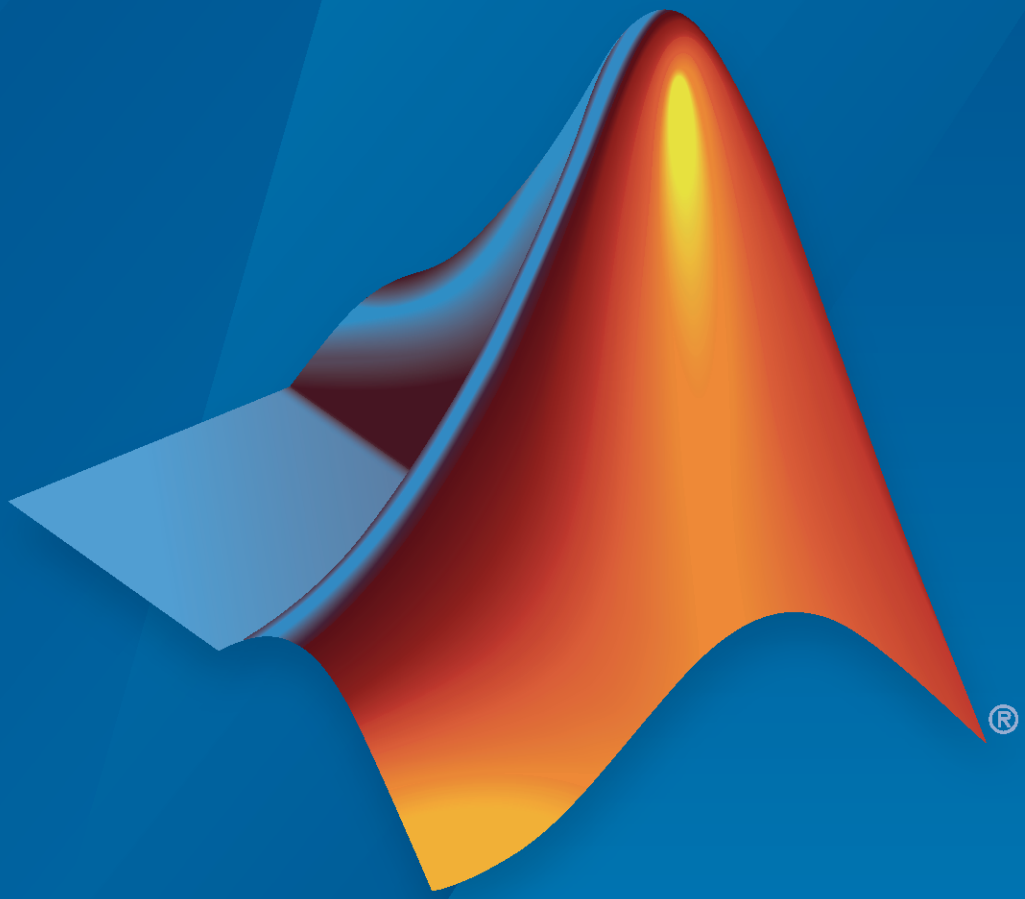
Image Processing Toolbox™

Reference



MATLAB®

R2022a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Image Processing Toolbox™ Reference

Revision History

August 1993	First printing	Version 1
May 1997	Second printing	Version 2
April 2001	Third printing	Revised for Version 3.0
June 2001	Online only	Revised for Version 3.1 (Release 12.1)
July 2002	Online only	Revised for Version 3.2 (Release 13)
May 2003	Fourth printing	Revised for Version 4.0 (Release 13.0.1)
September 2003	Online only	Revised for Version 4.1 (Release 13.SP1)
June 2004	Online only	Revised for Version 4.2 (Release 14)
August 2004	Online only	Revised for Version 5.0 (Release 14+)
October 2004	Fifth printing	Revised for Version 5.0.1 (Release 14SP1)
March 2005	Online only	Revised for Version 5.0.2 (Release 14SP2)
September 2005	Online only	Revised for Version 5.1 (Release 14SP3)
March 2006	Online only	Revised for Version 5.2 (Release 2006a)
September 2006	Online only	Revised for Version 5.3 (Release 2006b)
March 2007	Online only	Revised for Version 5.4 (Release 2007a)
September 2007	Online only	Revised for Version 6.0 (Release 2007b)
March 2008	Online only	Revised for Version 6.1 (Release 2008a)
October 2008	Online only	Revised for Version 6.2 (Release 2008b)
March 2009	Online only	Revised for Version 6.3 (Release 2009a)
September 2009	Online only	Revised for Version 6.4 (Release 2009b)
March 2010	Online only	Revised for Version 7.0 (Release 2010a)
September 2010	Online only	Revised for Version 7.1 (Release 2010b)
April 2011	Online only	Revised for Version 7.2 (Release 2011a)
September 2011	Online only	Revised for Version 7.3 (Release 2011b)
March 2012	Online only	Revised for Version 8.0 (Release 2012a)
September 2012	Online only	Revised for Version 8.1 (Release 2012b)
March 2013	Online only	Revised for Version 8.2 (Release 2013a)
September 2013	Online only	Revised for Version 8.3 (Release 2013b)
March 2014	Online only	Revised for Version 9.0 (Release 2014a)
October 2014	Online only	Revised for Version 9.1 (Release 2014b)
March 2015	Online only	Revised for Version 9.2 (Release 2015a)
September 2015	Online only	Revised for Version 9.3 (Release 2015b)
March 2016	Online only	Revised for Version 9.4 (Release 2016a)
September 2016	Online only	Revised for Version 9.5 (Release 2016b)
March 2017	Online only	Revised for Version 10.0 (Release 2017a)
September 2017	Online only	Revised for Version 10.1 (Release 2017b)
March 2018	Online only	Revised for Version 10.2 (Release 2018a)
September 2018	Online only	Revised for Version 10.3 (Release 2018b)
March 2019	Online only	Revised for Version 10.4 (Release 2019a)
September 2019	Online only	Revised for Version 11.0 (Release 2019b)
March 2020	Online only	Revised for Version 11.1 (Release 2020a)
September 2020	Online only	Revised for Version 11.2 (Release 2020b)
March 2021	Online only	Revised for Version 11.3 (Release 2021a)
September 2021	Online only	Revised for Version 11.4 (Release 2021b)
March 2022	Online only	Revised for Version 11.5 (Release 2022a)

Contents

Functions

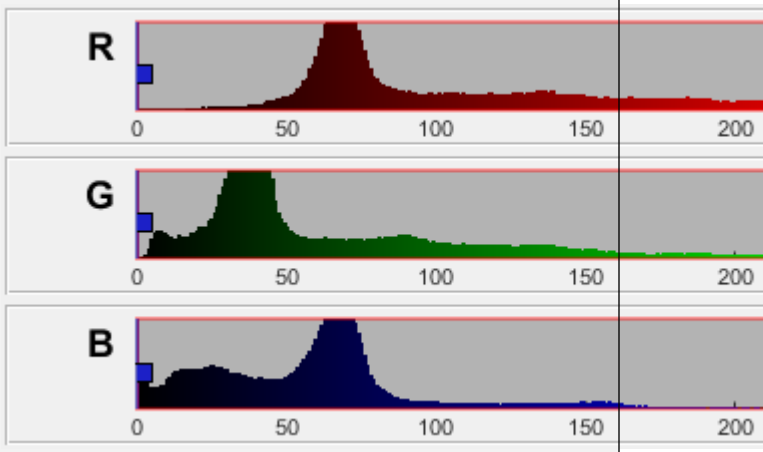
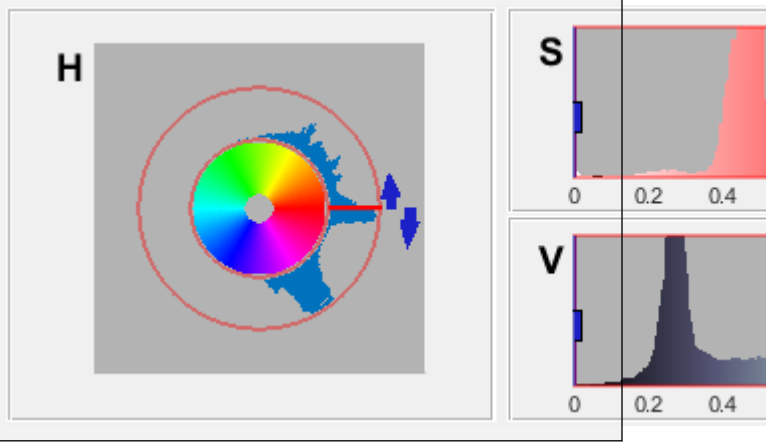
Color Thresholder

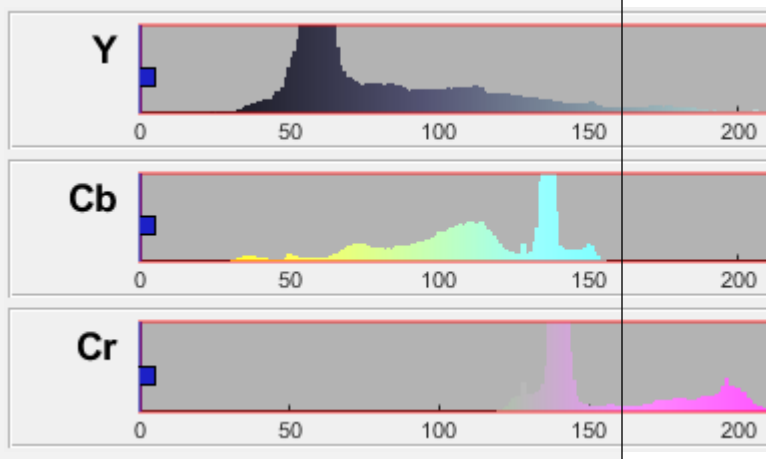
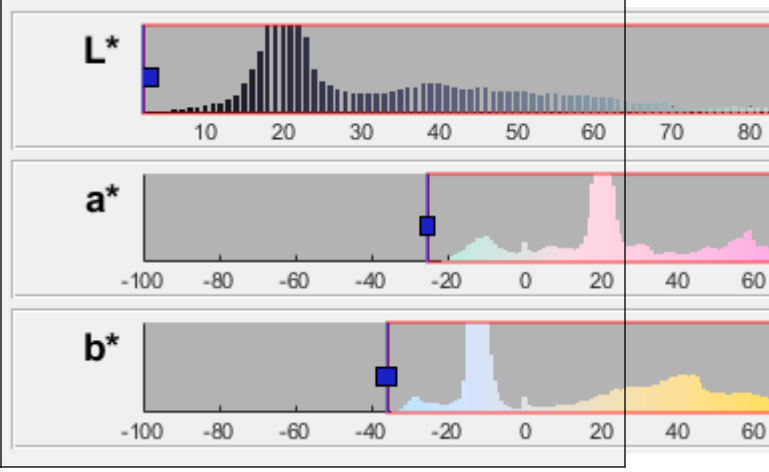
Threshold color image

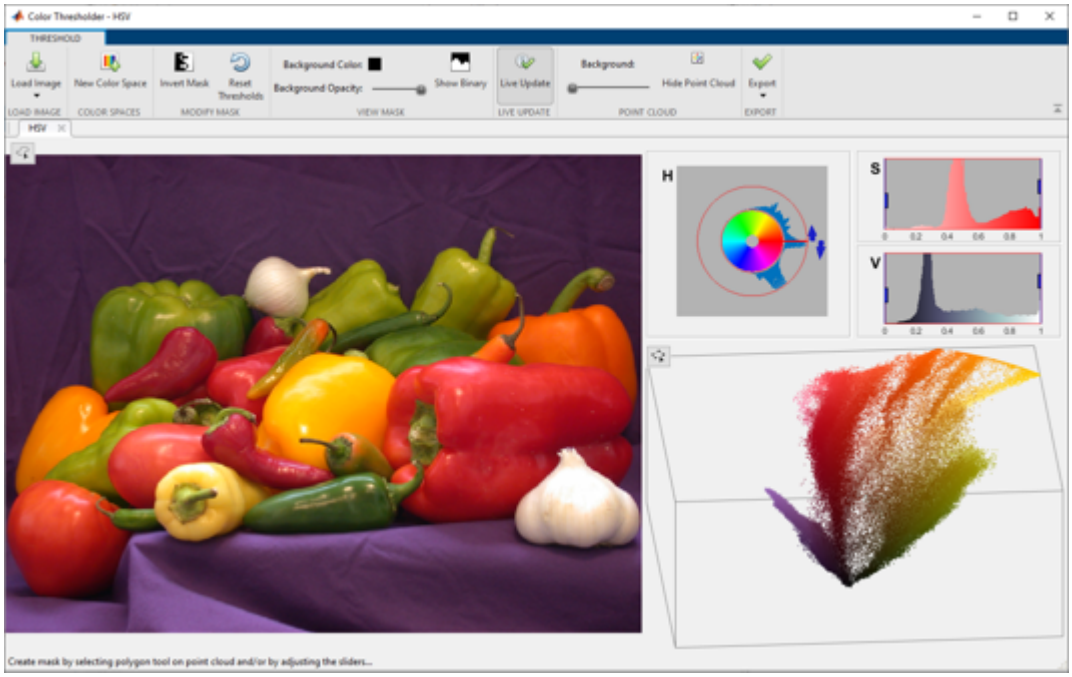
Description

The **Color Thresholder** app lets you segment color images by thresholding the color channels based on different color spaces. Using this app, you can create a binary segmentation mask for a color image.

Color Thresholder supports segmentation in four color spaces. In each color space, the app displays the image, the three color channels, and the color value of all pixels as points in a 3-D color space plot. You can select the colors included in the mask by windowing the color channel values or by drawing an ROI in the image or 3-D color space plot. For an example of using the app, see “Segment Image and Create Mask Using Color Thresholder App”.

Color Space	Color Channel Thresholding Controls
RGB	
HSV	

Color Space	Color Channel Thresholding Controls
YCbCr	
L*a*b*	



Open the Color Thresholder App

- MATLAB® Toolstrip: On the **Apps** tab, under **Image Processing and Computer Vision**, click the **Color Thresholder** app icon.
- From the MATLAB command prompt, use the `colorThresholder` function with a syntax described in “Programmatic Use” on page 1-4.

Examples

- “Segment Image and Create Mask Using Color Thresholder App”

Programmatic Use

`colorThresholder` opens **Color Thresholder**, which enables you to create a segmentation mask of a color image based on the exploration of different color spaces.

`colorThresholder(RGB)` opens **Color Thresholder**, loading the image `RGB` into the app.

`colorThresholder close` closes all open instances of **Color Thresholder**.

See Also

Apps
Image Segmenter

Functions
`imcontrast`

Topics
“Segment Image and Create Mask Using Color Thresholder App”
“Understanding Color Spaces and Color Space Conversion”

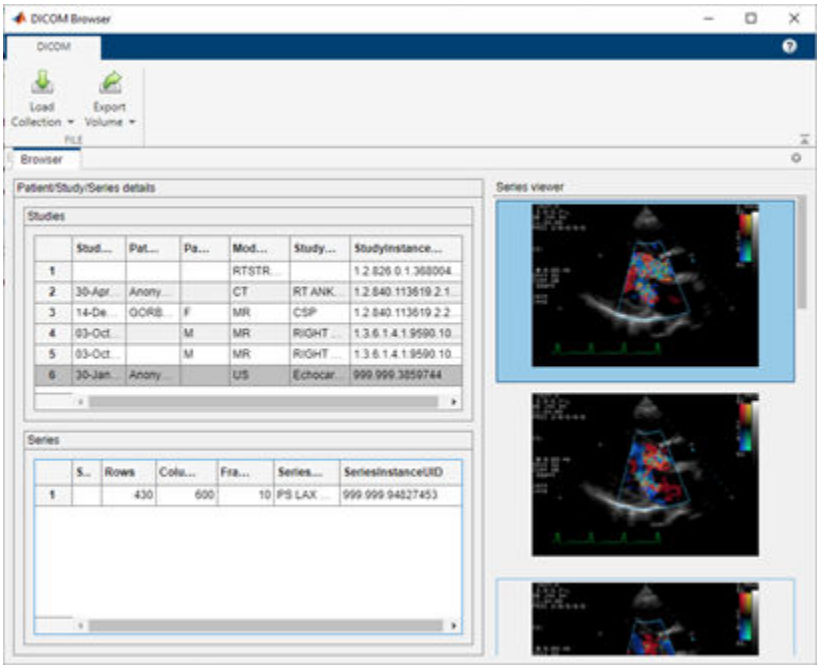
Introduced in R2014a

DICOM Browser

Explore collection of DICOM files

Description

The **DICOM Browser** app lets you explore the contents of collections of DICOM files. The app sorts images by study and series. You can select a series and save it to the MATLAB workspace. The DICOM Browser stores the data as a volume, with separate variables for a colormap and for spatial details.



Open the DICOM Browser App

- MATLAB Toolstrip: On the **Apps** tab, under **Image Processing and Computer Vision**, click the DICOM Browser app icon.
- MATLAB command prompt: Enter `dicomBrowser`.

Examples

Explore by Folder Name

Open the DICOM Browser, displaying DICOM files from the sample image folder.

```
dicomBrowser(fullfile(matlabroot,'toolbox/images/imdata'))
```

Explore by DICOMDIR File

Open the DICOM Browser and explore a DICOM folder by using the DICOMDIR file.

```
dicomBrowser(fullfile(matlabroot,'toolbox/images/imdata/DICOMDIR'))
```

Programmatic Use

`dicomBrowser` opens the DICOM Browser app for exploring the contents of collections of DICOM files.

`dicomBrowser(DIR)` opens the DICOM Browser app, displaying details about the files in the folder `DIR` and its subfolders. `DIR` can contain a full path name, a relative path name to the file, or the name of a file on the MATLAB search path.

`dicomBrowser(DICOMDIR)` opens the DICOM Browser app and gathers details from the DICOM directory file, named `DICOMDIR`. A DICOM directory file is a special DICOM file that serves as a directory to a collection of DICOM files stored on removable media, such as CD/DVD ROMs. `DICOMDIR` can contain a full path name or a relative path name to the file. The name of this file is `DICOMDIR`, with no file extension.

See Also

Functions

`dicomanon` | `dicomdict` | `dicomdisp` | `dicominfo` | `dicomlookup` | `dicomwrite` | `dicomuid`

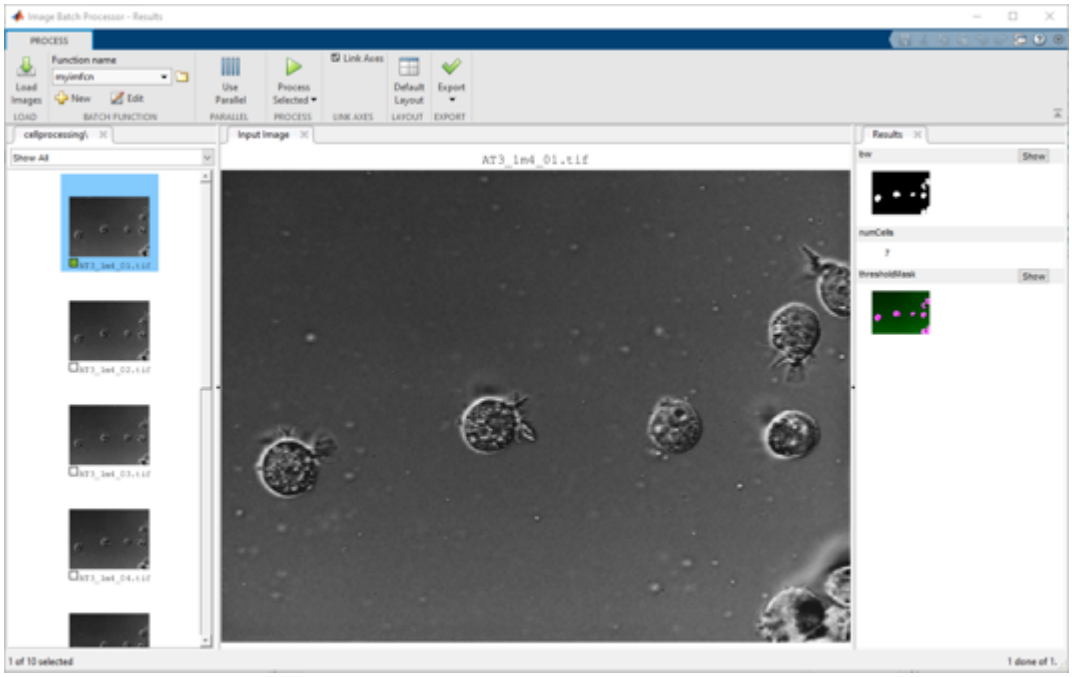
Introduced in R2017b

Image Batch Processor

Apply a function to multiple images

Description

The **Image Batch Processor** app lets you process a folder of images using a function you specify. The function must have the following signature: `out = fcn(in)`. The app creates an output folder containing the processed images, using the same name and subfolder structure as the input folder.



Open the Image Batch Processor App

- MATLAB Toolstrip: On the **Apps** tab, under **Image Processing and Computer Vision**, click the Image Batch Processor app icon.
- MATLAB command prompt: Enter `imageBatchProcessor`.

Examples

- “Process Folder of Images Using Image Batch Processor App”

Programmatic Use

`imageBatchProcessor` opens the Image Batch Processor app, which enables you to process a folder of images.

`imageBatchProcessor close` closes all open instances of the Image Batch Processor app.

See Also

Functions
`imread` | `imwrite`

Topics
“Process Folder of Images Using Image Batch Processor App”

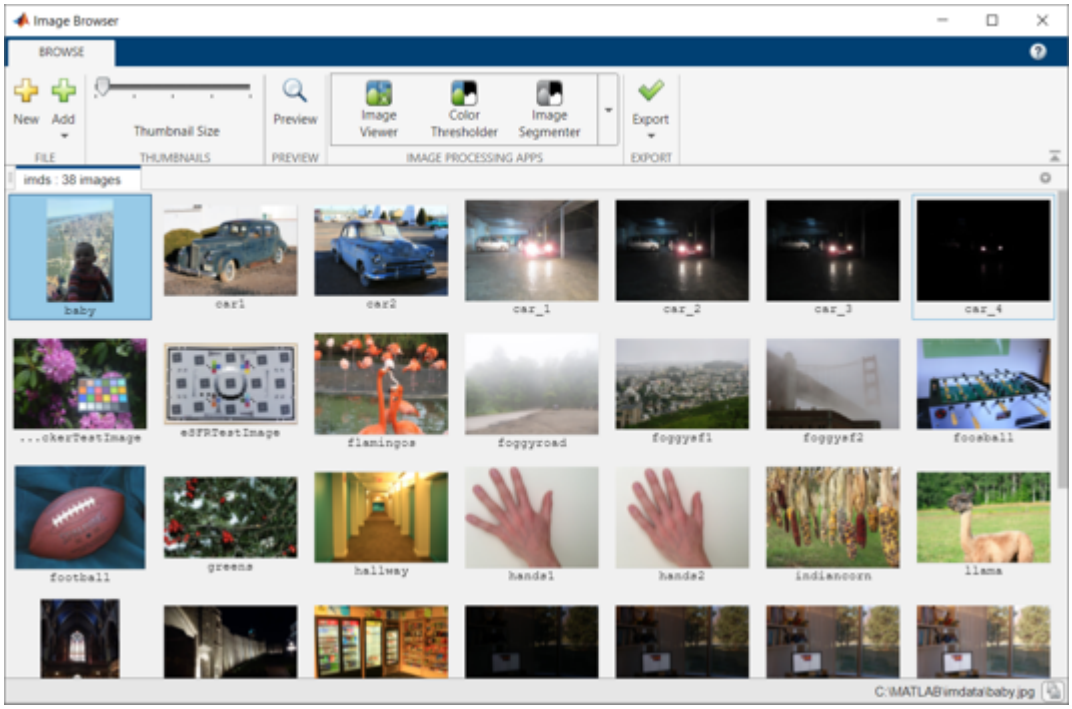
Introduced in R2015a

Image Browser

Browse images using thumbnails

Description

The **Image Browser** app lets you view thumbnails of a collection of images in folders or image datastore. You can select images to view information such as the image size and data type, or open an image in one of several Image Processing Toolbox apps. You can save images displayed in the app to the MATLAB workspace as an `ImageDatastore` object.



Open the Image Browser App

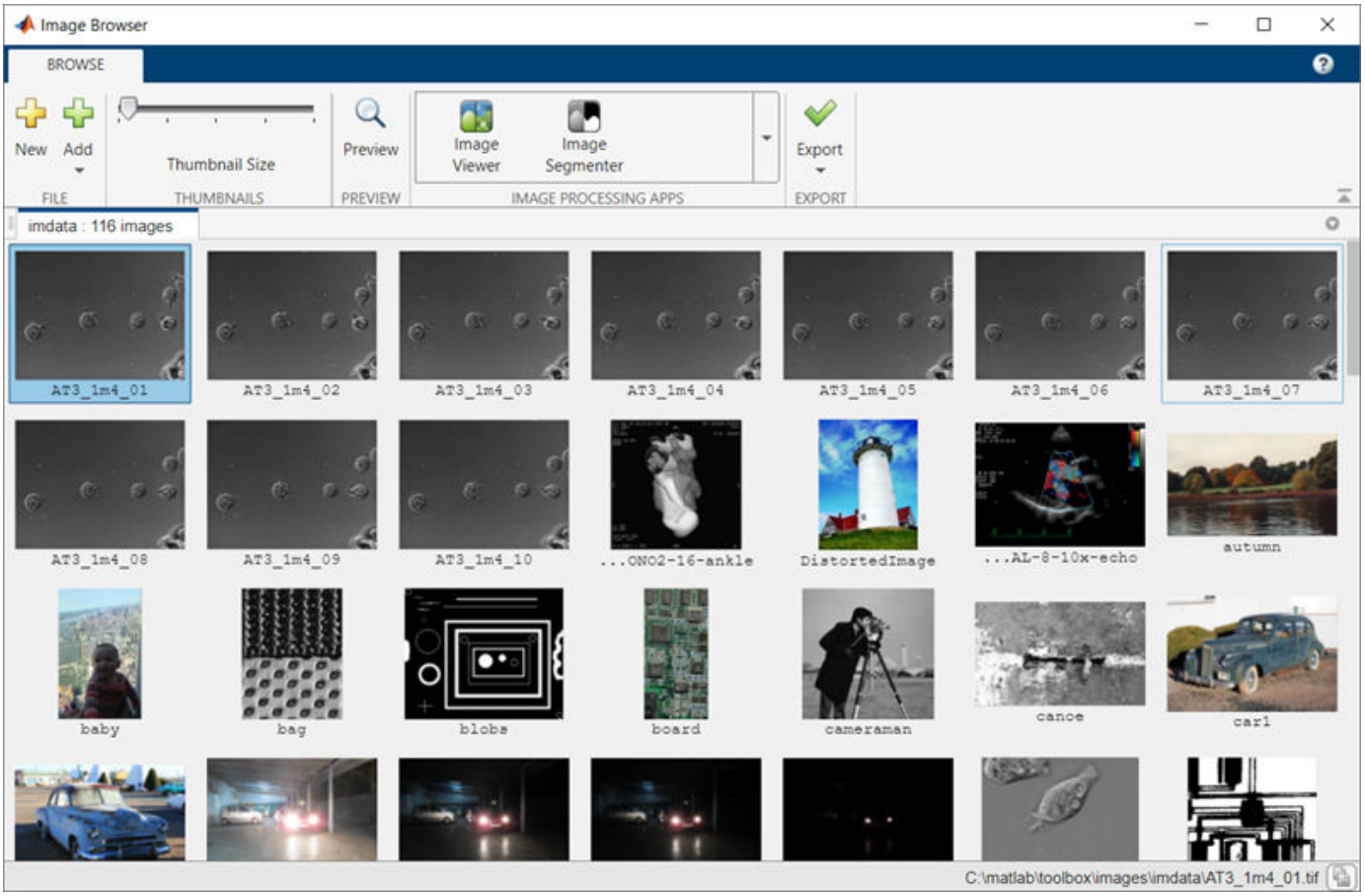
- MATLAB Toolstrip: On the **Apps** tab, under **Image Processing and Computer Vision**, click the Image Browser app icon.
- MATLAB command prompt: Enter `imageBrowser`.

Examples

Browse Images in Folder and Subfolders

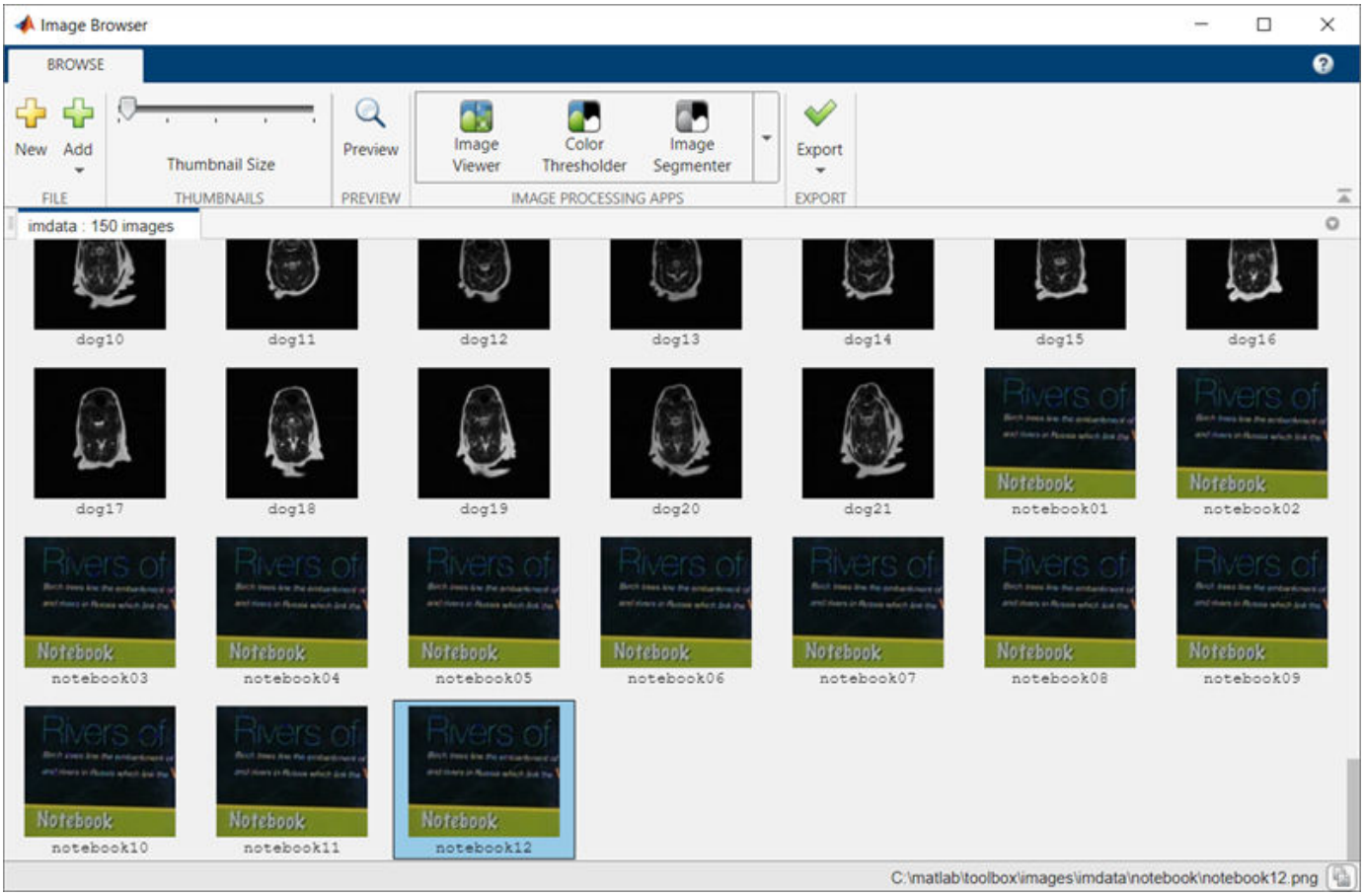
Open the app using the `imageBrowser` function. Specify the name of the folder that contains the images that you want to view. When you specify the name of the folder, **Image Browser** displays only thumbnails of the images in that folder. The app ignores the images in subfolders. For example, to view thumbnails of the images in the `imdata` folder, use this command:

```
imageBrowser(fullfile(matlabroot,'toolbox/images/imdata/'))
```



If you want to view thumbnails of the images in the folder and all subfolders, you must load the images into the app using the **Add** button on the app toolstrip.

First, clear the current collection of images from the app by clicking **New**. Then, click **Add > Folder, include subfolders** on the app toolstrip. In the Select Folder dialog box, select the top level folder. The app displays thumbnails of the images in subfolders after the thumbnails of images in the top level folder.



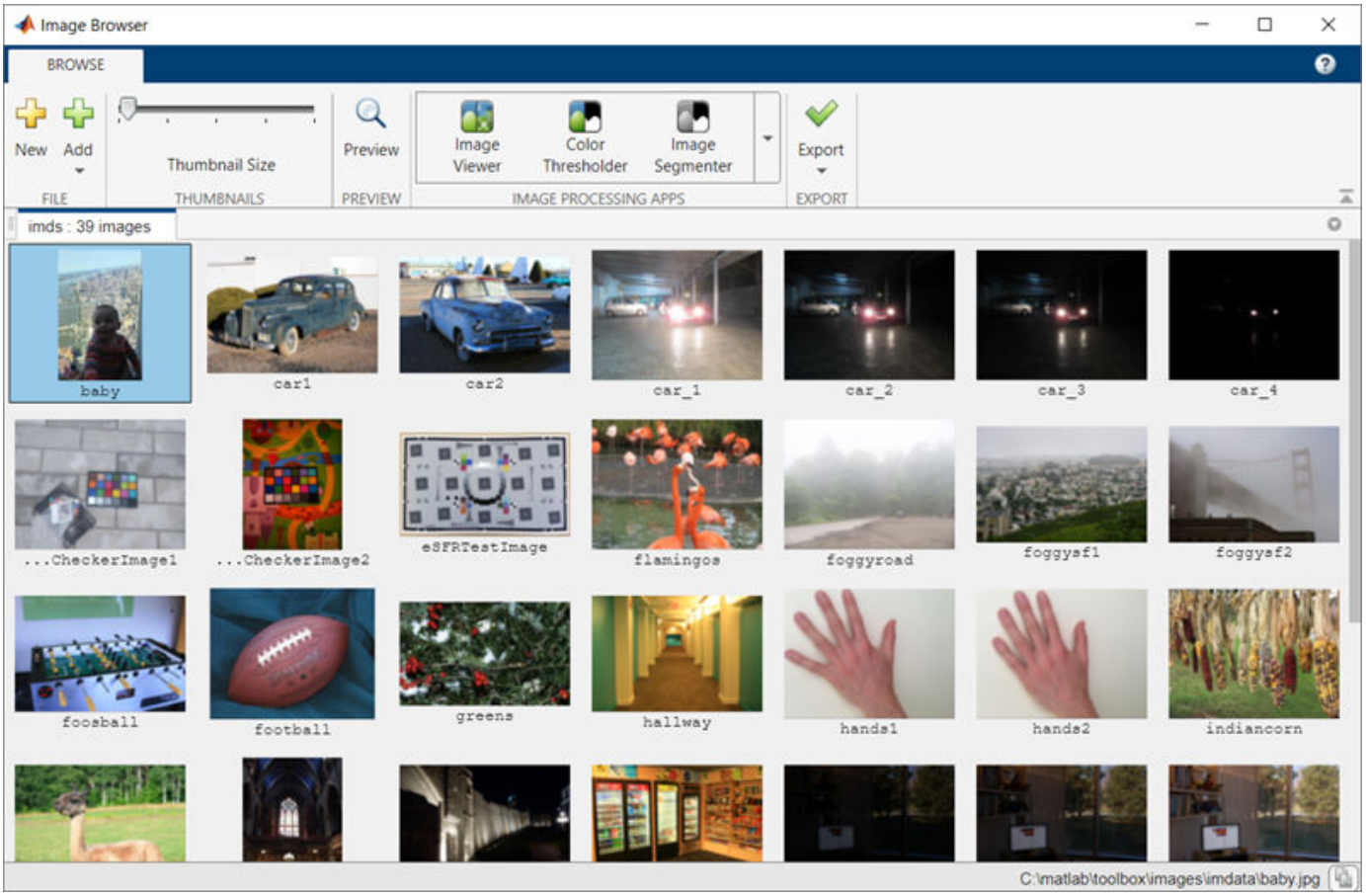
Browse Images in Datastore

Create an image datastore consisting of the JPEG images in the `imdata` folder.

```
imDir = fullfile(matlabroot,'toolbox/images/imdata');
imds = imageDatastore(imDir,FileExtensions=[".jpg",".jpeg"]);
```

Open the app using the `imageBrowser` function, specifying the name of the datastore.

```
imageBrowser(imds)
```



- “View and Edit Collection of Images in Folder or Datastore”

Programmatic Use

`imageBrowser` opens the **Image Browser** app.

`imageBrowser(folder)` opens the **Image Browser** app and loads all images in the folder `folder`. The app ignores images in subfolders.

`imageBrowser(imds)` opens the **Image Browser** app and loads all images in the image datastore `imds`.

See Also

Apps
Image Batch Processor

Functions
`imageDatastore`

Topics
“View and Edit Collection of Images in Folder or Datastore”
“Getting Started with Datastore”

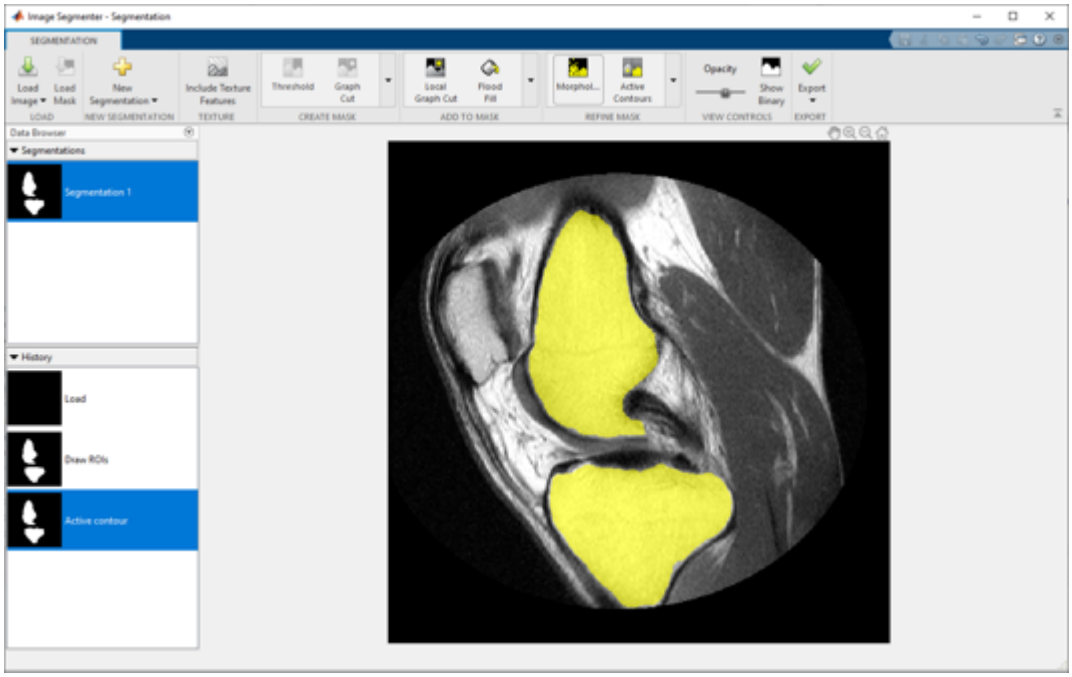
Introduced in R2016b

Image Segmenter

Segment an image by refining regions

Description

The **Image Segmenter** app lets you create a segmentation mask using automatic algorithms such as flood fill, semi-automatic techniques such as graph cut, and manual techniques such as drawing ROIs. You can also refine masks using morphology or an iterative approach such as active contours (also called snakes). For more information about creating, refining, and exporting a segmentation mask, see “Getting Started with Image Segmenter App”.



Open the Image Segmenter App

- MATLAB Toolstrip: Open the **Apps** tab, under **Image Processing and Computer Vision**, click the Image Segmenter app icon.
- MATLAB command prompt: Enter `imageSegmenter`.

Examples

- “Segment Image Using Thresholding in Image Segmenter”
- “Segment Image by Drawing Regions Using Image Segmenter”
- “Segment Image Using Active Contours in Image Segmenter”
- “Segment Image Using Auto Cluster in Image Segmenter”
- “Segment Image Using Graph Cut in Image Segmenter”

- “Segment Image Using Find Circles in Image Segmenter”
- “Segment Image Using Local Graph Cut (Grabcut) in Image Segmenter”
- “Refine Segmentation Using Morphology in Image Segmenter”

Programmatic Use

`imageSegmenter` opens the Image Segmenter app, which enables you to create a segmentation mask of an image by using active contours.

`imageSegmenter(I)` opens the Image Segmenter app, loading the image `I` into the app.

`imageSegmenter close` closes all open instances of the Image Segmenter app.

See Also

Functions
`activecontour` | `imbinarize` | `grayconnected` | `lazysnapping` | `grabcut` | `imfindcircles`

Topics
“Segment Image Using Thresholding in Image Segmenter”
“Segment Image by Drawing Regions Using Image Segmenter”
“Segment Image Using Active Contours in Image Segmenter”
“Segment Image Using Auto Cluster in Image Segmenter”
“Segment Image Using Graph Cut in Image Segmenter”
“Segment Image Using Find Circles in Image Segmenter”
“Segment Image Using Local Graph Cut (Grabcut) in Image Segmenter”
“Refine Segmentation Using Morphology in Image Segmenter”
“Getting Started with Image Segmenter App”

Introduced in R2014b

Image Region Analyzer

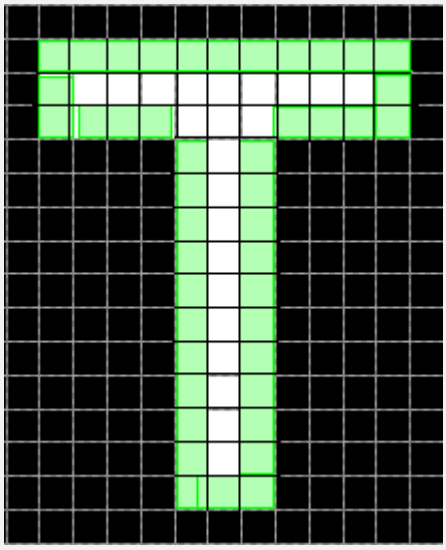
Browse and filter connected components in an image

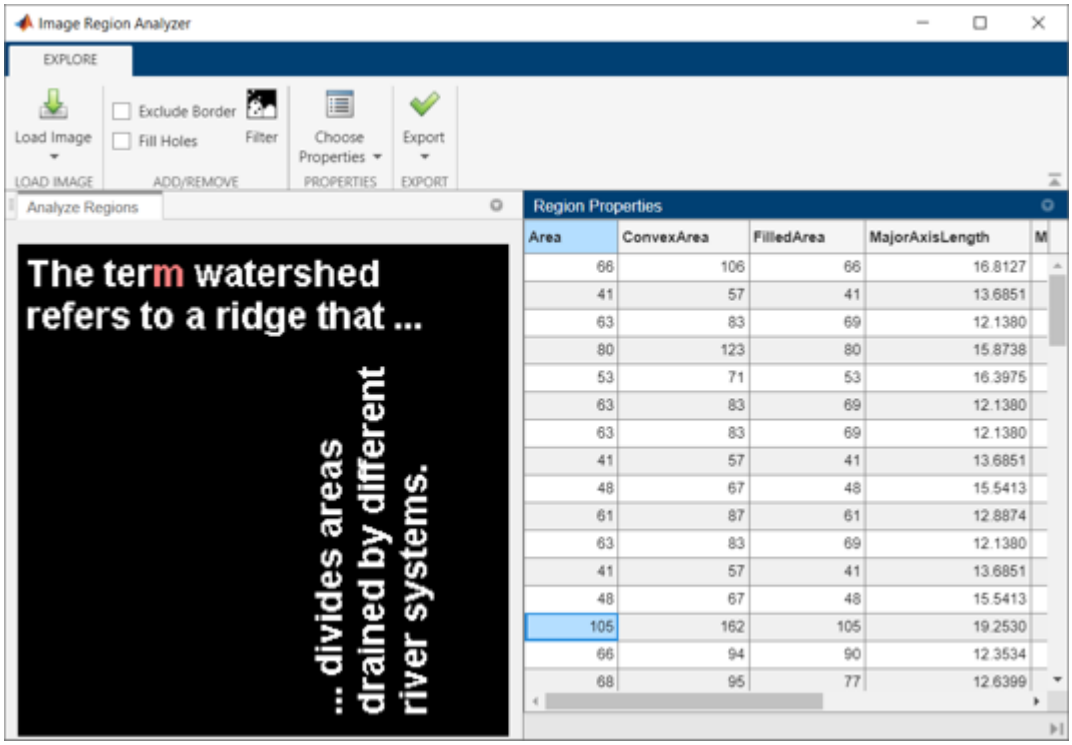
Description

The **Image Region Analyzer** app measures a set of properties for each connected component (also called an object or region) in a binary image and displays this information in a table. You can also use this app to create other binary images by filtering the image on region properties.

Image Region Analyzer can calculate these properties.

Property Name	Description
`"Area"`	Number of pixels in the region.
`"ConvexArea"`	Number of pixels in the convex hull. The convex hull is the smallest convex polygon that can contain the region. For more information about classifying pixels on the boundary of the hull, see “Classify Pixels That Are Partially Enclosed by ROI”.
`"Eccentricity"`	Eccentricity of the ellipse that has the same second-moments as the region. The eccentricity is the ratio of the distance between the foci of the ellipse and its major axis length. The value is between 0 and 1. (0 and 1 are degenerate cases. An ellipse whose eccentricity is 0 is actually a circle, while an ellipse whose eccentricity is 1 is a line segment.)
`"EquivDiameter"`	Diameter (in pixels) of a circle with the same area as the region, calculated as `sqrt(4*Area/pi)`.
`"EulerNumber"`	Euler number (also known as the Euler characteristic), calculated as 1 minus the number of holes in the region.
`"Extent"`	Ratio of pixels in the region to pixels in the total bounding box, calculated as `Area` divided by the area of the bounding box.
`"FilledArea"`	Number of pixels in the region after filling all holes in the region.
`"MajorAxisLength"`	Length (in pixels) of the major axis of the ellipse that has the same normalized second central moments as the region.
`"MinorAxisLength"`	Length (in pixels) of the minor axis of the ellipse that has the same normalized second central moments as the region.

Property Name	Description
`"Orientation"`	Angle (in degrees) between the *x*-axis and the major axis of the ellipse that has the same second-moments as the region. The value is in the range (–90, 90]. This figure illustrates the axes and orientation of the ellipse. The left side of the figure shows an image region and its corresponding ellipse. The right side shows the same ellipse with the solid blue lines representing the axes. The red dots are the foci. The orientation is the angle between the horizontal dotted line and the major axis. 
`"Perimeter"`	Distance (in pixels) around the boundary of the region, calculated by adding the distance between each adjoining pair of pixels around the border of the region. This figure illustrates the pixels included in the perimeter calculation for a sample region. 
`"Solidity"`	Proportion of the pixels in the convex hull that are also in the region, calculated as `Area/ConvexArea`.



Open the Image Region Analyzer App

- MATLAB Toolstrip: On the **Apps** tab, under **Image Processing and Computer Vision**, click the **Image Region Analyzer** app icon.
- MATLAB command prompt: Enter `imageRegionAnalyzer`.

Examples

- “Calculate Properties of Image Regions Using Image Region Analyzer”
- “Filter Images on Properties Using Image Region Analyzer App”

Programmatic Use

`imageRegionAnalyzer` opens the **Image Region Analyzer** app, which enables you to create other binary images and get information about the regions within binary images.

`imageRegionAnalyzer(I)` opens the **Image Region Analyzer** app, loading the image `I` into the app.

`imageRegionAnalyzer close` closes all open instances of the **Image Region Analyzer** app.

Tips

- **Image Region Analyzer** uses the `regionprops` function to identify regions in the image and calculate properties of those regions. Note that the `regionprops` offers additional capabilities for calculating region properties:
  - Measure properties of images with discontiguous regions
  - Measure additional properties such as the centroid and the maximum and minimum Feret diameter and angle
  - Measure pixel value properties of grayscale images
  - Measure a subset of properties for N-D images
  - Perform measurements on a GPU and generate code
- **Image Region Analyzer** uses the `bwpropfilt` and `bwareafilt` functions to filter binary images.

See Also

Functions
`regionprops` | `bwpropfilt` | `bwareafilt`

Topics
“Calculate Properties of Image Regions Using Image Region Analyzer”
“Filter Images on Properties Using Image Region Analyzer App”

Introduced in R2014b

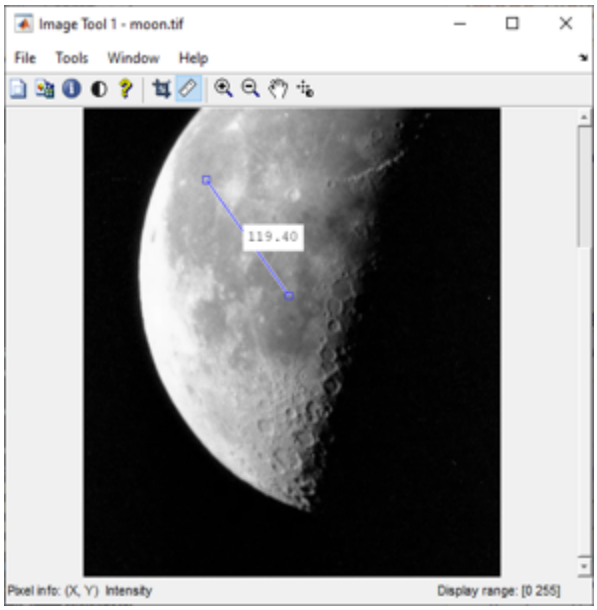
Image Viewer

View and explore images

Description

The **Image Viewer** app presents an integrated environment for displaying images and performing common image processing tasks.

Image Viewer provides all the image display capabilities of `imshow`, which optimizes figure, axes, and image object property settings for image display. **Image Viewer** also provides access to several tools for navigating and exploring images, such as the Pixel Region tool, Image Information tool, and the Adjust Contrast tool. To learn more about the available tools, see “Get Started with Image Viewer App”.



Open the Image Viewer App

- MATLAB Toolstrip: On the **Apps** tab, under **Image Processing and Computer Vision**, click the **Image Viewer** app icon.
- MATLAB command prompt: Use the `imtool` function.

Examples

Display Different Types of Images

Display a color image from a file.

```
imtool('board.tif')
```

Display an indexed image.

```
[X,map] = imread('trees.tif');
imtool(X,map)
```

Display a grayscale image.

```
I = imread('cameraman.tif');
imtool(I)
```

Display a grayscale image, adjusting the display range.

```
h = imtool(I,[0 80]);
close(h)
```

- “Get Started with Image Viewer App”
- “Get Pixel Information in Image Viewer App”
- “Measure Distance Between Pixels in Image Viewer App”
- “Adjust Image Contrast in Image Viewer App”
- “Crop Image Using Image Viewer App”

Programmatic Use

`imtool` opens **Image Viewer** in an empty state.

- To load an from a file, select **File** > **Open**.
- To load an image stored as a variable in the workspace, select **File** > **Import from Workspace**.

`imtool(I)` opens **Image Viewer** and displays the grayscale, RGB, or binary image `I`. Specify `I` as one of these values.

- An *m*-by-*n* numeric matrix representing a grayscale image. **Image Viewer** displays the image using the default display range of the image data type.
- An *m*-by-*n*-by-3 numeric array representing an RGB image.
- An *m*-by-*n* logical matrix representing a binary image.

`imtool(X,cmap)` open **Image Viewer** and displays the indexed image `X` with colormap `cmap`.

- Specify `X` as an *m*-by-*n* matrix of data type `single`, `double`, `uint8`, or `logical`.
- Specify `cmap` as a *c*-by-3 numeric matrix containing the RGB values of *c* colors. `cmap` can be of data type `single`, `double`, `uint8`, or `uint16`, or `int16`.

`imtool(filename)` opens **Image Viewer** and displays the image file with file name `filename`. Specify `filename` as a character vector or string scalar.

`imtool( ___ ,'Colormap',cmap)` displays the grayscale or binary image in **Image Viewer** using the colormap `cmap`. Setting the colormap of an RGB image has no effect. Specify `cmap` as a *c*-by-3 numeric matrix with values in the range [0, 1], where *c* is the number of colors in the colormap. You can also create a colormap matrix using a predefined colormap function, such as `parula` or `jet`.

For example, `imtool(I,'Colormap',parula)` displays grayscale image `I` using the `parula` colormap.

`imtool( ___ ,'DisplayRange',dispRange)` displays a grayscale or indexed image in **Image Viewer** and scales the display range to the values in `dispRange`. Setting the display range of an RGB or binary image has no effect. Specify `dispRange` as one of these values.

- 2-element vector of the form `[low high]` — **Image Viewer** displays pixels with the value low (and any value less than low) as black. **Image Viewer** displays pixels with the value high (and any value greater than high) as white. Pixel values within the display range are displayed as intermediate shades of gray using the default number of gray levels.

  For example, `imtool(I,'DisplayRange',[15 140])` scales the display range of grayscale image `I` of data type `uint8` such that pixels less than or equal to `15` appear black and pixels greater than or equal to `140` appear white.
- `[]` — `Image Viewer` sets the display range to `[min(I(:)) max(I(:))]`. The minimum value in `I` is displayed as black, and the maximum value is displayed as white.

  For example, `imtool(I,'DisplayRange',[])` scales the display range of grayscale image `I` of data type `double` such that pixels with the minimum value appear black and pixels with the maximum value appear white.

`imtool( ___ ,'InitialMagnification',initMag)` displays the image with initial magnification `initMag`. Specify `initMag` as one of these values.

- `"adaptive"` — The entire image is visible on initial display. If the image is too large to display on the screen, then **Image Viewer** displays the image at the largest magnification that fits on the screen.
- `"fit"` — **Image Viewer** resizes the entire image to fit in the window.
- A positive number — **Image Viewer** resizes the entire image as a percentage of the original image size. For example, if you specify 100, then **Image Viewer** displays the image at 100% magnification (one screen pixel for each image pixel).

  For example, `imtool(I,'InitialMagnification',50)` displays image `I` at 50% of the original image dimensions.

Note When the image aspect ratio is such that less than one pixel would be displayed in either dimension at the requested magnification, **Image Viewer** issues a warning and displays the image at 100% magnification.

By default, the initial magnification is set to the value returned by `iptgetpref('ImtoolInitialMagnification')`. To change the default initial magnification behavior, see “Specify Default Display Behavior” on page 1-24.

`imtool( ___ ,'Interpolation',interp)` specifies the interpolation technique `interp` used to resize the image. Specify `interp` as `"nearest"` for nearest neighbor interpolation or `"bilinear"` for bilinear interpolation. The default interpolation technique is `"nearest"`.

For example, `imtool(I,'Interpolation',"bilinear")` resizes image `I` using bilinear interpolation.

`hfigure = imtool( ___ )` returns `hfigure`, the figure object created by **Image Viewer**.

`imtool close all` closes all open instances of **Image Viewer**.

More About

Specify Default Display Behavior

You can specify the default display behavior of **Image Viewer** by using the Image Processing Preferences dialog box. To access the dialog, select **File** > **Preferences** in the MATLAB desktop or **Image Viewer** menu. You can also set preferences programmatically by using the `iptsetpref` function.

Preference	Description
`'ImtoolInitialMagnification'`	Controls the initial magnification for image display. To override this toolbox preference, specify the `'InitialMagnification'` name-value argument when you call the `imtool` function, as follows: `imtool(___,'InitialMagnification',initial_mag)`
`'ImtoolStartWithOverview'`	Controls whether the Overview tool opens automatically when you open an image using **Image Viewer**. Possible values: • `true` — Overview tool opens when you open an image. • `false` — Overview tool does not open when you open an image. This is the default behavior.

For more information about these preferences, see `iptprefs`.

Large Data Support

To view very large TIFF or NITF images that will not fit into memory, you can use `rsetwrite` to create a reduced resolution dataset (R-Set) viewable in **Image Viewer**. R-Sets can also improve performance of **Image Viewer** for large images that fit in memory.

The following tools can be used with an R-Set: Overview, Zoom, Pan, Image Information, and Distance. Other tools, however, will not work with an R-Set. You cannot use the Pixel Region, Adjust Contrast, Crop Image, and Window/Level tools. Please note that the Pixel Information tool displays only the *x* and *y* coordinates of a pixel and not the associated intensity, index, or RGB values.

Tips

- If you want to set the display range when calling `imtool`, then the `'DisplayRange'` name is optional unless you specify the image using a file name. The syntax `imtool(I,[low high])` is equivalent to `imtool(I,'DisplayRange',[low high])`. However, you must specify the `'DisplayRange'` argument when calling `imtool` with a file name, as in the syntax `imtool(filename,'DisplayRange',[low high])`.
- **Image Viewer** does not close when you call the `close all` command. If you want to close multiple instances of the **Image Viewer** app, use the syntax `imtool close all` or select **Close all** from the **Image Viewer File** menu. You can close a specific **Image Viewer** specified by the handle `hfigure` by using the command `close(hfigure)`.

See Also

Apps
Video Viewer | Volume Viewer

Functions
`imshow`

Topics
“Get Started with Image Viewer App”
“Get Pixel Information in Image Viewer App”
“Measure Distance Between Pixels in Image Viewer App”
“Adjust Image Contrast in Image Viewer App”
“Crop Image Using Image Viewer App”

Introduced in R2014b

Registration Estimator

Register 2-D grayscale images

Description

The **Registration Estimator** app aligns 2-D grayscale images using automatic image registration. Using this app, you can:

- Compare feature-based, intensity-based, and nonrigid registration techniques interactively
- Obtain the registered image and the geometric transformation

Feature-Based Techniques

Registration Estimator offers these registration techniques that use feature detection and matching:

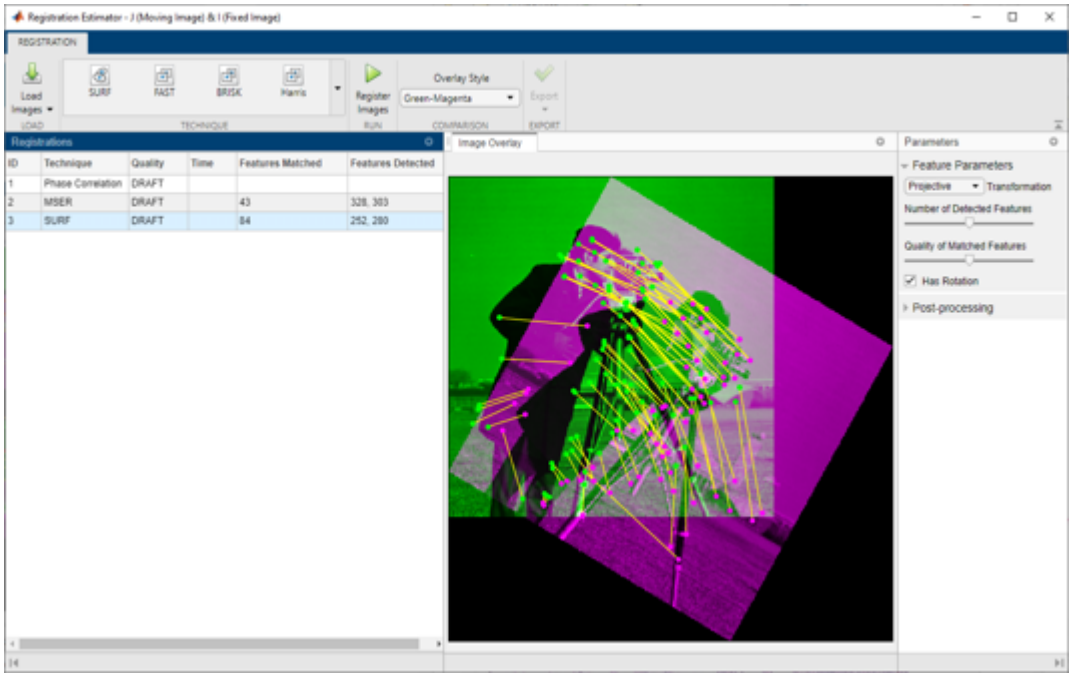
- BRISK
- FAST
- Harris
- KAZE
- MinEigen
- MSER
- ORB
- SURF

Intensity-Based Techniques

Registration Estimator offers three registration techniques that use intensity metric optimization:

- Monomodal intensity
- Multimodal intensity
- Phase correlation

For more details of the available techniques, see “Techniques Supported by Registration Estimator App”.



Open the Registration Estimator App

- MATLAB Toolstrip: On the **Apps** tab, under **Image Processing and Computer Vision**, click the **Registration Estimator** app icon.
- MATLAB command prompt: Enter `registrationEstimator`.

Examples

- “Register Images Using Registration Estimator App”

Programmatic Use

`registrationEstimator` opens the **Registration Estimator** app, which enables you to perform intensity-based, feature-based, and nonrigid image registration.

`registrationEstimator(moving,fixed)` opens the **Registration Estimator**, loading the grayscale images `moving` and `fixed` into the app.

`registrationEstimator close` closes all open instances of the **Registration Estimator** app.

See Also

Functions

`imregister` | `imregtform` | `imregconfig` | `imregdemons` | `imwarp`

Topics
“Register Images Using Registration Estimator App”
“Techniques Supported by Registration Estimator App”
“Approaches to Registering Images”

Introduced in R2017a

Video Viewer

View videos and image sequences

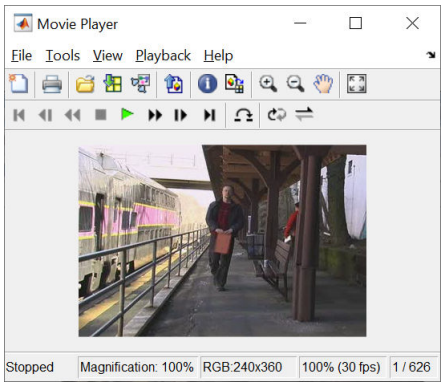
Description

The **Video Viewer** app plays movies, videos, or image sequences. The app offers basic video playback aids, including the ability to jump to a specific frame, to adjust the frame rate of the display, and to play in forwards and reverse directions.

Using this app, you can adjust other aspects of the video display and explore video data in more depth.

- Get information about frame size, color format, data type, and frame count.
- Change the colormap of grayscale and binary image sequences.
- Adjust the display range of grayscale image sequences.
- Open a Pixel Region tool that displays an extreme close-up view of a small region of pixels in the target image.
- Export a frame to the **Image Viewer** app.
- View video signals in Simulink® models (requires Simulink). For more information, see “View Video in Simulink” (Computer Vision Toolbox).

Video Viewer does not play audio tracks.



Open the Video Viewer App

- MATLAB Toolstrip: On the **Apps** tab, under **Image Processing and Computer Vision**, click the **Video Viewer** app icon.
- MATLAB command prompt: Enter `implay`.

Examples

Play Three Types of Videos

Animate a sequence of grayscale images at 10 frames per second.

```
load cellsequence
implay(cellsequence,10)
```

Visually explore a stack of grayscale MRI images.

```
load mristack
implay(mristack)
```

Play an AVI file.

```
implay("rhinos.avi")
```

- “View Image Sequences in Video Viewer App”

Programmatic Use

`implay` opens the **Video Viewer** app in an empty state.

- To load a video from a file, select **File** > **Open**.
- To load an image sequence stored as a variable in the workspace, select **File** > **Import from Workspace**.

`implay(filename)` opens the **Video Viewer** app and loads the content of the Audio Video Interleaved (AVI) file with file name `filename`. Specify `filename` as a character vector or string scalar. The AVI file determines the default frame rate.

`implay(I)` opens the **Video Viewer** app and displays the first frame in the multiframe image sequence `I`. Specify `I` as one of these values:

- An *m*-by-*n*-by-*k* numeric or logical array representing a grayscale or binary image sequence of *k* frames, respectively.
- An *m*-by-*n*-by-1-by-*k* numeric or logical array representing a grayscale or binary image sequence of *k* frames, respectively.
- An *m*-by-*n*-by-3-by-*k* numeric array representing a truecolor (RGB) image sequence of *k* frames.
- A MATLAB movie structure. For more information, see `immovie`.

For numeric data, the preferred data type of `I` is `uint8`. The actual data type used to display pixels may differ from the source data type.

The default frame rate is 20 frames per second. To change the frame rate, specify the second input argument, `fps`.

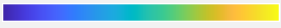
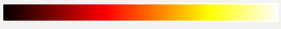
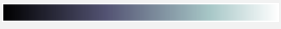
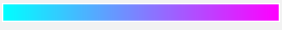
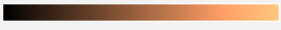
`implay( ___ ,fps)` also specifies the frame rate `fps` in frames per second. Specify `fps` as a positive number.

More About

Change Colormap of Grayscale or Binary Image Sequence

Change the colormap of a grayscale or binary image sequence by selecting **Tools** > **Colormap**. You cannot change the colormap of a truecolor (RGB) image sequence.

To change the colormap of a grayscale image sequence, select one of the seven built-in colormaps listed in the table. For a binary image sequence, only the `parula`, `jet`, `cool`, and `copper` colormaps modify the appearance of the images. **Video Viewer** does not support custom colormaps.

Colormap Name	Color Scale
`gray`	
`parula`	
`jet`	
`hot`	
`bone`	
`cool`	
`copper`	

Adjust Contrast of Grayscale Image Sequence

Adjust the contrast of a grayscale image sequence by selecting **Tools** > **Colormap**. You cannot change the contrast of a binary or truecolor (RGB) image sequence.

Enable contrast adjustment by selecting **Specify range of displayed pixel values**. The range of values depends on the data type of the image sequence. Specify the new minimum and maximum values of the display range.

- Pixel values less than or equal to the minimum display range value display as black, or the first value in the colormap for a nondefault colormap.
- Pixel values greater than or equal to the maximum display range value display as white, or the last value in the colormap for a nondefault colormap.
- Intermediate pixel values map linearly to the intermediate grayscale values, or the intermediate values in the colormap for a nondefault colormap.

Open Pixel Region Tool to Display Close-Up View

You can open a Pixel Region tool that displays an extreme close-up view of a small region of pixels in the target image. Access this tool by selecting **Tools** > **Pixel Region**. For more information about using the Pixel Region tool, see `impixelregion`.

See Also

Apps
Image Viewer

Functions
`immovie` | `montage` | `movie` | `VideoWriter`

Topics
“View Image Sequences in Video Viewer App”
“Work with Image Sequences as Multidimensional Arrays”
“Convert Multiframe Image to Movie”

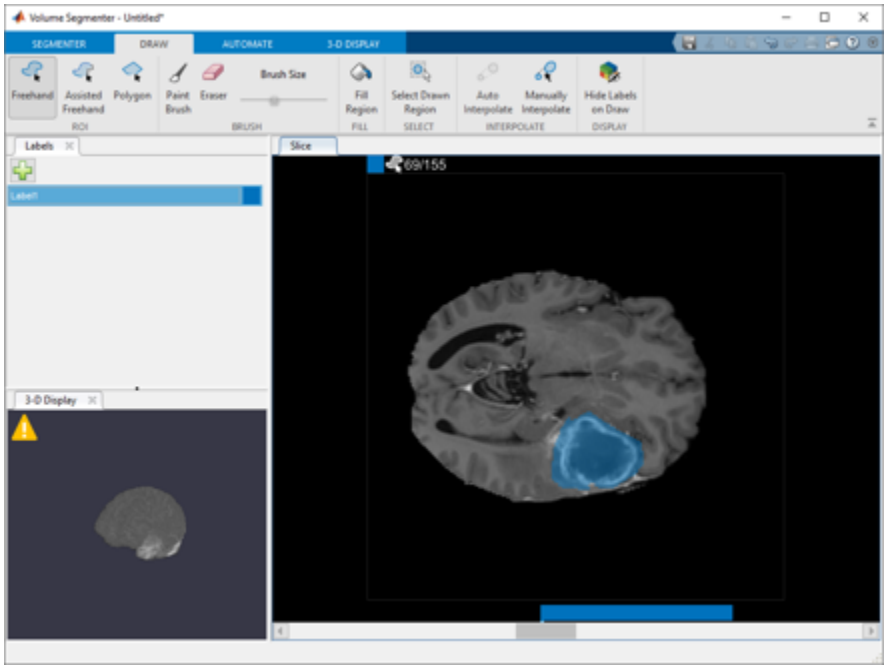
Introduced in R2014b

Volume Segmenter

Segment 3-D grayscale or RGB volumetric images

Description

The app can be used to create and refine a binary or semantic segmentation mask for a 3-D grayscale or an RGB image using automated, semi-automated, and manual techniques.



Open the Volume Segmenter App

- MATLAB Toolstrip: Open the **Apps** tab, under **Image Processing and Computer Vision**, click the Volume Segmenter app icon.
- MATLAB command prompt: Enter `volumeSegmenter`.

Examples

Load Volume and Labeled Volume into the Volume Segmenter

1. Load a volume into the workspace.

   ```
   load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled','images','vol_001.mat'));
   ```

2. Load the corresponding labeled volume into the workspace.

   ```
   load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled','labels','label_001.mat'));
   ```

3. Open the Volume Segmenter specifying both the volume and the labeled volume.

   ```
   volumeSegmenter(vol,label)
   ```

- “Create Binary Mask Using Volume Segmenter”
- “Create Semantic Segmentation Using Volume Segmenter”

Programmatic Use

`volumeSegmenter` opens a volume segmentation app.

`volumeSegmenter(V)` opens the Volume Segmenter app, loading the volume `V` into the app. Volume `V` is a scalar valued *m*-by-*n*-by-*p* or *m*-by-*n*-by-*p*-by-3 image of class `uint8`, `uint16`, `uint32`, `int8`, `int16`, `int32`, `single`, or `double`.

`volumeSegmenter(V,L)` opens the Volume Segmenter app, loading the volume `V` and the labeled volume `L` into the app. The labeled volume `L` is a scalar valued *m*-by-*n*-by-*p* image of class `logical`, `categorical`, `uint8`, `uint16`, `uint32`, `int8`, `int16`, `int32`, `single`, or `double`.

`volumeSegmenter( ___ ,'Show3DDisplay',TF)` logical value that specifies whether the Volume Segmenter includes a visualization of the 3-D volume in the app. The default value is `true`. However, the default is `false` on platforms where 3-D display is not supported, such as, Linux platforms, or on Windows platforms that use software versions of OpenGL.

See Also

Functions
Image Segmenter | Volume Viewer

Topics
“Create Binary Mask Using Volume Segmenter”
“Create Semantic Segmentation Using Volume Segmenter”

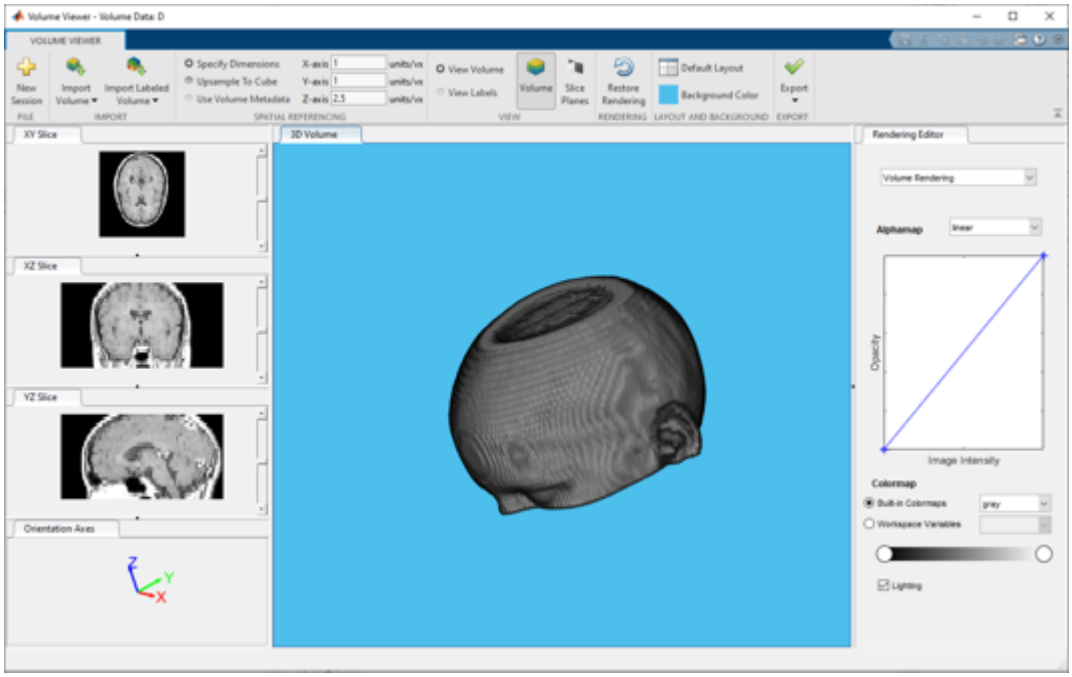
Introduced in R2020b

Volume Viewer

View volumetric data and labeled volumetric data

Description

The **Volume Viewer** app lets you view 3-D volumetric data and 3-D labeled volumetric data. Using this app, you can view the data as a volume or as plane slices. You can also view the data as a maximum intensity projection or an isosurface. Using the Rendering Editor component you can manipulate opacity to see the structures in the volume that you want to see and make transparent those structures in the volume that you do not want to see.



Open the Volume Viewer App

- MATLAB toolstrip: Open the **Apps** tab, under **Image Processing and Computer Vision**, click the Volume Viewer app icon.
- MATLAB command prompt: Enter `volumeViewer`.

Examples

Load Labeled Volume into Volume Viewer

1. Load a labeled volume into the workspace.

```
load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled','labels','label_001.mat'));
```

2. Open the labeled volume in the Volume Viewer. Use the `'VolumeType'` parameter to identify the volume as a labeled volume.

```
volumeViewer(label,'VolumeType','labels')
```

- “Explore 3-D Volumetric Data with Volume Viewer App”
- “Explore 3-D Labeled Volumetric Data with Volume Viewer App”

Programmatic Use

`volumeViewer` opens a volume visualization app.

`volumeViewer(V)` loads the intensity volume `V` into the app. `V` is a scalar-valued *m*-by-*n*-by-*p* image of class `logical`, `uint8`, `uint16`, `uint32`, `int8`, `int16`, `int32`, `single`, or `double`.

`volumeViewer(V,L)` loads the intensity volume `V` and the labeled volume `L` into the Volume Viewer. `L` is a scalar-valued *m*-by-*n*-by-*p* image of class `categorical`, `uint8`, `uint16`, `uint32`, `int8`, `int16`, `int32`, `single`, or `double`

`volumeViewer( ___ ,'VolumeType',vtype)` loads the volumetric data into the app, where `'VolumeType'` defines the type of volume being loaded. `vtype` can be either `'Volume'` or `'Labels'`. If the volume is of class `categorical`, the default `VolumeType` is `'Labels'`. For volumes of any other class, the default `VolumeType` is `'Volume'`. If you specify both an intensity volume and a labeled volume, the Volume Viewer ignores this name-value pair

`volumeViewer( ___ ,'ScaleFactors',sfactors)` loads the volumetric data into the app, where `'ScaleFactors'` specifies the scale factors used to rescale volumes. `'ScaleFactors'` is a 1-by-3 positive numeric array of the form `[x y z]`, where the values are scale factors applied in the *x*, *y*, and *z* directions. The default value is `[1 1 1]`. If `'VolumeType'` is `'Labels'`, the Volume Viewer ignores this name-value pair.

`volumeViewer close` closes all open Volume Viewer apps.

See Also

Functions
`isosurface` | `slice` | `volshow` | `labelvolshow` | `sliceViewer` | `obliqueslice` | `orthosliceViewer`

Topics
“Explore 3-D Volumetric Data with Volume Viewer App”
“Explore 3-D Labeled Volumetric Data with Volume Viewer App”

Introduced in R2017a

activecontour

Segment image into foreground and background using active contours (snakes) region growing technique

Syntax

```
BW = activecontour(A,mask)
BW = activecontour(A,mask,n)
BW = activecontour(A,mask,method)
BW = activecontour(A,mask,n,method)
BW = activecontour(___,Name,Value)
```

Description

The active contours technique, also called snakes, is an iterative region-growing image segmentation algorithm. Using the active contour algorithm, you specify initial curves on an image and then use the `activecontour` function to evolve the curves towards object boundaries.

`BW = activecontour(A,mask)` segments the image `A` into foreground (object) and background regions using active contours.

The `mask` argument is a binary image that specifies the initial state of the active contour. The boundaries of the object regions (white) in `mask` define the initial contour position used for contour evolution to segment the image. The output image `BW` is a binary image where the foreground is white (logical true) and the background is black (logical false).

To obtain faster and more accurate segmentation results, specify an initial contour position that is close to the desired object boundaries.

`BW = activecontour(A,mask,n)` segments the image by evolving the contour for a maximum of `n` iterations.

`BW = activecontour(A,mask,method)` specifies the active contour method used for segmentation as either `'Chan-Vese'` or `'edge'`. For RGB images, the method must be `'Chan-Vese'`.

`BW = activecontour(A,mask,n,method)` segments the image by evolving the contour for a maximum of `n` iterations using the specified `method`.

`BW = activecontour(___,Name,Value)` specifies name-value pair arguments that control various aspects of the segmentation.

Examples

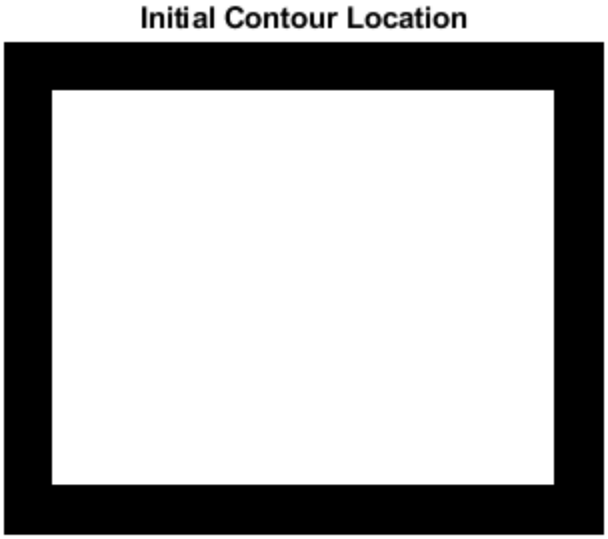
Segment Image Using Active Contours

Read and display a grayscale image.

```
I = imread('coins.png');
imshow(I)
title('Original Image')
```

Specify an initial contour surrounding the objects of interest. Display the contour.

```
mask = zeros(size(I));
mask(25:end-25,25:end-25) = 1;
imshow(mask)
title('Initial Contour Location')
```

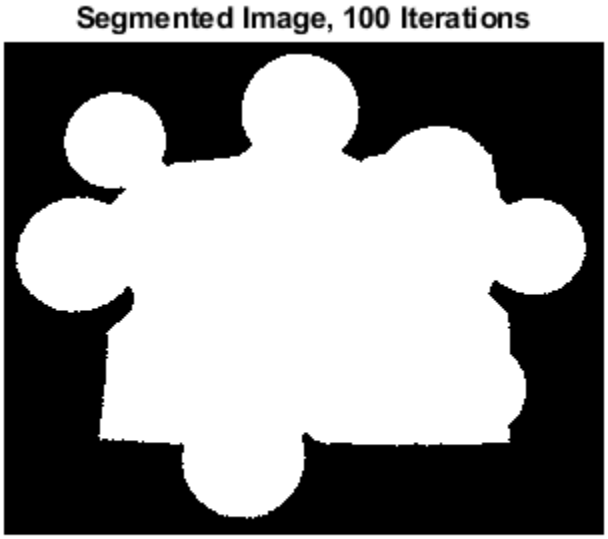


Segment the image by using the `activecontour` function. By default, the function evolves the segmentation through 100 iterations.

```
bw = activecontour(I,mask);
```

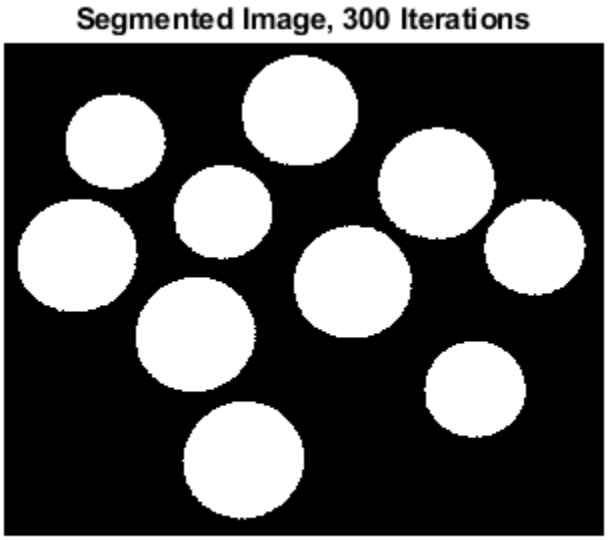
Display the result. After 100 iterations, objects are not fully segmented from the background because the original contour is not close to the object boundaries.

```
imshow(bw)
title('Segmented Image, 100 Iterations')
```



To continue evolving the segmentation, increase the number of iterations. After 300 iterations, objects are fully segmented from the background.

```
bw = activecontour(I,mask,300);
imshow(bw)
title('Segmented Image, 300 Iterations')
```



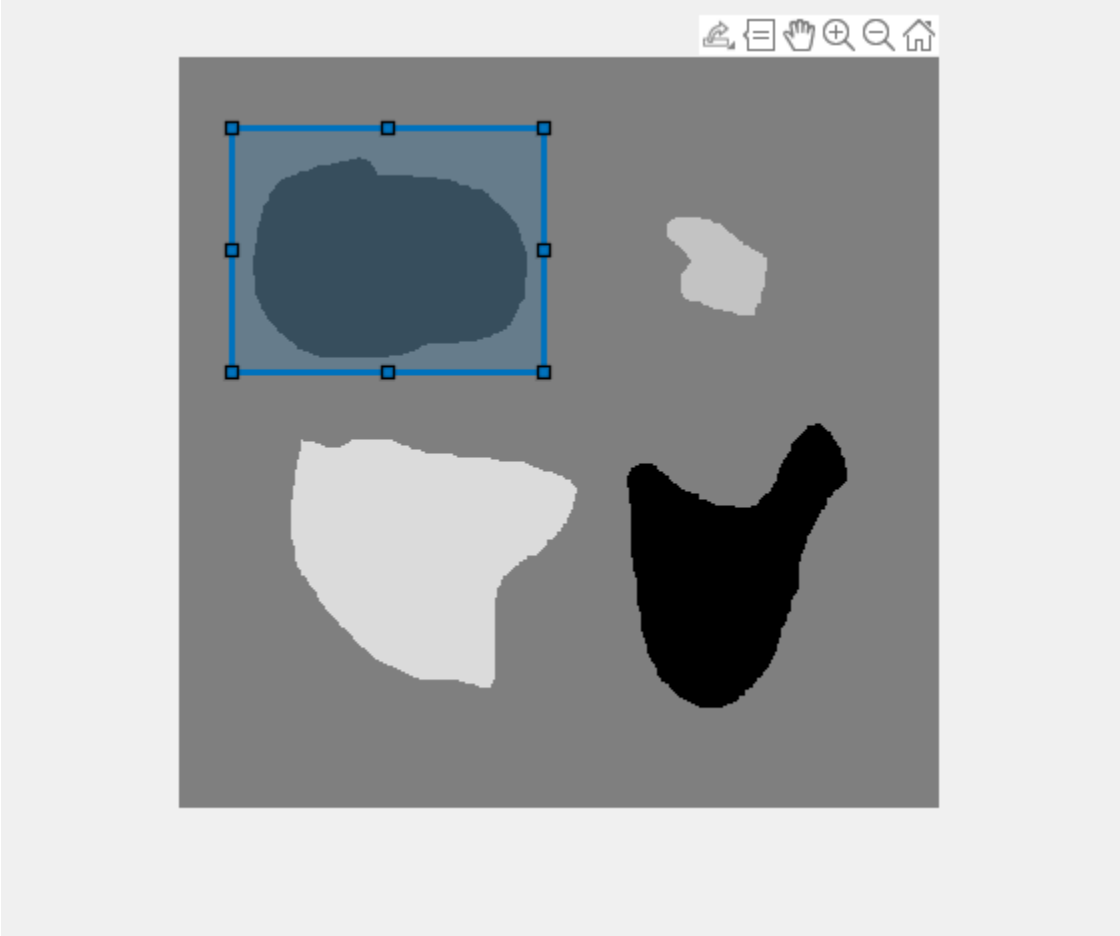
Segment Image Using Active Contours with Interactive Mask

Read and display a grayscale image.

```
I = imread('toyobjects.png');
imshow(I)
```

Draw an initial contour close to the object of interest by using the `drawrectangle` function. After drawing the contour, create a mask by using the `createMask` function.

```
r = drawrectangle;
```



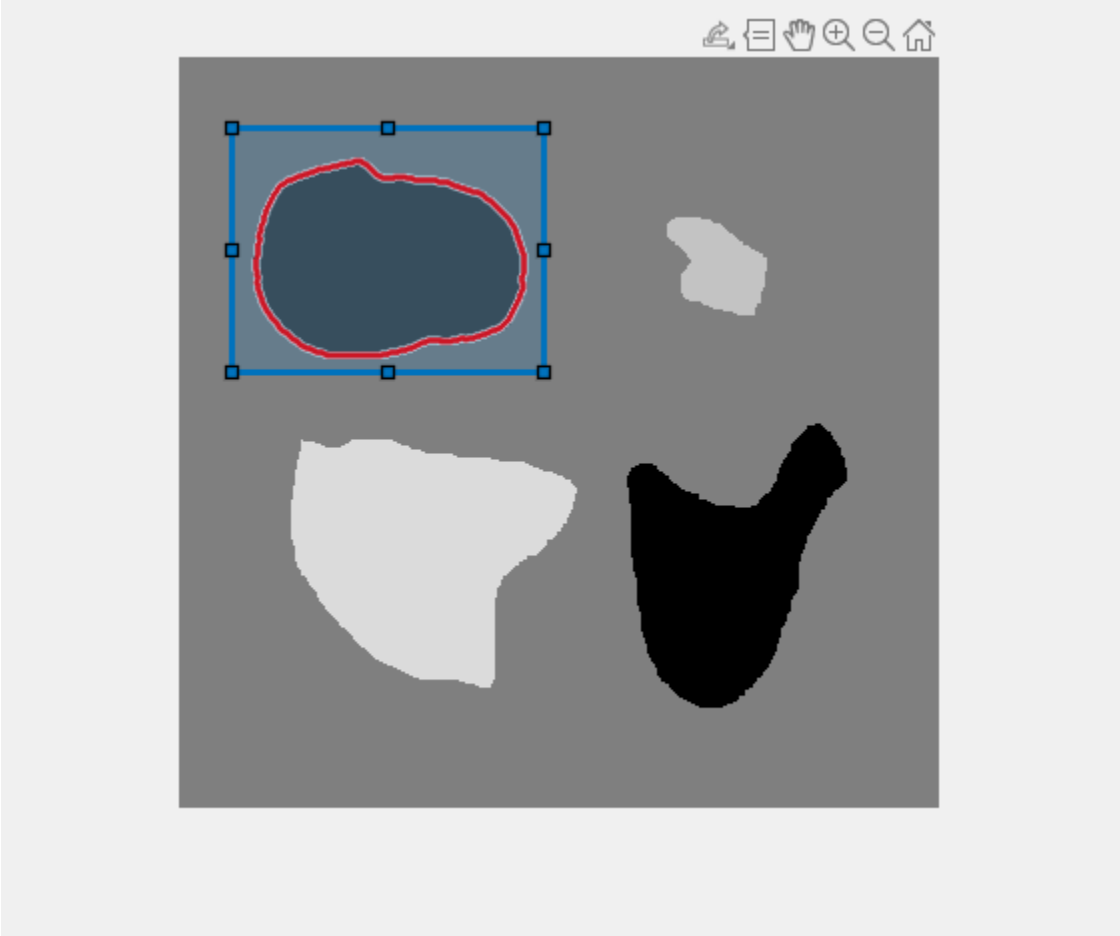
```
mask = createMask(r);
```

Segment the image using the `'edge'` method and 200 iterations.

```
bw = activecontour(I,mask,200,'edge');
```

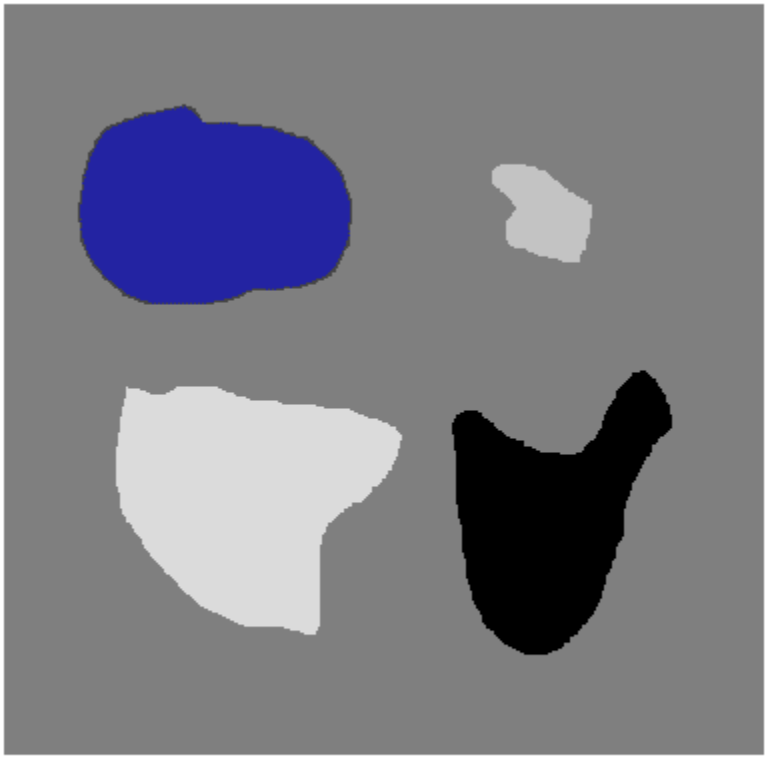
Display the final contour over the original image in red.

```
hold on;
visboundaries(bw,'Color','r');
```



Display the result of the segmentation over the original image. The object in the foreground has a blue color.

```
figure
imshow(labeloverlay(I,bw));
```



Perform 3-D Segmentation Using 2-D Initial Seed Mask

Load 3-D volumetric image data, removing the singleton dimension.

```
D = load('mri.mat');
A  = squeeze(D.D);
```

Create 2-D mask for initial seed points.

```
seedLevel = 10;
seed = A(:,:,seedLevel) > 75;
figure
imshow(seed)
```

Create an empty 3-D seed mask and put the seed points into it.

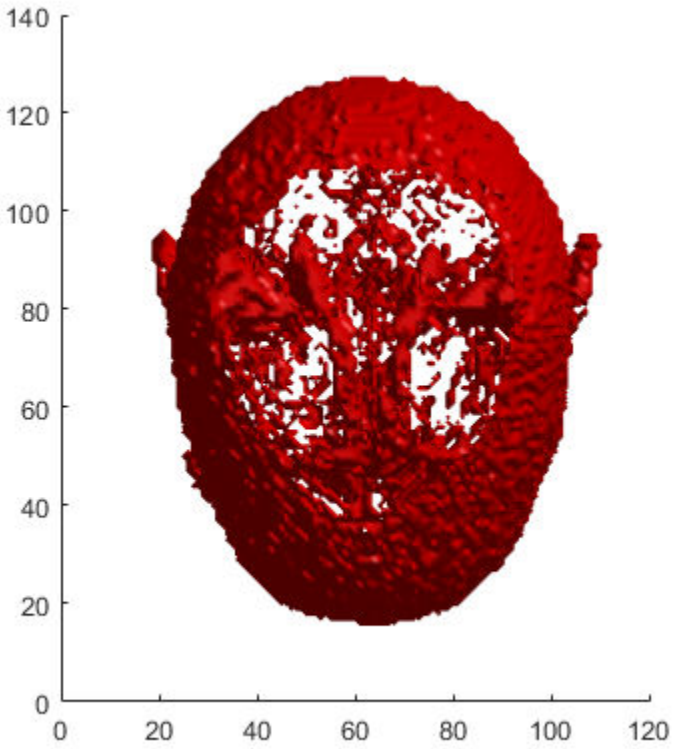
```
mask = zeros(size(A));
mask(:,:,seedLevel) = seed;
```

Perform the segmentation using active contours, specifying the seed mask.

```
bw = activecontour(A,mask,300);
```

Display the 3-D segmented image.

```
figure;
p = patch(isosurface(double(bw)));
p.FaceColor = 'red';
p.EdgeColor = 'none';
daspect([1 1 27/128]);
camlight;
lighting phong
```



Input Arguments

`A` — Image to be segmented

2-D numeric matrix | 3-D numeric array

Image to segmented, specified as a 2-D numeric matrix or 3-D numeric array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`mask` — Initial contour

binary image

Initial contour at which the evolution of the segmentation begins, specified as a binary image of the same size as `A`. For 2-D and 3-D grayscale images, the size of `mask` must match the size of the image `A`. For color and multi-channel images, `mask` must be a 2-D logical array where the first two dimensions match the first two dimensions of the image `A`.

You can create a mask interactively by using ROI objects. For example, draw a polygonal ROI by using the `drawpolygon` function, then create a mask from the ROI by using the `createMask` function.

Data Types: `logical`

`n` — Maximum number of iterations

`100` (default) | positive integer

Maximum number of iterations to perform in evolution of the segmentation, specified as a positive integer. `activecontour` stops the evolution of the active contour when it reaches the maximum number of iterations. `activecontour` also stops the evolution if the contour position in the current iteration is the same as the contour position in one of the most recent five iterations.

If the initial contour position (specified by `mask`) is far from the object boundaries, specify larger values of `n` to achieve desired segmentation results.

Data Types: `double`

`method` — Active contour method
`'Chan-Vese'` (default) | `'edge'`

Active contour method used for segmentation, specified as `'Chan-Vese'` or `'edge'`. The Chan-Vese region-based energy model is described in [1] on page 1-47. The edge-based model, similar to geodesic active contours, is described in [2] on page 1-47.

For RGB images, the method must be `'Chan-Vese'`.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'SmoothFactor',1.5`

`SmoothFactor` — Degree of smoothness
positive number

Degree of smoothness or regularity of the boundaries of the segmented regions, specified as the comma-separated pair consisting of `'SmoothFactor'` and a positive number. Higher values produce smoother region boundaries but can also smooth out finer details. Lower values produce more irregularities (less smoothing) in the region boundaries but allow finer details to be captured. The default smoothness value is `0` for the `'Chan-Vese'` method and `1` for the `'edge'` method.

Example: `'SmoothFactor',1.5`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`ContractionBias` — Tendency of the contour to grow outwards or shrink inwards
numeric scalar

Tendency of the contour to grow outwards or shrink inwards, specified as the comma-separated pair consisting of `'ContractionBias'` and a numeric scalar. Positive values bias the contour to shrink inwards (contract). Negative values bias the contour to grow outwards (expand). This parameter does not guarantee that the contour contracts or expands. It is possible that even with a positive value for this parameter, the contour could actually expand. However, by specifying a bias, you slow the expansion when compared to an unbiased contour. Typical values for this parameter are between -1 and 1. The default contraction bias is `0` for the `'Chan-Vese'` method and `0.3` for the `'edge'` method.

Example: `'ContractionBias',0.4`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output Arguments

`BW` — Segmented image

binary image

Segmented image, returned as a binary image of the same size as the input image `A`. The foreground is white (logical true) and the background is black (logical false).

Tips

- `activecontour` uses the boundaries of the regions in `mask` as the initial state of the contour from where the evolution starts. Holes in the mask can cause unpredictable results. Use `imfill` to fill any holes in the regions in `mask`.
- If a region touches the image borders, then `activecontour` removes a single-pixel layer from the region, before further processing, so that the region does not touch the image border.
- To get faster and more accurate results, specify an initial contour position that is close to the desired object boundaries, especially for the `'edge'` method.
- For the `'edge'` method, the active contour is naturally biased towards shrinking inwards (collapsing). In the absence of any image gradient, the active contour shrinks on its own. Conversely, with the `'Chan-Vese'` method, where the contour is unbiased, the contour is free to either shrink or expand based on the image features.
- To achieve an accurate segmentation with the `'edge'` method, specify an initial contour that lies outside the boundaries of the object. The active contour with the `'edge'` method is biased to shrink, by default.
- If object regions are of significantly different grayscale intensities, then the `'Chan-Vese'` method [1] might not segment all objects in the image. For example, if the image contains objects that are brighter than the background and some that are darker, the `'Chan-Vese'` method typically segments out either the dark or the bright objects only.

Algorithms

`activecontour` uses the Sparse-Field level-set method, similar to the method described in [3], for implementing active contour evolution.

References

[1] T. F. Chan, L. A. Vese, *Active contours without edges*. IEEE Transactions on Image Processing, Volume 10, Issue 2, pp. 266-277, 2001.

[2] V. Caselles, R. Kimmel, G. Sapiro, *Geodesic active contours*. International Journal of Computer Vision, Volume 22, Issue 1, pp. 61-79, 1997.

[3] R. T. Whitaker, *A level-set approach to 3d reconstruction from range data*. International Journal of Computer Vision, Volume 29, Issue 3, pp. 203-231, 1998.

See Also

`multithresh` | `poly2mask` | `roipoly` | `drawfreehand` | `drawellipse` | `drawpolygon` | **Image Segmenter**

Introduced in R2013a

adapthisteq

Contrast-limited adaptive histogram equalization (CLAHE)

Syntax

```
J = adapthisteq(I)
J = adapthisteq(I,Name,Value)
```

Description

`J = adapthisteq(I)` enhances the contrast of the grayscale image `I` by transforming the values using contrast-limited adaptive histogram equalization (CLAHE) [1].

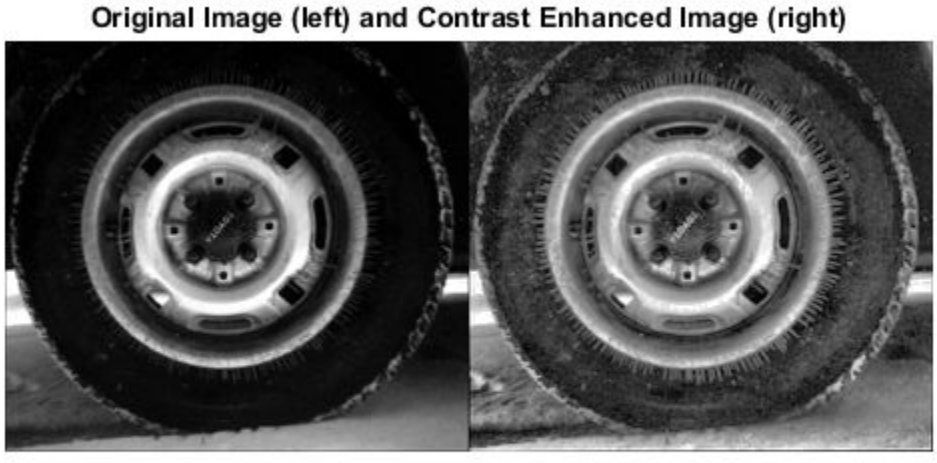
`J = adapthisteq(I,Name,Value)` uses name-value pairs to control aspects of the contrast enhancement.

Examples

Apply Contrast-Limited Adaptive Histogram Equalization (CLAHE)

Apply CLAHE to an image and display the results.

```
I = imread('tire.tif');
J = adapthisteq(I,'clipLimit',0.02,'Distribution','rayleigh');
imshowpair(I,J,'montage');
title('Original Image (left) and Contrast Enhanced Image (right)')
```



Apply CLAHE to Indexed Color Image

Read the indexed color image into the workspace.

```
[X, MAP] = imread('shadow.tif');
```

Convert the indexed image into a truecolor (RGB) image, then convert the RGB image into the L*a*b* color space.

```
RGB = ind2rgb(X,MAP);
LAB = rgb2lab(RGB);
```

Scale values to the range expected by the `adapthisteq` function, [0 1].

```
L = LAB(:,:,1)/100;
```

Perform CLAHE on the L channel. Scale the result to get back to the range used by the L*a*b* color space.

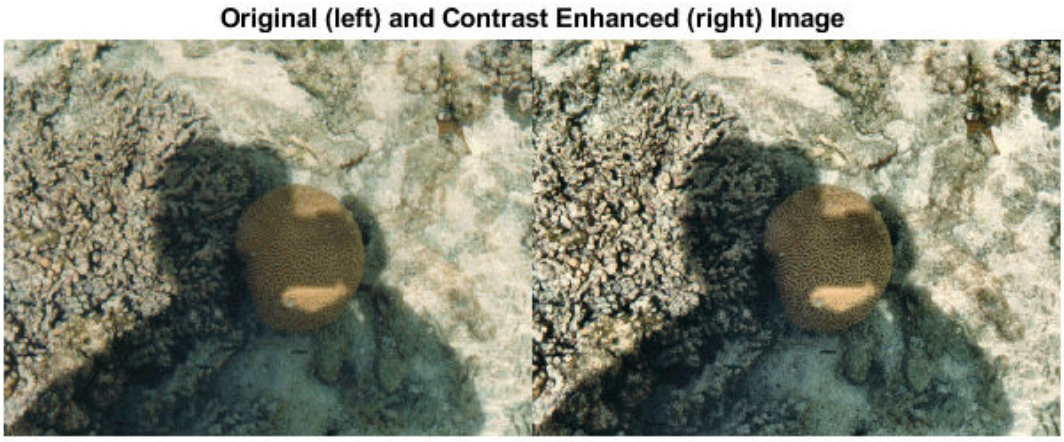
```
L = adapthisteq(L,'NumTiles',[8 8],'ClipLimit',0.005);
LAB(:,:,1) = L*100;
```

Convert the resulting image back into the RGB color space.

```
J = lab2rgb(LAB);
```

Display the original image and the processed image.

```
figure
imshowpair(RGB,J,'montage')
title('Original (left) and Contrast Enhanced (right) Image')
```



Shadows in the enhanced image look darker and highlights look brighter. The overall contrast is improved.

Input Arguments

`I` — Grayscale image

2-D numeric matrix

Grayscale image, specified as a 2-D numeric matrix.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'NumTiles',[8 16]` divides the image into 8 rows and 16 columns of tiles.

`NumTiles` — Number of tiles

`[8,8]` (default) | 2-element vector of positive integers

Number of rectangular contextual regions (tiles) into which `adapthisteq` divides the image, specified as a 2-element vector of positive integers. With the original image divided into `M` rows and `N` columns of tiles, the value of `'NumTiles'` is `[M N]`. Both `M` and `N` must be at least `2`. The total number of tiles is equal to `M*N`. The optimal number of tiles depends on the type of the input image, and it is best determined through experimentation.

Data Types: `double`

`ClipLimit` — Contrast enhancement limit

`0.01` (default) | number in the range [0, 1]

Contrast enhancement limit, specified as a number in the range [0, 1]. Higher limits result in more contrast.

`'ClipLimit'` is a contrast factor that prevents oversaturation of the image specifically in homogeneous areas. These areas are characterized by a high peak in the histogram of the particular image tile due to many pixels falling inside the same gray level range. Without the clip limit, the adaptive histogram equalization technique could produce results that, in some cases, are worse than the original image.

Data Types: `double`

`NBins` — Number of histogram bins used to build a contrast enhancing transformation

`256` (default) | positive integer

Number of histogram bins used to build a contrast enhancing transformation, specified as a positive integer. Higher values result in greater dynamic range at the cost of slower processing speed.

Data Types: `double`

Range — Range of output data

`'full'` (default) | `'original'`

Range of the output image data, specified as one of these values.

Value	Description
`'full'`	Use the full range of the output class (such as [0 255] for `uint8`).
`'original'`	Limit the range to `[min(I(:)) max(I(:))]`.

Data Types: `char` | `string`

`Distribution` — Desired histogram shape
`'uniform'` (default) | `'rayleigh'` | `'exponential'`

Desired histogram shape, specified as one of the following values:

Value	Description
`'uniform'`	Create a flat histogram.
`'rayleigh'`	Create a bell-shaped histogram.
`'exponential'`	Create a curved histogram.

`'Distribution'` specifies the distribution that `adapthisteq` uses as the basis for creating the contrast transform function. The distribution you select should depend on the type of the input image. For example, underwater imagery appears to look more natural when the Rayleigh distribution is used.

Data Types: `char` | `string`

`Alpha` — Distribution parameter
`0.4` (default) | nonnegative number

Distribution parameter, specified as a nonnegative number. `'Alpha'` is only used when `'Distribution'` is set to `'rayleigh'` or `'exponential'`.

Data Types: `double`

Output Arguments

`J` — Contrast enhanced image
2-D matrix

Contrast enhanced image, returned as a 2-D matrix of the same data type as the input image `I`.

Algorithms

CLAHE operates on small regions in the image, called *tiles*, rather than the entire image. `adapthisteq` calculates the contrast transform function for each tile individually. Each tile's contrast is enhanced, so that the histogram of the output region approximately matches the histogram specified by the `'Distribution'` value. The neighboring tiles are then combined using bilinear interpolation to eliminate artificially induced boundaries. The contrast, especially in homogeneous areas, can be limited to avoid amplifying any noise that might be present in the image.

References

[1] Zuiderveld, Karel. “Contrast Limited Adaptive Histograph Equalization.” *Graphic Gems IV*. San Diego: Academic Press Professional, 1994. 474–485.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`histeq`

Introduced before R2006a

adaptthresh

Adaptive image threshold using local first-order statistics

Syntax

```
T = adaptthresh(I)
T = adaptthresh(I,sensitivity)
T = adaptthresh( ___ ,Name,Value)
```

Description

`T = adaptthresh(I)` computes a locally adaptive threshold for 2-D grayscale image or 3-D grayscale volume `I`. The `adaptthresh` function chooses the threshold based on the local mean intensity (first-order statistics) in the neighborhood of each pixel. The threshold `T` can be used with the `imbinarize` function to convert the grayscale image to a binary image.

`T = adaptthresh(I,sensitivity)` computes a locally adaptive threshold with sensitivity factor specified by `sensitivity`. `sensitivity` is a scalar in the range [0,1] that indicates sensitivity towards thresholding more pixels as foreground.

`T = adaptthresh( ___ ,Name,Value)` computes a locally adaptive threshold using name-value pairs to control aspects of the thresholding.

Examples

Find Threshold and Segment Bright Rice Grains from Dark Background

Read image into the workspace.

```
I = imread('rice.png');
```

Use `adaptthresh` to determine threshold to use in binarization operation.

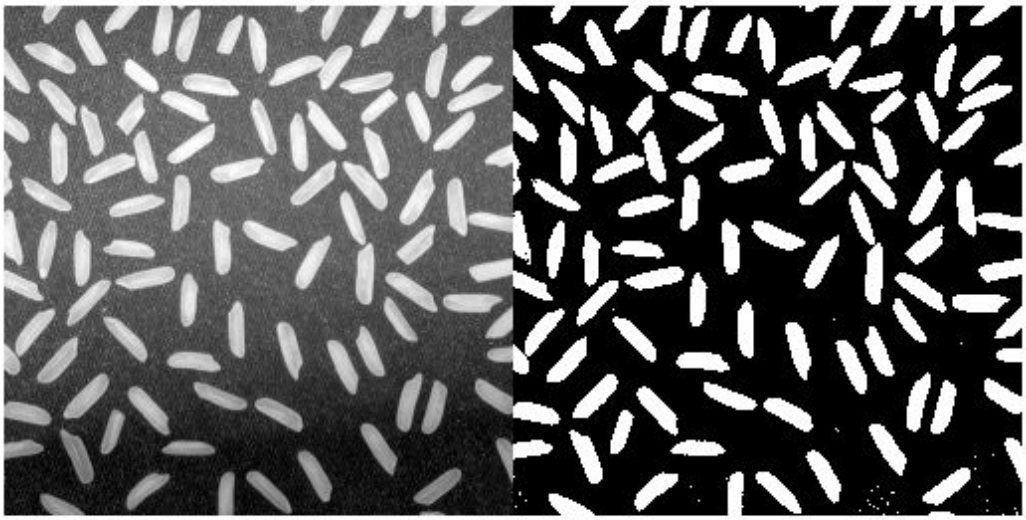
```
T = adaptthresh(I, 0.4);
```

Convert image to binary image, specifying the threshold value.

```
BW = imbinarize(I,T);
```

Display the original image with the binary version, side-by-side.

```
figure
imshowpair(I, BW, 'montage')
```



Find Threshold and Segment Dark Text from Bright Background

Read image into the workspace.

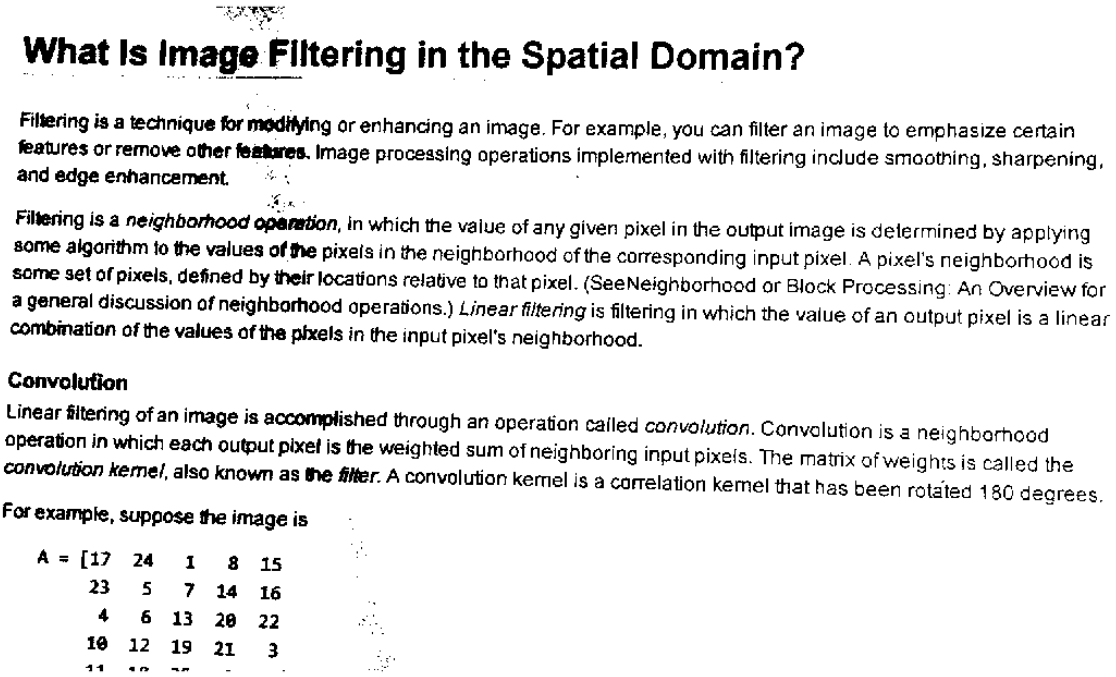
```
I = imread('printedtext.png');
```

Using `adaptthresh` compute adaptive threshold and display the local threshold image. This represents an estimate of average background illumination.

```
T = adaptthresh(I,0.4,'ForegroundPolarity','dark');
figure
imshow(T)
```

Binarize image using locally adaptive threshold

```
BW = imbinarize(I,T);
figure
imshow(BW)
```



What Is Image Filtering in the Spatial Domain?

Filtering is a technique for modifying or enhancing an image. For example, you can filter an image to emphasize certain features or remove other features. Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.

Filtering is a *neighborhood operation*, in which the value of any given pixel in the output image is determined by applying some algorithm to the values of the pixels in the neighborhood of the corresponding input pixel. A pixel's neighborhood is some set of pixels, defined by their locations relative to that pixel. (SeeNeighborhood or Block Processing: An Overview for a general discussion of neighborhood operations.) *Linear filtering* is filtering in which the value of an output pixel is a linear combination of the values of the pixels in the input pixel's neighborhood.

Convolution

Linear filtering of an image is accomplished through an operation called *convolution*. Convolution is a neighborhood operation in which each output pixel is the weighted sum of neighboring input pixels. The matrix of weights is called the *convolution kernel*, also known as the *filter*. A convolution kernel is a correlation kernel that has been rotated 180 degrees.

For example, suppose the image is

```
A = [17  24   1   8  15
     23   5   7  14  16
      4   6  13  20  22
     10  12  19  21   3
```

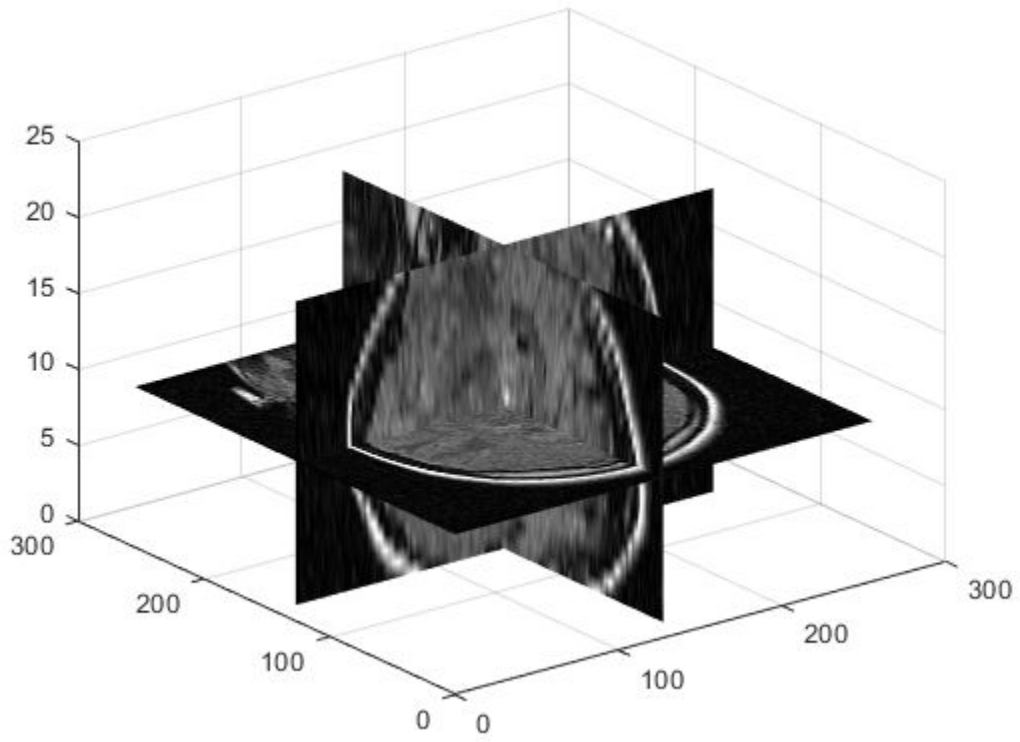
Calculate Threshold for 3-D Volume

Load 3-D volume into the workspace.

```
load mristack;
V = mristack;
```

Display the data.

```
figure
slice(double(V),size(V,2)/2,size(V,1)/2,size(V,3)/2)
colormap gray
shading interp
```

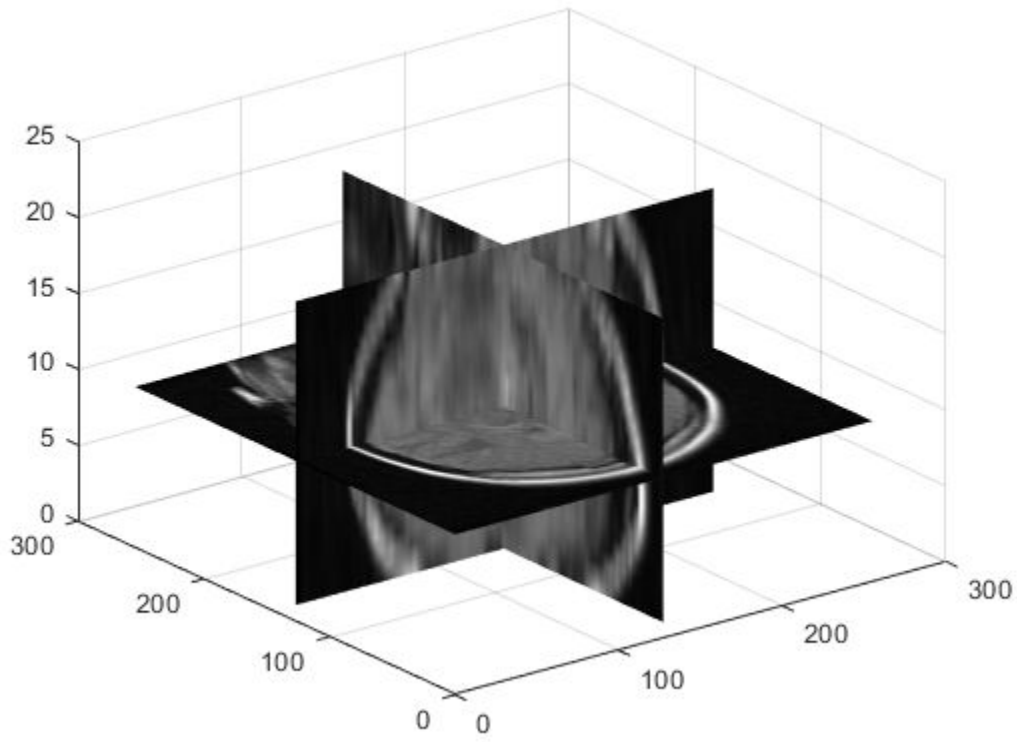


Calculate the threshold.

```
J = adaptthresh(V,'neigh',[3 3 3],'Fore','bright');
```

Display the threshold.

```
figure
slice(double(J),size(J,2)/2,size(J,1)/2,size(J,3)/2)
colormap gray
shading interp
```



Input Arguments

`I` — Grayscale image or volume

2-D numeric matrix | 3-D numeric array

Grayscale image or volume, specified as a 2-D numeric matrix or 3-D numeric array.

If the image contains `Inf`s or `NaN`s, the behavior of `adaptthresh` is undefined. Propagation of `Inf`s or `NaN`s might not be localized to the neighborhood around `Inf` or `NaN` pixels.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`sensitivity` — Determine which pixels get thresholded as foreground pixels

`0.5` (default) | number in the range [0, 1]

Determine which pixels get thresholded as foreground pixels, specified as a number in the range [0, 1]. High sensitivity values lead to thresholding more pixels as foreground, at the risk of including some background pixels.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `T = adaptthresh(I,0.4,'ForegroundPolarity','dark');`

`NeighborhoodSize` — Size of neighborhood used to compute local statistic around each pixel
`2*floor(size(I)/16)+1` (default) | positive odd integer | 2-element vector of positive odd integers

Size of neighborhood used to compute local statistic around each pixel, specified as a positive odd integer or a 2-element vector of positive odd integers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ForegroundPolarity` — Determine which pixels are considered foreground pixels
`'bright'` (default) | `'dark'`

Determine which pixels are considered foreground pixels, specified using one of the following:

Value	Meaning
`'bright'`	The foreground is brighter than the background.
`'dark'`	The foreground is darker than the background

Data Types: `char` | `string`

`Statistic` — Statistic used to compute local threshold
`'mean'` (default) | `'median'` | `'gaussian'`

Statistic used to compute local threshold at each pixel, specified as one of the following:

Value	Meaning
`'mean'`	The local mean intensity in the neighborhood. This technique is also called Bradley's method [1].
`'median'`	The local median in the neighborhood. Computation of this statistic can be slow. Consider using a smaller neighborhood size to obtain faster results.
`'gaussian'`	The Gaussian weighted mean in the neighborhood.

Data Types: `char` | `string`

Output Arguments

`T` — Normalized intensity values
numeric matrix | numeric array

Normalized intensity values, returned as a numeric matrix or numeric array of the same size as the input image or volume, `I`. Values are normalized to the range [0, 1].

Data Types: `double`

References

[1] Bradley, D., G. Roth, "Adapting Thresholding Using the Integral Image," *Journal of Graphics Tools*. Vol. 12, No. 2, 2007, pp.13–21.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `adaptthresh` supports the generation of C code (requires MATLAB Coder™). Note that if you choose the generic `MATLAB Host Computer` target platform, `adaptthresh` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- The `ForegroundPolarity` and `Statistic` arguments must be compile-time constants.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

`imbinarize` | `otsuthresh` | `graythresh`

Introduced in R2016a

addPix2PixHDLocalEnhancer

Add local enhancer network to pix2pixHD generator network

Syntax

```
netWithEnhancer = addPix2PixHDLocalEnhancer(net)
netWithEnhancer = addPix2PixHDLocalEnhancer(net,Name,Value)
```

Description

`netWithEnhancer = addPix2PixHDLocalEnhancer(net)` adds a local enhancer network to a pix2pixHD generator network, `net`. For more information about the network architecture, see “pix2pixHD Local Enhancer Network” on page 1-66.

This function requires Deep Learning Toolbox™.

`netWithEnhancer = addPix2PixHDLocalEnhancer(net,Name,Value)` controls aspects of the local enhancer network creation using name-value arguments.

Examples

Add Local Enhancer to Pix2pixHD Generator Network

Specify the network input size for 32-channel data of size 512-by-1024.

```
inputSize = [512 1024 32];
```

Create a pix2pixHD global generator network.

```
pix2pixHD = pix2pixHDGlobalGenerator(inputSize)
```

```
pix2pixHD =
  dlnetwork with properties:

         Layers: [84x1 nnet.cnn.layer.Layer]
    Connections: [92x2 table]
     Learnables: [110x3 table]
          State: [0x3 table]
     InputNames: {'GlobalGenerator_inputLayer'}
    OutputNames: {'GlobalGenerator_fActivation'}
    Initialized: 1
```

Add a local enhancer network to the pix2pixHD network.

```
pix2pixHDEnhanced = addPix2PixHDLocalEnhancer(pix2pixHD)
```

```
pix2pixHDEnhanced =
  dlnetwork with properties:

         Layers: [113x1 nnet.cnn.layer.Layer]
```

```
    Connections: [124x2 table]
     Learnables: [146x3 table]
          State: [0x3 table]
     InputNames: {'LocalEnhancer_inputLayer'  'GlobalGenerator_inputLayer'}
    OutputNames: {'LocalEnhancer_fActivation'}
    Initialized: 1
```

Display the network with the local enhancer.

```
analyzeNetwork(pix2pixHDEnhanced)
```

Input Arguments

`net` — pix2pixHD generator network

`dlnetwork` object

Pix2pixHD generator network, specified as a `dlnetwork` object. You can create a pix2pixHD generator network using the `pix2pixHDGlobalGenerator` function.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'FilterSizeInFirstAndLastBlocks',[5 7]` adds a local enhancer whose first and last convolution layers have a size of 5-by-7

`FilterSizeInFirstAndLastBlocks` — Filter size in first and last convolution layers

7 (default) | positive odd integer | 2-element vector of positive odd integers

Filter size in the first and last convolution layers of the local enhancer network, specified as a positive odd integer or 2-element vector of positive odd integers of the form [*height width*]. When you specify the filter size as a scalar, the filter has equal height and width.

`FilterSizeInIntermediateBlocks` — Filter size in intermediate convolution layers

3 (default) | 2-element vector of positive odd integers | positive odd integer

Filter size in intermediate convolution layers in the local enhancer network, specified as a positive odd integer or 2-element vector of positive odd integers of the form [*height width*]. The intermediate convolution layers are the convolution layers excluding the first and last convolution layer. When you specify the filter size as a scalar, the filter has identical height and width. Typical values are between 3 and 7.

`NumResidualBlocks` — Number of residual blocks

3 (default) | positive integer

Number of residual blocks in the local enhancer network, specified as a positive integer. Each residual block consists of a set of convolution, normalization and nonlinear layers with skip connections between every block.

`ConvolutionPaddingValue` — Style of padding
`"symmetric-exclude-edge"` (default) | `"symmetric-include-edge"` | `"replicate"` | numeric scalar

Style of padding used in the local enhancer network, specified as one of these values.

PaddingValue	Description	Example
Numeric scalar	Pad with the specified numeric value	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}2&2&2&2&2&2&2\\2&2&2&2&2&2&2\\2&2&3&1&4&2&2\\2&2&1&5&9&2&2\\2&2&2&6&5&2&2\\2&2&2&2&2&2&2\\2&2&2&2&2&2&2\end{bmatrix}$
`'symmetric-include-edge'`	Pad using mirrored values of the input, including the edge values	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}5&1&1&5&9&9&5\\1&3&3&1&4&4&1\\1&3&3&1&4&4&1\\5&1&1&5&9&9&5\\6&2&2&6&5&5&6\\6&2&2&6&5&5&6\\5&1&1&5&9&9&5\end{bmatrix}$
`'symmetric-exclude-edge'`	Pad using mirrored values of the input, excluding the edge values	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}5&6&2&6&5&6&2\\9&5&1&5&9&5&1\\4&1&3&1&4&1&3\\9&5&1&5&9&5&1\\5&6&2&6&5&6&2\\9&5&1&5&9&5&1\\4&1&3&1&4&1&3\end{bmatrix}$
`'replicate'`	Pad using repeated border elements of the input	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}3&3&3&1&4&4&4\\3&3&3&1&4&4&4\\3&3&3&1&4&4&4\\1&1&1&5&9&9&9\\2&2&2&6&5&5&5\\2&2&2&6&5&5&5\\2&2&2&6&5&5&5\end{bmatrix}$

`UpsampleMethod` — Method used to upsample activations
`"transposedConv"` (default) | `"bilinearResize"` | `"pixelShuffle"`

Method used to upsample activations in the local enhancer network, specified as one of these values:

- `"transposedConv"` — Use a `transposedConv2dLayer` with a stride of [2 2]
- `"bilinearResize"` — Use a `convolution2dLayer` with a stride of [1 1] followed by a `resize2dLayer` with a scale of [2 2]

- `"pixelShuffle"` — Use a `convolution2dLayer` with a stride of [1 1] followed by a `depthToSpace2dLayer` with a block size of [2 2]

Data Types: `char` | `string`

`ConvolutionWeightsInitializer` — Weight initialization used in convolution layers
`"narrow-normal"` (default) | `"glorot"` | `"he"` | function

Weight initialization used in convolution layers of the local enhancer network, specified as `"glorot"`, `"he"`, `"narrow-normal"`, or a function handle. For more information, see “Specify Custom Weight Initialization Function” (Deep Learning Toolbox).

`ActivationLayer` — Activation function
`"relu"` (default) | `"leakyRelu"` | `"elu"` | layer object

Activation function to use in the local enhancer network, specified as one of these values. For more information and a list of available layers, see “Activation Layers” (Deep Learning Toolbox).

- `"relu"` — Use a `reluLayer`
- `"leakyRelu"` — Use a `leakyReluLayer` with a scale factor of 0.2
- `"elu"` — Use an `eluLayer`
- A layer object

`NormalizationLayer` — Normalization operation
`"instance"` (default) | `"none"` | `"batch"` | layer object

Normalization operation to use after each convolution in the local enhancer network, specified as one of these values. For more information and a list of available layers, see “Normalization, Dropout, and Cropping Layers” (Deep Learning Toolbox).

- `"instance"` — Use an `instanceNormalizationLayer`
- `"batch"` — Use a `batchNormalizationLayer`
- `"none"` — Do not use a normalization layer
- A layer object

`Dropout` — Probability of dropout
`0` (default) | number in the range [0, 1]

Probability of dropout in the local enhancer network, specified as a number in the range [0, 1]. If you specify a value of `0`, then the network does not include dropout layers. If you specify a value greater than `0`, then the network includes a `dropoutLayer` in each residual block.

`NamePrefix` — Prefix to all layer names
`"LocalEnhancer_"` (default) | string | character vector

Prefix to all layer names in the local enhancer network, specified as a string or character vector.

Data Types: `char` | `string`

Output Arguments

`netWithEnhancer` — pix2pixHD generator network with local enhancer
`dlnetwork` object

Pix2pixHD generator network with local enhancer, returned as a `dlnetwork` object.

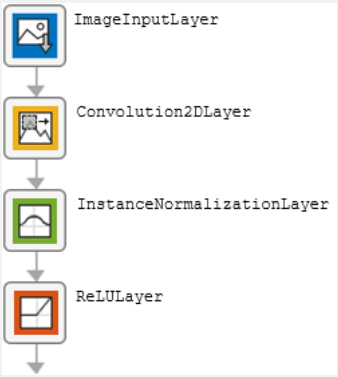
More About

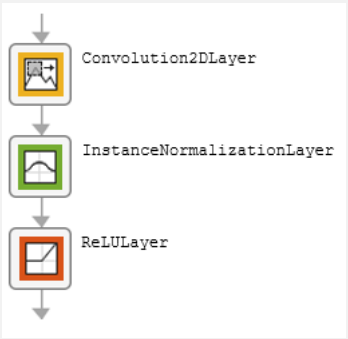
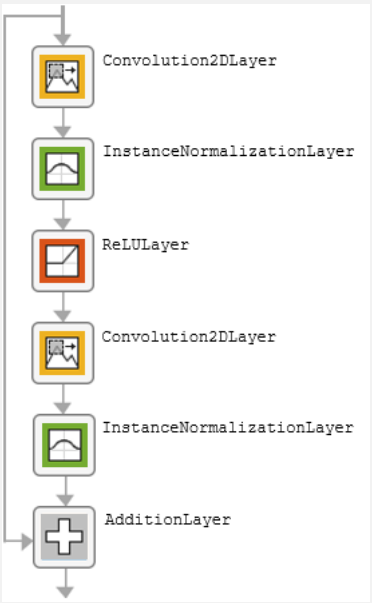
pix2pixHD Local Enhancer Network

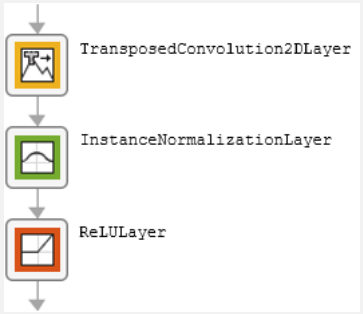
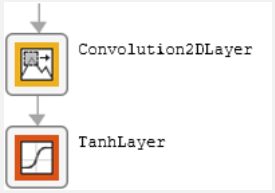
The `addPix2PixHDLocalEnhancer` function performs these operations to add a local enhancer network to a pix2pixHD global generator network. The default enhanced network follows the architecture proposed by Wang et. al. "References" on page 1-68.

1. The local enhancer network has an initial block of layers that accepts images of size [2*H 2*W C], where H is the height, W is the width, and C is the number of channels of the input to the global generator network, `net`. When `net` has multiple image input layers, the input image size of the local enhancer network is twice the input size with the maximum resolution.
2. After the initial block, the local enhancer network has a single downsampling block that downsamples the data by a factor of two. Therefore, the output after downsampling has size [H W 2*C].
3. The `addPix2PixHDLocalEnhancer` function trims the final block from the global generator network. The function then adds the output of the last upsampling block in the global generator network to the output of the downsampled data from the enhancer network using an `additionLayer`.
4. The output of the addition then passes through `NumResidualBlocks` residual blocks from the local enhancer.
5. The residual blocks are followed by a single upsampling block that upsamples data to size [2*H 2*W C].
6. The `addPix2PixHDLocalEnhancer` function adds a final block to the enhanced network. The convolution layer has properties specified by the arguments of `addPix2PixHDLocalEnhancer`. If the global generator network has a final activation layer, then the function adds the same type of activation layer to the enhanced network.

The table describes the blocks of layers that comprise the local enhancer network.

Block Type	Layers	Diagram of Default Block
Initial block	• An `imageInputLayer` that accepts images of twice the size as pix2pixHD global generator network, `net`. • A `convolution2dLayer` with a stride of [1 1] and a filter size of `FilterSizeInFirstAndLastBlocks` • An optional normalization layer, specified by the `NormalizationLayer` name-value argument. • An activation layer specified by the `ActivationLayer` name-value argument.	

Block Type	Layers	Diagram of Default Block
Downsampling block	• A `convolution2dLayer` with a stride of [2 2] to perform downsampling. The convolution layer has a filter size of `FilterSizeInIntermediateBlocks`. • An optional normalization layer, specified by the `NormalizationLayer` name-value argument. • An activation layer specified by the `ActivationLayer` name-value argument.	
Residual block	• A `convolution2dLayer` with a stride of [1 1] and a filter size of `FilterSizeInIntermediateBlocks`. • An optional normalization layer, specified by the `NormalizationLayer` name-value argument. • An activation layer specified by the `ActivationLayer` name-value argument. • An optional `dropoutLayer`. By default, residual blocks omit a dropout layer. Include a dropout layer by specifying the `Dropout` name-value argument as a value in the range (0, 1]. • A second `convolution2dLayer`. • An optional second normalization layer. • An `additionLayer` that provides a skip connection between every block.	

Block Type	Layers	Diagram of Default Block
Upsampling block	• An upsampling layer that upsamples by a factor of 2 according to the `UpsampleMethod` name-value argument. The convolution layer has a filter size of `FilterSizeInIntermediateBlocks`. • An optional normalization layer, specified by the `NormalizationLayer` name-value argument. • An activation layer specified by the `ActivationLayer` name-value argument.	
Final block	• A `convolution2dLayer` with a stride of [1 1] and a filter size of `FilterSizeInFirstAndLastBlocks`. • An optional activation layer according to the global generator network, `net`.	

References

[1] Wang, Ting-Chun, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. "High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs." In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 8798–8807. Salt Lake City, UT, USA: IEEE, 2018. https://doi.org/10.1109/CVPR.2018.00917.

See Also

`pix2pixHDGlobalGenerator`

Topics

“Get Started with GANs for Image-to-Image Translation”
“Create Modular Neural Networks”
“List of Deep Learning Layers” (Deep Learning Toolbox)

Introduced in R2021a

affine2d

2-D affine geometric transformation

Description

An `affine2d` object stores information about a 2-D affine geometric transformation and enables forward and inverse transformations.

Creation

You can create an `affine2d` object in these ways:

- `imregtform` — Estimate a geometric transformation that maps a moving image to a fixed image using similarity optimization.
- `imregcorr` — Estimate a geometric transformation that maps a moving image to a fixed image using phase correlation.
- `fitgeotrans` — Estimate a geometric transformation that maps pairs of control points between two images.
- `randomAffine2d` — Create a randomized 2-D affine transformation.
- The `affine2d` function described here.

Syntax

```
tform = affine2d
tform = affine2d(T)
```

Description

`tform = affine2d` creates an `affine2d` object with default property settings that correspond to the identity transformation.

`tform = affine2d(T)` sets the property `T` as the specified valid affine transformation matrix.

Properties

T — Forward 2-D affine transformation

nonsingular 3-by-3 numeric matrix

Forward 2-D affine transformation, specified as a nonsingular 3-by-3 numeric matrix.

The matrix `T` uses the convention:

```
[x y 1] = [u v 1] * T
```

where `T` has the form:

```
[a b 0;
 c d 0;
 e f 1];
```

The default of T is the identity transformation.

Data Types: `double` | `single`

`Dimensionality` — Dimensionality of geometric transformation
2

Dimensionality of the geometric transformation for both input and output points, specified as the value 2.

Object Functions

invert	Invert geometric transformation
isRigid	Determine if transformation is rigid transformation
isSimilarity	Determine if transformation is similarity transformation
isTranslation	Determine if transformation is pure translation
outputLimits	Find output spatial limits given input spatial limits
transformPointsForward	Apply forward geometric transformation
transformPointsInverse	Apply inverse geometric transformation

Examples

Define 2-D Affine Transformation from Transformation Matrix

Create an `affine2d` object that defines a 30 degree rotation in the counterclockwise direction around the origin.

```
theta = 30;
tform = affine2d([ ...
    cosd(theta) sind(theta) 0;...
    -sind(theta) cosd(theta) 0; ...
    0 0 1])
```

```
tform =
  affine2d with properties:

                T: [3x3 double]
    Dimensionality: 2
```

Apply the forward geometric transformation to a point (10,0).

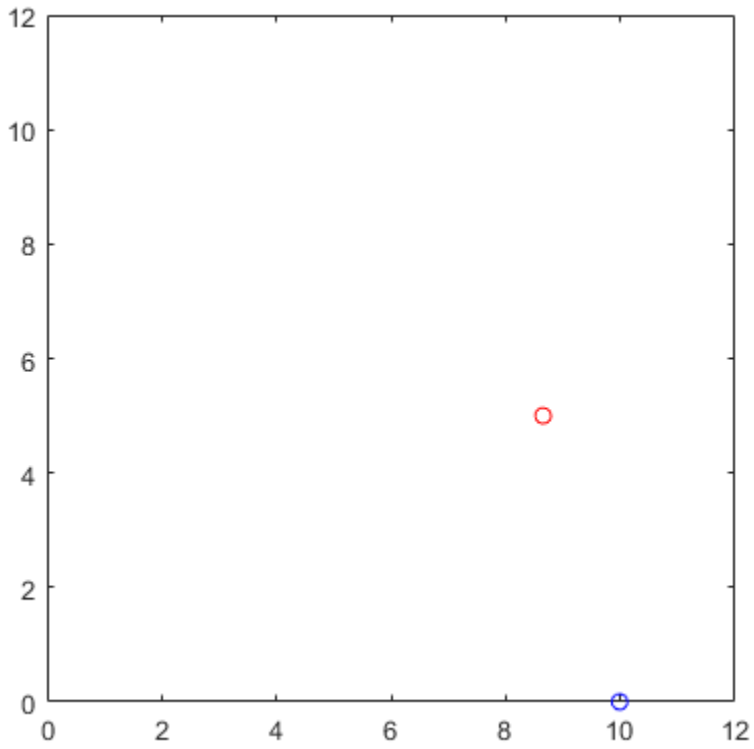
```
[x,y] = transformPointsForward(tform,10,0)
```

```
x = 8.6603
```

```
y = 5
```

Validate the transformation by plotting the original point (in blue) and the transformed point (in red).

```
plot(10,0,'bo',x,y,'ro')
axis([0 12 0 12])
axis square
```



Transform Image Using 2-D Affine Transformation Object

Read an image into the workspace.

```
A = imread('pout.tif');
```

Create an `affine2d` object that defines an affine geometric transformation. This example combines vertical shear and horizontal stretch.

```
tform = affine2d([2 0.33 0; 0 1 0; 0 0 1])
```

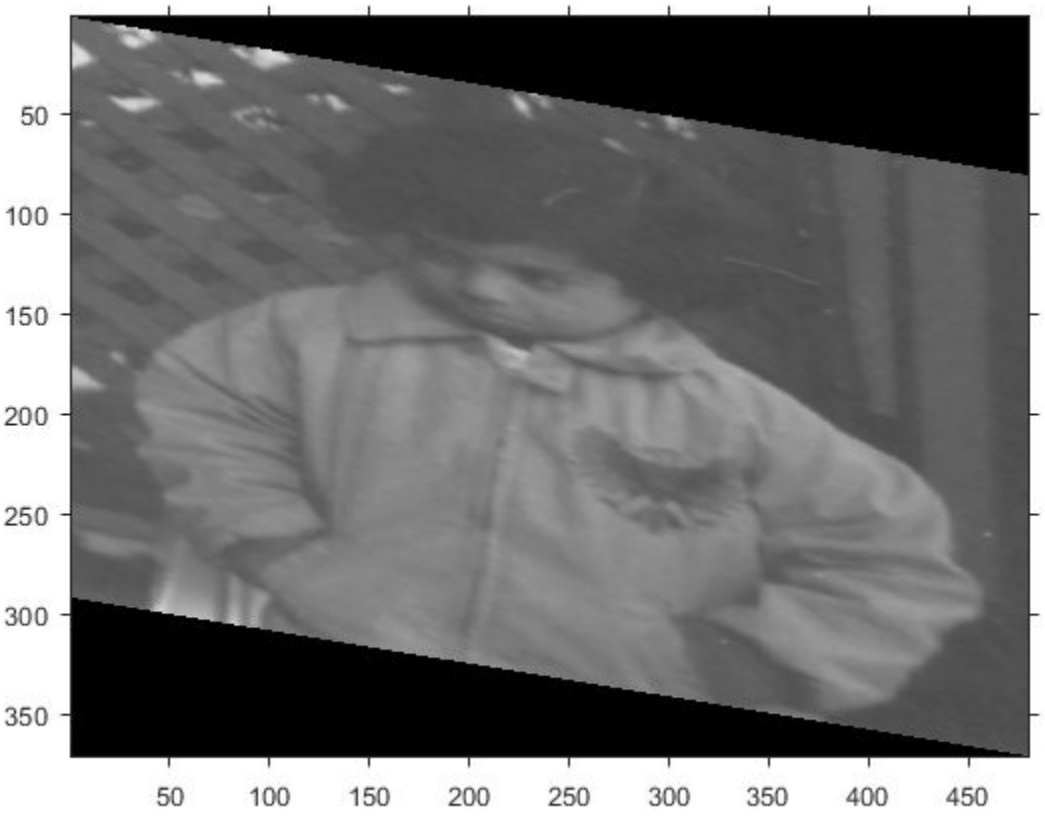
```
tform =
  affine2d with properties:

                T: [3x3 double]
    Dimensionality: 2
```

Apply the geometric transformation to the image using `imwarp`.

```
B = imwarp(A,tform);
```

Display the resulting image.

```
figure
imshow(B);
axis on equal;
```



Randomly Rotate Image

Read and display an image.

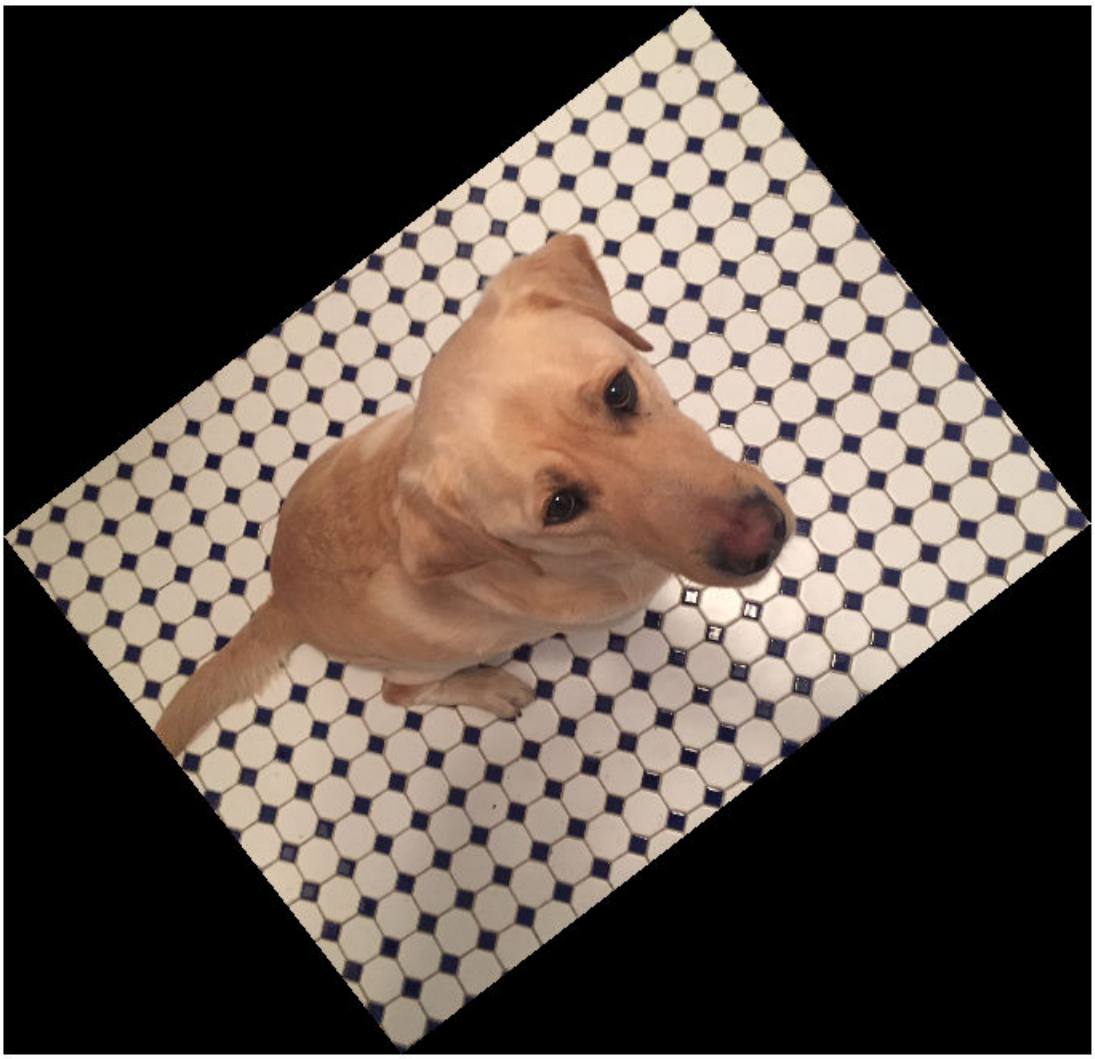
```
I = imread('kobi.png');
imshow(I)
```

Create an `affine2d` transformation object that rotates images. The `randomAffine2d` function picks a rotation angle randomly from a continuous uniform distribution within the interval [35, 55] degrees.

```
tform1 = randomAffine2d('Rotation',[35 55]);
```

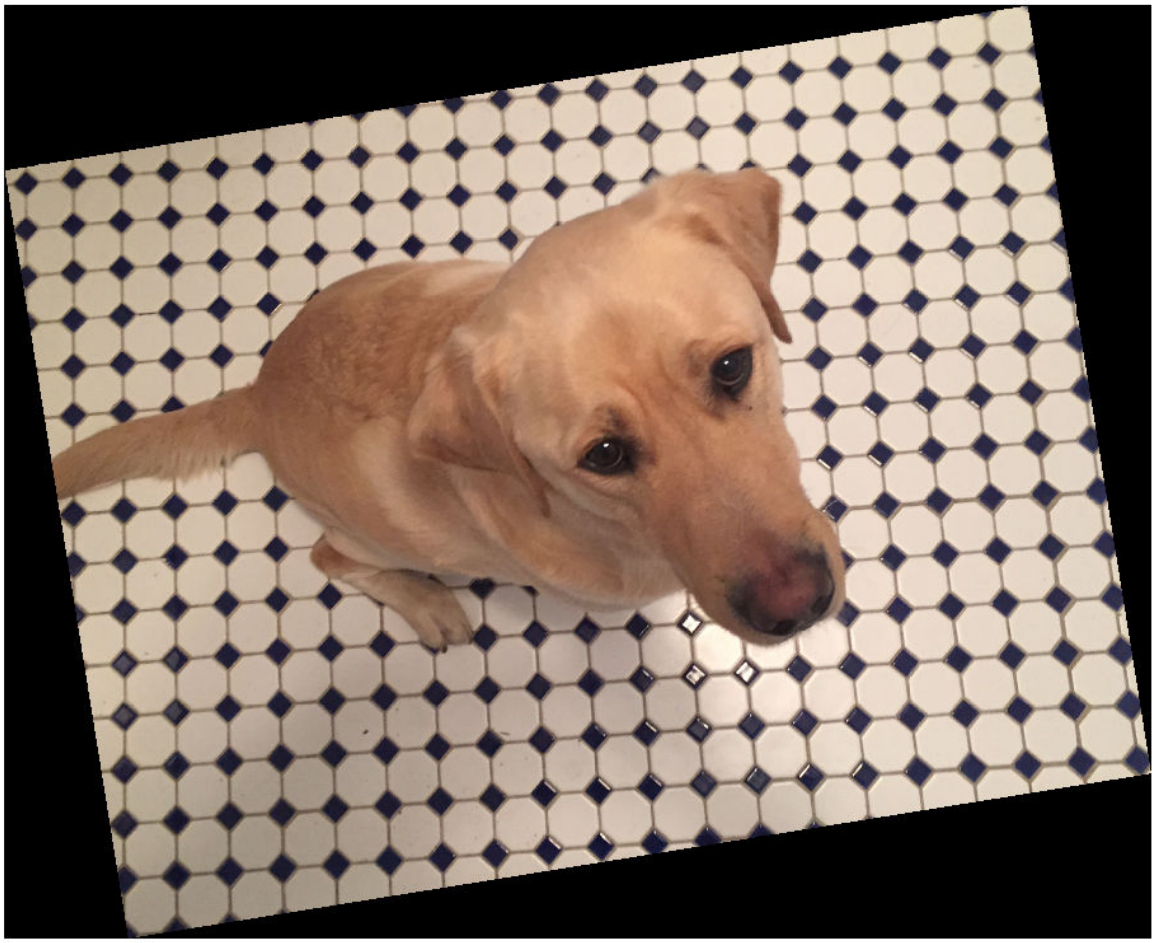
Rotate the image and display the result.

```
J = imwarp(I,tform1);
imshow(J)
```



The transformation object, `tform1`, rotates all images by the same amount. To rotate an image by a different randomly selected amount, create a new `affine2d` transformation object.

```
tform2 = randomAffine2d('Rotation',[-10 10]);
J2 = imwarp(I,tform2);
imshow(J2)
```

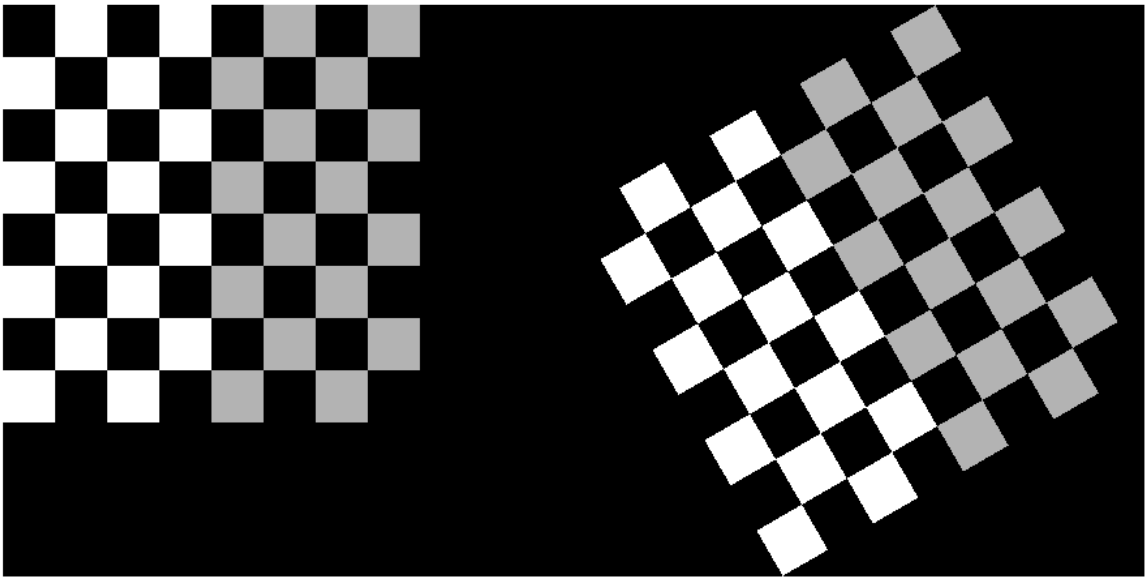


Create Geometric Transformation for Image Alignment

This example shows how to create a geometric transformation that can be used to align two images.

Create a checkerboard image and rotate it to create a misaligned image.

```
I = checkerboard(40);
J = imrotate(I,30);
imshowpair(I,J,'montage')
```



Define some matching control points on the fixed image (the checkerboard) and moving image (the rotated checkerboard). You can define points interactively using the Control Point Selection tool.

```
fixedPoints = [41 41; 281 161];
movingPoints = [56 175; 324 160];
```

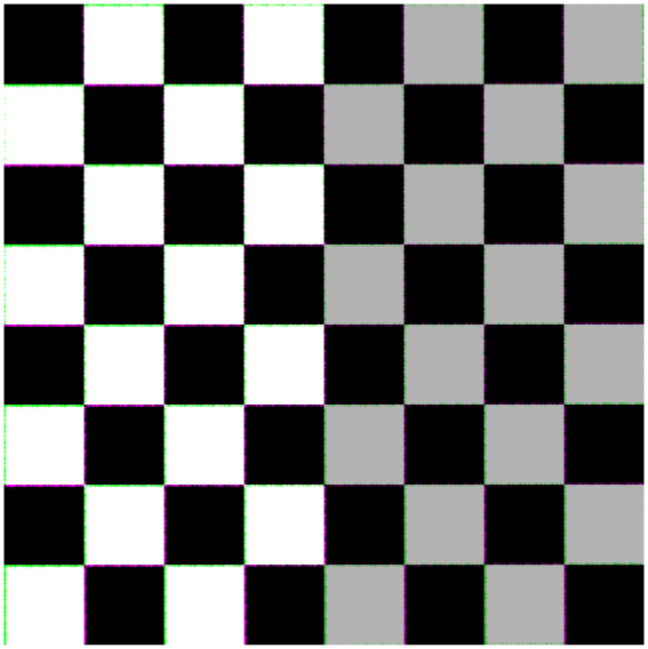
Create a geometric transformation that can be used to align the two images, returned as an `affine2d` geometric transformation object.

```
tform = fitgeotrans(movingPoints,fixedPoints,'NonreflectiveSimilarity')
```

```
tform =
  affine2d with properties:

                T: [3x3 double]
    Dimensionality: 2
```

Use the `tform` estimate to resample the rotated image to register it with the fixed image. The regions of color (green and magenta) in the false color overlay image indicate error in the registration. This error comes from a lack of precise correspondence in the control points.

```
Jregistered = imwarp(J,tform,'OutputView',imref2d(size(I)));
figure
imshowpair(I,Jregistered)
```



Recover angle and scale of the transformation by checking how a unit vector parallel to the x-axis is rotated and stretched.

```
u = [0 1];
v = [0 0];
[x, y] = transformPointsForward(tform, u, v);
dx = x(2) - x(1);
dy = y(2) - y(1);
angle = (180/pi) * atan2(dy, dx)
```

```
angle = 29.7686
```

```
scale = 1 / sqrt(dx^2 + dy^2)
```

```
scale = 1.0003
```

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `affine2d` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

- When generating code, you can only specify singular objects—arrays of objects are not supported.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, you can only specify singular objects—arrays of objects are not supported.

Thread-Based Environment
Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Functions
`imwarp` | `fitgeotrans` | `imregtform` | `imregister` | `imregcorr`

Objects
`affine3d` | `rigid2d` | `projective2d` | `geometricTransform2d` | `LocalWeightedMeanTransformation2D` | `PiecewiseLinearTransformation2D` | `PolynomialTransformation2D`

Topics
“2-D and 3-D Geometric Transformation Process Overview”
“Matrix Representation of Geometric Transformations”

Introduced in R2013a

affine3d

3-D affine geometric transformation

Description

An `affine3d` object stores information about a 3-D affine geometric transformation and enables forward and inverse transformations.

Creation

You can create an `affine3d` object using the following methods:

- `imregtform` — Estimates a geometric transformation that maps a moving image to a fixed image using similarity optimization
- `randomAffine3d` — Creates a randomized 3-D affine transformation
- The `affine3d` function described here

Syntax

```
tform = affine3d
tform = affine3d(A)
```

Description

`tform = affine3d` creates an `affine3d` object with default property settings that correspond to the identity transformation.

`tform = affine3d(A)` sets the property `T` with a valid affine transformation defined by nonsingular matrix `A`.

Properties

T — Forward 3-D affine transformation

nonsingular 4-by-4 numeric matrix

Forward 3-D affine transformation, specified as a nonsingular 4-by-4 numeric matrix.

The matrix `T` uses the convention:

```
[x y z 1] = [u v w 1] * T
```

where `T` has the form:

```
 [a b c 0;
  d e f 0;
  g h i 0;
  j k l 1];
```

The default of T is the identity transformation.

Data Types: `double` | `single`

`Dimensionality` — Describes the dimensionality of the geometric transformation
3

Describes the dimensionality of the geometric transformation for both input and output points, specified as the value 3.

Object Functions

invert	Invert geometric transformation
isRigid	Determine if transformation is rigid transformation
isSimilarity	Determine if transformation is similarity transformation
isTranslation	Determine if transformation is pure translation
outputLimits	Find output spatial limits given input spatial limits
transformPointsForward	Apply forward geometric transformation
transformPointsInverse	Apply inverse geometric transformation

Examples

Define 3-D Affine Transformation Object for Anisotropic Scaling

Create an `affine3d` object that scales a 3-D image by a different factor in each dimension.

```
Sx = 1.2;
Sy = 1.6;
Sz = 2.4;
tform = affine3d([Sx 0 0 0; 0 Sy 0 0; 0 0 Sz 0; 0 0 0 1])
```

```
tform =
  affine3d with properties:

                T: [4x4 double]
    Dimensionality: 3
```

Load a 3-D volume into the workspace.

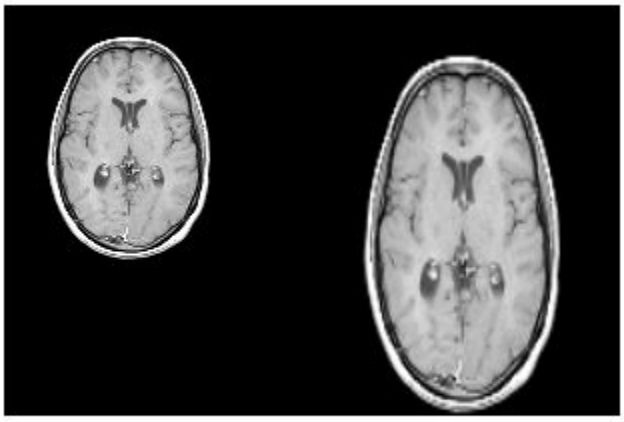
```
load('mri');
D = squeeze(D);
```

Apply the geometric transformation to the image using `imwarp`.

```
B = imwarp(D,tform);
```

Visualize an axial slice through the center of each volume to see the effect of scale translation. Note that the center slice of the transformed volume has a different index than the center slice of the original volume because of the scaling in the *z*-dimension.

```
figure
imshowpair(D(:,:,14),B(:,:,33),'montage');
```



The original image is on the left, and the transformed image is on the right. The transformed image is scaled more in the vertical direction than in the horizontal direction, as expected since `Sy` is larger than `Sx`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `affine3d` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- When generating code, you can only specify singular objects—arrays of objects are not supported.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Functions

`imwarp` | `imregtform` | `imregister`

Objects

`affine2d` | `rigid3d` | `geometricTransform3d`

Topics

“2-D and 3-D Geometric Transformation Process Overview”

“Matrix Representation of Geometric Transformations”

Introduced in R2013a

affineOutputView

Create output view for warping images

Syntax

```
Rout = affineOutputView(sizeA,tform)
Rout = affineOutputView(sizeA,tform,'BoundsStyle',style)
```

Description

`Rout = affineOutputView(sizeA,tform)` takes the size of an input image, `sizeA`, and an affine geometric transformation, `tform`, and returns a spatial referencing object, `Rout`. You can use this object as input to `imwarp` to control the output limits and grid spacing of a warped image.

`Rout = affineOutputView(sizeA,tform,'BoundsStyle',style)` also specifies constraints on the spatial limits of the output view, such as whether the output view should completely contain the output image or whether the output view should match the input limits.

Examples

Warp Image Using Different Output View Styles

Read and display an image. To see the spatial extents of the image, make the axes visible.

```
A = imread('kobi.png');
iptsetpref('ImshowAxesVisible','on')
imshow(A)
```

Create a 2-D affine transformation. This example creates a randomized transformation that consists of scale by a factor in the range [1.2, 2.4], rotation by an angle in the range [-45, 45] degrees, and horizontal translation by a distance in the range [100, 200] pixels.

```
tform = randomAffine2d('Scale',[1.2,2.4],'XTranslation',[100 200],'Rotation',[-45,45]);
```

Create three different output views for the image and transformation.

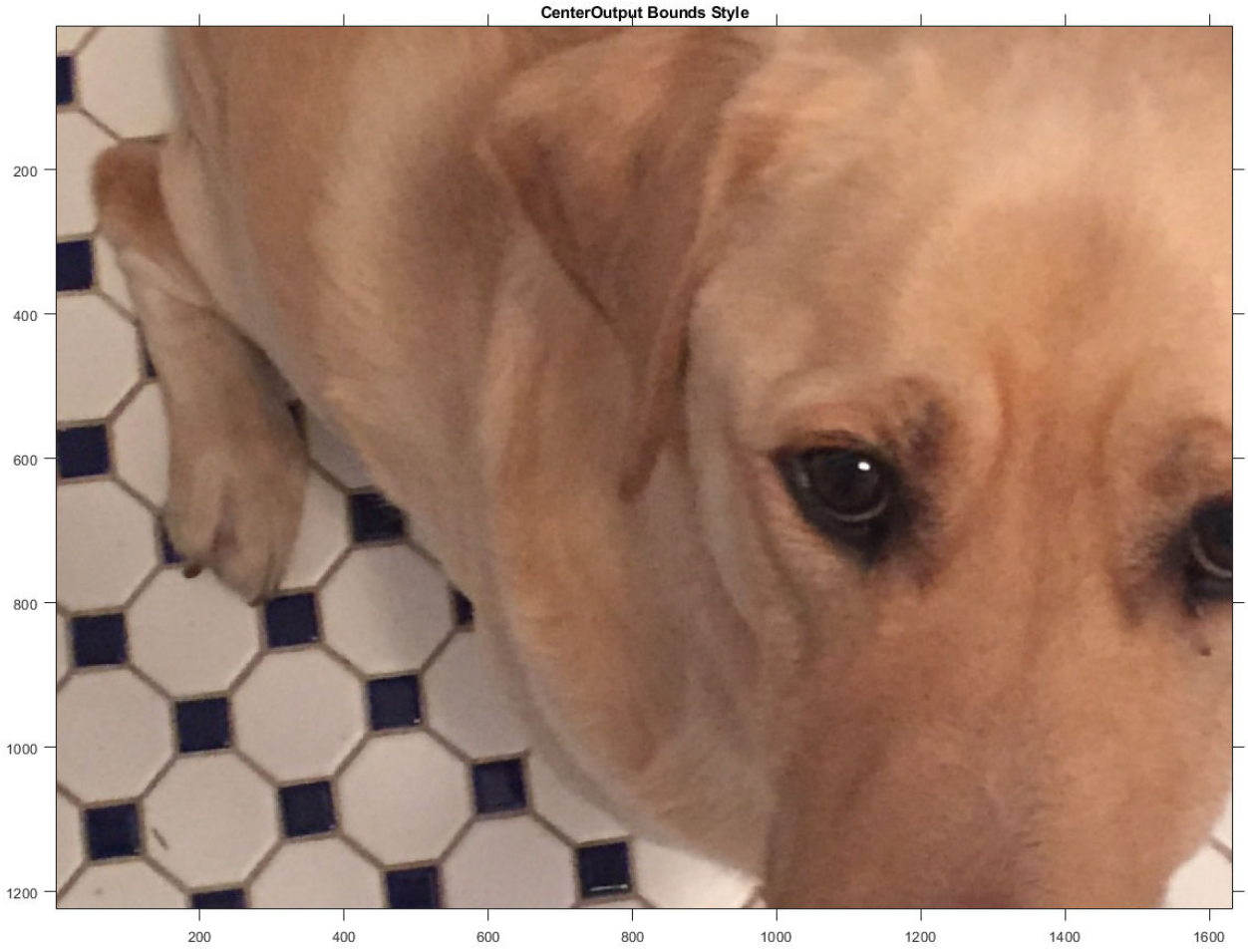
```
centerOutput = affineOutputView(size(A),tform,'BoundsStyle','CenterOutput');
followOutput = affineOutputView(size(A),tform,'BoundsStyle','FollowOutput');
sameAsInput = affineOutputView(size(A),tform,'BoundsStyle','SameAsInput');
```

Apply the transformation to the input image using each of the different output view styles.

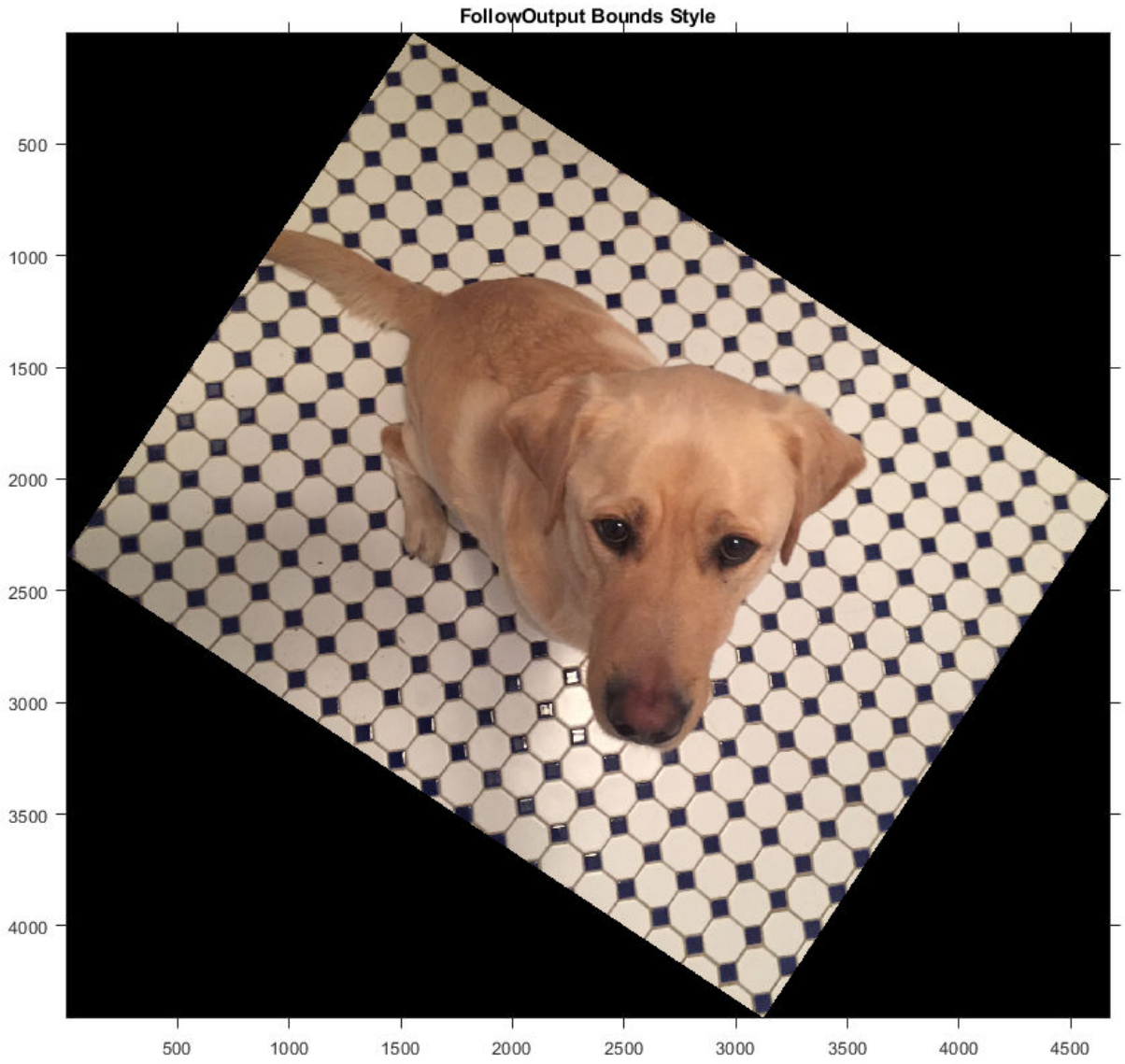
```
BCenterOutput = imwarp(A,tform,'OutputView',centerOutput);
BFollowOutput = imwarp(A,tform,'OutputView',followOutput);
BSameAsInput = imwarp(A,tform,'OutputView',sameAsInput);
```

Display the resulting images.

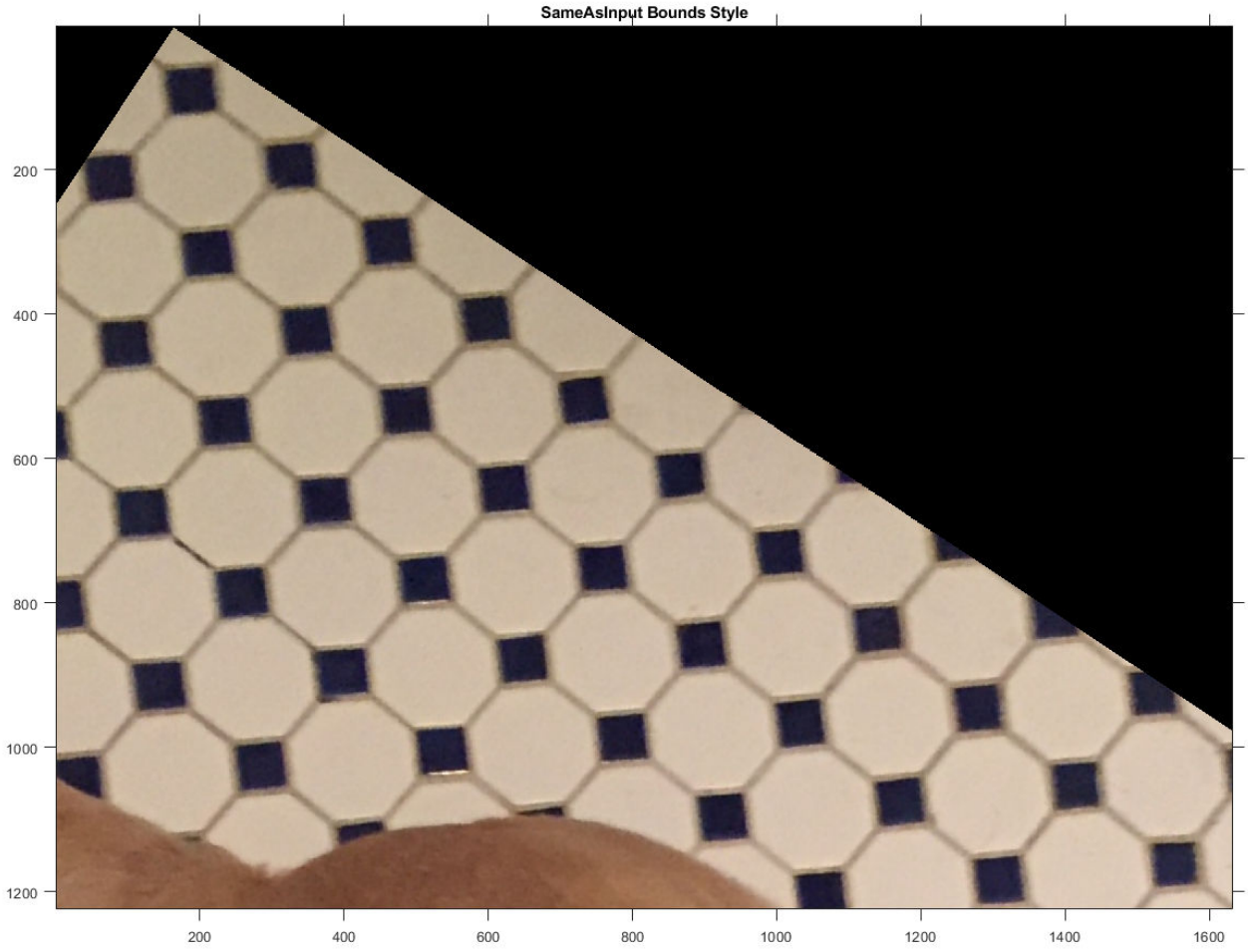
```
imshow(BCenterOutput)
title('CenterOutput Bounds Style');
```




```
imshow(BFollowOutput)
title('FollowOutput Bounds Style');
```




```
imshow(BSameAsInput)
title('SameAsInput Bounds Style');
```




```
iptsetpref('ImshowAxesVisible','off')
```

Input Arguments

`sizeA` — Input image size

2-element numeric vector | 3-element numeric vector

Input image size, specified as a 2-element numeric vector for 2-D image input or a 3-element numeric vector for 3-D volumetric image input.

`tform` — Affine geometric transformation

`affine2d` object | `affine3d` object

Affine geometric transformation, specified as an `affine2d` or `affine3d` object.

`style` — Bounds style

`'CenterOutput'` (default) | `'FollowOutput'` | `'SameAsInput'`

Bounds style, specified as one of the following values.

Style	Description
'CenterOutput'	Center the view at the center of the image in output space while allowing translation to move the output image out of view.
'FollowOutput'	Set the limits of the output view to completely contain the output image.
'SameAsInput'	Set the output limits to be the same as the input limits.

Output Arguments

Rout — Spatial referencing

imref2d object | imref3d object

Spatial referencing, returned as an imref2d or imref3d object. Use Rout as the OutputView argument of the imwarp function to specify the spatial referencing of the warped output.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® backgroundPool or accelerate code with Parallel Computing Toolbox™ ThreadPool.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

imwarp | randomAffine2d | randomAffine3d

Topics

“Define World Coordinate System of Image”

Introduced in R2019b

analyze75info

Read metadata from header file of Analyze 7.5 data set

Syntax

```
info = analyze75info(filename)
info = analyze75info(filename,'ByteOrder',byteOrder)
```

Description

`info = analyze75info(filename)` reads the header file of the Analyze 7.5 data set specified by `filename`. The function returns `info`, a structure whose fields contain information about the data set. Analyze 7.5 is a 3-D biomedical image visualization and analysis product developed by the Biomedical Imaging Resource of the Mayo Clinic. An Analyze 7.5 data set is made of two files, a header file and an image file. The files have the same name with different file extensions. The header file has the file extension `.hdr` and the image file has the file extension `.img`.

`info = analyze75info(filename,'ByteOrder',byteOrder)` reads the Analyze 7.5 header file with Big Endian or Little Endian byte ordering.

Examples

Get Information about an Analyze 7.5 Data Set

Get information about an Analyze 7.5 data set. An Analyze 7.5 data set is made up of two files: a header file with the file extension `.hdr` and an image file with the file extension `.img`. You don't need to specify a file extension when calling `analyze75info`.

```
info = analyze75info('brainMRI');
```

Get information about an Analyze 7.5 data set, this time specifying the byte ordering of the data set. If you specify the wrong byte order, `analyze75info` attempts to read the file with the other supported byte order.

```
info = analyze75info('brainMRI', 'ByteOrder', 'ieee-le');
```

Input Arguments

`filename` — Name of Analyze 7.5 data set

character vector | string

Name of Analyze 7.5 data set, specified as a string or character vector. You don’t need to specify a file extension.

Example: `info = analyze75info('brainMRI');`

Data Types: `char` | `string`

`byteOrder` — Endianness of data

character vector | string

Endianness of the data, specified as one of the strings or character vectors in the following table. If the specified value results in a read error, `analyze75info` issues a warning message and attempts to read the header file with the opposite `ByteOrder` format.

Value	Meaning
`'ieee-le'`	Byte ordering is Little Endian
`'ieee-be'`	Byte ordering is Big Endian

Data Types: `char` | `string`

Output Arguments

`info` — Information about Analyze 7.5 data set

structure

Information about Analyze 7.5 data set, returned as a structure.

See Also

`analyze75read`

Introduced before R2006a

analyze75read

Read image data from image file of Analyze 7.5 data set

Syntax

```
X = analyze75read(filename)
X = analyze75read(info)
```

Description

`X = analyze75read(filename)` reads the image data from the image file of an Analyze 7.5 format data set specified by the character vector `filename`. The function returns the image data in `X`.

Analyze 7.5 is a 3-D biomedical image visualization and analysis product developed by the Biomedical Imaging Resource of the Mayo Clinic. An Analyze 7.5 data set is made of two files, a header file and an image file. The files have the same name with different file extensions. The header file has the file extension `.hdr` and the image file has the file extension `.img`.

Note By default, `analyze75read` returns image data in radiological orientation (LAS). For more information, see “Read Image Data from Analyze 7.5 File” on page 1-91.

`X = analyze75read(info)` reads the image data from the image file specified in the metadata structure `info`. `info` must be a valid metadata structure returned by the `analyze75info` function.

Examples

Read Image Data from Analyze 7.5 File

Read image data from an Analyze 7.5 file.

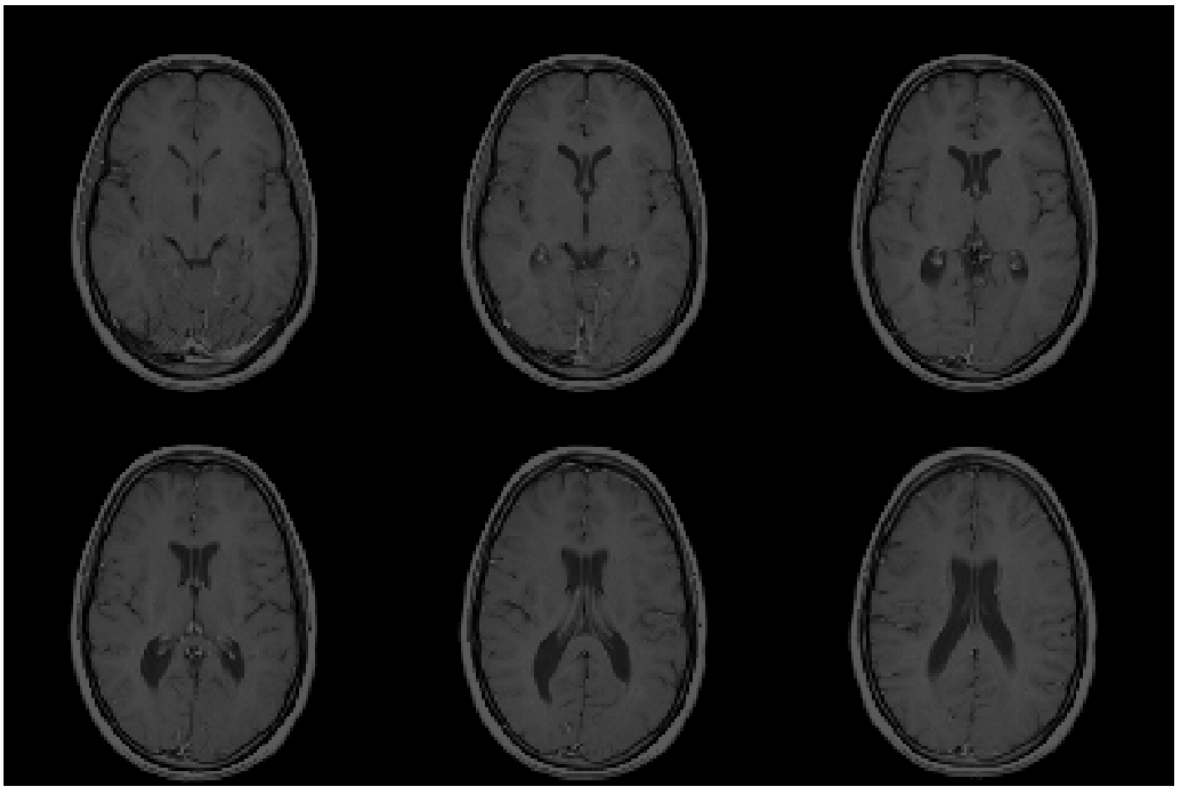
```
X = analyze75read('brainMRI');
```

View the data. First, because Analyze 7.5 format uses radiological orientation (LAS), flip the data for correct image display in MATLAB.

```
X = flip(X);
```

Then, reshape the data to create an array that can be displayed using `montage`. Select frames 12 to 17.

```
Y = reshape(X(:,:,12:17),[size(X,1) size(X,2) 1 6]);
montage(Y);
```



Read Image Data Using the Info Structure

Read image data from an Analyze 7.5 data set, using the structure returned by `analyze75info` to specify the data set. First, use `analyze75info` to create the info structure.

```
info = analyze75info('brainMRI');
```

Call `analyze75read` to read image data from the data set, specifying the info structure returned by `analyze75info`.

```
X = analyze75read(info);
```

Input Arguments

`filename` — Name of Analyze 7.5 data set

character vector

Name of Analyze 7.5 data set, specified as a character vector. You don't need to specify a file extension.

Example: `info = analyze75info('brainMRI');`

Data Types: `char`

`info` — Information about Analyze 7.5 data set

structure

Information about the Analyze 7.5 data set, specified as a structure returned by the `analyze75info` function.

Data Types: `struct`

Output Arguments

X — Image data from Analyze 7.5 data set

array

Image data from Analyze 7.5 data set, returned as an array. `X` can be `logical`, `uint8`, `int16`, `int32`, `single`, or `double`. `analyze75read` uses a data type for `X` that is consistent with the data type specified in the data set header file. Complex and RGB data types are not supported. For single-frame, grayscale images, `X` is an *m*-by-*n* array.

See Also

`analyze75info`

Introduced before R2006a

applycform

Apply device-independent color space transformation

Syntax

```
B = applycform(A,C)
```

Description

`B = applycform(A,C)` converts the color values in `A` to the color space specified in the color transformation structure `C`.

Examples

Convert sRGB to L*a*b* Color Space using Applycform

Read color image that uses the sRGB color space into the workspace.

```
rgb = imread('peppers.png');
```

Create a color transformation structure that defines an sRGB to L*a*b* conversion.

```
C = makecform('srgb2lab');
```

Perform the transformation by using the `applycform` function.

```
lab = applycform(rgb,C);
```

Input Arguments

A — Input color space

2-D numeric matrix | 3-D numeric array | string | character vector

Input color space, specified as one of the following:

- 2-D numeric matrix. `applycform` interprets each row as a color unless the color transformation structure, `C`, contains a grayscale ICC profile. In that case, `applycform` interprets each pixel in `A` as a color.
- 3-D numeric matrix. Each row-column location is interpreted as a color. `size(A,3)` is typically 1 or more, depending on the input color space.
- string or character vector. `A` is only a string or character vector if `C` is created with the following syntax:

  ```
  C = makecform('named', profile, space)
  ```

Data Types: `double` | `uint8` | `uint16` | `char` | `string`

C — Color transformation

structure

Color transformation, specified as a structure. The color transformation structure specifies various parameters of the transformation. You can create a color transformation structure using `makecform`.

Output Arguments

B — Output color space

numeric array

Output color space, returned as a numeric array. The size of `B` depends on the dimensionality and size of the input color space, `A`:

- When `A` is two-dimensional, `B` has the same number of rows and one or more columns, depending on the output color space. (The ICC specification currently supports up to 15-channel device spaces).
- When `A` is three-dimensional, `B` is the same number of rows and columns as `A`, and `size(B,3)` is 1 or more, depending on the output color space.

See Also

`lab2double` | `lab2uint8` | `lab2uint16` | `makecform` | `whitepoint` | `xyz2double` | `xyz2uint16`

Topics

“Understanding Color Spaces and Color Space Conversion”

Introduced before R2006a

applylut

Neighborhood operations on binary images using lookup tables

Note `applylut` is not recommended. Use `bwlookup` instead.

Syntax

```
A = applylut(BW,lut)
```

Description

`A = applylut(BW,lut)` performs a 2-by-2 or 3-by-3 neighborhood operation on binary image `BW` by using a lookup table, `lut`. The lookup table consists of the output values for all possible 2-by-2 or 3-by-3 neighborhoods.

Examples

Perform Erosion Using a 2-by-2 Neighborhood

Create the LUT.

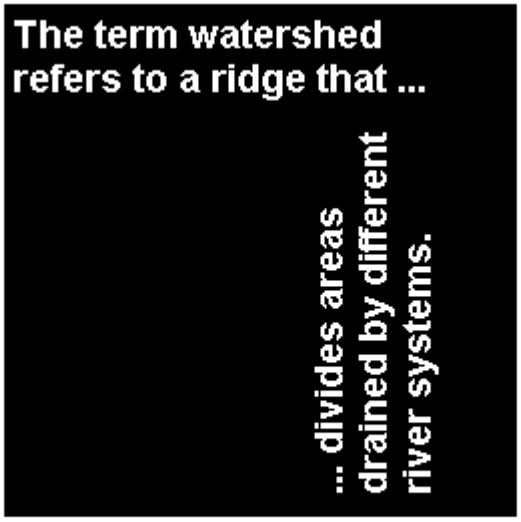
```
lutfun = @(x)(sum(x(:))==4);
lut    = makelut(lutfun,2);
```

Read image into the workspace and then apply the LUT to the image. An output pixel is `on` only if all four of the input pixel's neighborhood pixels are `on` .

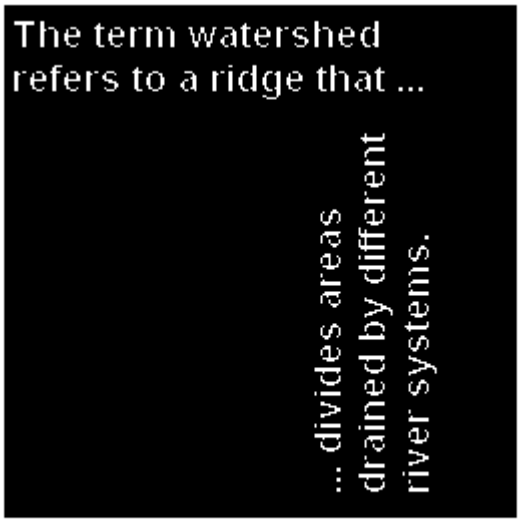
```
BW1    = imread('text.png');
BW2    = applylut(BW1,lut);
```

Show the original image and the eroded image.

```
figure, imshow(BW1);
```




```
figure, imshow(BW2);
```



Input Arguments

BW — Input image

2-D binary image

Input image, specified as a 2-D binary image. For numeric input, any nonzero pixels are considered to be 1 (`true`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

lut — Lookup table of output pixel values

16-element numeric vector | 512-element numeric vector

Lookup table of output pixel values, specified as a 16- or 512-element vector as returned by `makelut`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

A — Output image

binary image | grayscale image

Output image, returned as a grayscale or binary image whose distribution of pixel values are determined by the content of the lookup table, `lut`. The output image `J` is the same size as the input image `I`.

- If all elements of `lut` are `0` or `1`, then `A` has data type `logical`.
- If all elements of `lut` are integers between 0 and 255, then `A` has data type `uint8`.
- For all other cases, `A` has data type `double`.

Data Types: `double` | `uint8` | `logical`

Algorithms

`applylut` performs a neighborhood operation on a binary image by producing a matrix of indices into `lut`, and then replacing the indices with the actual values in `lut`. The specific algorithm used depends on whether you use 2-by-2 or 3-by-3 neighborhoods.

2-by-2 Neighborhoods

For 2-by-2 neighborhoods, `length(lut)` is 16. There are four pixels in each neighborhood, and two possible states for each pixel, so the total number of permutations is $2^4 = 16$.

To produce the matrix of indices, `applylut` convolves the binary image `BW` with this matrix.

```
8     2
4     1
```

The resulting convolution contains integer values in the range [0, 15]. `applylut` uses the central part of the convolution, of the same size as `BW`, and adds 1 to each value to shift the range to [1, 16]. The function then constructs `A` by replacing the values in the cells of the index matrix with the values in `lut` that the indices point to.

3-by-3 Neighborhoods

For 3-by-3 neighborhoods, `length(lut)` is 512. There are nine pixels in each neighborhood, and two possible states for each pixel, so the total number of permutations is $2^9 = 512$.

To produce the matrix of indices, `applylut` convolves the binary image `BW` with this matrix.

```
256     32      4
128     16      2
 64      8      1
```

The resulting convolution contains integer values in the range [0, 511]. `applylut` uses the central part of the convolution, of the same size as `BW`, and adds 1 to each value to shift the range to [1, 512]. It then constructs `A` by replacing the values in the cells of the index matrix with the values in `lut` that the indices point to.

Compatibility Considerations

`applylut` is not recommended

Not recommended starting in R2012b

Starting in R2012b, use `bwlookup` to perform neighborhood operations on binary images using lookup tables. For `bwlookup`, the data type of the returned image is the same as the data type of the lookup table. `bwlookup` supports code generation. There are no plans to remove `applylut` at this time.

To update your code, replace instances of `applylut` with `bwlookup`. You do not need to change the input arguments.

See Also

`makelut`

Introduced before R2006a

axes2pix

Convert axes coordinates to pixel coordinates

Syntax

```
pixelCoord = axes2pix(n,extent,axesCoord)
```

Description

`pixelCoord = axes2pix(n,extent,axesCoord)` converts an axes coordinate into an intrinsic ("pixel") coordinate.

Note The `imref2d` object has several methods that facilitate conversion between intrinsic coordinates, world coordinates and array indices.

Examples

Convert Axes Coordinate into Intrinsic Coordinate

Display image.

```
h = imshow('pout.tif');
```

Get the size of the image.

```
[nrows,ncols] = size(get(h,'CData'));
```

Get the image `XData` and `YData`.

```
xdata = get(h,'XData')
```

```
xdata = 1×2

     1   240
```

```
ydata = get(h,'YData')
```

```
ydata = 1×2

     1   291
```

Convert an axes coordinate into an intrinsic coordinate for the x and y dimensions.

```
px = axes2pix(ncols,xdata,30)
```

```
px = 30
```

```
py = axes2pix(nrows,ydata,30)
```

```
py = 30
```

Convert Axes Coordinate to Intrinsic Coordinate with Nondefault XData and YData

Read an image and display it. Get the size of the image.

```
I = imread('pout.tif');
[nrows,ncols] = size(I)
```

```
nrows = 291
```

```
ncols = 240
```

Create a spatial referencing object for this image, with default property settings. By default, the upper-left corner of the image has intrinsic coordinate (1,1).

```
RI = imref2d(size(I));
h = imshow(I,RI);
```


```
xData = get(h,'XData')
```

```
xData = 1×2

     1   240
```

```
yData = get(h,'YData')
```

```
yData = 1×2

     1   291
```

For illustrative purposes, specify an arbitrary image extent in the *x*- and *y*-directions. This example shifts the image up by 20 pixels and to the right by 400 pixels. The example also shifts the image to the right by 100 pixels and compresses the image horizontally by a factor of 2.

```
xWorldLimits = 0.5*xData + 400;
yWorldLimits = yData - 20;
RA = imref2d(size(I),xWorldLimits,yWorldLimits);
imshow(I,RA)
```

Select a pixel, such as a pixel near the nose of the child. This pixel occurs around the axes coordinate (x, y) = (450, 90) in the modified image.

Convert the axes coordinate to an intrinsic coordinate.

```
px = axes2pix(ncols,xWorldLimits,450)
```

```
px = 100
```

```
py = axes2pix(nrows,yWorldLimits,90)
```

```
py = 110
```

The intrinsic coordinate of the point is at (100, 110). This agrees with the location of the nose in the original image.

Input Arguments

`n` — Number of image rows or columns

positive integer

Number of image rows or columns, specified as a positive integer. `n` is the number of image columns for the *x*-coordinate, or the number of image rows for the *y*-coordinate.

`extent` — Image world extent

2-element numeric vector

Image world extent, specified as a 2-element numeric vector. `extent` is returned by `get(image_handle,'XData')` or `get(image_handle,'YData')`.

`axesCoord` — Axes coordinates to convert

numeric vector

Axes coordinate to convert to intrinsic coordinates, specified as a numeric vector.

Output Arguments

`pixelCoord` — Intrinsic coordinates

numeric vector

Intrinsic coordinates, returned as a numeric vector.

Data Types: `double`

Tips

- `axes2pix` performs minimal checking on the validity of the `n`, `axesCoord`, or `extent` arguments. For example, `axes2pix` can extrapolate from `extent` to return a negative coordinate. The function calling `axes2pix` bears responsibility for error checking.

See Also

`imref2d` | `impixelinfo` | `bwselect` | `impixel` | `improfile` | `roipoly`

Topics

“Image Coordinate Systems”

Introduced before R2006a

bestblk

Determine optimal block size for block processing

Syntax

```
siz = bestblk([M N],k)
[m,n] = bestblk([M N],k)
```

Description

`siz = bestblk([M N],k)` returns the optimal block size for block processing of an `M`-by-`N` image. The optimal block size minimizes the padding required along the outer partial blocks. `k` specifies the maximum row and column dimensions for the block.

`[m,n] = bestblk([M N],k)` returns the row and column dimensions for the block in `m` and `n`, respectively.

Examples

Determine Optimal Block Size

```
siz = bestblk([640 800],72)
```

```
siz = 1×2

    64    50
```

Input Arguments

`[M N]` — Image size

2-element vector of positive integers

Image size, specified as a 2-element vector of positive integers. `M` is the number of rows and `N` is the number of columns in the image.

Data Types: `double`

`k` — Maximum number of block rows or columns

`100` (default) | positive integer

Maximum number of block rows or columns, specified as a positive integer.

Data Types: `double`

Output Arguments

`siz` — Optimal block size

2-element numeric row vector

Optimal block size, returned as a 2-element numeric row vector. `siz` is equivalent to [`m` `n`].

`m`, `n` — Optimal number of block rows or columns
numeric scalar

Optimal number of block rows or columns, returned as a numeric scalar.

Algorithms

The algorithm for determining the optimal value of `m` from `M` and `k` is:

- If `M` is less than or equal to `k`, return `M`.
- If `M` is greater than `k`, consider all values between `min(M/10,k/2)` and `k`. Return the value that minimizes the padding required.

The same algorithm is used to find the optimal value of `n` from `N` and `k`.

See Also
`blockproc`

Introduced before R2006a

bfscore

Contour matching score for image segmentation

Syntax

```
score = bfscore(prediction,groundTruth)
[score,precision,recall] = bfscore(prediction,groundTruth)
[___] = bfscore(prediction,groundTruth,threshold)
```

Description

`score = bfscore(prediction,groundTruth)` computes the BF (Boundary F1) contour matching score between the predicted segmentation in `prediction` and the true segmentation in `groundTruth`. `prediction` and `groundTruth` can be a pair of logical arrays for binary segmentation, or a pair of label or categorical arrays for multiclass segmentation.

`[score,precision,recall] = bfscore(prediction,groundTruth)` also returns the precision and recall values for the `prediction` image compared to the `groundTruth` image.

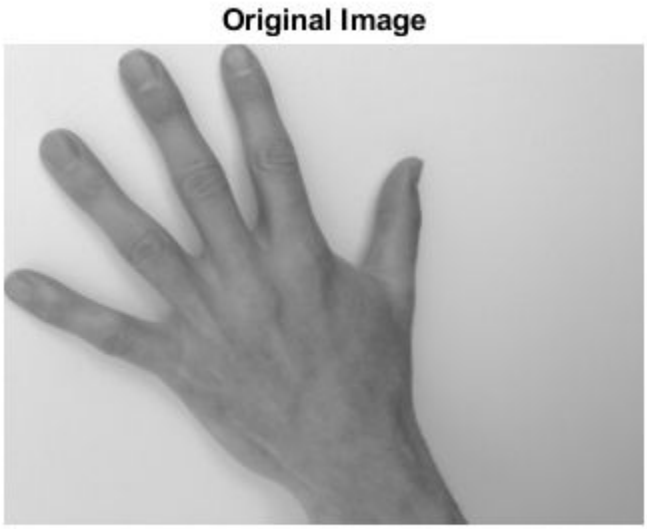
`[___] = bfscore(prediction,groundTruth,threshold)` computes the BF score using a specified threshold as the distance error tolerance, to decide whether a boundary point has a match or not.

Examples

Compute BF Score for Binary Segmentation

Read an image with an object to segment. Convert the image to grayscale, and display the result.

```
A = imread('hands1.jpg');
I = im2gray(A);
figure
imshow(I)
title('Original Image')
```



Use the active contours (snakes) method to segment the hand.

```
mask = false(size(I));
mask(25:end-25,25:end-25) = true;
BW = activecontour(I, mask, 300);
```

Read the ground truth segmentation.

```
BW_groundTruth = imread('hands1-mask.png');
```

Compute the BF score of the active contours segmentation against the ground truth.

```
similarity = bfscore(BW, BW_groundTruth);
```

Display the masks on top of each other. Colors indicate differences in the masks.

```
figure
imshowpair(BW, BW_groundTruth)
title(['BF Score = ' num2str(similarity)])
```

Compute BF Score for Multi-Region Segmentation

This example shows how to segment an image into multiple regions. The example then computes the BF score for each region.

Read an image with several regions to segment.

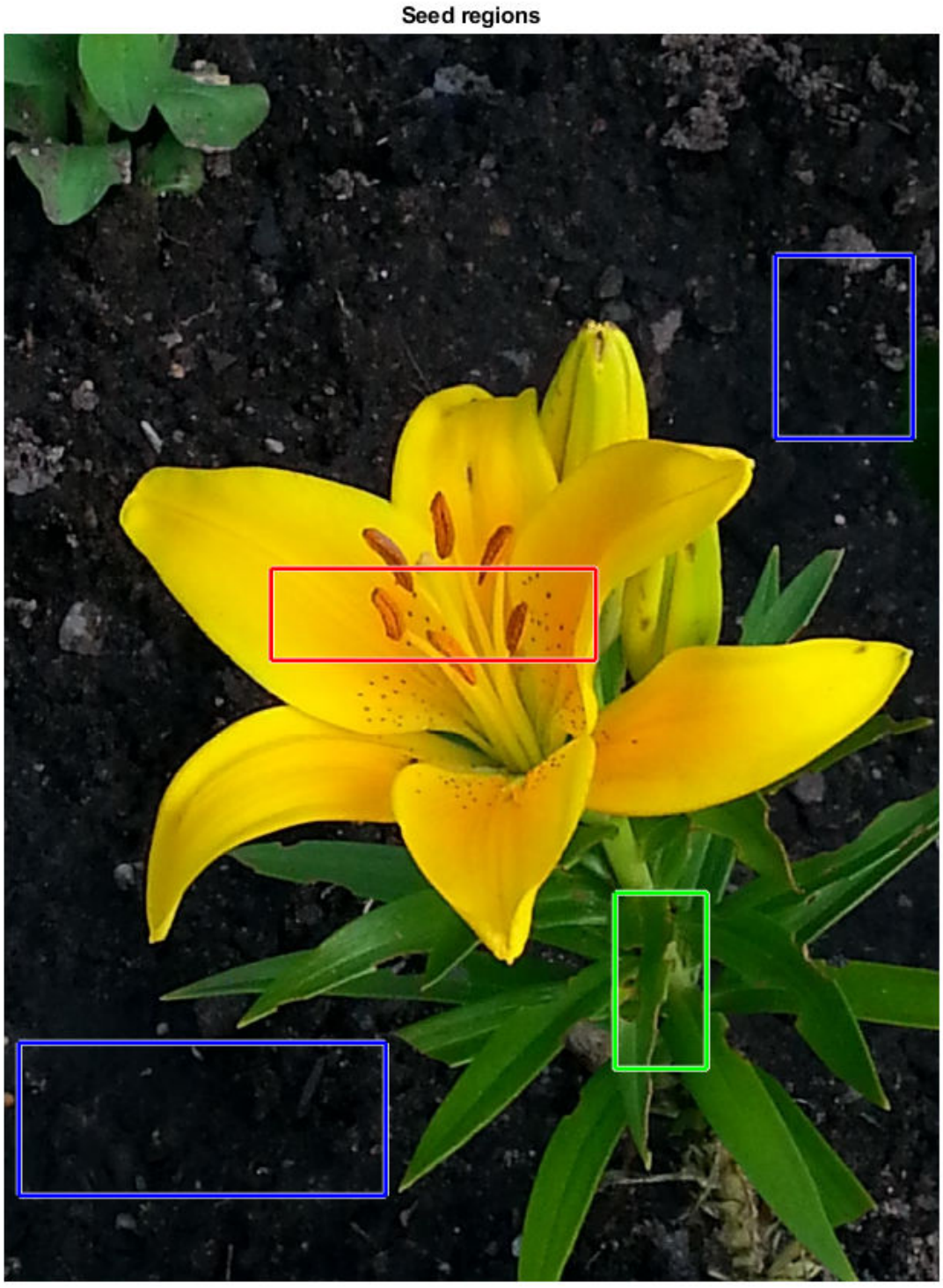
```
RGB = imread('yellowlily.jpg');
```

Create scribbles for three regions that distinguish their typical color characteristics. The first region classifies the yellow flower. The second region classifies the green stem and leaves. The last region classifies the brown dirt in two separate patches of the image. Regions are specified by a 4-element vector, whose elements indicate the x- and y-coordinate of the upper left corner of the ROI, the width of the ROI, and the height of the ROI.

```
region1 = [350 700 425 120]; % [x y w h] format
BW1 = false(size(RGB,1),size(RGB,2));
BW1(region1(2):region1(2)+region1(4),region1(1):region1(1)+region1(3)) = true;

region2 = [800 1124 120 230];
BW2 = false(size(RGB,1),size(RGB,2));
BW2(region2(2):region2(2)+region2(4),region2(1):region2(1)+region2(3)) = true;

region3 = [20 1320 480 200; 1010 290 180 240];
BW3 = false(size(RGB,1),size(RGB,2));
BW3(region3(1,2):region3(1,2)+region3(1,4),region3(1,1):region3(1,1)+region3(1,3)) = true;
BW3(region3(2,2):region3(2,2)+region3(2,4),region3(2,1):region3(2,1)+region3(2,3)) = true;
```

Display the seed regions on top of the image.

```
figure
imshow(RGB)
hold on
visboundaries(BW1,'Color','r');
visboundaries(BW2,'Color','g');
visboundaries(BW3,'Color','b');
title('Seed regions')
```



Segment the image into three regions using geodesic distance-based color segmentation.

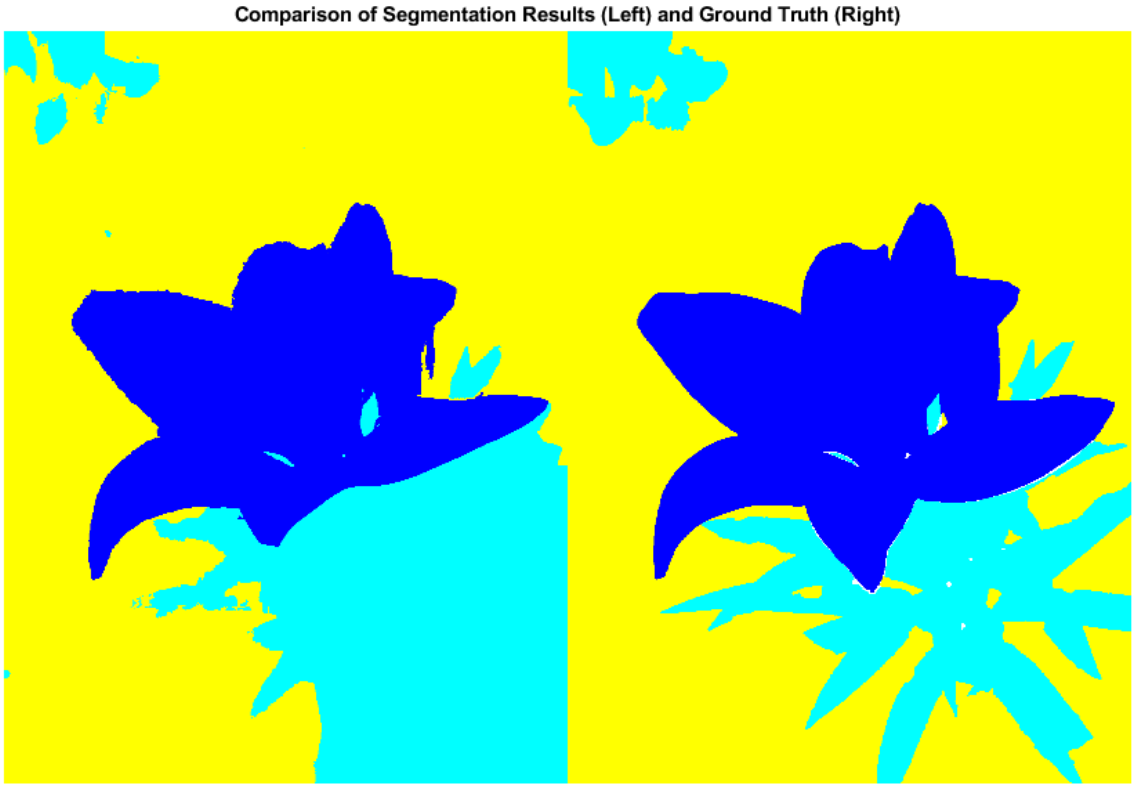
```
L = imseggeodesic(RGB,BW1,BW2,BW3,'AdaptiveChannelWeighting',true);
```

Load a ground truth segmentation of the image.

```
L_groundTruth = double(imread('yellowlily-segmented.png'));
```

Visually compare the segmentation results with the ground truth.

```
figure
imshowpair(label2rgb(L),label2rgb(L_groundTruth),'montage')
title('Comparison of Segmentation Results (Left) and Ground Truth (Right)')
```



Compute the BF score for each segmented region.

```
similarity = bfscore(L, L_groundTruth)
```

```
similarity = 3×1

    0.7992
    0.5333
    0.7466
```

The BF score is noticeably smaller for the second region. This result is consistent with the visual comparison of the segmentation results, which erroneously classifies the dirt in the lower right corner of the image as leaves.

Input Arguments

`prediction` — Predicted segmentation

2-D or 3-D logical, numeric, or categorical array

Predicted segmentation, specified as a 2-D or 3-D logical, numeric, or `categorical` array. If `prediction` is a numeric array, then it represents a label array and must contain nonnegative integers of data type `double`.

Data Types: `logical` | `double` | `categorical`

`groundTruth` — Ground truth segmentation

2-D or 3-D logical, numeric, or categorical array

Ground truth segmentation, specified as a 2-D or 3-D logical, numeric, or `categorical` array of the same size and data type as `prediction`. If `groundTruth` is a numeric array, then it represents a label array and must contain nonnegative integers of data type `double`.

Data Types: `logical` | `double` | `categorical`

`threshold` — Distance error tolerance threshold

positive scalar

Distance error tolerance threshold in pixels, specified as a positive scalar. The threshold determines whether a boundary point has a match or not. If `threshold` is not specified, then the default value is 0.75% of the length of the image diagonal.

Example: 3

Data Types: `double`

Output Arguments

`score` — BF score

numeric scalar | numeric vector

BF score, returned as a numeric scalar or vector with values in the range [0, 1]. A `score` of 1 means that the contours of objects in the corresponding class in `prediction` and `groundTruth` are a perfect match. If the input arrays are:

- logical arrays, `score` is a scalar and represents the BF score of the foreground.
- label or categorical arrays, `score` is a vector. The first coefficient in `score` is the BF score for the first foreground class, the second coefficient is the score for the second foreground class, and so on.

`precision` — Precision

numeric scalar | numeric vector

Precision, returned as a numeric scalar or numeric vector with values in the range [0, 1]. Each element indicates the precision of object contours in the corresponding foreground class.

Precision is the ratio of the number of points on the boundary of the predicted segmentation that are close enough to the boundary of the ground truth segmentation to the length of the predicted boundary. In other words, precision is the fraction of detections that are true positives rather than false positives.

`recall` — Recall
numeric scalar | numeric vector

Recall, returned as a numeric scalar or numeric vector with values in the range [0, 1]. Each element indicates the recall of object contours in the corresponding foreground class.

Recall is the ratio of the number of points on the boundary of the ground truth segmentation that are close enough to the boundary of the predicted segmentation to the length of the ground truth boundary. In other words, recall is the fraction of true positives that are detected rather than missed.

More About

BF (Boundary F1) Score

The BF score measures how close the predicted boundary of an object matches the ground truth boundary.

The BF score is defined as the harmonic mean (F1-measure) of the `precision` and `recall` values with a distance error tolerance to decide whether a point on the predicted boundary has a match on the ground truth boundary or not.

```
score = 2 * precision * recall / (recall + precision)
```

References

[1] Csurka, G., D. Larlus, and F. Perronnin. "What is a good evaluation measure for semantic segmentation?" *Proceedings of the British Machine Vision Conference*, 2013, pp. 32.1-32.11.

See Also

`jaccard` | `dice`

Introduced in R2017b

bigimage

Out-of-core processing of very large images

Note `bigimage` is not recommended. Use the `blockedImage` object instead. For more information, see “Compatibility Considerations”.

Description

A `bigimage` object stores information about a large TIFF image file and the image data it contains. A `bigimage` represents images as smaller blocks of data that can be independently loaded and processed.

Use a `bigimage` object to visualize and process images that are too large to fit in memory, or when processing the image requires more memory than is available. Additionally, the object can:

- Read, process, and display images at different multiple resolution levels (image pyramids).
- Read arbitrary regions of the image.
- Read, set, and write blocks of data.

For big images with multiple resolution levels, the lowest or *coarsest* resolution level is the level where each pixel covers the largest area. The highest or *finest* resolution level is the level where each pixel covers the smallest area.

Creation

Syntax

```
bigimg = bigimage(filename)
bigimg = bigimage(dirname)
bigimg = bigimage(varname)
bigimg = bigimage(spatialReferencing,channels,classUnderlying)
bigimg = bigimage(levelSizes,channels,classUnderlying)
bigimg = bigimage( ___ ,'Classes',classes,'PixelLabelIDs',pixelLabelIDs)
bigimg = bigimage( ___ ,Name,Value)
```

Description

`bigimg = bigimage(filename)` creates a `bigimage` object from the big image file with name `filename`.

`bigimg = bigimage(dirname)` creates a `bigimage` object from a directory with name `dirname` containing files with big image data.

`bigimg = bigimage(varname)` creates a `bigimage` object from the variable `varname` in the workspace.

`bigimg = bigimage(spatialReferencing,channels,classUnderlying)` creates a writeable `bigimage` object and sets the SpatialReferencing, Channels, and ClassUnderlying properties, without initializing the image data.

`bigimg = bigimage(levelSizes,channels,classUnderlying)` creates a writeable `bigimage` object and sets the LevelSizes, Channels, and ClassUnderlying properties, without initializing the image data.

`bigimg = bigimage( ___ ,'Classes',classes,'PixelLabelIDs',pixelLabelIDs)` creates a `bigimage` object with categorical data. Specify class names using the Classes property and the mapping of image pixel label values to categorical class names using the PixelLabelIDs property.

`bigimg = bigimage( ___ ,Name,Value)` uses name-value pairs to set one or more of the BlockSize, SpatialReferencing, UndefinedID, UnloadedValue properties. You can specify multiple name-value pairs. Enclose each property name in quotes.

For example, `bigimage(bigfile,'BlockSize',[256 256],'UnloadedValue',128)` creates a big image from file `bigfile` that has a block size of 256-by-256 pixels and a default pixel value of `128`.

Input Arguments

`filename` — Name of big image file
character vector | string scalar

Name of the big image file, specified as a character vector or string scalar. Supported file formats are TIFF and BigTIFF. This argument sets the DataSource property.

`dirname` — Name of big image directory
character vector

Name of the big image directory, specified as a character vector or string scalar. This argument sets the DataSource property.

`varname` — Big image variable
numeric array

Big image variable in the workspace, specified as a numeric array of size *m*-by-*n* for a single-channel image or *m*-by-*n*-by-*c* for an image with *c* color channels.

Properties

Image File Properties

`DataSource` — Location of data
character vector

Location of the data backing the big image, specified as a character vector. Supported file formats are TIFF and BigTIFF.

If you create a `bigimage` object without specifying the name of a big image file, then the value of `DataSource` is `''`. If you create a `bigimage` object from a variable in the workspace, then the value of `DataSource` is `'variable'`.

Data Types: `string`

`SourceDetails` — Source metadata

struct

This property is read-only.

Source metadata, specified as a struct such as returned by `imfinfo`.

Image Data Properties

`BlockSize` — Block size

2-element row vector

Block size, specified as a 2-element row vector of positive integers of the form `[numrows numcols]`. The block size is the smallest unit of data that the `bigimage` object can read or write.

Data Types: `double`

`Channels` — Number of channels

positive integer

This property is read-only.

Number of color or multispectral channels, specified as a positive integer.

Data Types: `double`

`Classes` — Class names

string array | cell array of character vectors

Class names of categorical data, specified as a string array or cell array of character vectors. `Classes` can contain duplicate names to map multiple pixel label IDs to the same categorical class.

Data Types: `char` | `string`

`ClassUnderlying` — Data type of image pixels

`"double"` | `"single"` | `"uint8"` | `"uint16"` | ...

This property is read-only.

Data type of image pixels, specified as one of the following strings.

`"double"`	`"uint8"`	`"int8"`
`"single"`	`"uint16"`	`"int16"`
`"logical"`	`"uint32"`	`"int32"`
`"categorical"`		

Data Types: `char` | `string`

`CoarsestResolutionLevel` — Coarsest resolution level

positive integer

This property is read-only.

Coarsest resolution level, specified as a positive integer. For single-resolution images, `CoarsestResolutionLevel` is `1`.

Data Types: `double`

`FinestResolutionLevel` — Finest resolution level
positive integer

This property is read-only.

Finest resolution level, specified as a positive integer. For single-resolution images, `FinestResolutionLevel` is `1`.

Data Types: `double`

`LevelSizes` — Image dimensions at each resolution level
r-by-2 matrix of positive integers

Image dimensions at each resolution level, specified as an *r*-by-2 matrix of positive integers. Each row specifies the `[numrows numcols]` image dimensions at one of the *r* resolution levels.

Data Types: `double`

`PixelLabelIDs` — Pixel label IDs
c-element numeric vector | *c*-by-3 numeric array of data type `uint8`

Pixel label IDs that map pixel label values to categorical class names, specified as one of the following.

- *c*-element numeric vector, where *c* is the number of classes.
- *c*-by-3 numeric array of data type `uint8`. Each row contains a 3-element vector representing the RGB pixel value to associate with each class name. Use this format when the pixel label data is stored as an RGB image.

If a pixel has a value that does not exist in `PixelLabelIDs`, then `bigimage` maps the pixel to the class `'<undefined>'`.

`SpatialReferencing` — Pixel locations and sizes
scalar `imref2d` object | *r*-by-1 vector of `imref2d` objects

Pixel locations and sizes, specified as a scalar `imref2d` object for a single-resolution big image or an *r*-by-1 vector of `imref2d` objects for a multi-resolution big image. Each element specifies the pixel size, image size, and world limits at one of the *r* resolution levels.

`UndefinedID` — Pixel label value for `'<undefined>'` categorical class
`0` (default) | numeric scalar | 1-by-3 numeric vector

Pixel label value for the `'<undefined>'` categorical class, specified as a numeric scalar or a 1-by-3 numeric vector. Do not specify this value as any of the values in PixelLabelIDs.

`UnloadedValue` — Default pixel value
logical scalar | numeric scalar | 1-by-1-by-`Channels` numeric vector | string scalar

Default pixel value used to fill blocks that do not exist in the DataSource, specified as a value in the table. If you do not specify `UnloadedValue`, then `bigimage` uses the pixel value of `0` for numeric and logical blocks and `missing` for categorical blocks.

Image Data Type	Format of `UnloadedValue`
Logical image	Logical scalar.
Numeric image	Numeric scalar for grayscale images or a 1-by-1-by-Channels numeric vector for truecolor and multispectral images. If you specify a numeric scalar when `Channels` is greater than 1, then `bigimage` extends the value to a 1-by-1-by-`Channels` numeric vector. The data type of `UnloadedValue` must match the data type specified by ClassUnderlying
Categorical image	String scalar specifying an element of the Classes property.

Object Functions

apply	(Not recommended) Process blocks of bigimage object
isequal	(Not recommended) Compare two bigimage objects for equality
getBlock	(Not recommended) Read block of bigimage object
getFullLevel	(Not recommended) Get all data in one level of bigimage object
getRegion	(Not recommended) Read arbitrary region of bigimage object
setBlock	(Not recommended) Put data in specific block of bigimage object
write	(Not recommended) Write bigimage object content to new file

Examples

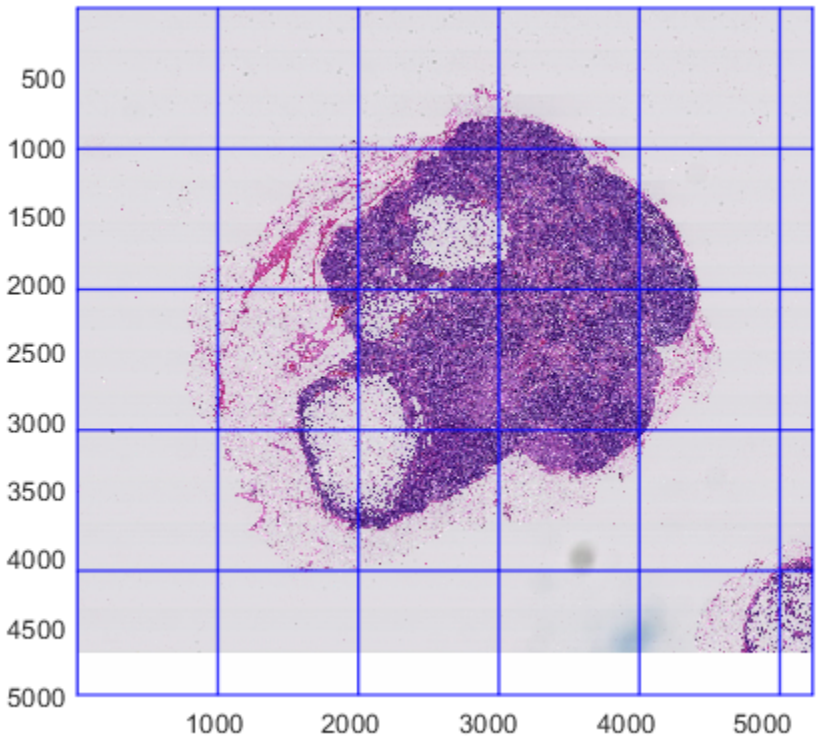
Create Single-Resolution Mask from Multiresolution Blocked Image

Create a blocked image from the sample image tumor_091R.tif. This sample image is a training image of a lymph node containing tumor tissue from the CAMELYON16 data set. The image has been modified to have three coarse resolution levels, and has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = blockedImage('tumor_091R.tif');
```

Display the entire blocked image at the finest resolution level, including a grid of the block boundaries.

```
bshow = bigimageshow(bim,'ResolutionLevel','fine', ...
    'GridVisible','on','GridLevel',1);
```



Create a mask of the coarsest resolution level.

First create a single-resolution image of the coarsest resolution level. By default, the `gather` function gets data from the coarsest resolution level.

```
imcoarse = gather(bim);
```

Convert the coarse image to grayscale.

```
graycoarse = rgb2gray(imcoarse);
```

Binarize the grayscale image. In the binarized image, the object of interest is black and the background is white.

```
bwcoarse = imbinarize(graycoarse);
```

Take the complement of the binarized image. The resulting mask follows the convention in which the object of interest is white and the background is black.

```
mask = imcomplement(bwcoarse);
```

Create a blocked image containing the mask.

Use the same spatial referencing as the original blocked image. Determine the coarsest resolution level and capture the spatial referencing information of the blocked image at the first two dimensions at that level.

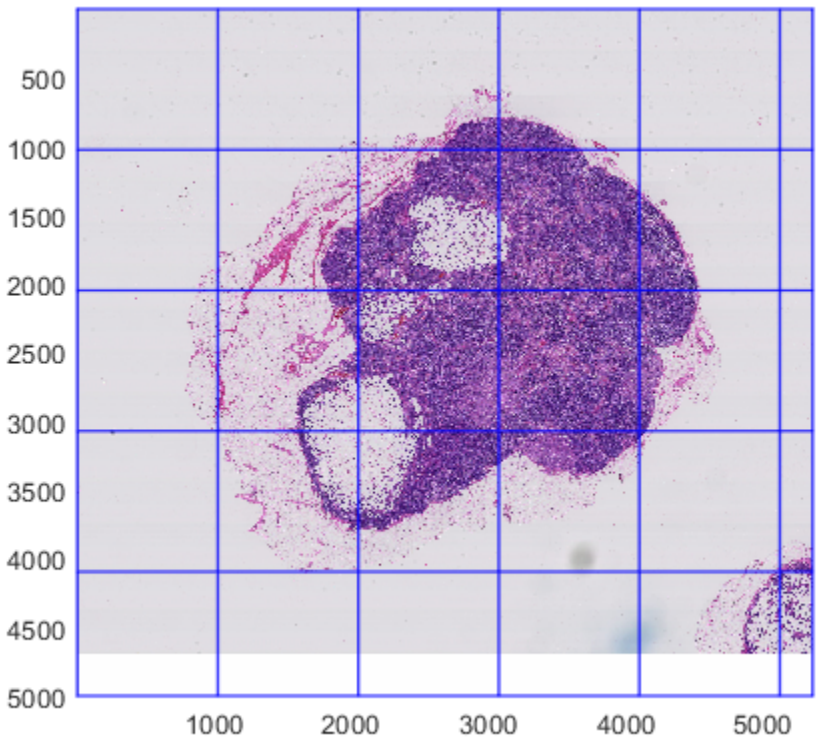
```
coarsestLevel = bim.NumLevels;
originalWorldStartCoarsest = bim.WorldStart(coarsestLevel,1:2);
originalWorldEndCoarsest = bim.WorldEnd(coarsestLevel,1:2);
```

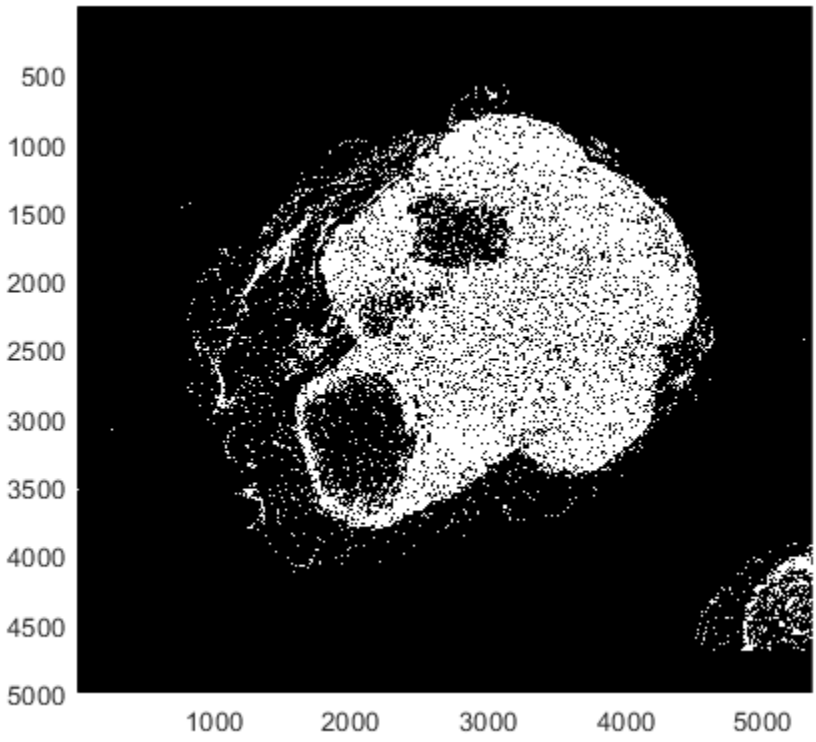
Create the blocked image for the mask.

```
bmask = blockedImage(mask,'WorldStart',originalWorldStartCoarsest, ...
   'WorldEnd',originalWorldEndCoarsest);
```

Display the mask image.

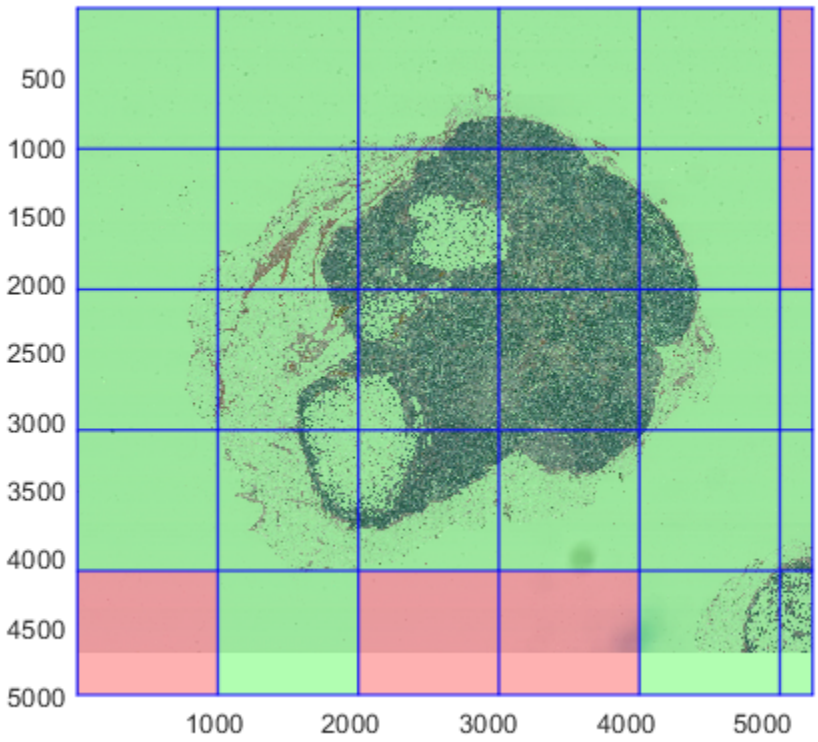
```
figure
bigimageshow(bmask)
```





Overlay the mask on the display of the original blocked image using the `showmask` function. To highlight all blocks that contain at least one nonzero mask pixel, specify an inclusion threshold of `0`.

```
showmask(bshow,bmask,'InclusionThreshold',0)
```



Tips

- You can write to `bigimage` objects that are created using the SpatialReferencing or LevelSizes syntaxes. Write to these `bigimage` objects by using the `setBlock` function. You cannot write to `bigimage` objects that are created using the `filename`, `dirname`, or `varname` syntaxes.
- A `bigimage` object uses the UnloadedValue property in two situations. The first situation is when you create a writeable `bigimage` object. Blocks of the writeable `bigimage` object are set to `UnloadedValue` until you write the block data by using the `setBlock` function. The second situation is when the `apply` function stops processing blocks of a `bigimage` object before all blocks are processed.

Compatibility Considerations

`bigimage` is not recommended
Not recommended starting in R2021a

Starting in R2021a, the `blockedImage` object replaces the existing `bigimage` object. The new blocked image object offers these advantages:

- Extension to N-D processing
- Introduction of a documented adapter interface to enable custom support for any data source that can be chunked into blocks
- Simpler interface afforded by use of direct pixel subscripts (`bigimage` uses world coordinates)

- Simpler interface for single-resolution level images (default resolution level is 1)

To display blocked image data, you still use the `bigimageshow` function.

Code Updates

Update all instances of the `bigimage` object.

Discouraged Usage	Recommended Replacement
This example creates a `bigimage` object. `bim = bigimage('tumor_091R.tif');`	Here is equivalent code, replacing the `bigimage` object with the new `blockedImage` object. `bim = blockedImage('tumor_091R.tif');`

Other Code Updates

The `blockedImage` object supports some different properties and object functions than the `bigimage` object. For example, to retrieve image data in a blocked image at a specified resolution level, replace calls to `getFullLevel` with use of the `gather` object function.

References

[1] Bejnordi, Babak Ehteshami, Mitko Veta, Paul Johannes van Diest, Bram van Ginneken, Nico Karssemeijer, Geert Litjens, Jeroen A. W. M. van der Laak, et al. “Diagnostic Assessment of Deep Learning Algorithms for Detection of Lymph Node Metastases in Women With Breast Cancer.” JAMA 318, no. 22 (December 12, 2017): 2199–2210. https://doi.org/10.1001/jama.2017.14585.

[2] Grand Challenge. https://camelyon17.grand-challenge.org/Data/.

See Also

`bigimageDatastore` | `bigimageshow` | `selectBlockLocations` | `blockLocationSet`

Topics

“Set Up Spatial Referencing for Blocked Images”
“Process Blocked Images Efficiently Using Partial Images or Lower Resolutions”
“Process Blocked Images Efficiently Using Mask”
“Explore Blocked Image Details with Interactive ROIs”
“Warp Blocked Image at Coarse and Fine Resolution Levels”

Introduced in R2019b

apply

(Not recommended) Process blocks of `bigimage` object

Note The `apply` function of the `bigimage` object is not recommended. Use the `apply` function associated with the `blockedImage` object instead. For more information, see “Compatibility Considerations”.

Syntax

```
newbig = apply(bigimg,level,fun)
newbig = apply(bigimg,level,fun,extraImages)
newbig = apply( ___ ,Name,Value)

[newbig,other1,other2,...] = apply( ___ )
```

Description

`newbig = apply(bigimg,level,fun)` processes all blocks of the big image `bigimg` at the specified resolution level using the function `fun` and returns a new big image `newbig` containing the processed data.

`newbig = apply(bigimg,level,fun,extraImages)` processes all blocks of big image `bigimg` and one or more extra big images `extraImages`. Use this syntax when the function `fun` accepts multiple image inputs, such as an image and a mask.

`newbig = apply( ___ ,Name,Value)` controls aspects of the processing, such as processing data in parallel or padding blocks on the edge of the image, using name-value pair arguments.

`[newbig,other1,other2,...] = apply( ___ )` returns multiple outputs. Use this syntax when the function `fun` returns multiple outputs, including image and non-image output.

Examples

Apply Filter to Big Image

Create a `bigimage` using a modified version of image "tumor_091.tif" from the CAMELYON16 data set. The original image is a training image of a lymph node containing tumor tissue. The original image has eight resolution levels, and the finest level has resolution 53760-by-61440. The modified image has only three coarse resolution levels. The spatial referencing of the modified image has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = bigimage('tumor_091R.tif');
```

Enhance structures in the image by applying an edge-preserving non-local means filter to each block at the finest resolution level, 1. For this example, the `apply` function performs these operations on each block of the input `bigimage`:

- Convert the block to the L*a*b* color space.
- Filter the block using `imnlmfilt`.
- Convert the block back to the RGB color space.

The `apply` function recombines the output blocks to form a new `bigimage`.

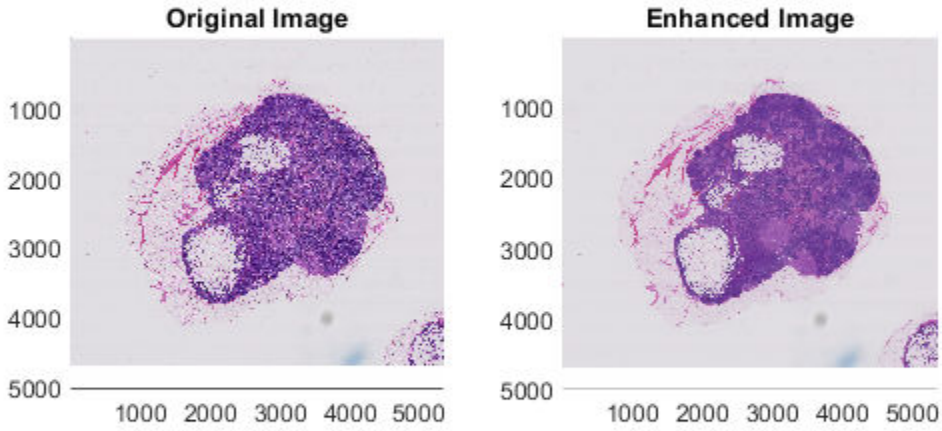
```
bim_enhanced = apply(bim,1, ...
    @(block)lab2rgb(imnlmfilt(rgb2lab(block),'DegreeOfSmoothing',15)));
```

Display the original image on the left side of a figure window using the `bigimageshow` function.

```
figure
ha1 = subplot(1,2,1);
bigimageshow(bim,'ResolutionLevel',1);
title("Original Image")
```

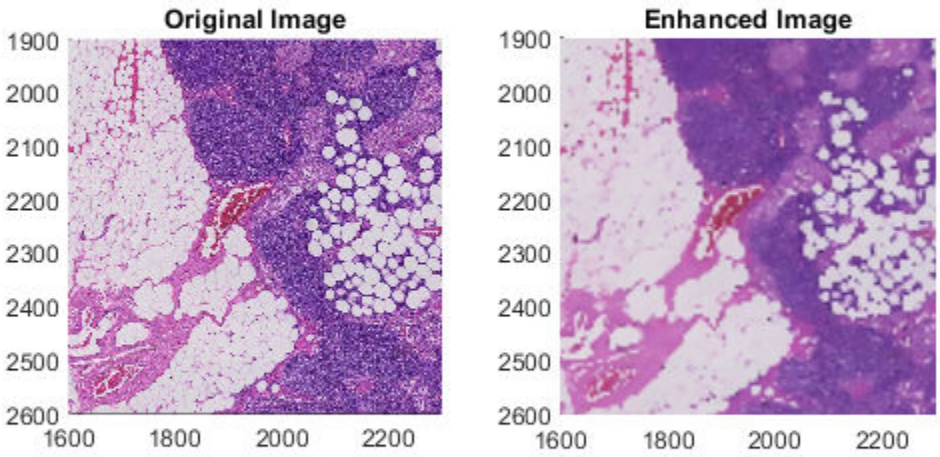
Display the enhanced image on the right side of the figure window.

```
ha2 = subplot(1,2,2);
bigimageshow(bim_enhanced);
title("Enhanced Image")
```



Ensure that both displays show the same extents, then zoom in on a feature.

```
linkaxes([ha1,ha2]);
xlim([1600,2300])
ylim([1900,2600])
```



Input Arguments

`bigimg` — Big image

`bigimage` object

Big image, specified as a `bigimage` object.

`level` — Resolution level

positive integer

Resolution level, specified as a positive integer that is less than or equal to the number of resolution levels of `bigimg`.

`fun` — Function handle

handle

Function handle, specified as a handle. For more information, see “Create Function Handle”.

Function Inputs

The function `fun` must accept at least one block as input.

Optionally, the function can accept additional inputs that are not blocks. To perform processing with non-block inputs, you must call the `apply` function and specify `fun` as an anonymous function. For more information, see “Anonymous Functions”.

The table shows sample function signatures for different types of input to `fun`. The table also shows sample syntax to use when calling `apply`.

Input Type	Function Signature	Example of Calling apply

Single block	`function outblock = myfun(inblock)`
	`  ...`
	`end`
		`newbig = apply(bigimg,level,@myfun);`

Two blocks	`function outblock = myfun(inblock1,inblock2)`
	`  ...`
	`end`
		Specify the second `bigimage` after the handle to the function `myfun`.
		
		`newbig = apply(bigimg,level,@myfun,otherbig);`

One block and one non-block	`function outblock = myfun(inblock,nonblock)`
	`  ...`
	`end`
		This example passes a scalar value `37` to the function `myfun`:
		
		`c = 37;`
		`mynewbig = apply(mybigimg,level,@(x) myfun(x,c`

Function Outputs

The function `fun` typically returns one or more image blocks of the same size as the input block. In this case, `apply` recombines the blocks and returns a `bigimage`. If you specify the `BorderSize` argument of `apply` and desire a `bigimage` output, then `apply` will crop the border from the output blocks. You can also crop the block directly within `fun`.

All of the examples in the above table demonstrate a function signature that returns a single block. However, the function `fun` can also return structs or other non-image outputs.

The table shows sample function signatures for different types of output of `fun`. The table also shows sample syntaxes to use when calling `apply`.

Output Type	Sample Processing Function	Example of Calling apply

Block of the same size as the input block	`function sameSizedBlock = myfun(inblock)`
	`  sameSizedBlock = imgaussfilt(inblock);`
	`end`
		`bigimageOutput = apply(bigimg,level,@myfun);`
		`bigimageOutput` is a single-resolution `bigimage`. In this example, `bigimageOutput` has the same number of channels and data type as the input `bigimg`.

Multiple blocks of the same size as the input block	`function [sameSizedBlock,maskBlock] = myfun(inblock)`
	`  sameSizedBlock = rgb2lightness(inblock);`
	`  maskBlock = imbinarize(sameSizedBlock);`
	`end`
		`[bigimageOutput1,bigimageOutput2] = apply(bigi`
		`bigimageOutput1` is a single-resolution `bigimage`. In this example, if `bigimgRGB` contains a color image, then `bigimageOutput1` has a different number of channels than the input `bigimgRGB`.
		
		`bigimageOutput2` is a single-resolution `bigimage` that has a different number of channels and a different data type than `bigimgRGB`.

Output Type	Sample Processing Function	Example of Calling apply

Non-image	`function nonimageOutput = myfun(inblock)`
	`  nonimageOutput = mean(inblock(:))`
	`end`
		`cellArrayOutput = apply(bigimg,level,@myfun);`
		`cellArrayOutput` is a cell array whose elements are the non-image output of each block. `cellArrayOutput` has one additional column that specifies the (x,y) origin of each block as a 1-by-2 vector.

Struct	`function structOutput = myfun(inblock)`
	`  structOutput.num = numel(inblock);`
	`  structOutput.mean = mean(inblock(:));`
	`  structOutput.max = max(inblock(:));`
	`end`
		`tableOutput = apply(mybigimg,level,@myfun);`
		`tableOutput` is a table with four variables: `num`, `mean`, `max`, and `BlockOrigin`. The `BlockOrigin` variable specifies the (x,y) origin of each block as a 1-by-2 vector.

Multiple outputs	`function [out1,out2,out3,out4,out5] = myfun(inblock)`
	`  % non-image output`
	`  out1 = min(inblock(:));`
	`  % image output of same size as input block`
	`  out2 = imgaussfilt(out2);`
	`  out3 = imbinarize(inblock);`
	`  % struct output`
	`  out4.originalMean = mean(inblock(:));`
	`  out4.filteredMean = mean(out2(:));`
	`  out4.fractionTrue = sum(out3(:))/numel(out3);`
	`  % non-image output`
	`  out5 = out4.fractionTrue;`
	`end`
		`[c1,b2,b3,t4] = apply(mybigimg,level,@myfun);`
		`c1` is a cell array because the first output of `myfun` is a non-image. `b2` and `b3` are `bigimage`s because the second and third outputs of `myfun` are image blocks of the same size as the input block. `t4` is a table because the fourth output of `myfun` is a struct.
		The `apply` function ignores the fifth output argument of `myfcn` because only four output arguments are specified in the call to `apply`.

`extraImages` — Additional input big images

vector of `bigimage` objects

Additional input big images, specified as a vector of `bigimage` objects. Each big image must have the same spatial extents as `bigimg`, but the blocks do not need to have the same size. The big images may have different values of the ClassUnderlying and Channels properties.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `newbig = apply(bigimg,level,@myfun,'UseParallel',true);`

`BatchSize` — Number of blocks supplied to `fun`

1 (default) | positive integer

Number of blocks supplied to the processing function `fun` in each batch, specified as the comma-separated pair consisting of `'BatchSize'` and a positive integer. When `BatchSize` is greater than 1, `PadPartialBlocks` must be `true`.

When BatchSize is greater than 1, apply supplies blocks as a numrows-by-numcols-by-channels-by-BatchSize array. For apply to return a bigimage, fun must return an array of the same size. For apply to return a cell array or table, fun must return a cell array of length BatchSize with one element for each block.

BlockSize — Block size
1-by-2 vector of positive integers

Block size, specified as the comma-separated pair consisting of 'BlockSize' and a 1-by-2 vector of positive integers of the form [numrows numcols]. If you specify 'BlockSize', then apply passes blocks of size [numrows numcols] to the processing function, fun. apply passes all channels of the block to fun.

BorderSize — Border size
[0 0] (default) | 1-by-2 vector of nonnegative integers

Border size, specified as the comma-separated pair consisting of 'BorderSize' and a 1-by-2 vector of nonnegative integers of the form [numrows numcols]. The function adds numrows rows above and below each block and numcols columns to the left and right of each block with data from the neighboring blocks. For blocks that lie on the edge of an image, data is padded according to PadMethod. By default, no border is added to blocks.

DisplayWaitbar — Display wait bar
true (default) | false

Display wait bar, specified as the comma-separated pair consisting of 'DisplayWaitbar' and true or false. When true, the apply function displays a wait bar for long running operations. If you close the wait bar, then apply returns a partial output, if available.

Data Types: logical

IncludeBlockInfo — Include block information
false (default) | true

Include block information, specified as the comma-separated pair consisting of 'IncludeBlockInfo' and false or true. When true, apply includes a struct as the last input to the processing function, fun. The struct has these fields that describe spatial referencing information about the block.

Field	Description
BlockStartWorld	World coordinates of the center of the top-left pixel of the block, excluding any border or padding.
BlockEndWorld	World coordinates of the center of the bottom-right pixel of the block, excluding any border or padding.
DataStartWorld	World coordinates of the center of the top-left pixel of the block, including any border or padding.
DataEndWorld	World coordinates of the center of the bottom-right pixel of the block, including any border or padding.

If BatchSize is greater than 1, then the values in the struct are arrays of length BatchSize.

Data Types: logical

`InclusionThreshold` — Inclusion threshold

`0.5` (default) | number in the range [0, 1]

Inclusion threshold for mask blocks, specified as the comma-separated pair consisting of `'InclusionThreshold'` and a number in the range [0, 1]. The inclusion threshold indicates the minimum fraction of nonzero pixels in a mask block required to process the image block.

- When the inclusion threshold is `0`, then `apply` processes a block when at least one pixel in the corresponding mask block is nonzero.
- When the inclusion threshold is `1`, then `apply` only processes a block when all pixels in the mask block are nonzero.

`Mask` — Mask

`[]` (default) | single-resolution `bigimage` object

Mask, specified as the comma-separated pair consisting of `'Mask'` and a `bigimage` object of the same size as `bigimg` and with a ClassUnderlying property value of `logical`.

The `apply` function only processes blocks that overlap with nonzero blocks of the mask. If you also specify `InclusionThreshold`, then the block to process must overlap with a minimum percentage of nonzero pixels in a mask block. If an image block sufficiently overlaps a mask block, then `apply` sends the entire image block to the processing function `fun`, and `fun` processes all pixels within the block. `fun` cannot access the mask directly.

An input block can overlap multiple mask blocks when the image is coarser than the mask or when the edge of the block does not align with the mask blocks. If an input block overlaps multiple mask blocks, then `apply` selects the mask that overlaps with the center of the input block.

`OutputFolder` — Location to save output `bigimages`

character vector

Location to save output `bigimage`s, specified as the comma-separated pair consisting of `'OutputFolder'` and `false` or `true`. Parallel processing requires Parallel Computing Toolbox™.

`PadMethod` — Pad method

numeric scalar | string scalar | `'replicate'` | `'symmetric'`

Pad method of incomplete edge blocks, specified as the comma-separated pair consisting of `'PadMethod'` and one of these values. By default, `apply` pads numeric blocks with `0` and categorical blocks with `missing`.

Value	Meaning
numeric scalar	Pad numeric array with elements of constant value.
string scalar	Pad categorical array with the specified class in the Classes property of the `bigimage`.
`'replicate'`	Pad by repeating border elements of array.
`'symmetric'`	Pad array with mirror reflections of itself.

`PadPartialBlocks` — Pad partial blocks

`false` (default) | `true`

Pad partial blocks, specified as the comma-separated pair consisting of `'PadPartialBlocks'` and `false` or `true`. Partial blocks arise when the image size is not exactly divisible by `BlockSize`. If they exist, partial blocks lie along the right and bottom edge of the image.

- When `false`, the processing function `fun` operates on partial blocks without padding and can return blocks smaller than `BlockSize`.
- When `true`, the `apply` function pads partial blocks using the specified `PadMethod`. The processing function `fun` operates on and returns full-sized blocks.

Set `PadPartialBlocks` to `true` when `BatchSize` is greater than 1.

Data Types: `logical`

`UseParallel` — Use parallel processing
`false` (default) | `true`

Use parallel processing, specified as the comma-separated pair consisting of `'UseParallel'` and `false` or `true`. Parallel processing requires Parallel Computing Toolbox.

When you specify `UseParallel` as `true`, then MATLAB automatically opens a parallel pool based on default parallel settings. `apply` processes the `bigimage` blocks across the available workers. The `DataSource` property of all input `bigimage`s should be valid paths on each of the parallel workers. If relative paths are used, then ensure workers and the client process are on the same working directory. If workers do not share the same file system as the client process, then specify `OutputFolder`.

Data Types: `logical`

Output Arguments

`newbig` — Processed big image
`bigimage` object

Processed big image, returned as a `bigimage` object with a single resolution level. The number of rows and columns of `newbig` is equal to the number of rows and columns of the input big image `bigimg` at the specified resolution `level`. However, `newbig` can have a different number of channels and a different underlying class.

`other` — Additional output
`bigimage` object | table | cell array

Additional output from processing function `fun`, returned as one of the following. If `fun` returns multiple output arguments, then `apply` can return the same number or fewer output arguments.

Value	Occurrence
`bigimage` object	`apply` returns a `bigimage` when the corresponding output argument of `fun` returns data as a numeric or logical array of the same size as the input block, or the padded size of the input block when you specify the `BorderSize` argument. The returned `bigimage` has a single resolution level, but it can have a different number of channels and underlying class.

Value	Occurrence
table	`apply` returns a `table` when the corresponding output argument of `fun` returns data as a `struct`. The table has an additional `BlockOrigin` variable that specifies the (x,y) origin of each block as a 1-by-2 vector in world coordinates.
cell array	`apply` returns a cell array when the corresponding output argument of `fun` returns data as a non-image. The cell array has an additional column that specifies the (x,y) origin of each block as a 1-by-2 vector in world coordinates.

Tips

- Setting `BatchSize` as greater than 1 is useful to optimally load GPUs when running inference deep learning networks inside the processing function `fun`.

Algorithms

`apply` passes data to the processing function, `fun`, one block at a time in the most efficient order to traverse the data (often row-major block order). `apply` processes each block only once.

It is inefficient to perform many processing operations by making multiple calls to `apply` because the data must be traversed multiple times. To optimize processing time, define `fun` such that it performs multiple processing operations. This minimizes reading and writing overhead and ensures data locality. You can further reduce processing time at a particular level by using masks created at coarser resolution levels to exclude regions of the image from processing.

Compatibility Considerations

`apply` function is not recommended

Not recommended starting in R2021a

The `apply` function of the `bigimage` object is not recommended. Use the `apply` function of the `blockedImage` object instead. The `blockedImage` object offers several advantages including extension to N-D processing, a simpler interface, and custom support for reading and writing nonstandard image formats.

Although there are no plans to remove the `bigimage` object and its `apply` function at this time, switch to `blockedImage` to take advantage of the additional capabilities and flexibility.

See Also

`apply` | `blockedImage`

Topics

“Process Blocked Images Efficiently Using Partial Images or Lower Resolutions”
“Process Blocked Images Efficiently Using Mask”
“Warp Blocked Image at Coarse and Fine Resolution Levels”

Introduced in R2019b

isequal

(Not recommended) Compare two `bigimage` objects for equality

Note The `isequal` function of the `bigimage` object is not recommended. Use the `blockedImage` object instead. For more information, see “Compatibility Considerations”.

Syntax

```
tf = isequal(bigimg1,bigimg2)
```

Description

`tf = isequal(bigimg1,bigimg2)` returns `true` if big images `bigimg1` and `bigimg2` have the same spatial referencing, underlying data type, and block size. Pixel values are only compared if the metadata is equal, and the comparison ends early when unequal pixel values are found. `isequal` does not consider file names, masks, and other class properties.

Input Arguments

`bigimg1` — First big image

`bigimage` object

First big image, specified as a `bigimage` object.

`bigimg2` — Second big image

`bigimage` object

Second big image, specified as a `bigimage` object.

Output Arguments

`tf` — Big images are equal

logical scalar

Big images `bigimg1` and `bigimg2` are equal, returned as a logical scalar.

Data Types: `logical`

Compatibility Considerations

`isequal` function is not recommended

Not recommended starting in R2021a

The `isequal` function of the `bigimage` object is not recommended. Use a `blockedImage` object instead. The `blockedImage` object offers several advantages including extension to N-D processing, a simpler interface, and custom support for reading and writing nonstandard image formats.

Although there are no plans to remove the `bigimage` object and its `isequal` at this time, switch to the `blockedImage` to take advantage of the additional capabilities and flexibility.

See Also

`blockedImage`

Introduced in R2019b

getBlock

(Not recommended) Read block of `bigimage` object

Note The `getBlock` function of the `bigimage` object is not recommended. Use the `getBlock` function associated with the `blockedImage` object instead. For more information, see “Compatibility Considerations”.

Syntax

```
data = getBlock(bigimg,level,locationWorld)
```

Description

`data = getBlock(bigimg,level,locationWorld)` reads the big image data in `bigimg` at the specified resolution level, and returns pixel data for the entire block that contains coordinate `locationWorld`.

Examples

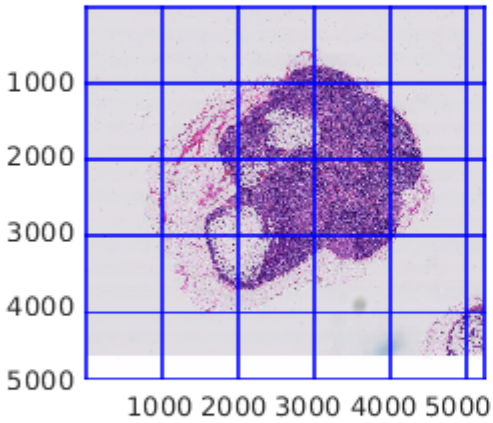
Select and Display bigimage Block Interactively

Create a `bigimage` using a modified version of image "tumor_091.tif" from the CAMELYON16 data set. The original image is a training image of a lymph node containing tumor tissue. The original image has eight resolution levels, and the finest level has resolution 53760-by-61440. The modified image has only three coarse resolution levels. The spatial referencing of the modified image has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = bigimage('tumor_091R.tif');
```

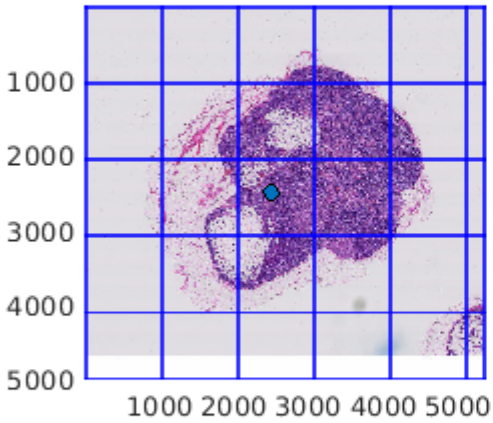
Display the `bigimage` by using the `bigimageshow` function. Overlay a grid that shows the block boundaries at the finest resolution level.

```
hb = subplot(1,2,1);
bigimageshow(bim,'GridVisible','on','GridLevel',1);
```



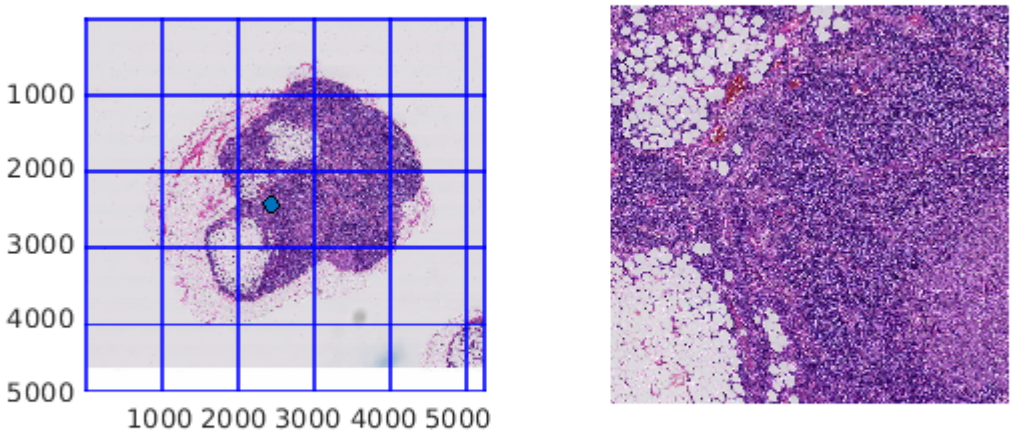
Specify the (x,y) coordinate of a block to display. Get the block containing the coordinate. Add a `Point` ROI over the displayed `bigimage` at the specified coordinate.

```
coord = [2500,2500];
blk = getBlock(bim,1,coord);
hp = drawpoint(hb,'Position',coord);
```



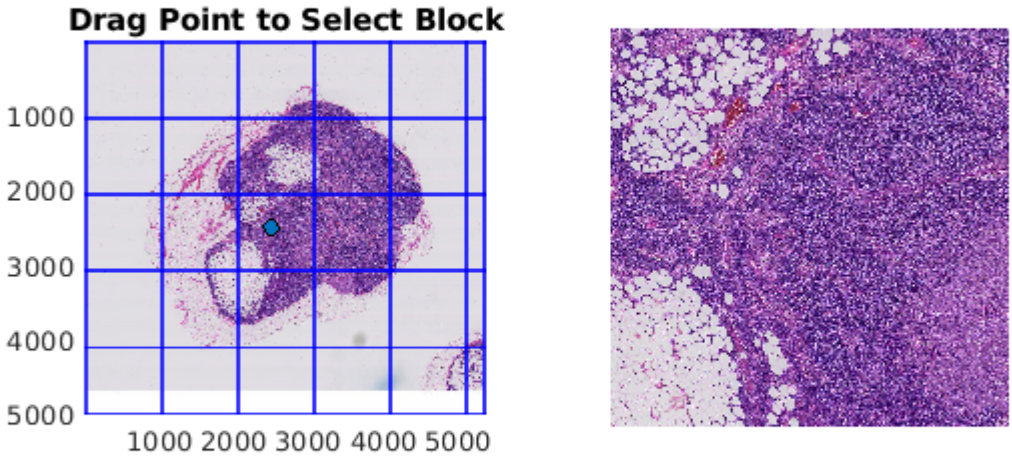
In the figure, display the block next to the entire `bigimage`. You can use `imshow` to display the block because the block fits in memory and has a single resolution level.

```
ha = subplot(1,2,2);
imshow(blk,'Parent',ha)
```



Add a listener to the `Point` ROI. When you drag the ROI with the mouse, the figure is updated to show the block containing the current ROI coordinates.

```
title(hb,'Drag Point to Select Block');
addlistener(hp, ...
    'ROIMoved',@(~,~) imshow(getBlock(bim,1,hp.Position),'Parent',ha));
```



Input Arguments

`bigimg` — Big image

`bigimage` object

Big image, specified as a `bigimage` object.

`level` — Resolution level

positive integer

Resolution level, specified as a positive integer that is less than or equal to the number of resolution levels of `bigimg`.

`locationWorld` — Coordinate of a point

1-by-2 numeric vector

Coordinate of a point, specified as a 1-by-2 numeric vector of the form `[x y]`. The location is specified in world coordinates, which are the pixel locations relative to the highest resolution level. The position must be a valid position within `bigimg`.

Output Arguments

`data` — Pixel data

numeric array

Pixel data, returned as a numeric array of the same data type as the big image, `bigimg.ClassUnderlying`.

Compatibility Considerations

`getBlock` function is not recommended

Not recommended starting in R2021a

The `getBlock` function of the `bigimage` object is not recommended. Use the `getBlock` function of the `blockedImage` object instead. The `blockedImage` object offers several advantages including extension to N-D processing, a simpler interface, and custom support for reading and writing nonstandard image formats.

Although there are no plans to remove the `bigimage` object and its `getBlock` function at this time, switch to `blockedImage` to take advantage of the additional capabilities and flexibility.

See Also

`getBlock` | `blockedImage`

Introduced in R2019b

getFullLevel

(Not recommended) Get all data in one level of `bigimage` object

Note The `getFullLevel` function of the `bigimage` object is not recommended. Use the `gather` function associated with the `blockedImage` object instead. For more information, see “Compatibility Considerations”.

Syntax

```
I = getFullLevel(bigimg)
I = getFullLevel(bigimg,level)
```

Description

`I = getFullLevel(bigimg)` reads the big image data in `bigimg` at the coarsest resolution level and returns the single-resolution image `I`.

`I = getFullLevel(bigimg,level)` reads the big image data in `bigimg` at the specified resolution level and returns the single-resolution image `I`.

Examples

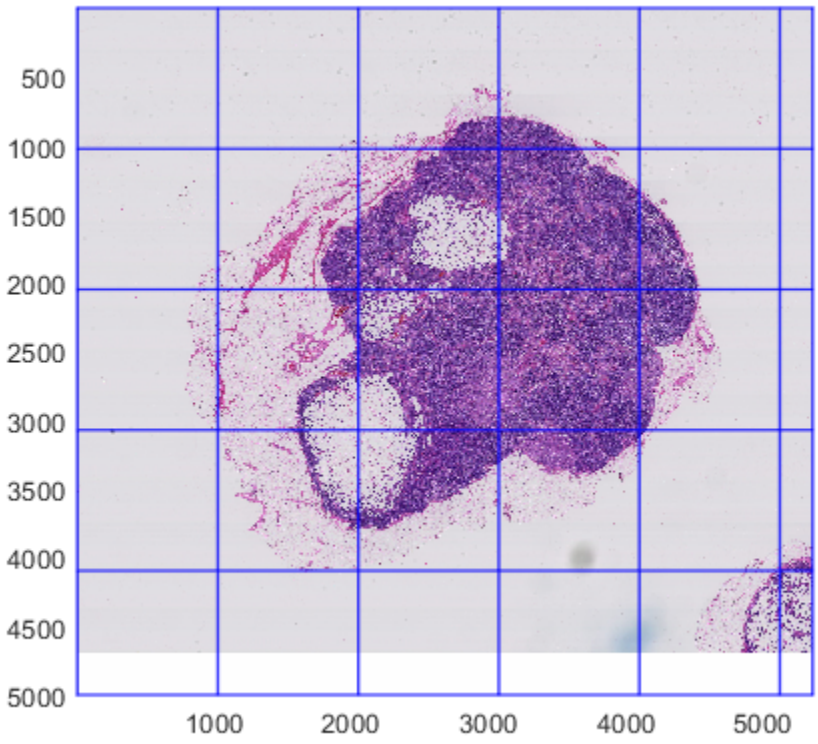
Create Single-Resolution Mask from Multiresolution Blocked Image

Create a blocked image from the sample image tumor_091R.tif. This sample image is a training image of a lymph node containing tumor tissue from the CAMELYON16 data set. The image has been modified to have three coarse resolution levels, and has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = blockedImage('tumor_091R.tif');
```

Display the entire blocked image at the finest resolution level, including a grid of the block boundaries.

```
bshow = bigimageshow(bim,'ResolutionLevel','fine', ...
    'GridVisible','on','GridLevel',1);
```



Create a mask of the coarsest resolution level.

First create a single-resolution image of the coarsest resolution level. By default, the `gather` function gets data from the coarsest resolution level.

```
imcoarse = gather(bim);
```

Convert the coarse image to grayscale.

```
graycoarse = rgb2gray(imcoarse);
```

Binarize the grayscale image. In the binarized image, the object of interest is black and the background is white.

```
bwcoarse = imbinarize(graycoarse);
```

Take the complement of the binarized image. The resulting mask follows the convention in which the object of interest is white and the background is black.

```
mask = imcomplement(bwcoarse);
```

Create a blocked image containing the mask.

Use the same spatial referencing as the original blocked image. Determine the coarsest resolution level and capture the spatial referencing information of the blocked image at the first two dimensions at that level.

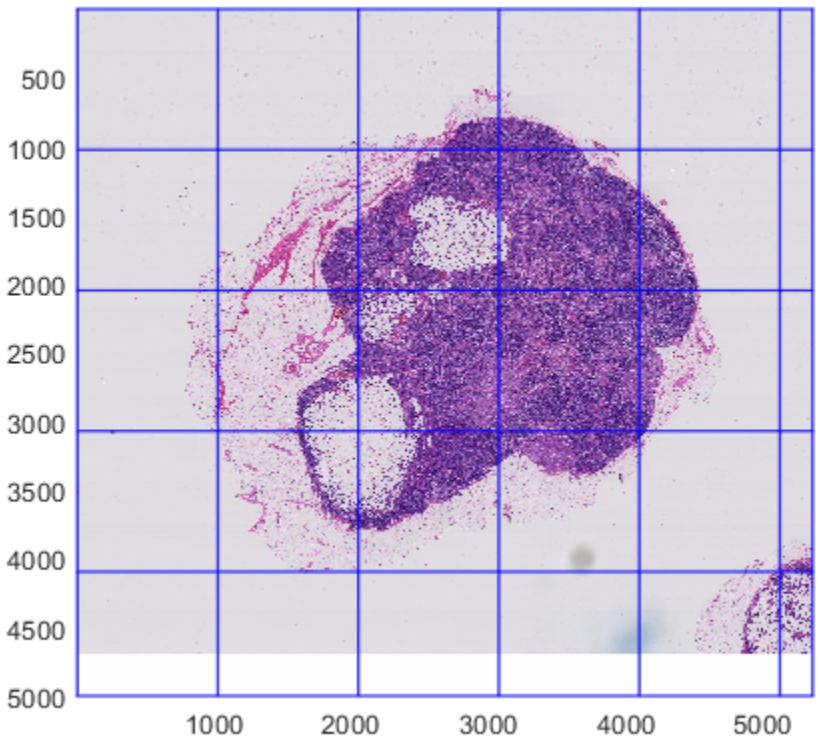
```
coarsestLevel = bim.NumLevels;
originalWorldStartCoarsest = bim.WorldStart(coarsestLevel,1:2);
originalWorldEndCoarsest = bim.WorldEnd(coarsestLevel,1:2);
```

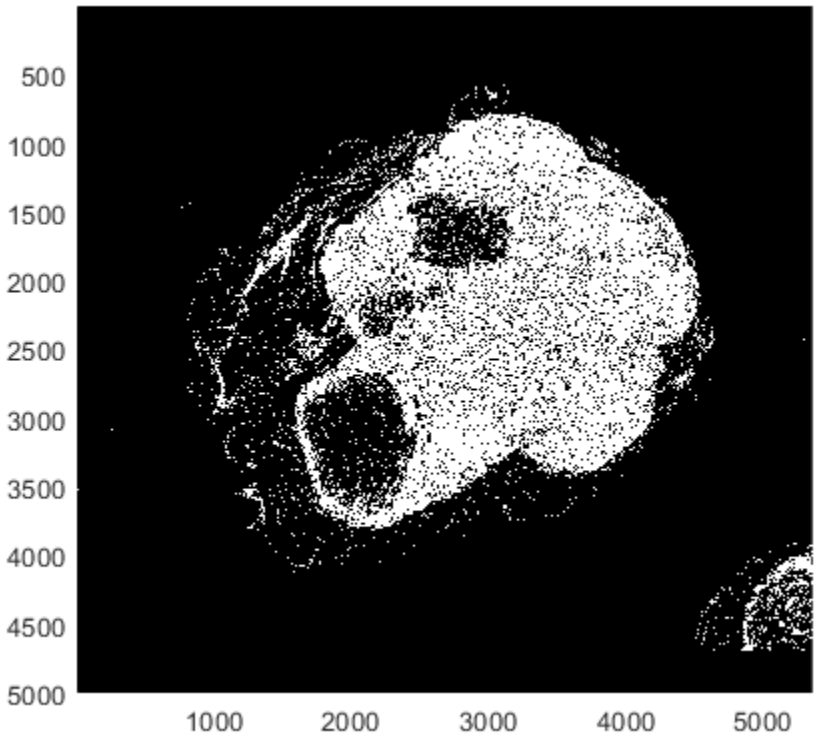
Create the blocked image for the mask.

```
bmask = blockedImage(mask,'WorldStart',originalWorldStartCoarsest, ...
   'WorldEnd',originalWorldEndCoarsest);
```

Display the mask image.

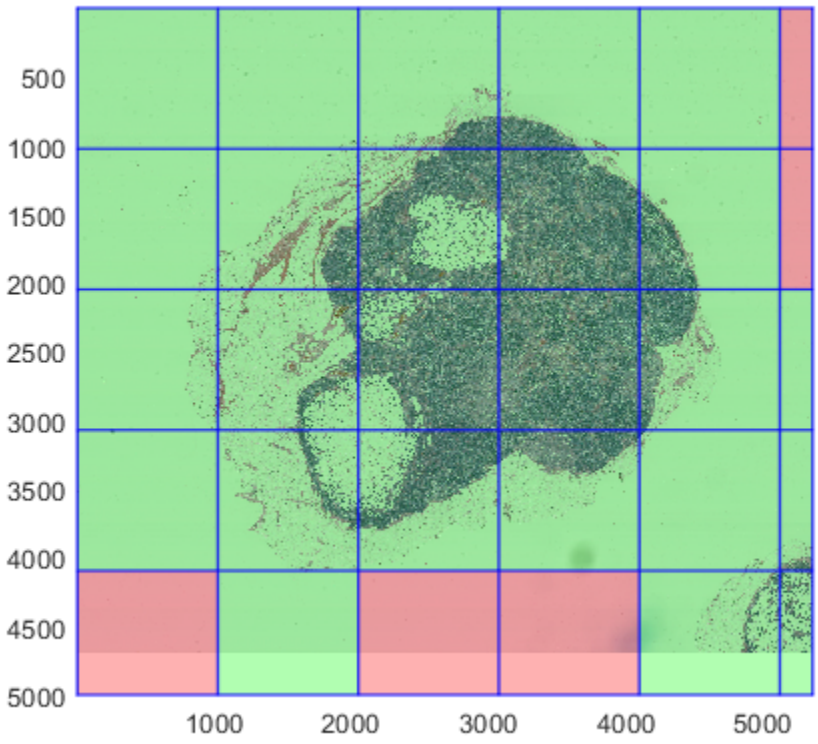
```
figure
bigimageshow(bmask)
```





Overlay the mask on the display of the original blocked image using the `showmask` function. To highlight all blocks that contain at least one nonzero mask pixel, specify an inclusion threshold of `0`.

```
showmask(bshow,bmask,'InclusionThreshold',0)
```



Input Arguments

`bigimg` — Big image

`bigimage` object

Big image, specified as a `bigimage` object.

`level` — Resolution level

positive integer

Resolution level, specified as a positive integer that is less than or equal to the number of resolution levels of `bigimg`. The default level is the coarsest resolution level, `bigimg.CoarsestResolutionLevel`.

Output Arguments

`I` — Single-resolution image

numeric array

Single-resolution image, returned as a numeric array.

Tips

- Check the `LevelSizes` property of the input big image `bigimg` to confirm that the size of image data at the specified level is small enough to fit in memory.

Compatibility Considerations

`getFullLevel` function is not recommended

Not recommended starting in R2021a

The `getFullLevel` function of the `bigimage` object is not recommended. Use the `gather` function of the `blockedImage` object instead. The `blockedImage` object offers several advantages including extension to N-D processing, a simpler interface, and custom support for reading and writing nonstandard image formats.

Although there are no plans to remove the `bigimage` object and its `getFullLevel` function at this time, switch to the `blockedImage` object and the `gather` function to take advantage of the additional capabilities and flexibility.

See Also

`gather` | `blockedImage`

Introduced in R2019b

getRegion

(Not recommended) Read arbitrary region of `bigimage` object

Note The `getRegion` function of the `bigimage` object is not recommended. Use the `getRegion` function associated with the `blockedImage` object instead. For more information, see “Compatibility Considerations”.

Syntax

```
data = getRegion(bigimg,level,regionStartWorld,regionEndWorld)
```

Description

`data = getRegion(bigimg,level,regionStartWorld,regionEndWorld)` reads the big image data in `bigimg` at the specified resolution level. The function returns all pixels whose extents touch or lie within the bounds of the rectangular region specified by `regionStartWorld` and `regionEndWorld`, inclusive.

Examples

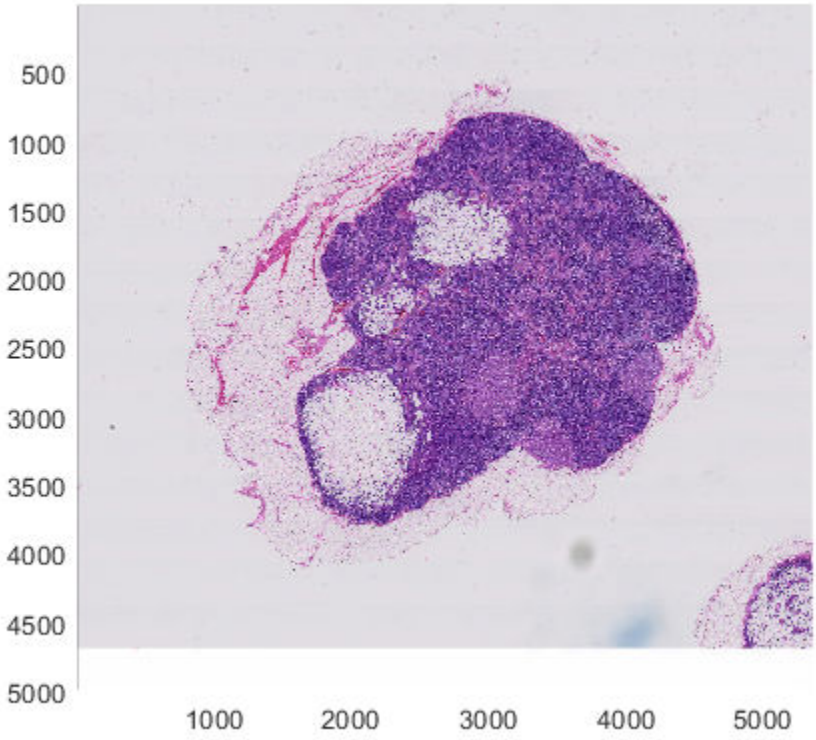
Get Region of Big Image at Different Resolution Levels

Create a `bigimage` using a modified version of image "tumor_091.tif" from the CAMELYON16 data set. The original image is a training image of a lymph node containing tumor tissue. The original image has eight resolution levels, and the finest level has resolution 53760-by-61440. The modified image has only three coarse resolution levels. The spatial referencing of the modified image has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = bigimage('tumor_091R.tif');
```

Display the entire `bigimage` at the finest resolution level.

```
bshow = bigimageshow(bim);
```



Define a rectangular region by specifying the starting and ending coordinates in the horizontal and vertical direction relative to the finest resolution level.

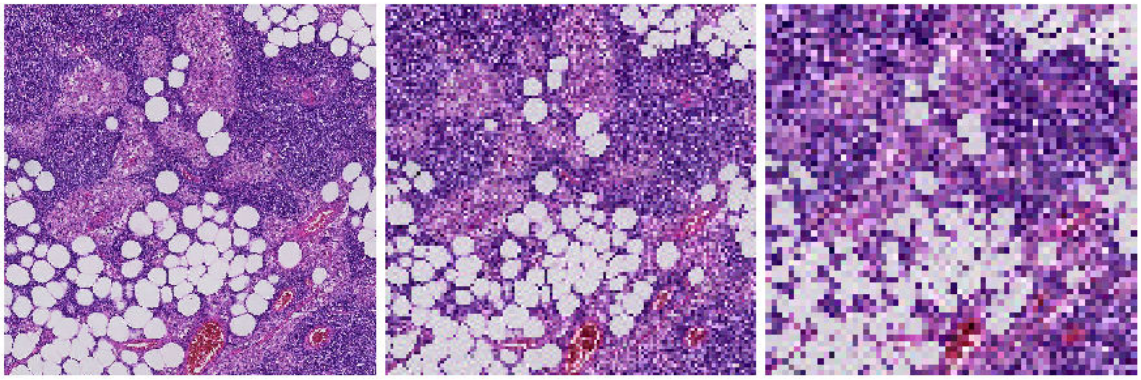
```
xyStart = [2100,1800];
xyEnd = [2600,2300];
```

Get the region of the `bigimage` at each resolution level.

```
imL1 = getRegion(bim,1,xyStart,xyEnd);
imL2 = getRegion(bim,2,xyStart,xyEnd);
imL3 = getRegion(bim,3,xyStart,xyEnd);
```

Display the three regions in a montage. The finest resolution level is on the left and the coarsest resolution level is on the right.

```
montage({imL1,imL2,imL3},'Size',[1 3], ...
    'BorderSize',5,'BackgroundColor','w');
```



Input Arguments

`bigimg` — Big image
`bigimage` object

Big image, specified as a `bigimage` object.

`level` — Resolution level
positive integer

Resolution level, specified as a positive integer that is less than or equal to the number of resolution levels of `bigimg`.

`regionStartWorld` — Top-left coordinates of rectangular region
1-by-2 numeric vector

Top-left coordinates of the rectangular region to read, specified as a 1-by-2 numeric vector of the form `[x y]`. The location is specified in world coordinates, which are the pixel locations relative to the highest resolution level.

`regionEndWorld` — Bottom-right coordinates of rectangular region
1-by-2 numeric vector

Bottom-right coordinates of the rectangular region to read, specified as a 1-by-2 numeric vector of the form `[x y]`. The location is specified in world coordinates, which are the pixel locations relative to the highest resolution level.

Output Arguments

`data` — Pixel data
numeric array

Pixel data, returned as a numeric array of the same data type as the big image, `bigimg.ClassUnderlying`.

Compatibility Considerations

`getRegion` function is not recommended
Not recommended starting in R2021a

The `getRegion` function of the `bigimage` object is not recommended. Use the `getRegion` function of the `blockedImage` object instead. The `blockedImage` object offers several advantages including extension to N-D processing, a simpler interface, and custom support for reading and writing nonstandard image formats.

Although there are no plans to remove the `bigimage` object and its `getRegion` function at this time, switch to `blockedImage` to take advantage of the additional capabilities and flexibility.

See Also

`getRegion` | `blockedImage`

Introduced in R2019b

setBlock

(Not recommended) Put data in specific block of `bigimage` object

Note The `setBlock` function of the `bigimage` object is not recommended. Use the `setBlock` function associated with the `blockedImage` object instead. For more information, see “Compatibility Considerations”.

Syntax

```
setBlock(bigimg,level,locationWorld,data)
```

Description

`setBlock(bigimg,level,locationWorld,data)` sets the pixel data in the block of big image `bigimg` that contains coordinate `locationWorld` at the specified resolution level.

Examples

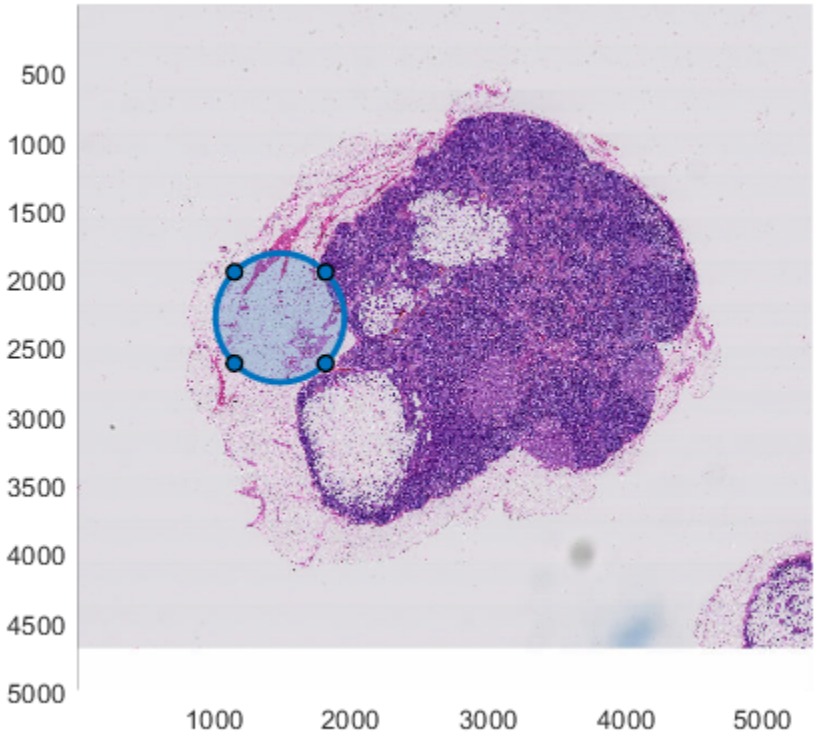
Set Blocks of Writeable Big Image

Create a `bigimage` using a modified version of image "tumor_091.tif" from the CAMELYON16 data set. The original image is a training image of a lymph node containing tumor tissue. The original image has eight resolution levels, and the finest level has resolution 53760-by-61440. The modified image has only three coarse resolution levels. The spatial referencing of the modified image has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = bigimage('tumor_091R.tif');
```

Display the `bigimage`, then create a circle ROI over the displayed image.

```
h = bigimageshow(bim);
hROI = drawcircle(gca,'Radius',470,'Position',[1477 2284]);
```



Choose the level at which to create a writeable `bigimage`. Level 3 is the coarsest resolution level.

```
maskLevel = 3;
```

Get the spatial referencing and pixel extents from the specified level.

```
ref = bim.SpatialReferencing(maskLevel);
pixelExtent = [ref.PixelExtentInWorldX,ref.PixelExtentInWorldY];
```

Create a writeable `bigimage` by specifying the spatial referencing instead of image data. This big image has one channel and is of data type `logical`.

```
bmask = bigimage(ref,1,'logical');
```

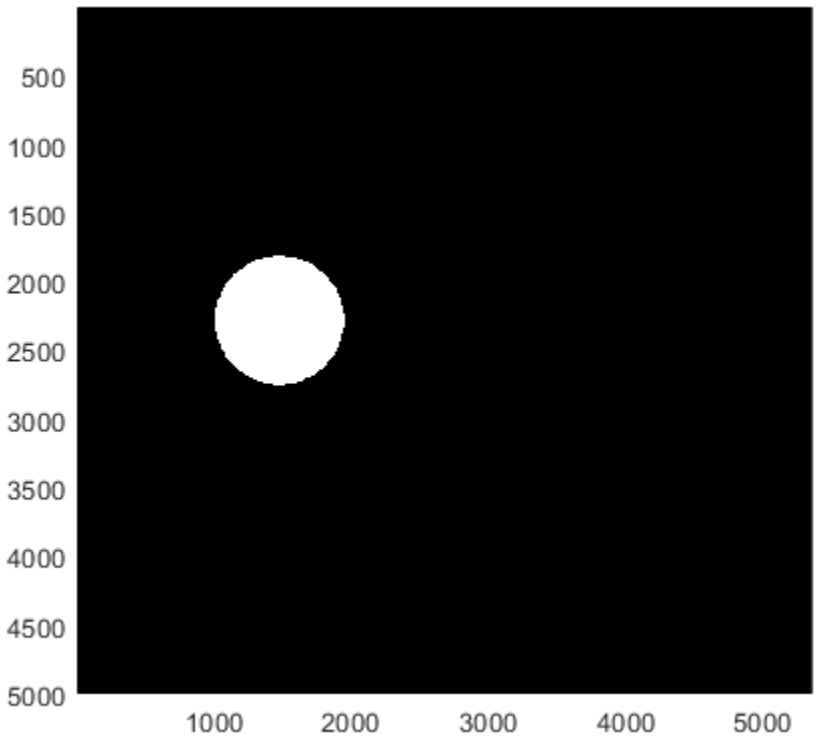
Loop through all blocks in the writeable big image to create a mask image. For each block, set the pixel values as `1` (`true`) for pixels inside the ROI and `0` (`false`) for pixels outside the ROI.

```
for cStart = 1:bmask.BlockSize(2):ref.ImageSize(2)
    for rStart = 1:bmask.BlockSize(1):ref.ImageSize(1)

        % Get the center of top left pixel of this block in world units.
        xyStart = [cStart,rStart].*pixelExtent;

        % Get the block size. The |'BlockSize'| property represents the
        % size as a 2-element vector of the form [row,column]. Switch the
        % order of the elements so that the block size is represented as
        % [x,y].
        bsize = bmask.BlockSize;
```

```
            numRows = bsize(1);
            numCols = bsize(2);

            % Determine which pixels have coordinates inside the ROI.
            roiPositions = hROI.Vertices;

            % Transform |roiPositions| from world coordinates to the intrinsic
            % image indices at the given resolution level.
            roiPositions = (roiPositions - xyStart) ./ pixelExtent + 1;

            blockMask = poly2mask(roiPositions(:,1),roiPositions(:,2), ...
                                  numRows, numCols);

            % Set the pixel values of the block.
            setBlock(bmask,1,xyStart,blockMask);
        end
    end
```

Display the mask.

```
figure
bigimageshow(bmask)
```



Input Arguments

`bigimg` — Big image

`bigimage` object

Big image, specified as a `bigimage` object.

`level` — Resolution level
positive integer

Resolution level, specified as a positive integer that is less than or equal to the number of resolution levels of `bigimg`.

`locationWorld` — Coordinate of a point
1-by-2 numeric vector

Coordinate of a point, specified as a 1-by-2 numeric vector of the form `[x y]`. The location is specified in world coordinates, which are the pixel locations relative to the highest resolution level. The position must be a valid position within `bigimg`.

`data` — Pixel data
numeric array

Pixel data, specified as a numeric array of the same data type as the big image, `bigimg.ClassUnderlying`. The first two dimensions of the data must match the block size at the specified `level`.

Tips

- Create a writeable `bigimage` by using a syntax that does not initialize image data. If you create a `bigimage` by specifying the file name, directory name, or variable name of image data, or by using the `apply` function, then the `bigimage` is not writeable and you cannot use the `setBlock` function.
- If the size of `data` is less than the block size `bigimg.BlockSize`, then `setBlock` pads the data with the default value, `bigimg.UnloadedValue`.
- `setBlock` trims data for partial edge blocks.

Compatibility Considerations

`setBlock` function is not recommended
Not recommended starting in R2021a

The `setBlock` function of the `bigimage` object is not recommended. Use the `setBlock` function of the `blockedImage` object instead. The `blockedImage` object offers several advantages including extension to N-D processing, a simpler interface, and custom support for reading and writing nonstandard image formats.

Although there are no plans to remove the `bigimage` object and its `setBlock` function at this time, switch to `blockedImage` to take advantage of the additional capabilities and flexibility.

See Also
`blockedImage` | `setBlock`

Introduced in R2019b

write

(Not recommended) Write `bigimage` object content to new file

Note The `write` function of the `bigimage` object is not recommended. Use the `write` function associated with the `blockedImage` object instead. For more information, see “Compatibility Considerations”.

Syntax

```
write(bigimg,filename)
write(bigimg,filename,'TIFFCompression',compression)
write(bigimg,dirname)
write( ___ ,Name,Value)
```

Description

`write(bigimg,filename)` writes a formatted version of big image `bigimg` to a TIFF file named `filename`. This syntax does not preserve the spatial referencing information of the big image.

`write(bigimg,filename,'TIFFCompression',compression)` also specifies the compression scheme for writing a formatted version of big image `bigimg` to a TIFF file named `filename`. This syntax does not preserve the spatial referencing information of the big image.

`write(bigimg,dirname)` writes a formatted version of big image `bigimg` to the directory named `dirname`. This syntax preserves the spatial referencing information of the big image.

`write( ___ ,Name,Value)` specifies additional options when writing categorical data using name-value pair arguments.

Examples

Write Big Image Data to Disk

Create a `bigimage` using a modified version of image "tumor_091.tif" from the CAMELYON16 data set. The original image is a training image of a lymph node containing tumor tissue. The original image has eight resolution levels, and the finest level has resolution 53760-by-61440. The modified image has only three coarse resolution levels. The spatial referencing of the modified image has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = bigimage('tumor_091R.tif');
```

Create a mask image from the coarsest resolution level, 3. The mask is `1` (`true`) for each pixel whose grayscale value is less than 100.

```
mask = apply(bim,3,@(im)rgb2gray(im)<100);
```

Write the mask image to a directory called 'maskDir'. The directory must not already exist. Before writing the mask image, check if the directory already exists, and if it does, delete it.

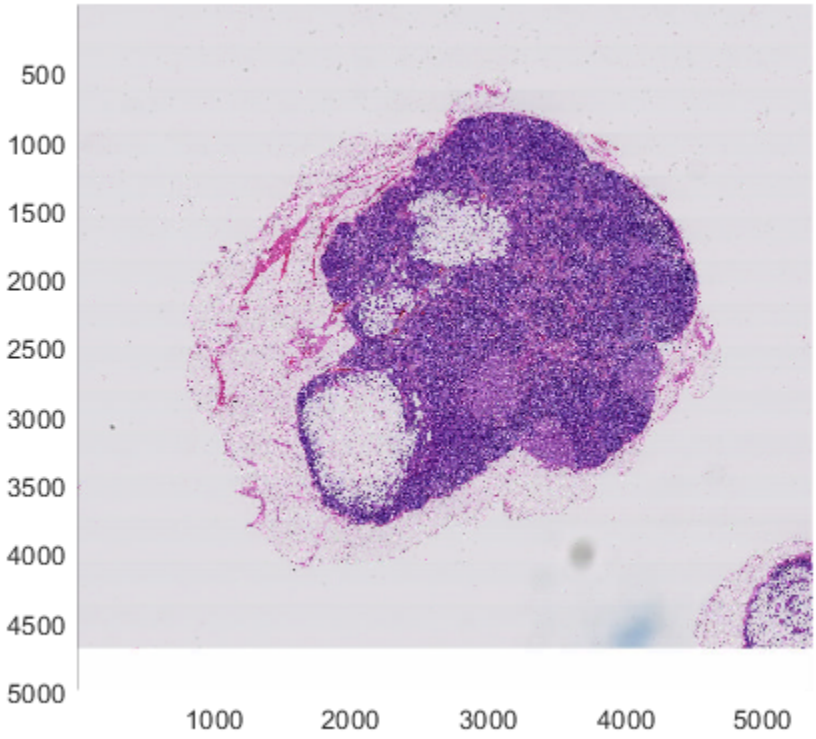
```
imageDir = 'maskDir';
if exist(imageDir,'dir')
    rmdir maskDir s;
end
write(mask,imageDir);
```

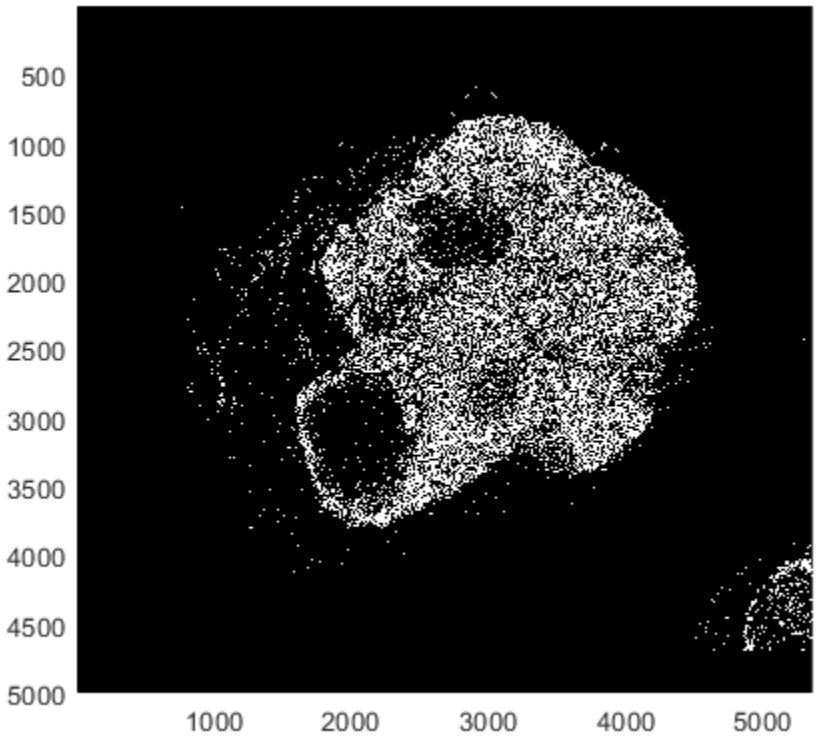
Load the mask image back into the workspace by creating a new `bigimage` from the data in the mask directory. The spatial referencing information of the mask is retained.

```
mask1 = bigimage('maskDir');
```

Display the original image and the mask image. The spatial referencing matches the original image, `bim`.

```
figure
bigimageshow(bim);
figure
bigimageshow(mask1);
```





Input Arguments

`bigimg` — Big image
`bigimage` object

Big image, specified as a `bigimage` object.

`filename` — File name
string | character vector

File name of written big image data, specified as a string or character vector. Supported file extensions are `'.tif'` and `'.tiff'`.

Data Types: `string`

`dirname` — Directory name
string | character vector

Directory name of written big image data, specified as a string or character vector.

Data Types: `string`

`compression` — TIFF compression scheme
`"LZW"` (default) | `"PackBits"` | `"Deflate"` | `"JPEG"` | `"None"`

TIFF compression scheme, specified as one of the following.

Compression Scheme	Description
"LZW"	Lempel-Ziv-Welch lossless compression
"PackBits"	PackBits lossless compression
"Deflate"	Adobe DEFLATE lossless compression
"JPEG"	JPEG-based lossy compression
"None"	No compression

Data Types: string

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Example: write(bigimg,filename,'Classes',["sky" "vegetation" "building"],'PixelLabelIDs',[1 2 3]) writes a categorical bigimage with three classes

Classes — Class names
string array | cell array of character vectors

Class names of categorical data, specified as the comma-separated pair consisting of 'Classes' and a string array or a cell array of character vectors. The default value is the value of the Classes property of the big image bigimg.

If a class has multiple pixel values in PixelLabelIDs, then write writes all instances of that class using the first pixel value.

Data Types: char | string

PixelLabelIDs — Pixel label IDs
d-element numeric vector | *d*-by-3 numeric array of data type uint8

Pixel label IDs that map pixel label values to categorical class names, specified as the comma-separated pair consisting of 'PixelLabelIDs' and one of the following.

- *d*-element numeric vector, where *d* is the number of classes
- *d*-by-3 numeric array of data type uint8. Each row contains a 3-element vector representing the RGB pixel value to associate with each class name. Use this format when the pixel label data is stored as an RGB image.

The data type of the written pixels matches the data type of PixelLabelIDs. The default value is the value of the PixelLabelIDs property of the big image bigimg.

If a class has multiple pixel values in PixelLabelIDs, then write writes all instances of that class using the first pixel value.

UndefinedID — Pixel label value for '<undefined>' categorical class
0 (default) | numeric scalar | 1-by-3 numeric vector

Pixel label value for the '<undefined>' categorical class and pixel values that do not exist in PixelLabelIDs, specified as the comma-separated pair consisting of 'UndefinedID' and a

numeric scalar or a 1-by-3 numeric vector. Do not specify this value as any of the values in `PixelLabelIDs`. The default value is the value of the UndefinedID property of the big image `bigimg`.

Compatibility Considerations

`write` function is not recommended

Not recommended starting in R2021a

The `write` function of the `bigimage` object is not recommended. Use the `write` function of the `blockedImage` object instead. The `blockedImage` object offers several advantages including extension to N-D processing, a simpler interface, and custom support for reading and writing nonstandard image formats.

Although there are no plans to remove the `bigimage` object and its `write` function at this time, switch to `blockedImage` to take advantage of the additional capabilities and flexibility.

See Also

`write` | `blockedImage`

Topics

“Create Labeled Blocked Image from ROIs and Masks”
“Preprocess Multiresolution Images for Training Classification Network”

Introduced in R2019b

bigimageDatastore

(Not recommended) Datastore to manage blocks of big image data

Note The `bigimageDatastore` object is not recommended. Use the `blockedImageDatastore` object instead. For more information, see “Compatibility Considerations”.

Description

A `bigimageDatastore` object manages a collection of image blocks that belong to one or more `bigimage` objects. A `bigimageDatastore` is analogous to an `imageDatastore`, which manages a collection of unrelated images.

Creation

Syntax

```
bigds = bigimageDatastore(images)
bigds = bigimageDatastore(images,levels)
bigds = bigimageDatastore(images,levels,Name,Value)

bigds = bigimageDatastore(images,'BlockLocationSet',blockLocationSet)
bigds = bigimageDatastore(images,'BlockLocationSet',blockLocationSet,
Name,Value)
```

Description

Create Datastore that Reads Blocks Over Entire Image

`bigds = bigimageDatastore(images)` creates a datastore that manages a collection of image blocks at the finest resolution level of one or more `bigimage` objects, Images.

`bigds = bigimageDatastore(images,levels)` creates a datastore that manages a collection of image blocks of one or more `bigimage` objects, Images, at the specified resolution levels, Levels.

`bigds = bigimageDatastore(images,levels,Name,Value)` also uses name-value pairs to set one or more “Properties” on page 1-162 except for `BlockLocationSet`. You can specify multiple name-value pairs. Enclose each property name in quotes.

Example: `bigimageDatastore(bigimg,3,'BlockSize',[128 128],'IncompleteBlocks','pad')` creates a datastore that reads blocks of size 128-by-128 at resolution level 3 from big image `bigimg` and zero-pads partial edge blocks.

Create Datastore that Reads Blocks at Specified Locations

`bigds = bigimageDatastore(images,'BlockLocationSet',blockLocationSet)` creates a datastore that reads blocks from `bigimage` objects, Images, using the resolution level, block size, and block positions specified by BlockLocationSet.

`bigds = bigimageDatastore(images,'BlockLocationSet',blockLocationSet, Name,Value)` also uses name-value pairs to set one or more of the BorderSize, IncompleteBlocks, PadMethod, and ReadSize properties. You can specify multiple name-value pairs. Enclose each property name in quotes.

Example: `bigimageDatastore(bigimg,'BlockLocationSet',bls,'ReadSize',4)` creates a datastore that reads four blocks at a time from big images `bigimg` according to the position, block size, and resolution level specified by `bls`.

Properties

`BlockLocationSet` — Block locations

`blockLocationSet` object

Block locations, specified as a `blockLocationSet` object.

`BlockOffsets` — Block offsets

1-by-2 vector of positive integers

Block offsets, specified as 1-by-2 vector of positive integers of the form `[numrows numcols]`.

The default value is equal to BlockSize. To overlap blocks during calls to `read`, specify a smaller value. To add a gap between blocks, specify a larger value.

`BlockSize` — Block size

1-by-2 vector of positive integers

Block size of read data, specified as a 1-by-2 vector of positive integers of the form `[numrows numcols]`. The default value is equal to the BlockSize property of the first big image in Images at the first resolution level in Levels.

`BorderSize` — Border size

`[0 0]` (default) | 1-by-2 vector of nonnegative integers

Border size, specified as a 1-by-2 vector of nonnegative integers of the form `[m n]`. The function adds `m` rows above and below each block and `n` columns to the left and right of each block with data from the neighboring blocks. For blocks that lie on the edge of an image, data is padded according to IncompleteBlocks. By default, the datastore does not add a border to blocks.

`Images` — Big images

b-element vector of `bigimage` objects

Big images that supply blocks for the `bigimageDatastore`, specified as a *b*-element vector of `bigimage` objects. To read different resolution levels from the same big image, specify the same image multiple times in the vector.

`IncompleteBlocks` — Method to handle edge blocks

`"same"` (default) | `"exclude"` | `"pad"`

Method to handle edge blocks that are smaller than BlockSize, specified as one of these values.

Value	Meaning
`"same"`	Return data of the same size as the edge block.

Value	Meaning
`"exclude"`	Do not include edge blocks in calls to `read`.
`"pad"`	Pad incomplete blocks to the same size as `BlockSize` using the pad method specified by PadMethod.

`Levels` — Resolution level

positive integer | *b*-element vector of positive integers

Resolution level of blocks from each big image in Images, specified as a positive integer scalar or a *b*-element vector of positive integers. If you specify a scalar value, then all big images supply blocks to the datastore at the same resolution level.

Data Types: `double`

`PadMethod` — Pad method

numeric scalar | string scalar | `"replicate"` | `"symmetric"`

Pad method of incomplete edge blocks, specified as one of these values. By default, the datastore pads numeric blocks with `0` and categorical blocks with `missing`.

Value	Meaning
numeric scalar	Pad numeric array with elements of constant value.
string scalar	Pad categorical array with the specified class in the Classes property of the underlying `bigimage`.
`"replicate"`	Pad by repeating border elements of array.
`"symmetric"`	Pad array with mirror reflections of itself.

`ReadSize` — Number of blocks to return

1 (default) | positive integer

Number of blocks to return in each call to `read`, specified as a positive integer.

Object Functions

combine	Combine data from multiple datastores
countEachLabel	(Not recommended) Count number of pixel labels for each class of bigimageDatastore object
hasdata	Determine if data is available to read
numpartitions	Number of datastore partitions
partition	(Not recommended) Partition bigimageDatastore
preview	Preview subset of data in datastore
read	(Not recommended) Read data from bigimageDatastore
readRelative	(Not recommended) Read neighboring block from bigimageDatastore using relative position
reset	Reset datastore to initial state
shuffle	Shuffle data in datastore
transform	Transform datastore
isPartitionable	Determine whether datastore is partitionable
isShuffleable	Determine whether datastore is shuffleable

Examples

Create Big Image Datastore and Specify Block Size

Create a `bigimage` using a modified version of image "tumor_091.tif" from the CAMELYON16 data set. The original image is a training image of a lymph node containing tumor tissue. The original image has eight resolution levels, and the finest level has resolution 53760-by-61440. The modified image has only three coarse resolution levels. The spatial referencing of the modified image has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = bigimage('tumor_091R.tif');
```

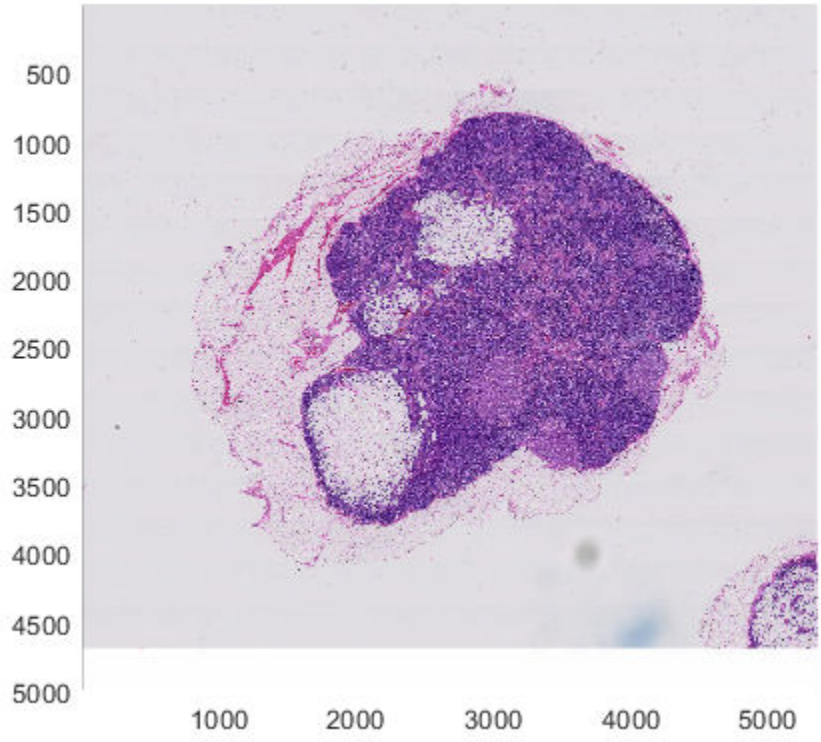
Display the default block size of the `bigimage` at each resolution level. The block size is a 2-element vector of the form [*numrows, numcols*].

```
t = table((1:3)',bim.BlockSize,'VariableNames',["Level" "Block Size"]);
disp(t)
```

```
    Level     Block Size
    _____    ____________

      1      1024    1024
      2      1024    1024
      3      1024    1024
```

Display the `bigimage` by using the `bigimageshow` function.

```
bigimageshow(bim);
```



Create a `bigimageDatastore` at resolution level 1. Specify a nondefault block size. Set the datastore to read four blocks at a time.

```
bimds = bigimageDatastore(bim,2,'BlockSize',[512 512],'ReadSize',4)
```

```
bimds =
  bigimageDatastore with properties:

           ReadSize: 4
         BorderSize: [0 0]
          PadMethod: 0
             Images: [1x1 bigimage]
             Levels: 2
          BlockSize: [512 512]
       BlockOffsets: [512 512]
   IncompleteBlocks: 'same'
   BlockLocationSet: [1x1 blockLocationSet]
```

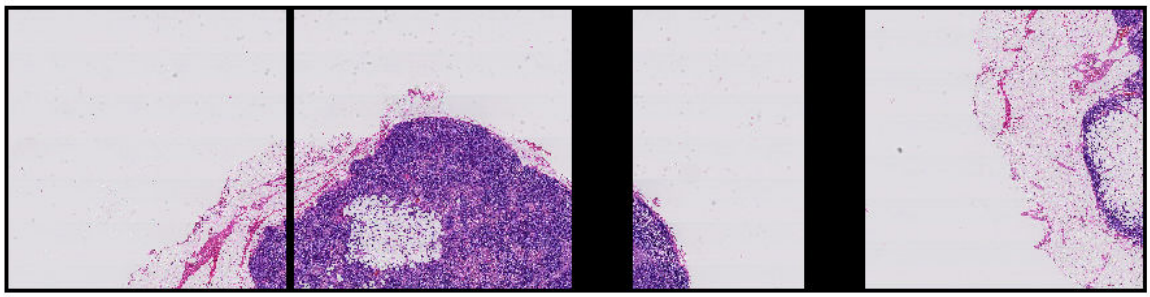
Read one batch of data from the datastore. Notice that the third block is a partial edge block and has a smaller size than interior blocks. Display the returned image patches as a montage. The montage displays the third block with a thicker border because the width of the block is smaller than the width of the complete blocks.

```
blocks = read(bimds)
```

```
blocks=4×1 cell array
    {512x512x3 uint8}
```

```
    {512x512x3 uint8}
    {512x316x3 uint8}
    {512x512x3 uint8}
```

```
montage(blocks,'Size',[1 bimds.ReadSize],'BorderSize',5,'BackgroundColor','k');
```



Read the next batch of data from the datastore and display the returned image patches as a montage. The montage displays partial blocks with a thicker border because the dimensions of the blocks are smaller than the dimensions of the full block.

```
blocks = read(bimds)
```

```
blocks=4×1 cell array
    {512x512x3 uint8}
    {512x316x3 uint8}
    {226x512x3 uint8}
    {226x512x3 uint8}
```

```
montage(blocks,'Size',[1 bimds.ReadSize],'BorderSize',5,'BackgroundColor','k');
```

Read the last batch of data from the datastore. The read operation returns a partial batch that contains the only remaining patch. Display the patch.

```
blocks = read(bimds)
```

```
blocks = 1x1 cell array
    {226x316x3 uint8}
```

```
montage(blocks,'Size',[1 bimds.ReadSize],'BorderSize',5,'BackgroundColor','k');
```

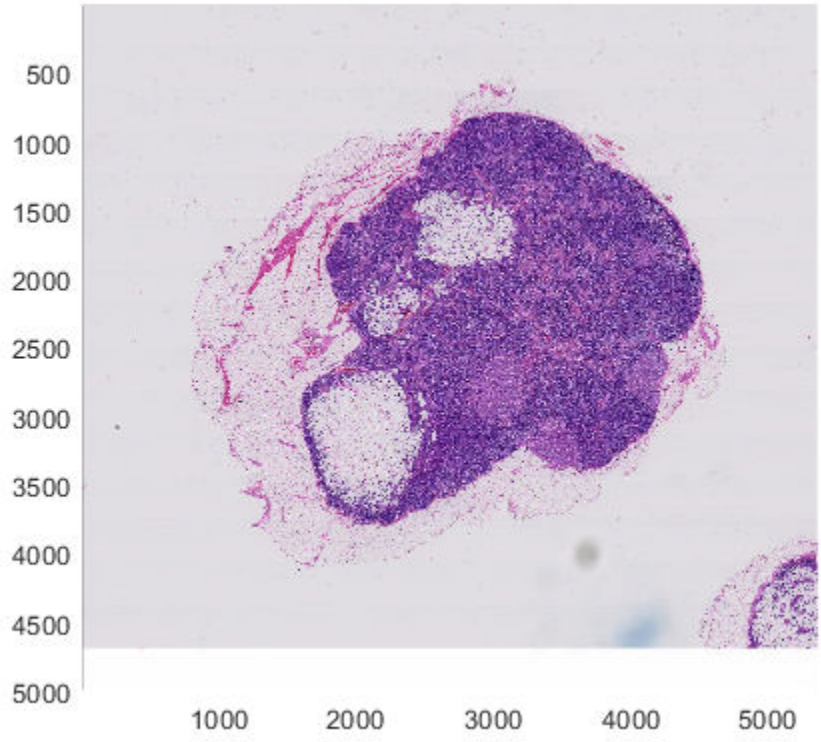
Create Big Image Datastore and Specify Mask

Create a `bigimage` using a modified version of image "tumor_091.tif" from the CAMELYON16 data set. The original image is a training image of a lymph node containing tumor tissue. The original image has eight resolution levels, and the finest level has resolution 53760-by-61440. The modified image has only three coarse resolution levels. The spatial referencing of the modified image has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = bigimage('tumor_091R.tif');
```

Display the `bigimage` by using the `bigimageshow` function.

```
h = bigimageshow(bim);
```



Create a mask at the coarsest resolution level, retaining the original spatial referencing information.

```
clevel = bim.CoarsestResolutionLevel;
imcoarse = getFullLevel(bim,clevel);
stainMask = ~imbinarize(rgb2gray(imcoarse));
bmask = bigimage(stainMask,'SpatialReferencing',bim.SpatialReferencing(clevel));
```

Specify the location of blocks to read from the `bigimage` by using the `selectBlockLocations` function. Set the block size as 256-by-256 pixels. Select blocks that are at least 75% within the ROI defined by the mask by specifying the `'InclusionThreshold'` name-value pair argument. By default, `selectBlockLocations` selects blocks from the finest resolution level of the big image.

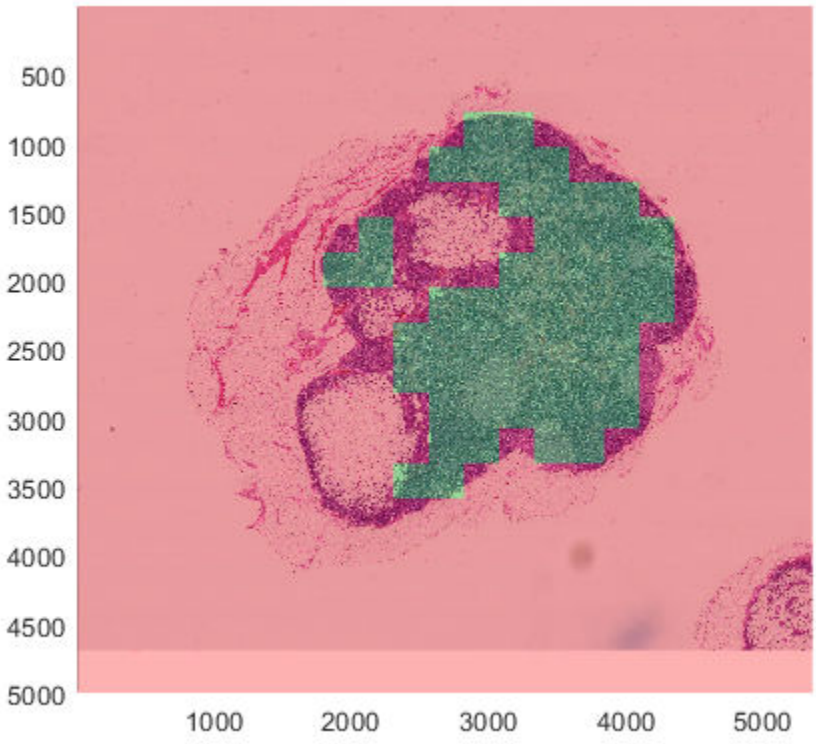
```
t = 0.75;
blockSize = [256 256];
blockSet = selectBlockLocations(bim,"BlockSize",blockSize, ...
    "Masks",bmask,"InclusionThreshold",t);
```

Create a `bigimageDatastore` that reads four blocks at a time from the locations specified by the `blockLocationSet`.

```
bimds = bigimageDatastore(bim,"BlockLocationSet",blockSet,"ReadSize",4);
```

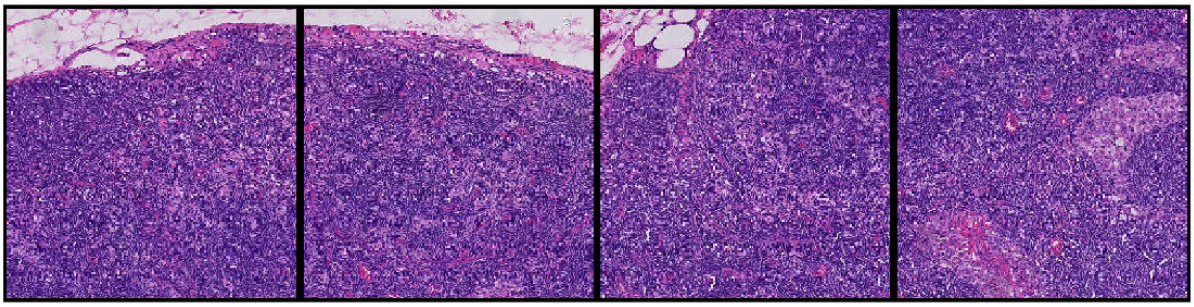
To preview which patches are read by the datastore, display the mask over the original `bigimage` using the same block size and inclusion threshold. The overlay highlights in green the patches that are at least 75% within the ROI defined by the mask.

```
showmask(h,bmask,'BlockSize',blockSize,'InclusionThreshold',t)
```



Read the first batch of data from the datastore and display the returned image patches as a montage. The content of these patches matches the green blocks of the overlay.

```
blocks = read(bimds);
montage(blocks,'Size',[1 bimds.ReadSize],'BorderSize',5,'BackgroundColor','k');
```



Read Big Image Blocks From Specified Block Locations

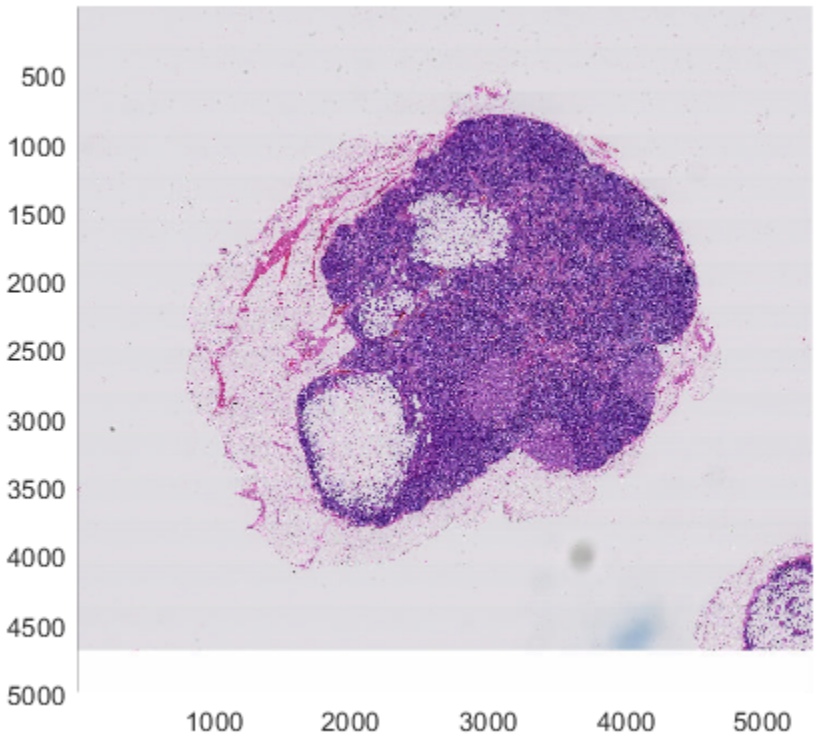
Create a `bigimage` using a modified version of image "tumor_091.tif" from the CAMELYON16 data set. The original image is a training image of a lymph node containing tumor tissue. The original

image has eight resolution levels, and the finest level has resolution 53760-by-61440. The modified image has only three coarse resolution levels. The spatial referencing of the modified image has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = bigimage('tumor_091R.tif');
```

Display the entire `bigimage` at the finest resolution level.

```
bshow = bigimageshow(bim);
```



Specify four [x y] block locations from the finest level. The first two blocks overlap in the vertical direction. The second two blocks are adjacent horizontally.

```
xyLocations = [ ...
    2800 1300; ...
    2800 1400; ...
    1500 2400; ...
    1800 2400];
blockSize = [300,300];
```

All blocks are from the same image. Specify the image number as `1` for all blocks.

```
imageNumber = [1 1 1 1]';
```

Create a `blockLocationSet` object that stores block locations.

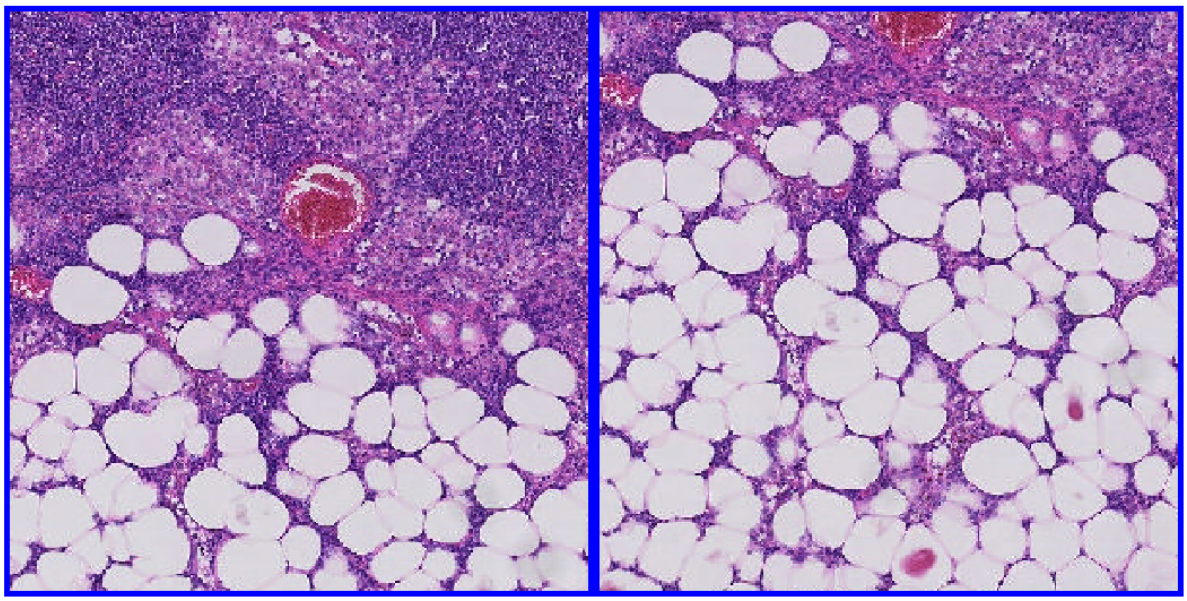
```
locationSet = blockLocationSet(imageNumber,xyLocations,blockSize);
```

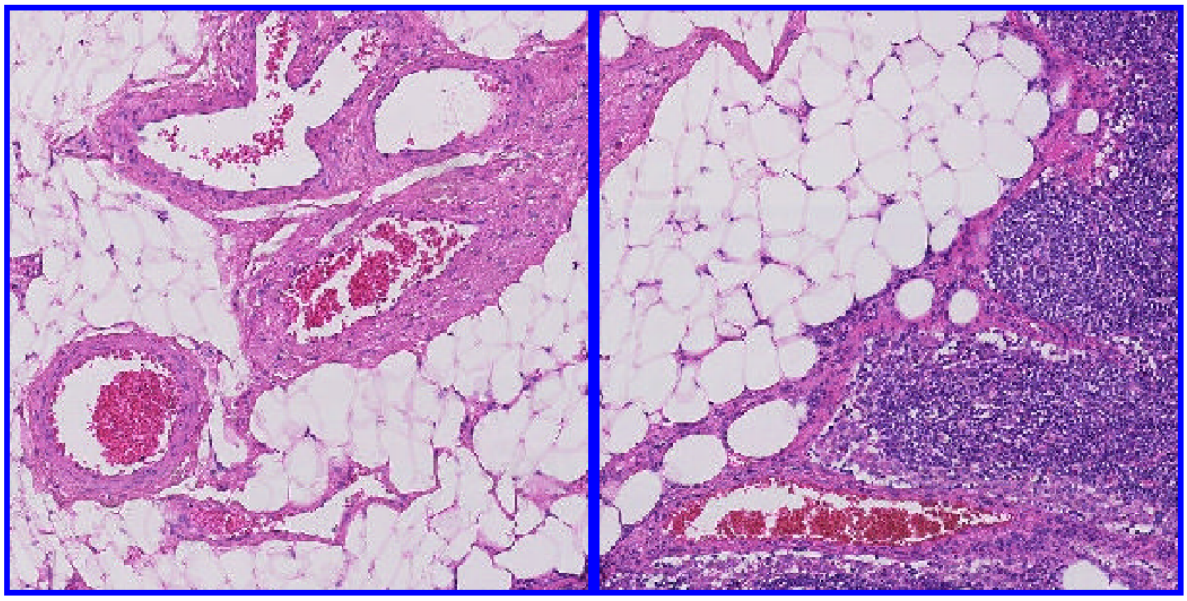
Create a `bigimageDatastore` object that reads blocks from big image `bim` at locations specified by the `blockLocationSet` object.

```
bimds = bigimageDatastore(bim,'BlockLocationSet',locationSet);
```

Read two blocks at a time from the datastore and display them in a montage.

```
bimds.ReadSize = 2;
while hasdata(bimds)
    figure
    blocks = read(bimds);
    montage(blocks,'BorderSize',5,'BackgroundColor','b');
end
```





Compatibility Considerations

`bigimageDatastore` is not recommended

Not recommended starting in R2021a

Starting in R2021a, the `bigimageDatastore` object, which reads data from `bigimage` objects, is no longer recommended. Instead, use the `blockedImageDatastore` object, which reads data from `blockedImage` objects. The `blockedImage` object offers several advantages including extension to N-D processing, a simpler interface, and custom support for reading and writing nonstandard image formats.

Code Updates

Update all instances of the `bigimageDatastore` object.

Discouraged Usage	Recommended Replacement

This example creates a `bigimageDatastore` object.
 	
`bim = bigimage('tumor_091R.tif');`
`bimds = bigimageDatastore(bim);`
	Here is equivalent code, replacing the `bigimageDatastore` object with the new `blockedImageDatastore` object.
	
	`bim = blockedImage('tumor_091R.tif');`
	`bimds = blockedImageDatastore(bim);`

The `blockedImageDatastore` object supports some different properties and functions than the `bigimageDatastore` object. For example, the `blockedImageDatastore` object does not support the `BlockOffset` and `IncompleteBlocks` properties, and it adds support for the `TotalNumBlocks` property. The `blockedImageDatastore` object does not support the `readRelative` function.

The `Mask` and `InclusionThreshold` properties of `bigimageDatastore` are not recommended

Starting in R2020b, the `Mask` and `InclusionThreshold` properties of `bigimageDatastore` are no longer recommended. Instead, first specify a mask and inclusion threshold as input arguments to the `selectBlockLocations` function. Then, specify the resulting `blockLocationSet` object as an input to `bigimageDatastore`.

The table shows some typical usages of `bigimageDatastore` and how to update your code to use `selectBlockLocations`.

Not Recommended	Recommended
`b = bigimageDatastore(bigimg,1,'Masks',m)`	`bls = selectBlockLocations(bigimg,'Masks',m);` `b = bigimageDatastore(bigimg,'BlockLocationSet',bls)`

`b = bigimageDatastore(bigimg,3,'Masks',m,'InclusionThreshold',t)`	`bls = selectBlockLocations(bigimg,'Levels',3, ...`
	`    'Masks',m,'InclusionThreshold',t);`
	`b = bigimageDatastore(bigimg,'BlockLocationSet',bls)`

References

[1] Bejnordi, Babak Ehteshami, Mitko Veta, Paul Johannes van Diest, Bram van Ginneken, Nico Karssemeijer, Geert Litjens, Jeroen A. W. M. van der Laak, et al. “Diagnostic Assessment of Deep Learning Algorithms for Detection of Lymph Node Metastases in Women With Breast Cancer.” JAMA 318, no. 22 (December 12, 2017): 2199–2210. https://doi.org/10.1001/jama.2017.14585.

See Also

`blockedImage` | `selectBlockLocations` | `blockLocationSet`

Topics

“Create Labeled Blocked Image from ROIs and Masks”
“Preprocess Multiresolution Images for Training Classification Network”
“Datastores for Deep Learning” (Deep Learning Toolbox)

External Websites

https://camelyon17.grand-challenge.org/Data/

Introduced in R2019b

countEachLabel

(Not recommended) Count number of pixel labels for each class of `bigimageDatastore` object

Note The `countEachLabel` function of the `bigimageDatastore` object is not recommended. Use the `countEachLabel` function associated with the `blockedImageDatastore` object instead. For more information, see “Compatibility Considerations”.

Syntax

```
counts = countEachLabel(bigds)
```

Description

`counts = countEachLabel(bigds)` returns the number of each pixel label for all big images in big image datastore `bigds`.

Examples

Calculate Class Weights of Labeled Big Images

Load pixel label data.

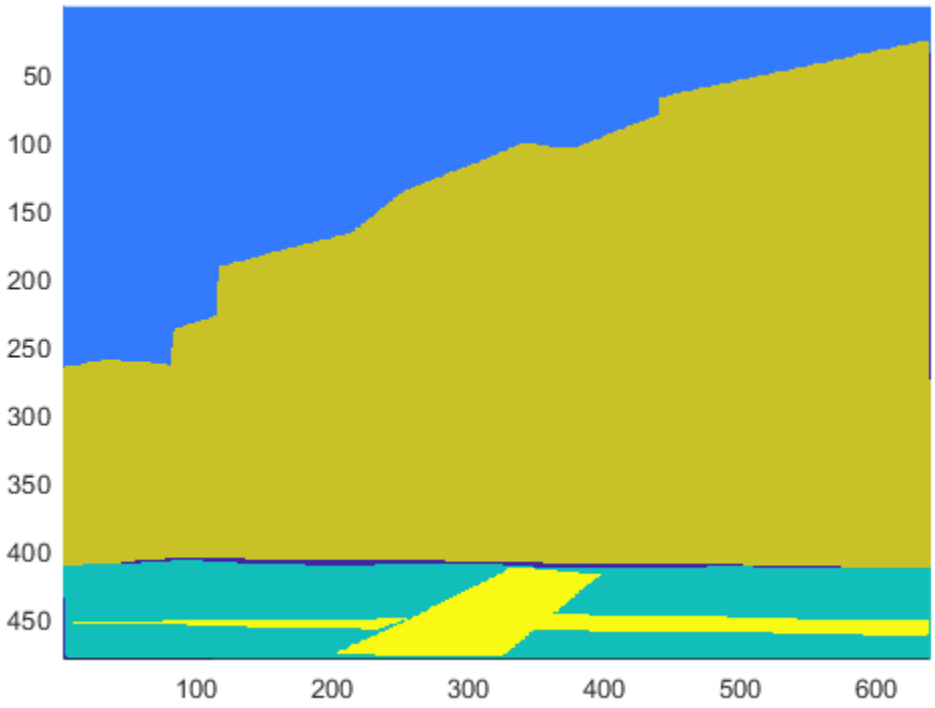
```
load('buildingPixelLabeled.mat');
```

Specify the classes and pixel label IDs of the pixel label data.

```
pixelLabelID = [1 2 3 4];
classNames = ["sky" "grass" "building" "sidewalk"];
```

Create a `bigimage` to manage the pixel label data.

```
bigLabeledImage = bigimage(uint8(label),'Classes',classNames,'PixelLabelIDs',pixelLabelID);
bigimageshow(bigLabeledImage)
```



Create a `bigimageDatastore` that reads blocks of size 200-by-150 pixels at the finest resolution level from `bigLabeledImage`.

```
level = 1;
blockSize = [200 150];
biglabelds = bigimageDatastore(bigLabeledImage,level,'BlockSize',blockSize);
```

Count the number of pixel labels for each class.

```
tbl = countEachLabel(biglabelds)
```

```
tbl=4×3 table
       Name       PixelCount    BlockPixelCount
    __________    __________    _______________

    "sky"              81525        1.58e+05
    "grass"            32983           51200
    "building"     1.8036e+05       3.072e+05
    "sidewalk"         10491           51200
```

Balance the classes by using uniform prior weighting.

```
prior = 1/numel(classNames);
uniformClassWeights = prior ./ tbl.PixelCount
```

```
uniformClassWeights = 4×1
10^-4 ×
```

```
    0.0307
    0.0758
    0.0139
    0.2383
```

Balance the classes by using inverse frequency weighting.

```
totalNumberOfPixels = sum(tbl.PixelCount);
freq = tbl.PixelCount / totalNumberOfPixels;
invFreqClassWeights = 1./freq
```

```
invFreqClassWeights = 4×1

    3.7456
    9.2580
    1.6931
   29.1067
```

Balance the classes by using median frequency weighting.

```
freq = tbl.PixelCount ./ tbl.BlockPixelCount;
medFreqClassWeights = median(freq) ./ freq
```

```
medFreqClassWeights = 4×1

    1.0689
    0.8562
    0.9394
    2.6917
```

Input Arguments

`bigds` — Big image datastore

`bigimageDatastore` object

Big image datastore, specified as a `bigimageDatastore` object.

Output Arguments

`counts` — Label information

table

Label information, returned as a table that contains three variables.

Pixel Count Variables	Description
`Name`	Pixel label class name
`PixelCount`	Number of pixels in class
`ImagePixelCount`	Total number of pixels in images that have an instance of the class

Tips

You can use the label information returned by `countEachLabel` to calculate class weights for class balancing. For example, for labeled pixel data information in `tbl`:

- Uniform class balancing weights each class such that each contains a uniform prior probability:

  ```
  numClasses = height(tbl)
  prior = 1/numClasses;
  classWeights = prior./tbl.PixelCount
  ```

- Inverse frequency balancing weights each class such that underrepresented classes are given higher weight:

  ```
  totalNumberOfPixels = sum(tbl.PixelCount)
  frequency = tbl.PixelCount / totalNumberOfPixels;
  classWeights = 1./frequency
  ```

- Median frequency balancing weights each class using the median frequency:

  ```
  imageFreq = tbl.PixelCount ./ tbl.ImagePixelCount
  classWeights = median(imageFreq) ./ imageFreq
  ```

You can pass the calculated class weights to a `pixelClassificationLayer`.

Compatibility Considerations

`countEachLabel` function is not recommended
Not recommended starting in R2021a

Starting in R2021a, the `bigimageDatastore` object and its object functions, which operate on data from `bigimage` objects, are no longer recommended. Instead, use the `blockedImageDatastore` object and its object functions, which operate on data from `blockedImage` objects. The `blockedImage` object offers several advantages including extension to N-D processing, a simpler interface, and custom support for reading and writing nonstandard image formats.

Although there are no plans to remove the `bigimageDatastore` object and its `countEachLabel` function at this time, switch to the `blockedImageDatastore` object and its `countEachLabel` function to take advantage of the additional capabilities and flexibility.

See Also

`pixelClassificationLayer` | `trainNetwork` | `blockedImageDatastore` | `countEachLabel`

Introduced in R2020a

partition

(Not recommended) Partition `bigimageDatastore`

Note The `partition` function of the `bigimageDatastore` object is not recommended. Use the `partition` function associated with the `blockedImageDatastore` object instead. For more information, see “Compatibility Considerations”.

Syntax

```
outds = partition(bigds,n,index)
```

Description

`outds = partition(bigds,n,index)` partitions big image datastore `bigds` into the number of parts specified by `n` and returns the partition corresponding to the index `index`.

Input Arguments

`bigds` — Big image datastore

`bigimageDatastore` object

Big image datastore, specified as a `bigimageDatastore` object.

`n` — Number of partitions

positive integer

Number of partitions, specified as a positive integer.

Example: `3`

Data Types: `double`

`index` — Index

positive integer

Index, specified as a positive integer.

Example: `1`

Data Types: `double`

Output Arguments

`outds` — Output datastore

`bigimageDatastore` object

Output datastore, returned as a `bigimageDatastore` object.

Compatibility Considerations

`partition` function is not recommended

Not recommended starting in R2021a

Starting in R2021a, the `bigimageDatastore` object and its object functions, which operate on data from `bigimage` objects, are no longer recommended. Instead, use the `blockedImageDatastore` object and its object functions, which operate on data from `blockedImage` objects. The `blockedImage` object offers several advantages including extension to N-D processing, a simpler interface, and custom support for reading and writing nonstandard image formats.

Although there are no plans to remove the `bigimageDatastore` object and its `partition` function at this time, switch to the `blockedImageDatastore` object and its `partition` function to take advantage of the additional capabilities and flexibility.

See Also

`blockedImageDatastore` | `partition` | `numpartitions`

Introduced in R2019b

read

(Not recommended) Read data from `bigimageDatastore`

Note The `read` function of the `bigimageDatastore` object is not recommended. Use the `read` function associated with the `blockedImageDatastore` object instead. For more information, see "Compatibility Considerations".

Syntax

```
data = read(bigds)
[data,info] = read(bigds)
```

Description

`data = read(bigds)` returns a batch of data from a big image datastore, `bigds`. Subsequent calls to the `read` function continue reading from the endpoint of the previous call.

`[data,info] = read(bigds)` also returns information about the extracted data, including metadata, in `info`.

Examples

Read Data from Big Image Datastore

Create a `bigimage` using a modified version of image "tumor_091.tif" from the CAMELYON16 data set. The original image is a training image of a lymph node containing tumor tissue. The original image has eight resolution levels, and the finest level has resolution 53760-by-61440. The modified image has only three coarse resolution levels. The spatial referencing of the modified image has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = bigimage('tumor_091R.tif');
```

Create a `bigimageDatastore` that manages blocks of the big image at the finest resolution level.

```
bimds = bigimageDatastore(bim,1)
```

```
bimds =
  bigimageDatastore with properties:

            ReadSize: 1
          BorderSize: [0 0]
           PadMethod: 0
              Images: [1x1 bigimage]
              Levels: 1
           BlockSize: [1024 1024]
        BlockOffsets: [1024 1024]
    IncompleteBlocks: 'same'
    BlockLocationSet: [1x1 blockLocationSet]
```

Change the `'ReadSize'` property of the datastore to `3`.

```
bimds.ReadSize = 3;
```

Read one batch of image data from the datastore.

```
[data,info] = read(bimds);
```

Display the returned image data in a montage with a black border around each image. The montage shows that the datastore reads blocks of the big image in row-major order.

```
montage(data,'Size',[1 3],"BorderSize",10)
```

Display the information about the returned data.

```
info
```

```
info = struct with fields:
              Level: [1 1 1]
        ImageNumber: [1 1 1]
    BlockStartWorld: [3x2 double]
      BlockEndWorld: [3x2 double]
     DataStartWorld: [3x2 double]
       DataEndWorld: [3x2 double]
```

Inspect the (x,y) coordinates of the center of the top-left pixel of each returned block of data.

```
info.BlockStartWorld
```

```
ans = 3×2

           1           1
        1025           1
        2049           1
```

Input Arguments

`bigds` — Big image datastore

`bigimageDatastore`

Big image datastore, specified as a `bigimageDatastore` object.

- The datastore contains one or more big images, `Images`, each with `Channels` number of channels.
- The datastore reads blocks from each big image at specified resolution levels, `Levels`.
- The datastore specifies the number of blocks to read in each batch, `ReadSize`.
- The datastore specifies the *m*-by-*n* pixel size of blocks to read, `BlockSize`.

Output Arguments

`data` — Output data

cell array

Output data, returned as a cell array with `ReadSize` elements. Each cell contains an *m*-by-*n*-by-`Channels` numeric array.

`info` — Information about output data

struct

Information about output data, returned as a struct containing these fields.

Field Name	Description
`Level`	Resolution level of the data, specified as a 1-by-`ReadSize` vector of positive integers.
`ImageNumber`	Index of the big image providing the data, specified as a 1-by-`ReadSize` vector of positive integers.
`BlockStartWorld`	(*x*,*y*) coordinates of the center of the top-left pixel of the data, excluding padding, specified as a `ReadSize`-by-2 numeric vector. Values are in world-coordinates.
`BlockEndWorld`	(*x*,*y*) coordinates of the center of the bottom-right pixel of the data, excluding padding, specified as a `ReadSize`-by-2 numeric vector. Values are in world-coordinates.
`DataStartWorld`	(*x*,*y*) coordinates of the center of the top-left pixel of the data, including padding, specified as a `ReadSize`-by-2 numeric vector. Values are in world-coordinates.
`DataEndWorld`	(*x*,*y*) coordinates of the center of the bottom-right pixel of the data, including padding, specified as a `ReadSize`-by-2 numeric vector. Values are in world-coordinates.

Compatibility Considerations

`read` function is not recommended

Not recommended starting in R2021a

Starting in R2021a, the `bigimageDatastore` object and its object functions, which operate on data from `bigimage` objects, are no longer recommended. Instead, use the `blockedImageDatastore` object and its object functions, which operate on data from `blockedImage` objects. The `blockedImage` object offers several advantages including extension to N-D processing, a simpler interface, and custom support for reading and writing nonstandard image formats.

Although there are no plans to remove the `bigimageDatastore` object and its `read` function at this time, switch to the `blockedImageDatastore` object and its `read` function to take advantage of the additional capabilities and flexibility.

See Also

`read` | `blockedImageDatastore`

Topics

“Create Labeled Blocked Image from ROIs and Masks”

Introduced in R2019b

readRelative

(Not recommended) Read neighboring block from `bigimageDatastore` using relative position

Note The `readRelative` function of the `bigimageDatastore` object is not recommended. Use the `blockedImageDatastore` object and its object functions instead. For more information, see "Compatibility Considerations".

Syntax

```
data = readRelative(bigds,sourceInfo,blockOffset)
[data,info] = readRelative(bigds,sourceInfo,blockOffset)
```

Description

`data = readRelative(bigds,sourceInfo,blockOffset)` returns the block from big image datastore `bigds` that neighbors the source block `sourceInfo` with offset `blockOffset`.

`[data,info] = readRelative(bigds,sourceInfo,blockOffset)` also returns information about the extracted data, including metadata, in `info`.

Examples

Read Neighboring Big Image Blocks

Create a `bigimage` using a modified version of image "tumor_091.tif" from the CAMELYON16 data set. The original image is a training image of a lymph node containing tumor tissue. The original image has eight resolution levels, and the finest level has resolution 53760-by-61440. The modified image has only three coarse resolution levels. The spatial referencing of the modified image has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = bigimage('tumor_091R.tif');
```

Create a `bigimageDatastore` that manages blocks of the big image at the finest resolution level.

```
bimds = bigimageDatastore(bim,1);
```

Read the first block from the datastore.

```
[b,binfo] = read(bimds);
b = b{1};
```

Read the neighboring blocks to the left and right of the block. The left neighboring block is empty because the block is out of the bounds of `bim`.

```
bLeft = readRelative(bimds,binfo,[0 -1]);
bRight = readRelative(bimds,binfo,[0 1]);
```

Display the blocks as a montage. The left neighboring block appears black because it is empty.

```
montage({bLeft,b,bRight},'Size',[1 3],'BorderSize',5,'BackgroundColor','b')
```

Input Arguments

`bigds` — Big image datastore

`bigimageDatastore` object

Big image datastore, specified as a `bigimageDatastore` object.

`sourceInfo` — Information about source block

struct

Information about source block, specified as a struct containing at least these fields. The value of `info` returned by `read` is a valid input for `sourceInfo`.

Field Name	Description
`Level`	Resolution level of the data, specified as a positive integers.
`ImageNumber`	Index of the big image providing the data, specified as a positive integer.
`BlockStartWorld`	(*x*,*y*) world coordinates of the top-left corner of the data, specified as a 1-by-2 numeric vector. The coordinates correspond to a position on the boundary of the block, not the center of the top-left pixel.

`blockOffset` — Block offset

1-by-2 vector of integers

Block offset, specified as a 1-by-2 vector of integers in units of blocks. The two elements specify the vertical and horizontal offset from the source block. respectively.

Output Arguments

`data` — Output data

numeric array

Output data, returned as a numeric array. If the requested block is outside the bounds of the source image, then `readRelative` returns an empty block, `[]`

`info` — Information about output data

struct

Information about output data, returned as a struct containing these fields.

Field Name	Description
`Level`	Resolution level of the data, specified as a 1-by-`ReadSize` vector of positive integers.
`ImageNumber`	Index of the big image providing the data, specified as a 1-by-`ReadSize` vector of positive integers.
`BlockStartWorld`	(*x*,*y*) coordinates of the center of the top-left pixel of the data, excluding padding, specified as a `ReadSize`-by-2 numeric vector. Values are in world-coordinates.
`BlockEndWorld`	(*x*,*y*) coordinates of the center of the bottom-right pixel of the data, excluding padding, specified as a `ReadSize`-by-2 numeric vector. Values are in world-coordinates.
`DataStartWorld`	(*x*,*y*) coordinates of the center of the top-left pixel of the data, including padding, specified as a `ReadSize`-by-2 numeric vector. Values are in world-coordinates.
`DataEndWorld`	(*x*,*y*) coordinates of the center of the bottom-right pixel of the data, including padding, specified as a `ReadSize`-by-2 numeric vector. Values are in world-coordinates.

Tips

- `readRelative` ignores masks.
- `readRelative` respects the `PadMethod` and `BorderSize` properties of the big image datastore.
- If the requested block is incomplete and `bigds.IncompleteBlocks` has a value of `'exclude'`, then `readRelative` returns an empty block

Compatibility Considerations

`readRelative` function is not recommended

Not recommended starting in R2021a

Starting in R2021a, the `bigimageDatastore` object and its object functions, which operate on data from `bigimage` objects, are no longer recommended. Instead, use the `blockedImageDatastore` object and its object functions, which operate on data from `blockedImage` objects. The

`blockedImage` object offers several advantages including extension to N-D processing, a simpler interface, and custom support for reading and writing nonstandard image formats.

Although there are no plans to remove the `bigimageDatastore` object and its `readRelative` function at this time, switch to the `blockedImageDatastore` object to take advantage of the additional capabilities and flexibility.

See Also

`blockedImageDatastore`

Introduced in R2019b

bigimageshow

Display 2-D `blockedImage` object

Description

A `bigimageshow` object displays data from a `blockedImage` object. The `bigimageshow` object progressively loads image data based on image extents and screen resolution.

Creation

Syntax

```
bigimageshow(bim)
bigimageshow(hax,bim)
b = bigimageshow( ___ )
b = bigimageshow( ___ ,Name,Value)
```

Description

`bigimageshow(bim)` displays the 2-D blocked image `bim`.

For categorical data, `bigimageshow` sets the axis colormap to `parula`. For numeric data, `gray` is the default colormap.

`bigimageshow(hax,bim)` displays the blocked image, `bim`, in the axes specified by `hax`.

`b = bigimageshow( ___ )` returns the `bigimageshow` object `b`. Use `b` to modify the display settings after you display the blocked image.

`b = bigimageshow( ___ ,Name,Value)` sets initial display properties on page 1-189 using name-value pairs. You can specify multiple name-value pairs. Enclose each argument or property name in quotes.

For example, `bigimageshow(bim,'GridVisible','on','GridLineStyle',':')` displays the blocked image, `bim`, and overlays dotted grid lines.

Input Arguments

`bim` — Blocked image
`blockedImage` object

Blocked image, specified as a `blockedImage` object.

`hax` — Parent axes
axes object

Parent axes of `bigimageshow` object, specified as an axes object.

Properties

`Parent` — Parent axes of `bigimageshow` object

`gca` (default) | `axes` object

Parent axes of the `bigimageshow` object, specified as an `axes` object. If you do not specify a parent, `bigimageshow` uses the handle to the current figure, `gca`. If a figure does not exist, `bigimageshow` creates a new figure.

`CData` — 2-D `blockedImage` object to display

`blockedImage` object

2-D `blockedImage` object to display, specified as a `blockedImage` object.

`CDataMapping` — Color data mapping method

`'direct'` (default) | `'scaled'`

Color data mapping method, specified as `'direct'` or `'scaled'`. Use this property to control the mapping of color data values in `CData` into the colormap. `CData` must be a vector or a matrix defining indexed colors. This property has no effect if `CData` is a 3-D array defining RGB colors.

The methods have these effects:

- `'direct'` — Interpret the values as indices into the current colormap. Values with a decimal portion are fixed to the nearest lower integer.
  - If the values are of type `double` or `single`, values of `1` or less map to the first color in the colormap. Values equal to or greater than the length of the colormap map to the last color in the colormap.
  - If the values are of type `uint8`, `uint16`, `uint32`, `uint64` , `int8`, `int16`, `int32`, or `int64`, values of `0` or less map to the first color in the colormap. Values equal to or greater than the length of the colormap map to the last color in the colormap (or up to the range limits of the type).
  - If the values are of type `logical`, values of `0` map to the first color in the colormap and values of `1` map to the second color in the colormap.
- `'scaled'` — Scale the values to range between the minimum and maximum color limits. The `CLim` property of the axes contains the color limits.

`AlphaData` — Transparency data

`1` (default) | numeric scalar | `blockedImage` object

Transparency data, specified in one of these forms:

- Numeric scalar — Use a consistent transparency across the entire image.
- 2-D `blockedImage` object — Transparency data must have the same rows and columns extent as the `CData` 2-D `blockedImage` object. The blocked image can have multiple resolution levels, in which case, `bigimageshow` selects the level closest to the current `ResolutionLevel` for display.

The `AlphaDataMapping` property controls how MATLAB interprets the alpha data transparency values.

Example: `0.5`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`AlphaDataMapping` — Interpretation of `AlphaData` values
`'none'` (default) | `'scaled'` | `'direct'`

Interpretation of `AlphaData` values, specified as one of these values:

- `'none'` — Interpret the values as transparency values. A value of 1 or greater is completely opaque, a value of 0 or less is completely transparent, and a value between 0 and 1 is semitransparent.
- `'scaled'` — Map the values into the figure's alphamap. The minimum and maximum alpha limits of the axes determine the alpha data values that map to the first and last elements in the alphamap, respectively. For example, if the alpha limits are `[3 5]`, alpha data values less than or equal to 3 map to the first element in the alphamap. Alpha data values greater than or equal to 5 map to the last element in the alphamap. The `ALim` property of the axes contains the alpha limits. The `Alphamap` property of the figure contains the alphamap.
- `'direct'` — Interpret the values as indices into the figure's alphamap. Values with a decimal portion are fixed to the nearest lower integer:
  - If the values are of type `double` or `single`, values of 1 or less map to the first element in the alphamap. Values equal to or greater than the length of the alphamap map to the last element in the alphamap.
  - If the values are of type integer, then values of 0 or less map to the first element in the alphamap. Values equal to or greater than the length of the alphamap map to the last element in the alphamap (or up to the range limits of the type). The integer types are `uint8`, `uint16`, `uint32`, `uint64`, `int8`, `int16`, `int32`, and `int64`.
  - If the values are of type `logical`, values of `0` map to the first element in the alphamap and values of `1` map to the second element in the alphamap.

`ResolutionLevel` — Resolution level
positive integer | `'fine'` | `'coarse'`

Resolution level of the 2-D `blockedImage` object to display, specified as a positive integer that identifies a resolution level of the 2-D `blockedImage` object in the CData property. Resolution level can also be specified as `'fine'` or `'coarse'` corresponding to these two limits. The default value is computed based on available screen space and resolution.

`ResolutionLevelMode` — Selection mode for resolution level
`'auto'` (default) | `'manual'`

Selection mode for resolution level, specified as one of these values:

- `'auto'` — Automatically select resolution level based on parent axes and available screen size.
- `'manual'` — Manually specify resolution level by setting the `ResolutionLevel` property.

`GridVisible` — Grid visibility
`'off'` (default) | `'on'`

Grid visibility, specified as `'off'` or `'on'`. `bigimageshow` spaces the grid in world units to include as many pixels as specified by `CData.BlockSize` at the current `GridResolutionLevel`.

`GridLevel` — Resolution level of blocked image at which to show grid

positive integer | `'fine'` | `'coarse'`

Resolution level of blocked image at which to show grid, specified as one of these values:

- positive integer — Display the grid specified as a numeric scalar that identifies a resolution level of the 2-D `blockedImage` object in CData property. Value is between 1 and the value of the `NumLevels` property of the blocked image in the `bigimageshow` `CData` property.
- `'fine'` — Display the grid at the finest resolution level.
- `'coarse'` — Display the grid at the coarsest resolution level.

By default, `GridLevel` has the same value as `ResolutionLevel` property.

`GridLevelMode` — Selection mode for grid level

`'auto'` (default) | `'manual'`

Selection mode for grid level, specified as one of these values:

- `'auto'` — Select the grid resolution level to match the image data resolution level `ResolutionLevel`.
- `'manual'` — Manually specify the grid resolution level by setting the `GridLevel` property.

`GridColor` — Grid line color

`'blue'` (default) | RGB triplet | hexadecimal color code | color name | short color name

Grid line color, specified as an RGB triplet, a hexadecimal color code, a color name, or a short color name. To display the grid lines, set the `GridVisible` property to `'on'`.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
'white'	'w'	[1 1 1]	'#FFFFFF'	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	'#0072BD'	
[0.8500 0.3250 0.0980]	'#D95319'	
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

Example: `b.GridColor = [1 0 0]`

Example: `b.GridColor = 'r'`

Example: `b.GridColor = 'red'`

Example: `b.GridColor = '#FF0000'`

`GridAlpha` — Grid line transparency
`0.8` (default) | value in the range [0,1]

Grid line transparency, specified as a value in the range [0, 1]. A value of `1` means completely opaque and a value of `0` means completely transparent. To display the grid lines, set the `GridVisible` property to `'on'`.

`GridLineWidth` — Grid line width
1 (default) | positive numeric value

Grid line width, specified as a positive numeric value, measured in points. To display the grid lines, set the `GridVisible` property to `'on'`.

`GridLineStyle` — Grid line style
`'-'` (default) | `'--'` | `':'` | `'-.'`

Grid line style, specified as one of the line styles in this table.

Line Style	Description	Resulting Line
'-'	Solid line	
'--'	Dashed line	
':'	Dotted line	
'-.'	Dash-dot line	

To display the grid lines, set the `GridVisible` property to `'on'`.

`Interpolation` — Interpolation method
`'linear'` (default) | `'nearest'`

Interpolation method used to resample pixels, specified as `'linear'` for bilinear interpolation, or `'nearest'` for nearest neighbor interpolation.

For categorical data, `bigimageshow` supports only nearest neighbor interpolation. For logical data, the default value is `'nearest'`.

On Windows systems with a software version of OpenGL, the only supported interpolation option is `'nearest'`.

`Visible` — Control image visibility
`'on'` (default) | `'off'`

Control image visibility, specified as one of these values:

- `'on'` — Display the `bigimageshow` object.
- `'off'` — Hide the object without deleting it. You still can access the properties of an invisible object.

Object Functions

showmask	Show mask overlay at specified inclusion threshold
hidemask	Hide mask overlay in bigimageshow object
showlabels	Display label overlay on bigimageshow object
hidelabels	Hide label overlay on bigimageshow object

Examples

Visualize 2-D Blocked Image at Different Resolution Levels

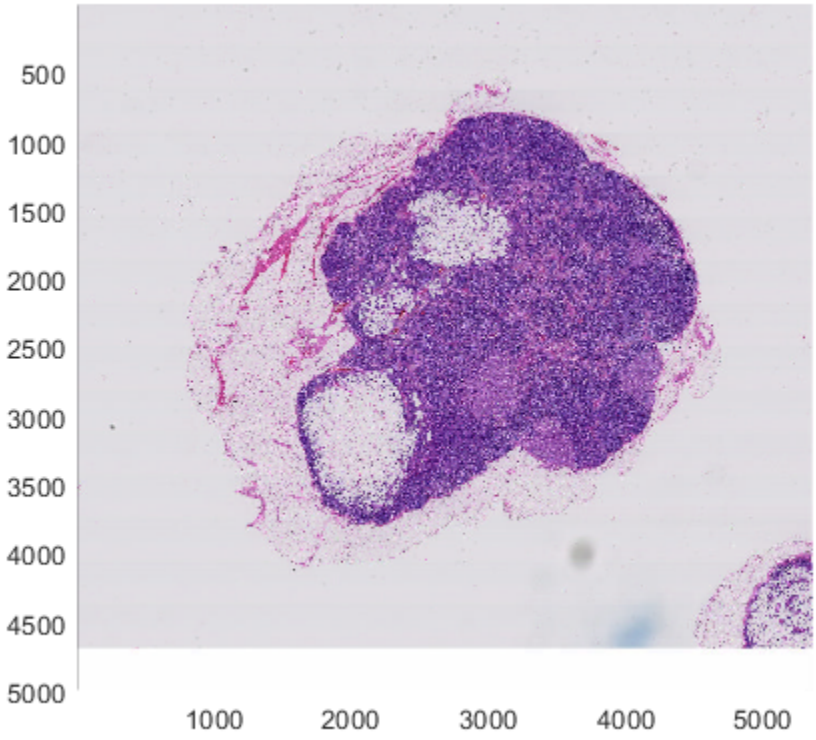
This example uses a modified version of a training image of a lymph node containing tumor tissue (tumor_091.tif) from the CAMELYON16 data set. The modified image has three coarse resolution levels and has been adjusted to enforce a consistent aspect ratio and to register features at each level.

Create a blocked image from the sample image.

```
bim = blockedImage('tumor_091R.tif');
```

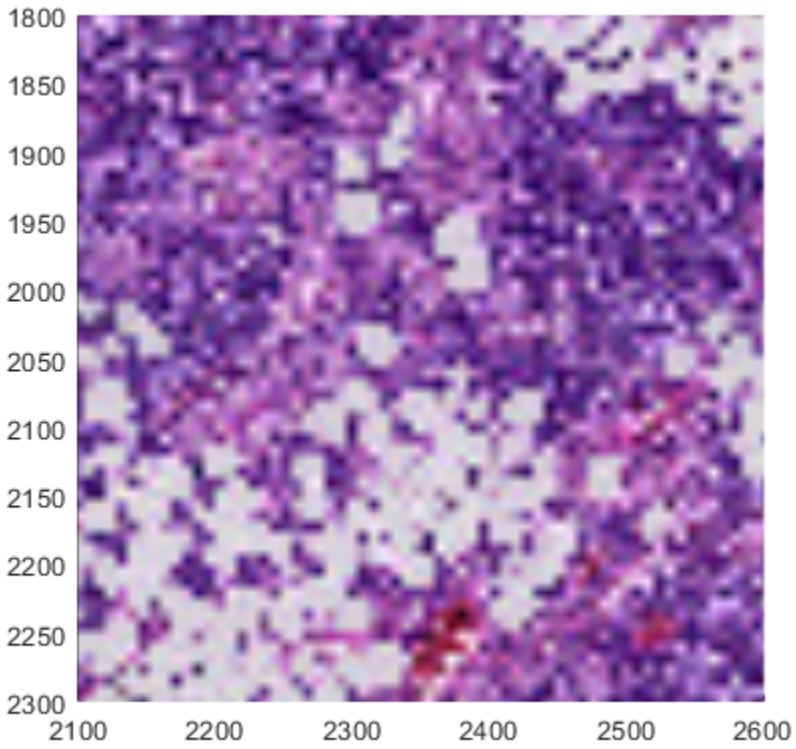
Display the blocked image.

```
h = bigimageshow(bim);
```



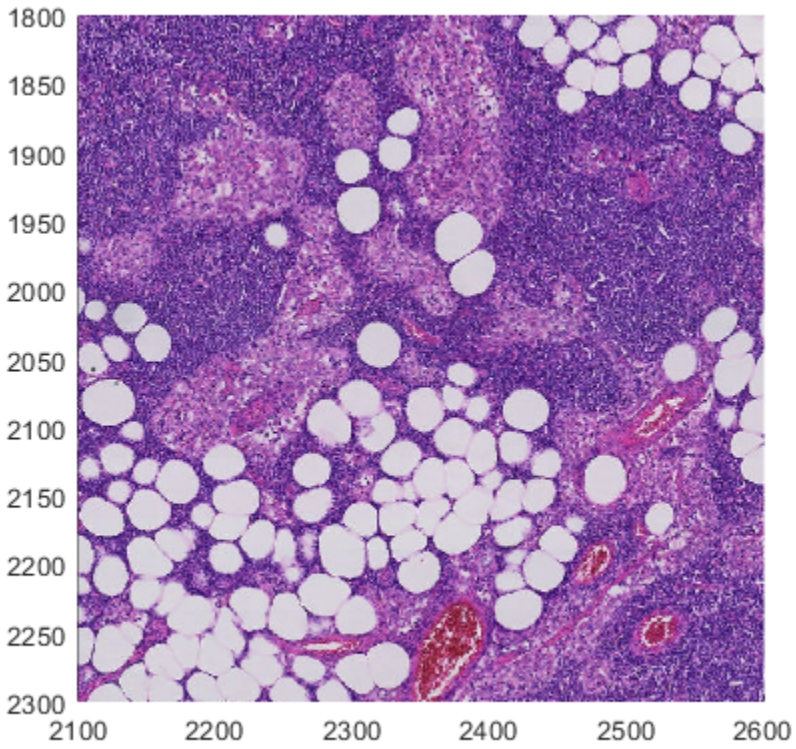
Zoom in on a region in the image.

```
xlim([2100, 2600])
ylim([1800 2300])
```



To view the image at the three resolution levels, specify a new value for the `ResolutionLevel` property. As you view each resolution level, note that the axis limits remain the same, but `bigimageshow` ensures that the images from other levels are correctly sized. When you set `ResolutionLevel` , the `ResolutionLevelMode` value changes to `'manual'` automatically.

```
h.ResolutionLevel = 3;
pause(1);
h.ResolutionLevel = 2;
pause(1);
h.ResolutionLevel = 1;
pause(1);
```



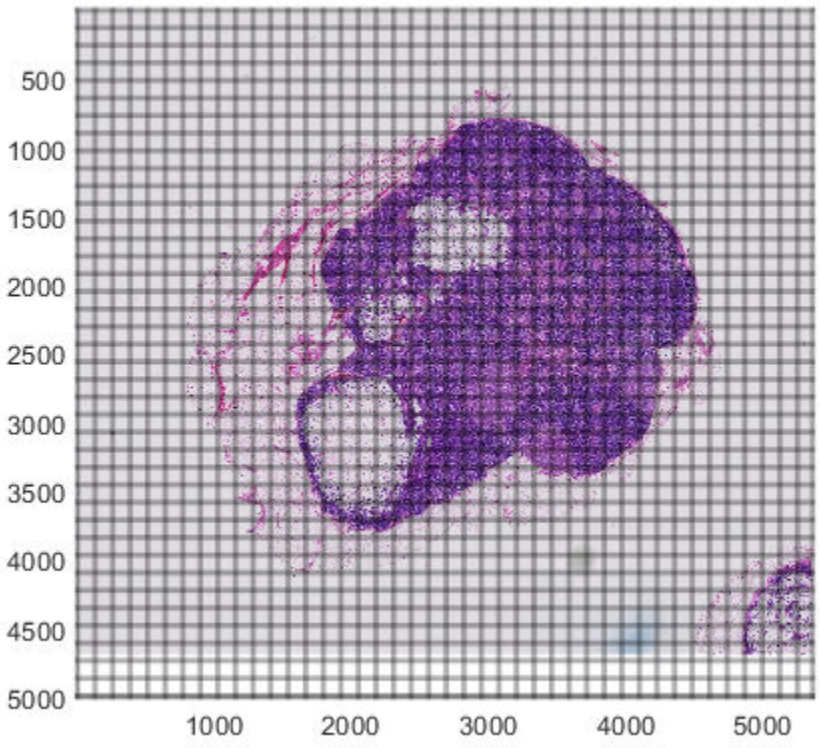
Visualize Blocked Image with Grid Lines Indicating Blocks

Create a blocked image from the sample image `tumor_091R.tif`. This sample image is a training image of a lymph node containing tumor tissue from the CAMELYON16 data set. The image has been modified to have three coarse resolution levels, and has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = blockedImage('tumor_091R.tif','BlockSize', [128 128]);
```

Display the blocked image with `bigimageshow`. Specify that you want the grid to be visible at the finest resolution level (level 1). Also specify the color, width, and transparency of the grid lines.

```
h = bigimageshow(bim,...
     'GridVisible','on','GridLevel',1,...
     'GridLineWidth', 2,'GridColor','k','GridAlpha',0.3);
```



Validate Mask using Alpha Layer

Create a blocked image from the sample image `tumor_091R.tif`. This sample image is a training image of a lymph node containing tumor tissue from the CAMELYON16 data set. The image has been modified to have three coarse resolution levels, and has been adjusted to enforce a consistent aspect ratio and to register features at each level.

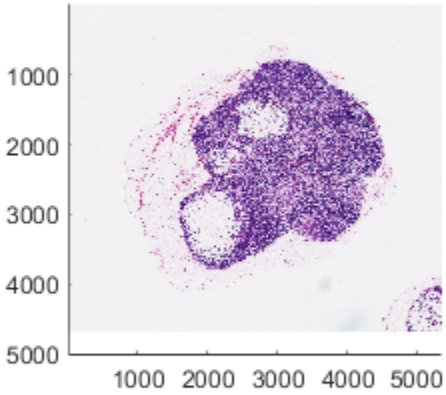
```
bim = blockedImage('tumor_091R.tif');
```

Create a coarse mask using the `blockedImage` `apply` object function.

```
bmask = apply(bim, @(bs)im2gray(bs.Data)<120, "Level", 3);
```

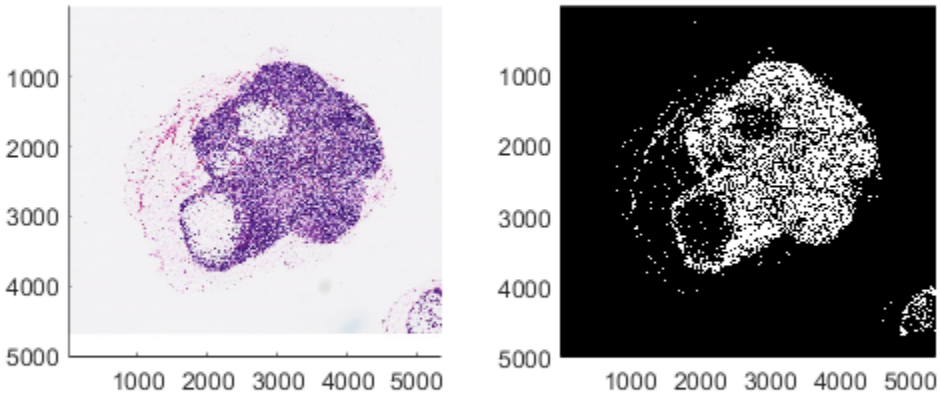
Overlay the mask as an alpha layer.

```
ha1 = subplot(1,2,1);
h = bigimageshow(bim);
h.AlphaData = bmask;
h.AlphaDataMapping = 'direct';
alphamap([0.4 1])
h.Parent.Color = 'r';
```



Independently visualize the mask.

```
ha2 = subplot(1,2,2);
bigimageshow(bmask);
linkaxes([ha1, ha2]);
%
```



Improve Mask Creation Using InclusionThreshold and BlockSize

Create a blocked image from the sample image `tumor_091R.tif`. This sample image is a training image of a lymph node containing tumor tissue from the CAMELYON16 data set. The image has been modified to have three coarse resolution levels, and has been adjusted to enforce a consistent aspect ratio and to register features at each level.

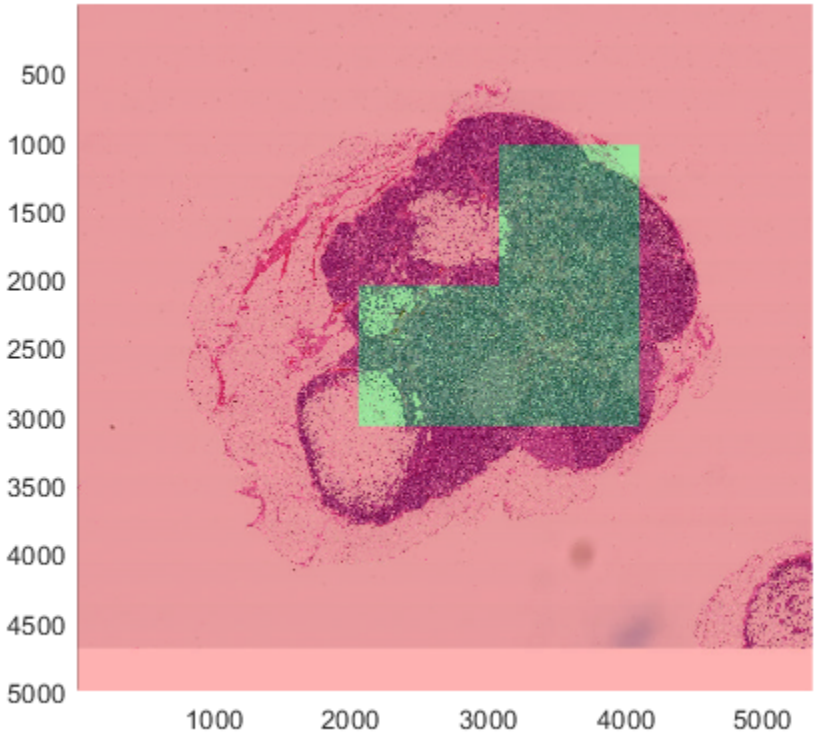
```
bim = blockedImage('tumor_091R.tif');
```

Create a mask using the coarsest resolution level of the blocked image.

```
bmask = apply(bim, @(im)im2gray(im.Data)<120, "Level", 3);
```

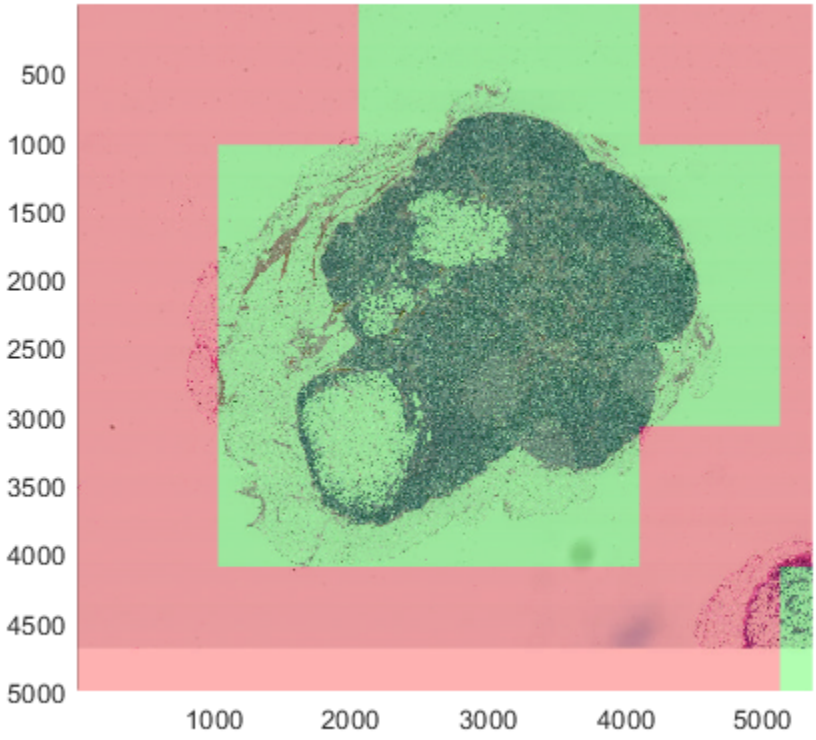
Display the blocked image with the mask.

```
h = bigimageshow(bim);
showmask(h, bmask);
```



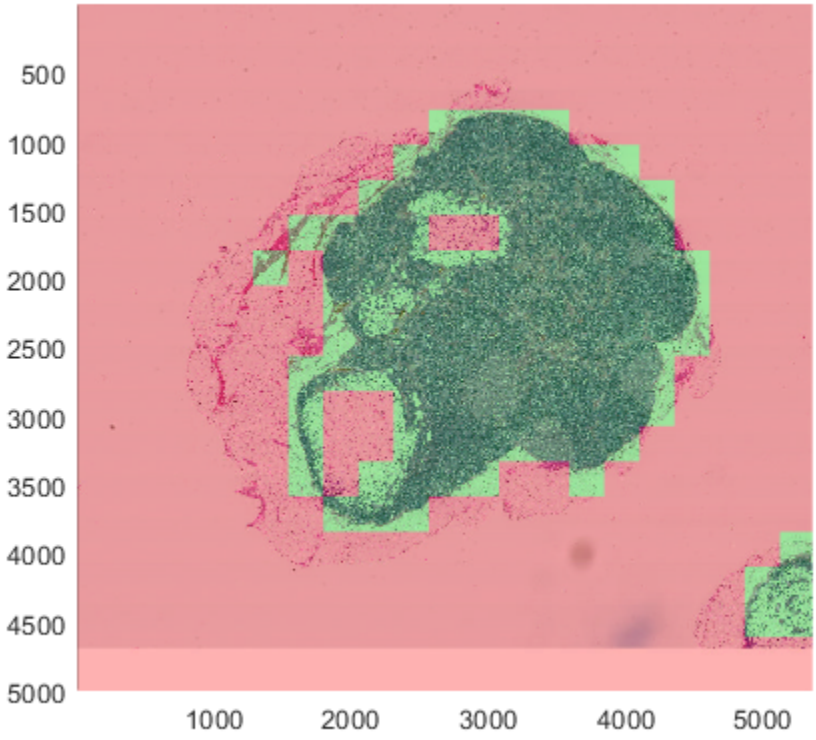
Experiment with different inclusion thresholds to get a better fit of the mask over the stained area. By default, the inclusion threshold is 0.5.

```
showmask(h, bmask, 'InclusionThreshold', 0.2);
showmask(h, bmask, 'InclusionThreshold', 0);
showmask(h, bmask, 'InclusionThreshold', 0.06);
```



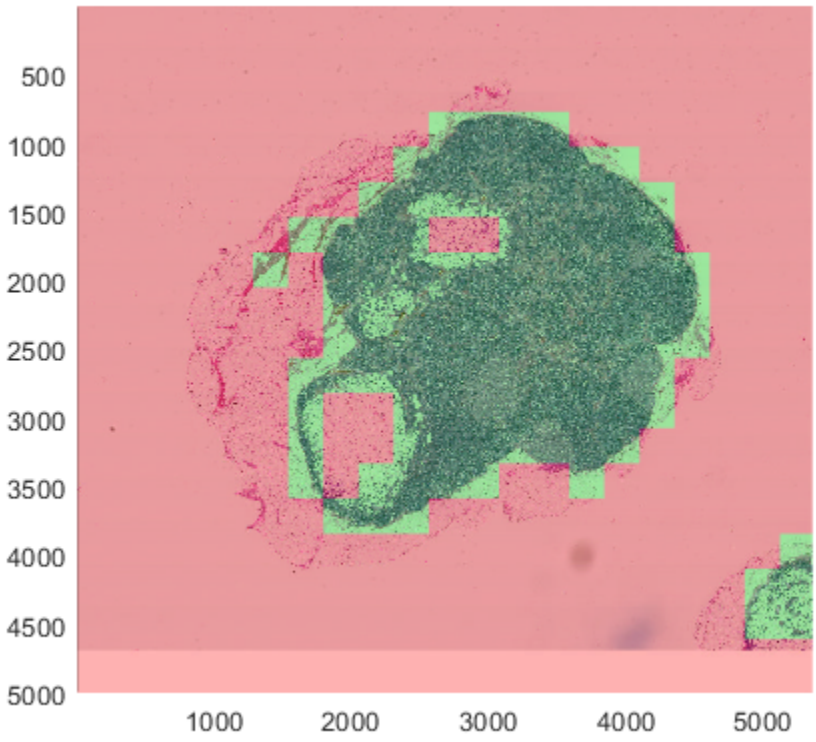
Experiment with different different block sizes, in conjunction with different inclusion thresholds, to get a better fit of the mask over the stained area. By default, the block size for the coarsest resolution level is 625-by-670.

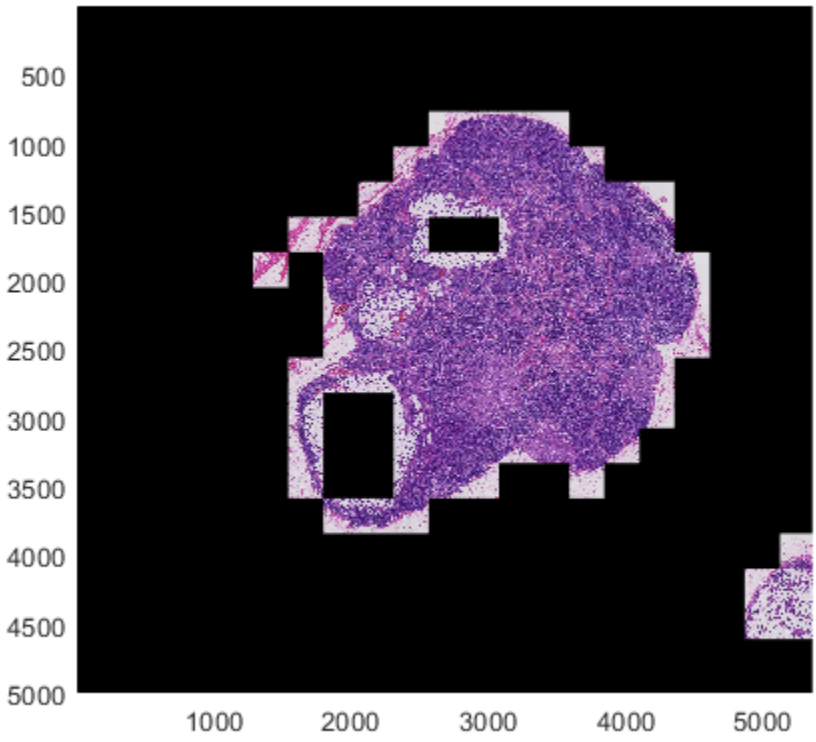
```
showmask(h, bmask, 'InclusionThreshold', 0.06, 'BlockSize', [256 256]);
showmask(h, bmask, 'InclusionThreshold', 0.14, 'BlockSize', [256 256]);
```



When you are satisfied with the mask, use it to segment the lymph node.

```
bls = selectBlockLocations(bim,'BlockSize', [256 256],...
                                'Mask', bmask, 'InclusionThreshold', 0.14);
bregion = apply(bim, @(bs)bs.Data, 'BlockLocationSet', bls);
figure
bigimageshow(bregion);
%
```


500
1000
1500
2000
2500
3000
3500
4000
4500
5000
1000
2000
3000
4000
5000



Display Blocked Image with Label Overlay

Create a blocked image from the sample image `tumor_091R.tif`. This sample image is a training image of a lymph node containing tumor tissue from the CAMELYON16 data set. The image has been modified to have three coarse resolution levels, and has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = blockedImage("tumor_091R.tif");
```

Create a label image at a coarse resolution level.

First get a single-resolution image. By default, `gather` gets data from the coarsest resolution level.

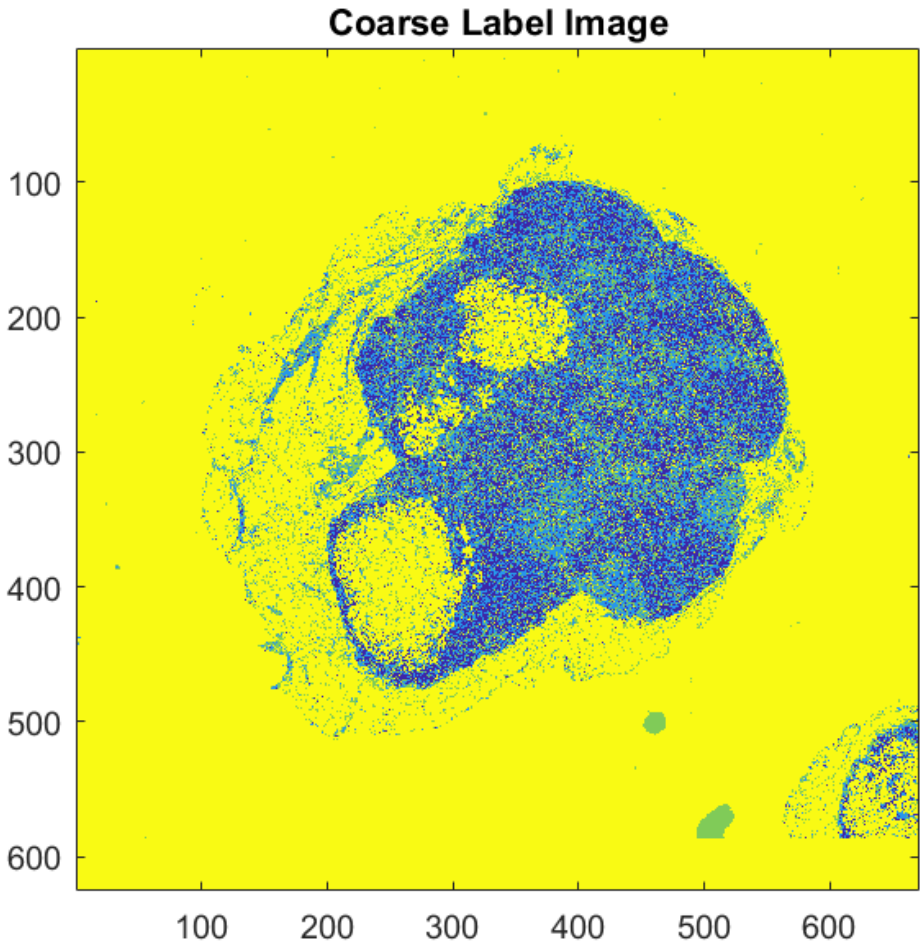
```
cim = gather(bim);
```

Convert the image to grayscale. Use `multithresh` to calculate three threshold values to convert the image into a four-level image.

```
cgim = im2gray(cim);
numClasses = 4;
thresh = multithresh(cgim,numClasses-1);
```

Segment the image into four regions using `imquantize`, specifying the threshold levels returned by `multithresh`.

```
labels = imquantize(cgim,thresh);
imagesc(labels)
axis square
title("Coarse Label Image")
```

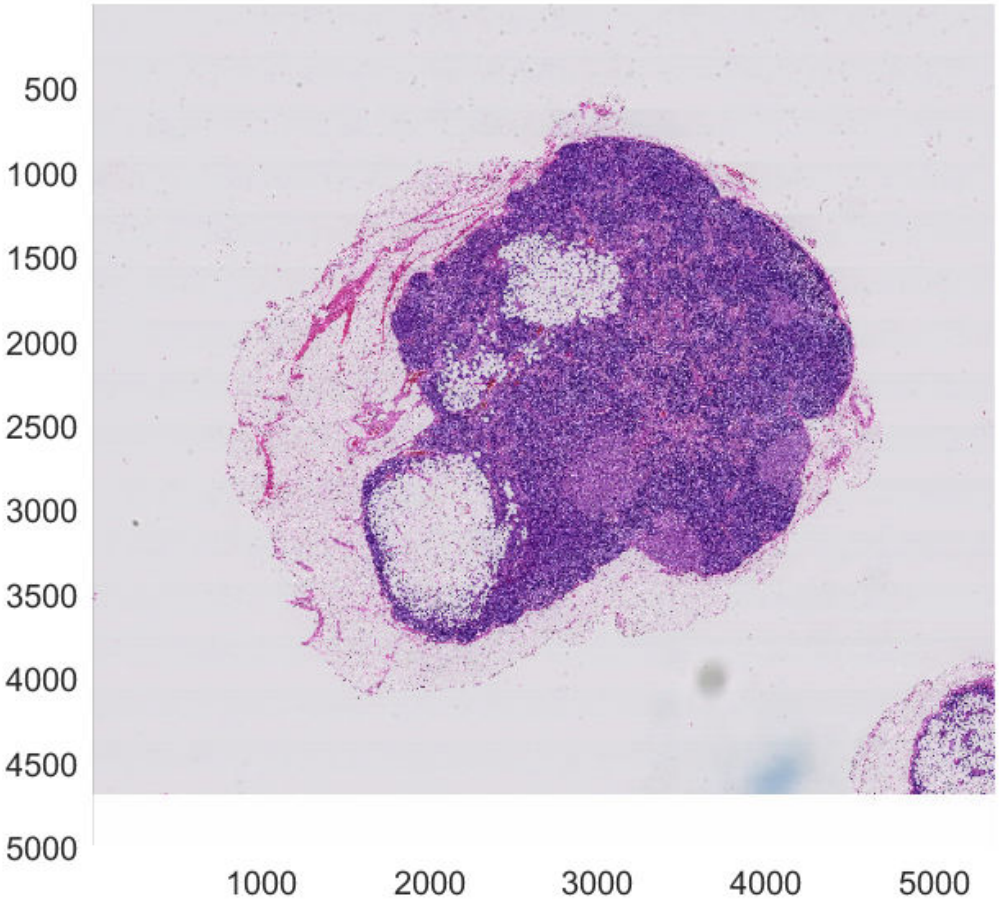


Convert the `labels` image back to a `blockedImage` object, using the same spatial referencing as the original image at the coarsest resolution level.

```
blabels = blockedImage(labels,WorldStart=bim.WorldStart(3,1:2),...
    WorldEnd=bim.WorldEnd(3,1:2));
```

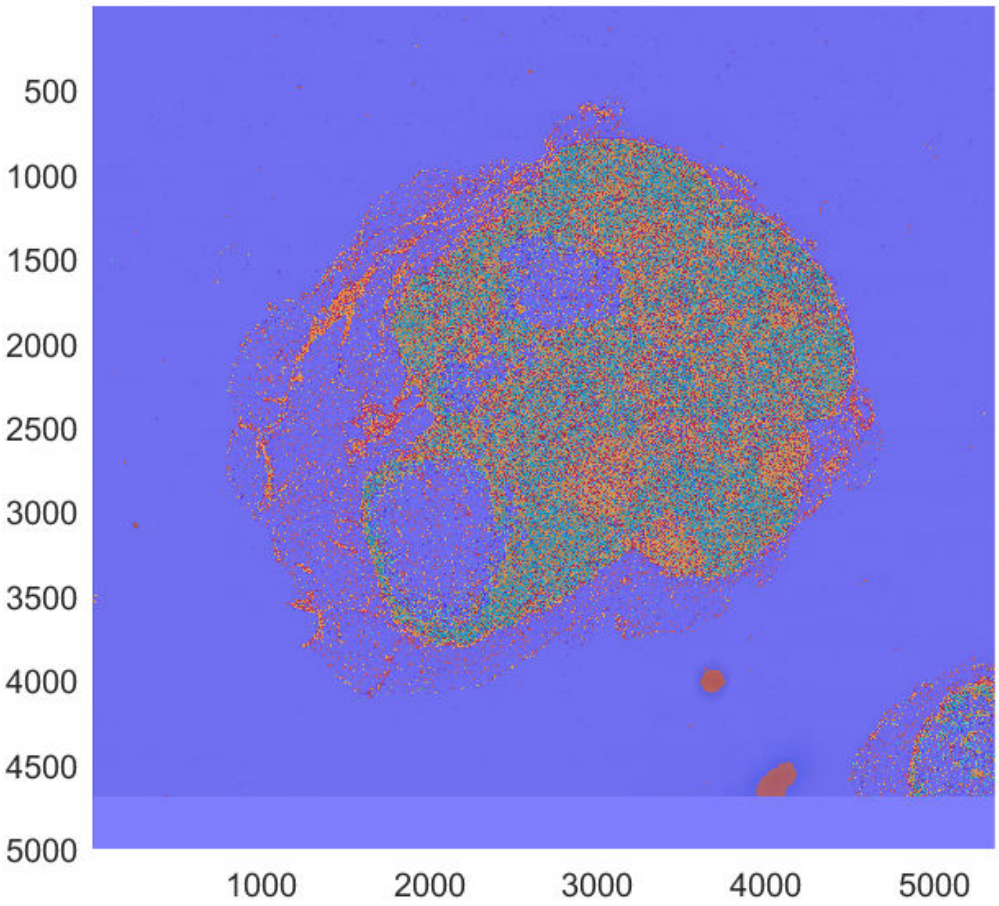
Display the original blocked image.

```
figure
hB = bigimageshow(bim);
```



Overlay the `labels` image on the original blocked image.

```
showlabels(hB,blabels)
```



References

[1] Bejnordi, Babak Ehteshami, Mitko Veta, Paul Johannes van Diest, Bram van Ginneken, Nico Karssemeijer, Geert Litjens, Jeroen A. W. M. van der Laak, et al. “Diagnostic Assessment of Deep Learning Algorithms for Detection of Lymph Node Metastases in Women With Breast Cancer.” JAMA 318, no. 22 (December 12, 2017): 2199–2210. https://doi.org/10.1001/jama.2017.14585.

[2] Grand Challenge. https://camelyon17.grand-challenge.org/Data/.

See Also

`blockedImage` | `imshow`

Introduced in R2019b

hidemask

Hide mask overlay in `bigimageshow` object

Syntax

```
hidemask(b)
```

Description

`hidemask(b)` hides the mask overlay in the specified `bigimageshow` object, `b`.

Input Arguments

`b` — `bigimageshow` object displaying blocked image data

`bigimageshow` object

`bigimageshow` object that is displaying blocked image data, specified as a `bigimageshow` object. The `CData` property of the `bigimageshow` object specifies the blocked image data that is being displayed.

See Also

`showmask` | `blockedImage` | `bigimageshow`

Introduced in R2019b

showmask

Show mask overlay at specified inclusion threshold

Syntax

```
showmask(b,mask)
showmask(b,mask,level)
showmask( ___ ,Name,Value)
```

Description

`showmask(b,mask)` displays the mask, `mask`, as an overlay on the blocked image displayed in the `bigimageshow` object, `b`. The overlay shows the blocks that the `blockedImage` `apply` object function processes with the specified mask. `mask` is a 2-D blocked image object the same size as the displayed image. If `mask` has multiple resolution levels, `bigimageshow` uses the finest level.

- `bigimageshow` displays blocks of the mask that exceed a minimum percentage of nonzero pixels (by default, 50%) with a green tint. These blocks are considered regions of interest and would be selected for processing by the `apply` object function of the `blockedImage` object.
- `bigimageshow` displays blocks of the mask below the minimum percentage of nonzero pixels with a red tint. These blocks are considered background and would not be processed by the `apply` object function.

`showmask(b,mask,level)` overlays a mask on a `bigimageshow` object, `b`, at the specified resolution level of the blocked image data.

`showmask( ___ ,Name,Value)` modifies the appearance of the mask blocks by using name-value pair arguments.

Examples

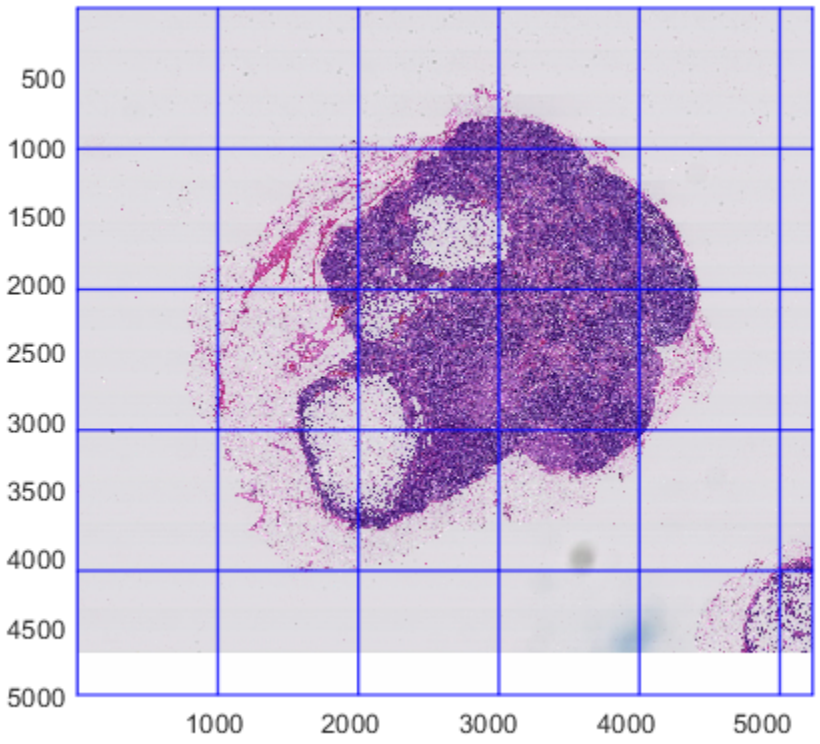
Create Single-Resolution Mask from Multiresolution Blocked Image

Create a blocked image from the sample image tumor_091R.tif. This sample image is a training image of a lymph node containing tumor tissue from the CAMELYON16 data set. The image has been modified to have three coarse resolution levels, and has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = blockedImage('tumor_091R.tif');
```

Display the entire blocked image at the finest resolution level, including a grid of the block boundaries.

```
bshow = bigimageshow(bim,'ResolutionLevel','fine', ...
    'GridVisible','on','GridLevel',1);
```



Create a mask of the coarsest resolution level.

First create a single-resolution image of the coarsest resolution level. By default, the `gather` function gets data from the coarsest resolution level.

```
imcoarse = gather(bim);
```

Convert the coarse image to grayscale.

```
graycoarse = rgb2gray(imcoarse);
```

Binarize the grayscale image. In the binarized image, the object of interest is black and the background is white.

```
bwcoarse = imbinarize(graycoarse);
```

Take the complement of the binarized image. The resulting mask follows the convention in which the object of interest is white and the background is black.

```
mask = imcomplement(bwcoarse);
```

Create a blocked image containing the mask.

Use the same spatial referencing as the original blocked image. Determine the coarsest resolution level and capture the spatial referencing information of the blocked image at the first two dimensions at that level.

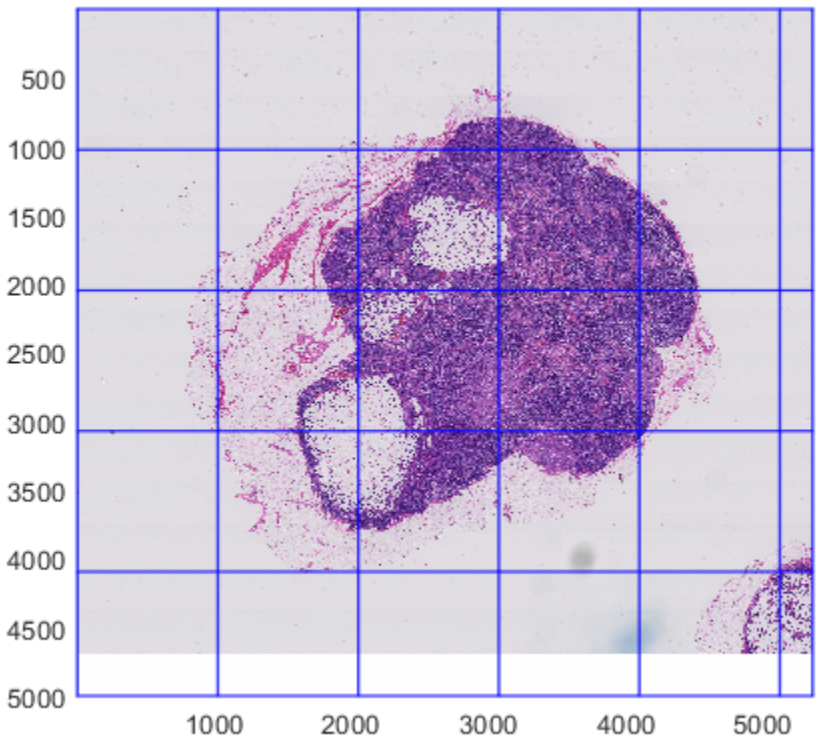
```
coarsestLevel = bim.NumLevels;
originalWorldStartCoarsest = bim.WorldStart(coarsestLevel,1:2);
originalWorldEndCoarsest = bim.WorldEnd(coarsestLevel,1:2);
```

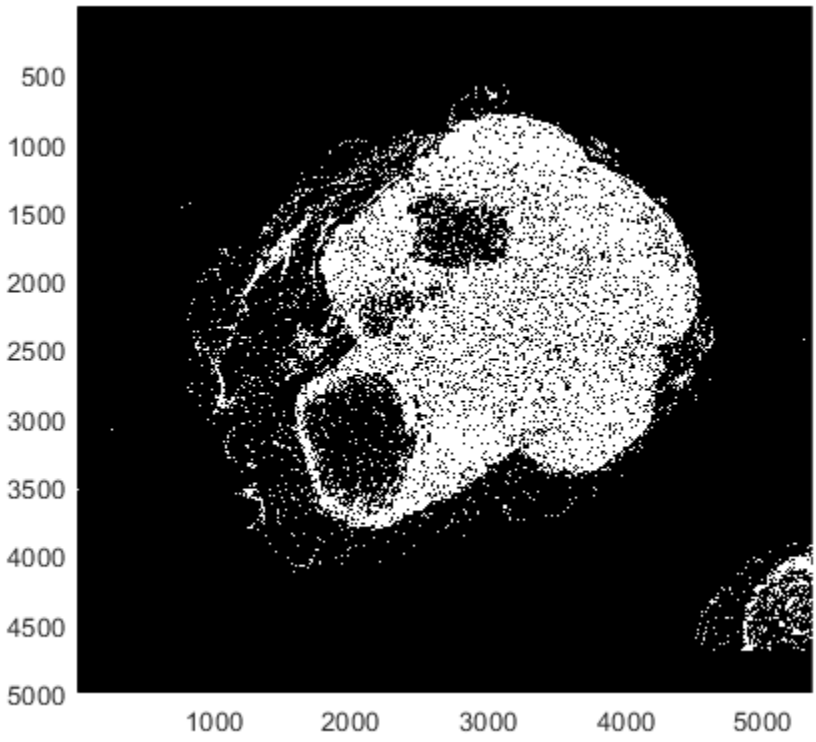
Create the blocked image for the mask.

```
bmask = blockedImage(mask,'WorldStart',originalWorldStartCoarsest, ...
   'WorldEnd',originalWorldEndCoarsest);
```

Display the mask image.

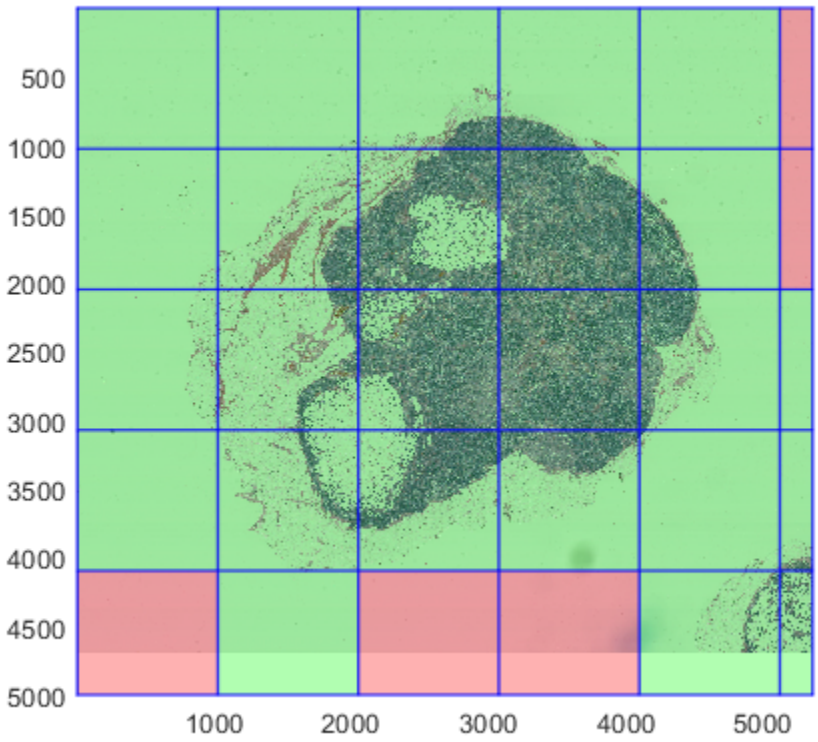
```
figure
bigimageshow(bmask)
```





Overlay the mask on the display of the original blocked image using the `showmask` function. To highlight all blocks that contain at least one nonzero mask pixel, specify an inclusion threshold of `0`.

```
showmask(bshow,bmask,'InclusionThreshold',0)
```



Input Arguments

`b` — `bigimageshow` object displaying blocked image data

`bigimageshow` object

`bigimageshow` object displaying blocked image data, specified as a `bigimageshow` object. The `CData` property of the `bigimageshow` object specifies the blocked image data that is being displayed.

`mask` — Mask

`[]` (default) | single-resolution `blockedImage` object

Mask, specified as a single-resolution `blockedImage` object with a `ClassUnderlying` property value of `logical`. The spatial extents of the mask must be the same as the blocked image data at the displayed resolution level.

`level` — Resolution level

positive integer

Resolution level at which to display the bocked image data, specified as a positive integer.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `showmask(b,mask,'InclusionThreshold',0.4)`

`Alpha` — Mask transparency
`0.3` (default) | scalar value in the range [0, 1]

Mask transparency, specified as a scalar value in the range [0, 1]. A value of `1` means the mask is completely opaque and a value of `0` means the mask is completely transparent.

`BlockSize` — Block size used with `apply` function
1-by-2 vector of positive integers

Block size used with the `apply` function, specified as a 1-by-2 vector of positive integers of the form `[numrows numcols]`. The default value is equal to the `BlockSize` property of the `blockedimage`object in `b`.

`InclusionThreshold` — Inclusion threshold
`0.5` (default) | number in the range [0, 1]

Inclusion threshold, specified as a number in the range [0, 1]. The inclusion threshold indicates the minimum fraction of nonzero pixels in a mask block required to consider the mask block as a region of interest.

- When the inclusion threshold is `0`, the `showmask` function displays a mask block when at least one pixel in the mask block is nonzero.
- When the inclusion threshold is `1`, the `showmask` function displays a mask block only when all pixels in the mask block are nonzero.

Tips

- When you call `showmask` for the first time, `bigimageshow` calculates the ratio of nonzero to zero mask pixels for each block at the finest level of the displayed `blockedImage`. This calculation takes some time to complete, so there can be some delay displaying the mask. When you make subsequent calls to `showmask`, the function reuses the computed inclusion values and updates the displayed mask more quickly.

See Also
`hidemask` | `blockedImage` | `apply` | `bigimageshow`

Topics
“Process Blocked Images Efficiently Using Mask”

Introduced in R2019b

hidelabels

Hide label overlay on `bigimageshow` object

Syntax

```
hidelabels(b)
```

Description

`hidelabels(b)` hides the label overlay on the specified `bigimageshow` object, `b`.

Input Arguments

`b` — `bigimageshow` object displaying blocked image data

`bigimageshow` object

`bigimageshow` object displaying blocked image data, specified as a `bigimageshow` object.

See Also

`showlabels` | `bigimageshow` | `blockedImage`

Introduced in R2021b

showlabels

Display label overlay on `bigimageshow` object

Syntax

```
showlabels(b,labels)
showlabels(b,labels,Name=Value)
```

Description

`showlabels(b,labels)` displays the labels in the 2-D blocked image `labels` as an overlay on the blocked image displayed in the `bigimageshow` object, `b`.

`showlabels(b,labels,Name=Value)` uses name-value arguments to modify the appearance of the overlay.

Example: `showlabels(b,labels,Colormap="jet")` displays the label overlay using the `"jet"` colormap.

Examples

Display Blocked Image with Label Overlay

Create a blocked image from the sample image `tumor_091R.tif`. This sample image is a training image of a lymph node containing tumor tissue from the CAMELYON16 data set. The image has been modified to have three coarse resolution levels, and has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = blockedImage("tumor_091R.tif");
```

Create a label image at a coarse resolution level.

First get a single-resolution image. By default, `gather` gets data from the coarsest resolution level.

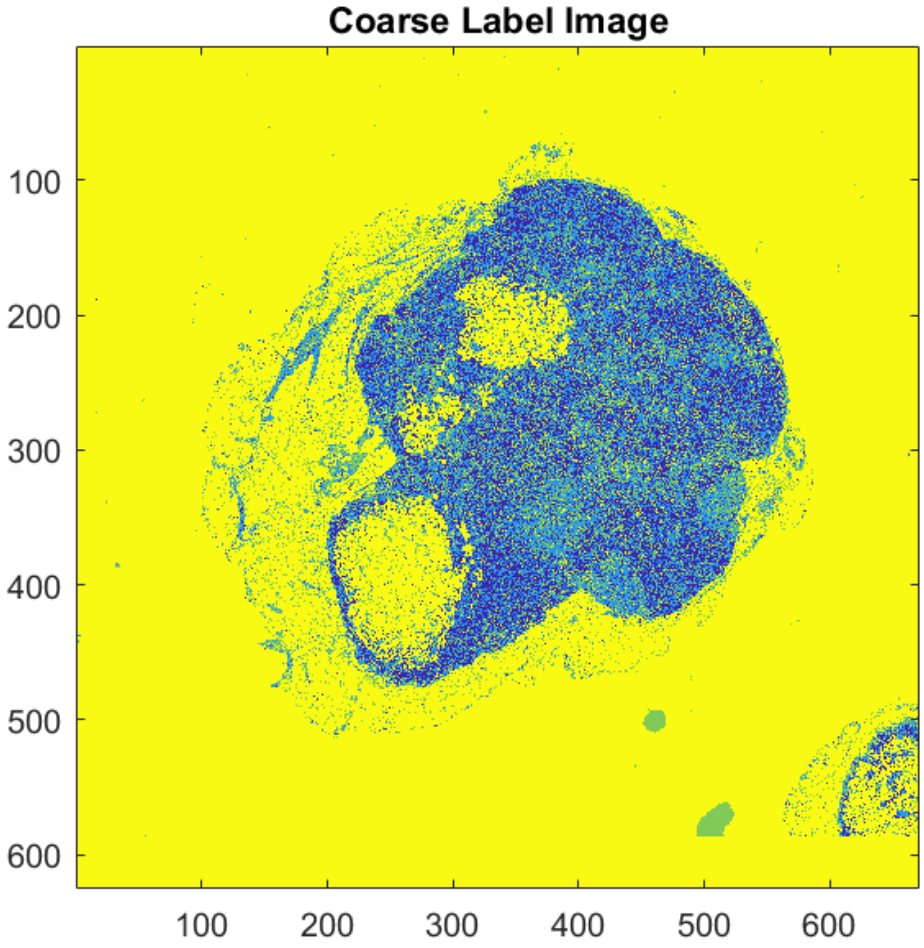
```
cim = gather(bim);
```

Convert the image to grayscale. Use `multithresh` to calculate three threshold values to convert the image into a four-level image.

```
cgim = im2gray(cim);
numClasses = 4;
thresh = multithresh(cgim,numClasses-1);
```

Segment the image into four regions using `imquantize`, specifying the threshold levels returned by `multithresh`.

```
labels = imquantize(cgim,thresh);
imagesc(labels)
axis square
title("Coarse Label Image")
```

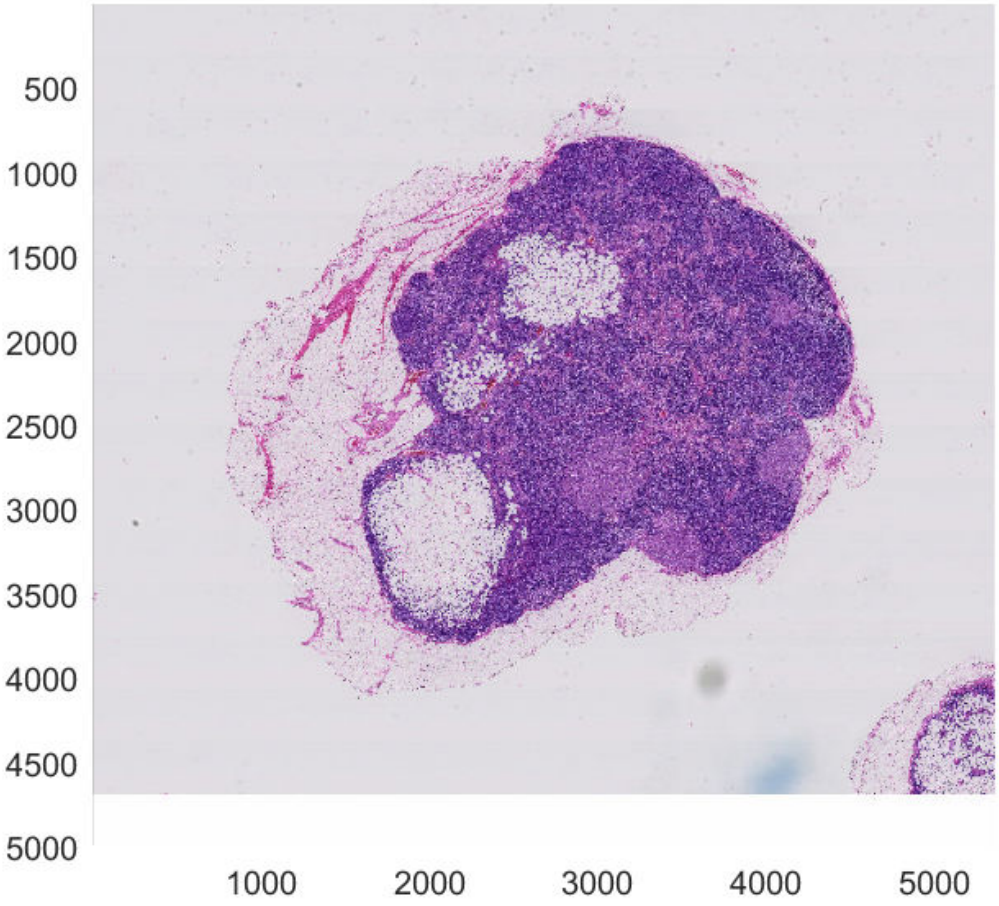


Convert the `labels` image back to a `blockedImage` object, using the same spatial referencing as the original image at the coarsest resolution level.

```
blabels = blockedImage(labels,WorldStart=bim.WorldStart(3,1:2),...
    WorldEnd=bim.WorldEnd(3,1:2));
```

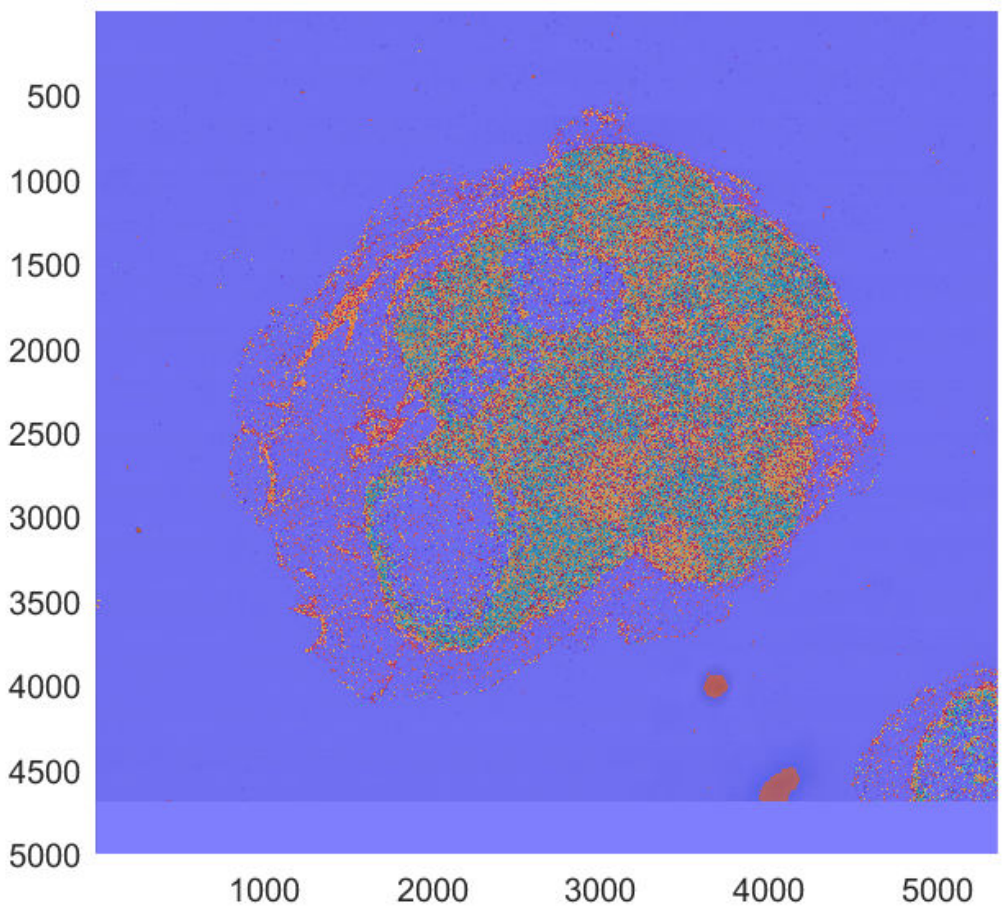
Display the original blocked image.

```
figure
hB = bigimageshow(bim);
```



Overlay the `labels` image on the original blocked image.

```
showlabels(hB,blabels)
```



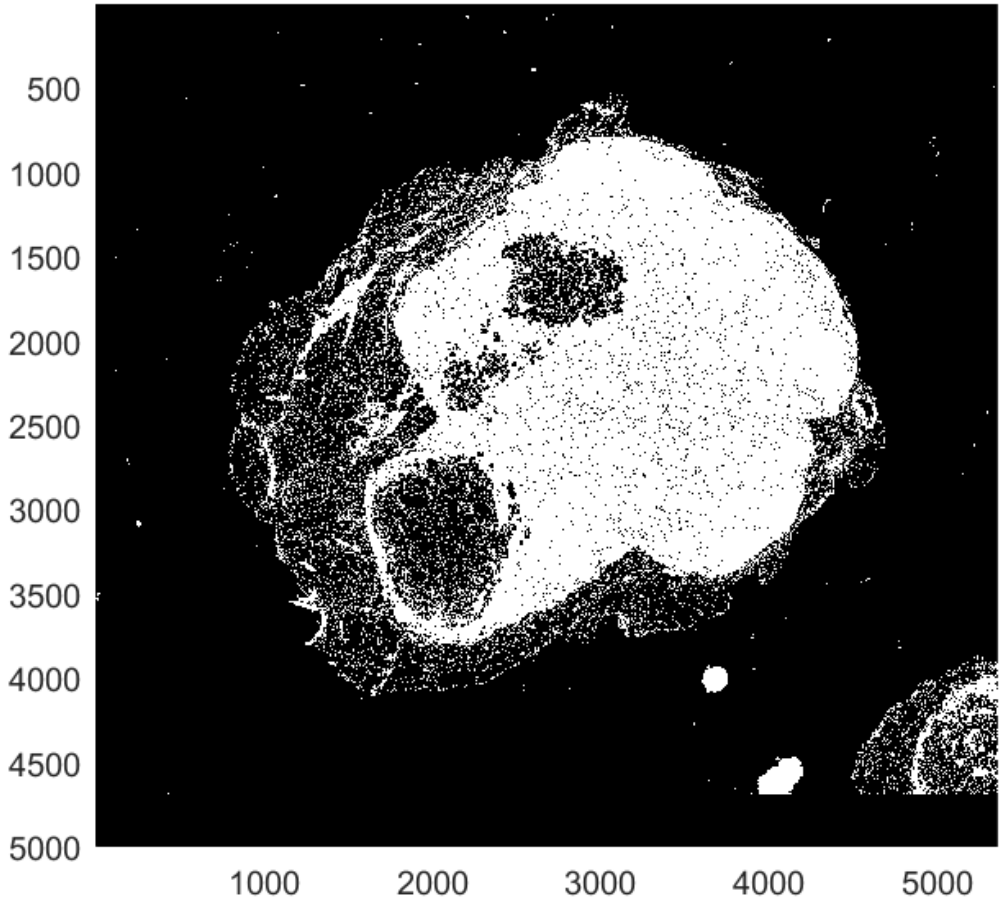
Modify Transparency of Blocked Image Label Overlay

Create a blocked image using a modified version of image `tumor_091R.tif` from the CAMELYON16 data set. The original image is a training image of a lymph node containing tumor tissue. The original image has eight resolution levels, and the finest level has resolution 53760-by-61440. The modified image has only three coarse resolution levels. The spatial referencing of the modified image has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = blockedImage("tumor_091R.tif");
```

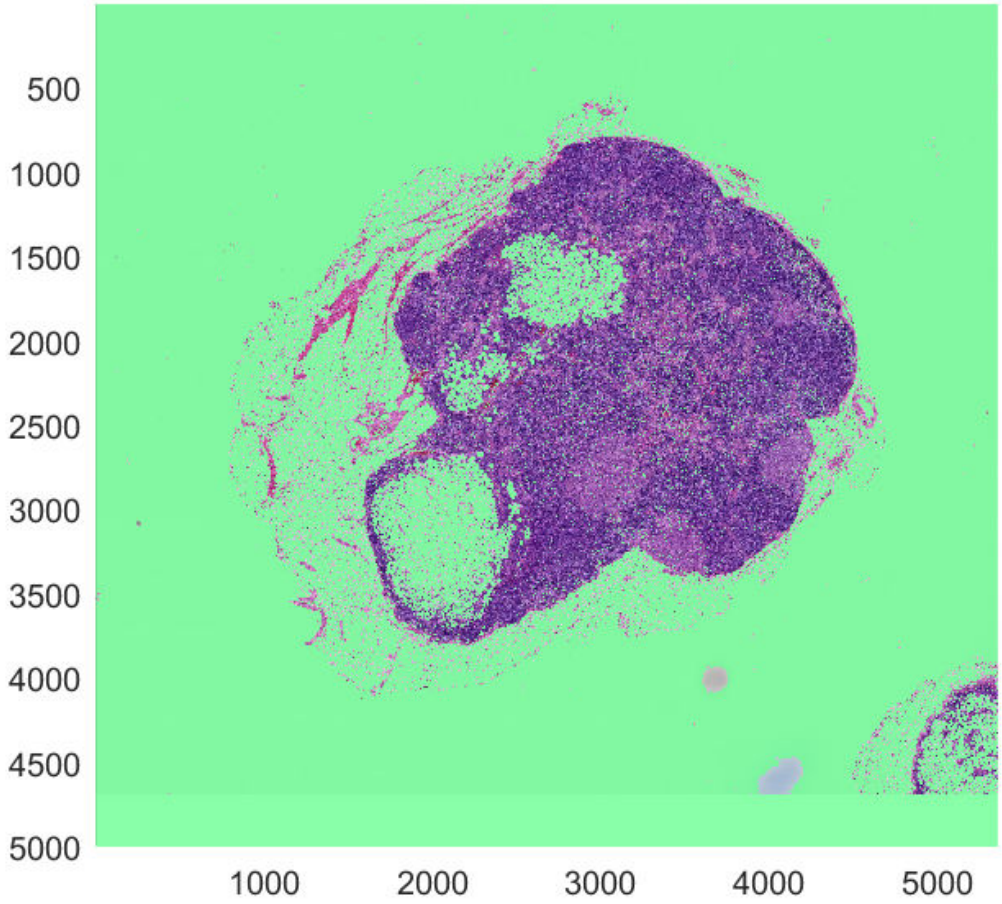
Create a mask at the coarsest resolution level and display it.

```
blabels = apply(bim,@(bs)rgb2lightness(bs.Data)<80,Level=3);
hbim = bigimageshow(blabels);
```



Display the original blocked image with the mask as a label overlay. Use the `Alphadata` and `Alphamap` name-value arguments to display the mask background overlay as translucent and the mask foreground overlay as fully transparent.

```
hbim = bigimageshow(bim);
showlabels(hbim,blabels,AlphaData=blabels,Alphamap=[0.8 0])
```



Display Categorical Labels as Blocked Image Overlay

Load a file containing an image, `img`, and its corresponding pixel label data, `label`. Convert the image to a `blockedImage` object.

```
load("buildingPixelLabeled.mat")
bim = blockedImage(img);
```

Create a blocked image of the categorical pixel label data. Display the order of the four label categories.

```
blabel = blockedImage(label);
labels = categories(blabel.InitialValue)
```

```
labels = 4x1 cell
    {'sky'     }
    {'grass'   }
    {'building'}
```

```
    {'sidewalk'}
```

Define a custom colormap specifying RGB triplets for each category. The first row of `cmap` corresponds to undefined pixel labels, and the remaining four rows correspond to each categorical label. Assign the sky overlay to display blue and the grass overlay to display green.

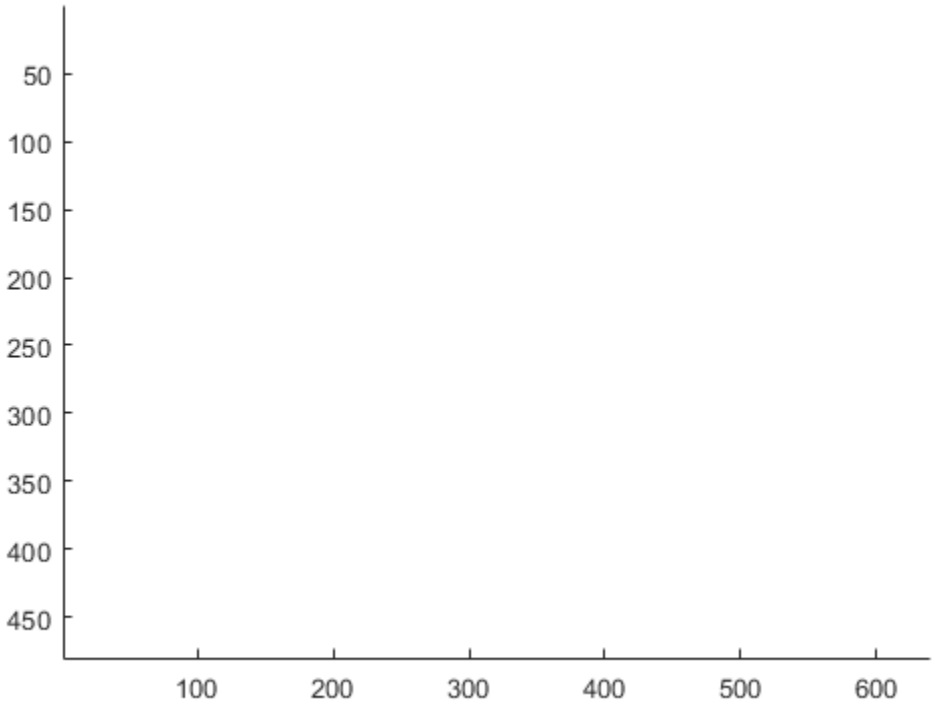
```
cmap = [0 0 0; 0 0 1; 0 1 0; 0 0 0; 0 0 0];
```

Define the transparency map.

```
amap = [.5 .5 0 0];
```

Display the original unlabeled blocked image and the label overlay on the same axes. The `AlphaData` and `Alphamap` name-value arguments map each defined category in `blabel` to the corresponding element in `amap`, making the sky and grass overlays translucent and the building and sidewalk overlays fully transparent. Undefined pixel labels map to the first element in `amap`.

```
hbim = bigimageshow(bim);
showlabels(hbim,blabel,AlphaData=blabel,Alphamap=amap,Colormap=cmap)
```



Input Arguments

b — `bigimageshow` object displaying 2-D blocked image data
`bigimageshow` object

`bigimageshow` object displaying 2-D blocked image data, specified as a `bigimageshow` object.

`labels` — Labels
2-D `blockedImage` object | numeric matrix | logical matrix

Labels, specified as a 2-D `blockedImage` object, numeric matrix, or logical matrix. If `labels` is a `blockedImage` object with multiple resolution levels, `showlabels` selects the level closest to the current `ResolutionLevel` of `b` for display. If `labels` is specified as a numeric or logical matrix, `showlabels` converts the matrix to a `blockedImage` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `showlabels(b,labels,AlphaData=labels,Alphamap=[0 0.1 0.1 0.5 1])` specifies five varying transparency values in the overlay by mapping each element in `AlphaData` to an index in `Alphamap`.

`AlphaData` — Transparency data
`1` (default) | numeric scalar | 2-D `blockedImage` object

Transparency data, specified in one of these forms:

- Numeric scalar — Applies consistent transparency across the entire image.
- 2-D `blockedImage` object — Applies varying transparency values based on the underlying pixel values. Transparency data must be the same size as the `blockedImage` object displayed in `b`.

The function interprets the numeric scalar or underlying pixel values as indices of the transparency map specified by `Alphamap`. Values with a decimal portion are fixed to the nearest lower integer:

- If the values are of type `double` or `single`, values of `1` or less map to the first element of `Alphamap`. Values equal to or greater than the length of `Alphamap` map to the last element of `Alphamap`.
- If the values are of an integer data type, then values of `0` or less map to the first element of `Alphamap`. Values equal to or greater than the length of `Alphmap` map to the last element of `Alphamap` (or up to the range limits of the data type). The integer data types are `uint8`, `uint16`, `uint32`, `uint64`, `int8`, `int16`, `int32`, and `int64`.
- If the values are of type `logical`, values of `0` map to the first element of `Alphamap` and values of `1` map to the second element of `Alphamap`.
- If `AlphaData` is a categorical `blockedImage` object, the defined categories map to corresponding elements of `Alphamap`. You can verify the order in which categories map to `Alphamap` by using the `categories` function.

`Alphamap` — Transparency map
`0.5` (default) | numeric scalar or *m*-element vector

Transparency map, specified as a numeric scalar or *m*-element vector. All values must be in the range [0, 1], where `0` is transparent and `1` is opaque. You can specify `Alphamap` as a 1-by-*m* or *m*-by-1 vector, where *m* is the number of transparency values.

`Colormap` — Colormap
`"jet"` (default) | *c*-by-3 colormap | string scalar | character vector

Colormap, specified as one of these values:

- A *c*-by-3 colormap, where *c* is the number of colors in the colormap. RGB triplets in each row of the colormap must be normalized to the range [0, 1]. When *c* is greater than the number of labels *l* in the blocked image `labels`, `showlabels` uses only the first *l* colors of the colormap.
- A string scalar or character vector corresponding to one of the valid inputs to the `colormap` function. `showlabels` permutes the specified colormap so that adjacent labels are more distinct.

Example: `[0.2 0.1 0.5; 0.1 0.5 0.8]`

Example: `"hot"`

Data Types: `single` | `double` | `char` | `string`

See Also

`hidelabels` | `bigimageshow` | `blockedImage`

Introduced in R2021b

blendexposure

Create well-exposed image from images with different exposures

Syntax

```
J = blendexposure(I1,I2,...,In)
J = blendexposure(I1,I2,...,In,Name,Value)
```

Description

`J = blendexposure(I1,I2,...,In)` blends grayscale or RGB images that have different exposures. `blendexposure` blends the images based on their contrast, saturation, and well-exposedness, and returns the well-exposed image, `J`.

`J = blendexposure(I1,I2,...,In,Name,Value)` blends images that have different exposures, using name-value pairs to adjust how each input image contributes to the blended image.

Examples

Blend Images with Strong Light Sources

Read a series of images with different exposures that were captured from a fixed camera with no moving objects in the scene.

```
I1 = imread('car_1.jpg');
I2 = imread('car_2.jpg');
I3 = imread('car_3.jpg');
I4 = imread('car_4.jpg');
```

Display the images. In the underexposed images, only bright regions like headlights have informative details. Conversely, the headlights are saturated in the overexposed images, and the best contrast comes from darker regions such as the brick floor and the roof.

```
montage({I1,I2,I3,I4})
```

Blend the images using exposure fusion. By default, the `blendexposure` function attempts to suppress highlights from strong light sources. For comparison, also blend the images without suppressing the highlights. Display the two results.

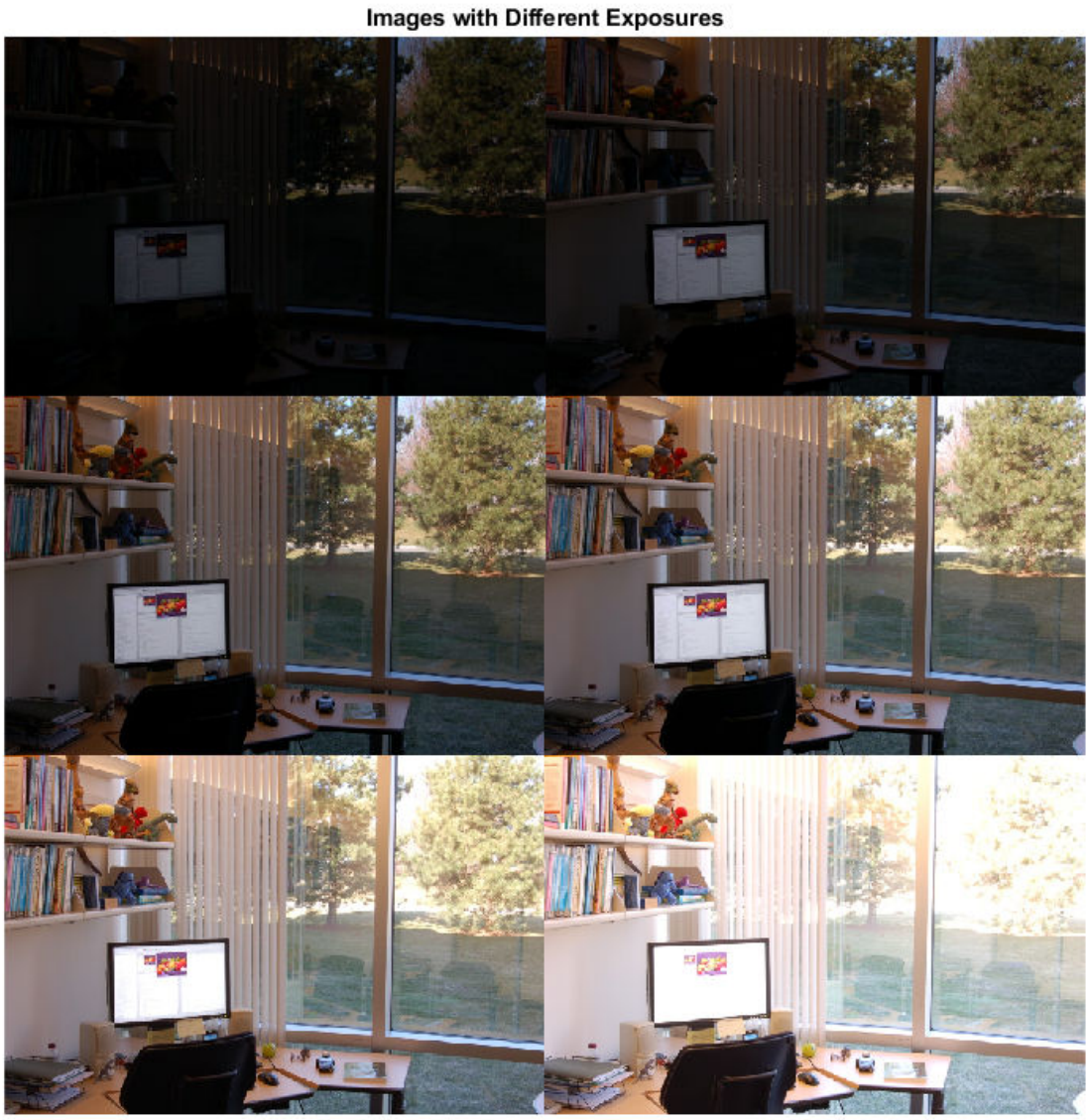
```
E = blendexposure(I1,I2,I3,I4);
F = blendexposure(I1,I2,I3,I4,'ReduceStrongLight',false);
montage({E,F})
title('Exposure Fusion With (Left) and Without (Right) Strong Light Suppression')
```

In the fused images, bright regions and dark regions retain informative details. With strong light suppression, the shape of the headlights is identifiable, and saturated pixels do not extend past the boundary of the headlights. Without strong light perception, the shape of the headlights is not identifiable, and there are saturated pixels in the reflection of the headlights on the ground and on some parts of the other cars.

Blend Images of Stationary Scene Using Exposure Fusion

Read a series of images with different exposures. The images were captured from a fixed camera, and there are no moving objects in the scene.

```
I1 = imread('office_1.jpg');
I2 = imread('office_2.jpg');
I3 = imread('office_3.jpg');
I4 = imread('office_4.jpg');
I5 = imread('office_5.jpg');
I6 = imread('office_6.jpg');
montage({I1,I2,I3,I4,I5,I6})
title('Images with Different Exposures')
```



Blend the registered images using exposure fusion, optionally varying the weight of contrast, saturation and well-exposedness in the fusion, and without reducing strong light sources. Display the result.

```
E = blendexposure(I1,I2,I3,I4,I5,I6,'contrast',0.8,...
    'saturation',0.8,'wellexposedness',0.8,'reduceStrongLight',false);
imshow(E)
title('Blended Image Using Exposure Fusion')
```

Input Arguments

`I1,I2,...,In` — Grayscale or RGB images

m-by-*n* numeric matrices | *m*-by-*n*-by-3 numeric arrays

Grayscale or RGB images, specified as a series of *m*-by-*n* numeric matrices or *m*-by-*n*-by-3 numeric arrays. All images must have the same size and data type.

Data Types: `single` | `double` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `blendexposure(I1,I2,I3,'Contrast',0.5,'Saturation',0.9)`

`Contrast` — Relative weight given to contrast

1 (default) | numeric scalar in the range [0, 1]

Relative weight given to contrast during blending, specified as the comma-separated pair consisting of `'Contrast'` and a numeric scalar in the range [0, 1].

`Saturation` — Relative weight given to saturation
1 (default) | numeric scalar in the range [0, 1]

Relative weight given to saturation during blending, specified as the comma-separated pair consisting of `'Saturation'` and a numeric scalar in the range [0, 1].

`Wellexposedness` — Relative weight given to exposure quality
1 (default) | numeric scalar in the range [0, 1]

Relative weight given to exposure quality during blending, specified as the comma-separated pair consisting of `'Wellexposedness'` and a numeric scalar in the range [0, 1]. The exposure quality of each image is based on the divergence of the pixel intensities from a model of pixels with good exposure.

`ReduceStrongLight` — Reduce strong light
`true` (default) | `false`

Reduce strong light, specified as the comma-separated pair consisting of `'ReduceStrongLight'` and `true` or `false`. If `'ReduceStrongLight'` is `true`, then `blendexposure` attempts to suppress highlights from strong light sources in the images.

Note If the input images do not have strong light sources and you specify `ReduceStrongLight` as `true`, then the output image `J` has less contrast.

Output Arguments

`J` — Fused image
numeric matrix or array

Fused image, returned as a numeric matrix or array of the same size and data type as the input images `I1,I2,...,In`.

Tips

- To blend images of moving scenes or with camera jitter, first register the images by using the `imregmtb` function. `imregmtb` considers only translations, not rotations or other types of geometric transformations, when registering the images.

Algorithms

The `blendexposure` function computes the weight of each quality measure as follows:

- Contrast weights are computed using Laplacian filtering.
- Saturation weights are computed from the standard deviation of each image.
- Well-exposedness is determined by comparing parts of the image to a Gaussian distribution with a mean of 0.5 and a standard deviation of 0.2.
- Strong light reduction weights are computed as a mixture of the other three weights, multiplied by a Gaussian distribution with a fixed mean and variance.

The weights are decomposed using Gaussian pyramids for seamless blending with a Laplacian pyramid of the corresponding image, which helps preserve scene details.

References

[1] Mertens, T., J. Kautz, and F. V. Reeth. "Exposure Fusion." *Pacific Graphics 2007: Proceedings of the Pacific Conference on Computer Graphics and Applications*. Maui, HI, 2007, pp. 382–390.

See Also

`imregmtb` | `makehdr` | `tonemap`

Introduced in R2018a

BINBlocks

Read and write blocks of blocked image data as binary files

Description

The `BINBlocks` object is an adapter that reads and writes blocked image data as binary files.

When writing to disk, the object creates an individual binary file with simple header information for each block. The object saves the binary files in a folder. For multiresolution images, the object creates one subfolder for each resolution level. The object also creates and saves a MAT file with information about the blocked image, including the image size, block size, and data type.

The table lists the support that the `BINBlocks` object has for various `blockedImage` capabilities.

Capabilities	Support
Data types	All numeric and logical data types of any dimension
Multiple resolution levels	Yes
Process blocks in parallel using the `apply` function	Yes
Resume block processing using the `apply` function	Yes

Creation

Syntax

```
adapter = images.blocked.BINBlocks
```

Description

`adapter = images.blocked.BINBlocks` creates a `BINBlocks` object that reads and writes blocked image data as binary files, with one binary file for each block.

See Also

`blockedImage`

Introduced in R2021a

GenericImage

Read and write blocked image data as single image file

Description

A `GenericImage` object is an adapter that reads and writes 2-D single-resolution blocked image data as a single image file.

When writing to disk, if the blocked image has any additional metadata in the `UserData` property, then the `GenericImage` object writes the data to a separate MAT file with the same file name.

By default, the object saves image data as a PNG file. To use a different file format, create the object and then change the file format using the `BlockFormat` property. For example, to write a blocked image as a JPG file, use this code.

```
adapter = images.blocked.GenericImage;
adapter.Extension = "jpg";
```

When reading from disk, the object reads all image data into memory as a single block. To access smaller blocks of image data, create a `blockedImage` object from the image file and specify a block size that is smaller than the full size of the image.

The table lists the support that the `GenericImage` object has for various `blockedImage` capabilities.

Capabilities	Support
Data types	This object supports 2-D images only: • Binary images of size *m*-by-*n* with data type `logical` • Grayscale images of size *m*-by-*n* with data type `uint8` • Truecolor (RGB) images of size *m*-by-*n*-by-3 with data type `uint8`
Multiple resolution levels	No
Process blocks in parallel using the `apply` function	No
Resume block processing using the `apply` function	Limited. Only useful when processing an array of `blockedImage` objects.

Creation

Description

`adapter = images.blocked.GenericImage` creates a `GenericImage` object that reads and writes blocked image data as a single image file.

Properties

`Extension` — Preferred file format

`"png"` (default) | string

Preferred file format, specified as a string. The `apply` function of `blockedImage` uses this value when creating the output locations automatically.

Example: `"jpg"`

Examples

Save Single Level Image Data in Single PNG File

Create a blocked image.

```
bim = blockedImage("tumor_091R.tif");
```

Write blocked image data to a PNG file using the `write` function. Create a `GenericImage` object as the adapter for the `write` function to use.

```
writeAdapter = images.blocked.GenericImage;
write(bim,"tumorL3.png","Adapter",writeAdapter,"Levels",3);
```

Create a blocked image from the PNG file that you just created. The `blockedImage` object automatically picks the appropriate adapter for the data.

```
bgi = blockedImage("tumorL3.png");
disp(bgi.Adapter.Format)
```

```
png
```

Compatibility Considerations

`Format` property is not recommended

Starting in R2021b, the `Format` property is no longer recommended. Use the `Extension` property instead. If you specify the `Format` property using dot notation, then the value is assigned to the `Extension` property.

See Also

`blockedImage` | `GenericImageBlocks` | `TIFF` | `InMemory`

Introduced in R2021a

GenericImageBlocks

Read and write blocks of blocked image data as image files

Description

The `GenericImageBlocks` object is an adapter that reads and writes 2-D blocked image data as image files.

When writing to disk, the object creates an individual image file for each block and saves the image files in a folder. For multiresolution images, the object creates one subfolder for each resolution level. The object also creates and saves a MAT file with information about the blocked image, including the image size, block size, and data type.

By default, the object writes images as PNG files. To use a different file format, create the object and then change the file format using the `BlockFormat` property. For example, to write images as JPG files, use this code.

```
adapter = images.blocked.GenericImageBlocks;
adapter.BlockFormat = "jpg";
```

The table lists the support that the `GenericImageBlocks` object has for various `blockedImage` capabilities.

Capabilities	Support
Data types	This object supports 2-D images only: • Binary images of size *m*-by-*n* with data type `logical` • Grayscale images of size *m*-by-*n* with data type `uint8` • Truecolor (RGB) images of size *m*-by-*n*-by-3 with data type `uint8`
Multiple resolution levels	Yes
Process blocks in parallel using the `apply` function	Yes
Resume block processing using the `apply` function	Yes

Creation

Description

`adapter = images.blocked.GenericImageBlocks` creates a `GenericImageBlocks` object that reads and writes blocked image data as image files, with one image file per block.

Properties

`BlockFormat` — Image file format for each block of data

`"png"` (default) | string

Image file format for each block of data, specified as a string that identifies one of the formats supported by `imwrite`.

Example: `"tif"`

Data Types: `string`

Examples

Save Image in Folder with One TIFF File Per Block

Create blocked image.

```
bim = blockedImage('tumor_091R.tif');
```

Write image data to files. Specify the `images.blocked.GenericImageBlocks` adapter.

```
wa = images.blocked.GenericImageBlocks();
wa.BlockFormat = "tif";
write(bim, "dirOfTIFFs", "Adapter", wa);
```

Create a blocked image from the folder of images. The `blockedImage` object automatically picks the appropriate adapter.

```
bt = blockedImage("dirOfTIFFs");
disp(bt.Adapter)
```

```
  GenericImageBlocks with properties:

    BlockFormat: "tif"
```

See Also

`blockedImage` | `GenericImage`

Introduced in R2021a

H5

Read and write blocked image data as single H5 file

Description

The `H5` object is an adapter that reads and writes blocked image data as a single chunked HDF5 file.

When writing to disk, if the blocked image has any additional metadata in the `UserData` property, then the `H5` object writes the data to a separate MAT file with the same file name.

The object supports lossless compression. By default, the compression level is set to `1`. To use a different compression level, create the object and then change the compression level using the `GZIPLevel` property. You can also use this property to turn off compression. For example, to use a compression level of `3`, use this code.

```
adapter = images.blocked.H5;
adapter.GZIPLevel = 3;
```

The table lists the support that the `H5` object has for various `blockedImage` capabilities.

Capabilities	Support
Data types	All numeric and logical data types of any dimension. The object writes logical data as `uint8`.
Multiple resolution levels	Yes
Process blocks in parallel using the `apply` function	No
Resume block processing using the `apply` function	No

Creation

Description

`adapter = images.blocked.H5` creates an `H5` object that reads and writes blocked image data as a single chunked HDF5 file.

Properties

`GZIPLevel` — GZIP compression level

`1` (default) | numeric scalar in the range [0, 9]

GZIP compression level, specified as a numeric scalar in the range [0, 9]. This value controls the level of GZIP (lossless) compression. The value `0` turns off compression. Higher values attempt to increase the level of compression and reduce the file size at the cost of higher runtimes.

Data Types: `double`

Examples

Save Single Level Image in Single HDF5 File

Create a blocked image.

```
bim = blockedImage('tumor_091R.tif');
```

Write blocked image data to a HDF5 file using the blocked image write object function. Specify the `images.blocked.H5` adapter for use by the `write` object function.

```
wa = images.blocked.H5();
wa.GZIPLevel = 5; % Slower, but results in smallest file size
write(bim, "tumor1.h5", "Adapter", wa);
```

Create a blocked image from the HDF5 file you just created. The `blockedImage` object automatically picks the appropriate adapter for the data.

```
bh5 = blockedImage("tumor1.h5");
disp(bh5.Adapter.Extension)
```

```
h5
```

See Also

`blockedImage` | `H5Blocks`

Introduced in R2021a

H5Blocks

Read and write blocks of blocked image data as H5 files

Description

The `H5Blocks` object is an adapter that reads and writes blocked image data as chunked H5 files, with one H5 file for each block.

When writing to disk, the object creates an individual H5 file for each block and saves the image files in a folder. For multiresolution images, the object creates one subfolder for each resolution level. The object also creates and saves a MAT file with information about the blocked image, including the image size, block size, and data type.

The object supports lossless compression. By default, the compression level is set to `1`. To use a different compression level, create the object and then change the compression level using the `GZIPLevel` property. You can also use this property to turn off compression. For example, to use a compression level of `3`, use this code.

```
adapter = images.blocked.H5Blocks;
adapter.GZIPLevel = 3;
```

The table lists the support that the `H5Blocks` object has for various `blockedImage` capabilities.

Capabilities	Support
Data types	All numeric and logical data types of any dimension. The object writes logical data as `uint8`.
Multiple resolution levels	Yes
Process blocks in parallel using the `apply` function	Yes
Resume block processing using the `apply` function	Yes

Creation

Description

`adapter = images.blocked.H5Blocked` creates a `H5Blocks` object that reads and writes blocked image data as chunked H5 files, with one file for each block.

Properties

`GZIPLevel` — GZIP compression level

1 (default) | number in the range [0, 9]

GZIP compression level, specified as a number in the range [0, 9]. This value controls the level of GZIP (lossless) compression. 0 turns off compression. Higher values attempt to increase the level of compression and reduce the file size at the cost of higher runtimes.

Data Types: `double`

Examples

Save Image Data in Folder with One HDF5 File Per Block

Create a blocked image.

```
bim = blockedImage('tumor_091R.tif');
```

Write image data to files. Specify the `images.blocked.H5Blocks` adapter.

```
wa = images.blocked.H5Blocks();
wa.GZIPLevel = 5;
write(bim, "H5sFolder", "Adapter", wa);
```

Create a blocked image from the folder of images. The `blockedImage` object automatically picks the appropriate adapter.

```
bh5 = blockedImage("H5sFolder");
```

See Also

`blockedImage` | `H5`

Introduced in R2021a

InMemory

Read and write blocked image data as workspace variable

Description

The `InMemory` object is an adapter that reads and writes single-resolution blocked image data as a variable in the workspace. This object has the fastest read/write performance of the adapter objects that support blocked images.

The table lists the support that the `InMemory` object has for various `blockedImage` capabilities.

Capabilities	Support
Data types	All numeric, logical, categorical, and structure data types of any dimension.
Multiple resolution levels	No
Process blocks in parallel using the `apply` function	No
Resume block processing using the `apply` function	No

Creation

Description

`adapter = images.blocked.InMemory` creates an `InMemory` object that reads and writes single-resolution blocked image data to a variable in the workspace.

See Also

`blockedImage` | `GenericImage`

Introduced in R2021a

JPEGBlocks

Read and write blocks of blocked image data as JPEG files

Description

The `JPEGBlocks` object is an adapter that reads and writes 2-D blocked image data in the JPEG format.

When writing to disk, the object creates an individual JPEG file for each block and saves the image files in a folder. For multiresolution images, the object creates one subfolder for each resolution level. The object also creates and saves a MAT file with information about the blocked image, including the image size, block size, and data type.

The object supports lossy and lossless compression. By default, the object writes JPEG image files with lossy compression and a quality factor of `75`. To use lossy compression with a different quality factor, create the object and then change the quality factor using the `JPEGQuality` property. To use lossless compression, create the object and then specify the `CompressionMode` property as `"Lossless"`. For example, to specify a quality factor of `90`, use this code.

```
adapter = images.blocked.JPEGBlocks;
adapter.JPEGQuality = 90;
```

The table lists the support that the `JPEGBlocks` object has for various `blockedImage` capabilities.

Capabilities	Support
Data types	This object supports 2-D images only: • Binary images of size *m*-by-*n* with data type `logical` • Grayscale images of size *m*-by-*n* with data type `uint8` • Truecolor (RGB) images of size *m*-by-*n*-by-3 with data type `uint8`
Multiple resolution levels	Yes
Process blocks in parallel using the `apply` function	Yes
Resume block processing using the `apply` function	Yes

Creation

Description

`adapter = images.blocked.JPEGBlocks` creates a `JPEGBlocks` object that reads and writes blocked image data as JPEG files, with one JPEG file for each block.

Properties

`JPEGQuality` — JPEG quality factor

75 (default) | number in the range [0, 100]

JPEG quality factor, specified as a number in the range [0, 100]. Higher numbers specify better quality because there is less image degradation due to compression, but the resulting file size is larger.

`CompressionMode` — JPEG compression mode

`"Lossy"` (default) | `"Lossless"`

JPEG compression mode, specified as the string scalar or character vector `"Lossy"` or `"Lossless"`. If you specify `"Lossless"`, then the adapter ignores the `JPEGQuality` property.

Data Types: `string` | `char`

Examples

Save Image Data in Folder with One JPEG File Per Block

Create a blocked image.

```
bim = blockedImage('tumor_091R.tif');
```

Write image data to files. Specify the `images.blocked.JPEGBlocks` adapter. Choose to write in highest quality. JPG uses lossy compression, so space required is still significantly lower than other formats.

```
wa = images.blocked.JPEGBlocks();
wa.JPEGQuality = 100;
write(bim, "folderOfJPGs", "Adapter", wa);
```

Create a blocked image from the folder of images. The `blockedImage` object automatically picks the appropriate adapter.

```
bjpeg = blockedImage("folderOfJPGs");
disp(bjpeg.Adapter)
```

```
  JPEGBlocks with properties:

        JPEGQuality: 100
    CompressionMode: "Lossy"
        BlockFormat: "jpeg"
```

See Also

`blockedImage` | `PNGBlocks` | `GenericImageBlocks` | `GenericImage`

Introduced in R2021a

MATBlocks

Read and write blocks of blocked image data as MAT files

Description

The `MATBlocks` object is an adapter that reads and writes blocked image data as MAT files, with one MAT file for each block. The object can read and write numeric, binary, categorical, and structure data types.

The object saves the image files in a folder. For multiresolution images, the object creates one subfolder for each resolution level.

The table lists the support that the `MATBlocks` object has for various `blockedImage` capabilities.

Capabilities	Support
Data types	All numeric, logical, categorical, and structure data types of any dimension.
Multiple resolution levels	Yes
Process blocks in parallel using the `apply` function	Yes
Resume block processing using the `apply` function	Yes

Creation

Description

`adapter = images.blocked.MATBlocks` creates a `MATBlocks` object that reads and writes blocked image data as MAT files, with one MAT file for each block.

See Also

`blockedImage`

Introduced in R2021a

PNGBlocks

Read and write blocks of blocked image data as PNG files

Description

The `PNGBlocks` object is an adapter that writes 2-D blocked image data in the PNG format.

When writing to disk, the object creates an individual PNG file for each block and saves the image files in a folder. For multiresolution images, the object creates one subfolder for each resolution level. The object also creates and saves a MAT file with information about the blocked image, including the image size, block size, and data type.

The table lists the support that the `PNGBlocks` object has for various `blockedImage` capabilities.

Capabilities	Support
Data types	This object supports 2-D images only: • Grayscale images of size *m*-by-*n* with data type `uint8` or `uint16` • Truecolor (RGB) images of size *m*-by-*n*-by-3 with data type `uint8` or `uint16`
Multiple resolution levels	Yes
Process blocks in parallel using the `apply` function	Yes
Resume block processing using the `apply` function	Yes

Creation

Description

`adapter = images.blocked.PNGBlocks` creates a `PNGBlocks` object that reads and writes blocked image data as PNG files, with one PNG file for each block.

Examples

Save Image Data in Folder with One PNG File Per Block

Create a blocked image.

```
bim = blockedImage('tumor_091R.tif');
```

Write image data to files. Specify the `images.blocked.PNGBlocks` adapter.

```
wa = images.blocked.PNGBlocks();
write(bim, "dirOfPNGs", "Adapter", wa);
```

Create a blocked image from the folder of images. The `blockedImage` object automatically picks the appropriate adapter.

```
bpng = blockedImage("dirOfPNGs");
disp(bpng.Adapter)
```

```
  PNGBlocks with properties:

    BlockFormat: "png"
```

See Also

`blockedImage` | `JPEGBlocks` | `GenericImageBlocks` | `GenericImage`

Introduced in R2021a

TIFF

Read and write blocked image data as single TIFF file

Description

The `TIFF` object is an adapter that reads and writes 2-D blocked image data as a single TIFF file.

When writing to disk, the TIFF format requires block sizes to be a multiple of 16. If the blocked image has any additional metadata in the `UserData` property, then the `TIFF` object writes the data to a separate MAT-file with the same file name.

The object supports lossy and lossless compression. By default, the object uses Lempel-Ziv-Welch lossless compression. To use a different compression scheme, create the object and then change the compression scheme using the `Compression` property. You can also use this property to turn off compression. For example, to use JPEG-based lossy compression, use this code.

```
adapter = images.blocked.TIFF;
adapter.Compression = JPEG;
```

The table lists the support that the `TIFF` object has for various `blockedImage` capabilities.

Capabilities	Support
Data types	This object supports 2-D images only: • Binary images of size *m*-by-*n* with data type `logical` Grayscale images of size *m*-by-*n* with data type `uint8`, `int16`, `uint16`, `int32`, `uint32`, `single`, or `double` • Truecolor (RGB) images of size *m*-by-*n*-by-3 with data type `uint8`, `uint16`, `uint32`, `single`, or `double`
Multiple resolution levels	Yes
Process blocks in parallel using the `apply` function	No
Resume block processing using the `apply` function	No

Creation

Description

`adapter = images.blocked.TIFF` creates a `TIFF` object that reads and writes blocked image data as a single TIFF file.

Properties

`Compression` — TIFF compression scheme

`"LZW"` (default) | `"PackBits"` | `"Deflate"` | `"JPEG"` | `"None"`

TIFF compression scheme, specified as one of the following.

Compression Scheme	Description
`"LZW"`	Lempel-Ziv-Welch lossless compression
`"PackBits"`	PackBits lossless compression
`"Deflate"`	Adobe DEFLATE lossless compression
`"JPEG"`	JPEG-based lossy compression
`"None"`	No compression

Data Types: `string`

`Extension` — Preferred file extension

`"tiff"` (default) | character vector | string scalar

Preferred file extension, specified as a string scalar or character vector.

Data Types: `char` | `string`

Examples

Save Two Images as Single Multiresolution TIFF file

Create a blocked image.

```
bim = blockedImage('tumor_091R.tif');
```

Create two separate images.

```
bim.BlockSize = [512 512 3];
bo1 = apply(bim, @(bs)im2gray(bs.Data));
bo3 = apply(bim, @(bs)im2gray(bs.Data), "Level", 3);
```

Create a single multiresolution TIFF file from the two images. You specify additional resolution levels using the `"LevelImages"` parameter.

```
wa = images.blocked.TIFF(); % Specify the TIFF adapter
wa.Compression = Tiff.Compression.JPEG; % Specify compression in the adapter
write(bo1, "tumor_091RGray.tif", "LevelImages", bo3, "Adapter", wa);
```

See Also

`blockedImage` | `GenericImage` | `InMemory`

Introduced in R2021a

blockedImage

Image made from discrete blocks

Description

A `blockedImage` object is an image made from discrete blocks. Use blocked images when an image or volume is too large to fit into memory. With a blocked image, you can perform processing without running out of memory.

Creation

Syntax

```
bim = blockedImage(source)
bims = blockedImage(sources)
___  = blockedImage( ___ ,Name,Value)

wbim = blockedImage(destination,size,blockSize,initialValue,"Mode","w")
```

Description

Create Read-only blockedImage Objects

`bim = blockedImage(source)` creates a `blockedImage` object from the specified source. The source can be an in-memory array or the name of a file or folder with image data.

`bims = blockedImage(sources)` creates an array of `blockedImage` objects from multiple sources. The source can be a collection of files or folders with image data. The length of `bims` is equal to the number of sources in `sources`.

`___  = blockedImage( ___ ,Name,Value)` creates a `blockedImage` object, using one or more name-value arguments to set object properties on page 1-251.

Create Writable blockedImage Object

`wbim = blockedImage(destination,size,blockSize,initialValue,"Mode","w")` creates a writeable `blockedImage` object at one or multiple resolution levels. `destination` specifies the location of the writable data. `size` indicates the image size at each resolution level. `initialValue` indicates the initial value for each array element.

Input Arguments

`source` — Source of image data

numeric array | categorical array | structure array | character vector | string scalar

Source of image data, specified as a numeric array, categorical array, or structure array, or a character vector or string scalar specifying the name of a file or folder.

Blocked images supports these file formats:

- Single TIFF file. If the file contains multiple Image File Directories (IFDs), the `blockedImage` object treats the IFDs as multiple resolution levels.
- Any image file that can be read by `imread`.
- Any source created by the adapters included with the toolbox, listed in `Adapter`.

`sources` — Sources of image data

cell array of character vectors | string array | `FileSet` object

Sources of image data, specified as an cell array of character vectors, a string array, or a `FileSet` object.

`destination` — Location to place writeable data

`[]` | character vector | string scalar

Location to place writable data, specified as a character vector or string scalar.

Destination Type	Image Format
Folder name (without a file extension)	The `blockedImage` object creates the folder and stores blocks of data as files within the folder. • For numeric image data, `blockedImage` stores each block as a binary file using the `BINBlocks` adapter. • For categorical and structure image data, `blockedImage` stores each block as a MAT file in the folder using the `MATBlocks` adapter.
File name with TIF or TIFF file extension	The `blockedImage` object stores data as a single TIFF image using the `TIFF` adapter. The `initialValue` must be numeric or logical with data type `uint8`, `int8`, `uint16`, `int16`,`uint32`, `int32`, `single`, `double` or `logical`.
File name with H5 file extension	The `blockedImage` object stores data as a single HDF5 image using the `H5` adapter. The `initialValue` must be numeric with data type `uint8`, `int8`, `uint16`, `int16`, `uint32`, `int32`, `single`, or `double`.
`[]`	The `blockedImage` object stores data as a variable in memory using the `InMemory` adapter.

To specify a custom adapter for other output formats, use the `Adapter` property.

`size` — Image size at each resolution level

L-by-*N* matrix

Image size at each resolution level, specified as an *L*-by-*N* matrix of positive integers, where *L* is the number of resolution levels and *N* is the number of dimensions of the image. The `blockedImage` object always sorts `size` in descending order by number of pixels, irrespective of how `Source` stores the levels.

`blockSize` — Size of blocks

L-by-*N* matrix

Size of blocks, specified as an *L*-by-*N* matrix of positive integers, where *L* is the number of resolution levels and *N* is the number of dimensions.

If the image has multiple resolution levels, then you can specify `blockSize` as a 1-by-*N* vector to use the same block size for all resolution levels.

Example: `[64 128]` specifies a block size of 64-by-128 pixels for a single resolution image

Example: `[128 128; 64 64; 32 32]` specifies three different block sizes for three resolution levels

`initialValue` — Default element value for unloaded blocks

numeric scalar | categorical scalar | struct scalar

Default pixel value for unloaded blocks, specified as one of these values. The blocked image uses this value to fill blocks which do not have data in the underlying source.

- Numeric scalar. The data type of `initialValue` specifies the value of the `ClassUnderlying` property. The default value is `0`.
- Categorical scalar. The default value is `<undefined>`.
- Structure with the same field names as the data. The default value is `<undefined>`.

Properties

`Adapter` — Read and write interface for blocked image object

`InMemory` object | `MATBlocks` object | `PNGBlocks` object | `TIFF` object | ...

Read and write interface for the blocked image object, specified as one of these adapter objects.

Adapter	Description
`BINBlocks`	Store each block as a binary file in a folder
`GenericImage`	Store blocks in a single image
`GenericImageBlocks`	Store each block as an image file in a folder
`H5`	Store blocks in a single HDF5 image
`H5Blocks`	Store each block as an HDF5 file in a folder
`InMemory`	Store blocks in a variable in main memory
`JPEGBlocks`	Store each block as a JPEG file in a folder
`MATBlocks`	Store each block as a MAT file in a folder
`PNGBlocks`	Store each block as a PNG file in a folder
`TIFF`	Store blocks in a single TIFF file

You can also create your own adapter using the `images.blocked.Adapter` class.

`AlternateFileSystemRoots` — Alternate file system path

string array

Alternate file system path for the files specified in source, specified as a string array containing one or more rows. Each row specifies a set of equivalent root paths.

Example: `["Z:\datasets", "/mynetwork/datasets"]`

Data Types: `char` | `string` | `cell`

`BlockSize` — Size of blocks
L-by-*N* matrix

Size of blocks, specified as an *L*-by-*N* matrix of positive integers, where *L* is the number of resolution levels and *N* is the number of dimensions. `BlockSize` serves as the default size of data that is loaded into main memory at any time for use. It is the smallest unit of data that can be manipulated with the `blockedImage` interface. If you specify `BlockSize` with less than *N* dimensions, `blockedImage` pads the image with elements from the `Size` property.

You cannot specify this property when you specify `Mode` as `'w'`.

Data Types: `double`

`ClassUnderlying` — Pixel data type
cell array of character vectors | string vector | structure | categorical array

This property is read-only.

Pixel data type, specified as a cell array of character vectors, string array, structure array, or categorical array, with *L* elements. *L* is number of resolution levels. Each element in the array is the data type of a pixel from the corresponding resolution level. Values are: `"logical"`, `"int8"`, `"uint8"`, `"int16"`, `"uint16"`, `"int32"`, `"uint32"`, `"single"`, and `"double"`.

Data Types: `char` | `string`

`InitialValue` — Default element value for unloaded blocks
numeric scalar | `categorical` scalar | `struct`

This property is read-only.

Default element value for unloaded blocks, specified as a numeric scalar of the type specified by `ClassUnderlying`, a `categorical` value for categorical images, or a `struct`. The `blockedImage` object uses this value to fill blocks which do not have data in the underlying source. The default value varies by data type: `0` for numeric types, `<undefined>` for categorical, and a scalar `struct`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `char` | `categorical` | `struct`

`IOBlockSize` — I/O block size of image source
numeric matrix

This property is read-only.

I/O block size of image source, specified as an *L*-by-*N* matrix of positive integers, where *L* is the number of resolution levels and *N* is the number of dimensions. `IOBlockSize` is the size of the underlying I/O block size the adapter uses to read from the image source. This represents the smallest unit of data that can be written or read. This read-only property reflects the format of the underlying image source.

Note You can set `BlockSize` to any value. The `blockedImage` object does the appropriate reading, cropping, stitching, and caching of the source I/O blocks to ensure efficient I/O.

Data Types: `double`

`Mode` — Current read or write mode

`'r'` (default) | `'w'`

Current read or write mode of the object, specified as `'r'` for read mode and `'w'` for write mode.

You can only set `Mode` to `'w'` when you create the object. You can change the value of `Mode` from `'w'` to `'r'`, at which point no further writes are possible. You cannot change `Mode` from `'r'` to `'w'`.

When opening a `blockedImage` in write mode, you must also specify values for the `ImageSize`, `BlockSize`, and `InitialValue` properties.

Data Types: `char`

`NumDimensions` — Number of dimensions in image

positive integer

This property is read-only.

Number of dimensions in the image, specified as a positive integer. For multiresolution level images that have varying number of dimensions, `NumDimensions` is the maximum taken across all levels. The blocked image extends other levels with singleton dimensions, if needed.

Data Types: `double`

`NumLevels` — Number of image resolution levels

positive integer

This property is read-only.

Number of image resolution levels, specified as a positive integer.

Data Types: `double`

`Size` — Image size at each level

L-by-*N* matrix

This property is read-only.

Image size at each level, specified as an *L*-by-*N* matrix of positive integers, where *L* is the number of resolution levels and *N* is the number of dimensions of the image. The `blockedImage` object always sorts `Size` in descending order by number of pixels, irrespective of how `Source` stores the levels.

Data Types: `double`

`SizeInBlocks` — Size expressed as number of blocks

L-by-*N* matrix

This property is read-only.

Size expressed as the number of blocks, specified as an *L*-by-*N* matrix of positive integers, where *L* is the number of resolution levels and *N* is the number of dimensions. This property is dependent on the `BlockSize` property. The value includes partial blocks.

Data Types: `double`

`Source` — Source of image data

`string` scalar | numeric array | `categorical` array | `struct` array

This property is read-only.

Source of image data, specified as an in-memory numeric, `categorical`, or `struct` array, or as a `string` scalar or `char` vector specifying a file or folder name.

Data Types: `string`

`WorldEnd` — World coordinates of ending edge of image

numeric matrix

World coordinates of ending edge of the image, specified as an *L*-by-*N* numeric matrix, where *L* is the number of resolution levels and *N* is the number of dimensions. By default, the value is `Size + 0.5` for each dimension and level, resulting in pixels that are one-unit wide. World coordinates of pixel centers coincide with pixel subscript locations for the first level.

Data Types: `double`

`WorldStart` — World coordinates of starting edge of image

numeric matrix

World coordinates of the starting edge of the image, specified as a *L*-by-*N* numeric matrix, where *L* is the number of levels and *N* is the number of dimensions. By default, the starting-edge value is `0.5` in each dimension and level.

Data Types: `double`

`UserData` — User data associated with the image

`struct`

User data associated with the image, specified as a `struct`. This field can be empty. You can update the value at any time. To make this value persist with the source, write the `blockedImage` to a file using the `write` function, or specify the data as parameter when you create the object.

Data Types: `struct`

Object Functions

apply	Process blocks of blocked image
crop	Create cropped version of blocked image
blocksub2sub	Convert block subscripts to pixel subscripts
gather	Collect blocks into current workspace
getBlock	Read specific block of blocked image
getRegion	Read arbitrary region of blocked image
setBlock	Put data in specific block of blocked image
sub2blocksub	Convert pixel subscripts to block subscripts
sub2world	Convert pixel subscripts to block subscripts
world2sub	Convert world coordinates to pixel subscripts
write	Write blocked image data to new destination

Examples

Create and Visualize Blocked Image

Create a blocked image from a sample image included with the toolbox.

```
bim = blockedImage('tumor_091R.tif');
```

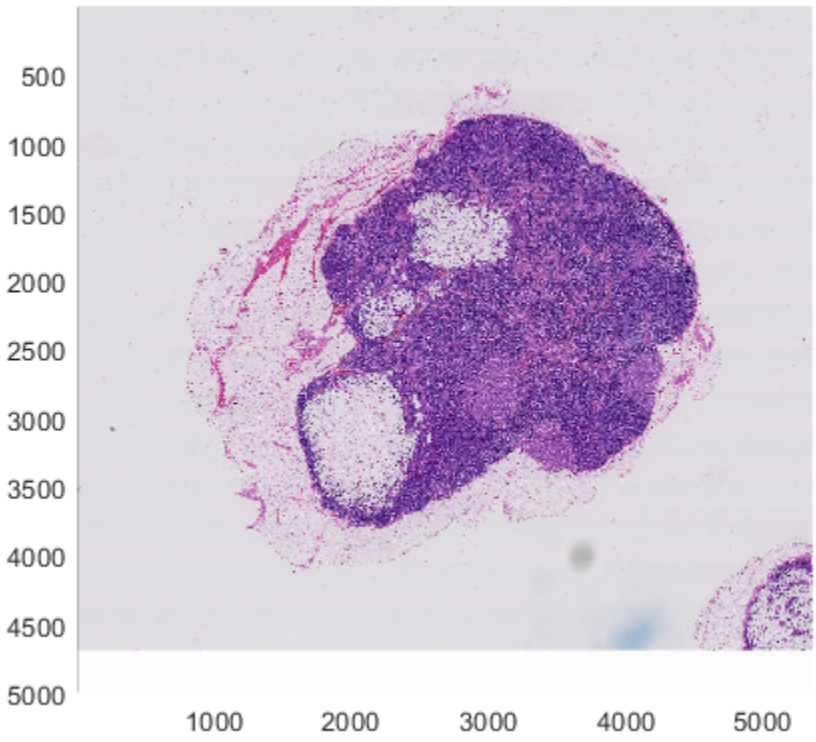
Display details of the blocked image at the command line.

```
disp(bim)
```

```
  blockedImage with properties:

   Read only properties
              Source: "B:\matlab\toolbox\images\imdata\tumor_091R.tif"
             Adapter: [1x1 images.blocked.TIFF]
                Size: [3x3 double]
       SizeInBlocks: [3x3 double]
    ClassUnderlying: [3x1 string]

   Settable properties
           BlockSize: [3x3 double]
```

View the blocked image in a figure window.

```
bigimageshow(bim)
```



Create Blocked Image from Workspace Variable

Read data into the workspace. For this example, read a sample volume that is included with the toolbox.

```
dmri = tiffreadVolume('mri.tif');
```

Create a blocked image from the volume.

```
bim = blockedImage(dmri);
```

Display details about the blocked image at the command line.

```
disp(bim)
```

```
  blockedImage with properties:

   Read only properties
             Source: [128x128x27 uint8]
            Adapter: [1x1 images.blocked.InMemory]
               Size: [128 128 27]
      SizeInBlocks: [1 1 1]
    ClassUnderlying: "uint8"

   Settable properties
          BlockSize: [128 128 27]
```

Create Array of Blocked Images

Create a file set of the images in the toolbox folder of sample images.

```
fs = matlab.io.datastore.FileSet( ...
         fullfile(matlabroot,'toolbox','images','imdata'), ...
         "FileExtensions",{'.jpg','.png'});
```

Create an array of blocked images from the images in the file set.

```
bims = blockedImage(fs);
```

Display details of the array of blocked images.

```
disp(bims)
```

```
  1x74 blockedImage array with properties:

   Read only properties
             Source: 'Various'
            Adapter: [1x1 images.blocked.GenericImage]
    ClassUnderlying: 'Various'

   Settable properties
    No properties.
```

Write Data to Blocked Image and Read It Back

Create a blocked image to which you can write data. You specify the format of the blocked image in the `destination` parameter. To write to memory, specify an empty matrix. You must also specify the size of the image and the size of the blocks into which you want the image chunked. The initial value parameter depends on the format you specified in destination. To create a writable blocked image, specify the `'Mode'` parameter with the value `'w'` for write mode.

```
destination = [];
imgsize = [5 7];
blocksize = [2 2];
initval = uint8(0);
bim = blockedImage(destination,imgsize,blocksize,initval, "Mode", 'w');
```

Write data to the specified blocks in the blocked image by using the `setBlock` object function. The `blocksubs` parameter specifies the coordinates of the block to which you want to write data. The `blockdata` parameter specifies the data to write to the specified block. The size of blockdata must match the block size.

```
blocksubs = [1 1];
blockdata = ones(2,2,"uint8");
setBlock(bim, blocksubs, blockdata)
```

Close the image for writing.

Switch the blocked image to read mode by setting the 'Mode' parameter to 'r' for read.

```
bim.Mode = 'r'
```

```
bim =

  blockedImage with properties:

   Read only properties
             Source: [5x7 uint8]
            Adapter: [1x1 images.blocked.InMemory]
               Size: [5 7]
      SizeInBlocks: [3 4]
    ClassUnderlying: "uint8"

   Settable properties
          BlockSize: [2 2]
```

Create the full image by using the `gather` function to collect all the individual blocks.

```
fullImage = gather(bim);
```

Display details of the blocked image at the command line.

```
disp(fullImage)
```

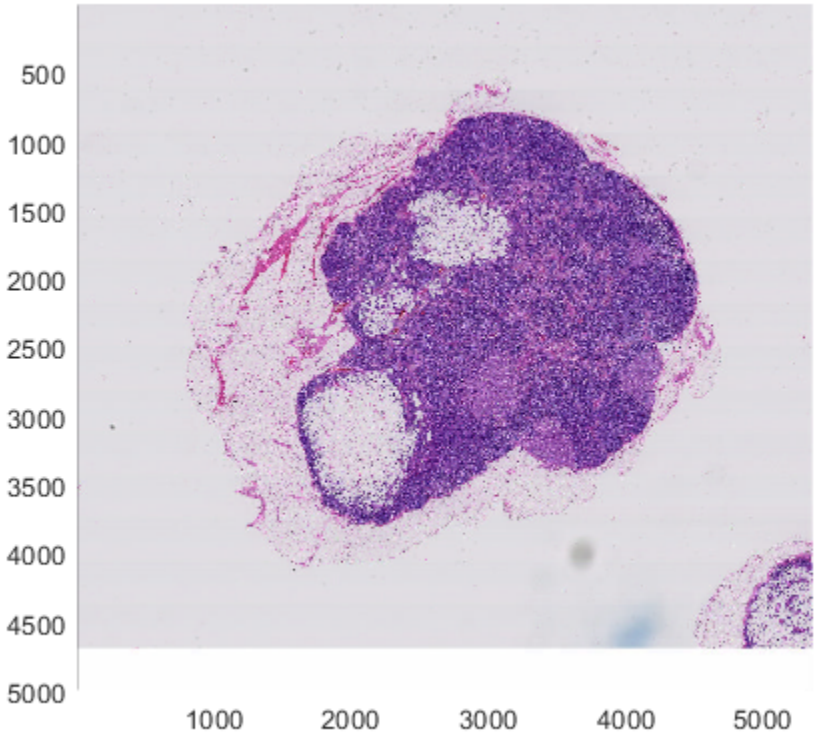
```
   1   1   0   0   0   0   0
   1   1   0   0   0   0   0
   0   0   0   0   0   0   0
   0   0   0   0   0   0   0
```

```
     0     0     0     0     0     0     0
```

Create Blocked Image Specifying Custom World Coordinates

Create a blocked image from a sample image included with the toolbox.

```
bim = blockedImage('tumor_091R.tif');
bigimageshow(bim)
```



Specify the distance between pixel centers at the finest level. This information obtained from the raw data available at https://camelyon17.grand-challenge.org/Data/.

```
pext = 0.000226316; % (in millimeters)
```

Assume the top-left edge of the first pixel starts at (0,0).

```
worldStart = zeros(bim.NumLevels, bim.NumDimensions);
```

Calculate the bottom right edge of the last pixel of the finest resolution level, using only the spatial dimensions. With the distance between each pixel center known, multiply the distance by the number of pixels.

```
worldEnd = bim.Size(1,:)*pext;
```

All resolution levels span the same world coordinates.

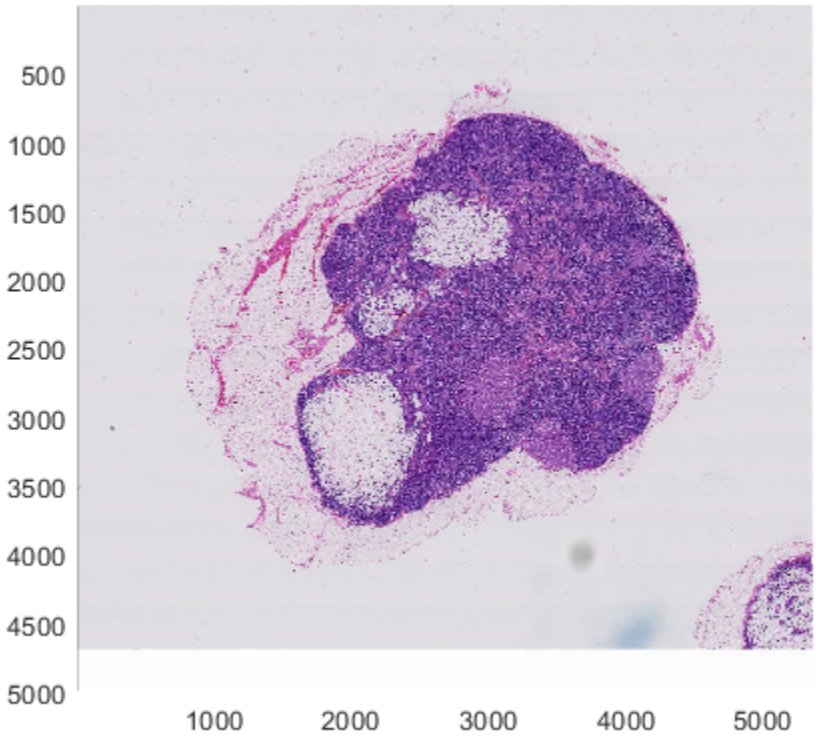
```
worldEnd = repmat(worldEnd,[bim.NumLevels,1]);
```

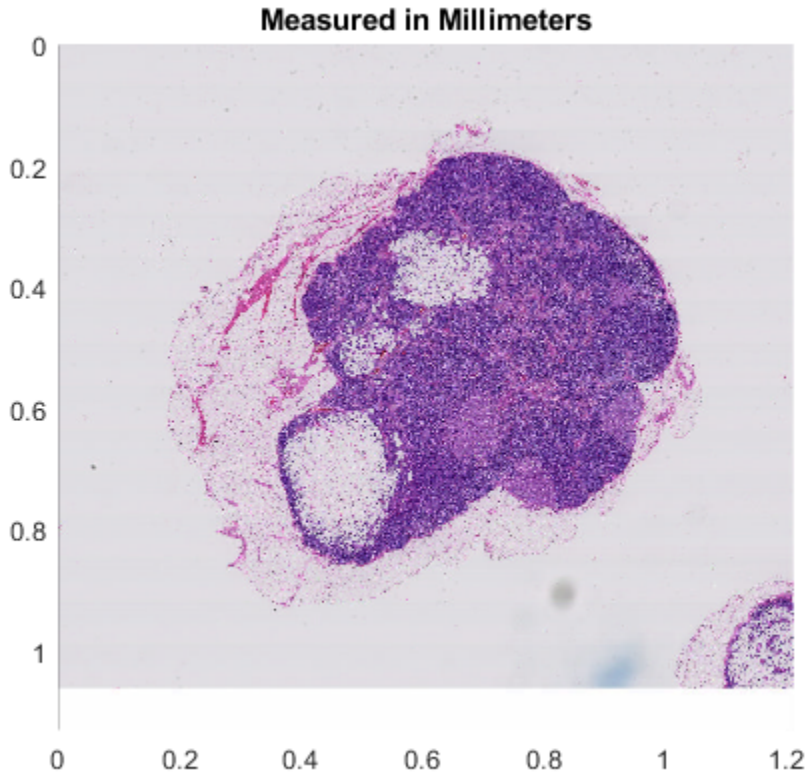
The third dimension holds the color channels, each with a pixel extent of 1. Update the world coordinates of the edges of the pixels to center them on integer values (in this case, 1, 2, and 3).

```
worldStart(:,3) = 0.5;
worldEnd(:,3) = 3.5;
```

View the image with updated coordinates.

```
bim = blockedImage('tumor_091R.tif', ...
    'WorldStart',worldStart,'WorldEnd',worldEnd);
figure
h = bigimageshow(bim);
title('Measured in Millimeters')
```





See Also

`blockedImageDatastore` | `bigimageshow`

Introduced in R2021a

apply

Process blocks of blocked image

Syntax

```
bres = apply(bim,fcn)
[bres1,bres2,...] = apply(bim,fcn)
[bres1s,bres2s,...] = apply(bims,fcn)
[___] = apply(___,Name,Value)
```

Description

`bres = apply(bim,fcn)` processes the entire blocked image `bim` by applying the function handle `fcn` to each block. Returns `bres`, a new blocked image containing the processed data.

`[bres1,bres2,...] = apply(bim,fcn)` returns multiple output arguments. The specified function handle, `fcn`, must point to a user function which returns the same number of arguments.

`[bres1s,bres2s,...] = apply(bims,fcn)` processes the array of blocked images `bims` by applying the function handle `fcn` to each block of each blocked image. Returns an array of blocked images containing the processed data.

`[___] = apply(___,Name,Value)` modifies aspects of the block processing using name-value arguments.

Examples

Enhance Image Details to Better Visualize Region Boundaries

Create blocked image.

```
bim = blockedImage("tumor_091R.tif");
```

Create a smoothing filter and apply it to the blocks in the blocked image.

```
smoothing = 2000;
```

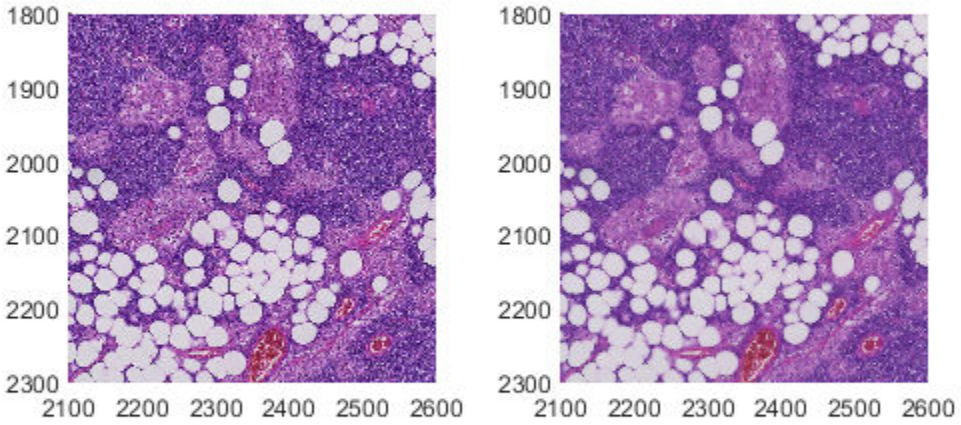
`imguidedfilter` operates on a default neighborhood of 5 pixels. Add a border to the input to read additional data. This border pixels automatically get trimmed from the output since its the same size as the input.

```
benh= apply(bim,...
    @(bs)imguidedfilter(bs.Data,bs.Data,"DegreeOfSmoothing", smoothing),...
    "BorderSize", [5 5]);
```

Display the original image and the enhanced image.

```
ha1 = subplot(1,2,1);
bigimageshow(bim,"ResolutionLevel",1);
ha2 = subplot(1,2,2);
```

```
bigimageshow(benh);
linkaxes([ha1, ha2]);
xlim([2100, 2600])
ylim([1800 2300])
```



Improve Efficiency By Using Mask to Limit Processed Region

Create a blocked image.

```
bim = blockedImage("tumor_091R.tif");
```

Create a mask at the coarsest level and display it.

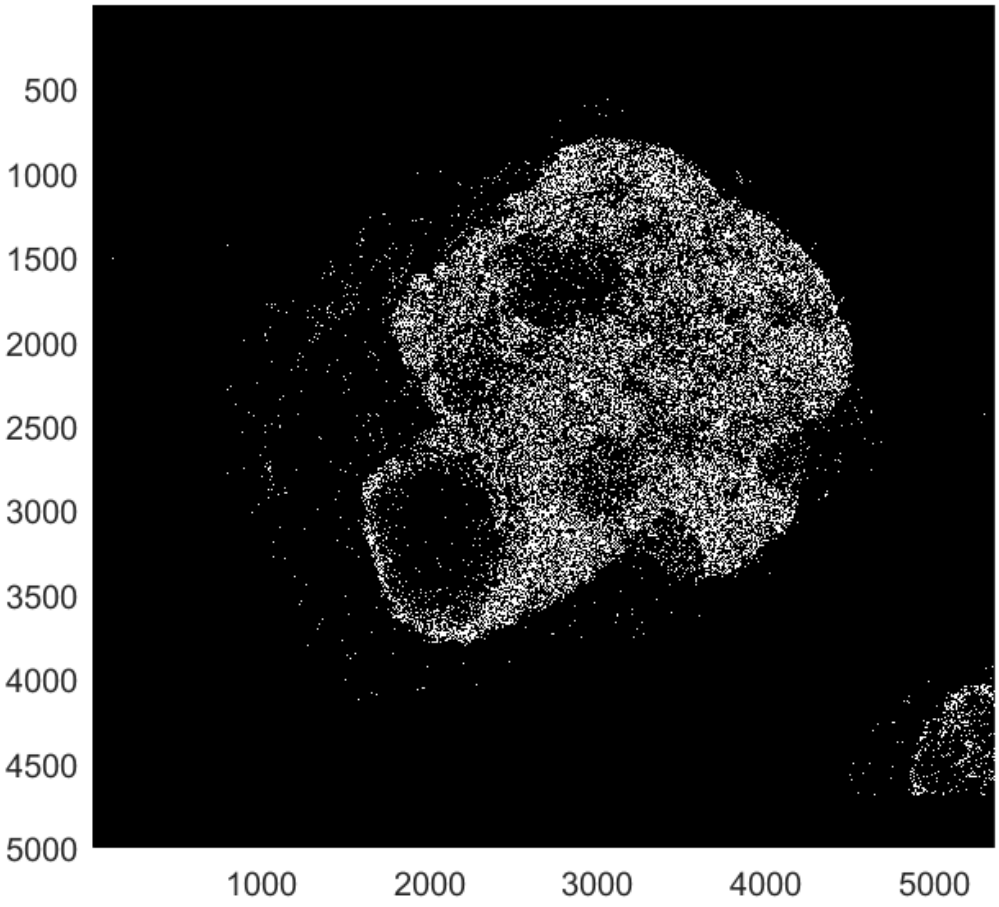
```
bmask = apply(bim, @(bs)rgb2gray(bs.Data)<80, "Level",3);
figure
bigimageshow(bmask)
```

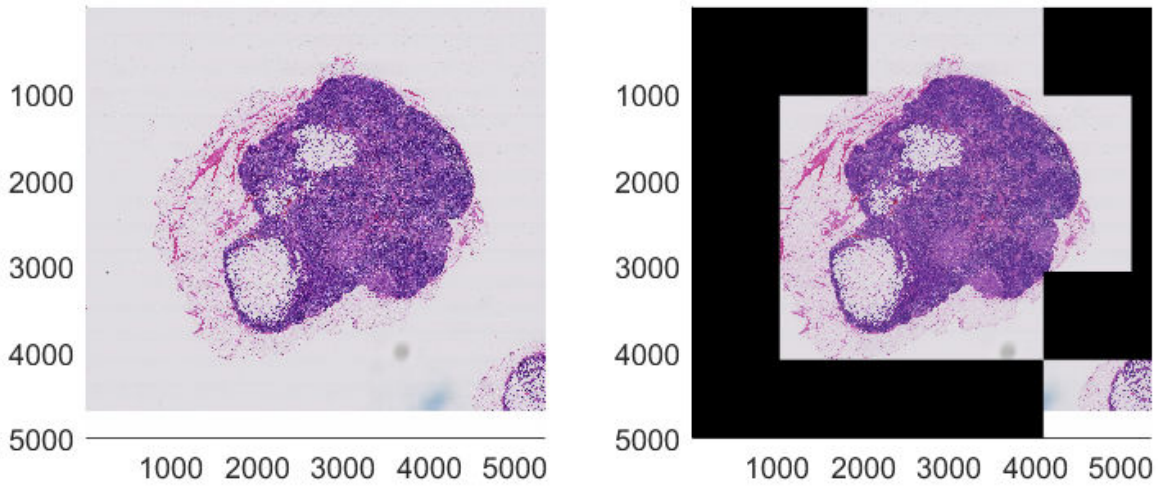
Use the mask to limit regions processed by the call to the `apply` object function.

```
bls = selectBlockLocations(bim, "Mask", bmask,"InclusionThreshold", 0.005);
benh = apply(bim, @(bs)imguidedfilter(bs.Data,bs.Data,"DegreeOfSmoothing", 2000),...
    "BorderSize", [5 5],...
    "BlockLocationSet", bls);
```

Display the original image and the enhanced image.

```
figure
ha1 = subplot(1,2,1);
bigimageshow(bim,"ResolutionLevel",1);
ha2 = subplot(1,2,2);
bigimageshow(benh);
linkaxes([ha1, ha2]);
```





Process Array of Blocked Images

Create a file set of all the JPEG images in the toolbox sample images folder.

```
fs = matlab.io.datastore.FileSet( ...
    fullfile(matlabroot,"toolbox","images","imdata"), ...
    "FileExtensions",".jpg");
```

Create an array of blocked images from the file set.

```
bims = blockedImage(fs);
```

Create an adapter that saves a blocked image to disk as a single image file.

```
outputFolder = tempname;
outputAdapter = images.blocked.GenericImage;
outputAdapter.Extension = "jpg";
```

Convert the images to binary images on disk.

```
bos = apply(bims, @(b)imbinarize(im2gray(b.Data)), ...
    "OutputLocation",outputFolder,"Adapter",outputAdapter);
```

View the contents of the output folder using the Image Browser app by running this command: `imageBrowser(outputFolder)`

Input Arguments

`bim` — Blocked image

`blockedImage` object

Blocked image, specified as a `blockedImage` object.

`bims` — Blocked images

array of `blockedImage` objects

Blocked images, specified as an array of `blockedImage` objects.

`fcn` — Processing function

function handle

Processing function, specified as a function handle. For more information, see “Create Function Handle”.The processing function `fcn` must accept a `bstruct` as input. To pass additional arguments, specify `fcn` as an anonymous function. For more information, see “Anonymous Functions”.

`bstruct` is a `struct` with these fields:

Field	Description
`Data`	Block of data from `bim`
`Start`	Subscripts of the first element in the block. If `BorderSize` is specified, this subscript can be out-of-bounds for edge blocks.
`End`	Subscripts of the last element in the block. If `BorderSize` is specified, this subscript can be out-of-bounds for edge blocks.
`Blocksub`	Block subscripts of the current block
`BorderSize`	Value of the `BorderSize` parameter.
`BlockSize`	Value of the `BlockSize` parameter. Note: `size(data)` can be less than this value for border blocks when `PadPartialValue` is false.
`BatchSize`	Value of the `BatchSize` parameter
`ImageNumber`	Index into `bim` array for the current image.
`Level`	The resolution level from which the data is being read.

The function `fcn` typically returns the results for one block. The results can be numeric, a `struct`, or `categorical`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"Level",3`

`Adapter` — Adapter used for writing blocked image data

adapter object

Adapter used for writing blocked image data, specified as an adapter object. To specify different adapters for different outputs, use a cell array. Scalar values are expanded.

This table lists the adapters included with the toolbox.

Adapter	Description
`BINBlocks`	Store each block as a binary file in a folder
`GenericImage`	Store blocks in a single image
`GenericImageBlocks`	Store each block as an image file in a folder
`H5`	Store blocks in a single HDF5 image
`H5Blocks`	Store each block as an HDF5 file in a folder
`InMemory`	Store blocks in a variable in main memory
`JPEGBlocks`	Store each block as a JPEG file in a folder
`MATBlocks`	Store each block as a MAT file in a folder
`PNGBlocks`	Store each block as a PNG file in a folder
`TIFF`	Store blocks in a single TIFF file

You can also specify a custom adapter that performs custom writing operations. For more information, see `images.blocked.Adapter`.

`BatchSize` — Number of blocks supplied to `fcn`

1 (default) | numeric scalar

Number of blocks supplied to the processing function `fcn` in each batch, specified as a numeric scalar. The `BatchSize` is the last dimension of input to `fcn`. All outputs of `fcn` must have the last dimension be the same as `BatchSize`. `BatchSize` greater than 1 is useful to optimally load GPUs when applying deep learning inference calls. When `BatchSize` is greater than 1, `PadPartialBlocks` must be `true`.

`BlockLocationSet` — Set of blocks to process

all non-overlapping `BlockSize`-sized blocks (default) | `blockLocationSet` object

Set of blocks to process, specified as a `blockLocationSet` object. The `ImageNumber` property indexes into `bims` array. Specifying the blocks to process can improve efficiency by limiting the number of blocks processed. For example, use `selectBlockLocations` with a mask to limit applying the processing function to certain regions. Blocks contained must be on a regular grid.

`BlockSize` — Block size
value of `BlockSize` property of blocked image (default) | integer-valued vector

Block size, specified as an integer-valued vector of length equal to the `NumDimensions` property of `bim`. Specifies the size of a block supplied as the input to `fcn`. If `BlockSize` contains fewer elements, the `apply` object function pads the missing dimensions with elements from the `Size` property.

`BorderSize` — Border size
`0` (default) | integer-valued vector

Border size, specified as an integer-valued vector of length equal to the `NumDimensions` property of `bim`. Specifies additional data from neighboring region to be included in a block. For edge blocks, the `apply` object function uses `'PadMethod'`. If `BorderSize` contains fewer elements, the `apply` object function pads the border with 0s.

`DisplayWaitbar` — Display wait bar
`true` (default) | `false`

Display wait bar, specified as a logical scalar. When set to `true`,the `apply` object function displays a wait bar for long-running operations. If you cancel the wait bar, the `apply` object function returns partial output, if available.

Data Types: `logical`

`ExtraImages` — Additional inputs to `fcn`
array of `blockedImage` objects

Additional inputs to `fcn`, specified as an array of `blockedImage` objects. Blocks from this array are provided to `fcn` as additional inputs after `bstruct`: `__ = fcn(bstruct, extrablock1,...)`. The apply object function extracts these blocks from the same world region as the main block from `bim`, represented in `bstruct`.

`ExtraLevels` — Resolution level
vector of integers

Resolution level, specified as a vector of integers of the same length as `'ExtraImages'`. Each value specifies the resolution level to use from the corresponding `blockedImage` object in `ExtraImages`.

`Level` — Resolution level to use
`1` (default) | integer scalar

Resolution level to use, specified as an integer scalar. For a multiresolution `blockedImage` object, this value determines the resolution level to use to obtain blocks for processing.

`OutputLocation` — Location of output folder
`char` vector | `string` scalar

Location of output folder, specified as a `string` scalar or `char` vector. If there is a single output, the `apply` object function writes it directly to this location. For multiple outputs, the `apply` object function creates subfolders of the format `output<N>/` for the *N*'th output. If the input is an array, the `apply` object function derives the output name from the `Source` property of the corresponding element. If the input is in-memory, the `apply` object function uses a numeric index. When the *UseParallel* property is `true`, `OutputLocation` should be a valid path on the client session. Use the

AlternateFileSystemRoots property of the input to specify the required mapping for worker sessions. All outputs inherit this value.

PadMethod — Pad method
`'replicate'` (default) | numeric scalar | `string` scalar | `char` vector

Pad method of incomplete edge blocks, specified as one of these values. The method specifies how to obtain padding pixels to honor `'BorderSize'` or the `'PadPartialBlocks'` parameters.

Value	Meaning
numeric scalar	Pad numeric array with elements of a constant value of type specified by the `ClassUnderlying` property of the blocked image.
`'replicate'`	Pad by repeating border elements of array.

PadPartialBlocks — Pad partial blocks
`false` (default) | `true`

Pad partial blocks, specified as logical scalar. Specifies if partial blocks that may exist on the edges need to be padded out to the specified block size. The apply object function uses the method specified in `PadMethod` to perform the padding operation.

- When `false`, the processing function `fcn` operates on partial blocks without padding and can return blocks smaller than `BlockSize`.
- When `true`, the `apply` function pads partial blocks using the specified `PadMethod`. The processing function `fcn` operates on and returns full-sized blocks.

When `BatchSize` is greater than 1, set `PadPartialBlocks` to `true`

Data Types: `logical`

Resume — Continue processing from where previous run stopped
`false` (default) | `true`

Continue processing from where the previous run stopped, specified as a logical scalar. If `true`, and the specified `'OutputLocation'` has content from a previous run, the current run will continue processing from where the previous run stopped. This support depends on the output adapter used. If `false`, the `apply` object function deletes the previous content.

UseParallel — Use parallel processing
`false` (default) | `true`

Use parallel processing, specified as a logical scalar. Determines if a new or existing parallel pool should be used. If no parallel pool is active, a new pool is opened based on the default parallel settings. All adapters specified by the `Adapter` property must support parallel processing. You must specify a valid `OutputLocation`. This argument requires Parallel Computing Toolbox.

Data Types: `logical`

Output Arguments

bres — New blocked image
`blockedImage` object

New blocked image, returned as a `blockedImage` object.

Tips

- The `apply` object function determines the output size by processing the first block. If processing the first block yields an output block of the same size as the input, then the final output size is set to match the input. Otherwise, the last block is processed to determine the final output size. The first block must not be a partial block.
- The `apply` object function sets the `InitialValue` property of the output based on the type of the output:
  - Numeric or logical outputs – `InitialValue` is set to `0`.
  - Categorical outputs – `InitialValue` is set to the `<undefined>` value of the corresponding type.
  - `struct` outputs – `InitialValue` is derived from the first block's output. All fields are set to empty.

See Also

`blockedImage` | `selectBlockLocations`

Introduced in R2021a

blocksub2sub

Convert block subscripts to pixel subscripts

Syntax

```
[pStart,pEnd] = blocksub2sub(bim,blocksub)
[pStart,pEnd] = blocksub2sub(bim,blocksub,'Level',L)
```

Description

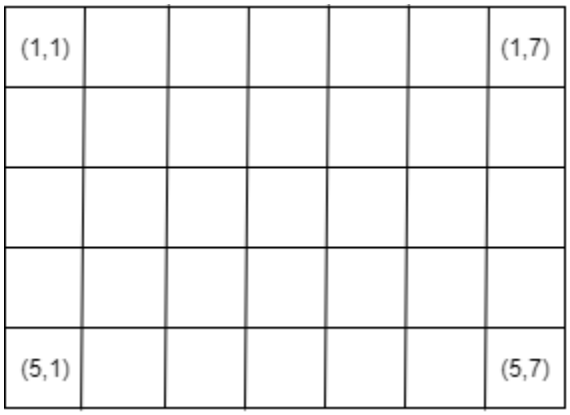
`[pStart,pEnd] = blocksub2sub(bim,blocksub)` converts a block subscript to pixel subscripts. `pStart` is the pixel subscripts of the first pixel in the block. `pEnd` is the pixel subscripts of the last pixel in the block. Note: For partial blocks `pEnd` represents the last valid pixel subscript in the image.

`[pStart,pEnd] = blocksub2sub(bim,blocksub,'Level',L)` additionally specifies the resolution level to use in a multiresolution image. `Level` defaults to `1`.

Examples

Convert Block Subscripts to Pixel Subscripts

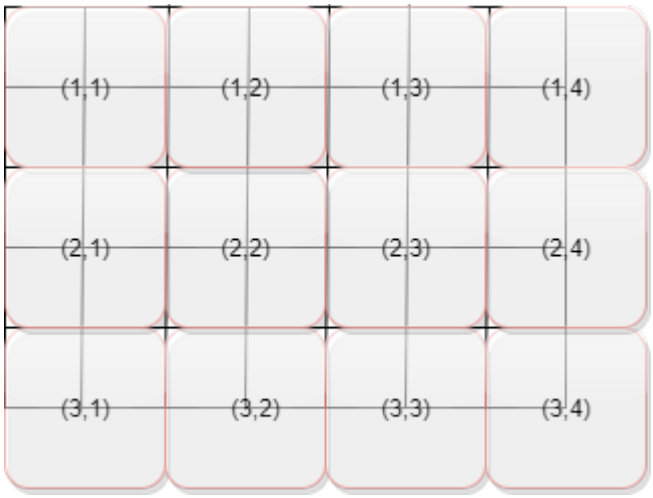
Create a small sample image as a 5-by-7 matrix of zeros. Here is an illustration of the small sample image, with the pixel coordinates of the four corners provided.



Create a blocked image from the sample image, specifying a 2-by-2 block size. To create this blocked image, use `blockedImage` in write mode.

```
bim = blockedImage([],[5 7],[2 2],uint8(0),"Mode",'w');
```

Here is an illustration of the blocked image overlaid on the original image. It is divided into 2-by-2 blocks. In the diagram, each block contains its block coordinates.

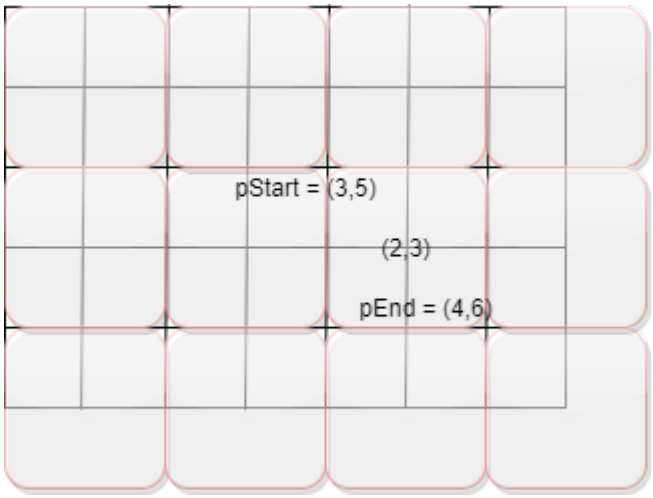


Convert block subscripts into pixel subscripts by using the `blocksub2sub` function. By default, if the image is a multiresolution image, `blocksub2sub` uses pixel coordinates from coarsest level, although you can specify any level. Since the sample image has only one resolution level, `blocksub2sub` converts level 1.

```
[pstart,pend] = blocksub2sub(bim,[2 3])
```

```
pstart = 1×2

     3     5


pend = 1×2

     4     6
```

This illustration show the block coordinate `[2 3]` converted to pixel coordinates.



Input Arguments

`bim` — Blocked image

`blockedImage` object

Blocked image, specified as a `blockedImage` object.

`blocksub` — Block subscripts

numeric array

Block subscripts, specified as a numeric array of positive integers.

Example: `[2 3]`

Output Arguments

`pStart` — Subscript of first pixel in block

numeric array

Subscript of first pixel in the specified block, returned as a numeric array.

`pEnd` — Subscript of last pixel in block

numeric array

Subscript of last pixel in the specified block, returned as a numeric array.

See Also

`blockedImage`

Introduced in R2021a

crop

Create cropped version of blocked image

Syntax

```
cbim = crop(bim,cstart,cend)
```

Description

`cbim = crop(bim,cstart,cend)` crops the blocked image `bim` to the crop window specified by the starting and ending pixel subscripts `cstart` and `cend`, inclusive. Returns `cbim`, a `blockedImage` which references the same `Source` as `bim`, but represents image data in the crop window, across all levels.

Examples

Crop Multiresolution Image

Create a blocked image from a sample image included with the toolbox.

```
bim = blockedImage('tumor_091R.tif');
figure
bigimageshow(bim);
```

Inspect image size and world coordinate properties

```
bim.Size
```

```
ans = 3×3

        5000        5358           3
        1250        1340           3
         625         670           3
```

```
bim.WorldStart
```

```
ans = 3×3

    0.5000    0.5000    0.5000
    0.5000    0.5000    0.5000
    0.5000    0.5000    0.5000
```

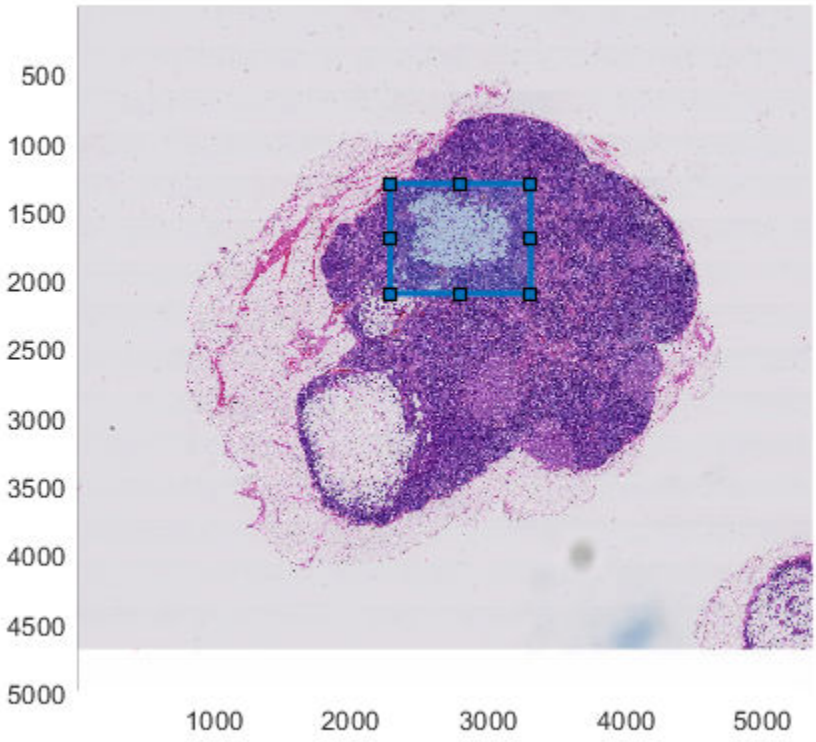
```
bim.WorldEnd
```

```
ans = 3×3
10^3 ×

    5.0005    5.3585    0.0035
    5.0005    5.3585    0.0035
```

```
    5.0005    5.3585    0.0035
```

Define a region of interest on the image that will be the crop area.

```
hrect = drawrectangle('Position', [2280 1300 1024 800]);
```



Obtain the world coordinates of the region.

```
wstartxy = hrect.Position(1:2);
wendxy = wstartxy + hrect.Position(3:4);
```

Convert to row-col order, include world coordinates of the last dimension.

```
wstart = [wstartxy(2), wstartxy(1), bim.WorldStart(1,3)];
wend   = [wendxy(2), wendxy(1), bim.WorldEnd(1,3)];
```

Convert to image subscripts, this is an optional step useful when using non-default world coordinates.

```
subs = world2sub(bim, [wstart; wend]);
cbim = crop(bim, subs(1,:), subs(2,:));
```

Inspect properties of the cropped image.

```
cbim.Size
```

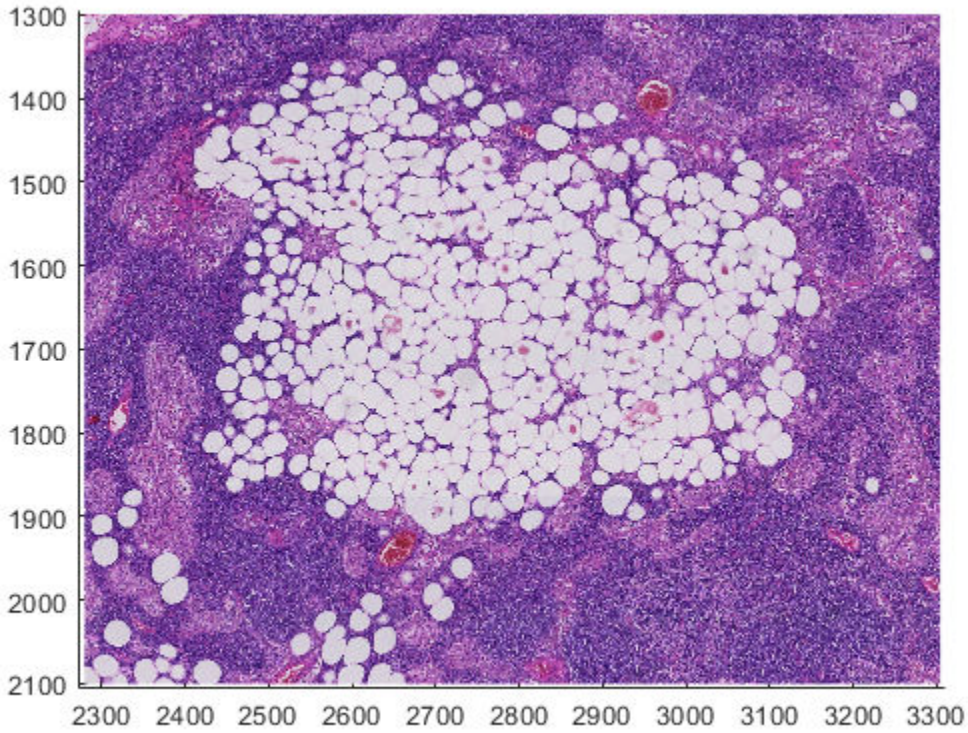
```
ans = 3×3
```

```
         801        1025           3
         201         258           3
         101         130           3
```

```
cbim.WorldStart
```

```
ans = 3×3
10^3 ×

    1.2995    2.2795    0.0005
    1.2965    2.2757    0.0005
    1.2965    2.2717    0.0005
```

```
figure
% Axes limits reflect cropped coordinates
bigimageshow(cbim);
```



Input Arguments

`bim` — Blocked image

`blockedImage` object

Blocked image, specified as a `blockedImage` object.

`cstart` — First pixel in crop window
1-by-*N* integer-valued vector

First pixel in the crop window, in pixel subscripts, specified as a 1-by-*N* integer-valued vector for an *N*-dimensional `blockedImage`. If `cstart` has fewer than *N* elements, `blockedImage` extends it with 1s.

`cend` — Last pixel in crop window
1-by-*N* integer-valued vector

Last pixel in the crop window, in pixel subscripts, specified as a 1-by-*N* integer-valued vector. If `cend` has fewer than *N* elements, `blockedImage` extends the image with the corresponding elements from `Size` at the finest level.

Output Arguments

`cbim` — Cropped blocked image
`blockedImage` object

Cropped blocked image, returned as a `blockedImage` object that contains image data in the crop window across all resolution levels.

See Also
`blockedImage`

Introduced in R2021a

gather

Collect blocks into current workspace

Syntax

```
data = gather(bim)
data = gather(bim,'Level',L)
```

Description

`data = gather(bim)` returns an image in the workspace formed by assembling all the blocks of the `blockedImage`, `bim`. By default, `gather` collects blocks from the coarsest level (level with the least amount of data).

`data = gather(bim,'Level',L)` returns an array formed by assembling all the blocks from the specified resolution level `L`.

Examples

Extract Finest and Coarsest Levels from Multiresolution Image

Create a blocked image using a modified version of image "tumor_091.tif" from the CAMELYON16 data set. The original image is a training image of a lymph node containing tumor tissue. The original image has eight resolution levels, and the finest level has resolution 53760-by-61440. The modified image has only three resolution levels. The spatial referencing of the modified image has been adjusted to enforce a consistent aspect ratio and to register features at each level.

```
bim = blockedImage('tumor_091R.tif');
```

Use `gather` to extract the coarsest resolution level. By default, `gather` returns the coarsest level

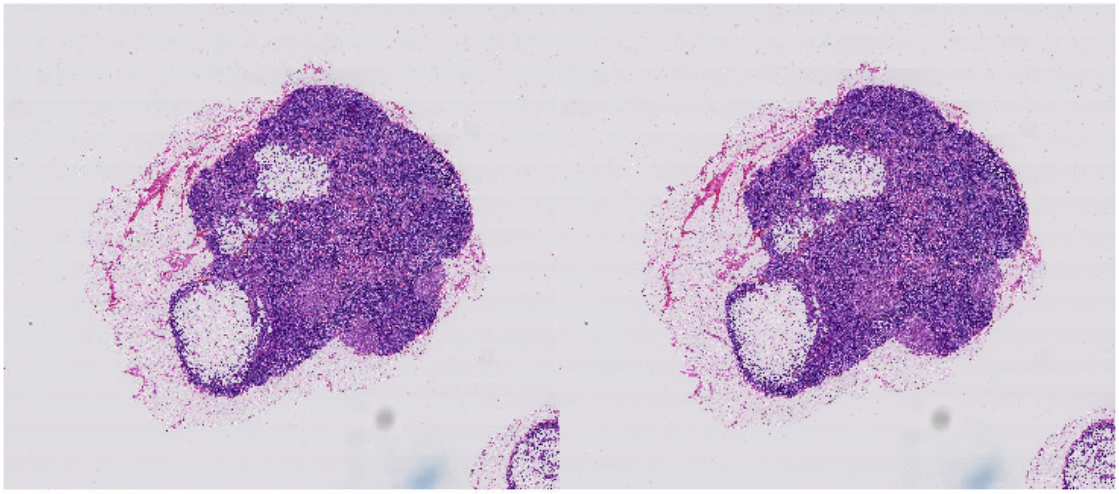
```
level_coarsest = gather(bim);
```

Use `gather` to extract the finest resolution level. You must specify the resolution level. The finest resolution level is always numbered 1.

```
level_finest = gather(bim, "Level", 1);
```

Display the coarsest resolution level and finest resolution level side-by-side.

```
montage({level_coarsest,level_finest})
```



Input Arguments

`bim` — Blocked image

`blockedImage` object

Blocked image, specified as a `blockedImage` object.

Output Arguments

`data` — Image formed from blocks

numeric array

Image formed from blocks, returned as a numeric array of the type specified by the `ClassUnderlying` property of the blocked image.

See Also

`blockedImage`

Introduced in R2021a

getBlock

Read specific block of blocked image

Syntax

```
blockdata = getBlock(bim,blocksub)
blockdata = getBlock(bim,blocksub,'Level',L)
[blockdata,blockinfo] = getBlock( ___ )
```

Description

`blockdata = getBlock(bim,blocksub)` returns `blockdata`, the block specified by the subscripts in `blocksub`.

`blockdata = getBlock(bim,blocksub,'Level',L)` gets the specified block from the `L`'th level of the multiresolution `blockedImage`, `bim`. By default, `L` is 1.

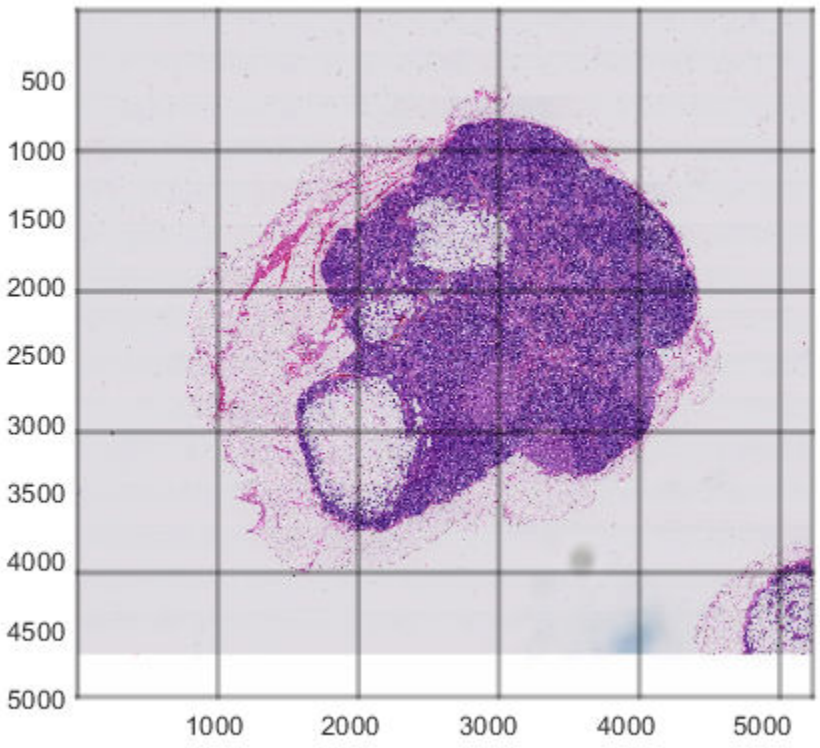
`[blockdata,blockinfo] = getBlock( ___ )` also returns block metadata in `blockinfo`.

Examples

Read Block from Blocked Image

Create a blocked image and display it.

```
bim = blockedImage('tumor_091R.tif');
bigimageshow(bim,...
    'GridVisible','on', 'GridLevel', 1,...
    'GridLineWidth', 2, 'GridColor','k','GridAlpha',0.3);
```



Display the value of the `SizeInBlocks` property.

```
disp(bim.SizeInBlocks)
```

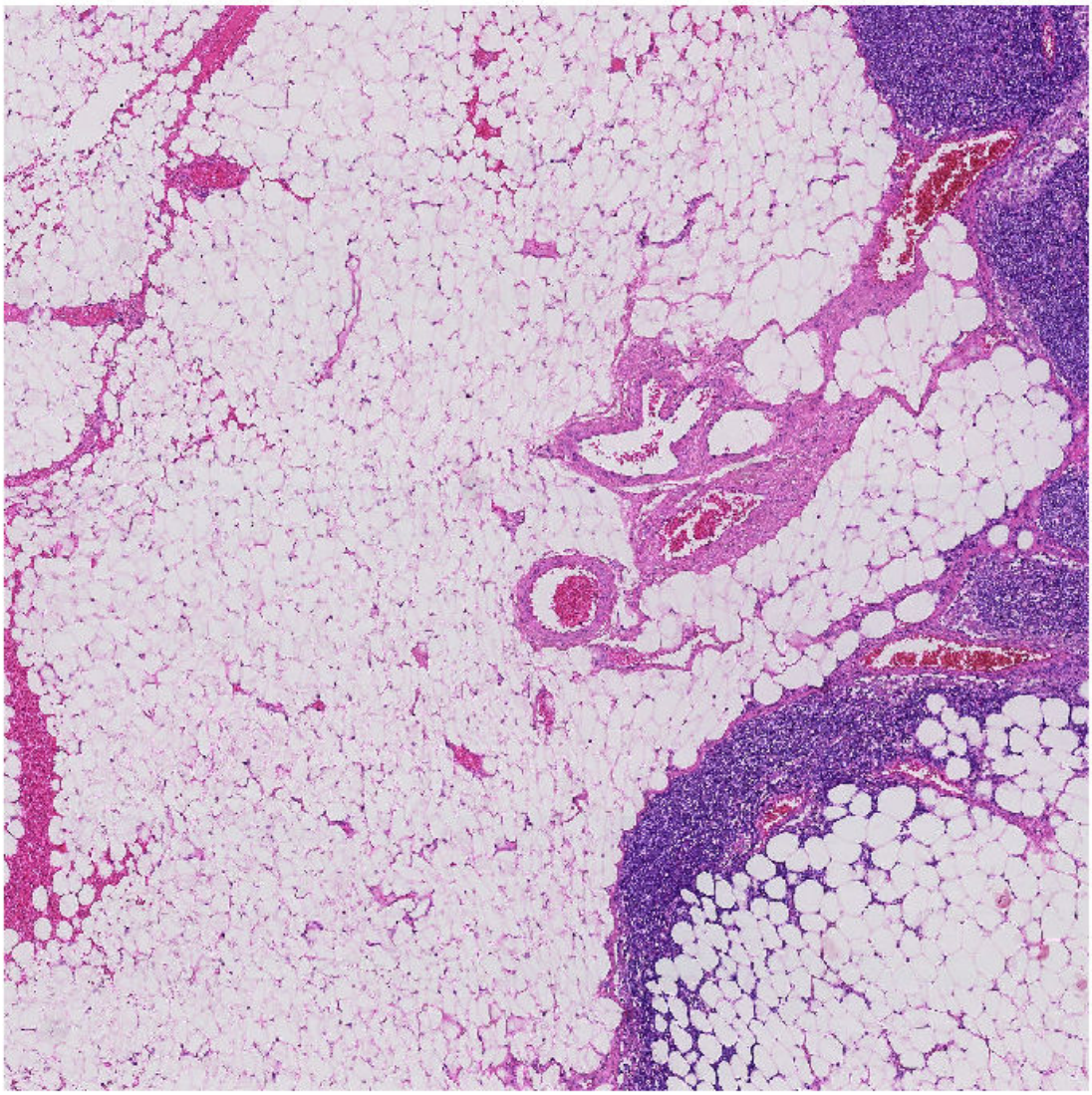
```
     5     6     1
     2     2     1
     1     1     1
```

Specify the block you want to read.

```
block = getBlock(bim, [3, 2, 1]);
```

Display the block.

```
imshow(block)
```



Input Arguments

`bim` — Blocked image
`blockedImage` object

Blocked image, specified as a `blockedImage` object.

`blocksub` — Block subscript vector
integer-valued vector

Block subscript vector, specified as a 1-by-*N* integer-valued vector. Valid elements range from 1 to the corresponding element in the `SizeInBlocks` property.

Example: `[3, 2, 1]`

Output Arguments

`blockdata` — Block of pixels from blocked image

numeric array

Block of pixels from blocked image, returned as a numeric array. Partial blocks around the edges can be smaller than the `BlockSize` property.

`blockinfo` — Metadata of block of pixels

scalar `struct`

Metadata of block of pixels, returned as a scalar `struct`. The `blockinfo` `struct` contains these fields.

Field	Description
`Start`	Subscripts of the first element in `blockdata`
`End`	Subscripts of the last element in `blockdata`
`Level`	Image level

See Also

`blockedImage` | `setBlock`

Introduced in R2021a

getRegion

Read arbitrary region of blocked image

Syntax

```
pixels = getRegion(bim,pixelStart,pixelEnd)
pixels = getRegion(bim,pixelStart,pixelEnd,Level=L)
```

Description

`pixels = getRegion(bim,pixelStart,pixelEnd)` returns all the pixels in the blocked image `bim`, in the region specified by `pixelStart` and `pixelEnd`.

`pixels = getRegion(bim,pixelStart,pixelEnd,Level=L)` gets the specified region from the `L`'th resolution level of the multiresolution `blockedImage` object, `bim`. `pixelStart` and `pixelEnd` are pixel subscripts at the `L`'th resolution level.

Examples

Read Same Region from Multiple Levels

Create a blocked image.

```
bim = blockedImage('tumor_091R.tif');
```

Specify a region in the blocked image and retrieve the data.

```
pstart_l1 = [2100, 1800, 1];
pend_l1 = [2600, 2300, 3];
imL1 = getRegion(bim, pstart_l1, pend_l1, "Level", 1);
```

Convert the start and end points of the region to world coordinates.

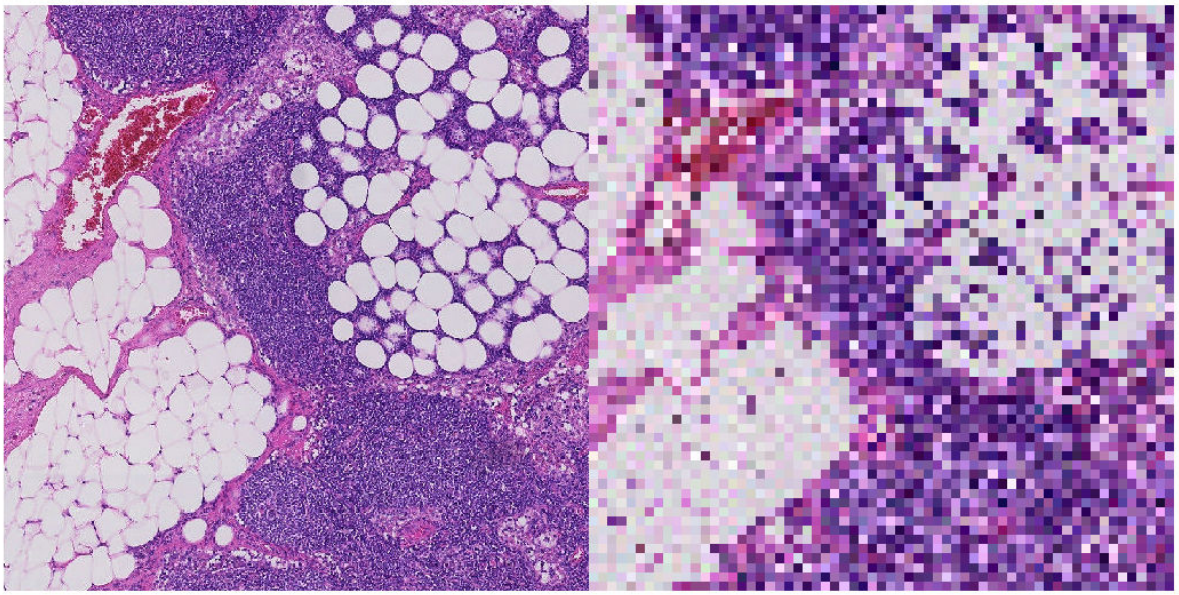
```
wstart_l1 = sub2world(bim, pstart_l1, "Level", 1);
wend_l1 = sub2world(bim, pend_l1, "Level", 1);
```

Convert the world coordinates to level 3 pixel subscripts.

```
pstart_l3 = world2sub(bim, wstart_l1, "level", 3);
pend_l3 = world2sub(bim, wend_l1, "level", 3);
imL3 = getRegion(bim, pstart_l3, pend_l3, "Level", 3);
```

Display the region in both resolution levels.

```
montage({imL1, imL3});
```



Input Arguments

`bim` — Blocked image

`blockedImage` object

Blocked image, specified as a `blockedImage` object.

`pixelStart` — Starting pixel of region

1-by-*N* vector

Starting pixel of the region, specified as a 1-by-*N* vector of pixel subscripts in the form `[row,column,...,subN]`, where *N* is the dimensionality of blocked image `bim`. Specify the pixel subscripts at the desired resolution level.

`pixelEnd` — Ending pixel of region

1-by-*N* vector

Ending pixel of the region, specified as a 1-by-*N* vector of pixel subscripts in the form `[row,column,...,subN]`, where *N* is the dimensionality of blocked image `bim`. Specify the pixel subscripts at the desired resolution level.

`L` — Resolution level

1 (default) | positive integer

Resolution level, specified as a positive integer that is less than or equal to the number of resolution levels of `bim`.

Output Arguments

`pixels` — Pixels in specified region

numeric array

Pixels in specified region, returned as a numeric array.

See Also

`blockedImage` | `getBlock` | `world2sub` | `sub2world` | `blocksub2sub` | `sub2blocksub`

Introduced in R2021a

setBlock

Put data in specific block of blocked image

Syntax

```
setBlock(bim,blocksub,blockdata)
setBlock(bim,blocksub,blockdata,'Level',L)
```

Description

`setBlock(bim,blocksub,blockdata)` sets the block content `blockdata` at the specified block subscript location, `blocksub` in the `blockedImage` object `bim`.

`setBlock(bim,blocksub,blockdata,'Level',L)` sets the block content at the `L`'th level of a multiresolution `blockedImage`. By default, `L` is `1`.

Examples

Write Data to Blocked Image and Read It Back

Create a blocked image to which you can write data. You specify the format of the blocked image in the `destination` parameter. To write to memory, specify an empty matrix. You must also specify the size of the image and the size of the blocks into which you want the image chunked. The initial value parameter depends on the format you specified in destination. To create a writable blocked image, specify the `'Mode'` parameter with the value `'w'` for write mode.

```
destination = [];
imgsize = [5 7];
blocksize = [2 2];
initval = uint8(0);
bim = blockedImage(destination,imgsize,blocksize,initval, "Mode", 'w');
```

Write data to the specified blocks in the blocked image by using the `setBlock` object function. The `blocksubs` parameter specifies the coordinates of the block to which you want to write data. The `blockdata` parameter specifies the data to write to the specified block. The size of blockdata must match the block size.

```
blocksubs = [1 1];
blockdata = ones(2,2,"uint8");
setBlock(bim, blocksubs, blockdata)
```

Close the image for writing.

Switch the blocked image to read mode by setting the 'Mode' parameter to 'r' for read.

```
bim.Mode = 'r'
```

```
bim =
```

```
  blockedImage with properties:

   Read only properties
             Source: [5x7 uint8]
            Adapter: [1x1 images.blocked.InMemory]
               Size: [5 7]
      SizeInBlocks: [3 4]
    ClassUnderlying: "uint8"

   Settable properties
          BlockSize: [2 2]
```

Create the full image by using the `gather` function to collect all the individual blocks.

```
fullImage = gather(bim);
```

Display details of the blocked image at the command line.

```
disp(fullImage)
```

```
   1   1   0   0   0   0   0
   1   1   0   0   0   0   0
   0   0   0   0   0   0   0
   0   0   0   0   0   0   0
   0   0   0   0   0   0   0
```

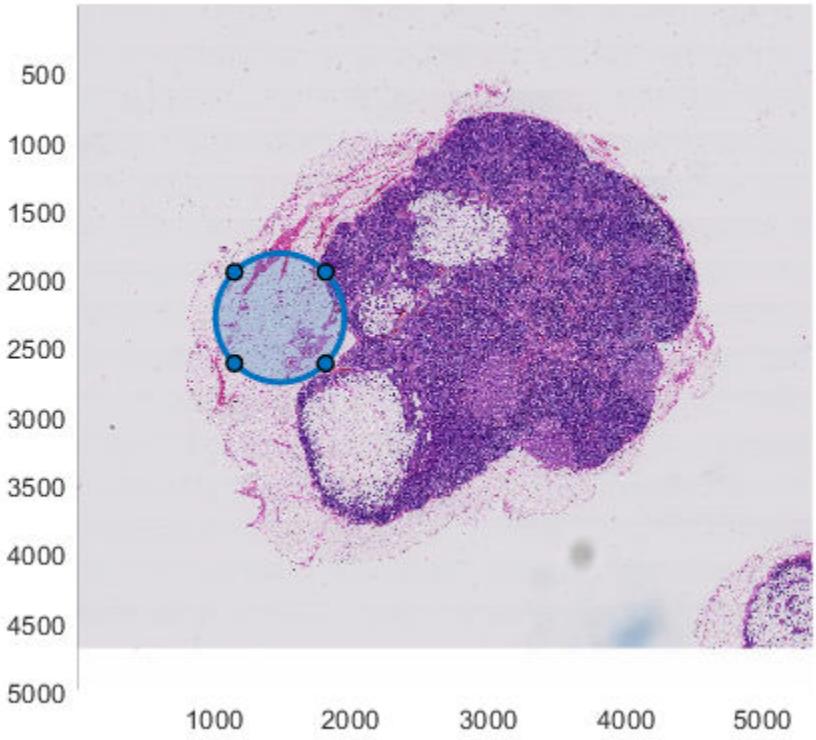
Create Mask from ROI

Create a blocked image.

```
bim = blockedImage('tumor_091R.tif');
```

Display the blocked image and draw a circular ROI on the image.

```
h = bigimageshow(bim);
hROI = drawcircle(gca, 'Radius', 470, 'Position', [1477 2284]);
```



Specify the resolution level at which to create the mask.

```
maskLevel = 3;
```

Create a writable blocked image in memory.

```
bmask = blockedImage([], [200 200], bim.Size(maskLevel,1:2), false, "Mode", "w");
```

Specify the start and ending points for the mask.

```
bmask.WorldStart = bim.WorldStart(maskLevel, 1:2);
bmask.WorldEnd = bim.WorldEnd(maskLevel, 1:2);
```

Display the number of blocks.

```
disp(bmask.SizeInBlocks);
```

```
     1     1
```

Convert the ROI coordinates to pixel level.

```
roiPositionsRC = fliplr(hROI.Vertices); % x,y to row,column
roiPosSub = world2sub(bmask, roiPositionsRC, "level", 1);

for cSub = 1:bmask.SizeInBlocks(2)
    for rSub = 1:bmask.SizeInBlocks(1)
        blockSub = [rSub, cSub];
        [pStart, pEnd] = blocksub2sub(bmask, blockSub, "Level", 1);
```

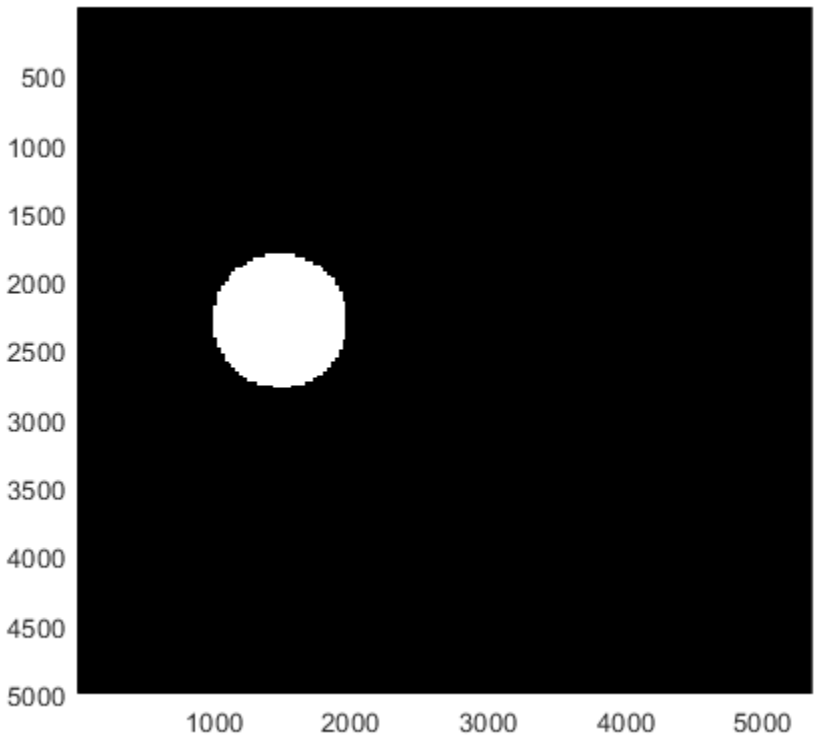
```
        % Create a grid encompassing all pixels in the block in X-Y order
        [xgrid,ygrid] = meshgrid(pStart(2):pEnd(2), pStart(1):pEnd(1));

        % Create in/out mask for this block
        tileMask = inpolygon(xgrid, ygrid,...
                roiPosSub(:,2), roiPosSub(:,1));

        % Write out the block
        setBlock(bmask, blockSub, tileMask);

    end
end
```

Switch the blocked image to read mode.

```
bmask.Mode = 'r';
```

Display the mask.

```
figure
bigimageshow(bmask)
```



Input Arguments

`bim` — Blocked image
`blockedImage` object

Blocked image, specified as a `blockedImage` object.

`blocksub` — Block subscript vector
integer-valued vector

Block subscript vector, specified as a 1-by-*N* integer-valued block subscript vector. Valid elements range from `1` to the corresponding element in `SizeInBlocks` property.

Example: `[3, 2, 1]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`blockdata` — Block of data
numeric array

Block of data, specified as a numeric array with dimensions that match `BlockSize`. The type matches the type specified by the `ClassUnderlying` property. `setBlock` automatically trims blocks along the edges to fit the `Size` property.

See Also

`blockedImage` | `getBlock`

Introduced in R2021a

sub2blocksub

Convert pixel subscripts to block subscripts

Syntax

```
blocksub = sub2blocksub(bim,pixelsub)
blocksub = sub2blocksub(bim,pixelsub,'Level',L)
```

Description

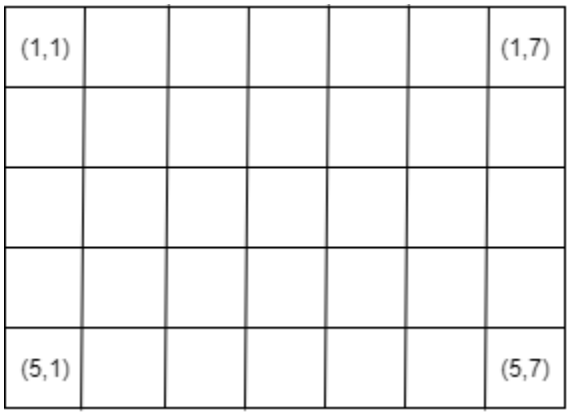
`blocksub = sub2blocksub(bim,pixelsub)` converts the pixel subscripts in `pixelsub` to the block subscripts identifying the block containing the specified pixel.

`blocksub = sub2blocksub(bim,pixelsub,'Level',L)` additionally specifies the resolution level to use in a multiresolution image. By default, `Level` is `1`.

Examples

Convert Pixel Subscripts to Block Subscripts

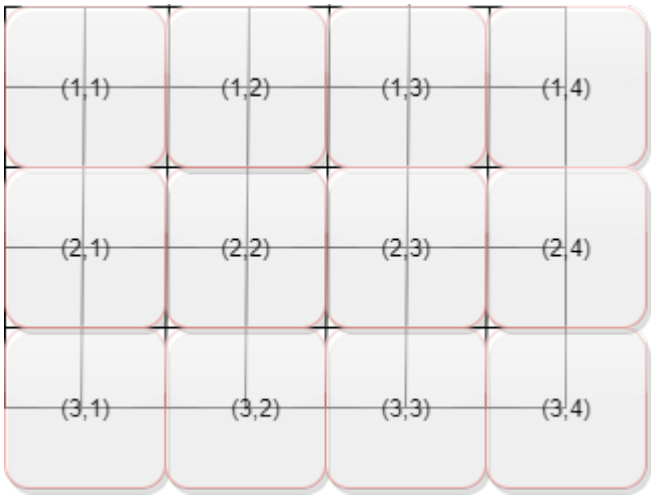
Create a small sample image as a 5-by-7 matrix of zeros. Here is an illustration of the small sample image, with the pixel coordinates of the four corners provided.



Create a blocked image from the sample image, specifying a 2-by-2 block size. To create this blocked image, use `blockedImage` in write mode.

```
bim = blockedImage([],[5 7],[2 2],uint8(0),"Mode",'w');
```

Here is an illustration of the blocked image overlaid on the original image. It is divided into 2-by-2 blocks. In the diagram, each block contains it's block coordinates.



To determine which block contains a particular pixel, convert the pixel subscripts into block subscripts by using the sub2`blocksub` function. By default, if the image is a multiresolution image, then sub2`blocksub` uses pixel coordinates from coarsest level, although you can specify any level. Since the sample image has only one resolution level, sub2`blocksub` converts level 1.

```
[blocksub] = sub2blocksub(bim,[2 3])
```

```
blocksub = 1×2

     1     2
```

Input Arguments

`bim` — Blocked image

`blockedImage` object

Blocked image, specified as a `blockedImage` object.

`pixelsub` — Pixel subscripts

K-by-*N* integer-valued matrix

Pixel subscripts, specified as a *K*-by-*N* integer-valued matrix, *N* is the number of dimensions and *K* is the number of coordinates.

Output Arguments

`blocksub` — Subscripts of block that contains pixel

K-by-*N* integer-valued matrix

Subscripts of the block that contains the pixel, returned as a *K*-by-*N* integer-valued matrix, for an *N*-dimensional blocked image. *K* is the number of coordinates.

See Also

`blockedImage`

Introduced in R2021a

sub2world

Convert pixel subscripts to block subscripts

Syntax

```
world = sub2world(bim,pixelsub)
world = sub2world(bim,pixelsub,'Level',L)
```

Description

`world = sub2world(bim,pixelsub)` converts the pixel subscripts, `pixelsub`, to the block subscript of the block containing the corresponding pixel.

`world = sub2world(bim,pixelsub,'Level',L)` additionally specifies the resolution level to use in a multiresolution image. `Level` defaults to 1.

Examples

Convert Pixel Subscripts to World Coordinates

Convert pixel subscripts from one level to another via the world coordinates to refer to the same spatial region.

Create a blocked image from a sample image included with the toolbox.

```
bim = blockedImage('tumor_091R.tif');
```

Define a region of interest in the finest resolution level in pixel subscripts.

```
level1PixelSubStart = [1700, 1550 1];
level1PixelSubEnd = [2100, 2000 3];
```

Get the image data from the region of interest on the resolution level 1 image.

```
imr = getRegion(bim, level1PixelSubStart, level1PixelSubEnd, "Level", 1);
size(imr)
```

```
ans = 1×3

   401   451     3
```

Convert the pixel subscripts that define the region of interest into world coordinates. By default, `sub2world` converts the coordinates at level 1, the finest resolution.

```
worldRegion = sub2world(bim,[level1PixelSubStart; level1PixelSubEnd]);
```

Compute a binary mask at the coarsest level.

```
bbw = apply(bim, @(bs)imbinarize(im2gray(bs.Data)), "Level", 3);
```

Convert the world coordinates of the region of interest to pixel subscripts of the mask. Note that the mask has only two dimensions.

```
worldRegion = worldRegion(:,1:2);
maskPixelSubs = world2sub(bbw,worldRegion);
```

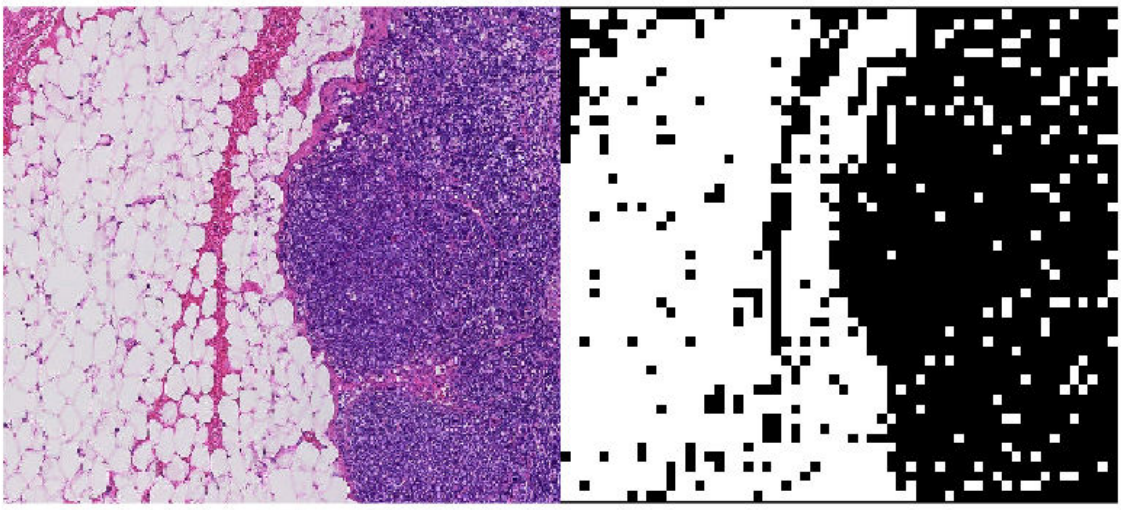
Corresponding mask region.

```
bwr = getRegion(bbw, maskPixelSubs(1,:), maskPixelSubs(2,:));
size(bwr)
```

```
ans = 1×2

    51    58
```

View the original image and the mask.

```
montage({imr,bwr})
```



Input Arguments

`bim` — Blocked image

`blockedImage` object

Blocked image, specified as a `blockedImage` object.

`pixelsub` — Pixel subscripts

K-by-*N* integer-valued vector

Pixel subscripts, specified as a *K*-by-*N* integer-valued vector, where *N* is the number of dimensions and *K* is the number of coordinates.

Output Arguments

`world` — World subscripts

K-by-*N* numeric array

World subscripts, returned as a *K*-by-*X* numeric array. The world coordinates are in the same order as the pixel subscripts. For an N-dimensional `blockedImage`, and *K* subscripts (each row is a separate subscript), `pixelsub` is a *K*-by-*N* integer-valued matrix and `world` is a *K*-by-*N* numeric matrix.

See Also

`blockedImage` | `world2sub`

Introduced in R2021a

world2sub

Convert world coordinates to pixel subscripts

Syntax

```
pixelsub = world2sub(bim,world)
pixelsub = world2sub(bim,world,'Level',L)
```

Description

`pixelsub = world2sub(bim,world)` converts the world coordinates, `world`, to the corresponding pixel subscripts, `pixelsub`. The world coordinates should be in the same order as the pixel subscripts.

`pixelsub = world2sub(bim,world,'Level',L)` additionally specifies the resolution level to use in a multi-resolution image. By default, `Level` is `1`.

Examples

Convert Pixel Subscripts to World Coordinates

Convert pixel subscripts from one level to another via the world coordinates to refer to the same spatial region.

Create a blocked image from a sample image included with the toolbox.

```
bim = blockedImage('tumor_091R.tif');
```

Define a region of interest in the finest resolution level in pixel subscripts.

```
level1PixelSubStart = [1700, 1550 1];
level1PixelSubEnd = [2100, 2000 3];
```

Get the image data from the region of interest on the resolution level 1 image.

```
imr = getRegion(bim, level1PixelSubStart, level1PixelSubEnd, "Level", 1);
size(imr)
```

```
ans = 1×3

   401   451     3
```

Convert the pixel subscripts that define the region of interest into world coordinates. By default, `sub2world` converts the coordinates at level 1, the finest resolution.

```
worldRegion = sub2world(bim,[level1PixelSubStart; level1PixelSubEnd]);
```

Compute a binary mask at the coarsest level.

```
bbw = apply(bim, @(bs)imbinarize(im2gray(bs.Data)), "Level", 3);
```

Convert the world coordinates of the region of interest to pixel subscripts of the mask. Note that the mask has only two dimensions.

```
worldRegion = worldRegion(:,1:2);
maskPixelSubs = world2sub(bbw,worldRegion);
```

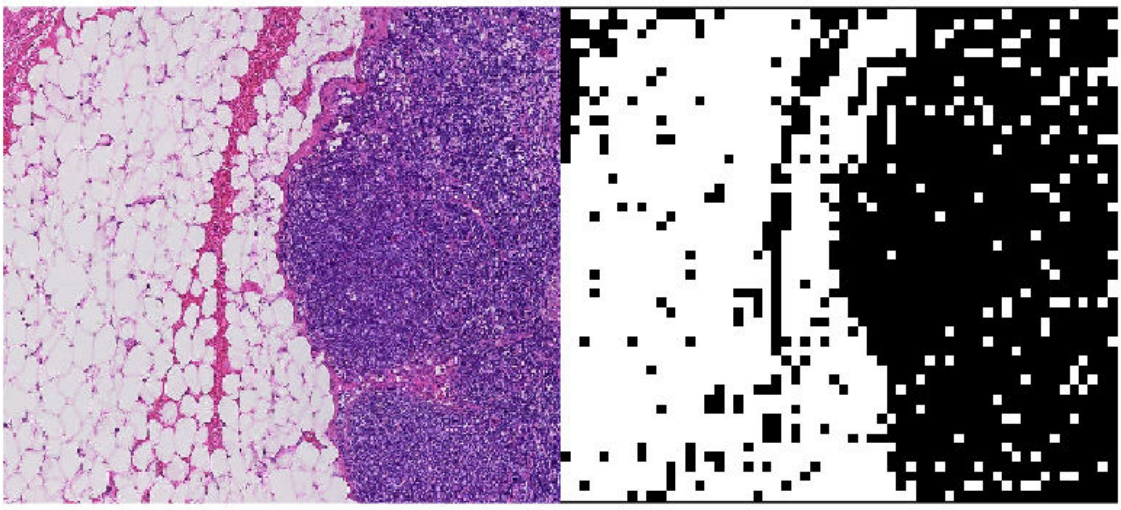
Corresponding mask region.

```
bwr = getRegion(bbw, maskPixelSubs(1,:), maskPixelSubs(2,:));
size(bwr)
```

```
ans = 1×2

    51    58
```

View the original image and the mask.

```
montage({imr,bwr})
```



Input Arguments

`bim` — Blocked image

`blockedImage` object

Blocked image, specified as a `blockedImage` object.

`world` — World coordinates

numeric matrix

World coordinates, specified as a *K*-by-*N* numeric matrix, where *K* is the number of world coordinate vectors and *N* is the number of dimensions of the `blockedImage`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

`pixelsub` — Pixel subscripts

K-by-*N* integer-valued matrix

Pixel subscripts, returned as a *K*-by-*N* integer-valued matrix, where *N* is the number of dimensions and *K* is the number of coordinates.

Tips

- World coordinates are theoretically continuous domain values represented by floating point numbers. Subscripts are discrete integer values that can be used to index into the underlying array. Floating point computation and rounding may cause small changes in world coordinates around the edge of pixels to map to different neighboring subscript locations. `world2sub` rounds up world coordinate values on the edge of two pixels, except for pixels on the border, where it rounds down to the last pixel.

See Also

`blockedImage` | `sub2world`

Introduced in R2021a

write

Write blocked image data to new destination

Syntax

```
write(bim,destination)
write(bim,destination,Name,Value)
```

Description

`write(bim,destination)` writes the blocked image data `bim` to the location specified by `destination`.

`write(bim,destination,Name,Value)` specifies additional options for writing the blocked image data using name-value arguments.

Examples

Create Pyramidal Representation of Single Image

Create a blocked image and view the value of the `Size` property.

```
bim = blockedImage('cameraman.tif');
disp(bim.Size)
```

```
   256   256
```

Resize the blocked image using the blocked image `apply` object function to call `imresize`.

```
bsub1 = bim.apply(@(bs)imresize(bs.Data,0.5));
```

Create the new resolution level for the image.

```
write(bim, "pyramid2.tif", "LevelImages", bsub1, "BlockSize", [ 32 32]);
```

Create a new, multiresolution blocked image and display the `Size` property.

```
bpyramid = blockedImage("pyramid2.tif");
disp(bpyramid.Size)
```

```
   256   256
   128   128
```

Write Two Levels from a Three Level Image

Create a blocked image and view the value of the `Size` property.

```
bim = blockedImage('tumor_091R.tif');
disp(bim.Size)
```

```
        5000        5358           3
        1250        1340           3
         625         670           3
```

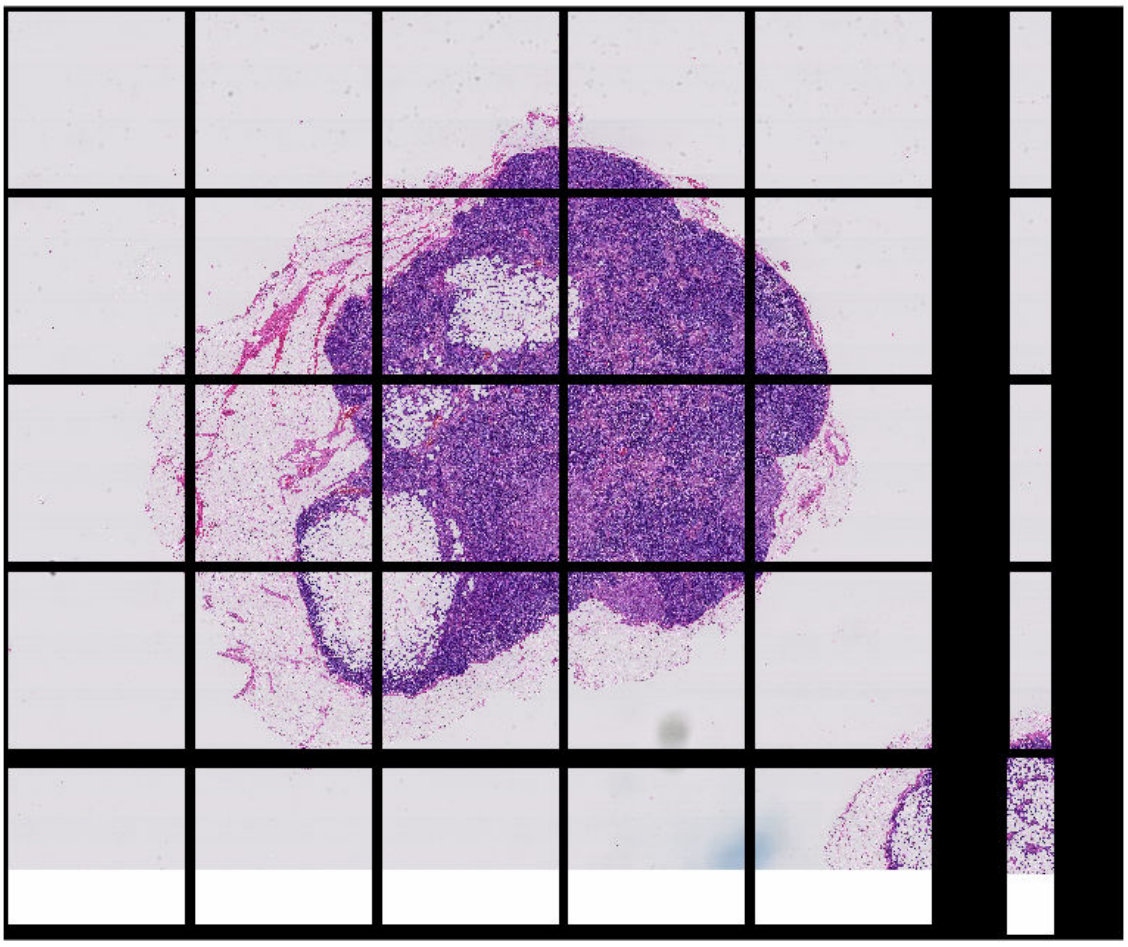
Write only two levels from a three level image. Write to a folder where each block is a saved in a separate PNG file.

```
write(bim, "lev1_and_3", "Levels", [1 3], "Adapter", images.blocked.PNGBlocks);
bim2 = blockedImage("lev1_and_3");
disp(bim2.Size)
```

```
        5000        5358           3
         625         670           3
```

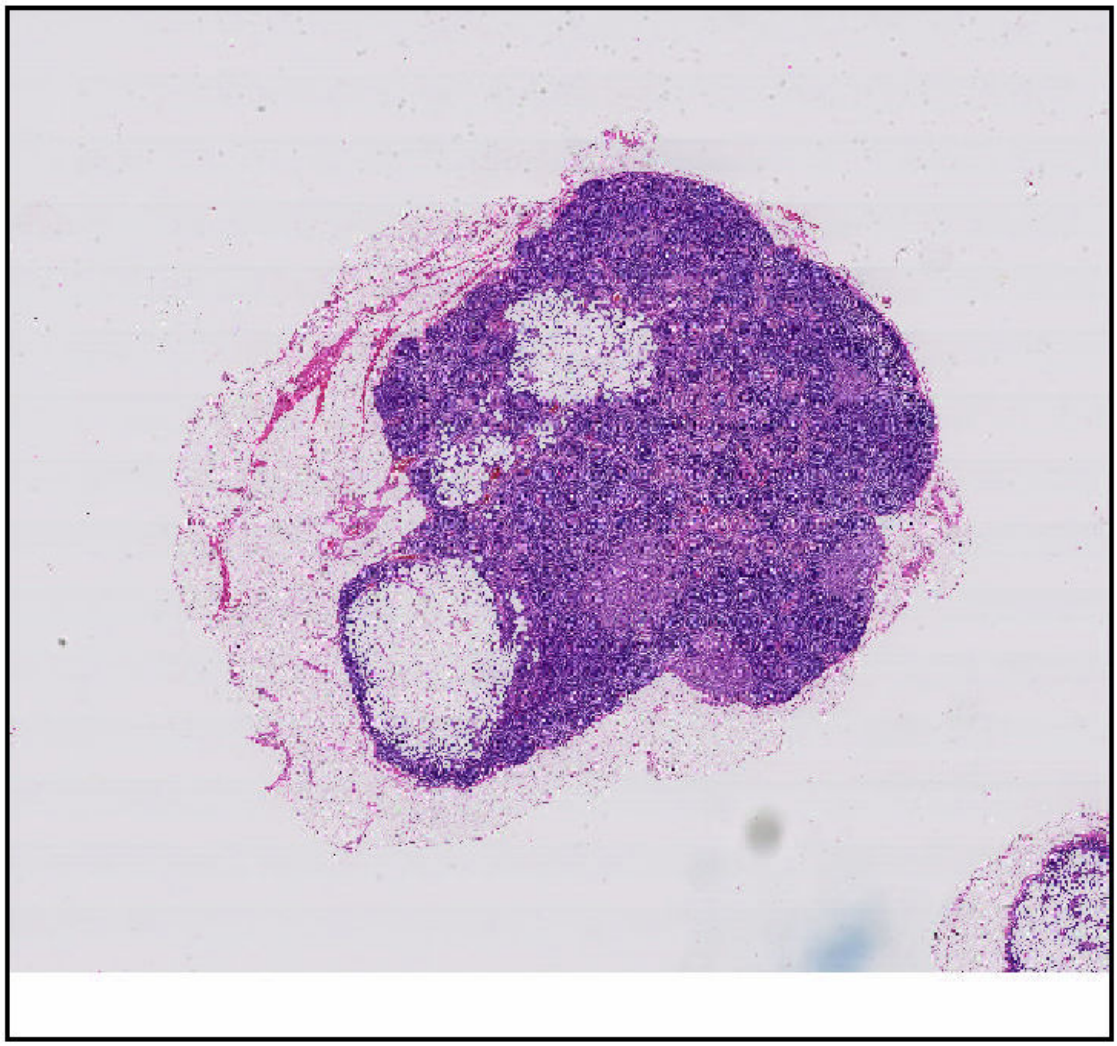
Inspect the output folder structure, using the Image Browser app, or view a montage of the images in the folders `lev1_and_3/L1` and `lev1_and_3/L2`.

```
figure; montage(imageDatastore('lev1_and_3/L1'), 'BorderSize', 5);
```



View a montage of the images in the folder `lev1_and_3/L2`.

```
figure; montage(imageDatastore('lev1_and_3/L2'), 'BorderSize', 5);
```



Write HDF5 Compressed Data for Archival Use

Create a blocked image.

```
bim = blockedImage('tumor_091R.tif');
```

Write the data from the three resolution levels of the blocked image to an H5 file. Specify the blocked image adapter for HDF5 files and the compression level when you write the data.

```
wadapter = images.blocked.H5;
wadapter.GZIPLevel = 5;
write(bim, "tumor_091.h5", "Adapter", wadapter);
```

Display information about the HDF5 image that was created.

```
h5disp("tumor_091.h5");
```

```
HDF5 tumor_091.h5
Group '/'
    Group '/blockedImage'
        Attributes:
            'Size':  3x3 H5T_FLOAT
            'IOBlockSize':  3x3 H5T_FLOAT
            'Datatype':  'uint8', 'uint8', 'uint8'
        Dataset 'L1'
            Size:  5000x5358x3
            MaxSize:  5000x5358x3
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  1024x1024x3
            Filters:  deflate(5)
            FillValue:  0
        Dataset 'L2'
            Size:  1250x1340x3
            MaxSize:  1250x1340x3
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  1024x1024x3
            Filters:  deflate(5)
            FillValue:  0
        Dataset 'L3'
            Size:  625x670x3
            MaxSize:  625x670x3
            Datatype:   H5T_STD_U8LE (uint8)
            ChunkSize:  625x670x3
            Filters:  deflate(5)
            FillValue:  0
```

Input Arguments

`bim` — Blocked image

`blockedImage` object

Blocked image, specified as a `blockedImage` object.

`destination` — Location to place writable data

char vector | string scalar

Location to place writable data, specified as a character vector or string scalar.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"Levels",[1 3]`

`Adapter` — Adapter used for writing blocked image data
adapter object

Adapter used for writing blocked image data, specified as an adapter object.

Adapter	Description
`BINBlocks`	Store each block as a binary file in a folder
`GenericImage`	Store blocks in a single image
`GenericImageBlocks`	Store each block as an image file in a folder
`H5`	Store blocks in a single HDF5 image
`H5Blocks`	Store each block as an HDF5 file in a folder
`InMemory`	Store blocks in a variable in main memory
`JPEGBlocks`	Store each block as a JPEG file in a folder
`MATBlocks`	Store each block as a MAT file in a folder
`PNGBlocks`	Store each block as a PNG file in a folder
`TIFF`	Store blocks in a single TIFF file

You can also specify a custom adapter that performs custom writing operations. For more information, see `images.blocked.Adapter`.

You must specify a value for the `'OutputLocation'` property for all adapters except `InMemory`. If you do not specify a value for the `OutputLocation` parameter, `write` uses `InMemory` as the default adapter. Otherwise, the default adapter is `BINBlocks` for numeric and logical data and `MATBlocks` for `struct` and categorical data.

`BlockSize` — Output block size
1-by-*D* vector of positive integers

Output block size, specified as 1-by-*D* vector of positive integers, where *D* is the value of the `NumDimensions` property of the blocked image `bim`. The default block size is equal to the `BlockSize` property of `bim`.

`DisplayWaitbar` — Display wait bar
`true` (default) | `false`

Display wait bar, specified as `true` or `false`. When set to `true`, the `write` object function displays a wait bar for long-running operations. If you cancel the wait bar, the `write` function returns a partial output, if available.

`LevelImages` — Additional single-level blocked images
vector of `blockedImage` objects

Additional single-level blocked images, specified as a vector of single-level `blockedImage` objects. The `write` function appends these additional single-level blocked image to the specified blocked image (`bim`) to create a multiresolution blocked image. The single-level blocked images should have the same `NumDimensions` property value as `bim`.

`Levels` — Levels of multiresolution blocked image to write
`[ ]` (default) | vector of positive integers

Levels of a multiresolution blocked image to write, specified as vector of positive integers. Use this argument to selectively write the specified levels of the blocked image to the destination.

See Also

`blockedImage` | `images.blocked.Adapter`

Introduced in R2021a

blockedImageDatastore

Datastore for use with blocks from `blockedImage` objects

Description

A `blockedImageDatastore` object manages a collection of image blocks that belong to one or more `blockedImage` objects. A `blockedImageDatastore` is analogous to an `imageDatastore`, which manages a collection of unrelated images.

Creation

Syntax

```
bimds = blockedImageDatastore(Images)
bimds = blockedImageDatastore(sources)
bimds = blockedImageDatastore( ___ ,Name,Value)
```

Description

`bimds = blockedImageDatastore(Images)` creates a `blockedImageDatastore` object that manages a collection of image blocks of one or more `blockedImage` objects, Images.

If `Images` contains an object with multiple resolution levels, then `blockedImageDatastore` chooses only blocks from the finest resolution level. The `BlockSize` property of the first element in `Images` is the default datastore block size.

`bimds = blockedImageDatastore(sources)` creates a datastore from the files specified by `sources`.

`bimds = blockedImageDatastore( ___ ,Name,Value)` also uses name-value arguments to set one or more properties on page 1-306, except for `Images` and `TotalNumBlocks`.

Input Arguments

`sources` — Name of blocked image files
cell array of character vectors | string array | `FileSet` object

Name of the blocked image files, specified as a cell array of character vectors, a string scalar, or a `FileSet` object. The `blockedImageDatastore` object converts the images in the files into blocked images and sets those images as the Images property.

Properties

`BlockLocationSet` — Blocks to include in datastore
`blockLocationSet` object

Blocks to include in the datastore, specified as a `blockLocationSet` object. The object specifies which blocks to include from the blocked image `bims`. You can repeat or omit individual blocks. To obtain the default value, `blockedImageDatastore` calls the `selectBlockLocations` function.

You cannot change the `BlockLocationSet` property after creating the `blockedImageDatastore`.

`BlockSize` — Block size
1-by-*D* numeric vector

Block size, specified as a 1-by-*D* numeric vector, where *D* is the number of dimensions of the first blocked image in `Images`, at the first resolution level in `Levels`.

You cannot change the `BlockSize` property after creating the `blockedImageDatastore`.

`BorderSize` — Size of additional block border elements
1-by-D vector of nonnegative integers

Size of additional block border elements in each dimension, specified as a 1-by-*D* numeric vector, where *D* is the number of dimensions of the first blocked image in `Images`, at the first resolution level in `Levels`. The default value is `zeros(1,D)`.

`Images` — Blocked images
array of `blockedImage` objects

Blocked images that supply blocks for the `blockedImageDatastore`, specified as an array of `blockedImage` objects. All elements of `Images` must have the same number of dimensions and be of the same type.

You cannot change the `Images` property after creating the `blockedImageDatastore`.

`PadMethod` — Method used for padding incomplete blocks
numeric scalar | `'replicate'`

Method used for padding incomplete blocks, specified as one of the values in this table. By default, the datastore pads numeric blocks with the value of the `InitialValue` property of the first blocked image in the array of blocked images, `Images`.

Value	Meaning
numeric scalar	Pad incomplete blocks with the specified scalar value. The type of the value depends on the `ClassUnderlying` of the blocked images in `Images`.
`'replicate'`	Pad by repeating border elements of array.

`PadPartialBlocks` — Pad partial blocks
`true` (default) | `false`

Pad partial blocks that exist on the edge, specified as a logical scalar `true` or `false`. When `true`, the blocked image datastore add padding according to the padding method specified in the `PadMethod` property.

`ReadSize` — Number of blocks to return in each call to `read` function
1 (default) | positive integer

Number of blocks to return in each call to the `read` function, specified as a positive integer. Each call to the `read` function reads at most `ReadSize` blocks

`TotalNumBlocks` — Total number of blocks available
numeric scalar

This property is read-only.

Total number of blocks available, specified as a numeric scalar.

Object Functions

combine	Combine data from multiple datastores
countEachLabel	Counts number of pixel labels for each class
hasdata	Returns true if more data is available in blockedImageDatastore
numpartitions	Number of datastore partitions
partition	Return partitioned part of blockedImageDatastore
preview	Preview subset of data in datastore
read	Read data and metadata from blockedImageDatastore
readall	Read all data from the blockedImageDatastore
reset	Reset datastore to initial state
shuffle	Shuffle data in datastore
subset	Create subset of datastore or file-set
transform	Transform datastore

Examples

Create blockedImageDatastore at Specific level and Block Size

Create a blockedImage.

```
bim = blockedImage('tumor_091R.tif');
```

Create a datastore, specifying the resolution level and the blocksize.

```
bls = selectBlockLocations(bim,"Levels",2,"BlockSize",[512, 512]);
bimds = blockedImageDatastore(bim, "BlockLocationSet", bls);
```

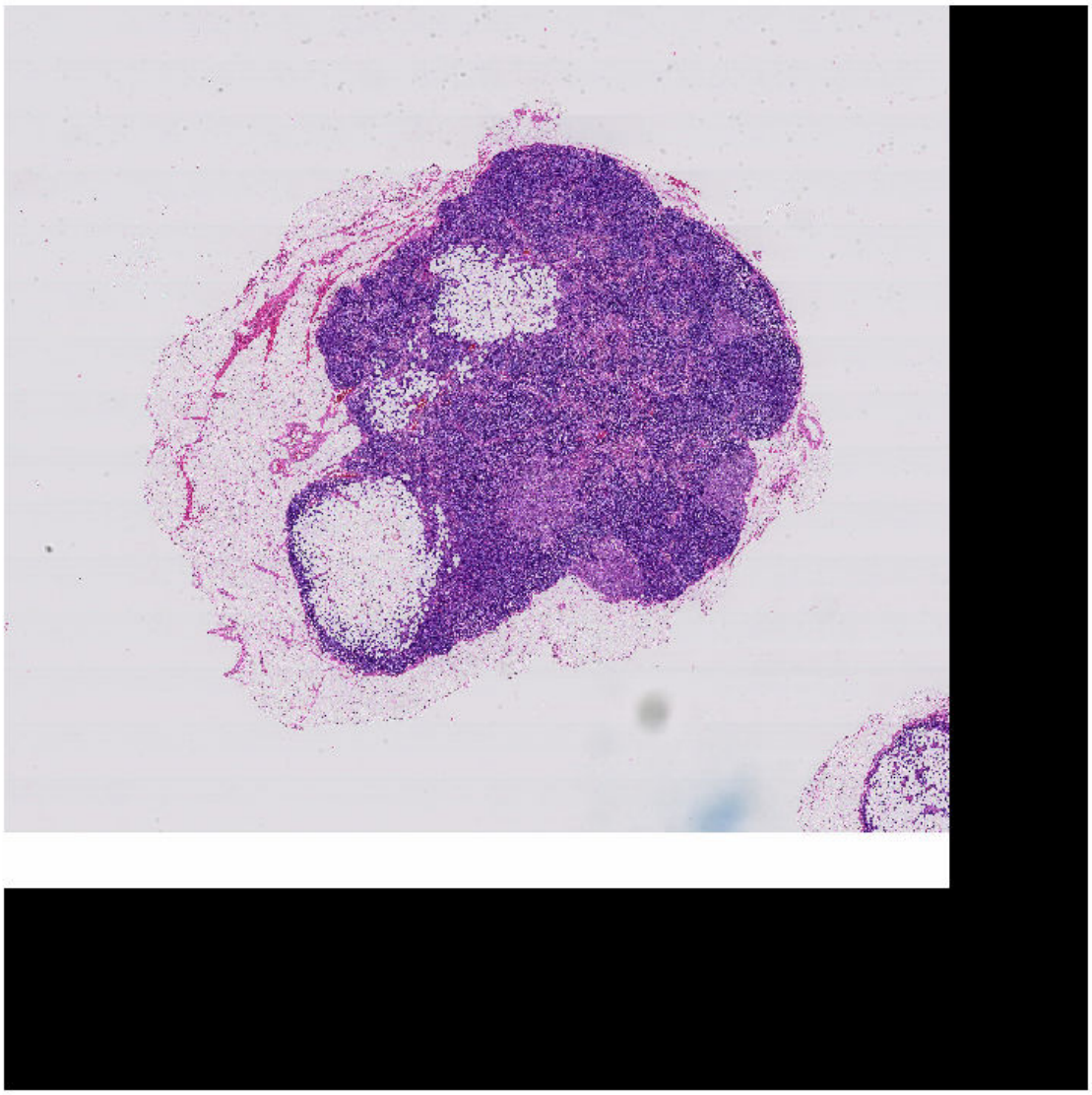
Read all the blocks in the datastore.

```
b = readall(bimds)
```

```
b=9×1 cell array
    {512×512×3 uint8}
    {512×512×3 uint8}
    {512×512×3 uint8}
    {512×512×3 uint8}
    {512×512×3 uint8}
    {512×512×3 uint8}
    {512×512×3 uint8}
    {512×512×3 uint8}
    {512×512×3 uint8}
```

Display the blocked image

```
montage(b)
```



Create blockedImageDatastore from Multiple Files

Create a `FileSet` object containing multiple image files of the PNG file format.

```
fs = matlab.io.datastore.FileSet( ...
     fullfile(matlabroot,"toolbox","images","imdata"), ...
    "FileExtensions",".png");
```

Create a `blockedImage` object, specifying an adapter. This saves time by skipping the need to inspect each file to pick a suitable adapter.

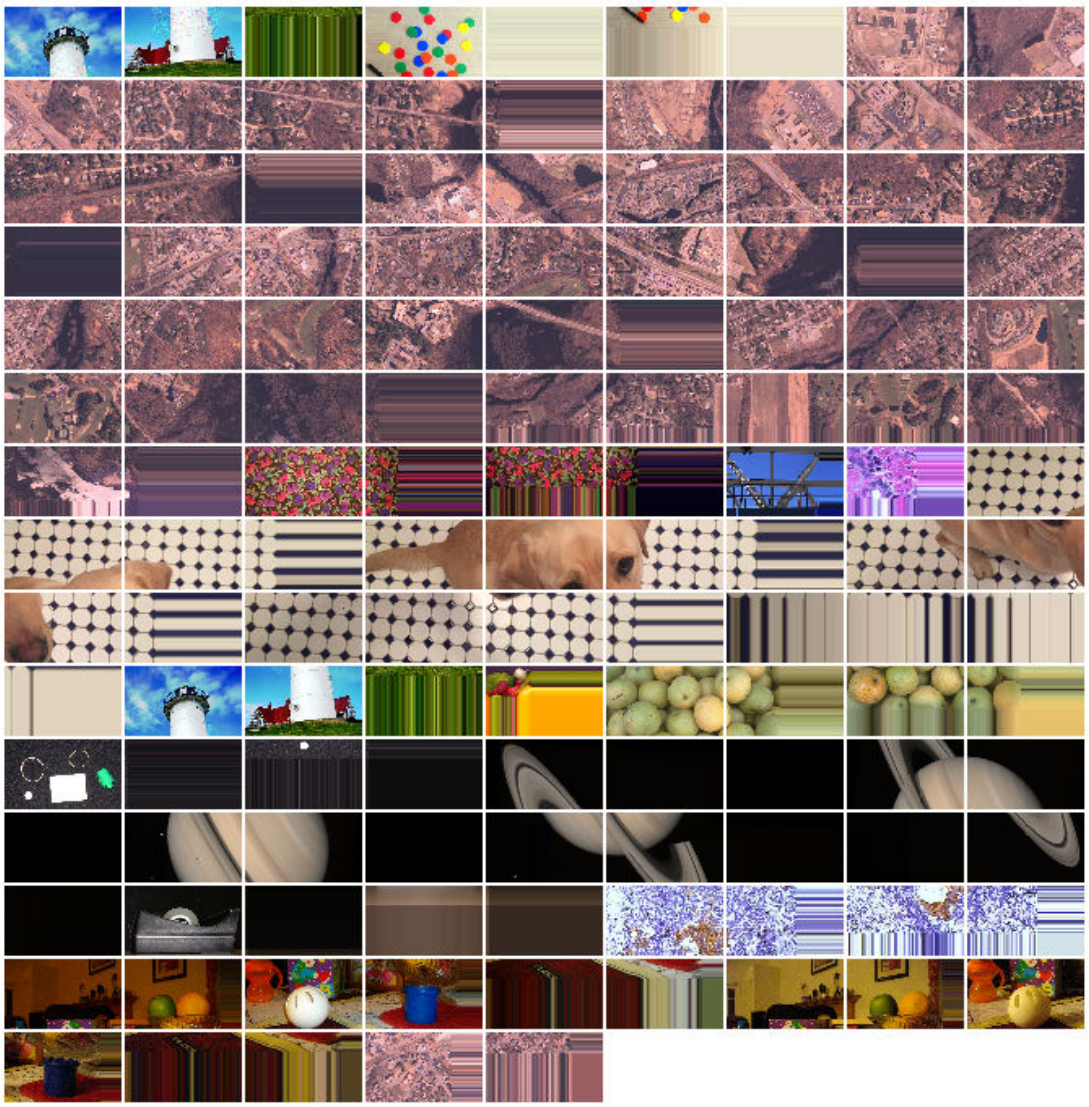
```
readAdapter = images.blocked.GenericImage;
bims = blockedImage(fs,"Adapter",readAdapter);
```

All images must have the same number of dimensions, so only retain RGB images.

```
bims = bims([bims.NumDimensions]==3);
bimds = blockedImageDatastore(bims,"BlockSize",[300 500], ...
     "PadMethod","replicate");
```

Display all of the blocks in the `blockedImageDatastore`.

```
montage(readall(bimds),"Border",2,"BackgroundColor","w");
```



Create blockedImageDatastore with Overlapping Blocks

Create a `blockedImage`.

```
bim = blockedImage('tumor_091R.tif');
```

Specify overlapping blocks.

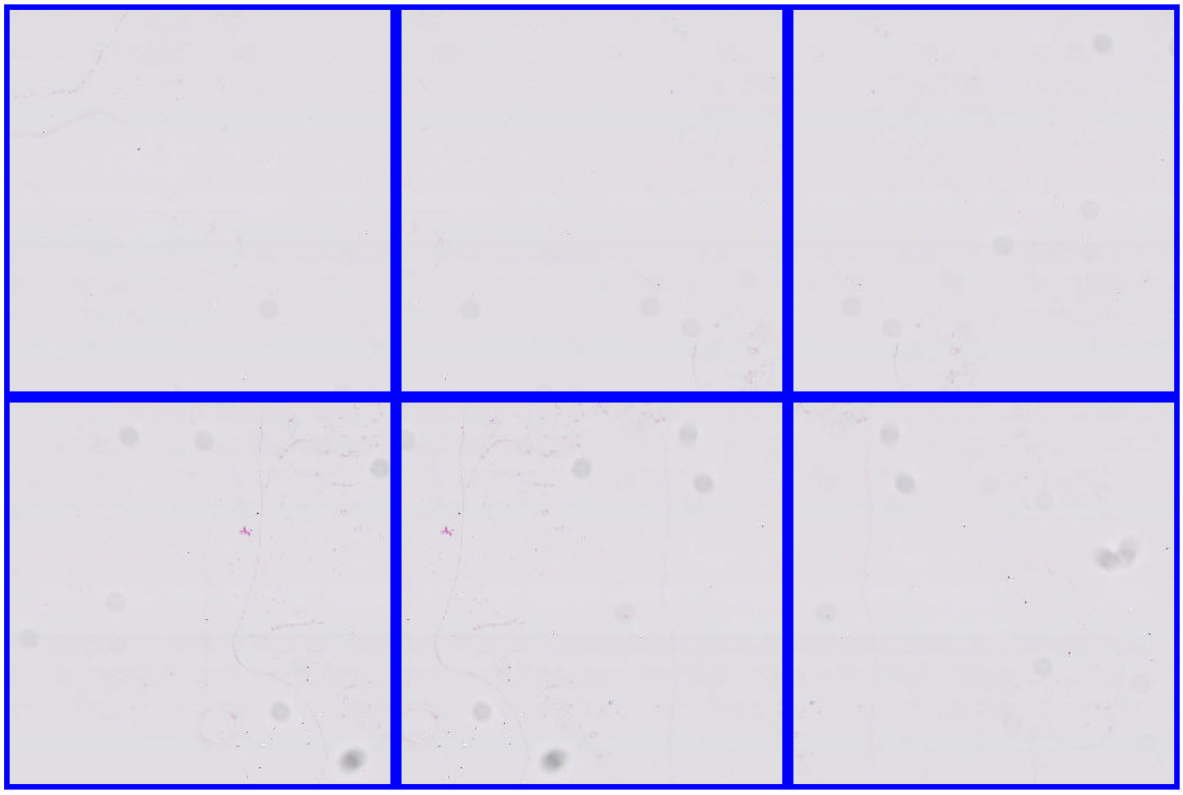
```
blockSize = [512 512];
overlapPct = 0.5;
blockOffsets = round(blockSize.*overlapPct);
bls = selectBlockLocations(bim,...
      'BlockSize', blockSize,...
      'BlockOffSets', blockOffsets,...
      'ExcludeIncompleteBlocks', true);
```

Create the `blockedImageDatastore`.

```
bimds = blockedImageDatastore(bim, "BlockLocationSet", bls);
```

Display the overlapping blocks.

```
bimds.ReadSize = 6;
blocks = read(bimds);
montage(blocks, "BorderSize", 5, "BackgroundColor", 'b');
```



Create blockedImageDatastore using Coarse Level Mask

Create a blockedImage.

```
bim = blockedImage('tumor_091R.tif');
```

Create a mask at the coarsest level.

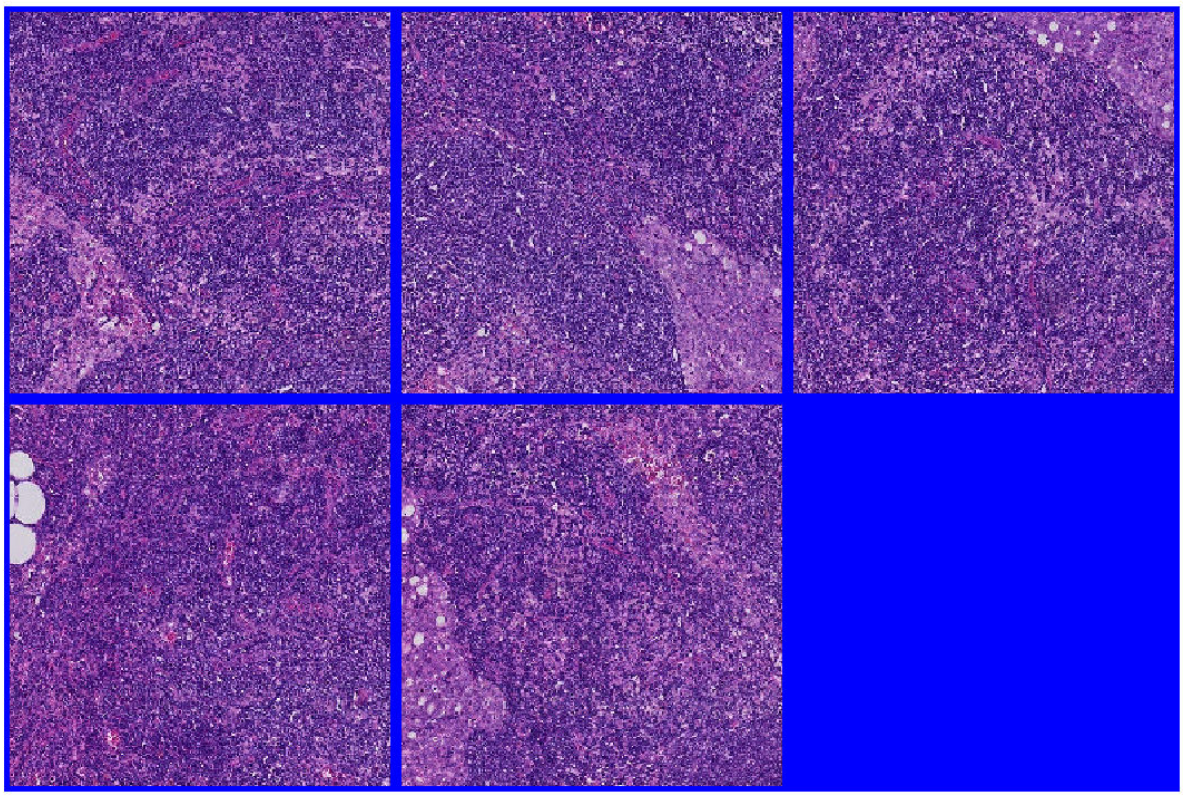
```
bmask = apply(bim, @(bs)~imbinarize(im2gray(bs.Data)),"Level",3);
```

Create a `blockedImageDatastore` for blocks which have at least 90% pixels 'on' in the stained region as defined by the mask.

```
mbls = selectBlockLocations(bim,...
     'Levels', 1, ...
     'Masks', bmask, 'InclusionThreshold', 0.90,...
     'BlockSize', [256 256]);
bimds = blockedImageDatastore(bim, 'BlockLocationSet', mbls);
```

Read blocks and display them.

```
bimds.ReadSize = 5;
blocks = read(bimds);
montage(blocks, "BorderSize", 5, "BackgroundColor", 'b')
```



Create Categorical Data from Labelled Numeric Data

Create blocked images from numeric and labeled data.

```
bim = blockedImage('yellowlily.jpg', 'BlockSize', [512 512]);
lbim = blockedImage('yellowlily-segmented.png', 'BlockSize', [512 512]);
```

Create `blockedImageDatastore` objects for each blocked image.

```
bimds = blockedImageDatastore(bim);
lbimds = blockedImageDatastore(lbim);
```

Transform the labeled numeric data into categorical data.

```
catbimds = transform(lbimds,...
      @(bs){categorical(bs{1}, [0,1, 2, 3], ["Unknown", "Flower", "Leaf", "Background"])});
```

Combine the original `blockedImageDatastore` with the categorical datastore.

```
cbimds = combine(bimds,catbimds);
```

Read data from the datastore and display it. The first cell is image data, second is categorical labels.

```
data = read(cbimds);
imshow(labeloverlay(data{1},data{2}));
```

See Also

`blockedImage` | `blockLocationSet` | `selectBlockLocations`

Introduced in R2021a

countEachLabel

Counts number of pixel labels for each class

Syntax

```
counts = countEachLabel(bimds)
counts = countEachLabel( ___ ,Name,Value)
```

Description

`counts = countEachLabel(bimds)` counts the occurrence of each pixel label in all the blocks represented by the blocked image datastore `bimds`.

`counts = countEachLabel( ___ ,Name,Value)` specifies additional parameters.

If `bimds` contains categorical data, `countEachLabel` obtains the class names from the categories specified in the `InitialValue` property of the first blocked image. In this case, do not specify values for the `'Classes'` and `'PixelLabelIDs'` parameters. If `bimds` contains numeric data, you must provide values for the `'Classes'` and `'PixelLabelIDs'` parameters.

Examples

Count Pixel Labels from Numeric Data

Create a blocked image from a sample label image.

```
label_bim = blockedImage('yellowlily-segmented.png', 'BlockSize', [512 512]);
```

Create a blocked image datastore from the blocked image.

```
lbimds = blockedImageDatastore(label_bim);
```

Count the labels in the blocked image datastore. Labels 0 and 3 both map to `'Background'`.

```
countEachLabel(lbimds, ...
      "Classes", ["Background", "Flower", "Leaf", "Background"],...
      "PixelLabelIDs", [0, 1, 2, 3])
```

```
ans=3×3 table
        Name         PixelCount    BlockPixelCount
    _____________    __________    _______________

    "Background"    2.3706e+06      3.1457e+06
    "Flower"        4.3349e+05      1.5729e+06
    "Leaf"          3.4159e+05      2.0972e+06
```

Input Arguments

`bimds` — Blocked image datastore

`blockedImageDatastore` object

Blocked image datastore, specified as a `blockedImageDatastore` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `countEachLabel(lbimds, ... "Classes", ["Background","Flower","Leaf","Background"],... "PixelLabelIDs",[0,1,2,3])`

`Classes` — Class names

string array | cell array of `char` vectors

Class names, specified as a string array or a cell array of `char` vectors.

Example: `"Classes",["Background","Flower","Leaf"]`

Data Types: `char` | `string` | `cell`

`PixelLabelIDs` — Values for each label

numeric array

Values for each label, specified as a numeric array of values with the same length as `'Classes'`. This parameter provides the mapping from numeric values to the label class.

Example: `"PixelLabelIDs",[0,1,2,3]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`UseParallel` — Use new or existing parallel pool

`false` (default) | `true`

Use new or existing parallel pool, specified as a logical scalar `true` or `false`. If no parallel pool is active, `countEachLabel` opens a new pool based on the default parallel settings. This syntax requires Parallel Computing Toolbox.

Data Types: `logical`

Output Arguments

`counts` — Counts the occurrence of each pixel label

table

Counts the occurrence of each pixel label in all blocks represented by the blocked image datastore, returned as a table that contains three variables.

Pixel Count Variables	Description
`Name`	Pixel label class name
`PixelCount`	Number of pixels of a given class in all blocks
`ImagePixelCount`	Total number of pixels in blocks that have an instance of the given class

Tips

You can use the label information returned by `countEachLabel` to calculate class weights for class balancing. For example, for labeled pixel data information in `tbl`:

- Uniform class balancing weights each class such that each contains a uniform prior probability:

  ```
  numClasses = height(tbl)
  prior = 1/numClasses;
  classWeights = prior./tbl.PixelCount
  ```

- Inverse frequency balancing weights each class such that underrepresented classes are given higher weight:

  ```
  totalNumberOfPixels = sum(tbl.PixelCount)
  frequency = tbl.PixelCount / totalNumberOfPixels;
  classWeights = 1./frequency
  ```

- Median frequency balancing weights each class using the median frequency. The weight for each class c is defined as `median(imageFreq)/imageBlockFreq(c)` where `imageBlockFreq(c)` is the number of pixels of a given class divided by the total number of pixels in image blocks that had an instance of the given class c.

  ```
  imageBlockFreq = tbl.PixelCount ./ tbl.BlockPixelCount
  classWeights = median(imageBlockFreq) ./ imageBlockFreq
  ```

You can pass the calculated class weights to a `pixelClassificationLayer`.

See Also

`blockedImage` | `blockedImageDatastore` | `pixelClassificationLayer`

Introduced in R2021a

hasdata

Returns `true` if more data is available in `blockedImageDatastore`

Syntax

```
tf = hasdata(bimds)
```

Description

`tf = hasdata(bimds)` returns a logical scalar, `true` or `false`, indicating the availability of data in the `blockedImageDatastore` `bimds`. Use `hasdata` in conjunction with the `read` function to read all the data within the datastore. Call `hasdata` before calling `read`.

Examples

Read Until All Data in Datastore Has Been Read

Create a blocked image from a sample image.

```
bim = blockedImage('tumor_091R.tif');
```

Create a `blockedImageDatastore` object from the blocked image, `bim`.

```
bimds = blockedImageDatastore(bim);
```

Read blocks from the `blockedImageDatastore` until there is no more data to read, that is, until `hasdata` returns `false`.

```
while hasdata(bimds)
    [data, info] = read(bimds);
    disp(info);
end
```

```
       BlockSub: [1 1 1]
          Start: [1 1 1]
            End: [1024 1024 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

       BlockSub: [1 2 1]
          Start: [1 1025 1]
            End: [1024 2048 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

       BlockSub: [1 3 1]
          Start: [1 2049 1]
```

```
        End: [1024 3072 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [1 4 1]
      Start: [1 3073 1]
        End: [1024 4096 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [1 5 1]
      Start: [1 4097 1]
        End: [1024 5120 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [1 6 1]
      Start: [1 5121 1]
        End: [1024 6144 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [2 1 1]
      Start: [1025 1 1]
        End: [2048 1024 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [2 2 1]
      Start: [1025 1025 1]
        End: [2048 2048 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [2 3 1]
      Start: [1025 2049 1]
        End: [2048 3072 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [2 4 1]
      Start: [1025 3073 1]
        End: [2048 4096 3]
      Level: 1
```

```
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [2 5 1]
      Start: [1025 4097 1]
        End: [2048 5120 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [2 6 1]
      Start: [1025 5121 1]
        End: [2048 6144 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [3 1 1]
      Start: [2049 1 1]
        End: [3072 1024 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [3 2 1]
      Start: [2049 1025 1]
        End: [3072 2048 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [3 3 1]
      Start: [2049 2049 1]
        End: [3072 3072 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [3 4 1]
      Start: [2049 3073 1]
        End: [3072 4096 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [3 5 1]
      Start: [2049 4097 1]
        End: [3072 5120 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
```

```
  BlockSize: [1024 1024 3]

   BlockSub: [3 6 1]
      Start: [2049 5121 1]
        End: [3072 6144 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [4 1 1]
      Start: [3073 1 1]
        End: [4096 1024 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [4 2 1]
      Start: [3073 1025 1]
        End: [4096 2048 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [4 3 1]
      Start: [3073 2049 1]
        End: [4096 3072 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [4 4 1]
      Start: [3073 3073 1]
        End: [4096 4096 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [4 5 1]
      Start: [3073 4097 1]
        End: [4096 5120 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [4 6 1]
      Start: [3073 5121 1]
        End: [4096 6144 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]
```

```
    BlockSub: [5 1 1]
       Start: [4097 1 1]
         End: [5120 1024 3]
       Level: 1
 ImageNumber: 1
  BorderSize: [0 0 0]
   BlockSize: [1024 1024 3]

    BlockSub: [5 2 1]
       Start: [4097 1025 1]
         End: [5120 2048 3]
       Level: 1
 ImageNumber: 1
  BorderSize: [0 0 0]
   BlockSize: [1024 1024 3]

    BlockSub: [5 3 1]
       Start: [4097 2049 1]
         End: [5120 3072 3]
       Level: 1
 ImageNumber: 1
  BorderSize: [0 0 0]
   BlockSize: [1024 1024 3]

    BlockSub: [5 4 1]
       Start: [4097 3073 1]
         End: [5120 4096 3]
       Level: 1
 ImageNumber: 1
  BorderSize: [0 0 0]
   BlockSize: [1024 1024 3]

    BlockSub: [5 5 1]
       Start: [4097 4097 1]
         End: [5120 5120 3]
       Level: 1
 ImageNumber: 1
  BorderSize: [0 0 0]
   BlockSize: [1024 1024 3]

    BlockSub: [5 6 1]
       Start: [4097 5121 1]
         End: [5120 6144 3]
       Level: 1
 ImageNumber: 1
  BorderSize: [0 0 0]
   BlockSize: [1024 1024 3]
```

Input Arguments

`bimds` — Blocked image datastore

`blockedImageDatastore` object

Blocked image datastore, specified as a `blockedImageDatastore` object.

Output Arguments

`tf` — Data availability

`true` | `false`

Data availability, returned as a logical scalar, `true` or `false`.

See Also

`blockedImageDatastore`

Introduced in R2021a

partition

Return partitioned part of `blockedImageDatastore`

Syntax

```
subbimds = partition(bimds,n,index)
```

Description

`subbimds = partition(bimds,n,index)` partitions the blocked image datastore `bimds` into the number of parts, specified by `n`, and returns the partition corresponding to the index `index`.

Examples

Partition blockedImageDatastore and Read Each Partition

Create a blocked image.

```
bim = blockedImage('tumor_091R.tif');
```

Create a blocked image datastore from the blocked image.

```
bimds = blockedImageDatastore(bim);
```

Partition the blocked image datastore into two partitions and create a new `blockedImageDatastore` object of the data in the first partition.

```
bimdsp1 = partition(bimds, 2, 1);
```

Read data from the first partition.

```
disp('Partition 1');
```

```
Partition 1
```

```
while hasdata(bimdsp1)
      [data, info] = read(bimdsp1);
      disp(info);
end
```

```
       BlockSub: [1 1 1]
          Start: [1 1 1]
            End: [1024 1024 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

       BlockSub: [1 2 1]
          Start: [1 1025 1]
            End: [1024 2048 3]
          Level: 1
```

```
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [1 3 1]
      Start: [1 2049 1]
        End: [1024 3072 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [1 4 1]
      Start: [1 3073 1]
        End: [1024 4096 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [1 5 1]
      Start: [1 4097 1]
        End: [1024 5120 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [1 6 1]
      Start: [1 5121 1]
        End: [1024 6144 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [2 1 1]
      Start: [1025 1 1]
        End: [2048 1024 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [2 2 1]
      Start: [1025 1025 1]
        End: [2048 2048 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [2 3 1]
      Start: [1025 2049 1]
        End: [2048 3072 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
```

```
      BlockSize: [1024 1024 3]

       BlockSub: [2 4 1]
          Start: [1025 3073 1]
            End: [2048 4096 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

       BlockSub: [2 5 1]
          Start: [1025 4097 1]
            End: [2048 5120 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

       BlockSub: [2 6 1]
          Start: [1025 5121 1]
            End: [2048 6144 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

       BlockSub: [3 1 1]
          Start: [2049 1 1]
            End: [3072 1024 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

       BlockSub: [3 2 1]
          Start: [2049 1025 1]
            End: [3072 2048 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

       BlockSub: [3 3 1]
          Start: [2049 2049 1]
            End: [3072 3072 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]
```

Partition the blocked image datastore and create a new `blockedImageDatastore` object of the data in the second partition.

```
bimdsp2 = partition(bimds, 2, 2);
```

Read data from the second partition.

```
disp('Partition 2');
```

```
Partition 2

while hasdata(bimdsp2)
      [data, info] = read(bimdsp2);
      disp(info);
end

       BlockSub: [3 4 1]
          Start: [2049 3073 1]
            End: [3072 4096 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

       BlockSub: [3 5 1]
          Start: [2049 4097 1]
            End: [3072 5120 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

       BlockSub: [3 6 1]
          Start: [2049 5121 1]
            End: [3072 6144 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

       BlockSub: [4 1 1]
          Start: [3073 1 1]
            End: [4096 1024 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

       BlockSub: [4 2 1]
          Start: [3073 1025 1]
            End: [4096 2048 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

       BlockSub: [4 3 1]
          Start: [3073 2049 1]
            End: [4096 3072 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

       BlockSub: [4 4 1]
          Start: [3073 3073 1]
            End: [4096 4096 3]
```

```
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [4 5 1]
      Start: [3073 4097 1]
        End: [4096 5120 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [4 6 1]
      Start: [3073 5121 1]
        End: [4096 6144 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [5 1 1]
      Start: [4097 1 1]
        End: [5120 1024 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [5 2 1]
      Start: [4097 1025 1]
        End: [5120 2048 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [5 3 1]
      Start: [4097 2049 1]
        End: [5120 3072 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [5 4 1]
      Start: [4097 3073 1]
        End: [5120 4096 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [5 5 1]
      Start: [4097 4097 1]
        End: [5120 5120 3]
      Level: 1
ImageNumber: 1
```

```
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

        BlockSub: [5 6 1]
           Start: [4097 5121 1]
             End: [5120 6144 3]
           Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]
```

Input Arguments

`bimds` — Blocked image datastore

`blockedImageDatastore` object

Blocked image datastore, specified as a `blockedImageDatastore` object.

`n` — Number of partitions

numeric scalar

Number of partitions, specified as a numeric scalar. To get an estimate for a reasonable value for `N`, use the `numpartitions` function.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`index` — Partition to read

numeric scalar

Partition to read, specified as a numeric scalar.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

`subbimds` — Subset of datastore

`blockedImageDatastore` object

Subset of datastore, returned as a `blockedImageDatastore` object.

See Also

`blockedImage` | `blockedImageDatastore`

Introduced in R2021a

read

Read data and metadata from `blockedImageDatastore`

Syntax

```
b = read(bimds)
[b,info] = read(bimds)
```

Description

`b = read(bimds)` returns the data extracted from the `blockedImageDatastore`, `bimds`. `b` is a cell array of block data. The value of the `ReadSize` property of the `blockedImageDatastore` object determines the length of the cell array.

`[b,info] = read(bimds)` also returns `info`, a structure containing information about where the data was extracted from the `blockedImageDatastore`.

Examples

Read Data and Metadata from blockedImageDatastore

Create a blocked image.

```
bim = blockedImage('tumor_091R.tif');
```

Create a `blockedImageDatastore` from the blocked image.

```
bimds = blockedImageDatastore(bim);
```

Read data and metadata from the `blockedImageDatastore`. Display the metadata.

```
while hasdata(bimds)
      [data, info] = read(bimds);
      disp(info);
end
```

```
       BlockSub: [1 1 1]
          Start: [1 1 1]
            End: [1024 1024 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]

       BlockSub: [1 2 1]
          Start: [1 1025 1]
            End: [1024 2048 3]
          Level: 1
    ImageNumber: 1
     BorderSize: [0 0 0]
      BlockSize: [1024 1024 3]
```

```
   BlockSub: [1 3 1]
      Start: [1 2049 1]
        End: [1024 3072 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [1 4 1]
      Start: [1 3073 1]
        End: [1024 4096 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [1 5 1]
      Start: [1 4097 1]
        End: [1024 5120 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [1 6 1]
      Start: [1 5121 1]
        End: [1024 6144 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [2 1 1]
      Start: [1025 1 1]
        End: [2048 1024 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [2 2 1]
      Start: [1025 1025 1]
        End: [2048 2048 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [2 3 1]
      Start: [1025 2049 1]
        End: [2048 3072 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [2 4 1]
```

```
      Start: [1025 3073 1]
        End: [2048 4096 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [2 5 1]
      Start: [1025 4097 1]
        End: [2048 5120 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [2 6 1]
      Start: [1025 5121 1]
        End: [2048 6144 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [3 1 1]
      Start: [2049 1 1]
        End: [3072 1024 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [3 2 1]
      Start: [2049 1025 1]
        End: [3072 2048 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [3 3 1]
      Start: [2049 2049 1]
        End: [3072 3072 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [3 4 1]
      Start: [2049 3073 1]
        End: [3072 4096 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [3 5 1]
      Start: [2049 4097 1]
        End: [3072 5120 3]
```

```
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [3 6 1]
      Start: [2049 5121 1]
        End: [3072 6144 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [4 1 1]
      Start: [3073 1 1]
        End: [4096 1024 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [4 2 1]
      Start: [3073 1025 1]
        End: [4096 2048 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [4 3 1]
      Start: [3073 2049 1]
        End: [4096 3072 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [4 4 1]
      Start: [3073 3073 1]
        End: [4096 4096 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [4 5 1]
      Start: [3073 4097 1]
        End: [4096 5120 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [4 6 1]
      Start: [3073 5121 1]
        End: [4096 6144 3]
      Level: 1
ImageNumber: 1
```

```
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [5 1 1]
      Start: [4097 1 1]
        End: [5120 1024 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [5 2 1]
      Start: [4097 1025 1]
        End: [5120 2048 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [5 3 1]
      Start: [4097 2049 1]
        End: [5120 3072 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [5 4 1]
      Start: [4097 3073 1]
        End: [5120 4096 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [5 5 1]
      Start: [4097 4097 1]
        End: [5120 5120 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]

   BlockSub: [5 6 1]
      Start: [4097 5121 1]
        End: [5120 6144 3]
      Level: 1
ImageNumber: 1
 BorderSize: [0 0 0]
  BlockSize: [1024 1024 3]
```

Input Arguments

`bimds` — Blocked image datastore

`blockedImageDatastore` object

Blocked image datastore, specified as a `blockedImageDatastore` object.

Output Arguments

`b` — Data from `blockedImageDatastore`

cell array

Data from `blockedImageDatastore`, returned as a cell array of block data of length `ReadSize`.

`info` — Metadata from `blockedImageDatastore`

scalar `struct`

Metadata from `blockedImageDatastore`, returned as a scalar `struct` with these fields. If `ReadSize>1`, these fields are arrays.

Field	Description
`Level`	Resolution level from which this data was read.
`ImageNumber`	Index into the `bimds.Images` array corresponding to the `blockedImage` from which this block was read.
`Start`	Subscripts of the first element in the block. If `BorderSize` is specified, this subscript can be out-of-bounds for edge blocks.
`End`	Subscripts of the last element in the block. If `BorderSize` is specified, this subscript can be out-of-bounds for edge blocks.
`Blocksub`	Block subscripts of the current block
`BorderSize`	Value of the `BorderSize` parameter
`BlockSize`	Value of the `BlockSize` parameter

See Also

`blockedImage` | `blockedImageDatastore`

Introduced in R2021a

readall

Read all data from the `blockedImageDatastore`

Syntax

```
b = readall(bimds)
```

Description

`b = readall(bimds)` read all the data from the `blockedImagesDatastore`, `bimds`. `readall` returns `b`, a cell array containing an element for every individual block. All the data returned from the individual reads should be able to be concatenated vertically. The datatype of the output should be the same as that of the `read` method.

Examples

Read All the Blocks from a blockedImageDatastore

Create a blocked image. The `blockedImage` object chunks the image into a 5-by-6 grid of 1024-by-1024 sized blocks, totaling 30 blocks.

```
bim = blockedImage('tumor_091R.tif');
```

Create a `blockedImageDatastore` from the blocked image.

```
bimds = blockedImageDatastore(bim);
```

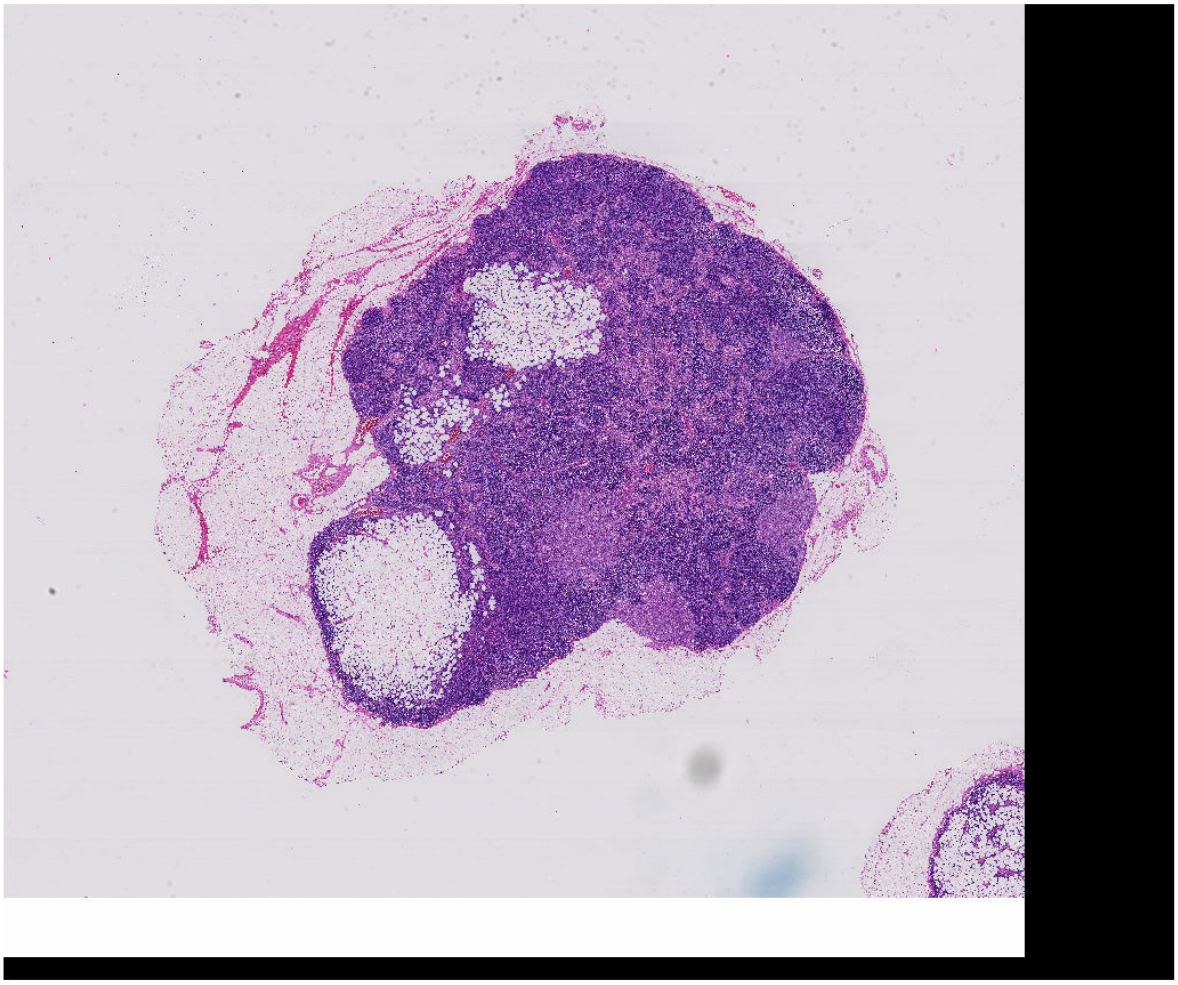
Read all the blocks from the `blockedImageDatastore`. The `readall` object function returns a cell array containing the 30 blocks.

```
b = readall(bimds)
```

```
b=30×1 cell array
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
    {1024x1024x3 uint8}
      ⋮
```

Display all the blocks.

```
montage(b)
```



Input Arguments

`bimds` — Blocked image datastore

`blockedImageDatastore` object

Blocked image datastore, specified as a `blockedImageDatastore` object.

Data Types: `blockedImageDatastore`

Output Arguments

`b` — Data from `blockedImageDatastore`

cell array

Data from `blockedImageDatastore`, returned as a cell array of block data of length `ReadSize`.

See Also

`blockedImage` | `blockedImageDatastore`

Introduced in R2021a

blockedNetwork

Create network with repeating block structure

Syntax

```
net = blockedNetwork(fun,numBlocks)
net = blockedNetwork(fun,numBlocks,'NamePrefix',namePrefix)
```

Description

`net = blockedNetwork(fun,numBlocks)` creates an uninitialized network, `net`, that consists of `numBlocks` blocks of layers connected sequentially. The function `fun` creates each block of layers.

This function requires Deep Learning Toolbox.

`net = blockedNetwork(fun,numBlocks,'NamePrefix',namePrefix)` adds the prefix `namePrefix` to all layer names in the network.

Examples

Create U-Net Style Encoder

Define a function that creates an array of layers. The first block has 32 filters in the convolution layers. The number of filters doubles in each successive block.

```
unetBlock = @(block) [
    convolution2dLayer(3,2^(5+block))
    reluLayer
    convolution2dLayer(3,2^(5+block))
    reluLayer
    maxPooling2dLayer(2,"Stride",2)];
```

Create a network that consists of four repeating blocks of layers. Add the prefix "encoder_" to all layer names in the network.

```
net = blockedNetwork(unetBlock,4,"NamePrefix","encoder_")
```

```
net =
  dlnetwork with properties:

         Layers: [20x1 nnet.cnn.layer.Layer]
    Connections: [19x2 table]
     Learnables: [16x3 table]
          State: [0x3 table]
     InputNames: {'encoder_Block1Layer1'}
    OutputNames: {'encoder_Block4Layer5'}
    Initialized: 0
```

Initialize network weights for input of size [224 224 3].

```
net = initialize(net,dlarray(zeros(224,224,3),"SSC"));
```

Display the network.

```
analyzeNetwork(net)
```

Create U-Net from Pretrained GoogLeNet

Create a GAN encoder network with four downsampling operations from a pretrained GoogLeNet network.

```
depth = 4;
[encoder,outputNames] = pretrainedEncoderNetwork('googlenet',depth);
```

Determine the input size of the encoder network.

```
inputSize = encoder.Layers(1).InputSize;
```

Determine the output size of the activation layers in the encoder network by creating a sample data input and then calling `forward`, which returns the activations.

```
exampleInput = dlarray(zeros(inputSize),'SSC');
exampleOutput = cell(1,length(outputNames));
[exampleOutput{:}] = forward(encoder,exampleInput,'Outputs',outputNames);
```

Determine the number of channels in the decoder blocks as the length of the third channel in each activation.

```
numChannels = cellfun(@(x) size(extractdata(x),3),exampleOutput);
numChannels = fliplr(numChannels(1:end-1));
```

Define a function that creates an array of layers for one decoder block.

```
decoderBlock = @(block) [
    transposedConv2dLayer(2,numChannels(block),'Stride',2)
    convolution2dLayer(3,numChannels(block),'Padding','same')
    reluLayer
    convolution2dLayer(3,numChannels(block),'Padding','same')
    reluLayer];
```

Create the decoder module with the same number of upsampling blocks as there are downsampling blocks in the encoder module.

```
decoder = blockedNetwork(decoderBlock,depth);
```

Create the U-Net network by connecting the encoder module and decoder module and adding skip connections.

```
net = encoderDecoderNetwork([224 224 3],encoder,decoder, ...
   'OutputChannels',3,'SkipConnections','concatenate')
```

```
net =
  dlnetwork with properties:

         Layers: [139x1 nnet.cnn.layer.Layer]
    Connections: [167x2 table]
```

```
     Learnables: [116x3 table]
          State: [0x3 table]
     InputNames: {'data'}
    OutputNames: {'encoderDecoderFinalConvLayer'}
    Initialized: 1
```

Display the network.

```
analyzeNetwork(net)
```

Input Arguments

`fun` — Function that creates blocks of layers

function

Function that creates blocks of layers, specified as a function with this signature:

```
block = fun(blockIndex)
```

- The input to `fun`, `blockIndex`, is an integer in the range [1, `numBlocks`].
- The output from `fun`, `block`, is a layer or layer array.

`numBlocks` — Number of blocks

positive integer

Number of blocks in the network, specified as a positive integer.

`namePrefix` — Prefix to all layer names

`""` (default) | string | character vector

Prefix to all layer names in the network, specified as a string or character vector.

Data Types: `char` | `string`

Output Arguments

`net` — Network with repeating block structure

`dlnetwork` object

Network with a repeating block structure, returned as a `dlnetwork` object.

Tips

- The `dlnetwork` returned by `blockedNetwork` is uninitialized and not ready for use with training or inference. To initialize the network, use the `initialize` function.
- Connect an encoder network to a decoder network using the `encoderDecoderNetwork` function.

See Also

`encoderDecoderNetwork`

Topics

“Create Modular Neural Networks”

“Get Started with GANs for Image-to-Image Translation”

Introduced in R2021a

blockLocationSet

List of block locations in large images

Description

A `blockLocationSet` object provides locations of blocks used for class balancing in semantic segmentation and object detection training workflows. It is used by `blockedImageDatastore` and `boxLabelDatastore` objects to specify block locations to read data.

Creation

You can create a `blockLocationSet` object in these ways.

- `selectBlockLocations` — Select blocks from an entire blocked image or within the masked region of a blocked image. Optionally specify the overlap and spacing between blocks.
- `balancePixelLabels` — Select blocks from labeled blocked images with pixel label data (requires Computer Vision Toolbox™). Use this function to perform class balancing in semantic segmentation workflows.
- `balanceBoxLabels` — Select blocks from labeled blocked images with bounding box data (requires Computer Vision Toolbox). Use this function to perform class balancing in object detection workflows.
- The `blockLocationSet` function described here. Use this function when you know the coordinates of blocks within the blocked images.

Syntax

```
locationSet = blockLocationSet(ImageNumber,BlockOrigin,BlockSize)
locationSet = blockLocationSet(ImageNumber,BlockOrigin,BlockSize,Levels)
```

Description

`locationSet = blockLocationSet(ImageNumber,BlockOrigin,BlockSize)` creates a `blockLocationSet` object that stores the locations `BlockOrigin` and size `BlockSize` of blocks to be read from a set of blocked image files indexed by `ImageNumber`.

`locationSet = blockLocationSet(ImageNumber,BlockOrigin,BlockSize,Levels)` also specifies the resolution level at which to read blocks from the blocked images.

Properties

`ImageNumber` — Image number
N-by-1 vector of positive integers

Image number of image files containing the read blocks, specified as an *N*-by-1 vector of positive integers, where *N* is the number of blocks specified by the `blockLocationSet`. Values cannot exceed the number of blocked images in the `blockedImageDatastore`.

Example: `[1 1 1 2]` specifies that a `blockedImageDatastore` reads four blocks total, with the first three blocks coming from the first `blockedImage` and the fourth block coming from the second `blockedImage` in the datastore.

Data Types: `double`

`BlockOrigin` — Block origin
n-by-2 numeric matrix

Block origin, specified as an *n*-by-*P* numeric matrix where *n* is the number of blocks specified by the `blockLocationSet` and *P* refers to the number of blocks. Each row specifies the `[x y]` coordinate of the upper left corner of a block.

Data Types: `double`

`BlockSize` — Block size
1-by-*N* vector of positive integers

Block size, specified as a 1-by-*N* vector of positive integers. The block size is the same for all blocks in the `blockLocationSet`. *N* matches the `NumDimensions` property of the `blockedImage` object.

`Levels` — Resolution levels
1 (default) | positive integer | vector of positive integers

Resolution level of each `blockedImage` in a `blockedImageDatastore`, specified as a positive integer or a vector of positive integers.

- When you specify `Levels` as a positive integer scalar, the `blockedImageDatastore` reads all blocks from the same resolution level.
- When you specify `Levels` as a vector of positive integers, each element indicates the resolution level at which the `blockedImageDatastore` reads blocks from the corresponding `blockedImage`. The length of `Levels` must equal the number of `blockedImage` objects in the `blockedImageDatastore`.

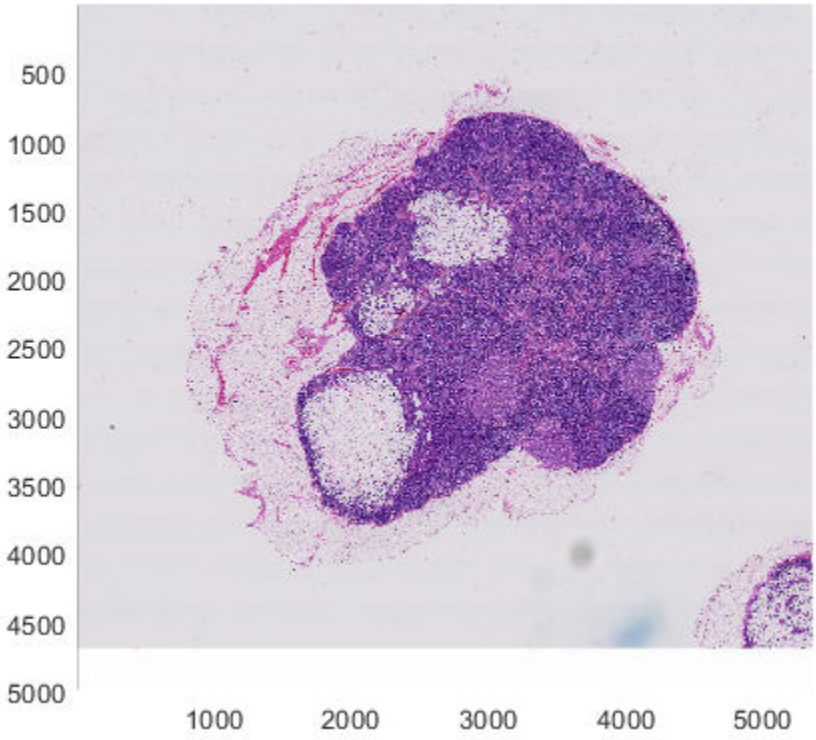
Example: `[1 1 2 2 1]` specifies that a `blockedImageDatastore` containing five `blockedImage` objects reads blocks at the first resolution level from the first, second, and fifth `blockedImage` objects and blocks at the second resolution level from the third and fourth `blockedImage` objects.

Examples

Specify Locations to Select Blocks from Blocked Images

Create a blocked image.

```
bim = blockedImage('tumor_091R.tif');
bigimageshow(bim)
```



Since all the blocks are from the same image, `imageNumber` is 1 for all the blocks. For resolution level, choose the finest resolution. Specify block locations in x,y coordinates.

```
imageNumber = [1, 1, 1, 1]';
levels = 1;
xyLocations = [[20 30 1]; [30 40 1]; [40 50 1]; [50 60 1]]
```

```
xyLocations = 4×3

    20    30     1
    30    40     1
    40    50     1
    50    60     1
```

```
blockSize = [300,300, 3];
locationSet = blockLocationSet(imageNumber,xyLocations,blockSize,levels);
```

Use the block location set to create a `blockedImageDatastore` containing just the specified blocks.

```
bimds = blockedImageDatastore(bim,'BlockLocationSet',locationSet);
```

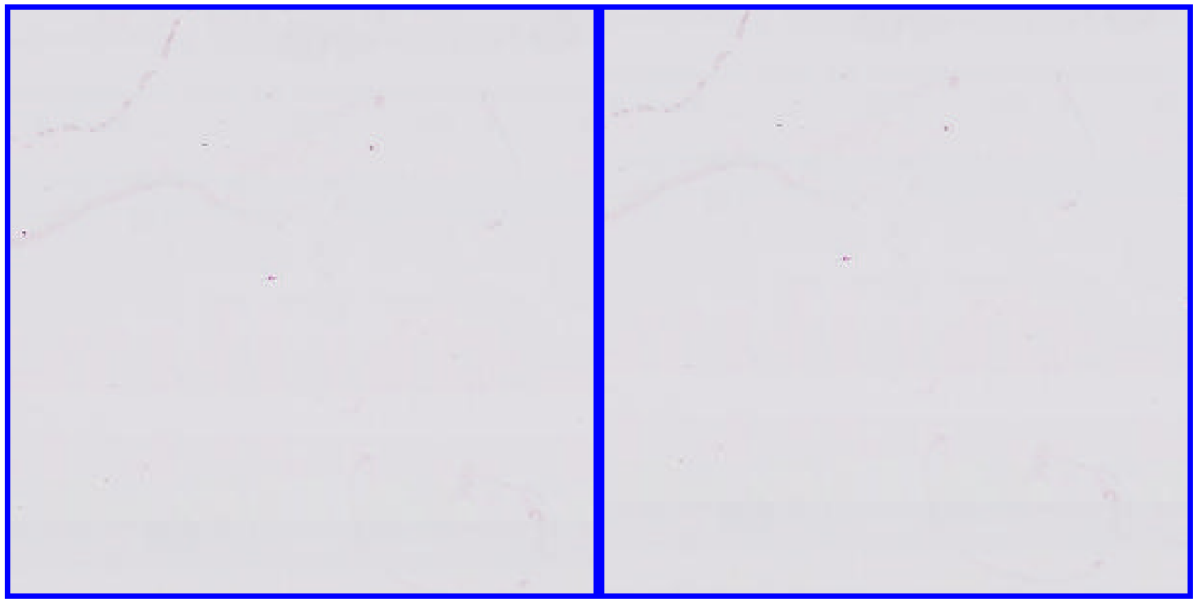
Read two blocks at a time from the `blockedImageDatastore`.

```
bimds.ReadSize = 2;
while hasdata(bimds)
    blocks = read(bimds)
end
```

```
blocks=2×1 cell array
    {300×300×3 uint8}
    {300×300×3 uint8}


blocks=2×1 cell array
    {300×300×3 uint8}
    {300×300×3 uint8}
```

Display the read blocks.

```
montage(blocks,'BorderSize',5,'BackgroundColor','b');
```



Tips

- The `blockLocationSet` object does not read or store data from blocked image files.

See Also

`blockedImage` | `blockedImageDatastore` | `boxLabelDatastore` | `balancePixelLabels` | `balanceBoxLabels` | `selectBlockLocations`

Introduced in R2020a

blockproc

Distinct block processing for image

Syntax

```
B = blockproc(A,[m n],fun)
B = blockproc(src_filename,[m n],fun)
B = blockproc(adapter,[m n],fun)
B = blockproc( ___ ,Name,Value)
```

Description

`B = blockproc(A,[m n],fun)` processes the input image `A` by applying the function `fun` to each distinct block of size `[m n]` and concatenating the results into the output image, `B`.

`B = blockproc(src_filename,[m n],fun)` processes the image with file name `src_filename`, reading and processing one block at a time. This syntax is useful for processing large images.

`B = blockproc(adapter,[m n],fun)` processes the source image specified by `adapter`, an `ImageAdapter` object. Use this syntax if you need a custom API for reading and writing to a particular image file format.

`B = blockproc( ___ ,Name,Value)` uses name-value pair arguments to control various aspects of the block behavior.

Examples

Create Thumbnail of Image

Read image into the workspace.

```
I = imread('pears.png');
```

Create block processing function.

```
fun = @(block_struct) imresize(block_struct.data,0.15);
```

Process the image, block-by-block.

```
I2 = blockproc(I,[100 100],fun);
```

Display the original image and the processed image.

```
figure;
imshow(I);
```

```
figure;
imshow(I2);
```

Set Pixels in 32-by-32 blocks to Standard Deviation

Create block processing function.

```
fun = @(block_struct) ...
   std2(block_struct.data) * ones(size(block_struct.data));
```

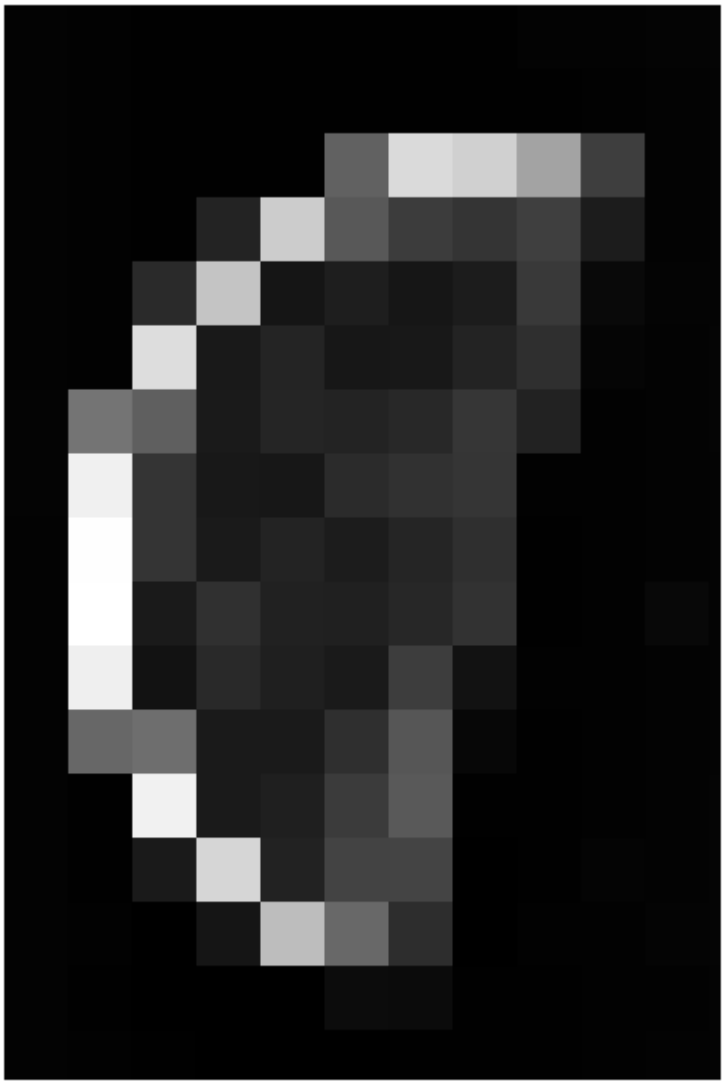
Perform the block processing operation, specifying the input image by filename.

```
I2 = blockproc('moon.tif',[32 32],fun);
```

Display the original image and the processed version.

```
figure;
imshow('moon.tif');
```

```
figure;
imshow(I2,[]);
```



Switch Red and Green Bands of RGB Image

Read image into the workspace.

```
I = imread('peppers.png');
```

Create block processing function.

```
fun = @(block_struct) block_struct.data(:,:,[2 1 3]);
```

Perform the block processing operation.

```
blockproc(I,[200 200],fun,'Destination','grb_peppers.tif');
```

Display original image and the processed image.

```
figure;
imshow('peppers.png');
```

```
figure;
imshow('grb_peppers.tif');
```

Convert Large TIFF Image into JPEG2000 Image

Note: To run this example, you must replace `"largeImage.tif"` with the name of your file.

Create block processing function.

```
fun = @(block_struct) block_struct.data;
```

Convert a TIFF image into a new JPEG2000 image. Replace `"largeImage.tif"` with the name of an actual image file.

```
blockproc("largeImage.tif",[1024 1024],fun,"Destination","New.jp2");
```

Input Arguments

A — Image to process

numeric array

Image to process, specified as a numeric array.

`src_filename` — Source file name

character vector | string scalar

Source file name, specified as a character vector or string scalar. Files must have one of these file types and must be named with one of the listed file extensions.

- TIFF (*.tif, *.tiff)
- JPEG2000 (*.jp2, *.jpf, *.jpx, *.j2c, *.j2k)

Data Types: `char` | `string`

`adapter` — Image adapter

`ImageAdapter` object

Image adapter, specified as an `ImageAdapter` object. An `ImageAdapter` is a user-defined class that provides `blockproc` with a common API for reading and writing to a particular image file format. For more information, see “Perform Block Processing on Image Files in Unsupported Formats”.

`[m n]` — Block size

2-element vector

Block size, specified as a 2-element vector. `m` is the number of rows and `n` is the number of columns in the block.

`fun` — Function handle

handle

Function handle, specified as a handle. The function must accept a *block_struct on page 1-355* as input and return an array, vector, or scalar. If `fun` returns empty, then `blockproc` does not generate any output and returns empty after processing all blocks.

For more information about function handles, see “Create Function Handle”.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `B = blockproc(A,[m,n],fun,BorderSize=[8 4])` creates a border with 8 rows and 4 columns around each block.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `B = blockproc(A,[m,n],fun,"BorderSize",[8 4])` creates a border with 8 rows and 4 columns around each block.

`Destination` — Destination

character vector | string scalar | `ImageAdapter` object

Destination for the output, specified as one of the following. When you specify the `Destination` argument, `blockproc` does not return the processed image as an output argument, but instead writes the output to the destination file.

- A character vector or string scalar with a destination file name. Files must have one of these file types and must be named with one of the listed file extensions.

- TIFF (*.tif, *.tiff)
- JPEG2000 (*.jp2, *.j2c, *.j2k)

If a file with this name exists, then it is overwritten.

- An `ImageAdapter` object, which provides a common API for reading and writing to a particular image file format. For more information, see “Perform Block Processing on Image Files in Unsupported Formats”.

The `Destination` argument is useful when you expect your output to be too large to fit into memory. It provides a workflow for file-to-file image processing for arbitrarily large images.

Note You cannot request an output argument when you specify the `Destination` argument.

`BorderSize` — Border size

`[0 0]` (default) | 2-element vector of positive integers

Number of border pixels to add to each block, specified as a 2-element vector of positive integers, of the form `[v h]`. The function adds `v` rows above and below each block and `h` columns left and right of each block. The size of each resulting block is:

```
[m+2*v, n+2*h]
```

By default, the function automatically removes the border from the result of `fun`. See the `TrimBorder` argument for more information.

The function pads blocks with borders extending beyond the image edges with zeros.

`PadPartialBlocks` — Pad partial blocks

`false` (default) | `true`

Pad partial blocks to make them full-sized, specified as `false` or `true`. Partial blocks arise when the image size is not exactly divisible by the block size. If they exist, partial blocks lie along the right and bottom edge of the image.

When set to `true`, `blockproc` pads partial blocks to make them full-sized `m`-by-`n` blocks. The default is `false`, meaning that the function does not pad the partial blocks, but processes them as-is. `blockproc` uses zeros to pad partial blocks when necessary.

`PadMethod` — Pad method

`0` (default) | `"replicate"` | `"symmetric"` | numeric scalar

Method used to pad the image boundary, specified as one of the following.

Value	Description
`"replicate"`	Repeat border elements.
`"symmetric"`	Pad image with mirror reflections of itself.
numeric scalar	Pad image with a scalar value. By default, the image boundary is padded with the value `0`.

Data Types: `char` | `string`

`TrimBorder` — Remove border pixels
`true` (default) | `false`

Remove border pixels from the output of the user function, specified as `true` or `false`. When set to `true`, the `blockproc` function removes border pixels from the output of the user function, `fun`. The function removes `v` rows from the top and bottom of the output of `fun`, and `h` columns from the left and right edges. The `BorderSize` argument defines `v` and `h`.

`UseParallel` — Use parallel processing
`false` (default) | `true`

Use parallel processing, specified as `false` or `true`. If you have Parallel Computing Toolbox installed, when set to `true`, MATLAB automatically opens a parallel pool of workers on your local machine. `blockproc` runs the computation across the available workers. For more information, see "Parallel Block Processing on Large Image Files".

`DisplayWaitbar` — Display wait bar
`true` (default) | `false`

Display wait bar, specified as `true` or `false`. When set to `true`, `blockproc` displays a wait bar to indicate progress for long-running operations. To prevent `blockproc` from displaying a wait bar, set `DisplayWaitbar` to `false`.

Output Arguments

`B` — Processed image
numeric array

Processed image, returned as a numeric array.

More About

Block Struct

A *block struct* is a MATLAB structure that contains the block data and other information about the block. Fields in the *block struct* are:

Field	Description
`border`	2-element vector of the form `[v h]`. The `border` field specifies the size of the vertical and horizontal padding around the block of data. See the `BorderSize` argument for more information.
`blockSize`	2-element vector of the form `[rows cols]`. The `blockSize` field specifies the size of the block data. If a border has been specified, the size does not include the border pixels.
`data`	`m`-by-`n` or `m`-by-`n`-by-`p` array of block data.
`imageSize`	2-element vector of the form `[rows cols]`. The `imageSize` field specifies the full size of the input image.

Field	Description
`location`	2-element vector of the form `[row col]`. The `location` field specifies the position of the first pixel (minimum-row, minimum-column) of the block data in the input image. If a border has been specified, the location refers to the first pixel of the discrete block data, not the added border pixels.

Tips

- Choosing an appropriate block size can significantly improve performance. For more information, see “Block Size and Performance”.
- If the output image `B` is too large to fit into memory, then omit the output argument and instead use the `Destination` name-value pair argument to write the output to a file.
- `blockproc` can read BigTIFF images but has limited support for writing BigTIFF images to file. If you write an image to file, then `blockproc` automatically selects the file type according to the size of the file. If the image is less than or equal to 4.0 Gb, then `blockproc` saves the image as a standard TIFF image. If the size of the file is larger than 4.0 Gb, then `blockproc` saves the image as a BigTIFF image.

  `blockproc` does not provide an argument that enables you to specify the file type as BigTIFF when the file size is less than or equal to 4.0 Gb. If you want to write a small image as a BigTIFF file, then specify a custom image adapter using the `adapter` argument. For more information, see TIFF, BigTIFF, and blockproc.
- To determine whether a written TIFF file is standard TIFF or BigTIFF, query the image format signature using the `imfinfo` function:

  ```
  tiffinfo = imfinfo(Destination);
  tiffformat = tiffinfo.FormatSignature
  ```

  If the last nonzero value of `tiffformat` is 42, then the file is in the standard TIFF format. If the last nonzero value is 43, then the file is in the BigTIFF format.

Extended Capabilities

Automatic Parallel Support

Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

Usage notes and limitations:

- This function supports automatic parallel processing (requires Parallel Computing Toolbox). To run in parallel, specify the `UseParallel` argument as `true`. For more information, see “Parallel Block Processing on Large Image Files”.
- Control parallel behavior with the parallel preferences, including scaling up to a cluster. See `parpool` for information on configuring your parallel environment.
- To run in parallel, this function requires a parallel pool with SPMD enabled.
- Parallel processing does not support an `adapter` source image.

See Also

`blockedImage` | `colfilt` | `ImageAdapter` | `nlfilter`

Topics
“Distinct Block Processing”
“Parallel Block Processing on Large Image Files”

Introduced in R2009b

boundarymask

Find region boundaries of segmentation

Syntax

```
mask = boundarymask(L)
mask = boundarymask(BW)
mask = boundarymask( ___ ,conn)
```

Description

`mask = boundarymask(L)` computes a mask that represents the region boundaries for the input label matrix `L`. The output, `mask`, is a logical image that is `true` at boundary locations and `false` at non-boundary locations.

`mask = boundarymask(BW)` computes the region boundaries for the input binary image `BW`.

`mask = boundarymask( ___ ,conn)` computes the region boundaries using a connectivity specified by `conn`.

Examples

Create Rasterized Grid of Region Boundaries

Read an image into the workspace.

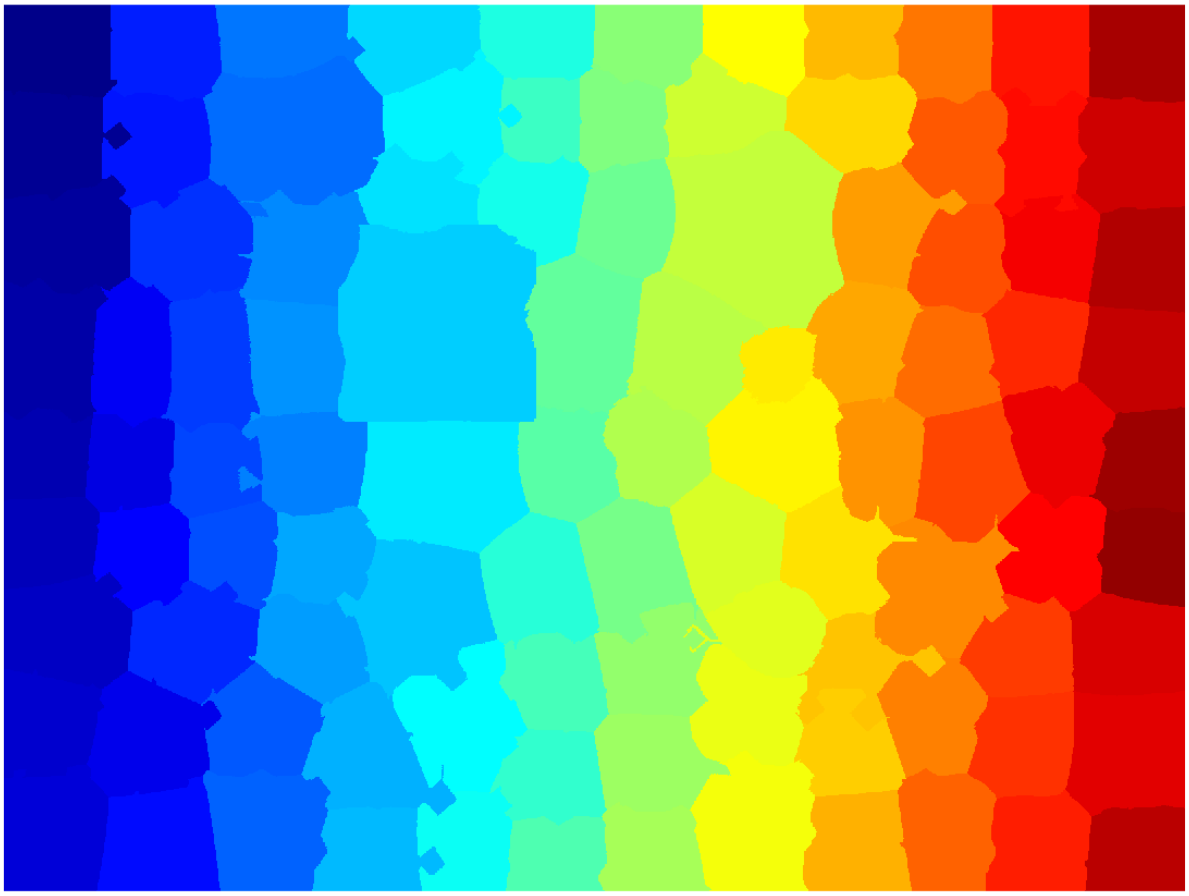
```
I = imread('kobi.png');
```

Create a superpixel representation of the image, returned as a label matrix.

```
L = superpixels(I,100);
```

Display the label matrix.

```
imshow(label2rgb(L))
```

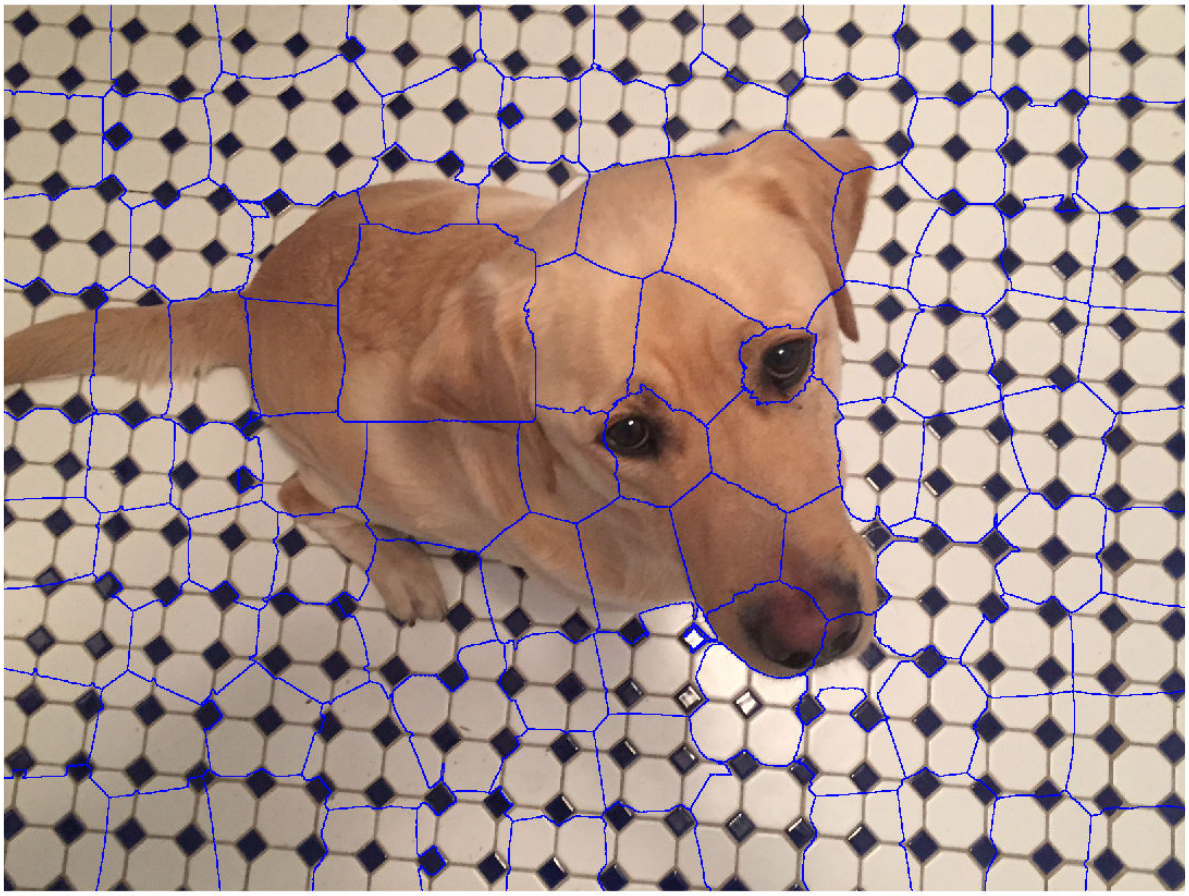


Find the region boundaries of the label matrix.

```
mask = boundarymask(L);
```

Display the boundary mask over the original image by using the `labeloverlay` function. The region boundaries of the label matrix appear as 1-pixel wide cyan lines.

```
imshow(labeloverlay(I,mask,'Transparency',0))
```



Input Arguments

L — Label matrix

2-D numeric matrix | 2-D logical matrix

Label matrix, specified as a 2-D numeric matrix of nonnegative numbers or a 2-D logical matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

BW — Binary image

numeric matrix | logical matrix

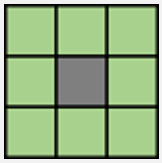
Binary image, specified as a numeric or logical matrix of the same size as L.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `logical`

conn — Pixel connectivity

8 (default) | 4

Pixel connectivity, specified as 4 or 8.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. The neighborhood of a pixel are the adjacent pixels in the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. The neighborhood of a pixel are the adjacent pixels in the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output Arguments

`mask` — Rasterized grid of region boundaries

2-D logical matrix

Rasterized grid of region boundaries, specified as a 2-D logical matrix of the same size as the input image. A pixel in `mask` is `true` when the corresponding pixel in the input image with value *P* has a neighboring pixel with a different value than *P*.

Data Types: `logical`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `boundarymask` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `boundarymask` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, the input argument `conn` must be a compile-time constant.

See Also

`superpixels` | `imoverlay` | `label2idx`

Introduced in R2016a

brisque

Blind/Referenceless Image Spatial Quality Evaluator (BRISQUE) no-reference image quality score

Syntax

```
score = brisque(A)
score = brisque(A,model)
```

Description

`score = brisque(A)` calculates the no-reference image quality score for image `A` using the Blind/Referenceless Image Spatial Quality Evaluator (BRISQUE). `brisque` compare `A` to a default model computed from images of natural scenes with similar distortions. A smaller score indicates better perceptual quality.

`score = brisque(A,model)` calculates the image quality score using a custom feature model.

Examples

Calculate BRISQUE Score Using Default Feature Model

Compute the BRISQUE score for a natural image and its distorted versions using the default model.

Read an image into the workspace. Create copies of the image with noise and blurring distortions.

```
I = imread('lighthouse.png');
Inoise = imnoise(I,'salt & pepper',0.02);
Iblur = imgaussfilt(I,2);
```

Display the images.

```
montage({I,Inoise,Iblur},'Size',[1 3],'ThumbnailSize',([]))
title('Original Image | Noisy Image | Blurry Image')
```

Calculate the BRISQUE score for each image using the default model, and display the score.

```
brisqueI = brisque(I);
fprintf('BRISQUE score for original image is %0.4f.\n',brisqueI)
```

```
BRISQUE score for original image is 20.6586.
```

```
brisqueInoise = brisque(Inoise);
fprintf('BRISQUE score for noisy image is %0.4f.\n',brisqueInoise)
```

```
BRISQUE score for noisy image is 52.6074.
```

```
brisqueIblur = brisque(Iblur);
fprintf('BRISQUE score for blurry image is %0.4f.\n',brisqueIblur)
```

```
BRISQUE score for blurry image is 47.7552.
```

The original undistorted image has the best perceptual quality and therefore the lowest BRISQUE score.

Calculate BRISQUE Score Using Custom Feature Model

Train a custom BRISQUE model from a set of quality-aware features and corresponding human opinion scores. Use the custom model to calculate a BRISQUE score for an image of a natural scene.

Save images from an image datastore. These images all have compression artifacts resulting from JPEG compression.

```
setDir = fullfile(toolboxdir('images'),'imdata');
imds = imageDatastore(setDir,'FileExtensions',{'.jpg'});
```

Specify the opinion score for each image. The following differential mean opinion score (DMOS) values are for illustrative purposes only. They are not real DMOS values obtained through experimentation.

```
opinionScores = 100*rand(1,size(imds.Files,1));
```

Create the custom model of quality-aware features using the image datastore and the opinion scores. Because the scores are random, the property values will vary.

```
model = fitbrisque(imds,opinionScores')
```

```
Extracting features from 38 images.
......
Completed 15 of 38 images.  Time: Calculating...
.....Training support vector regressor...

Done.

model =
  brisqueModel with properties:

             Alpha: [35x1 double]
              Bias: 58.1250
    SupportVectors: [35x36 double]
            Kernel: 'gaussian'
             Scale: 0.2767
```

Read an image of a natural scene that has the same type of distortion as the training images. Display the image.

```
I = imread('car1.jpg');
imshow(I)
```

Calculate the BRISQUE score for the image using the custom model. Display the score.

```
brisqueI = brisque(I,model);
fprintf('BRISQUE score for the image is %0.4f.\n',brisqueI)
```

```
BRISQUE score for the image is 72.7492.
```

Input Arguments

`A` — Input image

2-D grayscale image | 2-D RGB image

Input image, specified as a 2-D grayscale or RGB image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`model` — Custom model

`brisqueModel` object

Custom model trained on a set of quality-aware features, specified as a `brisqueModel` object. `model` is derived from natural scene statistics.

Output Arguments

`score` — No-reference image quality score
nonnegative scalar

No-reference image quality score, returned as a nonnegative scalar. The BRISQUE score is usually in the range [0, 100]. Lower values of `score` reflect better perceptual quality of image `A` with respect to the input `model`.

Data Types: `double`

Algorithms

`brisque` predicts the BRISQUE score by using a support vector regression (SVR) model trained on an image database with corresponding differential mean opinion score (DMOS) values. The database contains images with known distortion such as compression artifacts, blurring, and noise, and it contains pristine versions of the distorted images. The image to be scored must have at least one of the distortions for which the model was trained.

References

[1] Mittal, A., A. K. Moorthy, and A. C. Bovik. "No-Reference Image Quality Assessment in the Spatial Domain." *IEEE Transactions on Image Processing*. Vol. 21, Number 12, December 2012, pp. 4695–4708.

[2] Mittal, A., A. K. Moorthy, and A. C. Bovik. "Referenceless Image Spatial Quality Evaluation Engine." Presentation at the 45th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, November 2011.

See Also

Functions
`fitbrisque` | `niqe` | `fitniqe` | `piqe`

Objects
`brisqueModel`

Topics
“Image Quality Metrics”

Introduced in R2017b

brisqueModel

Blind/Referenceless Image Spatial Quality Evaluator (BRISQUE) model

Description

A `brisqueModel` object encapsulates a model used to calculate the Blind/Referenceless Image Spatial Quality Evaluator (BRISQUE) perceptual quality score of an image. The object contains a support vector regressor (SVR) model.

Creation

You can create a `brisqueModel` object using the following methods:

- `fitbrisque` — Train a BRISQUE model containing a custom trained support vector regressor (SVR) model. Use this function if you do not have a pretrained model.
- The `brisqueModel` function described here. Use this function if you have a pretrained SVR model, or if the default model is sufficient for your application.

Syntax

```
m = brisqueModel
m = brisqueModel(alpha,bias,supportVectors,scale)
```

Description

`m = brisqueModel` creates a BRISQUE model object with default property values that are derived from the LIVE IQA image database [1] [2].

`m = brisqueModel(alpha,bias,supportVectors,scale)` creates a custom BRISQUE model and sets the `Alpha on page 1-0` , `Bias on page 1-0` , `SupportVectors on page 1-0` , and `Scale on page 1-0` properties. You must provide all four arguments to create a custom model.

Note It is difficult to predict good property values without running an optimization routine. Use this syntax only if you are creating a `brisqueModel` object using a pretrained SVR model with known property values.

Properties

`Alpha` — Coefficients obtained by solving dual problem

m-by-1 numeric vector

Coefficients obtained by solving the dual problem, specified as an *m*-by-1 numeric vector. The length of `Alpha` must match the number of support vectors (the number of rows of `SupportVectors on page 1-0`).

Example: `rand(10,1)`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`Bias` — Bias term in SVM model
`43.4582` (default) | numeric scalar

Bias term in SVM model, specified as a numeric scalar.

Example: `47.4`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`SupportVectors` — Support vectors
m-by-36 numeric vector

Support vectors, specified as an *m*-by-36 numeric vector. The number of rows, *m*, matches the length of `Alpha on page 1-0` .

Example: `rand(10,36)`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`Kernel` — Kernel function
`'gaussian'` (default)

This property is read-only.

Kernel function, specified as `'gaussian'`.

`Scale` — Kernel scale factor
`0.3210` (default) | numeric scalar

Kernel scale factor, specified as a numeric scalar. The scale factor divides predictor values in the SVR kernel.

Example: `0.25`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Examples

Create BRISQUE Model Object with Default Properties

```
model = brisqueModel
```

```
model =
  brisqueModel with properties:

             Alpha: [593x1 double]
              Bias: 43.4582
    SupportVectors: [593x36 double]
            Kernel: 'gaussian'
             Scale: 0.3210
```

Create BRISQUE Model Object with Custom Properties

Create a `brisqueModel` object using precomputed `Alpha`, `Bias`, `SupportVectors`, and `Scale` properties. Random initializations are shown for illustrative purposes only.

```
model = brisqueModel(rand(10,1),47,rand(10,36),0.25)
```

```
model =
  brisqueModel with properties:

             Alpha: [10x1 double]
              Bias: 47
    SupportVectors: [10x36 double]
            Kernel: 'gaussian'
             Scale: 0.2500
```

You can use the custom model to calculate the BRISQUE score for an image.

```
I = imread('lighthouse.png');
score = brisque(I,model)
```

```
score = 47
```

Algorithms

The support vector regressor (SVR) calculates regression scores for predictor matrix `X` as:

$$F = G(X,\texttt{SupportVectors}) \times \texttt{Alpha} + \texttt{Bias}$$

`G(X,SupportVectors)` is an *n*-by-*m* matrix of kernel products for *n* rows in `X` and *m* rows in `SupportVectors`. The SVR has 36 predictors, which determine the number of columns in `SupportVectors`.

The SVR computes a kernel product between vectors `x` and `z` using `Kernel` (`x/Scale`, `z/Scale`).

References

[1] Mittal, A., A. K. Moorthy, and A. C. Bovik. "No-Reference Image Quality Assessment in the Spatial Domain." *IEEE Transactions on Image Processing*. Vol. 21, Number 12, December 2012, pp. 4695–4708.

[2] Mittal, A., A. K. Moorthy, and A. C. Bovik. "Referenceless Image Spatial Quality Evaluation Engine." Presentation at the 45th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, November 2011.

See Also

Functions
`brisque` | `fitbrisque`

Objects
`niqeModel` | `CompactRegressionSVM`

Topics
“Image Quality Metrics”
“Train and Use No-Reference Quality Assessment Model”

Introduced in R2017b

burstinterpolant

Create high-resolution image from set of low-resolution burst mode images

Syntax

```
B = burstinterpolant(imds,tforms,scale)
B = burstinterpolant(images,tforms,scale)
```

Description

`B = burstinterpolant(imds,tforms,scale)` creates a high-resolution image, `B` from a set of low-resolution burst mode images stored as an `ImageDatastore` object, `imds`. `scale` specifies the magnification value for high-resolution image. The size of `B` is `scale` times the size of input images.

`B = burstinterpolant(images,tforms,scale)` creates a high-resolution image, `B` from a set of low-resolution burst mode images stored in cell array `images`. The size of `B` is `scale` times the size of input images.

Examples

Create High-Resolution Image from Burst Mode Images in Image Datastore

Specify the location of low-resolution burst mode images to be stored as an image datastore object. The input images are 2-D RGB images.

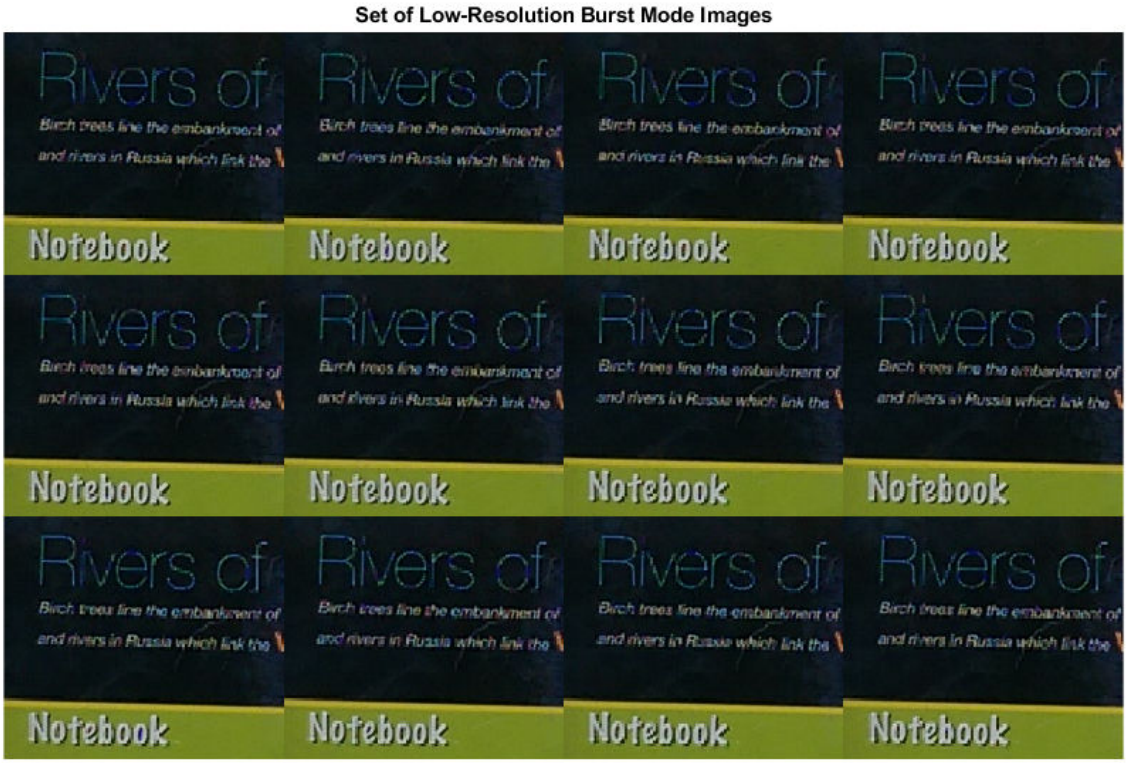
```
setDir = fullfile(toolboxdir('images'),'imdata','notebook');
```

Use the `imageDatastore` function to read and store the low-resolution burst mode images as an image datastore object.

```
imds = imageDatastore(setDir,'FileExtensions',{'.png'});
```

Display the images as a montage.

```
montage(imds)
title('Set of Low-Resolution Burst Mode Images')
```



Compute Geometric Transformation Parameters

To compute geometric transformation parameters, convert all the RGB images into lightness images by using `rgb2lightness` function. The burst mode lightness images are stored as an image datastore object.

```
imdsTransformed = transform(imds,@(x) rgb2lightness(x));
```

Read the first lightness image into the workspace and use it as the reference image for estimating geometric transformations.

```
refImg = read(imdsTransformed);
```

Get the optimal configuration parameters required for registration of the burst mode lightness images by using `imregconfig` function. Specify the image capture modality as `'monomodal'`.

```
[optimizer,metric] = imregconfig('monomodal');
```

Find the total number of images stored in the image datastore object by using `numpartitions` function.

```
numImages = numpartitions(imds);
```

Create an array of 2-D affine transformation object to store 2-D affine transformations of each low-resolution burst mode lightness image excluding the reference image. Set the number of rows in the transformation array as total number of images in the image datastore object minus one.

```
tforms = repmat(affine2d(),numImages-1,1);
```

Use the `imregtform` function to estimate the rigid geometric transformations for each low-resolution burst mode lightness image with respect to the reference image.

```
idx = 1;
while hasdata(imdsTransformed)
    movingImg = read(imdsTransformed);
    tforms(idx) = imregtform(refImg,movingImg,'rigid',optimizer,metric);
    idx = idx + 1;
end
```

Construct High-Resolution Image

Specify the scale factor for generating the high-resolution image.

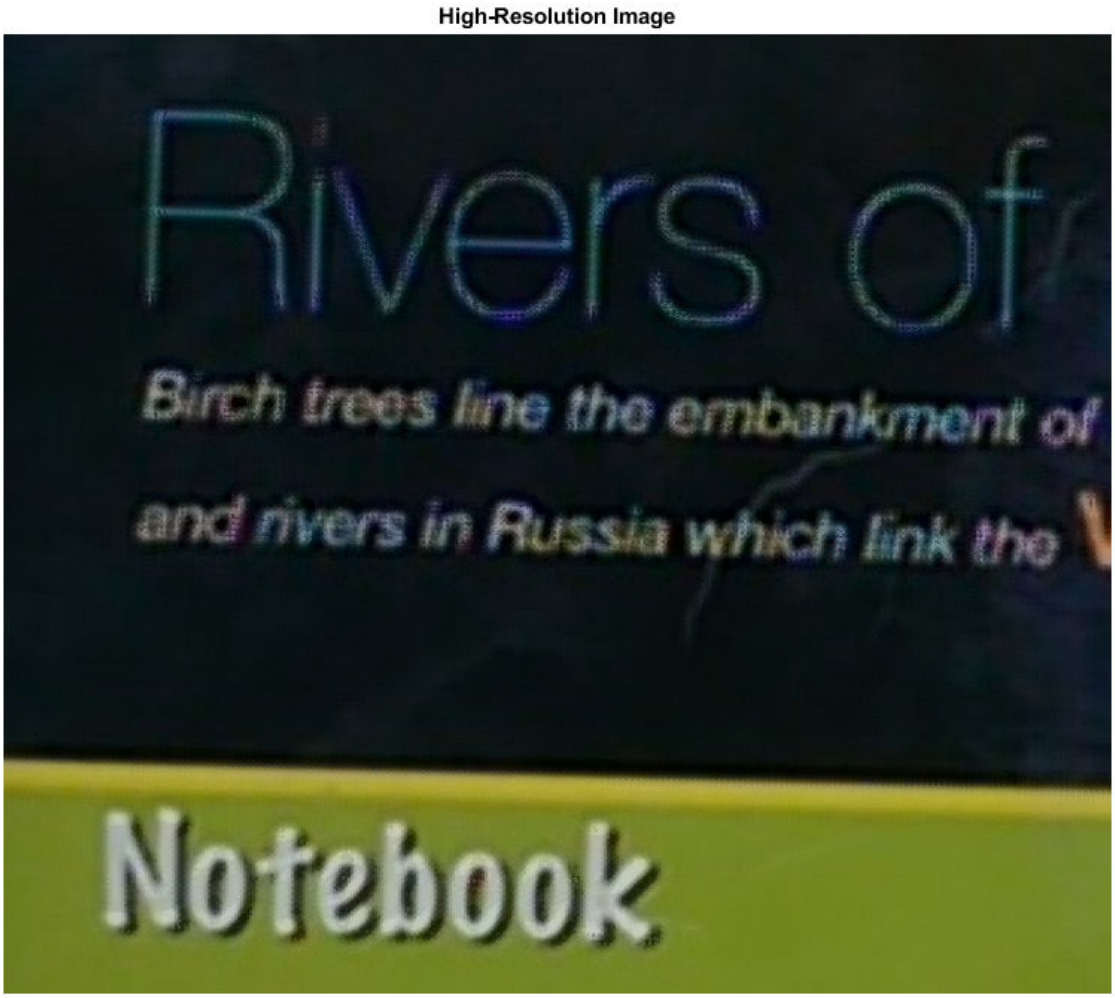
```
scale = 4;
```

Create the high-resolution image from the set of low-resolution burst mode RGB images. Specify the transformation parameter to robustly estimate the high-resolution pixel values.

```
B = burstinterpolant(imds,tforms,scale);
```

Display the high-resolution image.

```
figure('WindowState','maximized')
imshow(B)
title ('High-Resolution Image')
```



Read a low-resolution burst mode RGB image from the image datastore and display its size.

```
Img = read(imds);
inputDim = [size(Img,1) size(Img,2)]
```

```
inputDim = 1×2

   161   186
```

Display the size of the high-resolution image. Because the scale factor is 4, the size of the high-resolution image is 4 times the size of the low-resolution burst mode RGB images.

```
outputDim = [size(B,1) size(B,2)]
```

```
outputDim = 1×2
```

644 744

Create High-Resolution Image from Cell Array of Burst Mode Images

Load cell array data containing the low-resolution burst mode image into the workspace. The input images are monomodal and 2-D RGB images.

```
load('LRData')
```

Display images in the cell array data as a montage.

```
montage(images,'Size',[2 4],'BackgroundColor',[1 1 1]);
title('Set of Low-Resolution Burst Mode Images')
```

Compute Geometric Transformation Parameters

To compute geometric transformation parameters, convert all the RGB images into lightness images by using `rgb2lightness` function.

```
imagesT = cellfun(@rgb2lightness,images,'UniformOutput',false);
```

Read the first lightness image into the workspace and use it as the reference image for estimating geometric transformations.

```
refImg = imagesT{1};
```

Get the optimal configuration parameters required for registration of the burst mode lightness images by using `imregconfig`. Specify the image capture modality as `'monomodal'`.

```
[optimizer,metric] = imregconfig('monomodal');
```

Find the total number of images stored in the cell array.

```
numImages = length(images);
```

Create an array of 2-D affine transformation object to store 2-D affine transformations of each low-resolution burst mode lightness image excluding the reference image. Set the number of rows in the transformation array as total number of images in the cell array minus one.

```
tforms = repmat(affine2d(),numImages-1,1);
```

Use the `imregtform` function to estimate the rigid geometric transformations for each low-resolution burst mode lightness image with respect to the reference image.

```
for i= 2:length(images)
    movingImg = imagesT{i};
    tforms(i-1) = imregtform(refImg,movingImg,'rigid',optimizer,metric);
end
```

Construct High-Resolution Image

Specify the scale factor for generating the high-resolution image.

```
scale = 3;
```

Construct the high-resolution image from the set of low-resolution burst mode RGB images. Specify the transformation parameter to robustly estimate the high-resolution pixel values.

```
B = burstinterpolant(images,tforms,scale);
```

Display the high-resolution image.

```
figure
imshow(B);
title ('High-Resolution Image')
```

Read a low-resolution burst mode RGB image from the cell array and display its size.

```
Img = images{1};
inputDim = [size(Img,1) size(Img,2)]
```

```
inputDim = 1×2

   154   265
```

Display the size of the high-resolution image. Because the scale factor is 3, the size of the high-resolution image is 3 times the size of the low-resolution burst mode images.

```
ouputDim = [size(B,1) size(B,2)]
```

```
ouputDim = 1×2

   462   795
```

Input Arguments

`imds` — Input image datastore

`ImageDatastore` object

Input image datastore, specified as an `ImageDatastore` object. The input image datastore contains multiple low-resolution burst mode images used for creating the high-resolution image output.

- Images in the input image datastore must be 2-D grayscale images of size *m*-by-*n* or 2-D RGB images of size *m*-by-*n*-by-3.
- All images in the input image datastore must be of the same size and data type.
- The number of images in the input image datastore must be greater than or equal to 2.

Data Types: `single` | `double` | `uint8` | `uint16`

`images` — Input images

k-by-1 cell array

Input images, specified as a *k*-by-1 cell array. `k` is the number of input images stored in the cell array. All the input images must have same size.

Data Types: `single` | `double` | `uint8` | `uint16`

`tforms` — Transformation parameter

`affine2d` object array

Transformation parameter, specified as an `affine2d` object array of size (*k*-1)-by-1 or 1-by-(*k*-1). *k* is the number of images in input `imds` or `images`.

`scale` — Resize factor

scalar greater than or equal to 1

Resize factor, specified as a scalar greater than or equal to 1.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`B` — High-resolution image

2-D grayscale image | 2-D RGB image

High-resolution image, returned as a 2-D grayscale image or 2-D RGB image. `B` is of the same data type as the input images. The size of `B` is the value of `scale` times the size of the images in input `imds` or `images`.

For example, let *L* be the value of `scale`, and *m*-by-*n* be the size of the low-resolution burst mode images. Then, the size of the high-resolution image is *mL*-by-*nL*.

Tips

- Compute `tforms` with respect to each input image using the `imregtform` function. The first image in the input can be used as the reference image for estimating rigid geometric transformations (rotations and translations only).
- Compute input arguments `optimizer` and `metric` in `imregtform` using `imregconfig` function. `optimizer` must be a `RegularStepGradientDescent` object and `metric` must be a `MeanSquares` object.
- To improve the high-resolution output, you can modify the input argument value of `RegularStepGradientDescent` optimizer object in `imregtform`. For more details about these modifications, see the properties of `RegularStepGradientDescent`.

Algorithms

The `burstinterpolant` function uses the inverse distance weighting method [1] to generate high-resolution image from a set of low-resolution burst mode images. The function predicts a high-resolution pixel value from a set of pixels in the low-resolution burst mode images, selected based on the transformation parameter. The use of transformation parameter `tforms` makes the pixel selection robust to any rigid geometric transformations (rotations and translations only).

Note

- If the input images are 2-D RGB images, estimate `tforms` from the lightness component. You can use the `rgb2lightness` function to compute lightness values from the RGB color values.

References

[1] Shepard, Donald. “A Two-Dimensional Interpolation Function for Irregularly-Spaced Data”, In *Proceedings of the 1968 23rd ACM National Conference*, 517-524. New York, NY: ACM, 1968.

See Also

`scatteredInterpolant` | `imregtform` | `imregconfig` | `rgb2lightness`

Introduced in R2019a

bwarea

Area of objects in binary image

Syntax

```
total = bwarea(BW)
```

Description

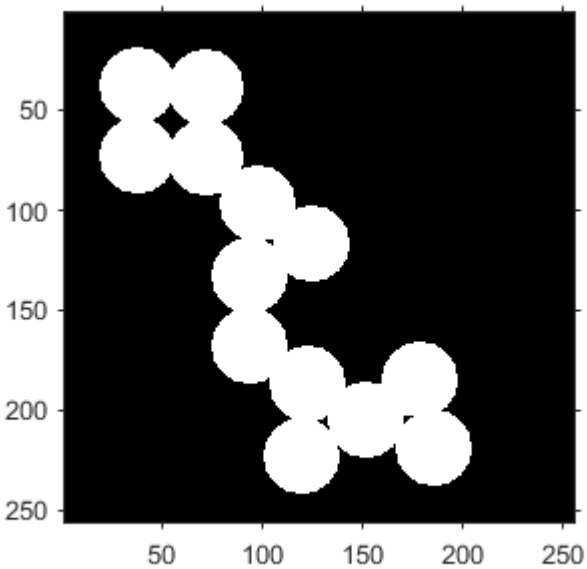
`total = bwarea(BW)` estimates the area of the objects in binary image `BW`. `total` is a scalar whose value corresponds roughly to the total number of `on` pixels in the image, but might not be exactly the same because different patterns of pixels are weighted differently.

Examples

Calculate Area of Objects in Binary Image

Read a binary image and display it.

```
BW = imread('circles.png');
imshow(BW)
```



Calculate the area of objects in the image.

```
bwarea(BW)
```

```
ans = 1.4187e+04
```

Input Arguments

BW — Binary image

2-D numeric matrix | 2-D logical matrix

Binary image, specified as a 2-D numeric or logical matrix. For numeric input, any nonzero pixels are considered to be `1` (`true`).

Example: `BW = imread('text.png'); L = bwlabel(BW);`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

total — Estimated number of on pixels

numeric scalar

Estimated number of `on` pixels in binary image `BW`, returned as a numeric scalar.

Data Types: `double`

Algorithms

`bwarea` estimates the area of all of the `on` pixels in an image by summing the areas of each pixel in the image. The area of an individual pixel is determined by looking at its 2-by-2 neighborhood. There are six different patterns, each representing a different area:

- Patterns with zero `on` pixels (area = 0)
- Patterns with one `on` pixel (area = 1/4)
- Patterns with two adjacent `on` pixels (area = 1/2)
- Patterns with two diagonal `on` pixels (area = 3/4)
- Patterns with three `on` pixels (area = 7/8)
- Patterns with all four `on` pixels (area = 1)

Each pixel is part of four different 2-by-2 neighborhoods. This means, for example, that a single `on` pixel surrounded by `off` pixels has a total area of 1.

References

[1] Pratt, William K., *Digital Image Processing*, New York, John Wiley & Sons, Inc., 1991, p. 634.

See Also

`bweuler` | `bwperim` | `bwferet`

Introduced before R2006a

bwareafilt

Extract objects from binary image by size

Syntax

```
BW2 = bwareafilt(BW,range)
BW2 = bwareafilt(BW,n)
BW2 = bwareafilt(BW,n,keep)
BW2 = bwareafilt( ___ ,conn)
```

Description

`BW2 = bwareafilt(BW,range)` extracts all connected components (objects) from the binary image `BW`, where the area of the objects is in the specified `range`, producing another binary image `BW2`. `bwareafilt` returns a binary image `BW2` containing only those objects that meet the criteria.

`BW2 = bwareafilt(BW,n)` keeps the `n` largest objects. In the event of a tie for `n`-th place, only the first `n` objects are included in `BW2`.

`BW2 = bwareafilt(BW,n,keep)` specifies whether to keep the `n` largest objects or the `n` smallest objects.

`BW2 = bwareafilt( ___ ,conn)` specifies the pixel connectivity that defines the objects.

Examples

Filter Binary Image by Area of Objects

Read image.

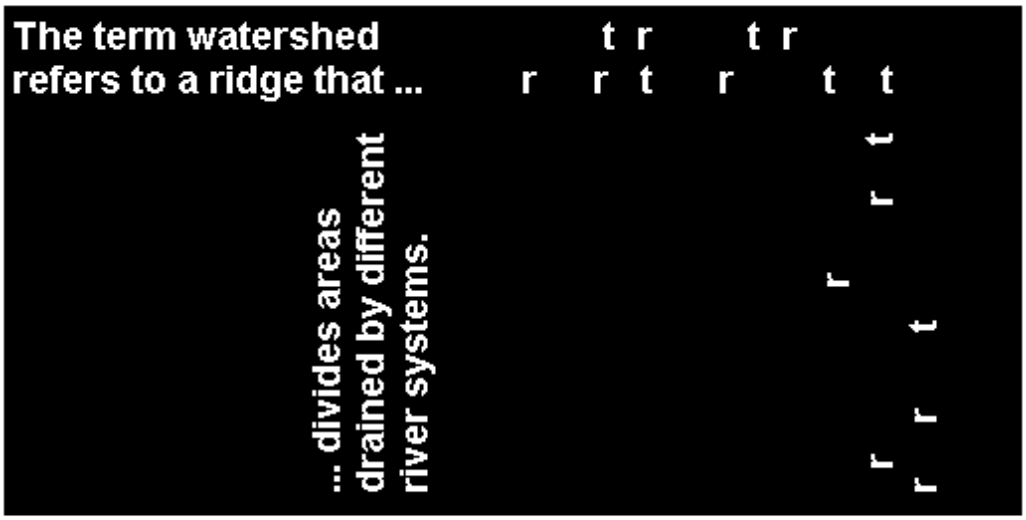
```
BW = imread('text.png');
```

Filter image, retaining only those objects with areas between 40 and 50.

```
BW2 = bwareafilt(BW,[40 50]);
```

Display the original image and filtered image side by side.

```
imshowpair(BW,BW2,'montage')
```



Filter Binary Image by Size of Objects

Read image.

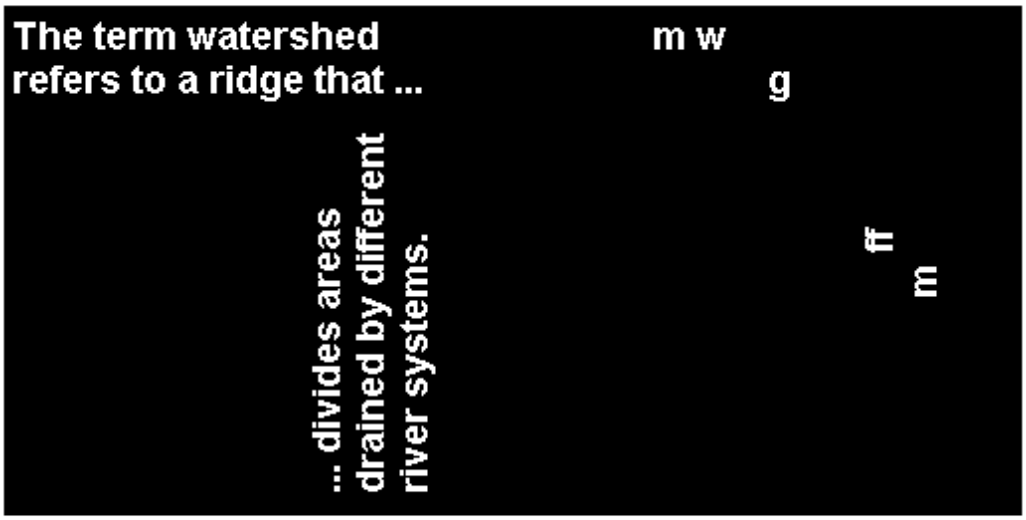
```
BW = imread('text.png');
```

Filter image, retaining only the 5 objects with the largest areas.

```
BW2 = bwareafilt(BW,5);
```

Display the original image and the filtered image side by side.

```
imshowpair(BW,BW2,'montage')
```



Input Arguments

`BW` — Image to be filtered

binary image

Image to be filtered, specified as a binary image.

Data Types: `logical`

`range` — Minimum and maximum areas

2-by-1 numeric vector

Minimum and maximum values of the area, specified as a 2-by-1 numeric vector of the form `[low high]`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`n` — Number of objects to include

numeric scalar

Number of objects to include when filtering image objects by size, specified as a numeric scalar.

Data Types: `double`

`keep` — Size of objects to include

`'largest'` (default) | `'smallest'`

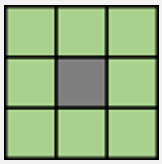
Size of objects to include in the output image, specified as `'largest'` or `'smallest'`. In the event of a tie for `n`-th place, `bwareafilt` includes only the first `n` objects.

Data Types: `char` | `string`

conn — Pixel connectivity

8 (default) | 4 | 3-by-3 matrix of 0s and 1s

Pixel connectivity, specified as one of these values.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.

Connectivity can also be defined in a more general way by specifying a 3-by-3 matrix of 0s and 1s. The 1-valued elements define neighborhood locations relative to the center element of `conn`. The matrix must be symmetric about its center element.

Data Types: `double` | `logical`

Output Arguments

BW2 — Filtered image

binary image

Filtered image, returned as a binary image of the same size and class as the input image `BW`.

See Also

`bwareaopen` | `bwconncomp` | `bwpropfilt` | `conndef` | `regionprops`

Topics

“Filter Images on Properties Using Image Region Analyzer App”

Introduced in R2014b

bwareaopen

Remove small objects from binary image

Syntax

```
BW2 = bwareaopen(BW,P)
BW2 = bwareaopen(BW,P,conn)
```

Description

`BW2 = bwareaopen(BW,P)` removes all connected components (objects) that have fewer than `P` pixels from the binary image `BW`, producing another binary image, `BW2`. This operation is known as an area opening.

`BW2 = bwareaopen(BW,P,conn)` removes all connected components, where `conn` specifies the desired connectivity.

Examples

Remove Objects in Image Containing Fewer Than 50 Pixels

Read binary image.

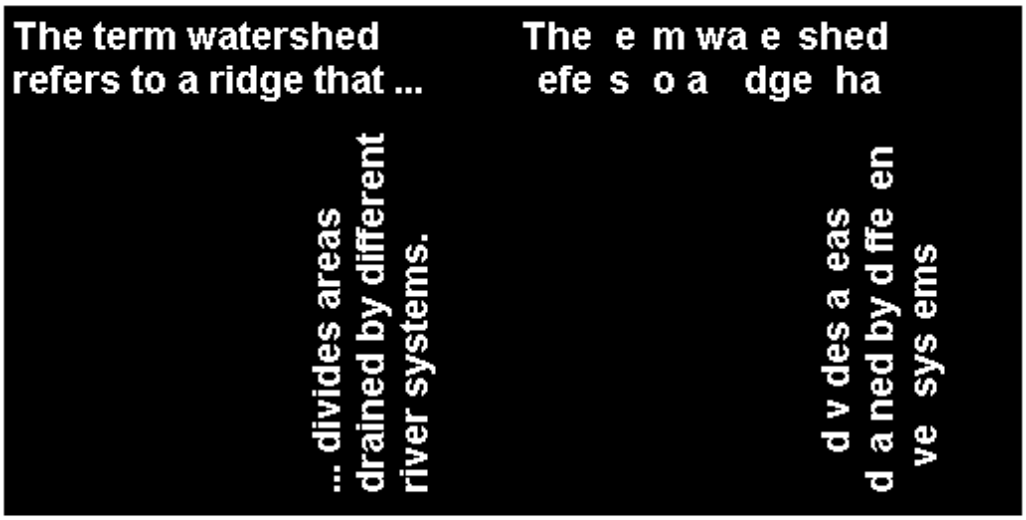
```
BW = imread('text.png');
```

Remove objects containing fewer than 50 pixels using `bwareaopen` function.

```
BW2 = bwareaopen(BW, 50);
```

Display original image next to morphologically opened image.

```
imshowpair(BW,BW2,'montage')
```



Input Arguments

BW — Binary image

logical array | numeric array

Binary image, specified as a logical or numeric array of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

P — Maximum number of pixels in objects

nonnegative integer

Maximum number of pixels in objects, specified as a nonnegative integer.

Example: `50`

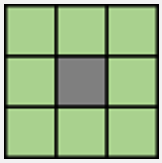
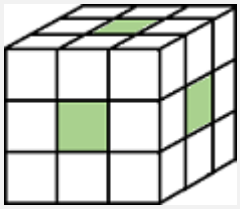
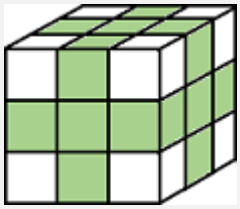
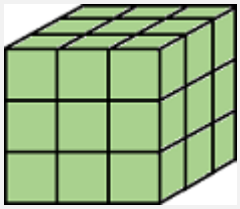
Data Types: `double`

conn — Pixel connectivity

4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 matrix of `0`s and `1`s

Pixel connectivity, specified as one of the values in this table. The default connectivity is 8 for 2-D images, and 26 for 3-D images.

Value	Meaning
Two-Dimensional Connectivities	

Value	Meaning	
4	Pixels are connected if their edges touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		
6	Pixels are connected if their faces touch. Two adjoining pixels are part of the same object if they are both on and are connected in: • One of these directions: in, out, left, right, up, and down	 Current pixel is center of cube.
18	Pixels are connected if their faces or edges touch. Two adjoining pixels are part of the same object if they are both on and are connected in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. Two adjoining pixels are part of the same object if they are both on and are connected in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `bwareaopen` uses the default value `conndef(ndims(BW),'maximal')`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the

center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Data Types: `double` | `logical`

Output Arguments

`BW2` — Area-opened image

logical array

Area-opened image, returned as a logical array of the same size as `BW`.

Algorithms

The basic steps are

1. Determine the connected components:

   ```
   CC = bwconncomp(BW, conn);
   ```

2. Compute the area of each component:

   ```
   S = regionprops(CC, 'Area');
   ```

3. Remove small objects:

   ```
   L = labelmatrix(CC);
   BW2 = ismember(L, find([S.Area] >= P));
   ```

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `bwareaopen` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- `BW` must be a 2-D binary image. N-D arrays are not supported.
- `conn` can only one of the two-dimensional connectivities (4 or 8) or a 3-by-3 matrix. The 3-D connectivities (6, 18, and 26) are not supported. Matrices of size 3-by-3-by-...-by-3 are not supported.
- `conn` must be a compile-time constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- `BW` must be a 2-D binary image. N-D arrays are not supported.
- `conn` must be one of the two-dimensional connectivities (4 or 8) or a 3-by-3 matrix. The 3-D connectivities (6, 18, and 26) are not supported. Matrices of size 3-by-3-by-...-by-3 are not supported.

- `conn` must be a compile-time constant.

See Also

`bwconncomp` | `conndef`

Introduced before R2006a

bwboundaries

Trace region boundaries in binary image

Syntax

```
B = bwboundaries(BW)
B = bwboundaries(BW,conn)
B = bwboundaries(BW,conn,options)
[B,L]= bwboundaries( ___ )
[B,L,n,A] = bwboundaries( ___ )
```

Description

`B = bwboundaries(BW)` traces the exterior boundaries of objects, as well as boundaries of holes inside these objects, in the binary image `BW`. `bwboundaries` also descends into the outermost objects (parents) and traces their children (objects completely enclosed by the parents). Returns `B`, a cell array of boundary pixel locations.

`B = bwboundaries(BW,conn)` traces the exterior boundaries of objects, where `conn` specifies the connectivity to use when tracing parent and child boundaries.

`B = bwboundaries(BW,conn,options)` traces the exterior boundaries of objects, where `options` is either `'holes'` or `'noholes'`, specifying whether you want to include the boundaries of holes inside other objects.

`[B,L]= bwboundaries( ___ )` returns a label matrix `L` where objects and holes are labeled.

`[B,L,n,A] = bwboundaries( ___ )` returns `n`, the number of objects found, and `A`, an adjacency matrix.

Examples

Overlay Region Boundaries on Image

Read grayscale image into the workspace.

```
I = imread('rice.png');
```

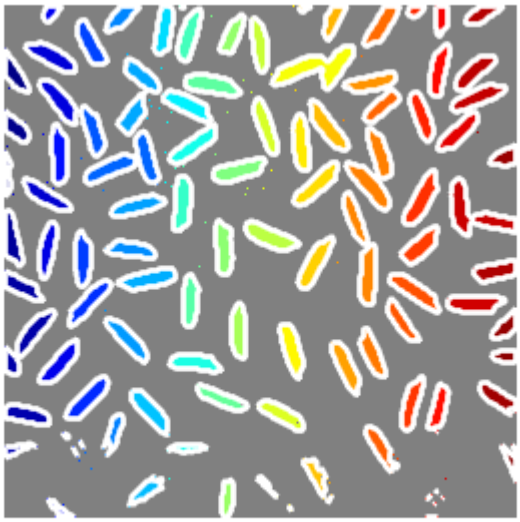
Convert grayscale image to binary image using local adaptive thresholding.

```
BW = imbinarize(I);
```

Calculate boundaries of regions in image and overlay the boundaries on the image.

```
[B,L] = bwboundaries(BW,'noholes');
imshow(label2rgb(L, @jet, [.5 .5 .5]))
hold on
for k = 1:length(B)
   boundary = B{k};
```

```
   plot(boundary(:,2), boundary(:,1), 'w', 'LineWidth', 2)
end
```



Overlay Region Boundaries on Image and Annotate with Region Numbers

Read binary image into the workspace.

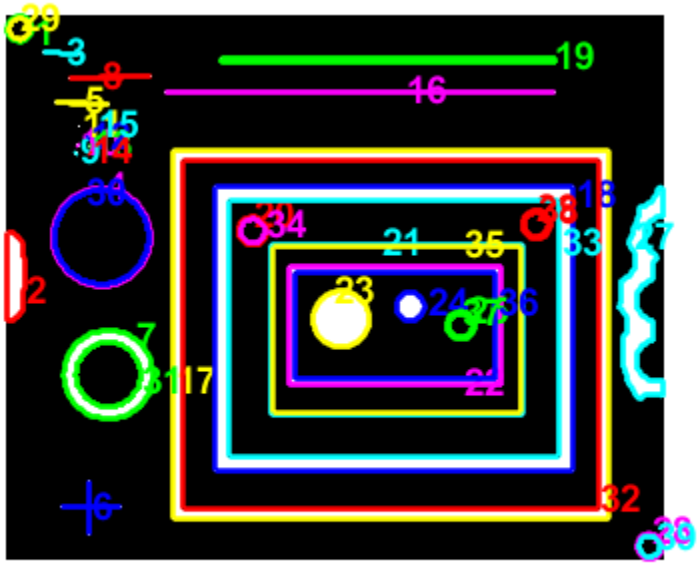
```
BW = imread('blobs.png');
```

Calculate boundaries of regions in the image.

```
[B,L,N,A] = bwboundaries(BW);
```

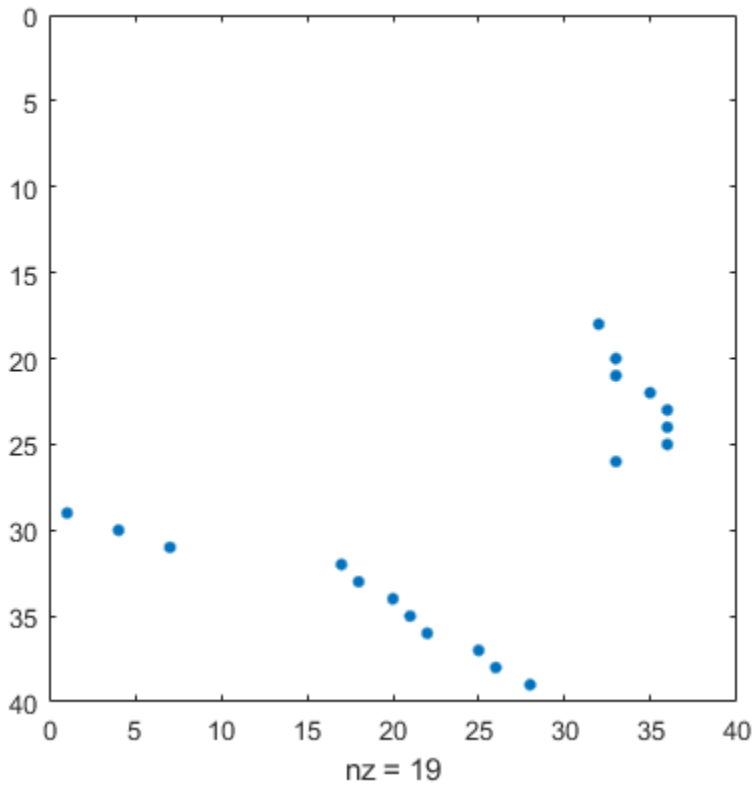
Display the image with the boundaries overlaid. Add the region number next to every boundary (based on the label matrix). Use the zoom tool to read individual labels.

```
imshow(BW); hold on;
colors=['b' 'g' 'r' 'c' 'm' 'y'];
for k=1:length(B),
  boundary = B{k};
  cidx = mod(k,length(colors))+1;
  plot(boundary(:,2), boundary(:,1),...
       colors(cidx),'LineWidth',2);

  %randomize text position for better visibility
  rndRow = ceil(length(boundary)/(mod(rand*k,7)+1));
  col = boundary(rndRow,2); row = boundary(rndRow,1);
  h = text(col+1, row-1, num2str(L(row,col)));
  set(h,'Color',colors(cidx),'FontSize',14,'FontWeight','bold');
end
```



Display the adjacency matrix using the `spy` function.

```
figure
spy(A);
```



Display Object Boundaries in Red and Hole Boundaries in Green

Read binary image into workspace.

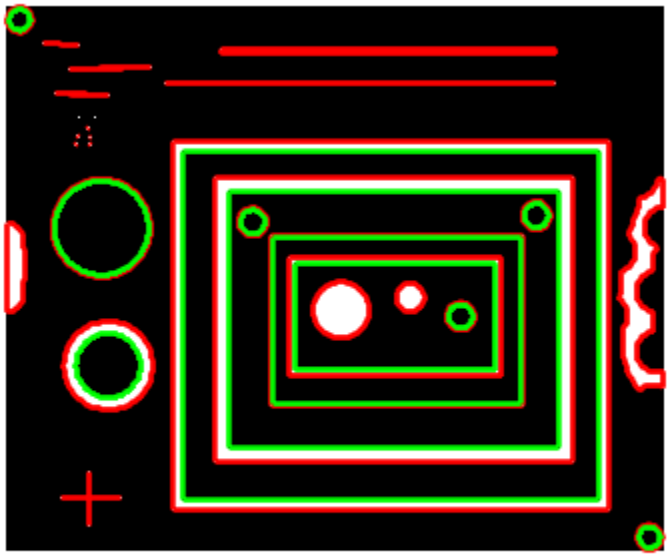
```
BW = imread('blobs.png');
```

Calculate boundaries.

```
[B,L,N] = bwboundaries(BW);
```

Display object boundaries in red and hole boundaries in green.

```
imshow(BW); hold on;
for k=1:length(B),
   boundary = B{k};
   if(k > N)
     plot(boundary(:,2), boundary(:,1), 'g','LineWidth',2);
   else
     plot(boundary(:,2), boundary(:,1), 'r','LineWidth',2);
   end
end
```



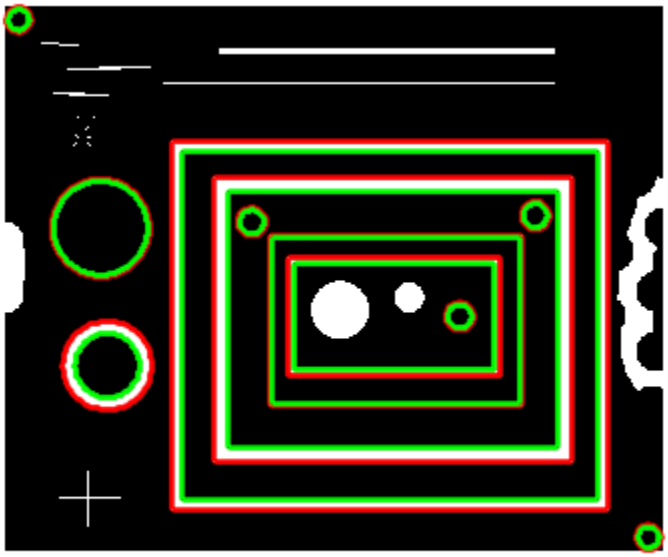
Display Parent Boundaries in Red and Holes in Green

Read image into workspace.

```
BW = imread('blobs.png');
```

Display parent boundaries in red and their holes in green.

```
[B,L,N,A] = bwboundaries(BW);
figure; imshow(BW); hold on;
% Loop through object boundaries
for k = 1:N
    % Boundary k is the parent of a hole if the k-th column
    % of the adjacency matrix A contains a non-zero element
    if (nnz(A(:,k)) > 0)
        boundary = B{k};
        plot(boundary(:,2),...
            boundary(:,1),'r','LineWidth',2);
        % Loop through the children of boundary k
        for l = find(A(:,k))'
            boundary = B{l};
            plot(boundary(:,2),...
                boundary(:,1),'g','LineWidth',2);
        end
    end
end
```

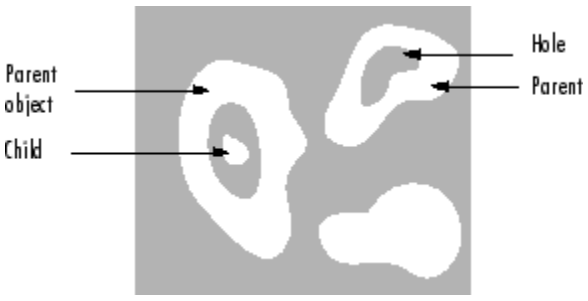


Input Arguments

BW — Input binary image

2-D numeric matrix | 2-D logical matrix

Binary input image, specified as a 2-D logical or numeric matrix. `BW` must be a binary image where nonzero pixels belong to an object and zero-valued pixels constitute the background. The following figure illustrates these components.

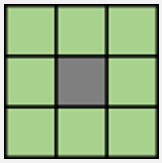


Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

conn — Pixel connectivity

8 (default) | 4

Pixel connectivity, specified as one of the values in this table.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.

Data Types: `double`

`options` — Determine whether to search for both parent and child boundaries
`'holes'` (default) | `'noholes'`

Determine whether to search for both parent and child boundaries, specified as either of the following:

Option	Meaning
`'holes'`	Search for both object and hole boundaries. This is the default.
`'noholes'`	Search only for object (parent and child) boundaries. This can provide better performance.

Data Types: `char` | `string`

Output Arguments

`B` — Row and column coordinates of boundary pixels
p-by-1 cell array

Row and column coordinates of boundary pixels, returned as a *p*-by-1 cell array, where *p* is the number of objects and holes. Each cell in the cell array contains a *q*-by-2 matrix. Each row in the matrix contains the row and column coordinates of a boundary pixel. *q* is the number of boundary pixels for the corresponding region.

`L` — Label matrix
2-D matrix of nonnegative integers

Label matrix of contiguous regions, returned as a 2-D matrix of nonnegative integers. The *k*th region includes all elements in `L` that have value *k*. The number of objects and holes represented by `L` is equal to `max(L(:))`. The zero-valued elements of `L` make up the background.

Data Types: `double`

`n` — Number of objects found
nonnegative integer

Number of objects found, returned as a nonnegative integer.

Data Types: `double`

`A` — Parent-child dependencies between boundaries and holes
square, sparse, logical matrix

Parent-child dependencies between boundaries and holes, returned as a square, sparse, logical matrix of class `double` with side of length `max(L(:))`. The rows and columns of `A` correspond to the positions of boundaries stored in `B`. The first `n` cells in `B` are object boundaries. `A(i,j)=1` means that object `i` is a child of object `j`. .The boundaries that enclose or are enclosed by the *k*-th boundary can be found using `A` as follows:

```
enclosing_boundary  = find(A(m,:));
enclosed_boundaries = find(A(:,m));
```

Algorithms

The `bwboundaries` function implements the Moore-Neighbor tracing algorithm modified by Jacob's stopping criteria. This function is based on the `boundaries` function presented in the first edition of *Digital Image Processing Using MATLAB*, by Gonzalez, R. C., R. E. Woods, and S. L. Eddins, New Jersey, Pearson Prentice Hall, 2004.

References

[1] Gonzalez, R. C., R. E. Woods, and S. L. Eddins, *Digital Image Processing Using MATLAB*, New Jersey, Pearson Prentice Hall, 2004.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `bwboundaries` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `bwboundaries` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- The parameter `conn` must be a compile-time constant.
- The parameter `options` must be a compile-time constant.
- The return value `A` can only be a full matrix, not a sparse matrix.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The parameter `conn` must be a compile-time constant.
- The parameter `options` must be a compile-time constant.
- The return value `A` can only be a full matrix, not a sparse matrix.

See Also

`bwlabel` | `bwlabeln` | `bwtraceboundary` | `bwperim`

Introduced before R2006a

bwconncomp

Find and count connected components in binary image

Syntax

```
CC = bwconncomp(BW)
CC = bwconncomp(BW,conn)
```

Description

`CC = bwconncomp(BW)` finds and counts the connected components `CC` in the binary image `BW`. The `CC` output structure contains the total number of connected components, such as regions of interest (ROIs), in the image and the pixel indices assigned to each component. `bwconncomp` uses a default connectivity of 8 for two dimensions and 26 for three dimensions.

`CC = bwconncomp(BW,conn)` specifies the desired connectivity `conn` for the connected components.

Examples

Calculate Centroids of 3-D Objects

Create a small sample 3-D array.

```
BW = cat(3, [1 1 0; 0 0 0; 1 0 0],...
            [0 1 0; 0 0 0; 0 1 0],...
            [0 1 1; 0 0 0; 0 0 1]);
```

Find the connected components in the array.

```
CC = bwconncomp(BW)
```

```
CC = struct with fields:
    Connectivity: 26
       ImageSize: [3 3 3]
      NumObjects: 2
    PixelIdxList: {[5x1 double]  [3x1 double]}
```

Calculate centroids of the objects in the array.

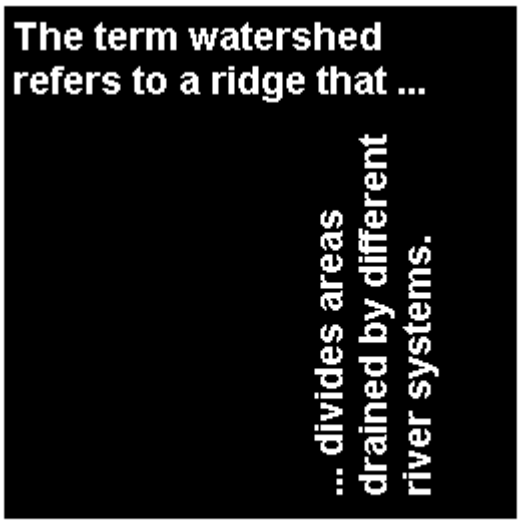
```
S = regionprops(CC,'Centroid')
```

```
S=2×1 struct array with fields:
    Centroid
```

Erase Largest Component from Image

Read image into the workspace and display it.

```
BW = imread('text.png');
imshow(BW)
```



Find the number of connected components in the image.

```
CC = bwconncomp(BW)
```

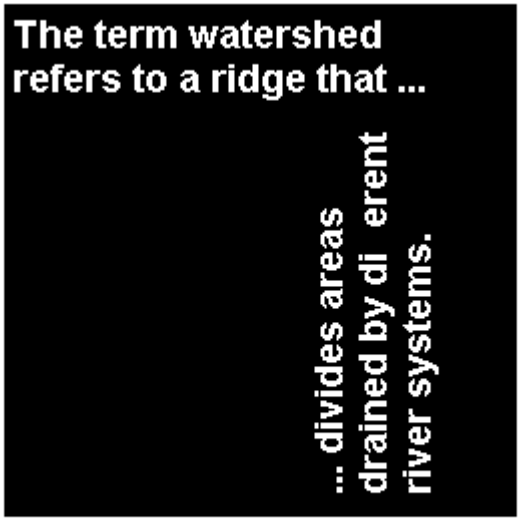
```
CC = struct with fields:
    Connectivity: 8
       ImageSize: [256 256]
      NumObjects: 88
    PixelIdxList: {1x88 cell}
```

Determine which is the largest component in the image and erase it (set all the pixels to 0).

```
numPixels = cellfun(@numel,CC.PixelIdxList);
[biggest,idx] = max(numPixels);
BW(CC.PixelIdxList{idx}) = 0;
```

Display the image, noting that the largest component happens to be the two consecutive f's in the word different.

```
figure
imshow(BW)
```



Input Arguments

BW — Binary image

numeric array | logical array

Binary image, specified as a numeric or logical array of any dimension. For numeric input, any nonzero pixels are considered to be 1 (`true`).

Example: `BW = imread('text.png');`

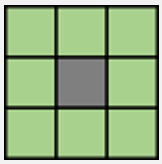
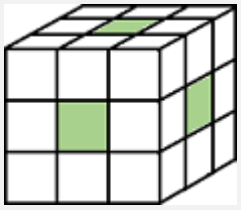
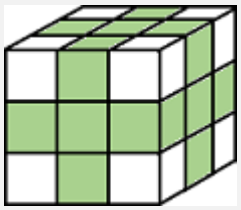
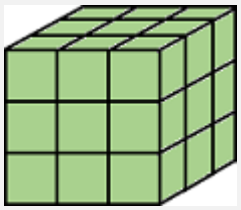
Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

conn — Pixel connectivity

4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 array of 0s and 1s

Pixel connectivity, specified as one of the values in this table. The default connectivity is 8 for 2-D images, and 26 for 3-D images.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal or vertical direction.	 Current pixel is shown in gray.

Value	Meaning	
8	Pixels are connected if their edges or corners touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		
6	Pixels are connected if their faces touch. Two adjoining pixels are part of the same object if they are both on and are connected in: • One of these directions: in, out, left, right, up, and down	 Current pixel is center of cube.
18	Pixels are connected if their faces or edges touch. Two adjoining pixels are part of the same object if they are both on and are connected in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. Two adjoining pixels are part of the same object if they are both on and are connected in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `bwconncomp` uses the default value `conndef(ndims(BW),'maximal')`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Data Types: `double` | `logical`

Output Arguments

`CC` — Connected components

struct

Connected components, returned as a structure with four fields.

Field	Description
`Connectivity`	Connectivity of the connected components (objects)
`ImageSize`	Size of `BW`
`NumObjects`	Number of connected components (objects) in `BW`
`PixelIdxList`	1-by-`NumObjects` cell array where the *k*-th element in the cell array is a vector containing the linear indices of the pixels in the *k*-th object.

Tips

- The functions `bwlabel`, `bwlabeln`, and `bwconncomp` all compute connected components for binary images. `bwconncomp` replaces the use of `bwlabel` and `bwlabeln`. It uses significantly less memory and is sometimes faster than the other functions.

Function	Input Dimension	Output Form	Memory Use	Connectivity
`bwlabel`	2-D	Label matrix with double-precision	High	4 or 8
`bwlabeln`	N-D	Double-precision label matrix	High	Any
`bwconncomp`	N-D	`CC` struct	Low	Any

- To extract features from a binary image using `regionprops` with default connectivity, just pass `BW` directly into `regionprops` using the command `regionprops(BW)`.
- To compute a label matrix having more memory-efficient data type (for instance, `uint8` versus `double`), use the `labelmatrix` function on the output of `bwconncomp`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `bwconncomp` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- `bwconncomp` only supports 2-D inputs.
- The `conn` arguments must be a compile-time constant and the only connectivities supported are 4 or 8. You can also specify connectivity as a 3-by-3 matrix, but it can only be `[0 1 0;1 1 1;0 1 0]` or `ones(3)`
- The `PixelIdxList` field in the `CC` struct return value is not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- `bwconncomp` only supports 2-D inputs.
- The `conn` arguments must be a compile-time constant and the only connectivities supported are 4 or 8. You can also specify connectivity as a 3-by-3 matrix, but it can only be `[0 1 0;1 1 1;0 1 0]` or `ones(3)`
- The `PixelIdxList` field in the `CC` struct return value is not supported.

See Also

`bwlabel` | `bwlabeln` | `labelmatrix` | `regionprops`

Topics

“Label and Measure Connected Components in a Binary Image”
“Pixel Connectivity”

Introduced in R2009a

bwconvhull

Generate convex hull image from binary image

Syntax

```
CH = bwconvhull(BW)
CH = bwconvhull(BW,method)
CH = bwconvhull(BW,'objects',conn)
```

Description

`CH = bwconvhull(BW)` computes the convex hull of all objects in `BW` and returns `CH`, a binary convex hull image.

`CH = bwconvhull(BW,method)` specifies the desired method for computing the convex hull image.

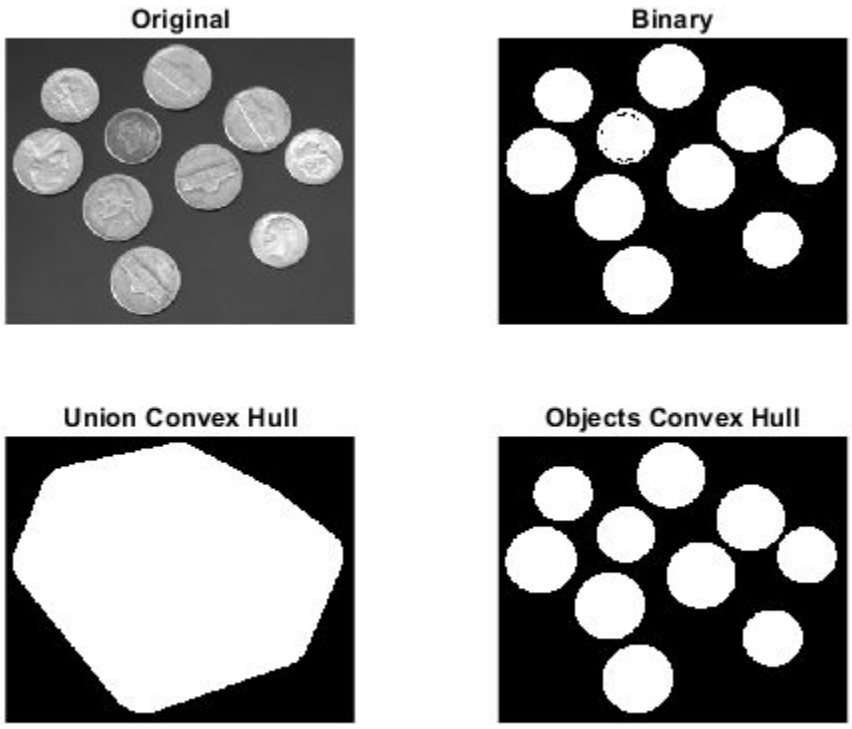
`CH = bwconvhull(BW,'objects',conn)` specifies the desired connectivity used when defining individual foreground objects.

Examples

Display Binary Convex Hull of Image

Read a grayscale image into the workspace. Convert it into a binary image and calculate the union binary convex hull. Finally, calculate the objects convex hull and display all the images in one figure window.

```
subplot(2,2,1);
I = imread('coins.png');
imshow(I);
title('Original');

subplot(2,2,2);
BW = I > 100;
imshow(BW);
title('Binary');

subplot(2,2,3);
CH = bwconvhull(BW);
imshow(CH);
title('Union Convex Hull');

subplot(2,2,4);
CH_objects = bwconvhull(BW,'objects');
imshow(CH_objects);
title('Objects Convex Hull');
```



Input Arguments

`BW` — Input binary image

2-D logical matrix

Input binary image, specified as a 2-D logical matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`method` — Method used to compute the convex hull

`'union'` (default) | `'objects'`

Method used to compute the convex hull, specified as one of the following:

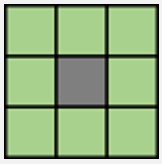
Value	Description
`'union'`	Compute the convex hull of all foreground objects, treating them as a single object
`'objects'`	Compute the convex hull of each connected component of `BW` individually. `CH` contains the convex hulls of each connected component.

Data Types: `char` | `string`

conn — Pixel connectivity

8 (default) | 4 | 3-by-3 matrix of 0s and 1s

Pixel connectivity, specified as one of these values The `conn` parameter is only valid when the `method` is `'objects'`.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.

Connectivity can also be defined in a more general way by specifying a 3-by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. The matrix must be symmetric about its center element.

Data Types: `double`

Output Arguments

CH — Binary mask of the convex hull of all foreground objects in the input image

2-D logical matrix

Binary mask of the convex hull of all foreground objects in the input image, returned as a 2-D logical matrix.

See Also

`bwconncomp` | `bwlabel` | `labelmatrix` | `regionprops`

Introduced in R2011a

bwdist

Distance transform of binary image

Syntax

```
D = bwdist(BW)
[D,idx] = bwdist(BW)
[D,idx] = bwdist(BW,method)
```

Description

`D = bwdist(BW)` computes the Euclidean distance transform of the binary image `BW`. For each pixel in `BW`, the distance transform assigns a number that is the distance between that pixel and the nearest nonzero pixel of `BW`.

`[D,idx] = bwdist(BW)` also computes the closest-pixel map in the form of an index array, `idx`. Each element of `idx` contains the linear index of the nearest nonzero pixel of `BW`. The closest-pixel map is also called the feature map, feature transform, or nearest-neighbor transform.

`[D,idx] = bwdist(BW,method)` computes the distance transform using an alternate distance metric, specified by `method`.

Examples

Compute the Euclidean Distance Transform

This example shows how to compute the Euclidean distance transform of a binary image, and the closest-pixel map of the image.

Create a binary image.

```
bw = zeros(5,5);
bw(2,2) = 1;
bw(4,4) = 1
```

```
bw = 5×5

     0     0     0     0     0
     0     1     0     0     0
     0     0     0     0     0
     0     0     0     1     0
     0     0     0     0     0
```

Calculate the distance transform.

```
[D,IDX] = bwdist(bw)
```

```
D = 5x5 single matrix

    1.4142    1.0000    1.4142    2.2361    3.1623
```

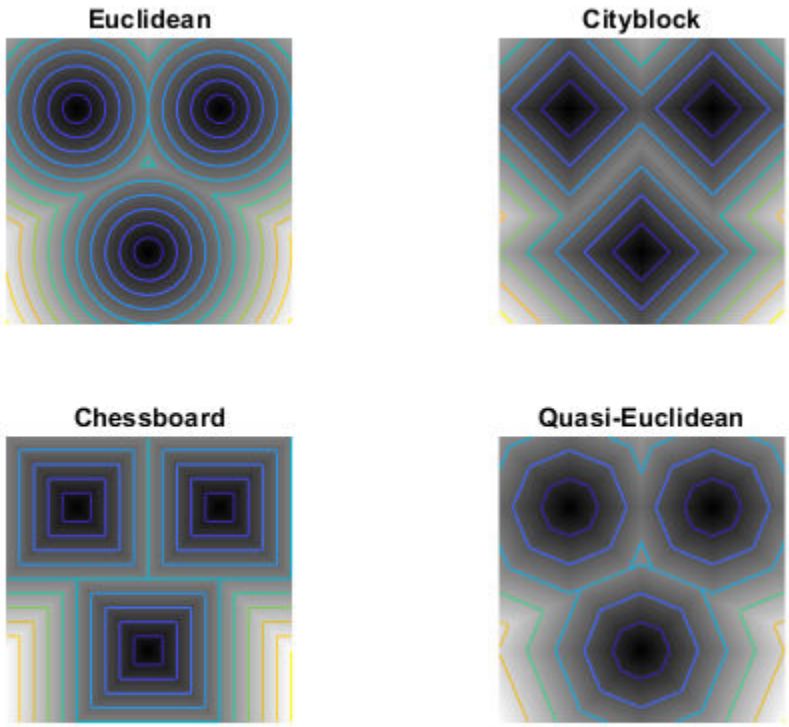
```
    1.0000         0    1.0000    2.0000    2.2361
    1.4142    1.0000    1.4142    1.0000    1.4142
    2.2361    2.0000    1.0000         0    1.0000
    3.1623    2.2361    1.4142    1.0000    1.4142

IDX = 5x5 uint32 matrix

    7    7    7    7    7
    7    7    7    7   19
    7    7    7   19   19
    7    7   19   19   19
    7   19   19   19   19
```

In the nearest-neighbor matrix `IDX` the values 7 and 19 represent the position of the nonzero elements using linear matrix indexing. If a pixel contains a 7, its closest nonzero neighbor is at linear position 7.

Compare 2-D Distance Transforms for Supported Distance Methods

This example shows how to compare the 2-D distance transforms for supported distance methods. In the figure, note how the quasi-Euclidean distance transform best approximates the circular shape achieved by the Euclidean distance method.

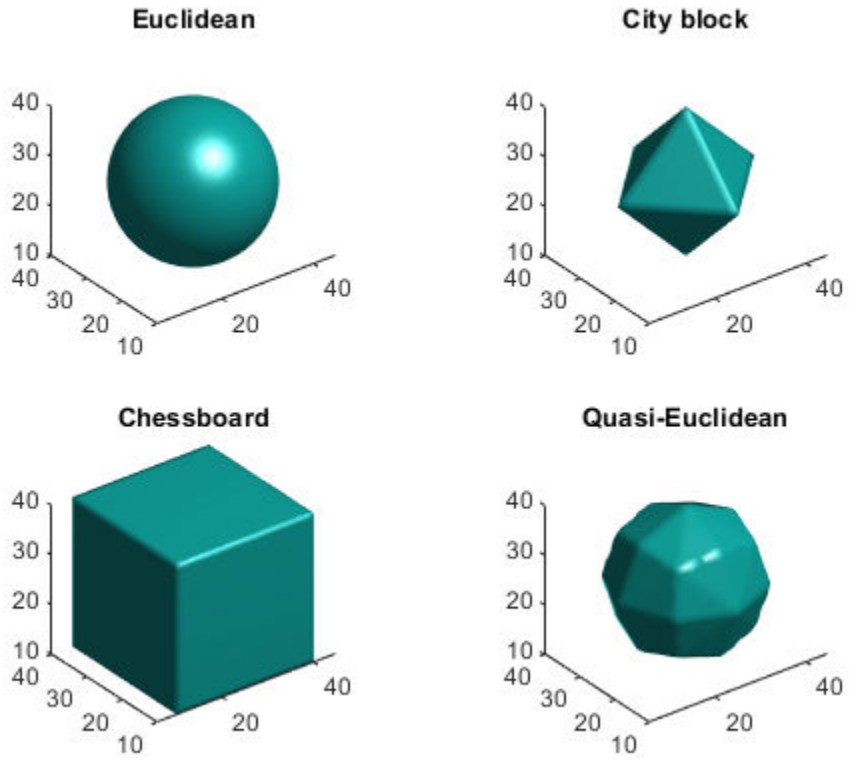
```
bw = zeros(200,200);
bw(50,50) = 1; bw(50,150) = 1; bw(150,100) = 1;
D1 = bwdist(bw,'euclidean');
D2 = bwdist(bw,'cityblock');
D3 = bwdist(bw,'chessboard');
D4 = bwdist(bw,'quasi-euclidean');
RGB1 = repmat(rescale(D1), [1 1 3]);
RGB2 = repmat(rescale(D2), [1 1 3]);
RGB3 = repmat(rescale(D3), [1 1 3]);
RGB4 = repmat(rescale(D4), [1 1 3]);

figure
subplot(2,2,1), imshow(RGB1), title('Euclidean')
hold on, imcontour(D1)
subplot(2,2,2), imshow(RGB2), title('Cityblock')
hold on, imcontour(D2)
subplot(2,2,3), imshow(RGB3), title('Chessboard')
hold on, imcontour(D3)
subplot(2,2,4), imshow(RGB4), title('Quasi-Euclidean')
hold on, imcontour(D4)
```



Compare Isosurface Plots for Distance Transforms of 3-D Image

This example shows how to compare isosurface plots for the distance transforms of a 3-D image containing a single nonzero pixel in the center.

```
bw = zeros(50,50,50); bw(25,25,25) = 1;
D1 = bwdist(bw);
D2 = bwdist(bw,'cityblock');
D3 = bwdist(bw,'chessboard');
D4 = bwdist(bw,'quasi-euclidean');
figure
subplot(2,2,1), isosurface(D1,15), axis equal, view(3)
camlight, lighting gouraud, title('Euclidean')
subplot(2,2,2), isosurface(D2,15), axis equal, view(3)
camlight, lighting gouraud, title('City block')
subplot(2,2,3), isosurface(D3,15), axis equal, view(3)
camlight, lighting gouraud, title('Chessboard')
subplot(2,2,4), isosurface(D4,15), axis equal, view(3)
camlight, lighting gouraud, title('Quasi-Euclidean')
```



Input Arguments

`BW` — Binary image

numeric array | logical array

Binary image, specified as a numeric or logical array of any dimension. For numeric input, any nonzero pixels are considered to be `1` (`true`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`method` — Distance metric

`'euclidean'` (default) | `'chessboard'` | `'cityblock'` | `'quasi-euclidean'`

Distance metric, specified as one of the these values.

Method	Description
`'chessboard'`	In 2-D, the chessboard distance between (x_1,y_1) and (x_2,y_2) is $\max(\lvert x_1 - x_2\rvert, \lvert y_1 - y_2\rvert)$.
`'cityblock'`	In 2-D, the cityblock distance between (x_1,y_1) and (x_2,y_2) is $\lvert x_1 - x_2\rvert + \lvert y_1 - y_2\rvert$

Method	Description
`'euclidean'`	In 2-D, the Euclidean distance between (x_1,y_1) and (x_2,y_2) is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.
`'quasi-euclidean'`	In 2-D, the quasi-Euclidean distance between (x_1,y_1) and (x_2,y_2) is $\lvert x_1 - x_2\rvert + (\sqrt{2} - 1)\lvert y_1 - y_2\rvert,\ \lvert x_1 - x_2\rvert > \lvert y_1 - y_2\rvert$ $(\sqrt{2} - 1)\lvert x_1 - x_2\rvert + \lvert y_1 - y_2\rvert,$ otherwise.

For more information, see “Distance Transform of a Binary Image”.

Data Types: `char` | `string`

Output Arguments

`D` — Distance array

numeric array

Distance, returned as a numeric array of the same size as `BW`. The value of each element is the distance between that pixel and the nearest nonzero pixel in `BW`, as defined by the distance metric, `method`.

Data Types: `single`

`idx` — Index array

numeric array

Index array, returned as a numeric array of the same size as `BW`. Each element of `idx` contains the linear index of the nearest nonzero pixel of `BW`. The class of `idx` depends on the number of elements in the input image, and is determined as follows.

Class	Range
`'uint32'`	`numel(BW) <=` $2^{32} - 1$
`'uint64'`	`numel(BW) >=` 2^{32}

Data Types: `uint32` | `uint64`

Tips

- `bwdist` uses fast algorithms to compute the true Euclidean distance transform, especially in the 2-D case. The other methods are provided primarily for pedagogical reasons. However, the alternative distance transforms are sometimes significantly faster for multidimensional input images, particularly those that have many nonzero elements.
- The function `bwdist` changed in version 6.4 (R2009b). Previous versions of the Image Processing Toolbox used different algorithms for computing the Euclidean distance transform and the associated label matrix. If you need the same results produced by the previous implementation, use the function `bwdist_old`.

Algorithms

- For Euclidean distance transforms, `bwdist` uses the fast algorithm. [1]
- For cityblock, chessboard, and quasi-Euclidean distance transforms, `bwdist` uses the two-pass, sequential scanning algorithm. [2]
- The different distance measures are achieved by using different sets of weights in the scans, as described in [3].

References

[1] Maurer, Calvin, Rensheng Qi, and Vijay Raghavan, "A Linear Time Algorithm for Computing Exact Euclidean Distance Transforms of Binary Images in Arbitrary Dimensions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 25, No. 2, February 2003, pp. 265-270.

[2] Rosenfeld, Azriel and John Pfaltz, "Sequential operations in digital picture processing," *Journal of the Association for Computing Machinery*, Vol. 13, No. 4, 1966, pp. 471-494.

[3] Paglieroni, David, "Distance Transforms: Properties and Machine Vision Applications," *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*, Vol. 54, No. 1, January 1992, pp. 57-58.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `bwdist` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `bwdist` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, the optional second input argument, `method`, must be a compile-time constant. Input images must have less than 2^{32} pixels.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, the optional second input argument, `method`, must be a compile-time constant. Input images must have fewer than 2^{32} pixels.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Input images must be 2-D and have less than 2^{32} elements.
- The `method` argument only supports the `'euclidean'` distance metric.

For more information, see “Image Processing on a GPU”.

See Also

`bwulterode` | `watershed`

Topics

“Distance Transform of a Binary Image”

Introduced before R2006a

bwdistgeodesic

Geodesic distance transform of binary image

Syntax

```
D = bwdistgeodesic(BW,mask)
D = bwdistgeodesic(BW,C,R)
D = bwdistgeodesic(BW,idx)
D = bwdistgeodesic( ___ ,method)
```

Description

`D = bwdistgeodesic(BW,mask)` computes the geodesic distance transform, given the binary image `BW` and the seed locations specified by `mask`. Regions where `BW` is `true` represent valid regions that can be traversed in the computation of the distance transform. Regions where `BW` is `false` represent constrained regions that cannot be traversed in the distance computation. For each `true` pixel in `BW`, the geodesic distance transform assigns a number that is the constrained distance between that pixel and the nearest `true` pixel in `mask`. Output matrix `D` contains geodesic distances.

`D = bwdistgeodesic(BW,C,R)` computes the geodesic distance transform of the binary image `BW`. Vectors `C` and `R` contain the column and row coordinates of the seed locations.

`D = bwdistgeodesic(BW,idx)` computes the geodesic distance transform of the binary image `BW`. `idx` is a vector of linear indices of seed locations.

`D = bwdistgeodesic( ___ ,method)` computes the geodesic distance transform using an alternate distance metric specified by `method`.

Examples

Compute Geodesic Distance Transformation of Binary Image

Create a sample binary image for this example.

```
BW = [1 1 1 1 1 1 1 1 1 1;...
     1 1 1 1 1 1 0 0 1 1;...
     1 1 1 1 1 1 0 0 1 1;...
     1 1 1 1 1 1 0 0 1 1;...
     0 0 0 0 0 1 0 0 1 0;...
     0 0 0 0 1 1 0 1 1 0;...
     0 1 0 0 1 1 0 0 0 0;...
     0 1 1 1 1 1 1 0 1 0;...
     0 1 1 0 0 0 1 1 1 0;...
     0 0 0 0 1 0 0 0 0 0];
 BW = logical(BW);
```

Create two vectors of seed locations.

```
C = [1 2 3 3 3];
R = [3 3 3 1 2];
```

Calculate the geodesic distance transform. Output pixels for which `BW` is false have undefined geodesic distance and contain `NaN` values. Because there is no connected path from the seed locations to element `BW(10,5)`, the output `D(10,5)` has a value of `Inf`.

```
D = bwdistgeodesic(BW,C,R)
```

```
D = 10x10 single matrix

     2     1     0     1     2     3     4     5     6     7
     1     1     0     1     2     3   NaN   NaN     6     7
     0     0     0     1     2     3   NaN   NaN     7     7
     1     1     1     1     2     3   NaN   NaN     8     8
   NaN   NaN   NaN   NaN   NaN     3   NaN   NaN     9   NaN
   NaN   NaN   NaN   NaN     4     4   NaN    10    10   NaN
   NaN     8   NaN   NaN     5     5   NaN   NaN   NaN   NaN
   NaN     8     7     6     6     6     6   NaN     8   NaN
   NaN     8     7   NaN   NaN   NaN     7     7     8   NaN
   NaN   NaN   NaN   NaN   Inf   NaN   NaN   NaN   NaN   NaN
```

Input Arguments

`BW` — Binary image

numeric array | logical array

Binary image, specified as a numeric array or logical array of any dimension. For numeric input, any nonzero pixels are considered to be `1` (`true`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`mask` — Seed locations

logical array

Seed locations, specified as a logical array of the same size as `BW`.

`C` — Column coordinates of seed locations

vector of positive integers

Column coordinates of seed locations, specified as a vector of positive integers of the same length as `R`.

`R` — Row coordinates of seed locations

vector of positive integers

Row coordinates of seed locations, specified as a vector of positive integers of the same length as `C`.

`idx` — Linear indices of seed locations

vector of positive integers

Linear indices of seed locations, specified as a vector of positive integers.

`method` — Distance metric

`'chessboard'` (default) | `'cityblock'` | `'quasi-euclidean'`

Distance metric, specified as one of the following.

Method	Description
`'chessboard'`	In 2-D, the chessboard distance between (x_1,y_1) and (x_2,y_2) is `max(abs(`x_1`-`x_2`),abs(`y_1`-`y_2`))`
`'cityblock'`	In 2-D, the cityblock distance between (x_1,y_1) and (x_2,y_2) is `abs(`x_1`-`x_2`) + abs(`y_1`-`y_2`)`
`'quasi-euclidean'`	In 2-D, the quasi-Euclidean distance between (x_1,y_1) and (x_2,y_2) is $$|x_1 - x_2| + (\sqrt{2} - 1)|y_1 - y_2|, \ |x_1 - x_2| > |y_1 - y_2|$$ $$(\sqrt{2} - 1)|x_1 - x_2| + |y_1 - y_2|, \ \text{otherwise.}$$

Data Types: `char` | `string`

Output Arguments

`D` — Geodesic distances

numeric array

Geodesic distances, returned as a numeric array of the same size as `BW`.

Data Types: `single`

Algorithms

`bwdistgeodesic` uses the geodesic distance algorithm described in Soille, P., *Morphological Image Analysis: Principles and Applications, 2nd Edition*, Secaucus, NJ, Springer-Verlag, 2003, pp. 219–221.

See Also

`bwdist` | `graydist`

Introduced in R2011b

bweuler

Euler number of binary image

Syntax

```
eul = bweuler(BW,conn)
```

Description

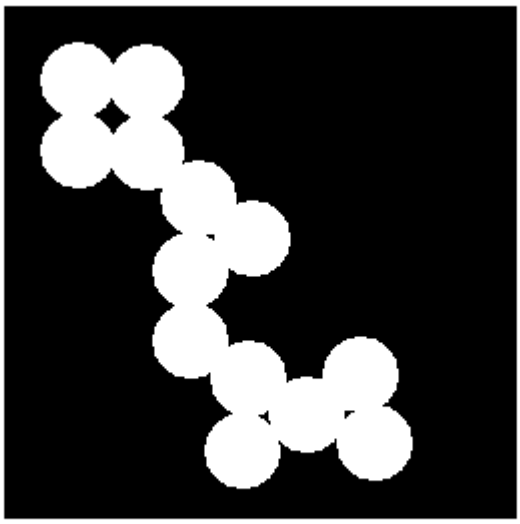
`eul = bweuler(BW,conn)` returns the Euler number for the binary image `BW`. The Euler number (also known as the Euler characteristic) is the total number of objects in the image minus the total number of holes in those objects. `conn` specifies the connectivity. Objects are connected sets of `on` pixels, that is, pixels having a value of 1.

Examples

Calculate Euler Number for Binary Image

Read binary image into workspace, and display it.

```
BW = imread('circles.png');
imshow(BW)
```



Calculate the Euler number. In this example, all the circles touch so they create one object. The object contains four "holes", which are the black areas created by the touching circles. Thus the Euler number is 1 minus 4, or -3.

```
bweuler(BW)

ans = -3
```

Input Arguments

`BW` — Binary image

2-D numeric matrix | 2-D logical matrix

Binary image, specified as a 2-D numeric matrix or 2-D logical matrix. For numeric input, any nonzero pixels are considered to be `on`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`conn` — Connectivity

8 (default) | 4

Connectivity, specified as the values `4` for 4-connected objects or `8` for 8-connected objects.

Data Types: `double`

Output Arguments

`eul` — Euler number

numeric scalar

Euler number, returned as a numeric scalar.

Data Types: `double`

Algorithms

`bweuler` computes the Euler number by considering patterns of convexity and concavity in local 2-by-2 neighborhoods. See [2] on page 1-419 for a discussion of the algorithm used.

References

[1] Horn, Berthold P. K., *Robot Vision,* New York, McGraw-Hill, 1986, pp. 73-77.

[2] Pratt, William K., *Digital Image Processing*, New York, John Wiley & Sons, Inc., 1991, p. 633.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `bweuler` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `bweuler` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance

optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`bwperim` | `bwmorph`

Introduced before R2006a

bwferet

Measure Feret properties

Syntax

```
out = bwferet(BW,properties)
out = bwferet(CC,properties)
out = bwferet(L,properties)
out = bwferet(input)
[out,LM] = bwferet( ___ )
```

Description

`out = bwferet(BW,properties)` measures the Feret properties of objects in an image and returns the measurements in a table. The input `properties` specifies the Feret properties to be measured for each object in input binary image `BW`. The measured Feret properties include the minimum and maximum Feret diameters, Feret angles, and endpoint coordinates of Feret diameters.

`out = bwferet(CC,properties)` measures the Feret properties for each connected component in structure `CC`.

`out = bwferet(L,properties)` measures the Feret properties for each object in the input label matrix `L`.

`out = bwferet(input)` measures the maximum Feret diameter, its relative angle, and coordinate values measured from the `input`. The function returns the measurements in a table. The `input` can be binary image `BW`, connected component `CC`, or label matrix `L`.

`[out,LM] = bwferet( ___ )` also returns a label matrix containing label values that represent the row indices of the table `out`. You can use any of the input arguments from previous syntaxes. Each row entry in `out` corresponds to a labeled region (object) in label matrix `LM`.

Examples

Measure Feret Properties of Objects in Binary Image

Read an image into the workspace.

```
I = imread('toyobjects.png');
```

Convert the image into a binary image.

```
bw = imbinarize(I,'adaptive');
```

Extract the first two largest objects from the binary image.

```
bw = bwareafilt(bw,2);
```

Fill holes in the extracted object regions.

```
bw = imfill(bw,'holes');
```

Calculate the minimum Feret properties and the label matrix of the extracted objects.

```
[out,LM] = bwferet(bw,'MinFeretProperties');
```

Get the maximum number of objects in the output label matrix.

```
maxLabel = max(LM(:));
```

Display the output containing the table of minimum Feret properties.

```
out
```

```
out=2×3 table
    MinDiameter    MinAngle    MinCoordinates
    ___________    ________    ______________

      116.23        99.462      {2x2 double}
      132.08       -159.27      {2x2 double}
```

Display the minimum Feret properties of the object with label-value 1 from the output label matrix.

```
out.MinDiameter(1)
```

```
ans = 116.2301
```

```
out.MinAngle(1)
```

```
ans = 99.4623
```

```
out.MinCoordinates{1}
```

```
ans = 2×2

  120.5000  311.5000
  139.6081  196.8514
```

Display the minimum Feret properties of the object with label-value 2 from the output label matrix.

```
out.MinDiameter(2)
```

```
ans = 132.0776
```

```
out.MinAngle(2)
```

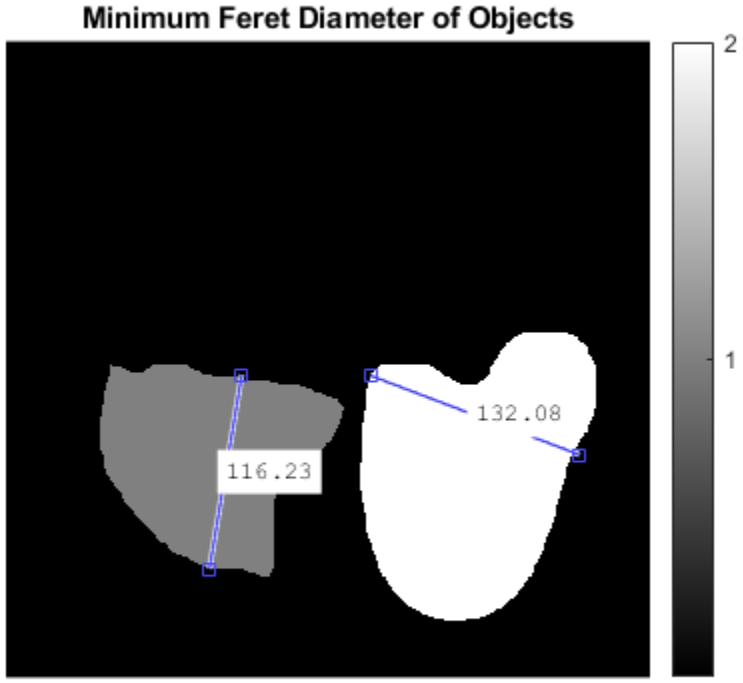
```
ans = -159.2744
```

```
out.MinCoordinates{2}
```

```
ans = 2×2

  215.5000  197.5000
  339.0304  244.2412
```

Display the output label matrix. Plot the endpoint coordinates and minimum Feret diameter of objects with different label values from the output label matrix.

```
h = imshow(LM,[]);
axis = h.Parent;
for labelvalues = 1:maxLabel
    xmin = [out.MinCoordinates{labelvalues}(1,1) out.MinCoordinates{labelvalues}(2,1)];
    ymin = [out.MinCoordinates{labelvalues}(1,2) out.MinCoordinates{labelvalues}(2,2)];
    imdistline(axis,xmin,ymin);
end
title(axis,'Minimum Feret Diameter of Objects');
colorbar('Ticks',1:maxLabel)
```



Measure Feret Properties of Connected Components

Read an image into the workspace.

```
I = imread('toyobjects.png');
```

Convert the image into a binary image.

```
bw = imbinarize(I,'adaptive');
```

Fill holes in the object regions of the input binary image.

```
bw = imfill(bw,'holes');
```

Use the bwconncomp function to generate connected components from the resulting image.

```
cc = bwconncomp(bw);
```

Measure the maximum Feret properties of the connected components.

```
[out,LM] = bwferet(cc,'MaxFeretProperties');
```

Get the maximum number of objects in the output label matrix.

```
maxLabel = max(LM(:));
```

Inspect the table to verify the measured maximum Feret properties.

```
out
```

```
out=4×3 table
    MaxDiameter    MaxAngle    MaxCoordinates
    ___________    ________    ______________

       162.6       -175.06      {2x2 double}
      156.21       -127.46      {2x2 double}
      187.96        121.07      {2x2 double}
      63.781       -131.19      {2x2 double}
```

Display the maximum Feret diameters of objects with different label values from output label matrix.

```
out.MaxDiameter(1:maxLabel)
```

```
ans = 4×1

  162.6038
  156.2082
  187.9628
   63.7809
```

Display the directional angles of the maximum Feret diameters specific to objects with different label values from output label matrix.

```
out.MaxAngle(1:maxLabel)
```

```
ans = 4×1

 -175.0608
 -127.4568
  121.0683
 -131.1859
```

Display the endpoint coordinates of the maximum Feret diameters specific to objects with different label values from output label matrix.

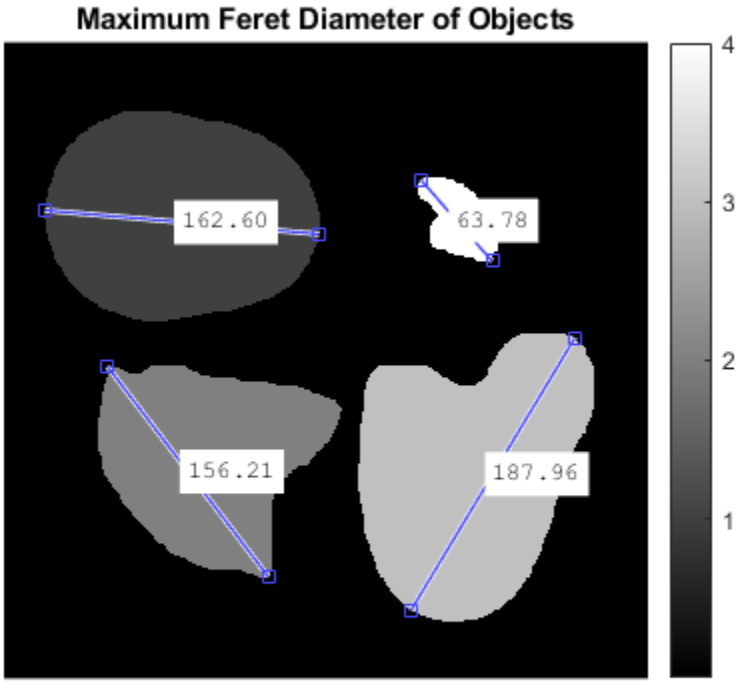
```
out.MaxCoordinates{1:maxLabel}
```

```
ans = 2×2
```

```
  186.5000  113.5000
   24.5000   99.5000


ans = 2×2

  156.5000  315.5000
   61.5000  191.5000


ans = 2×2

  337.5000  174.5000
  240.5000  335.5000


ans = 2×2

  288.5000  129.5000
  246.5000   81.5000
```

Display the output label matrix. Plot the endpoint coordinates and the maximum Feret diameter of objects with different label values from output label matrix.

```
h = imshow(LM,[]);
axis = h.Parent;
for labelvalues = 1:maxLabel
    xmax = [out.MaxCoordinates{labelvalues}(1,1) out.MaxCoordinates{labelvalues}(2,1)];
    ymax = [out.MaxCoordinates{labelvalues}(1,2) out.MaxCoordinates{labelvalues}(2,2)];
    imdistline(axis,xmax,ymax);
end
title(axis,'Maximum Feret Diameter of Objects');
colorbar('Ticks',1:maxLabel)
```



Input Arguments

`BW` — Binary image

numeric matrix | logical matrix

Binary image, specified as a logical or numeric matrix. `BW` must be a binary image where nonzero pixels correspond to an object and zero-valued pixels correspond to the background.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`CC` — Connected components

structure

Connected components, specified as a structure with the four fields shown in this table.

Field	Description
`Connectivity`	Connectivity of the connected components (objects)
`ImageSize`	Size of input binary image
`NumObjects`	Number of connected components (objects) in the input binary image

Field	Description
`PixelIdxList`	1-by-`NumObjects` cell array, where the *k*th element is a vector containing the linear indices of the pixels in the *k*th object

You can use the `bwconncomp` function to generate connected components from a binary image.

Data Types: `struct`

`L` — Label matrix
matrix of nonnegative integers

Label matrix of contiguous regions, specified as a matrix of nonnegative integers. The pixels labeled 0 are the background. The pixels labeled 1 make up one object; the pixels labeled 2 make up a second object; and so on. The number of objects represented by `L` is equal to the maximum value of `L`. You can use the `bwlabel` function to generate label matrix from a binary image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`properties` — Label for Feret properties
`MaxFeretProperties` | `MinFeretProperties` | `all`

Label for Feret properties, specified as `MaxFeretProperties`, `MinFeretProperties`, or `all`.

Data Types: `char` | `string`

`input` — Generic input
numeric matrix | logical matrix | structure | matrix of nonnegative integers

Generic input, specified as one of these values:

- Numeric matrix or logical matrix — When `input` is a binary image, `BW`.
- Structure — When `input` is the connected component, `CC`.
- Matrix of nonnegative integers — When `input` is the label matrix, `L`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical` | `struct`

Output Arguments

`out` — Table of Feret properties
m-by-*n* table

Table of Feret properties, returned as an *m*-by-*n* table. *m* is the number of objects for which the Feret properties are measured. *n* is 3 or 6, depending on the `properties` input.

- If `properties` is `'MaxFeretProperties'`, then the table `out` is of size *m*-by-3 with columns `MaxDiameter`, `MaxAngle`, and `MaxCoordinates`.
- If `properties` is `'MinFeretProperties'`, then the table `out` is of size *m*-by-3 with columns `MinDiameter`, `MinAngle`, and `MinCoordinates`.
- If `properties` is `'all'`, then the table `out` is of size *m*-by-6 with all columns listed in this table.

Column Name	Description
`MaxDiameter`	Maximum Feret diameter of an object, measured as the maximum distance between any two boundary points on the antipodal vertices of the convex hull that encloses that object
`MaxAngle`	Directional angle of the maximum Feret diameter with respect to the horizontal axis of the image. The value, in degrees, is in the range [–180°,180°]
`MaxCoordinates`	Endpoint coordinates of the maximum Feret diameter, returned in the form $\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}$
`MinDiameter`	Minimum Feret diameter of an object, measured as the minimum distance between any two boundary points on the antipodal vertices of the convex hull that encloses that object
`MinAngle`	Directional angle of the minimum Feret diameter with respect to the horizontal axis of the image. The value, in degrees, is in the range [–180°,180°]
`MinCoordinates`	Endpoint coordinates of the minimum Feret diameter, returned in the form $\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}$

`LM` — Label matrix of contiguous regions
matrix of nonnegative integers

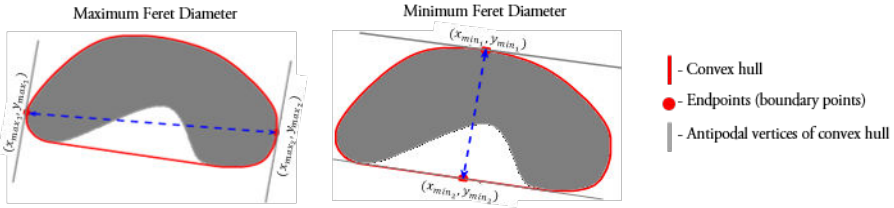
Label matrix of contiguous regions, specified as a matrix of nonnegative integers. The pixels labeled 0 are the background. The pixels labeled 1 make up one object; the pixels labeled 2 make up a second object; and so on. The Feret properties in the *k*th row entry of `out` correspond to the *k*th region (object) in `LM` that have the value *k*. The number of objects represented by `LM` is equal to the maximum value of `LM`.

Note If the input to `bwferet` is a label matrix, then the output label matrix `LM` is same as the input label matrix.

Data Types: `uint8`

Algorithms

The Feret properties of an object are measured by using boundary points on the antipodal vertices of the convex hull that encloses that object.



Given the endpoint coordinates of the maximum (or minimum) Feret diameter, $\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}$, the maximum (or minimum) Feret angle is measured as $angle = \tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1}\right)$.

See Also

`bwconncomp` | `bwlabel` | `bwlabeln` | `labelmatrix` | `regionprops`

Introduced in R2019a

bwhitmiss

Binary hit-miss operation

Syntax

```
BW2 = bwhitmiss(BW,SE1,SE2)
BW2 = bwhitmiss(BW,interval)
```

Description

`BW2 = bwhitmiss(BW,SE1,SE2)` performs the hit-miss operation defined by the structuring elements `SE1` and `SE2`. The hit-miss operation preserves pixels in binary image `BW` whose neighborhoods match the shape of `SE1` and don't match the shape of `SE2`.

This syntax is equivalent to `imerode(BW,SE1) & imerode(~BW,SE2)`.

`BW2 = bwhitmiss(BW,interval)` performs the hit-miss operation defined in terms of a single array, called an *interval*. An interval is an array whose elements are `1`, `0`, or `-1`. The `1`-valued elements make up the domain of `SE1`, the `-1`-valued elements make up the domain of `SE2`, and the `0`-valued elements are ignored.

This syntax is equivalent to `bwhitmiss(BW,interval==1,interval==-1)`.

Examples

Perform Hit-miss Operation on Binary Image

Create sample binary image for this example.

```
bw = [0 0 0 0 0 0
      0 0 1 1 0 0
      0 1 1 1 1 0
      0 1 1 1 1 0
      0 0 1 1 0 0
      0 0 1 0 0 0]
```

```
bw = 6×6

     0     0     0     0     0     0
     0     0     1     1     0     0
     0     1     1     1     1     0
     0     1     1     1     1     0
     0     0     1     1     0     0
     0     0     1     0     0     0
```

Define an interval.

```
interval = [0 -1 -1
            1  1 -1
            0  1  0];
```

Perform hit-miss operation.

```
bw2 = bwhitmiss(bw,interval)
```

```
bw2 = 6x6 logical array

   0   0   0   0   0   0
   0   0   0   1   0   0
   0   0   0   0   1   0
   0   0   0   0   0   0
   0   0   0   0   0   0
   0   0   0   0   0   0
```

Input Arguments

`BW` — Binary image

numeric array | logical array

Binary image, specified as a numeric or logical array of any dimension. For numeric input, any nonzero pixels are considered to be `1` (`true`).

`SE1, SE2` — Structuring element

`strel` object | numeric array

Flat structuring element, specified as a `strel` object or a numeric matrix with values of `1` and `0`. The neighborhoods of `SE1` and `SE2` should not have overlapping elements.

`interval` — Interval

numeric array

Interval, specified as a numeric array with values of `1`, `0`, and `-1`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64`

Output Arguments

`BW2` — Processed binary image

logical array

Processed binary image after the hit-miss operation, specified as a logical array of the same size as `BW`.

Data Types: `logical`

See Also

`imdilate` | `imerode` | `strel`

Introduced before R2006a

bwlabel

Label connected components in 2-D binary image

Syntax

```
L = bwlabel(BW)
L = bwlabel(BW,conn)
[L,n] = bwlabel( ___ )
```

Description

`L = bwlabel(BW)` returns the label matrix `L` that contains labels for the 8-connected objects found in `BW`.

`L = bwlabel(BW,conn)` returns a label matrix, where `conn` specifies the connectivity.

`[L,n] = bwlabel( ___ )` also returns `n`, the number of connected objects found in `BW`.

Examples

Label Components Using 4-connected Objects

Create a small binary image.

```
BW = logical ([1     1     1     0     0     0     0     0
               1     1     1     0     1     1     0     0
               1     1     1     0     1     1     0     0
               1     1     1     0     0     0     1     0
               1     1     1     0     0     0     1     0
               1     1     1     0     0     0     1     0
               1     1     1     0     0     1     1     0
               1     1     1     0     0     0     0     0]);
```

Create the label matrix using 4-connected objects.

```
L = bwlabel(BW,4)
```

```
L = 8×8

     1     1     1     0     0     0     0     0
     1     1     1     0     2     2     0     0
     1     1     1     0     2     2     0     0
     1     1     1     0     0     0     3     0
     1     1     1     0     0     0     3     0
     1     1     1     0     0     0     3     0
     1     1     1     0     0     3     3     0
     1     1     1     0     0     0     0     0
```

Use the `find` command to get the row and column coordinates of the object labeled "2".

```
[r, c] = find(L==2);
rc = [r c]

rc = 4×2

     2     5
     3     5
     2     6
     3     6
```

Input Arguments

BW — Binary image

2-D numeric matrix | 2-D logical matrix

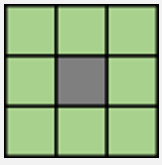
Binary image, specified as a 2-D numeric matrix or 2-D logical matrix. For numeric input, any nonzero pixels are considered to be `1` (`true`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

conn — Pixel connectivity

8 (default) | 4

Pixel connectivity, specified as one of these values.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.

Data Types: `double` | `logical`

Output Arguments

L — Label matrix

matrix of nonnegative integers

Label matrix of contiguous regions, returned as matrix of nonnegative integers with the same size as `BW`. The pixels labeled `0` are the background. The pixels labeled `1` make up one object; the pixels labeled `2` make up a second object; and so on.

Data Types: `double`

n — Number of connected objects
nonnegative integer

Number of connected objects in `BW`, returned as a nonnegative integer.

Data Types: `double`

Tips

- The functions `bwlabel`, `bwlabeln`, and `bwconncomp` all compute connected components for binary images. `bwconncomp` replaces the use of `bwlabel` and `bwlabeln`. It uses significantly less memory and is sometimes faster than the other functions.

	Input Dimension	Output Form	Memory Use	Connectivity
`bwlabel`	2-D	Double-precision label matrix	High	4 or 8
`bwlabeln`	N-D	Double-precision label matrix	High	Any
`bwconncomp`	N-D	CC struct	Low	Any

- You can use the MATLAB `find` function in conjunction with `bwlabel` to return vectors of indices for the pixels that make up a specific object. For example, to return the coordinates for the pixels in object 2, enter the following:.

  ```
  [r,c] = find(bwlabel(BW)==2)
  ```

  You can display the output matrix as a pseudocolor indexed image. Each object appears in a different color, so the objects are easier to distinguish than in the original image. For more information, see `label2rgb`.
- To extract features from a binary image using `regionprops` with default connectivity, just pass `BW` directly into `regionprops` using the command `regionprops(BW)`.
- The `bwlabel` function can take advantage of hardware optimization for data types `logical`, `uint8`, and `single` to run faster. Hardware optimization requires `marker` and `mask` to be 2-D images and `conn` to be either 4 or 8.

Algorithms

`bwlabel` uses the general procedure outlined in reference [1], pp. 40-48:

1. Run-length encode the input image.
2. Scan the runs, assigning preliminary labels and recording label equivalences in a local equivalence table.
3. Resolve the equivalence classes.
4. Relabel the runs based on the resolved equivalence classes.

References

[1] Haralick, Robert M., and Linda G. Shapiro, *Computer and Robot Vision, Volume I,* Addison-Wesley, 1992, pp. 28-48.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `bwlabel` supports the generation of C code (requires MATLAB Coder). For more information, see "Code Generation for Image Processing".
- When generating code, the parameter `n` must be a compile-time constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, the parameter `n` must be a compile-time constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see "Image Processing on a GPU".

See Also

`bwconncomp` | `bwlabeln` | `bwselect` | `labelmatrix` | `label2rgb` | `regionprops`

Introduced before R2006a

bwlabeln

Label connected components in binary image

Syntax

```
L = bwlabeln(BW)
L = bwlabeln(BW,conn)
[L,n] = bwlabeln( ___ )
```

Description

`L = bwlabeln(BW)` returns a label matrix, `L`, containing labels for the connected components in `BW`.

`L = bwlabeln(BW,conn)` returns a label matrix, where `conn` specifies the connectivity.

`[L,n] = bwlabeln( ___ )` also returns `n`, the number of connected objects found in `BW`.

Examples

Calculate Centroids of 3-D Objects

Create simple sample 3-D binary image.

```
BW = cat(3, [1 1 0; 0 0 0; 1 0 0],...
            [0 1 0; 0 0 0; 0 1 0],...
            [0 1 1; 0 0 0; 0 0 1])
```

```
BW =
BW(:,:,1) =

     1     1     0
     0     0     0
     1     0     0


BW(:,:,2) =

     0     1     0
     0     0     0
     0     1     0


BW(:,:,3) =

     0     1     1
     0     0     0
     0     0     1
```

Label connected components in the image.

```
bwlabeln(BW)
```

```
ans =
ans(:,:,1) =

     1     1     0
     0     0     0
     2     0     0


ans(:,:,2) =

     0     1     0
     0     0     0
     0     2     0


ans(:,:,3) =

     0     1     1
     0     0     0
     0     0     2
```

Input Arguments

BW — Binary image

numeric array | logical array

Binary image, specified as a numeric or logical array of any dimension. For numeric input, any nonzero pixels are considered to be `on`.

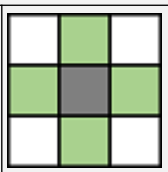
Example: `BW = imread('text.png');`

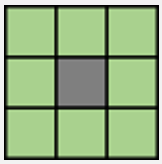
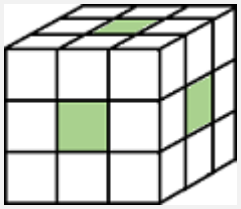
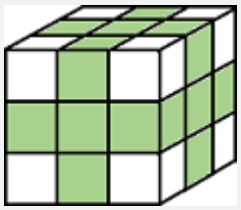
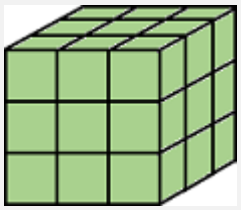
Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

conn — Pixel connectivity

4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 matrix of `0`s and `1`s

Pixel connectivity, specified as one of the values in this table. The default connectivity is `8` for 2-D images, and `26` for 3-D images.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal or vertical direction.	 Current pixel is shown in gray.

Value	Meaning	
8	Pixels are connected if their edges or corners touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		
6	Pixels are connected if their faces touch. Two adjoining pixels are part of the same object if they are both on and are connected in: • One of these directions: in, out, left, right, up, and down	 Current pixel is center of cube.
18	Pixels are connected if their faces or edges touch. Two adjoining pixels are part of the same object if they are both on and are connected in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. Two adjoining pixels are part of the same object if they are both on and are connected in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `bwlabeln` uses the default value `conndef(ndims(BW),'maximal')`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Data Types: `double` | `logical`

Output Arguments

L — Label matrix

array of nonnegative integers

Label matrix, returned as an array of nonnegative integers with the same size as `BW`. The pixels labeled `0` are the background. The pixels labeled `1` make up one object; the pixels labeled `2` make up a second object; and so on.

Data Types: `double`

n — Number of connected objects

nonnegative integer

Number of connected objects in `BW`, returned as a nonnegative integer.

Data Types: `double`

Tips

- The functions `bwlabel`, `bwlabeln`, and `bwconncomp` all compute connected components for binary images. `bwconncomp` replaces the use of `bwlabel` and `bwlabeln`. It uses significantly less memory and is sometimes faster than the other functions.

Function	Input Dimension	Output Form	Memory Use	Connectivity
`bwlabel`	2-D	Label matrix with double-precision	High	4 or 8
`bwlabeln`	N-D	Double-precision label matrix	High	Any
`bwconncomp`	N-D	`CC` struct	Low	Any

- To extract features from a binary image using `regionprops` with default connectivity, just pass `BW` directly into `regionprops` using the command `regionprops(BW)`.

Algorithms

`bwlabeln` uses the following general procedure:

1. Scan all image pixels, assigning preliminary labels to nonzero pixels and recording label equivalences in a union-find table.
2. Resolve the equivalence classes using the union-find algorithm [1].
3. Relabel the pixels based on the resolved equivalence classes.

References

[1] Sedgewick, Robert, *Algorithms in C*, 3rd Ed., Addison-Wesley, 1998, pp. 11-20.

See Also

`bwconncomp` | `bwlabel` | `labelmatrix` | `label2rgb` | `regionprops`

Introduced before R2006a

bwlookup

Nonlinear filtering using lookup tables

Syntax

```
J = bwlookup(BW,lut)
```

Description

`J = bwlookup(BW,lut)` performs a 2-by-2 or 3-by-3 nonlinear neighborhood filtering operation on binary image `BW`. The neighborhood processing determines an integer index value used to access values in lookup table `lut`. The fetched `lut` value becomes the pixel value in output image `J` at the targeted position.

Examples

Perform Erosion Along Edges of Binary Image

Construct the vector `lut` such that the filtering operation places a `1` at the targeted pixel location in the input image only when all four pixels in the 2-by-2 neighborhood of BW are set to `1`.

```
lutfun = @(x)(sum(x(:))==4);
lut = makelut(lutfun,2)
```

```
lut = 16×1

     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     ⋮
```

Load a binary image.

```
BW1 = imread('text.png');
```

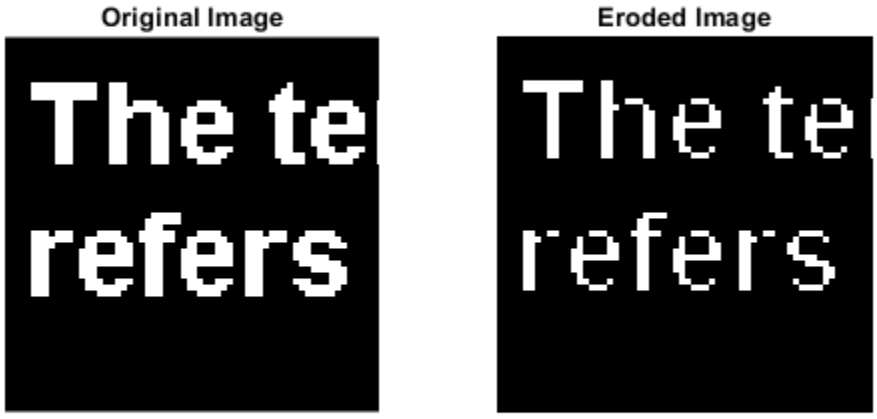
Perform 2-by-2 neighborhood processing with 16-element vector `lut` .

```
BW2 = bwlookup(BW1,lut);
```

Show zoomed before and after images.

```
figure;
h1 = subplot(1,2,1); imshow(BW1), axis off; title('Original Image')
```

```
h2 = subplot(1,2,2); imshow(BW2); axis off; title('Eroded Image')
% 16X zoom to see effects of erosion on text
set(h1,'Ylim',[1 64],'Xlim',[1 64]);
set(h2,'Ylim',[1 64],'Xlim',[1 64]);
```



Input Arguments

BW — Binary image

2-D logical matrix | 2-D numeric matrix

Binary image to be transformed by the nonlinear neighborhood filtering operation, specified as a 2-D logical matrix or 2-D numeric matrix. For numeric input, any nonzero pixels are considered to be 1 (`true`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

lut — Lookup table of output pixel values

16-element vector | 512-element vector

Lookup table of output pixel values, specified as a 16- or 512-element vector. The size of `lut` determines which of the two neighborhood operations is performed. You can use the `makelut` function to create a lookup table.

- If `lut` contains 16 data elements, then the neighborhood matrix is 2-by-2.

- If `lut` contains 512 data elements, then the neighborhood matrix is 3-by-3.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

`J` — Output image

binary image | grayscale image

Output image, returned as a grayscale or binary image whose distribution of pixel values are determined by the content of the lookup table, `lut`. The output image `J` is the same size as the input image `BW` and the same data type as `lut`.

Algorithms

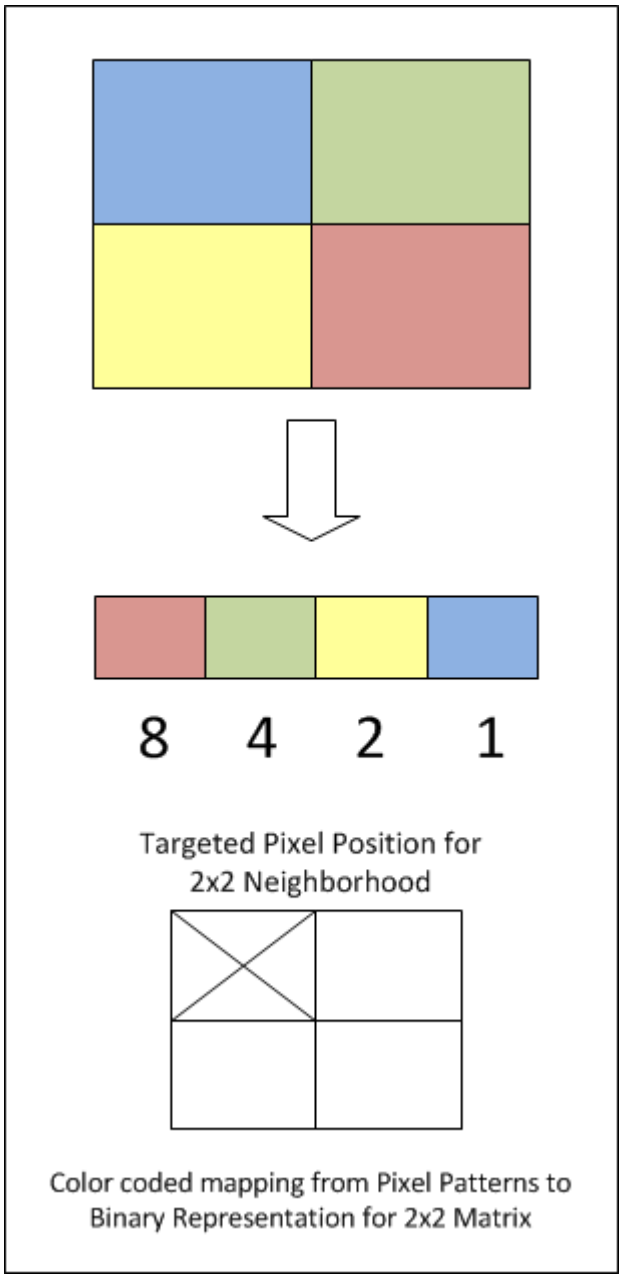
The first step in each iteration of the filtering operation performed by `bwlookup` entails computing the `index` into vector `lut` based on the binary pixel pattern of the neighborhood matrix on image `BW`. The value in `lut` accessed at `index`, `lut(index)`, is inserted into output image `J` at the targeted pixel location. This results in image `J` being the same data type as vector `lut`.

Since there is a 1-to-1 correspondence in targeted pixel locations, image `J` is the same size as image `BW`. If the targeted pixel location is on an edge of image `BW` and if any part of the 2-by-2 or 3-by-3 neighborhood matrix extends beyond the image edge, then these non-image locations are padded with 0 in order to perform the filtering operation.

The following figures show the mapping from binary 0 and 1 patterns in the neighborhood matrices to its binary representation. Adding 1 to the binary representation yields `index` which is used to access `lut`.

2-by-2 Neighborhood Lookup

For 2-by-2 neighborhoods, `length(lut)` is 16. There are four pixels in each neighborhood, and two possible states for each pixel, so the total number of permutations is $2^4 = 16$.



To illustrate, this example shows how the pixel pattern in a 2-by-2 matrix determines which entry in `lut` is placed in the targeted pixel location.

1 Create random 16-element `lut` vector containing `uint8` data.

```
scurr = rng;        % save current random number generator seed state
rng('default')    % always generate same set of random numbers
lut = uint8( round( 255*rand(16,1) ) ) % generate lut
rng(scurr);         % restore
```

```
lut =

  208
```

```
  231
   32
  233
  161
   25
   71
  139
  244
  246
   40
  248
  244
  124
  204
   36
```

2 Create a 2-by-2 image and assume for this example that the targeted pixel location is location `BW(1,1)`.

```
BW = [1 0; 0 1]

BW =

     1     0
     0     1
```

3 By referring to the color coded mapping figure above, the binary representation for this 2-by-2 neighborhood can be computed as shown in the code snippet below. The logical 1 at `BW(1,1)` corresponds to blue in the figure which maps to the Least Significant Bit (LSB) at position 0 in the 4-bit binary representation (,2^0= 1). The logical 1 at `BW(2,2)` is red which maps to the Most Significant Bit (MSB) at position 3 in the 4-bit binary representation (2^3= 8) .

```
% BW(1,1): blue square; sets bit position 0 on right
% BW(2,2): red  square; sets bit position 3 on left
binNot = '1 0 0 1';    % binary representation of 2x2 neighborhood matrix

X = bin2dec( binNot );    % convert from binary to decimal
index = X + 1    % add 1 to compute index value for uint8 vector lut
A11 = lut(index)    % value at A(1,1)

index =

    10

A11 =

  246
```

4 The above calculation predicts that output image A should contain the value 246 at targeted position `A(1,1)`.

```
A = bwlookup(BW,lut)   % perform filtering

A =

  246   32
  161  231
```

`A(1,1)` does in fact equal 246.

3-by-3 Neighborhood Lookup

For 3-by-3 neighborhoods, `length(lut)` is 512. There are nine pixels in each neighborhood, and two possible states for each pixel, so the total number of permutations is $2^9 = 512$.

The process for computing the binary representation of 3-by-3 neighborhood processing is the same as for 2-by-2 neighborhoods.

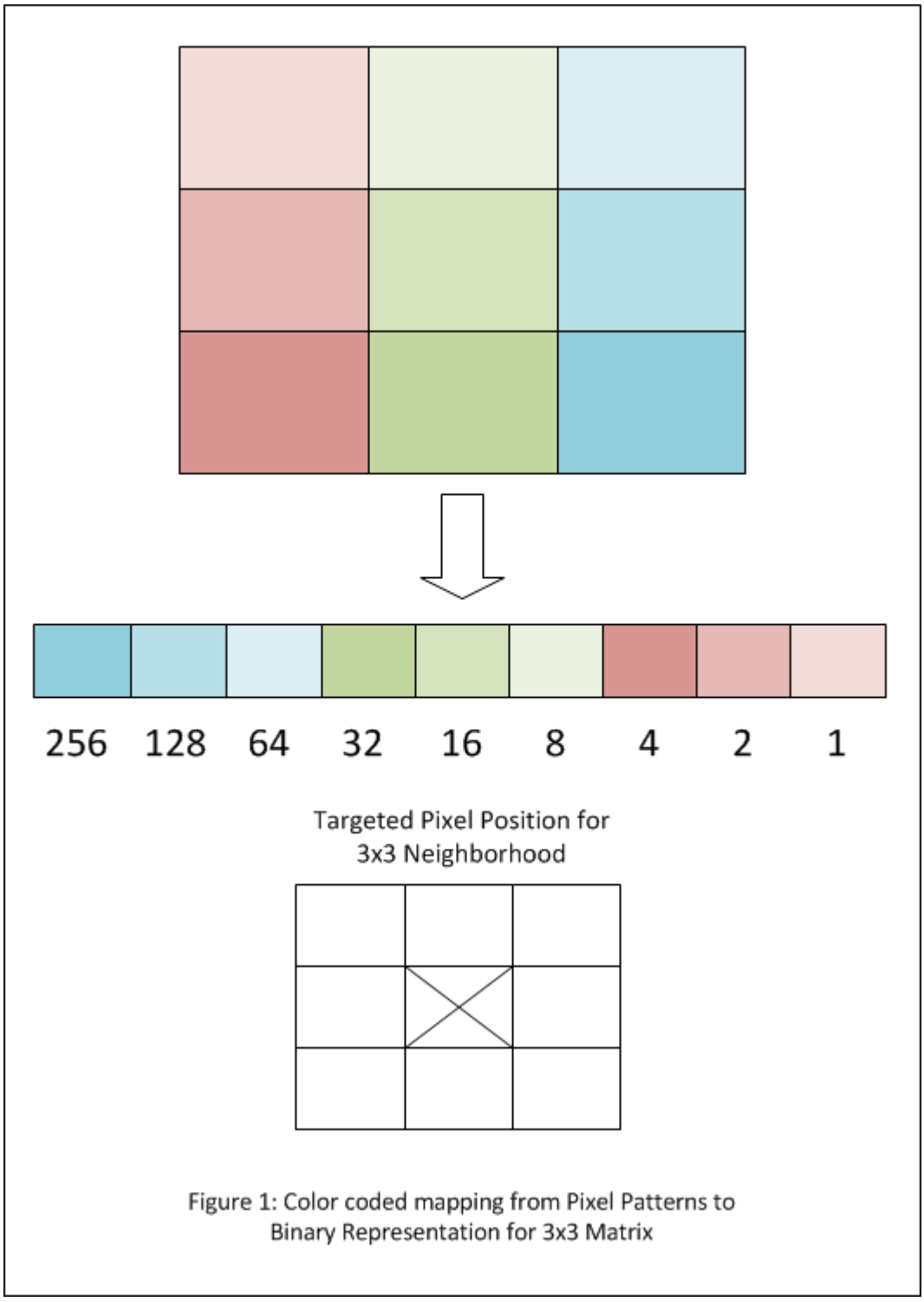


Figure 1: Color coded mapping from Pixel Patterns to Binary Representation for 3x3 Matrix

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `bwlookup` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `bwlookup` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, specify an input image of class `logical`.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, specify an input image of class `logical`.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also
`makelut`

Introduced in R2012b

bwmorph

Morphological operations on binary images

Syntax

```
BW2 = bwmorph(BW,operation)
BW2 = bwmorph(BW,operation,n)
```

Description

`BW2 = bwmorph(BW,operation)` applies a specific morphological operation to the binary image `BW`.

Note To perform morphological operations on a 3-D volumetric image, use `bwmorph3`.

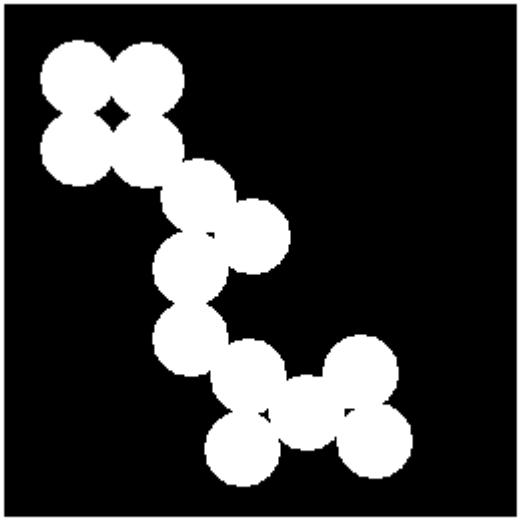
`BW2 = bwmorph(BW,operation,n)` applies the operation `n` times. `n` can be `Inf`, in which case the operation is repeated until the image no longer changes.

Examples

Perform Morphological Operations on Binary Image

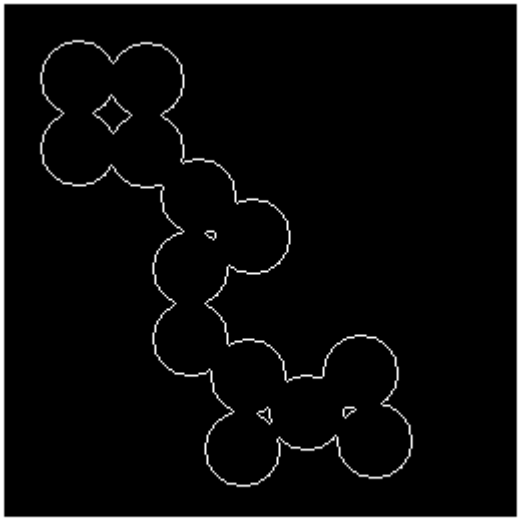
Read binary image and display it.

```
BW = imread('circles.png');
imshow(BW);
```



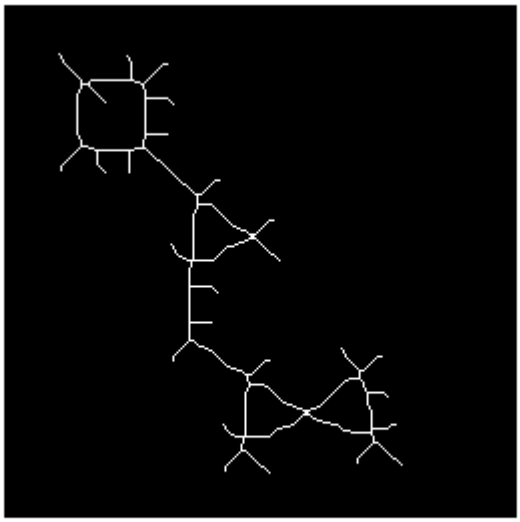
Remove interior pixels to leave an outline of the shapes.

```
BW2 = bwmorph(BW,'remove');
figure
imshow(BW2)
```



Get the image skeleton.

```
BW3 = bwmorph(BW,'skel',Inf);
figure
imshow(BW3)
```



Input Arguments

BW — Binary image

2-D numeric matrix | 2-D logical matrix

Binary image, specified as a 2-D numeric matrix or 2-D logical matrix. For numeric input, any nonzero pixels are considered to be 1 (`true`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

operation — Morphological operation to perform

character vector | string scalar

Morphological operation to perform, specified as one of the following.

Operation	Description
`'bothat'`	Perform the morphological bottom hat operation, returning the image minus the morphological closing of the image. The `bwmorph` function performs morphological closing using the neighborhood `ones(3)`. If you want to perform a morphological bottom hat operation with a different neighborhood, then use the `imbothat` function.

Operation	Description

`'branchpoints'`	Find branch points of skeleton. For example:
	
	`0 0 1 0 0 0 0 0 0 0`
	`0 0 1 0 0 0 0 0 0 0`
	`1 1 1 1 1 becomes 0 0 1 0 0`
	`0 0 1 0 0 0 0 0 0 0`
	`0 0 1 0 0 0 0 0 0 0`
	
	Note: To find branch points, the image must be skeletonized. To create a skeletonized image, use `bwmorph(BW,'skel')`.

`'bridge'`	Bridge unconnected pixels, that is, sets `0`-valued pixels to `1` if they have two nonzero neighbors that are not connected. For example:
	
	`1 0 0 1 1 0`
	`1 0 1 becomes 1 1 1`
	`0 0 1 0 1 1`

`'clean'`	Remove isolated pixels (individual `1`s that are surrounded by `0`s), such as the center pixel in this pattern.
	
	`0 0 0`
	`0 1 0`
	`0 0 0`

`'close'`	Perform morphological closing (dilation followed by erosion). The `bwmorph` function performs morphological closing using the neighborhood `ones(3)`. If you want to perform a morphological closing operation with a different neighborhood, then use the `imclose` function.

`'diag'`	Use diagonal fill to eliminate 8-connectivity of the background. For example:
	
	`0 1 0 0 1 0`
	`1 0 0 becomes 1 1 0`
	`0 0 0 0 0 0`

`'endpoints'`	Find end points of skeleton. For example:
	
	`1 0 0 0 1 0 0 0`
	`0 1 0 0 becomes 0 0 0 0`
	`0 0 1 0 0 0 1 0`
	`0 0 0 0 0 0 0 0`
	
	Note: To find end points, the image must be skeletonized. To create a skeletonized image, use `bwmorph(BW,'skel')`.

`'fill'`	Fill isolated interior pixels (individual `0`s that are surrounded by `1`s), such as the center pixel in this pattern.
	
	`1 1 1`
	`1 0 1`
	`1 1 1`

Operation	Description

`'hbreak'`	Remove H-connected pixels. For example:
	
	`1 1 1 1 1 1`
	`0 1 0 becomes 0 0 0`
	`1 1 1 1 1 1`

`'majority'`	Set a pixel to `1` if five or more pixels in its 3-by-3 neighborhood are `1`; otherwise, set the pixel to `0`.
`'open'`	Perform morphological opening (erosion followed by dilation). The `bwmorph` function performs morphological opening using the neighborhood `ones(3)`. If you want to perform a morphological opening operation with a different neighborhood, then use the `imopen` function.
`'remove'`	Remove interior pixels. This option sets a pixel to `0` if all its 4-connected neighbors are `1`, thus leaving only the boundary pixels on.
`'shrink'`	With `n = Inf`, shrink objects to points by removing pixels from the boundaries of objects. Objects without holes shrink to a point, and objects with holes shrink to a connected ring halfway between each hole and the outer boundary. This option preserves the Euler number (also known as the Euler characteristic).
`'skel'`	With `n = Inf`, remove pixels on the boundaries of objects without allowing objects to break apart. The pixels remaining make up the image skeleton. This option preserves the Euler number. When working with 3-D volumes, or when you want to prune a skeleton, use the `bwskel` function.

`'spur'`	Remove spur pixels. For example:
	
	`0 0 0 0 0 0 0 0`
	`0 0 0 0 0 0 0 0`
	`0 0 1 0 becomes 0 0 0 0`
	`0 1 0 0 0 1 0 0`
	`1 1 0 0 1 1 0 0`

`'thicken'`	With `n = Inf`, thicken objects by adding pixels to the exterior of objects until doing so would result in previously unconnected objects being 8-connected. This option preserves the Euler number.
`'thin'`	With `n = Inf`, thin objects to lines by removing pixels from the boundary of objects. An object without holes shrinks to a minimally connected stroke, and an object with holes shrinks to a connected ring halfway between each hole and the outer boundary. This option preserves the Euler number. See “Algorithms” on page 1-454 for more detail.
`'tophat'`	Perform the morphological top hat operation, returning the image minus the morphological opening of the image. The `bwmorph` function performs morphological opening using the neighborhood `ones(3)`. If you want to perform a morphological top hat operation with a different neighborhood, then use the `imtophat` function.

Tip To perform morphological erosion or dilation, use the `imerode` or `imdilate` function, respectively. If you want to replicate the dilation or erosion performed by the `bwmorph` function, then specify the neighborhood as `ones(3)`.

Data Types: `char` | `string`

n — Number of times to perform operation

positive integer | `Inf`

Number of times to perform the operation, specified as a positive integer or `Inf`. When you specify `n` as `Inf`, the `bwmorph` function repeats the operation until the image no longer changes.

Example: `100`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

BW2 — Image after morphological operations

2-D logical matrix

Image after morphological operations, returned as a 2-D logical matrix.

Data Types: `logical`

Algorithms

When used with the `'thin'` option, `bwmorph` uses the following algorithm [3]:

1. In the first subiteration, delete pixel p if and only if the conditions G_1, G_2, and G_3 are all satisfied.
2. In the second subiteration, delete pixel p if and only if the conditions G_1, G_2, and G_3' are all satisfied.

Condition G1:

$$X_H(p) = 1$$

where

$$X_H(p) = \sum_{i=1}^{4} b_i$$

$$b_i = \begin{cases} 1, & \text{if } x_{2i-1} = 0 \text{ and } (x_{2i} = 1 \text{ or } x_{2i+1} = 1) \\ 0, & \text{otherwise} \end{cases}$$

x_1, x_2, ..., x_8 are the values of the eight neighbors of p, starting with the east neighbor and numbered in counter-clockwise order.

Condition G2:

$$2 \le \min\{n_1(p), n_2(p)\} \le 3$$

where

$$n_1(p) = \sum_{k=1}^{4} x_{2k-1} \vee x_{2k}$$

$$n_2(p) = \sum_{k=1}^{4} x_{2k} \vee x_{2k+1}$$

Condition G3:

$$(x_2 \vee x_3 \vee \bar{x}_8) \wedge x_1 = 0$$

Condition G3':

$$(x_6 \vee x_7 \vee \bar{x}_4) \wedge x_5 = 0$$

The two subiterations together make up one iteration of the thinning algorithm. When the user specifies an infinite number of iterations (`n=Inf`), the iterations are repeated until the image stops changing. The conditions are all tested using `applylut` with precomputed lookup tables.

References

[1] Haralick, Robert M., and Linda G. Shapiro, *Computer and Robot Vision*, Vol. 1, Addison-Wesley, 1992.

[2] Kong, T. Yung and Azriel Rosenfeld, *Topological Algorithms for Digital Image Processing*, Elsevier Science, Inc., 1996.

[3] Lam, L., Seong-Whan Lee, and Ching Y. Suen, "Thinning Methodologies-A Comprehensive Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol 14, No. 9, September 1992, page 879, bottom of first column through top of second column.

[4] Pratt, William K., *Digital Image Processing*, John Wiley & Sons, Inc., 1991.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `bwmorph` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `bwmorph` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, the character vectors or string scalars specifying the operation must be a compile-time constant and, for best results, the input image must be of class `logical`.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, the character vectors or string scalars specifying the operation must be a compile-time constant and, for best results, the input image must be of class `logical`.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`bweuler` | `bwskel` | `bwmorph3` | `imdilate` | `bwperim` | `imerode`

Topics

“Types of Morphological Operations”
“Pixel Connectivity”

Introduced before R2006a

bwmorph3

Morphological operations on binary volume

Syntax

```
J = bwmorph3(V,operation)
```

Description

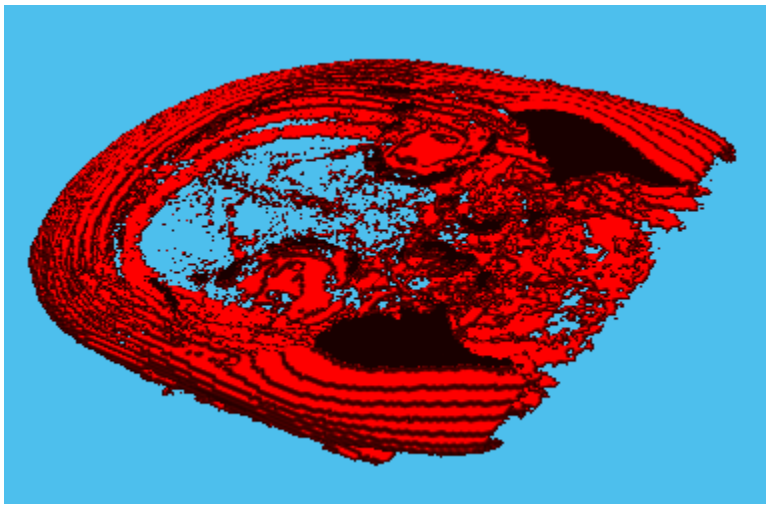
`J = bwmorph3(V,operation)` applies the morphological operation specified by the string or character vector `operation` to the binary volume `V`. `bwmorph3` returns the results of the operation in logical volume `J`.

Examples

Compare the Clean and Majority Operations of bwmorph3

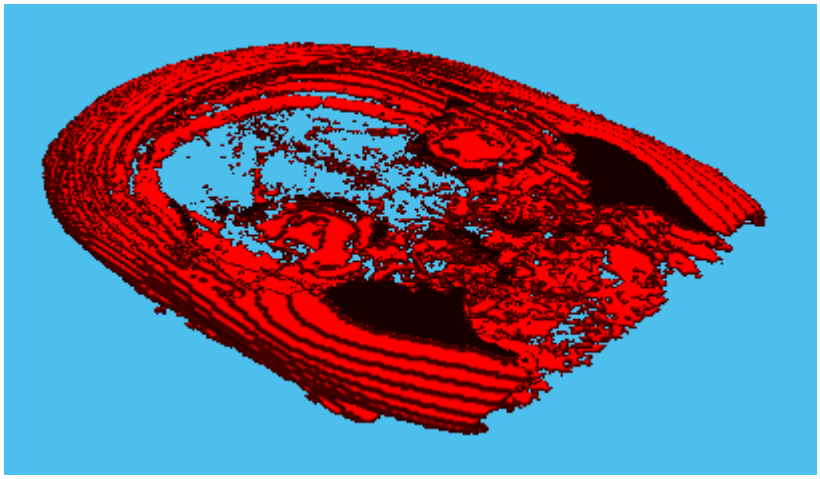
Load 3-D MRI volumetric data and create a binary volume. Use `volshow` to view the volumetric data.

```
load mristack;
BW1 = mristack > 127;
volshow(BW1);
```



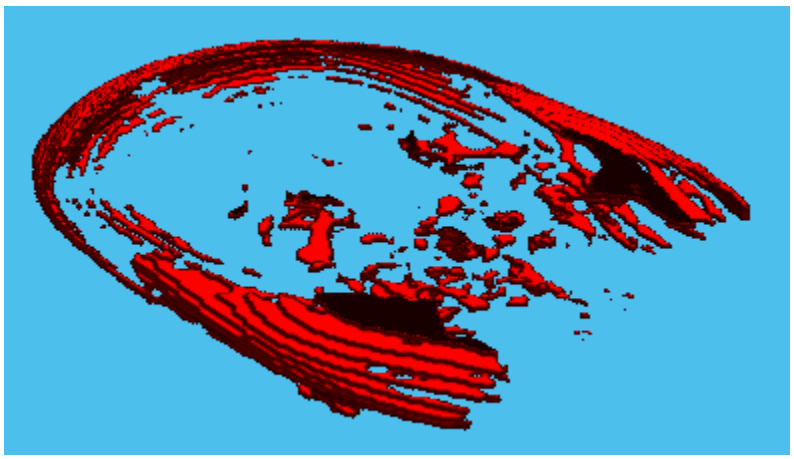
To remove voxels that are set to 1 and that are also surrounded by voxels set to 0, perform the `'clean'` operation on the volumetric data. When determining which voxels to remove, the `'clean'` operation considers 26 neighboring voxels. Use `volshow` to view the results.

```
BW2 = bwmorph3(BW1,'clean');
volshow(BW2);
```



For comparison, perform the 'majority' operation on the volumetric data. The 'majority' operation performs a similar task to the 'clean' operation but only retains voxels if more than half (the majority) of the voxels in the neighborhood of the target voxel are set to 1. When determining which voxels to retain, the 'majority' operation also considers 26 neighboring voxels. Use volshow to view the results.

```
BW3 = bwmorph3(BW1,'majority');
volshow(BW3);
```



Illustrations of Morphological Operations

This example shows how each of the morphological operations supported by bwmorph3 works on simple volumes.

Make a 9-by-9-by-3 cuboid of 0s that contains a 3-by-3-by-3 cube of 1s at its center.

```
innercube = ones(3,3,3);
cube_center = padarray(innercube,[3 3],0,'both')
```

```
cube_center =
cube_center(:,:,1) =
```

```
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0

cube_center(:,:,2) =

     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0

cube_center(:,:,3) =

     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
```

Turning Pixels Off with the Remove Operation

Set the center voxel of the inner cube to `0` using the `'remove'` operation. This operation sets the value of any `'on'` voxel completely surrounded by `'on'` voxels to `'off'`.

```
remove_center = bwmorph3(cube_center,'remove')

remove_center = 9x9x3 logical array
remove_center(:,:,1) =

   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
```

```
remove_center(:,:,2) =

   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   0   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0


remove_center(:,:,3) =

   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
```

Setting Pixels to On with the Fill Operation

Set the center voxel of the inner cube to 1 using the `'fill'` operation. This operation sets the value of any `'off'` voxel completely surrounded by `'on'` voxels to `'on'`.

```
fill_center = bwmorph3(remove_center,'fill')
```

```
fill_center = 9x9x3 logical array
fill_center(:,:,1) =

   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0


fill_center(:,:,2) =

   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
```

```
   0   0   0   0   0   0   0   0   0


fill_center(:,:,3) =

   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
```

Removing Unconnected Pixels with the Clean Operation

Use the `'clean'` operation to remove any stray voxels that are set to 1 but are not connected to a component in the volume. The example creates a stray voxel by setting a random voxel on the second plane to 1 and then uses the `'clean'` operation to remove it.

```
cube_center(2,2,2) = 1
```

```
cube_center =
cube_center(:,:,1) =

     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0


cube_center(:,:,2) =

     0     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0


cube_center(:,:,3) =

     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     1     1     1     0     0     0
```

```
     0     0     0     1     1     1     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
```

```
cube_cleaned = bwmorph3(cube_center,'clean')
```

```
cube_cleaned = 9x9x3 logical array
cube_cleaned(:,:,1) =

   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0


cube_cleaned(:,:,2) =

   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0


cube_cleaned(:,:,3) =

   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
```

Finding the Majority

Find the majority of the `cube_center` using the `'majority'` operation. This operation retains a voxel only if more than half (the majority) of the voxels in the 26-connected neighborhood around the voxel are set to 1.

```
cube_major = bwmorph3(cube_center,'majority')
```

```
cube_major = 9x9x3 logical array
cube_major(:,:,1) =
```

```
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   1   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0

cube_major(:,:,2) =

   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   1   0   0   0   0
   0   0   0   1   1   1   0   0   0
   0   0   0   0   1   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0

cube_major(:,:,3) =

   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   1   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
```

Creating a Shape Similar to a Skeleton

To illustrate the branch points and end points options, create another small matrix, this time with a linear shape, like a skeleton.

```
x1 = eye(5);
x2 = zeros(5);
x2(3,3) = 1;
x3 = x2;
shape = cat(3,x1,x2,x3)

shape =
shape(:,:,1) =

     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
```

```
shape(:,:,2) =

     0     0     0     0     0
     0     0     0     0     0
     0     0     1     0     0
     0     0     0     0     0
     0     0     0     0     0


shape(:,:,3) =

     0     0     0     0     0
     0     0     0     0     0
     0     0     1     0     0
     0     0     0     0     0
     0     0     0     0     0
```

Finding End Points

Find the end points of the shape using the `'endpoints'` operation. The shape has three end points, one at each end of the diagonal in the first plane and one at the end of the line through the center, on the third plane.

```
shape_endpts = bwmorph3(shape,'endpoints')
```

```
shape_endpts = 5x5x3 logical array
shape_endpts(:,:,1) =

   1   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   1


shape_endpts(:,:,2) =

   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0


shape_endpts(:,:,3) =

   0   0   0   0   0
   0   0   0   0   0
   0   0   1   0   0
   0   0   0   0   0
   0   0   0   0   0
```

Finding Branch Points

Find the branch points of the shape using the `'branchpoints'` operation. The shape has a single branch point, where the diagonal line and the horizontal line meet.

```
shape_brpts = bwmorph3(shape,'branchpoints')
```

```
shape_brpts = 5x5x3 logical array
shape_brpts(:,:,1) =

   0   0   0   0   0
   0   1   0   0   0
   0   0   1   0   0
   0   0   0   1   0
   0   0   0   0   0


shape_brpts(:,:,2) =

   0   0   0   0   0
   0   0   0   0   0
   0   0   1   0   0
   0   0   0   0   0
   0   0   0   0   0


shape_brpts(:,:,3) =

   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
```

Input Arguments

`V` — Input volume

numeric array | logical array

Input volume, specified as a numeric or logical array. For numeric input, any nonzero pixels are considered to be `1` (`true`).

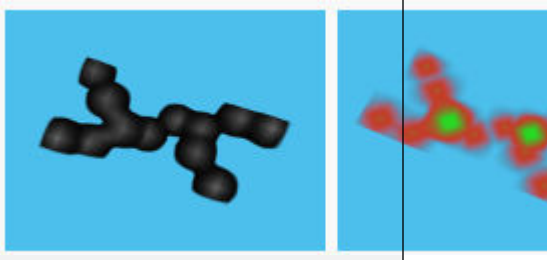
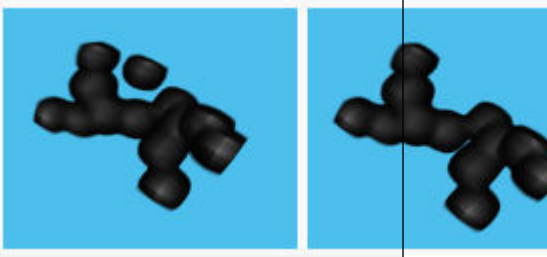
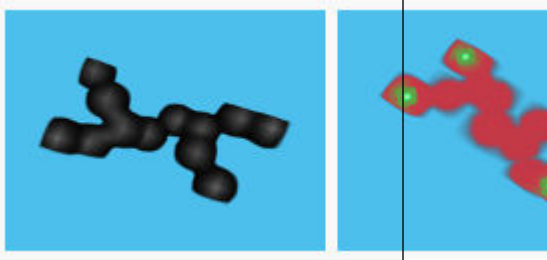
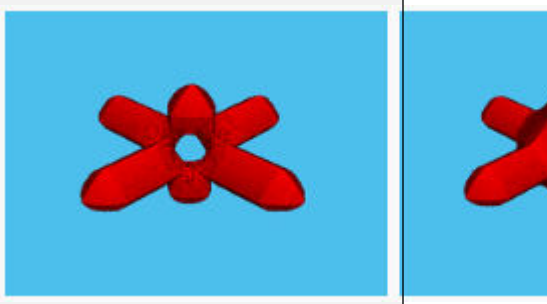
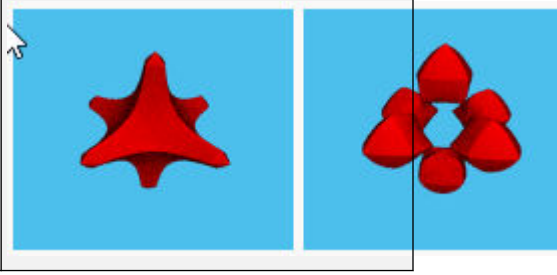
`bwmorph3` accepts 1-D, 2-D, or 3-D arrays. If you specify 1-D or 2-D input arrays, then `bwmorph3` performs the morphological operation as defined for a 3-D volume. If you want 2-D behavior, use `bwmorph` instead.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`operation` — Morphological operation to perform

character vector | string scalar

Morphological operation to perform, specified as one of the following character vectors or string scalar. For examples of these operations, see “Illustrations of Morphological Operations” on page 1-458.

Operation	Description	Illustration
`'branchpoints'`	Find branch points of skeleton. Branch points are the voxels at the junction where multiple branches meet. To find branch points, the image must be skeletonized. To create a skeletonized image, use `bwskel`.	
`'clean'`	Remove isolated voxels, setting them to 0. An isolated voxel is an individual, 26-connected voxel that is set to `1` that are surrounded by voxels set to `0`.	
`'endpoints'`	Find end points of skeleton. End points are voxels at the ends of branches. Note: To find end points, the image must be skeletonized. To create a skeletonized image, use `bwskel`.	
`'fill'`	Fill isolated interior voxels, setting them to `1`. Isolated interior voxels are individual voxels that are set to `0` that are surrounded (6-connected) by voxels set to `1`.	
`'majority'`	Keep a voxel set to `1` if 14 or more voxels (the majority) in its 3-by-3-by-3, 26-connected neighborhood are set to `1`; otherwise, set the voxel to `0`.	See “Illustrations of Morphological Operations” on page 1-458.
`'remove'`	Remove interior voxels, setting it to `0`. Interior voxels are individual voxels that are set to `1` that are surrounded (6-connected) by voxels set to `1`.	

Data Types: `char` | `string`

Output Arguments

`J` — Volume after morphological operations

logical array

Volume after morphological operations, returned as a logical array of the same size as input volume `V`.

Tips

- To perform the morphological operations erosion or dilation on 3-D volumes, use the `imerode` or `imdilate` functions, specifying the structuring element `ones(3,3,3)`.
- To perform morphological closing, opening, top-hat filtering, or bottom-hat filtering on 3-D volumes, use the `imclose`, `imopen`, `imtophat`, or `imbothat` functions, specifying the structuring element `ones(3,3,3)`.

See Also

`imdilate` | `imerode` | `imclose` | `imopen` | `imbothat` | `imtophat` | `bwmorph` | `bwskel`

Topics

“Types of Morphological Operations”

Introduced in R2018a

bwpack

Pack binary image

Syntax

```
BWP = bwpack(BW)
```

Description

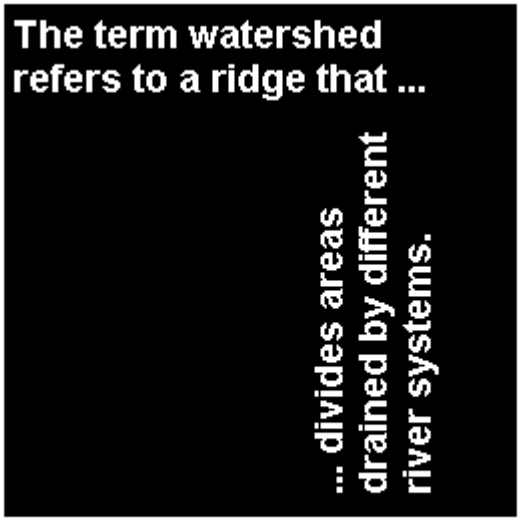
`BWP = bwpack(BW)` packs the binary image `BW` into the `uint32` array `BWP`, which is known as a *packed binary image*. Because each pixel value in the binary image has only two possible values, `1` and `0`, `bwpack` can map each pixel to a single bit in the packed output image.

Examples

Pack, Dilate, and Unpack Binary Image

Read binary image into the workspace.

```
BW = imread('text.png');
imshow(BW)
```



Pack the image.

```
BWp = bwpack(BW);
```

Dilate the packed image.

```
BWp_dilated = imdilate(BWp,ones(3,3),'ispacked');
```

Unpack the dilated image and display it.

```
BW_dilated = bwunpack(BWp_dilated, size(BW,1));
imshow(BW_dilated)
```

Input Arguments

BW — Binary image

2-D numeric matrix | 2-D logical matrix

Binary image, specified as a 2-D numeric or logical matrix. For numeric input, any nonzero pixels are considered to be 1 (`true`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

BWP — Packed binary image

numeric matrix

Packed binary image, returned as a numeric matrix of type `uint32`.

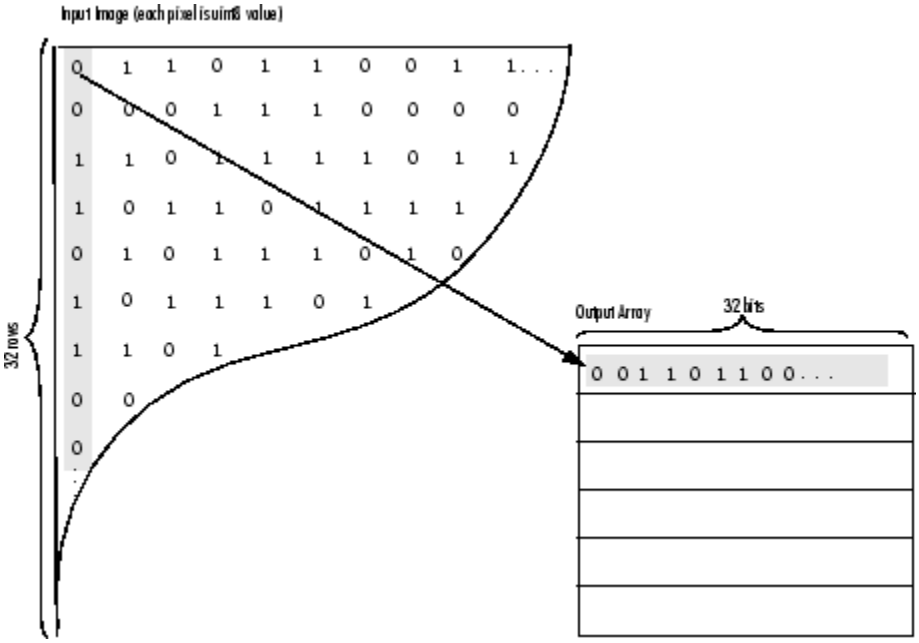
Data Types: `uint32`

Tips

- Binary image packing is used to accelerate some binary morphological operations, such as dilation and erosion. If the input to `imdilate` or `imerode` is a packed binary image, then the function uses a specialized routine to perform the operation faster.
- Use `bwunpack` to unpack packed binary images.

Algorithms

`bwpack` processes the input image pixels by column, mapping groups of 32 pixels into the bits of a `uint32` value. The first pixel in the first row corresponds to the least significant bit of the first `uint32` element of the output array. The first pixel in the 32nd input row corresponds to the most significant bit of this same element. The first pixel of the 33rd row corresponds to the least significant bit of the second output element, and so on. If `BW` is `M`-by-`N`, then `BWP` is `ceil(M/32)-by-N`. This figure illustrates how `bwpack` maps the pixels in a binary image to the bits in a packed binary image.



Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `bwpack` supports the generation of C code (requires MATLAB Coder). The code generated for `bwpack` uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

See Also

`bwunpack` | `imdilate` | `imerode`

Introduced before R2006a

bwperim

Find perimeter of objects in binary image

Syntax

```
BW2 = bwperim(BW)
BW2 = bwperim(BW,conn)
bwperim( ___ )
```

Description

`BW2 = bwperim(BW)` returns a binary image that contains only the perimeter pixels of objects in the input image `BW`. A pixel is part of the perimeter if it is nonzero and it is connected to at least one zero-valued pixel.

`BW2 = bwperim(BW,conn)` also specifies the pixel connectivity, `conn`.

`bwperim( ___ )` without output arguments displays the binary image of the perimeter in a new figure window. To use this syntax, `BW` must be a 2-D binary image.

Examples

Find Perimeter of Objects in Binary Image

Read binary image into workspace.

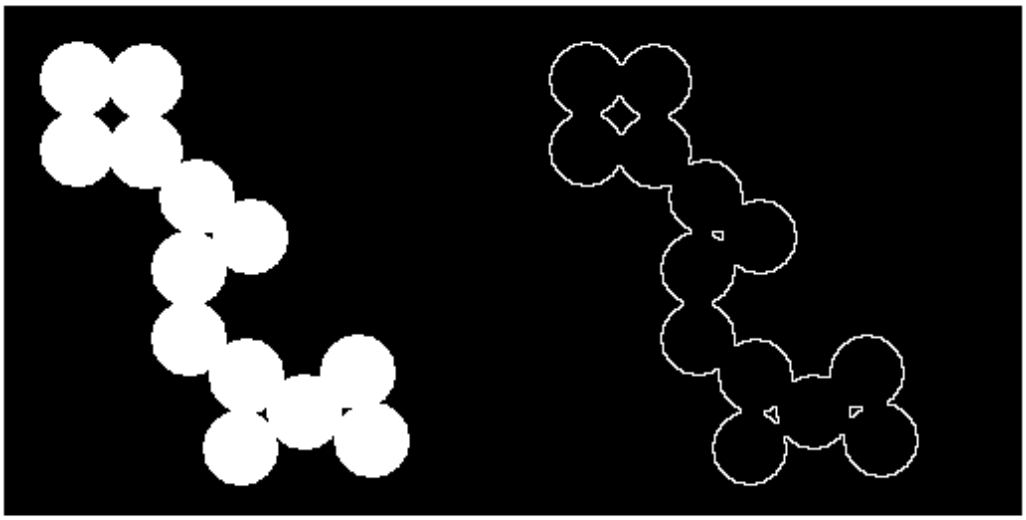
```
BW = imread('circles.png');
```

Calculate the perimeters of objects in the image.

```
BW2 = bwperim(BW,8);
```

Display the original image and the perimeters side-by-side.

```
imshowpair(BW,BW2,'montage')
```



Find Perimeter Pixels in Binary Image

This example shows how to find the perimeter pixels in a binary image using the `bwperim` function.

Read a binary image into the workspace.

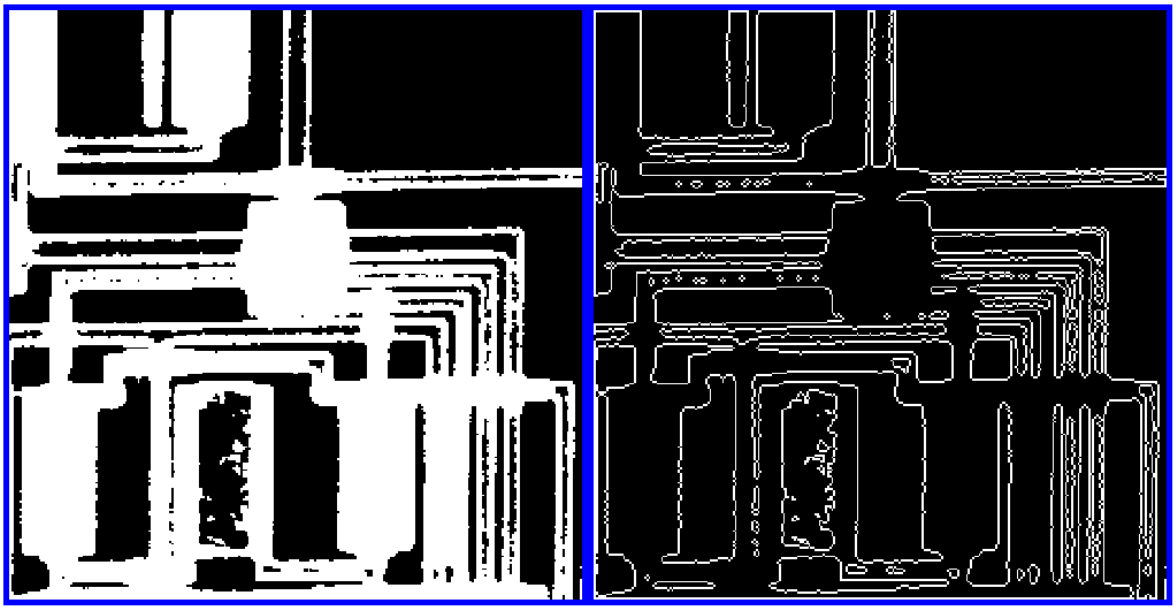
```
BW1 = imread('circbw.tif');
```

Find the perimeters of objects in the image.

```
BW2 = bwperim(BW1);
```

Display the original image and the image showing perimeters side-by-side.

```
montage({BW1,BW2},'BackgroundColor','blue','BorderSize',5)
```



Input Arguments

BW — Input binary image

numeric array | logical array

Input binary image, specified as a numeric or logical array of any dimension. For numeric input, any nonzero pixels are considered to be 1 (`true`).

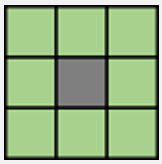
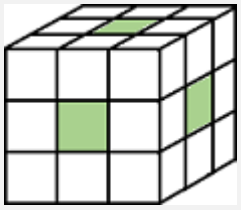
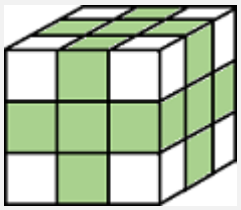
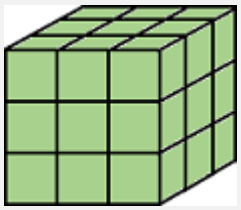
Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

conn — Pixel connectivity

4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 matrix of 0s and 1s

Pixel connectivity, specified as one of the values in this table. The default connectivity is 4 for 2-D images, and 6 for 3-D images.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal or vertical direction.	 Current pixel is shown in gray.

Value	Meaning	
8	Pixels are connected if their edges or corners touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		
6	Pixels are connected if their faces touch. Two adjoining pixels are part of the same object if they are both on and are connected in: • One of these directions: in, out, left, right, up, and down	 Current pixel is center of cube.
18	Pixels are connected if their faces or edges touch. Two adjoining pixels are part of the same object if they are both on and are connected in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. Two adjoining pixels are part of the same object if they are both on and are connected in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `bwperim` uses the default value `conndef(ndims(BW),"minimal")`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Data Types: `double` | `logical`

Output Arguments

`BW2` — Output binary image containing only perimeter pixels of objects
logical array

Output image containing only perimeter pixels of objects, returned as a logical array.

Data Types: `logical`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `bwperim` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `bwperim` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see "Types of Code Generation Support in Image Processing Toolbox".
- `bwperim` supports only 2-D images.
- `bwperim` does not support a no-output-argument syntax.
- The connectivity matrix input argument, `conn`, must be a constant.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- `bwperim` supports only 2-D images.
- `bwperim` does not support a no-output-argument syntax.
- The connectivity matrix input argument, `conn`, must be a constant.

See Also
`bwarea` | `imfill` | `conndef` | `bweuler` | `bwboundaries` | `bwtraceboundary` | `bwferet`

Topics
"Types of Morphological Operations"
"Pixel Connectivity"

Introduced before R2006a

bwpropfilt

Extract objects from binary image using properties

Syntax

```
BW2 = bwpropfilt(BW,prop,range)
BW2 = bwpropfilt(BW,prop,n)
BW2 = bwpropfilt(BW,prop,n,keep)
BW2 = bwpropfilt(BW,I,prop,___)
BW2 = bwpropfilt( ___ ,conn)
```

Description

`BW2 = bwpropfilt(BW,prop,range)` extracts all connected components (objects) from a binary image `BW` whose value of property `prop` is in the specified `range`. `bwpropfilt` returns a binary image `BW2` containing only those objects that meet the criteria.

`BW2 = bwpropfilt(BW,prop,n)` sorts the objects based on the value of the specified property, `prop`, returning a binary image that contains only the top `n` largest objects. In the event of a tie for `n`-th place, `bwpropfilt` keeps only the first `n` objects in `BW2`.

`BW2 = bwpropfilt(BW,prop,n,keep)` specifies whether to keep the `n` largest objects or the `n` smallest objects when sorted by property `prop`.

`BW2 = bwpropfilt(BW,I,prop,___)` sorts objects based on the intensity values in the grayscale image `I` and the property `prop`.

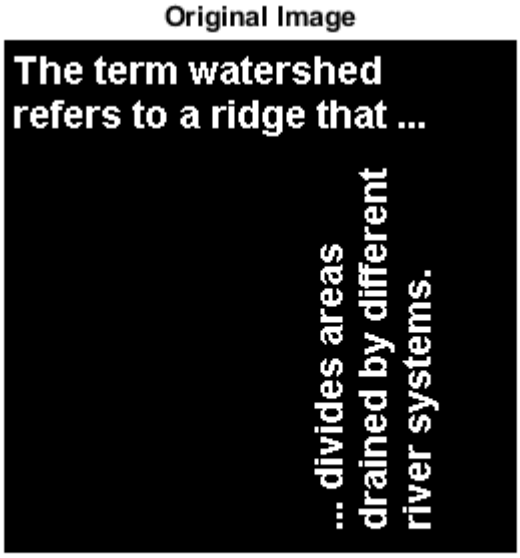
`BW2 = bwpropfilt( ___ ,conn)` specifies the pixel connectivity, `conn`.

Examples

Find Regions Without Holes

Read image and display it.

```
BW = imread('text.png');
figure
imshow(BW)
title('Original Image')
```



Use filtering to create a second image that contains only those regions in the original image that do not have holes. For these regions, the Euler number property is equal to 1. Display filtered image.

```
BW2 = bwpropfilt(BW,'EulerNumber',[1 1]);
figure
imshow(BW2)
title('Regions with Euler Number == 1')
```

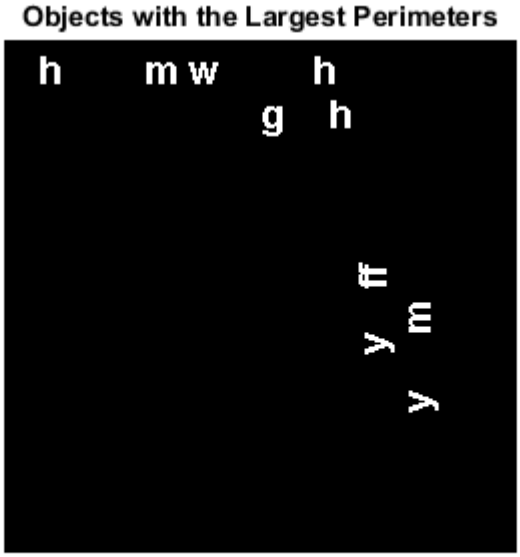
Find Which Ten Objects Have Largest Perimeters

Read image.

```
BW = imread('text.png');
```

Find the ten objects in the image with the largest perimeters and display filtered image.

```
BW2 = bwpropfilt(BW,'perimeter',10);
figure;
imshow(BW2)
title('Objects with the Largest Perimeters')
```



Input Arguments

`BW` — Image to be filtered

binary image

Image to be filtered, specified as a binary image.

Data Types: `logical`

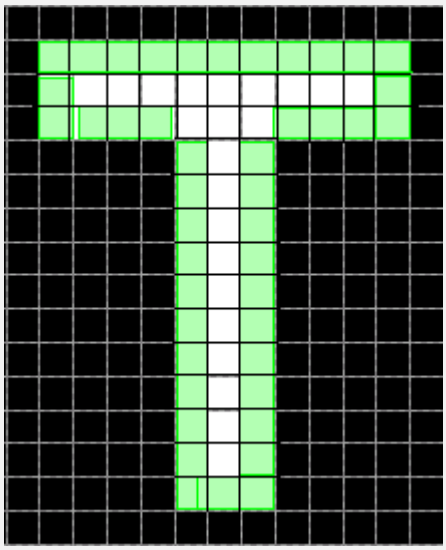
`prop` — Name of property on which to filter

character vector | string scalar

Name of property on which to filter, specified as one of these values.

Pixel Value Measurements

Property Name	Description
`"Area"`	Number of pixels in the region.
`"ConvexArea"`	Number of pixels in the convex hull. The convex hull is the smallest convex polygon that can contain the region. For more information about classifying pixels on the boundary of the hull, see “Classify Pixels That Are Partially Enclosed by ROI”.
`"Eccentricity"`	Eccentricity of the ellipse that has the same second-moments as the region. The eccentricity is the ratio of the distance between the foci of the ellipse and its major axis length. Values are numbers in the range [0, 1]. (0 and 1 are degenerate cases. An ellipse whose eccentricity is 0 is actually a circle, while an ellipse whose eccentricity is 1 is a line segment.)
`"EquivDiameter"`	Diameter (in pixels) of a circle with the same area as the region, calculated as `sqrt(4*Area/pi)`.
`"EulerNumber"`	Euler number (also known as the Euler characteristic), calculated as 1 minus the number of holes in the region.
`"Extent"`	Ratio of pixels in the region to pixels in the total bounding box, calculated as `Area` divided by the area of the bounding box.
`"FilledArea"`	Number of pixels in the region after filling all holes in the region.
`"MajorAxisLength"`	Length (in pixels) of the major axis of the ellipse that has the same normalized second central moments as the region.
`"MinorAxisLength"`	Length (in pixels) of the minor axis of the ellipse that has the same normalized second central moments as the region.
`"Orientation"`	Angle (in degrees) between the *x*-axis and the major axis of the ellipse that has the same second-moments as the region. The value is in the range (–90, 90]. This figure illustrates the axes and orientation of the ellipse. The left side of the figure shows an image region and its corresponding ellipse. The right side shows the same ellipse with the solid blue lines representing the axes. The red dots are the foci. The orientation is the angle between the horizontal dotted line and the major axis. 

Property Name	Description
`"Perimeter"`	Distance (in pixels) around the boundary of the region, calculated by adding the distance between each adjoining pair of pixels around the border of the region. This figure illustrates the pixels included in the perimeter calculation for a sample region. 
`"Solidity"`	Proportion of the pixels in the convex hull that are also in the region, calculated as `Area/ConvexArea`.

You can specify a pixel value measurement property from this table when you include a marker image, `I`, as an input argument to the function.

Pixel Value Measurements

Property Name	Description
`"MaxIntensity"`	Value of the pixel with the greatest intensity in the region.
`"MeanIntensity"`	Mean of all the intensity values in the region.
`"MinIntensity"`	Value of the pixel with the lowest intensity in the region.

Data Types: `char` | `string`

`range` — Minimum and maximum property values
1-by-2 numeric vector

Minimum and maximum property values, specified as a 1-by-2 numeric vector of the form [*low high*]. Values must be nondecreasing.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`n` — Number of objects to return
positive integer

Number of objects to return, specified as a positive integer.

Data Types: `double`

keep — Objects to retain
`"largest"` (default) | `"smallest"`

Objects to retain, specified as `"largest"` or `"smallest"`.

Data Types: `char` | `string`

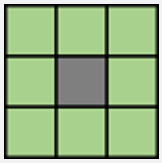
I — Marker image
grayscale image

Marker image, specified as a grayscale image of the same size as the input binary image. Intensity values in the grayscale image define regions in the input binary image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

conn — Pixel connectivity
8 (default) | 4 | 3-by-3 matrix of `0`s and `1`s

Pixel connectivity, specified as one of these values.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.

Connectivity can also be defined in a more general way by specifying a 3-by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. The matrix must be symmetric about its center element.

Data Types: `double` | `logical`

Output Arguments

BW2 — Filtered image
binary image

Filtered image, returned as a binary image of the same size as `BW`.

Tips

- `bwpropfilt` finds the connected components using the `bwconncomp` function. `bwpropfilt` then calculates the properties of those connected components using the `regionprops` function.

See Also

Image Region Analyzer | `regionprops` | `bwareafilt` | `bwareaopen` | `bwconncomp` | `conndef`

Topics

“Filter Images on Properties Using Image Region Analyzer App”

Introduced in R2014b

bwselect

Select objects in binary image

Syntax

```
BW2 = bwselect(BW,c,r)
BW2 = bwselect(BW,c,r,n)
[BW2,idx] = bwselect( ___ )
[x,y,BW2,idx,xi,yi] = bwselect( ___ )
[ ___ ] = bwselect(x,y,BW,xi,yi,n)

[ ___ ] = bwselect(BW,n)
[ ___ ] = bwselect
```

Description

`BW2 = bwselect(BW,c,r)` returns a binary image containing the objects that overlap the pixel (`r`, `c`). Objects are connected sets of `on` pixels, that is, pixels having a value of `1`.

`BW2 = bwselect(BW,c,r,n)` also specifies the object connectivity, `n`, as 4-connected or 8-connected.

`[BW2,idx] = bwselect( ___ )` returns the linear indices of the pixels belonging to the selected objects.

`[x,y,BW2,idx,xi,yi] = bwselect( ___ )` returns the *x* and *y* extents of the image and the (`xi`, `yi`) coordinates of the pixels. By default, `bwselect` uses the intrinsic coordinate system so that *x* and *y* are the image `XData` and `YData`.

`[ ___ ] = bwselect(x,y,BW,xi,yi,n)` establishes a nondefault world coordinate system for `BW` from the vectors `x` and `y`. The arguments `xi` and `yi` specify pixel coordinates in the world coordinate system.

`[ ___ ] = bwselect(BW,n)` displays the image `BW` in a figure and lets you select the (`r`, `c`) coordinates interactively using the mouse. With this syntax and the other interactive syntax, `bwselect` blocks the MATLAB command line until you finish selecting points.

For more information about selecting points interactively, see “Interactive Behavior” on page 1-487.

`[ ___ ] = bwselect` without an input argument lets you select the (`r`, `c`) coordinates of the image in the current axes interactively.

Examples

Select Objects in Binary Image

Select objects in a binary image and create a new image containing only those objects.

Read binary image into the workspace.

```
BW = imread('text.png');
```

Specify the locations of objects in the image using row and column indices.

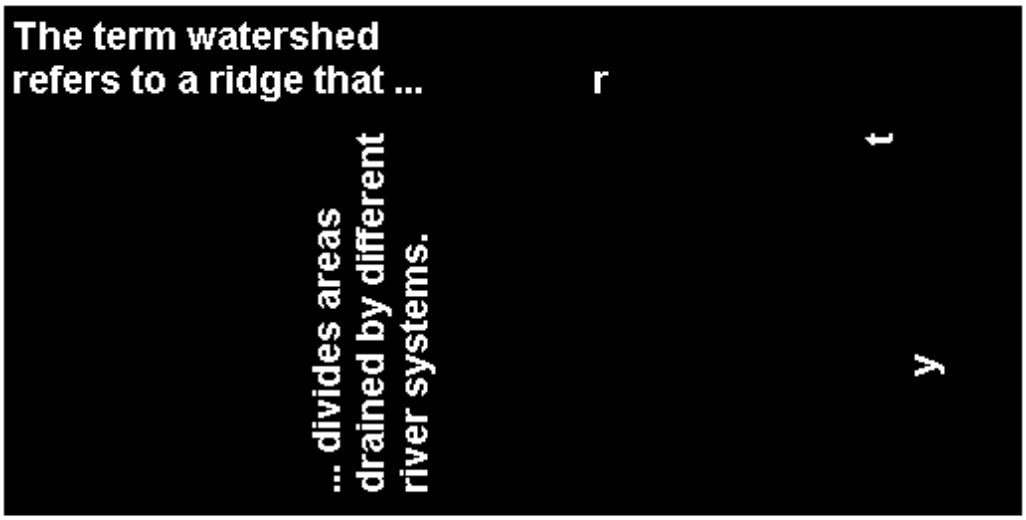
```
c = [43 185 212];
r = [38 68 181];
```

Create a new binary image containing only the selected objects. This example specifies 4-connected objects.

```
BW2 = bwselect(BW,c,r,4);
```

Display the original image and the new image side-by-side.

```
imshowpair(BW,BW2,'montage');
```



Input Arguments

BW — Binary image

2-D numeric matrix | 2-D logical matrix

Binary image, specified as a 2-D numeric matrix or 2-D logical matrix.

Example: `BW = imread('text.png');`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

c — Column index

numeric scalar | numeric vector

Column index, specified as a numeric scalar or numeric vector. If `r` and `c` are equal-length vectors, then `BW2` contains the sets of objects overlapping with any of the pixels `(r(k),c(k))`.

Example: `c = [43 185 212];`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`r` — Row index
numeric scalar | numeric vector

Row index, specified as a numeric scalar or numeric vector. If `r` and `c` are equal-length vectors, then `BW2` contains the sets of objects overlapping with any of the pixels `(r(k),c(k))`.

Example: `r = [38 68 181];`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`n` — Connectivity
8 (default) | 4

Connectivity, specified as 4 or 8.

Value	Description
4	4-connected objects
8	8-connected objects

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`x` — World *x*-axis coordinates
numeric scalar | numeric vector

World *x*-axis coordinates, specified as a numeric scalar or numeric vector of the same length as `y`. Use *x* and *y* to establish a nondefault spatial coordinate system. By default, if you do not specify *x* and *y*, then `bwselect` uses the intrinsic coordinate system in which `x` is [1, `size(BW,2)`].

Example: `x = [19.5 23.5];`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`y` — World *y*-axis coordinates
numeric scalar | numeric vector

World *y*-axis coordinates, specified as a numeric scalar or numeric vector of the same length as `x`. Use *x* and *y* to establish a nondefault spatial coordinate system. By default, if you do not specify *x* and *y*, then `bwselect` uses the intrinsic coordinate system in which `y` is [1, `size(BW,1)`].

Example: `y = [8.0 12.0];`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`xi` — *x*-coordinates of points
numeric scalar | numeric vector

x-coordinates of points in the world coordinate system, specified as a numeric scalar or numeric vector.

Example: `x = [19.5 23.5];`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`yi` — *y*-coordinates of points
numeric scalar | numeric vector

y-coordinates of points in the world coordinate system, specified as a numeric scalar or numeric vector.

Example: `y = [8.0 12.0];`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`BW2` — Binary image containing objects that overlap specified pixels
logical array

Binary image containing objects that overlap the specified pixels, returned as a logical array. `BW2` contains the set of objects overlapping with any of the pixels specified by `r` and `c` or `xi` and `yi`.

If you do not specify an output argument, then `bwselect` displays the output image in a new figure.

`idx` — Linear indices of pixels belonging to selected objects
numeric vector

Linear indices of the pixels belonging to the selected objects, returned as a numeric vector.

More About

Interactive Behavior

When you run `bwselect` without specifying pixel coordinates, `bwselect` enables you to select points interactively from an image in a figure window. Select points using these commands.

Interactive Behavior	Description
Add points	Left-click points in the image.
Remove previous point	Press **Backspace** or **Delete**.
Add final point and complete selection	Right-click, double-click, or press **Shift** and left-click simultaneously.
Complete selection without adding final point	Press **Return**.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `bwselect` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `bwselect` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance

optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

- When generating code, `bwselect` only supports the following syntaxes:
  - `BW2 = bwselect(BW,c,r)`
  - `[BW2,idx] = bwselect(BW,c,r)`
  - `BW2 = bwselect(BW,c,r,n)`
  - `[BW2,idx] = bwselect(BW,c,r,n)`
- In addition, the optional fourth input argument, `n`, must be a compile-time constant.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, `bwselect` supports only these syntaxes:
  - `BW2 = bwselect(BW, c, r)`
  - `[BW2, idx] = bwselect(BW, c, r)`
  - `BW2 = bwselect(BW, c, r, n)`
  - `[BW2, idx] = bwselect(BW, c, r, n)`
- In addition, the optional fourth input argument, `n`, must be a compile-time constant.

See Also

`bwselect3` | `bwlabel` | `imfill` | `grayconnected` | `roipoly` | `regionfill`

Topics
“Image Coordinate Systems”
“Shift X- and Y-Coordinate Range of Displayed Image”

Introduced before R2006a

bwselect3

Select objects in binary volume

Syntax

```
J = bwselect3(V,c,r,p)
J = bwselect3(V,c,r,p,n)
[J,idx] = bwselect3( ___ )
[x,y,z,J,idx,xi,yi,zi] = bwselect3( ___ )
[ ___ ] = bwselect3(x,y,z,V,xi,yi,zi)
```

Description

`J = bwselect3(V,c,r,p)` returns the binary volume `J` containing the objects that overlap the voxel (`r`,`c`,`p`). Objects are connected sets of voxels with the value `1`.

`J = bwselect3(V,c,r,p,n)` also specifies the connectivity, `n`, used to define objects.

`[J,idx] = bwselect3( ___ )` returns in `idx` the linear indices of voxels belonging to the selected objects.

`[x,y,z,J,idx,xi,yi,zi] = bwselect3( ___ )` also returns the *x*, *y*, and *z* extents of the binary volume and the (`xi`,`yi`,`zi`) coordinates of selected voxels. By default, `bwselect3` uses the intrinsic coordinate system so that `x`,`y`, and `z` are the volume `XData`, `YData`, and `ZData`.

`[ ___ ] = bwselect3(x,y,z,V,xi,yi,zi)` establishes a nondefault world coordinate system for `V` from the vectors `x`, `y`, and *z*. The arguments `xi`, `yi`, and `zi` specify voxel coordinates in the world coordinate system.

Examples

Find Objects in Volume

Load a volume and change its name to `V`.

```
load mristack;
V = mristack;
```

Define a set of points in the volume.

```
C = [126 87 11];
R = [34 120 20];
P = [20 2 12];
```

Return a volume that contains objects that intersect with the points specified.

```
J = bwselect3(V,C,R,P);
```

Input Arguments

`V` — Binary volume

3-D numeric array | 3-D logical array

Binary volume, specified as a 3-D numeric array or 3-D logical array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`r` — Row index of voxels

numeric scalar | numeric vector

Row index of voxels in objects of interest, specified as a numeric scalar or numeric vector. If you specify a vector, then `r` must be the same length as `c` and `p`. The output binary volume `J` contains the sets of objects overlapping with any of the voxels (`r`(k), `c`(k), `p`(k)), where k is an index into the vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`c` — Column index of object

numeric scalar | numeric vector

Column index of voxels in objects of interest, specified as a numeric scalar or numeric vector. If you specify a vector, then `c` must be the same length as `r` and `p`. The output binary volume `J` contains the sets of objects overlapping with any of the voxels (`r`(k), `c`(k), `p`(k)), where k is an index into the vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`p` — Plane index of object

numeric scalar | numeric vector

Plane index of voxels in objects of interest, specified as a numeric scalar or numeric vector. If you specify a vector, then `p` must be the same length as `r` and `c`. The output binary volume `J` contains the sets of objects overlapping with any of the voxels (`r`(k), `c`(k), `p`(k)), where k is an index into the vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`n` — Connectivity

26 (default) | 6 | 18

Connectivity, specified as one of these values.

Connectivities

Value	Connectivity
6	6-connected objects (Face-Face)
18	18-connected objects (Face-Face and Edge-Edge)
26	26-connected objects (Face-Face, Edge-Edge, and Vertex-Vertex)

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`x` — World *x*-axis coordinates

numeric scalar | numeric vector

World *x*-axis coordinates, specified as a numeric scalar or numeric vector of the same length as `y` and `z`. Use *x*, *y*, and *z* to establish a nondefault spatial coordinate system. If you do not specify a coordinate system, then by default `bwselect3` uses the intrinsic coordinate system in which `x` is [1, `size(J,2)`].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`y` — World *y*-axis coordinates
numeric scalar | numeric vector

World *y*-axis coordinates, specified as a numeric scalar or numeric vector of the same length as `x` and `z`. Use *x*, *y*, and *z* to establish a nondefault spatial coordinate system. If you do not specify a coordinate system, then by default `bwselect3` uses the intrinsic coordinate system in which `y` is [1, `size(J,1)`].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`z` — World *z*-axis coordinates
numeric scalar | numeric vector

World *z*-axis coordinates, specified as a numeric scalar or numeric vector of the same length as `x` and `y`. Use *x*, *y*, and *z* to establish a nondefault spatial coordinate system. If you do not specify a coordinate system, then by default `bwselect3` uses the intrinsic coordinate system in which `z` is [1, `size(J,3)`].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`xi` — *x*-coordinates of voxels
numeric scalar | numeric vector

x-coordinates of voxels in the world coordinate system, specified as a numeric scalar or numeric vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`yi` — *y*-coordinates of voxels
numeric scalar | numeric vector

y-coordinates of voxels in the world coordinate system, specified as a numeric scalar or numeric vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`zi` — *z*-coordinates of voxels
numeric scalar | numeric vector

z-coordinates of voxels in the world coordinate system, specified as a numeric scalar or numeric vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`J` — Binary volume containing objects that overlap specified voxels
3-D logical array

Binary volume containing objects that overlap specified voxels, returned as a 3-D logical array. `J` contains the set of objects overlapping with any of the voxels specified by `r`,`c`, and `p`, or `xi`,`yi`, and `zi`.

`idx` — Linear indices of voxels belonging to selected objects
numeric vector

Linear indices of the voxels belonging to the selected objects, returned as a numeric vector.

See Also

`bwlabel` | `imfill` | `roipoly` | `regionfill` | `bwselect`

Topics
“Image Coordinate Systems”
“Shift X- and Y-Coordinate Range of Displayed Image”

Introduced in R2017b

bwskel

Reduce all objects to lines in 2-D binary image or 3-D binary volume

Syntax

```
B = bwskel(A)
B = bwskel(V)
B = bwskel( ___ ,'MinBranchLength',N)
```

Description

`B = bwskel(A)` reduces all objects in the 2-D binary image `A` to 1-pixel wide curved lines, without changing the essential structure of the image. This process, called skeletonization, extracts the centerline while preserving the topology and Euler number (also known as the Euler characteristic) of the objects.

`B = bwskel(V)` returns the skeleton of a 3-D binary volume.

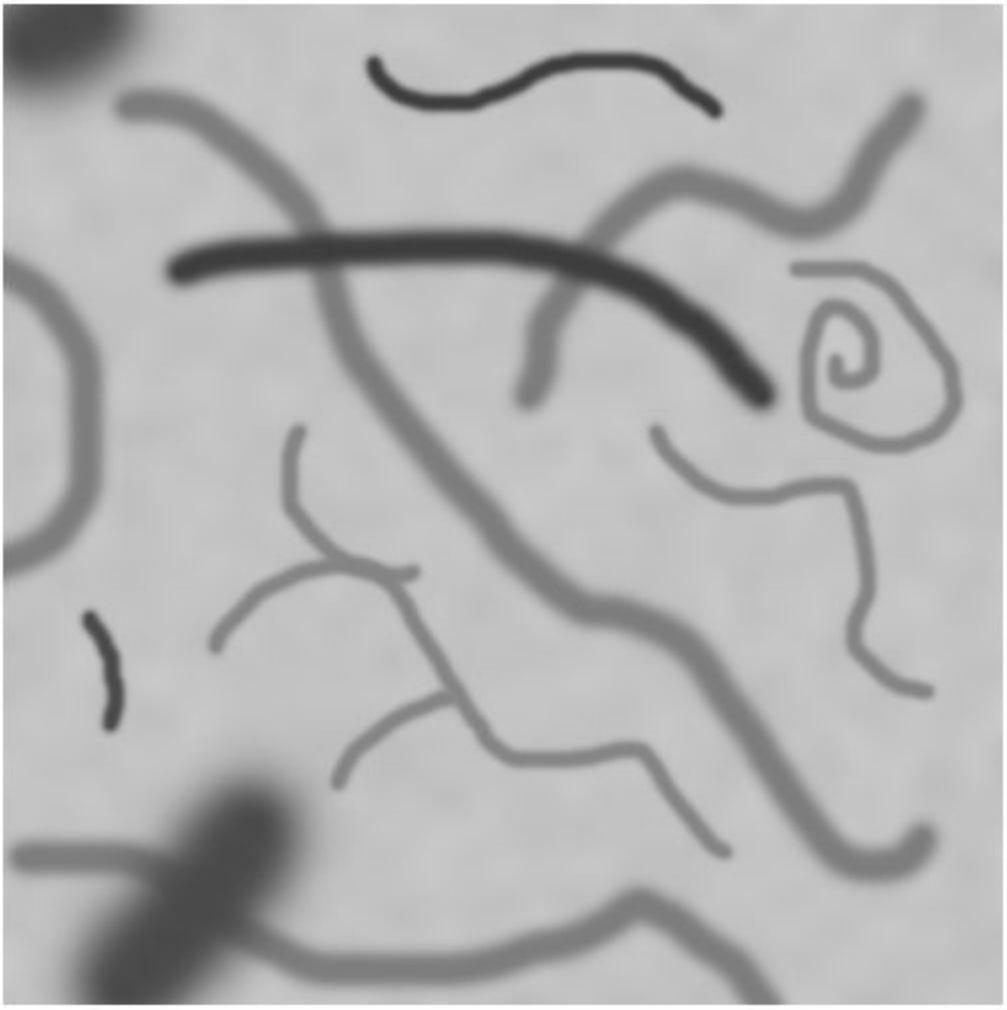
`B = bwskel( ___ ,'MinBranchLength',N)` specifies the minimum branch length `N` of the skeleton. `bwskel` removes (prunes) all branches shorter than the specified length. `bwskel` calculates the length as the number of pixels in a branch using 8-connectivity for 2-D and 26-connectivity for 3-D.

Examples

Skeletonize 2-D Grayscale Image

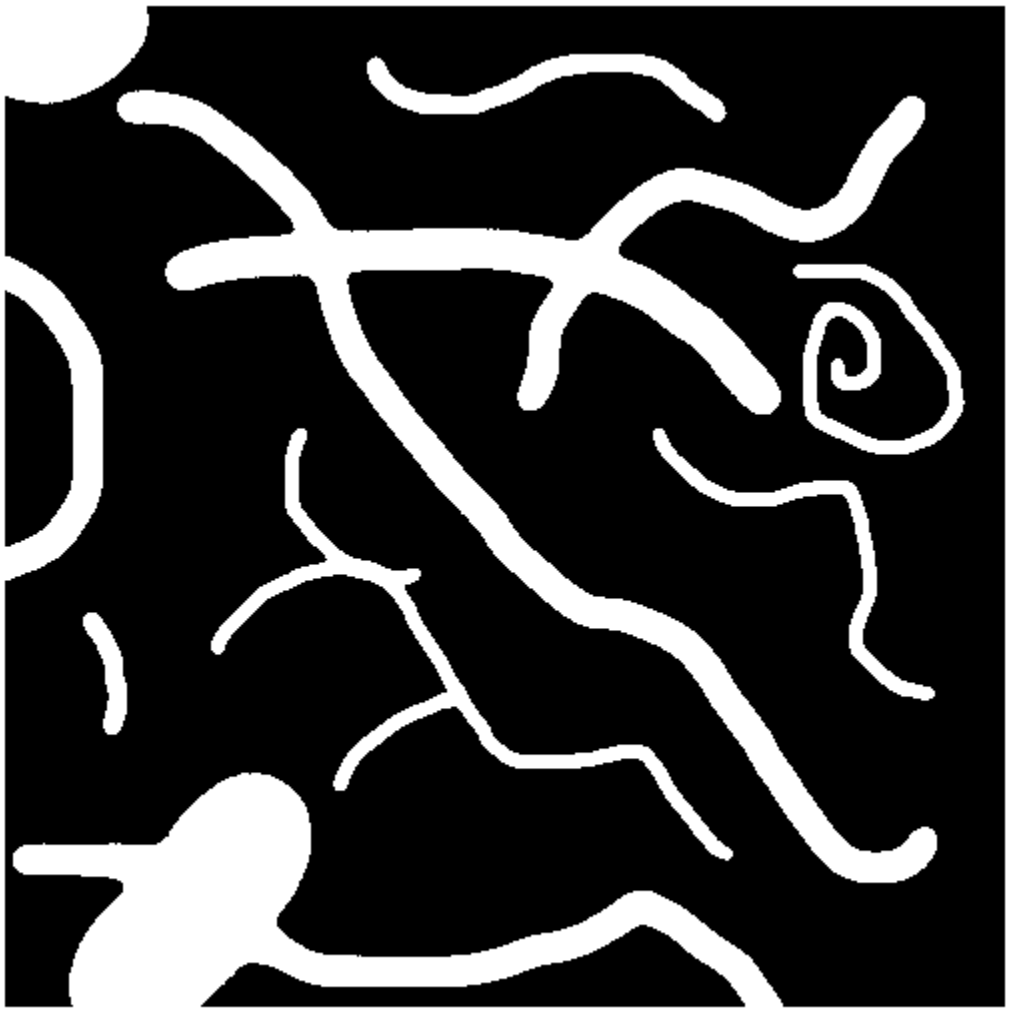
Read a 2-D grayscale image into the workspace. Display the image. Objects of interest are dark threads against a light background.

```
I = imread('threads.png');
imshow(I)
```



Skeletonization requires a binary image in which foreground pixels are `1` (white) and the background is `0` (black). To make the original image suitable for skeletonization, take the complement of the image so that the objects are light and the background is dark. Then, binarize the result.

```
Icomplement = imcomplement(I);
BW = imbinarize(Icomplement);
imshow(BW)
```

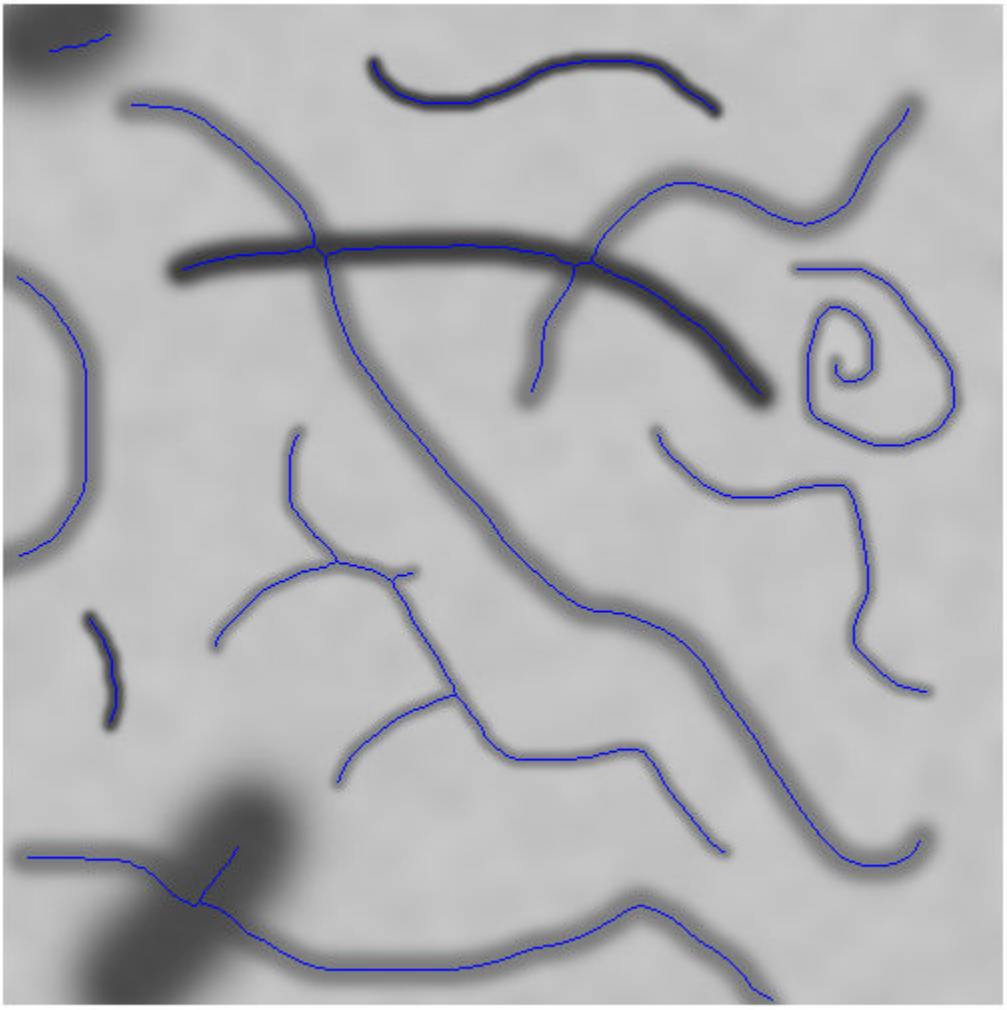


Perform skeletonization of the binary image using `bwskel`.

```
out = bwskel(BW);
```

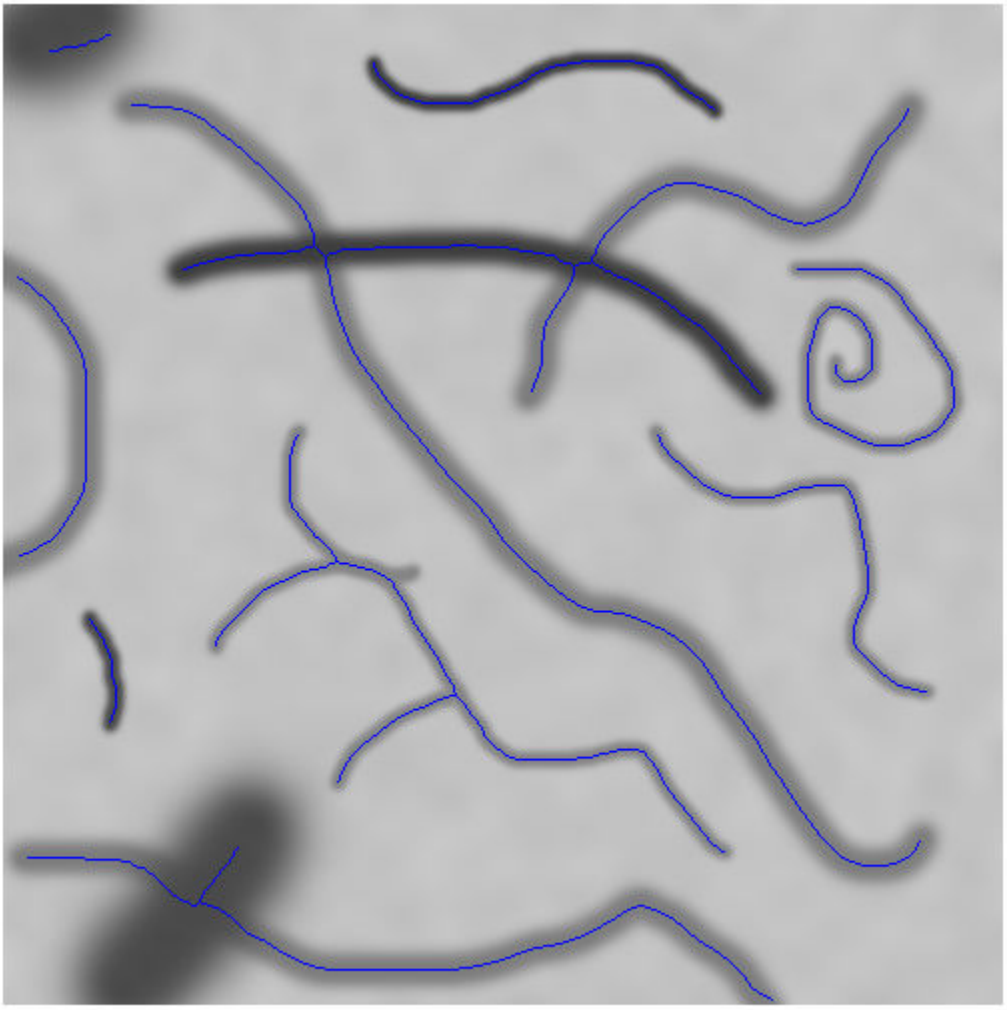
Display the skeleton over the original image by using the `labeloverlay` function. The skeleton appears as a 1-pixel wide blue line over the dark threads.

```
imshow(labeloverlay(I,out,'Transparency',0))
```



Prune small spurs that appear on the skeleton and view the result. One short branch is pruned from a thread near the center of the image.

```
out2 = bwskel(BW,'MinBranchLength',15);
imshow(labeloverlay(I,out2,'Transparency',0))
```



Skeletonize Binary Image

Read a binary image into the workspace.

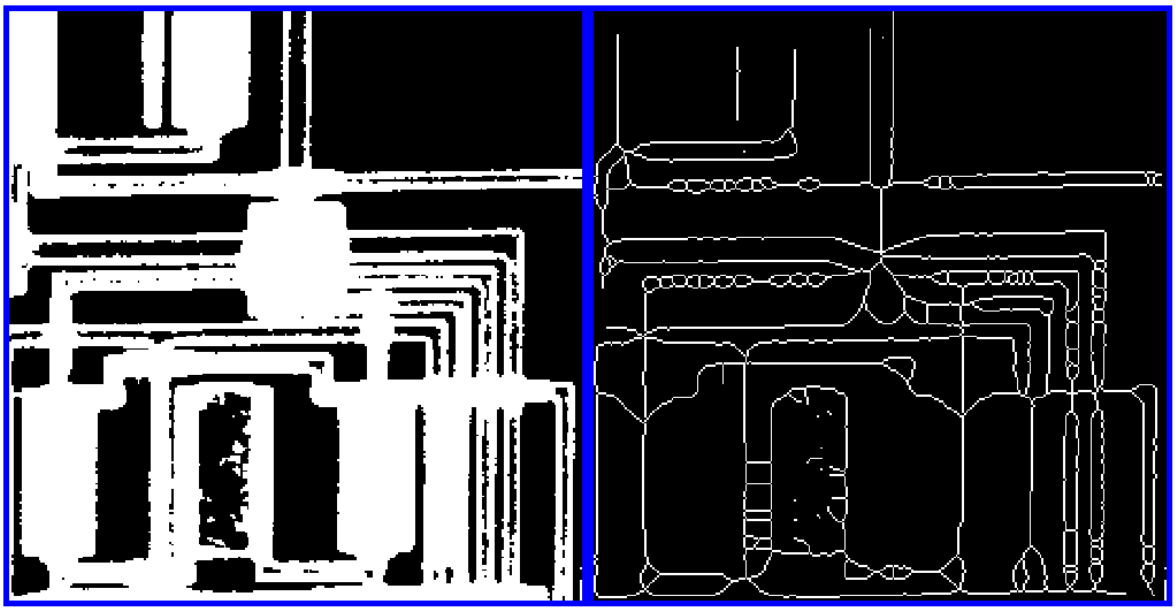
```
BW1 = imread('circbw.tif');
```

Skeletonize objects in the image by using the `bwskel` function.

```
BW2 = bwskel(BW1);
```

View the original image and the skeletonized image side by side.

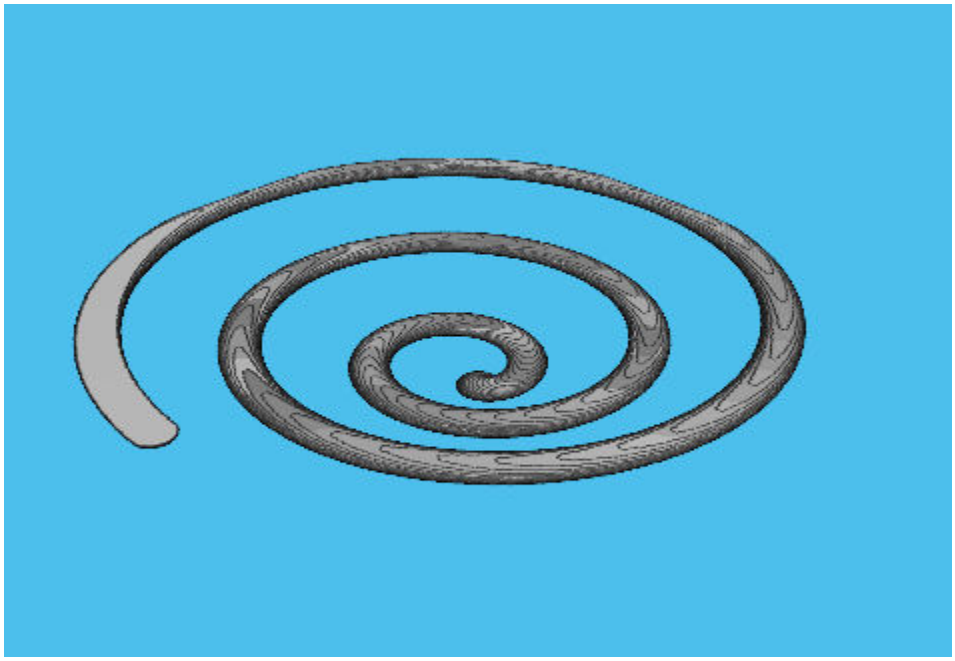
```
montage({BW1,BW2},'BackgroundColor','blue','BorderSize',5)
```



Skeletonize 3-D Volume

Load a volumetric data set into the workspace. The name of the data set is `spiralVol`. Display the volume using `volshow`.

```
load spiralVol.mat;
volshow(spiralVol);
```

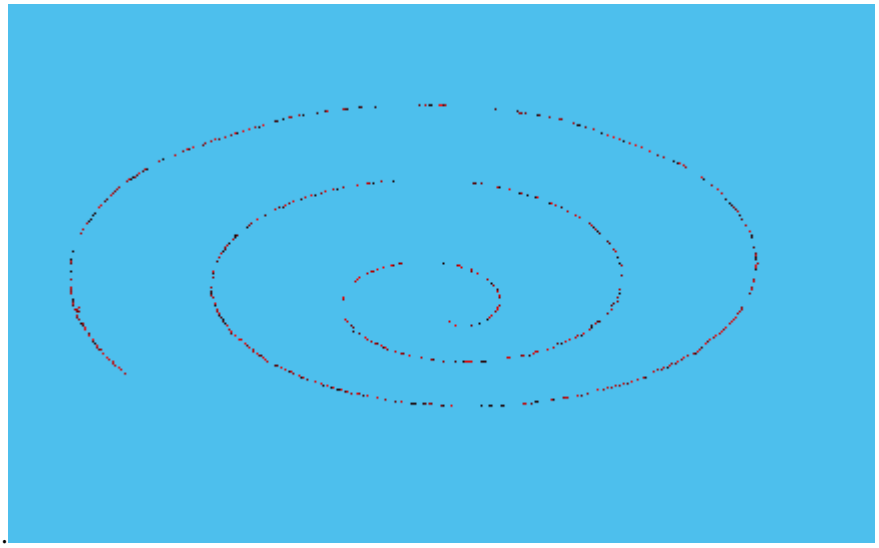


Convert the `spiralVol` data set to a binary format which is required by the `bwskel` function.

```
spiralVolLogical = imbinarize(spiralVol);
```

Skeletonize the spiral shape in the data set. Display the skeletonized volume with `volshow`.

```
spiralVolSkel = bwskel(spiralVolLogical);
```



Input Arguments

A — Binary image

2-D logical matrix

Binary image, specified as a 2-D logical matrix.

Data Types: `logical`

`V` — 3-D binary volume
3-D logical array

3-D binary volume, specified as a 3-D logical array.

Data Types: `logical`

`N` — Minimum branch length
`0` (default) | nonnegative integer

Minimum branch length, specified as a nonnegative integer. `bwskel` prunes branches shorter than `N`. By default, `bwskel` does not prune branches.

Output Arguments

`B` — Skeletonized image or volume
2-D logical matrix | 3-D logical array

Skeletonized image or volume, returned as a 2-D logical matrix or 3-D logical array of the same size as the input image or volume.

Tips

- While both `bwskel` and `bwmorph` can skeletonize 2-D images, you can get different results using `bwmorph` than when using `bwskel`. Because they use different algorithms, the `bwskel` function uses 4-connectivity with 2-D images; `bwmorph` uses 8-connectivity.
- `bwskel` assumes that foreground objects in the binary image are white (logical `true`). If your image has a white background and black objects, then use the complement of your image as the input to `bwskel`. You can compute the complement by using `imcomplement`.

Algorithms

- The `bwskel` function uses the medial axis transform.

References

[1] Ta-Chih Lee, Rangasami L. Kashyap and Chong-Nam Chu. *Building skeleton models via 3-D medial surface/axis thinning algorithms*. Computer Vision, Graphics, and Image Processing, 56(6):462-478, 1994.

[2] Kerschnitzki, M, Kollmannsberger, P, Burghammer, M. et al. *Architecture of the osteocyte network correlates with bone material quality*. Journal of Bone and Mineral Research, 28(8):1837-1845, 2013.

See Also
`bwmorph` | `bwmorph3`

Topics
“Types of Morphological Operations”

Introduced in R2018a

bwtraceboundary

Trace object in binary image

Syntax

```
B = bwtraceboundary(BW,P,fstep)
B = bwtraceboundary(BW,P,fstep,conn)
B = bwtraceboundary(BW,P,fstep,conn,m,dir)
```

Description

`B = bwtraceboundary(BW,P,fstep)` traces the outline of an object in binary image `BW`. Nonzero pixels belong to an object and zero-valued pixels constitute the background. `P` specifies the row and column coordinates of the point on the object boundary where you want the tracing to begin. `fstep` specifies the initial search direction for the next object pixel connected to `P`. `B` holds the row and column coordinates of the boundary pixels for the region.

`B = bwtraceboundary(BW,P,fstep,conn)` traces the boundary, where `conn` specifies the desired connectivity.

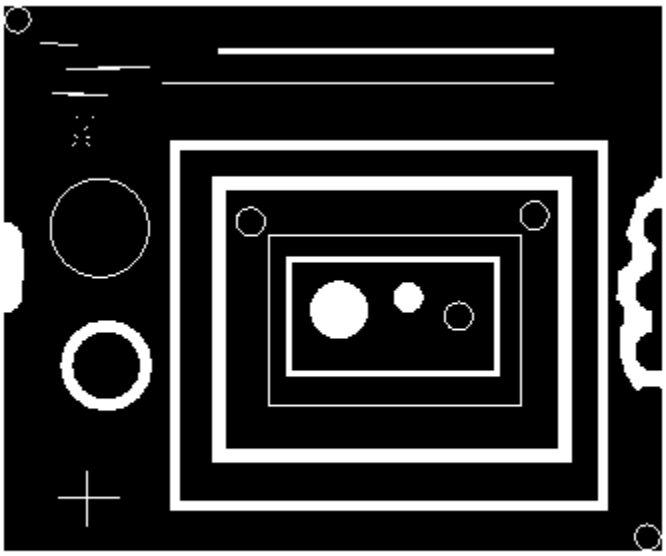
`B = bwtraceboundary(BW,P,fstep,conn,m,dir)` specifies `m`, the maximum number of boundary pixels to extract, and `dir`, the direction in which to trace the boundary. By default, `bwtraceboundary` identifies all the pixels on the boundary.

Examples

Trace Boundary and Visualize Contours

Read an image and display it.

```
BW = imread('blobs.png');
imshow(BW)
```

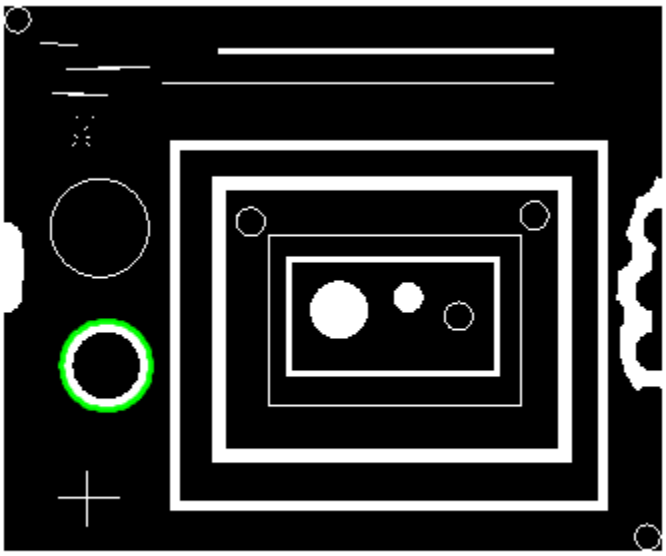


Pick an object in the image and trace the boundary. To select an object, specify a pixel on its boundary. This example uses the coordinates of a pixel on the boundary of the thick white circle, obtained through visual inspection using `impixelinfo`. By default, `bwtraceboundary` identifies all pixels on the boundary.

```
r1 = 163;
c1 = 37;
contour = bwtraceboundary(BW,[r1 c1],'W');
```

Plot the contour on the image.

```
hold on
plot(contour(:,2),contour(:,1),'g','LineWidth',2)
```



Pick a point on the boundary of a second object. This example uses the coordinates of a pixel near the upper-left corner of the largest rectangle. Trace the first fifty boundary pixels in the clockwise direction.

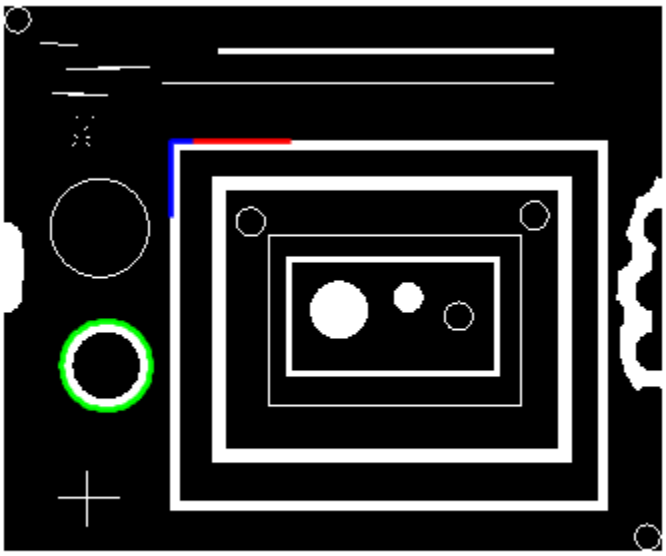
```
r2 = 68;
c2 = 95;
contourCW = bwtraceboundary(BW,[r2 c2],'W',8,50,'clockwise');
```

Starting at the same point on the second object boundary, trace the first fifty boundary pixels in the counterclockwise direction.

```
contourCCW = bwtraceboundary(BW,[r2 c2],'W',8,50,'counterclockwise');
```

Plot the clockwise contour on the image in red. Plot the counterclockwise contour on the image in blue.

```
plot(contourCW(:,2),contourCW(:,1),'r','LineWidth',2)
plot(contourCCW(:,2),contourCCW(:,1),'b','LineWidth',2)
```



Input Arguments

`BW` — Binary image

2-D numeric matrix | 2-D logical matrix

Binary image, specified as a 2-D numeric or logical matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`P` — Coordinates of starting point

2-element vector

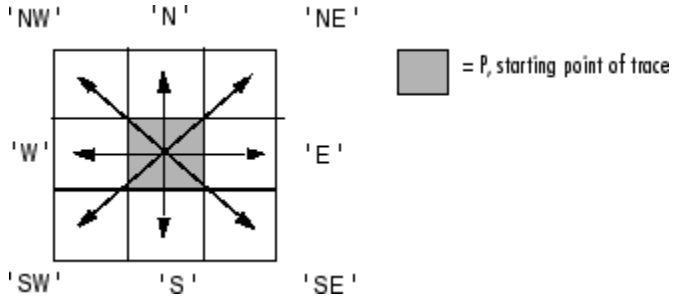
Coordinates of starting point on the object boundary where you want the tracing to begin, specified as a 2-element vector of the format `[row column]`.

Data Types: `double`

`fstep` — Initial search direction

`'N'` | `'NE'` | `'E'` | `'SE'` | `'S'` | `'SW'` | `'W'` | `'NW'`

Initial search direction for the next object pixel connected to `P`, specified as a character vector or string scalar as depicted in the diagram.



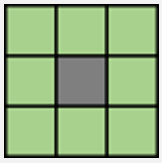
Note When the connectivity `conn` is `4`, `fstep` is limited to the values `'N'`, `'E'`, `'S'`, and `'W'`.

Data Types: `char` | `string`

`conn` — Pixel connectivity

`8` (default) | `4`

Pixel connectivity, specified as `8` or `4`.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. The neighborhood of a pixel are the adjacent pixels in the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. The neighborhood of a pixel are the adjacent pixels in the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.

Data Types: `double`

`m` — Maximum number of boundary pixels to extract

`Inf` (default) | positive integer

Maximum number of boundary pixels to extract, specified as a positive integer. By default, `m` is `Inf` and `bwtraceboundary` identifies all the pixels on the boundary.

Data Types: `double`

`dir` — Direction in which to trace boundary

`'clockwise'` (default) | `'counterclockwise'`

Direction in which to trace boundary, specified as `'clockwise'` or `'counterclockwise'`.

Data Types: `char` | `string`

Output Arguments

B — Row and column coordinates of boundary pixels

q-by-2 matrix

Row and column coordinates of the boundary pixels for the region, returned as a *q*-by-2 matrix. Each row in `B` has the form `[row column]`.

Algorithms

The `bwtraceboundary` function implements the Moore-Neighbor tracing algorithm modified by Jacob's stopping criteria. This function is based on the `boundaries` function presented in the first edition of *Digital Image Processing Using MATLAB*, by Gonzalez, R. C., R. E. Woods, and S. L. Eddins, New Jersey, Pearson Prentice Hall, 2004.

References

[1] Gonzalez, R. C., R. E. Woods, and S. L. Eddins, *Digital Image Processing Using MATLAB*, New Jersey, Pearson Prentice Hall, 2004.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `bwtraceboundary` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- When generating code, the `dir`, `fstep`, and `conn` arguments must be compile-time constants.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, the `dir`, `fstep`, and `conn` arguments must be compile-time constants.

See Also

`bwboundaries` | `bwperim`

Introduced before R2006a

bwulterode

Ultimate erosion

Syntax

```
BW2 = bwulterode(BW)
BW2 = bwulterode(BW,method)
BW2 = bwulterode( ___ ,conn)
```

Description

`BW2 = bwulterode(BW)` computes the ultimate erosion of the binary image `BW`. The ultimate erosion of `BW` consists of the regional maxima of the Euclidean distance transform of the complement of `BW`.

`BW2 = bwulterode(BW,method)` specifies the distance transform method.

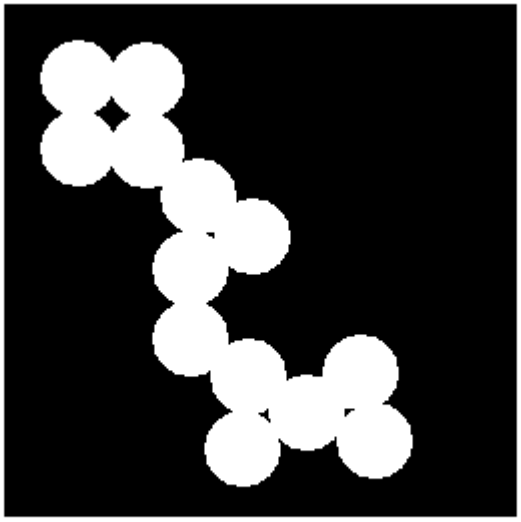
`BW2 = bwulterode( ___ ,conn)` specifies the pixel connectivity.

Examples

Perform Ultimate Erosion of Binary Image

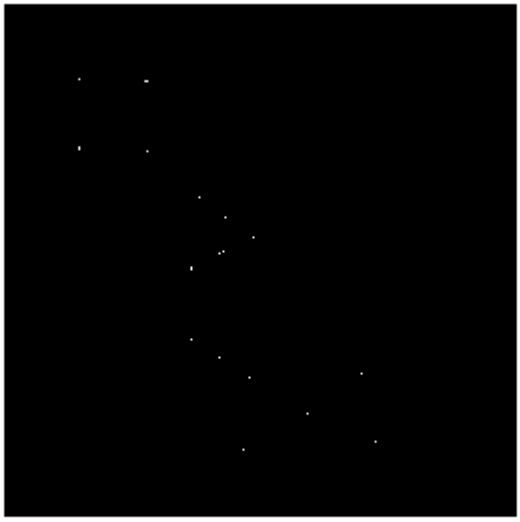
Read a binary image into the workspace and display it.

```
originalBW = imread('circles.png');
imshow(originalBW)
```



Perform the ultimate erosion of the image and display it.

```
ultimateErosion = bwulterode(originalBW);
figure, imshow(ultimateErosion)
```



Input Arguments

BW — Binary image

numeric array | logical array

Binary image, specified as a numeric or logical array of any dimension. For numeric input, any nonzero pixels are considered to be 1 (`true`).

Example: `BW = imread('circles.png');`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

method — Distance transform method

`'euclidean'` (default) | `'quasi-euclidean'` | `'cityblock'` | `'chessboard'`

Distance transform method, specified as one of the values in this table.

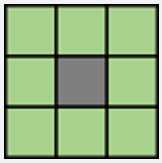
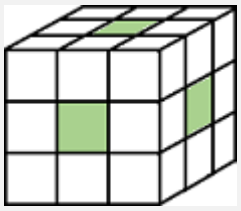
Method	Description
`'chessboard'`	In 2-D, the chessboard distance between (x_1,y_1) and (x_2,y_2) is $\max(\lvert x_1 - x_2\rvert, \lvert y_1 - y_2\rvert)$.

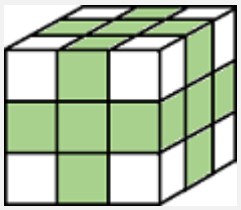
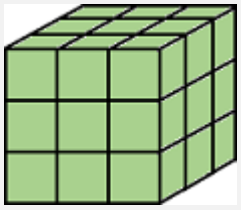
Method	Description
`'cityblock'`	In 2-D, the cityblock distance between (x_1,y_1) and (x_2,y_2) is $\lvert x_1 - x_2\rvert + \lvert y_1 - y_2\rvert$
`'euclidean'`	In 2-D, the Euclidean distance between (x_1,y_1) and (x_2,y_2) is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.
`'quasi-euclidean'`	In 2-D, the quasi-Euclidean distance between (x_1,y_1) and (x_2,y_2) is $\lvert x_1 - x_2\rvert + (\sqrt{2} - 1)\lvert y_1 - y_2\rvert,\ \lvert x_1 - x_2\rvert > \lvert y_1 - y_2\rvert$ $(\sqrt{2} - 1)\lvert x_1 - x_2\rvert + \lvert y_1 - y_2\rvert,$ otherwise.

For more information, see “Distance Transform of a Binary Image”.

conn — Pixel connectivity
4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 matrix of `0`s and `1`s

Pixel connectivity, specified as one of the values in this table. The default connectivity is `8` for 2-D images, and `26` for 3-D images.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. The neighborhood of a pixel are the adjacent pixels in the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. The neighborhood of a pixel are the adjacent pixels in the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		
6	Pixels are connected if their faces touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down	 Current pixel is shown in gray.

Value	Meaning	
18	Pixels are connected if their faces or edges touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `bwulterode` uses the default value `conndef(ndims(BW),'maximal')`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Data Types: `double` | `logical`

Output Arguments

`BW2` — Eroded image

logical array

Eroded image, returned as a logical array of the same size as `BW`.

Data Types: `logical`

See Also

`bwdist` | `conndef` | `imregionalmax`

Topics

“Distance Transform of a Binary Image”

Introduced before R2006a

bwunpack

Unpack binary image

Syntax

```
BW = bwunpack(BWP,m)
```

Description

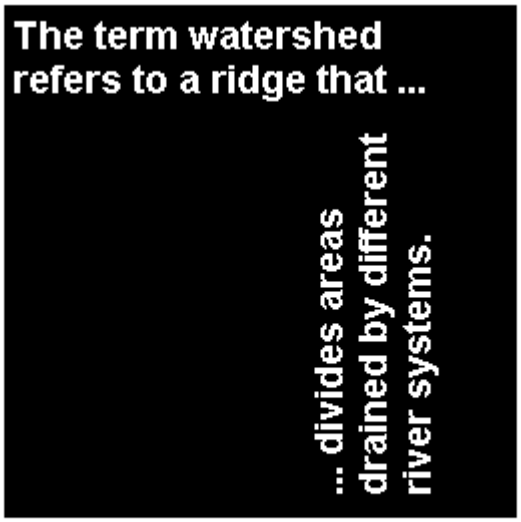
`BW = bwunpack(BWP,m)` unpacks the packed binary image `BWP` into binary image `BW` with `m` rows.

Examples

Pack, Dilate, and Unpack Binary Image

Read binary image into the workspace.

```
BW = imread('text.png');
imshow(BW)
```



Pack the image.

```
BWp = bwpack(BW);
```

Dilate the packed image.

```
BWp_dilated = imdilate(BWp,ones(3,3),'ispacked');
```

Unpack the dilated image and display it.

```
BW_dilated = bwunpack(BWp_dilated, size(BW,1));
imshow(BW_dilated)
```

Input Arguments

BWP — Packed binary image

2-D numeric matrix

Packed binary image, specified as a 2-D numeric array of data type `uint32`.

Data Types: `uint32`

m — Number of image rows

positive integer

Number of image rows, specified as a positive integer. The default value of `m` is `32*size(BWP,1)`.

Data Types: `uint32`

Output Arguments

BW — Unpacked binary image

`m`-by-`n` logical matrix

Unpacked binary image, returned as a logical matrix with `m` rows.

Data Types: `logical`

Algorithms

When `bwunpack` unpacks `BWP`, the function maps the least significant bit of the first row of `BWP` to the first pixel in the first row of `BW`. The most significant bit of the first element of `BWP` maps to the first pixel in the 32nd row of `BW`, and so on.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `bwunpack` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `bwunpack` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, all input arguments must be compile-time constants.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, all input arguments must be compile-time constants.

See Also

`bwpack` | `imdilate` | `imerode`

Introduced before R2006a

camresponse

Estimate camera response function

Syntax

```
crf = camresponse(files)
crf = camresponse(imds)
crf = camresponse( ___ ,'ExposureTimes',expTimes)
crf = camresponse(images,'ExposureTimes',expTimes)
```

Description

`crf = camresponse(files)` estimates the camera response function from a set of spatially registered, low dynamic range (LDR) images listed in `files`.

`crf = camresponse(imds)` estimates the camera response function from a set of spatially registered LDR images stored as an `ImageDatastore` object, `imds`.

`crf = camresponse( ___ ,'ExposureTimes',expTimes)` specifies the exposure time for each image in the input set using a name-value pair. You can specify this name-value pair in addition to the input argument from any of the previous syntaxes.

`crf = camresponse(images,'ExposureTimes',expTimes)` estimates the camera response function from a set of spatially registered LDR images stored as a cell array. Specify the exposure time for each image in the input set by using a name-value pair.

Examples

Estimate Camera Response Function from Set of Images

Specify a set of six low dynamic range (LDR) images that are spatially registered. These LDR images have same f-stop values and varying exposure times.

```
files = ["office_1.jpg","office_2.jpg","office_3.jpg",...
         "office_4.jpg","office_5.jpg","office_6.jpg"];
```

Estimate the camera response function from the set of specified images.

```
crf = camresponse(files);
```

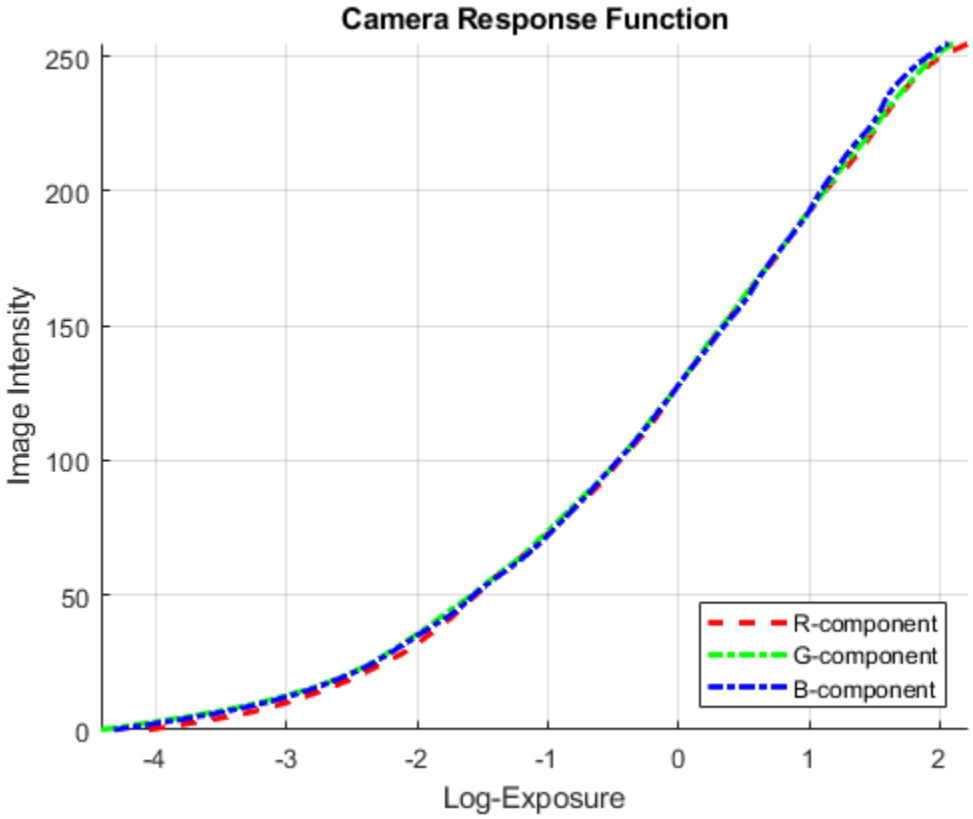
Specify the range of intensity levels in the input images.

```
range = 0:length(crf)-1;
```

Plot the estimated camera response function for each of the red, green, and blue color components. The plot shows the relationship between log-exposure and image intensity.

```
figure
hold on
plot(crf(:,1),range,'--r','LineWidth',2);
```

```
plot(crf(:,2),range,'-.g','LineWidth',2);
plot(crf(:,3),range,'-.b','LineWidth',2);
xlabel('Log-Exposure');
ylabel('Image Intensity');
title('Camera Response Function');
grid on
axis('tight')
legend('R-component','G-component','B-component','Location','southeast')
```



Estimate Camera Response Function from Images in Datastore

Create an `ImageDatastore` object containing six low dynamic range (LDR) images.

```
setDir = fullfile(toolboxdir('images'),'imdata','office_*');
imds = imageDatastore(setDir);
```

Display the images in a montage.

```
montage(imds,'Size',[6 1])
```

Specify the exposure time for each image in the `ImageDatastore` object.

```
expTimes = [0.0333 0.1000 0.3333 0.6250 1.3000 4.0000];
```

Estimate the camera response function from the images in the datastore, specifying the exposure times.

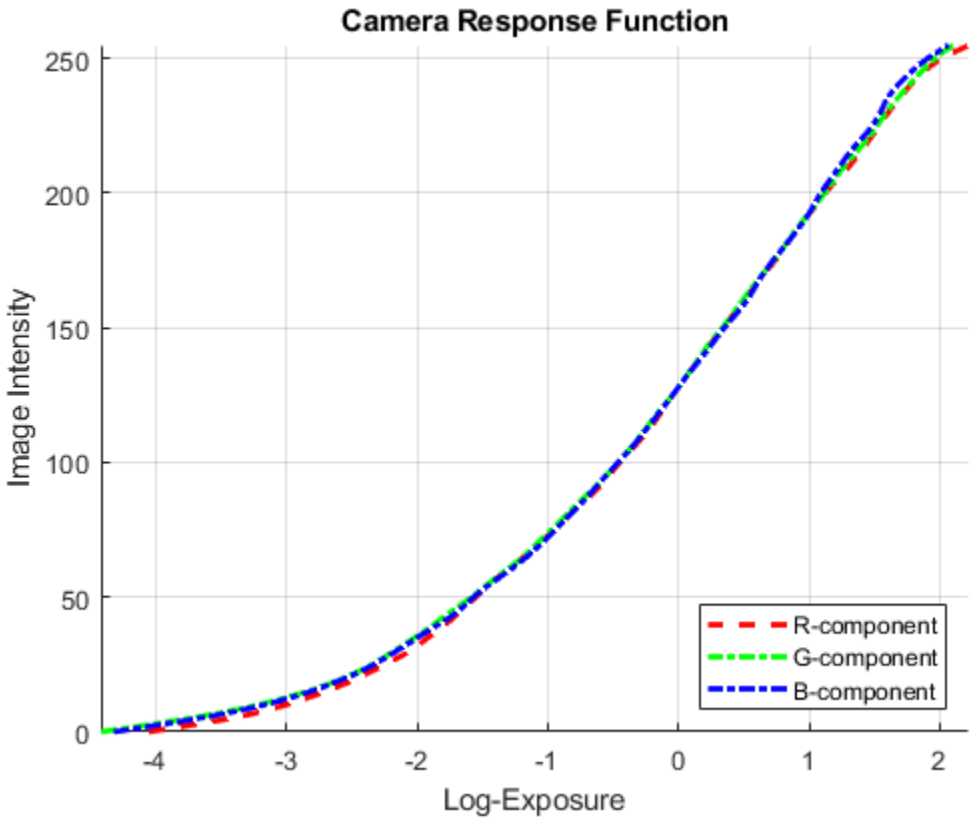
```
crf = camresponse(imds,'ExposureTimes',expTimes);
```

Specify the range of intensity values in the input images.

```
range = 0:length(crf)-1;
```

Plot the estimated camera response function for each of the R, G, and B color components. The plot shows the relationship between log-exposure and image intensity.

```
figure
hold on
plot(crf(:,1),range,'--r','LineWidth',2);
plot(crf(:,2),range,'-.g','LineWidth',2);
plot(crf(:,3),range,'-.b','LineWidth',2);
xlabel('Log-Exposure');
ylabel('Image Intensity');
title('Camera Response Function');
grid on
axis('tight')
legend('R-component','G-component','B-component','Location','southeast')
```



Input Arguments

`files` — Set of spatially registered LDR images

string array | cell array of character vectors

Set of spatially registered LDR images, specified as a string array or a cell array of character vectors. These images can be color or grayscale of any bit depth. However, the preferred bit depth for LDR images is 8 or 16.

Data Types: `char` | `string` | `cell`

`imds` — Set of spatially registered LDR images

`ImageDatastore` object

Set of spatially registered LDR images, specified as an `ImageDatastore` object. These images can be color or grayscale of any bit depth. However, the preferred bit depth for LDR images is 8 or 16.

`images` — Set of spatially registered LDR images

cell array

Set of spatially registered LDR images, specified as a cell array. These images can be color or grayscale of any bit depth. However, the preferred bit depth for LDR images is 8 or 16.

`expTimes` — Exposure time of input images

numeric vector of positive values

Exposure time of input images, specified as a numeric vector of positive values. The *k*th element in the vector corresponds to the *k*th LDR image in the input set. If you specify `expTimes`, the function overrides the EXIF exposure metadata.

Example: `camresponse(files,'ExposureTimes',[0.1 0.3 0.4]);`

Data Types: `single` | `double`

Note When input is a cell array of LDR images, you must specify exposure time as the second input argument by using the name-value pair `'ExposureTimes'`.

Output Arguments

`crf` — Estimate of camera response function

n-by-1 vector | *n*-by-3 matrix

Estimate of camera response function, returned as an *n*-by-1 vector for grayscale images and *n*-by-3 matrix for color images. The camera response function maps the log-exposure value (scene radiance) to the intensity levels in the input images. The value of *n* is $2^{bit\ depth}$. For example, if the bit depth of the input set of images is 8, then *n* is 256.

Data Types: `double`

Note

- This function requires a minimum of two images with different exposure times. A larger number of images yields a better estimate of `crf` at the expense of more processing time.

- The input image files in `files` and `imds` must contain the Exchangeable Image File Format (EXIF) exposure metadata. To estimate the `crf` values, the function reads the exposure time in the EXIF metadata. If you specify `expTimes`, the function overrides the exposure time in the EXIF metadata.

References

[1] Debevec, P.E., and J. Malik. "Recovering High Dynamic Range Radiance Maps from Photographs." In *ACM SIGGRAPH 2008 classes*, Article No. 31. New York, NY: ACM, 2008.

See Also

`makehdr` | `hdrread` | `hdrwrite`

Introduced in R2019a

centerCropWindow2d

Create rectangular center cropping window

Syntax

```
win = centerCropWindow2d(inputSize,targetSize)
```

Description

`win = centerCropWindow2d(inputSize,targetSize)` determines the window to crop from a 2-D input image of size `inputSize` such that the size of the cropped image is `targetSize`. The coordinates of the window are centered in the input image.

Examples

Center Crop Image To Target Size

Read and display an image.

```
chips = imread('coloredChips.png');
imshow(chips)
```

Specify the target size of the cropping window.

```
targetSize = [256 256];
```

Create a center crop window.

```
win1 = centerCropWindow2d(size(chips),targetSize);
```

Crop the original image using the center crop window.

```
B1 = imcrop(chips,win1);
```

Display the cropped image.

```
imshow(B1)
```

Read and display a second image of a different size.

```
kobi = imread('kobi.png');
imshow(kobi)
```

Try applying the center crop window to this image. The cropped region does not come from the center of the image because the center crop window uses the spatial extents of the `chips` image.

```
B2 = imcrop(kobi,win1);
imshow(B2)
```

To crop the `kobi` image from the center, specify a new center crop window.

```
win2 = centerCropWindow2d(size(kobi),targetSize);
B3 = imcrop(kobi,win2);
imshow(B3)
```

Input Arguments

`inputSize` — Input image size

2-element vector of positive integers | 3-element vector of positive integers

Input image size, specified as one of the following.

Type of Input Image	Format of `inputSize`
2-D grayscale or binary image	2-element vector of positive integers of the form `[height width]`
2-D RGB or multispectral image of size	3-element vector of positive integers of the form `[height width channels]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`targetSize` — Target image size

2-element vector of positive integers | 3-element vector of positive integers

Target image size, specified as one of the following.

Type of Target Image	Format of `targetSize`
2-D grayscale or binary image	2-element vector of positive integers of the form `[height width]`
2-D RGB or multispectral image of size	3-element vector of positive integers of the form `[height width channels]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`win` — Cropping window

`Rectangle` object

Cropping window, returned as a `Rectangle` object.

See Also

`centerCropWindow3d` | `randomWindow2d` | `imcrop`

Topics

“Augment Images for Deep Learning Workflows Using Image Processing Toolbox” (Deep Learning Toolbox)

Introduced in R2019b

centerCropWindow3d

Create cuboidal center cropping window

Syntax

```
win = centerCropWindow3d(inputSize,targetSize)
```

Description

`win = centerCropWindow3d(inputSize,targetSize)` determines the window to crop from a 3-D input image of size `inputSize` such that the size of the cropped image is `targetSize`. The coordinates of the window are centered in the input image.

Examples

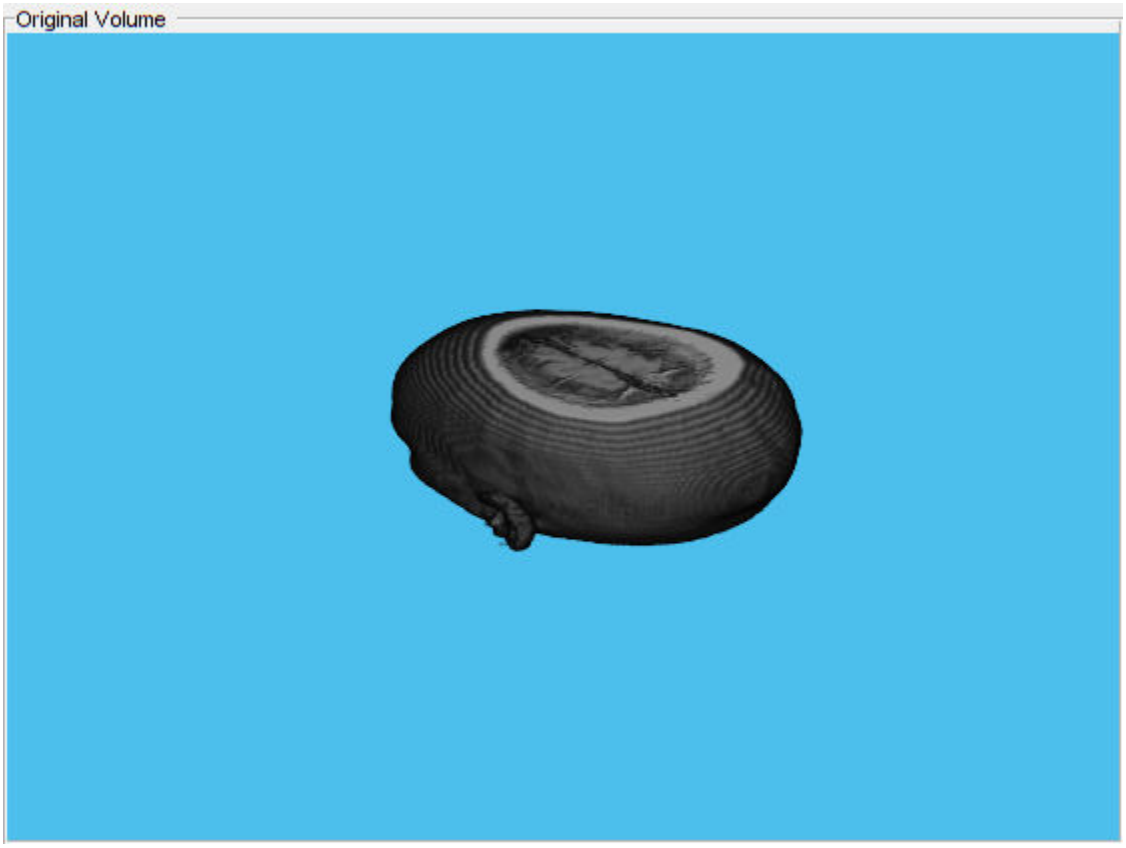
Center Crop 3-D Image to Target Size

Load a 3-D MRI image. Use the `squeeze` function to remove any singleton dimensions.

```
load mri;
D = squeeze(D);
```

Display the image.

```
fullViewPnl = uipanel(figure,'Title','Original Volume');
volshow(D,'Parent',fullViewPnl);
```



Specify the target size of the cropping window.

```
targetSize = [64 64 10];
```

Create a center cropping window that crops the specified image from its center.

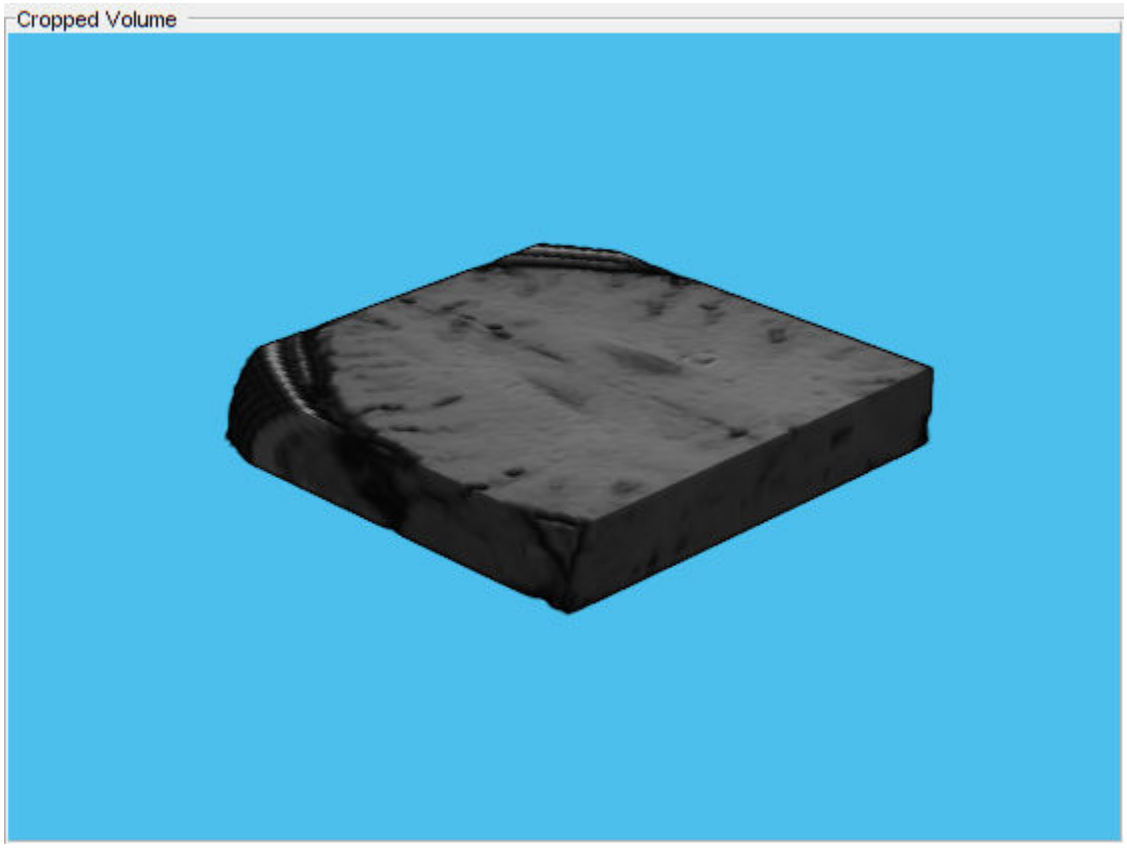
```
win = centerCropWindow3d(size(D),targetSize);
```

Crop the image using the center cropping window.

```
Dcrop = imcrop3(D,win);
```

Display the cropped image in a display panel.

```
fullViewPnl = uipanel(figure,'Title','Cropped Volume');
volshow(Dcrop,'Parent',fullViewPnl);
```



Input Arguments

`inputSize` — Input image size

3-element vector of positive integers | 4-element vector of positive integers

Input image size, specified as one of the following.

Type of Input Image	Format of `inputSize`
3-D grayscale or binary image	3-element vector of positive integers of the form `[height width depth]`
3-D RGB or multispectral image	4-element vector of positive integers of the form `[height width depth channels]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`targetSize` — Target image size

3-element vector of positive integers | 4-element vector of positive integers

Target image size, specified as one of the following.

Type of Target Image	Format of `targetSize`
3-D grayscale or binary image	3-element vector of positive integers of the form `[height width depth]`
3-D RGB or multispectral image	4-element vector of positive integers of the form `[height width depth channels]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`win` — Cropping window

`Cuboid` object

Cropping window, returned as a `Cuboid` object.

See Also

`centerCropWindow2d` | `randomCropWindow3d` | `imcrop3`

Topics

“Augment Images for Deep Learning Workflows Using Image Processing Toolbox” (Deep Learning Toolbox)

Introduced in R2019b

checkerboard

Create checkerboard image

Syntax

```
I = checkerboard
I = checkerboard(n)
I = checkerboard(n,p,q)
```

Description

`I = checkerboard` creates an 8-by-8 square checkerboard image that has four identifiable corners. The checkerboard pattern is made up of tiles. Each tile contains four squares, each with a default of 10 pixels per side. The light squares on the left half of the checkerboard are white. The light squares on the right half of the checkerboard are gray.

```
TILE = [DARK LIGHT; LIGHT DARK]
```

`I = checkerboard(n)` creates an 8-by-8 square checkerboard image where each square has `n` pixels per side.

`I = checkerboard(n,p,q)` creates a rectangular checkerboard image where `p` specifies the number of rows of tiles and `q` specifies the number of columns of tiles. If you omit `q`, the number of columns defaults to `p` and the checkerboard is square. Each square has `n` pixels per side.

Examples

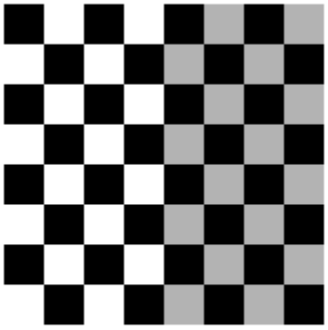
Create Square Checkerboard

Create a checkerboard where the side of every square is 20 pixels in length.

```
I = checkerboard(20);
```

Display the checkerboard.

```
imshow(I)
```



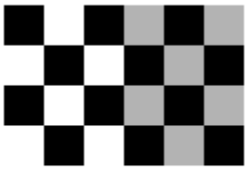
Create Rectangular Checkerboard

Create a rectangular checkerboard that is 2 tiles high and 3 tiles wide. The side of every square is 20 pixels in length.

```
J = checkerboard(20,2,3);
```

Display the checkerboard.

```
figure
imshow(J)
```



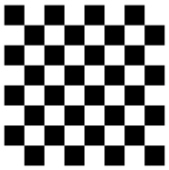
Create Black and White Checkerboard

Create a black and white checkerboard with the default tile size and the default number of rows and columns.

```
K = (checkerboard > 0.5);
```

Display the checkerboard.

```
figure
imshow(K)
```



Input Arguments

n — Side length in pixels of each square

`10` (default) | positive integer

Side length in pixels of each square in the checkerboard pattern, specified as a positive integer.

p — Number of rows of tiles

`4` (default) | positive integer

Number of rows of tiles in the checkerboard pattern, specified as a positive integer. Since there are four squares per tile, there are `2*p` rows of squares in the checkerboard.

q — Number of columns of tiles

positive integer

Number of columns of tiles in the checkerboard pattern, specified as a positive integer. If you omit `q`, the value defaults to `p` and the checkerboard is square. Since there are four squares per tile, there are `2*q` columns of squares in the checkerboard.

Output Arguments

I — Rectangular image with checkerboard pattern

2-D numeric array

Rectangular image with a checkerboard pattern, returned as a 2-D numeric array. The light squares on the left half of the checkerboard are white. The light squares on the right half of the checkerboard are gray.

Data Types: `double`

See Also

`fitgeotrans` | `imwarp`

Introduced before R2006a

chromadapt

Adjust color balance of RGB image with chromatic adaptation

Syntax

```
B = chromadapt(A,illuminant)
B = chromadapt(A,illuminant,Name,Value)
```

Description

`B = chromadapt(A,illuminant)` adjusts the color balance of sRGB image `A` according to the scene illuminant. The illuminant must be in the same color space as the input image.

`B = chromadapt(A,illuminant,Name,Value)` adjusts the color balance of `A` using name-value pairs to control additional options.

Examples

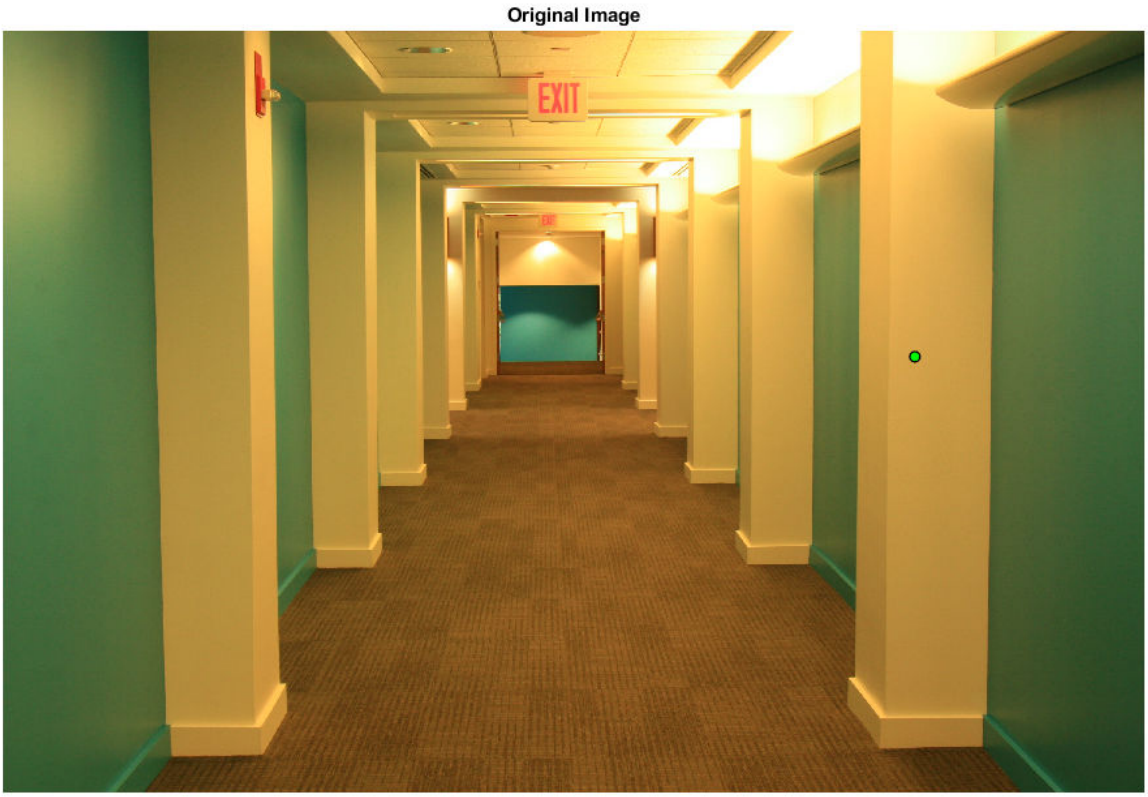
Color Balance Image by Specifying Gray Pixel

Read and display an image with a strong yellow color cast.

```
A = imread('hallway.jpg');
imshow(A)
title('Original Image')
```

Pick a pixel in the image that should look white or gray, such as a point on a pillar. Do not pick a saturated pixel, such as a point on the ceiling light. Display the selected point in green.

```
x = 2800;
y = 1000;
gray_val = impixel(A,x,y);
drawpoint('Position',[x y],'Color','g');
```

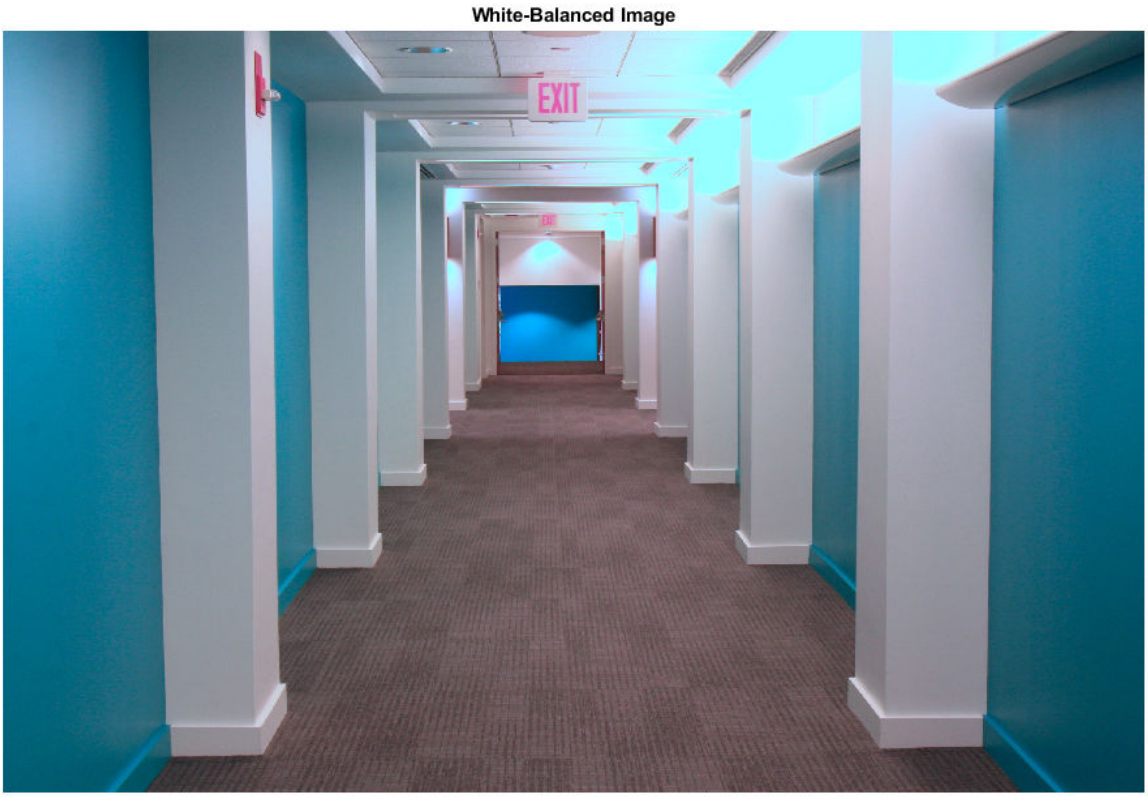


Use the selected color as reference for the scene illumination, and correct the white balance of the image.

```
B = chromadapt(A,gray_val);
```

Display the corrected image. The pillars now appear white as expected, and the rest of the image has no yellow tint.

```
imshow(B)
title('White-Balanced Image')
```



Color Balance Image in Linear RGB Color Space

Open an image file containing minimally processed linear RGB intensities.

```
A = imread("foosballraw.tiff");
```

The image data is the raw sensor data after correcting the black level and scaling to 16 bits per pixel. Interpolate the intensities to reconstruct color. The color filter array pattern is RGGB.

```
A = demosaic(A,"rggb");
```

Display the image. Because the image is in linear RGB color space, apply gamma correction so the image appears correctly on the screen.

```
A_sRGB = lin2rgb(A);
imshow(A_sRGB)
title("Original Image")
```

The image has a ColorChecker® chart in the scene. To get the color of the ambient light, display the RGB values of a pixel in one of the neutral patches of the chart. The intensity of the red channel is lower than the intensity of the other two channels, which indicates that the light is bluish green.

```
x = 1510;
y = 1250;
light_color = [A(y,x,1) A(y,x,2) A(y,x,3)]
```

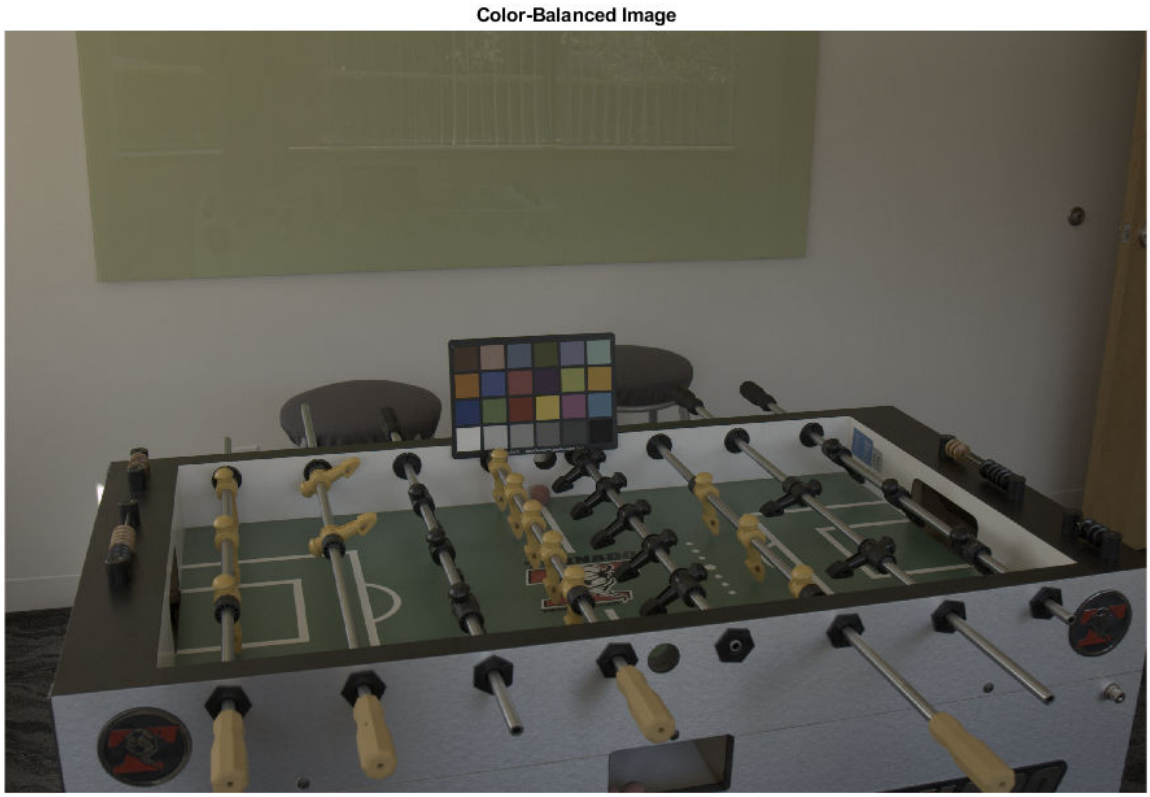
```
light_color = 1x3 uint16 row vector

    7361   14968   10258
```

Balance the color channels of the image. Use the `ColorSpace` name-value argument to specify that the image and the illuminant are expressed in linear RGB.

```
B = chromadapt(A,light_color,"ColorSpace","linear-rgb");
```

Display the color-balanced image with gamma correction.

```
B_sRGB = lin2rgb(B);
imshow(B_sRGB)
title("Color-Balanced Image")
```



Confirm that the gray patch has been color balanced. The three color channels in the color-balanced gray patch have similar intensities, as expected.

```
patch_color = [B(y,x,1) B(y,x,2) B(y,x,3)]
```

```
patch_color = 1x3 uint16 row vector

   13010   13010   13010
```

Input Arguments

A — RGB image

m-by-*n*-by-3 numeric array

RGB image, specified as an *m*-by-*n*-by-3 numeric array.

Data Types: `single` | `double` | `uint8` | `uint16`

illuminant — Scene illuminant

3-element numeric vector

Scene illuminant, specified as a 3-element numeric vector. The illuminant must be in the same color space as the input image, `A`.

Data Types: `single` | `double` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `I2 = chromadapt(I,uint8([22 97 118]),'ColorSpace','linear-rgb')` adjusts the color balance of an image, `I`, in linear RGB color space.

`ColorSpace` — Color space

`'srgb'` (default) | `'adobe-rgb-1998'` | `'linear-rgb'`

Color space of the input image and illuminant, specified as the comma-separated pair consisting of `'ColorSpace'` and `'srgb'`, `'adobe-rgb-1998'`, or `'linear-rgb'`. Use the `'linear-rgb'` option to adjust the color balance of an RGB image whose intensities are linear.

Data Types: `char` | `string`

`Method` — Chromatic adaptation method

`'bradford'` (default) | `'vonkries'` | `'simple'`

Chromatic adaptation method used to scale the RGB values in `A`, specified as the comma-separated pair consisting of `'Method'` and one of:

- `'bradford'`—Scale using the Bradford cone response model
- `'vonkries'`—Scale using the von Kries cone response model
- `'simple'`—Scale using the illuminant

Data Types: `char` | `string`

Output Arguments

`B` — Color-balanced RGB image

m-by-*n*-by-3 numeric array

Color-balanced RGB image, returned as an *m*-by-*n*-by-3 numeric array of the same data type as `A`.

References

[1] Lindbloom, Bruce. Chromatic Adaptation. http://www.brucelindbloom.com/index.html?Eqn_ChromAdapt.html.

See Also

`whitepoint` | `colorangle` | `illumgray` | `illumpca` | `illumwhite`

Introduced in R2017b

col2im

Rearrange matrix columns into blocks

Syntax

```
A = col2im(B,[m n],[M N])
A = col2im(B,[m n],[M N],'sliding')
A = col2im(B,[m n],[M N],'distinct')
```

Description

`A = col2im(B,[m n],[M N])` or

`A = col2im(B,[m n],[M N],'sliding')` rearranges the row vector `B` into neighborhoods of size `m`-by-`n` to create the matrix `A` of size `(M-m+1)`-by-`(N-n+1)`.

The row vector `B` is usually the result of processing the output of `im2col(...,'sliding')` using a column compression function, such as `sum`.

`A = col2im(B,[m n],[M N],'distinct')` rearranges each column of matrix `B` into a distinct `m`-by-`n` block to create the matrix `A` of size `M`-by-`N`.

For example, if `B` consists of column vectors `Bi(:)` with length `m*n`, arranged as `B = [B1(:) B2(:) B3(:) B4(:)]`, then `A = [B1 B3; B2 B4]` where each block `Bi` has size `m`-by-`n`.

Examples

Rearrange Matrix Values into Row-wise Orientation

Create a matrix.

```
B = reshape(uint8(1:25),[5 5])'
```

```
B = 5x5 uint8 matrix

    1    2    3    4    5
    6    7    8    9   10
   11   12   13   14   15
   16   17   18   19   20
   21   22   23   24   25
```

Rearrange the values in the matrix into a column-wise arrangement.

```
C = im2col(B,[1 5])
```

```
C = 5x5 uint8 matrix

    1    6   11   16   21
    2    7   12   17   22
    3    8   13   18   23
```

```
    4    9   14   19   24
    5   10   15   20   25
```

Rearrange the values in the matrix back into their original row-wise orientation.

```
A = col2im(C,[1 5],[5 5],'distinct')
```

```
A = 5x5 uint8 matrix

    1    2    3    4    5
    6    7    8    9   10
   11   12   13   14   15
   16   17   18   19   20
   21   22   23   24   25
```

Input Arguments

B — Image blocks

matrix | row vector

Image blocks, specified as one of the following.

- For distinct block processing, `B` is a numeric or logical matrix with `m*n` rows. Each column corresponds to one block.
- For sliding neighborhood processing, `B` is a numeric or logical row vector of size 1-by-`(M-m+1)*(N-n+1)`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

[m n] — Block size

2-element vector of positive integers

Block size, specified as a 2-element vector of positive integers. `m` is the number of rows and `n` is the number of columns in each block. `m*n` must be equal to the number of rows of `B`.

Data Types: `double`

[M N] — Image size

2-element vector of positive integers

Image size, specified as a 2-element vector of positive integers. `M` is the number of rows and `N` is the number of columns in the image.

Data Types: `double`

Output Arguments

A — Reconstructed image

numeric matrix

Reconstructed image, returned as a numeric matrix of size `M`-by-`N` for distinct block processing, or `(M-m+1)`-by-`(N-n+1)` for sliding block processing. `A` has the same data type as `B`.

See Also

`blockproc` | `colfilt` | `im2col` | `nlfilter` | `reshape`

Introduced before R2006a

colfilt

Column-wise neighborhood operations

Syntax

```
B = colfilt(A,[m n],block_type,fun)
B = colfilt(A,[m n],[mblock nblock],block_type,fun)
B = colfilt(A,'indexed',___)
```

Description

`B = colfilt(A,[m n],block_type,fun)` processes the image `A` by rearranging each `m`-by-`n` block of `A` into a column of a temporary matrix, and then applying the function `fun` to this matrix. `colfilt` zero-pads `A`, if necessary.

`B = colfilt(A,[m n],[mblock nblock],block_type,fun)` subdivides `A` into regions of size `mblock`-by-`nblock` blocks to save memory. Note that the result of the operation does not change when using the `[mblock nblock]` argument.

For example, if `[mblock nblock]` is `[3 4]` and the size of each block is 16-by-16 pixels, then `colfilt` subdivides the image into regions of size 48-by-64 pixels and processes each region separately.

`B = colfilt(A,'indexed',___)` processes `A` as an indexed image, padding with `0`s if the class of `A` is `uint8`, `uint16`, or `logical`, and padding with `1`s otherwise.

Examples

Perform Column-Wise Neighborhood Filtering on Image

This example shows how to set each output pixel to the mean value of the input pixel's 5-by-5 neighborhood using column-wise neighborhood processing.

Read a grayscale image into the workspace.

```
I = imread('tire.tif');
```

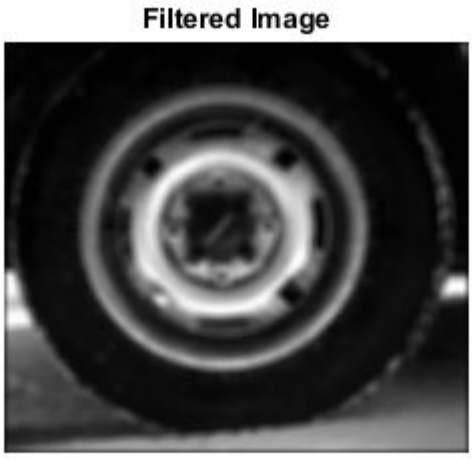
Perform column-wise filtering. The function `mean` is called on each 5-by-5 pixel neighborhood.

```
I2 = uint8(colfilt(I,[5 5],'sliding',@mean));
```

Display the original image and the filtered image.

```
imshow(I)
title('Original Image')
```


```
figure
imshow(I2)
title('Filtered Image')
```



Input Arguments

`A` — Image

array

Image, specified as an array of any class supported by `fun`.

`[m n]` — Block size

2-element vector of positive integers

Block size, specified as a 2-element vector of positive integers. `m` is the number of rows and `n` is the number of columns in each block.

`[mblock nblock]` — Block group size
2-element vector of positive integers

Block group size, specified as a 2-element vector of positive integers. `mblock` is the number of blocks in the group in the vertical direction, and `nblock` is the number of blocks in the group in the horizontal direction.

`block_type` — Block type
`'sliding'` | `'distinct'`

Block type, specified as `'sliding'` for sliding neighborhoods or `'distinct'` for distinct blocks.

Data Types: `char` | `string`

`fun` — Function handle
handle

Function handle, specified as a handle. The input and output arguments to this function depend on the value of `block_type`. For more information, see “Algorithms” on page 1-546.

For more information about function handles, see “Create Function Handle”.

Output Arguments

`B` — Filtered image
numeric matrix

Filtered image, returned as a numeric matrix.

Algorithms

The algorithm that `colfilt` uses to process images depends on the value of `block_type`.

Value	Description
`'distinct'`	• First, `colfilt` rearranges each `m`-by-`n` block of `A` into a column in a temporary matrix by using the `im2col` function. • Next, `colfilt` applies the function `fun` to this temporary matrix. `fun` must return a matrix the same size as the temporary matrix. • Finally, `colfilt` rearranges the columns of the matrix returned by `fun` into `m`-by-`n` distinct blocks, by using the `col2im` function.

Value	Description
`'sliding'`	• First, `colfilt` rearranges each `m`-by-`n` neighborhood of `A` into a column in a temporary matrix by using the `im2col` function. • Next, `colfilt` applies the function `fun` to this temporary matrix. `fun` must return a row vector containing a single value for each column in the temporary matrix. (Column compression functions such as `sum` return the appropriate type of output.) • Finally, `colfilt` reshapes the vector returned by `fun` into a matrix the same size as `A`, by using the `reshape` function.

To save memory, the `colfilt` function might divide `A` into subimages and process one subimage at a time. This implies that `fun` may be called multiple times, and that the first argument to `fun` may have a different number of columns each time.

See Also

`blockproc` | `col2im` | `im2col` | `nlfilter` | `reshape`

Topics

“Use Column-wise Processing to Speed Up Sliding Neighborhood or Distinct Block Operations”
“Border Padding Behavior in Sliding Neighborhood Operations”
“Anonymous Functions”
“Parameterizing Functions”
“Create Function Handle”

Introduced before R2006a

colorangle

Angle between two RGB vectors

Syntax

```
angle = colorangle(rgb1,rgb2)
```

Description

`angle = colorangle(rgb1,rgb2)` computes the angle in degrees between two RGB vectors.

Examples

Compare Accuracy of Illuminant Estimation Algorithms

Read a test image. The image is the raw data captured with a Canon EOS 30D digital camera after correcting the black level and scaling the intensities to 16 bits per pixel. No demosaicing, white balancing, color enhancement, noise filtering, or gamma correction has been applied.

```
RAW = imread("foosballraw.tiff");
```

Interpolate using the `demosaic` function to obtain a color image. The color filter array pattern is RGGB.

```
A = demosaic(RAW,"rggb");
```

Display the image. Because the image is in linear RGB color space, apply gamma correction so the image appears correctly on the screen.

```
A_sRGB = lin2rgb(A);
imshow(A_sRGB)
```

The image contains a ColorChecker® chart. Specify the ground truth illuminant, which was calculated in advance using the neutral patches of the chart.

```
illuminant_groundtruth = [0.0717 0.1472 0.0975];
```

To avoid skewing the estimation of the illuminant, exclude the ColorChecker chart by creating a mask.

```
mask = true(size(A,1), size(A,2));
mask(920:1330,1360:1900) = false;
```

Run three different illuminant estimation algorithms: `illumwhite`, `illumgray`, and `illumpca`.

```
illuminant_whitepatch = illumwhite(A,"Mask",mask);
illuminant_grayworld = illumgray(A,"Mask",mask);
illuminant_pca = illumpca(A,"Mask",mask);
```

Compare each estimation against the ground truth by calculating the angle between each estimated illuminant and the ground truth using the `colorangle` function. The smaller the angle, the better the estimation. The magnitude of the estimation does not matter because only the direction of the illuminant is used to white-balance an image with chromatic adaptation.

```
angle_whitepatch = colorangle(illuminant_whitepatch,illuminant_groundtruth)
```

```
angle_whitepatch = 5.0921
```

```
angle_grayworld = colorangle(illuminant_grayworld,illuminant_groundtruth)
```

```
angle_grayworld = 5.1036

angle_pca = colorangle(illuminant_pca,illuminant_groundtruth)

angle_pca = 5.0134
```

The value of `angle_pca` is smallest, indicating that the PCA illuminant estimation algorithm is closest to the ground truth illumination for this image.

Input Arguments

`rgb1` — First RGB vector

3-element numeric vector

First RGB vector, specified as a 3-element numeric vector.

Data Types: `single` | `double` | `uint8` | `uint16`

`rgb2` — Second RGB vector

3-element numeric vector

Second RGB vector, specified as a 3-element numeric vector.

Data Types: `single` | `double` | `uint8` | `uint16`

Output Arguments

`angle` — Angle between RGB vectors

numeric scalar

Angle between RGB vectors, returned as a numeric scalar.

Data Types: `double`

More About

Angular Error

Angular error is a useful metric to evaluate the estimation of an illuminant against the ground truth. The smaller the angle between the ground truth illuminant and the estimated illuminant, the better the estimate.

See Also

`whitepoint` | `chromadapt` | `illumgray` | `illumpca` | `illumwhite`

Introduced in R2017b

colorChecker

Calibrite ColorChecker test chart

Description

A `colorChecker` object stores the positions and measurements of the regions of interest (ROIs) of a Calibrite ColorChecker® Classic test chart (formerly produced by X-Rite® and GretagMacbeth®) [1].

Creation

Syntax

```
chart = colorChecker(A)
chart = colorChecker(A,Name,Value)
chart = colorChecker(A,"RegistrationPoints",p)
```

Description

`chart = colorChecker(A)` creates a `colorChecker` object from input image `A`. The input image sets the Image on page 1-0 property.

`chart = colorChecker(A,Name,Value)` controls the automatic chart detection using one or more name-value arguments.

`chart = colorChecker(A,"RegistrationPoints",p)` creates a `colorChecker` object and sets the RegistrationPoints property using the specified points in `p`.

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `chart = colorChecker(A,Downsample=false)` does not downsample the chart image for chart detection

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `chart = colorChecker(A,"Downsample",false)` does not downsample the chart image for chart detection

`Downsample` — Downsample chart image
`true` or `1` (default) | `false` or `0`

Downsample the chart image for chart detection, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `true`, `colorChecker` resizes the image while preserving the aspect ratio such that the size of the smallest dimension is 1000. Downsampling enables `colorChecker` to detect the chart more quickly.

`colorChecker` uses the downsampled image for detection only. The object calculates all properties using the original image.

`Sensitivity` — Sensitivity
`0.6` (default) | number in the range [0, 1]

Sensitivity of chart detection, specified as a number in the range [0, 1]. If you set a high sensitivity value, `colorChecker` detects more points of interest with which to register the test chart image.

Data Types: `single` | `double`

Properties

`Image` — Test chart image
RGB image

This property is read-only.

Test chart image, specified as an RGB image.

Data Types: `single` | `double` | `uint8` | `uint16`

`ColorROIs` — Position and intensity values of color patches
24-by-1 vector of structures

This property is read-only.

Position and intensity values of the color patches, specified as a 24-by-1 vector of structures. Each element in the vector corresponds to one ROI and contains these fields:

Field	Description
`ROI`	1-by-4 vector specifying the spatial extent of the ROI. The vector has the form [*X Y Width Height*]. *X* and *Y* are the coordinates of the top-left corner of the ROI. *Width* and *Height* are the width and height of the ROI, in pixels. `ROI` is of data type `double`.
`ROIIntensity`	Array of color values within the ROI. The array has dimensions *Height*-by-*Width*-by-3. The data type of `ROIIntensity` matches the data type of the `Image on page 1-0` property.

`RegistrationPoints` — Coordinates of registration points
4-by-2 numeric matrix

Coordinates of registration points, specified as a 4-by-2 numeric matrix. The registration points are the (*x*,*y*) coordinates of the plus-shaped (+) fiducials on the outer corners of the chart. Each row of the matrix contains the coordinates for one registration point. Specify the points in the order "black", "white", "dark skin", and "bluish green" according to the color of the nearest color patch.

Data Types: `double`

Object Functions

measureColor	Measure color reproduction using test chart
measureIlluminant	Measure scene illuminant using test chart
displayChart	Display test chart with overlaid regions of interest

Examples

Create ColorChecker Chart from Test Image

Read an image of a ColorChecker® chart into the workspace.

```
I = imread("colorCheckerTestImage.jpg");
```

Display the image.

```
imshow(I)
title("Captured Image of ColorChecker Chart")
text(size(I,2),size(I,1)+15,"Chart courtesy of Calibrite", ...
    "FontSize",10,"HorizontalAlignment","right")
```

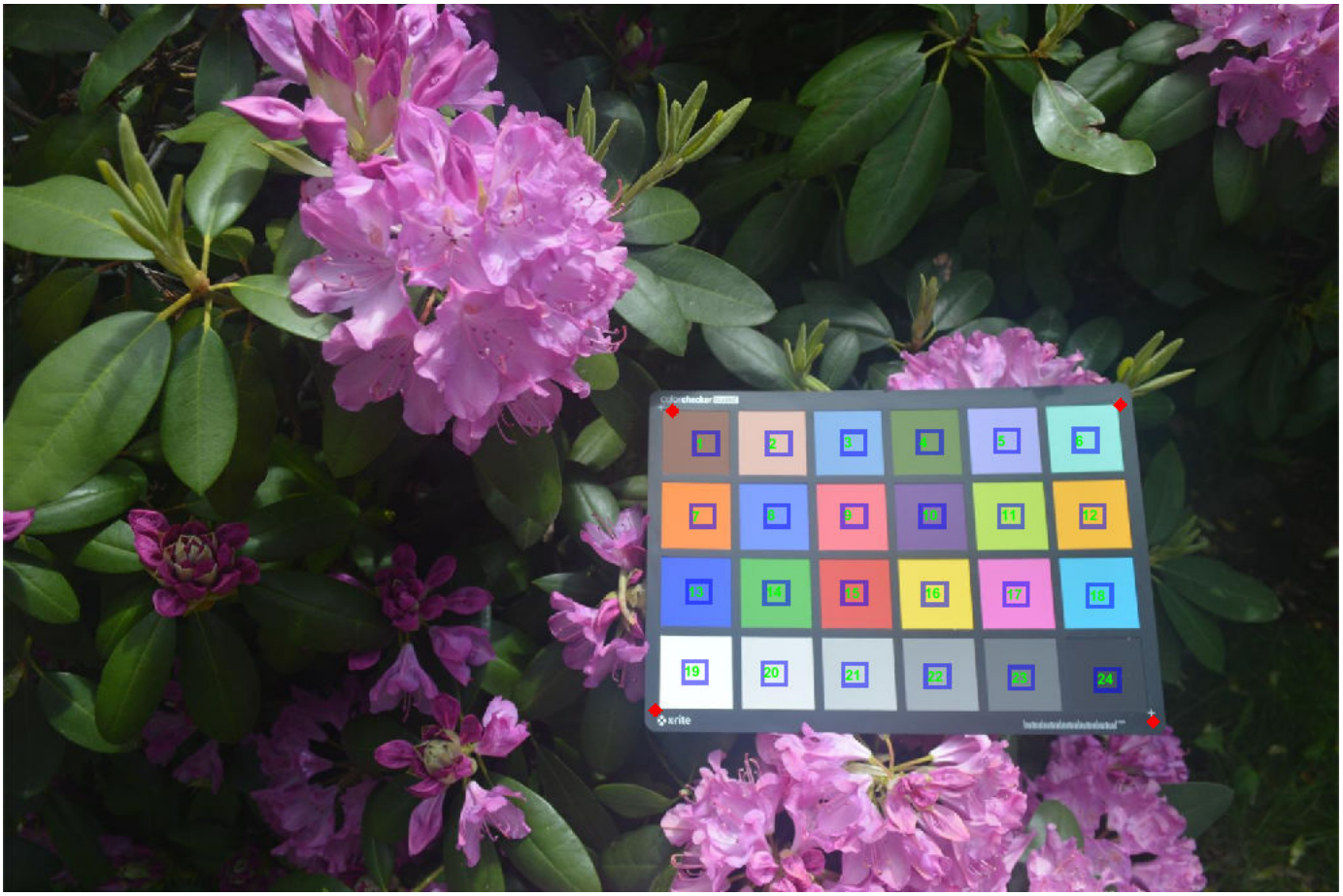
Create a `colorChecker` object by performing automatic chart detection on the image.

```
chart = colorChecker(I)
```

```
chart =
  colorChecker with properties:

                Image: [1024x1541x3 uint8]
    RegistrationPoints: [4x2 double]
             ColorROIs: [24x1 struct]
```

To confirm that the `colorChecker` object detected the chart correctly, display the chart and detected ROIs. Each ROI appears as a blue rectangle centered in the appropriate color patch. The registration points appear as red diamonds on the outer corners of the chart.

```
displayChart(chart)
```



Create ColorChecker Chart from Registration Points

Read an image of a ColorChecker® chart into the workspace.

```
I = imread("colorCheckerTestImage.jpg");
```

Display the image.

```
imshow(I)
title("Captured Image of ColorChecker Chart")
text(size(I,2),size(I,1)+15,"Chart courtesy of Calibrite", ...
    "FontSize",10,"HorizontalAlignment","right")
```

Draw point ROIs that overlap the plus-shaped (+) fiducials at the corners of the chart.

```
blackPoint = drawpoint;
whitePoint = drawpoint;
```

```
darkSkinPoint = drawpoint;
bluishGreenPoint = drawpoint;
```

Chart courtesy of Calibrite

Combine the (*x*,*y*) coordinates of the point ROIs into a 4-by-2 matrix.

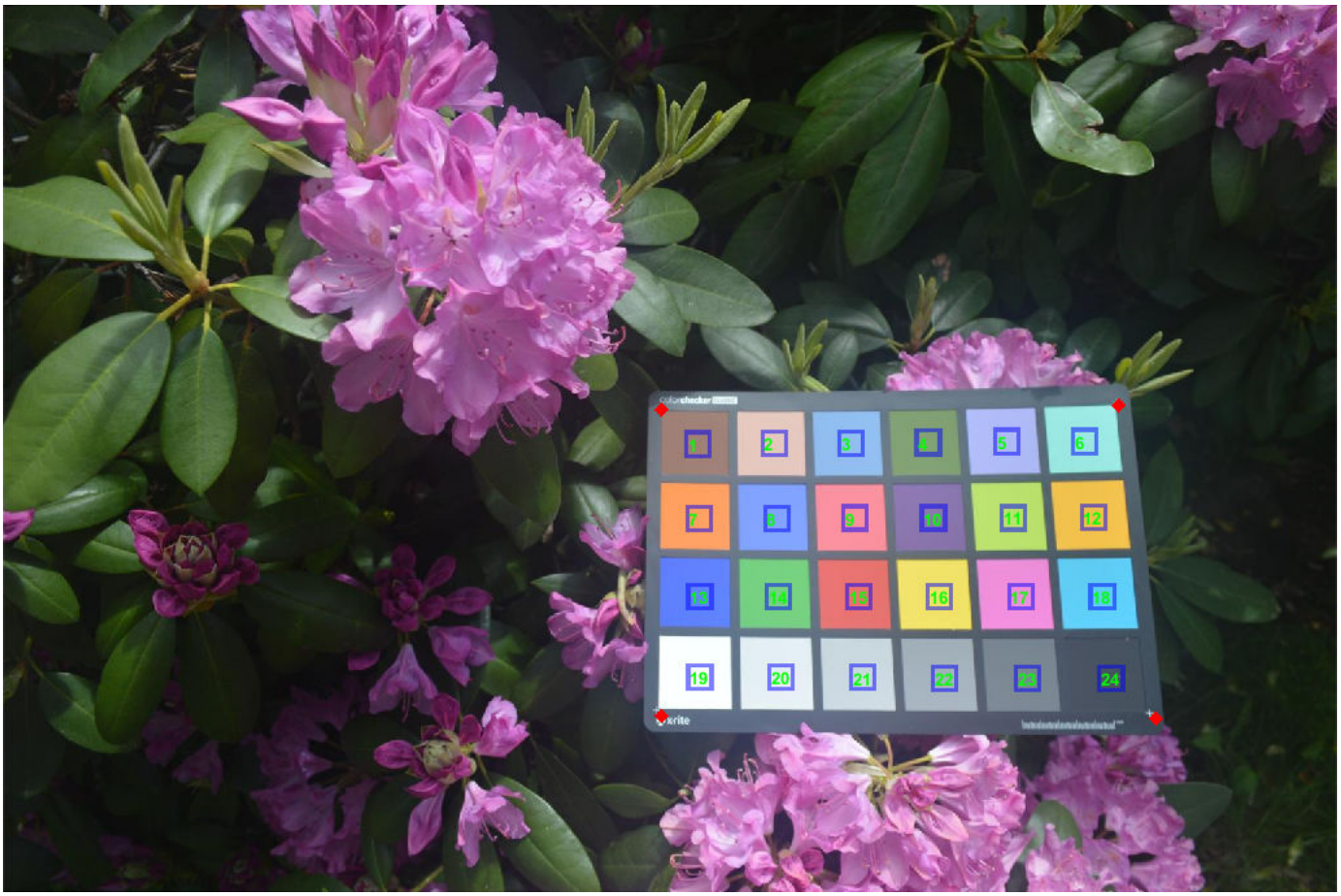
```
cornerPoints = [blackPoint.Position;
    whitePoint.Position;
    darkSkinPoint.Position;
    bluishGreenPoint.Position];
```

Create a `colorChecker` object by specifying the (*x*,*y*) coordinates of the corner registration points.

```
chart = colorChecker(I,"RegistrationPoints",cornerPoints);
```

To confirm that the `colorChecker` object detected the chart correctly, display the chart and detected ROIs.

```
displayChart(chart)
```



Tips

- There are two manufactured versions of the ColorChecker test chart that have slightly different reference values. The reference values of the `colorChecker` object match the "After November 2014" version of the chart.

References

[1] Calibrite. "ColorChecker Classic". https://calibrite.com/us/product/colorchecker-classic/.

[2] Fernandez, P. D. M., F. A. Guerrero-Peña, T. I. Ren, and G. J. J. Leandro, "Fast and robust multiple ColorChecker detection using deep convolutional neural networks," Image and Vision Computing, Volume 81, 2019, pp. 15-24.

See Also

`esfrChart`

Topics

“Calculate CIE94 Color Difference of Colors on Test Chart”
“Correct Colors Using Color Correction Matrix”

“Comparison of Auto White Balance Algorithms”

Introduced in R2020b

colorcloud

Display 3-D color gamut as point cloud in specified color space

Syntax

```
colorcloud(rgb)
colorcloud(rgb,colorspace)
colorcloud( ___ ,Name,Value)
hPanel = colorcloud( ___ )
```

Description

`colorcloud(rgb)` displays the full color gamut of the color image `rgb` as a point cloud. By default, `colorcloud` uses the RGB color space.

`colorcloud(rgb,colorspace)` displays the full color gamut of the color image `rgb` as a point cloud in the color space specified by `colorspace`.

`colorcloud( ___ ,Name,Value)` displays the full color gamut using name-value pairs to control aspects of the visualization.

`hPanel = colorcloud( ___ )` returns the uipanel object created by `colorcloud`.

Examples

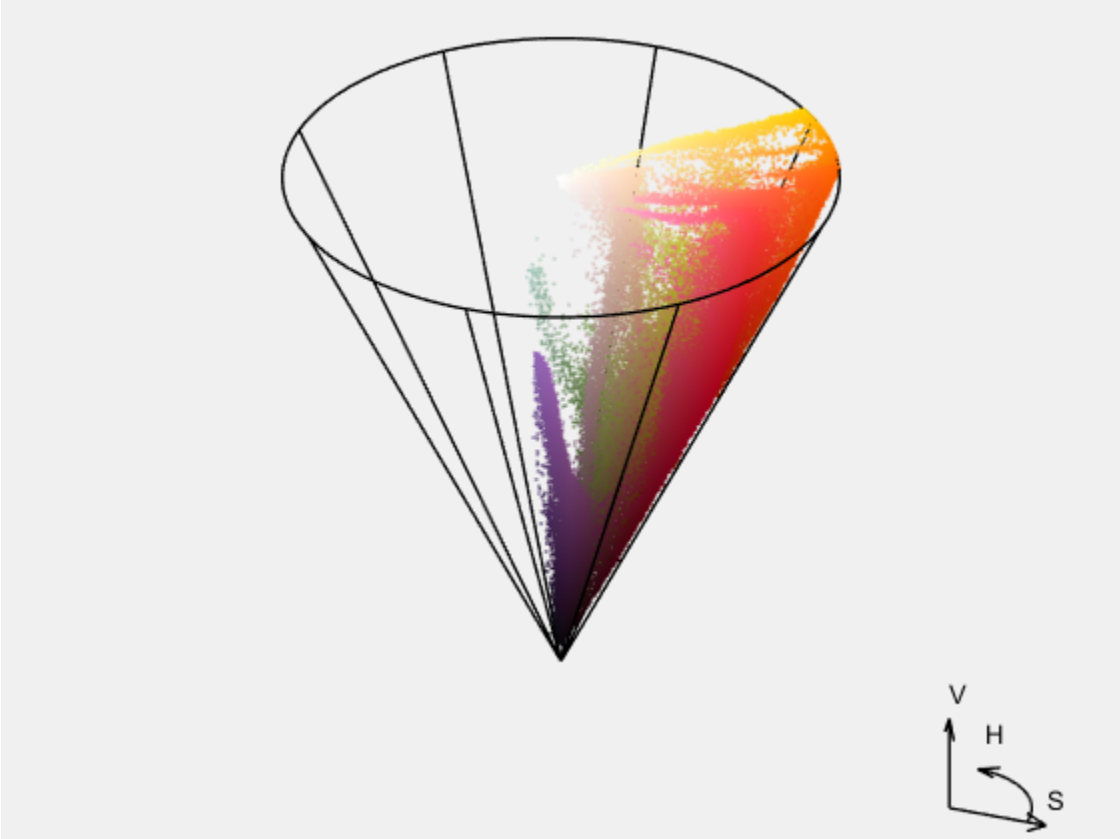
View 3D Color Gamut of RGB Image in HSV Color Space

Read in RGB image

```
RGB = imread('peppers.png');
```

View color gamut

```
colorcloud(RGB,'hsv');
```



Input Arguments

`rgb` — Color image

m-by-*n*-by-3 numeric array

Color image, specified as an *m*-by-*n*-by-3 numeric array.

Data Types: `single` | `double` | `uint8` | `uint16`

`colorspace` — Color space name

`'rgb'` (default) | `'hsv'` | `'ycbcr'` | `'lab'`

Color space name, specified as one of these values:

Value	Description
`'hsv'`	Color gamut in HSV color space
`'lab'`	Color gamut in CIE 1976 L*a*b* color space
`'rgb'`	Color gamut in RGB color space
`'ycbcr'`	Color gamut in YCbCr color space

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'BackgroundColor','w'`

`Parent` — Parent of object created by `colorcloud`

new figure (default)

Parent of the object created by `colorcloud`, specified as a figure or uipanel object. If you do not specify a valid object, then the `colorcloud` function creates a new figure window.

`BackgroundColor` — Background color

`[0.94 0.94 0.94]` (default) | RGB triplet | color name | short color name

Background color to the color cloud, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'BackgroundColor','r'`

Example: `'BackgroundColor','green'`

Example: `'BackgroundColor',[0 0.4470 0.7410]`

`WireFrameColor` — Color of wire frame

`'black'` (default) | `'none'` | RGB triplet | color name | short color name

Color of the wire frame, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify the value `'none'`, then `colorcloud` deletes the wire frame.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'WireFrameColor','r'`

Example: `'WireFrameColor','green'`

Example: `'WireFrameColor',[0.8500 0.3250 0.0980]`

`OrientationAxesColor` — Color of orientation axes and labels

`'black'` (default) | `'none'` | RGB triplet | color name | short color name

Color of the orientation axes and labels, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify the value `'none'`, then `colorcloud` deletes the labels.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'OrientationAxesColor','r'`

Example: `'OrientationAxesColor','green'`

Example: `'OrientationAxesColor',[0.9290 0.6940 0.1250]`

Output Arguments

`hPanel` — Color gamut point cloud

uipanel object

Color gamut point cloud, returned as a uipanel object.

See Also

Color Thresholder

Topics
“Understanding Color Spaces and Color Space Conversion”

Introduced in R2016b

conndef

Create connectivity array

Syntax

```
conn = conndef(num_dims,type)
```

Description

`conn = conndef(num_dims,type)` returns the pixel connectivity array defined by `type` for `num_dims` dimensions. Several Image Processing Toolbox functions use `conndef` to create the default connectivity input argument.

Examples

Create 2-D Connectivity Array with Minimal Connectivity

Create a 2-D connectivity array.

```
conn = conndef(2,'minimal')
```

```
conn = 3×3

     0     1     0
     1     1     1
     0     1     0
```

Create 2-D Connectivity Array with Maximal Connectivity

Create a 2-D connectivity array.

```
conn = conndef(2,'maximal')
```

```
conn = 3×3

     1     1     1
     1     1     1
     1     1     1
```

Create 3-D Connectivity Array with Minimal Connectivity

Create a 3-D connectivity array.

```
conndef(3,'minimal')
```

```
ans =
ans(:,:,1) =

     0     0     0
     0     1     0
     0     0     0


ans(:,:,2) =

     0     1     0
     1     1     1
     0     1     0


ans(:,:,3) =

     0     0     0
     0     1     0
     0     0     0
```

Input Arguments

`num_dims` — Number of dimensions

positive integer

Number of dimensions, specified as a positive integer.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`type` — Type of neighborhood connectivity

`'minimal'` | `'maximal'`

Type of neighborhood connectivity, specified as `'minimal'` or `'maximal'`

Value	Description
`'minimal'`	Defines a neighborhood whose neighbors are touching the central element on an (N-1)-dimensional surface, for the N-dimensional case.
`'maximal'`	Defines a neighborhood including neighbors that touch the central element in any way; it is `ones(repmat(3,1,NUM_DIMS))`.

Data Types: `char` | `string`

Output Arguments

`conn` — Pixel connectivity

3-by-3-by...-3 logical array

Pixel connectivity, returned as a 3-by-3-....-by-3 logical array. `conn` is symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `conndef` supports the generation of C code (requires MATLAB Coder). For more information, see "Code Generation for Image Processing".
- When generating code, the `num_dims` and `type` arguments must be compile-time constants.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, the `num_dims` and `type` arguments must be compile-time constants.

See Also

Introduced before R2006a

contains

Determine if image contains points in world coordinate system

Syntax

```
TF = contains(R,xWorld,yWorld)
TF = contains(R,xWorld,yWorld,zWorld)
```

Description

`TF = contains(R,xWorld,yWorld)` returns a logical array `TF`. Each element `TF(k)` is true if and only if the corresponding point (`xWorld(k)`,`yWorld(k)`) falls within the bounds of an image associated with 2-D spatial referencing object `R`.

`TF = contains(R,xWorld,yWorld,zWorld)` indicates whether each point falls within the bounds of an image associated with 3-D spatial referencing object `R`.

Examples

Check If Coordinates Fall Within 2-D Image Bounds

Read a 2-D image into the workspace.

```
I = imread('cameraman.tif');
```

Create an `imref2d` spatial referencing object associated with the image.

```
R = imref2d(size(I))
```

```
R =
  imref2d with properties:

           XWorldLimits: [0.5000 256.5000]
           YWorldLimits: [0.5000 256.5000]
              ImageSize: [256 256]
    PixelExtentInWorldX: 1
    PixelExtentInWorldY: 1
    ImageExtentInWorldX: 256
    ImageExtentInWorldY: 256
       XIntrinsicLimits: [0.5000 256.5000]
       YIntrinsicLimits: [0.5000 256.5000]
```

Check if certain world coordinates are in the image.

```
res = contains(R,[5 8 8],[5 10 257])
```

```
res = 1x3 logical array

   1   1   0
```

This result indicates that the points (5,5) and (8,10) are within the image bounds, and that the point (8, 257) is outside the image bounds. This conclusion is consistent with the `XWorldLimits` and `YWorldLimits` properties of the spatial referencing object `R`.

Check If Coordinates Fall Within 3-D Image Bounds

Read a 3-D image into the workspace. This image consists of 27 frames of 128-by-128 pixel images.

```
load mri;
D = squeeze(D);
```

Create an `imref3d` spatial referencing object associated with the image.

```
R = imref3d(size(D))
```

```
R =
  imref3d with properties:

           XWorldLimits: [0.5000 128.5000]
           YWorldLimits: [0.5000 128.5000]
           ZWorldLimits: [0.5000 27.5000]
              ImageSize: [128 128 27]
    PixelExtentInWorldX: 1
    PixelExtentInWorldY: 1
    PixelExtentInWorldZ: 1
    ImageExtentInWorldX: 128
    ImageExtentInWorldY: 128
    ImageExtentInWorldZ: 27
       XIntrinsicLimits: [0.5000 128.5000]
       YIntrinsicLimits: [0.5000 128.5000]
       ZIntrinsicLimits: [0.5000 27.5000]
```

Check if certain 3-D world coordinates are in the image.

```
res = contains(R,[5 6 6 8],[5 10 10 257],[1 27.5 28 1])
```

```
res = 1x4 logical array

   1   1   0   0
```

This result indicates that the points (5,5,1) and (6,10,27.5) are within the image bounds. The points (6,10,28) and (8,257,1) are outside the image bounds. This conclusion is consistent with the `XWorldLimits`, `YWorldLimits`, and `ZWorldLimits` properties of the spatial referencing object `R`.

Input Arguments

R — Spatial referencing object

`imref2d` or `imref3d` object

Spatial referencing object, specified as an `imref2d` or `imref3d` object. `R` is associated with an image.

`xWorld` — Coordinates along the *x*-dimension in the world coordinate system
numeric scalar or vector

Coordinates along the *x*-dimension in the world coordinate system, specified as a numeric scalar or vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`yWorld` — Coordinates along the *y*-dimension in the world coordinate system
numeric scalar or vector

Coordinates along the *y*-dimension in the world coordinate system, specified as a numeric scalar or vector. `yWorld` is the same length as `xWorld`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`zWorld` — Coordinates along the *z*-dimension in the world coordinate system
numeric scalar or vector

Coordinates along the *z*-dimension in the world coordinate system, specified as a numeric scalar or vector. `zWorld` is the same length as `xWorld` and `yWorld`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`TF` — Flag indicating whether coordinates exist within the bounds of the image
logical scalar or vector

Flag indicating whether coordinates exist within the bounds of the image, returned as a logical scalar or vector. `TF` is the same length as the input coordinate vectors `xWorld`, `yWorld`, and (when relevant) `zWorld`.

Data Types: `logical`

See Also

`imref2d` | `imref3d`

Introduced in R2013a

convmtx2

2-D convolution matrix

Syntax

```
T = convmtx2(H,m,n)
T = convmtx2(H,[m n])
```

Description

`T = convmtx2(H,m,n)` returns the convolution matrix `T` for the matrix `H`. If `X` is an `m`-by-`n` matrix, then `reshape(T*X(:),size(H)+[m n]-1)` is the same as `conv2(X,H)`.

`T = convmtx2(H,[m n])` returns the convolution matrix, where the dimensions `m` and `n` are a two-element vector.

Examples

Create a Convolution Matrix

Show that, for the convolution matrix `T` for the matrix `H`, if `X` is an *m*-by-*n* matrix, then `reshape(T*X(:),size(H)+[m n]-1)` is the same as `conv2(X,H)`

Description of first code block

```
H = ones(3,3)/9; % averaging filter 3-by-3
M = 5;
X = magic(M);
T = convmtx2(H,M,M);
Y1 = reshape(T*X(:), size(H)+[5 5]-1)
```

```
Y1 = 7×7

    1.8889    4.5556    4.6667    3.6667    2.6667    2.5556    1.6667
    4.4444    7.6667    8.5556    6.5556    6.7778    5.8889    3.4444
    4.8889    8.7778   11.1111   10.8889   12.8889   10.5556    5.8889
    4.1111    6.6667   11.0000   13.0000   15.0000   10.6667    4.5556
    2.7778    6.7778   13.1111   15.1111   14.8889    8.5556    3.7778
    2.3333    5.6667   10.5556   10.7778    8.7778    3.8889    1.3333
    1.2222    3.2222    6.0000    5.0000    4.0000    1.2222    1.0000
```

```
Y2 = conv2(X,H)
```

```
Y2 = 7×7

    1.8889    4.5556    4.6667    3.6667    2.6667    2.5556    1.6667
    4.4444    7.6667    8.5556    6.5556    6.7778    5.8889    3.4444
    4.8889    8.7778   11.1111   10.8889   12.8889   10.5556    5.8889
    4.1111    6.6667   11.0000   13.0000   15.0000   10.6667    4.5556
    2.7778    6.7778   13.1111   15.1111   14.8889    8.5556    3.7778
```

```
    2.3333    5.6667   10.5556   10.7778    8.7778    3.8889    1.3333
    1.2222    3.2222    6.0000    5.0000    4.0000    1.2222    1.0000
```

```
isequal(Y1,Y2) % They are the same.
```

```
ans = logical
   0
```

Input Arguments

`H` — Input matrix

numeric array

Input matrix, specified as a numeric array.

Data Types: `double`

`m` — Rows in convolution matrix

numeric scalar

Rows in convolution matrix, specified as a numeric scalar.

Data Types: `double`

`n` — Columns in convolution matrix

numeric scalar

Columns in convolution matrix, specified as a numeric scalar.

Data Types: `double`

`[m n]` — Dimensions of convolution matrix

numeric scalar

Dimensions of convolution matrix, specified as a two-element vector of the form `[m n]`, where `m` is the number of rows and `n` is the number of columns.

Data Types: `double`

Output Arguments

`T` — Convolution matrix

numeric array

Convolution matrix, returned as a numeric array. The output matrix `T` is of class `sparse`. The number of nonzero elements in `T` is no larger than `prod(size(H))*m*n`.

See Also

`conv2` | `convmtx`

Introduced before R2006a

corner

(Not recommended) Find corner points in image

Note `corner` is not recommended. Use `detectHarrisFeatures` or `detectMinEigenFeatures` in Computer Vision Toolbox™ instead.

Syntax

```
C = corner(I)
C = corner(I,method)
C = corner(I,N)
C = corner(I,method,N)
C = corner( ___ ,Name,Value)
```

Description

`C = corner(I)` detects corners in image `I` and returns their coordinates in matrix `C`.

`C = corner(I,method)` detects corners in image `I` using the specified `method`.

`C = corner(I,N)` detects corners in image `I` and returns a maximum of `N` corners.

`C = corner(I,method,N)` detects corners using the specified method and maximum number of corners.

`C = corner( ___ ,Name,Value)` specifies parameters and corresponding values that control various aspects of the corner detection algorithm.

Examples

Find Corners in Images

This example shows how to locate corners with the corner function and adjust your results by refining the maximum number of desired corners.

Create a checkerboard image.

```
I = checkerboard(40,2,2);
```

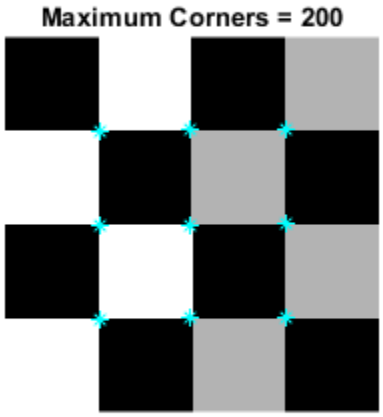
Find the corners in the image.

```
C = corner(I);
```

Display the corners when the maximum number of desired corners is the default setting of 200.

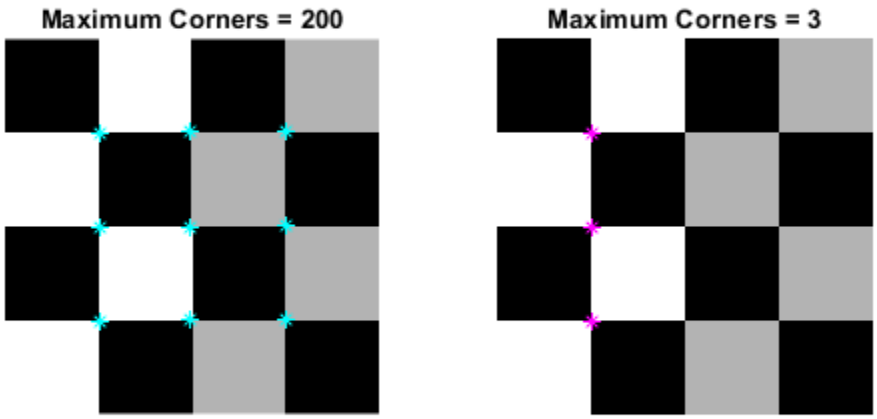
```
subplot(1,2,1);
imshow(I);
hold on
plot(C(:,1), C(:,2), '*', 'Color', 'c')
```

```
title('Maximum Corners = 200')
hold off
```



Display the corners when the maximum number of desired corners is 3.

```
corners_max_specified = corner(I,3);
subplot(1,2,2);
imshow(I);
hold on
plot(corners_max_specified(:,1), corners_max_specified(:,2), ...
   '*', 'Color', 'm')
title('Maximum Corners = 3')
hold off
```



Input Arguments

`I` — Grayscale or binary image

m-by-*n* numeric matrix

Grayscale or binary image, specified as an *m*-by-*n* numeric matrix.

`method` — Corner detection algorithm

`'Harris'` (default) | `'MinimumEigenvalue'`

Corner detection method, specified as `'Harris'` for the Harris corner detector, or `'MinimumEigenvalue'` for Shi & Tomasi's minimum eigenvalue method.

`N` — Maximum number of corners

`200` (default) | positive integer

Maximum number of corners that the `corner` function can return, specified as a positive integer.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `corner(I,'QualityLevel',0.2)` specifies the minimum quality level of corners in image `I` as `0.2`.

`FilterCoefficients` — Filter coefficients
numeric vector

Filter coefficients for the separable smoothing filter, specified as the comma-separated pair consisting of `'FilterCoefficients'` and a numeric vector. The vector, `V`, must have odd length and a minimum length of 3. The outer product, `V*V'`, gives the full filter kernel. The default filter coefficients are given by `fspecial('gaussian',[5 1],1.5)`.

`QualityLevel` — Minimum accepted quality
`0.01` (default) | numeric scalar

Minimum accepted quality of corners, specified as the comma-separated pair consisting of `'QualityLevel'` and a numeric scalar in the range (0, 1). For a quality level `Q`, the toolbox rejects candidate corners with corner metric values less than `Q * max(corner metric)`. Use larger values of *Q* to remove erroneous corners.

`SensitivityFactor` — Sensitivity factor
`0.04` (default) | numeric scalar

Sensitivity factor used in the Harris detection algorithm, specified as the comma-separated pair consisting of `'SensitivityFactor'` and a numeric scalar in the range (0, 0.25). The smaller the sensitivity factor, the more likely the algorithm is to detect sharp corners. Use this parameter with the `'Harris'` method only.

Output Arguments

`C` — Coordinates of corner points
p-by-2 matrix

x and *y* coordinates of the corner points detected in image `I`, returned as a *p*-by-2 matrix.

Data Types: `double`

Tips

The `corner` and `cornermetric` functions both detect corners in images. For most applications, use the streamlined `corner` function to find corners in one step. If you want greater control over corner selection, use the `cornermetric` function to compute a corner metric matrix and then write your own algorithm to find peak values.

Algorithms

The `corner` function performs nonmaxima suppression on candidate corners, and corners are at least two pixels apart.

See Also
`cornermetric` | `detectHarrisFeatures` | `detectMinEigenFeatures`

Introduced in R2010b

cornermetric

(Not recommended) Create corner metric matrix from image

Note `cornermetric` is not recommended. Use `detectHarrisFeatures` or `detectMinEigenFeatures` and the `cornerPoints` object in Computer Vision Toolbox™ instead. For more information, see “Compatibility Considerations”.

Syntax

```
C = cornermetric(I)
C = cornermetric(I,method)
C = cornermetric( ___ ,Name,Value)
```

Description

`C = cornermetric(I)` creates a corner metric matrix by detecting corner features in the input image `I`.

`C = cornermetric(I,method)` creates a corner metric matrix by detecting corner features in the input image `I`. The corner detection method specified by `method` is used for finding the corner features.

`C = cornermetric( ___ ,Name,Value)`specifies options using one or more name-value arguments in addition to the input arguments from any of the previous syntaxes.

Examples

Find Corner Features in a Binary Image

Read an input image into the workspace.

```
I = imread('circles.png');
```

Generate a corner metric matrix. Specify the filter coefficients. The corner detection method takes the default value '`Harris`'.

```
filter = [0.25 0.5 0.25];
C = cornermetric(I,'FilterCoefficients',filter);
```

Use `imregionalmax` to detect corner features (pixels) from the corner metric matrix.

```
corner_peaks = imregionalmax(C);
```

Set the value of the detected corner pixels to [255 0 0].

```
corner_idx = find(corner_peaks == true);
[r,g,b] = deal(I);
r(corner_idx) = 255;
g(corner_idx) = 0;
```

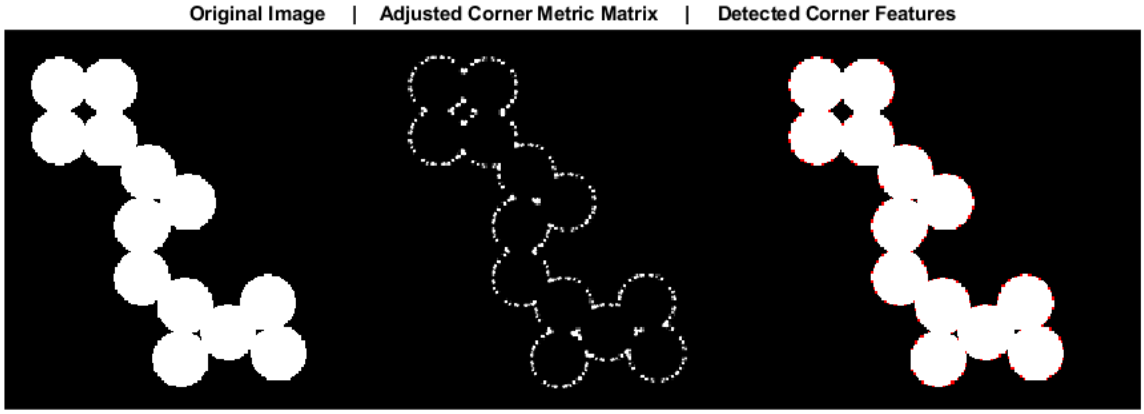
```
b(corner_idx) = 0;
RGB = cat(3,r,g,b);
```

Adjust the corner metric matrix for viewing.

```
C_adjusted = imadjust(C);
```

Display the original image, adjusted corner metric and the detected corner features as a montage. The detected corner features are displayed as red color pixels with RGB value as [255 0 0].

```
montage({I,C_adjusted,RGB},'Size',[1 3])
title('Original Image     |    Adjusted Corner Metric Matrix     |     Detected Corner Features')
```



Find Corner Features in a Grayscale Image

Read an input image into the workspace.

```
I = imread('bag.png');
```

Generate a corner metric matrix. Specify the method as '`MinimumEigenvalue`'.

```
C = cornermetric(I,'MinimumEigenvalue');
```

Use `imregionalmax` to detect corner features (pixels) from the corner metric matrix.

```
corner_peaks = imregionalmax(C);
```

Set the value of the detected corner pixels to [255 0 0].

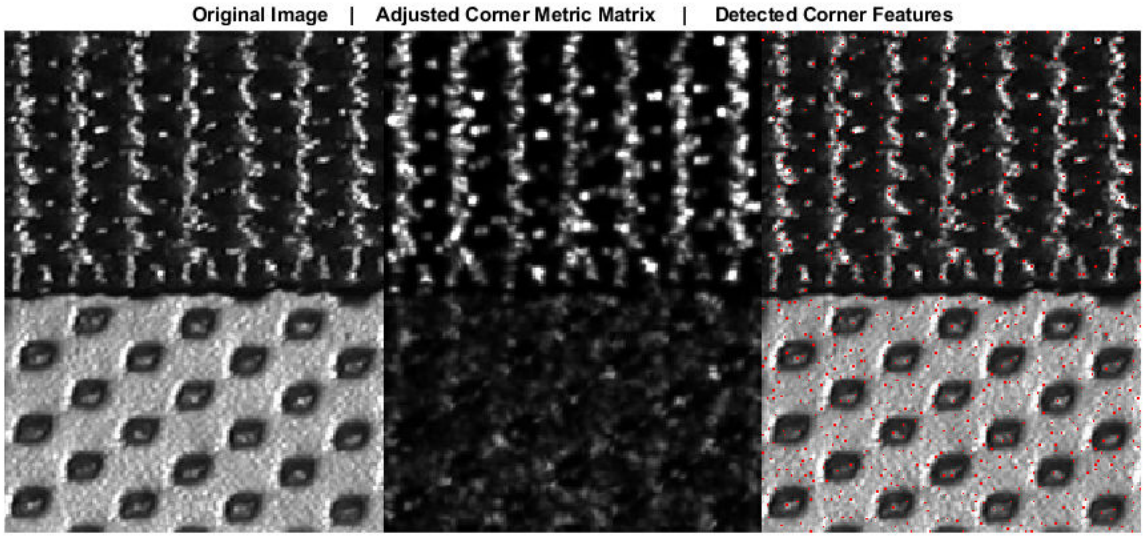
```
corner_idx = find(corner_peaks == true);
[r g b] = deal(I);
r(corner_idx) = 255;
g(corner_idx) = 0;
b(corner_idx) = 0;
RGB = cat(3,r,g,b);
```

Adjust the corner metric matrix for viewing.

```
C_adjusted = imadjust(C);
```

Display the original image, adjusted corner metric and the detected corner features as a montage. The detected corner features are displayed as red color pixels with RGB value as [255 255 0].

```
montage({I,C_adjusted,RGB},'Size',[1 3])
title('Original Image    |    Adjusted Corner Metric Matrix     |     Detected Corner Features')
```



Input Arguments

`I` — Input image

2-D binary image | 2-D grayscale image

Input image, specified as a 2-D binary image or 2-D grayscale image of size *m*-by-*n*.

Data Types: `single` | `double` | `uint8` | `uint16` | `uint32` | `int8` | `int16` | `int32` | `logical`

`method` — Corner detection method

`'Harris'` (default) | `'MinimumEigenvalue'`

Corner detection method, specified as either `'Harris'` or `'MinimumEigenvalue'`. If the method is:

- `'Harris'`, the function creates corner metric matrix by using the Harris corner detector.
- `'MinimumEigenvalue'`, the function creates corner metric matrix by using the Shi and Tomasi's minimum eigenvalue approach.

If `method` is not specified, the default value set as `'Harris'` and the function uses Harris corner detector for detecting corner features.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `cornermetric(I,'SensitivityFactor',0.1)`

`FilterCoefficients` — Coefficients of 1-D spatial filter mask
`[0.1201 0.2339 0.2921 0.2339 0.1201]` (default) | *n*-element vector

Coefficients of 1-D spatial filter mask, specified as a comma-separated pair consisting of `'FilterCoefficients'` and an *n*-element vector. The value of *n* must be odd and greater than or equal to 3. By default, the 1-D spatial filter mask is a 5-element vector and the default filter coefficients are computed using `fspecial('gaussian',[5 1],1.5)`.

`SensitivityFactor` — Sensitivity factor
`0.04` (default) | numeric scalar in the range (0, 0.25)

Sensitivity factor, specified as a comma-separated pair consisting of `'SensitivityFactor'` and a numeric scalar in the interval (0, 0.25). For smaller values of sensitivity factor, the algorithm is more likely to detect sharper corners.

Note The name-value pair `'SensitivityFactor'` is valid only if the input `method` is `'Harris'`.

Output Arguments

`C` — Corner metric matrix
m-by-*n* matrix

Corner metric matrix, returned as a *m*-by-*n* matrix of the same size as the input image `I`.

Data Types: `double`

Tips

The `corner` and `cornermetric` functions both detect corners in images. For most applications, use the streamlined `corner` function to find corners in one step. If you want greater control over corner selection, use the `cornermetric` function to compute a corner metric matrix. Then, write your own algorithm to find peak values in corner metric matrix.

Compatibility Considerations

`cornermetric` is not recommended
Not recommended starting in R2016a

`cornermetric` is not recommended. Instead, use the `detectHarrisFeatures` or `detectMinEigenFeatures` and the `cornerPoints` object in Computer Vision Toolbox.

Use `detectHarrisFeatures` to find corners in an image by using the Harris corner detector method. Use `detectMinEigenFeatures` to find corners in an image by using Shi and Tomasi's

minimum eigenvalue method. The `detectHarrisFeatures` and `detectMinEigenFeatures` functions return the `cornerPoints` object to which the detected corner points are stored.

See Also

`edge` | `corner`

Introduced in R2008b

corr2

2-D correlation coefficient

Syntax

```
R = corr2(A,B)
```

Description

`R = corr2(A,B)` returns the 2-D correlation coefficient `R` between arrays `A` and `B`.

Examples

Compute the correlation coefficient

Compute the correlation coefficient between an image and the same image processed with a median filter.

```
I = imread('pout.tif');
J = medfilt2(I);
R = corr2(I,J)
```

```
R = 0.9959
```

Input Arguments

A — First input array

numeric array | logical array

First input array, specified as a numeric or logical array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

B — Second input array

numeric array | logical array

Second input array, specified as a numeric or logical array. `B` has the same size as the first input array, `A`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

R — Correlation coefficient

numeric scalar

Correlation coefficient, returned as a numeric scalar.

Data Types: `double`

Algorithms

`corr2` computes the correlation coefficient using

$$r = \frac{\sum_m \sum_n (A_{mn} - \bar{A})(B_{mn} - \bar{B})}{\sqrt{\left(\sum_m \sum_n (A_{mn} - \bar{A})^2\right)\left(\sum_m \sum_n (B_{mn} - \bar{B})^2\right)}}$$

where $\bar{A}$ = `mean2(A)`, and $\bar{B}$ = `mean2(B)`.

Extended Capabilities

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`std2` | `corrcoef`

Introduced before R2006a

cp2tform

(Not recommended) Infer spatial transformation from control point pairs

Note `cp2tform` is not recommended. Use `fitgeotrans` instead.

Syntax

```
tform = cp2tform(movingPoints,fixedPoints,transformationType)
tform = cp2tform(movingPoints,fixedPoints,'polynomial',degree)
tform = cp2tform(movingPoints,fixedPoints,'lwm',n)
tform = cp2tform(movingPoints,fixedPoints,'piecewise linear')
[tform,usedMP,usedFP,badMP,badFP] = cp2tform(movingPoints,
fixedPoints,'piecewise linear')

tform = cp2tform(cpstruct,transformationType,___)
[tform,usedMP,usedFP] = cp2tform(cpstruct,transformationType,___)
```

Description

`tform = cp2tform(movingPoints,fixedPoints,transformationType)` infers a spatial transformation from control point pairs and returns this transformation as a `tform` structure. Some of the transformation types have optional additional parameters, shown in the following syntaxes.

`tform = cp2tform(movingPoints,fixedPoints,'polynomial',degree)` lets you specify the order of the polynomials to use.

`tform = cp2tform(movingPoints,fixedPoints,'lwm',n)` creates a mapping by inferring a polynomial at each control point using neighboring control points. The mapping at any location depends on a weighted average of these polynomials. You can optionally specify the number of points, `n`, used to infer each polynomial. The `n` closest points are used to infer a polynomial of order 2 for each control point pair.

`tform = cp2tform(movingPoints,fixedPoints,'piecewise linear')` creates a Delaunay triangulation of the fixed control points, and maps corresponding moving control points to the fixed control points. The mapping is linear (affine) for each triangle and continuous across the control points but not continuously differentiable as each triangle has its own mapping.

`[tform,usedMP,usedFP,badMP,badFP] = cp2tform(movingPoints, fixedPoints,'piecewise linear')` returns in `usedMP` and `usedFP` the control points that were used for the piecewise linear transformation. This syntax also returns in `badMP` and `badFP` the control points that were eliminated because they were middle vertices of degenerate fold-over triangles.

`tform = cp2tform(cpstruct,transformationType,___)` uses a `cpstruct` structure to store the control point coordinates of the moving and fixed images.

`[tform,usedMP,usedFP] = cp2tform(cpstruct,transformationType,___)` also returns in `usedMP` and `usedFP` the control points that were used for the transformation. Unmatched and predicted points are not used. See `cpstruct2pairs`.

Examples

Use Control Points to Create Nonreflective Similarity Transformation Structure

Transform an image, use the `cp2tform` function to return the transformation, and compare the angle and scale of the `tform` to the angle and scale of the original transformation:

```
I = checkerboard;
J = imrotate(I,30);
fixedPoints = [11 11; 41 71];
movingPoints = [14 44; 70 81];
cpselect(J,I,movingPoints,fixedPoints);

t = cp2tform(movingPoints,fixedPoints,'nonreflective similarity');
```

Recover angle and scale by checking how a unit vector parallel to the *x*-axis is rotated and stretched.

```
u = [0 1];
v = [0 0];
[x, y] = tformfwd(t,u,v);
dx = x(2) - x(1);
dy = y(2) - y(1);
angle = (180/pi) * atan2(dy, dx)
scale = 1 / sqrt(dx^2 + dy^2)
```

Input Arguments

`movingPoints` — Control points in the moving image

m-by-2 matrix

Control points in the moving image, specified as an *m*-by-2 matrix. Each row specifies the [*x* *y*] coordinates of a control point.

Example: `[11 11; 41 71]`

Data Types: `double`

`fixedPoints` — Control points in the fixed image

m-by-2 matrix

Control points in the fixed image, specified as an *m*-by-2 matrix. Each row specifies the [*x* *y*] coordinates of a control point.

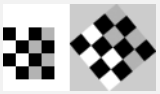
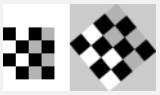
Example: `[14 44; 70 81]`

Data Types: `double`

`transformationType` — Type of transformation

`'nonreflective similarity'` | `'similarity'` | `'affine'` | `'projective'` | `'polynomial'` | `'piecewise linear'` | `'lwm'`

Type of transformation, specified as one of the following, listed in order of increasing complexity. The `cp2tform` function requires a minimum number of control point pairs to infer a structure of each transform type.

Transformation Type	Description	Minimum Number of Control Point Pairs	Example
`'nonreflective similarity'`	Use this transformation when shapes in the moving image are unchanged, but the image is distorted by some combination of translation, rotation, and scaling. Straight lines remain straight, and parallel lines are still parallel.	2	
`'similarity'`	Same as `'nonreflective similarity'` with the addition of optional reflection.	3	
`'affine'`	Use this transformation when shapes in the moving image exhibit shearing. Straight lines remain straight, and parallel lines remain parallel, but rectangles become parallelograms.	3	
`'projective'`	Use this transformation when the scene appears tilted. Straight lines remain straight, but parallel lines converge toward vanishing points that might or might not fall within the image.	4	
`'polynomial'`	Use this transformation when objects in the image are curved. The higher the order of the polynomial, the better the fit, but the result can contain more curves than the fixed image. You can specify the `degree` of the polynomial.	6 (order 2) 10 (order 3) 15 (order 4)	
`'piecewise linear'`	Use this transformation when parts of the image appear distorted differently.	4	
`'lwm'`	Use this transformation (local weighted mean), when the distortion varies locally and piecewise linear is not sufficient. You can specify the number `n` of points to use in the local weighed mean calculation.	6 (12 recommended)	

Data Types: `char`

`cpstruct` — Preselected control points
structure

Preselected control points, specified as a structure. `cpstruct` contains information about the *x*- and *y*-coordinates of all control points in the moving and fixed images, including unpaired and predicted control points. `cpstruct2pairs` eliminates unmatched and predicted control points, and returns the set of valid control point pairs.

`cpstruct` is a structure produced by the Control Point Selection tool (`cpselect`) when you choose the **Export Points to Workspace** option. For more information, see “Export Control Points to the Workspace”.

Data Types: `struct`

degree — Degree of the polynomial
3 (default) | 2 | 4

Degree of the polynomial transformation, specified as the integer 2, 3, or 4.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

n — Number of points to use in local weighted mean calculation
12 (default) | positive integer

Number of points to use in local weighted mean calculation, specified as a positive integer. `n` can be as small as 6, but making `n` small risks generating ill-conditioned polynomials

Output Arguments

tform — Transformation
`TFORM` structure

Transformation, returned as a `TFORM` structure.

usedMP — Used moving points
n-by-2 matrix

Moving control points that were used to infer the spatial transformation, returned as an *n*-by-2 matrix. Unmatched and predicted points are not used.

usedFP — Used fixed points
n-by-2 matrix

Fixed control points that were used to infer the spatial transformation, returned as an *n*-by-2 matrix. Unmatched and predicted points are not used.

badMP — Eliminated moving points
p-by-2 matrix

Moving control points that were eliminated because they were determined to be outliers, returned as a *p*-by-2 matrix.

badFP — Eliminated fixed points
p-by-2 matrix

Fixed control points that were eliminated because they were determined to be outliers, returned as a *p*-by-2 matrix.

Tips

- When `transformtype` is `'nonreflective similarity'`, `'similarity'`, `'affine'`, `'projective'`, or `'polynomial'`, and `movingPoints` and `fixedPoints` (or `cpstruct`) have the minimum number of control points needed for a particular transformation, `cp2tform` finds the coefficients exactly.
- If `movingPoints` and `fixedPoints` have more than the minimum number of control points, a least-squares solution is found. See `mldivide`.
- When either `movingPoints` or `fixedPoints` has a large offset with respect to their origin (relative to range of values that it spans), `cp2tform` shifts the points to center their bounding box

on the origin before fitting a `tform` structure. This enhances numerical stability and is handled transparently by wrapping the origin-centered `tform` within a custom `tform` that automatically applies and undoes the coordinate shift as needed. As a result, `fields(T)` can give different results for different coordinate inputs, even for the same transformation type.

Algorithms

`cp2tform` uses the following general procedure:

1. Use valid pairs of control points to infer a spatial transformation or an inverse mapping from output space (*x*,*y*) to input space (*x*,*y*) according to `transformtype`.
2. Return the `tform` structure containing spatial transformation.

The procedure varies depending on the `transformtype`.

Nonreflective Similarity

Nonreflective similarity transformations can include a rotation, a scaling, and a translation. Shapes and angles are preserved. Parallel lines remain parallel. Straight lines remain straight.

Let

```
sc = scale*cos(angle)
ss = scale*sin(angle)

[u v] = [x y 1] * [ sc -ss
                    ss  sc
                    tx  ty]
```

Solve for `sc`, `ss`, `tx`, and `ty`.

Similarity

Similarity transformations can include rotation, scaling, translation, and reflection. Shapes and angles are preserved. Parallel lines remain parallel. Straight lines remain straight.

Let

```
sc = s*cos(theta)
ss = s*sin(theta)

                    [ sc -a*-ss
 [u v] = [x y 1] *    ss  a*sc
                      tx  ty]
```

Solve for `sc`, `ss`, `tx`, `ty`, and `a`. If `a = -1`, reflection is included in the transformation. If `a = 1`, reflection is not included in the transformation.

Affine

In an affine transformation, the *x* and *y* dimensions can be scaled or sheared independently and there can be a translation. Parallel lines remain parallel. Straight lines remain straight. Nonreflective similarity transformations are a subset of affine transformations.

For an affine transformation,

```
[u v] = [x y 1] * Tinv
```

`Tinv` is a 3-by-2 matrix. Solve for the six elements of `Tinv`:

```
t_affine = cp2tform(movingPoints,fixedPoints,'affine');
```

The coefficients of the inverse mapping are stored in `t_affine.tdata.Tinv`.

At least three control-point pairs are needed to solve for the six unknown coefficients.

Projective

In a projective transformation, quadrilaterals map to quadrilaterals. Straight lines remain straight. Affine transformations are a subset of projective transformations.

For a projective transformation,

```
[up vp wp] = [x y w] * Tinv
```

where

```
u = up/wp
v = vp/wp
```

`Tinv` is a 3-by-3 matrix.

Assuming

```
Tinv = [ A D G;
         B E H;
         C F I ];
u = (Ax + By + C)/(Gx + Hy + I)
v = (Dx + Ey + F)/(Gx + Hy + I)
```

Solve for the nine elements of `Tinv`:

```
t_proj = cp2tform(movingPoints,fixedPoints,'projective');
```

The coefficients of the inverse mapping are stored in `t_proj.tdata.Tinv`.

At least four control-point pairs are needed to solve for the nine unknown coefficients.

Note An affine or projective transformation can also be expressed like this, for a 3-by-2 `Tinv`:

```
[u v]'  =  Tinv' * [x y 1]'
```

Or, like this, for a 3-by-3 `Tinv`:

```
[u v 1]'  =  Tinv' * [x y 1]'
```

Polynomial

In a polynomial transformation, polynomial functions of `x` and `y` determine the mapping.

Second-Order Polynomials

For a second-order polynomial transformation,

```
[u v] = [1  x  y  x*y  x^2  y^2] * Tinv
```

Both `u` and `v` are second-order polynomials of `x` and `y`. Each second-order polynomial has six terms. To specify all coefficients, `Tinv` has size 6-by-2.

```
t_poly_ord2 = cp2tform(movingPoints,fixedPoints,'polynomial');
```

The coefficients of the inverse mapping are stored in `t_poly_ord2.tdata`.

At least six control-point pairs are needed to solve for the 12 unknown coefficients.

Third-Order Polynomials

For a third-order polynomial transformation:

```
[u v] = [1  x  y  x*y  x^2  y^2  y*x^2  x*y^2  x^3  y^3] * Tinv
```

Both `u` and `v` are third-order polynomials of `x` and `y`. Each third-order polynomial has 10 terms. To specify all coefficients, `Tinv` has size 10-by-2.

```
t_poly_ord3 = cp2tform(movingPoints, fixedPoints,'polynomial',3);
```

The coefficients of the inverse mapping are stored in `t_poly_ord3.tdata`.

At least ten control-point pairs are needed to solve for the 20 unknown coefficients.

Fourth-Order Polynomials

For a fourth-order polynomial transformation:

```
[u v] = [1 x y x*y x^2 y^2 y*x^2 x*y^2 x^3 y^3 x^3*y x^2*y^2 x*y^3 x^4 y^4] * Tinv
```

Both `u` and `v` are fourth-order polynomials of `x` and `y`. Each fourth-order polynomial has 15 terms. To specify all coefficients, `Tinv` has size 15-by-2.

```
t_poly_ord4 = cp2tform(movingPoints, fixedPoints,'polynomial',4);
```

The coefficients of the inverse mapping are stored in `t_poly_ord4.tdata`.

At least 15 control-point pairs are needed to solve for the 30 unknown coefficients.

Piecewise Linear

In a piecewise linear transformation, linear (affine) transformations are applied separately to each triangular region of the image[1].

1. Find a Delaunay triangulation of the fixed control points.
2. Using the three vertices of each triangle, infer an affine mapping from fixed to moving coordinates.

Note At least four control-point pairs are needed. Four pairs result in two triangles with distinct mappings.

Local Weighted Mean

For each control point in `fixedPoints`:

1. Find the `N` closest control points.
2. Use these `N` points and their corresponding points in `movingPoints` to infer a second-order polynomial.
3. Calculate the radius of influence of this polynomial as the distance from the center control point to the farthest point used to infer the polynomial (using `fixedPoints`)[2].

Note At least six control-point pairs are needed to solve for the second-order polynomial. Ill-conditioned polynomials might result if too few pairs are used.

References

[1] Goshtasby, Ardeshir, "Piecewise linear mapping functions for image registration," *Pattern Recognition*, Vol. 19, 1986, pp. 459-466.

[2] Goshtasby, Ardeshir, "Image registration by local approximation methods," *Image and Vision Computing*, Vol. 6, 1988, pp. 255-261.

See Also

`cpcorr` | `cpselect` | `cpstruct2pairs`

Introduced before R2006a

cpcorr

Tune control point locations using cross-correlation

Syntax

```
movingPointsAdjusted = cpcorr(movingPoints,fixedPoints,moving,fixed)
```

Description

`movingPointsAdjusted = cpcorr(movingPoints,fixedPoints,moving,fixed)` adjusts the position of moving control points, `movingPoints`, with respect to fixed control points, `fixedPoints`, using normalized cross-correlation between the moving image `moving` and the fixed image `fixed`. The `cpcorr` function returns the adjusted moving control points in `movingPointsAdjusted`.

Examples

Fine-Tune Control-Point Locations using Cross Correlation

Read two images into the workspace.

```
moving = imread('onion.png');
fixed = imread('peppers.png');
```

Define sets of control points for both images.

```
movingPoints = [118 42;99 87];
fixedPoints = [190 114;171 165];
```

Display the images, and display the control points in white. The position of the moving points is slightly offset from the position of the fixed points.

```
imshow(fixed)
hold on
plot(fixedPoints(:,1),fixedPoints(:,2),'xw')
title('Fixed Image')
```


```
figure
imshow(moving)
hold on
plot(movingPoints(:,1),movingPoints(:,2),'xw')
title('Moving Image')
```

Adjust the moving control points using cross correlation.

```
movingPointsAdjusted = cpcorr(movingPoints,fixedPoints, ...
    moving(:,:,1),fixed(:,:,1))
```

```
movingPointsAdjusted = 2×2

  115.9000   39.1000
   97.0000   89.9000
```

Display the adjusted moving points in yellow. Compared to the original moving points (in white), the adjusted points more closely match the positions of the fixed points.

```
plot(movingPointsAdjusted(:,1),movingPointsAdjusted(:,2),'xy')
```

Input Arguments

`movingPoints` — Coordinates of control points in image to be transformed

m-by-2 matrix

Coordinates of control points in the image to be transformed, specified as an *m*-by-2 matrix. The two columns represent the *x*- and *y*-coordinates of the control points, respectively, in the intrinsic coordinate system of the image.

Example: `[127 93; 74 59]`

Data Types: `double`

`fixedPoints` — Coordinates of control points in reference image

p-by-2 matrix

Coordinates of control points in the reference image, specified as an *p*-by-2 matrix. The two columns represent the *x*- and *y*-coordinates of the control points, respectively, in the intrinsic coordinate system of the image.

Example: `[323 195; 269 161]`

Data Types: `double`

`moving` — Image to be registered

numeric array

Image to be registered, specified as a numeric array.

`fixed` — Reference image in target orientation
numeric array

Reference image in the target orientation, specified as a numeric array.

Output Arguments

`movingPointsAdjusted` — Adjusted coordinates of control points in the image to be transformed
numeric matrix

Adjusted coordinates of control points in the image to be transformed, returned as a numeric matrix of the same size as `movingPoints`.

Data Types: `double`

Tips

- The `moving` and `fixed` images must have the same scale for `cpcorr` to be effective.
- If `cpcorr` cannot correlate a pair of control points, `movingPointsAdjusted` contains the same coordinates as `movingPoints` for that pair.
- `cpcorr` cannot adjust a point if any of these conditions occur:
  - points are too near the edge of either image
  - regions of images around points contain `Inf` or `NaN`
  - region around a point in moving image has zero standard deviation
  - regions of images around points are poorly correlated

Algorithms

`cpcorr` only moves the position of a control point by up to four pixels. Adjusted coordinates are accurate up to one-tenth of a pixel. `cpcorr` is designed to get subpixel accuracy from the image content and coarse control point selection.

Extended Capabilities

Thread-Based Environment
Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also
`cpselect` | `fitgeotrans` | `normxcorr2` | `imwarp`

Topics
“Control Point Selection Procedure”

“Image Coordinate Systems”

Introduced before R2006a

cpselect

Control Point Selection tool

Syntax

```
cpselect(moving,fixed)
cpselect(moving,fixed,initialMovingPoints,initialFixedPoints)
cpselect(moving,fixed,cpstruct_in)
h = cpselect( ___ )
h = cpselect( ___ ,'Wait',false)
[selectedMovingPoints,selectedFixedPoints] = cpselect( ___ ,'Wait',true)
```

Description

`cpselect(moving,fixed)` starts the Control Point Selection tool that enables you to select control points in two related images. `moving` is the image to be warped, which brings it into the coordinate system of the `fixed` image.

When the Control Point Selection tool is open, you can add, move, and delete control points interactively with the mouse. When you are done modifying the control points, export them to the workspace by selecting **Export Points to Workspace** from the **File** menu. The tool can return the coordinates of valid selected pairs of moving and fixed control points in two numeric vectors. The tool can also return all selected control points and indexing information in a `cpstruct` structure that saves the state of the tool so that you can restart the tool later. For more information about using the tool, see “Control Point Selection Procedure”.

`cpselect(moving,fixed,initialMovingPoints,initialFixedPoints)` starts the Control Point Selection tool with an initial set of valid moving and fixed control point pairs, `initialMovingPoints` and `initialFixedPoints`.

`cpselect(moving,fixed,cpstruct_in)` starts the Control Point Selection tool with an initial set of control points and indexing information that are stored in `cpstruct_in`. Use this syntax to restart the Control Point Selection tool from a previously saved state.

`h = cpselect( ___ )` returns a handle `h` to the Control Point Selection tool. You can use the `close(h)` command to close the tool from the command line.

`h = cpselect( ___ ,'Wait',false)` returns a handle `h` to the Control Point Selection tool. You can use the `close(h)` syntax to close the tool from the command line. In contrast to setting `'Wait'` as `true`, this syntax lets you run `cpselect` at the same time as you run other programs in MATLAB.

`[selectedMovingPoints,selectedFixedPoints] = cpselect( ___ ,'Wait',true)` takes control of the MATLAB command line until you finish selecting control points. When you have finished selecting control points, return to the workspace by closing the tool. `cpselect` returns the coordinates of valid selected pairs of moving and fixed control points in `selectedMovingPoints` and `selectedFixedPoints`.

Examples

Start Control Point Selection Tool with Saved Images

Read the image `westconcordorthophoto.png` into the workspace. This image is an orthophoto that has already been registered to the ground.

```
fixed = imread('westconcordorthophoto.png');
```

Read the image `westconcordaerial.png` into the workspace. This image was taken from an airplane and is distorted relative to the orthophoto.

```
moving = imread('westconcordaerial.png');
```

Call `cpselect`, specifying the names of the image you want to register and the reference image. You can now add, move, and delete control points interactively with the mouse. When you are done adding control points, export them to the workspace by selecting **Export Points to Workspace** from the **File** menu.

```
cpselect('westconcordaerial.png','westconcordorthophoto.png');
```

Open Control Point Selection Tool with Predefined Control Points

Create a sample reference image. This image is the fixed image.

```
I = checkerboard;
```

Create a rotated and stretched copy of the sample image. This image is the moving image to be aligned with the fixed image.

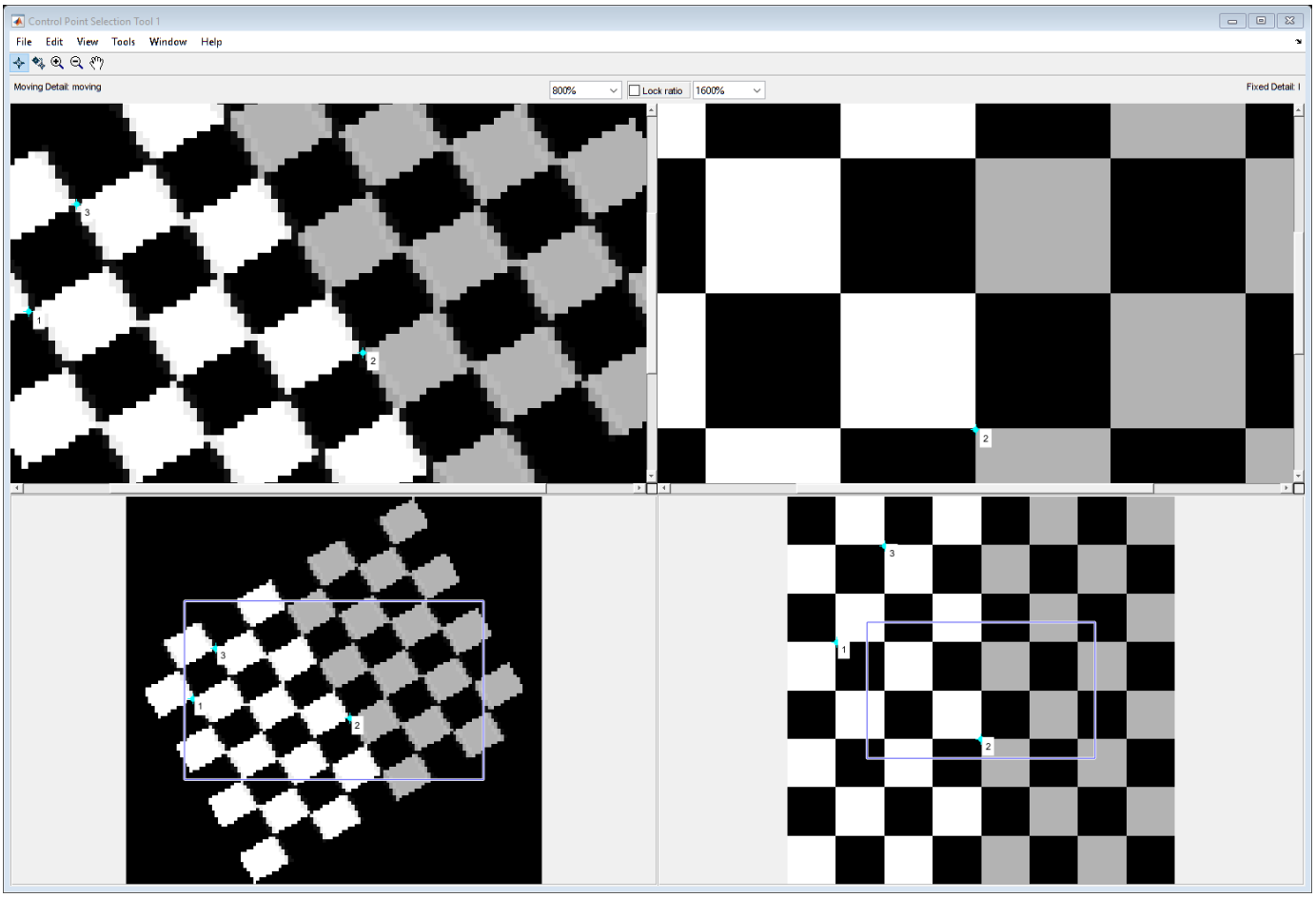
```
J = imresize(I,'Scale',[1 1.3]);
moving = imrotate(J,30);
```

Specify the (*x*,*y*) coordinates of three corresponding control points for the fixed and moving images.

```
fixedPoints = [10.7 30.6; 40.5 50.6; 20.6 10.7];
movingPoints = [21.6 64.2; 71.1 70.3; 28.7 48.3];
```

Open the Control Point Selection tool, specifying the sample fixed and moving images and the two sets of saved control points. You can now continue adding, moving, and deleting control points interactively with the mouse. When you are done modifying the control points, export them to the workspace by selecting **Export Points to Workspace** from the **File** menu.

```
h = cpselect(moving,I,movingPoints,fixedPoints);
```



Close the Control Point Selection tool programmatically by using the `close` function.

```
close(h)
```

Input Arguments

`moving` — Input image to be aligned

grayscale image | truecolor image | binary image | character vector | string

Input image to be aligned, specified as a grayscale, truecolor, or binary image, or a character vector or string that specifies the file name of an image of those types.

Image Type	Supported Data Types
Grayscale	`uint8`, `uint16`, `int16`, `single`, or `double`
Truecolor	`uint8`, `uint16`, `single`, or `double`
Binary	`logical`

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical` | `char` | `string`

`fixed` — Reference image

grayscale image | truecolor image | binary image | character vector | string

Reference image, specified as a grayscale, truecolor, or binary image, or a character vector or string that specifies the file name of an image of those types.

Image Type	Supported Data Types
Grayscale	`uint8`, `uint16`, `int16`, `single`, or `double`
Truecolor	`uint8`, `uint16`, `single`, or `double`
Binary	`logical`

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical` | `char` | `string`

`cpstruct_in` — Preselected control points

structure

Preselected control points, specified as a `cpstruct` structure. `cpstruct_in` contains information about *x*- and *y*-coordinates of all control points in the moving and fixed images, including unpaired and predicted control points. `cpstruct_in` also contains indexing information that allows the Control Point Selection tool to restore the state of the control points.

Create a `cpstruct` by exporting points from the Control Point Selection tool, described in “Export Control Points to the Workspace”.

Data Types: `struct`

`initialMovingPoints` — Preselected control points on moving image

m-by-2 numeric array

Preselected control points on the moving image, specified as an *m*-by-2 numeric array. The two columns represent the *x*- and *y*-coordinates of the control points.

Data Types: `double`

`initialFixedPoints` — Preselected control points on fixed image

m-by-2 numeric array

Preselected control points on the fixed image, specified as an *m*-by-2 numeric array. The two columns represent the *x*- and *y*-coordinates of the control points.

Data Types: `double`

Output Arguments

`h` — Control Point Selection tool

handle

Control Point Selection tool, returned as a handle.

`selectedMovingPoints` — Selected control points on moving image

p-by-2 numeric array

Selected control points on the moving image, specified as a *p*-by-2 numeric array. The two columns represent the *x*- and *y*-coordinates of the control points, respectively, in the intrinsic coordinate system of the image.

Data Types: `double`

`selectedFixedPoints` — Selected control points on fixed image

p-by-2 numeric array

Selected control points on the fixed image, specified as a *p*-by-2 numeric array. The two columns represent the *x*- and *y*-coordinates of the control points, respectively, in the intrinsic coordinate system of the image.

Data Types: `double`

Tips

- When calling `cpselect` in a script, specify the `'Wait'` option as `true`. The `'Wait'` option causes `cpselect` to block the MATLAB command line until control points have been selected and returned. If you do not use the `'Wait'` option, `cpselect` returns control immediately and your script continues without allowing time for control point selection. Additionally, without the `'Wait'` option, `cpselect` does not return the control points as return values.

Algorithms

`cpselect` uses the following general procedure for control-point prediction.

1. Find all valid pairs of control points.
2. Infer a spatial transformation between `moving` and `fixed` control points using a method that depends on the number of valid control point pairs.

Transformation Type	Minimum Number of Control Point Pairs
Nonreflective similarity	2
Affine	3
Projective	4

3. Apply the spatial transformation to the new point. This transformation generates the predicted point.
4. Display the predicted point.

See Also

`cpcorr` | `fitgeotrans` | `imwarp` | `cpstruct2pairs`

Topics

“Register Images with Projection Distortion Using Control Points”
“Control Point Selection Procedure”
“Export Control Points to the Workspace”
“Image Coordinate Systems”

Introduced before R2006a

cpstruct2pairs

Extract valid control point pairs from `cpstruct` structure

Syntax

```
[movingPoints,fixedPoints] = cpstruct2pairs(cpstruct_in)
```

Description

`[movingPoints,fixedPoints] = cpstruct2pairs(cpstruct_in)` extracts the valid control point pairs from `cpstruct_in`, returning two arrays `movingPoints` and `fixedPoints`.

Examples

Convert cpstruct to Sets of Control Point Pairs

Read an aerial photograph and an orthoregistered image into the workspace.

```
aerial = imread('westconcordaerial.png');
ortho = imread('westconcordorthophoto.png');
```

Load some preselected control points for these images.

```
load westconcordpoints
whos
```

```
  Name                Size                Bytes  Class     Attributes

  aerial            394x369x3            436158  uint8
  fixedPoints         4x2                    64  double
  movingPoints        4x2                    64  double
  ortho             366x364                133224  uint8
```

Open the Control Point Selection tool, specifying the two images along with the predefined control points.

```
cpselect(aerial,ortho,movingPoints,fixedPoints);
```

Create the `cpstruct` structure. Using the Control Point Selection tool, select **Export Points to Workspace** from the **File** menu to save the points to the workspace. On the **Export Points to Workspace** dialog box, check the **Structure with all points** check box, and clear **Moving points of valid pairs** and **Fixed points of valid pairs**. Click **OK**. Close the Control Point Selection tool.

Use `cpstruct2pairs` to extract the moving and fixed points from the `cpstruct`.

```
[mPoints,fPoints] = cpstruct2pairs(cpstruct);
```

Compare the stored set of points with the set of points you exported.

```
fixedPoints, fpoints
```

```
fixedPoints =

  164.5639  113.2890
  353.5325  130.0798
  143.4046  284.8935
  353.5325  311.9810


fpoints =

  164.5639  113.2890
  353.5325  130.0798
  143.4046  284.8935
  353.5325  311.9810
```

The two sets of points are identical, which indicates that all points in the stored set of points belong to valid control point pairs.

Input Arguments

`cpstruct_in` — Preselected control points

structure

Preselected control points, specified as a structure (`cpstruct`). `cpstruct_in` contains information about the *x*- and *y*-coordinates of all control points in the moving and fixed images, including unpaired and predicted control points. `cpstruct2pairs` eliminates unmatched and predicted control points, and returns the set of valid control point pairs.

`cpstruct_in` is a structure produced by the Control Point Selection tool (`cpselect`) when you choose the **Export Points to Workspace** option. For more information, see “Export Control Points to the Workspace”.

Data Types: `struct`

Output Arguments

`movingPoints` — Control point pairs from moving image being aligned

m-by-2 numeric array

Control point pairs from image being aligned, returned as an *m*-by-2 numeric array. The two columns represent the *x*- and *y*-coordinates of the control points, respectively, in the intrinsic coordinate system of the image.

Data Types: `double`

`fixedPoints` — Control point pairs from reference image

m-by-2 numeric array

Control point pairs from reference image, returned as an *m*-by-2 numeric array. The two columns represent the *x*- and *y*-coordinates of the control points, respectively, in the intrinsic coordinate system of the image.

Data Types: `double`

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

`cpselect` | `fitgeotrans`

Topics

“Control Point Selection Procedure”
“Export Control Points to the Workspace”
“Image Coordinate Systems”

Introduced before R2006a

cycleGANGenerator

Create CycleGAN generator network for image-to-image translation

Syntax

```
net = cycleGANGenerator(inputSize)
net = cycleGANGenerator(inputSize,Name,Value)
```

Description

`net = cycleGANGenerator(inputSize)` creates a CycleGAN generator network for input of size `inputSize`. For more information about the network architecture, see “CycleGAN Generator Network” on page 1-611.

This function requires Deep Learning Toolbox.

`net = cycleGANGenerator(inputSize,Name,Value)` modifies aspects of the CycleGAN network using name-value arguments.

Examples

Create CycleGAN Generator

Specify the network input size for RGB images of size 256-by-256.

```
inputSize = [256 256 3];
```

Create a CycleGAN generator that generates RGB images of the input size.

```
net = cycleGANGenerator(inputSize)
```

```
net =
  dlnetwork with properties:

         Layers: [72x1 nnet.cnn.layer.Layer]
    Connections: [80x2 table]
     Learnables: [94x3 table]
          State: [0x3 table]
     InputNames: {'inputLayer'}
    OutputNames: {'fActivation'}
    Initialized: 1
```

Display the network.

```
analyzeNetwork(net)
```

Create CycleGAN Generator with Six Residual Blocks

Specify the network input size for RGB images of size 128-by-128 pixels.

```
inputSize = [128 128 3];
```

Create a CycleGAN generator with six residual blocks. Add the prefix "cycleGAN6_" to all layer names.

```
net = cycleGANGenerator(inputSize,"NumResidualBlocks",6, ...
    "NamePrefix","cycleGAN6_")
```

```
net =
  dlnetwork with properties:

         Layers: [54x1 nnet.cnn.layer.Layer]
    Connections: [59x2 table]
     Learnables: [70x3 table]
          State: [0x3 table]
     InputNames: {'cycleGAN6_inputLayer'}
    OutputNames: {'cycleGAN6_fActivation'}
    Initialized: 1
```

Display the network.

```
analyzeNetwork(net)
```

Input Arguments

`inputSize` — Network input size

3-element vector of positive integers

Network input size, specified as a 3-element vector of positive integers. `inputSize` has the form [*H W C*], where *H* is the height, *W* is the width, and *C* is the number of channels.

Example: `[28 28 3]` specifies an input size of 28-by-28 pixels for a 3-channel image.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'NumFiltersInFirstBlock',32` creates a network with 32 filters in the first convolution layer

`NumDownsamplingBlocks` — Number of downsampling blocks

2 (default) | positive integer

Number of downsampling blocks in the network encoder module, specified as a positive integer. In total, the network downsamples the input by a factor of `2^NumDownsamplingBlocks`. The decoder module consists of the same number of upsampling blocks.

`NumFiltersInFirstBlock` — Number of filters in first convolution layer
64 (default) | positive even integer

Number of filters in the first convolution layer, specified as a positive even integer.

`NumOutputChannels` — Number of output channels
`"auto"` (default) | positive integer

Number of output channels, specified as `"auto"` or a positive integer. When you specify `"auto"`, the number of output channels is the same as the number of input channels.

`FilterSizeInFirstAndLastBlocks` — Filter size in first and last convolution layers
7 (default) | positive odd integer | 2-element vector of positive odd integers

Filter size in the first and last convolution layers, specified as a positive odd integer or 2-element vector of positive odd integers of the form [*height width*]. When you specify the filter size as a scalar, the filter has identical height and width.

`FilterSizeInIntermediateBlocks` — Filter size in intermediate convolution layers
3 (default) | 2-element vector of positive odd integers | positive odd integer

Filter size in intermediate convolution layers, specified as a positive odd integer or 2-element vector of positive odd integers of the form [*height width*]. The intermediate convolution layers are the convolution layers excluding the first and last convolution layer. When you specify the filter size as a scalar, the filter has identical height and width. Typical values are between 3 and 7.

`NumResidualBlocks` — Number of residual blocks
9 (default) | positive integer

Number of residual blocks, specified as a positive integer. Typically, this value is set to 6 for images of size 128-by-128 and 9 for images of size 256-by-256 or larger.

`ConvolutionPaddingValue` — Style of padding
`"symmetric-exclude-edge"` (default) | `"replicate"` | `"symmetric-include-edge"` | numeric scalar

Style of padding used in the network, specified as one of these values.

PaddingValue	Description	Example
Numeric scalar	Pad with the specified numeric value	$\begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \\ 2 & 6 & 5 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 3 & 1 & 4 & 2 & 2 \\ 2 & 2 & 1 & 5 & 9 & 2 & 2 \\ 2 & 2 & 2 & 6 & 5 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{bmatrix}$

PaddingValue	Description	Example
`'symmetric-include-edge'`	Pad using mirrored values of the input, including the edge values	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}5&1&1&5&9&9&5\\1&3&3&1&4&4&1\\1&3&3&1&4&4&1\\5&1&1&5&9&9&5\\6&2&2&6&5&5&6\\6&2&2&6&5&5&6\\5&1&1&5&9&9&5\end{bmatrix}$
`'symmetric-exclude-edge'`	Pad using mirrored values of the input, excluding the edge values	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}5&6&2&6&5&6&2\\9&5&1&5&9&5&1\\4&1&3&1&4&1&3\\9&5&1&5&9&5&1\\5&6&2&6&5&6&2\\9&5&1&5&9&5&1\\4&1&3&1&4&1&3\end{bmatrix}$
`'replicate'`	Pad using repeated border elements of the input	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}3&3&3&1&4&4&4\\3&3&3&1&4&4&4\\3&3&3&1&4&4&4\\1&1&1&5&9&9&9\\2&2&2&6&5&5&5\\2&2&2&6&5&5&5\\2&2&2&6&5&5&5\end{bmatrix}$

`UpsampleMethod` — Method used to upsample activations
`"transposedConv"` (default) | `"bilinearResize"` | `"pixelShuffle"`

Method used to upsample activations, specified as one of these values:

- `"transposedConv"` — Use a `transposedConv2dLayer` with a stride of [2 2]
- `"bilinearResize"` — Use a `convolution2dLayer` with a stride of [1 1] followed by a `resize2dLayer` with a scale of [2 2]
- `"pixelShuffle"` — Use a `convolution2dLayer` with a stride of [1 1] followed by a `depthToSpace2dLayer` with a block size of [2 2]

Data Types: `char` | `string`

`ConvolutionWeightsInitializer` — Weight initialization used in convolution layers
`"narrow-normal"` (default) | `"glorot"` | `"he"` | function

Weight initialization used in convolution layers, specified as `"glorot"`, `"he"`, `"narrow-normal"`, or a function handle. For more information, see “Specify Custom Weight Initialization Function” (Deep Learning Toolbox).

`ActivationLayer` — Activation function
`"relu"` (default) | `"leakyRelu"` | `"elu"` | layer object

Activation function to use in the network, specified as one of these values. For more information and a list of available layers, see “Activation Layers” (Deep Learning Toolbox).

- `"relu"` — Use a `reluLayer`
- `"leakyRelu"` — Use a `leakyReluLayer` with a scale factor of 0.2
- `"elu"` — Use an `eluLayer`
- A layer object

`FinalActivationLayer` — Activation function after final convolution
`"tanh"` (default) | `"none"` | `"sigmoid"` | `"softmax"` | layer object

Activation function after the final convolution layer, specified as one of these values. For more information and a list of available layers, see “Output Layers” (Deep Learning Toolbox).

- `"tanh"` — Use a `tanhLayer`
- `"sigmoid"` — Use a `sigmoidLayer`
- `"softmax"` — Use a `softmaxLayer`
- `"none"` — Do not use a final activation layer
- A layer object

`NormalizationLayer` — Normalization operation
`"instance"` (default) | `"none"` | `"batch"` | layer object

Normalization operation to use after each convolution, specified as one of these values. For more information and a list of available layers, see “Normalization, Dropout, and Cropping Layers” (Deep Learning Toolbox).

- `"instance"` — Use an `instanceNormalizationLayer`
- `"batch"` — Use a `batchNormalizationLayer`
- `"none"` — Do not use a normalization layer
- A layer object

`Dropout` — Probability of dropout
`0` (default) | number in the range [0, 1]

Probability of dropout, specified as a number in the range [0, 1]. If you specify a value of `0`, then the network does not include dropout layers. If you specify a value greater than `0`, then the network includes a `dropoutLayer` in each residual block.

`NamePrefix` — Prefix to all layer names
`""` (default) | string | character vector

Prefix to all layer names in the network, specified as a string or character vector.

Data Types: `char` | `string`

Output Arguments

`net` — CycleGAN generator network
`dlnetwork` object

CycleGAN generator network, returned as a `dlnetwork` object.

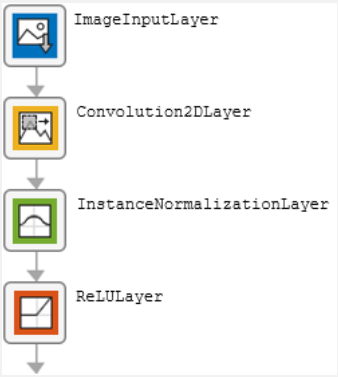
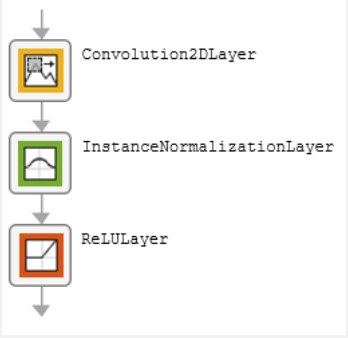
More About

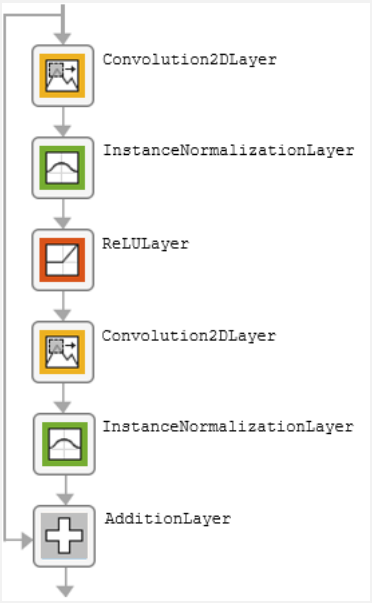
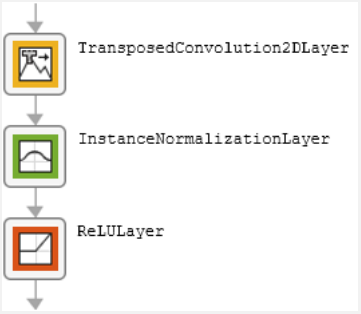
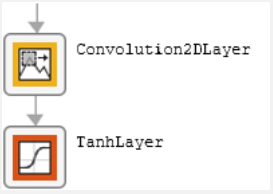
CycleGAN Generator Network

A cycleGAN generator network consists of an encoder module followed by a decoder module. The default network follows the architecture proposed by Zhu et. al. [1].

The encoder module downsamples the input by a factor of `2^NumDownsamplingBlocks`. The encoder module consists of an initial block of layers, `NumDownsamplingBlocks` downsampling blocks, and `NumResidualBlocks` residual blocks. The decoder module upsamples the input by a factor of `2^NumDownsamplingBlocks`. The decoder module consists of `NumDownsamplingBlocks` upsampling blocks and a final block.

The table describes the blocks of layers that comprise the encoder and decoder modules.

Block Type	Layers	Diagram of Default Block
Initial block	• An `imageInputLayer`. • A `convolution2dLayer` with a stride of [1 1] and a filter size of `FilterSizeInFirstAndLastBlocks`. • An optional normalization layer, specified by the `NormalizationLayer` name-value argument. • An activation layer specified by the `ActivationLayer` name-value argument.	
Downsampling block	• A `convolution2dLayer` with a stride of [2 2] to perform downsampling. The convolution layer has a filter size of `FilterSizeInIntermediateBlocks`. • An optional normalization layer, specified by the `NormalizationLayer` name-value argument. • An activation layer specified by the `ActivationLayer` name-value argument.	

Block Type	Layers	Diagram of Default Block
Residual block	• A `convolution2dLayer` with a stride of [1 1] and a filter size of `FilterSizeInIntermediateBlocks`. • An optional normalization layer, specified by the `NormalizationLayer` name-value argument. • An activation layer specified by the `ActivationLayer` name-value argument. • An optional `dropoutLayer`. By default, residual blocks omit a dropout layer. Include a dropout layer by specifying the `Dropout` name-value argument as a value in the range (0, 1]. • A second `convolution2dLayer`. • An optional second normalization layer. • An `additionLayer` that provides a skip connection between every block.	
Upsampling block	• An upsampling layer that upsamples by a factor of 2 according to the `UpsampleMethod` name-value argument. The convolution layer has a filter size of `FilterSizeInIntermediateBlocks`. • An optional normalization layer, specified by the `NormalizationLayer` name-value argument. • An activation layer specified by the `ActivationLayer` name-value argument.	
Final block	• A `convolution2dLayer` with a stride of [1 1] and a filter size of `FilterSizeInFirstAndLastBlocks`. • An optional activation layer specified by the `FinalActivationLayer` name-value argument.	

References

[1] Zhu, Jun-Yan, Taesung Park, Phillip Isola, and Alexei A. Efros. "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks." In *2017 IEEE International Conference on Computer Vision (ICCV)*, 2242–2251. Venice: IEEE, 2017. https://ieeexplore.ieee.org/document/8237506.

[2] Zhu, Jun-Yan, Taesung Park, and Tongzhou Wang. "CycleGAN and pix2pix in PyTorch." https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix.

See Also

`patchGANDiscriminator`

Topics

“Get Started with GANs for Image-to-Image Translation”
“Create Modular Neural Networks”
“List of Deep Learning Layers” (Deep Learning Toolbox)

Introduced in R2021a

dct2

2-D discrete cosine transform

Syntax

```
B = dct2(A)
B = dct2(A,m,n)
B = dct2(A,[m n])
```

Description

`B = dct2(A)` returns the two-dimensional discrete cosine transform of `A`. The matrix `B` contains the discrete cosine transform coefficients $B(k_1,k_2)$.

`B = dct2(A,m,n)` and

`B = dct2(A,[m n])` pad the matrix `A` with `0`s to size `m`-by-`n` before applying the transformation. If `m` or `n` is smaller than the corresponding dimension of `A`, then `dct2` crops `A` before the transformation.

Examples

Remove High Frequencies in Image using 2-D DCT

Read an image into the workspace, then convert the image to grayscale.

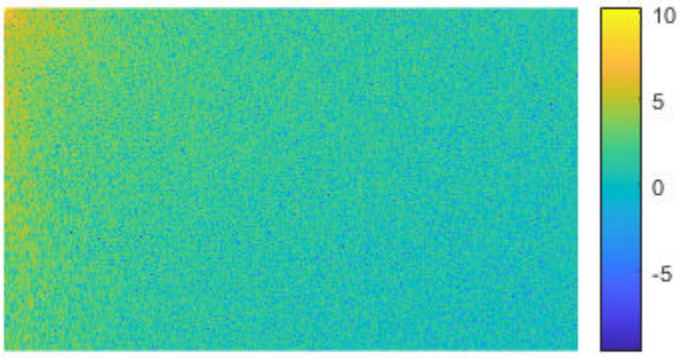
```
RGB = imread('autumn.tif');
I = im2gray(RGB);
```

Perform a 2-D DCT of the grayscale image using the `dct2` function.

```
J = dct2(I);
```

Display the transformed image using a logarithmic scale. Notice that most of the energy is in the upper left corner.

```
imshow(log(abs(J)),[])
colormap parula
colorbar
```



Set values less than magnitude 10 in the DCT matrix to zero.

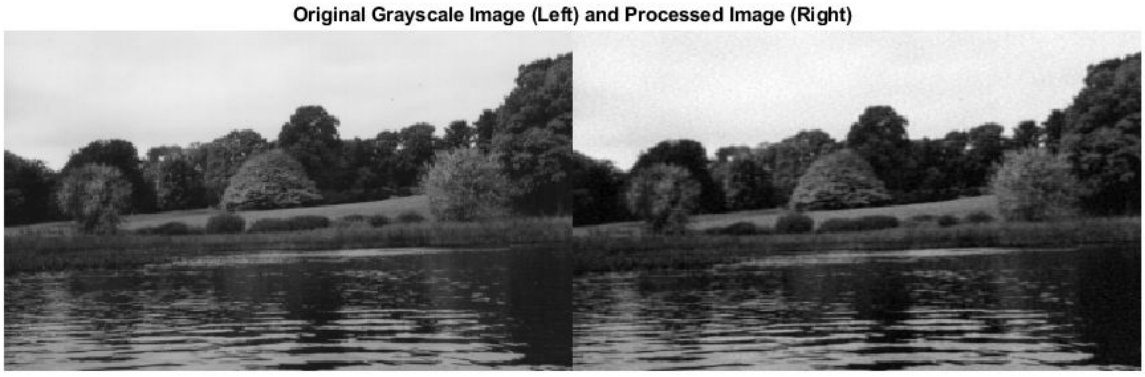
```
J(abs(J) < 10) = 0;
```

Reconstruct the image using the inverse DCT function `idct2`. Rescale the values to the range [0, 1] expected of images of data type `double`.

```
K = idct2(J);
K = rescale(K);
```

Display the original grayscale image alongside the processed image. The processed image has fewer high frequency details, such as in the texture of the trees.

```
montage({I,K})
title('Original Grayscale Image (Left) and Processed Image (Right)');
```



Input Arguments

A — Input matrix

2-D numeric matrix

Input matrix, specified as a 2-D numeric matrix.

m — Number of image rows

`size(A,1)` (default) | positive integer

Number of image rows, specified as a positive integer. `dct2` pads image `A` with `0`s or truncates image `A` so that it has `m` rows. By default, `m` is equal to `size(A,1)`.

n — Number of image columns

`size(A,2)` (default) | positive integer

Number of image columns, specified as a positive integer. `dct2` pads image `A` with `0`s or truncates image `A` so that it has `n` columns. By default, `n` is equal to `size(A,2)`

Output Arguments

B — Transformed matrix

`m`-by-`n` numeric matrix

Transformed matrix using a two-dimensional discrete cosine transform, returned as an `m`-by-`n` numeric matrix.

Data Types: `double`

More About

Discrete Cosine Transform

The discrete cosine transform (DCT) is closely related to the discrete Fourier transform. It is a separable linear transformation; that is, the two-dimensional transform is equivalent to a one-dimensional DCT performed along a single dimension followed by a one-dimensional DCT in the other dimension. The definition of the two-dimensional DCT for an input image `A` and output image `B` is

$$B_{pq} = \alpha_p \alpha_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \cos\frac{\pi(2m+1)p}{2M} \cos\frac{\pi(2n+1)q}{2N}, \quad \begin{matrix} 0 \le p \le M-1 \\ 0 \le q \le N-1 \end{matrix}$$

where

$$\alpha_p = \begin{cases} \frac{1}{\sqrt{M}}, & p = 0 \\ \sqrt{\frac{2}{M}}, & 1 \le p \le M-1 \end{cases}$$

and

$$\alpha_q = \begin{cases} \frac{1}{\sqrt{N}}, & q = 0 \\ \sqrt{\frac{2}{N}}, & 1 \le \text{q} \le \text{N-1} \end{cases}$$

M and *N* are the row and column size of `A`, respectively.

Tips

- If you apply the DCT to real data, the result is also real. The DCT tends to concentrate information, making it useful for image compression applications.
- To invert the DCT transformation, use `idct2`.

References

[1] Jain, Anil K., *Fundamentals of Digital Image Processing,* Englewood Cliffs, NJ, Prentice Hall, 1989, pp. 150–153.

[2] Pennebaker, William B., and Joan L. Mitchell, *JPEG: Still Image Data Compression Standard,* Van Nostrand Reinhold, 1993.

See Also

`fft2` | `idct2` | `ifft2`

Introduced before R2006a

dctmtx

Discrete cosine transform matrix

Syntax

```
D = dctmtx(n)
```

Description

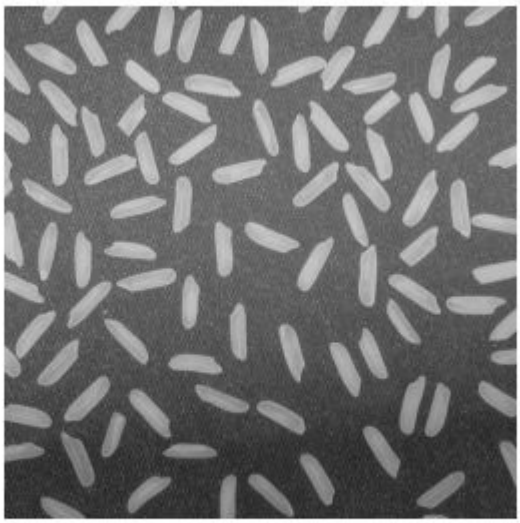
`D = dctmtx(n)` returns the `n`-by-`n` discrete cosine transform (DCT) matrix, which you can use to perform a 2-D DCT on an image.

Examples

Calculate Discrete Cosine Transform Matrix

Read an image into the workspace and cast it to class `double`.

```
A = im2double(imread('rice.png'));
imshow(A)
```

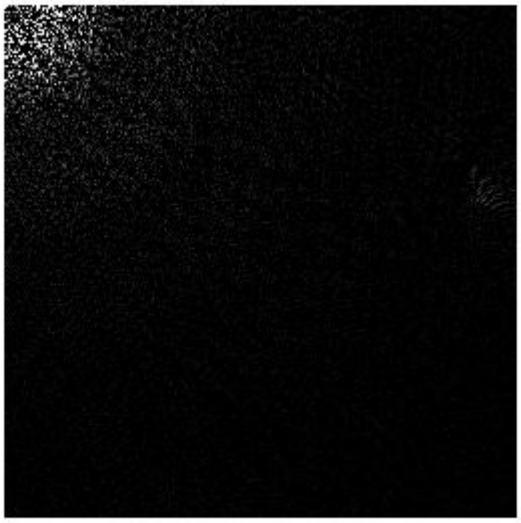


Calculate the discrete cosine transform matrix.

```
D = dctmtx(size(A,1));
```

Multiply the input image `A` by `D` to get the DCT of the columns of `A`, and by `D'` to get the inverse DCT of the columns of `A`.

```
dct = D*A*D';
imshow(dct)
```



Input Arguments

`n` — Size of DCT matrix

positive integer

Size of DCT matrix, specified as a positive integer.

Data Types: `double`

Output Arguments

`D` — DCT matrix

numeric matrix

DCT matrix, returned as a numeric matrix of size `n`-by-`n`.

Data Types: `double`

Tips

- If you have an `n`-by-`n` image, `A`, then `D*A` is the DCT of the columns of `A` and `D'*A` is the inverse DCT of the columns of `A`.

- The two-dimensional DCT of `A` can be computed as `D*A*D'`. This computation is sometimes faster than using `dct2`, especially if you are computing a large number of small DCTs, because `D` needs to be determined only once.

  For example, in JPEG compression, the DCT of each 8-by-8 block is computed. To perform this computation, use `dctmtx` to determine `D`, and then calculate each DCT using `D*A*D'` (where `A` is each 8-by-8 block). This is faster than calling `dct2` for each individual block.

See Also

`dct2`

Introduced before R2006a

decompose

Return sequence of decomposed structuring elements

Syntax

```
SEQ = decompose(SE)
```

Description

`SEQ = decompose(SE)` returns an array of structuring elements, `SEQ`, that are the decomposition of the structuring element `SE`. `SEQ` is equivalent to `SE`, but the elements of `SEQ` cannot be decomposed further.

Examples

View Decomposition of Structuring Element

Create a disk-shaped structuring element.

```
se = strel('square',5)
```

```
se =
strel is a square shaped structuring element with properties:

      Neighborhood: [5x5 logical]
    Dimensionality: 2
```

Extract the decomposition of the structuring element.

```
seq = decompose(se)
```

```
seq =
  2x1 strel array with properties:

    Neighborhood
    Dimensionality
```

To see that dilating sequentially with the decomposed structuring elements really does form a 5-by-5 square, use `imdilate` with the full option.

```
imdilate(1,seq,'full')
```

```
ans = 5×5

     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

Extract Decomposition of Structuring Element

Create a ball-shaped structuring element.

```
se = offsetstrel('ball',5, 6.5)
```

```
se =
offsetstrel is a ball shaped offset structuring element with properties:

            Offset: [11x11 double]
    Dimensionality: 2
```

Obtain the decomposition of the structuring element.

```
seq = decompose(se)
```

```
seq =
  1x8 offsetstrel array with properties:

    Offset
    Dimensionality
```

Input Arguments

SE — Structuring element

`strel` or `offsetstrel` object

Structuring element, specified as a `strel` or `offsetstrel` object.

Output Arguments

SEQ — Sequence of structuring elements

array of `strel` or `offsetstrel` objects

Sequence of structuring elements that approximate the desired shape, returned as an array of `strel` or `offsetstrel` objects.

See Also

Topics

“Structuring Elements”

Introduced before R2006a

deconvblind

Deblur image using blind deconvolution

Syntax

```
[J,psfr] = deconvblind(I,psfi)
[J,psfr] = deconvblind(I,psfi,iter)
[J,psfr] = deconvblind(I,psfi,iter,dampar)
[J,psfr] = deconvblind(I,psfi,iter,dampar,weight)
[J,psfr] = deconvblind(I,psfi,iter,dampar,weight,readout)
[J,psfr] = deconvblind(___,fun)
```

Description

`[J,psfr] = deconvblind(I,psfi)` deconvolves image `I` using the maximum likelihood algorithm and an initial estimate of the point-spread function (PSF), `psfi`. The `deconvblind` function returns both the deblurred image `J` and a restored PSF, `psfr`.

To improve the restoration, `deconvblind` supports several optional parameters, described below. Use `[]` as a placeholder if you do not specify an intermediate parameter.

`[J,psfr] = deconvblind(I,psfi,iter)` specifies the number of iterations, `iter`.

`[J,psfr] = deconvblind(I,psfi,iter,dampar)` controls noise amplification by suppressing iterations for pixels that deviate a small amount compared to the noise, specified by the damping threshold `dampar`. By default, no damping occurs.

`[J,psfr] = deconvblind(I,psfi,iter,dampar,weight)` specifies which pixels in the input image `I` are considered in the restoration. The value of an element in the `weight` array determines how much the pixel at the corresponding position in the input image is considered. For example, to exclude a pixel from consideration, assign it a value of `0` in the `weight` array. You can adjust the weight value assigned to each pixel according to the amount of flat-field correction.

`[J,psfr] = deconvblind(I,psfi,iter,dampar,weight,readout)` specifies the additive noise (such as background and foreground noise) and the variance of the read-out camera noise, `readout`.

`[J,psfr] = deconvblind(___,fun)`, where `fun` is a handle to a function that describes additional constraints on the PSF. `fun` is called at the end of each iteration. For more information about function handles, see “Create Function Handle”.

Examples

Deblur an Image Using Blind Deconvolution

Create a sample image with noise.

```
% Set the random number generator back to its default settings for
% consistency in results.
```

```
rng default;

I = checkerboard(8);
PSF = fspecial('gaussian',7,10);
V = .0001;
BlurredNoisy = imnoise(imfilter(I,PSF),'gaussian',0,V);
```

Create a weight array to specify which pixels are included in processing.

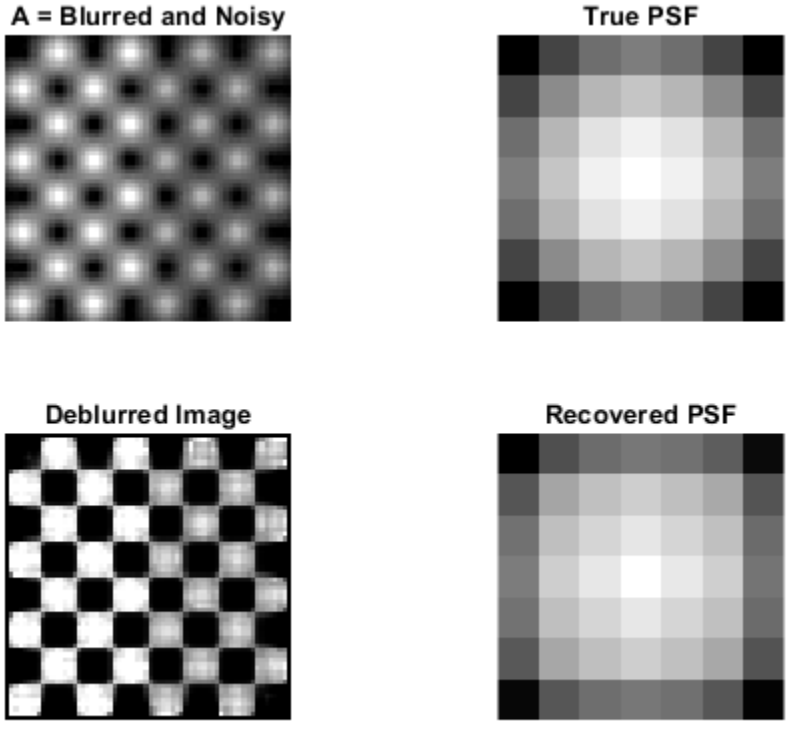
```
WT = zeros(size(I));
WT(5:end-4,5:end-4) = 1;
INITPSF = ones(size(PSF));
```

Perform blind deconvolution.

```
[J P] = deconvblind(BlurredNoisy,INITPSF,20,10*sqrt(V),WT);
```

Display the results.

```
subplot(221);imshow(BlurredNoisy);
title('A = Blurred and Noisy');
subplot(222);imshow(PSF,[]);
title('True PSF');
subplot(223);imshow(J);
title('Deblurred Image');
subplot(224);imshow(P,[]);
title('Recovered PSF');
```



Input Arguments

`I` — Blurry image

numeric array | cell array

Blurry image, specified as a numeric array of any dimension. You can also specify the image as a cell array to enable interrupted iterations. For more information, see “Tips” on page 1-626.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`psfi` — Initial estimate of PSF

numeric array

Initial estimate of PSF, specified as a numeric array. The PSF restoration is affected strongly by the size of the initial guess `psfi` and less by the values it contains. For this reason, specify an array of `1`s as your `psfi`.

You can also specify `psfi` as a cell array to enable interrupted iterations. For more information, see “Tips” on page 1-626.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`iter` — Number of iterations

`10` (default) | positive integer

Number of iterations, specified as a positive integer.

Data Types: `double`

`dampar` — Threshold for damping

`0` (default) | numeric scalar

Threshold for damping, specified as a numeric scalar. Damping occurs for pixels whose deviation between iterations is less than the threshold. `dampar` has the same data type as `I`.

`weight` — Weight of each pixel

numeric array

Weight value of each pixel, specified as a numeric array with values in the range [0, 1]. `weight` has the same size as the input image, `I`. By default, all elements in `weight` have the value `1`, so all pixels are considered equally in the restoration.

Data Types: `double`

`readout` — Noise

`0` (default) | numeric scalar | numeric array

Noise, specified as a numeric scalar or numeric array. The value of `readout` corresponds to the additive noise (such as noise from the foreground and background) and the variance of the read-out camera noise. `readout` has the same data type as `I`.

`fun` — Function handle

handle

Function handle, specified as a handle. `fun` must accept the PSF as its first argument. The function must return one argument: a PSF that is the same size as the original PSF and that satisfies the positivity and normalization constraints.

Output Arguments

J — Deblurred image

numeric array | 1-by-4 cell array

Deblurred image, returned as a numeric array or a 1-by-4 cell array. `J` (or `J{1}` when `J` is a cell array) has the same data type as `I`. For more information about returning `J` as a cell array for interrupted iterations, see “Tips” on page 1-626.

psfr — Restored PSF

array of positive numbers | 1-by-4 cell array

Restored PSF, returned as an array of positive numbers or a 1-by-4 cell array. `psfr` has the same size as the initial estimate of the PSF, `psfi`, and it is normalized so the sum of elements is 1. For more information about returning `psfr` as a cell array for interrupted iterations, see “Tips” on page 1-626.

Data Types: `double`

Tips

- You can use `deconvblind` to perform a deconvolution that starts where a previous deconvolution stopped. To use this feature, pass the input image `I` and the initial guess at the PSF, `psfi`, as cell arrays: `{I}` and `{psfi}`. When you do, the `deconvblind` function returns the output image `J` and the restored point-spread function, `psfr`, as cell arrays, which can then be passed as the input arrays into the next `deconvblind` call. The output cell array `J` contains four elements:

  `J{1}` contains `I`, the original image.

  `J{2}` contains the result of the last iteration.

  `J{3}` contains the result of the next-to-last iteration.

  `J{4}` is an array generated by the iterative algorithm.
- The output image `J` could exhibit ringing introduced by the discrete Fourier transform used in the algorithm. To reduce the ringing, use `I = edgetaper(I,psfi)` before calling `deconvblind`.

References

[1] D.S.C. Biggs and M. Andrews, *Acceleration of iterative image restoration algorithms*, Applied Optics, Vol. 36, No. 8, 1997.

[2] R.J. Hanisch, R.L. White, and R.L. Gilliland, *Deconvolutions of Hubble Space Telescope Images and Spectra*, Deconvolution of Images and Spectra, Ed. P.A. Jansson, 2nd ed., Academic Press, CA, 1997.

[3] Timothy J. Holmes, et al, *Light Microscopic Images Reconstructed by Maximum Likelihood Deconvolution*, Handbook of Biological Confocal Microscopy, Ed. James B. Pawley, Plenum Press, New York, 1995.

See Also

`deconvlucy` | `deconvreg` | `deconvwnr` | `edgetaper` | `imnoise` | `otf2psf` | `padarray` | `psf2otf`

Topics
“Deblurring Images Using the Blind Deconvolution Algorithm”
“Image Deblurring”
“Adapt Blind Deconvolution for Various Image Distortions”

Introduced before R2006a

deconvlucy

Deblur image using Lucy-Richardson method

Syntax

```
J = deconvlucy(I,psf)
J = deconvlucy(I,psf,iter)
J = deconvlucy(I,psf,iter,dampar)
J = deconvlucy(I,psf,iter,dampar,weight)
J = deconvlucy(I,psf,iter,dampar,weight,readout)
J = deconvlucy(I,psf,iter,dampar,weight,readout,subsample)
```

Description

`J = deconvlucy(I,psf)` restores image `I` that was degraded by convolution with a point-spread function (PSF), `psf`, and possibly by additive noise. The algorithm is based on maximizing the likelihood that the resulting image `J` is an instance of the original image `I` under Poisson statistics.

To improve the restoration, `deconvlucy` supports several optional parameters, described below. Use `[]` as a placeholder if you do not specify an intermediate parameter.

`J = deconvlucy(I,psf,iter)` specifies the number of iterations, `iter`.

`J = deconvlucy(I,psf,iter,dampar)` controls noise amplification by suppressing iterations for pixels that deviate a small amount compared to the noise, specified by the damping threshold `dampar`. By default, no damping occurs.

`J = deconvlucy(I,psf,iter,dampar,weight)` specifies which pixels in the input image `I` are considered in the restoration. The value of an element in the `weight` array determines how much the pixel at the corresponding position in the input image is considered. For example, to exclude a pixel from consideration, assign it a value of `0` in the `weight` array. You can adjust the weight value assigned to each pixel according to the amount of flat-field correction.

`J = deconvlucy(I,psf,iter,dampar,weight,readout)` specifies the additive noise (such as background or foreground noise) and variance of the read-out camera noise, `readout`.

`J = deconvlucy(I,psf,iter,dampar,weight,readout,subsample)` uses subsampling when the PSF is given on a grid that is `subsample` times finer than the image.

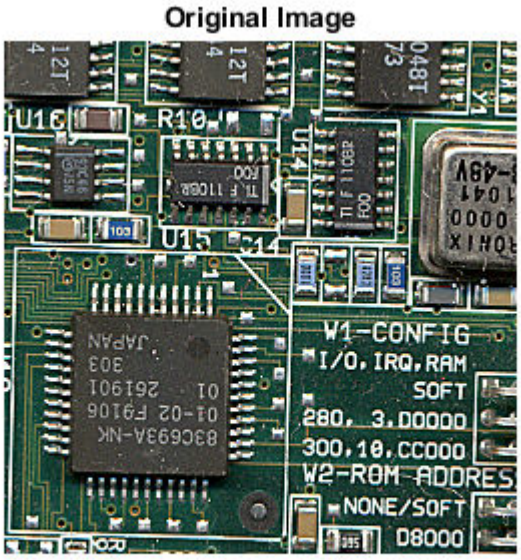
Examples

Remove Gaussian Blur Using deconvlucy

Read and display a pristine image that does not have blur or noise. This example optionally crops the image to a size of 256-by-256 with the top-left (x,y) coordinate at (2,50).

```
I = imread('board.tif');
I = imcrop(I,[2 50 255 255]);
```

```
imshow(I)
title('Original Image')
```



Create a PSF that represents a Gaussian blur with standard deviation 5 and filter of size 5-by-5.

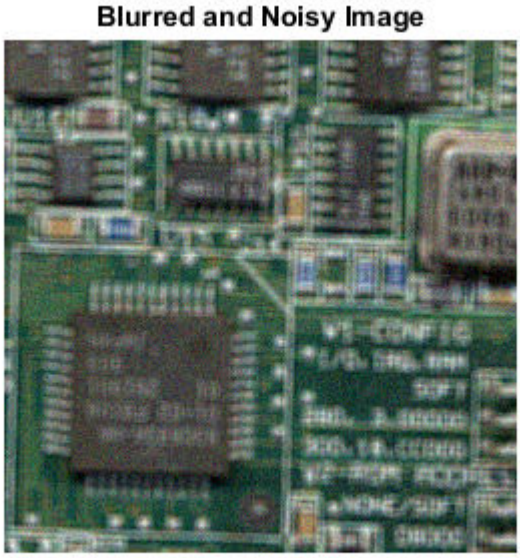
```
PSF = fspecial('gaussian',5,5);
```

Simulate blur in the image.

```
blurred = imfilter(I,PSF,'symmetric','conv');
```

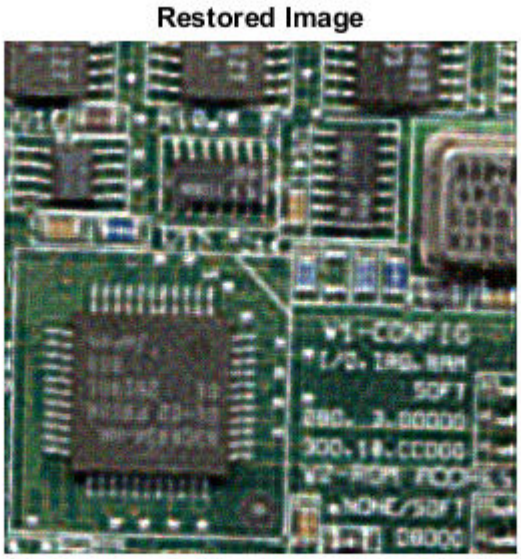
Add simulated zero-mean Gaussian noise.

```
V = 0.002;
blurred_noisy = imnoise(blurred,'gaussian',0,V);
imshow(blurred_noisy)
title('Blurred and Noisy Image')
```



Use `deconvlucy` to restore the blurred and noisy image. Specify the PSF used to create the blur and decrease the number of iterations to 5.

```
luc1 = deconvlucy(blurred_noisy,PSF,5);
imshow(luc1)
title('Restored Image')
```



Remove Blur Using Several `deconvlucy` Optional Syntaxes

Create a sample image and blur it.

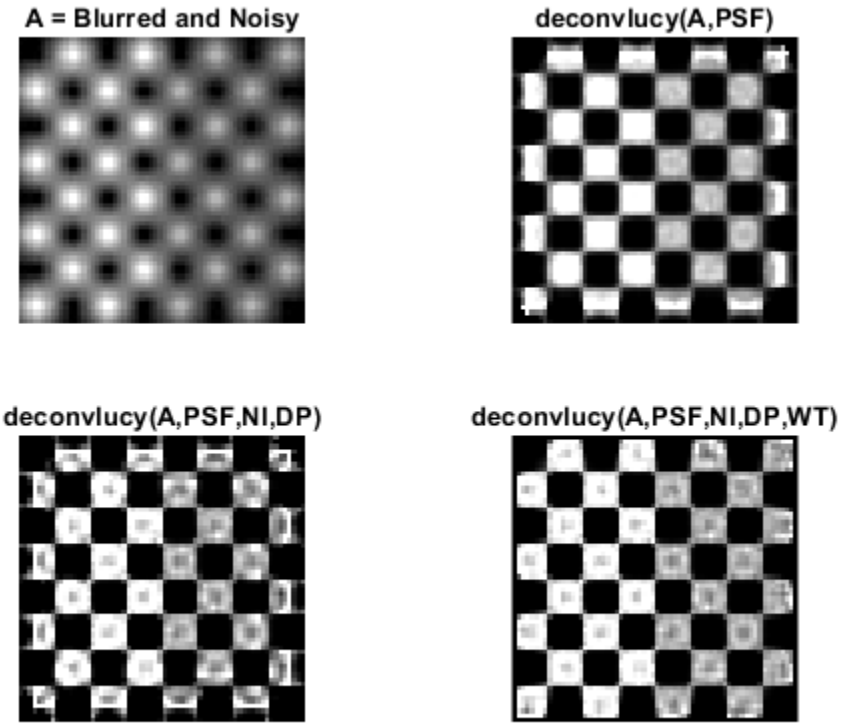
```
I = checkerboard(8);
PSF = fspecial('gaussian',7,10);
V = .0001;
BlurredNoisy = imnoise(imfilter(I,PSF),'gaussian',0,V);
```

Create a weight array and call `deconvlucy` using several optional parameters.

```
WT = zeros(size(I));
WT(5:end-4,5:end-4) = 1;
J1 = deconvlucy(BlurredNoisy,PSF);
J2 = deconvlucy(BlurredNoisy,PSF,20,sqrt(V));
J3 = deconvlucy(BlurredNoisy,PSF,20,sqrt(V),WT);
```

Display the results.

```
subplot(221);imshow(BlurredNoisy);
title('A = Blurred and Noisy');
subplot(222);imshow(J1);
title('deconvlucy(A,PSF)');
subplot(223);imshow(J2);
title('deconvlucy(A,PSF,NI,DP)');
subplot(224);imshow(J3);
title('deconvlucy(A,PSF,NI,DP,WT)');
```



Input Arguments

`I` — Blurry image

numeric array | cell array

Blurry image, specified as a numeric array of any dimension. You can also specify the image as a cell array to enable interrupted iterations. For more information, see “Tips” on page 1-633.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`psf` — PSF

numeric array

PSF, specified as a numeric array.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`iter` — Number of iterations

`10` (default) | positive integer

Number of iterations, specified as a positive integer.

Data Types: `double`

`dampar` — Threshold for damping

`0` (default) | numeric scalar

Threshold for damping, specified as a numeric scalar. Damping occurs for pixels whose deviation between iterations is less than the threshold. `dampar` has the same data type as `I`.

`weight` — Weight of each pixel

numeric array

Weight value of each pixel, specified as a numeric array with values in the range [0, 1]. `weight` has the same size as the input image, `I`. By default, all elements in `weight` have the value `1`, so all pixels are considered equally in the restoration.

Data Types: `double`

`readout` — Noise

`0` (default) | numeric scalar | numeric array

Noise, specified as a numeric scalar or numeric array. The value of `readout` corresponds to the additive noise (such as noise from the foreground and background) and the variance of the read-out camera noise. `readout` has the same data type as `I`.

`subsample` — Subsampling

`1` (default) | positive scalar

Subsampling, specified as a positive scalar.

Data Types: `double`

Output Arguments

J — Deblurred image

numeric array | 1-by-4 cell array

Deblurred image, returned as a numeric array or a 1-by-4 cell array. `J` (or `J{1}` when `J` is a cell array) has the same data type as `I`. For more information about returning `J` as a cell array for interrupted iterations, see “Tips” on page 1-633.

Tips

- You can use `deconvlucy` to perform a deconvolution that starts where a previous deconvolution stopped. To use this feature, pass the input image `I` as a cell array, `{I}`. When you do, the `deconvlucy` function returns the output image `J` as a cell array, which you can then pass as the input array into the next `deconvlucy` call. The output cell array `J` contains four elements:

  `J{1}` contains `I`, the original image.

  `J{2}` contains the result of the last iteration.

  `J{3}` contains the result of the next-to-last iteration.

  `J{4}` is an array generated by the iterative algorithm.
- The output image `J` could exhibit ringing introduced by the discrete Fourier transform used in the algorithm. To reduce the ringing, use `I = edgetaper(I,psf)` before calling `deconvlucy`.
- `deconvlucy` converts the PSF to `double` without normalization.
- `deconvlucy` may return values in the output image that are beyond the range of the input image.

References

[1] D.S.C. Biggs and M. Andrews, *Acceleration of iterative image restoration algorithms*, Applied Optics, Vol. 36, No. 8, 1997.

[2] R.J. Hanisch, R.L. White, and R.L. Gilliland, *Deconvolutions of Hubble Space Telescope Images and Spectra*, Deconvolution of Images and Spectra, Ed. P.A. Jansson, 2nd ed., Academic Press, CA, 1997.

See Also

`deconvblind` | `deconvreg` | `deconvwnr` | `edgetaper` | `otf2psf` | `padarray` | `psf2otf`

Topics

“Deblurring Images Using the Lucy-Richardson Algorithm”
“Image Deblurring”
“Adapt the Lucy-Richardson Deconvolution for Various Image Distortions”

Introduced before R2006a

deconvreg

Deblur image using regularized filter

Syntax

```
J = deconvreg(I,psf)
J = deconvreg(I,psf,np)
J = deconvreg(I,psf,np,lrange)
J = deconvreg(I,psf,np,lrange,regop)
[J,lagra] = deconvreg( ___ )
```

Description

`J = deconvreg(I,psf)` deconvolves image `I` using the regularized filter algorithm, returning deblurred image `J`. The assumption is that the image `I` was created by convolving a true image with a point-spread function (PSF), `psf`, and possibly by adding noise. The algorithm is a constrained optimum in the sense of least square error between the estimated and the true images under requirement of preserving image smoothness.

`J = deconvreg(I,psf,np)` specifies the additive noise power, `np`.

`J = deconvreg(I,psf,np,lrange)` specifies the range, `lrange`, where the search for the optimal solution is performed. The algorithm finds an optimal Lagrange multiplier `lagra` within the `lrange` range.

`J = deconvreg(I,psf,np,lrange,regop)` constrains the deconvolution using regularization operator `regop`. The default regularization operator is the Laplacian operator, to retain the image smoothness.

`[J,lagra] = deconvreg( ___ )` outputs the value of the Lagrange multiplier, `lagra` in addition to the restored image, `J`.

Examples

Deblur Image Using Regularized Filter

Create sample image.

```
I = checkerboard(8);
```

Create PSF and use it to create a blurred and noisy version of the input image.

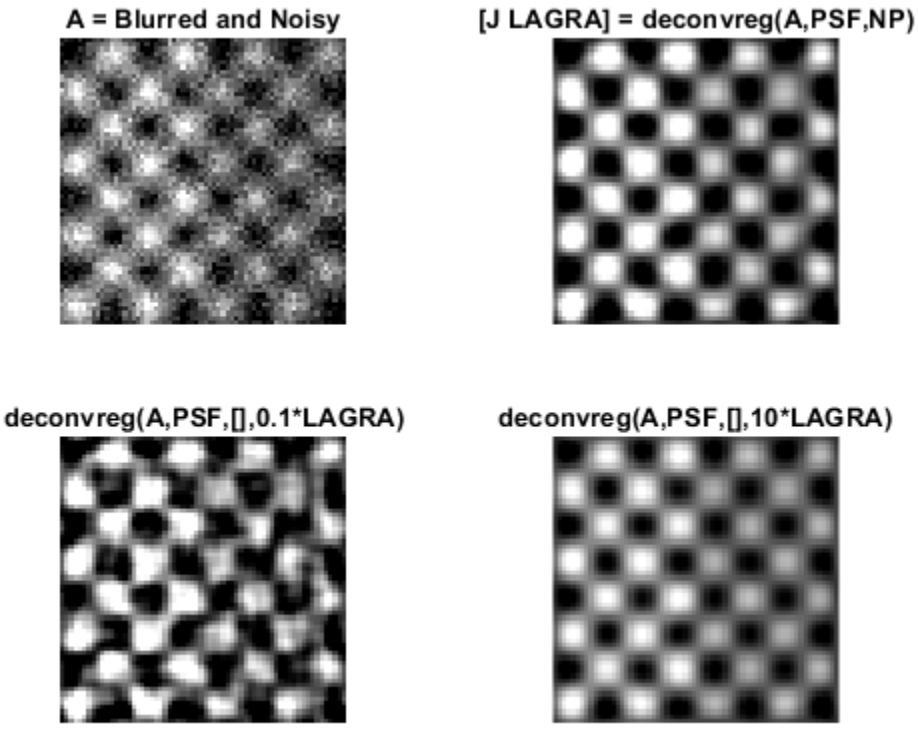
```
PSF = fspecial('gaussian',7,10);
V = .01;
BlurredNoisy = imnoise(imfilter(I,PSF),'gaussian',0,V);
NOISEPOWER = V*prod(size(I));
```

Deblur the image.

```
[J LAGRA] = deconvreg(BlurredNoisy,PSF,NOISEPOWER);
```

Display the various versions of the image.

```
subplot(221); imshow(BlurredNoisy);
title('A = Blurred and Noisy');
subplot(222); imshow(J);
title('[J LAGRA] = deconvreg(A,PSF,NP)');
subplot(223); imshow(deconvreg(BlurredNoisy,PSF,[],LAGRA/10));
title('deconvreg(A,PSF,[],0.1*LAGRA)');
subplot(224); imshow(deconvreg(BlurredNoisy,PSF,[],LAGRA*10));
title('deconvreg(A,PSF,[],10*LAGRA)');
```



Input Arguments

`I` — Blurry image

numeric array

Blurry image, specified as a numeric array of any dimension.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`psf` — PSF

numeric array

PSF, specified as a numeric array.

Data Types: `double`

np — Noise power
0 (default) | numeric scalar

Noise power, specified as a numeric scalar.

Data Types: double

lrange — Search range
[1e-9 1e9] (default) | numeric scalar | 2-element numeric vector

Search range, specified as a numeric scalar or a 2-element numeric vector. If lrange is a scalar, then the algorithm assumes that lagra is equal to lrange. If you specify lagra, then the function ignores the np value

Data Types: double

regop — Regularization operator
numeric array

Regularization operator, specified as a numeric array. The regop array dimensions must not exceed the dimensions of the image, I. Any nonsingleton dimensions must correspond to the nonsingleton dimensions of psf.

Data Types: double

Output Arguments

J — Deblurred image
numeric array

Deblurred image, returned as a numeric array. J has the same data type as I.

lagra — Lagrange multiplier
numeric scalar

Lagrange multiplier, returned as a numeric scalar.

Tips

- The output image J could exhibit ringing introduced by the discrete Fourier transform used in the algorithm. To reduce the ringing, use I = edgetaper(I,psf) before calling deconvreg.

References

[1] Gonzalez, R. C., and R. E. Woods. *Digital Image Processing*. Addison-Wesley Publishing Company, Inc., 1992.

See Also

deconvblind | deconvlucy | deconvwnr | edgetaper | otf2psf | padarray | psf2otf

Topics
“Deblur Images Using Regularized Filter”
“Image Deblurring”

Introduced before R2006a

deconvwnr

Deblur image using Wiener filter

Syntax

```
J = deconvwnr(I,psf,nsr)
J = deconvwnr(I,psf,ncorr,icorr)
J = deconvwnr(I,psf)
```

Description

`J = deconvwnr(I,psf,nsr)` deconvolves image `I` using the Wiener filter algorithm, returning deblurred image `J`. `psf` is the point-spread function (PSF) with which `I` was convolved. `nsr` is the noise-to-signal power ratio of the additive noise. The algorithm is optimal in a sense of least mean square error between the estimated and the true images.

`J = deconvwnr(I,psf,ncorr,icorr)` deconvolves image `I`, where `ncorr` is the autocorrelation function of the noise and `icorr` is the autocorrelation function of the original image.

`J = deconvwnr(I,psf)` deconvolves image `I` using the Wiener filter algorithm with no estimated noise. In the absence of noise, a Wiener filter is equivalent to an ideal inverse filter.

Examples

Deblur Image Using Wiener Filter

Read image into the workspace and display it.

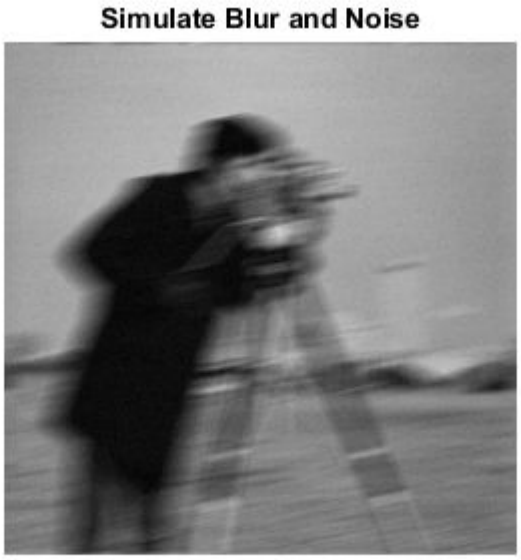
```
I = im2double(imread('cameraman.tif'));
imshow(I);
title('Original Image (courtesy of MIT)');
```

Simulate a motion blur.

```
LEN = 21;
THETA = 11;
PSF = fspecial('motion', LEN, THETA);
blurred = imfilter(I, PSF, 'conv', 'circular');
figure, imshow(blurred)
```

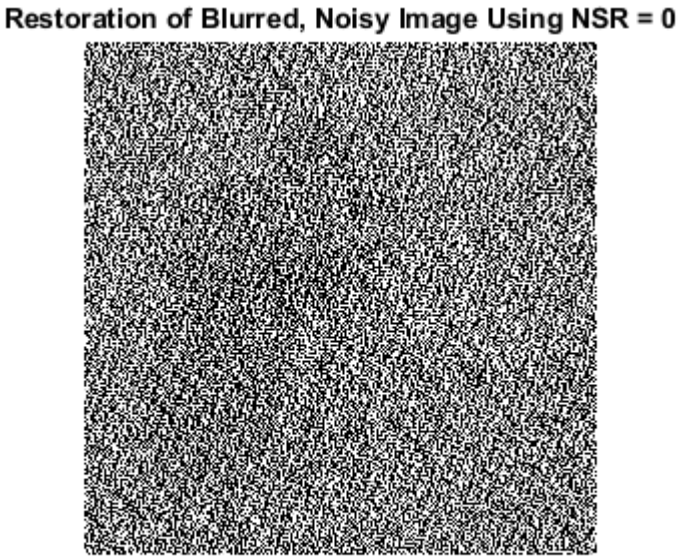
Simulate additive noise.

```
noise_mean = 0;
noise_var = 0.0001;
blurred_noisy = imnoise(blurred, 'gaussian', ...
                        noise_mean, noise_var);
figure, imshow(blurred_noisy)
title('Simulate Blur and Noise')
```



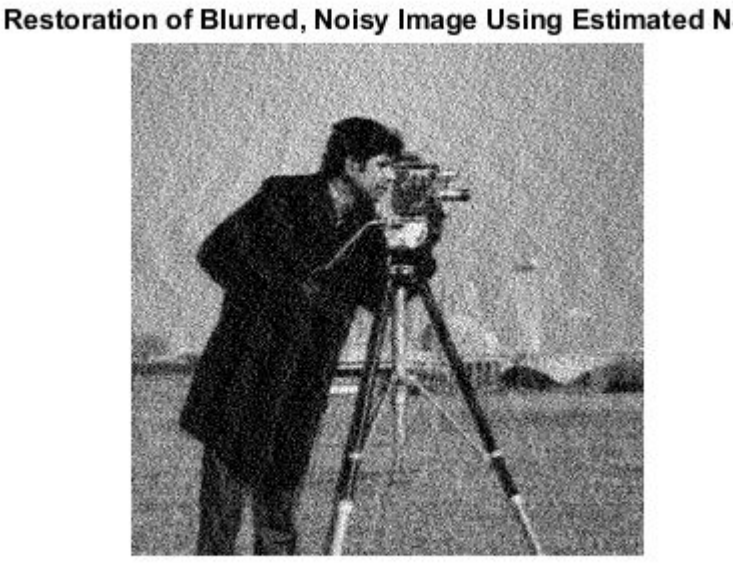
Try restoration assuming no noise.

```
estimated_nsr = 0;
wnr2 = deconvwnr(blurred_noisy, PSF, estimated_nsr);
figure, imshow(wnr2)
title('Restoration of Blurred, Noisy Image Using NSR = 0')
```



Try restoration using a better estimate of the noise-to-signal-power ratio.

```
estimated_nsr = noise_var / var(I(:));
wnr3 = deconvwnr(blurred_noisy, PSF, estimated_nsr);
figure, imshow(wnr3)
title('Restoration of Blurred, Noisy Image Using Estimated NSR');
```



Input Arguments

`I` — Blurry image

numeric array

Blurry image, specified as a numeric array of any dimension.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`psf` — Point-spread function

numeric array

Point-spread function, specified as a numeric array.

Data Types: `double`

`nsr` — Noise-to-signal ratio

`0` | nonnegative scalar

Noise-to-signal ratio, specified as a nonnegative scalar or numeric array of the same size as the image, `I`. If `nsr` is an array, then it represents the spectral domain. Specifying `0` for the `nsr` is equivalent to creating an ideal inverse filter.

Data Types: `double`

`ncorr` — Autocorrelation function of the noise

numeric array

Autocorrelation function of the noise, specified as a numeric array of any size or dimension, not exceeding the original image.

- If the dimensionality of `ncorr` matches the dimensionality of the image `I`, then the values correspond to the autocorrelation within each dimension.
- If `ncorr` is a vector and `psf` is also a vector, then the values in `ncorr` represent the autocorrelation function in the first dimension.
- If `ncorr` is a vector and `psf` is an array, then the 1-D autocorrelation function is extrapolated by symmetry to all non-singleton dimensions of `psf`.
- If `ncorr` is a scalar, then the value represents the power of the image noise.

Data Types: `double`

`icorr` — Autocorrelation function of the image

numeric array

Autocorrelation function of the image, specified as a numeric array of any size or dimension, not exceeding the original image.

- If the dimensionality of `icorr` matches the dimensionality of the image `I`, then the values correspond to the autocorrelation within each dimension.
- If `icorr` is a vector and `psf` is also a vector, then the values in `icorr` represent the autocorrelation function in the first dimension.
- If `icorr` is a vector and `psf` is an array, then the 1-D autocorrelation function is extrapolated by symmetry to all non-singleton dimensions of `psf`.

- If `icorr` is a scalar, then the value represents the power of the image noise.

Data Types: `double`

Output Arguments

`J` — Deblurred image

numeric array

Deblurred image, returned as a numeric array. `J` has the same data type as `I`.

Tips

- The output image `J` could exhibit ringing introduced by the discrete Fourier transform used in the algorithm. To reduce the ringing, use `I = edgetaper(I,psf)` before calling `deconvwnr`.

References

[1] Gonzalez, R. C., and R. E. Woods. *Digital Image Processing*. Addison-Wesley Publishing Company, Inc., 1992.

See Also

`deconvblind` | `deconvlucy` | `deconvreg` | `edgetaper` | `otf2psf` | `padarray` | `psf2otf`

Topics

“Deblur Images Using a Wiener Filter”
“Image Deblurring”

Introduced before R2006a

decorrstretch

Apply decorrelation stretch to multichannel image

Syntax

```
S = decorrstretch(A)
S = decorrstretch(A,Name,Value)
```

Description

`S = decorrstretch(A)` applies a decorrelation stretch to RGB or multispectral image `A` and returns the result in `S`. The mean and variance in each band of `S` are the same as in `A`.

The primary purpose of decorrelation stretch is visual enhancement. Decorrelation stretching is a way to enhance the color differences in an image.

`S = decorrstretch(A,Name,Value)` uses name-value pairs to control aspects of the decorrelation stretch, such as the target mean and standard deviation of each band.

Examples

Highlight Color Differences in Forest Scene

This example shows how to use decorrelation stretching to highlight elements in a forest image by exaggerating the color differences.

Read an image into the workspace.

```
[X, map] = imread('forest.tif');
```

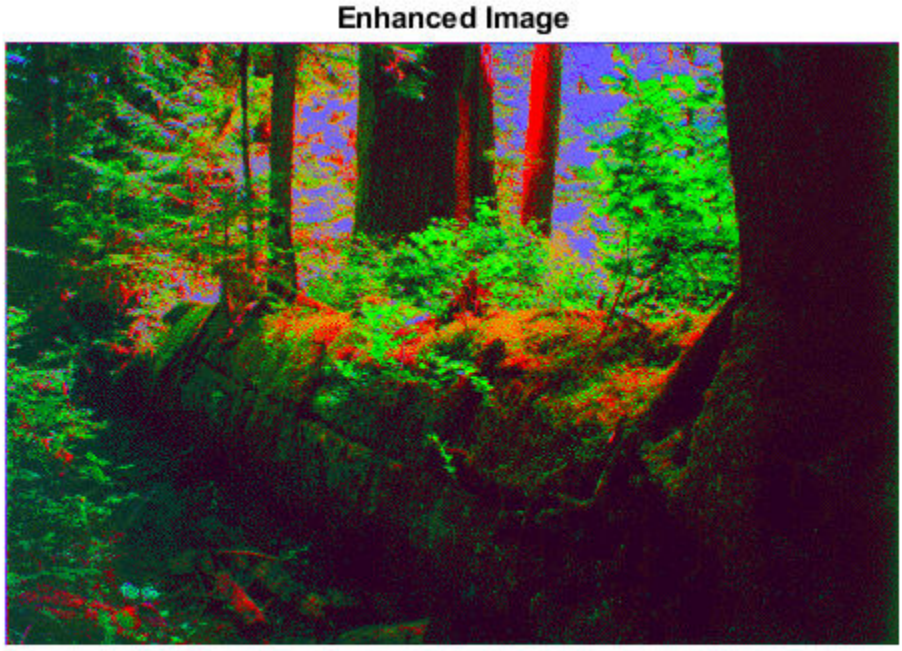
Apply decorrelation stretching using `decorrstretch`.

```
S = decorrstretch(ind2rgb(X,map),'tol',0.01);
```

Display the original image and the enhanced image.

```
figure
imshow(X,map)
title('Original Image')
```


```
figure
imshow(S)
title('Enhanced Image')
```



Input Arguments

A — Image to be enhanced

RGB image | multispectral image

Image to be enhanced, specified as an RGB image or multispectral image of size *m*-by-*n*-by-`nBands`. For an RGB image, `nBands` = 3.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Mode','covariance'`

Mode — Decorrelation method

`'correlation'` (default) | `'covariance'`

Decorrelation method, specified as the comma-separated pair consisting of `'Mode'` and of the following values.

- `'correlation'` — Uses the eigen decomposition of the band-to-band correlation matrix.

- `'covariance'` — Uses the eigen decomposition of the band-to-band covariance matrix.

Data Types: `char` | `string`

`TargetMean` — Target mean values
real scalar | vector of length `nBands`

Target mean values of the output bands, specified as the comma-separated pair consisting of `'TargetMean'` and a real scalar or vector of length `nBands`. By default, `TargetMean` is an 1-by-`nBands` vector containing the sample mean of each band, which preserves the band-wise means before and after the decorrelation stretch.

`TargetMean` must be of class `double`, but uses the same values as the pixels in the input image. For example, if `A` is class `uint8`, then `127.5` would be a reasonable value. If values need to be clamped to the standard range of the input/output image class, it can impact the results.

Data Types: `double`

`TargetSigma` — Target standard deviation values
positive scalar | vector of length `nBands`

Target standard deviation values of the output bands, specified as the comma-separated pair consisting of `'TargetSigma'` and a positive scalar or vector of length `nBands`. By default, `TargetSigma` is an 1-by-`nBands` vector containing the sample standard deviation of each band, which preserves the band-wise variance before and after the decorrelation stretch. The target standard deviation is ignored for uniform (zero-variance) bands.

`TargetSigma` must be class `double`, but uses the same values as the pixels in the input image. For example, if `A` is of class `uint8`, then `50.0` would be a reasonable value.

Data Types: `double`

`Tol` — Linear contrast stretch
numeric scalar | 2-element numeric vector

Linear contrast stretch following the decorrelation stretch, specified as the comma-separated pair consisting of `'Tol'` and a numeric scalar or 2-element numeric vector of class `double`. Specifying a value of `Tol` overrides the value of `TargetMean` or `TargetSigma`. If you do not specify `Tol`, then by default `decorrstretch` does not perform linear contrast stretch.

`Tol` has the same meaning as in `stretchlim`, where `Tol = [LOW_FRACT HIGH_FRACT]` specifies the fraction of the image to saturate at low and high intensities. If you specify `Tol` as a scalar value, then `LOW_FRACT = Tol` and `HIGH_FRACT = 1 - Tol`, saturating equal fractions at low and high intensities.

Small adjustments to `Tol` can strongly affect the visual appearance of the output.

Data Types: `double`

`SampleSubs` — Subset of `A` used to compute the band-means, covariance, and correlation
cell array containing two arrays of pixel subscripts `{rowsubs, colsubs}`

Subset of `A` used to compute the band-means, covariance, and correlation, specified as a cell array containing two arrays of pixel subscripts `{rowsubs, colsubs}`. `rowsubs` and `colsubs` are vectors or matrices of matching size that contain row and column subscripts, respectively.

Use this option to reduce the amount of computation, to keep invalid or non-representative pixels from affecting the transformation, or both. For example, you can use `rowsubs` and `colsubs` to exclude areas of cloud cover. If not specified, `decorrstretch` uses all the pixels in `A`.

Data Types: `double`

Output Arguments

S — Decorrelation stretched image

numeric array

Decorrelation stretched image, returned as a numeric array of the same size and class as the input image, `A`.

Tips

- The results of a straight decorrelation (without the contrast stretch option) may include values that fall outside the numerical range supported by the class `uint8` or `uint16` (negative values, or values exceeding `255` or `65535`, respectively). In these cases, `decorrstretch` clamps its output to the supported range.
- For class `double`, `decorrstretch` clamps the output only when you provide a value for `Tol`, specifying a linear contrast stretch followed by clamping to the interval `[0 1]`.
- The optional parameters do not interact, except that a linear stretch usually alters both the band-wise means and band-wise standard deviations. Thus, while you can specify `TargetMean` and `TargetSigma` along with `Tol`, their effects will be modified.

Algorithms

A decorrelation stretch is a linear, pixel-wise operation in which the specific parameters depend on the values of actual and desired (target) image statistics. The vector `a` containing the value of a given pixel in each band of the input image `A` is transformed into the corresponding pixel `b` in output image `B` as follows:

`b = T * (a - m) + m_target`.

`a` and `b` are `nBands`-by-1 vectors, `T` is an `nBands`-by-`nBands` matrix, and `m` and `m_target` are `nBands`-by-1 vectors such that

- `m` contains the mean of each band in the image, or in a subset of image pixels that you specify
- `m_target` contains the desired output mean in each band. The default choice is `m_target = m`.

The linear transformation matrix `T` depends on the following:

- The band-to-band sample covariance of the image, or of a subset of the image that you specify (the same subset as used for `m`), represented by matrix `Cov`
- A desired output standard deviation in each band. This is conveniently represented by a diagonal matrix, `SIGMA_target`. The default choice is `SIGMA_target = SIGMA`, where `SIGMA` is the diagonal matrix containing the sample standard deviation of each band. `SIGMA` should be computed from the same pixels that were used for `m` and `Cov`, which means simply that:

  `SIGMA(k,k) = sqrt(Cov(k,k), k = 1,..., nBands)`.

`Cov`, `SIGMA`, and `SIGMA_target` are `nBands`-by-`nBands`, as are the matrices `Corr`, `LAMBDA`, and `V`, defined below.

The first step in computing `T` is to perform an eigen-decomposition of either the covariance matrix `Cov` or the correlation matrix

`Corr = inv(SIGMA) * Cov * inv(SIGMA)`.

- In the correlation-based method, `Corr` is decomposed: `Corr = V LAMBDA V'`.
- In the covariance-based method, `Cov` is decomposed: `Cov = V LAMBDA V'`.

`LAMBDA` is a diagonal matrix of eigenvalues and `V` is the orthogonal matrix that transforms either `Corr` or `Cov` to `LAMBDA`.

The next step is to compute a stretch factor for each band, which is the inverse square root of the corresponding eigenvalue. It is convenient to define a diagonal matrix `S` containing the stretch factors, such that:

`S(k,k) = 1 / sqrt(LAMBDA(k,k))`.

Finally, matrix `T` is computed from either

`T = SIGMA_target V S V' inv(SIGMA)` (correlation-based method)

or

`T = SIGMA_target V S V'` (covariance-based method).

The two methods yield identical results if the band variances are uniform.

Substituting `T` into the expression for `b`:

```
b = m_target + SIGMA_target V S V' inv(SIGMA) * (a - m)
```

or

```
b = m_target + SIGMA_target V S V' * (a - m)
```

and reading from right to left, you can see that the decorrelation stretch:

1. Removes a mean from each band
2. Normalizes each band by its standard deviation (correlation-based method only)
3. Rotates the bands into the eigenspace of `Corr` or `Cov`
4. Applies a stretch `S` in the eigenspace, leaving the image decorrelated and normalized in the eigenspace
5. Rotates back to the original band-space, where the bands remain decorrelated and normalized
6. Rescales each band according to `SIGMA_target`
7. Restores a mean in each band.

See Also

`stretchlim` | `imadjust`

Introduced before R2006a

deltaE

Color difference based on CIE76 standard

Syntax

```
dE = deltaE(I1,I2)
dE = deltaE(I1,I2,'isInputLab',isLab)
```

Description

`dE = deltaE(I1,I2)` calculates the color difference between two RGB images or sets of colors using the CIE76 standard.

`dE = deltaE(I1,I2,'isInputLab',isLab)` also specifies whether the input color data is in the RGB color space or the L*a*b* color space.

Examples

Calculate Color Difference of Two Colors using CIE76 Standard

Specify two RGB color values.

```
pureRed = uint8([255 0 0]);
darkRed = uint8([255 10 50]);
```

Calculate the color difference of the colors.

```
dE = deltaE(pureRed,darkRed)
```

```
dE = single
    18.6206
```

Calculate Color Difference of RGB Images

Read a color image into the workspace.

```
I1 = imread('peppers.png');
imshow(I1)
```

Alter the local color contrast in the image.

```
I2 = localcontrast(I1);
imshow(I2)
```

Calculate the color difference of the images.

```
dE = deltaE(I1,I2);
```

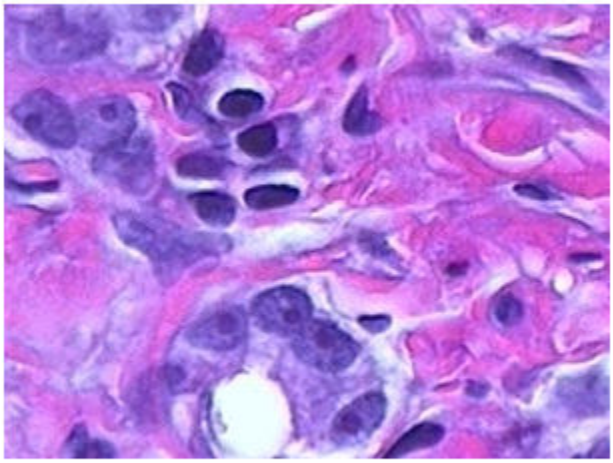
Display the color difference as an image. The maximum value of `dE` exceeds the range [0, 1] expected of images of data type `single`, so display the image using the full display range of the data. Bright pixels indicate a large color difference and therefore a larger amount of contrast enhancement.

```
imshow(dE,[])
```

Calculate Color Difference of L*a*b* Images

Read and display an image of tissue stained with hemotoxylin and eosin (H&E).

```
he = imread('hestain.png');
imshow(he)
```



Convert the image to the L*a*b* color space.

```
lab = rgb2lab(he);
```

Make a copy of the image, then increase the signal of the a* channel. Red tones in the image become more saturated while the image overall brightness and the blue tones are unchanged.

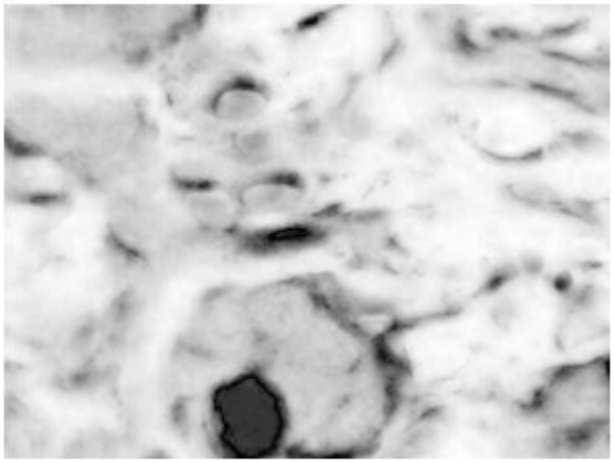
```
lab2 = lab;
scaleFactor = 1.2;
lab2(:,:,2) = scaleFactor*lab(:,:,2);
```

Calculate the color difference of the original and enhanced image in the L*a*b* color space.

```
dE = imcolordiff(lab,lab2,'isInputLab',true);
```

Display the color difference as an image. Scale the display range to match the range of pixel values in `dE`. Bright regions indicate the greatest color difference and correspond with the pink regions of tissue.

```
imshow(dE,[])
```



Input Arguments

`I1` — First set of color data

m-by-*n*-by-3 numeric array | *c*-by-3 numeric matrix

First set of color data, specified as an *m*-by-*n*-by-3 numeric array representing an image or a *c*-by-3 numeric matrix representing a set of *c* colors. `I1` and `I2` must be the same size with values in the same color space.

By default, the `deltaE` function interprets the color data as RGB color values. To calculate the color difference in the L*a*b* color space, specify the `isLab` argument as `true`. L*a*b* color values can be of data type `single` or `double` only.

Data Types: `single` | `double` | `uint8` | `uint16`

`I2` — Second set of color data

m-by-*n*-by-3 numeric array | *c*-by-3 numeric matrix

Second set of color data, specified as an *m*-by-*n*-by-3 numeric array representing an image or a *c*-by-3 numeric matrix representing a set of *c* colors. `I1` and `I2` must be the same size with values in the same color space.

By default, the `deltaE` function interprets the color data as RGB color values. To calculate the color difference in the L*a*b* color space, specify the `isLab` argument as `true`. L*a*b* color values can be of data type `single` or `double` only.

Data Types: `single` | `double` | `uint8` | `uint16`

`isLab` — Color values are in L*a*b* color space

`false` or `0` (default) | `true` or `1`

Color values are in the L*a*b* color space, specified as a numeric or logical `0` (`false`) or `1` (`true`).

Output Arguments

dE — Color difference

m-by-*n* matrix | *c*-element column vector

Color difference (delta E), returned as one of the following.

- An *m*-by-*n* matrix when the input color data `I1` and `I2` represent images
- A *c*-element column vector when `I1` and `I2` represent a set of *c* colors

If `I1` or `I2` is of data type `double`, then `dE` is of data type `double`. Otherwise, `dE` is of data type `single`.

Data Types: `single` | `double`

Tips

- To improve the accuracy of the color difference calculation, use the `imcolordiff` function. This function follows the CIE94 and CIEDE2000 standards and offers parameters to improve perceptual uniformity for different applications.

See Also

`imcolordiff` | `colorangle` | `measureColor`

Topics

“Understanding Color Spaces and Color Space Conversion”

Introduced in R2020b

demosaic

Convert Bayer pattern encoded image to truecolor image

Syntax

```
RGB = demosaic(I,sensorAlignment)
```

Description

`RGB = demosaic(I,sensorAlignment)` converts the Bayer pattern encoded image, `I`, to the truecolor image, `RGB`, using gradient-corrected linear interpolation. `sensorAlignment` specifies the Bayer pattern.

A Bayer filter mosaic, or color filter array, refers to the arrangement of color filters that let each sensor in a single-sensor digital camera record only red, green, or blue data. The patterns emphasize the number of green sensors to mimic the human eye's greater sensitivity to green light. The `demosaic` function uses interpolation to convert the two-dimensional Bayer-encoded image into the truecolor image.

Examples

Convert a Bayer Pattern Encoded Image To an RGB Image

Convert a Bayer pattern encoded image that was photographed by a camera with a sensor alignment of `'bggr'` .

```
I = imread('mandi.tif');
J = demosaic(I,'bggr');
imshow(I);
```


```
figure, imshow(J);
```

Input Arguments

`I` — Bayer-pattern encoded image

M-by-*N* array of intensity values

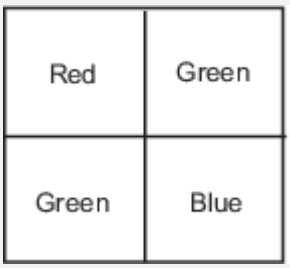
Bayer-pattern encoded image, specified as an *M*-by-*N* array of intensity values. `I` must have at least 5 rows and 5 columns.

Data Types: `uint8` | `uint16` | `uint32`

`sensorAlignment` — Bayer pattern

`'gbrg'` | `'grbg'` | `'bggr'` | `'rggb'`

Bayer pattern, specified as one of the values in the following table. Each value represents the order of the red, green, and blue sensors by describing the four pixels in the upper-left corner of the image (left-to-right, top-to-bottom).

Pattern	2-by-2 Sensor Alignment
`'gbrg'`	
`'grbg'`	
`'bggr'`	
`'rggb'`	

Data Types: `char` | `string`

Output Arguments

RGB — RGB image

M-by-*N*-by-3 numeric array

RGB image, returned as an *M*-by-*N*-by-3 numeric array the same class as `I`.

References

[1] Malvar, H.S., L. He, and R. Cutler, *High quality linear interpolation for demosaicing of Bayer-patterned color images*. ICASPP, Volume 34, Issue 11, pp. 2274-2282, May 2004.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `demosaic` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- `sensorAlignment` must be a compile-time constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- `sensorAlignment` must be a compile-time constant.

See Also

Introduced in R2007b

depthToSpace

Rearrange `dlarray` data from depth dimension into spatial blocks

Syntax

```
Y = depthToSpace(X,blockSize)
Y = depthToSpace(X,blockSize,Name,Value)
```

Description

`Y = depthToSpace(X,blockSize)` rearranges data of the formatted `dlarray` object, `X`, from the depth dimension into spatial blocks of size `blockSize`.

Given an input feature map of size [*H W C*height*width*] and blocks of size [*height width*], the output feature map size is [*H*height W*width C*].

This function requires Deep Learning Toolbox.

`Y = depthToSpace(X,blockSize,Name,Value)` modifies aspects of the depth-to-space rearranging operation using name-value arguments. If `X` is an unformatted `dlarray`, then you must specify the `DataFormat` name-value pair argument.

Examples

Rearrange Formatted dlarray Data from Depth to Spatial Dimension

Create a numeric array of height 2 and width 2 that simulates the depthwise concatenation of blocks of size 2-by-2.

```
X = reshape(1:48,2,2,12);
```

Create a `dlarray` object that contains the numeric data, specifying the format of the data as 'SSC' (spatial, spatial, channel).

```
X = dlarray(X,'SSC')
```

```
X =
  2(S) x 2(S) x 12(C) dlarray


(:,:,1) =

     1     3
     2     4


(:,:,2) =

     5     7
     6     8
```

```
(:,:,3) =

     9    11
    10    12


(:,:,4) =

    13    15
    14    16


(:,:,5) =

    17    19
    18    20


(:,:,6) =

    21    23
    22    24


(:,:,7) =

    25    27
    26    28


(:,:,8) =

    29    31
    30    32


(:,:,9) =

    33    35
    34    36


(:,:,10) =

    37    39
    38    40


(:,:,11) =

    41    43
    42    44


(:,:,12) =

    45    47
```

```
    46    48

  2(S) x 2(S) x 12(C) dlarray
```

Specify a 2-by-2 block size for reordering input activations.

```
blockSize = 2;
```

Rearrange blocks of data from the depth dimension to the spatial dimensions.

```
Z = depthToSpace(X,blockSize)
```

```
Z =
  4(S) x 4(S) x 3(C) dlarray


(:,:,1) =

     1    13     3    15
    25    37    27    39
     2    14     4    16
    26    38    28    40


(:,:,2) =

     5    17     7    19
    29    41    31    43
     6    18     8    20
    30    42    32    44


(:,:,3) =

     9    21    11    23
    33    45    35    47
    10    22    12    24
    34    46    36    48
```

Rearrange Unformatted Data from Depth to Spatial Dimensions

Create a numeric array of height 2 and width 2 that simulates the depthwise concatenation of blocks of size 2-by-2.

```
X = reshape(1:48,2,2,12);
```

Create an unformatted `dlarray` object that contains the numeric data.

```
dlX = dlarray(X);
```

Specify a 2-by-2 block size for reordering input activations.

```
blockSize = 2;
```

Rearrange blocks of data from the depth dimension to the spatial dimensions, specifying the data format. Order the data by column, row, and then depth.

```
dlZ = depthToSpace(dlX,blockSize,"DataFormat","SSC","Mode","CRD")
```

```
dlZ =
  4x4x3 dlarray


(:,:,1) =

     1     5     3     7
     9    13    11    15
     2     6     4     8
    10    14    12    16


(:,:,2) =

    17    21    19    23
    25    29    27    31
    18    22    20    24
    26    30    28    32


(:,:,3) =

    33    37    35    39
    41    45    43    47
    34    38    36    40
    42    46    44    48
```

Input Arguments

X — Deep learning data to rearrange

`dlarray` object

Deep learning data to rearrange, specified as a `dlarray` object.

blockSize — Block size to reorder input activation

positive integer | vector of two positive integers

Block size to reorder the input activation, specified as a positive integer or vector of two positive integers of the form `[h w]`, where `h` is the height and `w` is the width. When you specify `blockSize` as a scalar, the function uses the same value for both dimensions.

Example: `[2 4]` specifies blocks of height 2 and width 4.

Example: 32 specifies blocks of height and width 32.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'DataFormat','SSC'` specifies an array with two spatial dimensions and one channel dimension, appropriate for 2-D RGB image data.

`DataFormat` — Dimension labels
`"SSCB"` (default) | string scalar | character vector

Dimension labels when the input deep learning data `X` is unlabeled, specified as a string scalar or character vector. The number of labels must match the number of dimensions of the input data, `X`. Each character in '`DataFormat`' must be one of these labels:

- `S` — Spatial
- `C` — Channel
- `B` — Batch observations

The "T" (time or sequence) and "U" (unspecified) labels are not supported. Do not specify the '`DataFormat`' argument when the input deep learning data is a formatted `dlarray` object.

Example: `"SSCB"` indicates the array has two spatial dimensions, one channel dimension, and one batch dimension.

Data Types: `char` | `string`

`Mode` — Order of rearranged dimensions
`"DCR"` (default) | `"CRD"`

Order of rearranged dimensions from the input deep learning data `X`, specified as `"DCR"` or `"CRD"`. When you specify `"DCR"`, the function orders data by depth, column, and then row. When you specify `"CRD"`, the function orders data by column, row, and then depth.

Data Types: `char` | `string`

Output Arguments

`Y` — Rearranged deep learning data
`dlarray` object

Rearranged deep learning data, returned as a `dlarray` object.

Extended Capabilities

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also
`spaceToDepth` | `dlresize`

Topics
“List of Functions with dlarray Support” (Deep Learning Toolbox)

Introduced in R2021a

denoiseImage

Denoise image using deep neural network

Syntax

```
B = denoiseImage(A,net)
```

Description

`B = denoiseImage(A,net)` estimates denoised image `B` from noisy image `A` using a denoising deep neural network specified by `net`.

This function requires that you have Deep Learning Toolbox.

Examples

Remove Image Noise Using Pretrained Neural Network

Load the pretrained denoising convolutional neural network, `'DnCNN'`.

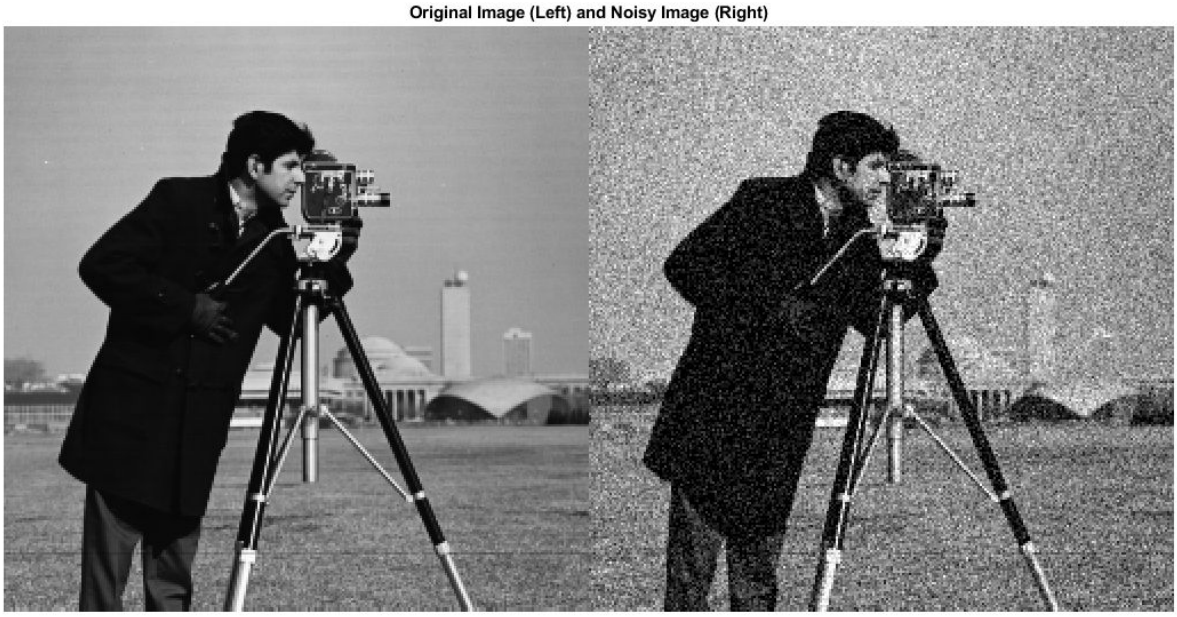
```
net = denoisingNetwork('DnCNN');
```

Load a grayscale image into the workspace, then create a noisy version of the image.

```
I = imread('cameraman.tif');
noisyI = imnoise(I,'gaussian',0,0.01);
```

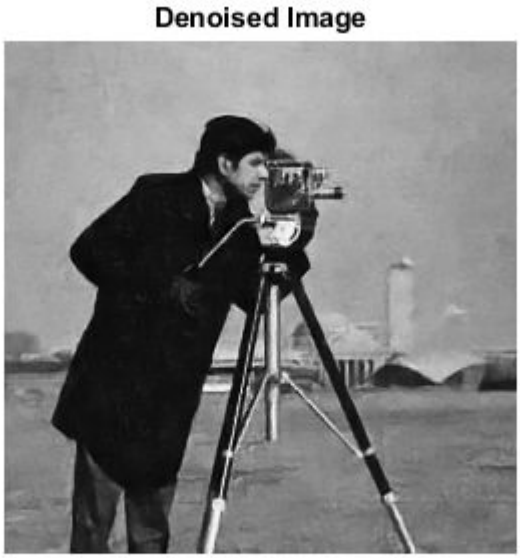
Display the two images as a montage.

```
montage({I,noisyI})
title('Original Image (Left) and Noisy Image (Right)')
```



Remove noise from the noisy image, then display the result.

```
denoisedI = denoiseImage(noisyI,net);
imshow(denoisedI)
title('Denoised Image')
```



Input Arguments

A — Noisy image

2-D image | stack of 2-D images

Noisy image, specified as a single 2-D image or a stack of 2-D images. A can be:

- A 2-D grayscale image with size *m*-by-*n*.
- A 2-D multichannel image with size *m*-by-*n*-by-*c*, where *c* is the number of image channels. For example, *c* is 3 for RGB images, and 4 for four-channel images such as RGB images with an infrared channel.
- A stack of equally-sized 2-D images. In this case, A has size *m*-by-*n*-by-*c*-by-*p*, where *p* is the number of images in the stack.

Data Types: single | double | uint8 | uint16

net — Denoising deep neural network

SeriesNetwork object

Denoising deep neural network, specified as a SeriesNetwork object. The network should be trained to handle images with the same channel format as A.

If the noisy image or stack of images A has only one channel and has Gaussian noise, then you can get a pretrained network using the denoisingNetwork function. For more information about creating a denoising network for multichannel images or for a different noise model, see “Train and Apply Denoising Neural Networks”.

Output Arguments

B — Denoised image

2-D image | stack of 2-D images

Denoised image, returned as a single 2-D image or a stack of 2-D images. B has the same size and data type as A.

Tips

- The denoiseImage function relies on the activations function to estimate the noise of the input image, A. The denoiseImage function specifies the OutputAs name-value argument of activations as "channels" so that A can be larger than the network input size. In contrast, the predict function requires that the image size match the network input size.

See Also

denoisingNetwork | dnCNNLayers | denoisingImageDatastore

Topics

“Train and Apply Denoising Neural Networks”

Introduced in R2017b

denoisingImageDatastore

Denoising image datastore

Description

Use a `denoisingImageDatastore` object to generate batches of noisy image patches and corresponding noise patches from images in an `ImageDatastore`. The patches are used to train a denoising deep neural network.

This object requires that you have Deep Learning Toolbox.

Note When you use a denoising image datastore as a source of training data, the datastore adds random noise to the image patches for each epoch, so that each epoch uses a slightly different data set. The actual number of training images at each epoch is increased by a factor of `PatchesPerImage`. The noisy image patches and corresponding noise patches are not stored in memory.

Creation

Syntax

```
dnimds = denoisingImageDatastore(imds)
dnimds = denoisingImageDatastore(imds,Name,Value)
```

Description

`dnimds = denoisingImageDatastore(imds)` creates a denoising image datastore, `dnimds` using images from image datastore `imds`. To generate noisy image patches, the denoising image datastore randomly crops pristine images from `imds` then adds zero-mean Gaussian white noise with a standard deviation of `0.1` to the image patches.

`dnimds = denoisingImageDatastore(imds,Name,Value)` uses name-value pairs to specify the two-dimensional image patch size or to set the `PatchesPerImage`, `GaussianNoiseLevel`, `ChannelFormat`, and `DispatchInBackground` properties. You can specify multiple name-value pairs. Enclose each argument or property name in quotes.

For example, `denoisingImageDatastore(imds,'PatchesPerImage',40)` creates a denoising image datastore and randomly generates 40 noisy patches from each image in the image datastore, `imds`.

Input Arguments

`imds` — Image datastore
`ImageDatastore` object

Image datastore, specified as an `ImageDatastore` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `denoisingImageDatastore(imds,'patchSize',48)` creates a denoising image datastore that has a square patch size of 48 pixels.

`patchSize` — Patch size

`50` (default) | scalar | 2-element vector

Patch size, specified as the comma-separated pair consisting of `'patchSize'` and a scalar or 2-element vector with positive integer values. This argument sets the first two elements of the `PatchSize` property.

- If `'patchSize'` is a scalar, then the patches are square.
- If `'patchSize'` is a 2-element vector of the form [*r c*], then the first element specifies the number of rows in the patch, and the second element specifies the number of columns.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Properties

`ChannelFormat` — Channel format

`'grayscale'` (default) | `'rgb'`

Channel format, specified as `'grayscale'` or `'rgb'`.

Data Types: `char`

`DispatchInBackground` — Dispatch observations in background

`false` (default) | `true`

Dispatch observations in the background during training, prediction, and classification, specified as `false` or `true`. To use background dispatching, you must have Parallel Computing Toolbox. If `DispatchInBackground` is `true` and you have Parallel Computing Toolbox, then `denoisingImageDatastore` asynchronously reads patches, adds noise, and queues patch pairs.

`GaussianNoiseLevel` — Gaussian noise standard deviation

`0.1` (default) | scalar | 2-element vector

Gaussian noise standard deviation as a fraction of the image class maximum, specified as a scalar or 2-element vector with values in the range [0, 1].

- If `GaussianNoiseLevel` is a scalar, then the standard deviation of the added zero-mean Gaussian white noise is identical for all image patches.
- If `GaussianNoiseLevel` is a 2-element vector, then it specifies a range of standard deviations [*stdmin stdmax*]. The standard deviation of the added zero-mean Gaussian white noise is unique for each image patch, and is randomly sampled from a uniform distribution with the range [*stdmin stdmax*].

Data Types: `single` | `double`

`MiniBatchSize` — Number of observations in each batch
`128` | positive integer

Number of observations that are returned in each batch. You can change the value of `MiniBatchSize` only after you create the datastore. For training, prediction, or classification, the `MiniBatchSize` property is set to the mini-batch size defined in `trainingOptions`.

`NumObservations` — Total number of observations in the datastore
positive integer

This property is read-only.

Total number of observations in the denoising image datastore. The number of observations is the length of one training epoch.

`PatchesPerImage` — Number of random patches per image
`512` (default) | positive integer

Number of random patches per image, specified as a positive integer.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`PatchSize` — Patch size
`[50 50 1]` (default) | 3-element vector of positive integers

This property is read-only.

Patch size, specified as a 3-element vector of positive integers. If you create a denoising image datastore by specifying a `'patchSize'` name-value pair argument, then the first two elements of the `PatchSize` property are set according to the value of the `patchSize` argument.

The `ChannelFormat` property determines the third element of the `PatchSize` property.

- If `ChannelFormat` is `'Grayscale'`, then all color images are converted to grayscale and the third element of `PatchSize` is `1`.
- If `ChannelFormat` is `'RGB'`, then grayscale images are replicated to simulate an RGB image and the third element of `PatchSize` is `3`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Object Functions

combine	Combine data from multiple datastores
hasdata	Determine if data is available to read
partitionByIndex	Partition denoisingImageDatastore according to indices
preview	Preview subset of data in datastore
read	Read data from denoisingImageDatastore
readall	Read all data in datastore
readByIndex	Read data specified by index from denoisingImageDatastore
reset	Reset datastore to initial state
shuffle	Shuffle data in datastore
transform	Transform datastore
isPartitionable	Determine whether datastore is partitionable
isShuffleable	Determine whether datastore is shuffleable

Examples

Create Denoising Image Datastore

Get an image datastore. The datastore in this example contains color images.

```
setDir = fullfile(toolboxdir('images'),'imdata');
imds = imageDatastore(setDir,'FileExtensions',{'.jpg'});
```

Create a `denoisingImageDatastore` object that creates many patches from each image in the image datastore, and adds Gaussian noise to the patches. Set the optional `PatchesPerImage`, `PatchSize`, `GaussianNoiseLevel`, and `ChannelFormat` properties of the `denoisingImageDatastore` using name-value pairs. When you set the `ChannelFormat` property to '`grayscale`', the `denoisingImageDatastore` converts all color images to grayscale.

```
dnds = denoisingImageDatastore(imds,...
    'PatchesPerImage',512,...
    'PatchSize',50,...
    'GaussianNoiseLevel',[0.01 0.1],...
    'ChannelFormat','grayscale')
```

```
dnds =
  denoisingImageDatastore with properties:

         PatchesPerImage: 512
               PatchSize: [50 50 1]
      GaussianNoiseLevel: [0.0100 0.1000]
           ChannelFormat: 'grayscale'
           MiniBatchSize: 128
         NumObservations: 19456
    DispatchInBackground: 0
```

Tips

- Training a deep neural network for a range of Gaussian noise standard deviations is a much more difficult problem than training a network for a single Gaussian noise standard deviation. You should create more patches compared to a single noise level case, and training might take more time.
- To visualize the data in a denoising image datastore, you can use the `preview` function, which returns a subset of data in a table. The `input` variable contains the noisy image patches and the `response` variable contains the corresponding noise patches. Visualize all of the noisy image patches or noise patches in the same figure by using the `montage` function. For example, this code displays data in a denoising image datastore called `dnimds`.

  ```
  minibatch = preview(dnimds);
  montage(minibatch.input)
  figure
  montage(minibatch.response)
  ```
- Each time images are read from the denoising image datastore, a different random amount of Gaussian noise is added to each image.

See Also

`denoiseImage` | `denoisingNetwork` | `dnCNNLayers` | `trainNetwork`

Topics

“Train and Apply Denoising Neural Networks”

Introduced in R2018a

partitionByIndex

Partition `denoisingImageDatastore` according to indices

Syntax

```
dnimds2 = partitionByIndex(dnimds,ind)
```

Description

`dnimds2 = partitionByIndex(dnimds,ind)` partitions a subset of observations in a denoising image datastore, `dnimds`, into a new datastore, `dnimds2`. The desired observations are specified by indices, `ind`.

Input Arguments

`dnimds` — Denoising image datastore

`denoisingImageDatastore`

Denoising image datastore, specified as a `denoisingImageDatastore` object.

`ind` — Indices

vector of positive integers

Indices of observations, specified as a vector of positive integers.

Output Arguments

`dnimds2` — Output datastore

`denoisingImageDatastore` object

Output datastore, returned as a `denoisingImageDatastore` object containing a subset of files from `dnimds`.

See Also

`denoisingImageDatastore` | `read` | `readall` | `readByIndex`

Introduced in R2018a

read

Read data from `denoisingImageDatastore`

Syntax

```
data = read(dnimds)
[data,info] = read(dnimds)
```

Description

`data = read(dnimds)` returns a batch of data from a denoising image datastore, `dnimds`. Subsequent calls to the `read` function continue reading from the endpoint of the previous call.

`[data,info] = read(dnimds)` also returns information about the extracted data, including metadata, in `info`.

Input Arguments

`dnimds` — Denoising image datastore

`denoisingImageDatastore`

Denoising image datastore, specified as a `denoisingImageDatastore` object. The datastore specifies a `MiniBatchSize` number of observations in each batch, and a `numObservations` total number of observations.

Output Arguments

`data` — Output data

table

Output data, returned as a table with `MiniBatchSize` number of rows.

For the last batch of data in the datastore `dnimds`, if `numObservations` is not cleanly divisible by `MiniBatchSize`, then `read` returns a partial batch containing all the remaining observations in the datastore.

`info` — Information about read data

structure array

Information about read data, returned as a structure array. The structure array can contain the following fields.

Field Name	Description
`CurrentFileIndices`	Current read index of the denoising image datastore.

See Also

`denoisingImageDatastore` | `read (Datastore)` | `readByIndex` | `readall`

Introduced in R2018a

readByIndex

Read data specified by index from `denoisingImageDatastore`

Syntax

```
data = readByIndex(dnimds,ind)
[data,info] = readByIndex(dnimds,ind)
```

Description

`data = readByIndex(dnimds,ind)` returns a subset of observations from a denoising image datastore, `dnimds`. The desired observations are specified by indices, `ind`.

`[data,info] = readByIndex(dnimds,ind)` also returns information about the observations, including metadata, in `info`.

Input Arguments

`dnimds` — Denoising image datastore

`denoisingImageDatastore`

Denoising image datastore, specified as a `denoisingImageDatastore` object.

`ind` — Indices

vector of positive integers

Indices of observations, specified as a vector of positive integers.

Output Arguments

`data` — Observations from datastore

table

Observations from the datastore, returned as a table with `length(ind)` number of rows.

`info` — Information about read data

structure array

Information about read data, returned as a structure array. The structure array can contain the following fields.

Field Name	Description
`CurrentFileIndices`	Numeric vector containing the indices of all read files of the denoising image datastore.

See Also

`denoisingImageDatastore` | `read` | `readall` | `partitionByIndex`

Introduced in R2018a

denoisingImageSource

(To be removed) Create denoising image datastore

Note `denoisingImageSource` will be removed in a future release. Use `denoisingImageDatastore` instead. For more information, see Compatibility Considerations.

Syntax

```
dnimds = denoisingImageSource(imds)
dnimds = denoisingImageSource(imds,Name,Value)
```

Description

`dnimds = denoisingImageSource(imds)` creates a denoising image datastore, `dnimds`, that generates pairs of randomly cropped pristine and noisy image patches from images in image datastore `imds`.

`dnimds = denoisingImageSource(imds,Name,Value)` sets properties on page 1-673 of the denoising image datastore using name-value pairs. You can specify multiple name-value pairs. Enclose each argument name in quotes.

Examples

Create Denoising Image Datastore Using `denoisingImageSource`

Create an image datastore. This datastore contains color JPG images.

```
setDir = fullfile(toolboxdir("images"),"imdata");
imds = imageDatastore(setDir,"FileExtensions",[".jpg"]);
```

Create a `denoisingImageDatastore` object using the `denoisingImageSource` function. The image datastore creates many patches from each image in the datastore, and adds Gaussian noise to the patches. Set the optional `PatchesPerImage`, `PatchSize`, `GaussianNoiseLevel`, and `ChannelFormat` properties of the `denoisingImageDatastore` using name-value pairs.

```
dnimds = denoisingImageSource(imds, ...
    "PatchesPerImage",512, ...
    "PatchSize",50, ...
    "GaussianNoiseLevel",[0.01 0.1], ...
    "ChannelFormat","RGB")
```

```
dnimds =
  denoisingImageDatastore with properties:

         PatchesPerImage: 512
               PatchSize: [50 50 3]
      GaussianNoiseLevel: [0.0100 0.1000]
           ChannelFormat: 'rgb'
           MiniBatchSize: 128
```

```
      NumObservations: 18944
DispatchInBackground: 0
```

Input Arguments

`imds` — Image datastore

`ImageDatastore` object

Image datastore, specified as an `ImageDatastore` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"PatchSize",48` creates a denoising image datastore that has a square patch size of 48 pixels.

`PatchSize` — Patch size

`50` (default) | scalar | 2-element vector

Patch size, specified as a scalar or 2-element vector with positive integer values. This argument sets the first two elements of the `PatchSize on page 1-0` property of the returned denoising image datastore, `dnimds`.

- When `"PatchSize"` is a scalar, the patches are square
- When `"PatchSize"` is a 2-element vector of the form [*r c*], the first element specifies the number of rows in the patch, and the second element specifies the number of columns

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`PatchesPerImage` — Number of random patches per image

`512` (default) | positive integer

Number of random patches per image, specified as a positive integer. This argument sets the `PatchesPerImage on page 1-0` property of the returned denoising image datastore, `dnimds`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`GaussianNoiseLevel` — Gaussian noise standard deviation

`0.1` (default) | scalar | 2-element vector

Gaussian noise standard deviation as a fraction of the image class maximum, specified as a scalar or 2-element vector with values in the range [0, 1]. This argument sets the `GaussianNoiseLevel on page 1-0` property of the returned denoising image datastore, `dnimds`.

- If `GaussianNoiseLevel` is a scalar, then the standard deviation of the added zero-mean Gaussian white noise is identical for all image patches.
- If `GaussianNoiseLevel` is a 2-element vector, then it specifies a range of standard deviations [*stdmin stdmax*]. The standard deviation of the added zero-mean Gaussian white noise is unique

for each image patch, and is randomly sampled from a uniform distribution with the range [*stdmin stdmax*].

Data Types: `single` | `double`

`ChannelFormat` — Channel format
`"Grayscale"` (default) | `"RGB"`

Channel format, specified as `"Grayscale"` or `"RGB"`. This argument sets the `ChannelFormat on page 1-0` property of the returned denoising image datastore, `dnimds`.

Data Types: `char`

`BackgroundExecution` — Preprocess training patches in parallel
`false` (default) | `true`

Preprocess training patches in parallel, specified as `true` or `false`. This argument sets the `DispatchInBackground on page 1-0` property of the returned denoising image datastore, `dnimds`. If `BackgroundExecution` is `true` and you have Parallel Computing Toolbox, then the denoising image datastore asynchronously reads patches, adds noise, and queues patch pairs.

Data Types: `char`

Output Arguments

`dnimds` — Denoising image datastore
`denoisingImageDatastore` object

Denoising image datastore, returned as an `denoisingImageDatastore` object.

Compatibility Considerations

`denoisingImageSource` object is removed

In R2017b, you could create a `denoisingImageSource` object for training deep learning networks. Starting in R2018a, the `denoisingImageSource` object has been removed. Use a `denoisingImageDatastore` object instead.

A `denoisingImageDatastore` has additional properties and methods to assist with data preprocessing. Unlike `denoisingImageSource`, which could be used for training only, you can use a `denoisingImageDatastore` for both training and prediction.

To create a `denoisingImageDatastore` object, you can use either the `denoisingImageDatastore` function (recommended) or the `denoisingImageSource` function.

`denoisingImageSource` function will be removed
Not recommended starting in R2018a

The `denoisingImageSource` function will be removed in a future release. Create a `denoisingImageDatastore` using the `denoisingImageDatastore` function instead.

To update your code, change instances of the function name `denoisingImageSource` to `denoisingImageDatastore`. You do not need to change the input arguments.

See Also

`denoisingImageDatastore`

Introduced in R2017b

denoisingNetwork

Get image denoising network

Syntax

```
net = denoisingNetwork(modelName)
```

Description

`net = denoisingNetwork(modelName)` returns a pretrained image denoising deep neural network specified by `modelName`.

This function requires that you have Deep Learning Toolbox.

Examples

Get Pretrained Image Denoising Network

Get the pretrained image denoising convolutional neural network, 'DnCNN'.

```
net = denoisingNetwork('DnCNN')
```

```
net =
  SeriesNetwork with properties:

         Layers: [59x1 nnet.cnn.layer.Layer]
     InputNames: {'InputLayer'}
    OutputNames: {'FinalRegressionLayer'}
```

See `denoiseImage` for an example of how to denoise an image using the pretrained network.

Input Arguments

modelName — Name of neural network

`'DnCnn'`

Name of pretrained denoising deep neural network, specified as the character vector `'DnCnn'`. This is the only pretrained denoising network currently available, and it is trained for grayscale images only.

Data Types: `char` | `string`

Output Arguments

net — Denoising deep neural network

`SeriesNetwork` object

Pretrained denoising deep neural network, returned as a `SeriesNetwork` object.

References

[1] Zhang, K., W. Zuo, Y. Chen, D. Meng, and L. Zhang. "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising." *IEEE Transactions on Image Processing*. Vol. 26, Number 7, Feb. 2017, pp. 3142-3155.

See Also

`denoiseImage` | `dnCNNLayers` | `denoisingImageDatastore`

Topics

“Train and Apply Denoising Neural Networks”

Introduced in R2017b

dice

Sørensen-Dice similarity coefficient for image segmentation

Syntax

```
similarity = dice(BW1,BW2)
similarity = dice(L1,L2)
similarity = dice(C1,C2)
```

Description

`similarity = dice(BW1,BW2)` computes the Sørensen-Dice similarity coefficient between binary images `BW1` and `BW2`.

`similarity = dice(L1,L2)` computes the Dice index for each label in label images `L1` and `L2`.

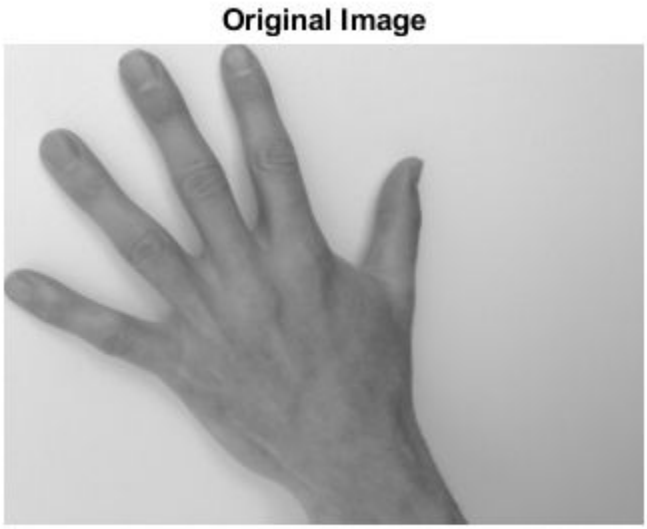
`similarity = dice(C1,C2)` computes the Dice index for each category in categorical images `C1` and `C2`.

Examples

Compute Dice Similarity Coefficient for Binary Segmentation

Read an image with an object to segment. Convert the image to grayscale, and display the result.

```
A = imread('hands1.jpg');
I = im2gray(A);
figure
imshow(I)
title('Original Image')
```



Use active contours (snakes) to segment the hand.

```
mask = false(size(I));
mask(25:end-25,25:end-25) = true;
BW = activecontour(I, mask, 300);
```

Read in the ground truth segmentation.

```
BW_groundTruth = imread('hands1-mask.png');
```

Compute the Dice index of the active contours segmentation against the ground truth.

```
similarity = dice(BW, BW_groundTruth);
```

Display the masks on top of each other. Colors indicate differences in the masks.

```
figure
imshowpair(BW, BW_groundTruth)
title(['Dice Index = ' num2str(similarity)])
```

Compute Dice Similarity Coefficient for Multi-Region Segmentation

This example shows how to segment an image into multiple regions. The example then computes the Dice similarity coefficient for each region.

Read an image with several regions to segment.

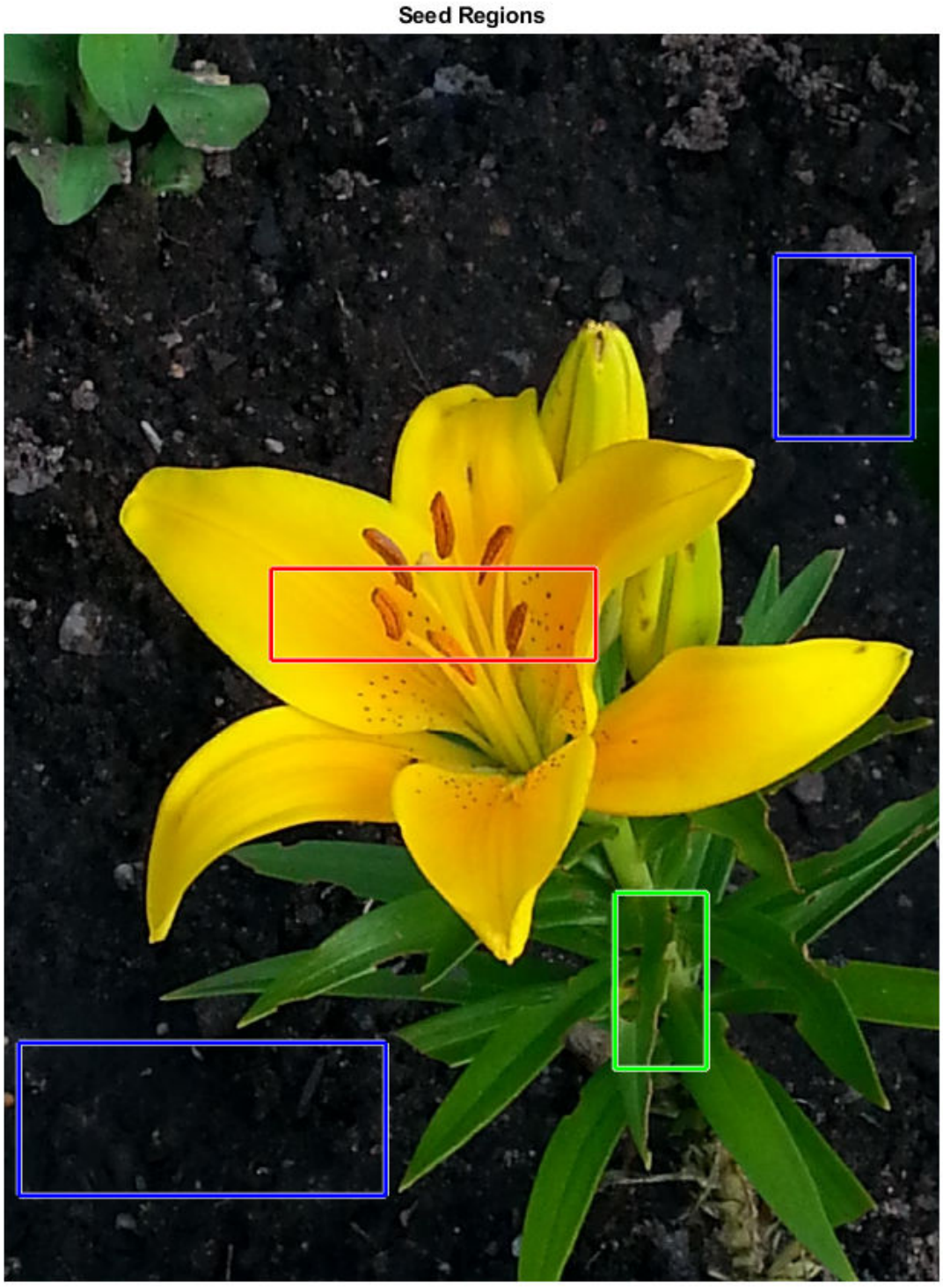
```
RGB = imread('yellowlily.jpg');
```

Create scribbles for three regions that distinguish their typical color characteristics. The first region classifies the yellow flower. The second region classifies the green stem and leaves. The last region classifies the brown dirt in two separate patches of the image. Regions are specified by a 4-element vector, whose elements indicate the x- and y-coordinate of the upper left corner of the ROI, the width of the ROI, and the height of the ROI.

```
region1 = [350 700 425 120]; % [x y w h] format
BW1 = false(size(RGB,1),size(RGB,2));
BW1(region1(2):region1(2)+region1(4),region1(1):region1(1)+region1(3)) = true;

region2 = [800 1124 120 230];
BW2 = false(size(RGB,1),size(RGB,2));
BW2(region2(2):region2(2)+region2(4),region2(1):region2(1)+region2(3)) = true;

region3 = [20 1320 480 200; 1010 290 180 240];
BW3 = false(size(RGB,1),size(RGB,2));
BW3(region3(1,2):region3(1,2)+region3(1,4),region3(1,1):region3(1,1)+region3(1,3)) = true;
BW3(region3(2,2):region3(2,2)+region3(2,4),region3(2,1):region3(2,1)+region3(2,3)) = true;
```

Display the seed regions on top of the image.

```
imshow(RGB)
hold on
visboundaries(BW1,'Color','r');
visboundaries(BW2,'Color','g');
visboundaries(BW3,'Color','b');
title('Seed Regions')
```



Segment the image into three regions using geodesic distance-based color segmentation.

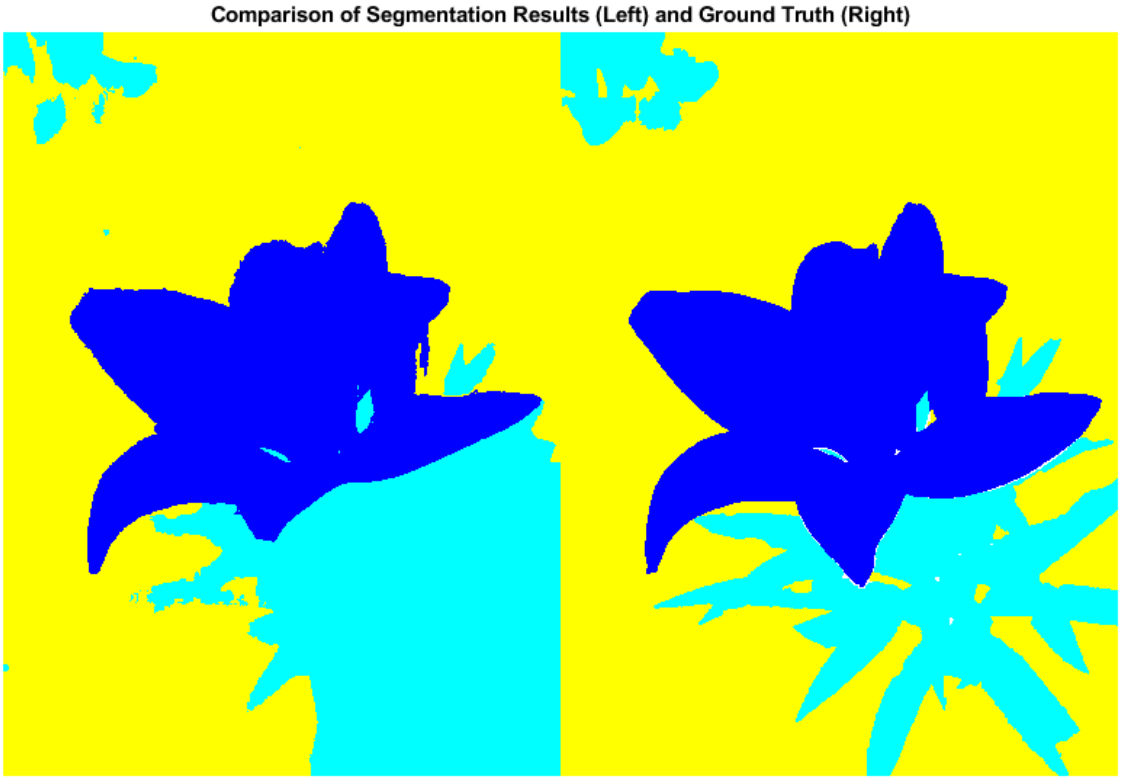
```
L = imseggeodesic(RGB,BW1,BW2,BW3,'AdaptiveChannelWeighting',true);
```

Load a ground truth segmentation of the image.

```
L_groundTruth = double(imread('yellowlily-segmented.png'));
```

Visually compare the segmentation results with the ground truth.

```
figure
montage({label2rgb(L),label2rgb(L_groundTruth)})
title('Comparison of Segmentation Results (Left) and Ground Truth (Right)')
```



Compute the Dice similarity index for each segmented region. The Dice similarity index is noticeably smaller for the second region. This result is consistent with the visual comparison of the segmentation results, which erroneously classifies the dirt in the lower right corner of the image as leaves.

```
similarity = dice(L, L_groundTruth)
```

```
similarity = 3×1

    0.9396
    0.7247
    0.9139
```

Input Arguments

`BW1` — First binary image

logical array

First binary image, specified as a logical array of any dimension.

Data Types: `logical`

`BW2` — Second binary image

logical array

Second binary image, specified as a logical array of the same size as `BW1`.

Data Types: `logical`

`L1` — First label image

array of nonnegative integers

First label image, specified as an array of nonnegative integers, of any dimension.

Data Types: `double`

`L2` — Second label image

array of nonnegative integers

Second label image, specified as an array of nonnegative integers, of the same size as `L1`.

Data Types: `double`

`C1` — First categorical image

`categorical` array

First categorical image, specified as a `categorical` array of any dimension.

Data Types: `category`

`C2` — Second categorical image

`categorical` array

Second categorical image, specified as a `categorical` array of the same size as `C1`.

Data Types: `category`

Output Arguments

`similarity` — Dice similarity coefficient

numeric scalar | numeric vector

Dice similarity coefficient, returned as a numeric scalar or numeric vector with values in the range [0, 1]. A `similarity` of 1 means that the segmentations in the two images are a perfect match. If the input arrays are:

- binary images, `similarity` is a scalar.
- label images, `similarity` is a vector, where the first coefficient is the Dice index for label 1, the second coefficient is the Dice index for label 2, and so on.

- categorical images, `similarity` is a vector, where the first coefficient is the Dice index for the first category, the second coefficient is the Dice index for the second category, and so on.

Data Types: `double`

More About

Dice Similarity Coefficient

The Dice similarity coefficient of two sets A and B is expressed as:

$$\texttt{dice}(A,B) = 2 * |\, \texttt{intersection}(A,B)\,| \,/\, (\,|A| + |B|\,)$$

where $|A|$ represents the cardinal of set A. The Dice index can also be expressed in terms of true positives (TP), false positives (FP) and false negatives (FN) as:

$$\texttt{dice}(A,B) = 2 * TP \,/\, (\, 2 * TP + FP + FN)$$

The Dice index is related to the Jaccard index according to:

$$\texttt{dice}(A,B) = 2 * \texttt{jaccard}(A,B) \,/\, (1 + \texttt{jaccard}(A,B)\,)$$

See Also

`jaccard` | `bfscore`

Introduced in R2017b

dicomanon

Anonymize DICOM file

Syntax

```
dicomanon(file_in,file_out)
dicomanon(file_in,file_out,"keep",fields)
dicomanon( ___ ,"update",attributes)
dicomanon( ___ ,Name,Value)
```

Description

`dicomanon(file_in,file_out)` removes confidential medical information from the DICOM file `file_in` and creates a new file `file_out` with the modified values. Image data and other attributes are unmodified.

`dicomanon(file_in,file_out,"keep",fields)` modifies all of the confidential data except for those listed in `fields`. This syntax is useful for keeping metadata that does not uniquely identify the patient but is useful for diagnostic purposes (such as `PatientAge` and `PatientSex`).

Note Keeping certain fields might compromise patient confidentiality.

`dicomanon( ___ ,"update",attributes)` modifies the confidential data and updates particular confidential data listed in `attributes`, in addition to any combination of input arguments from previous syntaxes. Use this syntax to preserve the Study/Series/Image hierarchy, or to replace a specific value with a more generic attribute (such as removing `PatientBirthDate` but keeping a computed `PatientAge`).

`dicomanon( ___ ,Name,Value)` uses name-value arguments to provide additional options to the parser.

Examples

Remove All Confidential Metadata from DICOM File

Create a version of a DICOM file with all the personal information removed.

```
dicomanon("US-PAL-8-10x-echo.dcm","US-PAL-anonymized.dcm");
```

Create a version of a DICOM file with personal information removed, keeping certain fields that could be useful for training.

```
dicomanon("US-PAL-8-10x-echo.dcm","US-PAL-anonymized.dcm","keep",...
         ["PatientAge","PatientSex","StudyDescription"])
```

Anonymize a series of images, keeping the study and series hierarchy.

```
values.StudyInstanceUID = dicomuid;
values.SeriesInstanceUID = dicomuid;
```

```
d = dir("*.dcm");
for p = 1:numel(d)
    dicomanon(d(p).name, sprintf("anon%d.dcm",p), ...
        "update",values)
end
```

Input Arguments

`file_in` — Name of DICOM file to read

character vector | string scalar

Name of the DICOM file to read, specified as a character vector or string scalar.

Data Types: `char` | `string`

`file_out` — Name of anonymized DICOM file to write

character vector | string scalar

Name of the anonymized DICOM file to write, specified as a character vector or string scalar.

Data Types: `char` | `string`

`fields` — Names of fields to preserve

string array | cell array of character vectors

Names of the fields to preserve, specified as a string array or cell array of character vectors containing metadata attribute names.

`attributes` — Attributes to update

structure

Attributes to update, specified as a structure whose fields are the names of the DICOM metadata attributes you want to update. The value of each field specifies the new value for the attribute.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `dicomanon("CT-MONO2-16-ankle.dcm","CT-MONO2-16-ankle_anon.dcm",UseVRHeuristic=false)` reads the metadata from the original DICOM file without using a heuristic.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `dicomanon("CT-MONO2-16-ankle.dcm","CT-MONO2-16-ankle_anon.dcm","UseVRHeuristic",false)` reads the metadata from the original DICOM file without using a heuristic.

`WritePrivate` — Write nonstandard attributes to anonymized file

`false` or `0` (default) | `true` or `1`

Write nonstandard attributes to the anonymized file, specified as a logical `0` (`false`) or `1` (`true`).

When set to `true`, the function includes private metadata attributes in the file. These attributes could compromise patient confidentiality.

Data Types: `logical`

`UseVRHeuristic` — Read noncompliant DICOM files that switch VR modes incorrectly
`true` or `1` (default) | `false` or `0`

Read noncompliant DICOM files that switch value representation (VR) modes incorrectly, specified as a logical `1` (`true`) or `0` (`false`).

When set to `true`, `dicomanon` uses a heuristic to help read certain noncompliant DICOM files that switch VR modes incorrectly. `dicomanon` displays a warning if it uses this heuristic. If this heuristic is enabled, a small number of compliant files are not read correctly. Set `UseVRHeuristic` to `false` to read these compliant files.

Data Types: `logical`

Tips

- For information about the fields that will be modified or removed, see DICOM Supplement 55 from https://www.dicomstandard.org/.

See Also

`dicominfo` | `dicomdict` | `dicomdisp` | `dicomwrite` | `dicomlookup` | `dicomread` | `dicomuid`

Topics
“Remove Confidential Information from DICOM File”

Introduced before R2006a

dicomCollection

Gather details about related series of DICOM files

Syntax

```
collection = dicomCollection(directory)
collection = dicomCollection(directory,"IncludeSubfolders",tf)
collection = dicomCollection(DICOMDIR)
```

Description

`collection = dicomCollection(directory)` gathers details about the DICOM files contained in `directory` and returns the file details in the table `collection`. The `dicomCollection` function aggregates details by DICOM series, using the value of the `SeriesInstanceUID` metadata field in each file to determine series membership. A DICOM series is a logically related set of images from an imaging operation.

`collection = dicomCollection(directory,"IncludeSubfolders",tf)` recursively searches for DICOM files below `directory` when `tf` is `true` (the default). When `tf` is `false`, `dicomCollection` searches only within `directory`.

`collection = dicomCollection(DICOMDIR)` gathers details about the DICOM files referenced in the DICOM directory file `DICOMDIR`. A DICOM directory file (DICOMDIR) is a special DICOM file that serves as a directory to a collection of DICOM files stored on removable media, such as CD- or DVD-ROMs.

Examples

Gather Details from DICOM Files in Sample Image Folder

Gather information about the DICOM files in the Image Processing Toolbox™ sample image folder.

```
details = dicomCollection(fullfile(matlabroot,"toolbox/images/imdata"))
```

```
details=6×14 table
              StudyDateTime              SeriesDateTime           PatientName      PatientSex
          ____________________________    ____________________________    _________________    __________

    s1    {0×0 double            }    {0×0 double            }    ""                   ""
    s2    {[30-Apr-1993 11:27:24]}    {[30-Apr-1993 11:27:24]}    "Anonymized"         ""
    s3    {[14-Dec-2013 15:47:31]}    {[14-Dec-2013 15:54:33]}    "GORBERG MITZI"      "F"
    s4    {[03-Oct-2011 19:18:11]}    {[03-Oct-2011 18:59:02]}    ""                   "M"
    s5    {[03-Oct-2011 19:18:11]}    {[03-Oct-2011 19:05:04]}    ""                   "M"
    s6    {[30-Jan-1994 11:25:01]}    {0×0 double            }    "Anonymized"         ""
```

Gather Details About DICOM Files from DICOMDIR File

Gather information about DICOM files in a folder from a DICOMDIR file.

```
details = dicomCollection(fullfile(matlabroot,"toolbox/images/imdata/DICOMDIR"))
```

```
details=4×14 table
            StudyDateTime        SeriesDateTime    PatientName     PatientSex    Modality    Rov
            _____________________    ______________    ____________    __________    ________    ___

    s1    30-Apr-1993 11:27:24      {0×0 char}      "Anonymized"        ""          "CT"      512
    s2    30-Jan-1994 11:25:01      {0×0 char}      "Anonymized"        ""          "US"      430
    s3    03-Oct-2011 19:18:11      {0×0 char}      ""                  ""          "MR"      512
    s4    03-Oct-2011 19:18:11      {0×0 char}      ""                  ""          "MR"      512
```

Input Arguments

`directory` — Folder containing DICOM files

string scalar | character vector

Name of a folder containing DICOM files, specified as a string scalar or character vector.

Example: `details = dicomCollection(fullfile(matlabroot,"toolbox/images/imdata"))`

Data Types: `char` | `string`

`DICOMDIR` — DICOM directory file

character vector | string scalar

DICOM directory file, specified as a string scalar or character vector.

A DICOM directory file (DICOMDIR) is a special DICOM file that serves as a directory to a collection of DICOM files stored on removable media, such as CD- or DVD-ROMs. When devices write DICOM files to removable media, they typically write a DICOMDIR file on the disk to serve as a list of the disk contents.

You can specify `DICOMDIR` using its absolute path, relative path, or location on the MATLAB path.

Example: `details = dicomCollection(fullfile(matlabroot,"toolbox/images/imdata/DICOMDIR"))`

Data Types: `char` | `string`

`tf` — Include subfolders in search

`true` or `1` (default) | `false` or `0`

Include subfolders in search, specified as a logical `1` (`true`) or `0` (`false`). When set to `true`, the `dicomCollection` function recursively searches for DICOM files below `directory`. When set to `false`, the `dicomCollection` function only includes files within `directory` in the search.

Data Types: `logical`

Output Arguments

`collection` — Metadata from DICOM files

table

Metadata from DICOM files, returned as a table. The `dicomCollection` function aggregates the information by DICOM series.

See Also

`dicominfo` | `dicomread` | `dicomreadVolume` | **DICOM Browser**

Introduced in R2017b

dicomContours

Extract ROI data from DICOM-RT structure set

Description

The `dicomcontours` object extracts and stores region of interest (ROI) data from the metadata in DICOM-RT structure set files. You can use the object functions to add, delete, display, modify, and create masks from this ROI data. For more information, see Object Functions on page 1-703.

Creation

Syntax

```
contour = dicomContours(info)
```

Description

`contour = dicomContours(info)` creates a `dicomCountours` object that stores ROI data from the structure set and ROI contour modules of the specified DICOM metadata `info`.

Input Arguments

`info` — DICOM metadata
structure array

DICOM metadata, specified as a structure array. The metadata must correspond to a valid RT structure set file. You can use the `dicominfo` function to read metadata from DICOM-RT structure set files.

Data Types: `struct`

Properties

`ROIs` — ROI data
M-by-5 table

This property is read-only.

ROI data, returned as an *M*-by-5 table, where *M* is the number of ROI sequences defined in the DICOM metadata. The entries in each row of the table define an ROI sequence. The table has these columns.

Column	Description
`Number`	Identification number of the ROI, returned as a scalar integer. The number references the ROI number in the structure set ROI sequence.

Column	Description
`Name`	Name of the ROI, returned as a character vector. The name references the ROI name in the structure set ROI sequence.
`ContourData`	Points defining the contours of the ROI, returned as a cell array. The number of cells corresponds to the number of axial slices defined in the ROI. Each cell contains an *N*-by-3 matrix with rows of the form [*x y z*] that specify coordinate data for one axial slice. *N* is the number of points in the contour. These coordinates define contours in the patient-based coordinate system.
`GeometricType`	Geometric type of the contour, returned as a character vector. The value for geometric type can be any of these options: • `POINT` • `OPEN_PLANAR` • `OPEN_NONPLANAR` • `CLOSED_PLANAR`
`Color`	Display color of the ROI, returned as an RGB triplet [*r g b*] with values in the range [0, 255].

Data Types: `table`

Object Functions

addContour	Add ROI sequence to ROI data
convertToInfo	Write ROI data to DICOM metadata
createMask	Create volumetric mask from dicomContours object
deleteContour	Delete ROI sequence from ROI data
plotContour	Plot ROI contour data in DICOM-RT structure set

Examples

Extract ROI Data from DICOM RT-Structure Set

Read DICOM metadata from a DICOM-RT structure set file by using the `dicominfo` function.

```
info = dicominfo("rtstruct.dcm");
```

Extract ROI data from the structure set and ROI contour modules of the DICOM metadata. The output is a `dicomContours` object that stores the extracted ROI data.

```
contour = dicomContours(info);
```

Display the details of the `dicomContours` object.

```
contour
```

```
contour =
  dicomContours with properties:
```

```
    ROIs: [2x5 table]
```

Display the `ROIs` property of the `dicomContours` object. The `ROIs` property is a table that contains the extracted ROI data.

```
contour.ROIs
```

```
ans=2×5 table
    Number          Name           ContourData    GeometricType       Color
    ______    ________________    ___________    _____________    ___________

      1       {'Body_Contour' }    {90x1 cell}     {90x1 cell}     {3x1 double}
      2       {'Tumor_Contour'}    {21x1 cell}     {21x1 cell}     {3x1 double}
```

See Also

Functions
`dicominfo` | `dicomwrite`

Topics
“Add and Modify ROIs of DICOM-RT Contour Data”
“Create and Display 3-D Mask of DICOM-RT Contour Data”

Introduced in R2020a

addContour

Add ROI sequence to ROI data

Syntax

```
contourOut = addContour(contourIn,number,name,contourData,geometry)
contourOut = addContour( ___ ,color)
```

Description

`contourOut = addContour(contourIn,number,name,contourData,geometry)` adds a user-defined region of interest (ROI) sequence to the `ROIs` property of the `dicomContours` object `contourIn`. You can use the `convertToInfo` function to export the new ROI data to the structure set and ROI contour modules of the DICOM metadata.

`contourOut = addContour( ___ ,color)` specifies the color for the contour data added to the input `dicomContours` object, in addition to all input arguments from the previous syntax.

Examples

Add ROI Sequence to ROI Data

Add an ROI sequence to the ROI data extracted from the structure set and ROI contour modules of the DICOM metadata.

Read DICOM metadata from a DICOM-RT structure set file.

```
info = dicominfo("rtstruct.dcm");
```

Extract ROI data from the structure set and ROI contour modules of the DICOM metadata. The output is a `dicomContours` object that stores the extracted ROI data.

```
contourIn = dicomContours(info);
```

Display the `ROIs` property of the `dicomContours` object.

```
contourIn.ROIs
```

```
ans=2×5 table
    Number          Name           ContourData    GeometricType       Color
    ______    _________________    ___________    _____________    ___________

      1       {'Body_Contour' }    {90x1 cell}     {90x1 cell}     {3x1 double}
      2       {'Tumor_Contour'}    {21x1 cell}     {21x1 cell}     {3x1 double}
```

Load another set of ROI contour data into the workspace. The contour data contains the 3-D coordinates of the contours in the ROI.

```
load("contours")
```

To create an ROI sequence that contain the new ROI contour data, specify these attributes of the ROI sequence:

- ROI number
- User-defined name for the ROI
- Geometric type of the contours
- Color of the ROI

Assign a unique ROI number for the ROI sequence. The ROI name can be any user-defined name. Because all points in the new ROI contour data are coplanar, and the last point is connected to the first point, specify the geometric type as `"Closed_planar"`.

```
number = 3;
name = "Organ";
geometricType = "Closed_planar";
```

Specify the color of the ROI. If you do not specify a color, the default value for color in the `ROIs` property is set to `[]`.

```
color = [0; 127; 127];
```

Add the new ROI sequence to the `ROIs` property of `dicomContours` object. The output is also a `dicomContours` object containing the new ROI sequence as well as the original sequences.

```
contourOut = addContour(contourIn,number,name,contours,geometricType,color)
```

```
contourOut =
  dicomContours with properties:

    ROIs: [3x5 table]
```

Display the details of the new `dicomContours` object by viewing its `ROIs` property. You can use the `convertToInfo` function to export the modified ROI data as DICOM metadata.

```
contourOut.ROIs
```

```
ans=3×5 table
    Number          Name           ContourData    GeometricType       Color
    ______    ________________    ___________    _____________    ___________

      1       {'Body_Contour' }    {90x1 cell}     {90x1 cell}     {3x1 double}
      2       {'Tumor_Contour'}    {21x1 cell}     {21x1 cell}     {3x1 double}
      3       {'Organ'        }    {21x1 cell}     {21x1 cell}     {3x1 double}
```

Input Arguments

`contourIn` — Input ROI data

`dicomContours` object

Input ROI data, specified as a `dicomContours` object.

`number` — ROI number

integer scalar

ROI number, specified as an integer scalar. The ROI number references the user-defined identification number for the ROI.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`name` — User-defined name for ROI
character vector | string scalar

User-defined name for the ROI, specified as a character vector or string scalar.

Data Types: `char` | `string`

`contourData` — 3-D coordinates of contours in ROI
cell array of *N*-by-3 matrices

3-D coordinates of contours in the ROI, specified as a cell array of *N*-by-3 matrices. Each row is of the form [*x y z*], and defines a contour in the patient-based coordinate system.

Data Types: `cell`

`geometry` — Geometric type of the contour
`"Point"` | `"Open_planar"` | `"Open_nonplanar"` | `"Closed_planar"`

Geometric type of the contour, specified as one of these values.

- `"Point"`
- `"Open_Planar"`
- `"Open_nonplanar"`
- `"Closed_planar"`

Data Types: `char` | `string`

`color` — Display color for ROI
RGB triplet

Display color for the ROI, specified as an RGB triplet whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0, 255].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`contourOut` — Output ROI data
`dicomContours` object

Output ROI data, returned as a `dicomContours` object. The `ROIs` property of the output object contains both the input ROI and the user-defined ROI sequence.

See Also

Objects
`dicomContours`

Functions
`plotContour` | `createMask` | `deleteContour` | `convertToInfo`

Topics
“Add and Modify ROIs of DICOM-RT Contour Data”

Introduced in R2020a

convertToInfo

Write ROI data to DICOM metadata

Syntax

```
info = convertToInfo(contour)
```

Description

`info = convertToInfo(contour)` creates metadata for a DICOM-RT structure set file by using the region of interest (ROI) data in the `dicomContours` object `contour`. The function parses the `ROIs` property of `contour`, and then writes to the structure set and ROI contour modules of the original DICOM metadata used to create the `dicomContours` object.

Examples

Export ROI Data to DICOM-RT Structure Set

Add an ROI contour sequence to existing ROI data, and then export the new ROI data to the DICOM-RT structure set format.

Read the DICOM metadata from a DICOM-RT structure set file.

```
info = dicominfo("rtstruct.dcm");
```

Extract the ROI data from the structure set and ROI contour modules of the DICOM metadata. The output is a `dicomContours` object that stores the extracted ROI data.

```
contourIn = dicomContours(info);
```

Display the `ROIs` property of the `dicomContours` object.

```
contourIn.ROIs
```

```
ans=2×5 table
    Number          Name           ContourData    GeometricType       Color
    ______    ___________________    ___________    _____________    ___________

      1       {'Body_Contour' }    {90x1 cell}     {90x1 cell}     {3x1 double}
      2       {'Tumor_Contour'}    {21x1 cell}     {21x1 cell}     {3x1 double}
```

Load another ROI contour into the workspace. The contour data contains the 3-D coordinates of the contours in the ROI.

```
load("contours")
```

To create an ROI sequence that contain the new ROI contour data, specify these attributes of the ROI sequence:

- ROI number
- User-defined name for the ROI
- Geometric type of the contours

Assign a unique ROI number for the ROI sequence. The ROI name can be any user-defined name. Because all points in the new ROI contour data are coplanar, and the last point is connected to the first point, specify the geometric type as `"Closed_planar"`.

```
number = 3;
name = "Organ";
geometricType = "Closed_planar";
```

Add the new ROI sequence to the `ROIs` property of the `dicomContours` object. The output is also a `dicomContours` object containing the new ROI sequence as well as the original ones.

```
contourOut = addContour(contourIn,number,name,contours,geometricType);
contourOut.ROIs
```

```
ans=3×5 table
    Number          Name           ContourData    GeometricType       Color
    ______    ________________    ___________    _____________    ___________

      1       {'Body_Contour' }    {90x1 cell}     {90x1 cell}     {3x1 double}
      2       {'Tumor_Contour'}    {21x1 cell}     {21x1 cell}     {3x1 double}
      3       {'Organ'        }    {21x1 cell}     {21x1 cell}     {0x0 double}
```

Export the modified ROI data as DICOM metadata.

```
info = convertToInfo(contourOut);
```

Write the metadata to a DICOM-RT structure set file by using the `dicomwrite` function. If the DICOM image associated with the ROI contour data is not available, specify the first input argument value in the `dicomwrite` function as an empty array. Set the `CreateMode` parameter to `"copy"` to copy the metadata to a new DICOM-RT structure set file, `rtfile.dcm`. Setting the create mode to `"copy"` copies the metadata directly without verifying the metadata attributes.

```
dicomwrite([],"rtfile.dcm",info,"CreateMode","copy");
```

Input Arguments

`contour` — ROI data

`dicomContours` object

ROI data, specified as a `dicomContours` object.

Output Arguments

`info` — DICOM metadata

structure

DICOM metadata, returned as a structure.

Tips

- A `dicomContours` object stores the metadata of the file used in its creation as a hidden property. Therefore, the metadata attributes of the output structure `info` match the original DICOM file, with updated ROI data specified by `contour`.

See Also

Objects
`dicomContours`

Functions
`addContour` | `createMask` | `plotContour` | `deleteContour`

Topics
“Add and Modify ROIs of DICOM-RT Contour Data”

Introduced in R2020a

createMask

Create volumetric mask from `dicomContours` object

Syntax

```
BW = createMask(rtContours,roiIndex,spatial)
```

Description

`BW = createMask(rtContours,roiIndex,spatial)` creates the volumetric mask `BW`, a voxel representation of the specified ROI `roiIndex` from the `dicomContours` object `rtContours`. The `roiIndex` argument specifies which contour in `rtContours` to create a mask from. `spatial` specifies the location, resolution, and orientation of the 3-D data in world coordinates.

Examples

Create Mask of `dicomContours` Object

Read the metadata of a DICOM-RT structure set file.

```
info = dicominfo("rtstruct.dcm");
```

Construct a `dicomContours` object from the metadata.

```
rtContours = dicomContours(info);
```

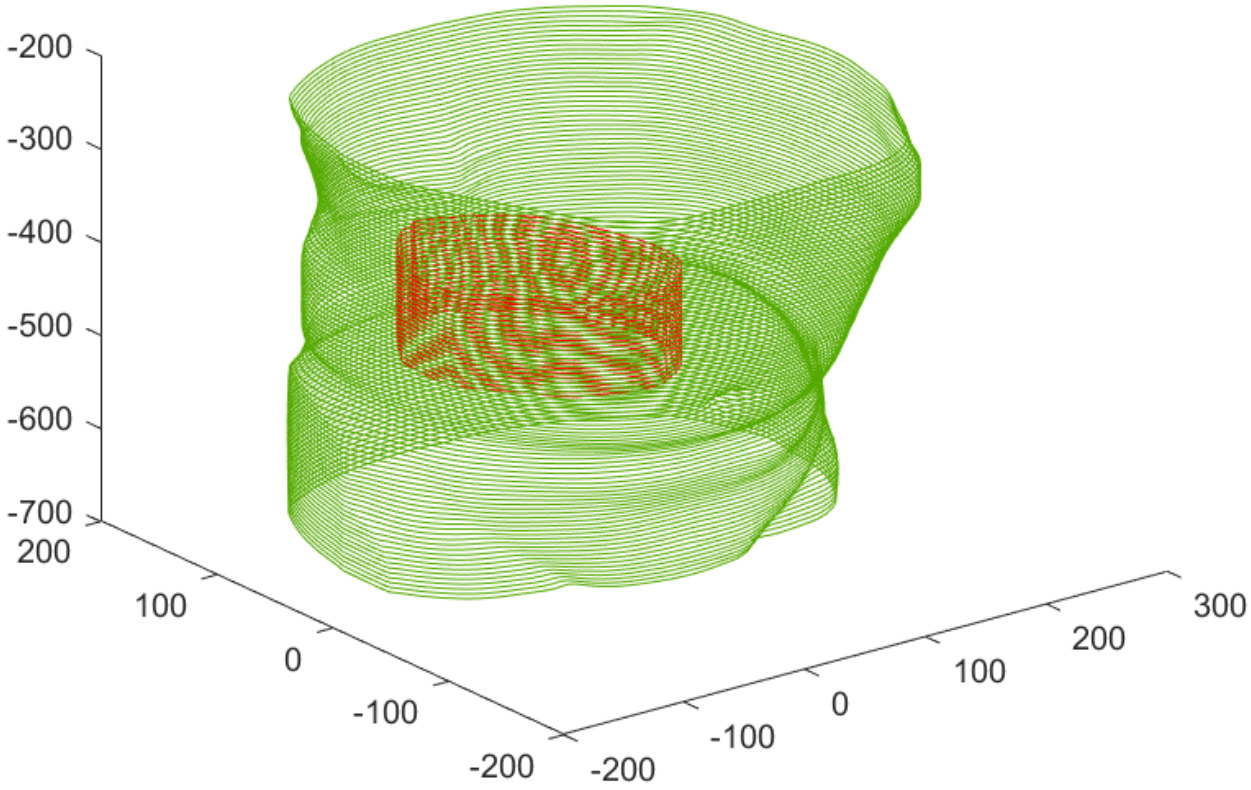
Display all of the ROI information as a table.

```
rtContours.ROIs
```

```
ans=2×5 table
    Number          Name           ContourData    GeometricType       Color
    ______    ________________    ___________    _____________    ___________

      1       {'Body_Contour' }    {90×1 cell}     {90×1 cell}     {3×1 double}
      2       {'Tumor_Contour'}    {21×1 cell}     {21×1 cell}     {3×1 double}
```

Plot the contours of all ROIs by using the `plotContours` object function. This function plots the contours in world coordinates.

```
plotContour(rtContours)
```



Create an `imref3d` object with the same world limits as the `plotContours` plot, so that the image is in the same space as the contours.

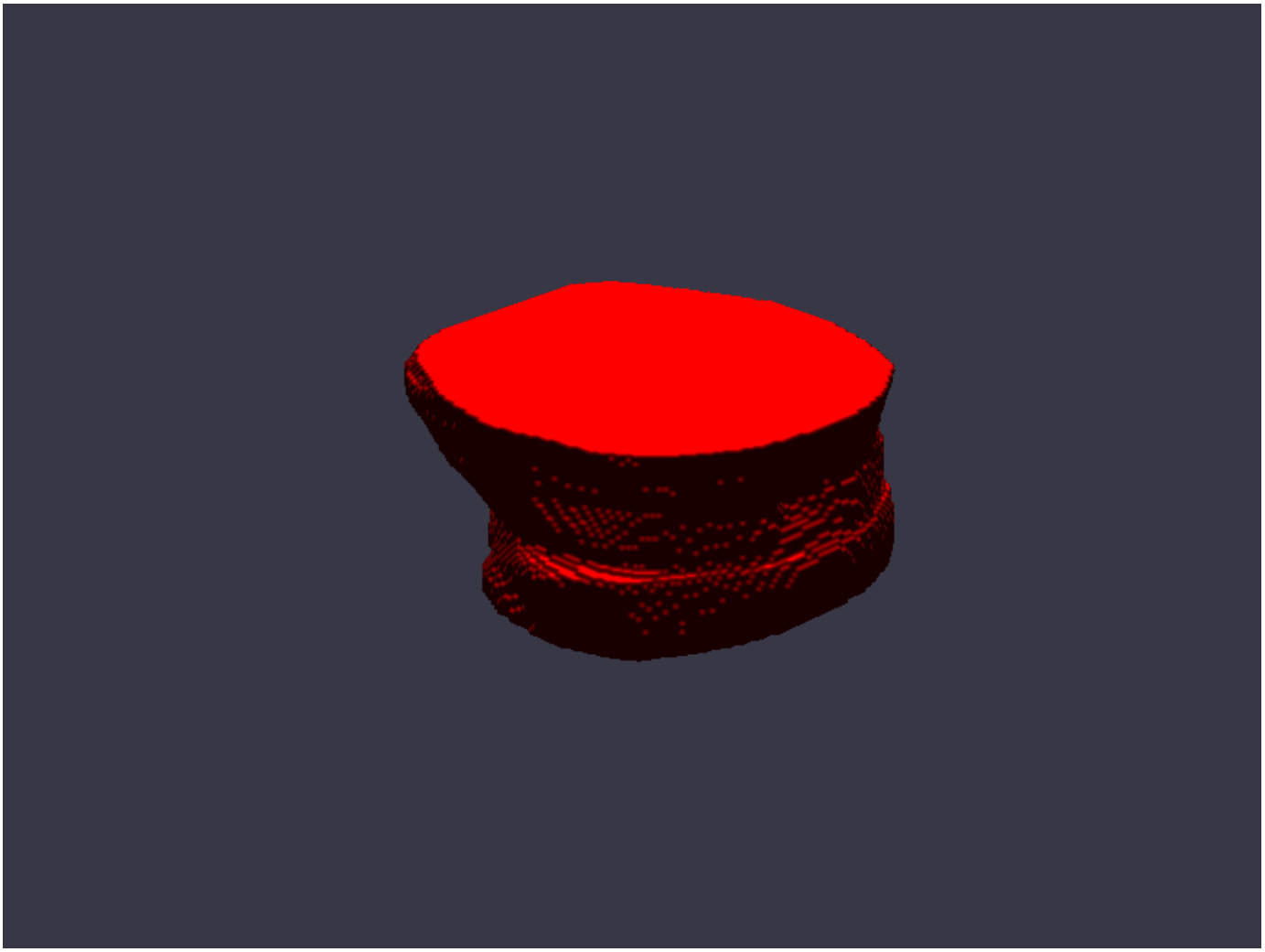
```
referenceInfo = imref3d([128 128 50],xlim,ylim,zlim);
```

Create a 3-D logical mask of the first contour, `'Body_Contour'`, from `rtContours`, with spatial referencing specified by the `imref3d` object.

```
contourIndex = 1;
rtMask = createMask(rtContours,contourIndex,referenceInfo);
```

View the mask using the **Volume Viewer**.

```
volshow(rtMask);
```



Input Arguments

`rtContours` — DICOM contours
`dicomContours` object

DICOM contours, specified as a `dicomContours` object.

Data Types: `dicomContours`

`roiIndex` — ROI in DICOM contours object
positive integer | character vector | string scalar

ROI in a DICOM contours object, specified as a positive integer, character vector, or string scalar. The value depends on which ROI identifier in the ROIs table of the `dicomContours` object you use.

ROI Identifier	Type	Example
`Number`	Row of the ROI in the ROIs table of the `rtContours` object, specified as a positive integer. `Number` is the first column in the ROIs table.	`rtMask = createMask(rtContours,1,spatialInfo)`
`Name`	Name of the ROI in the ROIs table of the `rtContours` object, specified as a character vector or string scalar. `Name` is the second column in the ROIs table.	`rtMask = createMask(rtContours,"Body_Contour",spatialInfo)`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

`spatial` — Spatial referencing information
structure | `imref3d` object

Spatial referencing information, specified as a structure or an `imref3d` object. You can use a structure returned by `dicomreadVolume` that contains the fields `PatientPosition`, `PixelSpacing`, and `PatientOrientation`. Spatial referencing information provides the location, resolution, and orientation of the 3-D coordinate data.

Output Arguments

`BW` — Volumetric mask
3-D logical array

Volumetric mask, returned as a 3-D logical array. The mask uses the intrinsic image coordinate system defined by `spatial`.

See Also

`dicomContours` | `addContour` | `dicominfo` | `dicomreadVolume` | `imref3d` | `plotContour` | `deleteContour` | `convertToInfo`

Topics
“Create and Display 3-D Mask of DICOM-RT Contour Data”

Introduced in R2020b

deleteContour

Delete ROI sequence from ROI data

Syntax

```
contourOut = deleteContour(contourIn,number)
```

Description

`contourOut = deleteContour(contourIn,number)` deletes one or more region of interest (ROI) sequences extracted from a DICOM-RT structure set file. Specify each sequence to delete by the ROI number `number`.

Use the `deleteContour` function to delete an ROI sequence from the `ROIs` property of a `dicomContours` object. You can then use the `convertToInfo` function to export the new ROI data to the structure set and ROI contour modules of DICOM metadata.

Examples

Delete ROI Sequence from ROI Data

Read the DICOM metadata from a DICOM-RT structure set file.

```
info = dicominfo("rtstruct.dcm");
```

Extract ROI data from the structure set and ROI contour modules of the DICOM metadata. The output is a `dicomContours` object that stores the extracted ROI data.

```
contourIn = dicomContours(info);
```

Display the `ROIs` property of the `dicomContours` object.

```
contourIn.ROIs
```

```
ans=2×5 table
    Number          Name           ContourData    GeometricType       Color
    ______    ________________    ___________    _____________    ___________

      1       {'Body_Contour' }    {90x1 cell}     {90x1 cell}     {3x1 double}
      2       {'Tumor_Contour'}    {21x1 cell}     {21x1 cell}     {3x1 double}
```

Delete the ROI sequence specified by ROI number 2.

```
contourOut = deleteContour(contourIn,2)
```

```
contourOut =
  dicomContours with properties:

    ROIs: [1x5 table]
```

Display the `ROIs` property of the output `dicomContours` object. You can use the `convertToInfo` function to export the modified ROI data to a DICOM-RT structure set file.

```
contourOut.ROIs
```

```
ans=1×5 table
    Number          Name          ContourData    GeometricType       Color
    ______    _________________    ___________    _____________    ___________

      1       {'Body_Contour'}    {90x1 cell}     {90x1 cell}     {3x1 double}
```

Input Arguments

`contourIn` — Input ROI data

`dicomContours` object

Input ROI data, specified as a `dicomContours` object.

`number` — ROI number

numeric scalar | numeric vector

ROI number, specified as a numeric scalar or vector. Specifying a vector of ROI numbers deletes multiple ROI sequences.

The ROI number is specified in the `Number` column of the table returned by the `ROIs` property of the `dicomContours` object.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`contourOut` — Output ROI data

`dicomContours` object

Output ROI data, returned as a `dicomContours` object.

See Also

Objects

`dicomContours`

Functions

`addContour` | `createMask` | `plotContour` | `convertToInfo`

Topics

“Add and Modify ROIs of DICOM-RT Contour Data”

Introduced in R2020a

plotContour

Plot ROI contour data in DICOM-RT structure set

Syntax

```
plotContour(contour)
plotContour(contour,number)
plotContour( ___ ,ax)
h = plotContour( ___ )
```

Description

`plotContour(contour)` plots one or more region of interest (ROI) sequences stored in the `dicomContours` object `contour`.

`plotContour(contour,number)` plots only the ROI contour data with the specified ROI number `number`.

`plotContour( ___ ,ax)` plots ROI contour data in the axes specified by `ax`, in addition to any combination of input arguments from previous syntaxes.

`h = plotContour( ___ )` returns the graphics object handles for the plot. You can use `h` to query and modify the properties of the plot. `h` is a group object. For more information on group object properties, see Group Properties.

Examples

Plot ROI Contour Data from DICOM-RT Structure Set

Read the DICOM metadata from a DICOM-RT structure set file.

```
info = dicominfo("rtstruct.dcm");
```

Extract the ROI data from the structure set and ROI contour modules of the DICOM metadata. The output is a `dicomContours` object that stores the extracted ROI data.

```
contour = dicomContours(info);
```

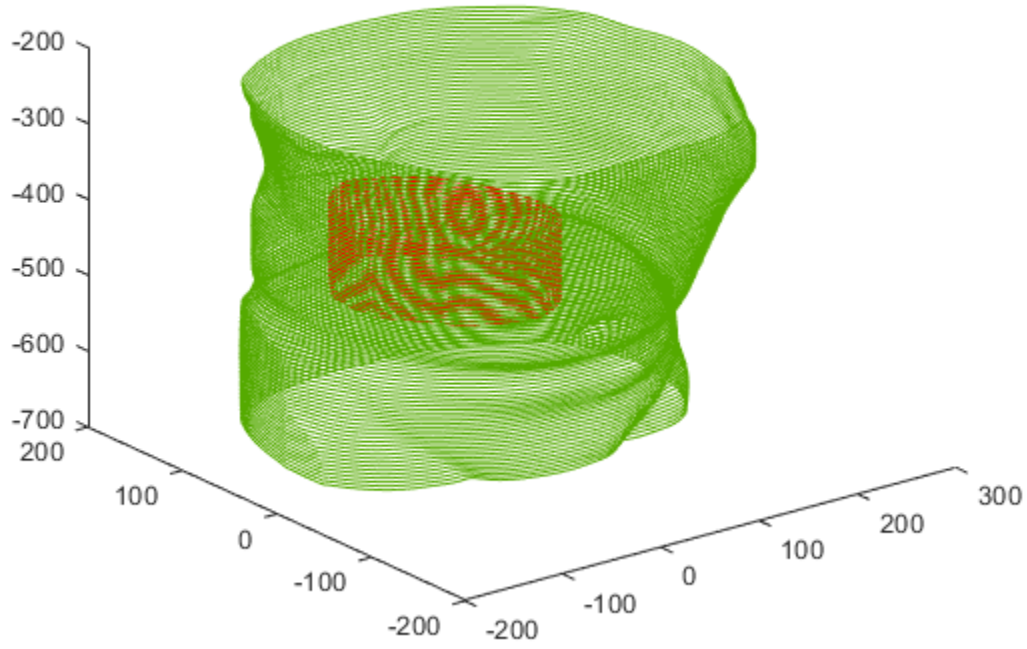
Display the `ROIs` property of the `dicomContours` object.

```
contour.ROIs
```

```
ans=2×5 table
    Number          Name           ContourData    GeometricType       Color
    ______    ________________    ___________    _____________    ___________

      1       {'Body_Contour' }    {90x1 cell}     {90x1 cell}     {3x1 double}
      2       {'Tumor_Contour'}    {21x1 cell}     {21x1 cell}     {3x1 double}
```

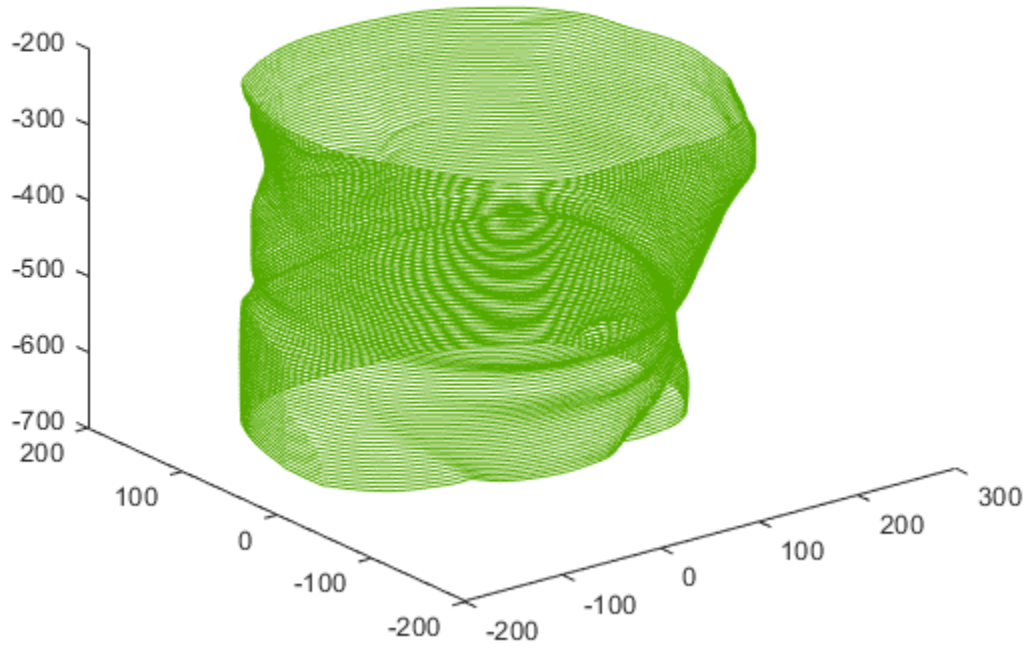
Plot all of the ROI contour data from the object.

```
figure
plotContour(contour)
```



You can also plot a specific ROI contour by using its ROI number. Plot only the ROI contour data specified by ROI number 1.

```
figure
plotContour(contour,1)
```



Specify Axes for Plotting ROI Contour Data

Read the DICOM metadata from a DICOM-RT structure set file.

```
info = dicominfo("rtstruct.dcm");
```

Extract the ROI data from the structure set and ROI contour modules of the DICOM metadata. The output is a `dicomContours` object that stores the extracted ROI data.

```
contour = dicomContours(info);
```

Create a 2-by-2 tiled chart layout to display multiple plots in a figure window.

```
figure("Position",[1 1 700 700])
tiledlayout(2,2)
```

Create an axes object by using the `nexttile` function. The axes span across the first two columns of the tiled chart layout. Plot all of the ROI contour data on these axes.

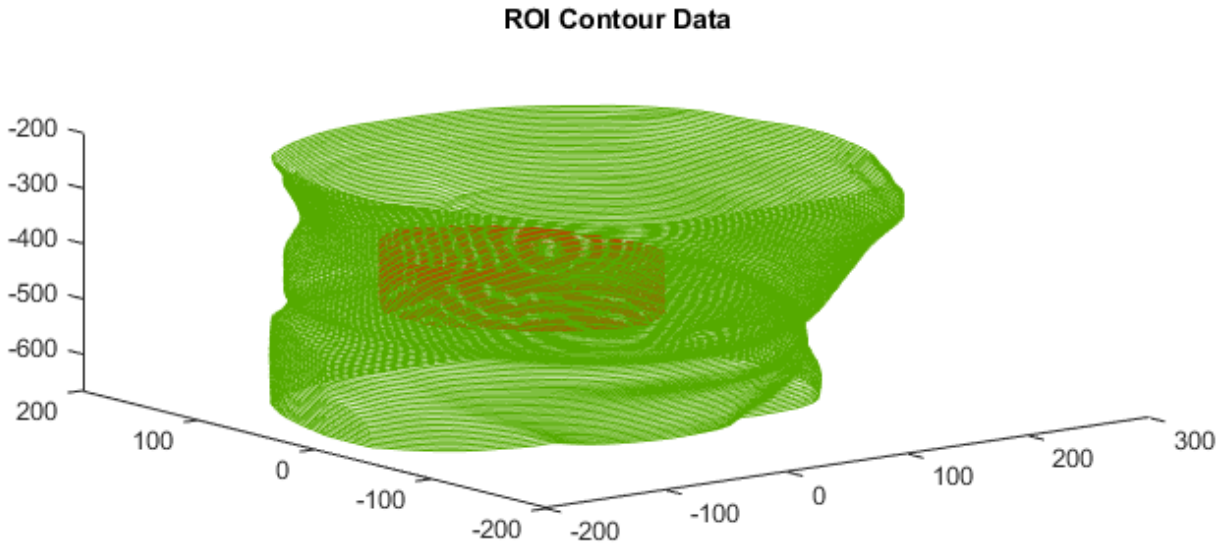
```
ax1 = nexttile(1,[1 2]);
plotContour(contour,ax1)
title("ROI Contour Data")
```

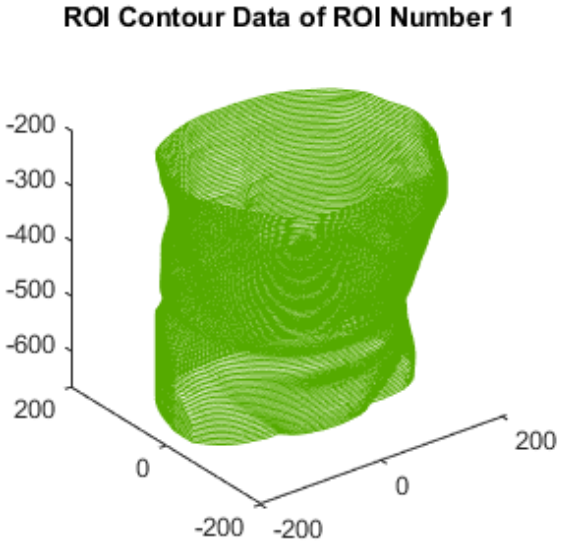
Create a second axes object and plot only the ROI contour data specified by ROI number `1`.

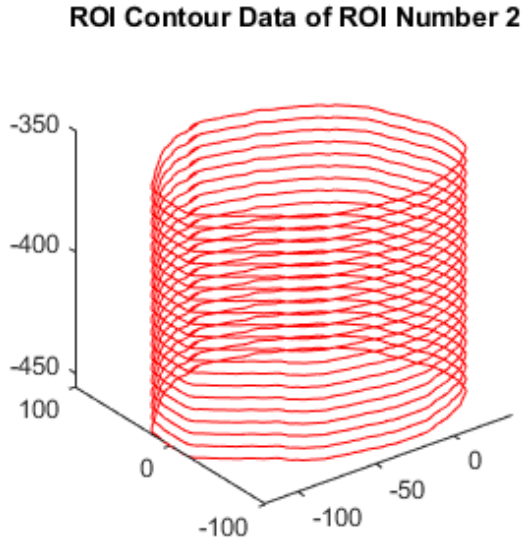
```
ax2 = nexttile;
plotContour(contour,1,ax2)
title("ROI Contour Data of ROI Number 1")
```

Create a third axes object and plot only the ROI contour data specified by ROI number 2.

```
ax3 = nexttile;
plotContour(contour,2,ax3)
title("ROI Contour Data of ROI Number 2")
```







Add Text Description to ROI Contour Data Plot

Read the DICOM metadata from a DICOM-RT structure set file.

```
info = dicominfo("rtstruct.dcm");
```

Extract the ROI data from the structure set and ROI contour modules of the DICOM metadata. The output is a `dicomContours` object that stores the extracted ROI data.

```
contour = dicomContours(info);
```

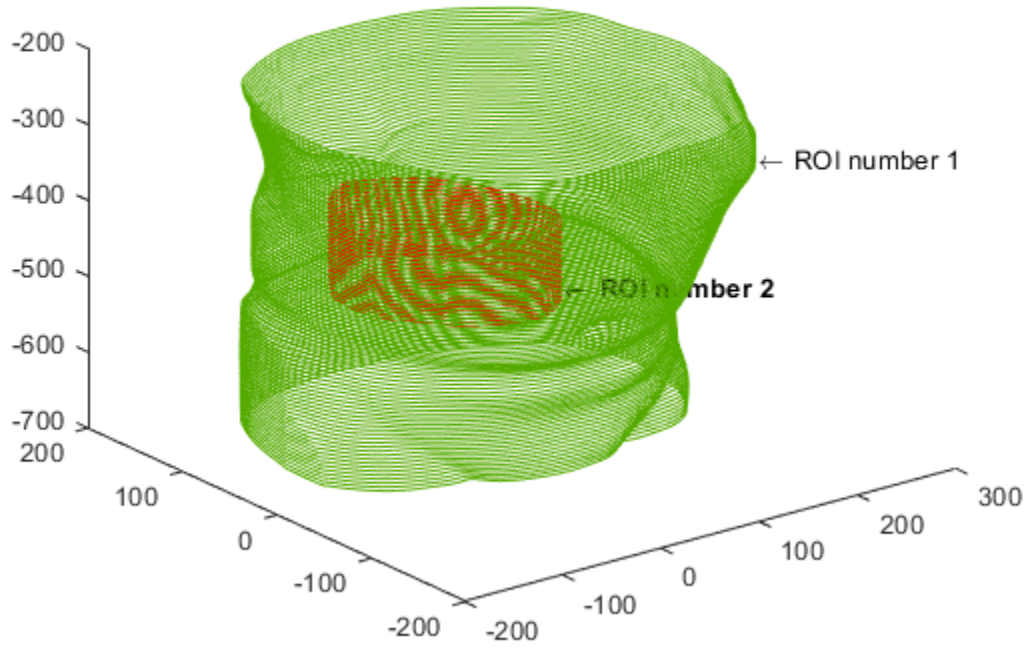
Plot the ROI contour data and get the parent axes. The returned parent axes is an `hggroup` object with separate handles for each ROI contour plot.

```
h = plotContour(contour)
```

```
h =
  2x1 Group array:

  Group    (Body_Contour)
  Group    (Tumor_Contour)
```

Add text descriptions for each ROI contour plot by using the returned handles.

```
text(290,0,-400,"\leftarrow ROI number 1","Parent",h(1))
text(90,0,-500,"\leftarrow ROI number 2","FontWeight","Bold","Parent",h(2))
```



Input Arguments

`contour` — ROI data
`dicomContours` object

ROI data, specified as a `dicomContours` object.

`number` — ROI number
numeric scalar | numeric vector

ROI number, specified as a numeric scalar or vector. Specifying a vector plots multiple contour sequences on the same axes.

The ROI number is specified in the `Number` column of the table returned by the `ROIs` property of the `dicomContours` object.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ax` — Target axes
handle object

Target axes, specified as a handle object returned by `axes` or `gca`.

Output Arguments

`h` — Graphics object handle
`hggroup` object | array of `hggroup` objects

Graphics object handle, returned as an `hggroup` object or an array of `hggroup` objects. If you specify `number` as a vector of ROIs, `h` is an array of `hggroup` objects with each element corresponding to one ROI.

If you specify `ax`, `h` is a child of the axes `ax`. Otherwise, `h` is a child of the current axes.

See Also

Objects
`dicomContours` | `hggroup`

Functions
`addContour` | `createMask` | `deleteContour` | `convertToInfo`

Topics
“Add and Modify ROIs of DICOM-RT Contour Data”

Introduced in R2020a

dicomdict

Get or set active DICOM data dictionary

Syntax

```
dictionaryOut = dicomdict("get")
dicomdict("set",dictionaryIn)
dicomdict("factory")
```

Description

`dictionaryOut = dicomdict("get")` returns the name of the active Digital Imaging and Communications in Medicine (DICOM) data dictionary file.

`dicomdict("set",dictionaryIn)` sets the file specified by input `dictionaryIn` as the active DICOM data dictionary. If the file is not found on the specified path, the function returns an error.

`dicomdict("factory")` restores the active DICOM data dictionary to its default value. The default value is a file in the MATLAB path:

`fullfile(matlabroot,"toolbox","images","iptformats","dicom-dict.txt")`

Examples

Get and Set Active DICOM Data Dictionary

Find the default active DICOM data dictionary.

```
dictionaryOut = dicomdict("get")
```

```
dictionaryOut =
'B:\matlab\toolbox\images\iptformats\dicom-dict.txt'
```

Specify the path to a new file to set as the active DICOM data dictionary.

```
dictionaryIn = "dicomdictnew.txt";
dicomdict("set",dictionaryIn)
```

Check if the active DICOM data dictionary has been updated to `dicomdictnew.txt`.

```
dictionaryOut1 = dicomdict("get");
```

Reset the active DICOM data dictionary to the default value. Verify the data dictionary path.

```
dicomdict("factory")
dictionaryOut2 = dicomdict("get")
```

```
dictionaryOut2 =
'B:\matlab\toolbox\images\iptformats\dicom-dict.txt'
```

Input Arguments

`dictionaryIn` — DICOM data dictionary file

character vector | string scalar

DICOM data dictionary file, specified as a character vector or string scalar. Data dictionary files must be of type `.txt`. DICOM-related functions use this dictionary unless you specify a different dictionary as input to function calls.

Data Types: `char` | `string`

Output Arguments

`dictionaryOut` — Active DICOM data dictionary file

`fullfile(matlabroot,"toolbox","images","iptformats","dicom-dict.txt")` (default) | character vector | string scalar

Active DICOM data dictionary file, returned as a character vector or string scalar. Data dictionary files are of type `.txt`. The default value is:

```
fullfile(matlabroot,"toolbox","images","iptformats","dicom-dict.txt")
```

See Also

`dicomanon` | `dicominfo` | `dicomdisp` | `dicomwrite` | `dicomlookup` | `dicomread` | `dicomuid`

Introduced before R2006a

dicomdisp

Display DICOM file structure

Syntax

```
dicomdisp(filename)
dicomdisp(filename,Name,Value)
```

Description

`dicomdisp(filename)` reads the metadata from the compliant DICOM file `filename` and displays the metadata at the command prompt. `dicomdisp` can be helpful when debugging issues with DICOM files.

`dicomdisp(filename,Name,Value)` reads the metadata using name-value arguments to control aspects of the operation.

Examples

Display Metadata from DICOM File

Display the metadata in a DICOM file.

```
dicomdisp("CT-MONO2-16-ankle.dcm")
```

```
File: B:\matlab\toolbox\images\imdata\CT-MONO2-16-ankle.dcm (525436 bytes)
Read on an IEEE little-endian machine.
File begins with group 0002 metadata at byte 132.
Transfer syntax: 1.2.840.10008.1.2 (Implicit VR Little Endian).
DICOM Information object: 1.2.840.10008.5.1.4.1.1.7 (Secondary Capture Image Storage).

Location Level     Tag     VR       Size         Name                              Data
-------------------------------------------------------------------------------------------
0000132    0   (0002,0000) UL          4 bytes - FileMetaInformationGroupLength   *Binary*
0000144    0   (0002,0001) OB          2 bytes - FileMetaInformationVersion       *Binary*
0000158    0   (0002,0002) UI         26 bytes - MediaStorageSOPClassUID          [1.2.840.10008
0000192    0   (0002,0003) UI         50 bytes - MediaStorageSOPInstanceUID       [1.2.840.113619
0000250    0   (0002,0010) UI         18 bytes - TransferSyntaxUID                [1.2.840.10008
0000276    0   (0002,0012) UI         18 bytes - ImplementationClassUID           [1.2.840.113619
0000302    0   (0002,0013) SH          6 bytes - ImplementationVersionName        [1_2_5 ]
0000316    0   (0002,0016) AE         12 bytes - SourceApplicationEntityTitle     [CTN_STORAGE ]
0000336    0   (0008,0000) UL          4 bytes - IdentifyingGroupLength           *Binary*
0000348    0   (0008,0008) CS         20 bytes - ImageType                        [DERIVED\SECOND
0000376    0   (0008,0016) UI         26 bytes - SOPClassUID                      [1.2.840.10008
0000410    0   (0008,0018) UI         50 bytes - SOPInstanceUID                   [1.2.840.113619
0000468    0   (0008,0020) DA         10 bytes - StudyDate                        [1993.04.30]
0000486    0   (0008,0021) DA         10 bytes - SeriesDate                       [1993.04.30]
0000504    0   (0008,0023) DA         10 bytes - ContentDate                      [1993.04.30]
0000522    0   (0008,0030) TM          8 bytes - StudyTime                        [11:27:24]
0000538    0   (0008,0031) TM          8 bytes - SeriesTime                       [11:27:24]
0000554    0   (0008,0033) TM          8 bytes - ContentTime                      [11:27:24]
```

```
0000570    0   (0008,0060) CS          2 bytes - Modality                          [CT]
0000580    0   (0008,0064) CS          4 bytes - ConversionType                    [WSD ]
0000592    0   (0008,0070) LO         18 bytes - Manufacturer                      [GE MEDICAL SYS
0000618    0   (0008,0080) LO         18 bytes - InstitutionName                   [JFK IMAGING CE
0000644    0   (0008,0090) PN         10 bytes - ReferringPhysicianName            [Anonymized]
0000662    0   (0008,1010) SH          8 bytes - StationName                       [CT01OC0 ]
0000678    0   (0008,1030) LO          8 bytes - StudyDescription                  [RT ANKLE]
0000694    0   (0008,1060) PN         10 bytes - NameOfPhysiciansReadingStudy      [Anonymized]
0000712    0   (0008,1070) PN         10 bytes - OperatorsName                    [Anonymized]
0000730    0   (0008,1090) LO         12 bytes - ManufacturerModelName            [GENESIS_ZEUS]
0000750    0   (0010,0000) UL          4 bytes - PatientGroupLength               *Binary*
0000762    0   (0010,0010) PN         10 bytes - PatientName                      [Anonymized]
0000780    0   (0018,0000) UL          4 bytes - AcquisitionGroupLength           *Binary*
0000792    0   (0018,1020) LO          2 bytes - SoftwareVersions                 [03]
0000802    0   (0020,0000) UL          4 bytes - RelationshipGroupLength          *Binary*
0000814    0   (0020,000D) UI         48 bytes - StudyInstanceUID                 [1.2.840.113619
0000870    0   (0020,000E) UI         48 bytes - SeriesInstanceUID                [1.2.840.113619
0000926    0   (0020,0011) IS          4 bytes - SeriesNumber                     [365 ]
0000938    0   (0028,0013) IS          2 bytes - InstanceNumber                   [1 ]
0000948    0   (0028,0000) UL          4 bytes - ImagePresentationGroupLength     *Binary*
0000960    0   (0028,0002) US          2 bytes - SamplesPerPixel                  *Binary*
0000970    0   (0028,0004) CS         12 bytes - PhotometricInterpretation        [MONOCHROME2 ]
0000990    0   (0028,0010) US          2 bytes - Rows                             *Binary*
0001000    0   (0028,0011) US          2 bytes - Columns                          *Binary*
0001010    0   (0028,0100) US          2 bytes - BitsAllocated                    *Binary*
0001020    0   (0028,0101) US          2 bytes - BitsStored                       *Binary*
0001030    0   (0028,0102) US          2 bytes - HighBit                          *Binary*
0001040    0   (0028,0103) US          2 bytes - PixelRepresentation              *Binary*
0001050    0   (0028,0106) US          2 bytes - SmallestImagePixelValue          *Binary*
0001060    0   (0028,0120) US          2 bytes - PixelPaddingValue                *Binary*
0001070    0   (0028,1050) DS          4 bytes - WindowCenter                     [1024]
0001082    0   (0028,1051) DS          4 bytes - WindowWidth                      [4095]
0001094    0   (0028,1052) DS          6 bytes - RescaleIntercept                 [-1024 ]
0001108    0   (0028,1053) DS          2 bytes - RescaleSlope                     [1 ]
0001118    0   (0028,1054) LO          2 bytes - RescaleType                      [US]
0001128    0   (7FE0,0000) UL          4 bytes - PixelDataGroupLength             *Binary*
0001140    0   (7FE0,0010) OB     524288 bytes - PixelData                        []
```

Input Arguments

`filename` — Name of DICOM file

character vector | string scalar

Name of DICOM file, specified as a string scalar or character vector .

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `dicomdisp("CT-MONO2-16-ankle.dcm",UseVRHeuristic=false)` reads the metadata from the DICOM file without using a heuristic.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `dicomdisp("CT-MONO2-16-ankle.dcm","UseVRHeuristic",false)` reads the metadata from the DICOM file without using a heuristic.

`dictionary` — Name of DICOM data dictionary
`'dicom-dict.txt'` (default) | string scalar | character vector

Name of DICOM data dictionary, specified as a string scalar or character vector. When specified, `dicomdisp` uses the data dictionary to read the DICOM file. The data dictionary file must be on the MATLAB search path.

Data Types: `char` | `string`

`UseVRHeuristic` — Read noncompliant DICOM files that switch VR modes incorrectly
`true` or `1` (default) | `false` or `0`

Read noncompliant DICOM files that switch value representation (VR) modes incorrectly, specified as a logical `1` (`true`) or `0` (`false`).

When set to `true`, `dicomdisp` uses a heuristic to help read certain noncompliant DICOM files that switch VR modes incorrectly. `dicomdisp` displays a warning if it uses this heuristic. If this heuristic is enabled, a small number of compliant files are not read correctly. Set `UseVRHeuristic` to `false` to read these compliant files.

Data Types: `logical`

See Also

`dicominfo` | `dicomread` | `dicomwrite` | `dicomdict` | `dicomuid` | `dicomanon` | `dicomlookup`

Topics
“Specify Value Representation”

Introduced in R2015a

dicomfind

Find location and value of target attribute in DICOM metadata

Syntax

```
attributeinfo = dicomfind(info,attribute)
attributeinfo = dicomfind(filename,attribute)
```

Description

`attributeinfo = dicomfind(info,attribute)` finds the location and value of the DICOM metadata field `attribute` of the DICOM metadata structure `info`. To extract the DICOM metadata structure from a DICOM file, use `dicominfo`. You can use `dicomfind` to help find values for metadata items that are deeply nested in the `info` structure.

`attributeinfo = dicomfind(filename,attribute)` finds the value of the metadata element `attribute` of the DICOM file `filename`.

Examples

Find Location and Value of DICOM Metadata in Info Structure

Create a DICOM metadata structure, `info`, by using the `dicominfo` function.

```
info = dicominfo("rtstruct.dcm");
```

Find the value and location in the `info` structure of the `ROINumber` metadata field. The `dicomfind` function returns the results in a table with two columns: `Location` and `Value`.

```
ROINumber = dicomfind(info,"ROINumber")
```

```
ROINumber=2×2 table
                       Location                          Value
    _____________________________________________    _____

    {'StructureSetROISequence.Item_1.ROINumber'}    {[1]}
    {'StructureSetROISequence.Item_2.ROINumber'}    {[2]}
```

Find Location and Value of DICOM Metadata in DICOM File

Find the value of the `ROINumber` metadata field and its location in the DICOM metadata structure. Specify the name of the metadata field and the name of the DICOM file. The function returns the results in a table with two columns: `Location` and `Value`.

```
ROINumber = dicomfind("rtstruct.dcm","ROINumber")
```

```
ROINumber=2×2 table
                       Location                          Value
```

```
    ____________________________________________    _____

    {'StructureSetROISequence.Item_1.ROINumber'}    {[1]}
    {'StructureSetROISequence.Item_2.ROINumber'}    {[2]}
```

Input Arguments

`info` — DICOM metadata

structure

DICOM metadata, specified as a structure. You can extract the DICOM metadata structure from a DICOM file by using the `dicominfo` function.

Data Types: `struct`

`attribute` — Name of DICOM metadata field

string scalar | character vector

Name of the DICOM metadata field, specified as a string scalar or character vector. The spelling and capitalization of `attribute` must match the full name of a metadata field in the input DICOM metadata structure, `info`.

Data Types: `string` | `char`

`filename` — Name of DICOM file

string scalar | character vector

Name of the DICOM file, specified as a string scalar or character vector.

Example: `"rtstruct.dcm"`

Data Types: `string` | `char`

Output Arguments

`attributeinfo` — Location and value of specified DICOM attribute

table

Location and value of the specified DICOM attribute, returned as a table with two columns: `Location` and `Value`. The `Location` column lists the attribute name using dot notation, providing its position within the nested DICOM metadata structure. The `Value` column lists the value assigned for each instance of the attribute within the DICOM metadata structure.

See Also

`dicomanon` | `dicomupdate` | `dicomread` | `dicomreadVolume` | `dicomdict` | `dicomdisp` | `dicominfo` | `dicomlookup` | `dicomwrite` | `dicomuid`

Introduced in R2021b

dicominfo

Read metadata from DICOM message

Syntax

```
info = dicominfo(filename)
info = dicominfo(filename,"dictionary",D)
info = dicominfo( ___ ,Name,Value)
```

Description

`info = dicominfo(filename)` reads the metadata from the compliant Digital Imaging and Communications in Medicine (DICOM) file or Digital Imaging and Communication in Security (DICOS) file, `filename`.

`info = dicominfo(filename,"dictionary",D)` reads the DICOM message by using the data dictionary file, `D`.

`info = dicominfo( ___ ,Name,Value)` specifies options using one or more name-value arguments in addition to any combination of input arguments from previous syntaxes.

Examples

Read Metadata from DICOM Message

Read metadata from a DICOM message.

```
info = dicominfo("CT-MONO2-16-ankle.dcm")
```

```
info = struct with fields:
                          Filename: 'B:\matlab\toolbox\images\imdata\CT-MONO2-16-ankle.dcm'
                       FileModDate: '18-Dec-2000 12:06:43'
                          FileSize: 525436
                            Format: 'DICOM'
                     FormatVersion: 3
                             Width: 512
                            Height: 512
                          BitDepth: 16
                         ColorType: 'grayscale'
    FileMetaInformationGroupLength: 192
        FileMetaInformationVersion: [2×1 uint8]
           MediaStorageSOPClassUID: '1.2.840.10008.5.1.4.1.1.7'
        MediaStorageSOPInstanceUID: '1.2.840.113619.2.1.2411.1031152382.365.1.736169244'
                  TransferSyntaxUID: '1.2.840.10008.1.2'
            ImplementationClassUID: '1.2.840.113619.6.5'
         ImplementationVersionName: '1_2_5'
      SourceApplicationEntityTitle: 'CTN_STORAGE'
            IdentifyingGroupLength: 414
                        ImageType: 'DERIVED\SECONDARY\3D'
                     SOPClassUID: '1.2.840.10008.5.1.4.1.1.7'
                  SOPInstanceUID: '1.2.840.113619.2.1.2411.1031152382.365.1.736169244'
```

```
                     StudyDate: '1993.04.30'
                    SeriesDate: '1993.04.30'
                   ContentDate: '1993.04.30'
                     StudyTime: '11:27:24'
                    SeriesTime: '11:27:24'
                   ContentTime: '11:27:24'
                      Modality: 'CT'
                ConversionType: 'WSD'
                  Manufacturer: 'GE MEDICAL SYSTEMS'
             InstitutionName: 'JFK IMAGING CENTER'
      ReferringPhysicianName: [1×1 struct]
                  StationName: 'CT01OC0'
            StudyDescription: 'RT ANKLE'
NameOfPhysiciansReadingStudy: [1×1 struct]
               OperatorsName: [1×1 struct]
       ManufacturerModelName: 'GENESIS_ZEUS'
          PatientGroupLength: 18
                 PatientName: [1×1 struct]
      AcquisitionGroupLength: 10
            SoftwareVersions: '03'
     RelationshipGroupLength: 134
            StudyInstanceUID: '1.2.840.113619.2.1.1.322987881.621.736170080.681'
           SeriesInstanceUID: '1.2.840.113619.2.1.2411.1031152382.365.736169244'
                SeriesNumber: 365
              InstanceNumber: 1
ImagePresentationGroupLength: 168
             SamplesPerPixel: 1
   PhotometricInterpretation: 'MONOCHROME2'
                        Rows: 512
                     Columns: 512
               BitsAllocated: 16
                  BitsStored: 16
                     HighBit: 15
         PixelRepresentation: 1
     SmallestImagePixelValue: 0
           PixelPaddingValue: 0
                WindowCenter: 1024
                 WindowWidth: 4095
           RescaleIntercept: -1024
                RescaleSlope: 1
                 RescaleType: 'US'
        PixelDataGroupLength: 524296
```

Input Arguments

`filename` — Name of DICOM file

character vector | string scalar

Name of the DICOM file, specified as a character vector or string scalar.

Data Types: `char` | `string`

`D` — Data dictionary file

`'dicom-dict.mat'` (default) | character vector | string scalar

Data dictionary file, specified as a character vector or string scalar. The file in `D` must be on the MATLAB search path.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `dicominfo("CT-MONO2-16-ankle.dcm",UseVRHeuristic=false)` reads the metadata from the DICOM file without using a heuristic.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `dicominfo("CT-MONO2-16-ankle.dcm","UseVRHeuristic",false)` reads the metadata from the DICOM file without using a heuristic.

`UseVRHeuristic` — Read noncompliant DICOM files that switch VR modes incorrectly
`true` or `1` (default) | `false` or `0`

Read noncompliant DICOM files that switch value representation (VR) modes incorrectly, specified as a logical `1` (`true`) or `0` (`false`).

When set to `true`, `dicominfo` uses a heuristic to help read certain noncompliant DICOM files that switch VR modes incorrectly. `dicominfo` displays a warning if it uses this heuristic. If this heuristic is enabled, a small number of compliant files are not read correctly. Set `UseVRHeuristic` to `false` to read these compliant files.

Data Types: `logical`

`UseDictionaryVR` — Conform data types to data dictionary
`false` or `0` (default) | `true` or `1`

Conform data types to the data dictionary, specified as a logical `0` (`false`) or `1` (`true`). If this argument is specified as `true`, the `info` output structure uses the data types in the data dictionary, regardless of what information is present in the file. If specified as `false`, the `info` output uses the VR codes of the file, even if they differ from the data dictionary. Because the file contents and the data dictionary almost always agree, specifying this argument as `true` is usually unnecessary.

However, if the file and the data dictionary disagree and `UseDictionaryVR` is set to `false`, then `dicominfo` issues a warning, and you can experience errors when passing `info` to `dicomwrite`. To resolve these errors, specify `UseDictionaryVR` as `true` to use the VR codes from the data dictionary.

Data Types: `logical`

Output Arguments

`info` — DICOM metadata
structure

DICOM metadata, returned as a structure.

See Also

dicomanon | dicomdict | dicomdisp | dicomwrite | dicomlookup | dicomread | dicomuid

Introduced before R2006a

dicomlookup

Find attribute in DICOM data dictionary

Syntax

```
nameOut = dicomlookup(group,element)
[groupOut,elementOut] = dicomlookup(name)
```

Description

`nameOut = dicomlookup(group,element)` looks into the current DICOM data dictionary for the attribute with the specified group tag `group` and element tag `element`. `dicomlookup` returns the name of the attribute.

`[groupOut,elementOut] = dicomlookup(name)` looks into the current DICOM data dictionary for the attribute specified by `name` and returns the group and element tags associated with the attribute.

Examples

Find DICOM Attributes in Data Dictionary

Find the names of DICOM attributes using their tags.

```
name1 = dicomlookup("7FE0","0010")
```

```
name1 =
'PixelData'
```

```
name2 = dicomlookup(40,4)
```

```
name2 =
'PhotometricInterpretation'
```

Look up the group and element of a DICOM attribute by the attribute name.

```
[group,element] = dicomlookup("TransferSyntaxUID")
```

```
group = 2
```

```
element = 16
```

Read the metadata of a DICOM file. Access the value of a metadata attribute in the file by specifying its group and element tag. This returns the same value even if the data dictionary changes.

```
metadata = dicominfo("CT-MONO2-16-ankle.dcm");
metadata.(dicomlookup("0028","0004"))
```

```
ans =
'MONOCHROME2'
```

Input Arguments

`group` — DICOM group tag

positive integer decimal | character vector | string scalar

DICOM group tag, specified as a positive integer decimal number or a character vector or string scalar that contains a hexadecimal value. `element` and `group` must use the same type of value:

- If `group` is a positive integer decimal number, then `element` must also be a positive integer decimal number.
- If `group` is a character vector or string scalar that contains a hexadecimal value, then `element` must be either a character vector or a string scalar that contains a hexadecimal value.

Example: `40`

Example: `'7FE0'` or `"7FE0"`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

`element` — DICOM element tag

positive integer decimal | character vector | string scalar

DICOM element tag, specified as a positive integer decimal number or a character vector or string scalar that contains a hexadecimal value. `element` and `group` must use the same type of value:

- If `group` is a positive integer decimal number, then `element` must also be a positive integer decimal number.
- If `group` is a character vector or string scalar that contains a hexadecimal value, then `element` must be either a character vector or a string scalar that contains a hexadecimal value.

Example: `4`

Example: `'0010'` or `"0010"`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

`name` — DICOM attribute name

character vector | string scalar

DICOM attribute name, specified as a character vector or string scalar.

Example: `'PhotometricInterpretation'` or `"PhotometricInterpretation"`

Data Types: `char` | `string`

Output Arguments

`groupOut` — Returned DICOM group tag

positive integer decimal

Returned DICOM group tag, returned as a positive integer decimal number.

Data Types: `double`

`elementOut` — Returned DICOM element tag
positive integer decimal

Returned DICOM element tag, returned as a positive integer decimal number.

Data Types: `double`

`nameOut` — Returned DICOM attribute name
character vector

Returned DICOM attribute name, returned as a character vector.

Data Types: `char`

See Also

`dicomanon` | `dicomdict` | `dicomdisp` | `dicominfo` | `dicomwrite` | `dicomread` | `dicomuid`

Introduced in R2006b

dicomread

Read DICOM image

Syntax

```
X = dicomread(filename)
X = dicomread(info)
X = dicomread( ___ ,"frames",f)
X = dicomread( ___ ,Name,Value)

[X,cmap] = dicomread( ___ )
[X,cmap,alpha] = dicomread( ___ )
[X,cmap,alpha,overlays] = dicomread( ___ )
```

Description

`X = dicomread(filename)` reads the image data from the compliant Digital Imaging and Communications in Medicine (DICOM) file `filename`. To read a group of DICOM files that contain a series of images that comprise a volume, use `dicomreadVolume`.

`X = dicomread(info)` reads DICOM image data from the message referenced in the DICOM metadata structure `info`.

`X = dicomread( ___ ,"frames",f)` reads only the specified frames `f` from the image, using any combination of input arguments from previous syntaxes.

`X = dicomread( ___ ,Name,Value)` reads DICOM image data using name-value arguments to configure the parser.

`[X,cmap] = dicomread( ___ )` also returns the colormap, `cmap`.

`[X,cmap,alpha] = dicomread( ___ )` also returns `alpha`, an alpha channel matrix for `X`.

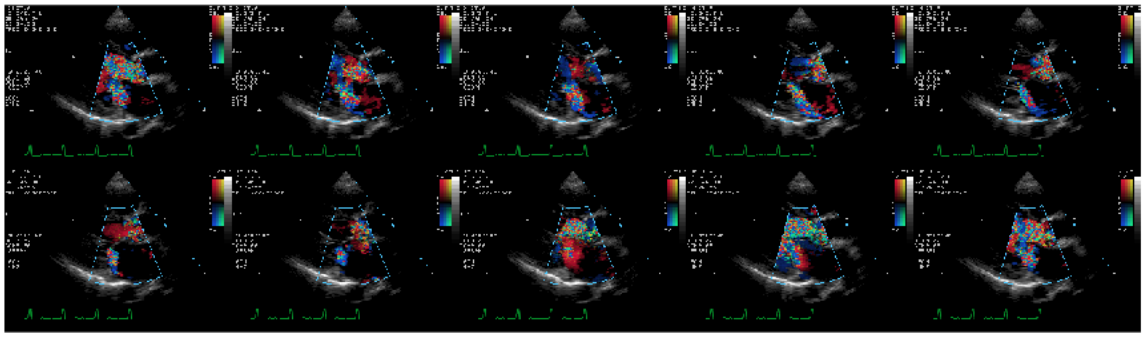
`[X,cmap,alpha,overlays] = dicomread( ___ )` also returns any overlays from the DICOM file.

Examples

Read DICOM Files

Read an indexed image from a DICOM file and display it using `montage`.

```
[X,map] = dicomread("US-PAL-8-10x-echo.dcm");
montage(X,map,"Size",[2 5]);
```



Read an image from another DICOM file and display it using `imshow`.

```
info = dicominfo("CT-MONO2-16-ankle.dcm");
Y = dicomread(info);
figure
imshow(Y,[]);
```

Input Arguments

`filename` — Name of DICOM file

character vector | string scalar

Name of the DICOM file, specified as a character vector or string scalar.

Data Types: `char` | `string`

`info` — DICOM metadata

structure

DICOM metadata, specified as a structure. You can create an `info` structure by using the `dicominfo` function.

`f` — Frames to read

`"all"` (default) | positive integer scalar | vector of positive integers

Frames to read, specified as a positive integer scalar, a vector of positive integers, or `"all"`. When `f` is numeric, `dicomread` reads only the frames of the specified numbers from the image. By default, `dicomread` reads all frames of the DICOM image.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `dicomread("CT-MONO2-16-ankle.dcm",UseVRHeuristic=false)` reads the image data from the DICOM file without using a heuristic.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `dicomread("CT-MONO2-16-ankle.dcm","UseVRHeuristic",false)` reads the image data from the DICOM file without using a heuristic.

`UseVRHeuristic` — Read noncompliant DICOM files that switch VR modes incorrectly

`true` or `1` (default) | `false` or `0`

Read noncompliant DICOM files that switch value representation (VR) modes incorrectly, specified as a logical `1` (`true`) or `0` (`false`).

When set to `true`, `dicomread` uses a heuristic to help read certain noncompliant DICOM files that switch VR modes incorrectly. `dicomread` displays a warning if it uses this heuristic. If this heuristic is enabled, a small number of compliant files are not read correctly. Set `UseVRHeuristic` to `false` to read these compliant files.

Data Types: `logical`

Output Arguments

`X` — DICOM image

m-by-*n* matrix | *m*-by-*n*-by-3 array | 4-D array

DICOM image, returned as one of these options:

- An *m*-by-*n* matrix representing a single-frame grayscale image, or an indexed image.
- An *m*-by-*n*-by-3 array representing a single-frame truecolor (RGB) image.
- A 4-D array representing a multiframe image.

Data Types: `int8` | `int16` | `uint8` | `uint16`

`cmap` — Colormap

c-by-3 matrix | `[]`

Colormap associated with image `X`, returned as one of these options:

- If `X` is an indexed image, then `cmap` is returned as a *c*-by-3 numeric matrix with values in the range [0, 1]. Each row is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap.

- If X is a grayscale or truecolor image, then cmap is empty ([]).

Data Types: double

alpha — Alpha channel matrix
m-by-*n* matrix of nonnegative integers | 4-D array of nonnegative integers

Alpha channel matrix for image X, returned as an *m*-by-*n* matrix or 4-D array of nonnegative integers. The values of alpha are 0 if the pixel is opaque. Otherwise, they are row indices into cmap. To use alpha, you should substitute the RGB value in cmap for the value in X. The alpha output has the same height and width as X, and is 4-D for a multiframe image. alpha has the same data type as X.

Data Types: int8 | int16 | uint8 | uint16

overlays — Overlays
binary *m*-by-*n* matrix | binary 4-D array | []

Overlays from the DICOM file, returned as an *m*-by-*n* matrix or 4-D array of binary values. Each overlay is a 1-bit black and white image with the same height and width as X. If multiple overlays are present in the file, then overlays is a 4-D multiframe image. If no overlays are in the file, then overlays is empty ([]).

Data Types: logical

Tips

- This function reads imagery from files with any of these pixel formats:
  - Little-endian, implicit VR, uncompressed
  - Little-endian, explicit VR, uncompressed
  - Big-endian, explicit VR, uncompressed
  - JPEG (lossy or lossless)
  - JPEG2000 (lossy or lossless)
  - Run-length Encoding (RLE)
  - GE implicit VR, LE with uncompressed BE pixels (1.2.840.113619.5.2)

See Also
dicomanon | dicomreadVolume | dicomdict | dicomdisp | dicominfo | dicomlookup | dicomwrite | dicomuid

Introduced before R2006a

dicomreadVolume

Create 4-D volume from set of DICOM images

Syntax

```
V = dicomreadVolume(source)
V = dicomreadVolume(sourcetable)
V = dicomreadVolume(sourcetable,rowname)
V = dicomreadVolume( ___ ,"MakeIsotropic",tf)
[V,spatial] = dicomreadVolume( ___ )
[V,spatial,dim] = dicomreadVolume( ___ )
```

Description

`V = dicomreadVolume(source)` creates a 4-D volume, `V`, from a set of Digital Imaging and Communications in Medicine (DICOM) files specified by `source`. The `dicomreadVolume` function identifies the correct order of the images and creates a 4-D volume.

Note If the input is a DICOM volume, then the function returns the volume data after checking the order of the image slices in the input volume. When the image slices are not in the appropriate order, the function corrects the order before returning the output.

`V = dicomreadVolume(sourcetable)` creates a 4-D DICOM volume from the input file listed in `sourcetable`. The table must contain only one row that specifies the metadata for a DICOM volume.

`V = dicomreadVolume(sourcetable,rowname)` creates a 4-D DICOM volume from the input file listed in `rowname` of the multirow table. Use this syntax when `sourcetable` contains multiple rows.

`V = dicomreadVolume( ___ ,"MakeIsotropic",tf)` creates an isotropic 4-D DICOM volume from the input DICOM image data using any combination of the input arguments from previous syntaxes. Use this syntax to create an isotropic DICOM volume from a set of nonisotropic DICOM image data.

`[V,spatial] = dicomreadVolume( ___ )` also returns a structure, `spatial`, that describes the location, resolution, and orientation of the input DICOM data.

`[V,spatial,dim] = dicomreadVolume( ___ )` also returns the dimension that has the largest amount of offset between two adjacent slices in the input DICOM data.

Examples

Create Volumetric Image from DICOM Files

Load volume data from a folder containing DICOM image files. Use the `squeeze` function to remove any singleton dimensions.

```
[V,spatial,dim] = dicomreadVolume(fullfile(matlabroot,"toolbox/images/imdata/dog"));
V = squeeze(V);
```

Display the 4-D DICOM volume. Generate a colormap and transparency map for magnetic resonance (MR) images.

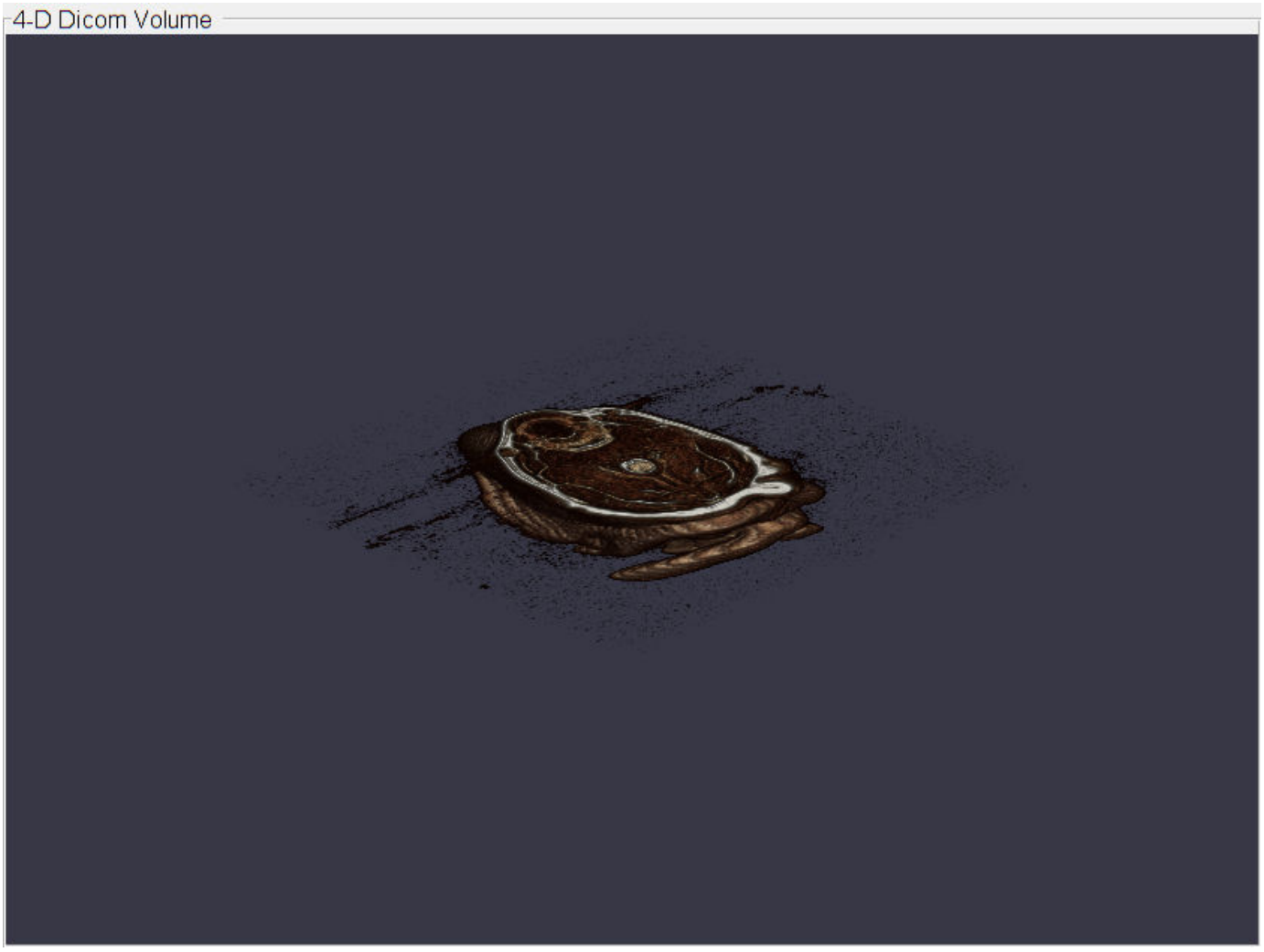
```
intensity = [0 20 40 120 220 1024];
alpha = [0 0 0.15 0.3 0.38 0.5];
color = ([0 0 0; 43 0 0; 103 37 20; 199 155 97; 216 213 201; 255 255 255])/ 255;
queryPoints = linspace(min(intensity),max(intensity),256);
alphamap = interp1(intensity,alpha,queryPoints)';
colormap = interp1(intensity,color,queryPoints);
```

Customize the display panel.

```
ViewPnl = uipanel(figure,"Title","4-D Dicom Volume");
```

View the volume with the custom colormap and transparency map.

```
volshow(V,"Colormap",colormap,"Alphamap",alphamap,"Parent",ViewPnl);
```



Display the returned spatial structure from `dicomreadVolume`. The structure contains spatial information about the input DICOM image files.

```
spatial
```

```
spatial = struct with fields:
       PatientPositions: [22×3 double]
          PixelSpacings: [22×2 double]
    PatientOrientations: [2×3×22 double]
              ImageSize: [512 512 22]
```

Display the dimension information from `dicomreadVolume`. The value specifies that the slice offset is largest along the *z*-dimension.

```
dim
```

```
dim = 3
```

Create Isotropic 4-D DICOM Volume

Gather details about the DICOM files contained in a folder by using the `dicomCollection` function. The function returns the details of the available DICOM metadata in the form of a table.

```
sourcetable = dicomCollection(fullfile(matlabroot,"toolbox/images/imdata"));
```

Display the table. The table has multiple rows, with each row containing the metadata for the DICOM image sets present in the specified folder.

```
sourcetable
```

```
sourcetable=6×14 table
               StudyDateTime              SeriesDateTime          PatientName      PatientSex
            ___________________________    ___________________________    _____________    __________

    s1      {0×0 double            }       {0×0 double            }       ""                  ""
    s2      {[30-Apr-1993 11:27:24]}       {[30-Apr-1993 11:27:24]}       "Anonymized"        ""
    s3      {[14-Dec-2013 15:47:31]}       {[14-Dec-2013 15:54:33]}       "GORBERG MITZI"     "F"
    s4      {[03-Oct-2011 19:18:11]}       {[03-Oct-2011 18:59:02]}       ""                  "M"
    s5      {[03-Oct-2011 19:18:11]}       {[03-Oct-2011 19:05:04]}       ""                  "M"
    s6      {[30-Jan-1994 11:25:01]}       {0×0 double            }       "Anonymized"        ""
```

Construct a 4-D DICOM volume from a DICOM image set in the table. Specify the row name that contains the desired DICOM image set. Set the name-value argument `MakeIsotropic` to `true` in order to create an isotropic volume. Use the `squeeze` function to remove any singleton dimensions.

```
V = dicomreadVolume(sourcetable,"s3","MakeIsotropic",true);
V = squeeze(V);
```

Display the isotropic 4-D DICOM volume by using the `volshow` function. Generate a colormap and transparency map for MR images.

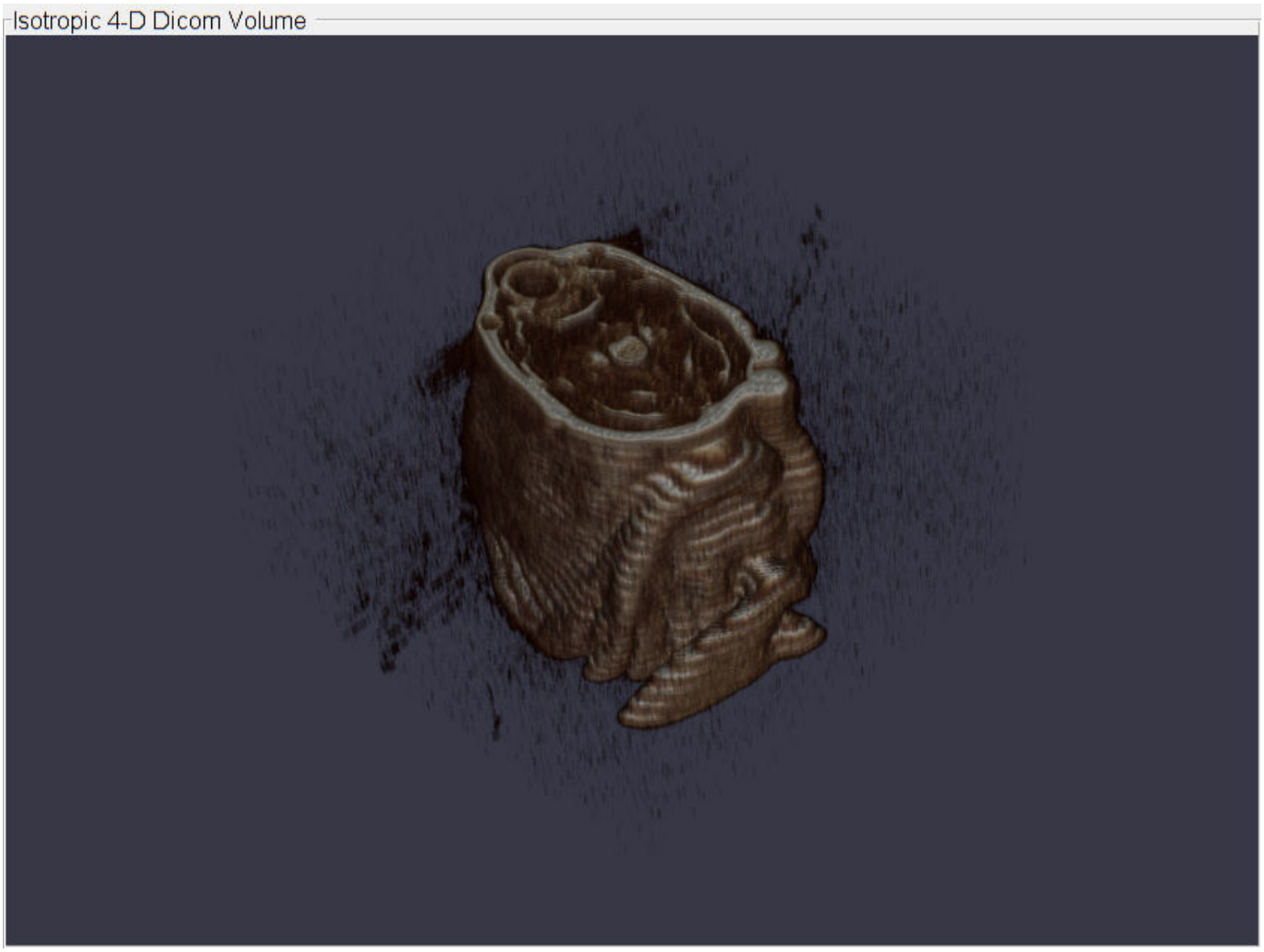
```
intensity = [0 20 40 120 220 1024];
alpha = [0 0 0.15 0.3 0.38 0.5];
color = ([0 0 0; 43 0 0; 103 37 20; 199 155 97; 216 213 201; 255 255 255])/255;
queryPoints = linspace(min(intensity),max(intensity),256);
alphamap = interp1(intensity,alpha,queryPoints)';
colormap = interp1(intensity,color,queryPoints);
```

Customize the display panel.

```
ViewPnl = uipanel(figure,"Position",[0 0 1 1],"Title","Isotropic 4-D Dicom Volume");
```

View the volume with the custom colormap and transparency map.

```
volshow(V,"Colormap",colormap,"Alphamap",alphamap,"CameraPosition",[3 3 4],"Parent",ViewPnl);
```



Input Arguments

`source` — Volume data folder or files

string scalar | character vector | string array | cell array of character vectors

Volume data folder or files, specified as a string scalar, character vector, string array, or cell array of character vectors.

Data Types: `char` | `string`

`sourcetable` — Collection of DICOM file metadata

table

Collection of DICOM file metadata, specified as a table returned by `dicomCollection`.

Data Types: `table`

`rowname` — Name of table row
string scalar | character vector

Name of the table row, specified as a string scalar or character vector. The name identifies one of the rows in the multirow table specified in `sourcetable`.

Data Types: `char` | `string`

`tf` — Create isotropic volume
`false` or `0` (default) | `true` or `1`

Create an isotropic volume, specified as a logical `0` (`false`) or `1` (`true`).

- `false` or 0 — Create a 4-D DICOM volume from the input data.
- `true` or 1 — Create an isotropic 4-D DICOM volume.

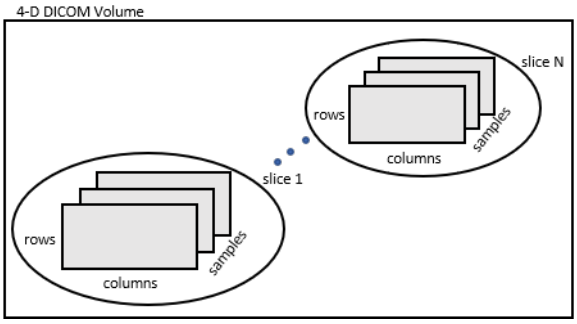
The input data specified by `source` can be either isotropic or nonisotropic DICOM data.

Output Arguments

`V` — 4-D DICOM volume
numeric array

4-D DICOM volume, returned as a numeric array.

The dimensions of `V` are [*rows*, *columns*, *samples*, *slices*], where *samples* is the number of color channels per voxel. For example, grayscale volumes have one sample, and RGB volumes have three samples. Use the `squeeze` function to remove any singleton dimensions, such as when the sample is 1.



`spatial` — Location, resolution, and orientation of input DICOM images
structure

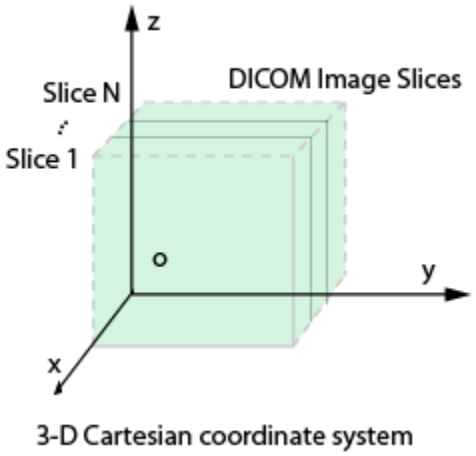
Location, resolution, and orientation of slices collected from the metadata of the input DICOM images, returned as a structure with these fields.

Field	Description
`PatientPositions`	(*x*, *y*, *z*) coordinate of the first pixel in each slice, measured in millimeters from the origin of the scanner coordinate system.
`PixelSpacings`	Distance between neighboring rows and columns within each slice, in millimeters.
`PatientOrientations`	Pair of direction-cosine triplets that designate the direction of the rows and columns in each slice relative to the patient position.

For more information about DICOM attributes, see part 3 of the DICOM standard, section C.7.6.2.

`dim` — Dimension with largest offset
1 | 2 | 3

Dimension with the largest offset, returned as 1, 2, or 3. The value denotes the dimension in a 3-D coordinate system that has the largest amount of offset between adjacent slices in the input DICOM data.



3-D Cartesian coordinate system

- If the largest offset is along the *x* dimension, then `dim` is 1.
- If the largest offset is along the *y* dimension, then `dim` is 2.
- If the largest offset is along the *z* dimension, then `dim` is 3.

See Also
`dicominfo` | `dicomread` | **DICOM Browser** | `dicomCollection` | `tiffreadVolume`

Introduced in R2017b

dicomuid

Generate DICOM globally unique identifier

Syntax

```
uid = dicomuid
```

Description

`uid = dicomuid` returns a new DICOM globally unique identifier, `uid`. The function generates a unique value each time it is called. Two calls to `dicomuid` always return different values.

Examples

Generate DICOM Globally Unique Identifier

Generate a new DICOM globally unique identifier `uid`.

```
uid = dicomuid
```

```
uid =
'1.3.6.1.4.1.9590.100.1.2.351696976011078532723666208590228528750'
```

Output Arguments

`uid` — DICOM globally unique identifier

character vector

DICOM globally unique identifier, returned as a character vector.

Data Types: `char`

See Also

`dicomanon` | `dicomdisp` | `dicominfo` | `dicomread` | `dicomwrite` | `images.dicom.decodeUID`

Introduced before R2006a

dicomupdate

Update value of target attribute in DICOM metadata

Syntax

```
newinfo = dicomupdate(info,attributeInfo)
newinfo = dicomupdate(info,attribute,value)
```

Description

`newinfo = dicomupdate(info,attributeInfo)` updates the values of the target attributes of the DICOM metadata structure `info`, and returns the updated metadata structure, `newinfo`. The `attributeInfo` argument specifies the locations and new values of the target attributes.

`newinfo = dicomupdate(info,attribute,value)` updates the value of a target attribute in a DICOM metadata structure by specifying the name of the target `attribute` and the new `value`.

Examples

Update Field in DICOM Metadata Structure

Create the DICOM metadata structure by reading it from a DICOM file.

```
info = dicominfo("rtstruct.dcm");
```

Find the value of a metadata field and its location in the DICOM metadata structure. The `dicomfind` function returns a table with two columns: `Location` and `Value`. Each element of the table is a cell array.

```
ROINumber_info = dicomfind(info,"ROINumber")
```

```
ROINumber_info=2×2 table
                      Location                       Value
    ___________________________________________    _____

    {'StructureSetROISequence.Item_1.ROINumber'}    {[1]}
    {'StructureSetROISequence.Item_2.ROINumber'}    {[2]}
```

Specify a new value for the `Value` field of the second `ROInumber`.

```
ROINumber_info.Value{2} = 4
```

```
ROINumber_info=2×2 table
                      Location                       Value
    ___________________________________________    _____

    {'StructureSetROISequence.Item_1.ROINumber'}    {[1]}
    {'StructureSetROISequence.Item_2.ROINumber'}    {[4]}
```

Update the DICOM metadata structure `info` by specifying the table that contains the new value for the second `ROInumber` field. The `dicomupdate` function creates a new, updated DICOM metadata structure.

```
newInfo = dicomupdate(info,ROINumber_info);
```

Check that the `newInfo` structure contains the updated value.

```
ROINumber_info = dicomfind(newInfo,"ROINumber")
```

```
ROINumber_info=2×2 table
                      Location                       Value
    ___________________________________________    _____

    {'StructureSetROISequence.Item_1.ROINumber'}    {[1]}
    {'StructureSetROISequence.Item_2.ROINumber'}    {[4]}
```

Update DICOM Metadata Structure with New Value

Create a DICOM metadata structure by using the `dicominfo` function.

```
info = dicominfo("rtstruct.dcm");
```

Find the value and location in the `info` structure of the `ROINumber` metadata field by using the `dicomfind` function.

```
ROINumber = dicomfind(info,"ROINumber")
```

```
ROINumber=2×2 table
                      Location                       Value
    ___________________________________________    _____

    {'StructureSetROISequence.Item_1.ROINumber'}    {[1]}
    {'StructureSetROISequence.Item_2.ROINumber'}    {[2]}
```

Update the `ROINumber` field in the DICOM metadata structure, `info`, by specifying the name of the field and its new value.

```
newInfo = dicomupdate(info,ROINumber=4);
```

Check that the `newInfo` structure contains the updated field. Notice that the metadata structure contains updated values for all instances of the specified attribute.

```
ROINumber_info = dicomfind(newInfo,"ROINumber")
```

```
ROINumber_info=2×2 table
                      Location                       Value
    ___________________________________________    _____

    {'StructureSetROISequence.Item_1.ROINumber'}    {[4]}
    {'StructureSetROISequence.Item_2.ROINumber'}    {[4]}
```

Input Arguments

`info` — DICOM metadata

structure

DICOM metadata, specified as a structure. You can extract the DICOM metadata structure from a DICOM file using the `dicominfo` function.

Data Types: `struct`

`attributeInfo` — Location and new value of target attribute

table

Location and new value of the target attribute, specified as a table.

`attribute` — Name of target DICOM metadata field

string scalar | character vector

Name of the target DICOM metadata field, specified as a string scalar or character vector.

Example: `"ROINumber"`

Data Types: `string` | `char`

`value` — New value for DICOM metadata attribute

numeric array | character vector | string scalar

New value for the DICOM metadata attribute, specified as a numeric array, string scalar, or character vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

Output Arguments

`newinfo` — Updated DICOM metadata

structure

Updated DICOM metadata, returned as a structure.

See Also

`dicomanon` | `dicomfind` | `dicomread` | `dicomreadVolume` | `dicomdict` | `dicomdisp` | `dicominfo` | `dicomlookup` | `dicomwrite` | `dicomuid`

Introduced in R2021b

dicomwrite

Write images as DICOM files

Syntax

```
dicomwrite(X,filename)
dicomwrite(X,cmap,filename)
dicomwrite( ___ ,meta_struct)
dicomwrite( ___ ,info)
dicomwrite( ___ ,"ObjectType",IOD)
dicomwrite( ___ ,"SOPClassUID",UID)
dicomwrite( ___ ,Name,Value)
status = dicomwrite( ___ )
```

Description

`dicomwrite(X,filename)` writes the binary, grayscale, or truecolor image `X` to the file `filename`. The `dicomwrite` function creates a Digital Imaging and Communications in Medicine (DICOM) file or a Digital Imaging and Communication in Security (DICOS) file. This syntax writes a file with the necessary metadata for the Secondary Capture DICOM Information Object (IOD).

`dicomwrite(X,cmap,filename)` writes the indexed image `X` with colormap `cmap`.

`dicomwrite( ___ ,meta_struct)` specifies optional metadata or file options in the structure `meta_struct`, in addition to any combination of arguments from previous syntaxes. The names of fields in `meta_struct` must be the names of DICOM file attributes or options. The value of a field is the value you want to assign to the attribute or option.

`dicomwrite( ___ ,info)` specifies metadata in the metadata structure `info`, which is produced by the `dicominfo` function.

`dicomwrite( ___ ,"ObjectType",IOD)` writes a file containing the necessary metadata for a particular type of DICOM Information Object (IOD). For the supported IODs, `dicomwrite` verifies that all required metadata attributes are present for the specified IOD, creates missing attributes if necessary, and specifies default values where possible. Additionally, `dicomwrite` removes attributes that are not part of the DICOM specification for the specified IOD. For more information, see Tips on page 1-758.

`dicomwrite( ___ ,"SOPClassUID",UID)` writes a file containing the necessary metadata for a particular type of IOD, specified using a DICOM Unique Identifier (UID).

`dicomwrite( ___ ,Name,Value)` specifies additional options using name-value arguments. You can also use this syntax to specify individual metadata attributes and their values to write to the DICOM file. To find a list of the DICOM attributes that you can specify, see the data dictionary file, `dicom-dict.txt`, included with the Image Processing Toolbox software. Enclose each attribute name in quotes.

`status = dicomwrite( ___ )` returns information about the metadata and the descriptions used to generate the DICOM file. This syntax can be useful when you specify an `info` structure to the `dicomwrite` function.

Examples

Write Data to DICOM File

Read a CT image from a DICOM file.

```
X = dicomread("CT-MONO2-16-ankle.dcm");
```

Write the CT image to a file, creating a secondary capture image. This operation creates a DICOM file with the necessary metadata attributes for the Secondary Capture DICOM Information Object (IOD).

```
dicomwrite(X,"sc_file.dcm");
```

Write the CT image to a DICOM file along with its metadata. Use the `dicominfo` function to retrieve the metadata from the DICOM file. By default, the `dicomwrite` function verifies that the metadata attributes in the new file are compliant with the Secondary Capture IOD standard.

```
metadata = dicominfo("CT-MONO2-16-ankle.dcm");
dicomwrite(X,"ct_file.dcm",metadata);
```

Write `X` to another DICOM file with a direct copy of its metadata. When you set the `CreateMode` parameter to `"copy"`, `dicomwrite` does not verify the metadata written to the file.

```
dicomwrite(X,"ct_copy.dcm",metadata,"CreateMode","copy");
```

Input Arguments

X — DICOM image

m-by-*n* matrix | *m*-by-*n*-by-3 array | 4-D array

DICOM image, specified as one of these options:

- An *m*-by-*n* matrix representing a single-frame grayscale image, or an indexed image.
- An *m*-by-*n*-by-3 array representing a single-frame truecolor (RGB) image.
- A 4-D array representing a multiframe image.

Note If `X` is empty, then the `dicomwrite` function writes a DICOM file with empty image data. The metadata attributes for the DICOM file are either set to default values or copied from `meta_struct`, if `CreateMode` is specified as `"Copy"`.

Data Types: `int8` | `int16` | `uint8` | `uint16`

cmap — Colormap

c-by-3 matrix

Colormap associated with the indexed image `X`, specified as a *c*-by-3 numeric matrix with values in the range [0, 1]. Each row is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap.

Data Types: `double`

`filename` — Name of DICOM file to write to
character vector | string scalar

Name of the DICOM file to write to, specified as a character vector or string scalar.

Data Types: `char` | `string`

`meta_struct` — Optional metadata or file options
structure

Optional metadata or file options, specified as a structure. The names of the fields in `meta_struct` must be the names of DICOM file attributes or options. The value of a field is the value you want to assign to the attribute or option.

`info` — Metadata produced by `dicominfo` function
structure

Metadata produced by the `dicominfo` function, specified as a structure.

`IOD` — DICOM Information Object
`"Secondary Capture Image Storage"` (default) | `"CT Image Storage"` | `"MR Image Storage"`

DICOM Information Object, specified as `"Secondary Capture Image Storage"`, `"CT Image Storage"`, or `"MR Image Storage"`.

Data Types: `char` | `string`

`UID` — DICOM unique identifier
character vector | string scalar

DICOM unique identifier corresponding to an `IOD`, specified as a character vector or string scalar.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `dicomwrite(X,"new_file.dcm",CompressionMode="JPEG lossless")` writes the image data `X` to a DICOM file `new_file.dcm` with JPEG lossless compression.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `dicomwrite(X,"new_file.dcm","CompressionMode","JPEG lossless")` writes the image data `X` to a DICOM file `new_file.dcm` with JPEG lossless compression.

`CompressionMode` — Compression mode
`"None"` (default) | `"JPEG lossless"` | `"JPEG lossy"` | ...

Compression mode to use when storing the image, specified as one of these options:

- `"None"`
- `"JPEG lossless"`

- "JPEG lossy"
- "JPEG2000 lossy"
- "JPEG2000 lossless"
- "RLE"

Note If you specify a compression mode, then dicomwrite ignores any values specified for the Endian or VR arguments.

CreateMode — Method used for creating data

"Create" (default) | "Copy"

Method used for creating data to put in the new file, specified as one of these options:

- "Create" — Verify input values, remove extraneous attributes, and generate missing data values.
- "Copy" — Copy all values from the input, and do not generate missing values.

For help selecting a creation method, see Tips on page 1-758.

Dictionary — Name of DICOM data dictionary

character vector | string scalar

Name of the DICOM data dictionary, specified as a character vector or string scalar. The default file is dicom-dict.mat.

Endian — Byte ordering

"ieee-le" (default) | "ieee-be"

Byte ordering of the file, specified as "ieee-le" or "ieee-be".

Note If VR is set to "Implicit", then Endian must be "ieee-le". The dicomwrite function ignores this value if you specify CompressionMode or TransferSyntax.

MultiframeSingleFile — Write multiframe image to one file

true or 1 (default) | false or 0

Write a multiframe image to one file, specified as a logical 1 (true) or 0 (false). When true, the function creates one file regardless of how many frames X contains. When false, the function creates one file for each frame in the image.

Data Types: logical

TransferSyntax — Transfer syntax

character vector | string scalar

Transfer syntax, specified as a character vector or string scalar. TransferSyntax is a UID that encodes values for the Endian, VR, and CompressionMode options.

Note If you specify a transfer syntax, then dicomwrite ignores any values specified for the Endian, VR, and CompressionMode options.

`UseMetadataBitDepths` — Preserve metadata values
`false` or `0` (default) | `true` or `1`

Preserve the metadata values, specified as a logical `0` (`false`) or `1` (`true`). When `true`, `dicomwrite` preserves the existing values of the `"BitStored"`, `"BitsAllocated"`, and`"HighBit"` fields of the metadata structure. When `false`, `dicomwrite` computes these values based on the data type of the pixel data.

Note When you specify `CreateMode` as `"Create"`, `dicomwrite` ignores this value.

Data Types: `logical`

`VR` — Write two-letter value representation (VR) code to file
`"implicit"` (default) | `"explicit"`

Write two-letter value representation (VR) code to file, specified as one of these options:

- `"implicit"` — Infer from the data dictionary.
- `"explicit"` — Write VR to the file.

Note If you specify the `Endian` value as `"ieee-be"`, then you must specify `VR` as `"explicit"`. The `dicomwrite` function ignores this value if you specify `TransferSyntax` or `CompressionMode`.

`WritePrivate` — Write private data to file
`false` or `0` (default) | `true` or `1`

Write private data to file, specified as a logical `0` (`false`) or `1` (`true`).

Data Types: `logical`

Output Arguments

`status` — Status of attributes
structure | `[]`

Status of attributes, returned as a structure. `status` contains information about the metadata and the descriptions used to generate the DICOM file. If no metadata is specified, `dicomwrite` returns an empty matrix (`[]`).

The `status` structure contains these fields.

Field	Description
`BadAttribute`	The internal description of the attribute is bad. It might be missing from the data dictionary or have incorrect data in its description.
`MissingCondition`	The attribute is conditional, but no condition has been provided for when to use it.
`MissingData`	No data provided for an attribute that must appear in the file.

Field	Description
SuspectAttribute	Data in the attribute does not match a list of enumerated values in the DICOM specification.

Tips

- The DICOM format specification lists several Information Object Definitions (IODs) that can be created. These IODs correspond to images and metadata produced by different real-world modalities (such as MRI, X-ray, and Ultrasound). For each type of IOD, the DICOM specification defines the set of metadata that must be present, as well as possible values for other metadata.

  - `dicomwrite` fully implements a limited number of `IOD`s (Secondary Capture, Computed Tomography, Magnetic Resonance). For these IODs, `dicomwrite` verifies that all required metadata attributes are present for the specified IOD, creates missing attributes if necessary, and specifies default values where possible. Additionally, `dicomwrite` removes attributes that are not part of the DICOM specification for the specified IOD. This behavior corresponds to the default value of the `CreateMode` name-value argument, `"Create"`. If you are working with DICOM files with one of the supported IODs, setting the `CreateMode` to `"Create"` is the best way to ensure that the files you create conform to the DICOM specification.
  - To write DICOM files for IODs that `dicomwrite` does not fully support, use the `"Copy"` value for the `CreateMode` name-value argument. In this mode, `dicomwrite` writes the image data to a file including the metadata that you specify using the `info` input argument. This option enables you to take metadata from an existing file of the same modality or IOD and use it to create a new DICOM file with different image pixel data. If the image data is empty, `dicomwrite` does not write image-related metadata attributes to the new DICOM file.

    Note Because `dicomwrite` copies metadata to the file without verification in `"Copy"` mode, you can create a DICOM file that does not conform to the DICOM standard. For example, the file may be missing required metadata, contain superfluous metadata, or the metadata may no longer correspond to the modality settings used to generate the original image. When using `"Copy"` mode, make sure that the metadata you use is from the same modality and IOD. If the copy you make is unrelated to the original image, use `dicomuid` to create new unique identifiers for series and study metadata. See the IOD descriptions in Part 3 of the DICOM specification for more information on appropriate IOD values.

- The `dicomwrite` function removes metadata attributes with "`Group Length`" in their names before writing a new DICOM file, regardless of the value of the `CreateMode` name-value argument. Including group length attributes is error-prone, and is not currently recommended by the DICOM specification. Other software or devices can read DICOM files without group length attributes.

See Also

`dicomanon` | `dicomdict` | `dicomdisp` | `dicominfo` | `dicomlookup` | `dicomread` | `dicomuid`

Topics

“Write Image Data to DICOM Files”
“Create New DICOM Series”
“Remove Confidential Information from DICOM File”

Introduced before R2006a

displayChart

Display test chart with overlaid regions of interest

Syntax

```
displayChart(chart)
displayChart(chart,Name,Value)
```

Description

`displayChart(chart)` displays an Imatest® eSFR chart [1] or a Calibrite ColorChecker Classic chart [2] with ROIs overlaid on detected features of the chart.

`displayChart(chart,Name,Value)` controls aspects of the chart display using name-value arguments.

Examples

Display Color Patch ROIs on an eSFR Chart

Read an image of an eSFR chart into the workspace.

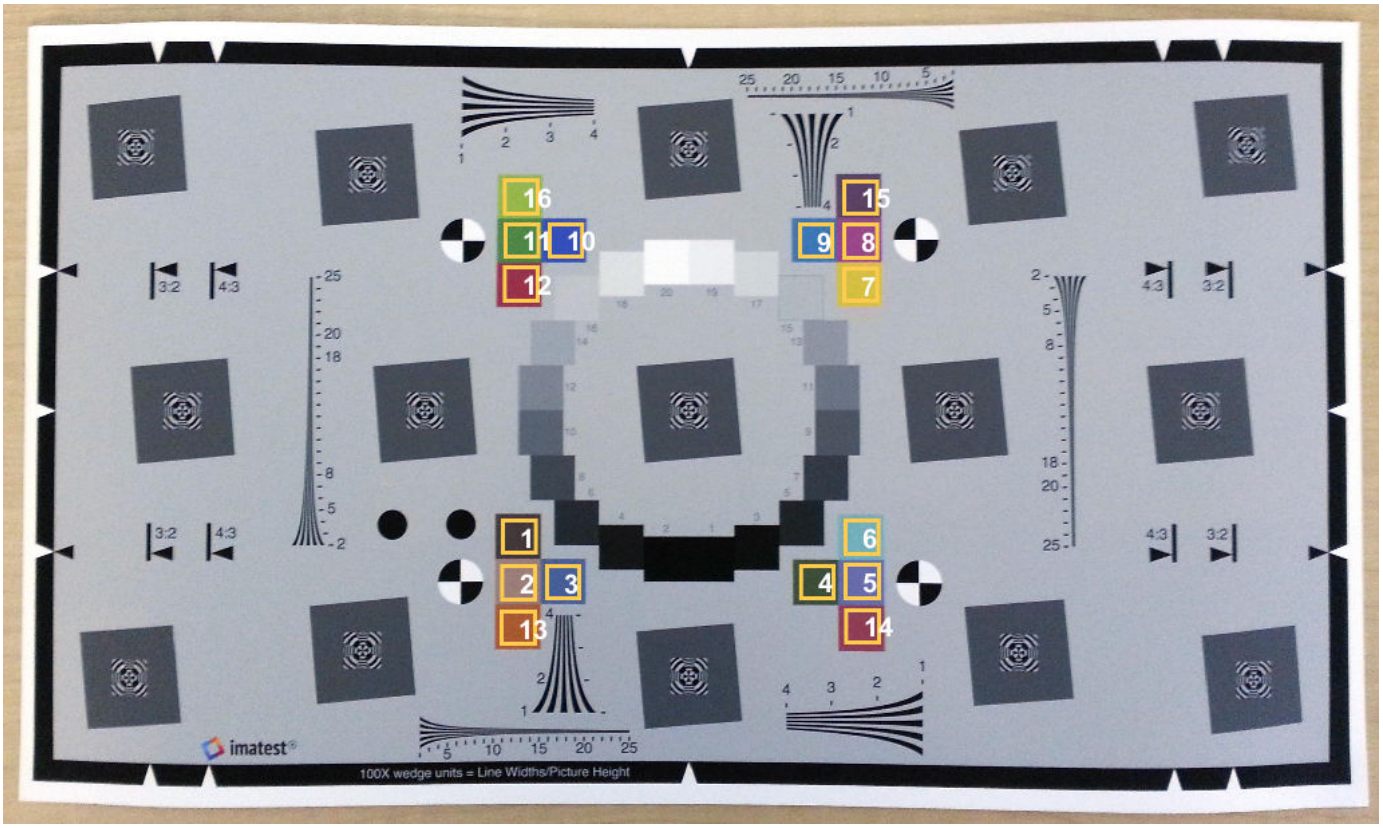
```
I = imread('eSFRTestImage.jpg');
```

Create an `esfrChart` object that stores information about the test chart.

```
chart = esfrChart(I);
```

Display only the color patch ROIs. To accomplish this, turn off the display of slanted edge ROIs, gray patch ROIs, and registration points.

```
displayChart(chart,'displayEdgeROIs',false,'displayGrayROIs',false,'displayRegistrationPoints',fa
```



Display Registration Points on ColorChecker Chart

Read an image of a ColorChecker® chart into the workspace.

```
I = imread("colorCheckerTestImage.jpg");
```

Create a `colorChecker` object by performing automatic chart detection on the image.

```
chart = colorChecker(I);
```

Display the chart with the detected corner registration points only. Turn off the display of the color patch ROIs.

```
displayChart(chart,"displayColorROIs",false)
```

Input Arguments

`chart` — Test chart

`esfrChart` object | `colorChecker` object

Test chart, specified as an `esfrChart` object or a `colorChecker` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'displayRegistrationPoints',false` turns off the overlay of registration points on the chart.

`displayEdgeROIs` — Display slanted edge ROIs

`true` or `1` (default) | `false` or `0`

Display slanted edge ROIs, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `displayEdgeROIs` is `true`, the 60 slanted-edge bounding boxes are overlaid on the image in pale yellow.

This argument is supported by eSFR test charts only.

`displayGrayROIs` — Display gray patch ROIs
`true` or `1` (default) | `false` or `0`

Display gray patch ROIs, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `displayGrayROIs` is `true`, the 20 gray patch bounding boxes are overlaid on the image in blue.

This argument is supported by eSFR test charts only.

`displayColorROIs` — Display color patch ROIs
`true` or `1` (default) | `false` or `0`

Display color patch ROIs, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `displayColorROIs` is `true`, the 16 color patch bounding boxes are overlaid on the image in dark yellow.

`displayRegistrationPoints` — Display registration points
`true` or `1` (default) | `false` or `0`

Display registration points, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `displayRegistrationPoints` is `true`, the four registration points are indicated with a red diamond overlay.

`Parent` — Axes handle of displayed image object
axes handle

Axes handle of the displayed image object, specified as an axes handle. `Parent` specifies the parent of the image object created by `displayChart`.

References

[1] Imatest. "Esfr". https://www.imatest.com/mathworks/esfr/.

[2] Calibrite. "ColorChecker Classic". https://calibrite.com/us/product/colorchecker-classic/.

See Also

Functions
`measureSharpness` | `measureChromaticAberration` | `measureNoise` | `measureColor` | `measureIlluminant`

Objects
`esfrChart` | `colorChecker`

Introduced in R2017b

displayColorPatch

Display measured and reference color as color patches

Syntax

```
displayColorPatch(colorTable)
displayColorPatch(colorTable,Name,Value)
```

Description

`displayColorPatch(colorTable)` displays measured and reference colors, `colorTable`, for color patch regions of interest (ROIs) in a test chart. The measured color values are displayed as squares surrounded by a thick boundary of the corresponding reference color.

`displayColorPatch(colorTable,Name,Value)` displays measured color values with additional name-value arguments to control aspects of the display.

Examples

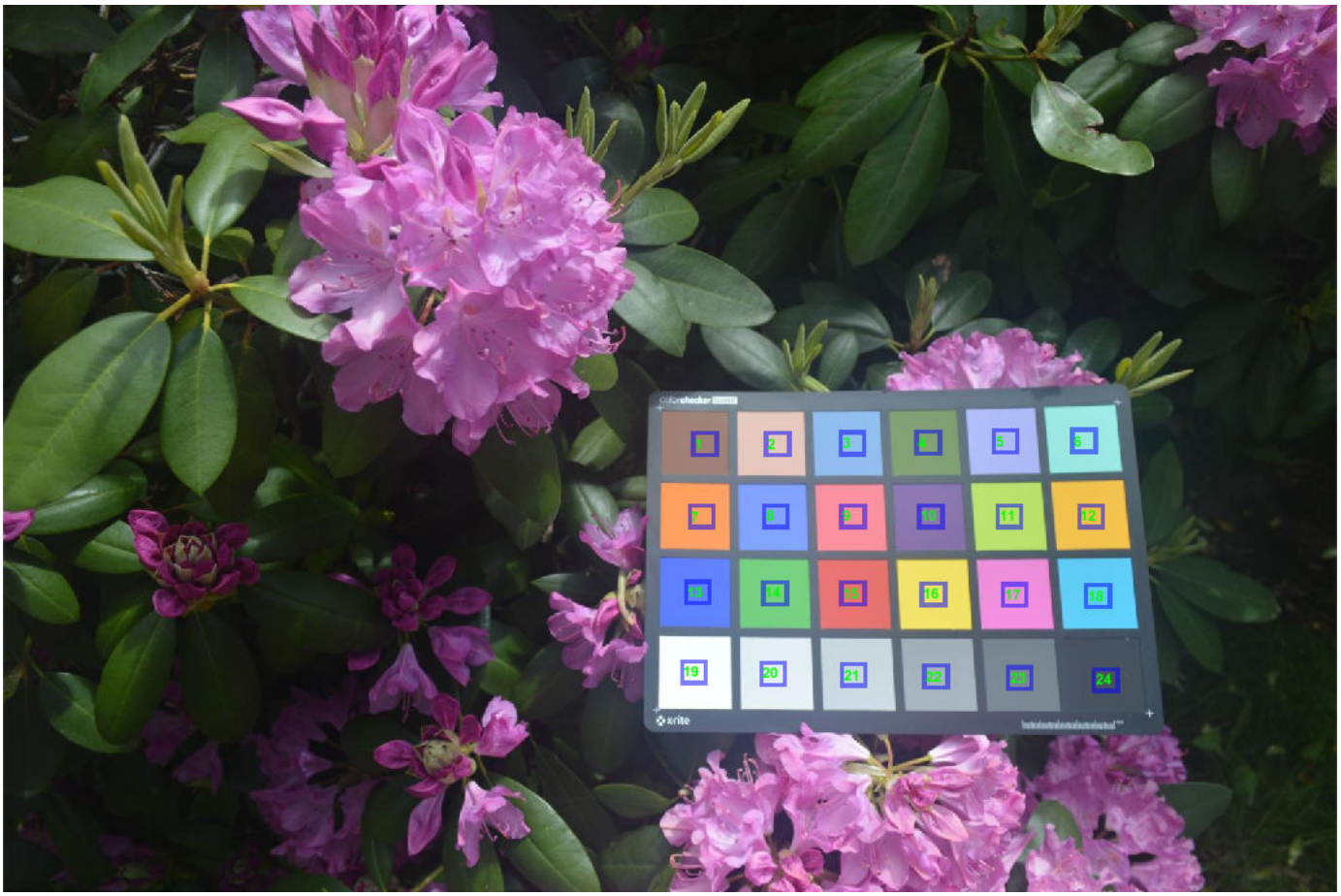
Display Color Patch Diagram from ColorChecker Chart

Read an image of a ColorChecker® chart into the workspace.

```
I = imread("colorCheckerTestImage.jpg");
```

Create a `colorChecker` object, then display the chart with ROI annotations.

```
chart = colorChecker(I);
displayChart(chart,"displayRegistrationPoints",false)
```

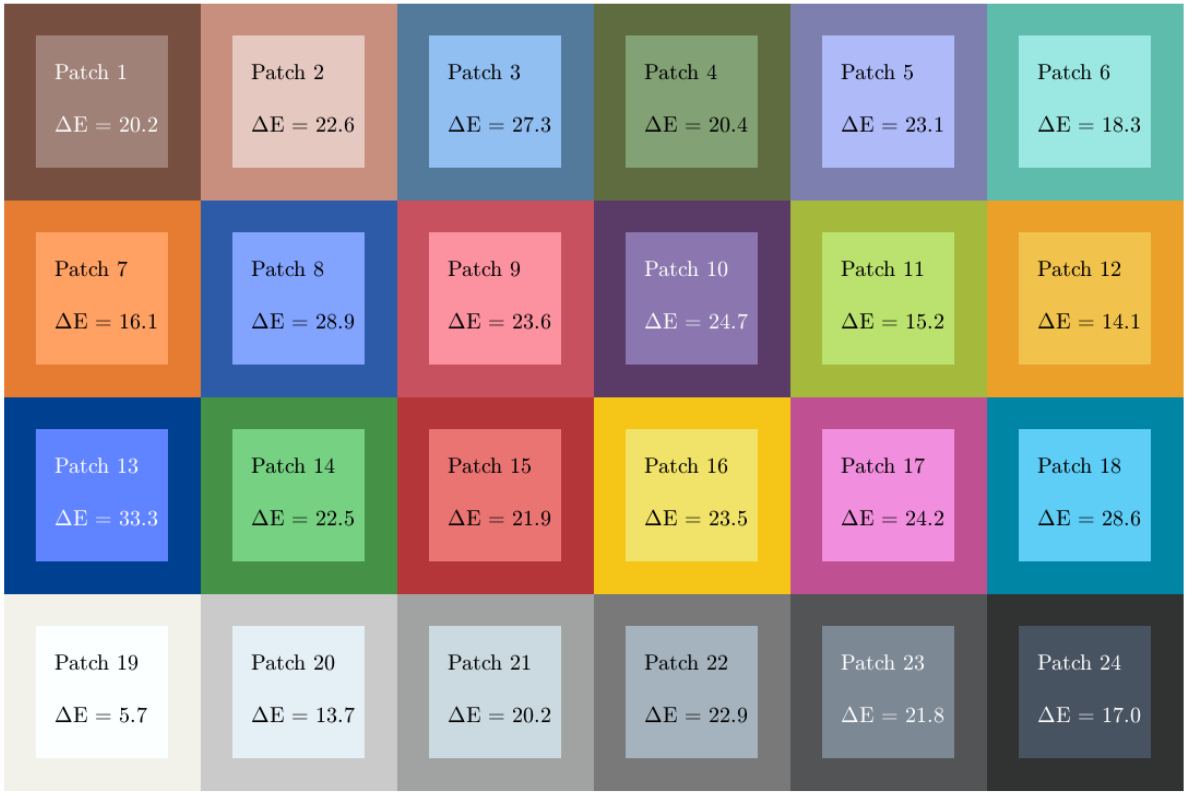


Measure the color in each color patch ROI.

```
colorTable = measureColor(chart);
```

On a color patch diagram, display the measured and reference colors and the color error (ΔE).

```
displayColorPatch(colorTable)
```



Display Color Patch Diagram from Color Accuracy Measurements

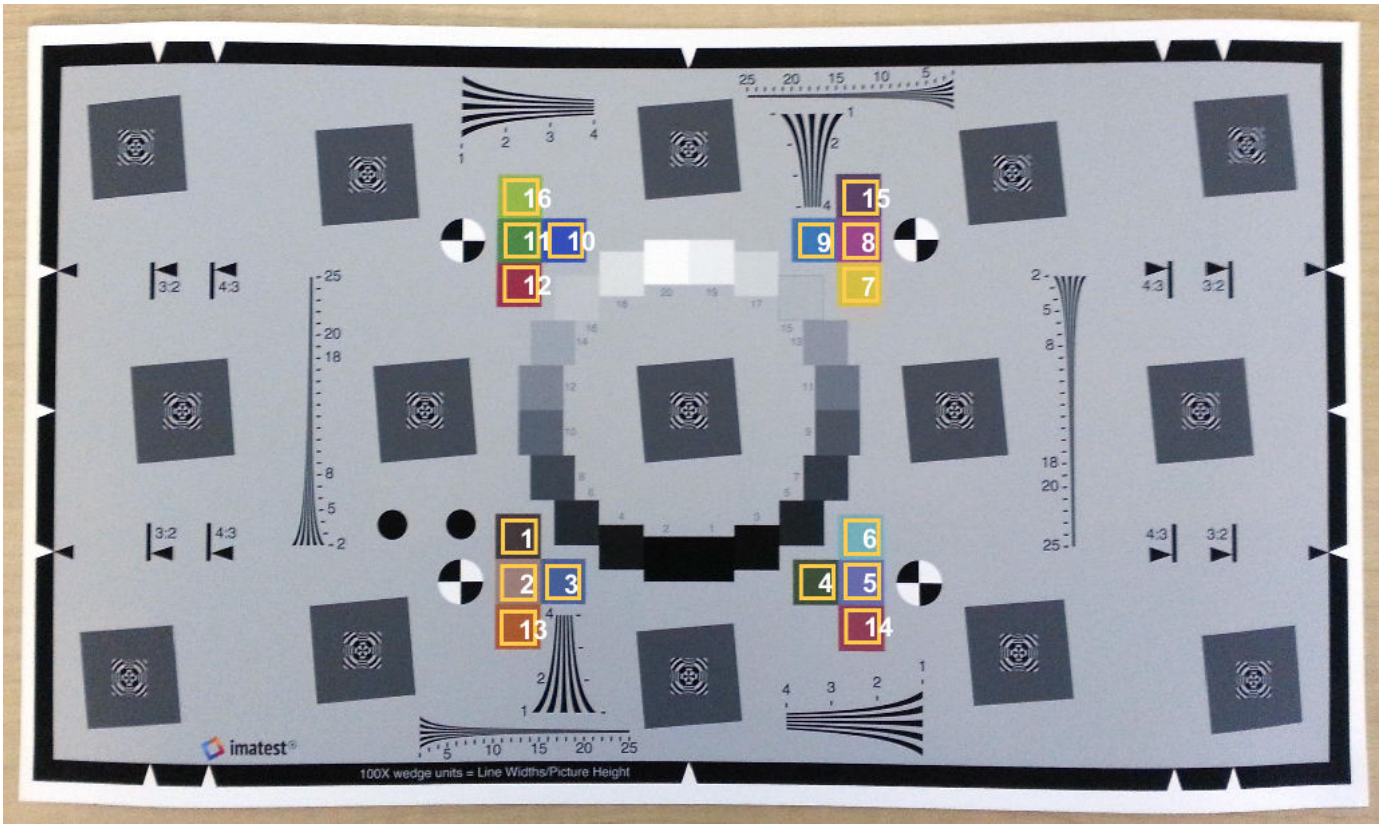
This example shows how to display the color patch diagram from measurements of color accuracy on an Imatest® eSFR chart.

Read an image of an eSFR chart into the workspace.

```
I = imread('eSFRTestImage.jpg');
```

Create an `esfrChart` object. Display the chart, highlighting the 16 color patches.

```
chart = esfrChart(I);
displayChart(chart,'displayEdgeROIs',false, ...
    'displayGrayROIs',false,'displayRegistrationPoints',false)
```

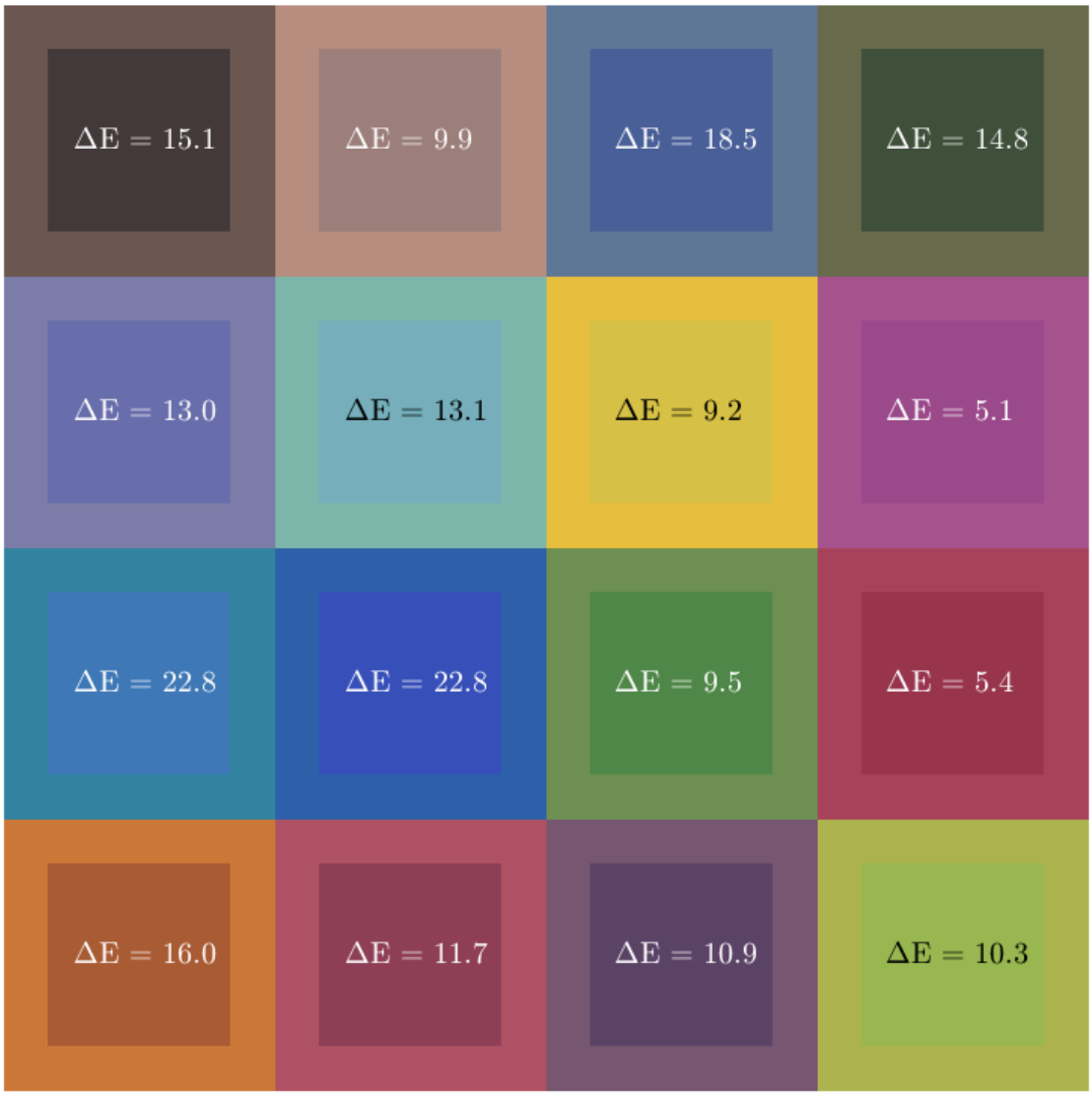


Measure the color in all color patch ROIs.

```
colorTable = measureColor(chart);
```

Display the color accuracy measurements without the ROI index overlay. Each square color patch is the measured color, and the thick surrounding border is the reference color for that ROI. The color accuracy measurement is displayed as `Delta_E`, the Euclidean distance between measured and reference colors in CIE 1976 L*a*b* color space. More accurate colors have a smaller `Delta_E`.

```
displayColorPatch(colorTable,'displayROIIndex',false)
```



Input Arguments

`colorTable` — Color values

color table

Color values in each color patch, specified as an *m*-by-8 color table, where *m* is the number of patches. The eight columns represent these variables:

Variable	Description
ROI	Index of the sampled ROI. The value of ROI is an integer in the range [1, 16]. The indices match the ROI numbers displayed by displayChart.
Measured_R	Mean value of red channel pixels in the ROI. Measured_R is a scalar of the same data type as chart.Image, which can be of type single, double, uint8, or uint16.
Measured_G	Mean value of green channel pixels in the ROI. Measured_G is a scalar of the same data type as chart.Image.
Measured_B	Mean value of blue channel pixels in the ROI. Measured_B is a scalar of the same data type as chart.Image.
Reference_L	Reference L* value of the ROI. Reference_L is a scalar of type double.
Reference_a	Reference a* value of the ROI. Reference_a is a scalar of type double.
Reference_b	Reference b* value of the ROI. Reference_b is a scalar of type double.
Delta_E	Euclidean color distance between the measured and reference color values in the L*a*b* color space, as outlined in CIE 1976. Delta_E is a scalar of type double.

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: displayColorPatch(colorTable,displayROIIndex=false) turns off the display of the ROI indices

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Example: displayColorPatch(colorTable,"displayROIIndex",false) turns off the display of the ROI indices

displayROIIndex — Display ROI index labels

true or 1 (default) | false or 0

Display ROI index labels, specified as a numeric or logical 1 (true) or 0 (false). When displayROIIndex is true, then displayColorPatch overlays color patch ROI index labels on the displayed color patches. The indices match the ROI numbers displayed by displayChart.

displayDeltaE — Display color error values

true or 1 (default) | false or 0

Display color error ("delta E") values, specified as a numeric or logical 1 (true) or 0 (false). When displayDeltaE is true, displayColorPatch overlays the color error values on the displayed color patches.

Parent — Axes handle of displayed image object

axes handle

Axes handle of the displayed image object, specified as an axes handle. Parent specifies the parent of the image object created by displayColorPatch.

Tips

- To obtain a color table of the correct format from an `esfrChart` or `colorChecker` object, use the `measureColor` function. You can also create your own color table containing measured and reference colors for an arbitrary number of color ROIs.
- The reference L*a*b* values of a `colorTable` measured from a `colorChecker` object are for the "After November 2014" version of the ColorChecker chart. The white point of the reference values is the CIE standard illuminant D50.

See Also

Functions
`measureColor` | `plotChromaticity` | `displayChart`

Objects
`esfrChart` | `colorChecker`

Topics
“Correct Colors Using Color Correction Matrix”

Introduced in R2017b

dlresize

Resize spatial dimensions of `dlarray` object

Syntax

```
Y = dlresize(X,'Scale',scale)
Y = dlresize(X,'OutputSize',outputSize)
Y = dlresize( ___ ,Name,Value)
```

Description

`Y = dlresize(X,'Scale',scale)` resizes the spatial dimensions of the `dlarray` object `X` by a scale factor `scale`.

This function requires Deep Learning Toolbox.

`Y = dlresize(X,'OutputSize',outputSize)` resizes the spatial dimensions of the `dlarray` object `X` so that the spatial dimension sizes are equal to `outputSize`.

`Y = dlresize( ___ ,Name,Value)` adjusts the resizing operation using name-value pair arguments. If `X` is not a formatted `dlarray`, then you must specify the `DataFormat` name-value pair argument.

Examples

Resize dlarray by Scale Factor

Read an RGB image.

```
A = imread('peppers.png');
```

Convert the image to data type `single` for use in a `dlarray`. Then, create a `dlarray` containing the input image.

```
A = im2single(A);
dlarrayA = dlarray(A,'SSC');
```

Rescale the `dlarray` by a factor of 1.5 vertically.

```
dlarrayB = dlresize(dlarrayA,'Scale',[1.5 1]);
```

Extract the image data from the resized `dlarray` `B` by using the `extractdata` (Deep Learning Toolbox) function.

```
B = extractdata(dlarrayB);
```

Display the original and resized images as a montage.

```
montage({A,B},"ThumbnailSize",size(B,[1 2]), ...
    "BorderSize",10,"BackgroundColor","white")
```

Input Arguments

`X` — Deep learning array to resize

`dlarray` object

Deep learning array to resize, specified as a `dlarray` object.

`scale` — Scale factor to resize input

positive number | vector of positive numbers

Scale factor to resize input, specified as a positive number or a vector of positive numbers of length equal to the number of spatial dimensions in `X`. If `scale` is a scalar, then `dlresize` applies the same scale factor to all spatial dimensions.

`outputSize` — Output size of resized input

vector of positive integers

Output size of resized input, specified as a vector of positive integers of length equal to the number of spatial dimensions in `X`. You can specify one element as a positive integer and specify the other elements as NaN, in which case the layer computes the other elements automatically to preserve the aspect ratio of the input.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Method',linear`

`DataFormat` — Dimension labels
string scalar | character vector

Dimension labels of the input `dlarray` object `X`, specified as the comma-separated pair consisting of `'DataFormat'` and a string scalar or character vector. Each character must be one of these labels:

- `S` — Spatial
- `C` — Channel
- `B` — Batch observations
- `T` — Time or sequence
- `U` — Unspecified

If `X` is not a formatted `dlarray`, then you must specify the `DataFormat` name-value pair argument. For more information, see `dlarray`.

Example: `'SSC'` indicates the array has two spatial dimensions and one channel dimension, appropriate for 2-D RGB image data.

`Method` — Interpolation method
`"nearest"` (default) | `"linear"`

Interpolation method, specified as the comma-separated pair consisting of `'Method'` and `"nearest"` for nearest neighbor interpolation or `"linear"` for bilinear interpolation.

`GeometricTransformMode` — Geometric transformation mode
`"half-pixel"` (default) | `"asymmetric"`

Geometric transformation mode to map points from input space to output space, specified as the comma-separated pair consisting of `'GeometricTransformMode'` and `"half-pixel"` or `"asymmetric"`.

`NearestRoundingMode` — Rounding mode for nearest neighbor interpolation
`"round"` (default) | `"floor"` | `"onnx-10"`

Rounding mode for nearest neighbor interpolation, specified as the comma-separated pair consisting of `'NearestRoundingMode'` and one of the following.

- `"round"` — use the same rounding behavior as the MATLAB `round` function.
- `"floor"` — use the same rounding behavior as the MATLAB `floor` function.
- `"onnx-10"` — reproduce the resizing behavior of the ONNX™ (Open Neural Network Exchange) opset 10 Resize operator.

This argument is used when you specify the `Method` as `'nearest'`.

Output Arguments

`Y` — Resized deep learning array
`dlarray` object

Resized deep learning array, returned as a `dlarray` object.

Extended Capabilities

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`dlarray` | `maxpool` | `dltranspconv`

Introduced in R2020b

dnCNNLayers

Get denoising convolutional neural network layers

Syntax

```
layers = dnCNNLayers
layers = dnCNNLayers('NetworkDepth',networkDepth)
```

Description

`layers = dnCNNLayers` returns layers of the denoising convolutional neural network (DnCNN) for grayscale images.

This function requires that you have Deep Learning Toolbox.

`layers = dnCNNLayers('NetworkDepth',networkDepth)` returns a denoising convolutional neural network with `networkDepth` number of convolutional layers.

Examples

Get Layers of Image Denoising Network

Get layers of the image denoising convolutional neural network, 'DnCNN'. Request the default number of layers, which returns 20 convolution layers.

```
layers = dnCNNLayers
```

```
layers =
  1x59 Layer array with layers:

     1   'InputLayer'             Image Input           50x50x1 images
     2   'Conv1'                  Convolution           64 3x3x1 convolutions with stride [1  1]
     3   'ReLU1'                  ReLU                  ReLU
     4   'Conv2'                  Convolution           64 3x3x64 convolutions with stride [1  1]
     5   'BNorm2'                 Batch Normalization   Batch normalization with 64 channels
     6   'ReLU2'                  ReLU                  ReLU
     7   'Conv3'                  Convolution           64 3x3x64 convolutions with stride [1  1]
     8   'BNorm3'                 Batch Normalization   Batch normalization with 64 channels
     9   'ReLU3'                  ReLU                  ReLU
    10   'Conv4'                  Convolution           64 3x3x64 convolutions with stride [1  1]
    11   'BNorm4'                 Batch Normalization   Batch normalization with 64 channels
    12   'ReLU4'                  ReLU                  ReLU
    13   'Conv5'                  Convolution           64 3x3x64 convolutions with stride [1  1]
    14   'BNorm5'                 Batch Normalization   Batch normalization with 64 channels
    15   'ReLU5'                  ReLU                  ReLU
    16   'Conv6'                  Convolution           64 3x3x64 convolutions with stride [1  1]
    17   'BNorm6'                 Batch Normalization   Batch normalization with 64 channels
    18   'ReLU6'                  ReLU                  ReLU
    19   'Conv7'                  Convolution           64 3x3x64 convolutions with stride [1  1]
    20   'BNorm7'                 Batch Normalization   Batch normalization with 64 channels
    21   'ReLU7'                  ReLU                  ReLU
```

```
  22   'Conv8'                  Convolution           64 3x3x64 convolutions with stride [1  1]
  23   'BNorm8'                 Batch Normalization   Batch normalization with 64 channels
  24   'ReLU8'                  ReLU                  ReLU
  25   'Conv9'                  Convolution           64 3x3x64 convolutions with stride [1  1]
  26   'BNorm9'                 Batch Normalization   Batch normalization with 64 channels
  27   'ReLU9'                  ReLU                  ReLU
  28   'Conv10'                 Convolution           64 3x3x64 convolutions with stride [1  1]
  29   'BNorm10'                Batch Normalization   Batch normalization with 64 channels
  30   'ReLU10'                 ReLU                  ReLU
  31   'Conv11'                 Convolution           64 3x3x64 convolutions with stride [1  1]
  32   'BNorm11'                Batch Normalization   Batch normalization with 64 channels
  33   'ReLU11'                 ReLU                  ReLU
  34   'Conv12'                 Convolution           64 3x3x64 convolutions with stride [1  1]
  35   'BNorm12'                Batch Normalization   Batch normalization with 64 channels
  36   'ReLU12'                 ReLU                  ReLU
  37   'Conv13'                 Convolution           64 3x3x64 convolutions with stride [1  1]
  38   'BNorm13'                Batch Normalization   Batch normalization with 64 channels
  39   'ReLU13'                 ReLU                  ReLU
  40   'Conv14'                 Convolution           64 3x3x64 convolutions with stride [1  1]
  41   'BNorm14'                Batch Normalization   Batch normalization with 64 channels
  42   'ReLU14'                 ReLU                  ReLU
  43   'Conv15'                 Convolution           64 3x3x64 convolutions with stride [1  1]
  44   'BNorm15'                Batch Normalization   Batch normalization with 64 channels
  45   'ReLU15'                 ReLU                  ReLU
  46   'Conv16'                 Convolution           64 3x3x64 convolutions with stride [1  1]
  47   'BNorm16'                Batch Normalization   Batch normalization with 64 channels
  48   'ReLU16'                 ReLU                  ReLU
  49   'Conv17'                 Convolution           64 3x3x64 convolutions with stride [1  1]
  50   'BNorm17'                Batch Normalization   Batch normalization with 64 channels
  51   'ReLU17'                 ReLU                  ReLU
  52   'Conv18'                 Convolution           64 3x3x64 convolutions with stride [1  1]
  53   'BNorm18'                Batch Normalization   Batch normalization with 64 channels
  54   'ReLU18'                 ReLU                  ReLU
  55   'Conv19'                 Convolution           64 3x3x64 convolutions with stride [1  1]
  56   'BNorm19'                Batch Normalization   Batch normalization with 64 channels
  57   'ReLU19'                 ReLU                  ReLU
  58   'Conv20'                 Convolution           1 3x3x64 convolutions with stride [1  1]
  59   'FinalRegressionLayer'   Regression Output     mean-squared-error
```

You can train a custom image denoising network by providing these layers and a `denoisingImageDatastore` to `trainNetwork` (Deep Learning Toolbox).

Input Arguments

`networkDepth` — Number of convolution layers

`20` (default) | positive integer

Number of convolution layers, specified as a positive integer with value greater than or equal to 3.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output Arguments

`layers` — Network layers

vector of `Layer` objects

Denoising convolutional neural network layers, returned as a vector of `Layer` objects.

Tips

- The DnCNN network can detect noise and other high-frequency image artifacts. For example, you can train the DnCNN network to increase image resolution or remove JPEG compression artifacts. The example “JPEG Image Deblocking Using Deep Learning” shows how to train a DnCNN to reduce JPEG compression artifacts in an image.

References

[1] Zhang, K., W. Zuo, Y. Chen, D. Meng, and L. Zhang. "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising." *IEEE Transactions on Image Processing*. Vol. 26, Issue 7, 2017, pp. 3142–3155.

See Also

`denoiseImage` | `denoisingNetwork` | `denoisingImageDatastore` | `trainNetwork`

Topics

“JPEG Image Deblocking Using Deep Learning”
“Train and Apply Denoising Neural Networks”

Introduced in R2017b

dpxinfo

Read metadata from DPX file

Syntax

```
metadata = dpxinfo(filename)
```

Description

`metadata = dpxinfo(filename)` reads information about the image contained in the DPX file specified by `filename`. `metadata` is a structure containing the file details.

Digital Picture Exchange (DPX) is an ANSI standard file format commonly used for still-frame storage in digital intermediate post-production facilities and film labs.

Examples

Read Metadata from DPX File

Read metadata from DPX file into the workspace.

```
m = dpxinfo('peppers.dpx')
```

```
m = struct with fields:
                     Filename: 'B:\matlab\toolbox\images\imdata\peppers.dpx'
                  FileModDate: '16-Mar-2015 09:57:26'
                     FileSize: 892828
                       Format: 'DPX'
                FormatVersion: '2.0'
                        Width: 512
                       Height: 384
                     BitDepth: 36
                    ColorType: 'R,G,B'
              FormatSignature: [88 80 68 83]
                    ByteOrder: 'Little-endian'
                  Orientation: 'Left-to-right, Top-to-bottom'
      NumberOfImageElements: 1
                    DataSign: {'Unsigned'}
  AmplitudeTransferFunction: {'ITU-R 709-4'}
                 Colorimetry: {'ITU-R 709-4'}
           ChannelBitDepths: 12
              PackingMethod: 0
                   Encoding: {'None'}
```

Input Arguments

`filename` — Name of the DPX file

character vector | string scalar

Name of a DPX file, specified as a string scalar or character vector. `filename` can contain the absolute path to the file, the name of a file on the MATLAB path, or a relative path.

Data Types: `char` | `string`

Output Arguments

`metadata` — Information about the DPX image data

structure

Information about the DPX image data, returned as a structure.

See Also

`dpxread`

Introduced in R2015b

dpxread

Read DPX image

Syntax

```
X = dpxread(filename)
```

Description

`X = dpxread(filename)` reads image data from the DPX file specified by `filename`, returning the image `X`.

Digital Picture Exchange (DPX) is an ANSI standard file format commonly used for still-frame storage in digital intermediate post-production facilities and film labs.

Examples

Read and Visualize 12-bit RGB Image

Read image from DPX file into the workspace.

```
RGB = dpxread('peppers.dpx');
```

Create a scale factor based on the data range of the image data. The image needs to be scaled to span the 16-bit data range expected by `imshow`.

```
maxOfDataRange = 2^12 - 1;
scaleFactor = intmax('uint16') / maxOfDataRange;
```

Display the image.

```
figure
imshow(RGB * scaleFactor)
```

Input Arguments

`filename` — Name of the DPX file

character vector | string scalar

Name of a DPX file, specified as a string scalar or character vector. `filename` can contain the absolute path to the file, the name of a file on the MATLAB path, or a relative path.

Example: `RGB = dpxread('peppers.dpx');`

Data Types: `char` | `string`

Output Arguments

X — Image data from DPX file

numeric array

Image data from DPX file, returned as a numeric array of class `uint8` or `uint16`, depending on the bit depth of the pixels in `filename`.

See Also

`dpxinfo`

Introduced in R2015b

drawassisted

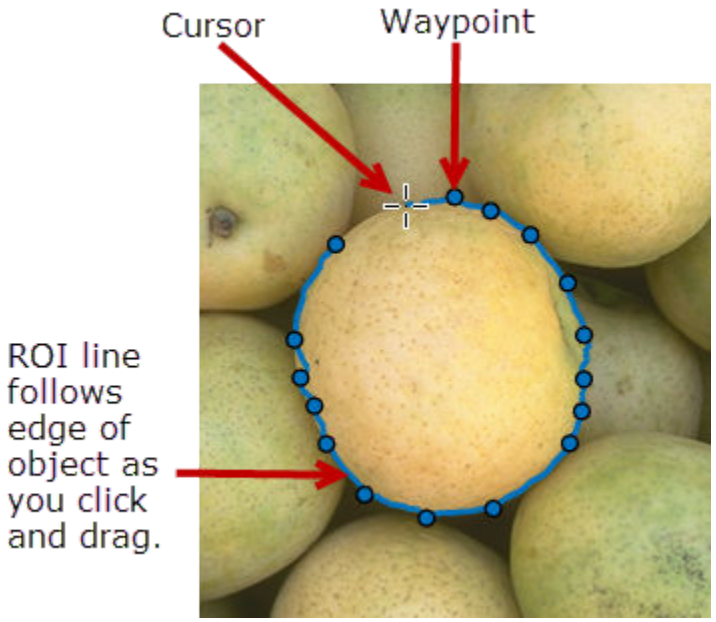
Create customizable freehand ROI with assistance from object edges

Syntax

```
roi = drawassisted
roi = drawassisted(hImage)
roi = drawassisted( ___ ,Name,Value)
```

Description

The `drawassisted` function creates a `AssistedFreehand` object that specifies the shape and position of a freehand region of interest (ROI) that follows the contours of objects in the image. You can create the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments. You can also specify the initial appearance and behavior of the ROI.



After you create the ROI, you can use object properties, object functions, and event notifications to customize the shape, position, appearance, and behavior of the ROI. For more information about using these capabilities, see “Tips” on page 1-793.

`roi = drawassisted` creates an `AssistedFreehand` object and enables interactive drawing of the hand-drawn region-of-interest (ROI) on the current axes. The `AssistedFreehand` ROI uses the edges in the underlying image to "assist" you as you draw the shape.

To draw the ROI, position the pointer on the image, click and release to place the first vertex (waypoint), and then move the pointer to draw a line. As you move the pointer to draw the shape, the line follows the contours of edges in the underlying image automatically. As you draw, click to place vertices along the line. To finish the ROI and close the shape, double-click. For more information

about using the ROI, including keyboard shortcuts and context menu options, see “Tips” on page 1-793.

`roi = drawassisted(hImage)` creates the ROI on the image specified by `hImage`.

`roi = drawassisted( ___ ,Name,Value)` modifies the appearance and behavior of the ROI using one or more name-value arguments.

Examples

Alpha Blend Source ROI into Target Image

Read an image into the workspace and display it.

```
im = imread('peppers.png');
imshow(im)
```

Draw an assisted freehand ROI.

```
h = drawassisted;
```

Create a mask of the ROI.

```
bw = createMask(h);
```

Create an alpha matrix that specifies the transparency of the source image at each pixel.

```
alphamat = imguidedfilter(single(bw),im,'DegreeOfSmoothing',2);
```

Display a target image.

```
target = imread('fabric.png');
imshow(target)
```

Resize the source image and the alpha matrix to the same size as the target image.

```
alphamat = imresize(alphamat,[size(target,1),size(target,2)]);
im = imresize(im,[size(target,1),size(target,2)]);
```

Alpha blend the source ROI into the target image.

```
fused = single(im).*alphamat + (1-alphamat).*single(target);
fused = uint8(fused);
imshow(fused)
```

Set Up Listener for AssistedFreehand Events

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Display the image. Use the `imshow` return value to get a handle to the image displayed. To create an `AssistedFreehand` ROI requires an underlying image.

```
img = imshow(I);
```

Draw an assisted freehand ROI on the image, with assistance from the underlying image.

```
roi = drawassisted(img,'Color','r');
```

Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays the previous position and the current position of the ROI.

```
function allevents(src,evt)
    evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving current position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved current position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

Input Arguments

`hImage` — Image on which to draw ROI

`Image` object

Image on which to draw the ROI, specified as an `Image` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `h = drawassisted(Color="y")` creates a yellow colored `AssistedFreehand` object

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `h = drawassisted("Color","y")` creates a yellow colored `AssistedFreehand` object

`Closed` — Close freehand ROI
`true` or `1` (default) | `false` or `0`

Close the freehand ROI, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `true`, the `drawassisted` function closes the ROI by connecting the last waypoint drawn to the first waypoint drawn.

`Color` — ROI color
`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

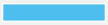
You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	

RGB Triplet	Appearance
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0.4470 0.7410]

ContextMenu — Context menu
ContextMenu object

Context menu that displays when you right-click the ROI, specified as a ContextMenu object. You can create a custom context menu by using the uicontextmenu function and then configuring context menu properties.

Deletable — Context menu provides option to delete the ROI
true or 1 (default) | false or 0

Context menu provides an option to delete the ROI, specified as a numeric or logical 1 (true) or 0 (false). When the value is true, you can delete the ROI interactively using the context menu. When the value is false, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the delete function.

FaceAlpha — Transparency of ROI face
0.2 (default) | number in the range [0, 1]

Transparency of the ROI face, specified as a number in the range [0, 1]. When the value is 1, the ROI face is completely opaque. When the value is 0, the ROI face is completely transparent.

FaceSelectable — ROI face can capture clicks
true or 1 (default) | false or 0

ROI face can capture clicks, specified as a numeric or logical 1 (true) or 0 (false). When true, the ROI face captures mouse clicks. When false, the ROI face does not capture mouse clicks.

HandleVisibility — Visibility of ROI handle in Children property of parent
'on' (default) | 'off' | 'callback'

Visibility of the ROI handle in the Children property of the parent, specified as one of the values in this table.

Value	Description
'on'	The object handle is always visible (default).
'off'	The object handle is hidden at all times.
'callback'	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

Image — Image on which to draw ROI
Image object

Image on which to draw the ROI, specified as an `Image` object.

`InteractionsAllowed` — Interactivity of ROI
`'all'` (default) | `'none'` | `'reshape'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable (default).
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area but not reshaped.
`'reshape'`	The ROI can be reshaped but not translated.

`Label` — ROI label
`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background
1 (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color
`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'LabelTextColor','r'

Example: 'LabelTextColor','green'

Example: 'LabelTextColor',[0 0.4470 0.7410]

LabelVisible — Label visibility
'on' (default) | 'off' | 'hover'

Label visibility, specified as one of these values.

Value	Description
'on'	Label is visible when the ROI is visible.
'hover'	Label is visible only when the mouse is hovering over the ROI.
'off'	Label is not visible.

LineWidth — Width of ROI border
positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

MarkerSize — Marker size
positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

Parent — ROI parent
Axes object | UIAxes object

ROI parent, specified as an Axes or UIAxes object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Position — Position of ROI
n-by-2 numeric matrix

Position of the ROI, specified as an *n*-by-2 numeric matrix where *n* is the number of vertices or points defining the ROI. Each row represents the [*x* *y*] coordinates of a vertex or point.

`Selected` — Selection state of ROI

`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`).

`SelectedColor` — Color of ROI when selected

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'SelectedColor','r'`

Example: `'SelectedColor','green'`

Example: `'SelectedColor',[0 0.4470 0.7410]`

`Smoothing` — Smoothing applied to edge of ROI

`1` (default) | nonnegative number

Smoothing applied to the edge of the ROI after interactive placement, specified as a nonnegative number. The `drawassisted` function filters the *x* and *y* coordinates of the ROI using a Gaussian smoothing kernel with a default standard deviation of 1. The size of the Gaussian filter is `2*ceil(2*Smoothing)+1`. You can see the smoothing effect only after completing the drawing.

`StripeColor` — Color of ROI stripe

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

`Tag` — Tag to associate with ROI

`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI
any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `drawassisted` object does not use this data.

`Visible` — ROI visibility
`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

`Waypoints` — Position point is waypoint
n-by-1 logical vector

Position point is waypoint, specified as an *n*-by-1 logical vector where *n* is the number of points defining the ROI. The length of `Waypoints` must match the number of rows of the `Position` name-value argument. Elements in `Waypoints` with the value `true` identify points in the `Position` matrix that are waypoints.

Waypoints appear as circular shapes on the ROI edge. You can use waypoints to reshape the ROI by clicking and dragging the waypoint with the mouse. Moving waypoints modifies the freehand-drawn region between the waypoint that you clicked and the adjacent waypoints.

Output Arguments

`roi` — Assisted freehand ROI
`AssistedFreehand` object

Assisted freehand ROI, returned as an `AssistedFreehand` object.

Tips

- This table describes how to perform common tasks with the `AssistedFreehand` ROI.

Behavior	Keyboard shortcut
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.

Behavior	Keyboard shortcut
Finish drawing (close) the ROI.	Double-click, which adds a point at the pointer position and draws a line connecting this point to the first point drawn, closing the ROI. Right-click, which draws a line connecting the last point to the first point drawn. Position the pointer over the first point and click. Press **Enter**, which draws a line connecting the last point to the first point drawn.
Resize (reshape) the ROI.	Position pointer over a waypoint and then click and drag. No assistance (snapping to edges) is available in this mode.
Add a waypoint.	Position the pointer on an edge of the ROI, right-click, and select **Add Waypoint**. You can also position the pointer on an edge of the ROI and double-click.
Remove a waypoint.	Position the pointer on a waypoint, right-click, and select **Remove Waypoint**.
Move the ROI.	Position the pointer over the ROI. The pointer changes to the fleur shape. Click and drag to move the ROI.
Delete an ROI.	Position the pointer on the ROI (not on a vertex), right-click, and select **Delete Freehand** from the context menu. You can also delete the ROI programmatically using the `delete` function.

- The `drawassisted` function creates an `AssistedFreehand` object. After you create the object, you can modify the shape, position, appearance, and behavior of the ROI by using these object capabilities.

Capability	Support
Object properties	ROI objects have properties that specify their shape, position, appearance, and behavior. After you create the ROI object, change properties using dot notation. For example, to change the color of the `roi` to yellow, set its `Color` property: `roi.Color = 'yellow'`
Object functions	ROI objects have object functions that operate on the ROIs. For example, if you want to pause the MATLAB command line after creating an ROI, use the `wait` function.

Capability	Support
Event notifications	ROI objects can notify your code when certain events occur, such as when the ROI is clicked or when the ROI is being moved. To receive event notifications, set up listeners. When the ROI notifies your application through the listener, it returns data specific to the event. For example, with the `ROIMoved` event, the ROI object returns its previous position and its current position. You can specify a callback function that executes when an event occurs.

For an example of using event listeners with the `AssistedFreehand` object, see “Set Up Listener for AssistedFreehand Events” on page 1-785.

Compatibility Considerations

`UIContextMenu` name-value argument is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` name-value argument to assign a context menu to an ROI object is not recommended. Use the `ContextMenu` name-value argument instead. The values are the same.

There are no plans to remove support for the `UIContextMenu` name-value argument at this time.

See Also

`AssistedFreehand` | `drawfreehand` | `drawpolygon` | `drawpolyline`

Topics

“Create ROI Shapes”
“Using ROIs in Apps Created with App Designer”
“Use Wait Function After Drawing ROI”

Introduced in R2018b

drawcircle

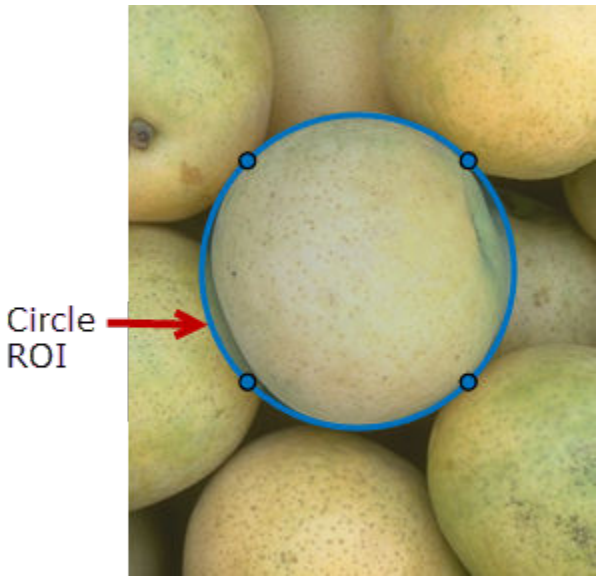
Create customizable circular ROI

Syntax

```
roi = drawcircle
roi = drawcircle(ax)
roi = drawcircle( ___ ,Name,Value)
```

Description

The `drawcircle` function creates a `Circle` object that specifies the size and position of a circular region of interest (ROI). You can create the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments. You can also specify the initial appearance and behavior of the ROI.



After you create the ROI, you can use object properties, object functions, and event notifications to customize the shape, position, appearance, and behavior of the ROI. For more information about using these capabilities, see “Tips” on page 1-809.

`roi = drawcircle` creates a `Circle` ROI object and enables interactive drawing of the ROI on the current axes.

To draw the ROI, position the pointer on the image. The cursor changes to a fleur shape. Click and drag to draw the circular ROI. To finish the ROI, release the mouse button. For more information about using the ROI, including keyboard shortcuts and context menu options, see “Tips” on page 1-809.

`roi = drawcircle(ax)` creates the ROI on the axes specified by `ax`.

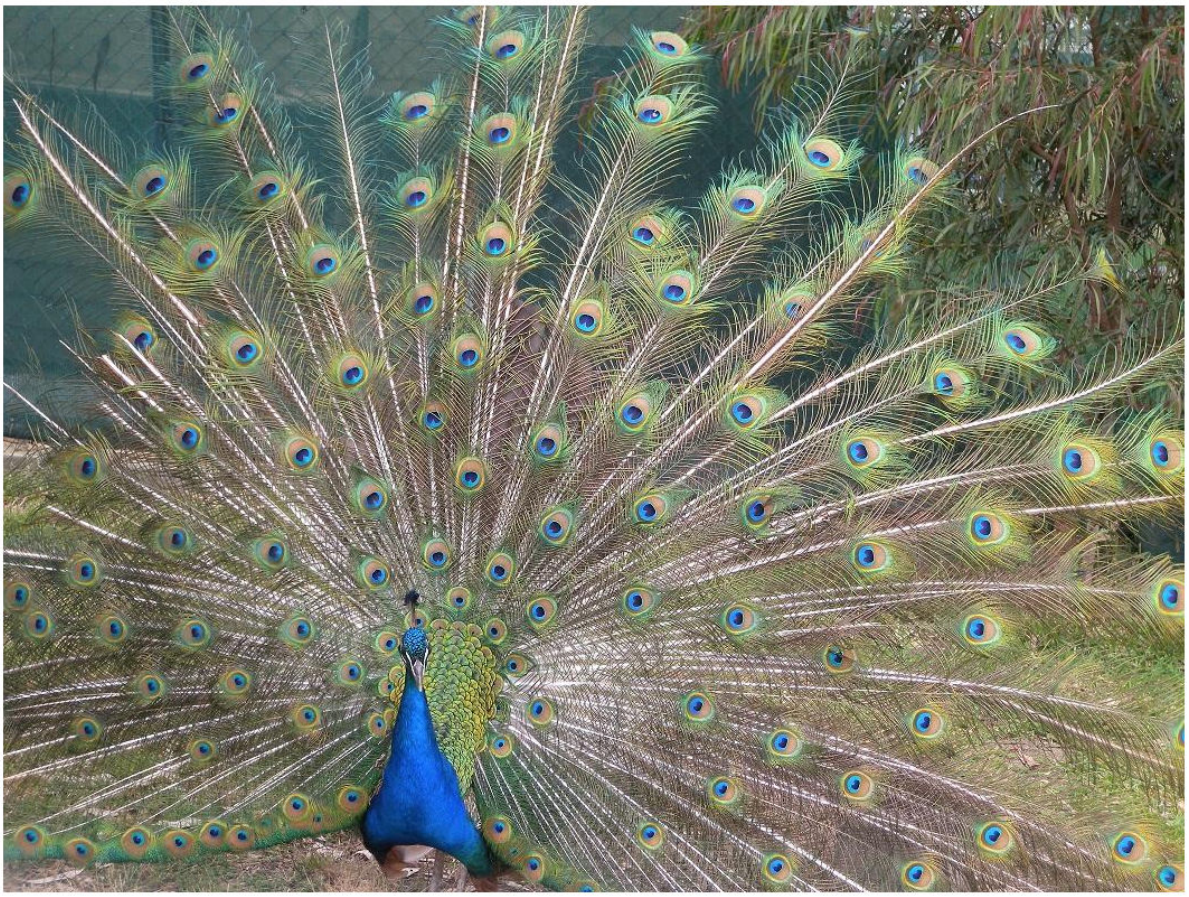
`roi = drawcircle( ___ ,Name,Value)` modifies the appearance and behavior of the ROI using one or more name-value arguments.

Examples

Create Black Circular ROI

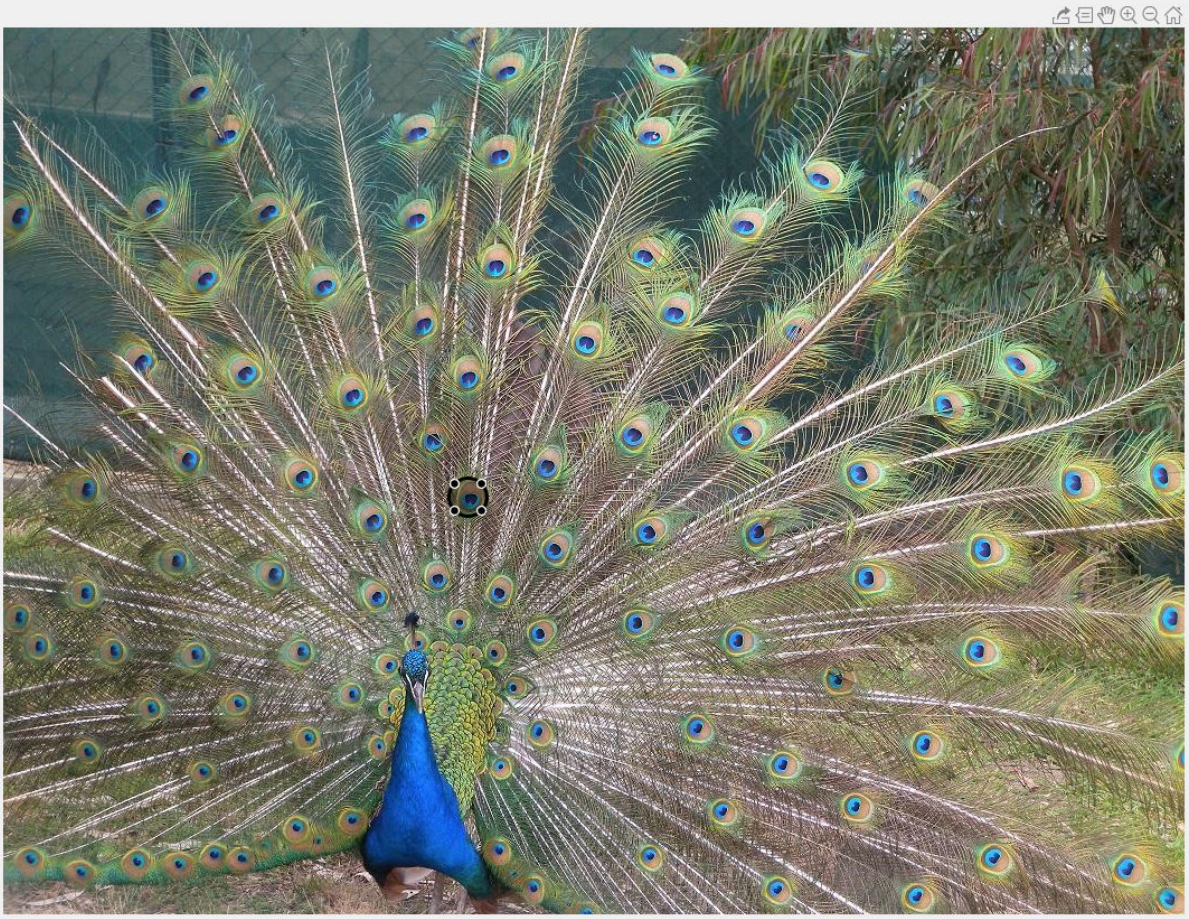
Read an image into the workspace and display it.

```
imshow(imread('peacock.jpg'))
```



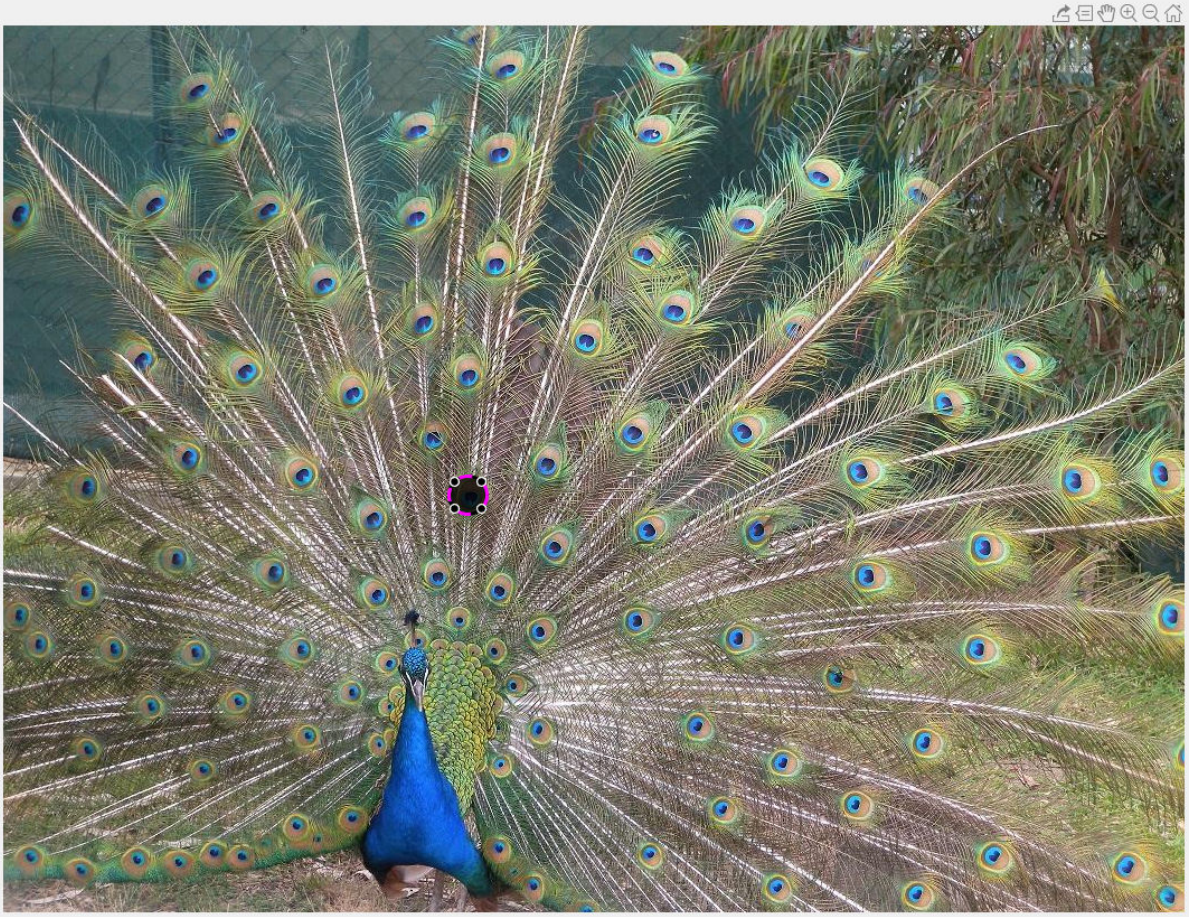
Interactively draw a partially-opaque black circular ROI.

```
h = drawcircle('Color','k','FaceAlpha',0.4);
```



Change the stripe color of the ROI to magenta, then increase the opacity of the ROI.

```
h.StripeColor = 'magenta';
h.FaceAlpha = 0.8;
```



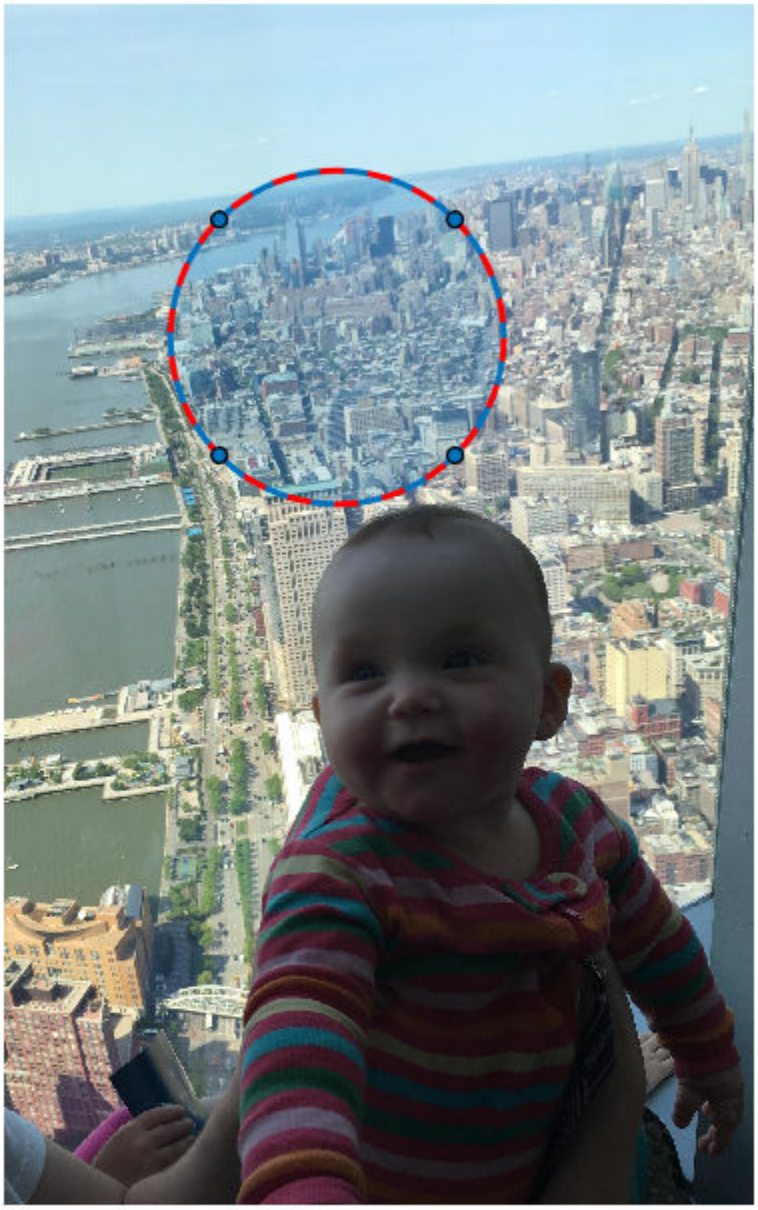
Create Circular ROI Programmatically

Read an image into the workspace and display it.

```
I = imread('baby.jpg');
figure
imshow(I)
```

Draw a circular ROI on the image, Use the `'Center'` name-value pair to specify the location of the circle and the `'Radius'` name-value pair to specify its size. Set the edge of the circle to be striped by specifying the `'StripeColor'` name-value pair.

```
h = drawcircle('Center',[1000,1000],'Radius',500,'StripeColor','red');
```



Set Up Listener for Circle ROI Events

Read an image into the workspace.

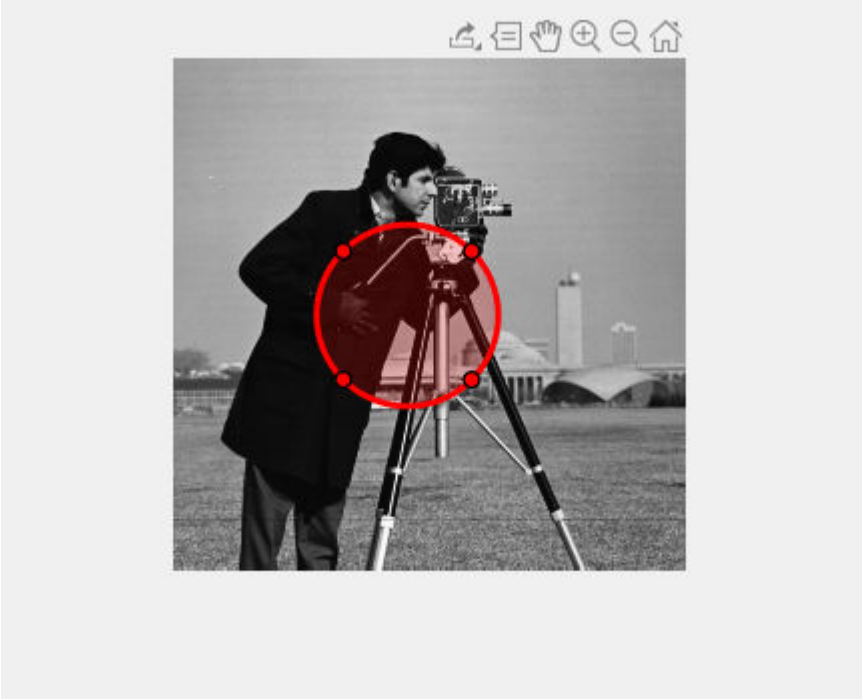
```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a circular ROI on the image.

```
roi = drawcircle('Color','r');
```



Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays the previous position and the current position of the ROI.

```
function allevents(src,evt)
    evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving current position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved current position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Color','y'` creates a yellow colored `Circle` object

`Center` — Center of ROI

1-by-2 numeric vector

Center of the ROI, specified as a 1-by-2 numeric vector of the form [*x* *y*].

`Color` — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

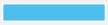
You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	

RGB Triplet	Appearance
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0.4470 0.7410]

ContextMenu — Context menu
ContextMenu object

Context menu that displays when you right-click the ROI, specified as a ContextMenu object. You can create a custom context menu by using the uicontextmenu function and then configuring context menu properties.

Deletable — Context menu provides option to delete the ROI
true or 1 (default) | false or 0

Context menu provides an option to delete the ROI, specified as a numeric or logical 1 (true) or 0 (false). When the value is true, you can delete the ROI interactively using the context menu. When the value is false, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the delete function.

DrawingArea — Area of axes in which you can interactively place ROI
'auto' (default) | 'unlimited' | [x,y,w,h]

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
'auto'	The drawing area is the current axes limits (default).
'unlimited'	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
[x,y,w,h]	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

FaceAlpha — Transparency of ROI face
0.2 (default) | number in the range [0, 1]

Transparency of the ROI face, specified as a number in the range [0, 1]. When the value is 1, the ROI face is completely opaque. When the value is 0, the ROI face is completely transparent.

FaceSelectable — ROI face can capture clicks
true or 1 (default) | false or 0

ROI face can capture clicks, specified as a numeric or logical 1 (true) or 0 (false). When true, the ROI face captures mouse clicks. When false, the ROI face does not capture mouse clicks.

`HandleVisibility` — Visibility of ROI handle in `Children` property of parent
`'on'` (default) | `'off'` | `'callback'`

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI
`'all'` (default) | `'none'` | `'reshape'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable (default).
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area but not reshaped.
`'reshape'`	The ROI can be reshaped but not translated.

`Label` — ROI label
`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background
1 (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color
`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'LabelTextColor','r'

Example: 'LabelTextColor','green'

Example: 'LabelTextColor',[0 0.4470 0.7410]

LabelVisible — Label visibility
'on' (default) | 'off' | 'hover'

Label visibility, specified as one of these values.

Value	Description
'on'	Label is visible when the ROI is visible.
'hover'	Label is visible only when the mouse is hovering over the ROI.
'off'	Label is not visible.

LineWidth — Width of ROI border
positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

MarkerSize — Marker size
positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

`Parent` — ROI parent
`Axes` object | `UIAxes` object

ROI parent, specified as an `Axes` or `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

`Radius` — Radius of circle
nonnegative number

Radius of the circle, specified as a nonnegative number.

`Selected` — Selection state of ROI
`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`).

`SelectedColor` — Color of ROI when selected
`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	

RGB Triplet	Appearance
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'SelectedColor','r'

Example: 'SelectedColor','green'

Example: 'SelectedColor',[0 0.4470 0.7410]

StripeColor — Color of ROI stripe
'none' (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or 'none'. If you specify 'none', then the ROI edge is a solid color specified by Color. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by Color and StripeColor.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

`Tag` — Tag to associate with ROI
`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI
any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `drawcircle` object does not use this data.

`Visible` — ROI visibility
`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Output Arguments

`roi` — Circular ROI
`Circle` object

Circular ROI, returned as a `Circle` object.

Tips

- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Resize the ROI.	Position the pointer over one of the vertices on the circle and then click and drag. The aspect ratio of the ROI remains constant (1:1).
Move the ROI.	Position the pointer over the ROI. The pointer changes to the fleur shape. Click and drag the ROI.

Behavior	Keyboard shortcut
Delete the ROI.	Position the pointer over the ROI and right-click to view its context menu. Select **Delete Circle** from the menu. You can also delete the ROI using the `delete` function.

- The `drawcircle` function creates a `Circle` object. After you create the object, you can modify the shape, position, appearance, and behavior of the ROI by using these object capabilities.

Capability	Support
Object properties	ROI objects have properties that specify their shape, position, appearance, and behavior. After you create the ROI object, change properties using dot notation. For example, to change the color of the `roi` to yellow, set its `Color` property: `roi.Color = 'yellow'`
Object functions	ROI objects have object functions that operate on the ROIs. For example, if you want to pause the MATLAB command line after creating an ROI, use the `wait` function.
Event notifications	ROI objects can notify your code when certain events occur, such as when the ROI is clicked or when the ROI is being moved. To receive event notifications, set up listeners. When the ROI notifies your application through the listener, it returns data specific to the event. For example, with the `ROIMoved` event, the ROI object returns its previous position and its current position. You can specify a callback function that executes when an event occurs.

  For an example of using event listeners with the `Circle` object, see “Set Up Listener for Circle ROI Events” on page 1-801.

Compatibility Considerations

`UIContextMenu` name-value argument is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` name-value argument to assign a context menu to an ROI object is not recommended. Use the `ContextMenu` name-value argument instead. The values are the same.

There are no plans to remove support for the `UIContextMenu` name-value argument at this time.

See Also

`Circle` | `drawellipse`

Topics
“Create ROI Shapes”
“Using ROIs in Apps Created with App Designer”
“Use Wait Function After Drawing ROI”

Introduced in R2018b

drawcrosshair

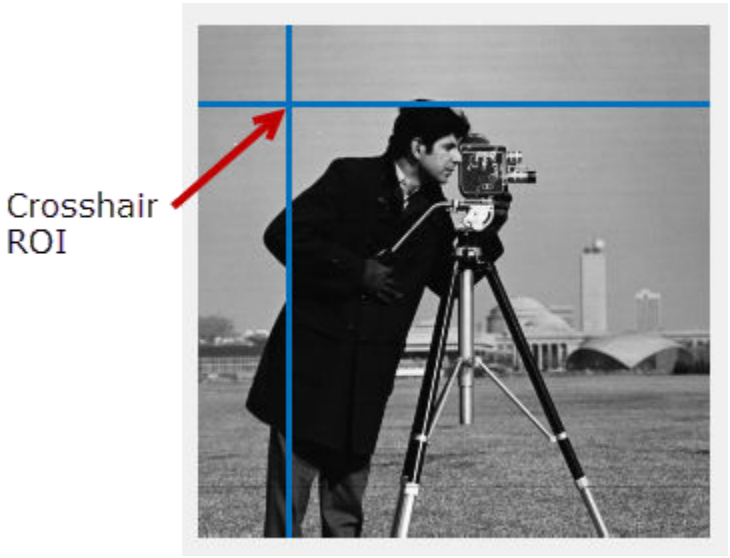
Create customizable crosshair ROI

Syntax

```
roi = drawcrosshair
roi = drawcrosshair(ax)
roi = drawcrosshair( ___ ,Name,Value)
```

Description

The `drawcrosshair` function creates a `Crosshair` object that specifies the position of a crosshair region of interest (ROI). You can create the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments. You can also specify the initial appearance and behavior of the ROI.



After you create the ROI, you can use object properties, object functions, and event notifications to customize the shape, position, appearance, and behavior of the ROI. For more information about using these capabilities, see “Tips” on page 1-823.

`roi = drawcrosshair` creates a `Crosshair` object and enables interactive drawing of the ROI on the current axes. The crosshair ROI is made up of two perpendicular lines that are the full width and height of the axes.

To draw the ROI, move the cursor over the axes and click. For more information about using the ROI, including keyboard shortcuts and context menu options, see “Tips” on page 1-823.

`roi = drawcrosshair(ax)` begins interactive placement of an ROI in the axes specified by `ax`.

`roi = drawcrosshair( ___ ,Name,Value)` customizes the appearance and behavior of the ROI using one or more name-value pairs. Unspecified name-value pairs are set to the default value.

Examples

Create Crosshair ROI and Modify ROI Properties

This example shows how to create a crosshair ROI both interactively and programmatically.

Create Crosshair ROI

Read an image into the workspace and display it.

```
figure;
imshow('pears.png')
```

Create a crosshair ROI on the image using the `drawcrosshair` function interactively. Move the cursor over the image anywhere and click to draw the ROI. Click on the ROI Center (the point where the horizontal line crosses the vertical line) to move the ROI on the image.

```
h = drawcrosshair();
```

Create Crosshair ROI Programmatically

Read an image into the workspace and display it.

```
figure;
imshow('pears.png')
```

Create a crosshair ROI on the image using the `drawcrosshair` function. Use Name/Value pair arguments to specify the initial position of the ROI.

```
h = drawcrosshair('Position',[100 100]);
```

Modify Crosshair ROI Appearance Using Properties

Read an image into the workspace and display it.

```
figure;
imshow('pears.png')
```

Create a crosshair ROI on the image using the `drawcrosshair` function. Use Name/Value pair arguments to specify the initial position of the ROI.

```
h = drawcrosshair('Position',[100 100]);
```

Use properties of the Crosshair object to change the lines in the ROI to be striped. The drawcrosshair function returns a `Crosshair` object that supports many properties. Use the `StripeColor` property to specify the stripe color.

```
h.StripeColor = 'green';
```

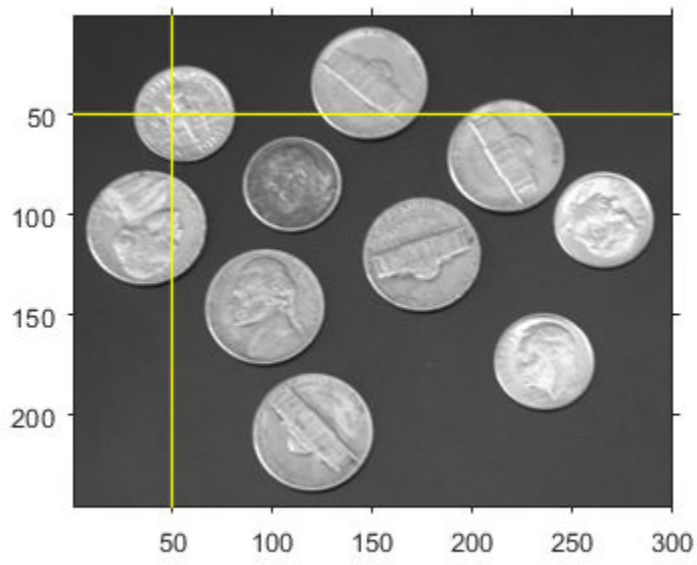
Display Value of Pixel Specified By Crosshair ROI

Read an image into the workspace and display it.

```
img = imread('coins.png');
hAx = gca;
imObj = imshow(img,'Parent',hAx);
imObj.Parent.Visible = 'on';
```

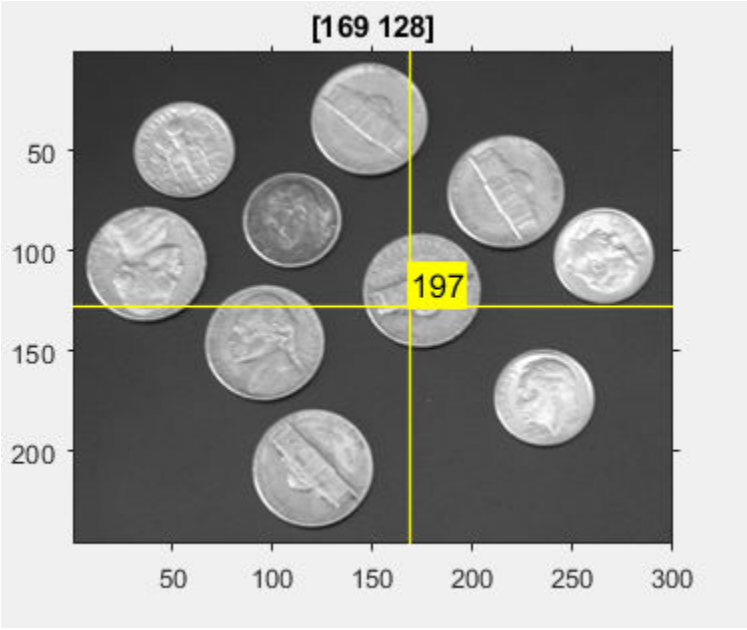
Create a crosshair ROI.

```
h = drawcrosshair('Parent',hAx,'Position',[50 50],'LineWidth',1,'Color','y');
```



Use the `addlistener` object function to receive notification when the ROI moves. Specify the callback function to execute when the event occurs. When you move the crosshair ROI, the code displays the position of the crosshair in the title and displays the pixel value at the location in the ROI label.

```
addlistener(h,'MovingROI',@(src,data)displayInfo(src,data,hAx,img));
```



This is the callback function that displays the value of the pixel specified by the crosshair ROI.

```
function displayInfo(src,data,hAx,img)
pos = ceil(data.CurrentPosition);
pixval = img(pos(2),pos(1));
src.Label = mat2str(pixval);
title(mat2str(pos));
end
```

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Color','y'` creates a yellow colored `Crosshair` object

Color — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0.4470 0.7410]

ContextMenu — Context menu
ContextMenu object

Context menu that displays when you right-click the ROI, specified as a ContextMenu object. You can create a custom context menu by using the uicontextmenu function and then configuring context menu properties.

Deletable — Context menu provides option to delete the ROI
true or 1 (default) | false or 0

Context menu provides an option to delete the ROI, specified as a numeric or logical 1 (true) or 0 (false). When the value is true, you can delete the ROI interactively using the context menu. When the value is false, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the delete function.

DrawingArea — Area of axes in which you can interactively place ROI
'auto' (default) | 'unlimited' | [x,y,w,h]

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
'auto'	The drawing area is the current axes limits (default).
'unlimited'	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
[x,y,w,h]	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

HandleVisibility — Visibility of ROI handle in Children property of parent
'on' (default) | 'off' | 'callback'

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI

`'all'` (default) | `'none'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable.
`'none'`	The ROI is not interactable, and no drag points are visible.

`Label` — ROI label

`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background

1 (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color

`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	

Color Name	Short Name	RGB Triplet	Appearance
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'LabelTextColor','r'

Example: 'LabelTextColor','green'

Example: 'LabelTextColor',[0 0.4470 0.7410]

LabelVisible — Label visibility
'on' (default) | 'off' | 'hover'

Label visibility, specified as one of these values.

Value	Description
'on'	Label is visible when the ROI is visible.
'hover'	Label is visible only when the mouse is hovering over the ROI.
'off'	Label is not visible.

LineWidth — Width of ROI border
positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

Parent — ROI parent
Axes object | UIAxes object

ROI parent, specified as an Axes or UIAxes object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Position — Position of ROI
1-by-2 numeric vector

Position of the ROI, specified as a 1-by-2 numeric vector of the form [*x* *y*]. The values *x* and *y* specify coordinates where the horizontal line crosses the vertical line in the crosshair ROI.

`Selected` — Selection state of ROI

`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`).

`SelectedColor` — Color of ROI when selected

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'SelectedColor','r'`

Example: `'SelectedColor','green'`

Example: `'SelectedColor',[0 0.4470 0.7410]`

`StripeColor` — Color of ROI stripe

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

Tag — Tag to associate with ROI
`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI
any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `drawcrosshair` object does not use this data.

`Visible` — ROI visibility
`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Output Arguments

`roi` — Crosshair ROI
`Crosshair` object

Crosshair ROI, returned as a `Crosshair` object.

Tips

- This table describes how to perform common tasks with a crosshair ROI.

Task	Description
Cancel the drawing operation.	Start drawing the ROI and press **Esc** before releasing the mouse. The function returns a valid ROI object with an empty `Position` property.
Move the ROI.	Position the pointer over the ROI. The pointer changes to a fleur shape. Click and drag to move the ROI.
Delete the ROI.	Position the pointer over the ROI and right-click to view its context menu. Select **Delete Crosshair** from the menu. You can also delete the ROI using the `delete` function.

- The `drawcrosshair` function creates a `Crosshair` object. After you create the object, you can modify the position, appearance, and behavior of the ROI by using these object capabilities.

Capability	Support
Object properties	ROI objects have properties that specify their shape, position, appearance, and behavior. After you create the ROI object, change properties using dot notation. For example, to change the color of the `roi` to yellow, set its `Color` property: `roi.Color = 'yellow'`
Object functions	ROI objects have object functions that operate on the ROIs. For example, if you want to pause the MATLAB command line after creating an ROI, use the `wait` function.
Event notifications	ROI objects can notify your code when certain events occur, such as when the ROI is clicked or when the ROI is being moved. To receive event notifications, set up listeners. When the ROI notifies your application through the listener, it returns data specific to the event. For example, with the `ROIMoved` event, the ROI object returns its previous position and its current position. You can specify a callback function that executes when an event occurs.

For an example of using event listeners with the `Crosshair` object, see “Display Value of Pixel Specified By Crosshair ROI” on page 1-815.

Compatibility Considerations

`UIContextMenu` name-value argument is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` name-value argument to assign a context menu to an ROI object is not recommended. Use the `ContextMenu` name-value argument instead. The values are the same.

There are no plans to remove support for the `UIContextMenu` name-value argument at this time.

See Also

`Crosshair` | `drawline` | `drawpoint` | `drawpolyline`

Topics

“Create ROI Shapes”

Introduced in R2019b

drawcuboid

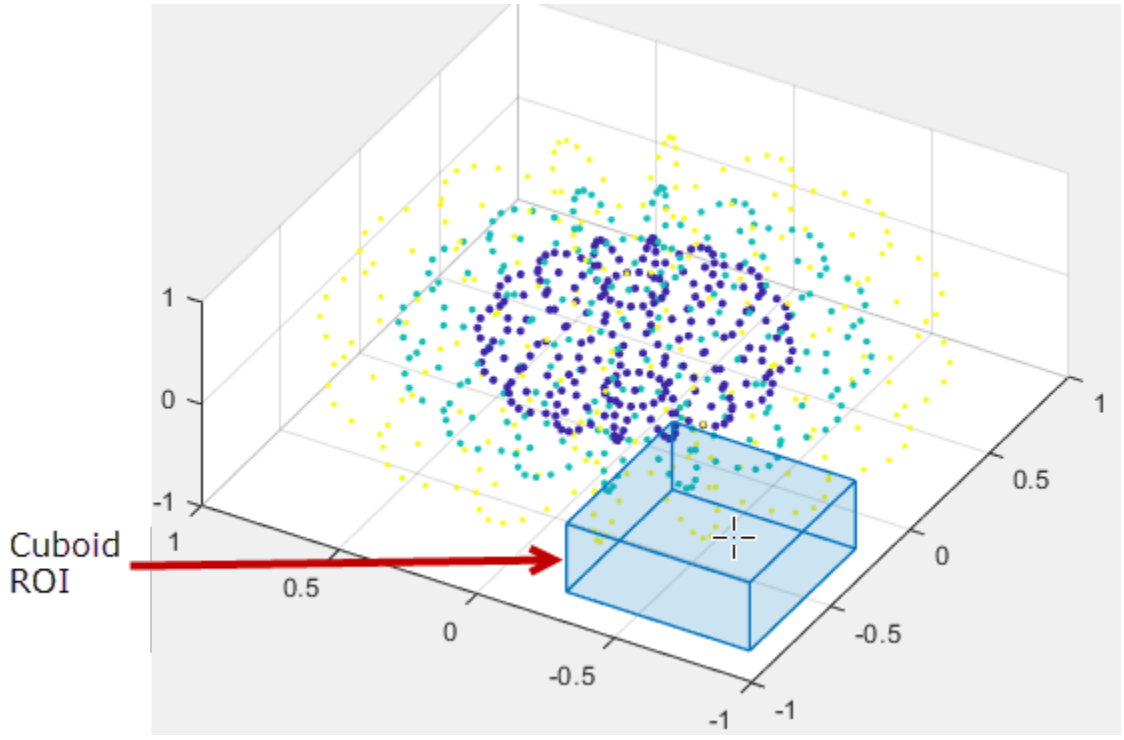
Create customizable cuboidal ROI

Syntax

```
roi = drawcuboid
roi = drawcuboid(ax)
roi = drawcuboid(S)
roi = drawcuboid( ___ ,Name,Value)
```

Description

The `drawcuboid` function creates a `Cuboid` object that specifies the shape and position of a cuboidal region of interest (ROI). You can create the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments. You can also specify the initial appearance and behavior of the ROI.



After you create the ROI, you can use object properties, object functions, and event notifications to customize the shape, position, appearance, and behavior of the ROI. For more information about using these capabilities, see “Tips” on page 1-837.

`roi = drawcuboid` creates a `Cuboid` ROI object and enables interactive drawing of a cuboidal region of interest (ROI) on the current axes.

To draw the ROI, call the `drawcuboid` function. The function draws a cuboidal ROI, centered in the volume. Move the pointer onto the image. The cursor changes to a fleur shape. Move the ROI

anywhere on the image. To finish the ROI, click the mouse button. For more information about using the ROI, including keyboard shortcuts and context menu options, see “Tips” on page 1-837.

`roi = drawcuboid(ax)` creates a `Cuboid` ROI object and enables interactive drawing of a cuboidal region of interest (ROI) on the axes specified by `ax`.

`roi = drawcuboid(S)` creates a `Cuboid` ROI object and enables interactive drawing of a cuboidal region of interest (ROI) on the `Scatter` object specified by `S`. During interactive placement, the cuboid snaps to the nearest point defined by the `Scatter` object.

`roi = drawcuboid(___,Name,Value)` modifies the appearance of the ROI using one or more name-value pairs.

Examples

Create Cuboid ROI on Scatter Plot

Create a 3-D scatter plot and interactively define a cuboid ROI over the data.

Define vectors for 3-D scatter data.

```
[x,y,z] = sphere(16);
X = [x(:)*.5 x(:)*.75 x(:)];
Y = [y(:)*.5 y(:)*.75 y(:)];
Z = [z(:)*.5 z(:)*.75 z(:)];
```

Specify the size and color of each marker.

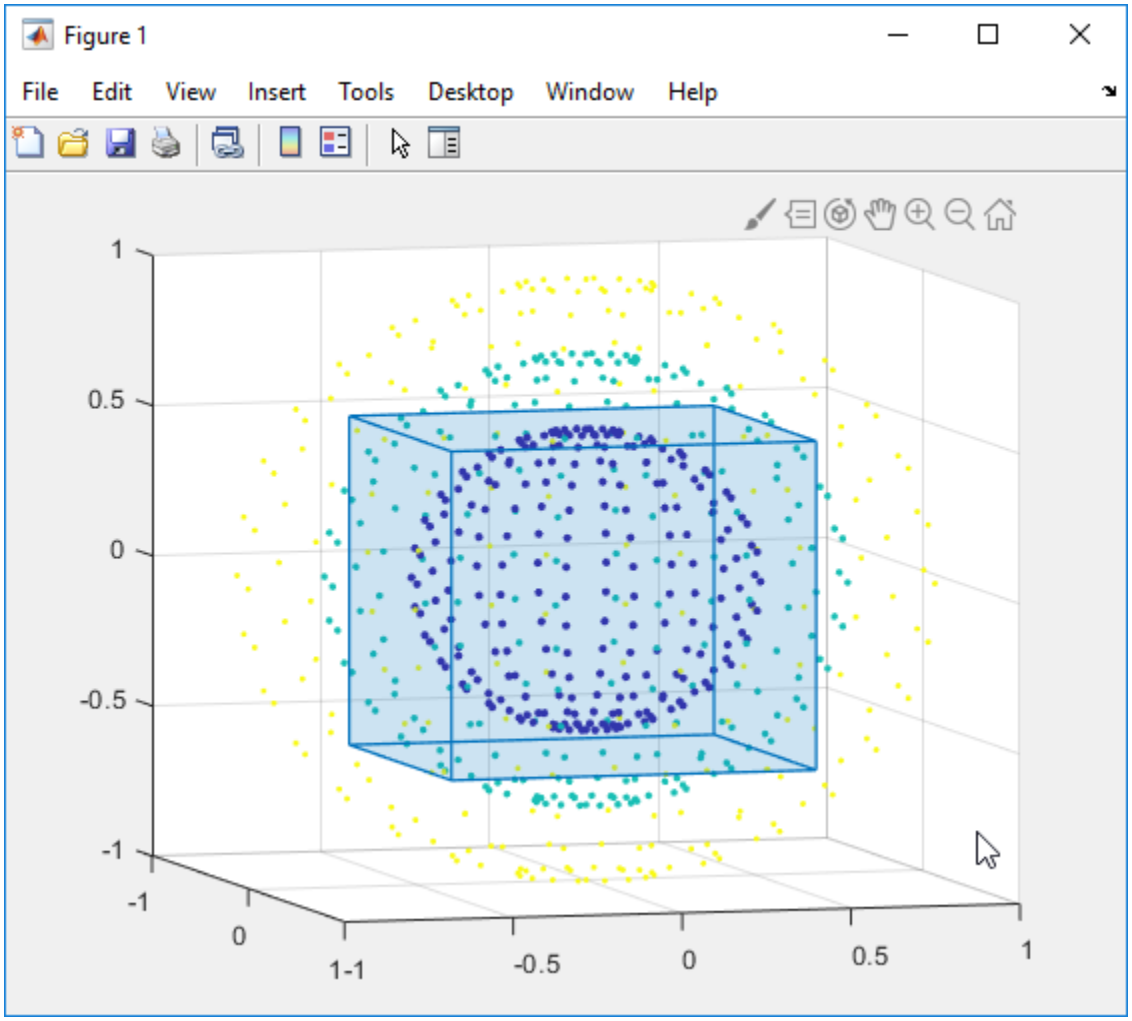
```
S = repmat([1 .75 .5]*10,numel(x),1);
C = repmat([1 2 3],numel(x),1);
```

Create a 3-D scatter plot. Use `view` to the change the angle of the axes in the figure.

```
figure
hScatter = scatter3(X(:),Y(:),Z(:),S(:),C(:),'filled');
view(-60,60);
```

Begin placing a cuboid ROI on the axes. The ROI snaps to the nearest point defined by the scatter plot. Adjust the size of the cuboid during interactive placement by using the scroll wheel.

```
drawcuboid(hScatter);
```



Set Up Listener for Cuboid ROI Events

Define vectors for 3-D scattered data.

```
[x,y,z] = sphere(16);
X = [x(:)*.5 x(:)*.75 x(:)];
Y = [y(:)*.5 y(:)*.75 y(:)];
Z = [z(:)*.5 z(:)*.75 z(:)];
```

Specify the size and color of each marker.

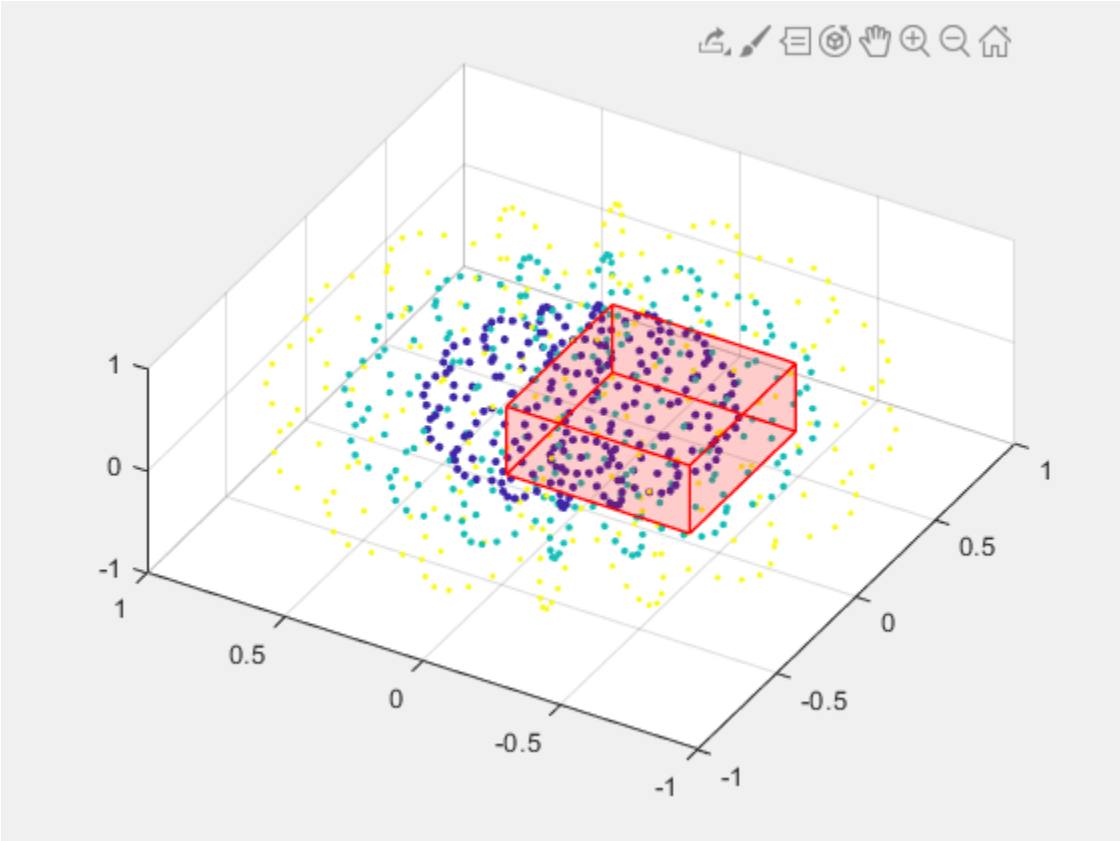
```
S = repmat([1 .75 .5]*10,numel(x),1);
C = repmat([1 2 3],numel(x),1);
```

Create a 3-D scatter plot and use view to change the angle of the axes in the figure.

```
figure
hScatter = scatter3(X(:),Y(:),Z(:),S(:),C(:),'filled');
view(-60,60);
```

Begin placing a cuboid in the axes that snaps to the nearest point from the scatter plot. Adjust the size of the cuboid during interactive placement by using the scroll wheel.

```
roi = drawcuboid(hScatter,'Color','r');
```



Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays the previous position and the current position of the ROI.

```
function allevents(src,evt)
    evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving current position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved current position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

S — Scatter plot

`Scatter` object

Scatter plot, specified as a `Scatter` object. The parent of the `Scatter` object becomes the parent of the ROI. For more information, see `scatter`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Color','y'` creates a yellow colored `Cuboid` object

Color — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	

RGB Triplet	Appearance
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0.4470 0.7410]

ContextMenu — Context menu
ContextMenu object

Context menu that displays when you right-click the ROI, specified as a ContextMenu object. You can create a custom context menu by using the uicontextmenu function and then configuring context menu properties.

Deletable — Context menu provides option to delete the ROI
true or 1 (default) | false or 0

Context menu provides an option to delete the ROI, specified as a numeric or logical 1 (true) or 0 (false). When the value is true, you can delete the ROI interactively using the context menu. When the value is false, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the delete function.

DrawingArea — Area of axes in which you can interactively place ROI
'auto' (default) | 'unlimited' | 1-by-6 numeric array

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
'auto'	The drawing area is a superset of the current axes limits and a bounding box that surrounds the ROI.
'unlimited'	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
[x,y,z,w,h,d]	The drawing area is restricted to a region beginning at (*x*,*y*,*z*), with width *w*, height *h*, and depth *d*.

EdgeAlpha — Transparency of ROI edge
1 (default) | number in the range [0, 1]

Transparency of ROI edge, specified as a number in the range [0, 1]. When set to 1, the ROI edge is completely opaque. When set to 0, the ROI edge is completely transparent.

`FaceAlpha` — Transparency of ROI faces
`0.2` (default) | number in the range [0, 1]

Transparency of the ROI faces, specified as a number in the range [0, 1]. When the value is `1`, the ROI faces are completely opaque. When the value is `0`, the ROI faces are completely transparent.

`FaceAlphaOnHover` — Transparency of ROI face directly underneath mouse pointer
`0.4` (default) | number in the range [0, 1] | `'none'`

Transparency of ROI face directly underneath the mouse pointer, specified as a number in the range [0, 1] or `'none'` to indicate no change to face transparency. When set to `1`, the face under the mouse pointer is fully opaque. When set to `0`, the face is completely transparent.

`FaceColorOnHover` — Color of ROI face directly underneath mouse pointer
`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI face directly underneath the mouse pointer, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify the value `'none'`, then the face color does not change on hover. (Hover means positioning the pointer over the surface of the cuboidal ROI.) When you are not hovering over a face of the ROI, the value of `Color` determines the face color.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	

RGB Triplet	Appearance
[0.6350 0.0780 0.1840]	

Example: 'FaceColorOnHover','r'

Example: 'FaceColorOnHover','green'

Example: 'FaceColorOnHover',[0.8500 0.3250 0.0980]

HandleVisibility — Visibility of ROI handle in Children property of parent
'on' (default) | 'off' | 'callback'

Visibility of the ROI handle in the Children property of the parent, specified as one of the values in this table.

Value	Description
'on'	The object handle is always visible (default).
'off'	The object handle is hidden at all times.
'callback'	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

InteractionsAllowed — Interactivity of ROI
'all' (default) | 'none' | 'translate'

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
'all'	The ROI is fully interactable.
'none'	The ROI is not interactable, and no drag points are visible.
'translate'	The ROI can be translated (moved) within the drawing area.

Label — ROI label
'' (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label ('').

LabelAlpha — Transparency of text background
1 (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to 1, the text background is completely opaque. When set to 0, the text background is completely transparent.

LabelTextColor — Label text color
'black' (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'LabelTextColor','r'`

Example: `'LabelTextColor','green'`

Example: `'LabelTextColor',[0 0.4470 0.7410]`

`LabelVisible` — Label visibility

`'on'` (default) | `'off'` | `'hover'`

Label visibility, specified as one of these values.

Value	Description
`'on'`	Label is visible when the ROI is visible.
`'hover'`	Label is visible only when the mouse is hovering over the ROI.
`'off'`	Label is not visible.

`LineWidth` — Width of ROI border

1 (default) | positive number

Width of the ROI border, specified as a positive number in points.

`Parent` — ROI parent
`Axes` object | `UIAxes` object

ROI parent, specified as an `Axes` or `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

`Position` — Position of cuboid
1-by-6 numeric vector

Position of the cuboid, specified as a 1-by-6 numeric vector of the form [*xmin*, *ymin*, *zmin*, *width*, *height*, *depth*].

`Rotatable` — Ability of cuboid to be rotated
`'none'` (default) | `'x'` | `'y'` | `'z'` | `'all'`

Ability of the cuboid to be rotated, specified as one of the values in this table.

Value	Description
`'all'`	ROI is fully rotatable.
`'x'`	ROI can only be rotated about the *x* axis
`'y'`	ROI can only be rotated about the *y* axis.
`'z'`	ROI can only be rotated about the *z* axis.
`'none'`	ROI is not rotatable.

`RotationAngle` — Angle of ROI rotation
`[0 0 0]` (default) | 1-by-3 numeric vector

Angle of ROI rotation, specified as a 1-by-3 numeric vector of the form [*x_angle y_angle z_angle*]. Rotation angle is measured in degrees about the *x*-, *y*-, and *z*-axis, respectively. Rotation is applied about the ROI centroid in this order: *z*, *y*, *x*.

The value of `RotationAngle` does not impact the values in `Position`. `Position` represents the cuboid before any rotation.

`ScrollWheelDuringDraw` — Ability of scroll wheel to adjust size
`'allresize'` (default) | `'xresize'` | `'yresize'` | `'zresize'` | `'none'`

Ability of the scroll wheel to adjust the size of the ROI, specified as one of the values in this table.

Value	Description
`'allresize'`	Scroll wheel impacts all ROI dimensions.
`'xresize'`	Scroll wheel impacts only the *x* dimension.
`'yresize'`	Scroll wheel impacts only the *y* dimension.
`'zresize'`	Scroll wheel impacts only the *z* dimension.
`'none'`	Scroll wheel has no effect.

`Selected` — Selection state of ROI
`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`).

`SelectedColor` — Color of ROI when selected

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'SelectedColor','r'`

Example: `'SelectedColor','green'`

Example: `'SelectedColor',[0 0.4470 0.7410]`

`StripeColor` — Color of ROI stripe

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

Tag — Tag to associate with ROI
`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI
any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `drawcuboid` object does not use this data.

`Visible` — ROI visibility
`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Output Arguments

`roi` — Cuboidal ROI
`Cuboid` object

Cuboidal ROI, returned as a `Cuboid` object.

Tips

- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Fine-tune ROI size while drawing.	Use the scroll wheel to make small changes to the size of the ROI while drawing.
Stop drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` field.
Resize (reshape) the ROI.	Position the pointer on one of the visible faces of the cuboid and click and drag the surface. You might have to rotate the cuboid to select a surface. If you press the **Shift**, dragging the mouse moves the ROI but does not change any of the dimensions.
Move the ROI.	Position the pointer on any of the visible surfaces of the ROI and click and drag while pressing **Shift**. Position the pointer on any visible surface of the ROI, right-click, and select **Lock Dimensions**. Click and drag to move the ROI.
Delete the ROI.	Position the pointer over the ROI and right-click to view its context menu. Select **Delete Cuboid** from the menu. You can also delete the ROI using the `delete` object function.

- The `drawcuboid` function creates a `Cuboid` object. After you create the object, you can modify the shape, position, appearance, and behavior of the ROI by using these object capabilities.

Capability	Support
Object properties	ROI objects have properties that specify their shape, position, appearance, and behavior. After you create the ROI object, change properties using dot notation. For example, to change the color of the `roi` to yellow, set its `Color` property: `roi.Color = 'yellow'`
Object functions	ROI objects have object functions that operate on the ROIs. For example, if you want to pause the MATLAB command line after creating an ROI, use the `wait` function.
Event notifications	ROI objects can notify your code when certain events occur, such as when the ROI is clicked or when the ROI is being moved. To receive event notifications, set up listeners. When the ROI notifies your application through the listener, it returns data specific to the event. For example, with the `ROIMoved` event, the ROI object returns its previous position and its current position. You can specify a callback function that executes when an event occurs.

For an example of using event listeners with the `Cuboid` object, see “Set Up Listener for Cuboid ROI Events” on page 1-827.

Compatibility Considerations

`UIContextMenu` name-value argument is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` name-value argument to assign a context menu to an ROI object is not recommended. Use the `ContextMenu` name-value argument instead. The values are the same.

There are no plans to remove support for the `UIContextMenu` name-value argument at this time.

See Also

`Cuboid` | `drawrectangle`

Topics

“Create ROI Shapes”
“Use Wait Function After Drawing ROI”

Introduced in R2019a

drawellipse

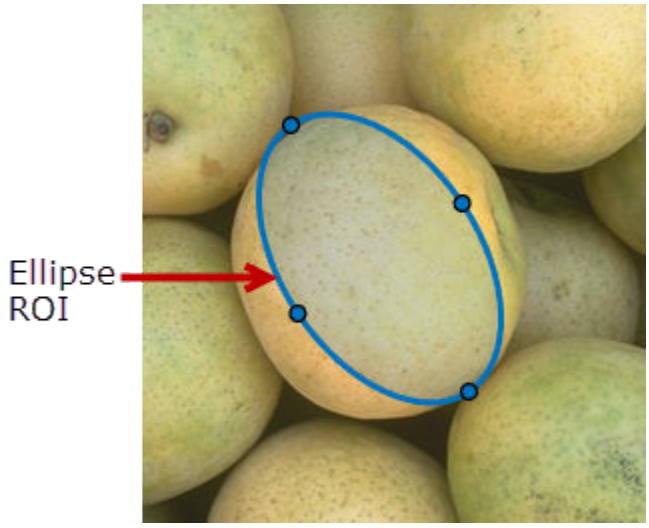
Create customizable elliptical ROI

Syntax

```
roi = drawellipse
roi = drawellipse(ax)
roi = drawellipse( ___ ,Name,Value)
```

Description

The `drawellipse` function creates an `Ellipse` object that specifies the shape and position of an elliptical region of interest (ROI). You can create the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments. You can also specify the initial appearance and behavior of the ROI.



After you create the ROI, you can use object properties, object functions, and event notifications to customize the shape, position, appearance, and behavior of the ROI. For more information about using these capabilities, see “Tips” on page 1-853.

`roi = drawellipse` creates an `Ellipse` object and enables interactive drawing of an elliptical ROI on the current axes.

To draw the ROI, position the pointer on the image. The cursor changes to a fleur shape. Click and drag to draw the elliptical ROI. To finish the ROI, release the mouse button. For more information about using the ROI, including keyboard shortcuts and context menu options, see “Tips” on page 1-853.

`roi = drawellipse(ax)` creates the ROI in the axes specified by `ax`.

`roi = drawellipse( ___ ,Name,Value)` modifies the appearance and behavior of the ROI using one or more name-value arguments.

Examples

Create Red Elliptical ROI

Read an image into the workspace and display it.

```
imshow(imread('llama.jpg'))
```

Interactively draw a red elliptical ROI.

```
h = drawellipse('Color','r');
```

Change the stripe color of the ROI to black, then increase the opacity of the ROI.

```
h.StripeColor = 'k';
h.FaceAlpha = 0.4;
```

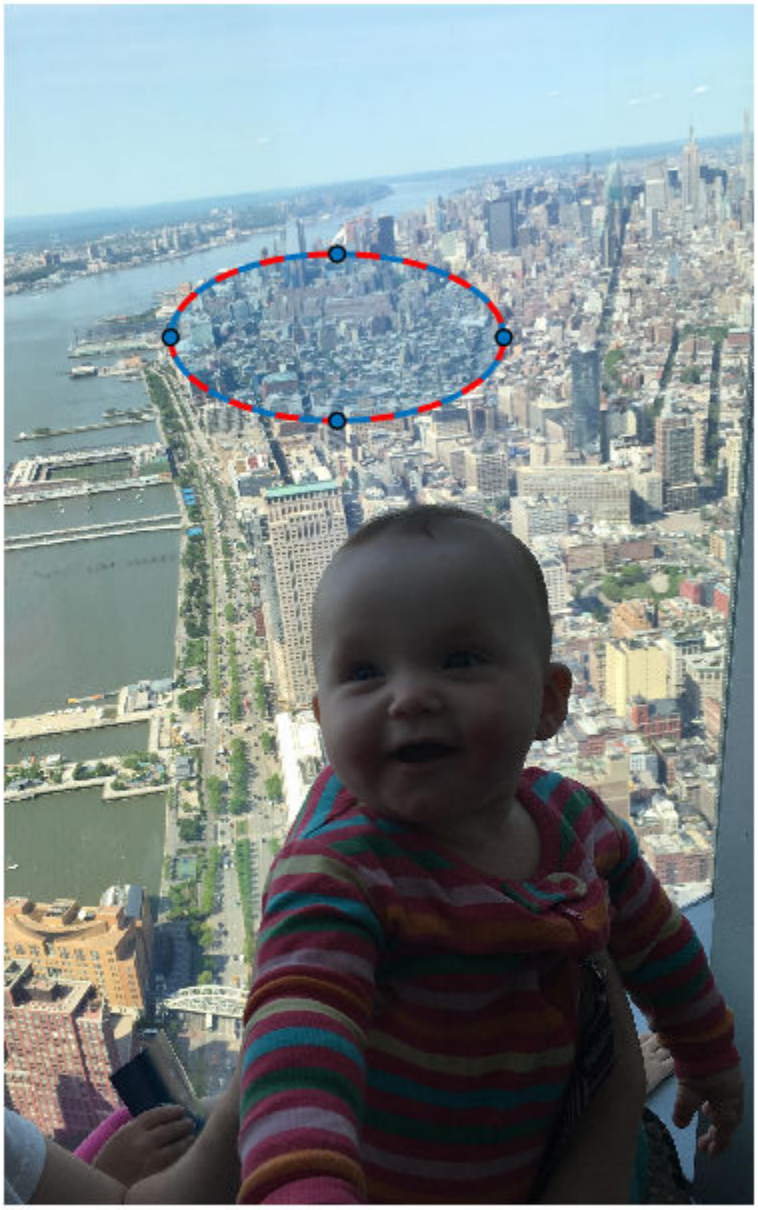
Create Elliptical ROI Programmatically

Read an image into the workspace and display it.

```
I = imread('baby.jpg');
figure
imshow(I)
```

Draw an elliptical ROI on the image. Use the `'Center'` name-value pair to specify the location of the ellipse and the `'SemiAxes'` name-value pair to specify the shape of the ellipse. Set the edge of the ellipse to be a striped red line by specifying the `'StripeColor'` name-value pair.

```
h = drawellipse('Center',[1000,1000],'SemiAxes',[500,250],'StripeColor','r');
```



Set Up Listener for Ellipse ROI Events

Read an image into the workspace.

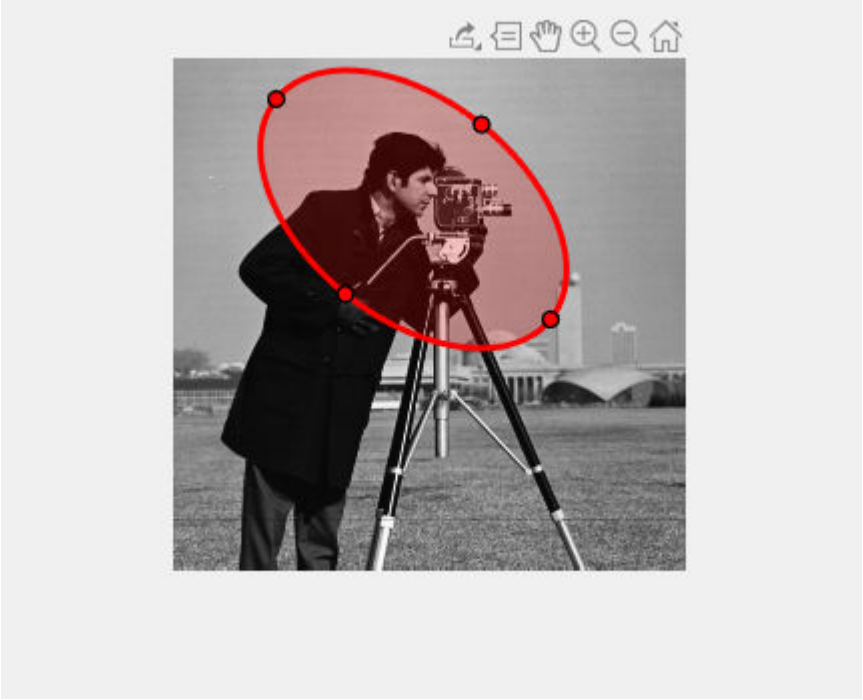
```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a circular ROI on the image.

```
roi = drawellipse('Color','r');
```



Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays the previous position and the current position of the ROI.

```
function allevents(src,evt)
    evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving current position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved current position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Color','y'` creates a yellow colored `Ellipse` object

`AspectRatio` — Aspect ratio of ellipse
nonnegative number

Aspect ratio of the ellipse, specified as a nonnegative number. The aspect ratio is defined as `SemiAxes(1)/SemiAxes(2)`.

`Center` — Center of ROI
1-by-2 numeric vector

Center of the ROI, specified as a 1-by-2 numeric vector of the form [*x y*].

`Color` — ROI color
`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	

RGB Triplet	Appearance
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'Color','r'`

Example: `'Color','green'`

Example: `'Color',[0 0.4470 0.7410]`

`ContextMenu` — Context menu
`ContextMenu` object

Context menu that displays when you right-click the ROI, specified as a `ContextMenu` object. You can create a custom context menu by using the `uicontextmenu` function and then configuring context menu properties.

`Deletable` — Context menu provides option to delete the ROI
`true` or `1` (default) | `false` or `0`

Context menu provides an option to delete the ROI, specified as a numeric or logical `1` (`true`) or `0` (`false`). When the value is `true`, you can delete the ROI interactively using the context menu. When the value is `false`, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the `delete` function.

`DrawingArea` — Area of axes in which you can interactively place ROI
`'auto'` (default) | `'unlimited'` | `[x,y,w,h]`

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
`'auto'`	The drawing area is the current axes limits (default).
`'unlimited'`	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
`[x,y,w,h]`	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

`FaceAlpha` — Transparency of ROI face
`0.2` (default) | number in the range [0, 1]

Transparency of the ROI face, specified as a number in the range [0, 1]. When the value is `1`, the ROI face is completely opaque. When the value is `0`, the ROI face is completely transparent.

`FaceSelectable` — ROI face can capture clicks
`true` or `1` (default) | `false` or `0`

ROI face can capture clicks, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `true`, the ROI face captures mouse clicks. When `false`, the ROI face does not capture mouse clicks.

`FixedAspectRatio` — Aspect ratio remains constant
`false` or `0` (default) | `true` or `1`

Aspect ratio remains constant during interaction, specified as a numeric or logical `0` (`false`) or `1` (`true`). When the value is `true`, the aspect ratio remains constant when you draw or resize the ROI. When the value is `false`, you can change the aspect ratio when drawing or resizing the ROI.

`HandleVisibility` — Visibility of ROI handle in `Children` property of parent
`'on'` (default) | `'off'` | `'callback'`

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI
`'all'` (default) | `'none'` | `'reshape'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable (default).
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area but not reshaped.
`'reshape'`	The ROI can be reshaped but not translated.

`Label` — ROI label
`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background
`1` (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color
`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'LabelTextColor','r'`

Example: `'LabelTextColor','green'`

Example: `'LabelTextColor',[0 0.4470 0.7410]`

`LabelVisible` — Label visibility

`'on'` (default) | `'off'` | `'hover'`

Label visibility, specified as one of these values.

Value	Description
`'on'`	Label is visible when the ROI is visible.
`'hover'`	Label is visible only when the mouse is hovering over the ROI.
`'off'`	Label is not visible.

`LineWidth` — Width of ROI border
positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

`MarkerSize` — Marker size
positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

`Parent` — ROI parent
`Axes` object | `UIAxes` object

ROI parent, specified as an `Axes` or `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

`RotationAngle` — Rotation angle
`0` (default) | number

Rotation angle of the ROI, specified as a number. The angle is measured in degrees in a clockwise direction around the center of the ROI.

`Selected` — Selection state of ROI
`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`).

`SelectedColor` — Color of ROI when selected
`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	

Color Name	Short Name	RGB Triplet	Appearance
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'SelectedColor','r'

Example: 'SelectedColor','green'

Example: 'SelectedColor',[0 0.4470 0.7410]

SemiAxes — Lengths of semiaxes of ellipse
1-by-2 numeric vector

Lengths of the semiaxis of the ellipse, specified as a 1-by-2 numeric vector of the form [*semiaxis1 semiaxis2*]. The `drawellipse` function assigns the length of the semiaxis that is closest to the *x* direction to *semiaxis1*.

StripeColor — Color of ROI stripe
'none' (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or 'none'. If you specify 'none', then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	

Color Name	Short Name	RGB Triplet	Appearance
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'StripeColor','r'

Example: 'StripeColor','green'

Example: 'StripeColor',[0 0.4470 0.7410]

Tag — Tag to associate with ROI
'' (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI
any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `drawellipse` object does not use this data.

`Visible` — ROI visibility
`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
'on'	Display the ROI.
'off'	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Output Arguments

`roi` — Elliptical ROI
`Ellipse` object

Elliptical ROI, returned as an `Ellipse` object.

Tips

- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Fine-tune width of ellipse as you are drawing.	As you draw the ellipse, use the scroll wheel to make small changes to the width of the ellipse.
Rotate the ROI.	Position the pointer near a vertex. The pointer changes to the rotate pointer. Click and rotate the ROI on its center. To make the rotation snap at 15 degree angles, press **Shift** as you rotate.
Maintain aspect ratio while drawing.	Hold the **Shift** key as you draw. Creates a circular ROI. To lock the aspect ratio, position the pointer on the ROI, right-click, and select **Fix Aspect Ratio** from the context menu
Resize (reshape) the ROI.	Position pointer over a vertex and then click and drag. To main the aspect ratio as you resize, Hold the **Shift** key.
Move the ROI.	Position the pointer over the ROI. The pointer changes to the fleur shape. Click and drag to move the ROI.
Delete the ROI.	Position the pointer over the ROI and right-click to view its context menu. Select **Delete Ellipse** from the menu. You can also delete the ROI using the `delete` function.

- The `drawellipse` function creates an `Ellipse` object. After you create the object, you can modify the shape, position, appearance, and behavior of the ROI by using these object capabilities.

Capability	Support
Object properties	ROI objects have properties that specify their shape, position, appearance, and behavior. After you create the ROI object, change properties using dot notation. For example, to change the color of the `roi` to yellow, set its `Color` property: `roi.Color = 'yellow'`

Capability	Support
Object functions	ROI objects have object functions that operate on the ROIs. For example, if you want to pause the MATLAB command line after creating an ROI, use the `wait` function.
Event notifications	ROI objects can notify your code when certain events occur, such as when the ROI is clicked or when the ROI is being moved. To receive event notifications, set up listeners. When the ROI notifies your application through the listener, it returns data specific to the event. For example, with the `ROIMoved` event, the ROI object returns its previous position and its current position. You can specify a callback function that executes when an event occurs.

For an example of using event listeners with the `Ellipse` object, see “Set Up Listener for Ellipse ROI Events” on page 1-844.

Compatibility Considerations

`UIContextMenu` name-value argument is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` name-value argument to assign a context menu to an ROI object is not recommended. Use the `ContextMenu` name-value argument instead. The values are the same.

There are no plans to remove support for the `UIContextMenu` name-value argument at this time.

See Also

`Ellipse` | `drawcircle`

Topics

“Create ROI Shapes”
“Using ROIs in Apps Created with App Designer”
“Use Wait Function After Drawing ROI”

Introduced in R2018b

drawfreehand

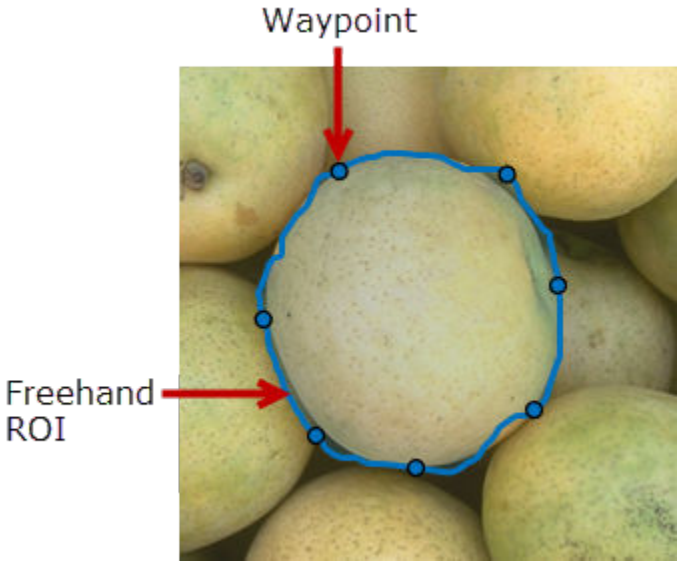
Create customizable freehand ROI

Syntax

```
h = drawfreehand
h = drawfreehand(ax)
h = drawfreehand( ___ ,Name,Value)
```

Description

The `drawfreehand` function creates a `Freehand` object that specifies the shape and position of a freehand region of interest (ROI). You can create the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments. You can also specify the initial appearance and behavior of the ROI.



After you create the ROI, you can use object properties, object functions, and event notifications to customize the shape, position, appearance, and behavior of the ROI. For more information about using these capabilities, see “Tips” on page 1-869.

`h = drawfreehand` creates a `Freehand` object and enables interactive drawing of a circular region-of-interest (ROI) on the current axes.

To draw the ROI, position the pointer on the image. The cursor changes to a fleur shape. Click and drag to draw the line. To finish the ROI, release the mouse button. For more information about using the ROI, including keyboard shortcuts and context menu options, see “Tips” on page 1-869.

`h = drawfreehand(ax)` creates the ROI in the axes specified by `ax`.

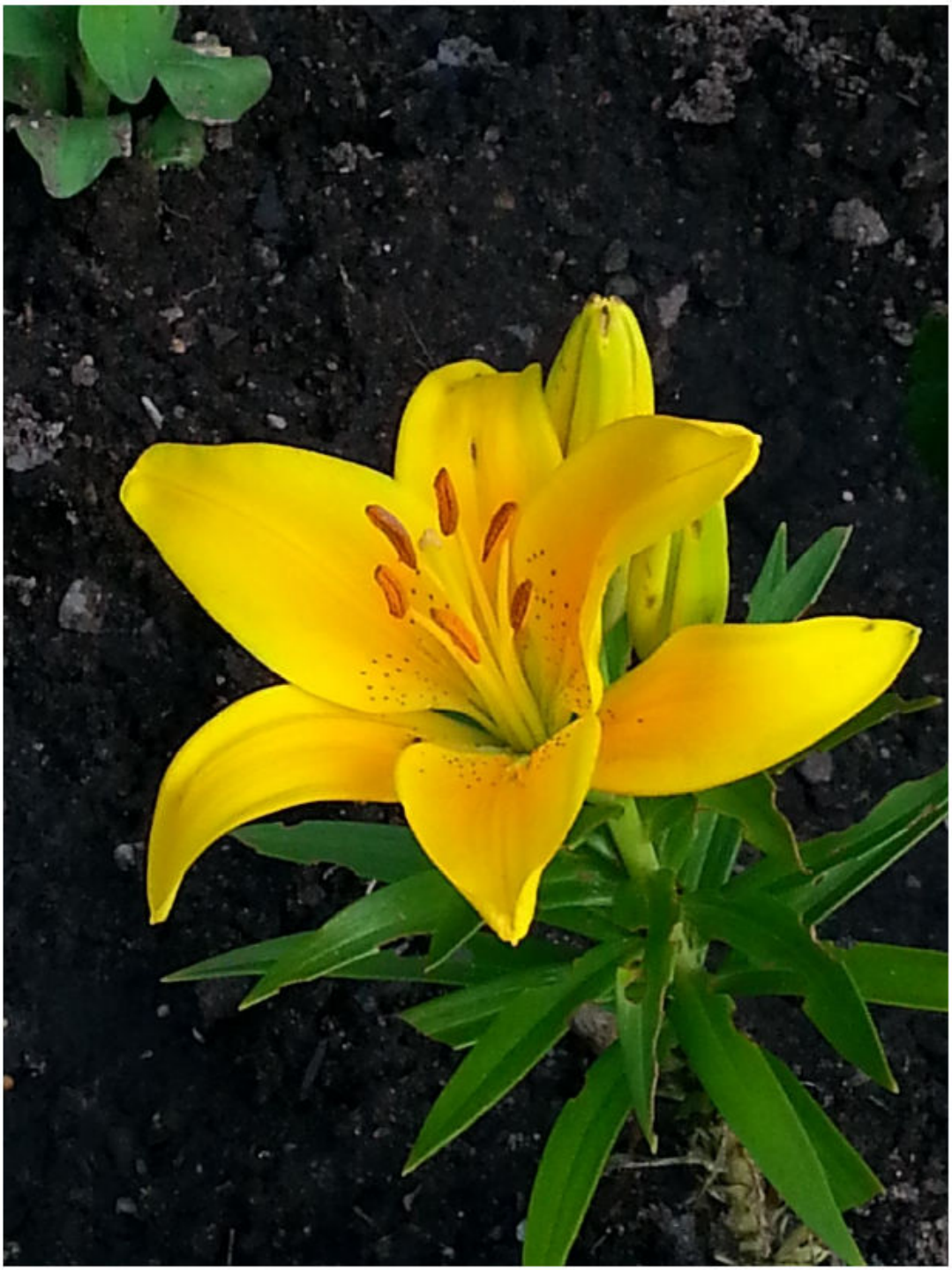
`h = drawfreehand( ___ ,Name,Value)` modifies the appearance and behavior of the ROI using one or more name-value arguments.

Examples

Create Freehand ROI That Is Not Selectable

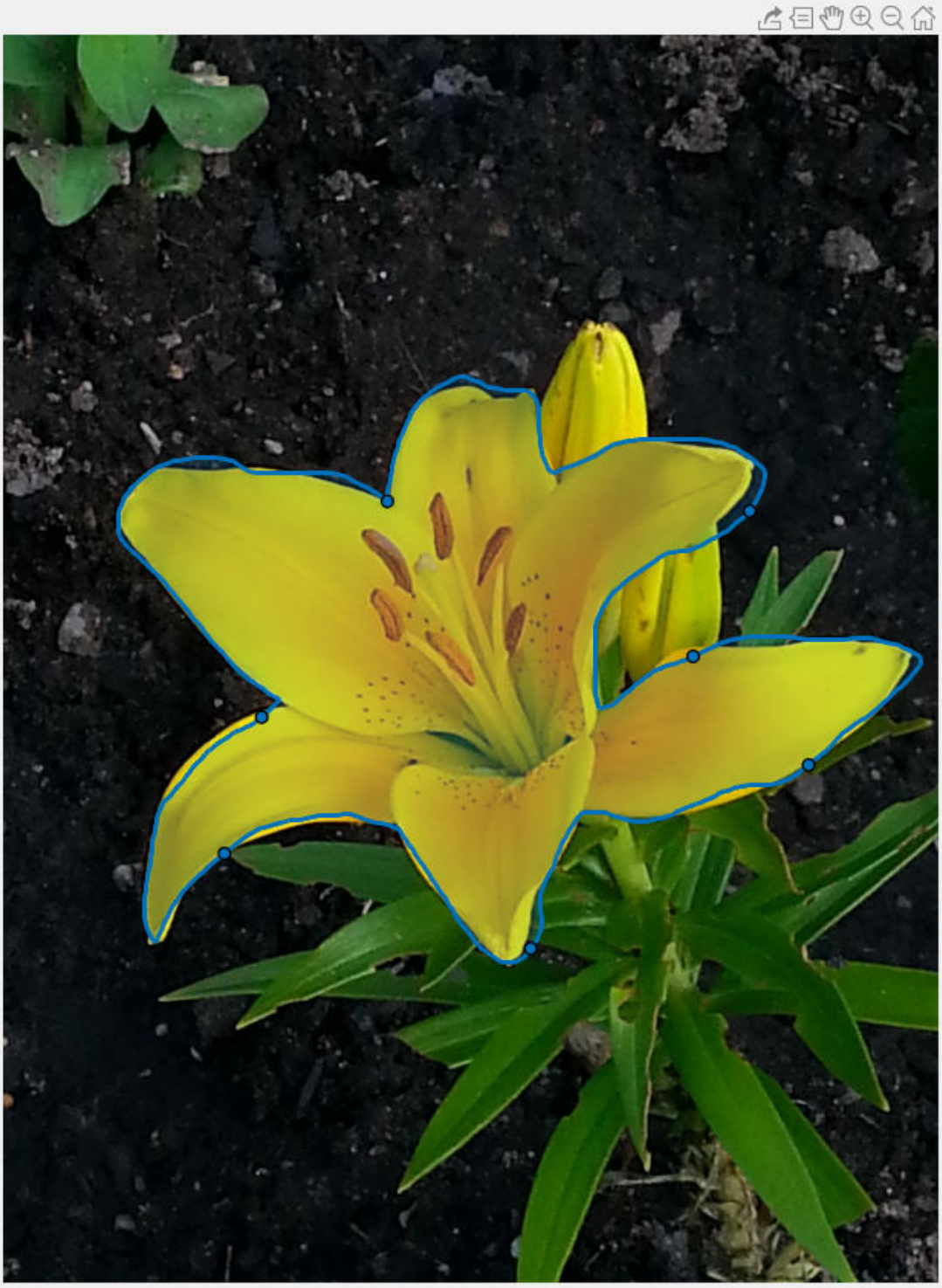
Read an image into the workspace and display it.

```
imshow(imread('yellowlily.jpg'))
```



Draw a freehand ROI.

```
h = drawfreehand;
```



Fill in the face of the freehand ROI and disable the ability to select the ROI. The ROI does not move when you click and drag the mouse.

```
h.FaceAlpha = 1;
h.FaceSelectable = false;
```

Set Up Listener for Freehand ROI Events

Read an image into the workspace.

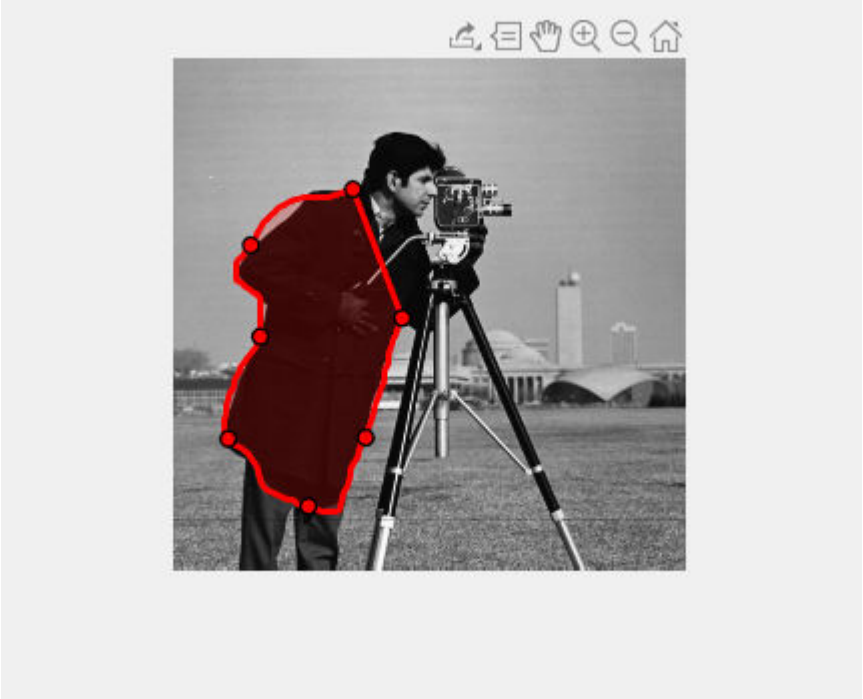
```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a freehand ROI on the image.

```
roi = drawfreehand('Color','r');
```



Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays the previous position and the current position of the ROI.

```
function allevents(src,evt)
    evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving current position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved current position: ' mat2str(evt.CurrentPosition)]);
```

```
    end
end
```

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `h = drawfreehand(Color="y")` creates a yellow colored `Freehand` object.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `h = drawfreehand("Color","y")` creates a yellow colored `Freehand` object.

Closed — Close freehand ROI

`true` or `1` (default) | `false` or `0`

Close the freehand ROI, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `true`, the `drawfreehand` function closes the ROI by connecting the last waypoint drawn to the first waypoint drawn.

Color — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	

Color Name	Short Name	RGB Triplet	Appearance
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0.4470 0.7410]

ContextMenu — Context menu
ContextMenu object

Context menu that displays when you right-click the ROI, specified as a ContextMenu object. You can create a custom context menu by using the uicontextmenu function and then configuring context menu properties.

Deletable — Context menu provides option to delete the ROI
true or 1 (default) | false or 0

Context menu provides an option to delete the ROI, specified as a numeric or logical 1 (true) or 0 (false). When the value is true, you can delete the ROI interactively using the context menu. When the value is false, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the delete function.

DrawingArea — Area of axes in which you can interactively place ROI
'auto' (default) | 'unlimited' | [x,y,w,h]

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
'auto'	The drawing area is the current axes limits (default).
'unlimited'	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.

Value	Description
`[x,y,w,h]`	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

`FaceAlpha` — Transparency of ROI face
`0.2` (default) | number in the range [0, 1]

Transparency of the ROI face, specified as a number in the range [0, 1]. When the value is `1`, the ROI face is completely opaque. When the value is `0`, the ROI face is completely transparent.

`FaceSelectable` — ROI face can capture clicks
`true` or `1` (default) | `false` or `0`

ROI face can capture clicks, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `true`, the ROI face captures mouse clicks. When `false`, the ROI face does not capture mouse clicks.

`HandleVisibility` — Visibility of ROI handle in `Children` property of parent
`'on'` (default) | `'off'` | `'callback'`

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI
`'all'` (default) | `'none'` | `'reshape'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable (default).
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area but not reshaped.
`'reshape'`	The ROI can be reshaped but not translated.

`Label` — ROI label
`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background
`1` (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color

`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'LabelTextColor','r'`

Example: `'LabelTextColor','green'`

Example: `'LabelTextColor',[0 0.4470 0.7410]`

`LabelVisible` — Label visibility

`'on'` (default) | `'off'` | `'hover'`

Label visibility, specified as one of these values.

Value	Description
`'on'`	Label is visible when the ROI is visible.
`'hover'`	Label is visible only when the mouse is hovering over the ROI.
`'off'`	Label is not visible.

`LineWidth` — Width of ROI border
positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

`MarkerSize` — Marker size
positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

`Multiclick` — Control freehand drawing style during interactive placement
`false` or `0` (default) | `true` or `1`

Control the freehand drawing style during interactive placement, specified as a numeric or logical `0` (`false`) or `1` (`true`). When the value is `false`, a single click and drag gesture completes the freehand ROI. When the value is `true`, multiple click and drag gestures can be combined with straight edges to make a more complex freehand ROI shape.

`Parent` — ROI parent
`Axes` object | `UIAxes` object

ROI parent, specified as an `Axes` or `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

`Position` — Position of ROI
n-by-2 numeric matrix

Position of the ROI, specified as an *n*-by-2 numeric matrix where *n* is the number of vertices or points defining the ROI. Each row represents the [*x* *y*] coordinates of a vertex or point.

`Selected` — Selection state of ROI
`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`).

`SelectedColor` — Color of ROI when selected
`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'SelectedColor','r'`

Example: `'SelectedColor','green'`

Example: `'SelectedColor',[0 0.4470 0.7410]`

`Smoothing` — Smoothing applied to edge of ROI

1 (default) | nonnegative number

Smoothing applied to the edge of the ROI after interactive placement, specified as a nonnegative number. The `drawfreehand` function filters the *x* and *y* coordinates of the ROI using a Gaussian smoothing kernel with a default standard deviation of 1. The size of the Gaussian filter is `2*ceil(2*Smoothing)+1`. You can see the smoothing effect only after completing the drawing.

`StripeColor` — Color of ROI stripe

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

Tag — Tag to associate with ROI
`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI
any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `drawfreehand` object does not use this data.

`Visible` — ROI visibility
`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

`Waypoints` — Position point is waypoint
n-by-1 logical vector

Position point is waypoint, specified as an *n*-by-1 logical vector where *n* is the number of points defining the ROI. The length of `Waypoints` must match the number of rows of the `Position` name-value argument. Elements in `Waypoints` with the value `true` identify points in the `Position` matrix that are waypoints.

Waypoints appear as circular shapes on the ROI edge. You can use waypoints to reshape the ROI by clicking and dragging the waypoint with the mouse. Moving waypoints modifies the freehand-drawn region between the waypoint that you clicked and the adjacent waypoints.

Output Arguments

`h` — Freehand ROI
`Freehand` object

Freehand ROI, returned as an `Freehand` object.

Tips

- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Finish drawing (close) the ROI.	Double-click, which adds a point at the pointer position and draws a line connecting this point to the first point drawn, closing the ROI. Right-click, which draws a line connecting the last point to the first point drawn. Position the pointer over the first point and click. Press **Enter**, which draws a line connecting the last point to the first point drawn.
Resize (reshape) the ROI.	Position pointer over a waypoint and then click and drag. No assistance (snapping to edges) is available in this mode.

Behavior	Keyboard shortcut
Add a waypoint.	Position the pointer on an edge of the ROI, right-click, and select **Add Waypoint**. You can also position the pointer on an edge of the ROI and double-click.
Remove a waypoint.	Position the pointer on a waypoint, right-click, and select **Remove Waypoint**.
Move the ROI.	Position the pointer over the ROI. The pointer changes to the fleur shape. Click and drag to move the ROI.
Delete an ROI.	Position the pointer on the ROI (not on a vertex), right-click, and select **Delete Freehand** from the context menu. You can also delete the ROI programmatically using the `delete` function.

- The `drawfreehand` function creates a `Freehand` object. After you create the object, you can modify the shape, position, appearance, and behavior of the ROI by using these object capabilities.

Capability	Support
Object properties	ROI objects have properties that specify their shape, position, appearance, and behavior. After you create the ROI object, change properties using dot notation. For example, to change the color of the `roi` to yellow, set its `Color` property: `roi.Color = 'yellow'`
Object functions	ROI objects have object functions that operate on the ROIs. For example, if you want to pause the MATLAB command line after creating an ROI, use the `wait` function.
Event notifications	ROI objects can notify your code when certain events occur, such as when the ROI is clicked or when the ROI is being moved. To receive event notifications, set up listeners. When the ROI notifies your application through the listener, it returns data specific to the event. For example, with the `ROIMoved` event, the ROI object returns its previous position and its current position. You can specify a callback function that executes when an event occurs.

For an example of using event listeners with the `Freehand` object, see “Set Up Listener for Freehand ROI Events” on page 1-861.

Compatibility Considerations

`UIContextMenu` name-value argument is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` name-value argument to assign a context menu to an ROI object is not recommended. Use the `ContextMenu` name-value argument instead. The values are the same.

There are no plans to remove support for the `UIContextMenu` name-value argument at this time.

See Also

`Freehand` | `drawassisted` | `drawpolygon` | `drawpolyline`

Topics

“Create ROI Shapes”
“Using ROIs in Apps Created with App Designer”
“Use Wait Function After Drawing ROI”

Introduced in R2018b

drawline

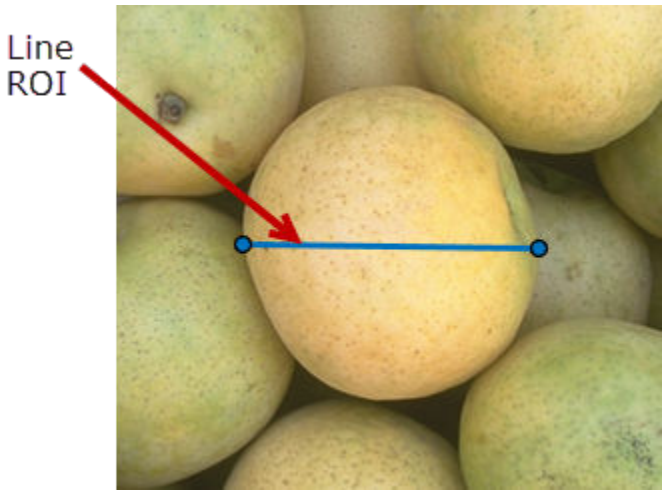
Create customizable linear ROI

Syntax

```
roi = drawline
roi = drawline(ax)
roi = drawline( ___ ,Name,Value)
```

Description

The `drawline` function creates a `Line` object that specifies the length and position of a line region of interest (ROI). You can create the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments. You can also specify the initial appearance and behavior of the ROI.



After you create the ROI, you can use object properties, object functions, and event notifications to customize the shape, position, appearance, and behavior of the ROI. For more information about using these capabilities, see “Tips” on page 1-885.

`roi = drawline` creates a `Line` object and enables interactive drawing of a linear region-of-interest (ROI) on the current axes.

To draw the ROI, position the pointer on the image. The cursor changes to a fleur shape. Click and drag to draw the Line ROI. To finish the ROI, release the mouse button. For more information about using the ROI, including keyboard shortcuts and context menu options, see “Tips” on page 1-885.

`roi = drawline(ax)` creates the ROI in the axes specified by `ax`.

`roi = drawline( ___ ,Name,Value)` modifies the appearance and behavior of the ROI using one or more name-value pairs. Unspecified name-value pairs are set to the default value.

Examples

Create Linear ROIs That Change Color When Selected

Read an image into the workspace and display it.

```
imshow(imread('car1.jpg'))
```

Draw two linear ROIs on the image. Use the `'SelectedColor'` name-value pair to specify the color of the ROI when selected.

```
h1 = drawline('SelectedColor','yellow');
h2 = drawline('SelectedColor','magenta');
```

To turn the first line yellow, select the ROI programatically. You can also select an ROI by clicking it with the mouse. Click the second ROI to turn it magenta.

```
h1.Selected = true;
```

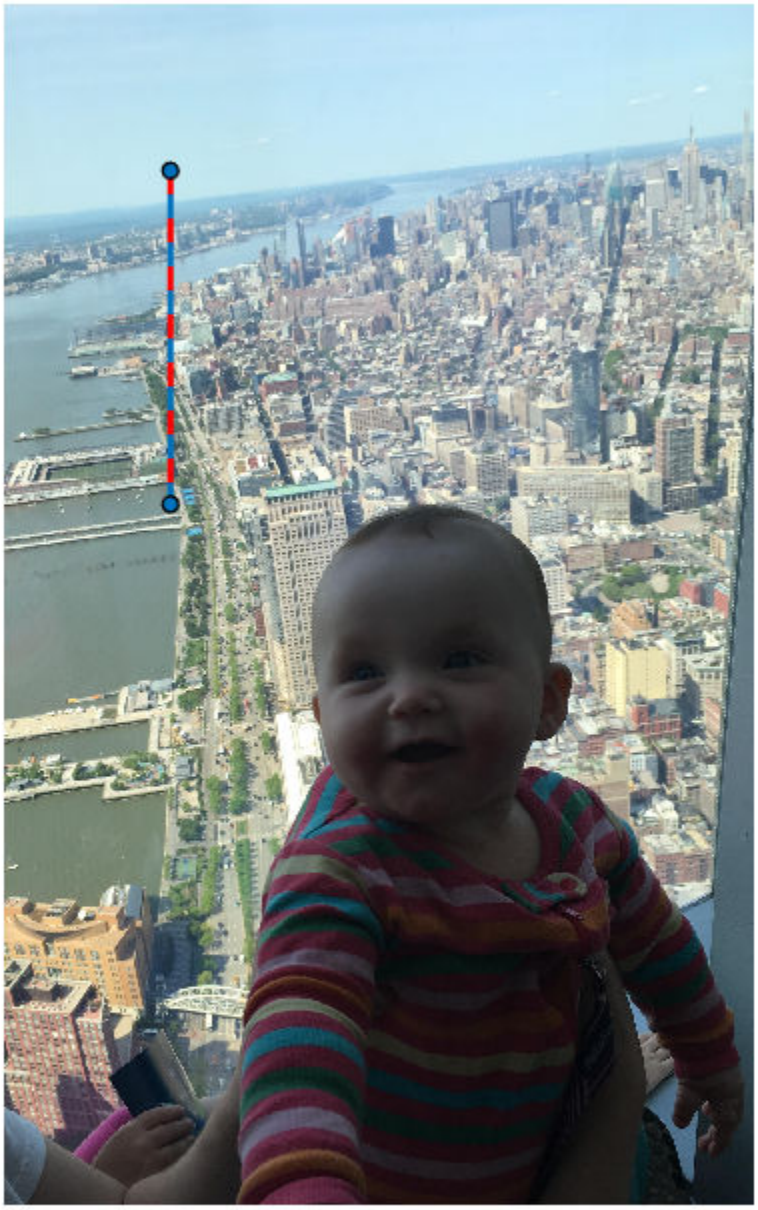
Create Linear ROI Programmatically

Read an image into the workspace and display it.

```
I = imread('baby.jpg');
figure
imshow(I)
```

Draw a linear ROI over the image. Use the `'Position'` name-value pair to specify the location and length of the linear ROI. Set the line to be striped red by specifying the `'StripeColor'` name-value pair.

```
h = drawline('Position',[500 500;500 1500],'StripeColor','r');
```



Set Up Listener for Line ROI Events

Read an image into the workspace.

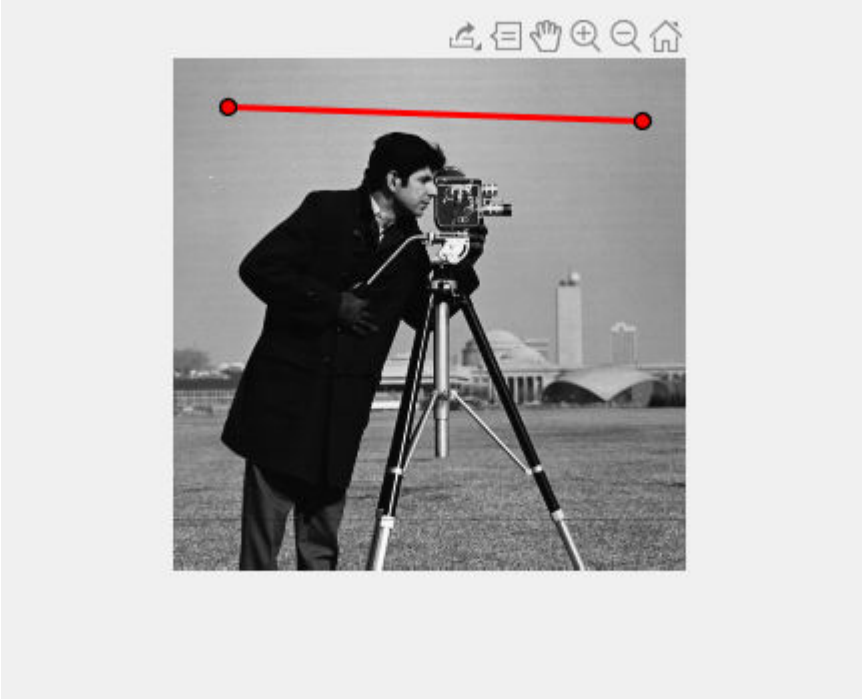
```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a Line ROI on the image.

```
roi = drawline('Color','r');
```



Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays the previous position and the current position of the ROI.

```
function allevents(src,evt)
    evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving current position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved current position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Color','y'` creates a yellow colored `Line` object

`Color` — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'Color','r'`

Example: `'Color','green'`

Example: `'Color',[0 0.4470 0.7410]`

`ContextMenu` — Context menu
`ContextMenu` object

Context menu that displays when you right-click the ROI, specified as a `ContextMenu` object. You can create a custom context menu by using the `uicontextmenu` function and then configuring context menu properties.

`Deletable` — Context menu provides option to delete the ROI
`true` or `1` (default) | `false` or `0`

Context menu provides an option to delete the ROI, specified as a numeric or logical `1` (`true`) or `0` (`false`). When the value is `true`, you can delete the ROI interactively using the context menu. When the value is `false`, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the `delete` function.

`DrawingArea` — Area of axes in which you can interactively place ROI
`'auto'` (default) | `'unlimited'` | `[x,y,w,h]`

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
`'auto'`	The drawing area is the current axes limits (default).
`'unlimited'`	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
`[x,y,w,h]`	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

`HandleVisibility` — Visibility of ROI handle in `Children` property of parent
`'on'` (default) | `'off'` | `'callback'`

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI
`'all'` (default) | `'none'` | `'reshape'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable (default).
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area but not reshaped.
`'reshape'`	The ROI can be reshaped but not translated.

`Label` — ROI label

`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background

1 (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color

`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	

RGB Triplet	Appearance
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'LabelTextColor','r'

Example: 'LabelTextColor','green'

Example: 'LabelTextColor',[0 0.4470 0.7410]

LabelVisible — Label visibility
'on' (default) | 'off' | 'hover'

Label visibility, specified as one of these values.

Value	Description
'on'	Label is visible when the ROI is visible.
'hover'	Label is visible only when the mouse is hovering over the ROI.
'off'	Label is not visible.

LineWidth — Width of ROI border
positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

MarkerSize — Marker size
positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

Parent — ROI parent
Axes object | UIAxes object

ROI parent, specified as an Axes or UIAxes object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Position — Position of ROI
2-by-2 numeric matrix

Position of the ROI, specified as a 2-by-2 numeric matrix. Each row represents the [*x* *y*] coordinates of an endpoint of the line.

Selected — Selection state of ROI
false or 0 (default) | true or 1

Selection state of the ROI, specified as a numeric or logical 0 (false) or 1 (true).

`SelectedColor` — Color of ROI when selected

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'SelectedColor','r'`

Example: `'SelectedColor','green'`

Example: `'SelectedColor',[0 0.4470 0.7410]`

`StripeColor` — Color of ROI stripe

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

Tag — Tag to associate with ROI
`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI
any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `drawline` object does not use this data.

`Visible` — ROI visibility
`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Output Arguments

`roi` — Linear ROI

`Line` object

Linear ROI, returned as a `Line` object.

Tips

- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Make drawn line snap to 15 degree angles.	Hold the **Shift** key while drawing.
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Resize the ROI.	Position pointer over either endpoint and then click and drag to resize the ROI. Hold the **Shift** key while resizing to snap the line drawn at 15 degree angles.
Move the ROI.	Position the pointer over the ROI. The pointer changes to the fleur shape. Then click and drag the ROI.
Delete the ROI.	Position the pointer anywhere on the ROI and right-click. Select **Delete Line** from the context menu. You can also delete the ROI using the `delete` function.

- The `drawline` function creates a `Line` object. After you create the object, you can modify the length, position, appearance, and behavior of the ROI by using these object capabilities.

Capability	Support
Object properties	ROI objects have properties that specify their shape, position, appearance, and behavior. After you create the ROI object, change properties using dot notation. For example, to change the color of the `roi` to yellow, set its `Color` property: `roi.Color = 'yellow'`

Capability	Support
Object functions	ROI objects have object functions that operate on the ROIs. For example, if you want to pause the MATLAB command line after creating an ROI, use the `wait` function.
Event notifications	ROI objects can notify your code when certain events occur, such as when the ROI is clicked or when the ROI is being moved. To receive event notifications, set up listeners. When the ROI notifies your application through the listener, it returns data specific to the event. For example, with the `ROIMoved` event, the ROI object returns its previous position and its current position. You can specify a callback function that executes when an event occurs.

For an example of using event listeners with the `Line` object, see “Set Up Listener for Line ROI Events” on page 1-877.

Compatibility Considerations

`UIContextMenu` name-value argument is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` name-value argument to assign a context menu to an ROI object is not recommended. Use the `ContextMenu` name-value argument instead. The values are the same.

There are no plans to remove support for the `UIContextMenu` name-value argument at this time.

See Also

`Line` | `drawcrosshair` | `drawpoint` | `drawpolyline`

Topics

“Create ROI Shapes”
“Using ROIs in Apps Created with App Designer”
“Use Wait Function After Drawing ROI”

Introduced in R2018b

drawpoint

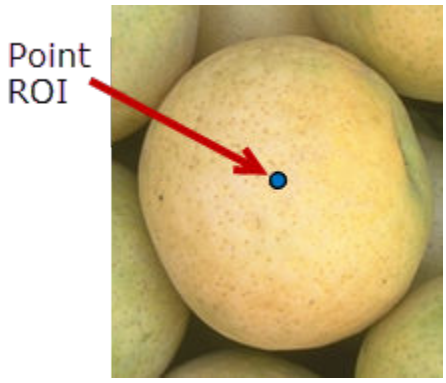
Create customizable point ROI

Syntax

```
roi = drawpoint
roi = drawpoint(ax)
roi = drawpoint( ___ ,Name,Value)
```

Description

The `drawpoint` function creates a `Point` object that specifies the position of a point region of interest (ROI). You can create the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments. You can also specify the initial appearance and behavior of the ROI.



After you create the ROI, you can use object properties, object functions, and event notifications to customize the shape, position, appearance, and behavior of the ROI. For more information about using these capabilities, see “Tips” on page 1-900.

`roi = drawpoint` creates a `Point` object and enables interactive drawing of a point region-of-interest (ROI) on the current axes.

To draw the ROI, position the pointer on the image. The cursor changes to a fleur shape. Click to draw the ROI. To finish the ROI, release the mouse button. For more information about using the ROI, including keyboard shortcuts and context menu options, see “Tips” on page 1-900.

`roi = drawpoint(ax)` creates the ROI in the axes specified by `ax`.

`roi = drawpoint( ___ ,Name,Value)` modifies the appearance and behavior of the ROI using one or more name-value pairs. Unspecified name-value pairs are set to the default value.

Examples

Create Point ROI Interactively

Read an image into the workspace and display it.

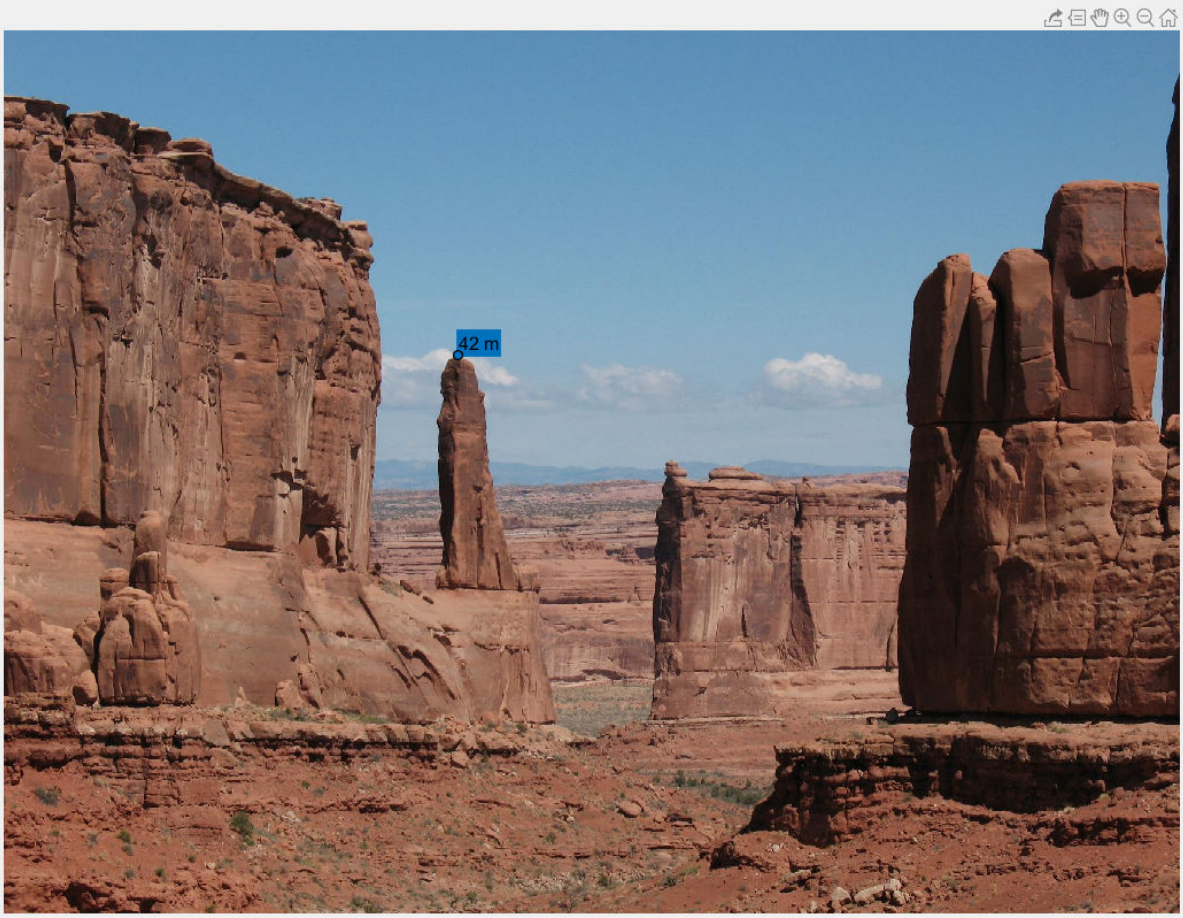
```
imshow(imread('parkavenue.jpg'))
```

Draw a point ROI on the image.

```
h = drawpoint;
```

Add a label to the ROI.

```
h.Label = '42 m';
```



Create Point ROI Programmatically

Read image into the workspace and display it.

```
I = imread('baby.jpg');
figure
imshow(I)
```

Draw a point ROI on the image, using named parameters to specify the location.

```
h = drawpoint('Position',[500 500]);
```

Set Up Listener for Point ROI Events

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a point ROI on the image.

```
roi = drawpoint('Color','r');
```

Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays the previous position and the current position of the ROI.

```
function allevents(src,evt)
    evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving current position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved current position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Color','y'` creates a yellow colored `Point` object

`Color` — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'Color','r'`

Example: `'Color','green'`

Example: `'Color',[0 0.4470 0.7410]`

`ContextMenu` — Context menu
`ContextMenu` object

Context menu that displays when you right-click the ROI, specified as a `ContextMenu` object. You can create a custom context menu by using the `uicontextmenu` function and then configuring context menu properties.

`Deletable` — Context menu provides option to delete the ROI
`true` or `1` (default) | `false` or `0`

Context menu provides an option to delete the ROI, specified as a numeric or logical `1` (`true`) or `0` (`false`). When the value is `true`, you can delete the ROI interactively using the context menu. When the value is `false`, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the `delete` function.

`DrawingArea` — Area of axes in which you can interactively place ROI
`'auto'` (default) | `'unlimited'` | `[x,y,w,h]`

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
`'auto'`	The drawing area is the current axes limits (default).
`'unlimited'`	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
`[x,y,w,h]`	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

`HandleVisibility` — Visibility of ROI handle in `Children` property of parent
`'on'` (default) | `'off'` | `'callback'`

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI
`'all'` (default) | `'none'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable.
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area.

`Label` — ROI label

`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background

1 (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color

`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	

RGB Triplet	Appearance
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'LabelTextColor','r'

Example: 'LabelTextColor','green'

Example: 'LabelTextColor',[0 0.4470 0.7410]

LabelVisible — Label visibility
'on' (default) | 'off' | 'hover'

Label visibility, specified as one of these values.

Value	Description
'on'	Label is visible when the ROI is visible.
'hover'	Label is visible only when the mouse is hovering over the ROI.
'off'	Label is not visible.

LineWidth — Width of ROI border
positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

MarkerSize — Marker size
positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

Parent — ROI parent
Axes object | UIAxes object

ROI parent, specified as an Axes or UIAxes object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Position — Position of ROI
1-by-2 numeric vector

Position of the ROI, specified as a 1-by-2 numeric vector that represents the [*x* *y*] coordinates of the point.

Selected — Selection state of ROI
false or 0 (default) | true or 1

Selection state of the ROI, specified as a numeric or logical 0 (false) or 1 (true).

SelectedColor — Color of ROI when selected
'none' (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'SelectedColor','r'`

Example: `'SelectedColor','green'`

Example: `'SelectedColor',[0 0.4470 0.7410]`

`StripeColor` — Color of ROI stripe

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

Tag — Tag to associate with ROI

`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI

any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `drawpoint` object does not use this data.

`Visible` — ROI visibility

`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Output Arguments

`roi` — Point ROI
`Point` object

Point ROI, returned as a `Point` object.

Tips

- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Move the ROI.	Position the pointer over the ROI. The pointer changes to a circle. Click and drag to move the ROI.
Delete the ROI.	Position the pointer anywhere on the ROI and right-click. Select **Delete Point** from the context menu. You can also delete the ROI using the `delete` function.

- The `drawpoint` function creates a `Point` object. After you create the object, you can modify the position, appearance, and behavior of the ROI by using these object capabilities.

Capability	Support
Object properties	ROI objects have properties that specify their shape, position, appearance, and behavior. After you create the ROI object, change properties using dot notation. For example, to change the color of the `roi` to yellow, set its `Color` property: `roi.Color = 'yellow'`
Object functions	ROI objects have object functions that operate on the ROIs. For example, if you want to pause the MATLAB command line after creating an ROI, use the `wait` function.

Capability	Support
Event notifications	ROI objects can notify your code when certain events occur, such as when the ROI is clicked or when the ROI is being moved. To receive event notifications, set up listeners. When the ROI notifies your application through the listener, it returns data specific to the event. For example, with the `ROIMoved` event, the ROI object returns its previous position and its current position. You can specify a callback function that executes when an event occurs.

For an example of using event listeners with the `Point` object, see “Set Up Listener for Point ROI Events” on page 1-892.

Compatibility Considerations

`UIContextMenu` name-value argument is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` name-value argument to assign a context menu to an ROI object is not recommended. Use the `ContextMenu` name-value argument instead. The values are the same.

There are no plans to remove support for the `UIContextMenu` name-value argument at this time.

See Also

`Point` | `drawcircle` | `drawcrosshair`

Topics

“Create ROI Shapes”
“Using ROIs in Apps Created with App Designer”
“Use Wait Function After Drawing ROI”

Introduced in R2018b

drawpolygon

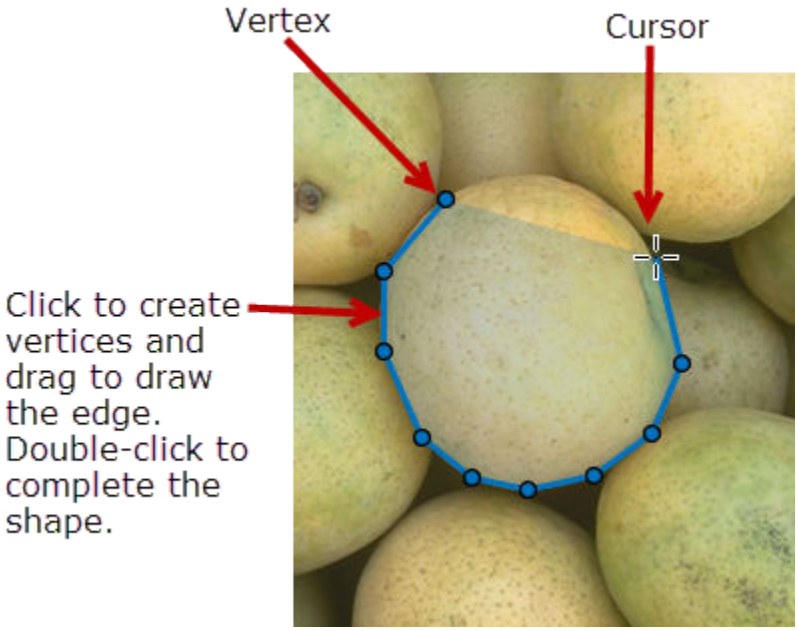
Create customizable polygonal ROI

Syntax

```
roi = drawpolygon
roi = drawpolygon(ax)
roi = drawpolygon( ___ ,Name,Value)
```

Description

The `drawpolygon` function creates a `Polygon` object that specifies the shape and position of a polygonal region of interest (ROI). You can create the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments. You can also specify the initial appearance and behavior of the ROI.



After you create the ROI, you can use object properties, object functions, and event notifications to customize the shape, position, appearance, and behavior of the ROI. For more information about using these capabilities, see “Tips” on page 1-915.

`roi = drawpolygon` creates a `Polygon` object and enables interactive drawing of a polygonal ROI on the current axes.

To draw the ROI, position the pointer on the image. The cursor changes to a fleur shape. Click to draw vertices of the polygon and drag to draw the lines between the vertices. To finish the ROI, double-click the mouse button. For more information about using the ROI, including keyboard shortcuts and context menu options, see “Tips” on page 1-915.

`roi = drawpolygon(ax)` creates the ROI on the axes specified by `ax`.

`roi = drawpolygon( ___ ,Name,Value)` modifies the appearance and behavior of the ROI using one or more name-value arguments.

Examples

Create Polygonal ROI Interactively

Read an image into the workspace and display it.

```
imshow(imread('strawberries.jpg'))
```

Draw a polygonal ROI on the image. Use the `'FaceAlpha'` name-value pair to make the face of the ROI transparent.

```
h = drawpolygon('FaceAlpha',0);
```

Change the color of the polygon outline by setting the `'Color'` property of the ROI.

```
h.Color = 'yellow';
```

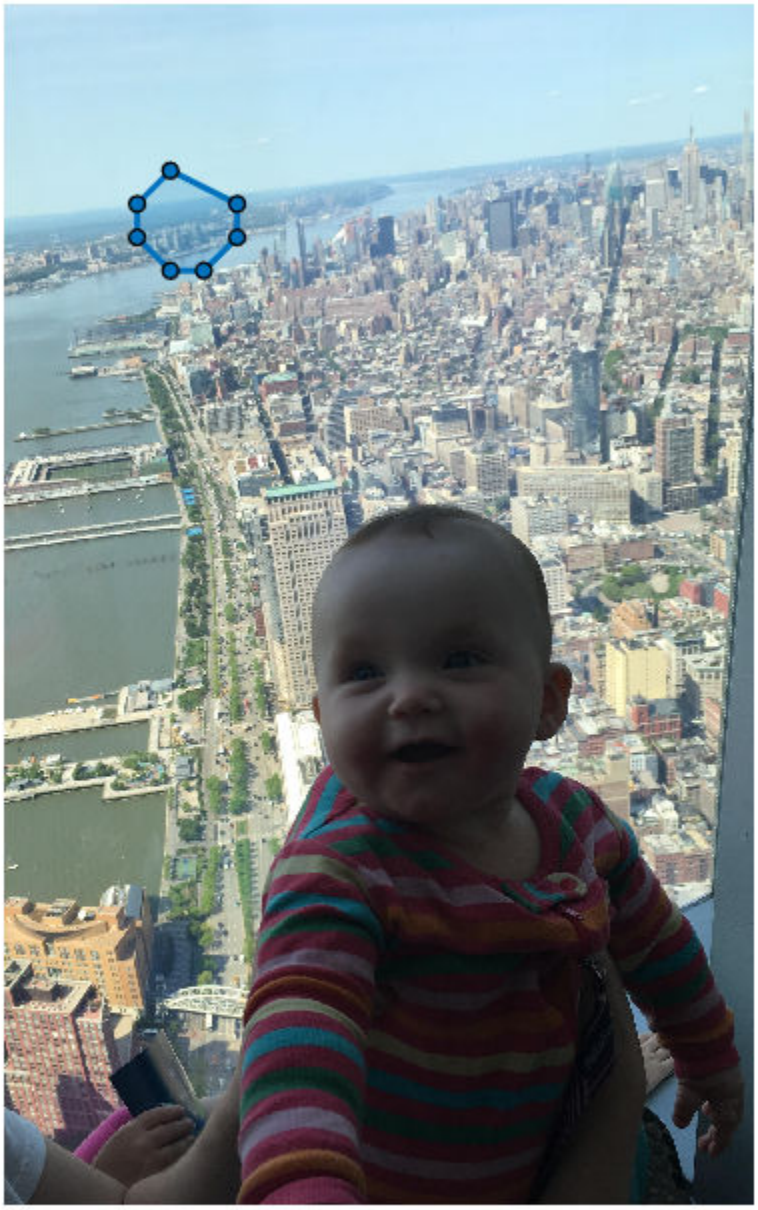
Create Polygonal ROI Programmatically

Read image into the workspace and display it.

```
I = imread('baby.jpg');
figure
imshow(I)
```

Draw a polygonal ROI over the image, using the `Position` parameter to specify the location of vertices.

```
my_vertices = [500 500;400 600;400 700;500 800;600 800;700 700; 700 600];
h = drawpolygon('Position',my_vertices);
```



Set Up Listener for Polygon ROI Events

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a polygonal ROI on the image.

```
roi = drawpolygon('Color','r');
```

Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays the previous position and the current position of the ROI.

```
function allevents(src,evt)
    evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving current position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved current position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Color','y'` creates a yellow colored `Polygon` object

`Color` — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'Color','r'`

Example: `'Color','green'`

Example: `'Color',[0 0.4470 0.7410]`

`ContextMenu` — Context menu
`ContextMenu` object

Context menu that displays when you right-click the ROI, specified as a `ContextMenu` object. You can create a custom context menu by using the `uicontextmenu` function and then configuring context menu properties.

`Deletable` — Context menu provides option to delete the ROI
`true` or `1` (default) | `false` or `0`

Context menu provides an option to delete the ROI, specified as a numeric or logical `1` (`true`) or `0` (`false`). When the value is `true`, you can delete the ROI interactively using the context menu. When the value is `false`, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the `delete` function.

`DrawingArea` — Area of axes in which you can interactively place ROI
`'auto'` (default) | `'unlimited'` | `[x,y,w,h]`

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
`'auto'`	The drawing area is the current axes limits (default).
`'unlimited'`	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
`[x,y,w,h]`	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

`FaceAlpha` — Transparency of ROI face
`0.2` (default) | number in the range [0, 1]

Transparency of the ROI face, specified as a number in the range [0, 1]. When the value is `1`, the ROI face is completely opaque. When the value is `0`, the ROI face is completely transparent.

`FaceSelectable` — ROI face can capture clicks
`true` or `1` (default) | `false` or `0`

ROI face can capture clicks, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `true`, the ROI face captures mouse clicks. When `false`, the ROI face does not capture mouse clicks.

`HandleVisibility` — Visibility of ROI handle in `Children` property of parent
`'on'` (default) | `'off'` | `'callback'`

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI
`'all'` (default) | `'none'` | `'reshape'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable (default).
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area but not reshaped.
`'reshape'`	The ROI can be reshaped but not translated.

`Label` — ROI label
`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background
`1` (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color
`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	

Color Name	Short Name	RGB Triplet	Appearance
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'LabelTextColor','r'`

Example: `'LabelTextColor','green'`

Example: `'LabelTextColor',[0 0.4470 0.7410]`

`LabelVisible` — Label visibility
`'on'` (default) | `'off'` | `'hover'`

Label visibility, specified as one of these values.

Value	Description
'on'	Label is visible when the ROI is visible.
'hover'	Label is visible only when the mouse is hovering over the ROI.
'off'	Label is not visible.

`LineWidth` — Width of ROI border
positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

`MarkerSize` — Marker size
positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

`Parent` — ROI parent
`Axes` object | `UIAxes` object

ROI parent, specified as an `Axes` or `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

`Position` — Position of ROI
n-by-2 numeric matrix

Position of the ROI, specified as an *n*-by-2 numeric matrix where *n* is the number of vertices or points defining the ROI. Each row represents the [*x* *y*] coordinates of a vertex or point.

`Selected` — Selection state of ROI
`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`).

`SelectedColor` — Color of ROI when selected
`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	

RGB Triplet	Appearance
[0.6350 0.0780 0.1840]	

Example: 'SelectedColor','r'

Example: 'SelectedColor','green'

Example: 'SelectedColor',[0 0.4470 0.7410]

StripeColor — Color of ROI stripe
'none' (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or 'none'. If you specify 'none', then the ROI edge is a solid color specified by Color. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by Color and StripeColor.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'StripeColor','r'

Example: 'StripeColor','green'

Example: 'StripeColor',[0 0.4470 0.7410]

`Tag` — Tag to associate with ROI
`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI
any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `drawpolygon` object does not use this data.

`Visible` — ROI visibility
`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Output Arguments

`roi` — Polygonal ROI
`Polygon` object

Polygonal ROI, returned as an `Polygon` object.

Tips

- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Make drawn line snap at 15 degree angles.	Hold the **Shift** key while drawing.
Finish drawing (close) the ROI.	Double-click, which adds a new vertex at the pointer position and draws a line to the first vertex to close the polygon. Press **Enter**, which adds a new vertex at the pointer position and draws a line to the first vertex to close the polygon. Right-click, which does not add a new vertex but closes the polygon from the previous vertex. Position pointer over the first vertex and click.

Behavior	Keyboard shortcut
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Add a new vertex to the ROI.	Position the pointer over the edge of the ROI and double-click. Position the pointer over the edge of the ROI, right-click, and select **Add Vertex** from the context menu.
Remove the most recently added vertex but keep drawing.	Press **Backspace**. The function redraws the line from the previous vertex to the current position of the pointer. You can only back up to the first vertex you drew.
Resize (reshape) the ROI	Position pointer over a vertex and then click and drag. Add a new vertex to the ROI and then click and drag. Remove a vertex. The ROI redraws the line connecting the two neighboring vertices.
Move the ROI.	Position the pointer over the ROI (not on a vertex). The pointer changes to a fleur shape. Click and drag to move the ROI.
Delete the ROI.	Position the pointer anywhere on the ROI and right-click. Select **Delete Polygon** from the context menu. You can also delete the ROI using the `delete` function.

- The `drawpolygon` function creates a `Polygon` object. After you create the object, you can modify the shape, position, appearance, and behavior of the ROI by using these object capabilities.

Capability	Support
Object properties	ROI objects have properties that specify their shape, position, appearance, and behavior. After you create the ROI object, change properties using dot notation. For example, to change the color of the `roi` to yellow, set its `Color` property: `roi.Color = 'yellow'`
Object functions	ROI objects have object functions that operate on the ROIs. For example, if you want to pause the MATLAB command line after creating an ROI, use the `wait` function.

Capability	Support
Event notifications	ROI objects can notify your code when certain events occur, such as when the ROI is clicked or when the ROI is being moved. To receive event notifications, set up listeners. When the ROI notifies your application through the listener, it returns data specific to the event. For example, with the `ROIMoved` event, the ROI object returns its previous position and its current position. You can specify a callback function that executes when an event occurs.

For an example of using event listeners with the `Polygon` object, see “Set Up Listener for Polygon ROI Events” on page 1-907.

Compatibility Considerations

`UIContextMenu` name-value argument is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` name-value argument to assign a context menu to an ROI object is not recommended. Use the `ContextMenu` name-value argument instead. The values are the same.

There are no plans to remove support for the `UIContextMenu` name-value argument at this time.

See Also

`Polygon` | `drawpolyline` | `drawrectangle` | `drawassisted` | `drawfreehand`

Topics

“Create ROI Shapes”
“Using ROIs in Apps Created with App Designer”
“Use Wait Function After Drawing ROI”

Introduced in R2018b

drawpolyline

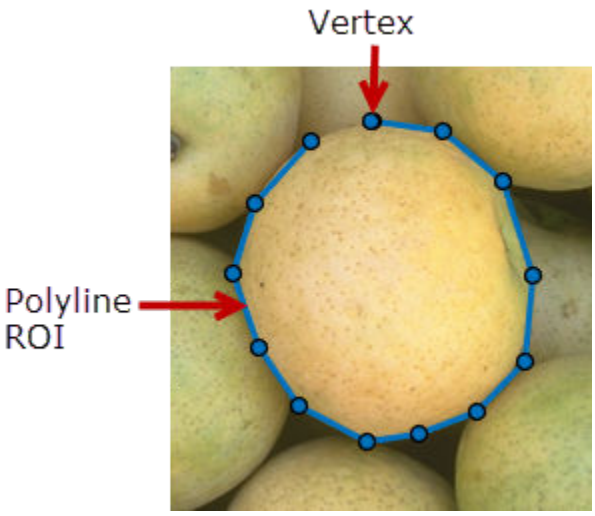
Create customizable polyline ROI

Syntax

```
roi = drawpolyline
roi = drawpolyline(ax)
roi = drawpolyline( ___ ,Name,Value)
```

Description

The `drawpolyline` function creates a `Polyline` object that specifies the shape and position of a polyline region of interest (ROI). You can create the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments. You can also specify the initial appearance and behavior of the ROI.



After you create the ROI, you can use object properties, object functions, and event notifications to customize the shape, position, appearance, and behavior of the ROI. For more information about using these capabilities, see “Tips” on page 1-931.

`roi = drawpolyline` creates a `Polyline` ROI object and enables interactive drawing of the ROI on the current axes.

To draw the ROI, position the pointer on the image. The cursor changes to a fleur shape. Click to draw vertices of the polyline and drag to draw the lines between the vertices. To finish the ROI, double-click the mouse button. For more information about using the ROI, including keyboard shortcuts and context menu options, see “Tips” on page 1-931.

`roi = drawpolyline(ax)` creates the ROI in the axes specified by `ax`.

`roi = drawpolyline( ___ ,Name,Value)` modifies the appearance and behavior of the ROI using one or more name-value pairs. Unspecified name-value pairs are set to the default value.

Examples

Create Polyline ROI Interactively

Read an image into the workspace and display it.

```
imshow(imread('westconcordaerial.png'))
```

Draw the polyline ROI on the image. Use the `'Color'` name-value pair to specify the color of the line.

```
h = drawpolyline('Color','green');
```

Decrease the width of the edge of the ROI by setting the `LineWidth` property.

```
h.LineWidth = 1;
```

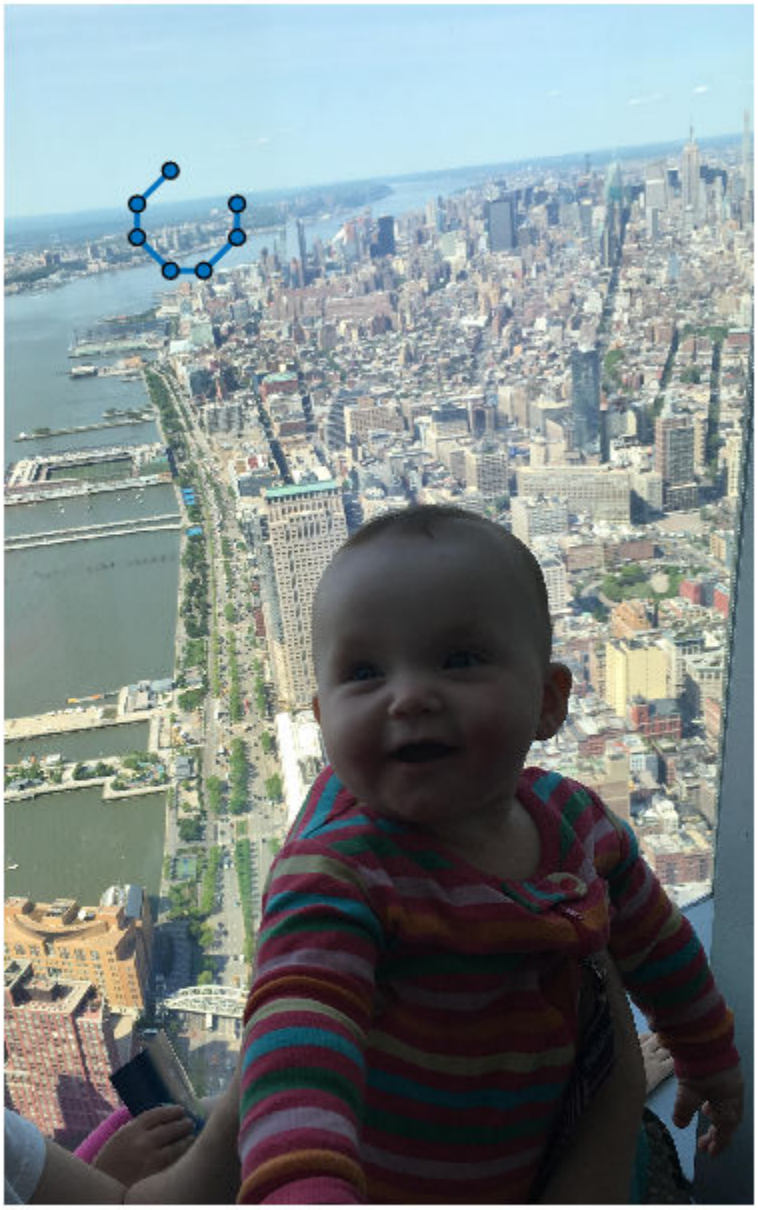
Create Polygonal ROI Programmatically

Read image into the workspace and display it.

```
I = imread('baby.jpg');
figure
imshow(I)
```

Draw a polygonal ROI over the image, using named parameters to specify the location and shape. The example also specifies that the edge of the polygon is a striped.

```
h = drawpolyline('Position',[500 500;400 600;400 700;500 800;600 800;700 700; 700 600]);
```



Set Up Listener for Polyline ROI Events

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a `Polyline` ROI on the image.

```
roi = drawpolyline('Color','r');
```

Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays the previous position and the current position of the ROI.

```
function allevents(src,evt)
    evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving current position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved current position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Color','y'` creates a yellow colored `Polyline` object

`Color` — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'Color','r'`

Example: `'Color','green'`

Example: `'Color',[0 0.4470 0.7410]`

`ContextMenu` — Context menu
`ContextMenu` object

Context menu that displays when you right-click the ROI, specified as a `ContextMenu` object. You can create a custom context menu by using the `uicontextmenu` function and then configuring context menu properties.

`Deletable` — Context menu provides option to delete the ROI
`true` or `1` (default) | `false` or `0`

Context menu provides an option to delete the ROI, specified as a numeric or logical `1` (`true`) or `0` (`false`). When the value is `true`, you can delete the ROI interactively using the context menu. When the value is `false`, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the `delete` function.

`DrawingArea` — Area of axes in which you can interactively place ROI
`'auto'` (default) | `'unlimited'` | `[x,y,w,h]`

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
`'auto'`	The drawing area is the current axes limits (default).
`'unlimited'`	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
`[x,y,w,h]`	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

`HandleVisibility` — Visibility of ROI handle in `Children` property of parent
`'on'` (default) | `'off'` | `'callback'`

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI
`'all'` (default) | `'none'` | `'reshape'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable (default).
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area but not reshaped.
`'reshape'`	The ROI can be reshaped but not translated.

`Label` — ROI label

`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background

1 (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color

`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	

RGB Triplet	Appearance
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'LabelTextColor','r'

Example: 'LabelTextColor','green'

Example: 'LabelTextColor',[0 0.4470 0.7410]

LabelVisible — Label visibility
'on' (default) | 'off' | 'hover'

Label visibility, specified as one of these values.

Value	Description
'on'	Label is visible when the ROI is visible.
'hover'	Label is visible only when the mouse is hovering over the ROI.
'off'	Label is not visible.

LineWidth — Width of ROI border
positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

MarkerSize — Marker size
positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

Parent — ROI parent
Axes object | UIAxes object

ROI parent, specified as an Axes or UIAxes object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Position — Position of ROI
n-by-2 numeric matrix

Position of the ROI, specified as an *n*-by-2 numeric matrix where *n* is the number of vertices or points defining the ROI. Each row represents the [*x* *y*] coordinates of a vertex or point.

Selected — Selection state of ROI
false or 0 (default) | true or 1

Selection state of the ROI, specified as a numeric or logical 0 (false) or 1 (true).

`SelectedColor` — Color of ROI when selected

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'SelectedColor','r'`

Example: `'SelectedColor','green'`

Example: `'SelectedColor',[0 0.4470 0.7410]`

`StripeColor` — Color of ROI stripe

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

Tag — Tag to associate with ROI
`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI
any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `drawpolyline` object does not use this data.

`Visible` — ROI visibility
`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Output Arguments

`roi` — Polyline ROI
`Polyline` object

Polyline ROI, returned as an `Polyline` object.

Tips

- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Make drawn line snap at 15 degree angles.	Hold the **Shift** key while drawing.
Finish drawing the ROI.	Double-click, which adds a final new vertex at the pointer position. Right-click, which adds a final new vertex at the pointer position. Press **Enter**, which adds a final new vertex at the pointer position..
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Add a new vertex to the ROI.	Position the pointer over the polygon and double-click. You can also position the pointer over the ROI, right-click, and choose **Add Vertex**.
Remove a vertex from the ROI.	Position the pointer over the ROI, right-click, and choose **Delete Vertex**.
Remove the most recently added vertex but keep drawing.	Press **Backspace**. The function redraws the line from the previous vertex to the current position of the pointer. You can only back up to the first vertex you drew.

Behavior	Keyboard shortcut
Resize (reshape) the ROI.	Position pointer over a vertex and then click and drag. Add a new vertex and then click and drag. Remove a vertex and the shape of the ROI adjusts.
Move the ROI.	Position the pointer over the line, not on a vertex. The pointer changes to the fleur shape. Click and drag the ROI.
Delete the ROI.	Position the pointer anywhere on the ROI and right-click. Select **Delete Polyline** from the context menu. You can also delete the ROI using the `delete` function.

- The `drawpolyline` function creates a `Polyline` object. After you create the object, you can modify the shape, position, appearance, and behavior of the ROI by using these object capabilities.

Capability	Support
Object properties	ROI objects have properties that specify their shape, position, appearance, and behavior. After you create the ROI object, change properties using dot notation. For example, to change the color of the `roi` to yellow, set its `Color` property: `roi.Color = 'yellow'`
Object functions	ROI objects have object functions that operate on the ROIs. For example, if you want to pause the MATLAB command line after creating an ROI, use the `wait` function.
Event notifications	ROI objects can notify your code when certain events occur, such as when the ROI is clicked or when the ROI is being moved. To receive event notifications, set up listeners. When the ROI notifies your application through the listener, it returns data specific to the event. For example, with the `ROIMoved` event, the ROI object returns its previous position and its current position. You can specify a callback function that executes when an event occurs.

  For an example of using event listeners with the `Polyline` object, see “Set Up Listener for Polyline ROI Events” on page 1-923.

Compatibility Considerations

`UIContextMenu` name-value argument is not recommended
Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` name-value argument to assign a context menu to an ROI object is not recommended. Use the `ContextMenu` name-value argument instead. The values are the same.

There are no plans to remove support for the `UIContextMenu` name-value argument at this time.

See Also

`Polyline` | `drawcrosshair` | `drawline` | `drawpolygon`

Topics

“Use Polyline to Create Angle Measurement Tool”
“Create ROI Shapes”
“Using ROIs in Apps Created with App Designer”
“Use Wait Function After Drawing ROI”

Introduced in R2018b

drawrectangle

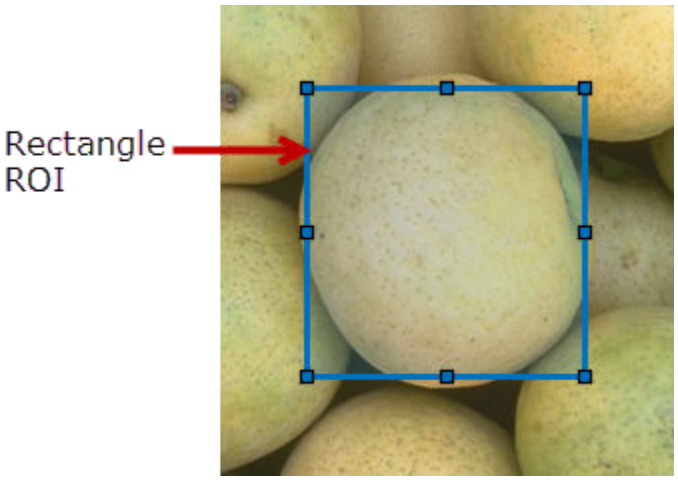
Create customizable rectangular ROI

Syntax

```
roi = drawrectangle
roi = drawrectangle(ax)
roi = drawrectangle( ___ ,Name,Value)
```

Description

The `drawrectangle` function creates a `Rectangle` object that specifies the shape and position of a rectangular region of interest (ROI). You can create the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments. You can also specify the initial appearance and behavior of the ROI.



After you create the ROI, you can use object properties, object functions, and event notifications to customize the shape, position, appearance, and behavior of the ROI. For more information about using these capabilities, see “Tips” on page 1-947.

`roi = drawrectangle` creates a `Rectangle` object and enables interactive drawing of the ROI on the current axes.

To draw the ROI, position the pointer on the image. The cursor changes to a fleur shape. Click and drag to draw the rectangular ROI. To finish the ROI, release the mouse button. For more information about using the ROI, including keyboard shortcuts and context menu options, see “Tips” on page 1-947.

`roi = drawrectangle(ax)` creates the ROI in the axes specified by `ax`.

`roi = drawrectangle( ___ ,Name,Value)` modifies the appearance and behavior of the ROI using one or more name-value pairs. Unspecified name-value pairs are set to the default value.

Examples

Draw Nested Rectangular ROIs

Read an image into the workspace and display it.

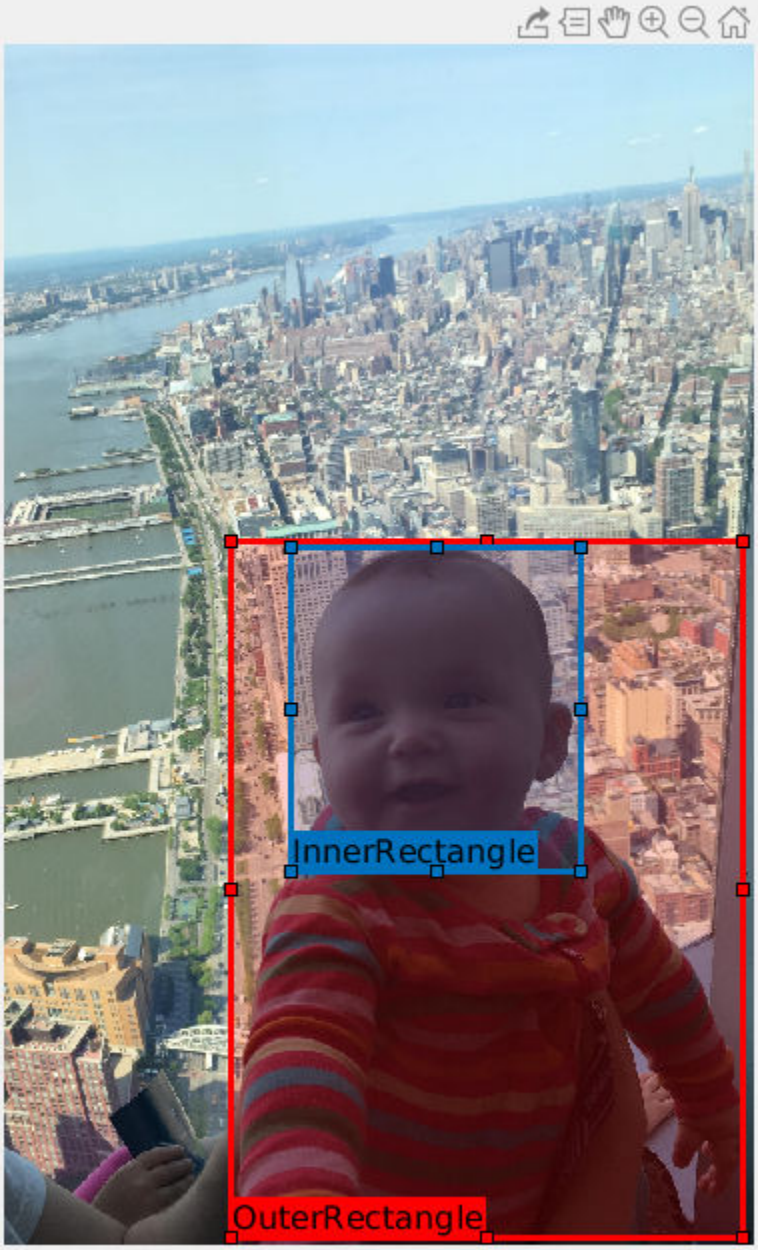
```
imshow(imread('baby.jpg'))
```

Draw a red rectangular ROI with the label `'OuterRectangle'`.

```
r1 = drawrectangle('Label','OuterRectangle','Color',[1 0 0]);
```

Draw another rectangular ROI, restricting the drawing area to the area inside the first rectangle.

```
r2 = drawrectangle('Label','InnerRectangle','DrawingArea',r1.Position);
```



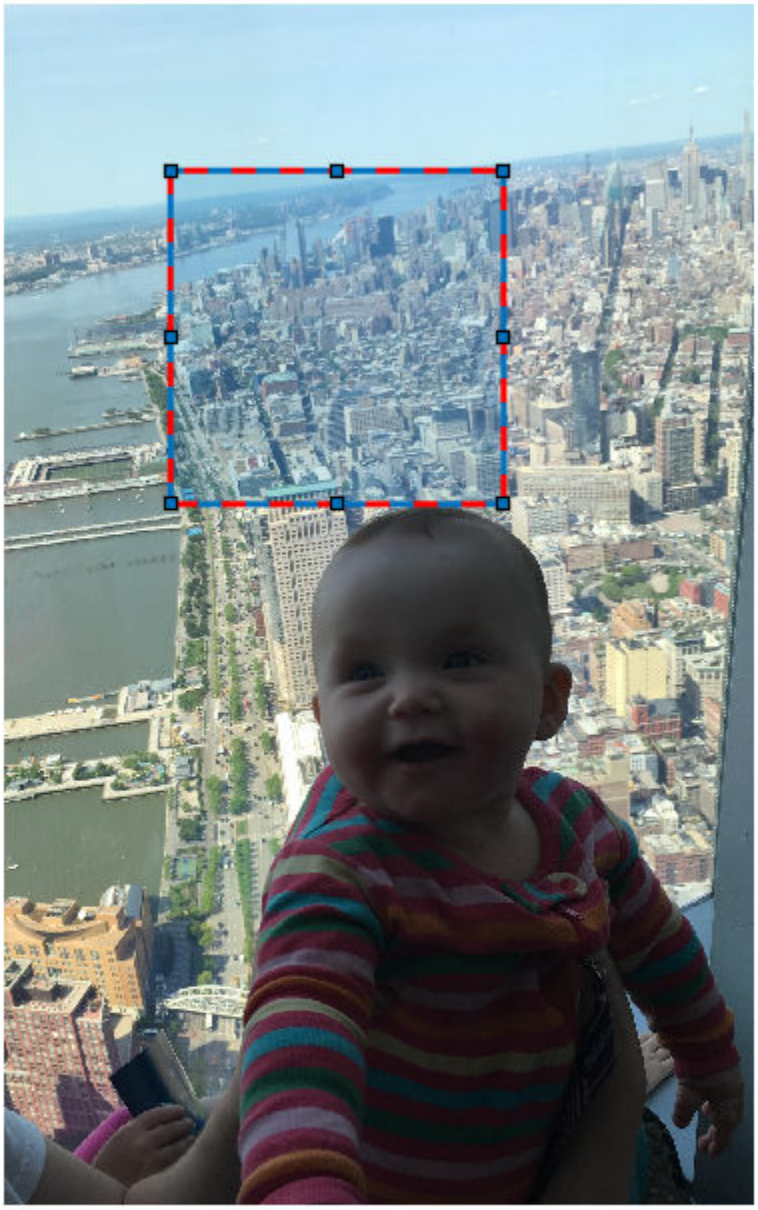
Create Rectangular ROI Programmatically

Read image into the workspace and display it.

```
I = imread('baby.jpg');
figure
imshow(I)
```

Draw a rectangular ROI over the image, using named parameters to specify the location and size of the rectangle. The example also specifies that the edge of the rectangle is a striped line.

```
h = drawrectangle('Position',[500,500,1000,1000],'StripeColor','r');
```



Set Up Listener for Rectangle ROI Events

Read an image into the workspace.

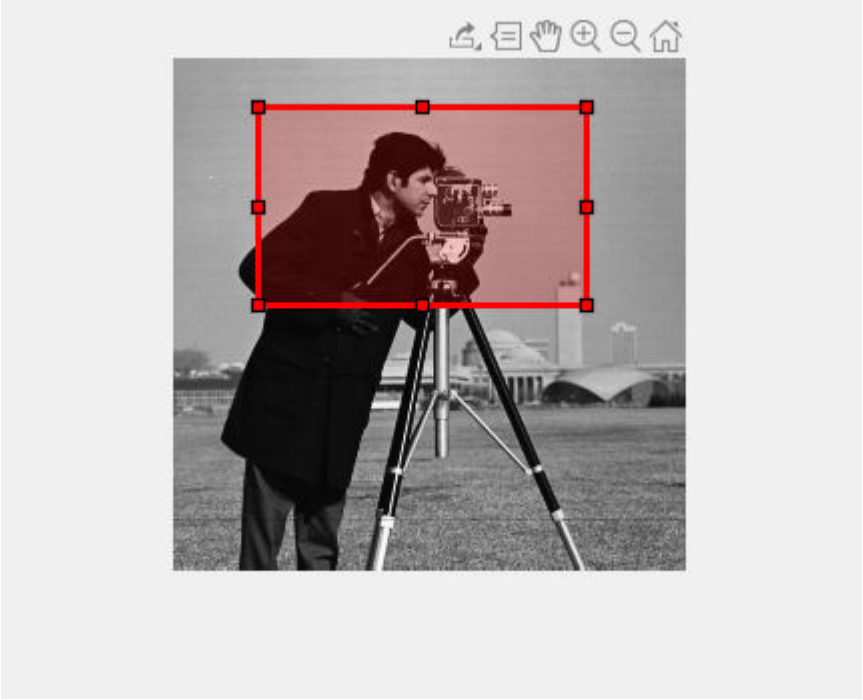
```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a rectangular ROI on the image.

```
roi = drawrectangle('Color','r');
```



Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays the previous position and the current position of the ROI.

```
function allevents(src,evt)
    evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving current position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved current position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Color','y'` creates a yellow colored `Rectangle` object

`AspectRatio` — Aspect ratio of rectangle

nonnegative number

Aspect ratio of the rectangle, specified as a nonnegative number. The aspect ratio is defined as `height/width`.

`Color` — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	

RGB Triplet	Appearance
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0.4470 0.7410]

ContextMenu — Context menu
ContextMenu object

Context menu that displays when you right-click the ROI, specified as a ContextMenu object. You can create a custom context menu by using the uicontextmenu function and then configuring context menu properties.

Deletable — Context menu provides option to delete the ROI
true or 1 (default) | false or 0

Context menu provides an option to delete the ROI, specified as a numeric or logical 1 (true) or 0 (false). When the value is true, you can delete the ROI interactively using the context menu. When the value is false, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the delete function.

DrawingArea — Area of axes in which you can interactively place ROI
'auto' (default) | 'unlimited' | [x,y,w,h]

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
'auto'	The drawing area is the current axes limits (default).
'unlimited'	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
[x,y,w,h]	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

FaceAlpha — Transparency of ROI face
0.2 (default) | number in the range [0, 1]

Transparency of the ROI face, specified as a number in the range [0, 1]. When the value is 1, the ROI face is completely opaque. When the value is 0, the ROI face is completely transparent.

FaceSelectable — ROI face can capture clicks
true or 1 (default) | false or 0

ROI face can capture clicks, specified as a numeric or logical 1 (true) or 0 (false). When true, the ROI face captures mouse clicks. When false, the ROI face does not capture mouse clicks.

`FixedAspectRatio` — Aspect ratio remains constant
`false` or `0` (default) | `true` or `1`

Aspect ratio remains constant during interaction, specified as a numeric or logical `0` (`false`) or `1` (`true`). When the value is `true`, the aspect ratio remains constant when you draw or resize the ROI. When the value is `false`, you can change the aspect ratio when drawing or resizing the ROI.

`HandleVisibility` — Visibility of ROI handle in `Children` property of parent
`'on'` (default) | `'off'` | `'callback'`

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI
`'all'` (default) | `'none'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable.
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area.

`Label` — ROI label
`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background
`1` (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color
`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'LabelTextColor','r'`

Example: `'LabelTextColor','green'`

Example: `'LabelTextColor',[0 0.4470 0.7410]`

`LabelVisible` — Visibility of ROI label

`'on'` (default) | `'hover'` | `'inside'` | `'off'`

Visibility of the ROI label, specified as one of these values.

Value	Description
`'on'`	Label is visible when the ROI is visible, and the `Label` property is nonempty (default).
`'hover'`	Label is visible only when the mouse hovers over the ROI.
`'inside'`	Label is visible only when there is adequate space inside the ROI to display it.
`'off'`	Label is not visible.

`LineWidth` — Width of ROI border

positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

`MarkerSize` — Marker size
positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

`Parent` — ROI parent
`Axes` object | `UIAxes` object

ROI parent, specified as an `Axes` or `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

`Position` — Position of ROI
1-by-4 numeric vector

Position of the ROI, specified as a 1-by-4 numeric vector of the form [*xmin*, *ymin*, *width*, *height*]. *xmin* and *ymin* specify the coordinates of the upper left corner of the rectangle. *width* and *height* specify the width and height of the rectangle and must be nonnegative.

`Rotatable` — Ability of ROI to be rotated
`false` or `0` (default) | `true` or `1`

Ability of the ROI to be rotated, specified as a numeric or logical `0` (`false`) or `1` (`true`). When the value is `true`, you can rotate the rectangle by clicking near the markers at the corners. When the value is `false`, you cannot rotate the rectangle.

`RotationAngle` — Rotation angle
`0` (default) | number

Rotation angle of the ROI, specified as a number. The angle is measured in degrees in a clockwise direction around the center of the ROI.

`Selected` — Selection state of ROI
`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`).

`SelectedColor` — Color of ROI when selected
`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'SelectedColor','r'`

Example: `'SelectedColor','green'`

Example: `'SelectedColor',[0 0.4470 0.7410]`

`StripeColor` — Color of ROI stripe

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	

Color Name	Short Name	RGB Triplet	Appearance
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

Tag — Tag to associate with ROI

`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI

any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `drawrectangle` object does not use this data.

`Visible` — ROI visibility

`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
'on'	Display the ROI.
'off'	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Output Arguments

`roi` — Rectangular ROI
`Rectangle` object

Rectangular ROI, returned as a `Rectangle` object.

Tips

- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Resize (reshape) the ROI.	Position pointer over a vertex and then click and drag. The rectangle has vertices at each corner and at the midpoint of each side. To preserve the aspect ratio while resizing, press the **Shift** key. To lock the aspect ratio, use the **Fix Aspect Ratio** in the right-click context menu.
Move the ROI.	Position the pointer over the ROI. The pointer changes to the fleur shape. Click and drag the ROI.
Delete the ROI.	Position the pointer anywhere on the ROI and right-click. Select **Delete Rectangle** from the context menu. You can also delete the ROI using the `delete` function.

- The `drawrectangle` function creates a `Rectangle` object. After you create the object, you can modify the shape, position, appearance, and behavior of the ROI by using these object capabilities.

Capability	Support
Object properties	ROI objects have properties that specify their shape, position, appearance, and behavior. After you create the ROI object, change properties using dot notation. For example, to change the color of the `roi` to yellow, set its `Color` property: `roi.Color = 'yellow'`
Object functions	ROI objects have object functions that operate on the ROIs. For example, if you want to pause the MATLAB command line after creating an ROI, use the `wait` function.

Capability	Support
Event notifications	ROI objects can notify your code when certain events occur, such as when the ROI is clicked or when the ROI is being moved. To receive event notifications, set up listeners. When the ROI notifies your application through the listener, it returns data specific to the event. For example, with the `ROIMoved` event, the ROI object returns its previous position and its current position. You can specify a callback function that executes when an event occurs.

For an example of using event listeners with the `Rectangle` object, see “Set Up Listener for Rectangle ROI Events” on page 1-938.

Compatibility Considerations

`UIContextMenu` name-value argument is not recommended
Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` name-value argument to assign a context menu to an ROI object is not recommended. Use the `ContextMenu` name-value argument instead. The values are the same.

There are no plans to remove support for the `UIContextMenu` name-value argument at this time.

See Also

`Rectangle` | `drawcuboid` | `drawpolygon`

Topics
“Rotate Image Interactively Using Rectangle ROI”
“Create ROI Shapes”
“Using ROIs in Apps Created with App Designer”
“Use Wait Function After Drawing ROI”

Introduced in R2018b

edge

Find edges in 2-D grayscale image

Syntax

```
BW = edge(I)
BW = edge(I,method)
BW = edge(I,method,threshold)
BW = edge(I,method,threshold,direction)
BW = edge(___,"nothinning")
BW = edge(I,method,threshold,sigma)
BW = edge(I,method,threshold,h)

[BW,threshOut] = edge(___)
[BW,threshOut,Gx,Gy] = edge(___)
```

Description

`BW = edge(I)` returns a binary image `BW` containing `1`s where the function finds edges in the grayscale or binary image `I` and `0`s elsewhere. By default, `edge` uses the Sobel edge detection method.

Tip To find edges in a 3-D grayscale or binary image, use the `edge3` function.

`BW = edge(I,method)` detects edges in image `I` using the edge-detection algorithm specified by `method`.

`BW = edge(I,method,threshold)` returns all edges that are stronger than `threshold`.

`BW = edge(I,method,threshold,direction)` specifies the orientation of edges to detect. The Sobel and Prewitt methods can detect edges in the vertical direction, horizontal direction, or both. The Roberts method can detect edges at angles of 45° from horizontal, 135° from horizontal, or both. This syntax is valid only when `method` is `"Sobel"`, `"Prewitt"`, or `"Roberts"`.

`BW = edge(___,"nothinning")` skips the edge-thinning stage, which can improve performance. This syntax is valid only when `method` is `"Sobel"`, `"Prewitt"`, or `"Roberts"`.

`BW = edge(I,method,threshold,sigma)` specifies `sigma`, the standard deviation of the filter. This syntax is valid only when `method` is `"log"` or `"Canny"`.

`BW = edge(I,method,threshold,h)` detects edges using the `"zerocross"` method with a filter, `h`, that you specify. This syntax is valid only when `method` is `"zerocross"`.

`[BW,threshOut] = edge(___)` also returns the threshold value.

`[BW,threshOut,Gx,Gy] = edge(___)` also returns the directional gradients. For the Sobel and Prewitt methods, `Gx` and `Gy` correspond to the horizontal and vertical gradients. For the Roberts methods, `Gx` and `Gy` correspond to the gradient at angles of 135° and 45° counterclockwise from

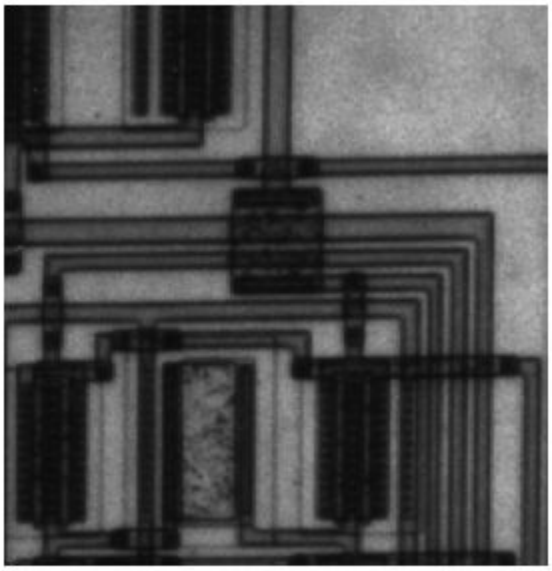
horizontal, respectively. This syntax is valid only when `method` is `"Sobel"`, `"Prewitt"`, or `"Roberts"`.

Examples

Compare Edge Detection Using Canny and Prewitt Methods

Read a grayscale image into the workspace and display it.

```
I = imread('circuit.tif');
imshow(I)
```



Find edges using the Canny method.

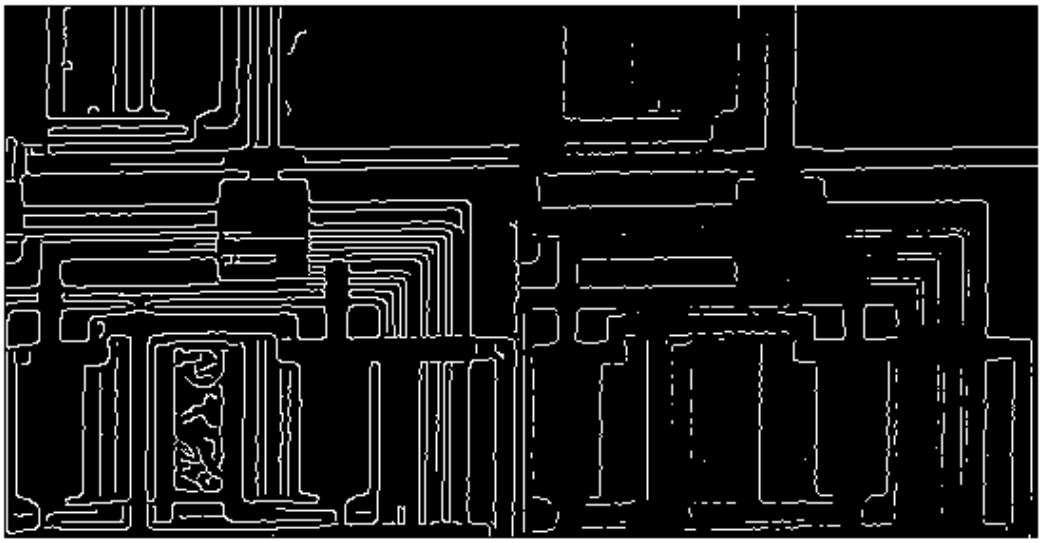
```
BW1 = edge(I,'Canny');
```

Find edges using the Prewitt method.

```
BW2 = edge(I,'Prewitt');
```

Display both results side-by-side.

```
imshowpair(BW1,BW2,'montage')
```



Input Arguments

`I` — Input image

2-D grayscale image | 2-D binary image

Input image, specified as a 2-D grayscale image or 2-D binary image.

For the `"approxcanny"` method, images of data type `single` or `double` must be normalized to the range [0, 1]. If `I` has values outside the range [0, 1], then you can use the `rescale` function to rescale values to the expected range.

Data Types: `double` | `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`method` — Edge detection method

`"Sobel"` (default) | `"Prewitt"` | `"Roberts"` | `"log"` | `"zerocross"` | `"Canny"` | `"approxcanny"`

Edge detection method, specified as one of the following.

Method	Description
`"Sobel"`	Finds edges at those points where the gradient of the image `I` is maximum, using the Sobel approximation to the derivative.
`"Prewitt"`	Finds edges at those points where the gradient of `I` is maximum, using the Prewitt approximation to the derivative.
`"Roberts"`	Finds edges at those points where the gradient of `I` is maximum, using the Roberts approximation to the derivative.

Method	Description
`"log"`	Finds edges by looking for zero-crossings after filtering `I` with a Laplacian of Gaussian (LoG) filter.
`"zerocross"`	Finds edges by looking for zero-crossings after filtering `I` with a filter that you specify, `h`
`"Canny"`	Finds edges by looking for local maxima of the gradient of `I`. The `edge` function calculates the gradient using the derivative of a Gaussian filter. This method uses two thresholds to detect strong and weak edges, including weak edges in the output if they are connected to strong edges. By using two thresholds, the Canny method is less likely than the other methods to be fooled by noise, and more likely to detect true weak edges.
`"approxcanny"`	Finds edges using an approximate version of the Canny edge detection algorithm that provides faster execution time at the expense of less precise detection. Floating point images are expected to be normalized to the range [0, 1].

`threshold` — Sensitivity threshold
numeric scalar | 2-element vector | `[]`

Sensitivity threshold, specified as a numeric scalar for any `method`, or a 2-element vector for the `"Canny"` and `"approxcanny"` methods. `edge` ignores all edges that are not stronger than `threshold`. For more information about this parameter, see “Algorithm” on page 1-954.

- If you do not specify `threshold`, or if you specify an empty array (`[]`), then `edge` chooses the value or values automatically.
- For the `"log"` and `"zerocross"` methods, if you specify the threshold value `0`, then the output image has closed contours because it includes all the zero-crossings in the input image.
- The `"Canny"` and `"approxcanny"` methods use two thresholds. `edge` disregards all edges with edge strength below the lower threshold, and preserves all edges with edge strength above the higher threshold. You can specify `threshold` as a 2-element vector of the form `[low high]` with `low` and `high` values in the range [0, 1]. You can also specify `threshold` as a numeric scalar, which `edge` assigns to the higher threshold. In this case, `edge` uses `threshold*0.4` as the lower threshold.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`direction` — Direction of edges to detect
`"both"` (default) | `"horizontal"` | `"vertical"`

Direction of edges to detect, specified as `"horizontal"`, `"vertical"`, or `"both"`. The `direction` argument is only valid when the `method` is `"Sobel"`, `"Prewitt"`, or `"Roberts"`.

Note If you select the Roberts `method`, then the `"horizontal"` direction actually detects edges at an angle of 135° counterclockwise from horizontal, and the `"vertical"` direction detects edges at an angle of 45° counterclockwise from horizontal.

Data Types: `char` | `string`

`h` — Filter
numeric matrix

Filter, specified as a numeric matrix. The `h` argument is supported by the `"zerocross"` method only.

Data Types: `double`

`sigma` — Standard deviation of filter

numeric scalar

Standard deviation of the filter, specified as a numeric scalar. The `sigma` argument is supported by the `"Canny"` and `"log"` methods only.

Method	Description
`"Canny"`	Scalar value that specifies the standard deviation of the Gaussian filter. The default is `sqrt(2)`. `edge` chooses the size of the filter automatically, based on `sigma`.
`"log"` (Laplacian of Gaussian)	Scalar value that specifies the standard deviation of the Laplacian of Gaussian filter. The default is `2`. The size of the filter is `n`-by-`n`, where `n=ceil(sigma*3)*2+1`.

Data Types: `double`

Output Arguments

`BW` — Output binary image

logical array

Output binary image, returned as a logical array of the same size as `I`, with `1`s where the function finds edges in `I` and `0`s elsewhere.

`threshOut` — Calculated threshold

numeric scalar | 2-element vector | `[]`

Calculated threshold value used in the computation, returned as a 2-element vector for the `"Canny"` `method`, an empty vector (`[]`) for the `"approxcanny"` method, or a numeric scalar for all other edge detection methods.

`Gx` — Horizontal gradient

numeric array

Horizontal gradient, returned as a numeric array of the same size as `I`. A large horizontal gradient magnitude indicates a strong vertical edge.

Note If you select the Roberts `method`, then `edge` returns the gradient calculated at an angle of 135° counterclockwise from horizontal.

`Gy` — Vertical gradient

numeric array

Vertical gradient, returned as a numeric array of the same size as `I`. A large vertical gradient magnitude indicates a strong horizontal edge.

Note If you select the Roberts `method`, then `edge` returns the gradient calculated at an angle of 45° counterclockwise from horizontal.

Algorithms

- For the gradient-magnitude edge detection methods (Sobel, Prewitt, and Roberts), `edge` uses `threshold` to threshold the calculated gradient magnitude.
- For the zero-crossing methods, including Laplacian of Gaussian, `edge` uses `threshold` as a threshold for the zero-crossings. In other words, a large jump across zero is an edge, while a small jump is not.
- The Canny method applies two thresholds to the gradient: a high threshold for low edge sensitivity and a low threshold for high edge sensitivity. `edge` starts with the low sensitivity result and then grows it to include connected edge pixels from the high sensitivity result. This helps fill in gaps in the detected edges.
- In all cases, `edge` chooses the default threshold heuristically, depending on the input data. The best way to vary the threshold is to run `edge` once, capturing the calculated threshold as the second output argument. Then, starting from the value calculated by `edge`, adjust the threshold higher to detect fewer edge pixels, or lower to detect more edge pixels.

Compatibility Considerations

edge Uses New Algorithm for Canny Method

Behavior changed in R2011a

The function `edge` changed in Version 7.2 (R2011a). Previous versions of the Image Processing Toolbox used a different algorithm for the Canny method. If you need the same results produced by the previous implementation, use the following syntax: `BW = edge(I,"canny_old",___)`

References

[1] Canny, John, "A Computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-8, No. 6, 1986, pp. 679-698.

[2] Lim, Jae S., *Two-Dimensional Signal and Image Processing*, Englewood Cliffs, NJ, Prentice Hall, 1990, pp. 478-488.

[3] Parker, James R., *Algorithms for Image Processing and Computer Vision,* New York, John Wiley & Sons, Inc., 1997, pp. 23-29.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `edge` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `edge` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but

limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

- The `method`, `direction`, and `sigma` arguments must be compile-time constants.
- The `"approxcanny"` method is not supported.
- The `Gx` and `Gh` output arguments are not supported.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The `method`, `direction`, and `sigma` arguments must be compile-time constants.
- The `"approxcanny"` method is not supported.
- The `Gx` and `Gy` output arguments are not supported.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- The `"Canny"` and `"approxcanny"` methods are not supported.

For more information, see “Image Processing on a GPU”.

See Also
`edge3` | `fspecial` | `imgradient` | `imgradientxy`

Introduced before R2006a

edge3

Find edges in 3-D grayscale volume

Syntax

```
BW = edge3(V,"approxcanny",thresh)
BW = edge3(V,"approxcanny",thresh,sigma)
BW = edge3(V,"Sobel",thresh)
BW = edge3(V,"Sobel",thresh,"nothinning")
```

Description

`BW = edge3(V,"approxcanny",thresh)` returns the edges found in the grayscale or binary volume `V` using the approximate Canny method. The approximate Canny method finds edges by looking for local maxima of the gradient of `V`. `edge3` calculates the gradient using the derivative of a Gaussian smoothed volume.

The approximate Canny method uses two thresholds to detect strong and weak edges, and includes the weak edges in the output only if they are connected to strong edges. This method is more likely than the Sobel method to detect true weak edges.

`BW = edge3(V,"approxcanny",thresh,sigma)` returns the edges found in the intensity or binary volume `V`, where `sigma` specifies the standard deviation of the Gaussian smoothing filter. `edge3` chooses the size of the filter automatically, based on `sigma`.

`BW = edge3(V,"Sobel",thresh)` accepts an intensity or a binary volume `V` and returns a binary volume `BW` with `1`s where the function finds edges in `V` and `0`s elsewhere.

The Sobel method finds edges using the Sobel approximation to the derivative. It returns edges at those points where the gradient of `V` is maximum. `edge3` ignores all edges that are not stronger than `thresh`.

`BW = edge3(V,"Sobel",thresh,"nothinning")` speeds up the operation of the algorithm by skipping the additional edge-thinning stage. By default, or when `"thinning"` is specified, `edge3` applies edge thinning.

Examples

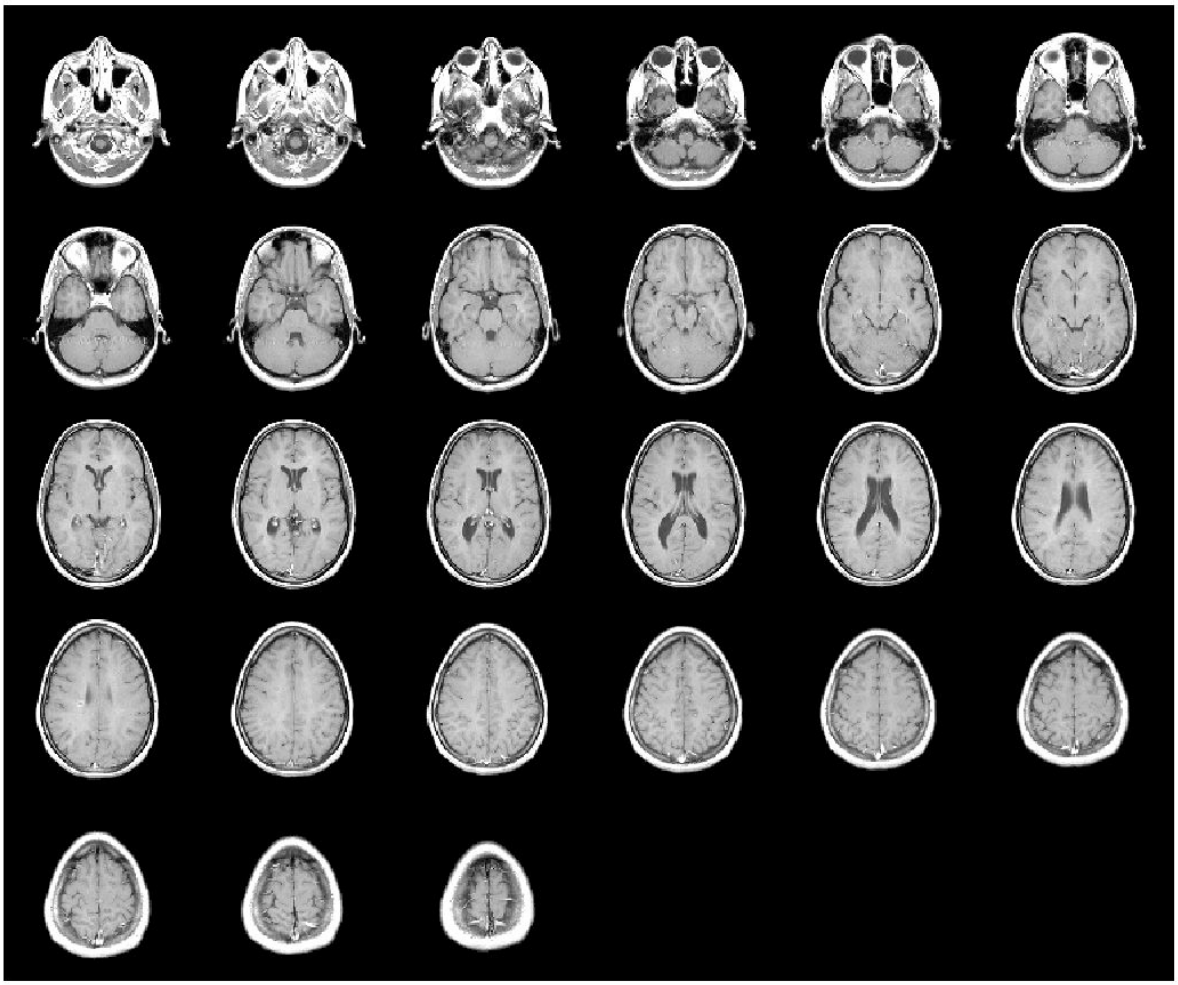
Find Edges of MRI Volume using Approximate Canny Method

Load volumetric data and remove any singleton dimensions.

```
load mri
V = squeeze(D);
```

Visualize original image.

```
montage(reshape(V,size(D)),map);
```

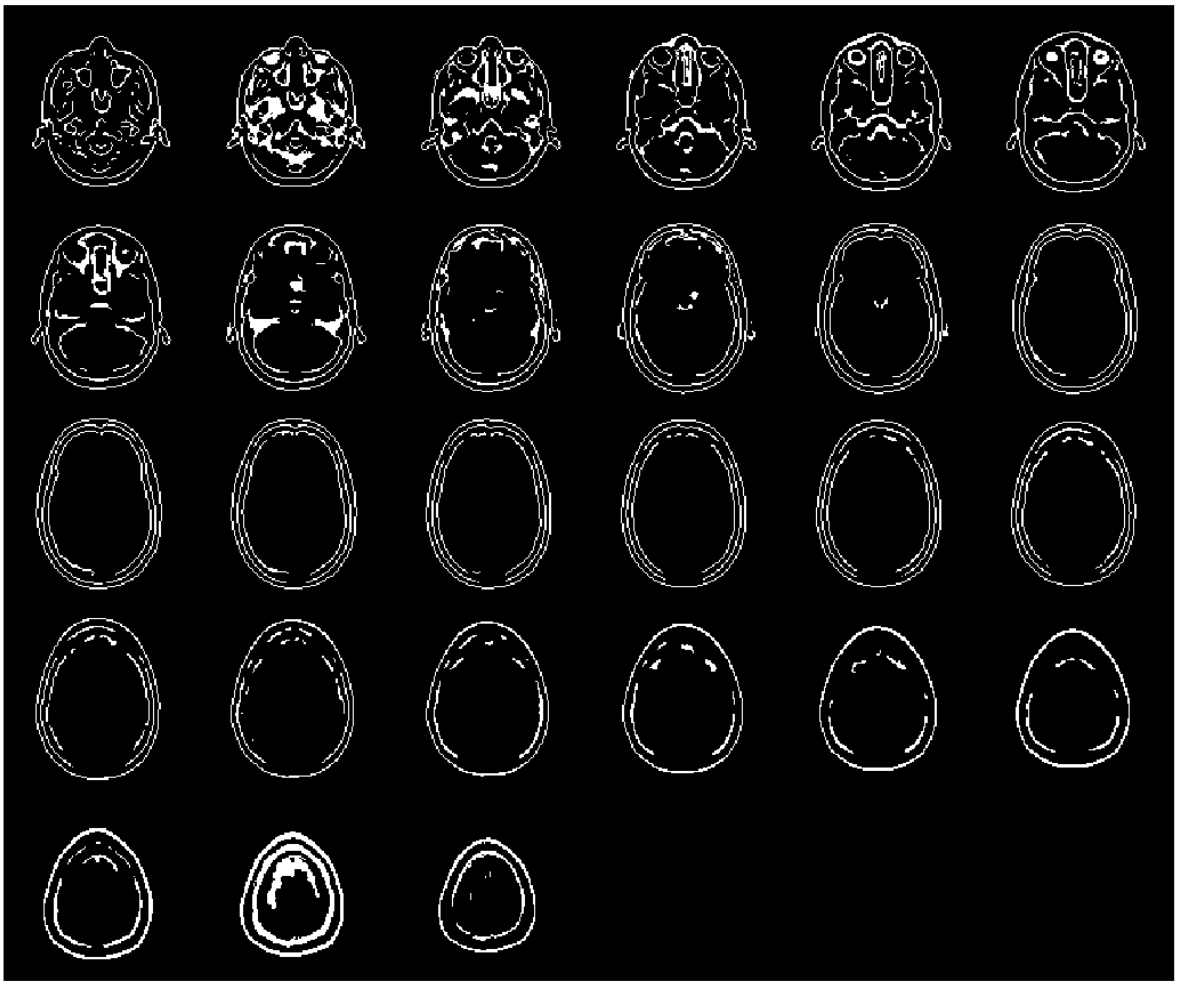


Detect edges in the volume.

```
BW = edge3(V,'approxcanny',0.6);
```

Visualize the detected edges. You can also view the result using the Volume Viewer app.

```
montage(reshape(BW,size(D)))
```



Input Arguments

`V` — Input volume

3-D numeric array

Input volume, specified as a 3-D numeric array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`thresh` — Sensitivity threshold

numeric scalar | 2-element numeric row vector

Sensitivity threshold, specified as one of the following.

Method	Threshold value
Canny	Numeric scalar
Approximate Canny	2-element numeric row vector. The first element is the low threshold, and the second element is the high threshold, `[lowthresh highthresh]`
	Numeric scalar representing the high threshold. `edge3` sets the low threshold as `0.4*thresh`.
Sobel	Numeric scalar

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`sigma` — Standard deviation of Gaussian filter
`sqrt(2)` | numeric scalar | 1-by-3 numeric vector

Standard deviation of the Gaussian filter, specified as a numeric scalar for isotropic volumes or a 1-by-3 numeric vector of the form `[SigmaX SigmaY SigmaZ]` for anisotropic volumes that have different scales in each direction.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

`BW` — Detected edges
3-D numeric array

Detected edges, returned as a 3-D numeric array of the same size as `V`. Pixel values of `1` indicate edges and pixel values of `0` indicate flat regions.

See Also
`edge`

Introduced in R2017b

edgetaper

Taper discontinuities along image edges

Syntax

```
J = edgetaper(I,PSF)
```

Description

`J = edgetaper(I,PSF)` blurs the edges of the input image `I` using the point spread function `PSF`.

The output image `J` is the weighted sum of the original image `I` and its blurred version. The weighting array, determined by the autocorrelation function of `PSF`, makes `J` equal to `I` in its central region, and equal to the blurred version of `I` near the edges.

The `edgetaper` function reduces the ringing effect in image deblurring methods that use the discrete Fourier transform, such as `deconvwnr`, `deconvreg`, and `deconvlucy`.

Examples

Blur the Edges of an Image

```
original = imread('cameraman.tif');
PSF = fspecial('gaussian',60,10);
edgesTapered = edgetaper(original,PSF);
figure, imshow(original,[]);
```

```
figure, imshow(edgesTapered,[]);
```

Input Arguments

`I` — Input image

numeric array

Input image, specified as a numeric array.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`PSF` — Point spread function

numeric array

Point spread function, specified as a numeric array. The size of the `PSF` cannot exceed half of the image size in any dimension.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

Output Arguments

`J` — Weighted sum of original image and its blurred version

numeric array

Weighted sum of original image and its blurred version, returned as a numeric array the same size and class as `I`. The weighting array, determined by the autocorrelation function of `PSF`, makes `J` equal to `I` in its central region, and equal to the blurred version of `I` near the edges.

See Also

deconvlucy | deconvreg | deconvwnr | otf2psf | padarray | psf2otf

Introduced before R2006a

encoderDecoderNetwork

Create encoder-decoder network

Syntax

```
net = encoderDecoderNetwork(inputSize,encoder,decoder)
net = encoderDecoderNetwork(inputSize,encoder,decoder,Name,Value)
```

Description

`net = encoderDecoderNetwork(inputSize,encoder,decoder)` connects an encoder network and a decoder network to create an encoder-decoder network, `net`.

This function requires Deep Learning Toolbox.

`net = encoderDecoderNetwork(inputSize,encoder,decoder,Name,Value)` modifies aspects of the encoder-decoder network using name-value arguments.

Examples

Create U-Net Network from Encoder and Decoder Blocks

Create the encoder module consisting of four encoder blocks.

```
encoderBlock = @(block) [
    convolution2dLayer(3,2^(5+block),"Padding",'same')
    reluLayer
    convolution2dLayer(3,2^(5+block),"Padding",'same')
    reluLayer
    maxPooling2dLayer(2,"Stride",2)];
encoder = blockedNetwork(encoderBlock,4,"NamePrefix","encoder_");
```

Create the decoder module consisting of four decoder blocks.

```
decoderBlock = @(block) [
    transposedConv2dLayer(2,2^(10-block),'Stride',2)
    convolution2dLayer(3,2^(10-block),"Padding",'same')
    reluLayer
    convolution2dLayer(3,2^(10-block),"Padding",'same')
    reluLayer];
decoder = blockedNetwork(decoderBlock,4,"NamePrefix","decoder_");
```

Create the bridge layers.

```
bridge = [
    convolution2dLayer(3,1024,"Padding",'same')
    reluLayer
    convolution2dLayer(3,1024,"Padding",'same')
    reluLayer
    dropoutLayer(0.5)];
```

Specify the network input size.

```
inputSize = [224 224 3];
```

Create the U-Net network by connecting the encoder module, bridge, and decoder module and adding skip connections.

```
unet = encoderDecoderNetwork(inputSize,encoder,decoder, ...
    "OutputChannels",3, ...
    "SkipConnections","concatenate", ...
    "LatentNetwork",bridge)
```

```
unet =
  dlnetwork with properties:

         Layers: [55x1 nnet.cnn.layer.Layer]
    Connections: [62x2 table]
     Learnables: [46x3 table]
          State: [0x3 table]
     InputNames: {'encoderImageInputLayer'}
    OutputNames: {'encoderDecoderFinalConvLayer'}
    Initialized: 1
```

Display the network.

```
analyzeNetwork(unet)
```

Create U-Net from Pretrained GoogLeNet

Create a GAN encoder network with four downsampling operations from a pretrained GoogLeNet network.

```
depth = 4;
[encoder,outputNames] = pretrainedEncoderNetwork('googlenet',depth);
```

Determine the input size of the encoder network.

```
inputSize = encoder.Layers(1).InputSize;
```

Determine the output size of the activation layers in the encoder network by creating a sample data input and then calling `forward`, which returns the activations.

```
exampleInput = dlarray(zeros(inputSize),'SSC');
exampleOutput = cell(1,length(outputNames));
[exampleOutput{:}] = forward(encoder,exampleInput,'Outputs',outputNames);
```

Determine the number of channels in the decoder blocks as the length of the third channel in each activation.

```
numChannels = cellfun(@(x) size(extractdata(x),3),exampleOutput);
numChannels = fliplr(numChannels(1:end-1));
```

Define a function that creates an array of layers for one decoder block.

```
decoderBlock = @(block) [
    transposedConv2dLayer(2,numChannels(block),'Stride',2)
```

```
    convolution2dLayer(3,numChannels(block),'Padding','same')
    reluLayer
    convolution2dLayer(3,numChannels(block),'Padding','same')
    reluLayer];
```

Create the decoder module with the same number of upsampling blocks as there are downsampling blocks in the encoder module.

```
decoder = blockedNetwork(decoderBlock,depth);
```

Create the U-Net network by connecting the encoder module and decoder module and adding skip connections.

```
net = encoderDecoderNetwork([224 224 3],encoder,decoder, ...
   'OutputChannels',3,'SkipConnections','concatenate')
```

```
net =
  dlnetwork with properties:

         Layers: [139x1 nnet.cnn.layer.Layer]
    Connections: [167x2 table]
     Learnables: [116x3 table]
          State: [0x3 table]
     InputNames: {'data'}
    OutputNames: {'encoderDecoderFinalConvLayer'}
    Initialized: 1
```

Display the network.

```
analyzeNetwork(net)
```

Input Arguments

`inputSize` — Network input size

3-element vector of positive integers

Network input size, specified as a 3-element vector of positive integers. `inputSize` has the form [*H W C*], where *H* is the height, *W* is the width, and *C* is the number of channels.

Example: `[28 28 3]` specifies an input size of 28-by-28 pixels for a 3-channel image.

`encoder` — Encoder network

`dlnetwork` object

Encoder network, specified as a `dlnetwork` object.

`decoder` — Decoder network

`dlnetwork` object

Decoder network, specified as a `dlnetwork` object. The network must have a single input and a single output.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'SkipConnections',"concatenate"` specifies the type of skip connection between the encoder and decoder networks as concatenation.

`LatentNetwork` — Network connecting encoder and decoder

`[]` (default) | layer object | array of layer objects

Network connecting the encoder and decoder, specified as a layer or array of layers.

`FinalNetwork` — Network connected to output of decoder

`[]` (default) | layer object | array of layer objects

Network connected to the output of the decoder, specified as a layer or array of layers. If you specify the '`OutputChannels`' argument, then the final network is connected after the final 1-by-1 convolution layer of the decoder.

`OutputChannels` — Number of output channels

`[]` (default) | positive integer

Number of output channels of the decoder network, specified as a positive integer. If you specify this argument, then the final layer of the decoder performs a 1-by-1 convolution operation with the specified number of channels.

`SkipConnectionNames` — Names of pairs of encoder/decoder layers

`"auto"` (default) | *M*-by-2 string array

Names of pairs of encoder/decoder layers whose activations are merged by skip connections, specified as one of these values.

- `"auto"` — The `encoderDecoderNetwork` function determines the names of pairs of encoder/decoder layers automatically.
- *M*-by-2 string array — The first column is the name of the encoder layer and the second column is the name of the respective decoder layer.

When you specify the '`SkipConnections`' argument as `"none"`, the `encoderDecoderNetwork` function ignores the value of '`SkipConnectionNames`'.

Data Types: `char` | `string`

`SkipConnections` — Type of skip connection

`"none"` (default) | `"auto"` | `"concatenate"`

Type of skip connection between the encoder and decoder networks, specified as `"none"`, `"auto"`, or `"concatenate"`.

Data Types: `char` | `string`

Output Arguments

net — Encoder/decoder network
`dlnetwork` object

Encoder/decoder network, returned as a `dlnetwork` object.

See Also

`blockedNetwork` | `pretrainedEncoderNetwork`

Topics
“Create Modular Neural Networks”
“Get Started with GANs for Image-to-Image Translation”

Introduced in R2021a

entropy

Entropy of grayscale image

Syntax

```
e = entropy(I)
```

Description

`e = entropy(I)` returns `e`, a scalar value representing the entropy of grayscale image `I`.

Examples

Calculate Entropy of Grayscale Image

Read image into the workspace.

```
I = imread('circuit.tif');
```

Calculate the entropy.

```
J = entropy(I)
```

```
J = 6.9439
```

Input Arguments

`I` — Grayscale image

numeric array

Grayscale image, specified as a numeric array of any dimension.

Data Types: `double` | `uint8` | `uint16` | `uint32` | `logical`

Output Arguments

`e` — Entropy

numeric scalar

Entropy of image `I`, returned as a numeric scalar.

Data Types: `double`

More About

Entropy

Entropy is a statistical measure of randomness that can be used to characterize the texture of the input image.

Entropy is defined as `-sum(p.*log2(p))`, where `p` contains the normalized histogram counts returned from `imhist`.

Tips

- By default, `entropy` uses two bins for logical arrays and 256 bins for `uint8`, `uint16`, or `double` arrays. `entropy` converts any class other than `logical` to `uint8` for the histogram count calculation so that the pixel values are discrete and directly correspond to a bin value.

References

[1] Gonzalez, R. C., R. E. Woods, and S. L. Eddins. *Digital Image Processing Using MATLAB*. New Jersey, Prentice Hall, 2003, Chapter 11.

See Also

`imhist` | `entropyfilt`

Introduced before R2006a

entropyfilt

Local entropy of grayscale image

Syntax

```
J = entropyfilt(I)
J = entropyfilt(I,nhood)
```

Description

`J = entropyfilt(I)` returns the array `J`, where each output pixel contains the entropy value of the 9-by-9 neighborhood around the corresponding pixel in the input image `I`.

For pixels on the borders of `I`, `entropyfilt` uses symmetric padding. In symmetric padding, the values of padding pixels are a mirror reflection of the border pixels in `I`.

`J = entropyfilt(I,nhood)` performs entropy filtering of the input image `I` using the neighborhood `nhood`.

Examples

Perform Entropy Filtering

This example shows how to perform entropy filtering using `entropyfilt`. Brighter pixels in the filtered image correspond to neighborhoods in the original image with higher entropy.

Read an image into the workspace.

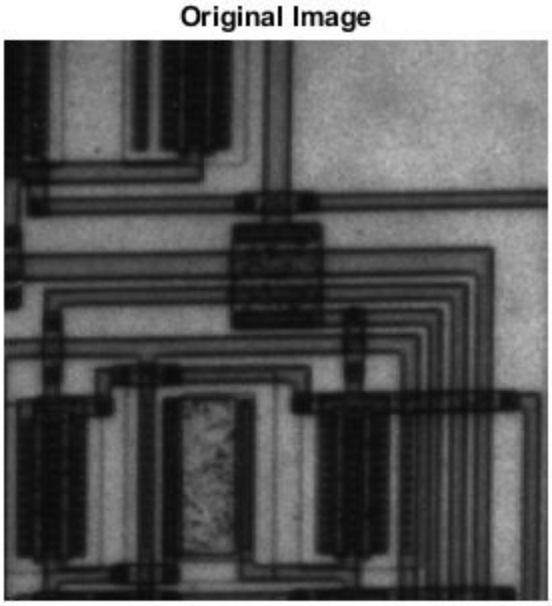
```
I = imread('circuit.tif');
```

Perform entropy filtering using `entropyfilt`.

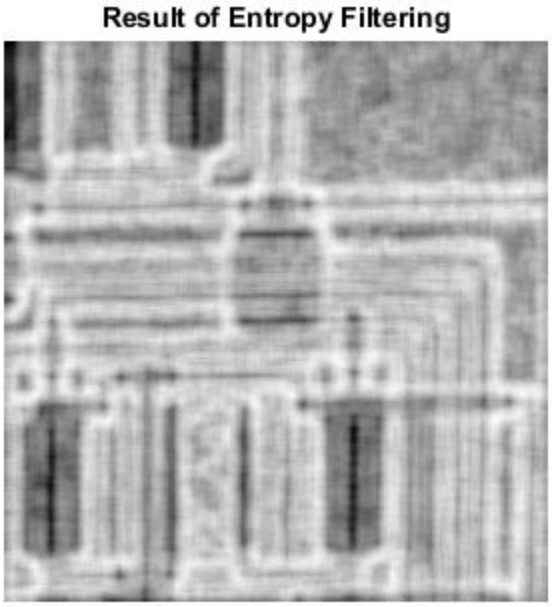
```
J = entropyfilt(I);
```

Show the original image and the processed image.

```
imshow(I)
title('Original Image')
```




```
figure
imshow(J,[])
title('Result of Entropy Filtering')
```



Input Arguments

I — Image to be filtered

numeric array

Image to be filtered, specified as a numeric array of any dimension. If the input image has more than two dimensions (`ndims(I) > 2`), such as for an RGB image, then `entropyfilt` filters all 2-D planes along the higher dimensions. For data types `double` and `single`, the intensity values of `I` must be in the range [0, 1].

Data Types: `double` | `uint8` | `uint16` | `uint32` | `logical`

nhood — Neighborhood

`true(9)` (default) | numeric array | logical array

Neighborhood, specified as a numeric or logical array containing `0`s and `1`s. The size of `nhood` must be odd in each dimension.

By default, `entropyfilt` uses the neighborhood `true(9)`. The center element of the neighborhood is `floor((size(nhood) + 1)/2)`.

To specify neighborhoods of other shapes, such as a disk, use the `strel` function to create a structuring element object of the desired shape. Then, extract the neighborhood from the structuring element object's `neighborhood` property.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

J — Filtered image

numeric array

Filtered image, returned as a numeric array the same size as the input image `I`. The data type of `J` is `double`. If the input image `I` is a binary image (data type `logical`), then the values of `J` are in the range [0, 1]. For all other data types of `I`, the values of `J` are in the range [0, 8].

Data Types: `double`

More About

Entropy

Entropy is a statistical measure of randomness that can be used to characterize the texture of the input image.

Entropy is defined as `-sum(p.*log2(p))`, where `p` contains the normalized histogram counts returned from `imhist`.

Tips

- By default, `entropyfilt` uses two bins for logical arrays. `entropyfilt` converts any other class to `uint8` for the histogram count calculation and uses 256 bins so that the pixel values are discrete and directly correspond to a bin value.

- If the input image `I` is a grayscale image, then the values of `J` can exceed the range [0, 1] that some Image Processing Toolbox functions expect for images of type `double`. To pass `J` as an input argument to these functions, use the `rescale` function to rescale the values of `J` to the range [0, 1].

References

[1] Gonzalez, R. C., R. E. Woods, and S. L. Eddins. *Digital Image Processing Using MATLAB*. New Jersey, Prentice Hall, 2003, Chapter 11.

See Also

`entropy` | `imhist` | `rangefilt` | `stdfilt`

Topics

“What Is Image Filtering in the Spatial Domain?”
“Image Types in the Toolbox”

Introduced before R2006a

esfrChart

Imatest edge spatial frequency response (eSFR) test chart

Description

An `esfrChart` object stores the positions and measurements of regions of interest (ROIs) of Imatest edge spatial frequency response (eSFR) test charts [1], [2].

The `esfrChart` object supports the Enhanced and Extended versions of the eSFR test chart. These test charts are based on the ISO 12233:2014 standard test chart, and they have extra visual features such as color ROIs and additional slanted edge ROIs. The `esfrChart` object also accepts versions of the Enhanced and Extended eSFR test charts with additional background wedges.

Creation

Syntax

```
chart = esfrChart(A)
chart = esfrChart(A,'Sensitivity',s)
chart = esfrChart(A,'RegistrationPoints',p)
chart = esfrChart( ___ ,Name,Value)
```

Description

`chart = esfrChart(A)` creates an `esfrChart` object from an image of a test chart, `A`. The `esfrChart` object performs automatic detection of the chart position and style.

`chart = esfrChart(A,'Sensitivity',s)` creates an `esfrChart` object using sensitivity `s` during the automatic chart detection.

`chart = esfrChart(A,'RegistrationPoints',p)` creates an `esfrChart` object by specifying the position, `p`, of registration points.

`chart = esfrChart( ___ ,Name,Value)` refines the automatic chart detection using one or more name-value arguments.

Input Arguments

A — Test chart image
m-by-*n*-by-3 numeric array | *m*-by-*n* numeric matrix

Test chart image, specified as an *m*-by-*n*-by-3 numeric array representing an RGB image or an *m*-by-*n* numeric matrix representing a grayscale image. This argument sets the Image property.

If you specify a grayscale image, then the `esfrChart` object simulates a color image by replicating the pixel intensity values across the three color channels. In this case, color measurements returned by the `measureColor` function are meaningless.

Data Types: `single` | `double` | `uint8` | `uint16`

s — Sensitivity

`0.5` (default) | numeric scalar in the range [0, 1]

Sensitivity of chart detection, specified as a numeric scalar in the range [0, 1]. If you set a high sensitivity value, then the `esfrChart` object detects more points of interest with which to register the test chart image.

Data Types: `single` | `double`

p — Position of registration points

4-by-2 numeric matrix

Position of registration points used to orient the image, specified as a 4-by-2 numeric matrix. The four rows correspond to the top-left, top-right, bottom-right, and bottom-left registration points, respectively. The two columns represent pixel coordinates in [*x*, *y*] format. This argument sets the RegistrationPoints property.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Style','Extended'`

Style — Style of test chart

`'Extended'` | `'Enhanced'` | `'WedgeExtended'` | `'WedgeEnhanced'`

Style of test chart, specified as `'Enhanced'`, `'Extended'`, `'WedgeEnhanced'`, or `'WedgeExtended'`. If you do not specify a chart style, then by default the `esfrChart` object estimates the chart style based on the number and position of points of interest. This argument sets the Style property.

Data Types: `char` | `string`

CameraParameters — Camera parameters

`cameraParameters` object

Camera parameters used to compensate for distortion, specified as a `cameraParameters` object. Use of this argument requires Computer Vision Toolbox.

RefinePoints — Refine position of slanted edge ROIs

`true` or `1` (default) | `false` or `0`

Refine the position of slanted edge ROIs, specified as a numeric or logical `1` (`true`) or `0` (`false`).

The `esfrChart` object first performs an initial estimate of ROI positions with respect to the registration points, RegistrationPoints. When you specify '`RefinePoints`' as `true`, the object then refines the slanted edge ROI positions using localized information about the image content. When `false`, the object does not refine the slanted edge ROI positions.

Properties

`Image` — Test chart image

m-by-*n*-by-3 numeric array

Test chart image, specified as an *m*-by-*n*-by-3 numeric array.

Data Types: `single` | `double` | `uint8` | `uint16`

`SlantedEdgeROIs` — Position and intensity values of slanted edges

60-by-1 vector of structures

Position and intensity values of slanted edges, specified as a 60-by-1 vector of structures. Each element in the vector corresponds to one ROI and contains these fields:

Field	Description
`ROI`	A 1-by-4 vector specifying the spatial extent of the ROI. The vector has the form `[X Y Width Height]`. `X` and `Y` are the coordinates of the top-left corner of the ROI. `Width` and `Height` are the width and height of the ROI, in pixels. `ROI` is of data type `double`.
`ROIIntensity`	Array of intensity values within the ROI, in RGB format. The array has dimensions `Height`-by-`Width`-by-3. The data type of `ROIIntensity` matches the data type of the Image property.

The `esfrChart` object excludes some slanted edge ROIs in the `'Enhanced'` and `'WedgeEnhanced'` style test charts.

- When the chart style is `'Enhanced'`, the `esfrChart` object excludes four ROIs, with indices 1, 19, 41, and 59.
- When the chart style is `'Enhanced'`, the `esfrChart` object excludes twelve ROIs, with indices 1, 2, 4, 18, 19, 20, 41, 42, 44, 58, 59, and 60.

For the excluded ROIs, the value of the `ROI` field is `[NaN NaN NaN NaN]` and the value of the `ROIIntensity` field is an empty array, `[]`.

`GrayROIs` — Position and intensity values of gray patches

20-by-1 vector of structures

Position and intensity values of gray patches, specified as a 20-by-1 vector of structures. Each element in the vector corresponds to one ROI and contains these fields:

Field	Description
`ROI`	A 1-by-4 vector specifying the spatial extent of the ROI. The vector has the form `[X Y Width Height]`. `X` and `Y` are the coordinates of the top-left corner of the ROI. `Width` and `Height` are the width and height of the ROI, in pixels. `ROI` is of data type `double`.
`ROIIntensity`	Array of intensity values within the ROI, in RGB format. The array has dimensions `Height`-by-`Width`-by-3. The data type of `ROIIntensity` matches the data type of the Image property.

`ColorROIs` — Position and intensity values of color patches

16-by-1 vector of structures

Position and intensity values of color patches, specified as a 16-by-1 vector of structures. Each element in the vector corresponds to one ROI and contains these fields:

Field	Description
`ROI`	A 1-by-4 vector specifying the spatial extent of the ROI. The vector has the form `[X Y Width Height]`. `X` and `Y` are the coordinates of the top-left corner of the ROI. `Width` and `Height` are the width and height of the ROI, in pixels. `ROI` is of data type `double`.
`ROIIntensity`	Array of intensity values within the ROI, in RGB format. The array has dimensions `Height`-by-`Width`-by-3. The data type of `ROIIntensity` matches the data type of the Image property.

`RegistrationPoints` — Position of registration points
4-by-2 numeric matrix

Position of registration points used to orient the image, specified as a 4-by-2 numeric matrix. The four rows correspond to the top-left, top-right, bottom-right, and bottom-left registration points, respectively. The two columns represent pixel coordinates in [*x*, *y*] format.

Data Types: `double`

`ReferenceGrayLab` — Reference values of gray ROIs
20-by-3 numeric matrix

Reference values of gray ROIs in the CIE 1976 L*a*b* color space, specified as a 20-by-3 numeric matrix. The three columns contain the L*, a*, and b* values of the gray patches, respectively. The rows contain the reference intensities of the 20 gray ROIs, in the same sequential order.

Note The `esfrChart` object includes default CIE 1976 L*a*b* values for the gray ROIs. However, the actual reference values can vary depending on several factors, such as print quality.

Data Types: `double`

`ReferenceColorLab` — Reference values of color ROIs
16-by-3 numeric matrix

Reference values of color ROIs in the CIE 1976 L*a*b* color space, specified as a 16-by-3 numeric matrix. The three columns contain the L*, a*, and b* values of the color patches, respectively. The rows contain the reference intensities of the 16 color ROIs, in the same sequential order.

Note The `esfrChart` object includes default CIE 1976 L*a*b* values for the color ROIs. However, the actual reference values can vary depending on several factors, such as print quality. Accurate reference color values result in more faithful color reproduction measurements.

Data Types: `double`

`Style` — Style of test chart
`'Enhanced'` | `'Extended'` | `'WedgeEnhanced'` | `'WedgeExtended'`

Style of test chart, specified as `'Enhanced'`, `'Extended'`, `'WedgeEnhanced'`, or `'WedgeExtended'`.

Object Functions

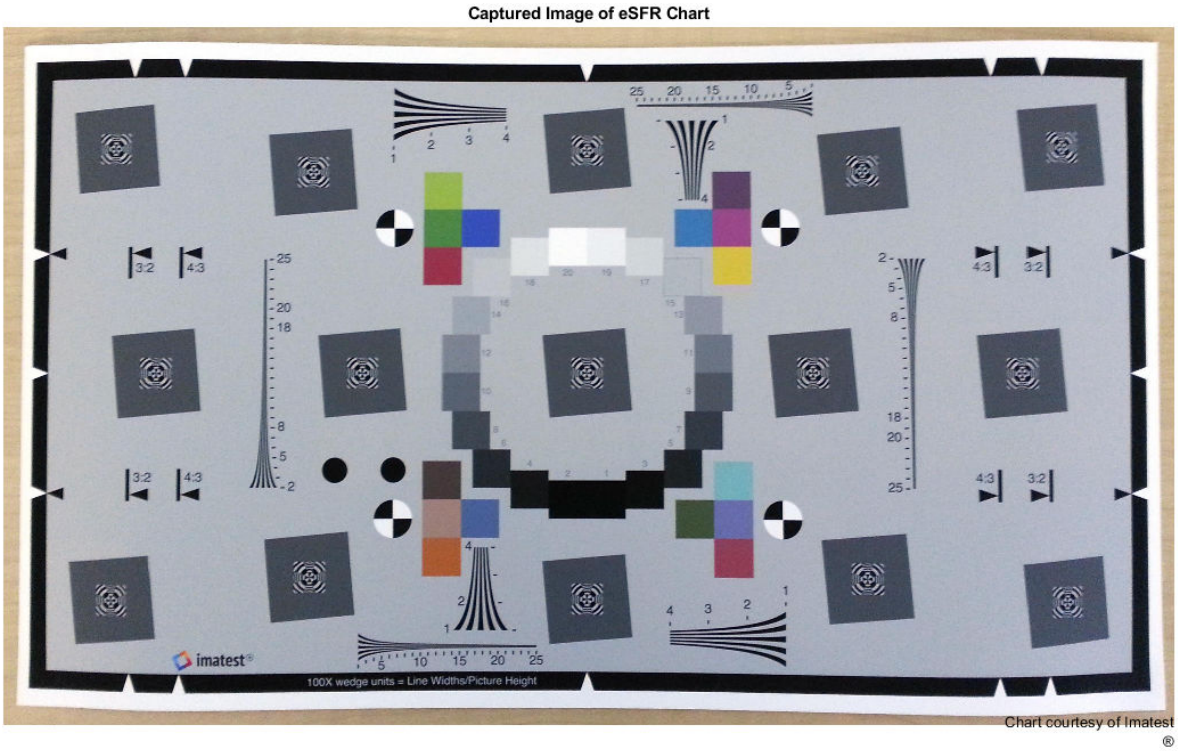
measureSharpness	Measure spatial frequency response using Imatest eSFR chart
measureChromaticAberration	Measure chromatic aberration at slanted edges using Imatest eSFR chart
measureNoise	Measure noise using Imatest eSFR chart
measureColor	Measure color reproduction using test chart
measureIlluminant	Measure scene illuminant using test chart
displayChart	Display test chart with overlaid regions of interest

Examples

Create eSFR Chart Object from Test Image

Read an image of an eSFR chart into the workspace. Display the image.

```
I = imread("eSFRTestImage.jpg");
imshow(I)
title("Captured Image of eSFR Chart")
text(size(I,2),size(I,1)+15,["Chart courtesy of Imatest",char(174)], ...
    "FontSize",10,"HorizontalAlignment","right");
```



Create an `esfrChart` object using the chart image.

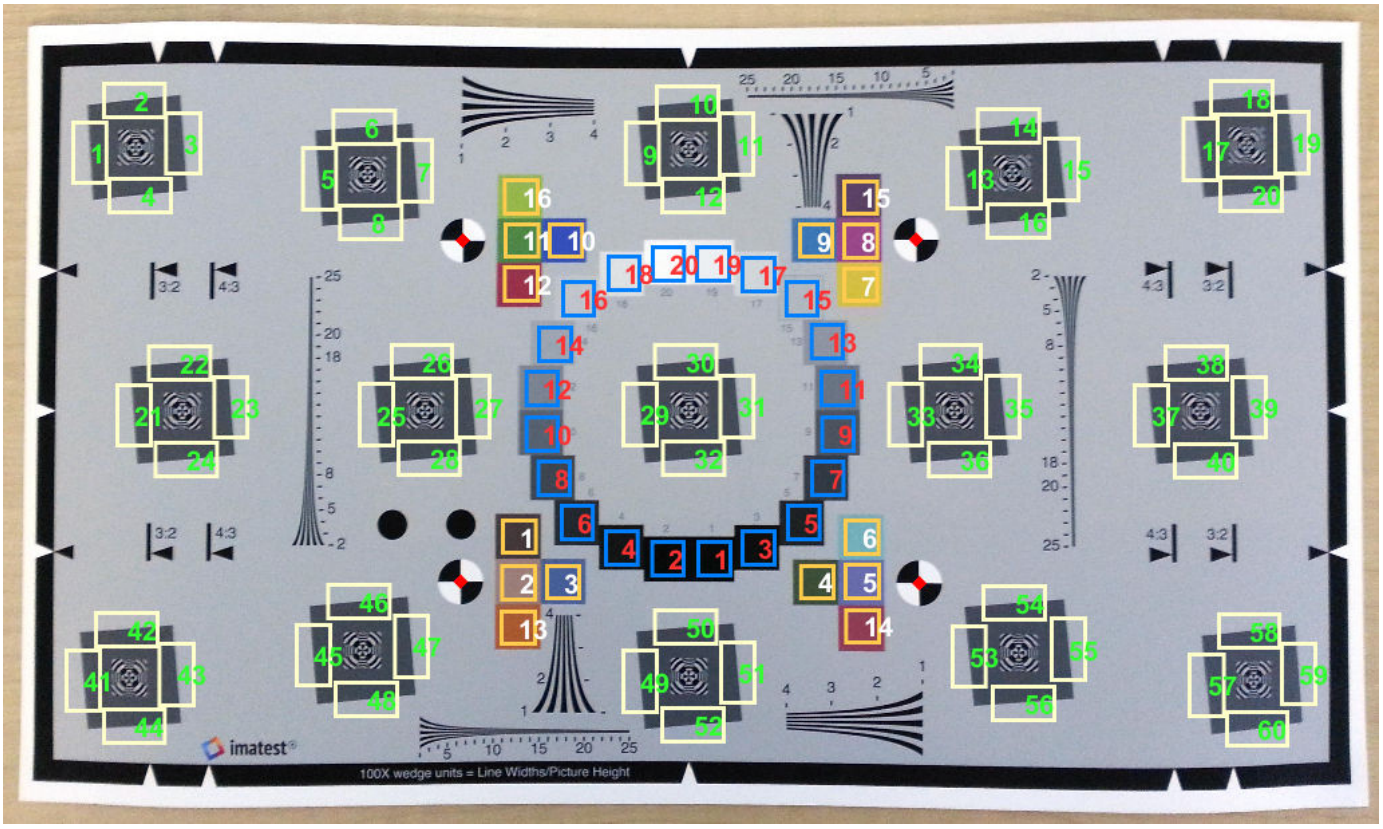
```
chart = esfrChart(I)
```

```
chart =
  esfrChart with properties:
```

```
                Image: [1836x3084x3 uint8]
    SlantedEdgeROIs: [60x1 struct]
            GrayROIs: [20x1 struct]
           ColorROIs: [16x1 struct]
  RegistrationPoints: [4x2 double]
               Style: 'Extended'
     ReferenceGrayLab: [20x3 double]
    ReferenceColorLab: [16x3 double]
```

Display the imported eSFR chart. Regions of interest (ROI) are highlighted and labeled.

```
displayChart(chart)
```



The chart is imported correctly. All 60 slanted edge ROIs (labeled with green numbers) are visible and centered on appropriate edges. 20 gray patch ROIs (labeled in red) and 16 color patch ROIs (labeled in white) are visible and are contained within the boundary of each patch.

Create eSFR Chart Object Using Specified Registration Points

Create an `esfrChart` object by specifying the coordinates of the four registration points. Registration points are located at the center of the black-and-white checkered circles.

Read an image of an eSFR chart into the workspace.

```
I = imread('eSFRTestImage.jpg');
```

Display the image and configure it to collect four registration points.

```
figure
imshow(I)
[X, Y] = ginput(4);
```

Click the registration points in this order: top-left, top-right, bottom-right, bottom-left.

Create an `esfrChart` object, specifying the four registration points. Display the imported eSFR chart. Regions of interest are highlighted and labeled. The registration points appear in red.

```
chart = esfrChart(I,'RegistrationPoints',[X, Y]);
displayChart(chart);
```

Tips

- For accurate and reliable results, acquire an image of the test chart according to standard specifications outlined in the ISO standard and by the manufacturer [2], [3]. As a simple guideline, align the chart horizontally on a light background. Cover over 90% of the field of view with the chart, but ensure that the top and bottom edges of the chart are still visible. For reliable measurements, set the minimum image width to at least 500 pixels.
- You can capture an image of the Extended eSFR test chart at the full 16:9 aspect ratio, or at an aspect ratio of 3:2 or 4:3, as specified on the chart.
- To ensure that the chart is properly imported, visually verify the test chart image using the `displayChart` function.

Compatibility Considerations

`esfrChart` object now refines slanted edge ROI positions

Behavior changed in R2021a

Starting in R2021a, by default the `esfrChart` object refines the position of slanted edge ROIs based on localized image content. In previous releases, the `esfrChart` object did not refine the position of the slanted edge ROIs after the initial estimate of the position based on the position of the registration points. The refinement results in more precise ROI positions.

To replicate the previous behavior, specify the '`RefinePoints`' name-value argument as `false` when creating an `esfrChart` object.

References

[1] Imatest. "Esfr". https://www.imatest.com/mathworks/esfr/.

[2] *Using eSFR ISO Part 1*. URL: https://www.imatest.com/docs/esfriso_instructions.

[3] ISO 12233:2014. "Photography – Electronic still picture imaging – Resolution and spatial frequency responses." *International Organization for Standardization; ISO/TC 42 Photography*. URL: https://www.iso.org/standard/59419.html.

See Also

`displayColorPatch` | `plotSFR` | `plotChromaticity`

Topics

“Anatomy of Imatest Extended eSFR Chart”
“Evaluate Quality Metrics on eSFR Test Chart”

Introduced in R2017b

fan2para

Convert fan-beam projections to parallel-beam

Syntax

```
P = fan2para(F,D)
P = fan2para(F,D,Name,Value)
[P,paraSensorPos,paraRotAngles] = fan2para( ___ )
```

Description

`P = fan2para(F,D)` converts the fan-beam data `F` to the parallel-beam data `P`. Each column of `F` contains the fan-beam data at one rotation angle. `D` is the distance from the fan-beam vertex to the center of rotation.

`P = fan2para(F,D,Name,Value)` uses name-value arguments to control aspects of the data conversion.

`[P,paraSensorPos,paraRotAngles] = fan2para( ___ )` returns the parallel-beam sensor locations in `paraSensorPos` and rotation angles in `paraRotAngles`.

Examples

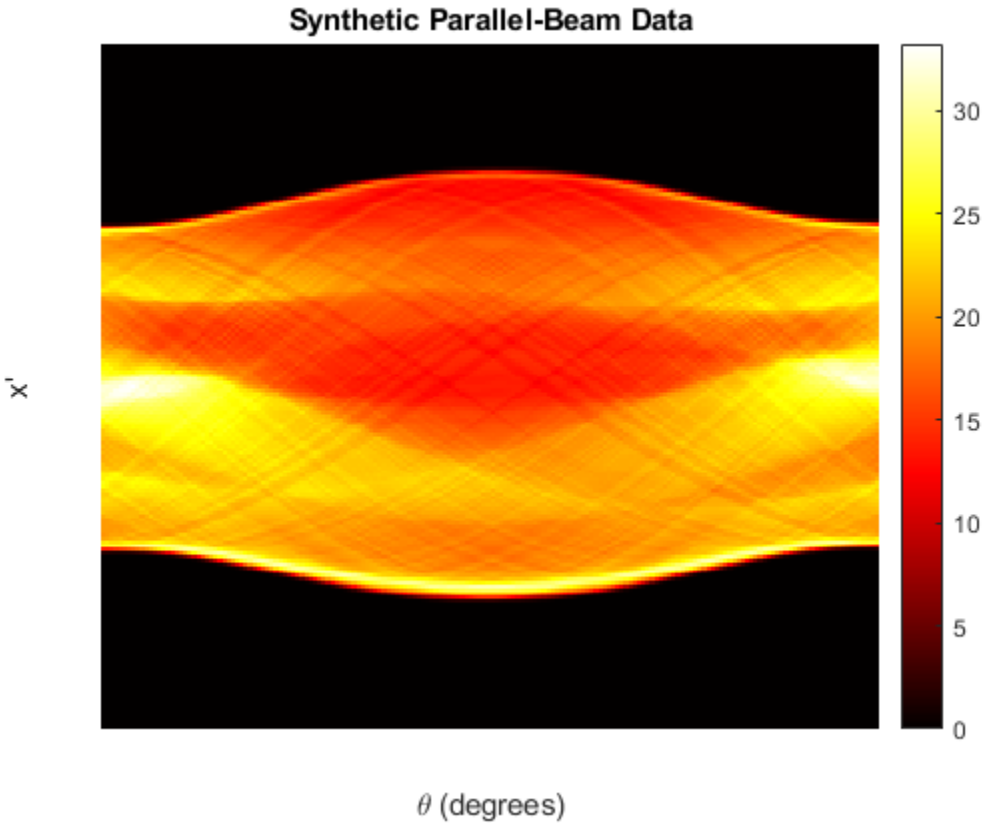
Recover Parallel-beam Data from Fan-beam Data

Create synthetic parallel-beam data.

```
ph = phantom(128);
```

Calculate the parallel beam transform and display it.

```
theta = 0:179;
[Psynthetic,xp] = radon(ph,theta);
imshow(Psynthetic,[],...
               'XData',theta,'YData',xp,'InitialMagnification','fit')
axis normal
title('Synthetic Parallel-Beam Data')
xlabel('\theta (degrees)')
ylabel('x''')
colormap(gca,hot), colorbar
```

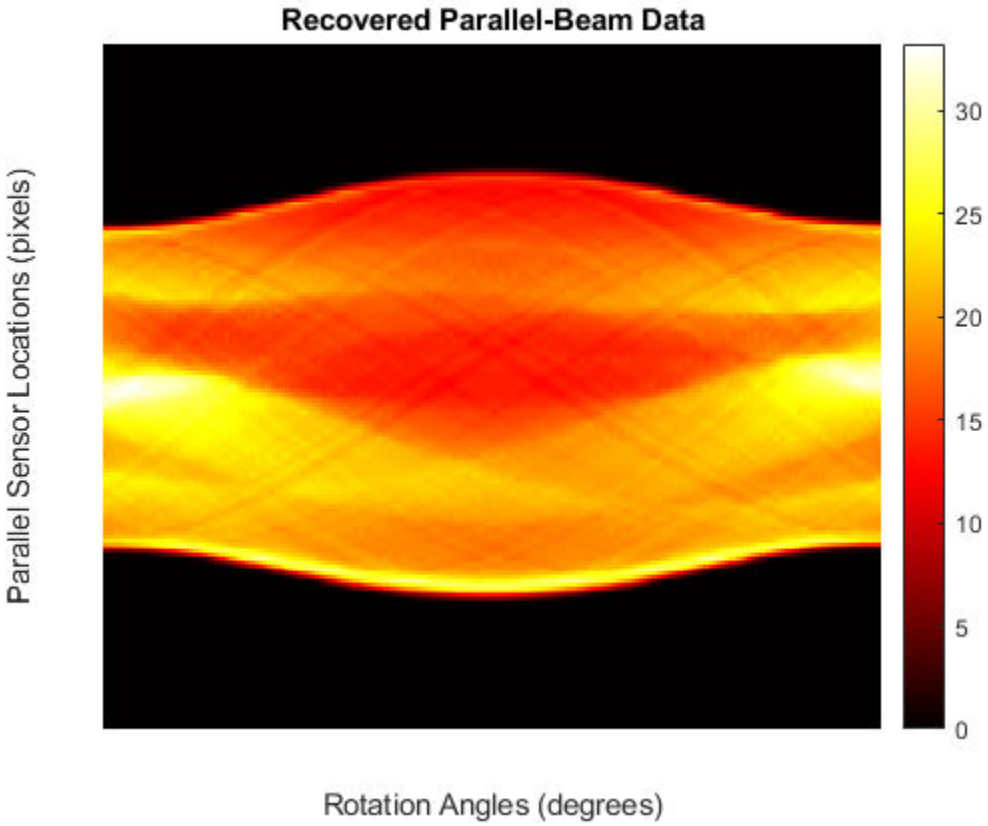


Convert the parallel-beam data to fan-beam.

```
Fsynthetic = para2fan(Psynthetic,100,'FanSensorSpacing',1);
```

Recover original parallel-beam data.

```
[Precovered,Ploc,Pangles] = fan2para(Fsynthetic,100,...
                                     'FanSensorSpacing',1,...
                                     'ParallelSensorSpacing',1);
figure
imshow(Precovered,[],...
       'XData',Pangles,'YData',Ploc,'InitialMagnification','fit')
axis normal
title('Recovered Parallel-Beam Data')
xlabel('Rotation Angles (degrees)')
ylabel('Parallel Sensor Locations (pixels)')
colormap(gca,hot), colorbar
```



Input Arguments

F — Fan-beam projection data

numeric matrix

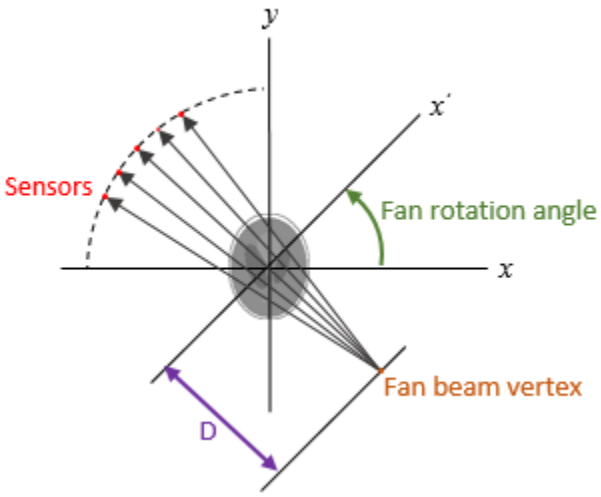
Fan-beam projection data, specified as a numeric matrix. Each column of `F` contains the fan-beam data at one rotation angle. The number of columns indicates the number of fan-beam rotation angles and the number of rows indicates the number of fan-beam sensors.

Data Types: `double` | `single`

D — Distance from fan beam vertex to center of rotation

positive number

Distance in pixels from the fan beam vertex to the center of rotation, specified as a positive number. `fan2para` assumes that the center of rotation is the center point of the projections, which is defined as `ceil(size(F,1)/2)`. The figure illustrates `D` in relation to the fan-beam vertex for one fan-beam projection.



Data Types: `double` | `single`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `P = fan2para(F,D,FanRotationIncrement=5)` specifies a fan rotation increment of 5 degrees.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `P = fan2para(F,D,"FanRotationIncrement",5)` specifies a fan rotation increment of 5 degrees.

`FanCoverage` — Range of fan-beam rotation

`"cycle"` (default) | `"minimal"`

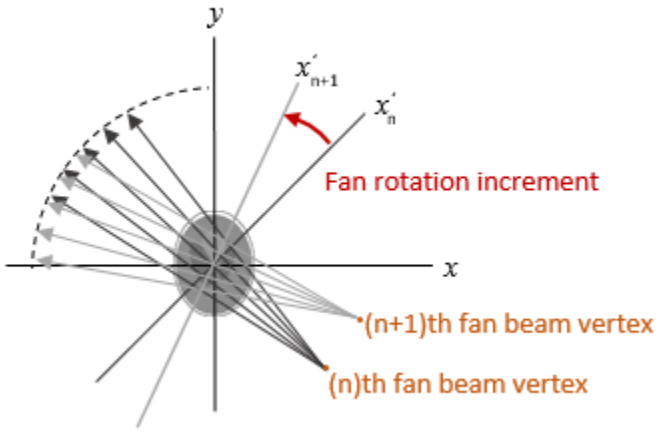
Range of fan-beam rotation, specified as `"cycle"` or `"minimal"`.

- `"cycle"` — Rotate through the full range [0, 360) degrees.
- `"minimal"` — Rotate through the minimum range necessary to represent the object.

`FanRotationIncrement` — Fan-beam rotation angle increment

1 (default) | positive scalar

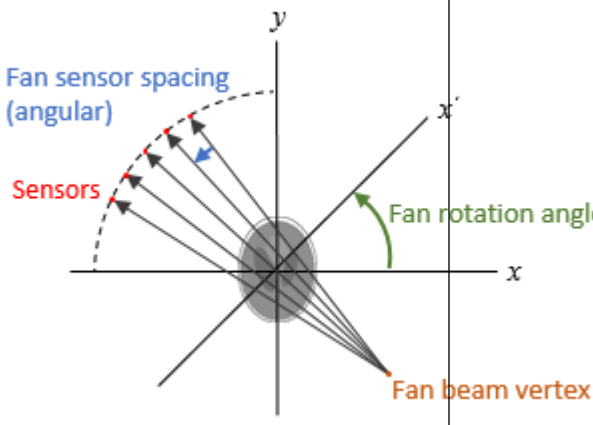
Fan-beam rotation angle increment in degrees, specified as a positive scalar.

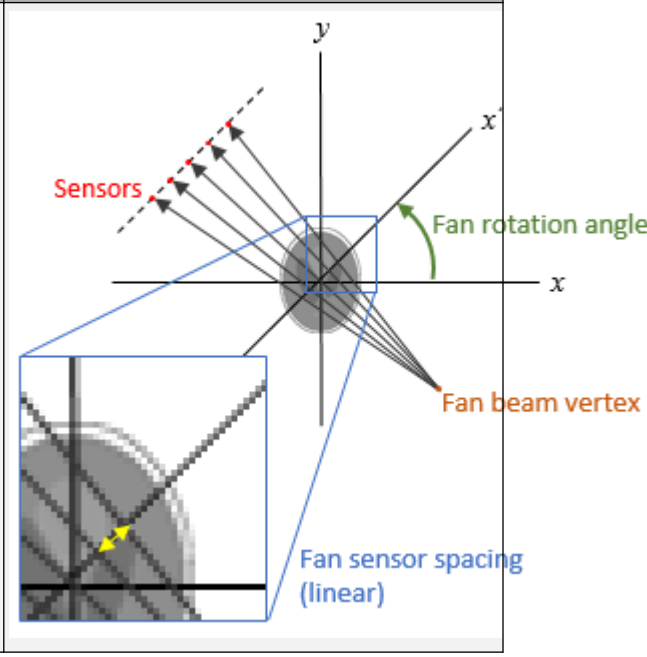


Data Types: `double`

`FanSensorGeometry` — Fan-beam sensor positioning
`"arc"` (default) | `"line"`

Fan-beam sensor positioning, specified as `"arc"` or `"line"`.

Value	Meaning	Diagram
`"arc"`	Sensors are spaced at equal angles along a circular arc. The center of the arc is the fan-beam vertex. `FanSensorSpacing` defines the angular spacing in degrees.	

Value	Meaning	Diagram
`"line"`	Sensors are spaced at equal distances along a line that is parallel to the x' axis. The closest sensor is distance `D` from the center of rotation. `FanSensorSpacing` defines the distance between fan-beams on the x' axis, in pixels.	

`FanSensorSpacing` — Fan-bean sensor spacing
1 (default) | positive scalar

Fan-bean sensor spacing, specified as a positive scalar.

- If `FanSensorGeometry` is `"arc"`, then `FanSensorSpacing` defines the angular spacing in degrees.
- If `FanSensorGeometry` is `"line"`, then `FanSensorSpacing` defines the linear distance between fan-beams, in pixels. Linear spacing is measured on the x' axis.

Data Types: `double`

`Interpolation` — Type of interpolation
`"Linear"` (default) | `"nearest"` | `"spline"` | `"pchip"`

Type of interpolation used between the parallel-beam and fan-beam data, specified as one of these values.

`"nearest"` — Nearest-neighbor

`"linear"` — Linear (the default)

`"spline"` — Piecewise cubic spline

`"pchip"` — Piecewise cubic Hermite (PCHIP)

`ParallelCoverage` — Range of parallel-beam rotation
`"halfcycle"` (default) | `"cycle"`

Range of parallel-beam rotation, specified as `"halfcycle"` or `"cycle"`.

- `"cycle"` — Parallel data covers the full range of [0, 360) degrees.
- `"halfcycle"` — Parallel data covers [0, 180) degrees.

`ParallelRotationIncrement` — Parallel-beam rotation angle increment
positive scalar

Parallel-beam rotation angle increment in degrees, specified as a positive scalar k such that `180/`k is an integer. If you do not specify `ParallelRotationIncrement`, then the default value is equal to `FanRotationIncrement`.

Data Types: `double`

`ParallelSensorSpacing` — Parallel-beam sensor spacing
positive scalar

Parallel-beam sensor spacing in pixels, specified as a positive scalar. The range of parallel-beam sensor locations is computed from the range of fan angles, *fanAngles*, according to: `[D*sin(min(`*`fanAngles`*`)) D*sin(max(`*`fanAngles`*`))]`.

If you do not specify `ParallelSensorSpacing`, then the spacing is assumed to be uniform and is set to the minimum spacing implied by the fan angles and sampled over the range implied by the fan angles.

Data Types: `double`

Output Arguments

`P` — Parallel-beam projection data
numeric matrix

Parallel-beam projection data, returned as a numeric matrix. Each column of `P` contains the parallel-beam data at one rotation angle. The number of columns indicates the total number of parallel-beam rotation angles and is equal to the length of `paraRotAngles`. The number of rows indicates the total number of parallel-beam sensors and is equal to the length of `paraSensorPos`.

Data Types: `double`

`paraSensorPos` — Parallel-beam sensor locations
numeric column vector

Parallel-beam sensor locations, returned as a numeric column vector.

Data Types: `double`

`paraRotAngles` — Parallel-beam rotation angles
numeric row vector

Parallel-beam rotation angles, returned as a numeric row vector.

Data Types: `double`

See Also

`fanbeam` | `ifanbeam` | `iradon` | `para2fan` | `phantom` | `radon`

Introduced before R2006a

fanbeam

Fan-beam transform

Syntax

```
F = fanbeam(I,D)
F = fanbeam(I,D,Name,Value)
[F,fanSensorPos,fanRotAngles] = fanbeam( ___ )
```

Description

`F = fanbeam(I,D)` calculates the fan-beam projection data (sinogram) `F` from the image `I`. Each column of `F` contains fan-beam projection data at one rotation angle. `D` is the distance from the fan-beam vertex to the center of rotation.

`F = fanbeam(I,D,Name,Value)` uses name-value arguments to specify the rotation increment and sensor spacing.

`[F,fanSensorPos,fanRotAngles] = fanbeam( ___ )` also returns the location of fan-beam sensors in `fanSensorPos` and the rotation angles where the fan-beam projections are calculated in `fanRotAngles`.

Examples

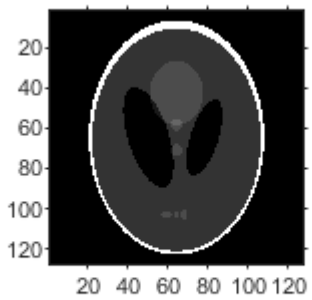
Compute Fan-beam Projections for Rotation Angles Over Entire Image

Set the IPT preference to make the axes visible.

```
iptsetpref('ImshowAxesVisible','on')
```

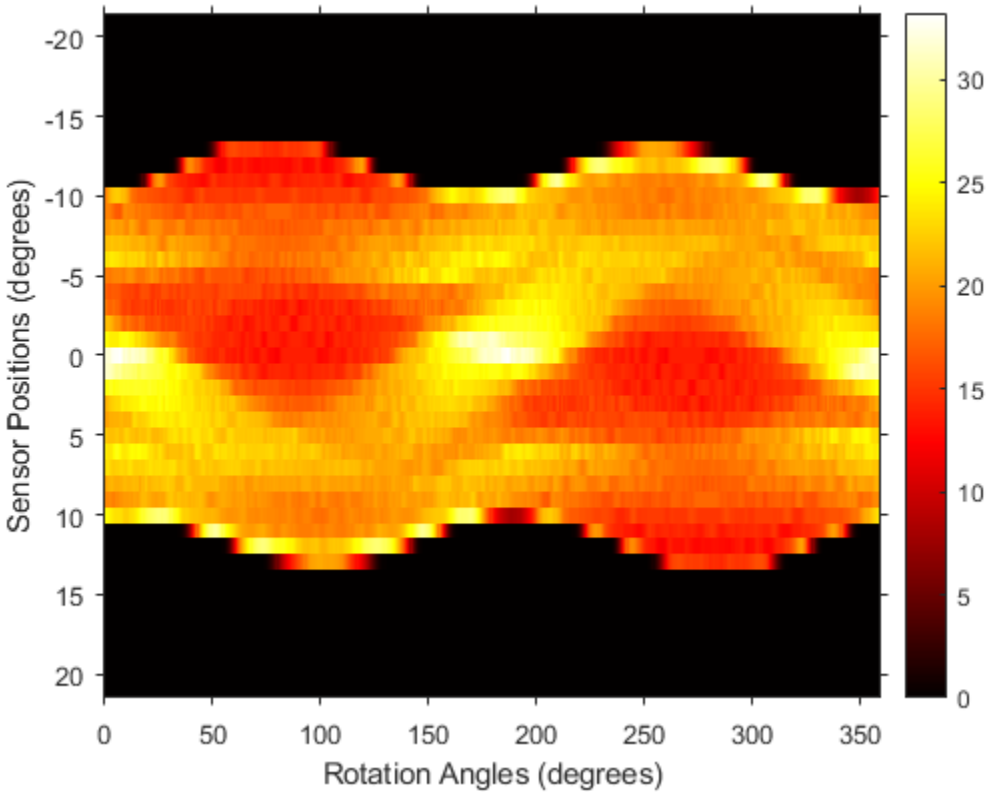
Create a sample image and display it.

```
ph = phantom(128);
imshow(ph)
```



Calculate the fanbeam projections and display them.

```
[F,Fpos,Fangles] = fanbeam(ph,250);
figure
imshow(F,[],'XData',Fangles,'YData',Fpos,...
            'InitialMagnification','fit')
axis normal
xlabel('Rotation Angles (degrees)')
ylabel('Sensor Positions (degrees)')
colormap(gca,hot), colorbar
```



Compute Radon and Fan-beam Projections and Compare Results

Compute fan-beam projections for 'arc' geometry.

```
I = ones(100);
D = 200;
dtheta = 45;
[Farc,FposArcDeg,Fangles] = fanbeam(I,D,...
                    'FanSensorGeometry','arc',...
                  'FanRotationIncrement',dtheta);
```

Convert angular positions to linear distance along x-prime axis.

```
FposArc = D*tan(FposArcDeg*pi/180);
```

Compute fan-beam projections for 'line' geometry.

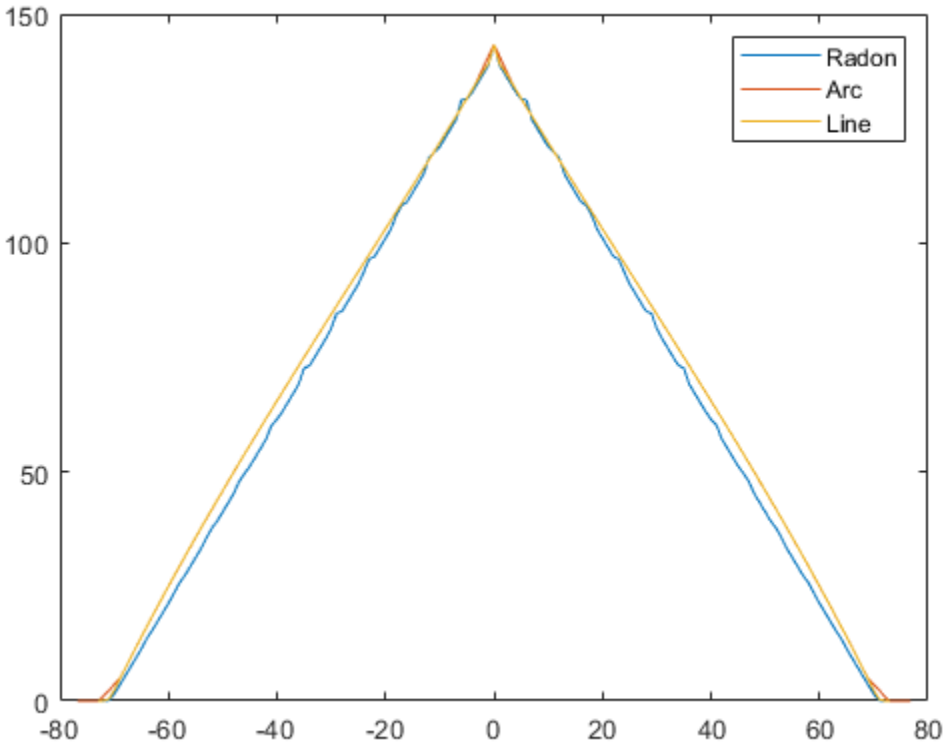
```
[Fline,FposLine] = fanbeam(I,D,...
            'FanSensorGeometry','line',...
         'FanRotationIncrement',dtheta);
```

Compute the corresponding Radon transform.

```
[R,Rpos]=radon(I,Fangles);
```

Display the three projections at one particular rotation angle. Note the three are very similar. Differences are due to the geometry of the sampling, and the numerical approximations used in the calculations.

```
figure
idx = find(Fangles==45);
plot(Rpos,R(:,idx),...
           FposArc,Farc(:,idx),...
          FposLine,Fline(:,idx))
legend('Radon','Arc','Line')
```



Input Arguments

I — Input image

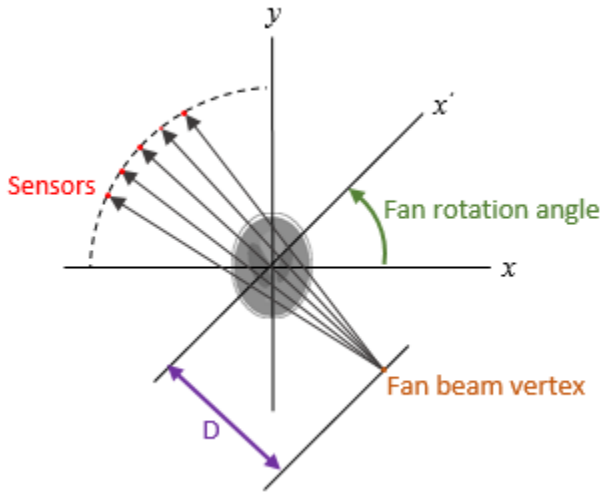
2-D numeric matrix | 2-D logical matrix

Input image, specified as a 2-D numeric matrix or 2-D logical matrix.

`D` — Distance from fan beam vertex to center of rotation

positive number

Distance in pixels from the fan beam vertex to the center of rotation, specified as a positive number. The center of rotation is the center pixel of the image, defined as `floor((size(I)+1)/2)`. `D` must be large enough to ensure that the fan-beam vertex is outside of the image at all rotation angles. See “Tips” on page 1-996 for guidelines on specifying `D`.



Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `fanbeam(I,D,FanRotationIncrement=5)`

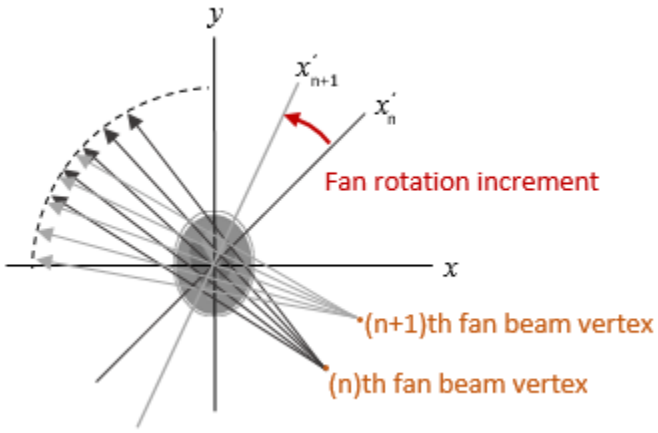
Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `fanbeam(I,D,"FanRotationIncrement",5)`

`FanRotationIncrement` — Fan-beam rotation angle increment

1 (default) | positive scalar

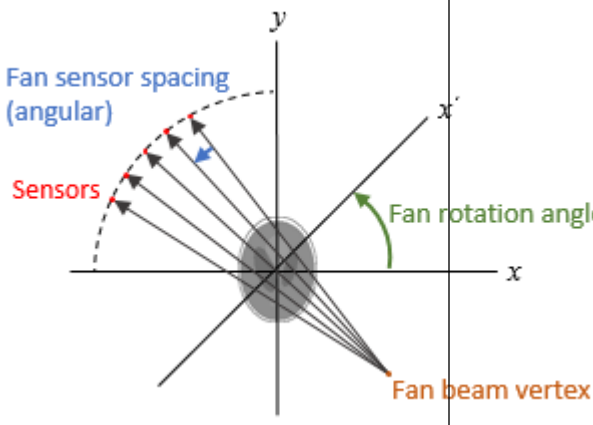
Fan-beam rotation angle increment in degrees, specified as a positive scalar.

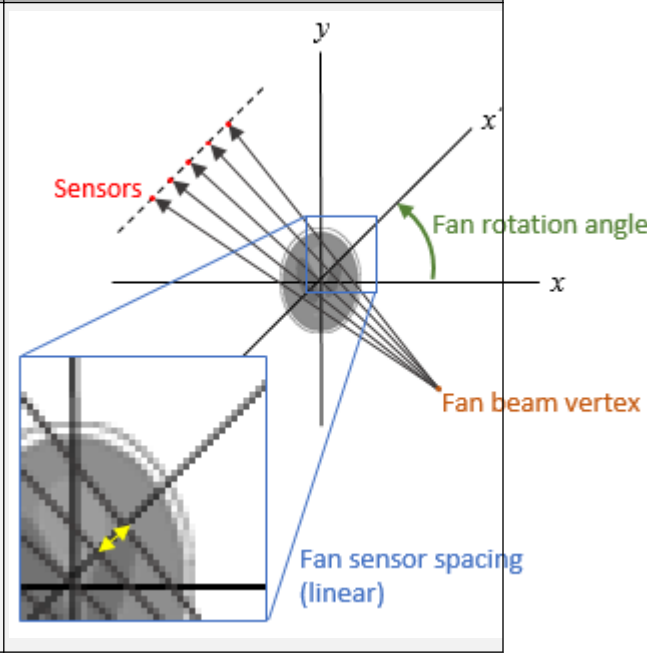


Data Types: `double`

`FanSensorGeometry` — Fan-beam sensor positioning
`"arc"` (default) | `"line"`

Fan-beam sensor positioning, specified as `"arc"` or `"line"`.

Value	Meaning	Diagram
`"arc"`	Sensors are spaced at equal angles along a circular arc. The center of the arc is the fan-beam vertex. `FanSensorSpacing` defines the angular spacing in degrees.	

Value	Meaning	Diagram
`"line"`	Sensors are spaced at equal distances along a line that is parallel to the x' axis. The closest sensor is distance `D` from the center of rotation. `FanSensorSpacing` defines the distance between fan-beams on the x' axis, in pixels.	

`FanSensorSpacing` — Fan-bean sensor spacing
1 (default) | positive scalar

Fan-bean sensor spacing, specified as a positive scalar.

- If `FanSensorGeometry` is `"arc"`, then `FanSensorSpacing` defines the angular spacing in degrees.
- If `FanSensorGeometry` is `"line"`, then `FanSensorSpacing` defines the linear distance between fan-beams, in pixels. Linear spacing is measured on the x' axis.

Data Types: `double`

Output Arguments

F — Fan-beam projection data
numSensors-by-*numAngles* numeric matrix

Fan-beam projection data, returned as a *numSensors*-by-*numAngles* numeric matrix. *numSensors* is the number of fan-beam sensors and *numAngles* is the number of fan-beam rotation angles. Each column of `F` contains the fan-beam sensor samples at one rotation angle.

`fanbeam` determines *numAngles* by calculating how many rotation angles are required to span 360 degrees using an angular spacing of `FanRotationIncrement`.

`fanbeam` determines *numSensors* by calculating how many beams are required to cover the entire image for any rotation angle. Fewer sensors are required to cover the image when the distance `D` between the fan-beam vertex and the center of rotation is large.

Data Types: `double`

`fanSensorPos` — Location of fan-beam sensors
numSensors-by-1 numeric vector

Location of fan-beam sensors, returned as a *numSensors*-by-1 numeric vector.

- If `FanSensorGeometry` is `"arc"` (the default), then `fanSensorPos` contains the fan-beam spread angles.
- If `FanSensorGeometry` is `"line"`, then `fanSensorPos` contains the fan-beam sensor positions along the *x*' axis. See `FanSensorSpacing` for more information.

Data Types: `double`

`fanRotAngles` — Rotation angle of fan-beam sensors
1-by-*numAngles* numeric vector

Rotation angle of fan-beam sensors, returned as a 1-by-*numAngles* numeric vector.

Data Types: `double`

Tips

As a guideline, try making `D` a few pixels larger than half the image diagonal dimension, calculated as follows.

```
sqrt(size(I,1)^2 + size(I,2)^2)
```

The values returned in `F` are a numerical approximation of the fan-beam projections. The algorithm depends on the Radon transform, interpolated to the fan-beam geometry. The results vary depending on the parameters used. You can expect more accurate results when the image is larger, `D` is larger, and for points closer to the middle of the image, away from the edges.

References

[1] Kak, Avinash C., and Malcolm Slaney. *Principles of Computerized Tomographic Imaging*. New York: IEEE Press, 1988.. pp. 92-93.

See Also

`fan2para` | `ifanbeam` | `iradon` | `para2fan` | `phantom` | `radon`

Introduced before R2006a

fibermetric

Enhance elongated or tubular structures in image using Frangi vesselness filter

Syntax

```
J = fibermetric(I)
J = fibermetric(I,thickness)
J = fibermetric( ___ ,Name,Value)
```

Description

`J = fibermetric(I)` enhances elongated or tubular structures in the 2-D or 3-D grayscale image `I` using a Hessian-based multiscale Frangi vesselness filter. The image returned, `J`, contains the maximum response of the filter at a thickness that approximately matches the size of the tubular structure in the image.

`J = fibermetric(I,thickness)` specifies the thickness of the tubular structures to enhance.

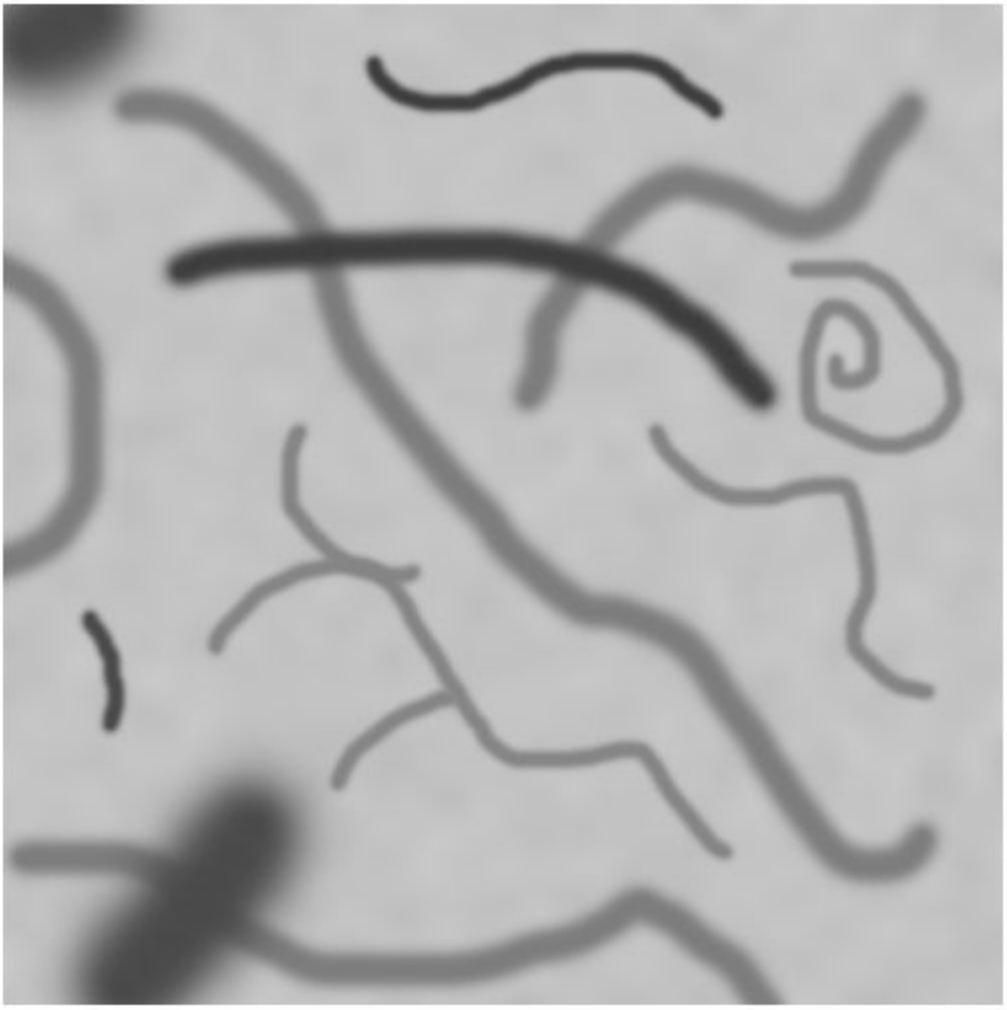
`J = fibermetric( ___ ,Name,Value)` uses name-value pair arguments to control different aspects of the filtering algorithm.

Examples

Find Threads Approximately Seven Pixels Thick

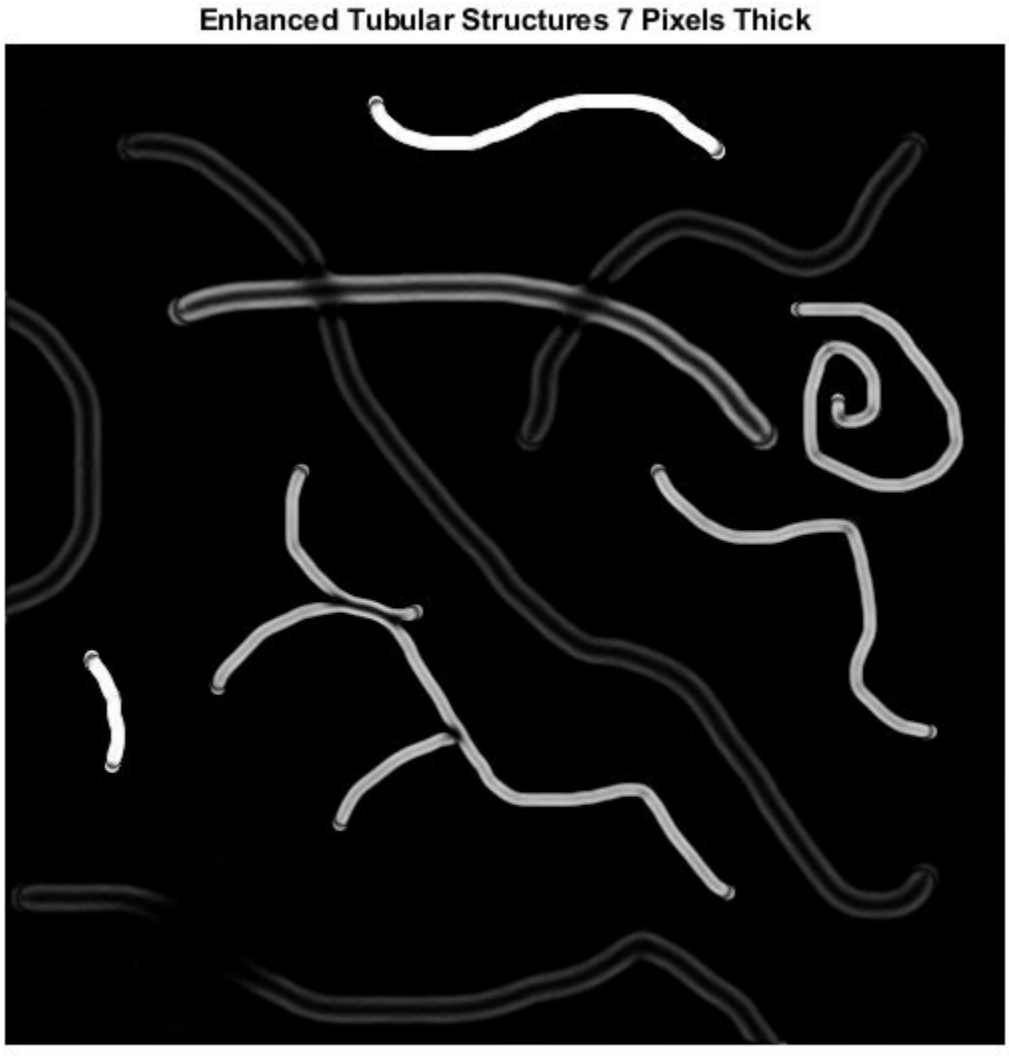
Read and display an image that contains tubular threads of varying thicknesses.

```
I = imread('threads.png');
imshow(I)
```



Create an enhanced version of the image that highlights threads that are seven pixels thick. Threads show up dark against a light background, therefore specify the object polarity as `'dark'`. Display the enhanced image.

```
B = fibermetric(I,7,'ObjectPolarity','dark');
imshow(B)
title('Enhanced Tubular Structures 7 Pixels Thick')
```

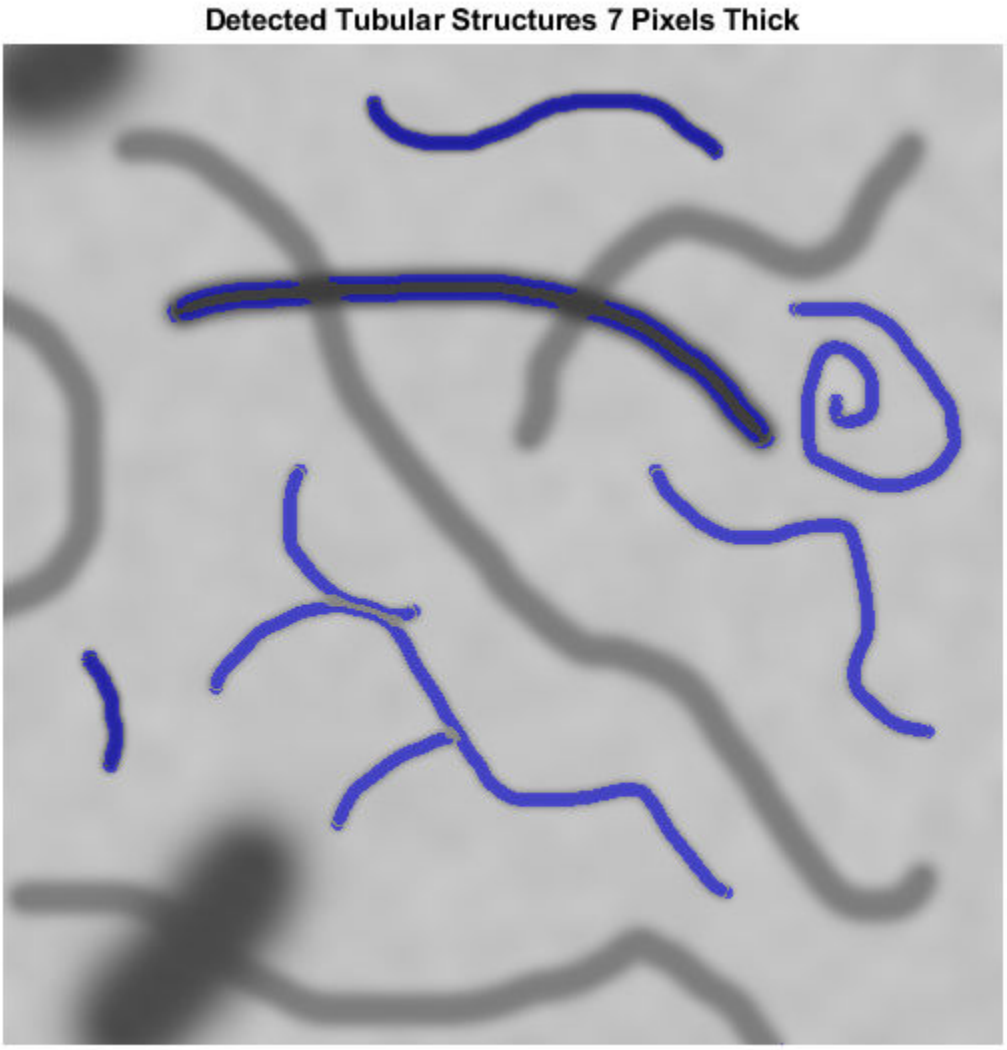


Threshold the enhanced image to create a binary mask image containing the threads with the specified thickness.

```
BW = imbinarize(B);
```

Display the mask over the original image by using the `labeloverlay` function. The overlay has a blue tint where the mask is `true` (where threads have the specified thickness).

```
imshow(labeloverlay(I,BW))
title('Detected Tubular Structures 7 Pixels Thick')
```



Input Arguments

`I` — Image with elongated or tubular structures

2-D grayscale image | 3-D grayscale volume

Image with elongated or tubular structures, specified as 2-D grayscale image or 3-D grayscale volume.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`thickness` — Thickness of tubular structures

`[4 6 8 10 12 14]` (default) | positive integer | vector of positive integers

Thickness of tubular structures in pixels, specified as a positive integer or vector of positive integers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `J = fibermetric(I,'StructureSensitivity',15)`

`StructureSensitivity` — Structure sensitivity
positive number

Structure sensitivity, specified as the comma-separated pair consisting of `'StructureSensitivity'` and a positive number. The structure sensitivity is a threshold for differentiating the tubular structure from the background.

The default value depends on the data type of image `I`, and is calculated as `0.01*diff(getrangefromclass(I))`. For example, the default threshold is `2.55` for images of data type `uint8`, and the default is `0.01` for images of data type `double` with pixel values in the range [0, 1].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ObjectPolarity` — Polarity of tubular structures
`'bright'` (default) | `'dark'`

Polarity of the tubular structures with the background, specified as the comma-separated pair consisting of `'ObjectPolarity'` and one of the following values:

Value	Description
`'bright'`	Structure is brighter than the background.
`'dark'`	Structure is darker than the background.

Data Types: `char` | `string`

Output Arguments

`J` — Enhanced image
numeric array

Enhanced image, returned as a numeric array of the same size as the input image `I`. If the data type of `I` is `double`, then the data type of `J` is also `double`. Otherwise, the data type of `J` is `single`.

Data Types: `single` | `double`

Tips

- The `fibermetric` function does not perform segmentation. The function enhances an image to highlight structures and is typically used as a preprocessing step for segmentation.

Compatibility Considerations

`fibermetric` calculates new default structural sensitivity
Behavior changed in R2018b

Starting in R2018b, `fibermetric` calculates the default value of the `StructureSensitivity` argument as `0.01*diff(getrangefromclass(I))`. The change in the default value increases computational efficiency and reduces memory usage. These improvements also enable processing 3-D volumetric images.

Previous versions of `fibermetric` defined the default value of `StructureSensitivity` as half of the maximum of the Hessian norm of the image. If you want to reproduce the prior default value for 2-D images, then specify `StructureSensitivity` as `0.5*maxhessiannorm(I)`. The `maxhessiannorm` function does not support 3-D input.

References

[1] Frangi, Alejandro F., et al. *Multiscale vessel enhancement filtering*. Medical Image Computing and Computer-Assisted Intervention — MICCAI'98. Springer Berlin Heidelberg, 1998. pp. 130–137.

See Also

`edge` | `imgradient`

Introduced in R2017a

findbounds

Find output bounds for spatial transformation

Syntax

```
outbounds = findbounds(tform,inbounds)
```

Description

`outbounds = findbounds(tform,inbounds)` estimates the output bounds corresponding to a given spatial transformation and a set of input bounds. `tform` is a spatial transformation structure. `inbounds` is a 2-by-`num_dims` matrix that specifies the lower and upper bounds of the output image. `outbounds` is an estimate of the smallest rectangular region completely containing the transformed rectangle represented by the input bounds, and has the same form as `inbounds`. Since `outbounds` is only an estimate, it might not completely contain the transformed input rectangle.

Examples

Calculate Boundaries of Transformed Output Image

Read an image into the workspace, and display the image.

```
I = imread('cameraman.tif');
figure
imshow(I)
```

Create a spatial transformation structure that stretches an image.

```
T = maketform('affine',[.5 0 0; .5 2 0; 0 0 1]);
```

Calculate the boundaries of the output image, given the size of the input image and the spatial transformation. The dimensions of the input image are 256-by-256. The boundaries of the output image reflect the transformation: 256-by-512.

```
outb = findbounds(T,[0 0;256 256])
```

```
outb = 2×2

     0     0
   256   512
```

Apply the transformation, and display the image.

```
transformedI = imtransform(I,T);
figure
imshow(transformedI)
```

Input Arguments

`tform` — Spatial transformation
structure

Spatial transformation, specified as a structure (`tform`).

Data Types: `struct`

`inbounds` — Bounds for each dimension of the input image
2-by-`num_dims` matrix

Bounds for each dimension of the input image, specified as a 2-by-`num_dims` matrix. The first row of `inbounds` specifies the lower bounds for each dimension, and the second row specifies the upper bounds. `num_dims` has to be consistent with the `ndims_in` field of `tform`.

Example: `outb = findbounds(T,[0 0;256 256])` where input image is 256-by-256.

Data Types: `double`

Output Arguments

`outbounds` — Bounds for each dimension of the output image

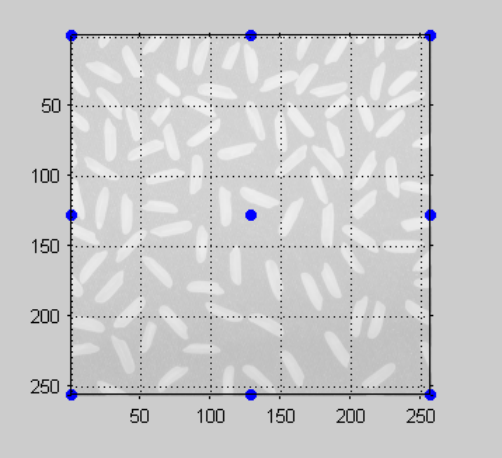
2-by-`num_dims` matrix of class `double`

Bounds for each dimension of the output image (output space bounding box), returned as a 2-by-`num_dims` matrix of class `double`.

Algorithms

1 `findbounds` first creates a grid of input-space points. These points are at the center, corners, and middle of each edge in the image.

```
I = imread('rice.png');
h = imshow(I);
set(h,'AlphaData',0.3);
axis on, grid on
in_points = [ ...
    0.5000    0.5000
    0.5000  256.5000
  256.5000    0.5000
  256.5000  256.5000
    0.5000  128.5000
  128.5000    0.5000
  128.5000  128.5000
  128.5000  256.5000
  256.5000  128.5000];
hold on
plot(in_points(:,1),in_points(:,2),'.','MarkerSize',18)
hold off
```



Grid of Input-Space Points

2 Next, `findbounds` transforms the grid of input-space points to output space. If `tform` contains a forward transformation (a nonempty `forward_fcn` field), then `findbounds` transforms the input-space points using `tformfwd`. For example:

```
tform = maketform('affine', ...
    [1.1067 -0.2341 0; 0.5872 1.1769 0; 1000 -300 1]);
out_points = tformfwd(tform, in_points)
```

```
  out_points =

   1.0e+03 *

    1.0008   -0.2995
    1.1512    0.0018
    1.2842   -0.3595
    1.4345   -0.0582
    1.0760   -0.1489
    1.1425   -0.3295
    1.2177   -0.1789
    1.2928   -0.0282
```

If `tform` does not contain a forward transformation, then `findbounds` estimates the output bounds using the Nelder-Mead optimization function `fminsearch`.

3 Finally, `findbounds` computes the bounding box of the transformed grid of points.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

`tformarray` | `tformfwd` | `tforminv`

Introduced before R2006a

fitbrisque

Fit custom model for BRISQUE image quality score

Syntax

```
model = fitbrisque(imds,opinionScores)
```

Description

`model = fitbrisque(imds,opinionScores)` creates a Blind/Referenceless Image Spatial Quality Evaluator (BRISQUE) model from a reference image datastore, `imds`, with corresponding human perceptual differential mean opinion score (DMOS) values, `opinionScore`.

Note To use the `fitbrisque` function, you must have Statistics and Machine Learning Toolbox™.

Examples

Calculate BRISQUE Score Using Custom Feature Model

Train a custom BRISQUE model from a set of quality-aware features and corresponding human opinion scores. Use the custom model to calculate a BRISQUE score for an image of a natural scene.

Save images from an image datastore. These images all have compression artifacts resulting from JPEG compression.

```
setDir = fullfile(toolboxdir('images'),'imdata');
imds = imageDatastore(setDir,'FileExtensions',{'.jpg'});
```

Specify the opinion score for each image. The following differential mean opinion score (DMOS) values are for illustrative purposes only. They are not real DMOS values obtained through experimentation.

```
opinionScores = 100*rand(1,size(imds.Files,1));
```

Create the custom model of quality-aware features using the image datastore and the opinion scores. Because the scores are random, the property values will vary.

```
model = fitbrisque(imds,opinionScores')
```

```
Extracting features from 38 images.
......
Completed 15 of 38 images.  Time: Calculating...
.....Training support vector regressor...

Done.

model =
  brisqueModel with properties:
```

```
          Alpha: [35x1 double]
           Bias: 58.1250
SupportVectors: [35x36 double]
         Kernel: 'gaussian'
          Scale: 0.2767
```

Read an image of a natural scene that has the same type of distortion as the training images. Display the image.

```
I = imread('car1.jpg');
imshow(I)
```

Calculate the BRISQUE score for the image using the custom model. Display the score.

```
brisqueI = brisque(I,model);
fprintf('BRISQUE score for the image is %0.4f.\n',brisqueI)
```

```
BRISQUE score for the image is 72.7492.
```

Input Arguments

`imds` — Reference image datastore

`ImageDatastore` object

Reference image datastore, specified as an `ImageDatastore` object. The datastore must contain 2-D grayscale or 2-D RGB images of data type `single`, `double`, `int16`, `uint8`, or `uint16`. The images must have a known set of distortions such as compression artifacts, blurring, or noise.

`opinionScores` — Human opinion scores
numeric vector

Human opinion scores, specified as a numeric vector with values in the range [0, 100]. Each element in `opinionScores` is the human perceptual DMOS value corresponding to an image in the datastore `imds`. The length of `opinionScores` is equal to the number of images in `imds`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output Arguments

`model` — Custom model of image features
`brisqueModel` object

Custom model of image features, returned as a `brisqueModel` object. `model` contains a support vector regressor (SVR) with a Gaussian kernel trained to predict the BRISQUE quality score.

References

[1] Mittal, A., A. K. Moorthy, and A. C. Bovik. "No-Reference Image Quality Assessment in the Spatial Domain." *IEEE Transactions on Image Processing*. Vol. 21, Number 12, December 2012, pp. 4695–4708.

[2] Mittal, A., A. K. Moorthy, and A. C. Bovik. "Referenceless Image Spatial Quality Evaluation Engine." Presentation at the 45th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, November 2011.

See Also

Functions
`brisque` | `niqe` | `fitniqe`

Objects
`brisqueModel`

Topics
“Image Quality Metrics”
“Train and Use No-Reference Quality Assessment Model”

Introduced in R2017b

fitgeotrans

Fit geometric transformation to control point pairs

Syntax

```
tform = fitgeotrans(movingPoints,fixedPoints,transformationType)
tform = fitgeotrans(movingPoints,fixedPoints,'polynomial',degree)
tform = fitgeotrans(movingPoints,fixedPoints,'pwl')
tform = fitgeotrans(movingPoints,fixedPoints,'lwm',n)
```

Description

`tform = fitgeotrans(movingPoints,fixedPoints,transformationType)` takes the pairs of control points, `movingPoints` and `fixedPoints`, and uses them to infer the geometric transformation specified by `transformationType`.

`tform = fitgeotrans(movingPoints,fixedPoints,'polynomial',degree)` fits a `PolynomialTransformation2D` object to control point pairs `movingPoints` and `fixedPoints`. Specify the degree of the polynomial transformation `degree`, which can be 2, 3, or 4.

`tform = fitgeotrans(movingPoints,fixedPoints,'pwl')` fits a `PiecewiseLinearTransformation2D` object to control point pairs `movingPoints` and `fixedPoints`. This transformation maps control points by breaking up the plane into local piecewise-linear regions. A different affine transformation maps control points in each local region.

`tform = fitgeotrans(movingPoints,fixedPoints,'lwm',n)` fits a `LocalWeightedMeanTransformation2D` object to control point pairs `movingPoints` and `fixedPoints`. The local weighted mean transformation creates a mapping, by inferring a polynomial at each control point using neighboring control points. The mapping at any location depends on a weighted average of these polynomials. The `n` closest points are used to infer a second degree polynomial transformation for each control point pair.

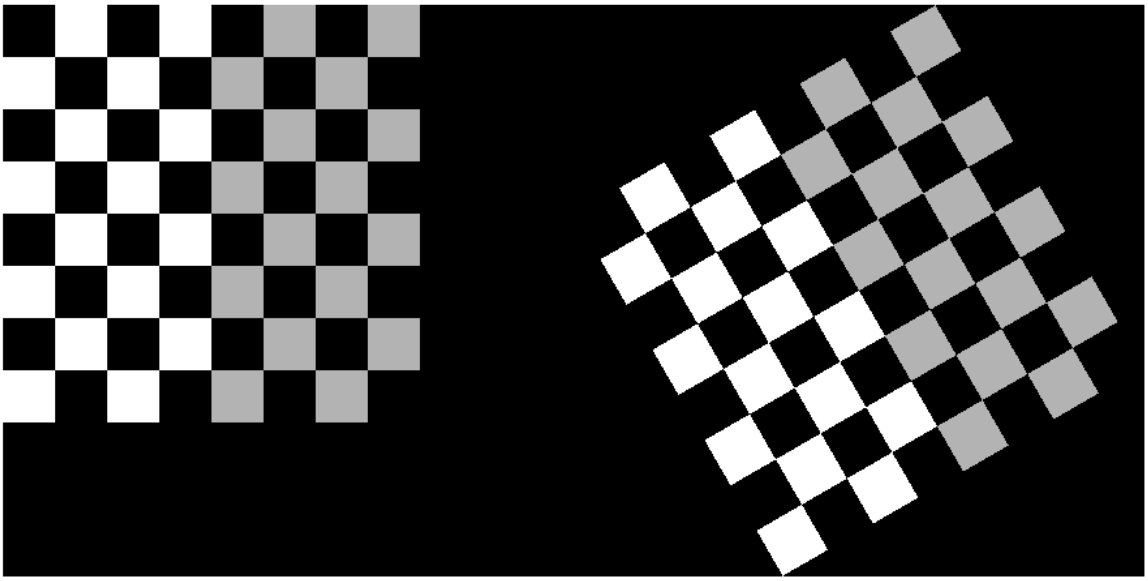
Examples

Create Geometric Transformation for Image Alignment

This example shows how to create a geometric transformation that can be used to align two images.

Create a checkerboard image and rotate it to create a misaligned image.

```
I = checkerboard(40);
J = imrotate(I,30);
imshowpair(I,J,'montage')
```



Define some matching control points on the fixed image (the checkerboard) and moving image (the rotated checkerboard). You can define points interactively using the Control Point Selection tool.

```
fixedPoints = [41 41; 281 161];
movingPoints = [56 175; 324 160];
```

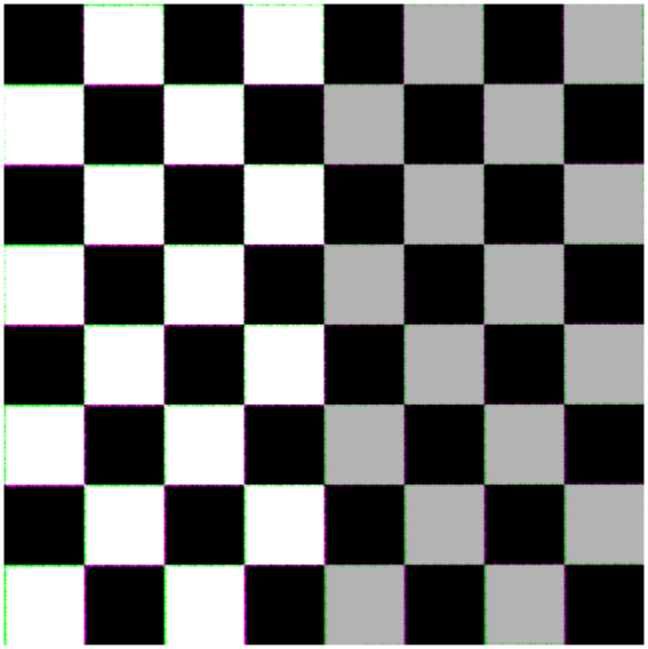
Create a geometric transformation that can be used to align the two images, returned as an `affine2d` geometric transformation object.

```
tform = fitgeotrans(movingPoints,fixedPoints,'NonreflectiveSimilarity')
```

```
tform =
  affine2d with properties:

                T: [3x3 double]
    Dimensionality: 2
```

Use the `tform` estimate to resample the rotated image to register it with the fixed image. The regions of color (green and magenta) in the false color overlay image indicate error in the registration. This error comes from a lack of precise correspondence in the control points.

```
Jregistered = imwarp(J,tform,'OutputView',imref2d(size(I)));
figure
imshowpair(I,Jregistered)
```



Recover angle and scale of the transformation by checking how a unit vector parallel to the x-axis is rotated and stretched.

```
u = [0 1];
v = [0 0];
[x, y] = transformPointsForward(tform, u, v);
dx = x(2) - x(1);
dy = y(2) - y(1);
angle = (180/pi) * atan2(dy, dx)
```

```
angle = 29.7686
```

```
scale = 1 / sqrt(dx^2 + dy^2)
```

```
scale = 1.0003
```

Input Arguments

`movingPoints` — *x*- and *y*-coordinates of control points in image to transform

m-by-2 matrix

x- and *y*-coordinates of control points in the image you want to transform, specified as an *m*-by-2 matrix.

Example: `movingPoints = [11 11; 41 71];`

Data Types: `double` | `single`

`fixedPoints` — *x*- and *y*-coordinates of control points in fixed image
m-by-2 matrix

x- and *y*- coordinates of control points in the fixed image, specified as an *m*-by-2 matrix.

Example: `fixedPoints = [14 44; 70 81];`

Data Types: `double` | `single`

`transformationType` — Type of transformation
`'nonreflectivesimilarity'` | `'similarity'` | `'affine'` | `'projective'`

Type of transformation, specified as one of the following: `'nonreflectivesimilarity'`, `'similarity'`, `'affine'`, or `'projective'`. For more information, see “Transformation Types” on page 1-1014.

Data Types: `char` | `string`

`degree` — Degree of the polynomial
2 | 3 | 4

Degree of the polynomial, specified as the integer 2, 3, or 4.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`n` — Number of points to use in local weighted mean calculation
positive integer

Number of points to use in local weighted mean calculation, specified as a positive integer. `n` can be as small as 6, but making `n` small risks generating ill-conditioned polynomials.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

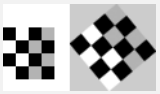
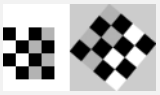
`tform` — Transformation
transformation object

Transformation, returned as a transformation object. The type of object depends on the transformation type. For example, if you specify the transformation type `'affine'`, then `tform` is an `affine2d` object. If you specify `'pwl'`, then `tform` is an `image.geotrans.PiecewiseLinearTransformation2d` object.

More About

Transformation Types

The table lists all the transformation types supported by `fitgeotrans` in order of complexity.

Transformation Type	Description	Minimum Number of Control Point Pairs	Example
`'nonreflectivesimilarity'`	Use this transformation when shapes in the moving image are unchanged, but the image is distorted by some combination of translation, rotation, and scaling. Straight lines remain straight, and parallel lines are still parallel.	2	
`'similarity'`	Same as `'nonreflectivesimilarity'` with the addition of optional reflection.	3	
`'affine'`	Use this transformation when shapes in the moving image exhibit shearing. Straight lines remain straight, and parallel lines remain parallel, but rectangles become parallelograms.	3	
`'projective'`	Use this transformation when the scene appears tilted. Straight lines remain straight, but parallel lines converge toward a vanishing point.	4	
`'polynomial'`	Use this transformation when objects in the image are curved. The higher the order of the polynomial, the better the fit, but the result can contain more curves than the fixed image.	6 (order 2) 10 (order 3) 15 (order 4)	
`'pwl'`	Use this transformation (piecewise linear) when parts of the image appear distorted differently.	4	
`'lwm'`	Use this transformation (local weighted mean) when the distortion varies locally and piecewise linear is not sufficient.	6 (12 recommended)	

References

[1] Goshtasby, Ardeshir, "Piecewise linear mapping functions for image registration," *Pattern Recognition*, Vol. 19, 1986, pp. 459-466.

[2] Goshtasby, Ardeshir, "Image registration by local approximation methods," *Image and Vision Computing*, Vol. 6, 1988, pp. 255-261.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `fitgeotrans` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

- When generating code, the `transformationType` argument must be a compile-time constant and only the following transformation types are supported: `'nonreflectivesimilarity'`, `'similarity'`, `'affine'`, and`'projective'`.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, the `transformationType` argument must be a compile-time constant and only the following transformation types are supported: `'nonreflectivesimilarity'`, `'similarity'`, `'affine'`, and `'projective'`.

Thread-Based Environment
Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Functions
`imwarp` | `cpselect`

Objects
`affine2d` | `projective2d` | `LocalWeightedMeanTransformation2D` | `PiecewiseLinearTransformation2D` | `PolynomialTransformation2D`

Topics
“Register Images with Projection Distortion Using Control Points”
“Control Point Selection Procedure”
“Matrix Representation of Geometric Transformations”
“Approaches to Registering Images”

Introduced in R2013b

fitniqe

Fit custom model for NIQE image quality score

Syntax

```
model = fitniqe(imds)
model = fitniqe(imds,Name,Value)
```

Description

`model = fitniqe(imds)` creates a Naturalness Image Quality Evaluator (NIQE) model from reference image datastore `imds`.

`model = fitniqe(imds,Name,Value)` creates a NIQE model using additional parameters to control the model calculation.

Examples

Calculate NIQE Score Using Custom Feature Model

Load a set of natural images into an image datastore. These images are shipped in Image Processing Toolbox™ in a directory named 'imdata'.

```
setDir = fullfile(toolboxdir('images'),'imdata');
imds = imageDatastore(setDir,'FileExtensions',{'.jpg'});
```

Train a custom NIQE model using the image datastore.

```
model = fitniqe(imds);
```

```
Extracting features from 38 images.
....
Completed 12 of 38 images.  Time: Calculating...
....
Completed 24 of 38 images.  Time: 00:21 of 00:32
...
Done.
```

Read an image of a natural scene. Display the image.

```
I = imread('car1.jpg');
imshow(I)
```

Calculate the NIQE score for the image using the custom model. Display the score.

```
niqeI = niqe(I,model);
fprintf('NIQE score for the image is %0.4f.\n',niqeI)
```

```
NIQE score for the image is 1.8728.
```

Fit Custom NIQE Model Using Specified Block Size

Load a set of natural images into an image datastore. These images are shipped in Image Processing Toolbox™ in a directory named 'imdata'.

```
setDir = fullfile(toolboxdir('images'),'imdata');
imds = imageDatastore(setDir,'FileExtensions',{'.jpg'});
```

Create the custom model of NSS features using the image datastore. Specify a block size and use the default sharpness threshold.

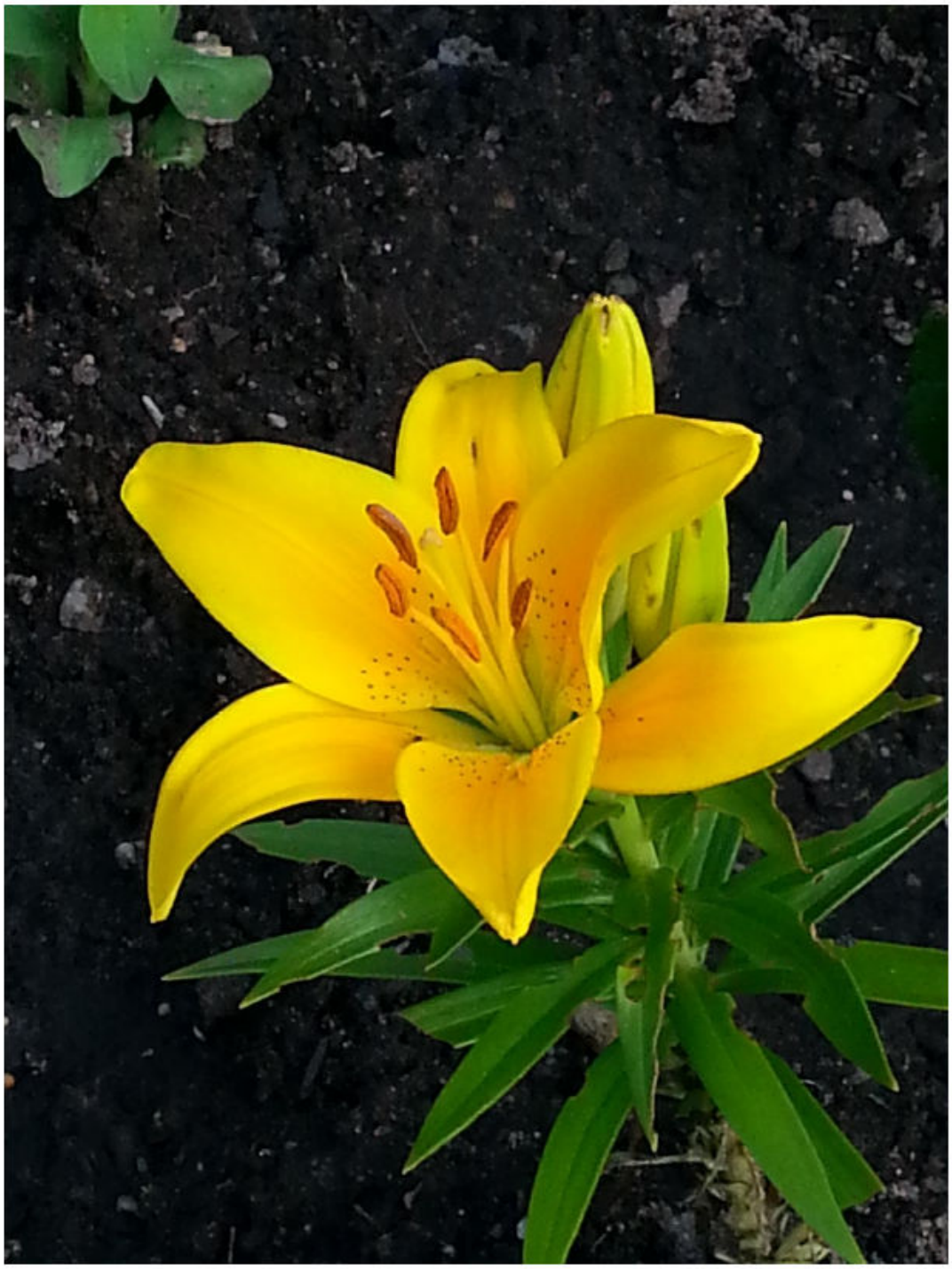
```
model = fitniqe(imds,'BlockSize',[48 96])
```

```
Extracting features from 38 images.
...
Completed 9 of 38 images.  Time: Calculating...
....
```

```
Completed 15 of 38 images.  Time: 00:22 of 01:10
....
Completed 32 of 38 images.  Time: 00:33 of 00:39
.
Done.

model =
  niqeModel with properties:

                 Mean: [1.9291 0.5854 0.6735 0.0667 0.0517 0.0889 ... ]
           Covariance: [36x36 double]
            BlockSize: [48 96]
    SharpnessThreshold: 0
```

Read a natural image into the workspace. Display the image.

```
I = imread('yellowlily.jpg');
imshow(I)
```



Calculate the NIQE score for the image using the custom model. Display the score.

```
niqeI = niqe(I,model);
fprintf('NIQE score for the image is %0.4f.\n',niqeI)
```

```
NIQE score for the image is 2.9554.
```

Input Arguments

`imds` — Reference image datastore

`ImageDatastore` object

Reference image datastore, specified as an `ImageDatastore` object. The datastore must contain 2-D grayscale or 2-D RGB images of data type `single`, `double`, `int16`, `uint8`, or `uint16`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `model = fitniqe(imds,'BlockSize',[48 36])` fits a NIQE model using 48-by-36 pixel blocks.

`BlockSize` — Block size used to partition the images

`[96 96]` (default) | 2-element row vector of positive even integers

Block size used to partition the images, specified as the comma-separated pair consisting of `'BlockSize'` and a 2-element row vector of positive even integers. Blocks are nonoverlapping. Natural scene statistics, which are calculated from the blocks, define the output `model`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`SharpnessThreshold` — Sharpness threshold

`0` (default) | numeric scalar in the range [0, 1]

Sharpness threshold, specified as the comma-separated pair consisting of `'SharpnessThreshold'` and a numeric scalar in the range [0, 1]. The sharpness threshold, `s`, controls which image blocks are used to compute the model. `fitniqe` computes the model using all blocks that have sharpness more than `s` times the maximum sharpness among all blocks.

Data Types: `single` | `double`

Output Arguments

`model` — Custom model of image features

`niqeModel` object

Custom model of image features, returned as a `niqeModel` object.

Tips

- The custom dataset specified in the image datastore `imds` should consist of images that are perceptually pristine to human subjects. However, the definition of pristine depends on the

application. For example, a pristine set of microscopy images has a different set of quality criteria than images of buildings or outdoor scenes. When training a custom NIQE model, use images with varied image content and with potentially different sets of quality criteria.

References

[1] Mittal, A., R. Soundararajan, and A. C. Bovik. "Making a Completely Blind Image Quality Analyzer." *IEEE Signal Processing Letters*. Vol. 22, Number 3, March 2013, pp. 209–212.

See Also

Functions
`brisque` | `fitbrisque` | `niqe`

Objects
`niqeModel`

Topics
“Image Quality Metrics”
“Train and Use No-Reference Quality Assessment Model”

Introduced in R2017b

fliptform

Flip input and output roles of spatial transformation structure

Syntax

```
tflip = fliptform(T)
```

Description

`tflip = fliptform(T)` creates a new `TFORM` spatial transformation structure by flipping the roles of the inputs and outputs in an existing `TFORM` structure.

Examples

Flip Spatial Transformation Structure

Create a spatial transformation structure.

```
T = maketform('affine', [.5 0 0; .5 2 0; 0 0 1])
```

```
T =

  struct with fields:

       ndims_in: 2
      ndims_out: 2
    forward_fcn: @fwd_affine
    inverse_fcn: @inv_affine
          tdata: [1×1 struct]
```

Create a new spatial transformation structure by flipping the roles of the inputs and outputs.

```
T2 = fliptform(T)
```

```
T2 =

  struct with fields:

       ndims_in: 2
      ndims_out: 2
    forward_fcn: @inv_affine
    inverse_fcn: @fwd_affine
          tdata: [1×1 struct]
```

After flipping the spatial transformation structures, the following statements are equivalent.

```
x = tformfwd([-3 7],T)
x = tforminv([-3 7],T2)
```

```
x =

     2    14
```

```
x =

     2    14
```

Input Arguments

T — Spatial transformation

TFORM spatial transformation structure

Spatial transformation, specified as a TFORM spatial transformation structure.

Data Types: struct

Output Arguments

tflip — Flipped spatial transformation

TFORM spatial transformation structure

Flipped spatial transformation, returned as a TFORM spatial transformation structure.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® backgroundPool or accelerate code with Parallel Computing Toolbox™ ThreadPool.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

maketform | tformfwd | tforminv

Introduced before R2006a

freqz2

2-D frequency response

Syntax

```
[H,f1,f2] = freqz2(h)
[H,f1,f2] = freqz2(h,[n1 n2])
[H,f1,f2] = freqz2(h,f1, f2)
[___] = freqz2(h,___,[dx dy])
freqz2(___)
```

Description

`[H,f1,f2] = freqz2(h)` returns `H`, the `64`-by-`64` frequency response of `h`, and the frequency vectors `f1` (of length `64`) and `f2` (of length `64`). `h` is a two-dimensional FIR filter, in the form of a computational molecule.

`freqz2` returns `f1` and `f2` as normalized frequencies in the range -1.0 to 1.0, where 1.0 corresponds to half the sampling frequency, or π radians.

`[H,f1,f2] = freqz2(h,[n1 n2])` returns `H`, the `n2`-by-`n1` frequency response of `h`, and the frequency vectors `f1` (of length `n1`) and `f2` (of length `n2`). You can also specify `[n1 n2]` as two separate arguments, `n1,n2`.

`[H,f1,f2] = freqz2(h,f1, f2)` returns the frequency response for the FIR filter `h` at frequency values in `f1` and `f2`. These frequency values must be in the range -1.0 to 1.0, where 1.0 corresponds to half the sampling frequency, or π radians. You can also specify `[f1 f2]` as two separate arguments, `f1, f2`.

`[___] = freqz2(h,___,[dx dy])` uses `[dx dy]` to override the intersample spacing in `h`. You can also specify a scalar to specify the same spacing in both the *x* and *y* dimensions.

`freqz2(___)` produces a mesh plot of the two-dimensional magnitude frequency response when no output arguments are specified.

Examples

View Frequency Response of Filter

This example shows how to create a two-dimensional filter using `fwind1` and how to view the filter's frequency response using `freqz2`.

Create an ideal frequency response.

```
Hd = zeros(16,16);
Hd(5:12,5:12) = 1;
Hd(7:10,7:10) = 0;
```

Create a 1-D window. This example uses a Bartlett window of length 16.

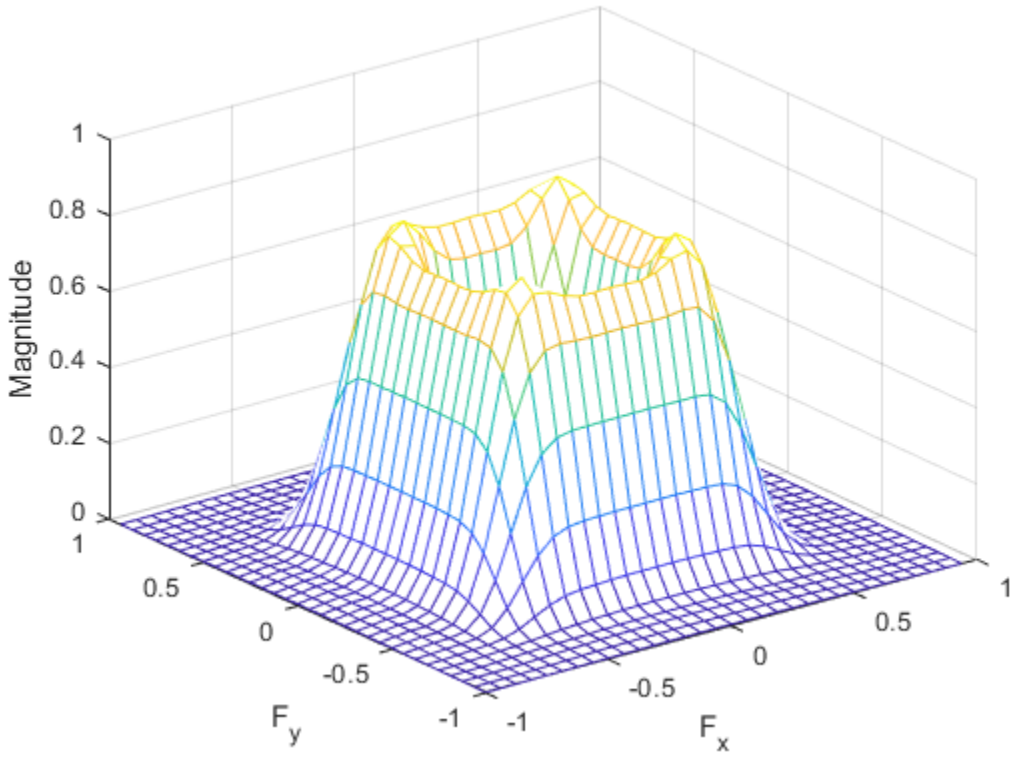
```
w = [0:2:16 16:-2:0]/16;
```

Create the 16-by-16 filter using `fwind1` and the 1-D window. This filter gives the closest match to the ideal frequency response.

```
h = fwind1(Hd,w);
```

Display the actual frequency response of the filter.

```
colormap(parula(64))
freqz2(h,[32 32]);
axis ([-1 1 -1 1 0 1])
```



Input Arguments

h — 2-D FIR filter

computational molecule

2-D FIR filter, specified in the form of a computational molecule.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

[n1 n2] — Number of points in the frequency response

`[64 64]` (default) | two-element vector

Number of points in the frequency response, specified as a two-element vector.

Data Types: `double`

`f1, f2` — Frequency vectors
numeric vectors

Frequency vectors, specified as numeric vectors.

Data Types: `double`

`[dx dy]` — Sample spacing
0.5 (default) | two-element vector or scalar

Sample spacing, specified as a two-element vector of the form `[dx dy]`. The default spacing is 0.5, which corresponds to a sampling frequency of 2.0. `dx` determines the spacing for the *x* dimension and `dy` determines the spacing for the *y* dimension. If you specify a scalar, `freqz2` uses the value to determine the intersample spacing in both dimensions.

Data Types: `double`

Output Arguments

`H` — Frequency response
numeric array

Frequency response, returned as a numeric array.

`f1` — Frequency vector
vector

Frequency vector, returned as a numeric vector.

Data Types: `double`

`f2` — Frequency vector
vector

Frequency vector, returned as a numeric vector.

See Also
`freqz`

Topics
“Design Linear Filters in the Frequency Domain”

Introduced before R2006a

fsamp2

2-D FIR filter using frequency sampling

Syntax

```
h = fsamp2(Hd)
h = fsamp2(f1,f2,Hd,[m n])
```

Description

`h = fsamp2(Hd)` designs a two-dimensional FIR filter with frequency response `Hd`, and returns the filter coefficients in matrix `h`. The filter `h` has a frequency response that passes through points in `Hd`. `fsamp2` designs two-dimensional FIR filters based on a desired two-dimensional frequency response sampled at points on the Cartesian plane.

`h = fsamp2(f1,f2,Hd,[m n])` produces an `m`-by-`n` FIR filter by matching the filter response at the points in the vectors `f1` and `f2`. The frequency vectors `f1` and `f2` are in normalized frequency, where 1.0 corresponds to half the sampling frequency, or π radians. The resulting filter fits the desired response as closely as possible in the least squares sense. For best results, there must be at least `m*n` desired frequency points. `fsamp2` issues a warning if you specify fewer than `m*n` points.

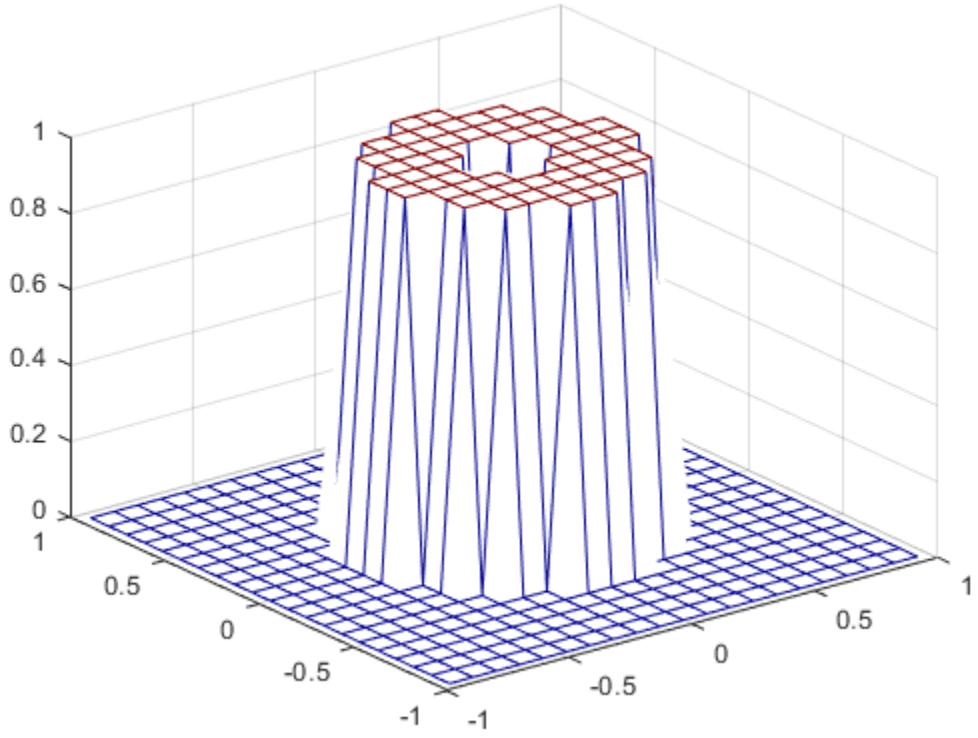
Examples

Create 2-D FIR Filter using Frequency Sampling

Use `fsamp2` to design an approximately symmetric, two-dimensional bandpass filter with passband between 0.1 and 0.5 (normalized frequency, where 1.0 corresponds to half the sampling frequency, or π radians).

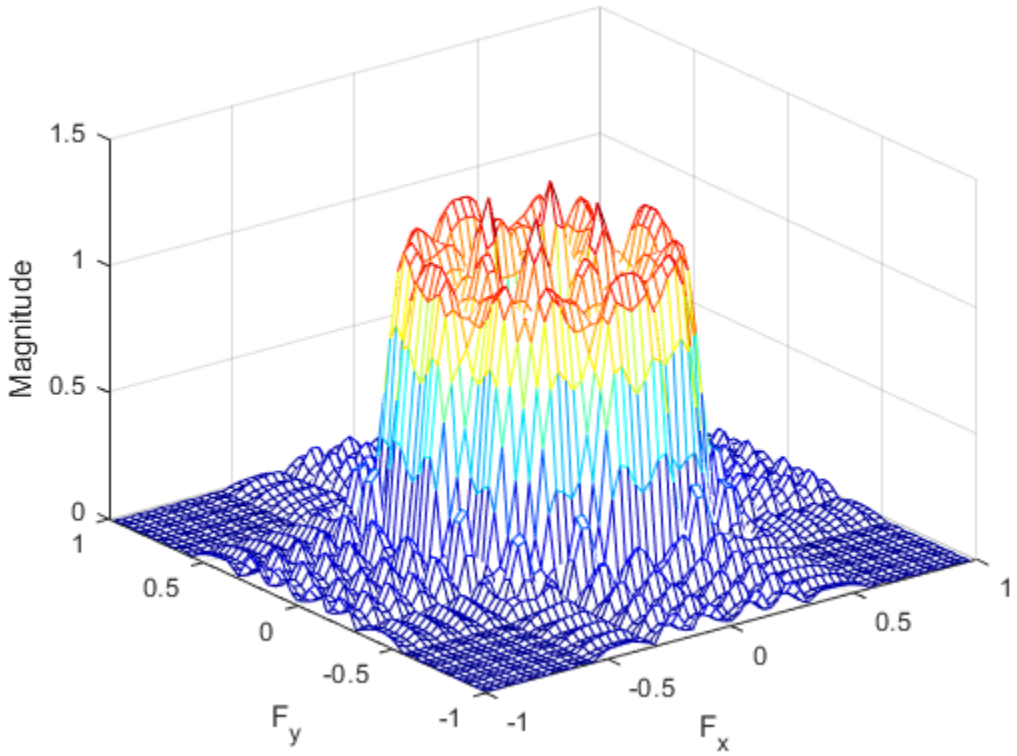
Create a matrix `Hd` that contains the desired bandpass response. Use `freqspace` to create the frequency vectors `f1` and `f2`.

```
[f1,f2] = freqspace(21,'meshgrid');
Hd = ones(21);
r = sqrt(f1.^2 + f2.^2);
Hd((r<0.1)|(r>0.5)) = 0;
colormap(jet(64))
mesh(f1,f2,Hd)
```



Design the filter that passes through this response.

```
h = fsamp2(Hd);
freqz2(h)
```



Input Arguments

Hd — Frequency response

numeric matrix

Frequency response, specified as a numeric matrix. `Hd` is a matrix containing the desired frequency response sampled at equally spaced points between -1.0 and 1.0 along the *x* and *y* frequency axes. The value 1.0 corresponds to half the sampling frequency, or π radians.

$$H_d(f_1, f_2) = H_d(\omega_1, \omega_2)|_{\omega_1 = \pi f_1, \omega_2 = \pi f_1}$$

For best results, use frequency points returned by `freqspace` to create `Hd`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

f1 — Frequency vector

numeric vector

Frequency vector, specified as a numeric vector.

Data Types: `double`

f2 — Frequency vector

numeric vector

Frequency vector, specified as a numeric vector.

Data Types: `double`

`[m n]` — Size of output FIR filter
2-element vector of positive integers

Size of output FIR filter `h`, specified as a 2-element vector of positive integers. The filter has `m` rows and `n` columns.

Data Types: `double`

Output Arguments

`h` — 2-D FIR filter
numeric array

2-D FIR filter, returned as a numeric array. `fsamp2` returns `h` as a computational molecule, which is the appropriate form to use with `filter2`. If you specify a frequency response matrix `Hd`, then `h` has the same size. If `Hd` is of class `single`, `h` is also of class `single`. Otherwise, `h` is of class `double`.

Data Types: `single` | `double`

Algorithms

`fsamp2` computes the filter `h` by taking the inverse discrete Fourier transform of the desired frequency response. If the desired frequency response is real and symmetric (zero phase), the resulting filter is also zero phase.

References

[1] Lim, Jae S., *Two-Dimensional Signal and Image Processing*, Englewood Cliffs, NJ, Prentice Hall, 1990, pp. 213-217.

See Also

`conv2` | `filter2` | `freqspace` | `ftrans2` | `fwind1` | `fwind2`

Topics
“Design Linear Filters in the Frequency Domain”

Introduced before R2006a

fspecial

Create predefined 2-D filter

Syntax

```
h = fspecial(type)
h = fspecial('average',hsize)
h = fspecial('disk',radius)
h = fspecial('gaussian',hsize,sigma)
h = fspecial('laplacian',alpha)
h = fspecial('log',hsize,sigma)
h = fspecial('motion',len,theta)
h = fspecial('prewitt')
h = fspecial('sobel')
```

Description

`h = fspecial(type)` creates a two-dimensional filter `h` of the specified `type`. Some of the filter types have optional additional parameters, shown in the following syntaxes. `fspecial` returns `h` as a correlation kernel, which is the appropriate form to use with `imfilter`.

`h = fspecial('average',hsize)` returns an averaging filter `h` of size `hsize`.

`h = fspecial('disk',radius)` returns a circular averaging filter (pillbox) within the square matrix of size `2*radius+1`.

`h = fspecial('gaussian',hsize,sigma)` returns a rotationally symmetric Gaussian lowpass filter of size `hsize` with standard deviation `sigma`. Not recommended. Use `imgaussfilt` or `imgaussfilt3` instead.

`h = fspecial('laplacian',alpha)` returns a 3-by-3 filter approximating the shape of the two-dimensional Laplacian operator, `alpha` controls the shape of the Laplacian.

`h = fspecial('log',hsize,sigma)` returns a rotationally symmetric Laplacian of Gaussian filter of size `hsize` with standard deviation `sigma`.

`h = fspecial('motion',len,theta)` returns a filter to approximate, once convolved with an image, the linear motion of a camera. `len` specifies the length of the motion and `theta` specifies the angle of motion in degrees in a counter-clockwise direction. The filter becomes a vector for horizontal and vertical motions. The default `len` is `9` and the default `theta` is `0`, which corresponds to a horizontal motion of nine pixels.

`h = fspecial('prewitt')` returns a 3-by-3 filter that emphasizes horizontal edges by approximating a vertical gradient. To emphasize vertical edges, transpose the filter `h'`.

```
[ 1  1  1
  0  0  0
 -1 -1 -1 ]
```

`h = fspecial('sobel')` returns a 3-by-3 filter that emphasizes horizontal edges using the smoothing effect by approximating a vertical gradient. To emphasize vertical edges, transpose the filter `h'`.

```
[ 1  2  1
  0  0  0
 -1 -2 -1 ]
```

Examples

Create Various Filters and Filter an Image

Read image and display it.

```
I = imread('cameraman.tif');
imshow(I);
```

Create a motion filter and use it to blur the image. Display the blurred image.

```
H = fspecial('motion',20,45);
MotionBlur = imfilter(I,H,'replicate');
imshow(MotionBlur);
```

Create a disk filter and use it to blur the image. Display the blurred image.

```
H = fspecial('disk',10);
blurred = imfilter(I,H,'replicate');
imshow(blurred);
```

Input Arguments

`type` — Type of filter

`'average'` | `'disk'` | `'gaussian'` | `'laplacian'` | `'log'` | `'motion'` | `'prewitt'` | `'sobel'`

Type of filter, specified as one of the following values:

Value	Description
`'average'`	Averaging filter
`'disk'`	Circular averaging filter (pillbox)
`'gaussian'`	Gaussian lowpass filter. Not recommended. Use `imgaussfilt` or `imgaussfilt3` instead.
`'laplacian'`	Approximates the two-dimensional Laplacian operator
`'log'`	Laplacian of Gaussian filter
`'motion'`	Approximates the linear motion of a camera
`'prewitt'`	Prewitt horizontal edge-emphasizing filter
`'sobel'`	Sobel horizontal edge-emphasizing filter

Data Types: `char` | `string`

`hsize` — Size of the filter

positive integer | 2-element vector of positive integers

Size of the filter, specified as a positive integer or 2-element vector of positive integers. Use a vector to specify the number of rows and columns in `h`. If you specify a scalar, then `h` is a square matrix.

When used with the `'average'` filter type, the default filter size is `[3 3]`. When used with the Laplacian of Gaussian (`'log'`) filter type, the default filter size is `[5 5]`.

Data Types: `double`

`radius` — Radius of a disk-shaped filter

`5` (default) | positive number

Radius of a disk-shaped filter, specified as a positive number. The filter is a square matrix of size `2*radius+1`.

Data Types: `double`

`sigma` — Standard deviation

`0.5` (default) | positive number

Standard deviation, specified as a positive number.

Data Types: `double`

`alpha` — Shape of the Laplacian

`0.2` (default) | number in the range [0, 1]

Shape of the Laplacian, specified as a number in the range [0, 1]. Specify `alpha` as `0` to obtain a 4-neighborhood Laplacian filter:

```
[ 0  1 0
  1 -4 1
  0  1 0 ]
```

Data Types: `double`

`len` — Linear motion of camera
`9` (default) | numeric scalar

Linear motion of camera, specified as a numeric scalar, measured in pixels.

Data Types: `double`

`theta` — Angle of camera motion
`0` (default) | numeric scalar

Angle of camera motion in degrees, specified as a numeric scalar. The angle is measured in a counter-clockwise direction from horizontal.

Data Types: `double`

Output Arguments

`h` — Correlation kernel
matrix

Correlation kernel, returned as a matrix.

Data Types: `double`

Algorithms

Averaging filters:

`ones(n(1),n(2))/(n(1)*n(2))`

Gaussian filters:

$$h_g(n_1, n_2) = e^{\frac{-(n_1^2 + n_2^2)}{2\sigma^2}}$$

$$h(n_1, n_2) = \frac{h_g(n_1, n_2)}{\sum_{n_1}\sum_{n_2} h_g}$$

Laplacian filters:

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

$$\nabla^2 = \frac{4}{(\alpha + 1)}\begin{bmatrix} \frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4} \\ \frac{1-\alpha}{4} & -1 & \frac{1-\alpha}{4} \\ \frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4} \end{bmatrix}$$

Laplacian of Gaussian (LoG) filters:

$$h_g(n_1, n_2) = e^{\frac{-(n_1^2 + n_2^2)}{2\sigma^2}}$$

$$h(n_1, n_2) = \frac{(n_1^2 + n_2^2 - 2\sigma^2)h_g(n_1, n_2)}{\sigma^4 \sum_{n_1} \sum_{n_2} h_g}$$

Note that `fspecial` shifts the equation to ensure that the sum of all elements of the kernel is zero (similar to the Laplace kernel) so that the convolution result of homogeneous regions is always zero.

Motion filters:

1. Construct an ideal line segment with the length and angle specified by the arguments `len` and `theta`, centered at the center coefficient of `h`.
2. For each coefficient location `(i,j)`, compute the nearest distance between that location and the ideal line segment.
3. `h = max(1 - nearest_distance,0);`
4. Normalize `h`: `h = h/(sum(h(:)))`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `fspecial` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- When generating code, all inputs must be constants at compilation time.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, all inputs must be constants at compilation time.

See Also

`conv2` | `del2` | `edge` | `imsharpen` | `imfilter` | `filter2` | `fspecial3`

Topics

“Filter Images Using Predefined Filter”
“What Is Image Filtering in the Spatial Domain?”

Introduced before R2006a

fspecial3

Create predefined 3-D filter

Syntax

```
h = fspecial3(type)
h = fspecial3('average',hsize)
h = fspecial3('ellipsoid',semiaxes)
h = fspecial3('gaussian',hsize,sigma)
h = fspecial3('laplacian',gamma1,gamma2)
h = fspecial3('log',hsize,sigma)
h = fspecial3('prewitt',direction)
h = fspecial3('sobel',direction)
```

Description

`h = fspecial3(type)` creates a three-dimensional filter `h` of the specified `type`. Some of the filter types have optional additional parameters, shown in the following syntaxes. `fspecial3` returns `h` as a correlation kernel, which is the appropriate form to use with `imfilter`.

`h = fspecial3('average',hsize)` returns an averaging filter `h` of size `hsize`. Not recommended. Use `imboxfilt3` instead.

`h = fspecial3('ellipsoid',semiaxes)` returns an ellipsoidal averaging filter with the length of the principal semiaxes specified by `semiaxes`. The filter `h` is returned in an array of size `2*ceil(semiaxes)+1`.

`h = fspecial3('gaussian',hsize,sigma)` returns a Gaussian lowpass filter of size `hsize` with standard deviation `sigma`. Not recommended. Use `imgaussfilt3` instead.

`h = fspecial3('laplacian',gamma1,gamma2)` returns a 3-by-3-by-3 filter approximating the shape of the three-dimensional Laplacian operator. `gamma1` and `gamma2` control the shape of the Laplacian [1][2].

`h = fspecial3('log',hsize,sigma)` returns a Laplacian of Gaussian filter of size `hsize` with standard deviation `sigma`.

`h = fspecial3('prewitt',direction)` returns a 3-by-3-by-3 filter that emphasizes gradients in the specified direction.

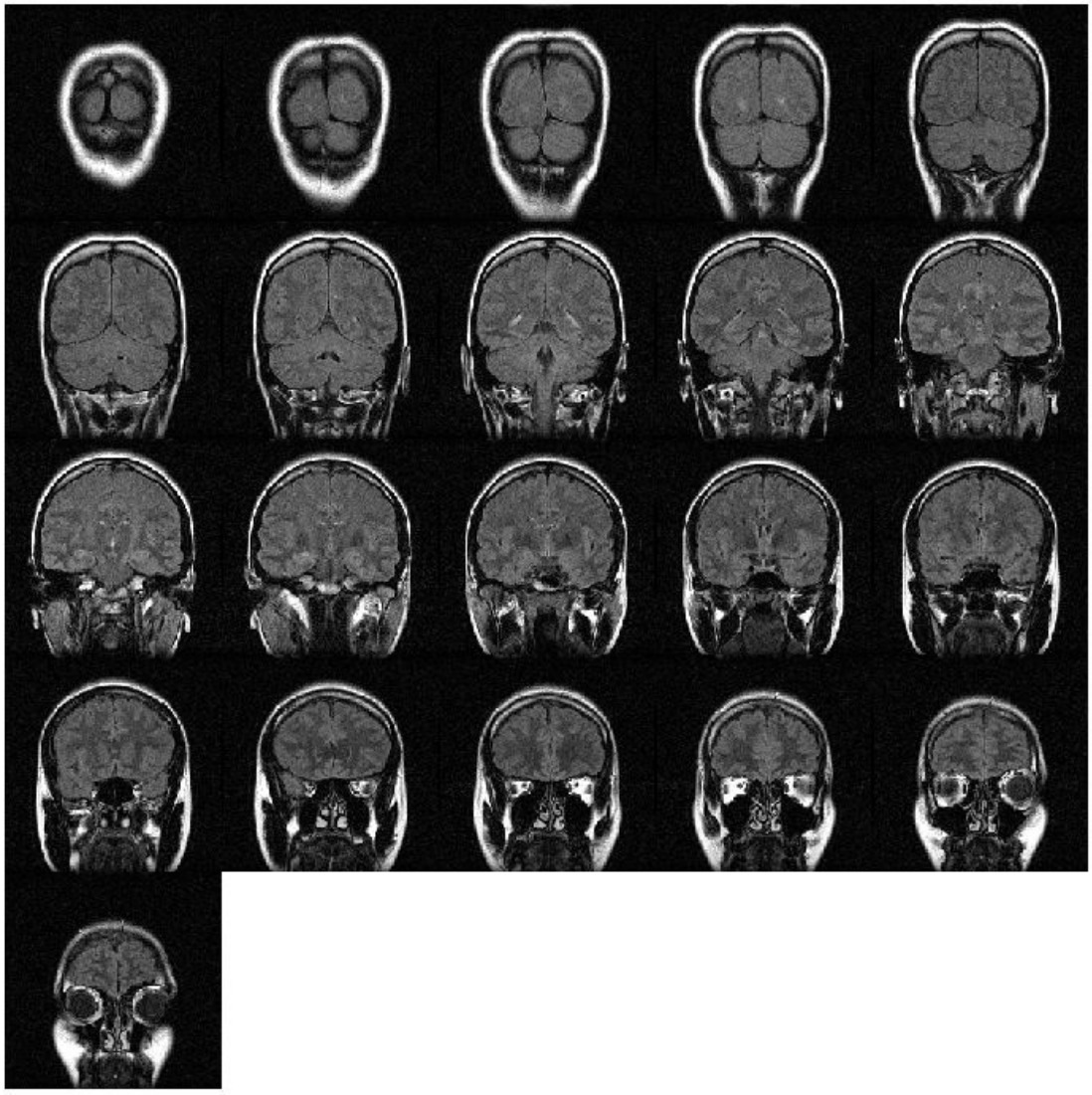
`h = fspecial3('sobel',direction)` returns a 3-by-3-by-3 filter that emphasizes gradients in the specified direction and smooths the other directions [3].

Examples

Smooth Volume Using 3-D Ellipsoidal Filter

Load a 3-D grayscale MRI volume. Display the planes of the volume.

```
load mristack;
montage(mristack,'BackgroundColor','w')
```



Create a 3-D ellipsoidal filter. Specify a semiaxis length of 7 pixels in the *y* (rows) and *x* (columns) directions, and a semiaxis length of 3 pixels in the *z* (planes) direction.

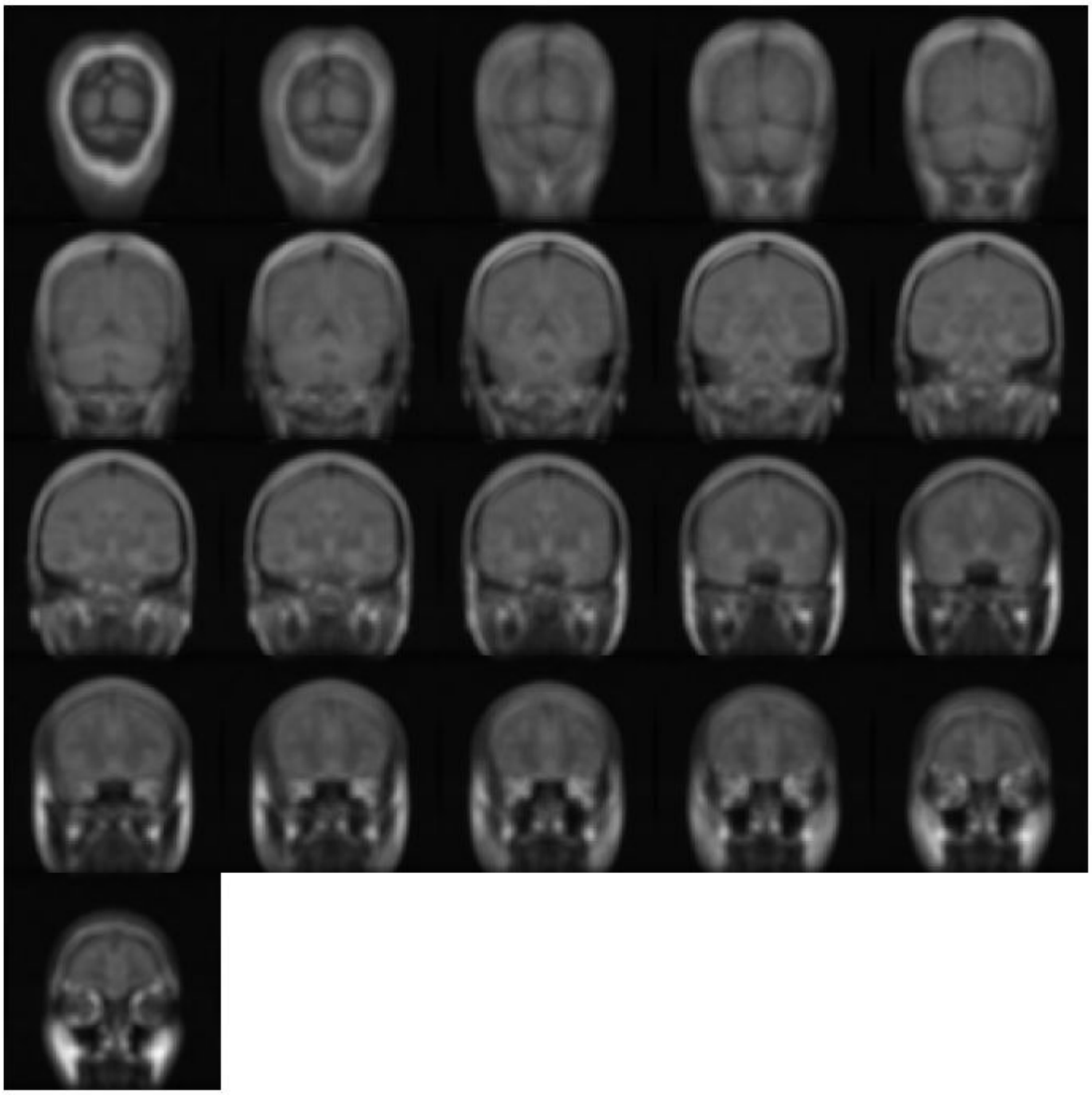
```
H = fspecial3('ellipsoid',[7 7 3]);
```

Smooth the volume with the filter.

```
volSmooth = imfilter(mristack,H,'replicate');
```

Display the planes of the smoothed volume.

```
montage(volSmooth,'BackgroundColor','w')
```



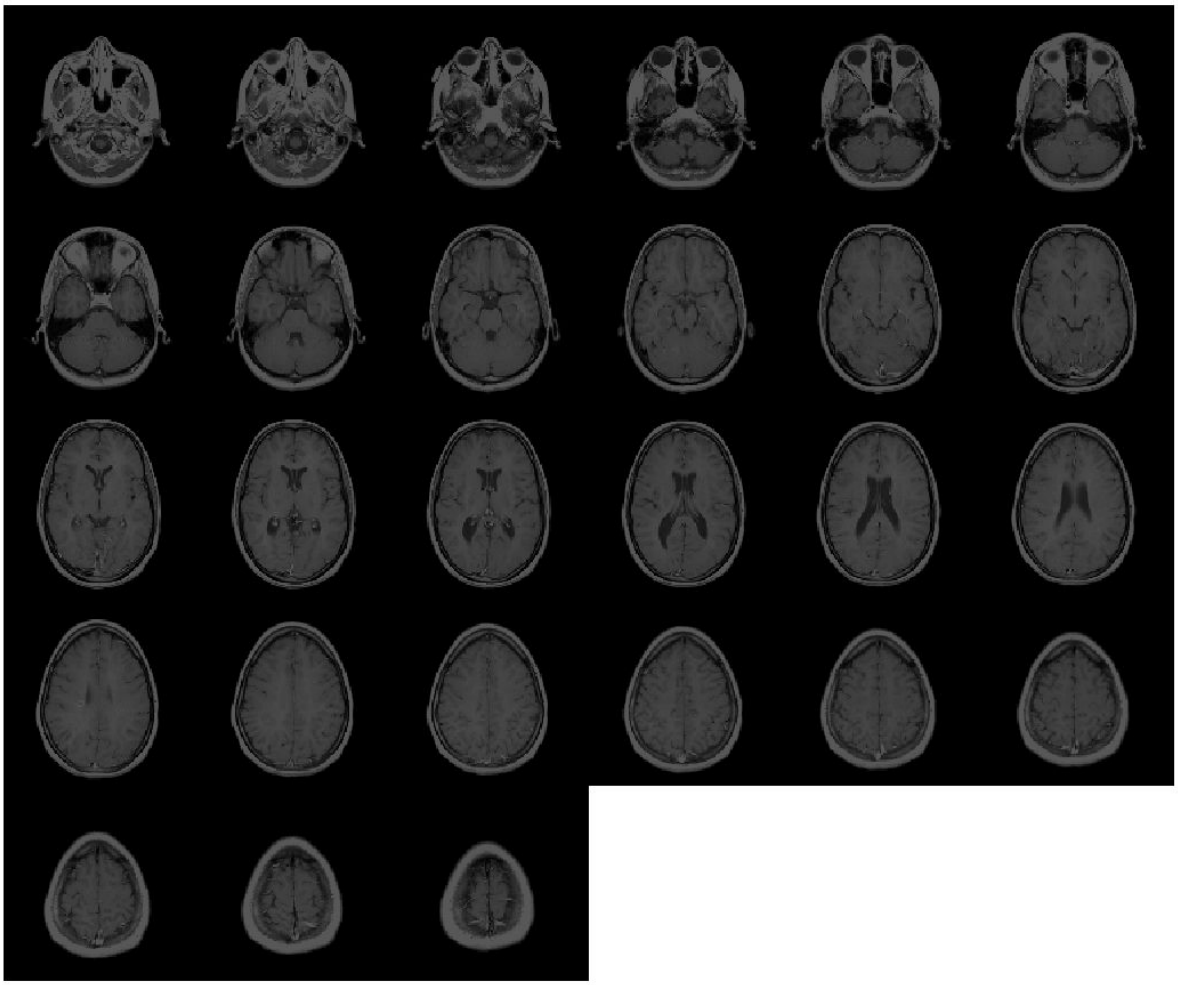
Detect Horizontal Edges Using 3-D Sobel Filter

Load an MRI volume. This volume is stored as a 4-D array with a singleton dimension. Create a 3-D grayscale volume by using the `squeeze` function to remove the singleton dimension.

```
load mri;
V = squeeze(D);
```

Display the planes of the volume.

```
montage(D,'BackgroundColor','w')
```



Create a 3-D Sobel filter that detects horizontal edges in the volume. Horizontal edges appear where there is a large gradient magnitude in the *y* direction, so specify the direction of the Sobel filter as `'Y'`. The Sobel filter smooths the gradient in the *x* and *z* directions.

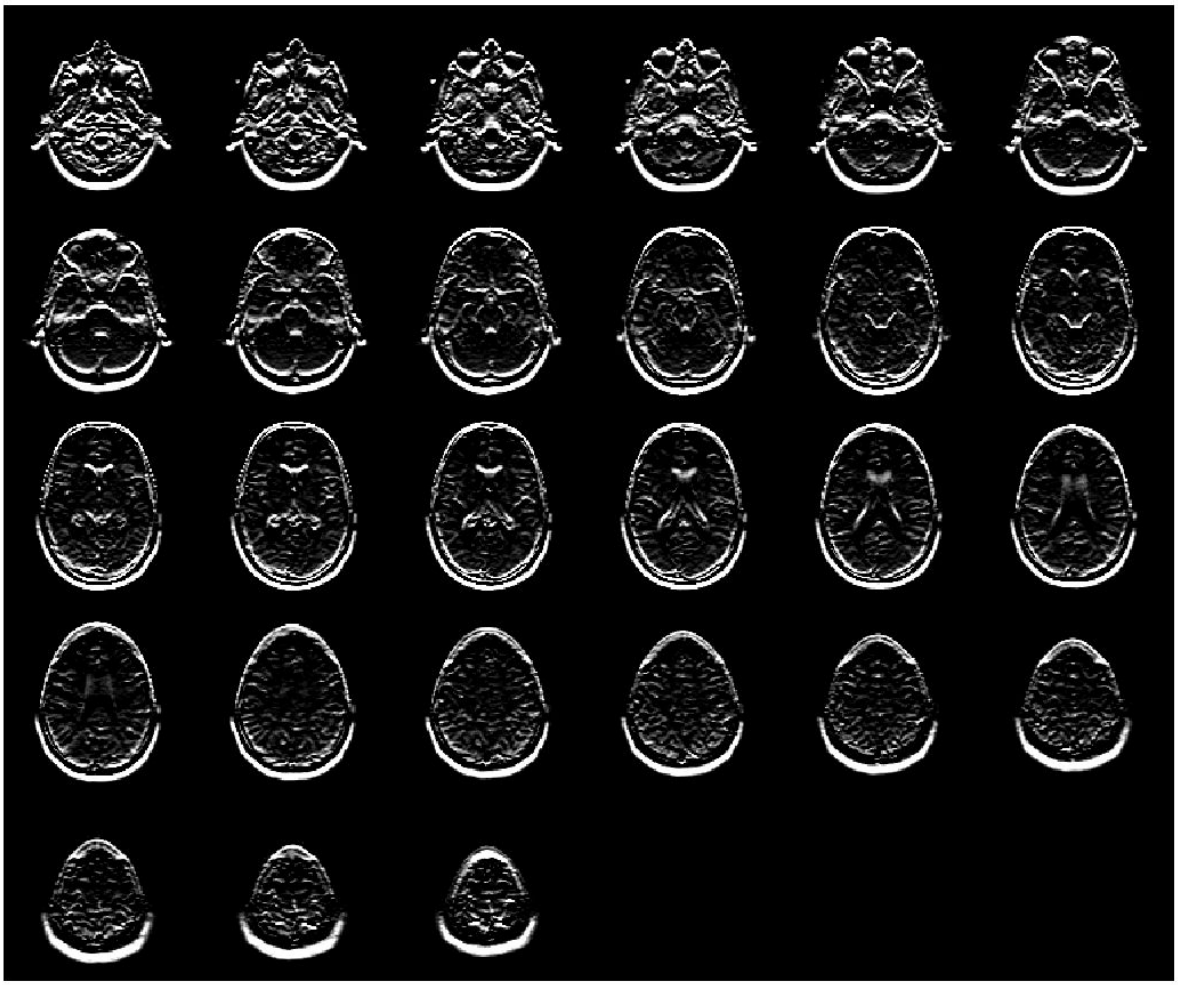
```
H = fspecial3('sobel','Y');
```

Filter the volume with the 3-D Sobel filter.

```
edgesHor = imfilter(V,H,'replicate');
```

Display the planes of the filtered volume.

```
montage(edgesHor)
```



Input Arguments

`type` — Type of filter

`'average'` | `'ellipsoid'` | `'gaussian'` | `'laplacian'` | `'log'` | `'prewitt'` | `'sobel'`

Type of filter, specified as one of the following values:

Value	Description
`'average'`	Averaging filter. Not recommended. Use `imboxfilt3` instead.
`'ellipsoid'`	Ellipsoidal averaging filter
`'gaussian'`	Gaussian lowpass filter. Not recommended. Use `imgaussfilt3` instead.
`'laplacian'`	Approximates the three-dimensional Laplacian operator
`'log'`	Laplacian of Gaussian filter

Value	Description
`'prewitt'`	Prewitt edge-emphasizing filter
`'sobel'`	Sobel edge-emphasizing filter

Data Types: `char` | `string`

`hsize` — Size of the filter
`[5 5 5]` (default) | positive integer | 3-element vector of positive integers

Size of the filter, specified as a positive integer or 3-element vector of positive integers. Use a vector to specify the number of rows, columns, and planes in `h`. Use a scalar to specify the side length of a cube.

For the `'gaussian'` and `'log'` filter types, if you specify `hsize` as `[]`, then `fspecial3` creates a filter with a default size of `2*ceil(2*sigma)+1`.

Data Types: `double`

`semiaxes` — Semiaxes length of ellipsoidal filter
`5` (default) | positive number | 3-element vector of positive numbers

Semiaxes length of an ellipsoidal filter, specified as a positive number or 3-element vector of positive numbers. Use a vector to specify the length of the three principal semiaxes in rows, columns, and planes. These values correspond to length in the Cartesian *y*, *x*, and *z* directions, respectively. Use a scalar to specify the radius of a sphere.

Data Types: `double`

`sigma` — Standard deviation of Gaussian filter
`1` (default) | positive number | 3-element vector of positive numbers

Standard deviation of Gaussian filter, specified as a positive number or 3-element vector of positive numbers. If you specify a scalar, then `fspecial3` creates a cubic Gaussian kernel.

Data Types: `double`

`gamma1`, `gamma2` — Shape of the Laplacian
`0` (default) | scalar in the range [0 1]

Shape of the Laplacian, specified as a scalar in the range [0 1]. The sum of `gamma1` and `gamma2` must not exceed 1.

Data Types: `double`

`direction` — Direction of gradients
`'X'` (default) | `'Y'` | `'Z'`

Direction of gradients for Prewitt and Sobel filtering, specified as `'X'`, `'Y'`, or `'Z'`.

Data Types: `char` | `string`

Output Arguments

`h` — Correlation kernel
numeric array

Correlation kernel, returned as a numeric array.

Data Types: `double`

References

[1] Lindeberg, T., *Scale-Space Theory in Computer Vision*. Boston, MA: Kluwer Academic Publishers, 1994.

[2] *Geometry-Driven Diffusion in Computer Vision*. Edited by B. M. ter Haar Romeny. Boston, MA: Kluwer Academic Publishers, 1994.

[3] Engel, K., M. Hadwiger, J. M. Kniss, C. Rezk-Salama, and D. Weiskopf. *Real-Time Volume Graphics*. Wellesley, MA: A K Peters, Ltd., 2006, pp. 112–114.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `fspecial3` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- When generating code, all inputs must be constants at compilation time.

See Also

`imfilter` | `fspecial` | `imboxfilt3` | `imgaussfilt3` | `edge3`

Topics

“What Is Image Filtering in the Spatial Domain?”

Introduced in R2018b

ftrans2

2-D FIR filter using frequency transformation

Syntax

```
h = ftrans2(b,t)
h = ftrans2(b)
```

Description

`h = ftrans2(b,t)` produces the two-dimensional FIR filter `h` that corresponds to the one-dimensional FIR filter `b` using the transform `t`. The transform matrix `t` contains coefficients that define the frequency transformation to use.

`h = ftrans2(b)` uses the McClellan transform matrix `t`.

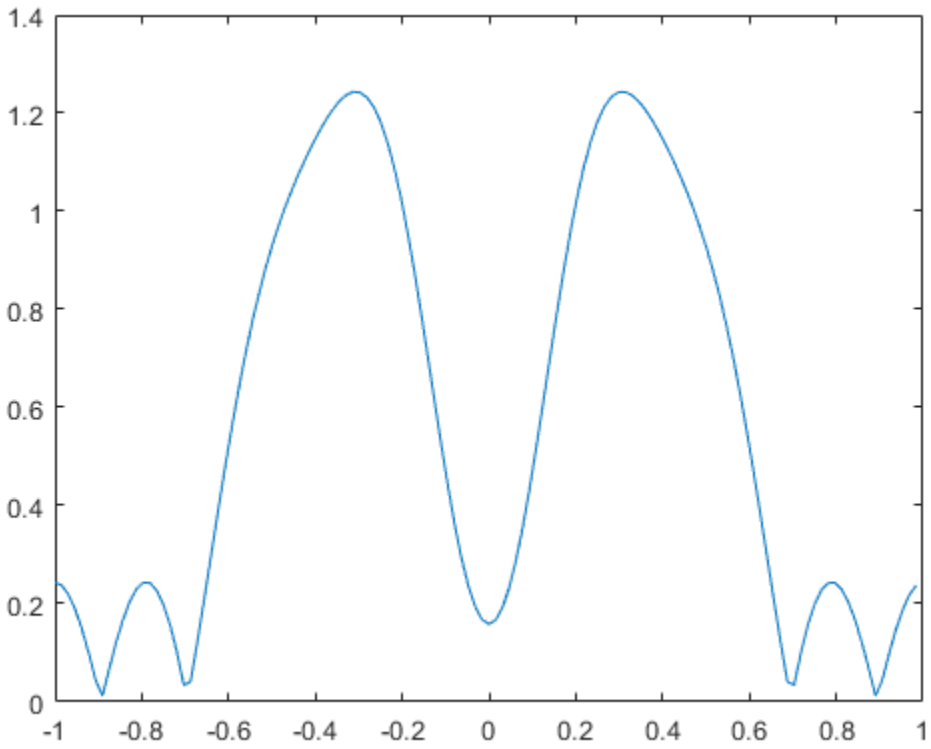
```
t = [1 2 1; 2 -4 2; 1 2 1]/8;
```

Examples

Design Circularly Symmetric 2-D Bandpass Filter

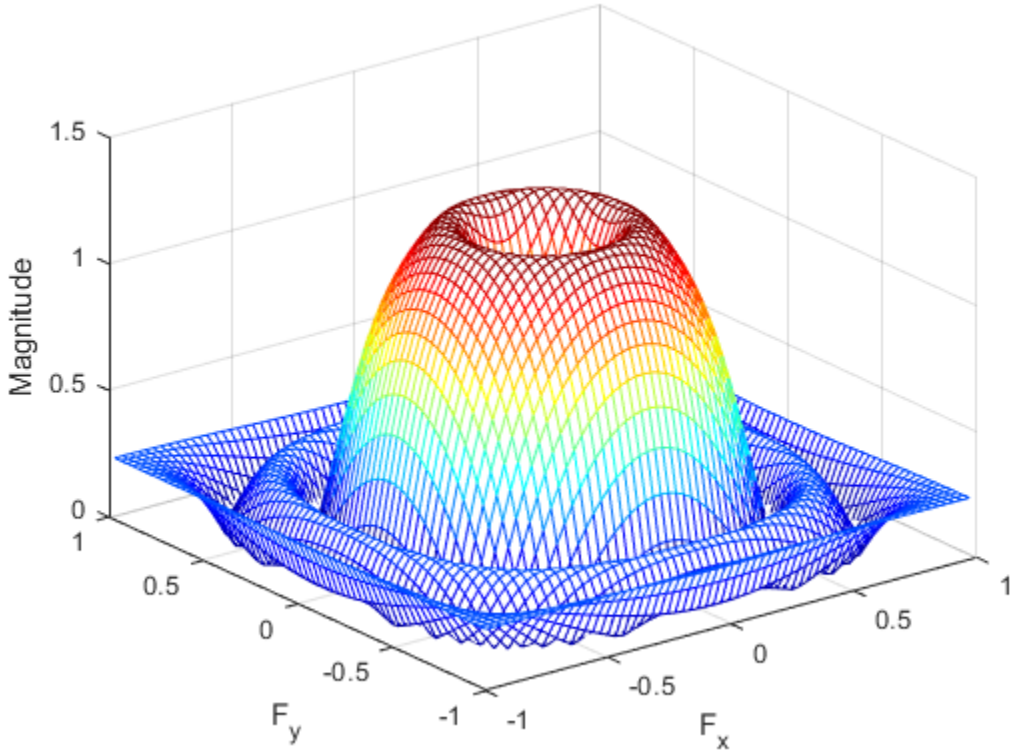
Use `ftrans2` to design an approximately circularly symmetric two-dimensional bandpass filter with passband between 0.1 and 0.6 (normalized frequency, where 1.0 corresponds to half the sampling frequency, or π radians). Since `ftrans2` transforms a one-dimensional FIR filter to create a two-dimensional filter, first design a one-dimensional FIR bandpass filter using the Signal Processing Toolbox function `firpm`.

```
colormap(jet(64))
b = firpm(10,[0 0.05 0.15 0.55 0.65 1],[0 0 1 1 0 0]);
[H,w] = freqz(b,1,128,'whole');
plot(w/pi-1,fftshift(abs(H)))
```



Use `ftrans2` with the default McClellan transformation to create the desired approximately circularly symmetric filter.

```
h = ftrans2(b);
freqz2(h)
```



Input Arguments

b — FIR filter

numeric matrix

FIR filter, specified as a numeric matrix. `b` must be a 1-D Type I (even symmetric, odd-length) filter such as can be returned by `fir1`, `fir2`, or `firpm`.

Data Types: `double`

t — Transform matrix

numeric matrix

The transform matrix, specified as a numeric matrix. `t` contains coefficients that define the frequency transformation to use. By default, `ftrans2` uses a McClellan transform matrix.

Data Types: `double`

Output Arguments

h — 2-D FIR filter

numeric matrix

2-D FIR filter, returned as a numeric matrix. `ftrans2` returns `h` as a computational molecule, which is the appropriate form to use with `filter2`. If `t` is `m`-by-`n` and `b` has length `Q`, then `h` is size `((m-1)*(Q-1)/2+1)`-by-`((n-1)*(Q-1)/2+1)`.

Algorithms

The transformation below defines the frequency response of the two-dimensional filter returned by `ftrans2`.

$$H_{(\omega_1,\omega_2)} = B(\omega)\Big|_{\cos\omega = T(\omega_1,\omega_2)},$$

where $B(\omega)$ is the Fourier transform of the one-dimensional filter `b`:

$$B(\omega) = \sum_{n=-N}^{N} b(n)e^{-j\omega n}$$

and $T(\omega_1,\omega_2)$ is the Fourier transform of the transformation matrix `t`:

$$T(\omega_1,\omega_2) = \sum_{n_2}\sum_{n_1} t(n_1,n_2)e^{-j\omega_1 n_1}e^{-j\omega_2 n_2}.$$

The returned filter `h` is the inverse Fourier transform of $H(\omega_1,\omega_2)$:

$$h(n_1,n_2) = \frac{1}{(2\pi)^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi} H(\omega_1,\omega_2)e^{j\omega_1 n_1}e^{j\omega_2 n_2}d\omega_1 d\omega_2.$$

References

[1] Lim, Jae S., *Two-Dimensional Signal and Image Processing*, Englewood Cliffs, NJ, Prentice Hall, 1990, pp. 218-237.

See Also

`conv2` | `filter2` | `fsamp2` | `fwind1` | `fwind2`

Topics

“Design Linear Filters in the Frequency Domain”

Introduced before R2006a

fwind1

2-D FIR filter using 1-D window method

Syntax

```
h = fwind1(Hd,win)
h = fwind1(Hd,win1,win2)
h = fwind1(f1,f2,Hd,___)
```

Description

The `fwind1` function designs 2-D FIR filters using the window method. `fwind1` uses a 1-D window specification to design a 2-D FIR filter based on the desired frequency response. `fwind1` works with 1-D windows only. Use `fwind2` to work with 2-D windows.

You can apply the 2-D FIR filter to images by using the `filter2` function.

`h = fwind1(Hd,win)` creates a 2-D FIR filter `h` based on the desired frequency response `Hd`. The `fwind1` function uses the 1-D window `win` to form an approximately circularly symmetric 2-D window using Huang's method.

`h = fwind1(Hd,win1,win2)` uses two 1-D windows, `win1` and `win2`, to create a separable 2-D window.

`h = fwind1(f1,f2,Hd,___)` enables you to specify the desired frequency response `Hd` at arbitrary frequencies `f1` and `f2` along the *x*- and *y*-axes.

Examples

Create 2-D FIR Filter using 1-D Window Method

This example shows how to design an approximately circularly symmetric two-dimensional bandpass filter using a 1-D window method.

Create the frequency range vectors `f1` and `f2` using `freqspace`. These vectors have length 21.

```
[f1,f2] = freqspace(21,'meshgrid');
```

Compute the distance of each position from the center frequency.

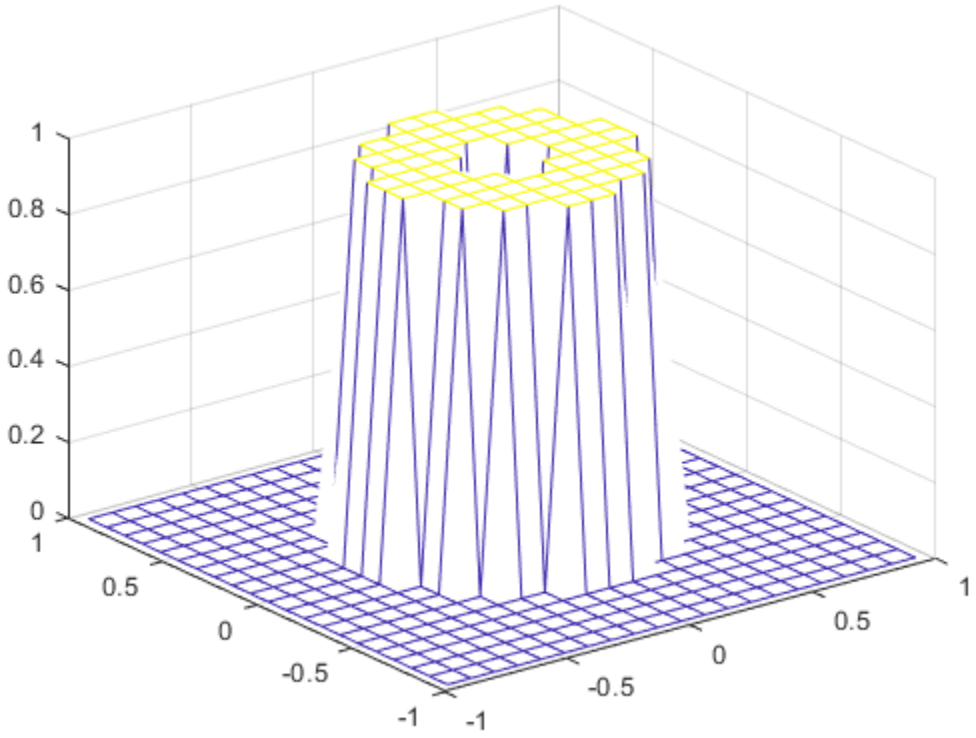
```
r = sqrt(f1.^2 + f2.^2);
```

Create a matrix `Hd` that contains the desired bandpass response. In this example, the desired passband is between 0.1 and 0.5 (normalized frequency, where 1.0 corresponds to half the sampling frequency, or π radians).

```
Hd = ones(21);
Hd((r<0.1)|(r>0.5)) = 0;
```

Display the ideal bandpass response.

```
colormap(parula(64))
mesh(f1,f2,Hd)
```

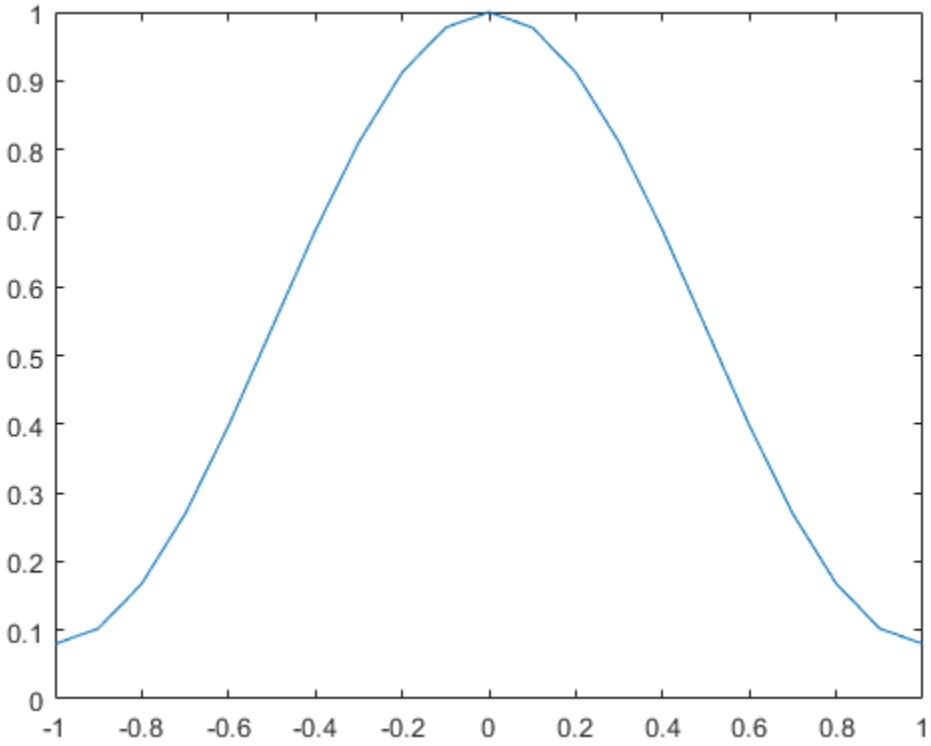


Design the 1-D window. This example uses a Hamming window of length 21.

```
win = 0.54 - 0.46*cos(2*pi*(0:20)/20);
```

Plot the 1-D window.

```
figure
plot(linspace(-1,1,21),win);
```

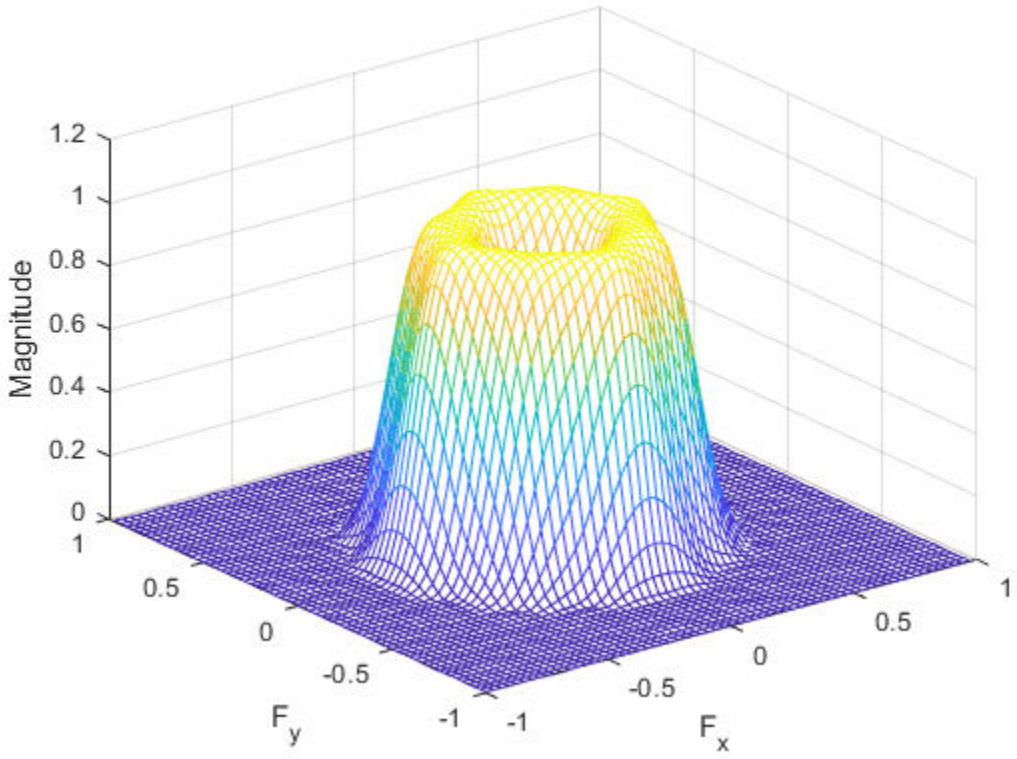


Using the 1-D window, design the filter that best produces this frequency response

```
h = fwind1(Hd,win);
```

Display the actual frequency response of this filter.

```
freqz2(h)
```



Input Arguments

`Hd` — Desired frequency response

numeric matrix

Desired frequency response, specified as a numeric matrix. `Hd` is sampled at equally spaced points between -1.0 and 1.0 (in normalized frequency, where 1.0 corresponds to half the sampling frequency, or π radians) along the *x* and *y* frequency axes. For accurate results, create `Hd` by using the `freqspace` function.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`win` — 1-D window

numeric matrix

1-D window, specified as a numeric matrix. If you have Signal Processing Toolbox™ software, then you can specify `win` using windows such as `hamming`, `hann`, `bartlett`, `blackman`, `kaiser`, or `chebwin`.

Data Types: `single` | `double`

`win1` — 1-D window

numeric matrix

1-D window, specified as a numeric matrix.

Data Types: `single` | `double`

`win2` — 1-D window
numeric matrix

1-D window, specified as a numeric matrix.

Data Types: `single` | `double`

`f1` — Desired frequency along the x-axis
vector

Desired frequency along the *x*-axis. The frequency vector should be in the range [-1, 1], where 1.0 corresponds to half the sampling frequency, or π radians.

Data Types: `single` | `double`

`f2` — Desired frequency along the y-axis
vector

Desired frequency along the *y*-axis. The frequency vector should be in the range [-1, 1], where 1.0 corresponds to half the sampling frequency, or π radians.

Data Types: `single` | `double`

Output Arguments

`h` — 2-D FIR filter
numeric matrix

2-D FIR filter, returned as a numeric matrix. The length of the window controls the size of the resulting filter.

- If you specify a single window `win` of length `n`, then the size of `h` is `n`-by-`n`.
- If you specify two windows `win1` and `win2` of length `n` and `m` respectively, then the size of `h` is `m`-by-`n`.

If `Hd` is of data type `single`, then `h` is of data type `single`. Otherwise, `h` is of data type `double`.

Data Types: `single` | `double`

Algorithms

The `fwind1` function takes a one-dimensional window specification and forms an approximately circularly symmetric two-dimensional window using Huang's method,

$$w(n_1, n_2) = w(t)|_{t = \sqrt{n_1^2 + n_2^2}},$$

where $w(t)$ is the one-dimensional window and $w(n_1,n_2)$ is the resulting two-dimensional window.

Given two windows, the `fwind1` function forms a separable two-dimensional window:

$$w(n_1, n_2) = w_1(n_1)w_2(n_2)\,.$$

The `fwind1` function calls the `fwind2` with the desired frequency response `Hd` and the two-dimensional window. The `fwind2` function calculates `h` using an inverse Fourier transform and multiplication by the two-dimensional window:

$$h_d(n_1, n_2) = \frac{1}{(2\pi)^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi} H_d(\omega_1, \omega_2)e^{j\omega_1 n_1}e^{j\omega_2 n_2}d\omega_1 d\omega_2$$

$$h(n_1, n_2) = h_d(n_1, n_2)w(n_1, n_2)$$

References

[1] Lim, Jae S., *Two-Dimensional Signal and Image Processing*, Englewood Cliffs, NJ, Prentice Hall, 1990.

See Also

`conv2` | `filter2` | `fsamp2` | `freqspace` | `ftrans2` | `fwind2`

Topics

“Design Linear Filters in the Frequency Domain”

Introduced before R2006a

fwind2

2-D FIR filter using 2-D window method

Syntax

```
h = fwind2(Hd,win)
h = fwind2(f1,f2,Hd,win)
```

Description

The `fwind2` function designs 2-D FIR filters using the window method. `fwind2` uses a 2-D window specification to design a 2-D FIR filter based on the desired frequency response. `fwind2` works with 2-D windows only. Use `fwind1` to create a 2-D FIR filter from a 1-D window.

You can apply the 2-D FIR filter to images by using the `filter2` function.

`h = fwind2(Hd,win)` creates a 2-D FIR filter `h` by using an inverse Fourier transform of the desired frequency response `Hd` and multiplication by the window `win`.

`h = fwind2(f1,f2,Hd,win)` lets you specify the desired frequency response `Hd` at arbitrary frequencies `f1` and `f2` along the *x*- and *y*-axes.

Examples

Create 2-D FIR Filter using 2-D Window Method

This example shows how to design an approximately circularly symmetric two-dimensional bandpass filter using a 2-D window method.

Create the frequency range vectors `f1` and `f2` using `freqspace`. These vectors have length 21.

```
[f1,f2] = freqspace(21,'meshgrid');
```

Compute the distance of each position from the center frequency.

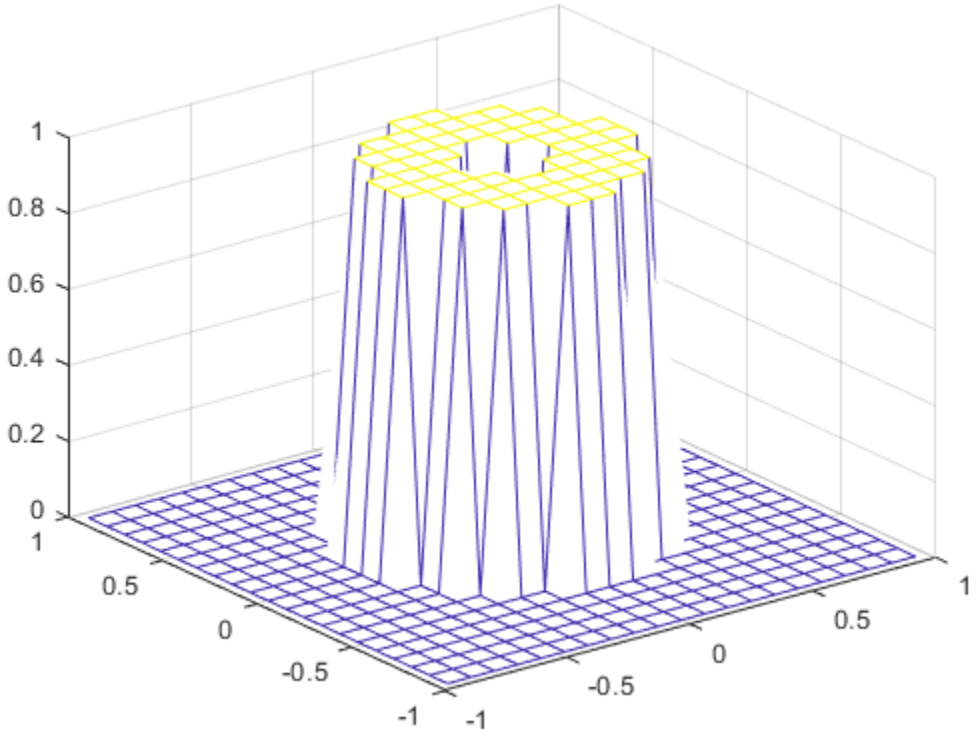
```
r = sqrt(f1.^2 + f2.^2);
```

Create a matrix `Hd` that contains the desired bandpass response. In this example, the desired passband is between 0.1 and 0.5 (normalized frequency, where 1.0 corresponds to half the sampling frequency, or π radians).

```
Hd = ones(21);
Hd((r<0.1)|(r>0.5)) = 0;
```

Display the ideal bandpass response.

```
colormap(parula(64))
mesh(f1,f2,Hd)
```

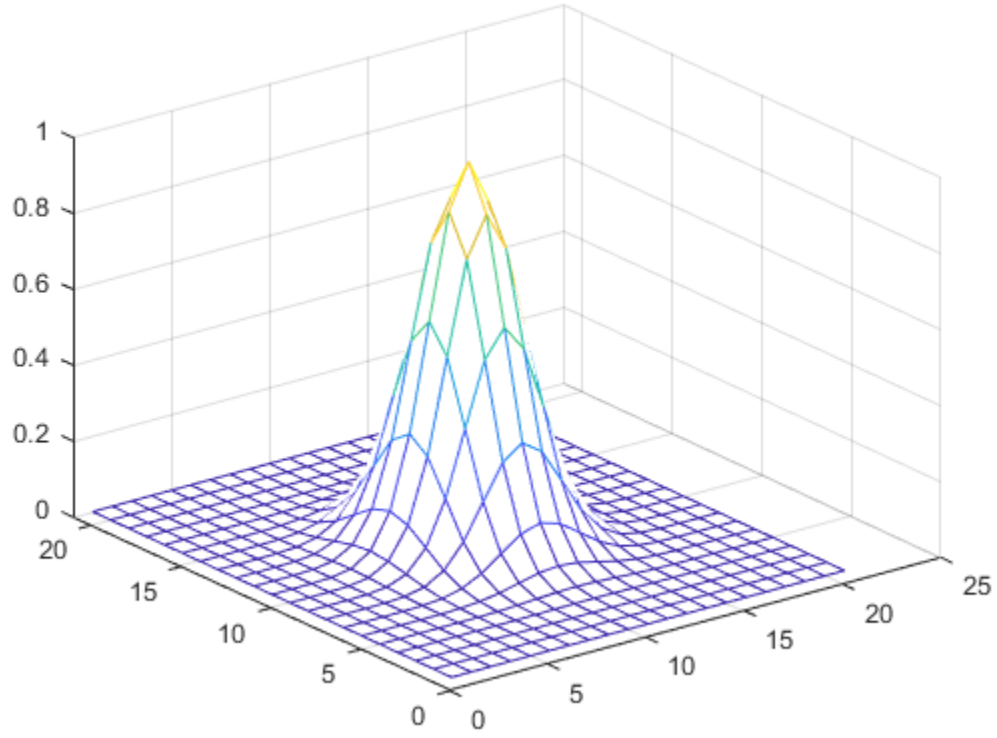


Create a 2-D Gaussian window using `fspecial`. Normalize the window.

```
win = fspecial('gaussian',21,2);
win = win ./ max(win(:));
```

Display the window.

```
mesh(win)
```

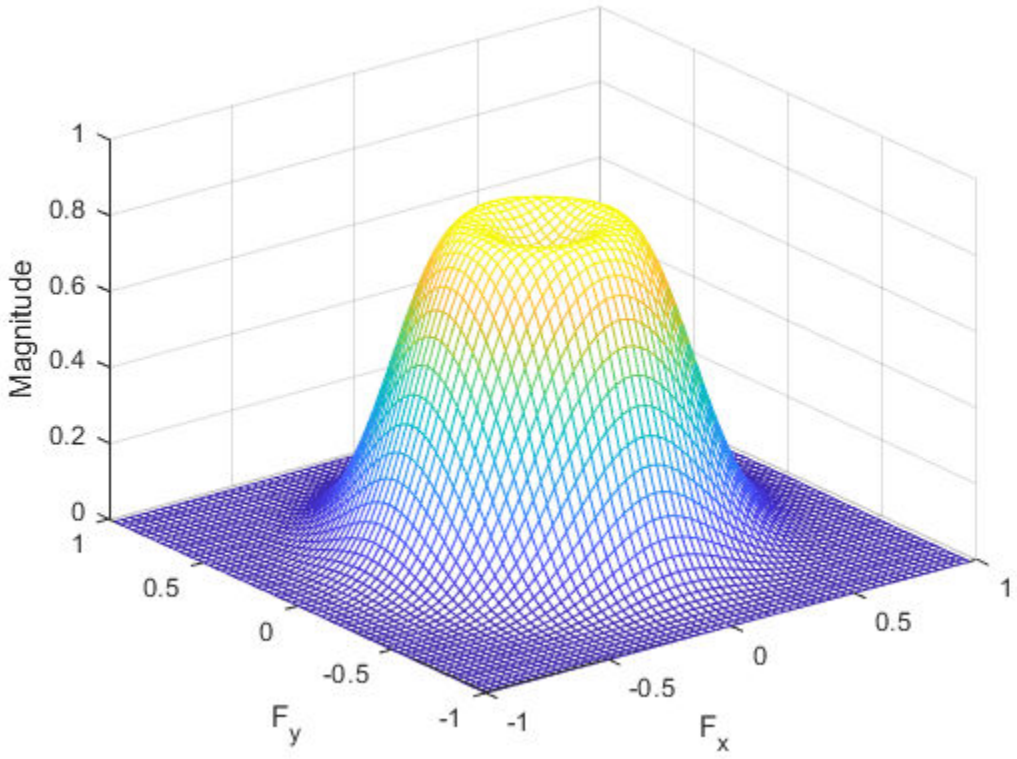


Using the 2-D window, design the filter that best produces the desired frequency response

```
h = fwind2(Hd,win);
```

Display the actual frequency response of this filter.

```
freqz2(h)
```



Input Arguments

`Hd` — Desired frequency response

numeric matrix

Desired frequency response at equally spaced points in the Cartesian plane, specified as a numeric matrix. For accurate results, create `Hd` by using the `freqspace` function.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`win` — 2-D window

numeric matrix

2-D window, specified as a numeric matrix.

Data Types: `single` | `double`

`f1` — Desired frequency along the x-axis

vector

Desired frequency along the *x*-axis. The frequency vector should be in the range [-1, 1], where 1.0 corresponds to half the sampling frequency, or π radians.

Data Types: `single` | `double`

`f2` — Desired frequency along the y-axis

vector

Desired frequency along the *y*-axis. The frequency vector should be in the range [-1, 1], where 1.0 corresponds to half the sampling frequency, or π radians.

Data Types: `single` | `double`

Output Arguments

`h` — 2-D FIR filter

numeric matrix

2-D FIR filter, returned as a numeric matrix of the same size as `win`.

Data Types: `double`

Algorithms

`fwind2` computes `h` using an inverse Fourier transform and multiplication by the two-dimensional window `win`.

$$h_d(n_1, n_2) = \frac{1}{(2\pi)^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi} H_d(\omega_1, \omega_2) e^{j\omega_1 n_1} e^{j\omega_2 n_2} d\omega_1 d\omega_2$$

$$h(n_1, n_2) = h_d(n_1, n_2) w(n_1, n_2)$$

References

[1] Lim, Jae S., *Two-Dimensional Signal and Image Processing*, Englewood Cliffs, NJ, Prentice Hall, 1990, pp. 202-213.

See Also

`conv2` | `filter2` | `fsamp2` | `freqspace` | `ftrans2` | `fwind1`

Introduced before R2006a

gabor

Create Gabor filter or Gabor filter bank

Description

A `gabor` object represents a linear Gabor filter that is sensitive to textures with a specified wavelength and orientation.

You can use the `gabor` function to create a single Gabor filter or a Gabor filter bank. A filter bank is a set of filters that represent combinations of multiple wavelengths, orientations, and other optional parameters. For example, if you specify two wavelengths and three orientations, then the Gabor filter bank consists of six filters for each combination of wavelength and orientation.

To apply a Gabor filter or a Gabor filter bank to an image, use the `imgaborfilt` function.

Creation

Syntax

```
g = gabor(Wavelength,Orientation)
g = gabor(wavelength,orientation,Name,Value)
```

Description

`g = gabor(Wavelength,Orientation)` creates a Gabor filter and sets the Wavelength and Orientation properties with the wavelength and orientation of the filter.

If you specify `Wavelength` or `Orientation` as vectors, then the `gabor` function creates an array of `gabor` objects that contain all the unique combinations of `Wavelength` and `Orientation`.

`g = gabor(wavelength,orientation,Name,Value)` also uses name-value pairs to set one or both of the SpatialFrequencyBandwidth and SpatialAspectRatio properties. You can specify multiple name-value pairs. Enclose each property name in quotes.

If you specify `SpatialFrequencyBandwidth` or `SpatialAspectRatio` as vectors, then the `gabor` function creates an array of `gabor` objects that represent all combinations of the input argument values.

Example: `gabor(wavelength,orientation,'SpatialFrequencyBandwidth',2)` creates a Gabor filter with a spatial frequency bandwidth of two octaves.

Properties

`Wavelength` — Wavelength of sinusoidal carrier
numeric scalar | numeric vector

Wavelength of the sinusoidal carrier, specified as a numeric scalar or numeric vector with values greater than or equal to 2, in pixels/cycle. Typical values of `Wavelength` range from 2 up to the hypotenuse length of the input image [1].

You cannot change the `Wavelength` property after creating the `gabor` object.

`Orientation` — Orientation of filter in degrees
numeric scalar | numeric vector

Orientation of filter in degrees, specified as a numeric scalar or numeric vector with values in the range [0, 360]. The orientation is defined as the normal direction to the sinusoidal plane wave.

If you are interested in only the Gabor magnitude response, then restrict the range of `Orientation` to [0, 180].

You cannot change the `Orientation` property after creating the `gabor` object.

`SpatialFrequencyBandwidth` — Spatial-frequency bandwidth
`1` (default) | positive number | vector of positive numbers

Spatial-frequency bandwidth in units of octaves, specified as a positive number or a vector of positive numbers. The spatial-frequency bandwidth determines the cutoff of the filter response as frequency content in the input image varies from the preferred frequency, *1/lambda*. Typical values for the spatial-frequency bandwidth are in the range [0.5, 2.5].

You cannot change the `SpatialFrequencyBandwidth` property after creating the `gabor` object.

`SpatialAspectRatio` — Ratio of semimajor and semiminor axes of Gaussian envelope
`0.5` (default) | positive number | vector of positive numbers

Ratio of the semimajor and semiminor axes of the Gaussian envelope (*semiminor*/*semimajor*), specified as a positive number or a vector of positive numbers. This property controls the ellipticity of the Gaussian envelope. Typical values for spatial aspect ratio are in the range [0.23, 0.92].

You cannot change the `SpatialAspectRatio` property after creating the `gabor` object.

`SpatialKernel` — Spatial kernel
numeric matrix

This property is read-only.

Spatial kernel, specified as a numeric matrix.

Examples

Construct Gabor Filter Array and Apply to Input Image

Create a sample image of a checkerboard.

```
A = checkerboard(20);
```

Create an array of Gabor filters.

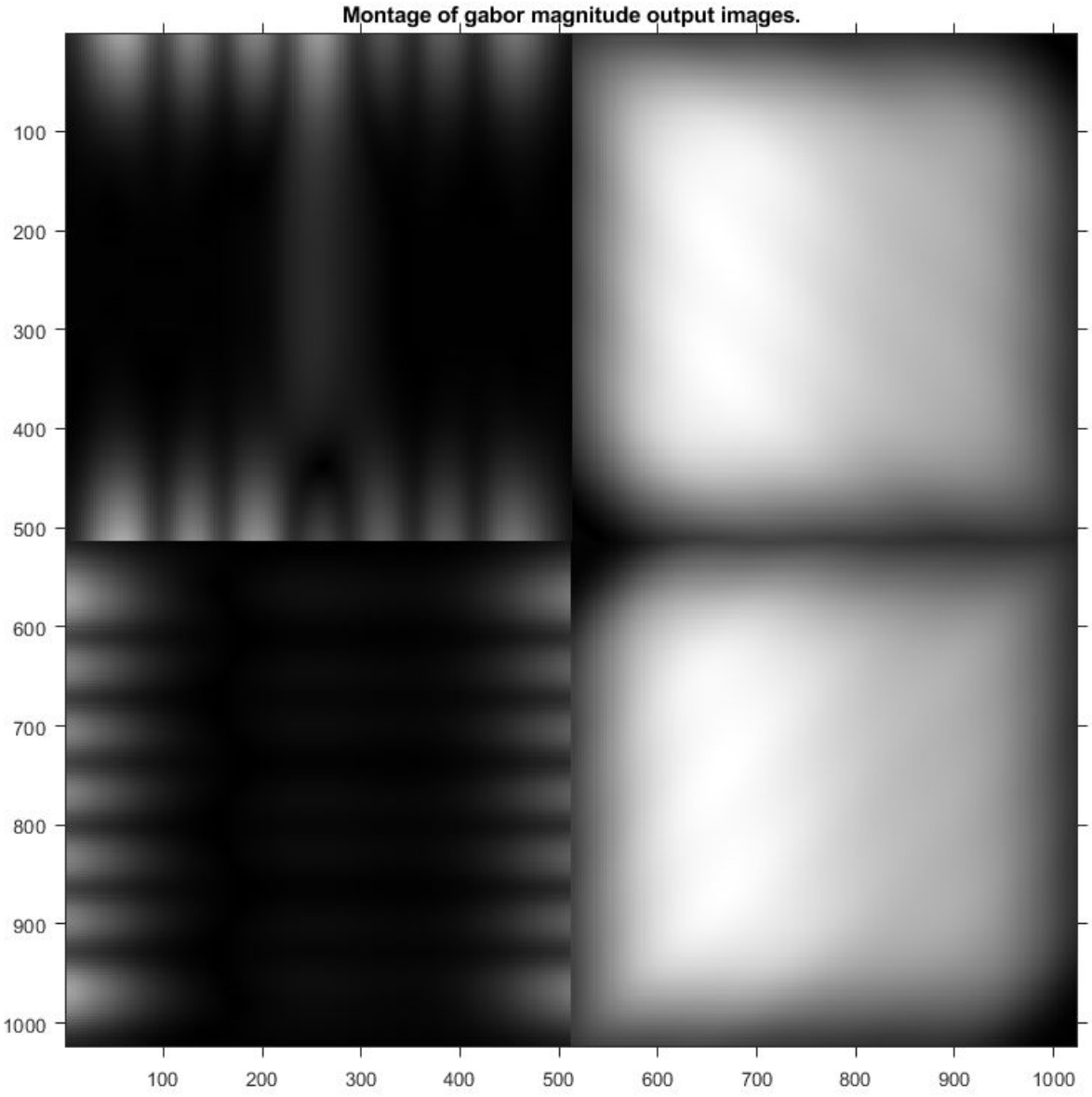
```
wavelength = 20;
orientation = [0 45 90 135];
g = gabor(wavelength,orientation);
```

Apply the filters to the checkerboard image.

```
outMag = imgaborfilt(A,g);
```

Display the results.

```
outSize = size(outMag);
outMag = reshape(outMag,[outSize(1:2),1,outSize(3)]);
figure, montage(outMag,'DisplayRange',[]);
title('Montage of gabor magnitude output images.');
```



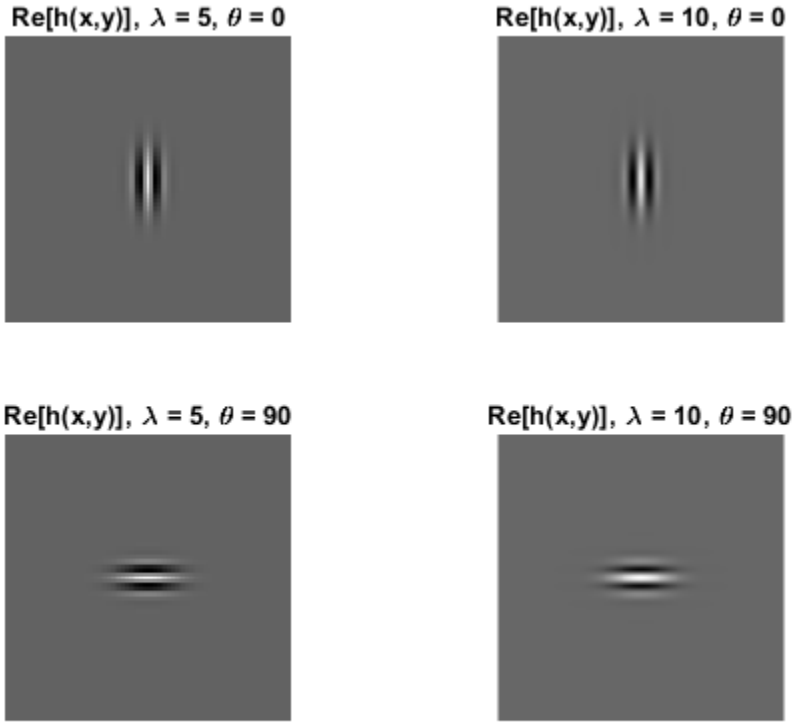
Construct Gabor Filter Array and Visualize Wavelength and Orientation

Create array of Gabor filters.

```
g = gabor([5 10],[0 90]);
```

Visualize the real part of the spatial convolution kernel of each Gabor filter in the array.

```
figure;
subplot(2,2,1)
for p = 1:length(g)
    subplot(2,2,p);
    imshow(real(g(p).SpatialKernel),[]);
    lambda = g(p).Wavelength;
    theta  = g(p).Orientation;
    title(sprintf('Re[h(x,y)], \\lambda = %d, \\theta = %d',lambda,theta));
end
```



References

[1] Jain, Anil K., and Farshid Farrokhnia. "Unsupervised Texture Segmentation Using Gabor Filters." *Pattern Recognition* 24, no. 12 (January 1991): 1167–86. https://doi.org/10.1016/0031-3203(91)90143-S.

See Also

`imgaborfilt`

Topics

“Texture Segmentation Using Gabor Filters”

Introduced in R2015b

geometricTransform2d

2-D geometric transformation object

Description

A `geometricTransform2d` object defines a custom 2-D geometric transformation using point-wise mapping functions.

Creation

Syntax

```
tform = geometricTransform2d(inverseFcn)
tform = geometricTransform2d(inverseFcn,forwardFcn)
```

Description

`tform = geometricTransform2d(inverseFcn)` creates a `geometricTransform2d` object and sets the inverse mapping `InverseFcn` property.

`tform = geometricTransform2d(inverseFcn,forwardFcn)` also sets the forward mapping property, `ForwardFcn`.

Properties

`InverseFcn` — Inverse mapping function
function handle

Inverse mapping function, specified as a function handle. The function should accept and return coordinates as a *n*-by-2 numeric matrix representing the packed (*x*,*y*) coordinates of *n* points.

For more information about function handles, see “Create Function Handle”.

Example: `ifcn = @(xy) [xy(:,1).^2, sqrt(xy(:,2))];`

`ForwardFcn` — Forward mapping function
function handle

Forward mapping function, specified as a function handle. The function should accept and return coordinates as a *n*-by-2 numeric matrix representing the packed (*x*,*y*) coordinates of *n* points.

For more information about function handles, see “Create Function Handle”.

Example: `ffcn = @(xy) [sqrt(xy(:,1)),(xy(:,2).^2)];`

Object Functions

transformPointsForward Apply forward geometric transformation

transformPointsInverse Apply inverse geometric transformation

Examples

Transform Packed Coordinates Using Custom 2-D Transformation

Specify the packed (x,y) coordinates of five input points. The packed coordinates are stored in a 5-by-2 matrix, where the *x*-coordinate of each point is in the first column, and the *y*-coordinate of each point is in the second column.

```
XY = [10 15;11 32;15 34;2 7;2 10];
```

Define the inverse mapping function. The function accepts and returns points in packed (x,y) format.

```
inversefn = @(c) [c(:,1)+c(:,2),c(:,1)-c(:,2)]
```

```
inversefn = function_handle with value:
    @(c)[c(:,1)+c(:,2),c(:,1)-c(:,2)]
```

Create a 2-D geometric transform object, `tform`, that stores the inverse mapping function.

```
tform = geometricTransform2d(inversefn)
```

```
tform =
  geometricTransform2d with properties:

        InverseFcn: @(c)[c(:,1)+c(:,2),c(:,1)-c(:,2)]
        ForwardFcn: []
    Dimensionality: 2
```

Apply the inverse geometric transform to the input points.

```
UV = transformPointsInverse(tform,XY)
```

```
UV = 5×2

    25    -5
    43   -21
    49   -19
     9    -5
    12    -8
```

Transform Coordinate Arrays Using Custom 2-D Transformation

Specify the x- and y-coordinates vectors of five points to transform.

```
x = [10 11 15 2 2];
y = [15 32 34 7 10];
```

Define the inverse and forward mapping functions. Both functions accept and return points in packed (x,y) format.

```
inversefn = @(c) [c(:,1).^2,sqrt(c(:,2))];
forwardfn = @(c) [sqrt(c(:,1)),c(:,2).^2];
```

Create a 2-D geometric transform object, `tform`, that stores the inverse mapping function and the optional forward mapping function.

```
tform = geometricTransform2d(inversefn,forwardfn)
```

```
tform =
  geometricTransform2d with properties:

       InverseFcn: @(c)[c(:,1).^2,sqrt(c(:,2))]
       ForwardFcn: @(c)[sqrt(c(:,1)),c(:,2).^2]
    Dimensionality: 2
```

Apply the inverse geometric transform to the input points.

```
[u,v] = transformPointsInverse(tform,x,y)
```

```
u = 1×5

   100   121   225     4     4


v = 1×5

    3.8730    5.6569    5.8310    2.6458    3.1623
```

Apply the forward geometric transform to the transformed points `u` and `v`.

```
[x,y] = transformPointsForward(tform,u,v)
```

```
x = 1×5

    10    11    15     2     2


y = 1×5

   15.0000   32.0000   34.0000    7.0000   10.0000
```

Transform Grayscale Image Using Custom 2-D Transformation

Define an inverse mapping function that applies anisotropic scaling. The function must accept and return packed (x,y) coordinates, where the *x*-coordinate of each point is in the first column, and the *y*-coordinate of each point is in the second column.

```
xscale = 0.3;
yscale = 0.5;
inversefn = @(xy) [xscale*xy(:,1), yscale*xy(:,2)];
```

Create a 2-D geometric transform object, `tform`, that stores the inverse mapping function.

```
tform = geometricTransform2d(inversefn)
```

```
tform =
  geometricTransform2d with properties:

        InverseFcn: @(xy)[xscale*xy(:,1),yscale*xy(:,2)]
        ForwardFcn: []
    Dimensionality: 2
```

Read an image to be transformed.

```
I = imread('cameraman.tif');
imshow(I)
```

Use `imwarp` to apply the inverse geometric transform to the input image. The image is enlarged vertically by a factor of 2 (the inverse of `yscale`) and horizontally by a factor of 10/3 (the inverse of `xscale`).

```
Itransformed = imwarp(I,tform);
imshow(Itransformed)
```

Transform Color Image Using Custom 2-D Transformation

Define an inverse mapping function that accepts packed (x,y) coordinates, where the *x*-coordinate of each point is in the first column, and the *y*-coordinate of each point is in the second column. The inverse mapping function in this example takes the square of the polar radial component.

```
r = @(c) sqrt(c(:,1).^2 + c(:,2).^2);
w = @(c) atan2(c(:,2), c(:,1));
f = @(c) [r(c).^2 .* cos(w(c)), r(c).^2 .* sin(w(c))];
g = @(c) f(c);
```

Create a 2-D geometric transform object, `tform`, that stores the inverse mapping function.

```
tform = geometricTransform2d(g);
```

Read a color image to be transformed.

```
I = imread('peppers.png');
imshow(I)
```

Create an `imref2d` object, specifying the size and world limits of the input and output images.

```
Rin = imref2d(size(I),[-1 1],[-1 1]);
Rout = imref2d(size(I),[-1 1],[-1 1]);
```

Apply the inverse geometric transform to the input image.

```
Itransformed = imwarp(I,Rin,tform,'OutputView',Rout);
imshow(Itransformed)
```

See Also

`affine2d` | `rigid2d` | `projective2d` | `imwarp` | `geometricTransform3d`

Topics

“2-D and 3-D Geometric Transformation Process Overview”

Introduced in R2018b

geometricTransform3d

3-D geometric transformation object

Description

A `geometricTransform3d` object defines a custom 3-D geometric transformation using point-wise mapping functions.

Creation

Syntax

```
tform = geometricTransform3d(inverseFcn)
tform = geometricTransform3d(inverseFcn,forwardFcn)
```

Description

`tform = geometricTransform3d(inverseFcn)` creates a `geometricTransform3d` object and sets the value of inverse mapping function property, `InverseFcn` to `inverseFcn`.

`tform = geometricTransform3d(inverseFcn,forwardFcn)` also sets the value of forward mapping function property, `ForwardFcn` to `forwardFcn`.

Properties

`InverseFcn` — Inverse mapping function
function handle

Inverse mapping function, specified as a function handle. The function must accept and return coordinates as an *n*-by-3 numeric matrix representing the packed (*x*,*y*,*z*) coordinates of *n* points.

For more information about function handles, see “Create Function Handle”.

Example: `ifcn = @(xyz) [xyz(:,1).^2,xyz(:,2).^2,xyz(:,3).^2];`

`ForwardFcn` — Forward mapping function
function handle

Forward mapping function, specified as a function handle. The function must accept and return coordinates as an *n*-by-3 numeric matrix representing the packed (*x*,*y*,*z*) coordinates of *n* points.

For more information about function handles, see “Create Function Handle”.

Example: `ffcn = @(xyz) [sqrt(xyz(:,1)),sqrt(xyz(:,2)),sqrt(xyz(:,3))];`

Object Functions

transformPointsForward Apply forward geometric transformation

transformPointsInverse Apply inverse geometric transformation

Examples

Transform Packed Coordinates Using Custom 3-D Transformation

Specify the packed (*x*,*y*,*z*) coordinates of five input points. The packed coordinates are stored as a 5-by-3 matrix, where the first, second, and third columns contain the *x*-, *y*-, and *z*-coordinates,respectively.

```
XYZ = [5 25 20;10 5 25;15 10 5;20 15 10;25 20 15];
```

Define an inverse mapping function that accepts and returns points in packed (*x*,*y*,*z*) format.

```
inverseFcn = @(c) [c(:,1)+c(:,2),c(:,1)-c(:,2),c(:,3).^2];
```

Create a 3-D geometric transformation object, `tform`, that stores this inverse mapping function.

```
tform = geometricTransform3d(inverseFcn)
```

```
tform =
  geometricTransform3d with properties:

        InverseFcn: @(c)[c(:,1)+c(:,2),c(:,1)-c(:,2),c(:,3).^2]
        ForwardFcn: []
    Dimensionality: 3
```

Apply the inverse transformation of this 3-D geometric transformation to the input points.

```
UVW = transformPointsInverse(tform,XYZ)
```

```
UVW = 5×3

    30   -20   400
    15     5   625
    25     5    25
    35     5   100
    45     5   225
```

Transform Coordinate Arrays Using Custom 3-D Transformation

Specify the *x*-, *y*- and the *z*-coordinate vectors of five points to transform.

```
x = [3 5 7 9 11];
y = [2 4 6 8 10];
z = [5 9 13 17 21];
```

Define the inverse and forward mapping functions that accept and return points in packed (*x*,*y*,*z*) format.

```
inverseFcn = @(c)[c(:,1).^2,c(:,2).^2,c(:,3).^2];
forwardFcn = @(c)[sqrt(c(:,1)),sqrt(c(:,2)),sqrt(c(:,3))];
```

Create a 3-D geometric transformation object, `tform`, that stores these inverse and forward mapping functions.

```
tform = geometricTransform3d(inverseFcn,forwardFcn)
```

```
tform =
  geometricTransform3d with properties:

        InverseFcn: @(c)[c(:,1).^2,c(:,2).^2,c(:,3).^2]
        ForwardFcn: @(c)[sqrt(c(:,1)),sqrt(c(:,2)),sqrt(c(:,3))]
    Dimensionality: 3
```

Apply the inverse transformation of this 3-D geometric transformation to the input points.

```
[u,v,w] = transformPointsInverse(tform,x,y,z)
```

```
u = 1×5

     9    25    49    81   121


v = 1×5

     4    16    36    64   100


w = 1×5

    25    81   169   289   441
```

Apply the forward geometric transform to the transformed points `u`, `v`, and `w`.

```
[x,y,z] = transformPointsForward(tform,u,v,w)
```

```
x = 1×5

     3     5     7     9    11


y = 1×5

     2     4     6     8    10


z = 1×5

     5     9    13    17    21
```

Transform 3-D Volumetric Image Using Custom 3-D Transformation

Define an inverse mapping function that performs reflection about horizontal axis. The function must accept and return packed (*x*,*y*,*z*) coordinates, where the first, second, and third columns contain the *x*-, *y*-, and *z*-coordinates, respectively.

```
inverseFcn = @(xyz)[xyz(:,1),-xyz(:,2),xyz(:,3)];
```

Create a 3-D geometric transformation object, `tform`, that stores this inverse mapping function.

```
tform = geometricTransform3d(inverseFcn)
```

```
tform =
  geometricTransform3d with properties:

        InverseFcn: @(xyz)[xyz(:,1),-xyz(:,2),xyz(:,3)]
        ForwardFcn: []
    Dimensionality: 3
```

Load and display an MRI volume to be transformed.

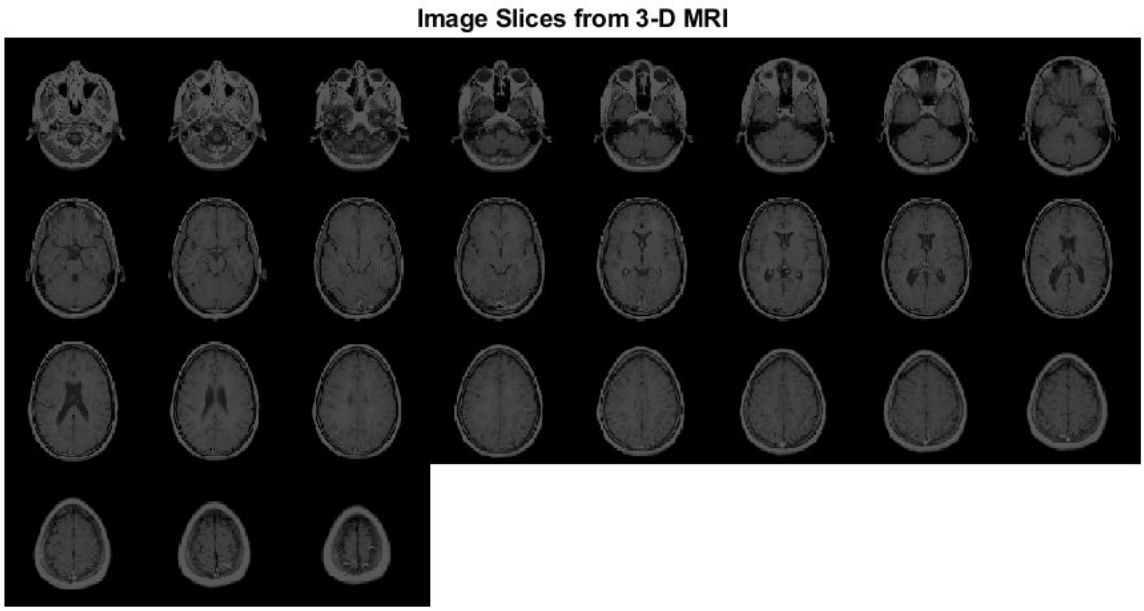
```
s = load('mri');
mriVolume = squeeze(s.D);
```

Use `imwarp` to apply the inverse geometric transform to the input MRI volume.

```
[mriVolumeTransformed] = imwarp(mriVolume,tform,'nearest','SmoothEdges',true);
```

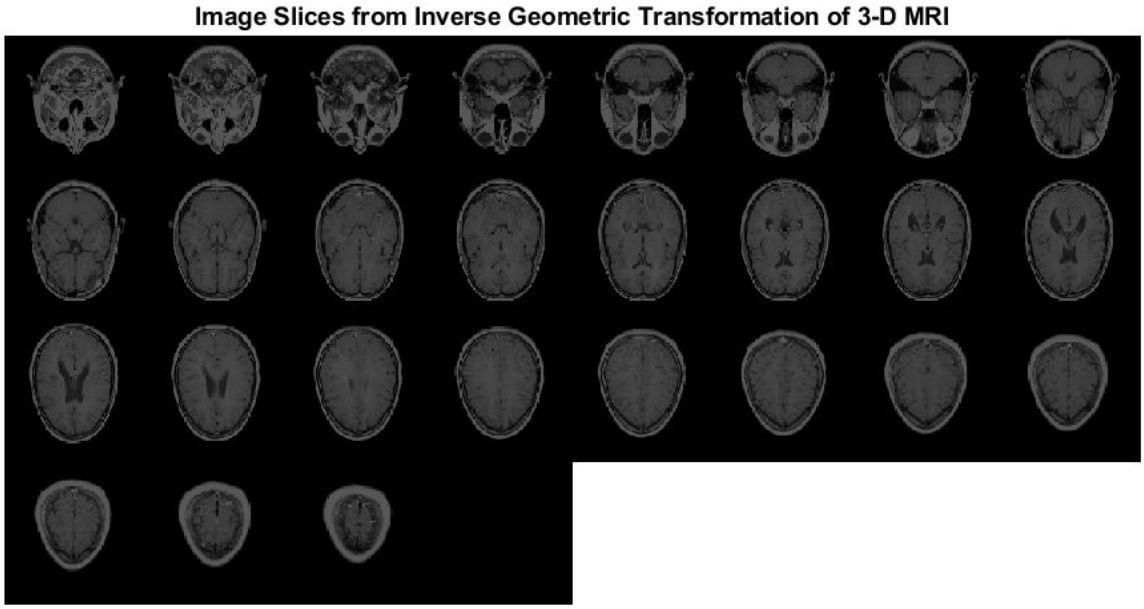
Display the image slices from the input MRI volume as montage.

```
montage(mriVolume,'Size',[4 8],'BackgroundColor','w')
title('Image Slices from 3-D MRI','FontSize',14)
```



Display the image slices from the transformed MRI volume as a montage. The transformed image slices are the reflection of the input image slices across the *x*-axis.

```
montage(mriVolumeTransformed,'Size',[4 8],'BackgroundColor','w')
title('Image Slices from Inverse Geometric Transformation of 3-D MRI','FontSize',14)
```



See Also

`affine3d` | `rigid3d` | `geometricTransform2d` | `imwarp`

Topics

"2-D and 3-D Geometric Transformation Process Overview"

Introduced in R2018b

LocalWeightedMeanTransformation2D

2-D local weighted mean geometric transformation

Description

A `LocalWeightedMeanTransformation2D` object encapsulates a 2-D local weighted mean geometric transformation.

Creation

You can create a `LocalWeightedMeanTransformation2D` object using the following methods:

- The `fitgeotrans` function, which estimates a geometric transformation that maps pairs of control points between two images.
- The `images.geotrans.LocalWeightedMeanTransformation2D` described here. This function creates a `LocalWeightedMeanTransformation2D` object using coordinates of fixed points and moving points, and a specified number of points to use in the local weighted mean calculation.

Syntax

```
tform = images.geotrans.LocalWeightedMeanTransformation2D(movingPoints,
fixedPoints,n)
```

Description

`tform = images.geotrans.LocalWeightedMeanTransformation2D(movingPoints, fixedPoints,n)` creates a `LocalWeightedMeanTransformation2D` object given control point coordinates in `movingPoints` and `fixedPoints`, which define matched control points in the moving and fixed images, respectively. The `n` closest points are used to infer a second degree polynomial transformation for each control point pair.

Input Arguments

`movingPoints` — *x*- and *y*-coordinates of control points in the moving image
m-by-2 matrix

x- and *y*-coordinates of control points in the moving image, specified as an *m*-by-2 matrix. The number of control points *m* must be greater than or equal to `n`.

Data Types: `double` | `single`

`fixedPoints` — *x*- and *y*-coordinates of control points in the fixed image
m-by-2 matrix

x- and *y*-coordinates of control points in the fixed image, specified as an *m*-by-2 matrix. The number of control points *m* must be greater than or equal to `n`.

Data Types: `double` | `single`

n — Number of points to use in local weighted mean calculation

numeric value

Number of points to use in local weighted mean calculation, specified as a numeric value. `n` can be as small as 6, but making `n` small risks generating ill-conditioned polynomials

Data Types: `double` | `single` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32`

Properties

Dimensionality — Dimensionality of the geometric transformation

2

Dimensionality of the geometric transformation for both input and output points, specified as the value 2.

Object Functions

outputLimits	Find output spatial limits given input spatial limits
transformPointsInverse	Apply inverse geometric transformation

Examples

Fit Set of Fixed and Moving Control Points Using Second Degree Polynomial

Fit a local weighted mean transformation to a set of fixed and moving control points that are actually related by a global second degree polynomial transformation across the entire plane.

Set up variables.

```
x = [10, 12, 17, 14, 7, 10];
y = [8, 2, 6, 10, 20, 4];

a = [1 2 3 4 5 6];
b = [2.3 3 4 5 6 7.5];

u = a(1) + a(2).*x + a(3).*y + a(4) .*x.*y + a(5).*x.^2 + a(6).*y.^2;
v = b(1) + b(2).*x + b(3).*y + b(4) .*x.*y + b(5).*x.^2 + b(6).*y.^2;

movingPoints = [u',v'];
fixedPoints = [x',y'];
```

Fit local weighted mean transformation to points.

```
tformLocalWeightedMean = images.geotrans.LocalWeightedMeanTransformation2D(movingPoints,fixedPoi
```

Verify the fit of the `LocalWeightedMeanTransformation2D` object at the control points.

```
movingPointsComputed = transformPointsInverse(tformLocalWeightedMean,fixedPoints);
```

```
errorInFit = hypot(movingPointsComputed(:,1)-movingPoints(:,1),...
                       movingPointsComputed(:,2)-movingPoints(:,2))
```

Algorithms

The local weighted mean transformation infers a polynomial at each control point using neighboring control points. The mapping at any location depends on a weighted average of these polynomials. The `n` closest points are used to infer a second degree polynomial transformation for each control point pair. `n` can be as small as 6, but making it small risks generating ill-conditioned polynomials.

See Also

Functions
`imwarp` | `fitgeotrans` | `cpselect`

Objects
`affine2d` | `projective2d` | `PiecewiseLinearTransformation2D` | `PolynomialTransformation2D`

Introduced in R2013b

PiecewiseLinearTransformation2D

2-D piecewise linear geometric transformation

Description

A `PiecewiseLinearTransformation2D` object encapsulates a 2-D piecewise linear geometric transformation.

Creation

You can create a `PiecewiseLinearTransformation2D` object using the following methods:

- The `fitgeotrans` function, which estimates a geometric transformation that maps pairs of control points between two images.
- The `images.geotrans.PiecewiseLinearTransformation2D` function described here. This function creates a `PiecewiseLinearTransformation2D` object using coordinates of fixed points and moving points.

Syntax

```
tform = images.geotrans.PiecewiseLinearTransformation2D(movingPoints,
fixedPoints)
```

Description

`tform = images.geotrans.PiecewiseLinearTransformation2D(movingPoints, fixedPoints)` creates a `PiecewiseLinearTransformation2D` object given control point coordinates in `movingPoints` and `fixedPoints`, which define matched control points in the moving and fixed images, respectively.

Input Arguments

`movingPoints` — *x*- and *y*-coordinates of control points in the moving image
m-by-2 matrix

x- and *y*-coordinates of control points in the moving image, specified as an *m*-by-2 matrix. The number of control points *m* must be greater than or equal to `n`.

Data Types: `double` | `single`

`fixedPoints` — *x*- and *y*-coordinates of control points in the fixed image
m-by-2 matrix

x- and *y*-coordinates of control points in the fixed image, specified as an *m*-by-2 matrix. The number of control points *m* must be greater than or equal to `n`.

Data Types: `double` | `single`

Properties

`Dimensionality` — Dimensionality of the geometric transformation

2

Dimensionality of the geometric transformation for both input and output points, specified as the value 2.

Object Functions

outputLimits	Find output spatial limits given input spatial limits
transformPointsInverse	Apply inverse geometric transformation

Examples

Fit Set of Control Points Related by Affine Transformation

Fit a piecewise linear transformation to a set of fixed and moving control points that are actually related by a single global affine2d transformation across the domain.

Create a 2D affine transformation.

```
theta = 10;
tformAffine = affine2d([cosd(theta) -sind(theta) 0; sind(theta) cosd(theta) 0; 0 0 1])
```

```
tformAffine =

  affine2d with properties:

                T: [3x3 double]
    Dimensionality: 2
```

Arbitrarily choose 6 pairs of control points.

```
fixedPoints = [10 20; 10 5; 2 3; 0 5; -5 3; -10 -20];
```

Apply forward geometric transformation to map fixed points to obtain effect of fixed and moving points that are related by some geometric transformation.

```
movingPoints = transformPointsForward(tformAffine,fixedPoints)
```

```
movingPoints =

   13.3210   17.9597
   10.7163    3.1876
    2.4906    2.6071
    0.8682    4.9240
   -4.4031    3.8227
  -13.3210  -17.9597
```

Estimate piecewise linear transformation that maps `movingPoints` to `fixedPoints`.

```
tformPiecewiseLinear = images.geotrans.PiecewiseLinearTransformation2D(movingPoints,fixedPoints)
```

```
tformPiecewiseLinear =
```

```
  PiecewiseLinearTransformation2D with properties:

    Dimensionality: 2
```

Verify the fit of the `PiecewiseLinearTransformation2D` object at the control points.

```
movingPointsComputed = transformPointsInverse(tformPiecewiseLinear,fixedPoints);

errorInFit = hypot(movingPointsComputed(:,1)-movingPoints(:,1),...
                   movingPointsComputed(:,2)-movingPoints(:,2))
```

```
errorInFit =

   1.0e-15 *

         0
         0
    0.4441
         0
         0
         0
```

See Also

Functions
`imwarp` | `fitgeotrans` | `cpselect`

Classes
`affine2d` | `projective2d` | `LocalWeightedMeanTransformation2D` | `PolynomialTransformation2D`

Introduced in R2013b

PolynomialTransformation2D

2-D polynomial geometric transformation

Description

A `PolynomialTransformation2D` object encapsulates a 2-D polynomial geometric transformation.

Creation

You can create a `PolynomialTransformation2D` object using the following methods:

- The `fitgeotrans` function, which estimates a geometric transformation that maps pairs of control points between two images.
- The `images.geotrans.PolynomialTransformation2D` function described here. This function creates a `PolynomialTransformation2D` object using coordinates of fixed points and moving points, or the known polynomial coefficients for the forward and inverse transformation.

Syntax

```
tform = images.geotrans.PolynomialTransformation2D(movingPoints,fixedPoints,
degree)
tform = images.geotrans.PolynomialTransformation2D(a,b)
```

Description

`tform = images.geotrans.PolynomialTransformation2D(movingPoints,fixedPoints, degree)` creates a `PolynomialTransformation2D` object and sets the `Degree` property. The function estimates the polynomial coefficients `A` and `B` from matrices `movingPoints` and `fixedPoints` that define matched control points in the moving and fixed images, respectively.

`tform = images.geotrans.PolynomialTransformation2D(a,b)` creates a `PolynomialTransformation2D` object and sets the `A` and `B` properties.

Input Arguments

`movingPoints` — *x*- and *y*-coordinates of control points in the moving image
m-by-2 matrix

x- and *y*-coordinates of control points in the moving image, specified as an *m*-by-2 matrix.

Data Types: `double` | `single`

`fixedPoints` — *x*- and *y*-coordinates of control points in the fixed image
m-by-2 matrix

x- and *y*-coordinates of control points in the fixed image, specified as an *m*-by-2 matrix.

Data Types: `double` | `single`

Properties

A — Polynomial coefficients used to determine *U* in the inverse transformation

n-element vector

Polynomial coefficients used to determine *U* in the inverse transformation, specified as an *n*-element vector. For polynomials of degree 2, 3, and 4, *n* is 6, 10, and 15, respectively.

The quadratic (degree 2) polynomial coefficient vector `A` is ordered as follows:

```
U = A(1) + A(2).*X + A(3).*Y + A(4).*X.*Y + A(5).*X.^2 + A(6).*Y.^2
```

The cubic (degree 3) polynomial coefficient vector adds these terms:

```
... + A(7).*X.^2.*Y + A(8).*X.*Y.^2 + A(9).*X.^3 + A(10).*Y.^3
```

The quartic (degree 4) polynomial coefficient vector adds these terms:

```
... + A(11).*X.^3.*Y + A(12).*X.^2.*Y.^2 + A(12).*X.*Y.^3 + A(14).*X.^3 + A(15).*Y.^4
```

Data Types: `double` | `single`

B — Polynomial coefficients used to determine *V* in the inverse transformation

n-element vector

Polynomial coefficients used to determine *V* in the inverse transformation, specified as an *n*-element vector. For polynomials of degree 2, 3, and 4, *n* is 6, 10, and 15, respectively.

The quadratic (degree 2) polynomial coefficient vector `B` is ordered as follows:

```
V = B(1) + B(2).*X + B(3).*Y + B(4).*X.*Y + B(5).*X.^2 + B(6).*Y.^2
```

The cubic (degree 3) polynomial coefficient vector adds these terms:

```
... + B(7).*X.^2.*Y + B(8).*X.*Y.^2 + B(9).*X.^3 + B(10).*Y.^3
```

The quartic (degree 4) polynomial coefficient vector adds these terms:

```
... + B(11).*X.^3.*Y + B(12).*X.^2.*Y.^2 + B(12).*X.*Y.^3 + B(14).*X.^3 + B(15).*Y.^4
```

Data Types: `double` | `single`

Degree — Degree of the polynomial transformation

2 | 3 | 4

Degree of the polynomial transformation, specified as the scalar values `2`, `3`, or `4`.

Dimensionality — Dimensionality of the geometric transformation

2

Dimensionality of the geometric transformation for both input and output points, specified as the value `2`.

Object Functions

outputLimits	Find output spatial limits given input spatial limits
transformPointsInverse	Apply inverse geometric transformation

Examples

Fit a Second Degree Polynomial Transformation to a Set of Fixed and Moving Control Points

Fit a second degree polynomial transformation to a set of fixed and moving control points that are actually related by an 2-D affine transformation.

Create 2-D affine transformation.

```
theta = 10;
tformAffine = affine2d([cosd(theta) -sind(theta) 0; sind(theta) cosd(theta) 0; 0 0 1]);
```

Arbitrarily choose six pairs of control points. A second degree polynomial requires six pairs of control points.

```
 fixedPoints = [10 20; 10 5; 2 3; 0 5; -5 3; -10 -20];
```

Apply forward geometric transformation to map fixed points to obtain effect of fixed and moving points that are related by some geometric transformation.

```
movingPoints = transformPointsForward(tformAffine,fixedPoints);
```

Estimate second degree `PolynomialTransformation2D` transformation that fits `fixedPoints` and `movingPoints`.

```
tformPolynomial = images.geotrans.PolynomialTransformation2D(movingPoints,fixedPoints,2);
```

Verify the fit of the `PolynomialTransformation2D` transformation at the control points.

```
movingPointsEstimated = transformPointsInverse(tformPolynomial,fixedPoints);
errorInFit = hypot(movingPointsEstimated(:,1)-movingPoints(:,1),...
                   movingPointsEstimated(:,2)-movingPoints(:,2))
```

More About

U and *V*

U and *V* are the *x*- and *y*-coordinates of control points in the original coordinate system. This is the same coordinate system as obtained by performing a forward transformation followed by its inverse transformation.

X and *Y*

X and *Y* are the *x*- and *y*-coordinates of control points in the forward transformed coordinate system.

See Also

Functions
`imwarp` | `fitgeotrans` | `cpselect`

Objects
`affine2d` | `projective2d` | `LocalWeightedMeanTransformation2D` | `PiecewiseLinearTransformation2D`

Introduced in R2013b

getheight

Height of structuring element

Note `getheight` will be removed in a future release. See `strel` for the current list of methods.

Syntax

```
h = getheight(SE)
```

Description

`h = getheight(SE)` returns the height of all neighbors of structuring element `SE`.

Examples

Get Height of Arbitrary Structuring Element

```
se = strel(ones(3,3),magic(3));
getheight(se)
```

```
ans =

     8     1     6
     3     5     7
     4     9     2
```

Input Arguments

SE — Structuring element

`strel` object

Structuring element, specified as a `strel` object.

Output Arguments

h — Height

numeric matrix

Height of the structuring element `SE`, returned as a numeric matrix of the same size as `getnhood(SE)`. For a flat structuring element, `h` is all zeros.

Data Types: `double`

See Also

`strel`

Topics
“Structuring Elements”

Introduced before R2006a

getimage

Image data from axes

Syntax

```
I = getimage(h)
[x,y,I] = getimage(h)
[___,flag] = getimage(h)
[___] = getimage
```

Description

`I = getimage(h)` returns the first image data contained in the graphics object `h`.

`[x,y,I] = getimage(h)` also returns the image extent in the *x* and `y` direction.

`[___,flag] = getimage(h)` also returns a flag that indicates the type of image that `h` contains.

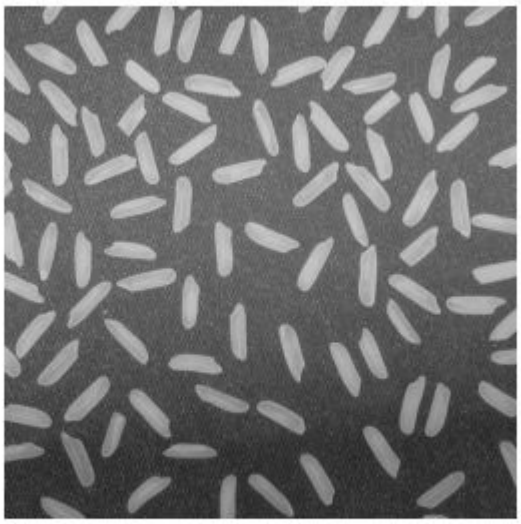
`[___] = getimage` returns information for the current axes object.

Examples

Import Data into Workspace from Image Displayed in Figure or App

Display image directly from a file using `imshow` and create a variable in the workspace that contains the image data.

```
imshow rice.png
```



```
I = getimage;
```

Display image directly from a file using the Image Viewer app (`imtool`) and create a variable in the workspace that contains the image data.

```
h = imtool('cameraman.tif');
```

```
I = getimage(imgca);
```

Input Arguments

h — Handle to graphics object

handle

Handle to a figure, axes, uipanel, or image graphics object, specified as a handle. If `h` is an axes or figure handle containing multiple images, then `getimage` uses the first image returned by `findobj(h,'Type','image')`.

Output Arguments

I — Image data

numeric array

Image data, returned as a numeric array. `I` is identical to the image `CData`; it contains the same values and is of the same class as the image `CData`. If `h` is not an image or does not contain an image, then `I` is empty.

x — Image extent in *x* direction

2-element numeric vector

Image extent in the *x* direction, returned as a 2-element numeric vector of the form `[xmin xmax]`. `x` is identical to the image `XData`.

Data Types: `double`

y — Image extent in *y* direction

2-element numeric vector

Image extent in the *y* direction, returned as a 2-element numeric vector of the form `[ymin ymax]`. `y` is identical to the image `YData`.

Data Types: `double`

flag — Image type

integer

Image type, returned as an integer with one of these values:

Flag	Type of Image
`0`	Not an image; `I` is returned as an empty matrix
`1`	Indexed image
`2`	Intensity image with values in standard range. The standard range for `single` and `double` images is [0,1].
`3`	Intensity data, but not in standard range
`4`	RGB image
`5`	Binary image

Data Types: `double`

See Also

Image Viewer | `imshow`

Introduced before R2006a

getimagemodel

Image model object from image object

Syntax

```
imgmodel = getimagemodel(himage)
```

Description

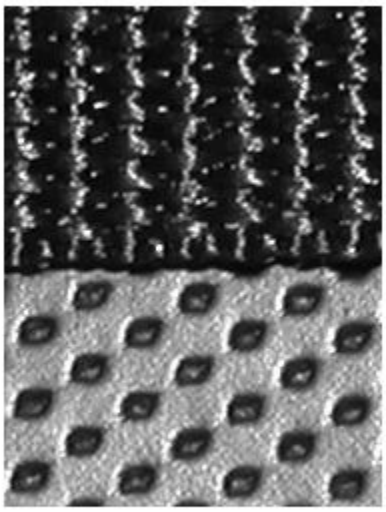
`imgmodel = getimagemodel(himage)` returns the image model object associated with image `himage`. If `himage` does not have an associated image model object, then `getimagemodel` creates one.

Examples

Retrieve imagemodel Object Associated with Image

Read an image into the workspace.

```
h = imshow('bag.png');
```



Retrieve the image model associated with this image.

```
imgmodel = getimagemodel(h)
```

```
imgmodel =
```

```
IMAGEMODEL object accessing an image with these properties:

       ClassType: 'uint8'
    DisplayRange: [0 255]
     ImageHeight: 250
       ImageType: 'intensity'
      ImageWidth: 189
    MinIntensity: 0
    MaxIntensity: 255
```

Input Arguments

himage — Target image

handle | array of handles

Target image, specified as a handle or array of handles to image objects.

Output Arguments

imgmodel — Image model

`imagemodel` object | array of `imagemodel` object

Image model, returned as an `imagemodel` object. If `himage` is an array of handles to image objects, then `imgmodel` is an array of image models.

See Also

`imagemodel` | `imattributes` | `imhandles`

Introduced before R2006a

getline

Select polyline with mouse

Note `getline` is not recommended. Use one of the ROI classes instead, described in “Create ROI Shapes”.

Syntax

```
[xi,yi] = getline
[xi,yi] = getline(fig)
[xi,yi] = getline(ax)
[xi,yi] = getline(___,'closed')
```

Description

`[xi,yi] = getline` lets you select a polyline in the current figure using the mouse. When you finish selecting the polyline, `getline` returns the coordinates of the polyline endpoints in `xi` and `yi`.

Use normal button clicks to add points to the polyline. A shift-, right-, or double-click adds a final point and ends the polyline selection. Pressing **Return** or **Enter** ends the polyline selection without adding a final point. Pressing **Backspace** or **Delete** removes the previously selected point from the polyline.

`[xi,yi] = getline(fig)` lets you select a polyline in the current axes of figure `fig`, using the mouse.

`[xi,yi] = getline(ax)` lets you select a polyline in axes `ax`, using the mouse.

`[xi,yi] = getline(___,'closed')` animates and returns a closed polygon.

Input Arguments

`fig` — Figure handle

handle

Figure handle, specified as a handle.

`ax` — Axes handle

handle

Axes handle, specified as a handle.

Output Arguments

`xi` — *x*-coordinates of selected polyline endpoints

numeric vector

x-coordinates of selected polyline endpoints, returned as a numeric vector.

Data Types: `double`

`yi` — *y*-coordinates of selected polyline endpoints
numeric vector

y-coordinates of selected polyline endpoints, returned as a numeric vector.

Data Types: `double`

See Also

`getpts` | `getrect` | `imline`

Introduced before R2006a

getneighbors

Structuring element neighbor locations and heights

Note `getneighbors` will be removed in a future release. See `strel` for the current list of functions recommended for use with structuring elements.

Syntax

```
[offsets,heights] = getneighbors(SE)
```

Description

`[offsets,heights] = getneighbors(SE)` returns the relative locations and corresponding heights for each of the neighbors in the structuring element `SE`.

Examples

Get Neighbor Location and Height of 2-D Structuring Element

Create a nonflat 2-D structuring element with two neighbors.

```
se = strel("arbitrary",[1 0 1],[5 0 -5]);
```

Get the row and column offset of each neighbor from the center of the structuring element. Also get the heights of the neighbors.

```
[offsets,heights] = getneighbors(se)
```

```
offsets =

     0    -1
     0     1


heights =

     5    -5
```

Input Arguments

SE — Structuring element

`strel` object

Structuring element, specified as a `strel` object with P neighbors and dimensionality N.

Output Arguments

`offsets` — Position of neighbors

P-by-*N* matrix

Position of neighbors relative to the center of the structuring element, in pixels, returned as a *P*-by-*N* matrix.

Data Types: `double`

`heights` — Height of neighbors

P-by-1 column vector

Height of neighbors, returned as a *P*-by-1 column vector.

Data Types: `double`

See Also

Topics

“Structuring Elements”

Introduced before R2006a

getnhood

Get structuring element neighborhood

Note `getnhood` will be removed in a future release. See `strel` for the current list of methods.

Syntax

```
nhood = getnhood(SE)
```

Description

`nhood = getnhood(SE)` returns the neighborhood associated with the structuring element `SE`.

Examples

Get Neighborhood of Flat Structuring Element

```
se = strel(eye(5));
nhood = getnhood(se)
```

```
nhood =

  5×5 logical array

   1   0   0   0   0
   0   1   0   0   0
   0   0   1   0   0
   0   0   0   1   0
   0   0   0   0   1
```

Input Arguments

SE — Structuring element

`strel` object | `offsetstrel` object

Structuring element, specified as a `strel` or `offsetstrel` object.

Output Arguments

nhood — Neighborhood of structuring element

logical array

Neighborhood of structuring element, returned as a logical array.

Note If `SE` is an `offsetstrel` object, then `nhood` indicates which pixels are in the neighborhood but does not return the offset of the pixels. You can get the offset from the property `SE.Offset`.

Data Types: `logical`

See Also

Topics
“Structuring Elements”

Introduced before R2006a

getpts

Specify points with mouse

Note `getpts` is not recommended. Use one of the ROI classes instead, described in “Create ROI Shapes”.

Syntax

```
[xi,yi] = getpts
[xi,yi] = getpts(fig)
[xi,yi] = getpts(ax)
```

Description

`[xi,yi] = getpts` lets you choose points in the current figure using the mouse. When you finish selecting points, `getpts` returns the coordinates of the selected points in `xi` and `yi`.

Use normal button clicks to add points. A shift-, right-, or double-click adds a final point and ends the selection. Pressing **Return** or **Enter** ends the selection without adding a final point. Pressing **Backspace** or **Delete** removes the previously selected point.

`[xi,yi] = getpts(fig)` lets you choose points in the current axes of figure `fig`, using the mouse.

`[xi,yi] = getpts(ax)` lets you choose points in axes `ax`, using the mouse.

Examples

Select Points in Image Interactively

Display an image using `imshow`.

```
figure
imshow('moon.tif')
```

Call `getpts` to choose points interactively in the displayed image using the mouse. Double-click to complete your selection. When you are done, `getpts` returns the coordinates of your points.

```
[x,y] = getpts
```

Input Arguments

`fig` — Figure handle

handle

Figure handle, specified as a handle.

`ax` — Axes handle

handle

Axes handle, specified as a handle.

Output Arguments

`xi` — *x*-coordinates of sampled points

numeric vector

x-coordinates of sampled points, returned as a numeric vector.

Data Types: `double`

`yi` — *y*-coordinates of sampled points

numeric vector

y-coordinates of sampled points, returned as a numeric vector.

Data Types: `double`

See Also

`getline` | `getrect` | `impixel` | `impoint`

Introduced before R2006a

getrect

Specify rectangle with mouse

Note `getrect` is not recommended. Use one of the ROI classes instead, described in “Create ROI Shapes”.

Syntax

```
rect = getrect
rect = getrect(fig)
rect = getrect(ax)
```

Description

`rect = getrect` lets you select a rectangle in the current axes using the mouse. When you finish selecting the rectangle, `getrect` returns information about the position and size of the rectangle in `rect`.

Use the mouse to click and drag the desired rectangle. To constrain the rectangle to be a square, use a shift- or right-click to begin the drag.

`rect = getrect(fig)` lets you select a rectangle in the current axes of figure `fig`, using the mouse.

`rect = getrect(ax)` lets you select a rectangle in axes `ax`, using the mouse.

Examples

Select Rectangle in Image Interactively

Display an image using `imshow`.

```
imshow('moon.tif')
```

Choose points interactively in the displayed image using the mouse. When you are done, `getrect` returns the size and position of your rectangle.

```
rect = getrect
```

Input Arguments

`fig` — Figure handle

handle

Figure handle, specified as a handle.

`ax` — Axes handle

handle

Axes handle, specified as a handle.

Output Arguments

`rect` — Selected rectangle

4-element numeric vector

Selected rectangle, returned as a 4-element numeric vector with the form `[xmin ymin width height]`.

See Also

`getline` | `getpts` | `imrect`

Introduced before R2006a

getsequence

Sequence of decomposed structuring elements

Note `getsequence` will be removed in a future release. See `strel` for the current list of methods.

Syntax

```
SEQ = getsequence(SE)
```

Description

`SEQ = getsequence(SE)` returns the array of structuring elements `SEQ`, containing the individual structuring elements that form the decomposition of `SE`.

Examples

Decompose Square Structuring Element

Create a 5-by-5 structuring element.

```
se = strel("square",5)
```

```
se =

strel is a square shaped structuring element with properties:

      Neighborhood: [5×5 logical]
    Dimensionality: 2
```

The `strel` function uses decomposition for square structuring elements larger than 3-by-3. Use `getsequence` to extract the decomposed structuring elements.

```
seq = getsequence(se)
```

```
seq =

  2×1 strel array with properties:

    Neighborhood
    Dimensionality
```

Use `imdilate` with the `"full"` option to confirm that dilating sequentially with the decomposed structuring elements forms a 5-by-5 square:

```
imdilate(1,seq,"full")
```

```
ans =

     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

```
     1     1     1     1     1
     1     1     1     1     1
```

Input Arguments

SE — Structuring elements

array of `strel` objects

Structuring elements, specified as an array of `strel` objects.

Output Arguments

SEQ — Decomposed structuring elements

array of `strel` objects

Decomposed structuring elements, returned as an array of `strel` objects. The elements of `SEQ` have no further decomposition.

See Also

`strel`

Topics

“Structuring Elements”

Introduced before R2006a

grabcut

Segment image into foreground and background using iterative graph-based segmentation

Syntax

```
BW = grabcut(A,L,ROI)
BW = grabcut(A,L,ROI,foremask,backmask)
BW = grabcut(A,L,ROI,foreind,backind)
BW = grabcut( ___ ,Name,Value)
```

Description

`BW = grabcut(A,L,ROI)` segments the image `A` into foreground and background regions. The label matrix `L` specifies the subregions of the image. `ROI` is a logical mask designating the initial region of interest.

`BW = grabcut(A,L,ROI,foremask,backmask)` segments the image `A`, where `foremask` and `backmask` are masks designating pixels in the image as foreground and background, respectively.

`BW = grabcut(A,L,ROI,foreind,backind)` segments the image `A`, where `foreind` and `backind` specify the linear indices of the pixels in the image marked as foreground and background, respectively.

`BW = grabcut( ___ ,Name,Value)` segments the image using name-value pairs to control aspects of the segmentation.

Examples

Segment Foreground from Background in Image Using Grabcut

Read an RGB image into the workspace.

```
RGB = imread('peppers.png');
```

Generate label matrix.

```
L = superpixels(RGB,500);
```

Specify a region of interest and create a mask image.

```
imshow(RGB)
h1 = drawpolygon('Position',[72,105; 1,231; 0,366; 104,359;...
        394,307; 518,343; 510,39; 149,72]);
```

```
roiPoints = h1.Position;
roi = poly2mask(roiPoints(:,1),roiPoints(:,2),size(L,1),size(L,2));
```

Perform the grab cut operation, specifying the original image, the label matrix and the ROI.

```
BW = grabcut(RGB,L,roi);
imshow(BW)
```

Create masked image.

```
maskedImage = RGB;
maskedImage(repmat(~BW,[1 1 3])) = 0;
imshow(maskedImage)
```

Segment 3-D Volume Using Grabcut

Load 3-D volumetric data.

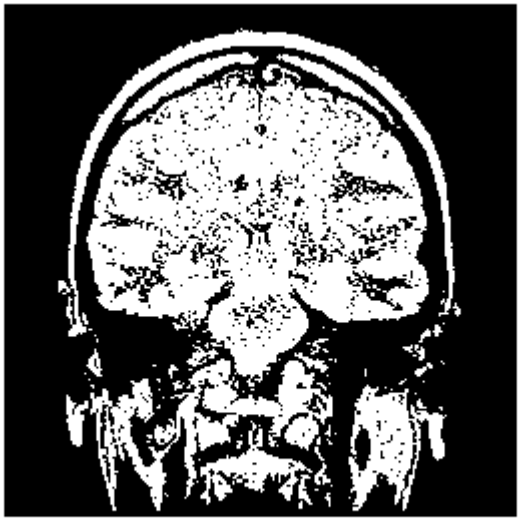
```
load mristack
V = mristack;
```

Create a 2-D mask for initial foreground and background seed points.

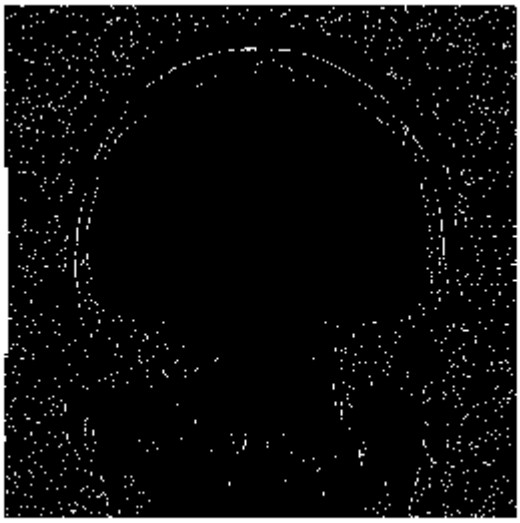
```
seedLevel = 10;
fseed = V(:,:,seedLevel) > 75;
bseed = V(:,:,seedLevel) == 0;
```

Display foreground and background seed points.

```
imshow(fseed)
```



```
imshow(bseed)
```



Place seed points into empty 3-D mask.

```
fmask = zeros(size(V));
bmask = fmask;
```

```
fmask(:,:,seedLevel) = fseed;
bmask(:,:,seedLevel) = bseed;
```

Create initial region of interest.

```
roi = false(size(V));
roi(10:end-10,10:end-10,:) = true;
```

Generate label matrix.

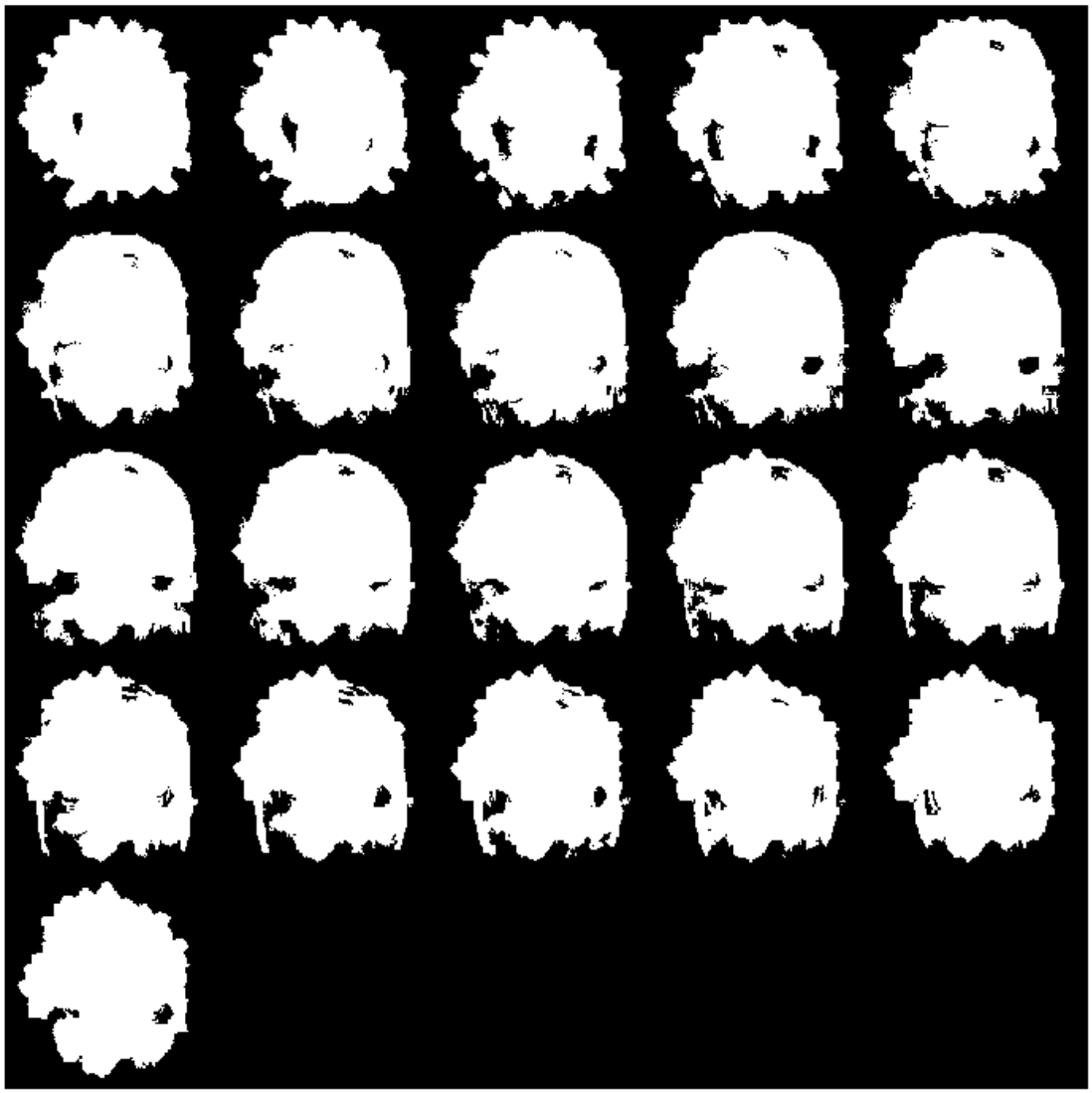
```
L = superpixels3(V,500);
```

Perform GrabCut.

```
bw = grabcut(V,L,roi,fmask,bmask);
```

Display 3D segmented image.

```
montage(reshape(bw,size(V)))
```



Input Arguments

A — Image to segment

2-D grayscale image | 2-D truecolor image | 3-D grayscale volume

Input image or volume, specified as a 2-D grayscale image, 2-D truecolor image, or 3-D grayscale volume. Only grayscale images can be data type `int16`.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

L — Label matrix

numeric array

Label matrix, specified as a numeric array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`ROI` — Region of interest
logical array

Region of interest, specified as a logical array. All pixels that define the region of interest are equal to `true`.

Data Types: `logical`

`foremask` — Foreground mask
logical array

Foreground mask, specified as a logical array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`backmask` — Background mask
logical array

Background mask, specified as a logical array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`foreind` — Indices of pixels in foreground
vector

Indices of pixels in foreground, specified as a vector of linear indices.

Data Types: `double`

`backind` — Indices of pixels in background
vector

Indices of pixels in background, specified as a vector of linear indices.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

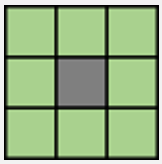
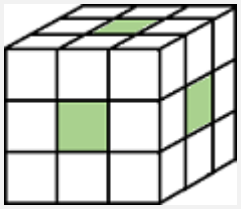
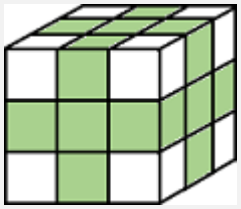
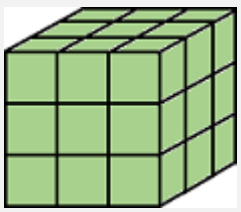
Example: `grabcut(A,L,ROI,Connectivity=4)`

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `grabcut(A,L,ROI,'Connectivity',4)`

`Connectivity` — Connectivity of connected components
4 | 8 | 6 | 18 | 26

Connectivity of connected components, specified as one of the following values. The default connectivity is 8 for 2–D images, and 26 for 3–D images.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. Two adjoining pixels are part of the same object if they are both on and are connected along the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		
6	Pixels are connected if their faces touch. Two adjoining pixels are part of the same object if they are both on and are connected in: • One of these directions: in, out, left, right, up, and down	 Current pixel is center of cube.
18	Pixels are connected if their faces or edges touch. Two adjoining pixels are part of the same object if they are both on and are connected in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. Two adjoining pixels are part of the same object if they are both on and are connected in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`MaximumIterations` — Maximum number of iterations
5 (default) | positive scalar

Maximum number of iterations performed by the algorithm. The algorithm can converge to a solution before reaching the maximum number of iterations.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`BW` — Segmented image
logical array

Segmented image, returned as binary image of the same size as the label matrix `L`.

Tips

- For `double` and `single` images, `grabcut` assumes the range of the image to be `[0 1]`. For `uint16`, `int16`, and `uint8` images, `grabcut` assumes the range to be the full range for the given data type.
- For grayscale images, the size of `L`, `foremask`, and `backmask` must match the size of the image `A`. For color and multi-channel images, `L`, `foremask`, and `backmask` must be 2-D arrays with the first two dimensions identical to the first two dimensions of the image `A`.

Algorithms

- The algorithm treats all subregions fully or spatially outside the ROI mask as belonging to the background. To get an optimal segmentation, make sure the object to be segmented is fully contained within the ROI, surrounded by a small number of background pixels.
- Do not mark a subregion of the label matrix as belonging to both the foreground mask and the background mask. If a region of the label matrix contains pixels belonging to both the foreground mask and background mask, the algorithm treats the region as unmarked.
- The algorithm assumes all subregions outside the region of interest belong to the background. Marking one of these subregions as belonging to foreground or background mask has no effect on the resulting segmentation.

References

[1] Rother, C., V. Kolmogorov, and A. Blake. "GrabCut - Interactive Foreground Extraction using Iterated Graph Cuts". *ACM Transactions on Graphics (SIGGRAPH)*. Vol. 23, Number 3, 2004, pp. 309–314.

See Also

Image Segmenter | `superpixels` | `lazysnapping` | `watershed`

Topics
“Label and Measure Connected Components in a Binary Image”

Introduced in R2018a

gradientweight

Calculate weights for image pixels based on image gradient

Syntax

```
W = gradientweight(I)
W = gradientweight(I,sigma)
W = gradientweight(___,Name,Value)
```

Description

`W = gradientweight(I)` calculates the pixel weight for each pixel in image `I` based on the gradient magnitude at that pixel, and returns the weight array `W`. The weight of a pixel is inversely related to the gradient values at the pixel location. Pixels with small gradient magnitude (smooth regions) have a large weight and pixels with large gradient magnitude (such as on the edges) have a small weight.

`W = gradientweight(I,sigma)` uses `sigma` as the standard deviation for the derivative of Gaussian that is used for computing the image gradient.

`W = gradientweight(___,Name,Value)` returns the weight array `W` using name-value pairs to control aspects of weight computation.

Examples

Segment Image Using Weights Derived from Image Gradient

This example segments an image using the Fast Marching Method based on the weights derived from the image gradient.

Read image and display it.

```
I = imread('coins.png');
imshow(I)
title('Original Image')
```

Compute weights based on image gradient.

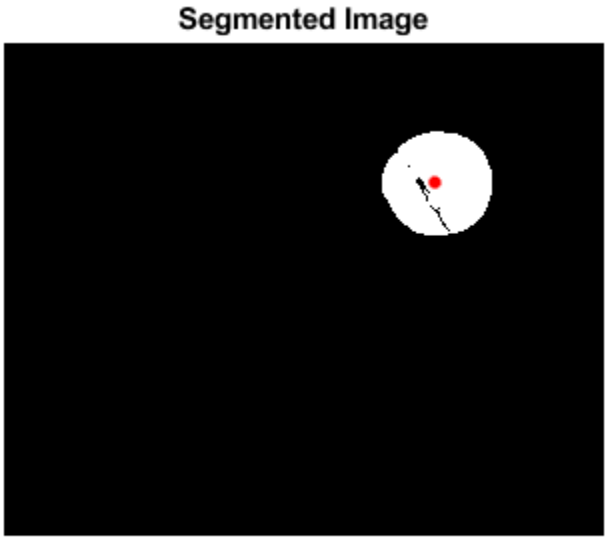
```
sigma = 1.5;
W = gradientweight(I, sigma, 'RolloffFactor', 3, 'WeightCutoff', 0.25);
```

Select a seed location.

```
R = 70; C = 216;
hold on;
plot(C, R, 'r.', 'LineWidth', 1.5, 'MarkerSize',15);
title('Original Image with Seed Location')
```

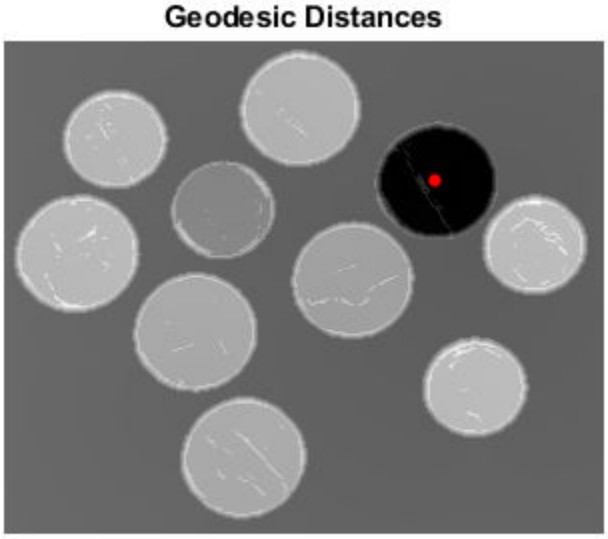
Segment the image using the weight array.

```
thresh = 0.1;
[BW, D] = imsegfmm(W, C, R, thresh);
figure, imshow(BW)
title('Segmented Image')
hold on;
plot(C, R, 'r.', 'LineWidth', 1.5, 'MarkerSize',15);
```



Geodesic distance matrix `D` can be thresholded using different thresholds to get different segmentation results.

```
figure, imshow(D)
title('Geodesic Distances')
hold on;
plot(C, R, 'r.', 'LineWidth', 1.5, 'MarkerSize',15);
```



Input Arguments

`I` — Grayscale image

numeric matrix

Grayscale image, specified as a numeric matrix.

Data Types: `single` | `double` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

`sigma` — Standard deviation for derivative of Gaussian

`1.5` (default) | positive number

Standard deviation for derivative of Gaussian, specified as a positive number.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `W = gradientweight(I,1.5,'RolloffFactor',3,'WeightCutoff',0.25);`

`RolloffFactor` — Output weight roll-off factor
3 (default) | positive scalar

Output weight roll-off factor, specified as the comma-separated pair consisting of `'RolloffFactor'` and a positive scalar of class `double`. Controls how fast weight values fall as a function of gradient magnitude. When viewed as a 2-D plot, pixel intensity values might vary gradually at the edges of regions, creating a gentle slope. In your segmented image, you might want the edge to be more well-defined. Using the roll-off factor, you control the slope of the weight value curve at points where intensity values start to change. If you specify a high value, the output weight values fall off sharply around the edges of smooth regions. If you specify a low value, the output weight has a more gradual fall-off around the edges. The suggested range for this parameter is `[0.5 4]`.

Data Types: `double`

`WeightCutoff` — Threshold for weight values
`0.25` (default) | positive number in the range `[1e-3 1]`

Threshold for weight values, specified as the comma-separated pair consisting of `'WeightCutoff'` and a positive number in the range `[1e-3 1]`. If you use this parameter to set a threshold on weight values, it suppresses any weight values less than the value you specify, setting these pixels to a small constant value (1e-3). This parameter can be useful in improving the accuracy of the output when you use the output weight array `W` as input to Fast Marching Method segmentation function, `imsegfmm`.

Data Types: `double`

Output Arguments

`W` — Weight array
numeric array

Weight array, returned as a numeric array of the same size as the input image, `I`. The weight array is of class `double`, unless `I` is `single`, in which case it is of class `single`.

Tips

- `gradientweight` uses double-precision floating point operations for internal computations for all classes of `I`, except when `I` is of class `single`, in which case `gradientweight` uses single-precision floating point operations internally.

See Also
`imsegfmm` | `graydiffweight`

Introduced in R2014b

gray2ind

Convert grayscale or binary image to indexed image

Syntax

```
[X,cmap] = gray2ind(I,c)
[X,cmap] = gray2ind(BW,c)
```

Description

`[X,cmap] = gray2ind(I,c)` converts the grayscale image `I` to an indexed image `X` with colormap `cmap` with `c` colors.

`[X,cmap] = gray2ind(BW,c)` converts the binary image `BW` to an indexed image.

Examples

Convert Grayscale Image to Indexed Image

Read grayscale image into the workspace.

```
I = imread('cameraman.tif');
```

Convert the image to an indexed image using `gray2ind`. This example creates an indexed image with 16 indices.

```
[X, map] = gray2ind(I, 16);
```

Display the indexed image.

```
imshow(X, map);
```

Input Arguments

`I` — Grayscale image

numeric array

Grayscale image, specified as a numeric array of any dimension.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`BW` — Binary image

numeric array

Binary image, specified as a numeric array of any dimension.

Data Types: `logical`

`c` — Number of colormap colors

positive integer

Number of colormap colors, specified as a positive integer between 1 and 65536.

- If the input image is grayscale, then the default value of `c` is `64`.
- If the input image is binary, then the default value of `c` is `2`.

Output Arguments

`X` — Indexed image

numeric array

Indexed image, returned as a numeric array of the same dimensionality as the input grayscale or binary image. If the colormap length is less than or equal to 256, then the class of the output image is `uint8` ; otherwise it is `uint16`.

Data Types: `uint8` | `uint16`

`cmap` — Colormap
c-by-3 numeric matrix

Colormap associated with indexed image `X`, returned as a *c*-by-3 numeric matrix with values in the range [0, 1]. Each row is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap. The colormap is equivalent to `gray(c)`.

Data Types: `double`

Extended Capabilities

Thread-Based Environment
Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also
`grayslice` | `ind2gray` | `mat2gray`

Introduced before R2006a

graycomatrix

Create gray-level co-occurrence matrix from image

Syntax

```
glcms = graycomatrix(I)
glcms = graycomatrix(I,Name,Value)
[glcms,SI] = graycomatrix( ___ )
```

Description

`glcms = graycomatrix(I)` creates a gray-level co-occurrence matrix (GLCM) from image `I`. Another name for a gray-level co-occurrence matrix is a gray-level spatial dependence matrix.

`graycomatrix` creates the GLCM by calculating how often a pixel with gray-level (grayscale intensity) value *i* occurs horizontally adjacent to a pixel with the value *j*. (You can specify other pixel spatial relationships using the `'Offsets'` parameter.) Each element (*i*,*j*) in `glcm` specifies the number of times that the pixel with value *i* occurred horizontally adjacent to a pixel with value *j*.

`glcms = graycomatrix(I,Name,Value)` returns one or more gray-level co-occurrence matrices, depending on the values of the optional name-value pair arguments.

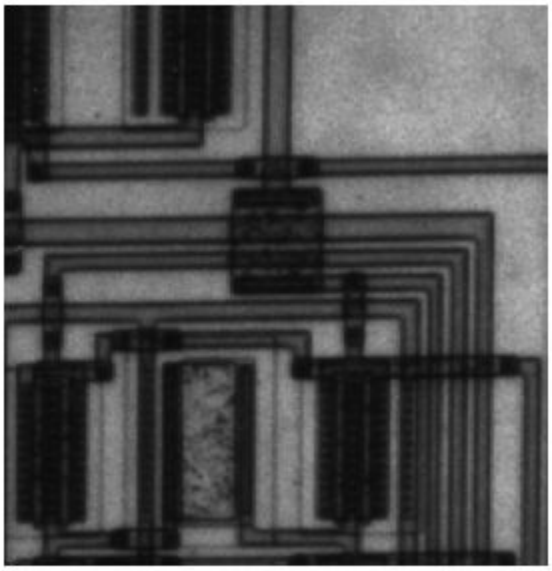
`[glcms,SI] = graycomatrix( ___ )` returns the scaled image, `SI`, used to calculate the gray-level co-occurrence matrix.

Examples

Create Gray-Level Co-occurrence Matrix for Grayscale Image

Read a grayscale image into the workspace.

```
I = imread('circuit.tif');
imshow(I)
```



Calculate the gray-level co-occurrence matrix (GLCM) for the grayscale image. By default, `graycomatrix` calculates the GLCM based on horizontal proximity of the pixels: [0 1]. That is the pixel next to the pixel of interest on the same row. This example specifies a different offset: two rows apart on the same column.

```
glcm = graycomatrix(I,'Offset',[2 0])
```

```
glcm = 8×8

       14205        2107         126           0           0           0           0           0
        2242       14052        3555         400           0           0           0           0
         191        3579        7341        1505          37           0           0           0
           0         683        1446        7184        1368           0           0           0
           0           7         116        1502       10256        1124           0           0
           0           0           0           2        1153        1435           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
```

Create Gray-Level Co-occurrence Matrix Returning Scaled Image

Create a simple 3-by-6 sample array.

```
I = [ 1 1 5 6 8 8; 2 3 5 7 0 2; 0 2 3 5 6 7]
```

```
I = 3×6

     1     1     5     6     8     8
```

```
     2     3     5     7     0     2
     0     2     3     5     6     7
```

Calculate the gray-level co-occurrence matrix (GLCM) and return the scaled image used in the calculation. By specifying empty brackets for the `GrayLimits` parameter, the example uses the minimum and maximum grayscale values in the input image as limits.

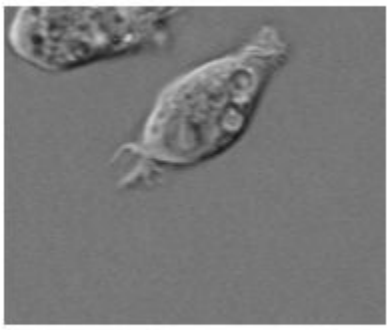
```
[glcm,SI] = graycomatrix(I,'NumLevels',9,'GrayLimits',[])
```

```
glcm = 9×9

     0     0     2     0     0     0     0     0     0
     0     1     0     0     0     1     0     0     0
     0     0     0     2     0     0     0     0     0
     0     0     0     0     0     2     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     2     1     0
     0     0     0     0     0     0     0     1     1
     1     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     1


SI = 3×6

     2     2     6     7     9     9
     3     4     6     8     1     3
     1     3     4     6     7     8
```

Calculate GLCMs using Four Different Offsets

Read a grayscale image into the workspace.

```
I = imread('cell.tif');
imshow(I)
```

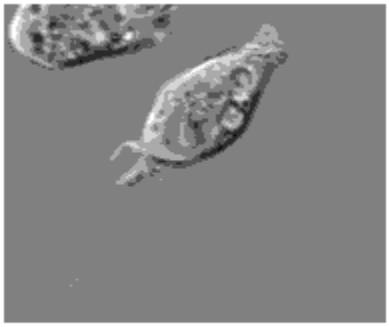


Define four offsets.

```
offsets = [0 1; -1 1;-1 0;-1 -1];
```

Calculate the GLCMs, returning the scaled image as well. Display the scaled image, performing an additional rescaling of data values to the range [0, 1].

```
[glcms,SI] = graycomatrix(I,'Offset',offsets);
imshow(rescale(SI))
```



Note how the function returns an array of four GLCMs.

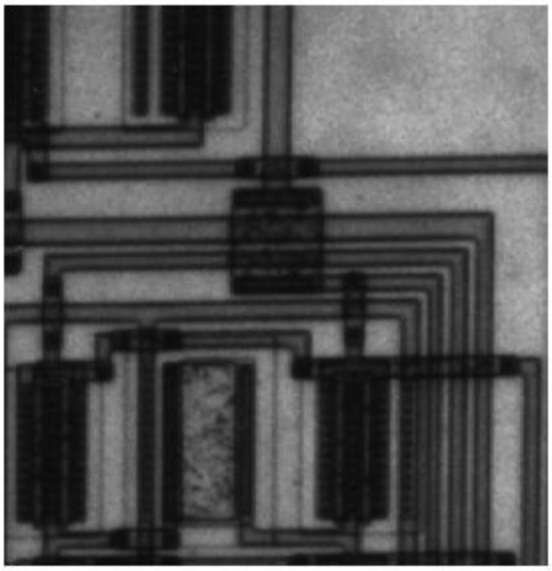
```
whos
```

```
  Name           Size              Bytes  Class     Attributes

  I            159x191             30369  uint8
  SI           159x191            242952  double
  glcms          8x8x4              2048  double
  offsets        4x2                  64  double
```

Calculate Symmetric GLCM for Grayscale Image

Read a grayscale image into the workspace.

```
I = imread('circuit.tif');
imshow(I)
```



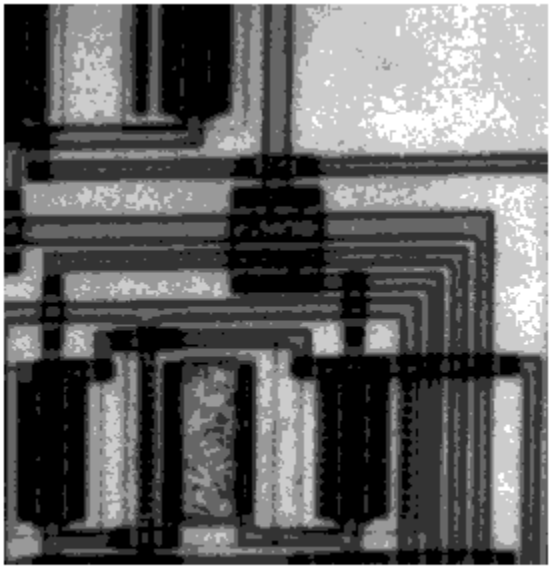
Calculate the GLCM using the `Symmetric` option, returning the scaled image as well. The GLCM created when you set `Symmetric` to `true` is symmetric across its diagonal, and is equivalent to the GLCM described by Haralick (1973).

```
[glcm,SI] = graycomatrix(I,'Offset',[2 0],'Symmetric',true);
glcm
```

```
glcm = 8×8

       28410        4349         317           0           0           0           0           0
        4349       28104        7134        1083           7           0           0           0
         317        7134       14682        2951         153           0           0           0
           0        1083        2951       14368        2870           2           0           0
           0           7         153        2870       20512        2277           0           0
           0           0           0           2        2277        2870           0           0
           0           0           0           0           0           0           0           0
           0           0           0           0           0           0           0           0
```

Display the scaled image, performing an additional rescaling of data values to the range [0, 1].

```
imshow(rescale(SI))
```



Input Arguments

`I` — Grayscale image

2-D numeric matrix | 2-D logical matrix

Input image, specified as a 2-D numeric matrix or 2-D logical matrix.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Offset',[2 0]`

`GrayLimits` — Range used for scaling input image into gray levels

2-element vector `[low high]`

Range used for scaling input image into gray levels, specified as a 2-element vector `[low high]`. If `N` is the number of gray levels (see parameter `'NumLevels'`) to use for scaling, the range `[low high]` is divided into `N` equal width bins and values in a bin get mapped to a single gray level. Grayscale values less than or equal to `low` are scaled to 1. Grayscale values greater than or equal to `high` are scaled to `'NumLevels'`. If `'GrayLimits'` is set to `[]`, `graycomatrix` uses the minimum and maximum grayscale values in `I` as limits, `[min(I(:)) max(I(:))]`, for example, `[0 1]` for class double and `[-32768 32767]` for class `int16`.

`NumLevels` — Number of gray levels

positive integer

Number of gray levels, specified as a positive integer. For example, if `NumLevels` is `8`, then `graycomatrix` scales the values in `I` so they are integers between 1 and 8. The number of gray-levels determines the size of the gray-level co-occurrence matrix (`glcm`). The default number of gray levels is `8` for numeric images and `2` for logical images.

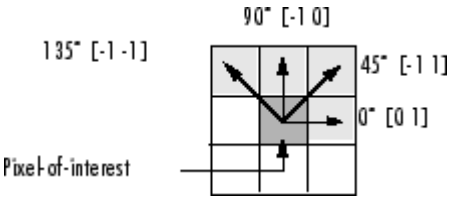
`Offset` — Distance between pixel of interest and its neighbor

`[0 1]` (default) | *p*-by-2 matrix of integers

Distance between the pixel of interest and its neighbor, specified as a *p*-by-2 matrix of integers. Each row in the matrix is a two-element vector, `[row_offset, col_offset]`, that specifies the relationship, or *offset*, of a pair of pixels. `row_offset` is the number of rows between the pixel-of-interest and its neighbor. `col_offset` is the number of columns between the pixel-of-interest and its neighbor. Because the offset is often expressed as an angle, the following table lists the offset values that specify common angles, given the pixel distance D.

Angle	Offset
0	`[0 D]`
45	`[-D D]`
90	`[-D 0]`
135	`[-D -D]`

The figure illustrates the array: `offset = [0 1; -1 1; -1 0; -1 -1]`



`Symmetric` — Consider ordering of values

`false` (default) | `true`

Consider ordering of values, specified as the Boolean value `true` or `false`. For example, when `'Symmetric'` is set to `true`, `graycomatrix` counts both 1,2 and 2,1 pairings when calculating the number of times the value 1 is adjacent to the value 2. When `'Symmetric'` is set to `false`, `graycomatrix` only counts 1,2 or 2,1, depending on the value of `'offset'`.

Data Types: `logical`

Output Arguments

`glcms` — Gray-level co-occurrence matrix

numeric array

Gray-level co-occurrence matrix (or matrices), returned as an `NumLevels`-by-`NumLevels`-by-`P` array, where `P` is the number of offsets in `Offset`.

Data Types: `double`

SI — Scaled image used in calculation of GLCM
numeric matrix

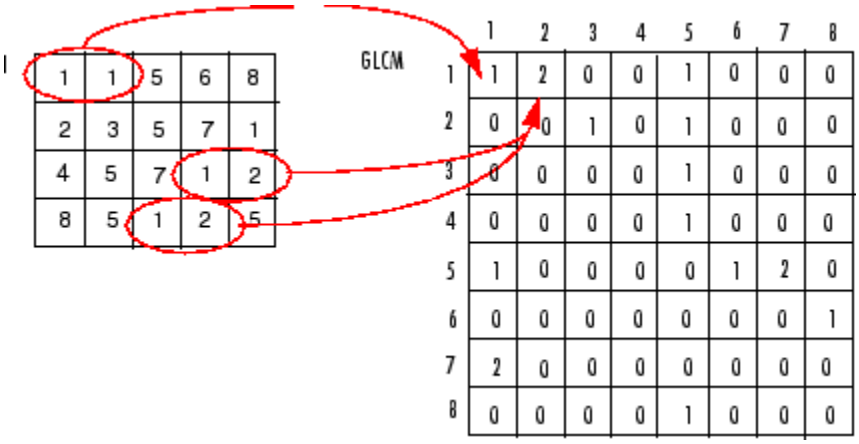
Scaled image used in calculation of GLCM, returned as a numeric matrix of the same size as the input image. The values in `SI` are between `1` and `NumLevels`.

Data Types: `double`

Algorithms

`graycomatrix` calculates the GLCM from a scaled version of the image. By default, if `I` is a binary image, `graycomatrix` scales the image to two gray-levels. If `I` is an intensity image, `graycomatrix` scales the image to eight gray-levels. You can specify the number of gray levels `graycomatrix` uses to scale the image by using the `'NumLevels'` parameter, and the way that `graycomatrix` scales the values using the `'GrayLimits'` name-value argument.

The following figure shows how `graycomatrix` calculates several values in the GLCM of the 4-by-5 image `I`. Element (1,1) in the GLCM contains the value `1` because there is only one instance in the image where two, horizontally adjacent pixels have the values `1` and `1`. Element `(1,2)` in the GLCM contains the value `2` because there are two instances in the image where two, horizontally adjacent pixels have the values `1` and `2`. `graycomatrix` continues this processing to fill in all the values in the GLCM.



`graycomatrix` ignores pixel pairs if either of the pixels contains a `NaN`, replaces positive `Infs` with the value `NumLevels`, and replaces negative `Infs` with the value `1`. `graycomatrix` ignores border pixels, if the corresponding neighbor pixel falls outside the image boundaries.

The GLCM created when `'Symmetric'` is set to `true` is symmetric across its diagonal, and is equivalent to the GLCM described by Haralick (1973). The GLCM produced by the following syntax, with `'Symmetric'` set to `true`

```
graycomatrix(I,'offset',[0 1],'Symmetric',true)
```

is equivalent to the sum of the two GLCMs produced by the following statements where `'Symmetric'` is set to `false`.

```
graycomatrix(I,'offset',[0 1],'Symmetric',false)
graycomatrix(I,'offset',[0 -1],'Symmetric',false)
```

References

[1] Haralick, R.M., K. Shanmugan, and I. Dinstein, "Textural Features for Image Classification", IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-3, 1973, pp. 610-621.

[2] Haralick, R.M., and L.G. Shapiro. Computer and Robot Vision: Vol. 1, Addison-Wesley, 1992, p. 459.

See Also

`graycoprops`

Topics

“Texture Analysis Using the Gray-Level Co-Occurrence Matrix (GLCM)”
“Derive Statistics from GLCM and Plot Correlation”

Introduced before R2006a

grayconnected

Select contiguous image region with similar gray values using flood-fill technique

Syntax

```
BW = grayconnected(I,row,column)
BW = grayconnected(I,row,column,tolerance)
```

Description

`BW = grayconnected(I,row,column)` finds a connected region of similar intensity in the grayscale image `I`. Specify the `row` and `column` indices of the starting point, the *seed pixel*. The function returns a binary mask, `BW`, that indicates which pixels are 8-connected to the seed pixel with a similar intensity.

`BW = grayconnected(I,row,column,tolerance)` specifies the range of intensity values to include in the mask, as in `[(seedvalue-tolerance),(seedvalue+tolerance)]`.

Examples

Segment Image Using Flood-Fill Technique

Read and display a grayscale image.

```
I = imread('cameraman.tif');
imshow(I)
```

Segment the sky in the image by using the flood-fill technique. Select a pixel in the sky to be the seed location. This example uses the pixel with (*row*, *column*) coordinate (50, 50). Call the `grayconnected` function, specifying the image to be segmented and this seed location.

```
J = grayconnected(I,50,50);
```

Display the segmented region in color over the original image by using the `labeloverlay` function. The segmented region includes sky pixels that are 8-connected to the seed pixel. The region does not include pixels of similar intensity that are disconnected, such as the sky visible between the legs of the tripod.

```
imshow(labeloverlay(I,J))
```

Segment the jacket of the cameraman by using the flood-fill. Select a pixel in the jacket to be the seed location. This example specifies the seed pixel with (*row*, *column*) coordinate (110, 65). Call the `grayconnected` function, specifying the image to be segmented and this seed location.

```
J2 = grayconnected(I,110,65);
```

Display the segmented image in color over the original image. The segmented region includes all pixels that are 8-connected to the seed pixel. The tripod and the hair of the cameraman have similar intensity to the jacket, so they are included in the segmented region.

```
imshow(labeloverlay(I,J2))
```

Create Binary Mask from Connected Pixels

Create small sample image.

```
I = uint8([20 22 24 23 25 20 100
           21 19 12 13 12 30 6
           22 11 13 12 24 25 5
           23 13 13 13 24 25 5
           24 27 13 12 12 13 5
           25 26  5 28 29 50 6]);
```

Specify the row and column indices of the seed location. The value at the seed location is 23.

```
seedrow = 4;
seedcol = 1;
```

Specify the tolerance.

```
tol = 3;
```

Create mask image, specifying the seed location and tolerance. The mask includes all pixels that are 8-connected to the seed pixel and have a value in the range [20, 26]. The mask excludes pixels with grayscale values within the tolerance range that are not 8-connected, such as the pixel with (row, column) coordinate (3, 6).

```
BW = grayconnected(I,seedrow,seedcol,tol)
```

```
BW = 6x7 logical array

   1   1   1   1   1   1   0
```

```
1   0   0   0   0   0   0
1   0   0   0   0   0   0
1   0   0   0   0   0   0
1   0   0   0   0   0   0
1   1   0   0   0   0   0
```

Input Arguments

`I` — Grayscale image

numeric matrix

Grayscale image, specified as a numeric matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`row` — Row index of seed pixel

positive integer

Row index of seed pixel, specified as a positive integer.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`column` — Column index of seed pixel

positive integer

Column index of seed pixel, specified as a positive integer.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`tolerance` — Tolerance of intensity values

numeric scalar

Tolerance of intensity values to include in the mask, specified as a numeric scalar. The mask includes all pixels with a value in the range `[(seedvalue-tolerance),(seedvalue+tolerance)]`. By default, the tolerance is `32` for integer-valued images and `0.1` for floating point images.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output Arguments

`BW` — Binary mask

logical matrix

Binary mask of the connected region, returned as a logical array of the same size as `I`. All of the foreground pixels indicate image pixels that are 8-connected to the seed pixel with similar intensity.

Data Types: `logical`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `grayconnected` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `grayconnected` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

See Also

`imfill` | `bwselect` | **Image Segmenter**

Introduced in R2015b

graycoprops

Properties of gray-level co-occurrence matrix

Syntax

```
stats = graycoprops(glcm,properties)
```

Description

`stats = graycoprops(glcm,properties)` calculates the statistics specified in `properties` from the gray-level co-occurrence matrix `glcm`.

`graycoprops` normalizes the gray-level co-occurrence matrix (GLCM) so that the sum of its elements is equal to 1. Each element (*r*,*c*) in the normalized GLCM is the joint probability occurrence of pixel pairs with a defined spatial relationship having gray level values *r* and *c* in the image. `graycoprops` uses the normalized GLCM to calculate `properties`.

Examples

Calculate Statistics from Gray-level Co-occurrence Matrix

Create simple sample GLCM.

```
glcm = [0 1 2 3;1 1 2 3;1 0 2 0;0 0 0 3]
```

```
glcm = 4×4

     0     1     2     3
     1     1     2     3
     1     0     2     0
     0     0     0     3
```

Calculate statistical properties of the GLCM.

```
stats = graycoprops(glcm)
```

```
stats = struct with fields:
       Contrast: 2.8947
    Correlation: 0.0783
         Energy: 0.1191
    Homogeneity: 0.5658
```

Calculate Contrast and Homogeneity from Multiple GLCMs

Read grayscale image into the workspace.

```
I = imread('circuit.tif');
```

Create two gray-level co-occurrence matrices (GLCM) from the image, specifying different offsets.

```
glcm = graycomatrix(I,'Offset',[2 0;0 2])
```

```
glcm =
glcm(:,:,1) =

  Columns 1 through 6

       14205        2107         126           0           0           0
        2242       14052        3555         400           0           0
         191        3579        7341        1505          37           0
           0         683        1446        7184        1368           0
           0           7         116        1502       10256        1124
           0           0           0           2        1153        1435
           0           0           0           0           0           0
           0           0           0           0           0           0

  Columns 7 through 8

           0           0
           0           0
           0           0
           0           0
           0           0
           0           0
           0           0
           0           0


glcm(:,:,2) =

  Columns 1 through 6

       13938        2615         204           4           0           0
        2406       14062        3311         630          23           0
         145        3184        7371        1650         133           0
           2         371        1621        6905        1706           0
           0           0         116        1477        9974        1173
           0           0           0           1        1161        1417
           0           0           0           0           0           0
           0           0           0           0           0           0

  Columns 7 through 8

           0           0
           0           0
           0           0
           0           0
           0           0
           0           0
           0           0
           0           0
```

Get statistics on contrast and homogeneity of the image from the GLCMs.

```
stats = graycoprops(glcm,{'contrast','homogeneity'})
```

```
stats = struct with fields:
       Contrast: [0.3420 0.3567]
    Homogeneity: [0.8567 0.8513]
```

Input Arguments

`glcm` — Gray-level co-occurrence matrix

matrix of nonnegative integers | array of nonnegative integers

Gray-level co-occurrence matrix, specified as one of the following. You can use the `graycomatrix` function to create a GLCM.

- An *m*-by-*n* matrix of nonnegative integers for a single gray-level co-occurrence matrix
- An *m*-by-*n*-by-*p* array of nonnegative integers for *p* valid gray-level co-occurrence matrices.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`properties` — Statistical properties

`'all'` (default) | comma-separated list | cell array | space-separated string scalar or character vector

Statistical properties of the image derived from GLCM, specified as a comma-separated list string scalars or character vectors, space-separated string scalar or character vector, cell array of string scalars or character vectors, or `'all'`. You can specify any of the property names listed in this table.

Property	Description	Formula
`'Contrast'`	Returns a measure of the intensity contrast between a pixel and its neighbor over the whole image. `Range = [0 (size(GLCM,1)-1)^2]` Contrast is 0 for a constant image. The property Contrast is also known as variance and inertia.	$\sum_{i,j} \lvert i-j \rvert^2 p(i,j)$
`'Correlation'`	Returns a measure of how correlated a pixel is to its neighbor over the whole image. `Range = [-1 1]` Correlation is 1 or -1 for a perfectly positively or negatively correlated image. Correlation is `NaN` for a constant image.	$\sum_{i,j} \frac{(i-\mu i)(j-\mu j)p(i,j)}{\sigma_i \sigma_j}$

Property	Description	Formula
`'Energy'`	Returns the sum of squared elements in the GLCM. `Range = [0 1]` Energy is 1 for a constant image. The property Energy is also known as uniformity, uniformity of energy, and angular second moment.	$\sum_{i,j} p(i,j)^2$
`'Homogeneity'`	Returns a value that measures the closeness of the distribution of elements in the GLCM to the GLCM diagonal. `Range = [0 1]` Homogeneity is 1 for a diagonal GLCM.	$\sum_{i,j} \frac{p(i,j)}{1+\lvert i-j\rvert}$

Data Types: `char` | `string` | `cell`

Output Arguments

`stats` — Statistics derived from GLCM

structure

Statistics derived from the GLCM, returned as a structure with fields that are specified by `properties`. Each field contains a 1-by-*p* array, where *p* is the number of gray-level co-occurrence matrices in `glcm`. For example, if `glcm` is an 8-by-8-by-3 array and properties is `'Energy'`, then `stats` is a structure containing the field `Energy`, which contains a 1-by-3 array.

See Also

`graycomatrix`

Topics

“Derive Statistics from GLCM and Plot Correlation”
“Texture Analysis Using the Gray-Level Co-Occurrence Matrix (GLCM)”
“Create a Gray-Level Co-Occurrence Matrix”

Introduced before R2006a

graydiffweight

Calculate weights for image pixels based on grayscale intensity difference

Syntax

```
W = graydiffweight(I,refGrayVal)
W = graydiffweight(I,mask)
W = graydiffweight(I,C,R)
W = graydiffweight(V,C,R,P)
W = graydiffweight(___, Name,Value)
```

Description

`W = graydiffweight(I,refGrayVal)` computes the pixel weight for each pixel in the grayscale image `I`. The weight is the absolute value of the difference between the intensity of the pixel and the reference grayscale intensity specified by the scalar `refGrayVal`. Pick a reference grayscale intensity value that is representative of the object you want to segment. The weights are returned in the array `W`, which is the same size as input image `I`.

The weight of a pixel is inversely related to the absolute value of the grayscale intensity difference at the pixel location. If the difference is small (intensity value close to `refGrayVal`), the weight value is large. If the difference is large (intensity value very different from `refGrayVal`), the weight value is small.

`W = graydiffweight(I,mask)` computes the pixel weights, where the reference grayscale intensity value is the average of the intensity values of all the pixels in `I` that are marked as logical `true` in `mask`. Using the average of several pixels to calculate the reference grayscale intensity value can be more effective than using a single reference intensity value, as in the previous syntax.

`W = graydiffweight(I,C,R)` computes the pixel weights, where the reference grayscale intensity value is the average of the intensity values of the pixel locations specified by the vectors `C` and `R`. `C` and `R` contain the column and row indices of the pixel locations that must be valid pixel indices in `I`.

`W = graydiffweight(V,C,R,P)` computes the weights for each voxel in the volume `V`, specified by the vectors `C`, `R`, and `P`. `C`, `R`, and `P` contain the column, row, and plane indices of the voxel locations that must be valid voxel indices in `V`.

`W = graydiffweight(___, Name,Value)` returns the weight array `W` using name-value pairs to control aspects of weight computation.

Examples

Calculate Grayscale Intensity Difference Weights

This example segments an object in an image using Fast Marching Method using grayscale intensity difference weights calculated from the intensity values at the seed locations.

Read image and display it.

```
I = imread('cameraman.tif');
imshow(I)
title('Original Image')
```

Specify row and column index of pixels for use a reference grayscale intensity value.

```
seedpointR = 159;
seedpointC = 67;
```

Calculate the grayscale intensity difference weight array for the image and display it. The example does log-scaling of `W` for better visualization.

```
W = graydiffweight(I, seedpointC, seedpointR,'GrayDifferenceCutoff',25);
figure, imshow(log(W),[])
```

Segment the image using the grayscale intensity difference weight array. Specify the same seed point vectors you used to create the weight array.

```
thresh = 0.01;
BW = imsegfmm(W, seedpointC, seedpointR, thresh);
figure, imshow(BW)
title('Segmented Image')
```

Input Arguments

`I` — Grayscale image

2-D numeric matrix

Grayscale image, specified as a 2-D numeric matrix.

Data Types: `single` | `double` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

`V` — Grayscale volume

3-D numeric array

Grayscale volume, specified as a 3-D numeric array.

Data Types: `single` | `double` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

`refGrayVal` — Reference grayscale intensity value

scalar

Reference grayscale intensity value, specified as a scalar.

Data Types: `double`

`mask` — Reference grayscale intensity mask

logical array

Reference grayscale intensity mask, specified as a logical array of the same size as `I`.

Data Types: `logical`

`C` — Column index of reference pixel (or voxel)

numeric vector

Column index of reference pixel (or voxel), specified as a numeric (integer-valued) vector.

Data Types: `double`

`R` — Row index of reference pixel (or voxel)

numeric vector

Row index of reference pixel (or voxel), specified as a numeric (integer-valued) vector.

Data Types: `double`

`P` — Plane index of reference voxel

numeric vector

Plane index of reference voxel, specified as a numeric (integer-valued) vector.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `W = graydiffweight(I, seedpointC, seedpointR,'GrayDifferenceCutoff',25);`

`RolloffFactor` — Output weight roll-off factor
0.5 (default) | positive scalar

Output weight roll-off factor, specified as the comma-separated pair consisting of `'RolloffFactor'` and a positive scalar of class `double`. Controls how fast the output weight falls as the function of the absolute difference between an intensity value and the reference grayscale intensity. When viewed as a 2-D plot, pixel intensity values can vary gradually at the edges of regions, creating a gentle slope. In your segmented image, you might want the edge to be more well-defined. Using the roll-off factor, you control the slope of the weight value curve at points where intensity values start to change. If you specify a high value, the output weight values fall off sharply around the regions of change intensity. If you specify a low value, the output weight has a more gradual fall-off around the regions of changing intensity. The suggested range for this parameter is `[0.5 4]`.

Data Types: `double`

`GrayDifferenceCutoff` — Threshold for absolute grayscale intensity difference values
`Inf` (default) | nonnegative scalar

Threshold for absolute grayscale intensity difference values, specified as the comma-separated pair consisting of `'GrayDifferenceCutoff'` and a nonnegative scalar of class `double`. When you put a threshold on intensity difference values, you strongly suppress output weight values greater than the cutoff value. `graydiffweight` assigns these pixels the smallest weight value. When the output weight array `W` is used for Fast Marching Method based segmentation (as input to `imsegfmm`), this parameter can be useful in improving the accuracy of the segmentation output. Default value of this parameter is `Inf`, which means that there is no hard cutoff.

Data Types: `double`

Output Arguments

`W` — Weight array
numeric array

Weight array, specified as numeric array of the same size as the input image `I` or volume `V`. `W` is of class `double`, unless the input image or volume is of class `single`, in which case `W` is of class `single`.

See Also

`imsegfmm` | `gradientweight` | `graydist`

Introduced in R2014b

graydist

Gray-weighted distance transform of grayscale image

Syntax

```
T = graydist(I,mask)
T = graydist(I,C,R)
T = graydist(I,ind)
T = graydist( ___ ,method)
```

Description

`T = graydist(I,mask)` computes the gray-weighted distance transform of the grayscale image `I`. Locations where `mask` is `true` are seed locations.

`T = graydist(I,C,R)` specifies the column and row coordinates of seed locations in vectors `C` and R.

`T = graydist(I,ind)` specifies the linear indices of seed locations, `ind`.

`T = graydist( ___ ,method)` specifies an alternate distance metric, `method`.

Examples

Compute Minimum Path in Magic Square

Create a magic square. Matrices generated by the magic function have equal row, column, and diagonal sums. The minimum path between the upper-left and lower-right corner is along the diagonal.

```
A  = magic(3)
```

```
A = 3×3

     8     1     6
     3     5     7
     4     9     2
```

Calculate the gray-weighted distance transform, specifying the upper left corner and the lower right corner of the square as seed locations.

```
T1 = graydist(A,1,1);
T2 = graydist(A,3,3);
```

Sum the two transforms to find the minimum path between the seed locations. As expected, there is a constant-value minimum path along the diagonal.

```
T  = T1 + T2
```

```
T = 3×3
```

```
    10    11    17
    13    10    13
    17    17    10
```

Input Arguments

`I` — Grayscale image

numeric array | logical array

Grayscale image, specified as a numeric or logical array.

`mask` — Binary mask

logical array

Binary mask that specifies seed locations, specified as a logical array the same size as `I`.

`C, R` — Column and row coordinates

vector of positive integers

Column and row coordinates of seed locations, specified as a vector of positive integers. Coordinate values are valid `C`,`R` subscripts in `I`.

`ind` — Indices

vector of positive integers

Indices of seed locations, specified as a vector of positive integers.

`method` — Distance metric

`'chessboard'` (default) | `'cityblock'` | `'quasi-euclidean'`

Distance metric, specified as one of these values.

Method	Description
`'chessboard'`	In 2-D, the chessboard distance between (x_1,y_1) and (x_2,y_2) is $\max(\lvert x_1 - x_2\rvert, \lvert y_1 - y_2\rvert)$.
`'cityblock'`	In 2-D, the cityblock distance between (x_1,y_1) and (x_2,y_2) is $\lvert x_1 - x_2\rvert + \lvert y_1 - y_2\rvert$
`'quasi-euclidean'`	In 2-D, the quasi-Euclidean distance between (x_1,y_1) and (x_2,y_2) is $\lvert x_1 - x_2\rvert + (\sqrt{2} - 1)\lvert y_1 - y_2\rvert,\ \lvert x_1 - x_2\rvert > \lvert y_1 - y_2\rvert$ $(\sqrt{2} - 1)\lvert x_1 - x_2\rvert + \lvert y_1 - y_2\rvert,$ otherwise.

For more information, see “Distance Transform of a Binary Image”.

Output Arguments

`T` — Gray-weighted distance transform

numeric array

Gray-weighted distance transform, returned as a numeric array of the same size as `I`. If the input numeric type of `I` is `double`, then the output numeric type of `T` is `double`. If the input is any other numeric type, then the output `T` is `single`.

Data Types: `single` | `double`

Algorithms

`graydist` uses the geodesic time algorithm [1]. The basic equation for geodesic time along a path is:

$$\tau_f(P) = \frac{f(p_o)}{2} + \frac{f(p_l)}{2} + \sum_{i=1}^{l-1} f(p_i)$$

`method` determines the chamfer weights that are assigned to the local neighborhood during outward propagation. Each pixel's contribution to the geodesic time is based on the chamfer weight in a particular direction multiplied by the pixel intensity.

References

[1] Soille, P. "Generalized geodesy via geodesic time." *Pattern Recognition Letters*. Vol.15, December 1994, pp. 1235–1240.

See Also

`bwdist` | `bwdistgeodesic` | `watershed`

Topics

“Distance Transform of a Binary Image”

Introduced in R2011b

grayslice

Convert grayscale image to indexed image using multilevel thresholding

Syntax

```
X = grayslice(I,N)
X = grayslice(I,thresholds)
```

Description

`X = grayslice(I,N)` converts a grayscale image to an indexed image by using multilevel thresholding approach. The function automatically calculates the threshold values based on `N`. To learn more about threshold calculation, see “Algorithms” on page 1-1155.

`X = grayslice(I,thresholds)` returns an indexed image by multilevel thresholding of input image using a specified set of thresholds.

Examples

Convert Grayscale Image to Indexed Image Using Thresholding

Read grayscale image into the workspace.

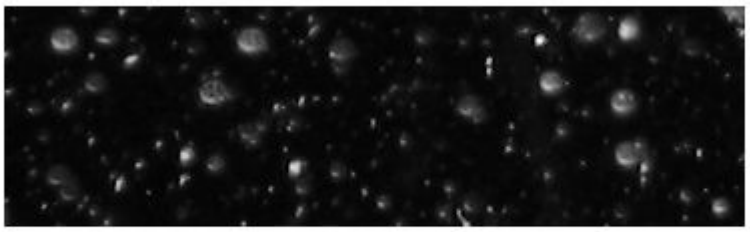
```
I = imread('snowflakes.png');
```

Threshold the intensity image, returning an indexed image.

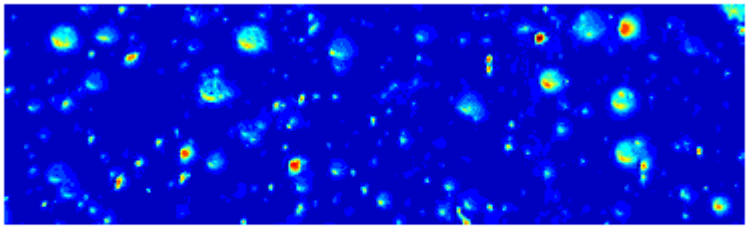
```
X = grayslice(I,16);
```

Display the original image and the indexed image, using one of the standard colormaps.

```
imshow(I)
```



```
figure
imshow(X,jet(16))
```



Convert Grayscale Image to Indexed Image Using Multilevel Thresholds

Read a grayscale image into the workspace. Display the image.

```
I = imread('coins.png');
imshow(I)
```

Specify the threshold values for multilevel thresholding.

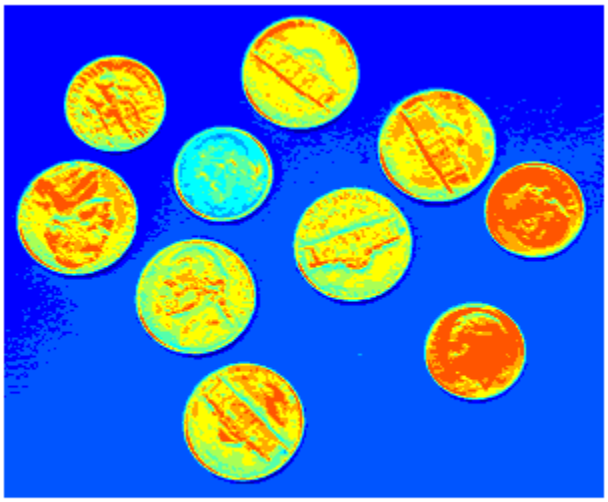
```
thresholds = [45 65 84 108 134 157 174 189 206 228];
```

Convert the input grayscale image to an indexed image.

```
X = grayslice(I,thresholds);
```

Display the indexed image. Set the colormap of the indexed image to `jet`. The length of colormap, `m`, is the maximum intensity value in the indexed image.

```
m = double(max(X(:)));

figure
imshow(X,colormap(jet(m)))
```



Input Arguments

`I` — Input grayscale image

m-by-*n* numeric matrix

Input grayscale image, specified as a *m*-by-*n* numeric matrix.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`N` — Number of threshold values

positive scalar

Number of threshold values, specified as a positive scalar. The value represents the total number of thresholds to be used for multilevel thresholding.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`thresholds` — Set of thresholds

numeric vector

Set of thresholds, specified as a numeric vector. The number of threshold values to be used for multilevel thresholding is equal to `length(thresholds)`.

Image Data Type	Range of Valid Threshold Values
`uint8`	[0, 255]
`int16` or `uint16`	[0, 65535]
`single` or `double`	[0, 1]

Note Before thresholding an image of data type `int16`, the `grayslice` function converts the image to `uint16` by adding 32,768 to each pixel. Consider this additive offset when specifying thresholds for input images of data type `int16`.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

Output Arguments

X — Output indexed image

m-by-*n* matrix

Output indexed image, returned as a *m*-by-*n* matrix of the same size as the input grayscale image. The data type of X depends on the number of threshold values used for multilevel thresholding.

- If the number of threshold values is less than 256, then X is of data type `uint8`. In this case, the range of intensity values in X is either [0, N-1] or [0, `length(thresholds)`].
- If the number of threshold values is greater than or equal to 256, then X is of data type `double`. In this case, the range of intensity values in X is either [1, N] or [1, `length(thresholds)`+1].

Data Types: `uint8` | `double`

Tips

- You can view the thresholded image using `imshow(X,map)` with a colormap of appropriate length.

Algorithms

The function performs multilevel thresholding of the input grayscale image and returns an indexed image as the output. If you specify the number of thresholds N, then `grayslice` assigns pixels to N indices according to the these thresholds.

- The first index in X consists of the grayscale pixels in the range $max\_intensity \times \left[0, \frac{1}{N}\right)$
- The *k*-th index in X consists of the grayscale pixels in the range $max\_intensity \times \left[\frac{k-1}{N}, \frac{k}{N}\right)$
- The last index in X consists of the grayscale pixels in the range $max\_intensity \times \left[\frac{N-1}{N}, 1\right]$.

max_intensity depends on the data type of the input image.

Image Data Type	*max_intensity*
`uint8`	255
`int16` or `uint16`	65535
`single` or `double`	1

Note Before thresholding an image of data type `int16`, the `grayslice` function converts the image to `uint16` by adding 32,768 to each pixel.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

`gray2ind`

Introduced before R2006a

graythresh

Global image threshold using Otsu's method

Syntax

```
T = graythresh(I)
[T,EM] = graythresh(I)
```

Description

`T = graythresh(I)` computes a global threshold `T` from grayscale image `I`, using Otsu's method [1]. Otsu's method chooses a threshold that minimizes the intraclass variance of the thresholded black and white pixels. The global threshold `T` can be used with `imbinarize` to convert a grayscale image to a binary image.

`[T,EM] = graythresh(I)` also returns the effectiveness metric, `EM`.

Examples

Convert Intensity Image to Binary Image Using Level Threshold

Read a grayscale image into the workspace.

```
I = imread('coins.png');
```

Calculate a threshold using `graythresh`. The threshold is normalized to the range [0, 1].

```
level = graythresh(I)
```

```
level = 0.4941
```

Convert the image into a binary image using the threshold.

```
BW = imbinarize(I,level);
```

Display the original image next to the binary image.

```
imshowpair(I,BW,'montage')
```

Input Arguments

`I` — Grayscale image

numeric array

Grayscale image, specified as a numeric array of any dimensionality. The `graythresh` function converts multidimensional arrays to 2-D arrays using the `reshape` function, and ignores any nonzero imaginary part of `I`.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

Output Arguments

`T` — Global threshold

nonnegative number

Global threshold, returned as a nonnegative number in the range [0, 1].

Data Types: `double`

`EM` — Effectiveness metric

nonnegative number

Effectiveness metric of the threshold, returned as a nonnegative number in the range [0, 1]. The lower bound is attainable only by images having a single gray level, and the upper bound is attainable only by two-valued images.

Data Types: `double`

Tips

- By default, the function `imbinarize` creates a binary image using a threshold obtained using Otsu's method. This default threshold is identical to the threshold returned by `graythresh`.

However, `imbinarize` only returns the binary image. If you want to know the level or the effectiveness metric, use `graythresh` before calling `imbinarize`.

References

[1] Otsu, N., "A Threshold Selection Method from Gray-Level Histograms." *IEEE Transactions on Systems, Man, and Cybernetics*. Vol. 9, No. 1, 1979, pp. 62–66.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`graythresh` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

See Also

`imbinarize` | `imquantize` | `multithresh` | `rgb2ind`

Introduced before R2006a

hdrread

Read high dynamic range (HDR) image

Syntax

```
hdr = hdrread(filename)
```

Description

`hdr = hdrread(filename)` reads the high dynamic range (HDR) image, `hdr`, from the file specified by `filename`. For scene-referred data sets, pixel values usually are scene illumination in radiance units.

Examples

Read and Display High Dynamic Range Image

Read high dynamic range image into the workspace.

```
hdr = hdrread('office.hdr');
```

Convert the HDR image to a lower dynamic range, suitable for display.

```
rgb = tonemap(hdr);
```

Display the image.

```
imshow(rgb);
```

Input Arguments

`filename` — File name

character vector | string scalar

File name of HDR image, specified as a character vector or string scalar.

Example: `'office.hdr'` or `"office.hdr"`

Data Types: `char` | `string`

Output Arguments

`hdr` — HDR image

m-by-*n*-by-3 numeric array

HDR image, returned as an *m*-by-*n*-by-3 numeric array with values in the range `[0,Inf)`.

Data Types: `single`

Tips

- To display HDR images, use an appropriate tone-mapping function, such as `tonemap`.

References

[1] Larson, Greg W. "Radiance File Formats". *http://radsite.lbl.gov/radiance/refer/filefmts.pdf*

See Also

`hdrwrite` | `makehdr` | `tonemap`

Introduced in R2007b

hdrwrite

Write high dynamic range (HDR) image file

Syntax

```
hdrwrite(hdr,filename)
```

Description

`hdrwrite(hdr,filename)` writes high dynamic range (HDR) image `hdr` to a file with name `filename`. The function uses run-length encoding to minimize file size.

Examples

Write High Dynamic Range Image to File

Read a high dynamic range image into the workspace.

```
hdr = hdrread('office.hdr');
```

Create a new HDR file, writing the high dynamic range data, `hdr`, to a file with a new filename.

```
hdrwrite(hdr,'newHDRfile.hdr');
```

Input Arguments

`hdr` — HDR image

m-by-*n*-by-3 numeric array

HDR image, specified as an *m*-by-*n*-by-3 numeric array of positive numbers.

Data Types: `single` | `double`

`filename` — File name

character vector | string scalar

File name of HDR image, specified as a character vector or string scalar ending with extension `'hdr'`.

Example: `'office.hdr'` or `"office.hdr"`

Data Types: `char` | `string`

See Also

`hdrread` | `makehdr` | `tonemap`

Introduced in R2008a

histeq

Enhance contrast using histogram equalization

Syntax

```
J = histeq(I)
J = histeq(I,n)
J = histeq(I,hgram)

newmap = histeq(X,map)
newmap = histeq(X,map,hgram)

[___,T] = histeq(___)
```

Description

`J = histeq(I)` transforms the grayscale image `I` so that the histogram of the output grayscale image `J` has 64 bins and is approximately flat.

`J = histeq(I,n)` transforms the grayscale image `I` so that the histogram of the output grayscale image `J` with `n` bins is approximately flat. The histogram of `J` is flatter when `n` is much smaller than the number of discrete levels in `I`.

`J = histeq(I,hgram)` transforms the grayscale image `I` so that the histogram of the output grayscale image `J` with `length(hgram)` bins approximately matches the target histogram `hgram`.

`newmap = histeq(X,map)` transforms the values in the colormap so that the histogram of the gray component of the indexed image `X` is approximately flat. The transformed colormap is `newmap`.

`newmap = histeq(X,map,hgram)` transforms the colormap associated with the indexed image `X` so that the histogram of the gray component of the indexed image (`X`,`newmap`) approximately matches the target histogram `hgram`. The `histeq` function returns the transformed colormap in `newmap`. `length(hgram)` must be the same as `size(map,1)`.

`[___,T] = histeq(___)` also returns the transformation `T` that maps the gray component of the input grayscale image or colormap to the gray component of the output grayscale image or colormap.

Examples

Enhance Contrast Using Histogram Equalization

Read an image into the workspace.

```
I = imread('tire.tif');
```

Enhance the contrast of an intensity image using histogram equalization.

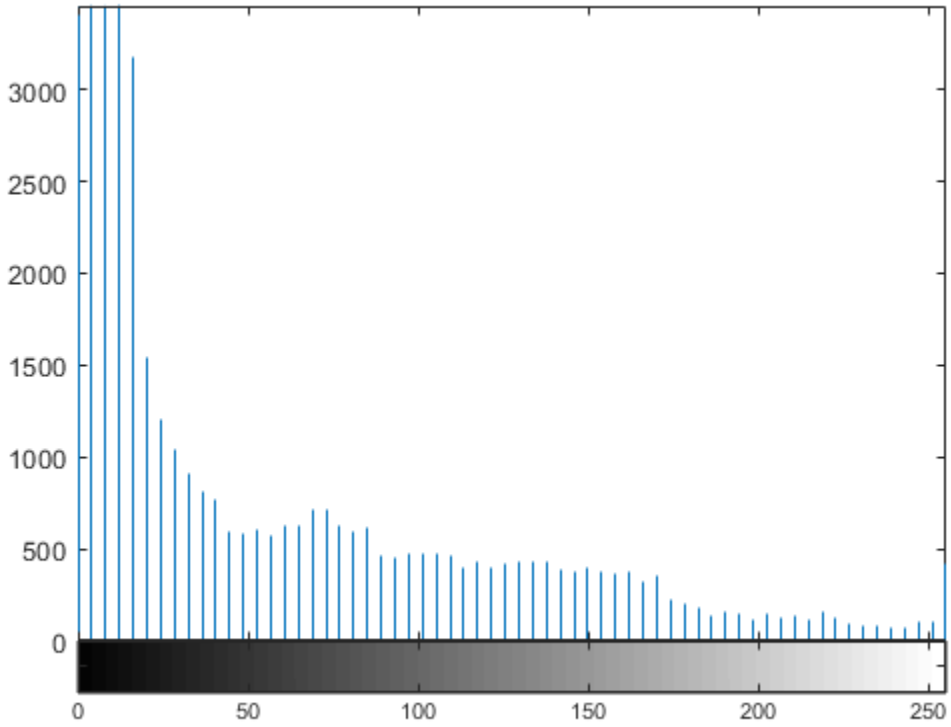
```
J = histeq(I);
```

Display the original image and the adjusted image.

```
imshowpair(I,J,'montage')
axis off
```

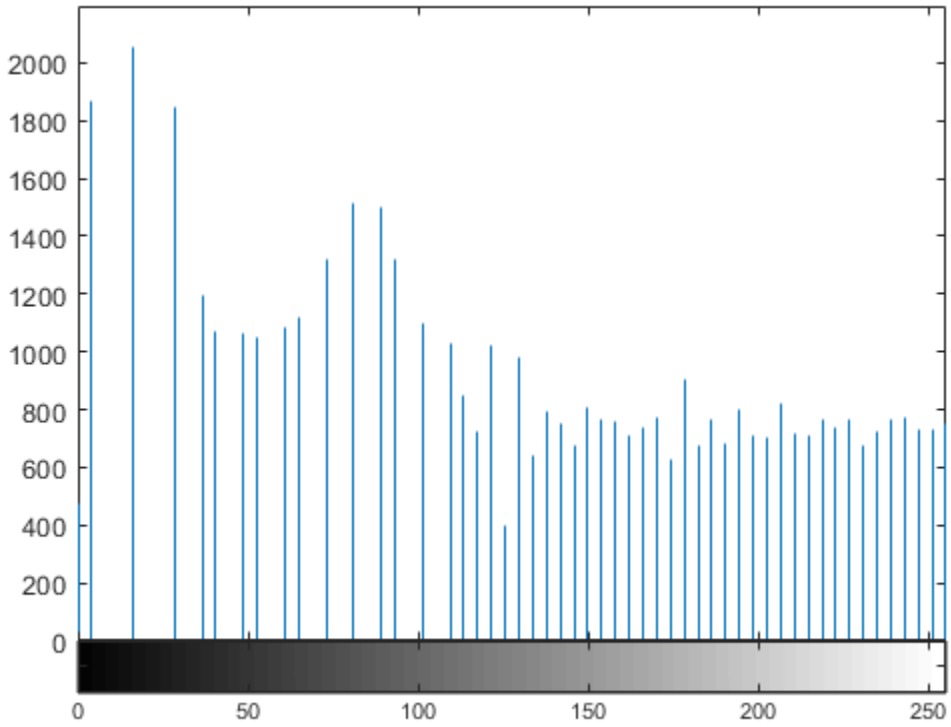
Display a histogram of the original image.

```
figure
imhist(I,64)
```



Display a histogram of the processed image.

```
figure
imhist(J,64)
```



Enhance Contrast of Volumetric Image Using Histogram Equalization

Load a 3-D dataset.

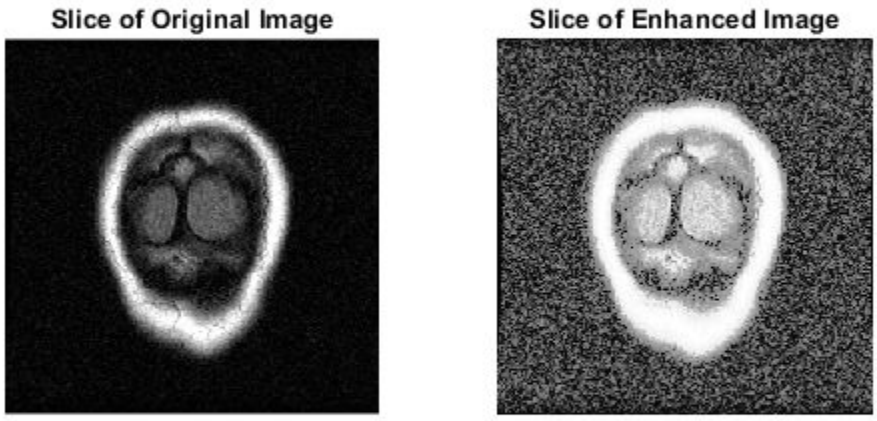
```
load mristack
```

Perform histogram equalization.

```
enhanced = histeq(mristack);
```

Display the first slice of data for the original image and the contrast-enhanced image.

```
figure
subplot(1,2,1)
imshow(mristack(:,:,1))
title('Slice of Original Image')
subplot(1,2,2)
imshow(enhanced(:,:,1))
title('Slice of Enhanced Image')
```



Input Arguments

`I` — Grayscale image

numeric array

Grayscale image, specified as a numeric array of any dimension.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`hgram` — Target histogram

numeric vector

Target histogram, specified as a numeric vector. `hgram` has equally spaced bins with intensity values in the appropriate range:

- [0, 1] for images of class `double` or `single`
- [0, 255] for images of class `uint8`
- [0, 65535] for images of class `uint16`
- [-32768, 32767] for images of class `int16`

`histeq` automatically scales `hgram` so that `sum(hgram)=numel(I)`. The histogram of `J` better matches `hgram` when `length(hgram)` is much smaller than the number of discrete levels in `I`.

Data Types: `single` | `double`

`n` — Number of discrete gray levels
64 (default) | positive integer

Number of discrete gray levels, specified as a positive integer.

Data Types: `single` | `double`

`X` — Indexed image
numeric array

Indexed image, specified as a numeric array of any dimension. The values in `X` are an index into the colormap `map`.

Data Types: `single` | `double` | `uint8` | `uint16`

`map` — Colormap
c-by-3 numeric matrix

Colormap associated with indexed image `X`, specified as a *c*-by-3 numeric matrix with values in the range [0, 1]. Each row is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap.

Data Types: `double`

Output Arguments

`J` — Transformed grayscale image
numeric array

Transformed grayscale image, returned as a numeric array of the same size and class as the input image `I`.

`T` — Grayscale transformation
numeric vector

Grayscale transformation, returned as a numeric vector. The transformation `T` maps gray levels in the image `I` to gray levels in `J`.

Data Types: `double`

`newmap` — Transformed colormap
n-by-3 numeric matrix

Transformed colormap, specified as an *n*-by-3 numeric matrix with values in the range [0, 1]. Each row is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap.

Data Types: `double`

Algorithms

When you supply a desired histogram `hgram`, `histeq` chooses the grayscale transformation `T` to minimize

$$|c_1(T(k)) - c_0(k)|,$$

c_0 is the cumulative histogram of the input image `I`, and c_1 is the cumulative sum of `hgram` for all intensities k. This minimization is subject to these constraints:

- `T` must be monotonic
- $c_1(T(a))$ cannot overshoot $c_0(a)$ by more than half the distance between the histogram counts at a

`histeq` uses the transformation $b = T(a)$ to map the gray levels in `X` (or the colormap) to their new values.

If you do not specify `hgram`, then `histeq` creates a flat `hgram`,

```
hgram = ones(1,n)*prod(size(A))/n;
```

and then applies the previous algorithm.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `histeq` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `histeq` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, `histeq` does not support indexed images.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`adapthisteq` | `brighten` | `imadjust` | `imhist`

Introduced before R2006a

hough

Hough transform

Syntax

```
[H,theta,rho] = hough(BW)
[H,theta,rho] = hough(BW,Name,Value)
```

Description

`[H,theta,rho] = hough(BW)` computes the Standard Hough Transform (SHT) of the binary image `BW`. The `hough` function is designed to detect lines. The function uses the parametric representation of a line: `rho = x*cos(theta) + y*sin(theta)`. The function returns `rho`, the distance from the origin to the line along a vector perpendicular to the line, and `theta`, the angle in degrees between the *x*-axis and this vector. The function also returns the SHT, `H`, which is a parameter space matrix whose rows and columns correspond to *rho* and *theta* values respectively. For more information, see “Algorithms” on page 1-1174.

`[H,theta,rho] = hough(BW,Name,Value)` computes the SHT of the binary image `BW` using name-value arguments to affect the computation.

Examples

Compute and Display Hough Transform

Read an image, and convert it to a grayscale image.

```
RGB = imread('gantrycrane.png');
I  = im2gray(RGB);
```

Extract edges.

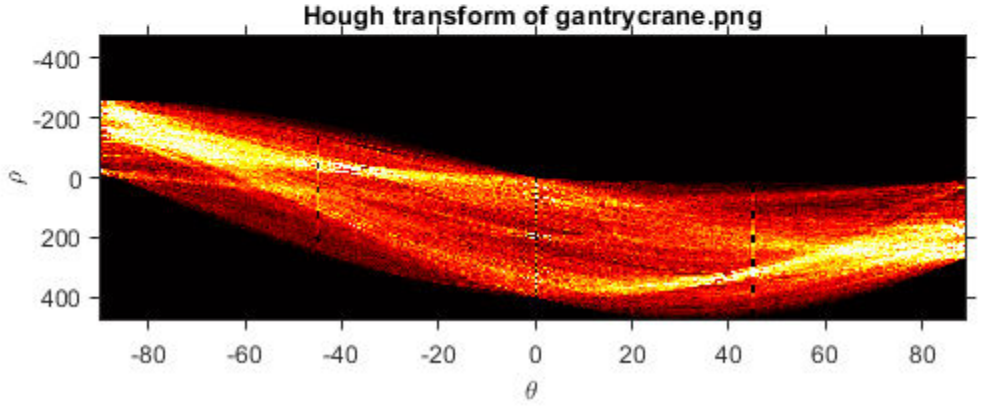
```
BW = edge(I,'canny');
```

Calculate Hough transform.

```
[H,T,R] = hough(BW,'RhoResolution',0.5,'Theta',-90:0.5:89);
```

Display the original image and the Hough matrix.

```
subplot(2,1,1);
imshow(RGB);
title('gantrycrane.png');
subplot(2,1,2);
imshow(imadjust(rescale(H)),'XData',T,'YData',R,...
      'InitialMagnification','fit');
title('Hough transform of gantrycrane.png');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
colormap(gca,hot);
```



Compute Hough Transform Over Limited Theta Range

Read an image, and convert it to grayscale.

```
RGB = imread('gantrycrane.png');
I  = im2gray(RGB);
```

Extract edges.

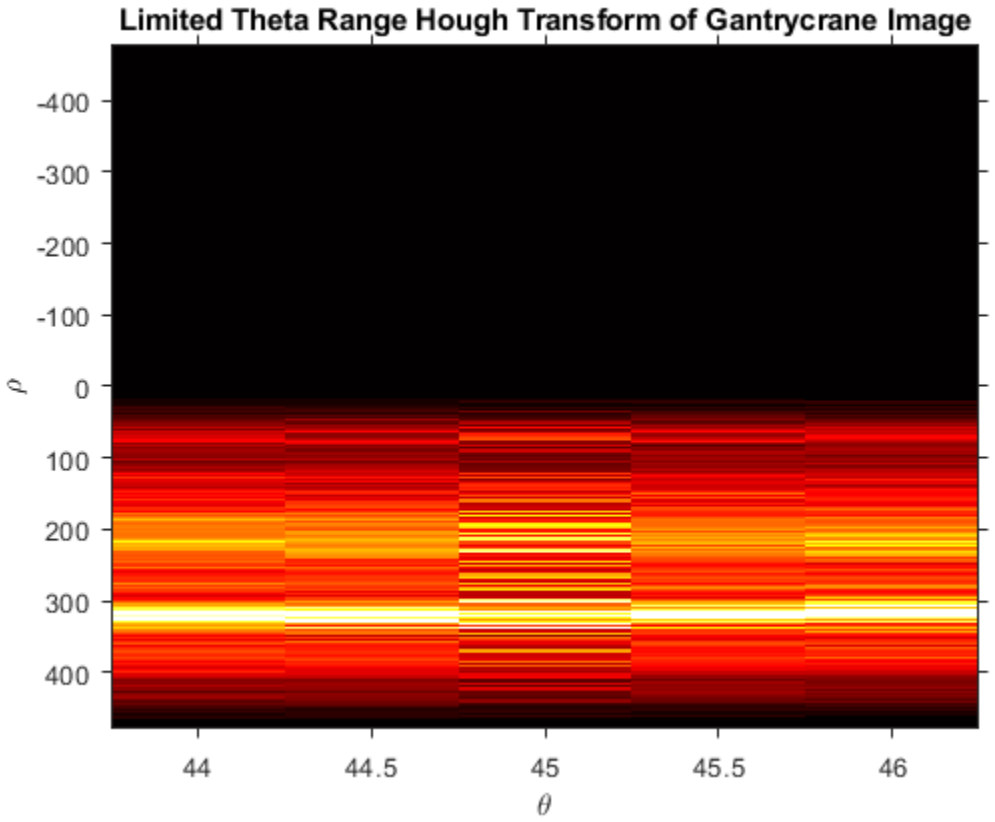
```
BW = edge(I,'canny');
```

Calculate the Hough transform over a limited range of angles.

```
[H,T,R] = hough(BW,'Theta',44:0.5:46);
```

Display the Hough transform.

```
figure
imshow(imadjust(rescale(H)),'XData',T,'YData',R,...
   'InitialMagnification','fit');
title('Limited Theta Range Hough Transform of Gantrycrane Image');
xlabel('\theta')
ylabel('\rho');
axis on, axis normal;
colormap(gca,hot)
```



Input Arguments

`BW` — Binary image

2-D logical matrix | 2-D numeric matrix

Binary image, specified as a 2-D logical matrix or 2-D numeric matrix. For numeric input, any nonzero pixels are considered to be `1` (`true`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `[H,theta,rho] = hough(BW,RhoResolution=0.5)`

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `[H,theta,rho] = hough(BW,"RhoResolution",0.5)`

`RhoResolution` — Spacing of Hough transform bins

`1` (default) | positive number

Spacing of Hough transform bins along the *rho* axis, specified as a positive number between 0 and `norm(size(BW))`, exclusive.

Data Types: `double`

`Theta` — *Theta* values for SHT
`-90:89` (default) | numeric vector

Theta values for the SHT, specified as a numeric vector with elements in the range [-90, 90).

Example: `-90:0.5:89.5`

Data Types: `double`

Output Arguments

`H` — Hough transform matrix
numeric matrix

Hough transform matrix, returned as a numeric matrix of size *nrho*-by-*ntheta*. The rows and columns correspond to `rho` and `theta` values. For more information, see “Algorithms” on page 1-1174.

`theta` — Angle between *x*-axis and *rho* vector
numeric matrix

Angle between the *x*-axis and the *rho* vector, in degrees, returned as a numeric matrix. For more information, see “Algorithms” on page 1-1174.

Data Types: `double`

`rho` — Distance from origin to line
numeric array

Distance from the origin to the line along a vector perpendicular to the line, returned as a numeric array. For more information, see “Algorithms” on page 1-1174.

Data Types: `double`

Algorithms

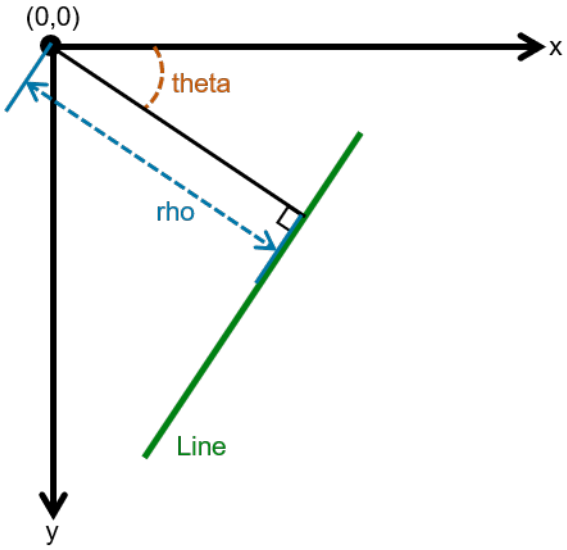
The Standard Hough Transform (SHT) uses the parametric representation of a line:

```
rho = x*cos(theta) + y*sin(theta)
```

The origin of the coordinate system is assumed to be at the center of the upper-left corner pixel.

The variable *rho* is the perpendicular distance from the origin to the line.

The variable *theta* is the angle of the perpendicular projection from the origin to the line, measured in degrees clockwise from the positive *x*-axis. The range of *theta* is –90° ≤ θ < 90°. The angle of the line itself is θ + 90°, also measured clockwise with respect to the positive *x*-axis.



The SHT is a parameter space matrix whose rows and columns correspond to *rho* and *theta* values, respectively. The elements in the SHT represent accumulator cells. Initially, the value in each cell is zero. Then, for every non-background point in the image, *rho* is calculated for every *theta*. *rho* is rounded off to the nearest allowed row in SHT. That accumulator cell is incremented. At the end of this procedure, a value of *Q* in *SHT(r,c)* means that *Q* points in the *xy*-plane lie on the line specified by *theta(c)* and *rho(r)*. Peak values in the SHT represent potential lines in the input image.

The Hough transform matrix, `H`, is *nrho*-by-*ntheta* where:

`nrho = 2*(ceil(D/RhoResolution)) + 1`, and
`D = sqrt((numRowsInBW - 1)^2 + (numColsInBW - 1)^2)`.
`rho` values range from `-diagonal` to `diagonal`, where
`diagonal = RhoResolution*ceil(D/RhoResolution)`.

`ntheta = length(theta)`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `hough` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- The optional parameters `Theta` and `RhoResolution` must be compile-time string constants.
- The optional `Theta` vector must have a bounded size.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The optional parameters `Theta` and `RhoResolution` must be compile-time string constants.
- The optional `Theta` vector must have a bounded size.

See Also

`houghlines` | `houghpeaks`

Topics

“Detect Lines in Images Using Hough”

Introduced before R2006a

houghlines

Extract line segments based on Hough transform

Syntax

```
lines = houghlines(BW,theta,rho,peaks)
lines = houghlines( ___ ,Name,Value)
```

Description

`lines = houghlines(BW,theta,rho,peaks)` extracts line segments in the image `BW` associated with particular bins in a Hough transform. `theta` and `rho` are vectors returned by function `hough`. `peaks` is a matrix returned by the `houghpeaks` function that contains the row and column coordinates of the Hough transform bins to use in searching for line segments. The return value `lines` contains information about the extracted line segments.

`lines = houghlines( ___ ,Name,Value)` uses name-value pair arguments to control various aspects of the line extraction.

Examples

Find Line Segments and Highlight longest segment

Read image into workspace.

```
I  = imread('circuit.tif');
```

Rotate the image.

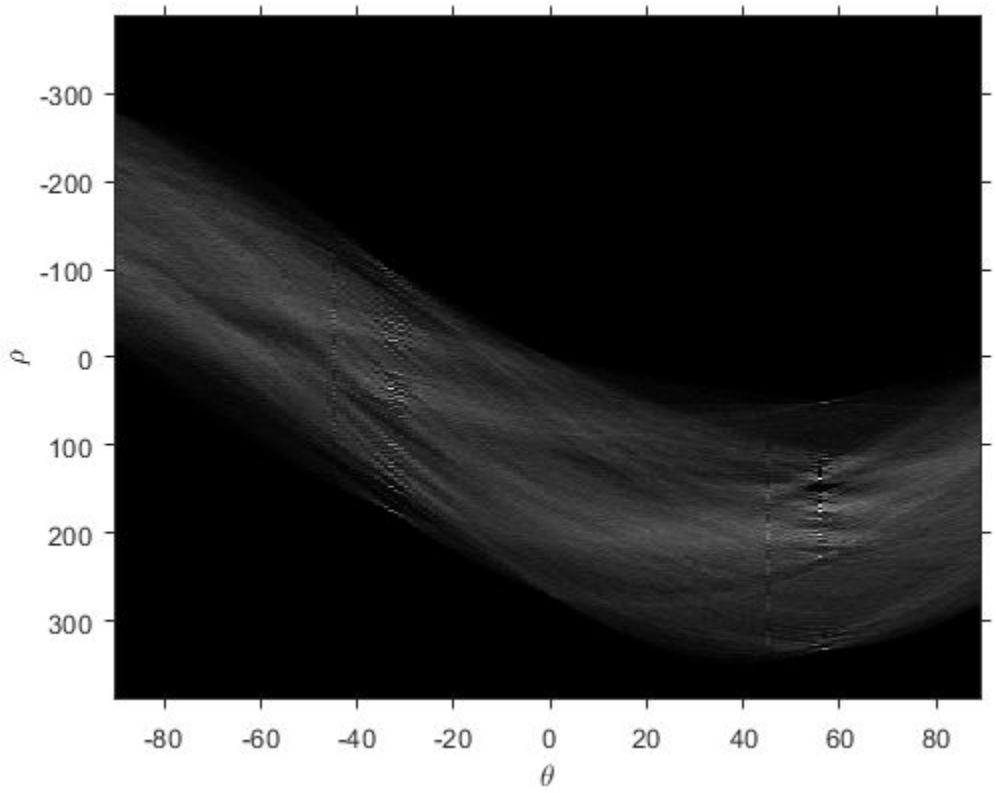
```
rotI = imrotate(I,33,'crop');
```

Create a binary image.

```
BW = edge(rotI,'canny');
```

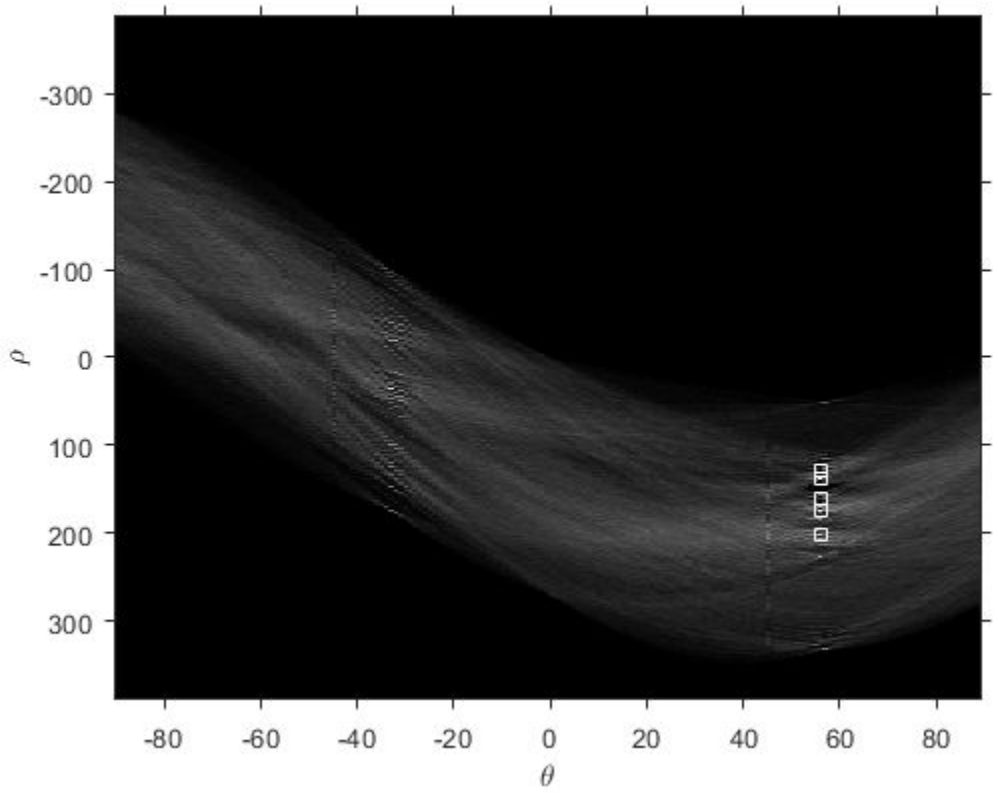
Create the Hough transform using the binary image.

```
[H,T,R] = hough(BW);
imshow(H,[],'XData',T,'YData',R,...
            'InitialMagnification','fit');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
```



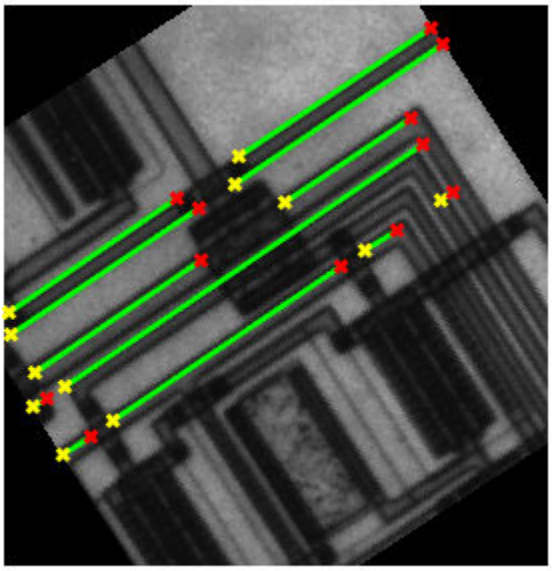
Find peaks in the Hough transform of the image.

```
P  = houghpeaks(H,5,'threshold',ceil(0.3*max(H(:))));
x = T(P(:,2)); y = R(P(:,1));
plot(x,y,'s','color','white');
```



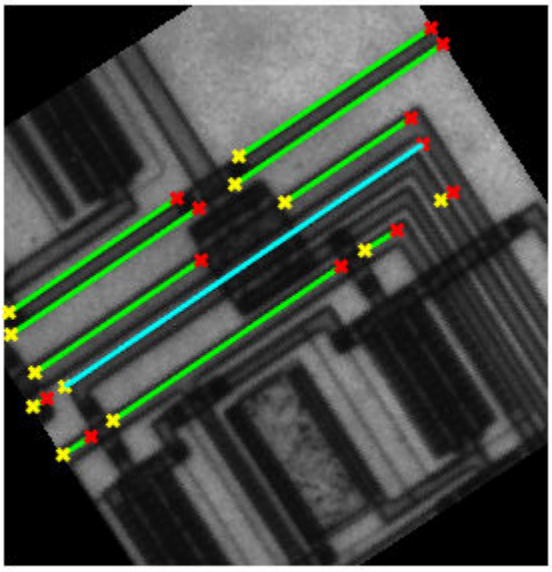
Find lines and plot them.

```
lines = houghlines(BW,T,R,P,'FillGap',5,'MinLength',7);
figure, imshow(rotI), hold on
max_len = 0;
for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');

   % Plot beginnings and ends of lines
   plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
   plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');

   % Determine the endpoints of the longest line segment
   len = norm(lines(k).point1 - lines(k).point2);
   if ( len > max_len)
      max_len = len;
      xy_long = xy;
   end
end
```



Highlight the longest line segment by coloring it cyan.

```
plot(xy_long(:,1),xy_long(:,2),'LineWidth',2,'Color','cyan');
```



Input Arguments

`BW` — Binary image

2-D logical matrix | 2-D numeric matrix

Binary image, specified as a 2-D logical matrix or 2-D numeric matrix. For numeric input, any nonzero pixels are considered to be `1` (`true`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`theta` — Line rotation angle

numeric matrix

Line rotation angle, in degrees, specified as a numeric matrix. The angle is measured between the *x*-axis and the `rho` vector.

Data Types: `double`

`rho` — Distance from origin to line

numeric matrix

Distance from the coordinate origin, specified as a numeric matrix. The coordinate origin is the top-left corner of the image (0,0).

Data Types: `double`

`peaks` — Row and column coordinates of Hough transform bins

numeric matrix

Row and column coordinates of Hough transform bins, specified as a numeric matrix.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `lines = houghlines(BW,T,R,P,'FillGap',5,'MinLength',7);`

`FillGap` — Distance between two line segments associated with the same Hough transform bin

`20` (default) | positive number

Distance between two line segments associated with the same Hough transform bin, specified as a positive number. When the distance between the line segments is less than the value specified, the `houghlines` function merges the line segments into a single line segment.

Data Types: `double`

`MinLength` — Minimum line length

`40` (default) | positive number

Minimum line length, specified as a positive number. `houghlines` discards lines that are shorter than the value specified.

Data Types: `double`

Output Arguments

`lines` — Detected lines

structure array

Detected lines, returned as a structure array whose length equals the number of merged line segments found. Each element of the structure array has these fields:

Field	Description
`point1`	Two element vector `[X Y]` specifying the coordinates of the end-point of the line segment
`point2`	Two element vector `[X Y]` specifying the coordinates of the end-point of the line segment
`theta`	Angle in degrees of the Hough transform bin
`rho`	`rho` axis position of the Hough transform bin

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `houghlines` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- The optional parameter names `'FillGap'` and `'MinLength'` must be compile-time constants. Their associated values need not be compile-time constants.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The optional parameter names `'FillGap'` and `'MinLength'` must be compile-time constants. Their associated values need not be compile-time constants.

See Also

`hough` | `houghpeaks`

Introduced before R2006a

houghpeaks

Identify peaks in Hough transform

Syntax

```
peaks = houghpeaks(H,numpeaks)
peaks = houghpeaks(H,numpeaks,Name,Value)
```

Description

`peaks = houghpeaks(H,numpeaks)` locates peaks in the Hough transform matrix, `H`, generated by the `hough` function. `numpeaks` specifies the maximum number of peaks to identify. The function returns `peaks` a matrix that holds the row and column coordinates of the peaks.

`peaks = houghpeaks(H,numpeaks,Name,Value)` controls aspects of the operation using name-value pair arguments.

Examples

Locate and Display Peaks in Hough Transform of Rotated Image

Read image into workspace.

```
I  = imread('circuit.tif');
```

Create binary image.

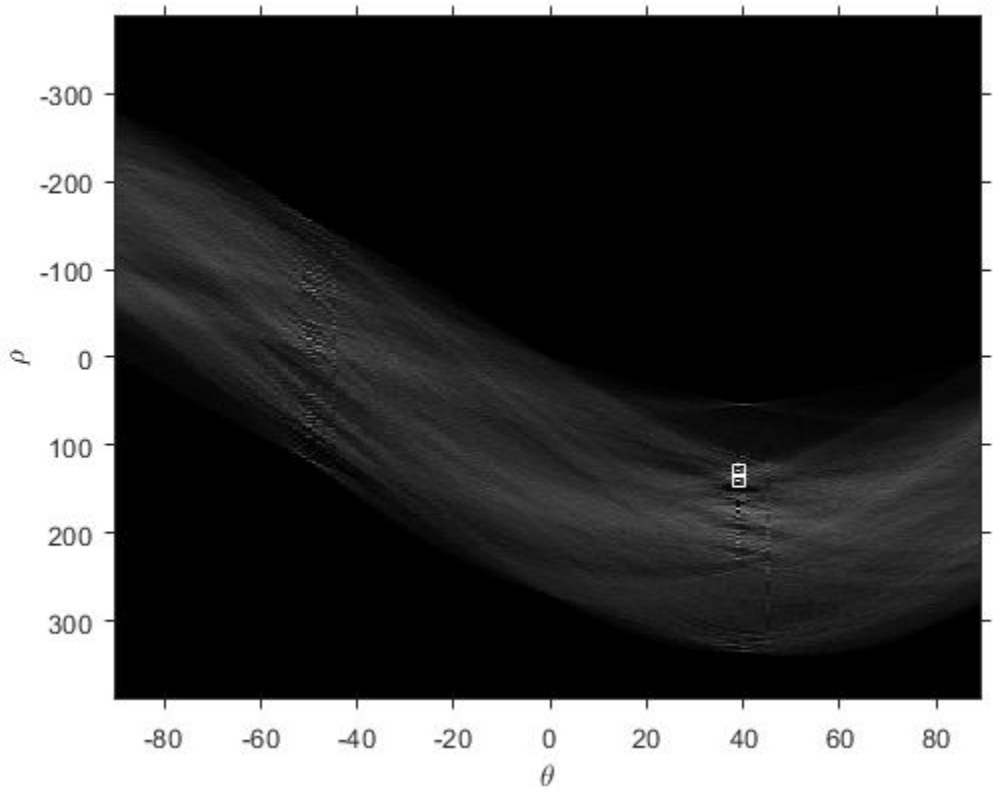
```
BW = edge(imrotate(I,50,'crop'),'canny');
```

Create Hough transform of image.

```
[H,T,R] = hough(BW);
```

Find peaks in the Hough transform of the image and plot them.

```
P  = houghpeaks(H,2);
imshow(H,[],'XData',T,'YData',R,'InitialMagnification','fit');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
plot(T(P(:,2)),R(P(:,1)),'s','color','white');
```



Input Arguments

`H` — Hough transform matrix

numeric array

Hough transform matrix, specified as a numeric array. The rows and columns correspond to `rho` and `theta` values. Use the `hough` function to create a Hough transform matrix.

Data Types: `double`

`numpeaks` — Maximum number of peaks to identify

1 (default) | positive integer

Maximum number of peaks to identify, specified as a positive integer.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `P = houghpeaks(H,2,'Threshold',15);`

`Threshold` — Minimum value to be considered a peak

`0.5*max(H(:))` (default) | nonnegative number

Minimum value to be considered a peak, specified as a nonnegative number.

Data Types: `double`

`NHoodSize` — Size of suppression neighborhood

2-element vector of positive odd integers

Size of the suppression neighborhood, specified as a 2-element vector of positive odd integers. The suppression neighborhood is the neighborhood around each peak that is set to zero after the peak is identified. The default value of `NHoodSize` is the smallest odd values greater than or equal to `size(H)/50`. The dimensions of `NHoodSize` must be smaller than the size of the Hough transform matrix, `H`.

Data Types: `double`

`Theta` — Hough transform theta values

`-90:89` (default) | numeric vector

Hough transform theta values, specified as a numeric vector returned by the `hough` function. Each element of the vector specifies the *theta* value for the corresponding column of the output matrix `H`. `houghpeaks` uses the *theta* values specified for peak suppression. Use the `hough` function to create a Hough transform matrix.

Note If you specify the `'Theta'` parameter as input to the `hough` function, you must specify the `theta` parameter with the `houghpeaks` function. Use the `theta` output value from the `hough` function as the `theta` input value for `houghpeaks`. Otherwise, peak suppression can result in unexpected results.

Data Types: `double`

Output Arguments

`peaks` — Row and column coordinates of found peaks

Q-by-2 matrix

Row and column coordinates of found peaks, returned as a *Q*-by-2 matrix. The value *Q* can range from `0` to `numpeaks`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `houghpeaks` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- The optional parameter names `'Threshold'` and `'NHoodSize'` must be compile-time constants. Their associated values need not be compile-time constants.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The optional parameter names `'Threshold'` and `'NHoodSize'` must be compile-time constants. Their associated values need not be compile-time constants.

See Also

`hough` | `houghlines`

Topics

“Hough Transform”

Introduced before R2006a

iccfind

Find ICC profiles

Syntax

```
profiles = iccfind(folder)
profiles = iccfind(folder,pattern)
[profiles,descriptions] = iccfind( ___ )
```

Description

`profiles = iccfind(folder)` finds and returns profile information for all of the International Color Consortium (ICC) profiles stored in the folder.

`profiles = iccfind(folder,pattern)` finds and returns profile information for ICC profiles in the folder whose profile names contain the value `pattern`. The function performs case-insensitive pattern matching to find the ICC profile with the desired profile name.

`[profiles,descriptions] = iccfind( ___ )` also returns the profile descriptions associated with every profile listed in `profiles`.

Examples

Find International Color Consortium Profiles

Use `iccroot` to find the default folder to which the International Color Consortium (ICC) profiles are stored.

```
folder = iccroot;
disp(folder)
```

```
C:\WINDOWS\System32\Spool\Drivers\Color
```

Find all the ICC profiles stored in default folder. Read the profile information of all ICC profiles as a cell array of structures.

```
profiles = iccfind(folder);
```

Display the size of `profiles` to know the number of ICC profiles available in the default folder.

```
size(profiles)
```

```
ans = 1×2

    23     1
```

Read profile information for the first ICC profile in `profiles`.

```
currentProfile = profiles{1}
```

```
currentProfile = struct with fields:
             Header: [1×1 struct]
           TagTable: {10×3 cell}
          Copyright: 'Copyright 2000 Adobe Systems Incorporated'
        Description: [1×1 struct]
    MediaWhitePoint: [0.9505 1 1.0891]
    MediaBlackPoint: [0 0 0]
             MatTRC: [1×1 struct]
        PrivateTags: {}
           Filename: 'C:\WINDOWS\System32\Spool\Drivers\Color\AdobeRGB1998.icc'
```

Inspect the `Description` field of `currentProfile`. The profile description is stored in the `String` field of `Description`.

```
currentProfile.Description
```

```
ans = struct with fields:
      String: 'Adobe RGB (1998)'
    Optional: [1×78 uint8]
```

Display the profile name of selected ICC profile.

```
ProfileName = currentProfile.Description.String
```

```
ProfileName =
'Adobe RGB (1998)'
```

Read Profile Name of ICC Profiles

Find all the ICC profiles stored in default folder. Read the profile information and the descriptions of all ICC profiles.

```
[~,descriptions] = iccfind(folder);
```

Display the `descriptions` of all the ICC profiles in `profiles`.

```
descriptions
```

```
descriptions = 23×1 cell
    {'Adobe RGB (1998)'                    }
    {'Apple RGB'                           }
    {'Coated FOGRA27 (ISO 12647-2:2004)'   }
    {'Coated FOGRA39 (ISO 12647-2:2004)'   }
    {'Coated GRACoL 2006 (ISO 12647-2:2004)'}
    {'ColorMatch RGB'                      }
    {'Japan Color 2001 Coated'             }
    {'Japan Color 2001 Uncoated'           }
    {'Japan Color 2002 Newspaper'          }
    {'Japan Color 2003 Web Coated'         }
    {'Japan Web Coated (Ad)'               }
    {'ProPhoto RGB'                        }
    {'Agfa : Swop Standard      '          }
    {'U.S. Sheetfed Coated v2'             }
    {'U.S. Sheetfed Uncoated v2'           }
    {'U.S. Web Coated (SWOP) v2'           }
    {'U.S. Web Uncoated v2'                }
    {'Uncoated FOGRA29 (ISO 12647-2:2004)'  }
```

```
    {'Web Coated FOGRA28 (ISO 12647-2:2004)'}
    {'Web Coated SWOP 2006 Grade 3 Paper'   }
    {'Web Coated SWOP 2006 Grade 5 Paper'   }
    {'change'                               }
    {'sRGB IEC61966-2.1'                    }
```

Find Specific ICC Profiles

Find ICC profiles with a specific pattern in the profile description. Specify the pattern to search in the profile description as `'rgb'`.

```
[profiles,descriptions] = iccfind(folder,'rgb');
```

Display the descriptions of all the ICC profiles in `profiles`. The function returns the profile information and the descriptions for ICC profiles containing the pattern `'rgb'` in profile description.

```
descriptions
```

```
descriptions = 5×1 cell
    {'Adobe RGB (1998)' }
    {'Apple RGB'        }
    {'ColorMatch RGB'   }
    {'ProPhoto RGB'     }
    {'sRGB IEC61966-2.1'}
```

Input Arguments

`folder` — Path to ICC profiles

character vector | string scalar

Path to ICC profiles, specified as a character vector or string scalar denotes the folder in which the ICC profiles are stored. The ICC profiles can have the file extension `.icc` or `.icm`.

Data Types: `char` | `string`

`pattern` — Search key

character vector | string scalar

Search key, specified as a character vector or string scalar. You can use this search key to find ICC profiles whose profile names contain the search key. The ICC profile names are stored in the profile descriptions.

Data Types: `char` | `string`

Output Arguments

`profiles` — Set of profile information

cell array of structures

Set of profile information, returned as a cell array of structures. Each structure in the cell array contains profile information for an ICC profile in the folder.

Data Types: `cell`

`descriptions` — Profile descriptions
cell array of character vectors

Profile descriptions, returned as a cell array of character vectors. Each profile description is the localized version of the ICC profile name.

Data Types: `cell`

Tips

- To improve performance, `iccfind` caches copies of the ICC profiles in memory. Adding or modifying profiles might not change the results of `iccfind`. To clear the cache, use the `clear functions` command.

References

[1] Abhay, S. "ICC Color Management: Architecture and Implementation." Color Image Processing: Methods and Applications (R. Lukac and K. N. Plataniotis, eds.). CRC Press, 2006.

See Also

`iccread` | `iccroot` | `iccwrite`

Introduced before R2006a

iccread

Read ICC profile

Syntax

```
profile = iccread(filename)
```

Description

`profile = iccread(filename)` reads the International Color Consortium (ICC) color profile data from the file specified by the input `filename`.

Note `iccread` can read profiles that conform with either Version 2 (ICC.1:2001-04) or Version 4 (ICC.1:2001-12) of the ICC specification. For more information about ICC profiles, visit the ICC website, https://www.color.org.

Examples

Read ICC Profile for Typical PC Computer Monitor

Read the International Color Consortium (ICC) profile that describes a typical PC computer monitor.

```
profile = iccread('sRGB.icm')
```

```
profile = struct with fields:
              Header: [1x1 struct]
            TagTable: {17x3 cell}
           Copyright: 'Copyright (c) 1999 Hewlett-Packard Company'
         Description: [1x1 struct]
      MediaWhitePoint: [0.9505 1 1.0891]
      MediaBlackPoint: [0 0 0]
        DeviceMfgDesc: [1x1 struct]
      DeviceModelDesc: [1x1 struct]
      ViewingCondDesc: [1x1 struct]
    ViewingConditions: [1x1 struct]
           Luminance: [76.0365 80 87.1246]
         Measurement: [1x1 struct]
          Technology: 'Cathode Ray Tube Display'
              MatTRC: [1x1 struct]
         PrivateTags: {}
            Filename: 'sRGB.icm'
```

Determine the source color space. The profile header provides general information about the profile, such as its class, color space, and PCS.

```
profile.Header.ColorSpace
```

```
ans =
'RGB'
```

Input Arguments

`filename` — Name of the file containing ICC profile

character vector | string scalar

Name of the file containing ICC profile, specified as a character vector or string scalar. The file can be either an ICC profile file or a TIFF file containing an embedded ICC profile. To determine if a TIFF file contains an embedded ICC profile, use the `imfinfo` function to get information about the file and look for the `ICCProfileOffset` field in the output.

Note If you specify only the file name without its path, `iccread` searches for the file in the current folder, a folder on the MATLAB path, or in the folder returned by `iccroot` in that order.

Data Types: `char` | `string`

Output Arguments

`profile` — ICC profile data

structure array

ICC profile data, returned as a structure array. The fields contain the data structures (called tags) defined in the ICC specification. The number of fields in `profile` depends on the profile class and the choices made by the profile creator. `iccread` returns all the tags for a given profile, both public and private. Private tags and certain public tags are left as encoded `uint8` data. The following table lists fields that are found in any profile structure generated by `iccread`.

Field	Data Type	Description
`Header`	1-by-1 struct array	Profile header fields.
`TagTable`	*n*-by-3 cell array	Profile tag table.
`Copyright`	Character vector	Profile copyright notice.
`Description`	1-by-1 struct array	Profile description. The `String` field in this structure contains a character vector describing the profile.
`MediaWhitePoint`	`double` array	XYZ stimulus values of the device's media white point.
`PrivateTags`	*m*-by-2 cell array	Contents of all the private tags or tags not defined in the ICC specifications. The tag signatures are in the first column, and the contents of the tags are in the second column. The `iccread` leaves the contents of these tags in unsigned 8-bit encoding.
`Filename`	Character vector	Name of the file containing the profile.

Also, `profile` might contain one or more of the following transforms:

- Three-component, matrix-based transform: A simple transform that is often used to transform between the RGB and XYZ color spaces. If this transform is present, `profile` contains a field called `MatTRC`.
- N-component look-up-table (LUT) based transform: A transform that is used for transforming between color spaces that have a more complex relationship. This type of transform is found in any of the following fields in `profile`:

`AToB0`	`BToA0`	`Preview0`
`AToB1`	`BToA1`	`Preview1`
`AToB2`	`BToA2`	`Preview2`
`AToB3`	`BToA3`	`Gamut`

Data Types: `struct`

Tips

- ICC profiles provide color management systems with the information necessary to convert color data between native device color spaces and device-independent color spaces, called the Profile Connection Space (PCS). You can use the profile as the source or destination profile with the `makecform` or `applycform` functions to compute color space transformations.

See Also

`applycform` | `iccfind` | `iccroot` | `iccwrite` | `isicc` | `makecform`

Introduced before R2006a

iccroot

Find system default ICC profile repository

Syntax

```
rootdir = iccroot
```

Description

`rootdir = iccroot` returns the system directory containing International Color Consortium (ICC) profiles. Additional profiles can be stored in other directories, but this is the default location used by the color management system.

Note This function is only supported on Windows and Mac OS X platforms.

Examples

Find System Directory Containing ICC Profiles

Find the default location of International Color Consortium (ICC) profile repository.

```
rootdir = iccroot
```

```
rootdir =
'C:\WINDOWS\System32\Spool\Drivers\Color'
```

See Also

`iccfind` | `iccread` | `iccwrite`

Introduced before R2006a

iccwrite

Write ICC color profile data

Syntax

```
outProfile = iccwrite(inProfile,filename)
```

Description

`outProfile = iccwrite(inProfile,filename)` writes an International Color Consortium (ICC) profile data in structure `inProfile` to the file specified by `filename`.

You can use this function to modify fields in an ICC profile data structure and write it to a file with name `filename`. For example, some applications use the string field in profile description to present choices to users. The ICC recommends modifying the profile description in ICC profile data before writing the data to a file. Each profile is recommended to have a unique profile description. You can therefore, use the `iccwrite` function to modify the profile description.

Note `iccwrite` can write profiles that conform with either Version 2 (ICC.1:2001-04) or Version 4 (ICC.1:2001-12) of the ICC specification. To determine the version of the ICC specification, use `version` field in the `Header` of profile data structure. Based on the version, format the `inProfile` for output. For more information about ICC profiles, visit the ICC website, https://www.color.org.

Examples

Write ICC Profile Data to a File

Read an ICC profile data into the workspace and display the profile name.

```
inProfile = iccread('monitor.icm');
inProfile.Description.String
```

```
ans =
'sgC4_050102_d50.pf'
```

Change the profile name to `'monitor_RGB'`.

```
inProfile.Description.String = 'monitor_RGB';
```

Write the updated ICC profile data to a new file and display the corresponding output ICC profile data. The new file is created in the current working folder.

```
outProfile = iccwrite(inProfile,'monitorcolor.icm')
```

```
outProfile = struct with fields:
            Header: [1x1 struct]
          TagTable: {11x3 cell}
       Description: [1x1 struct]
    MediaWhitePoint: [0.9642 1.0000 0.8249]
```

```
       Copyright: 'Copyright  Sequel Imaging Inc. 1996-2001'
MediaBlackPoint: [0 0 0]
         MatTRC: [1x1 struct]
    PrivateTags: {'vcgt'  [118 99 103 116 0 0 0 0 0 0 0 0 0 3 1 0 0 ... ]}
       Filename: 'monitorcolor.icm'
```

Verify the modified description in output ICC profile data.

```
outProfile.Description.String
```

```
ans =
'monitor_RGB'
```

Input Arguments

`inProfile` — Input ICC profile data

structure array

Input ICC profile data, specified as a structure array represents an ICC profile in the data format returned by `iccread`. The ICC profile data must contain all the tags and fields required by the ICC profile specification. The input ICC profile data is written to `filename`.

Data Types: `struct`

`filename` — Name of the file to write ICC profile data

character vector | string scalar

Name of the file to write ICC profile data, specified as a character vector or string scalar. Depending on the operating system, you can save the file with an extension `.icc` or `.icm`.

Note If you specify only the file name without its path, `iccwrite` writes the file to current working folder.

Data Types: `char` | `string`

Output Arguments

`outProfile` — Output ICC profile data

structure array

Output ICC profile data, returned as a structure array gives the ICC profile data written to the file `filename`.

Data Types: `struct`

Tips

`iccwrite` does not perform automatic conversions from one version of the ICC specification to another. Do the conversion manually by adding fields or modifying fields in ICC profile data. Use `isicc` to validate the converted ICC profile data.

See Also

`applycform` | `iccread` | `isicc` | `makecform`

Introduced before R2006a

idct2

2-D inverse discrete cosine transform

Syntax

```
B = idct2(A)
B = idct2(A,m,n)
B = idct2(A,[m n])
```

Description

`B = idct2(A)` returns the two-dimensional inverse discrete cosine transform (DCT) of `A`.

`B = idct2(A,m,n)` and

`B = idct2(A,[m n])` pads `A` with `0`s to size `m`-by-`n` before applying the inverse transformation. If `m` or `n` is smaller than the corresponding dimension of `A`, then `idct2` crops `A` before the transformation.

Examples

Remove High Frequencies in Image using 2-D DCT

Read an image into the workspace, then convert the image to grayscale.

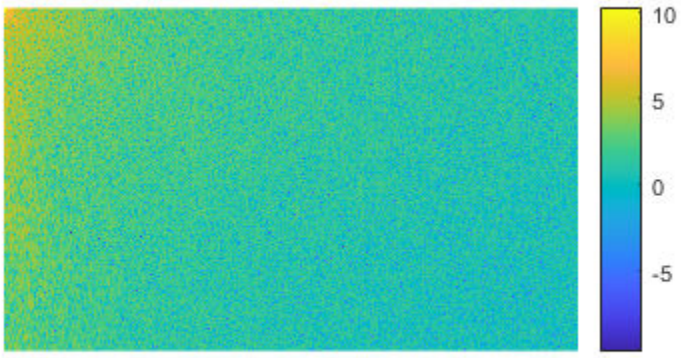
```
RGB = imread('autumn.tif');
I = im2gray(RGB);
```

Perform a 2-D DCT of the grayscale image using the `dct2` function.

```
J = dct2(I);
```

Display the transformed image using a logarithmic scale. Notice that most of the energy is in the upper left corner.

```
imshow(log(abs(J)),[])
colormap parula
colorbar
```



Set values less than magnitude 10 in the DCT matrix to zero.

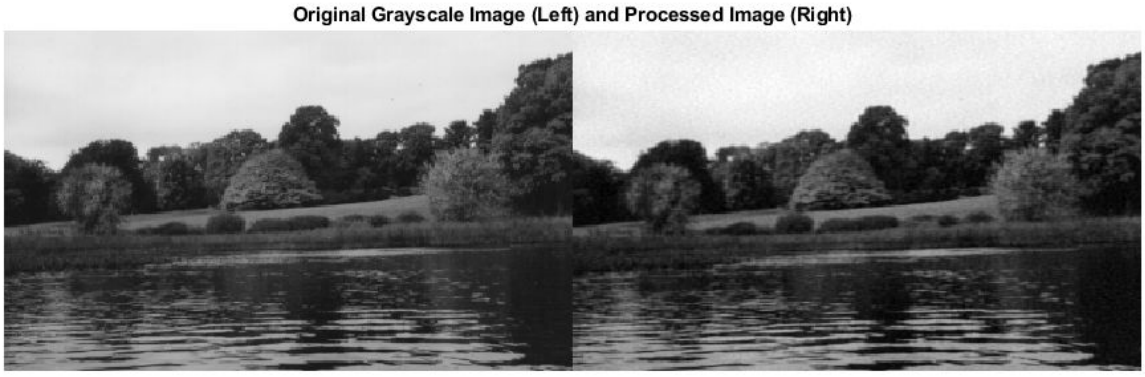
```
J(abs(J) < 10) = 0;
```

Reconstruct the image using the inverse DCT function `idct2`. Rescale the values to the range [0, 1] expected of images of data type `double`.

```
K = idct2(J);
K = rescale(K);
```

Display the original grayscale image alongside the processed image. The processed image has fewer high frequency details, such as in the texture of the trees.

```
montage({I,K})
title('Original Grayscale Image (Left) and Processed Image (Right)');
```



Input Arguments

A — Input matrix

2-D numeric matrix

Input matrix, specified as a 2-D numeric matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

m — Number of image rows

`size(A,1)` (default) | positive integer

Number of image rows, specified as a positive integer. `idct2` pads image `A` with `0`s or truncates image `A` so that it has `m` rows. By default, `m` is equal to `size(A,1)`.

n — Number of image columns

`size(A,2)` (default) | positive integer

Number of image columns, specified as a positive integer. `idct2` pads image `A` with `0`s or truncates image `A` so that it has `n` columns. By default, `n` is equal to `size(A,2)`

Output Arguments

B — Transformed matrix

`m`-by-`n` numeric matrix

Transformed matrix using a two-dimensional discrete cosine transform, returned as an `m`-by-`n` numeric matrix.

Data Types: `double`

Tips

- For any matrix `A`, `idct2(dct2(A))` equals `A` to within round-off error.

Algorithms

`idct2` computes the two-dimensional inverse DCT using:

$$A_{mn} = \sum_{p=0}^{M-1}\sum_{q=0}^{N-1} \alpha_p \alpha_q B_{pq} \cos\frac{\pi(2m+1)p}{2M} \cos\frac{\pi(2n+1)q}{2N}, \quad \begin{matrix} 0 \le m \le M-1 \\ 0 \le n \le N-1 \end{matrix},$$

where

$$\alpha_p = \begin{cases} \frac{1}{\sqrt{M}}, & p = 0 \\ \sqrt{\frac{2}{M}}, & 1 \le p \le M-1 \end{cases}$$

and

$$\alpha_q = \begin{cases} \frac{1}{\sqrt{N}}, & q = 0 \\ \sqrt{\frac{2}{N}}, & 1 \le q \le N-1 \end{cases}.$$

References

[1] Jain, A. K., *Fundamentals of Digital Image Processing,* Englewood Cliffs, NJ, Prentice Hall, 1989, pp. 150-153.

[2] Pennebaker, W. B., and J. L. Mitchell, *JPEG: Still Image Data Compression Standard,* New York, Van Nostrand Reinhold, 1993.

See Also

`dct2` | `dctmtx` | `fft2` | `ifft2`

Introduced before R2006a

ifanbeam

Inverse fan-beam transform

Syntax

```
I = ifanbeam(F,D)
I = ifanbeam(F,D,Name,Value)
[I,H] = ifanbeam( ___ )
```

Description

`I = ifanbeam(F,D)` reconstructs the image `I` from fan-beam projection data in `F`. Each column of `F` contains fan-beam projection data at one rotation angle. The angle between sensors is assumed to be uniform and equal to the increment between fan-beam rotation angles. `D` is the distance from the fan-beam vertex to the center of rotation.

`I = ifanbeam(F,D,Name,Value)` uses name-value arguments to control various aspects of the reconstruction.

`[I,H] = ifanbeam( ___ )` also returns the frequency response of the filter, `H`.

Examples

Recreate Image from Fan-beam Transformation

Create a sample image. The `phantom` function creates a phantom head image.

```
ph = phantom(128);
```

Create a fan-beam transformation of the phantom head image.

```
d = 100;
F = fanbeam(ph,d);
```

Reconstitute the phantom head image from the fan-beam representation. Display the original image and the reconstituted image.

```
I = ifanbeam(F,d);
imshow(ph)
```

```
figure
imshow(I);
```

Generate Fan-beam with Fancoverage Set to Minimal

Create a sample image. The phantom function creates a phantom head image.

```
ph = phantom(128);
```

Create a radon transformation of the image.

```
P = radon(ph);
```

Convert the transformation from parallel beam projection to fan-beam projection.

```
[F,obeta,otheta] = para2fan(P,100,...
                            'FanSensorSpacing',0.5,...
                            'FanCoverage','minimal',...
                            'FanRotationIncrement',1);
```

Reconstitute the image from fan-beam data.

```
phReconstructed = ifanbeam(F,100,...
                           'FanSensorSpacing',0.5,...
                           'Filter','Shepp-Logan',...
                           'OutputSize',128,...
                           'FanCoverage','minimal',...
                           'FanRotationIncrement',1);
```

Display the original and the transformed image.

```
imshow(ph)
```

```
figure
imshow(phReconstructed)
```

Input Arguments

F — Fan-beam projection data

numSensors-by-*numAngles* numeric matrix

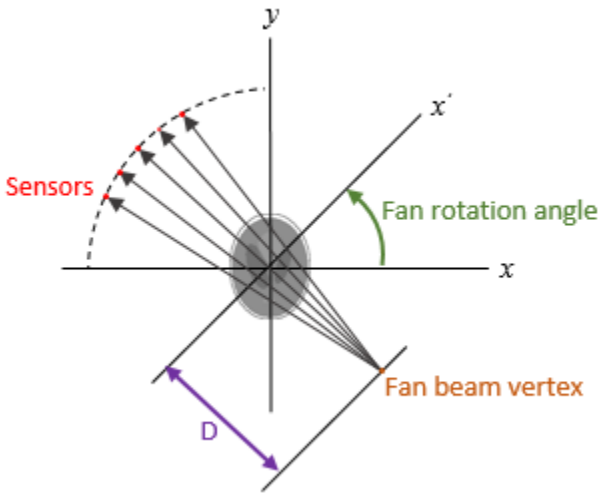
Fan-beam projection data, specified as a *numSensors*-by-*numAngles* numeric matrix. *numSensors* is the number of fan-beam sensors and *numAngles* is the number of fan-beam rotation angles. Each column of F contains the fan-beam sensor samples at one rotation angle.

Data Types: `double` | `single`

`D` — Distance from fan-beam vertex to center of rotation

positive number

Distance in pixels from the fan-beam vertex to the center of rotation, specified as a positive number. `ifanbeam` assumes that the center of rotation is the center point of the projections, which is defined as `ceil(size(F,1)/2)`.



Data Types: `double` | `single`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `I = ifanbeam(F,D,FanRotationIncrement=5)` specifies a fan rotation increment of 5 degrees.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `I = ifanbeam(F,D,"FanRotationIncrement",5)` specifies a fan rotation increment of 5 degrees.

`FanCoverage` — Range of fan-beam rotation

`"cycle"` (default) | `"minimal"`

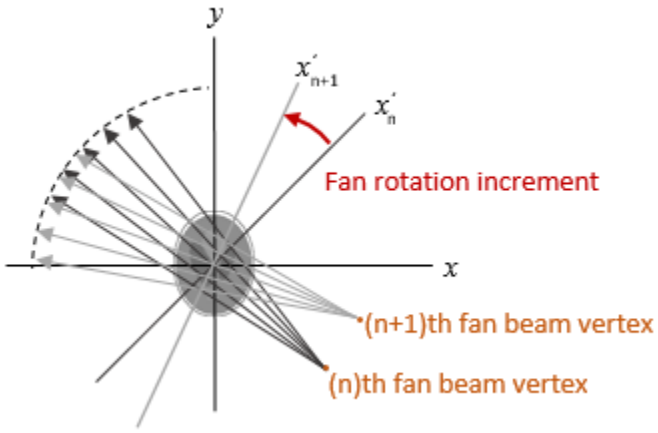
Range of fan-beam rotation, specified as `"cycle"` or `"minimal"`.

- `"cycle"` — Rotate through the full range [0, 360) degrees.
- `"minimal"` — Rotate through the minimum range necessary to represent the object.

`FanRotationIncrement` — Fan-beam rotation angle increment

1 (default) | positive scalar

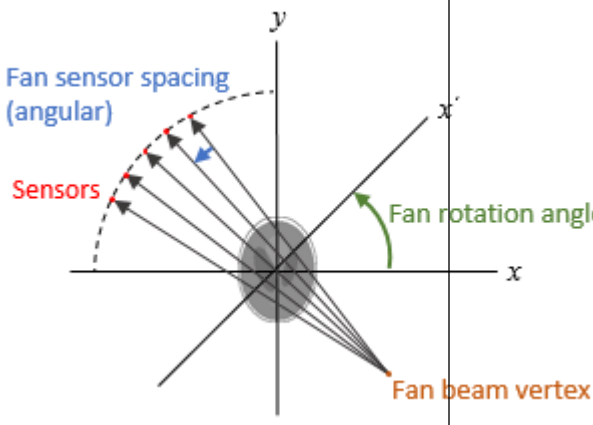
Fan-beam rotation angle increment in degrees, specified as a positive scalar.

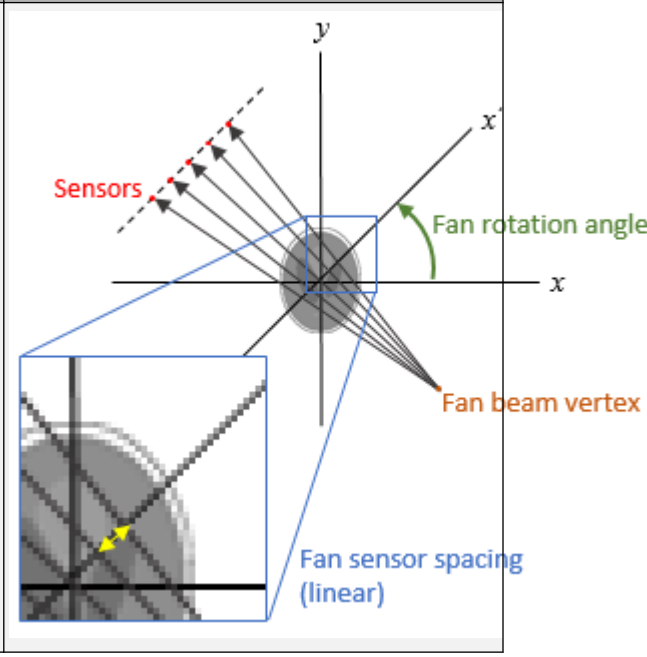


Data Types: `double`

`FanSensorGeometry` — Fan-beam sensor positioning
`"arc"` (default) | `"line"`

Fan-beam sensor positioning, specified as `"arc"` or `"line"`.

Value	Meaning	Diagram
`"arc"`	Sensors are spaced at equal angles along a circular arc. The center of the arc is the fan-beam vertex. `FanSensorSpacing` defines the angular spacing in degrees.	

Value	Meaning	Diagram
`"line"`	Sensors are spaced at equal distances along a line that is parallel to the x' axis. The closest sensor is distance `D` from the center of rotation. `FanSensorSpacing` defines the distance between fan-beams on the x' axis, in pixels.	

`FanSensorSpacing` — Fan-bean sensor spacing

1 (default) | positive scalar

Fan-bean sensor spacing, specified as a positive scalar.

- If `FanSensorGeometry` is `"arc"`, then `FanSensorSpacing` defines the angular spacing in degrees.
- If `FanSensorGeometry` is `"line"`, then `FanSensorSpacing` defines the linear distance between fan-beams, in pixels. Linear spacing is measured on the x' axis.

Data Types: `double`

`Filter` — Filter

`"Ram-Lak"` (default) | `"Shepp-Logan"` | `"Cosine"` | `"Hamming"` | `"Hann"` | `"None"`

Filter to use for frequency domain filtering, specified as one of the values in the table. For more information, see `iradon`.

Value	Description
`"Ram-Lak"`	Cropped Ram-Lak or ramp filter. The frequency response of this filter is | `f` |. Because this filter is sensitive to noise in the projections, one of the filters listed below might be preferable. These filters multiply the Ram-Lak filter by a window that de-emphasizes high frequencies.
`"Shepp-Logan"`	Multiplies the Ram-Lak filter by a `sinc` function
`"Cosine"`	Multiplies the Ram-Lak filter by a `cosine` function
`"Hamming"`	Multiplies the Ram-Lak filter by a Hamming window
`"Hann"`	Multiplies the Ram-Lak filter by a Hann window

Value	Description
"None"	No filtering. ifanbeam returns unfiltered data.

Data Types: char | string

FrequencyScaling — Scale factor
1 (default) | positive number in the range (0, 1]

Scale factor for rescaling the frequency axis, specified as a positive number in the range (0, 1]. If FrequencyScaling is less than 1, then the filter is compressed to fit into the frequency range [0, FrequencyScaling], in normalized frequencies; all frequencies above FrequencyScaling are set to 0. For more information, see iradon.

Data Types: double

Interpolation — Type of interpolation
"Linear" (default) | "nearest" | "spline" | "pchip"

Type of interpolation used between the parallel-beam and fan-beam data, specified as one of these values.

"nearest" — Nearest-neighbor

"linear" — Linear (the default)

"spline" — Piecewise cubic spline

"pchip" — Piecewise cubic Hermite (PCHIP)

Data Types: char | string

OutputSize — Size of reconstructed image
positive integer

Size of the reconstructed image, specified as a positive integer. The image has an equal number of rows and columns.

If you specify OutputSize, then ifanbeam reconstructs a smaller or larger portion of the image but does not change the scaling of the data.

Note If the projections were calculated with the fanbeam function, then the reconstructed image might not be the same size as the original image.

If you do not specify OutputSize, then the size is calculated automatically by:

```
OutputSize = 2*floor(size(R,1)/(2*sqrt(2)))
```

where R is the length of parallel-beam projection data used by iradon. For more information, see “Algorithms” on page 1-1209.

Data Types: double

Output Arguments

`I` — Reconstructed image

2-D numeric matrix

Reconstructed image, specified as a 2-D numeric matrix.

`H` — Frequency response

numeric vector

Frequency response of the filter, returned as a numeric vector.

Data Types: `double`

Tips

- To perform an inverse fan-beam reconstruction, you must give `ifanbeam` the same parameters that were used to calculate the projection data, `F`. If you use `fanbeam` to calculate the projection, then make sure the parameters are consistent when calling `ifanbeam`.

Algorithms

`ifanbeam` converts the fan-beam data to parallel beam projections and then uses the filtered back projection algorithm to perform the inverse Radon transform. The filter is designed directly in the frequency domain and then multiplied by the FFT of the projections. The projections are zero-padded to a power of 2 before filtering to prevent spatial domain aliasing and to speed up the FFT.

References

[1] Kak, Avinash C., and Malcolm Slaney. *Principles of Computerized Tomographic Imaging*. New York: IEEE Press, 1988.

See Also

`fan2para` | `fanbeam` | `iradon` | `para2fan` | `phantom` | `radon`

Introduced before R2006a

illumgray

Estimate illuminant using gray world algorithm

Syntax

```
illuminant = illumgray(A)
illuminant = illumgray(A,percentile)
illuminant = illumgray( ___ ,Name,Value)
```

Description

`illuminant = illumgray(A)` estimates the illumination of the scene in RGB image `A` by assuming that the average color of the scene is gray.

`illuminant = illumgray(A,percentile)` estimates the illumination, excluding the specified bottom and top percentiles of pixel values.

`illuminant = illumgray( ___ ,Name,Value)` estimates the illumination using name-value pairs to control additional options.

Examples

Correct White Balance Using Gray World Algorithm

Open an image and display it. Specify an optional magnification to shrink the size of the displayed image.

```
A = imread('foosball.jpg');
figure
imshow(A,'InitialMagnification',25)
title('Original Image')
```

The gray world algorithm assumes that the RGB values are linear. However, the JPEG file format saves images in the gamma-corrected sRGB color space. Undo the gamma correction by using the `rgb2lin` function.

```
A_lin = rgb2lin(A);
```

Estimate the scene illumination, excluding the top and bottom 10% of pixels. Because the input image has been linearized, `illumgray` returns the illuminant in the linear RGB color space.

```
percentiles = 10;
illuminant = illumgray(A_lin,percentiles)
```

```
illuminant = 1×3

    0.2206    0.2985    0.5219
```

The third coefficient of `illuminant` is the largest, which is consistent with the blue tint of the image.

Correct colors by providing the estimated illuminant to the `chromadapt` function.

```
B_lin = chromadapt(A_lin,illuminant,'ColorSpace','linear-rgb');
```

To display the white-balanced image correctly on the screen, apply gamma correction by using the `lin2rgb` function.

```
B = lin2rgb(B_lin);
```

Display the corrected image, setting the optional magnification.

```
figure
imshow(B,'InitialMagnification',25)
title(['White-Balanced Image Using Gray World with percentiles=[' ...
    num2str(percentiles) ' ' num2str(percentiles) ']'])
```

Input Arguments

A — RGB image

m-by-*n*-by-3 numeric array

RGB image, specified as an *m*-by-*n*-by-3 numeric array.

Data Types: `single` | `double` | `uint8` | `uint16`

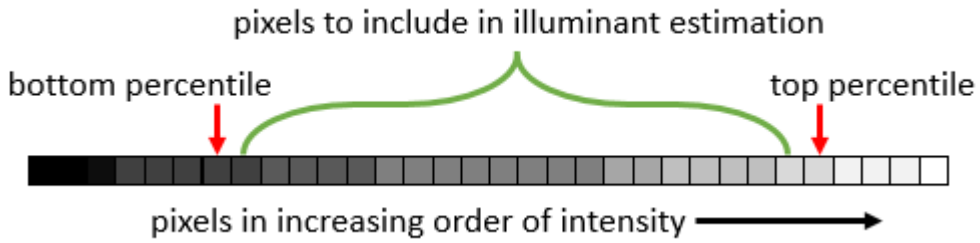
`percentile` — Percentile of pixels to exclude

1 (default) | numeric scalar | 2-element numeric vector

Percentile of pixels to exclude from the illuminant estimation, specified as a numeric scalar or 2-element numeric vector. Excluding pixels helps prevent overexposed and underexposed pixels from skewing the estimation.

- If `percentile` is a scalar, the same value is used for both the bottom percentile and the top percentile. In this case, `percentile` must be in the range [0, 50] so that the sum of the bottom and top percentiles does not exceed 100.
- If `percentile` is a 2-element vector, the first element is the bottom percentile and the second element is the top percentile. Both percentiles must be in the range [0, 100) and their sum cannot exceed 100.

The following image indicates the range of pixels that are included in the illuminant estimation. The selection is separate for each color channel.



Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `illuminant = illumgray(I,'Mask',m)` estimates the scene illuminant using a subset of pixels in image `I`, selected according to a binary mask, `m`.

`Mask` — Image mask
m-by-*n* logical or numeric array

Image mask, specified as the comma-separated pair consisting of `'Mask'` and an *m*-by-*n* logical or numeric array. The mask indicates which pixels of the input image `A` to use when estimating the illuminant. The computation excludes pixels in `A` that correspond to a mask value of 0. By default, the mask has all 1s, and all pixels in `A` are included in the estimation.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`Norm` — Type of vector norm (p-norm)
`1` (default) | positive numeric scalar

Type of vector norm (p-norm), specified as the comma-separated pair consisting of `'Norm'` and a positive numeric scalar. The p-norm affects the calculation of the average RGB value in the input image `A`. The p-norm is defined as sum(abs(x)^p) ^ (1/p).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`illuminant` — Estimate of scene illumination

3-element numeric row vector

Estimate of scene illumination, returned as a 3-element numeric row vector. The three elements correspond to the red, green, and blue values of the illuminant.

Data Types: `double`

Tips

- The gray world algorithm assumes uniform illumination and linear RGB values. If you are working with nonlinear sRGB or Adobe RGB images, use the `rgb2lin` function to undo the gamma correction before using `illumgray`. Also, make sure to convert the chromatically adapted image back to sRGB by using the `lin2rgb` function.
- When you specify `Mask on page 1-0` , the bottom percentile and top percentile apply to the masked image.
- You can adjust the color balance of the image to remove the scene illumination by using the `chromadapt` function.

References

[1] Ebner, Marc. "The Gray World Assumption." *Color Constancy*. Chichester, West Sussex: John Wiley & Sons, 2007.

See Also

`chromadapt` | `illumpca` | `illumwhite` | `lin2rgb` | `rgb2lin`

Introduced in R2017b

illumpca

Estimate illuminant using principal component analysis (PCA)

Syntax

```
illuminant = illumpca(A)
illuminant = illumpca(A,percentage)
illuminant = illumpca( ___ ,'Mask',mask)
```

Description

`illuminant = illumpca(A)` estimates the illumination of the scene in RGB image `A` from large color differences using principal component analysis (PCA).

`illuminant = illumpca(A,percentage)` estimates the illumination using the specified percentage of darkest and brightest pixels.

`illuminant = illumpca( ___ ,'Mask',mask)` estimates the illumination using only the pixels within the ROI defined by a binary mask.

Examples

Correct White Balance Using Principal Component Analysis

Open an image and display it. Specify an optional magnification to shrink the size of the displayed image.

```
A = imread('foosball.jpg');
figure
imshow(A,'InitialMagnification',25)
title('Original Image')
```

Principal component analysis assumes that the RGB values are linear. However, the JPEG file format saves images in the gamma-corrected sRGB color space. Undo the gamma correction by using the `rgb2lin` function.

```
A_lin = rgb2lin(A);
```

Estimate the scene illumination from the darkest and brightest 3.5% of pixels (the default percentage). Because the input image is linear, the `illumpca` function returns the illuminant in the linear RGB color space,

```
illuminant = illumpca(A_lin)
```

```
illuminant = 1×3

    0.4075    0.5547    0.7254
```

The third coefficient of `illuminant` is the largest, which is consistent with the blue tint of the image.

Correct colors by providing the estimated illuminant to the `chromadapt` function.

```
B_lin = chromadapt(A_lin,illuminant,'ColorSpace','linear-rgb');
```

To display the white-balanced image correctly on the screen, apply gamma correction by using the `lin2rgb` function.

```
B = lin2rgb(B_lin);
```

Display the corrected image, setting the optional magnification.

```
figure
imshow(B,'InitialMagnification',25)
title('White-Balanced Image using Principal Component Analysis')
```

Input Arguments

`A` — RGB image

m-by-*n*-by-3 numeric array

RGB image, specified as an *m*-by-*n*-by-3 numeric array.

Data Types: `single` | `double` | `uint8` | `uint16`

`percentage` — Percentage of pixels to retain

`3.5` (default) | numeric scalar

Percentage of pixels to retain for the illuminant estimation, specified as a numeric scalar in the range (0, 50].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`mask` — Image mask

m-by-*n* logical or numeric matrix

Image mask, specified as an *m*-by-*n* logical or numeric matrix. The mask indicates which pixels of the input image `A` to use when estimating the illuminant. The computation excludes pixels in `A` that correspond to a mask value of 0. By default, the mask has all 1s, and all pixels in `A` are included in the estimation.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

`illuminant` — Estimate of scene illumination

3-element numeric row vector

Estimate of scene illumination, returned as a 3-element numeric row vector. The three elements correspond to the red, green, and blue values of the illuminant.

Data Types: `double`

Tips

- The algorithm assumes uniform illumination and linear RGB values. If you are working with nonlinear sRGB or Adobe RGB images, use the `rgb2lin` function to undo the gamma correction before using `illumpca`. Also, make sure to convert the chromatically adapted image back to sRGB or Adobe RGB by using the `lin2rgb` function.

Algorithms

Pixel colors are represented as vectors in the RGB color space. The algorithm orders colors according to the brightness, or norm, of their projection on the average color in the image. The algorithm retains only the darkest and brightest colors, according to this ordering. Principal component analysis (PCA) is then performed on the subset of colors. The first component of PCA indicates the illuminant estimate.

References

[1] Cheng, Dongliang, Dilip K. Prasad, and Michael S. Brown. "Illuminant Estimation for Color Constancy: Why spatial-domain methods work and the role of the color distribution." *Journal of the Optical Society of America A*. Vol. 31, Number 5, 2014, pp. 1049–1058.

See Also

`chromadapt` | `illumgray` | `illumwhite` | `lin2rgb` | `rgb2lin`

Introduced in R2017b

illumwhite

Estimate illuminant using White Patch Retinex algorithm

Syntax

```
illuminant = illumwhite(A)
illuminant = illumwhite(A,topPercentile)
illuminant = illumwhite( ___ ,'Mask',mask)
```

Description

`illuminant = illumwhite(A)` estimates the scene illumination in RGB image `A` by assuming that the top 1% brightest red, green, and blue values represent the color white.

`illuminant = illumwhite(A,topPercentile)` estimates the illumination using the `topPercentile` percentage brightest red, green, and blue values.

`illuminant = illumwhite( ___ ,'Mask',mask)` estimates the illumination using only the pixels within the ROI defined by a binary mask.

Examples

Correct White Balance Using White Patch Retinex Algorithm

Open an image and display it. Specify an optional magnification to shrink the size of the displayed image.

```
A = imread('foosball.jpg');
figure
imshow(A,'InitialMagnification',25)
title('Original Image')
```

The JPEG file format saves images in the gamma-corrected sRGB color space. Undo the gamma correction by using the `rgb2lin` function.

```
A_lin = rgb2lin(A);
```

Estimate the scene illumination from the top 5% brightest pixels. Because the input image has been linearized, the `illumwhite` function returns the illuminant in the linear RGB color space.

```
topPercentile = 5;
illuminant = illumwhite(A,topPercentile)
```

```
illuminant = 1×3

    0.7333    0.8314    1.0000
```

The third coefficient of `illuminant` is the largest, which is consistent with the blue tint of the image.

Correct colors by providing the estimated illuminant to the `chromadapt` function.

```
B_lin = chromadapt(A_lin,illuminant,'ColorSpace','linear-rgb');
```

To display the white-balanced image correctly on the screen, apply gamma correction by using the `lin2rgb` function.

```
B = lin2rgb(B_lin);
```

Display the corrected image, setting the optional magnification.

```
figure
imshow(B,'InitialMagnification',25)
title(['White-Balanced Image using White Patch with topPercentile=' ...
    num2str(topPercentile)])
```

Input Arguments

A — RGB image

m-by-*n*-by-3 numeric array

RGB image, specified as an *m*-by-*n*-by-3 numeric array.

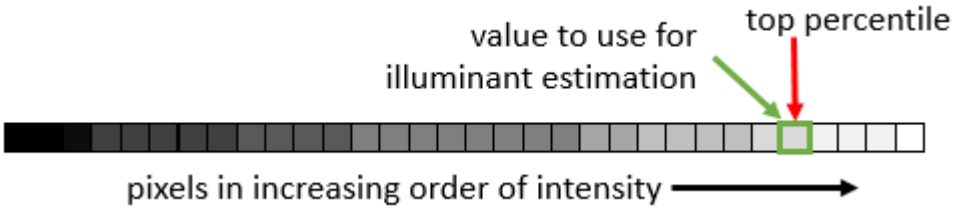
Data Types: `single` | `double` | `uint8` | `uint16`

topPercentile — Percentile of brightest colors

1 (default) | numeric scalar

Percentile of brightest colors to use for illuminant estimation, specified as a numeric scalar in the range [0, 100). To return the maximum red, green, and blue values, set `topPercentile` to 0.

The image indicates the red, green, and blue value that is selected to estimate the illuminant. The selection is separate for each color channel.



Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`mask` — Image mask
m-by-*n* logical or numeric matrix

Image mask, specified as an *m*-by-*n* logical or numeric matrix. The mask indicates which pixels of the input image `A` to use when estimating the illuminant. The computation excludes pixels in `A` that correspond to a mask value of 0. By default, the mask has all 1s, and all pixels in `A` are included in the estimation.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

`illuminant` — Estimate of scene illumination
3-element numeric row vector

Estimate of scene illumination, returned as a 3-element numeric row vector. The three elements correspond to the red, green, and blue values of the illuminant.

Data Types: `double`

References

[1] Ebner, Marc. "White Patch Retinex." *Color Constancy*. Chichester, West Sussex: John Wiley & Sons, 2007.

See Also

`whitepoint` | `chromadapt` | `illumgray` | `illumpca` | `lin2rgb` | `rgb2lin`

Introduced in R2017b

im2bw

Convert image to binary image, based on threshold

Note `im2bw` is not recommended. Use `imbinarize` instead. For more information, see "Compatibility Considerations".

Syntax

```
BW = im2bw(I,level)
BW = im2bw(X,cmap,level)
BW = im2bw(RGB,level)
```

Description

`BW = im2bw(I,level)` converts the grayscale image `I` to binary image `BW`, by replacing all pixels in the input image with luminance greater than `level` with the value `1` (white) and replacing all other pixels with the value `0` (black).

This range is relative to the signal levels possible for the image's class. Therefore, a `level` value of `0.5` corresponds to an intensity value halfway between the minimum and maximum value of the class.

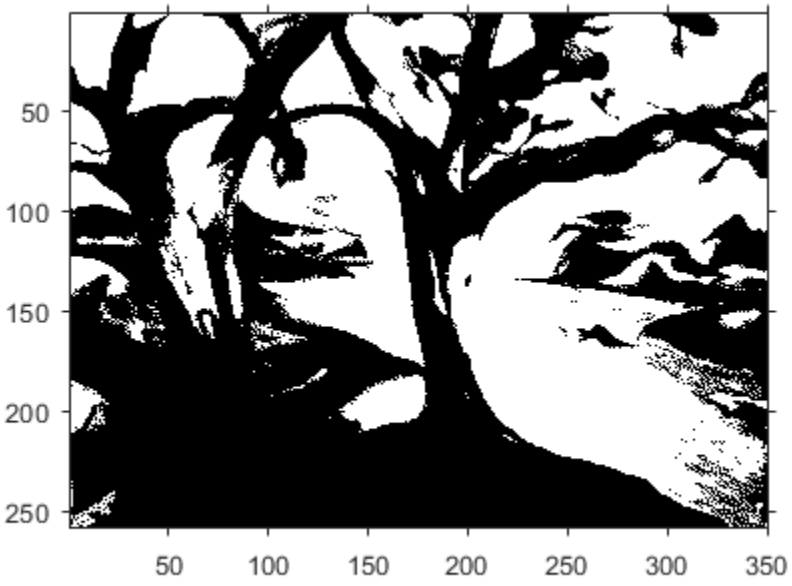
`BW = im2bw(X,cmap,level)` converts the indexed image `X` with colormap `cmap` to a binary image.

`BW = im2bw(RGB,level)` converts the truecolor image `RGB` to a binary image.

Examples

Convert an Indexed Image To a Binary Image

```
load trees
BW = im2bw(X,map,0.4);
imshow(X,map), figure, imshow(BW)
```



Input Arguments

I — 2-D grayscale image
m-by-*n* numeric matrix

2-D grayscale image, specified as an *m*-by-*n* numeric matrix.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`X` — 2-D indexed image
m-by-*n* numeric matrix

2-D indexed image, specified as an *m*-by-*n* numeric matrix.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`cmap` — Colormap
c-by-3 numeric matrix

Colormap associated with indexed image `X`, specified as a *c*-by-3 numeric matrix with values in the range [0, 1]. Each row is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`RGB` — 2-D RGB image
m-by-*n*-by-3 numeric matrix

2-D RGB image, specified as an *m*-by-*n*-by-3 numeric matrix.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`level` — Luminance threshold
`0.5` (default) | number in the range [0, 1]

Luminance threshold, specified as a number in the range [0, 1]. To compute `level`, you can use the `graythresh` function.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

Output Arguments

`BW` — Binary image
m-by-*n* logical matrix

Binary image, returned as an *m*-by-*n* logical matrix.

Data Types: `logical`

Algorithms

If the input image is not a grayscale image, `im2bw` converts the input image to grayscale using `ind2gray` or `rgb2gray`, and then converts this grayscale image to binary by thresholding.

Compatibility Considerations

`im2bw` is not recommended
Not recommended starting in R2016a

The default luminance threshold of `im2bw` is not optimal for most images. If you want to use a threshold appropriate for your image, you must compute the level using `graythresh` before calling `im2bw`.

In R2016a, the `imbinarize` function was introduced. This function computes the luminance threshold and performs binarization in one step. `imbinarize` has additional benefits, such as the ability to perform adaptive thresholding when the image has nonuniform shading. For more information, see Image Binarization - New 2016a Functions.

The table shows some typical usages of `im2bw` and how to update your code to use `imbinarize` instead.

Not Recommended	Recommended
`BW = im2bw(I);`	`BW = imbinarize(I,0.5);`
`thresh = graythresh(I);` `BW = im2bw(I,thresh);`	`BW = imbinarize(I);`

There are no plans to remove `im2bw` at this time.

See Also

`graythresh` | `ind2gray` | `rgb2gray` | `imbinarize`

Introduced before R2006a

im2col

Rearrange image blocks into columns

Syntax

```
B = im2col(A,[m n],'distinct')
B = im2col(A,[m n],'sliding')
B = im2col(A,[m n])
B = im2col(A,'indexed',___)
```

Description

`B = im2col(A,[m n],'distinct')` rearranges discrete image blocks of size `m`-by-`n` into columns, and returns the concatenated columns in matrix `B`. The `im2col` function pads image `A`, if necessary. For more information about the padding value, see “Tips” on page 1-1229.

The order of the columns in matrix `B` is determined by traversing the image `A` in a column-wise manner. For example, if `A` consists of distinct blocks `Aij` arranged as `A = [A11 A12; A21 A22]`, then `B = [A11(:) A21(:) A12(:) A22(:)]`.

`B = im2col(A,[m n],'sliding')` or

`B = im2col(A,[m n])` rearranges sliding image neighborhoods of size `m`-by-`n` into columns with no zero-padding, and returns the concatenated columns in matrix `B`.

`B = im2col(A,'indexed',___)` interprets `A` as an indexed image.

Examples

Calculate Local Mean Using [2 2] Neighborhood

Create a matrix.

```
A = reshape(linspace(0,1,16),[4 4])'
```

```
A = 4×4

         0    0.0667    0.1333    0.2000
    0.2667    0.3333    0.4000    0.4667
    0.5333    0.6000    0.6667    0.7333
    0.8000    0.8667    0.9333    1.0000
```

Rearrange the values into a column-wise arrangement.

```
B = im2col(A,[2 2])
```

```
B = 4×9

         0    0.2667    0.5333    0.0667    0.3333    0.6000    0.1333    0.4000    0.6667
    0.2667    0.5333    0.8000    0.3333    0.6000    0.8667    0.4000    0.6667    0.9333
```

```
    0.0667    0.3333    0.6000    0.1333    0.4000    0.6667    0.2000    0.4667    0.7333
    0.3333    0.6000    0.8667    0.4000    0.6667    0.9333    0.4667    0.7333    1.0000
```

Calculate the mean.

```
M = mean(B)
```

```
M = 1×9

    0.1667    0.4333    0.7000    0.2333    0.5000    0.7667    0.3000    0.5667    0.8333
```

Rearrange the values back into their original, row-wise orientation.

```
newA = col2im(M,[1 1],[3 3])
```

```
newA = 3×3

    0.1667    0.2333    0.3000
    0.4333    0.5000    0.5667
    0.7000    0.7667    0.8333
```

Input Arguments

`A` — Image

2-D grayscale image | 2-D binary image | 2-D indexed image

Image, specified as a 2-D grayscale image, 2-D binary image, or 2-D indexed image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`[m n]` — Block size

2-element vector

Block size, specified as a 2-element vector. `m` is the number of rows and `n` is the number of columns in the block.

Output Arguments

`B` — Image blocks

numeric matrix | logical matrix

Image blocks, returned as a numeric matrix or logical matrix with `m*n` rows. The number of columns depends on whether the image blocks are discrete blocks or sliding neighborhoods. Each column of `B` contains a block or neighborhood of `A` reshaped as a column vector.

- For distinct block processing, `B` has as many columns as there are `m`-by-`n` blocks in `A`. For example, if the size of `A` is `[mm nn]`, then `B` has `(mm/m)*(nn/n)` columns.
- For sliding neighborhood processing, `B` has as many columns as there are `m`-by-`n` neighborhoods of `A`. For example, if the size of `A` is `[mm nn]`, then `B` has `((mm-m+1)*(nn-n+1))` columns.

Tips

- For distinct block processing, `im2col` zero-pads `A`, if necessary, so its size is an integer multiple of `m`-by-`n`. The padding value is `0` when `A` is data type `uint8`, `uint16`, or `logical`. For other data types, the value of padding depends on whether `A` is interpreted as an indexed image.
  - The padding value is `1` when `A` is interpreted as an indexed image.
  - The padding value is `0` when `A` is not interpreted as an indexed image.
- `im2col` orders the columns of `B` so that they can be reshaped to form a matrix according to `reshape`.

  For example, suppose you use a function, such as `sum(B)`, that returns a scalar for each column of `B`. You can directly store the result in a matrix of size (`mm-m+1`)-by-(`nn-n+1`), using these calls.

```
B = im2col(A,[m n],'sliding');
C = reshape(sum(B),mm-m+1,nn-n+1);
```

See Also

`blockproc` | `col2im` | `colfilt` | `nlfilter` | `reshape`

Introduced before R2006a

im2int16

Convert image to 16-bit signed integers

Syntax

```
J = im2int16(I)
```

Description

`J = im2int16(I)` converts the grayscale, RGB, or binary image `I` to `int16`, rescaling the data if necessary.

If the input image is of class `int16`, then the output image is identical to the input image. If the input image is of class `logical`, then `im2int16` changes false-valued elements to -32768 and true-valued elements to 32767.

Examples

Convert Array from double to int16

Create an array of class `double`.

```
I = reshape(linspace(0,1,20),[5 4])
```

```
I = 5×4

         0    0.2632    0.5263    0.7895
    0.0526    0.3158    0.5789    0.8421
    0.1053    0.3684    0.6316    0.8947
    0.1579    0.4211    0.6842    0.9474
    0.2105    0.4737    0.7368    1.0000
```

Convert the array to class `int16`.

```
I2 = im2int16(I)
```

```
I2 = 5x4 int16 matrix

   -32768   -15522     1724    18970
   -29319   -12073     5173    22419
   -25870    -8624     8623    25869
   -22420    -5174    12072    29318
   -18971    -1725    15521    32767
```

Input Arguments

`I` — Input image

numeric array | logical array

Input image, specified as a numeric array or logical array of any size and dimension.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

Output Arguments

`J` — Image with class `int16`

numeric array

Image with class `int16`, returned as a numeric array of the same size as the input image `I`.

Data Types: `int16`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `im2int16` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `im2int16` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`im2double` | `im2single` | `im2uint8` | `im2uint16` | `int16`

Introduced before R2006a

im2java2d

(To be removed) Convert image to Java buffered image

Note `im2java2d` will be removed in a future release. For more information, see "Compatibility Considerations".

Syntax

```
javaImage = im2java2d(I)
javaImage = im2java2d(X,map)
```

Description

`javaImage = im2java2d(I)` converts the input image to an instance of the Java image class `java.awt.image.BufferedImage`.

`javaImage = im2java2d(X,map)` converts an indexed image with colormap `map` to an instance of the Java image class `java.awt.image.BufferedImage`.

Input Arguments

`I` — Input image

m-by-*n* matrix | *m*-by-*n*-by-3 array

Input image, specified as

- *m*-by-*n* matrix for grayscale and binary images.
- *m*-by-*n*-by-3 array for RGB color images.

Data Types: `double` | `uint8` | `uint16` | `logical`

`X` — Input indexed image

m-by-*n* matrix

Input indexed image, specified as a *m*-by-*n* matrix.

Data Types: `double` | `uint8` | `uint16`

`map` — Colormap

c-by-*3* numeric array

Colormap associated with input indexed image `X`, specified as a *c*-by-*3* numeric array. *c* represents the number of colors in the colormap.

Data Types: `double`

Output Arguments

`javaImage` — Output Java 2D image

`BufferedImage` class

Output Java 2D image, returned as a `BufferedImage` class of instance `java.awt.image.BufferedImage`. The output Java 2D image can be used with the Java 2D API and the Java Abstract Windowing Toolkit (AWT).

Compatibility Considerations

`im2java2d` function will be removed

Warns starting in R2020a

The `im2java2d` function will be removed in a future release. There is no replacement for this function.

Introduced before R2006a

im2single

Convert image to single precision

Syntax

```
J = im2single(I)
J = im2single(I,'indexed')
```

Description

`J = im2single(I)` converts the grayscale, RGB, or binary image `I` to `single`, rescaling or offsetting the data as necessary.

If the input image is of class `single`, then the output image is identical. If the input image is of class `logical`, then `im2single` changes true-valued elements to 65535.

`J = im2single(I,'indexed')` converts the indexed image `I` to `single`, offsetting the data if necessary.

Examples

Convert Array to Class Single

This example shows how to convert an array of class `uint8` into class `single` .

Create a numeric array of class `uint8` .

```
I = reshape(uint8(linspace(1,255,25)),[5 5])
```

```
I = 5x5 uint8 matrix

     1    54   107   160   213
    12    65   117   170   223
    22    75   128   181   234
    33    86   139   192   244
    43    96   149   202   255
```

Convert the array to class `single` .

```
I2 = im2single(I)
```

```
I2 = 5x5 single matrix

    0.0039    0.2118    0.4196    0.6275    0.8353
    0.0471    0.2549    0.4588    0.6667    0.8745
    0.0863    0.2941    0.5020    0.7098    0.9176
    0.1294    0.3373    0.5451    0.7529    0.9569
    0.1686    0.3765    0.5843    0.7922    1.0000
```

Input Arguments

`I` — Input image

numeric array | logical array

Input image, specified as a numeric array or logical array of any size and dimension.

- If `I` is a grayscale or RGB image, then it can be `uint8`, `uint16`, `double`, `logical`, `single`, or `int16`.
- If `I` is an indexed image, then it can be `uint8`, `uint16`, `double` or `logical`.
- If `I` is a binary image, then it must be `logical`.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

Output Arguments

`J` — Image with class `single`

numeric array

Image with class `single`, returned as a numeric array of the same size as the input image `I`.

Data Types: `single`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`im2single` supports the generation of C code (requires MATLAB Coder). For more information, see "Code Generation for Image Processing".

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see "Run MATLAB Functions in Thread-Based Environment".

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see "Image Processing on a GPU".

See Also

`im2double` | `im2int16` | `im2uint8` | `im2uint16` | `single`

Introduced before R2006a

im2uint16

Convert image to 16-bit unsigned integers

Syntax

```
J = im2uint16(I)
J = im2uint16(I,'indexed')
```

Description

`J = im2uint16(I)` converts the grayscale, RGB, or binary image `I` to `uint16`, rescaling or offsetting the data as necessary.

If the input image is of class `uint16`, then the output image is identical. If the input image is of class `logical`, then `im2uint16` changes true-valued elements to 65535.

`J = im2uint16(I,'indexed')` converts the indexed image `I` to `uint16`, offsetting the data if necessary.

Examples

Convert Array from double to uint16

Create an array of class `double`.

```
I = reshape(linspace(0,1,20),[5 4])
```

```
I = 5×4

         0    0.2632    0.5263    0.7895
    0.0526    0.3158    0.5789    0.8421
    0.1053    0.3684    0.6316    0.8947
    0.1579    0.4211    0.6842    0.9474
    0.2105    0.4737    0.7368    1.0000
```

Convert the array to class `uint16`.

```
I2 = im2uint16(I)
```

```
I2 = 5x4 uint16 matrix

       0   17246   34492   51738
    3449   20695   37941   55187
    6898   24144   41391   58637
   10348   27594   44840   62086
   13797   31043   48289   65535
```

Input Arguments

`I` — Input image

numeric array | logical array

Input image, specified as a numeric array or logical array of any size and dimension.

- If `I` is a grayscale or RGB image, then it can be `uint8`, `uint16`, `double`, `logical`, `single`, or `int16`.
- If `I` is an indexed image, then it can be `uint8`, `uint16`, `double` or `logical`.

  Note It is not always possible to convert an indexed image to `uint8`. If the indexed image is of class `double`, then the maximum value must be 65536 or less.

- If `I` is a binary image, then it must be `logical`.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

Output Arguments

`J` — Image with class `uint16`

numeric array

Image with class `uint16`, returned as a numeric array of the same size as the input image `I`.

Data Types: `uint16`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `im2uint16` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `im2uint16` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`im2uint8` | `double` | `im2double` | `uint8` | `uint16` | `imapprox`

Introduced before R2006a

im2uint8

Convert image to 8-bit unsigned integers

Syntax

```
J = im2uint8(I)
J = im2uint8(I,'indexed')
```

Description

`J = im2uint8(I)` converts the grayscale, RGB, or binary image `I` to `uint8`, rescaling or offsetting the data as necessary.

If the input image is of class `uint8`, then the output image is identical. If the input image is of class `logical`, then `im2uint8` changes true-valued elements to 255.

`J = im2uint8(I,'indexed')` converts the indexed image `I` to `uint8`, offsetting the data if necessary.

Examples

Convert uint16 Array to uint8 Array

Create an array of class `uint16`.

```
I = reshape(uint16(linspace(0,65535,25)),[5 5])
```

```
I = 5x5 uint16 matrix

       0   13653   27306   40959   54613
    2731   16384   30037   43690   57343
    5461   19114   32768   46421   60074
    8192   21845   35498   49151   62804
   10923   24576   38229   51882   65535
```

Convert the array to class `uint8` .

```
I2 = im2uint8(I)
```

```
I2 = 5x5 uint8 matrix

     0    53   106   159   213
    11    64   117   170   223
    21    74   128   181   234
    32    85   138   191   244
    43    96   149   202   255
```

Input Arguments

`I` — Input image

numeric array | logical array

Input image, specified as a numeric array or logical array of any size and dimension.

- If `I` is a grayscale or RGB image, then it can be `uint8`, `uint16`, `double`, `logical`, `single`, or `int16`. The intensity values for input image of class `single` or `double` must be in the range [0, 1].

  Note If `I` is of class `single` or `double` with values outside the range [0, 1] then you can use `rescale` function to rescale values to the expected range.

- If `I` is an indexed image, then it can be `uint8`, `uint16`, `double` or `logical`.

  Note It is not always possible to convert an indexed image to `uint8`. If the indexed image is of class `double`, then the maximum value must be 256 or less. If the indexed image is of class `uint16`, then the maximum value must be 255 or less.

- If `I` is a binary image, then it must be `logical`.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

Output Arguments

`J` — Image with class `uint8`

numeric array

Image with class `uint8`, returned as a numeric array of the same size as the input image `I`.

Data Types: `uint8`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `im2uint8` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `im2uint8` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`im2double` | `im2int16` | `im2single` | `im2uint16` | `uint8`

Introduced before R2006a

imabsdiff

Absolute difference of two images

Syntax

```
Z = imabsdiff(X,Y)
```

Description

`Z = imabsdiff(X,Y)` subtracts each element in array `Y` from the corresponding element in array `X` and returns the absolute difference in the corresponding element of the output array `Z`.

Examples

Display Absolute Difference between Filtered image and Original

Read image into workspace.

```
I = imread('cameraman.tif');
```

Filter the image.

```
J = uint8(filter2(fspecial('gaussian'), I));
```

Calculate the absolute difference of the two images.

```
K = imabsdiff(I,J);
```

Display the absolute difference image.

```
figure
imshow(K,[])
```

Input Arguments

X — Input image

numeric array

Input image, specified as a numeric array of any dimension.

Example: `x = imread('cameraman.tif');`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

Y — Input image

numeric array

Input image, specified as a numeric array. Y must be the same size and class as X.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

Output Arguments

Z — Difference image

numeric array

Difference image, returned as a numeric array. Z has the same class and size as X and Y. If X and Y are integer arrays, then `imabsdiff` truncates elements in the output that exceed the range of the integer type.

Tips

- If `X` is of class `double`, then use the expression `abs(X-Y)` instead of this function.
- If `X` is of class `logical`, then use the expression `XOR(X,Y)` instead of this function.
- When `X` and `Y` are of class `uint8`, `int16`, or `single`, then `imabsdiff` can use hardware optimization to run faster.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`imabsdiff` supports the generation of C code (requires MATLAB Coder). For more information, see "Code Generation for Image Processing".

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see "Image Processing on a GPU".

See Also

`imadd` | `imcomplement` | `imdivide` | `imlincomb` | `imsubtract` | `immultiply`

Introduced before R2006a

imadd

Add two images or add constant to image

Syntax

```
Z = imadd(X,Y)
```

Description

`Z = imadd(X,Y)` adds each element in array `X` with the corresponding element in array `Y` and returns the sum in the corresponding element of the output array `Z`.

Examples

Add Two uint8 Arrays

This example shows how to add two `uint8` arrays with truncation for values that exceed 255.

```
X = uint8([ 255 0 75; 44 225 100]);
Y = uint8([ 50 50 50; 50 50 50 ]);
Z = imadd(X,Y)
```

```
Z = 2x3 uint8 matrix

   255    50   125
    94   255   150
```

Add Two Images and Specify Output Class

Read two grayscale `uint8` images into the workspace.

```
I = imread('rice.png');
J = imread('cameraman.tif');
```

Add the images. Specify the output as type `uint16` to avoid truncating the result.

```
K = imadd(I,J,'uint16');
```

Display the result.

```
imshow(K,[])
```

Add a Constant to an Image

Read an image into the workspace.

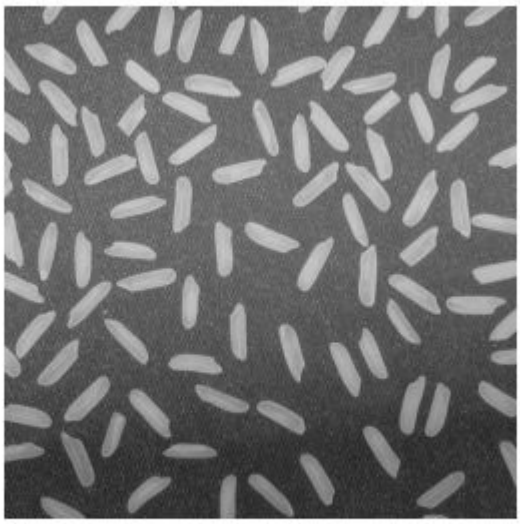
```
I = imread('rice.png');
```

Add a constant to the image.

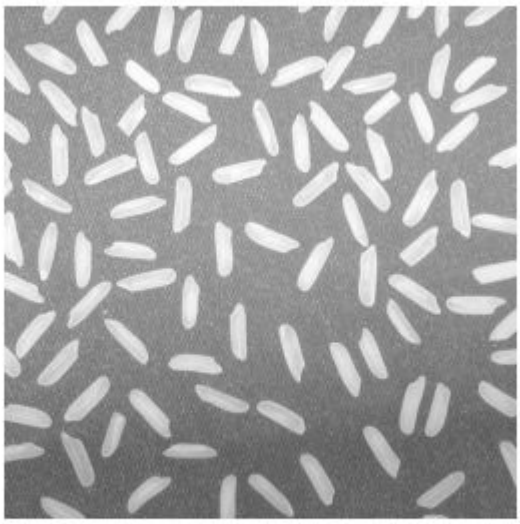
```
J = imadd(I,50);
```

Display the original image and the result.

```
imshow(I)
```



```
figure
imshow(J)
```



Input Arguments

X — First array

numeric array | logical array

First array, specified as a numeric array or logical array of any dimension.

Y — Second array

numeric scalar | numeric array | logical array

Second array to be added to X, specified as a numeric or logical array of the same size and class as X, or a numeric scalar of type `double`.

Output Arguments

Z — Sum

numeric array

Sum, returned as a numeric array of the same size as X. Z is the same class as X unless X is logical, in which case Z is data type `double`. If X is an integer array, elements of the output that exceed the range of the integer type are truncated, and fractional values are rounded.

See Also

`imabsdiff` | `imcomplement` | `imdivide` | `imlincomb` | `immultiply` | `imsubtract`

Introduced before R2006a

imadjust

Adjust image intensity values or colormap

Syntax

```
J = imadjust(I)
J = imadjust(I,[low_in high_in])
J = imadjust(I,[low_in high_in],[low_out high_out])
J = imadjust(I,[low_in high_in],[low_out high_out],gamma)

J = imadjust(RGB,[low_in high_in],___)
newmap = imadjust(cmap,[low_in high_in],___)
```

Description

`J = imadjust(I)` maps the intensity values in grayscale image `I` to new values in `J`. By default, `imadjust` saturates the bottom 1% and the top 1% of all pixel values. This operation increases the contrast of the output image `J`.

This syntax is equivalent to `imadjust(I,stretchlim(I))`.

`J = imadjust(I,[low_in high_in])` maps intensity values in `I` to new values in `J` such that values between `low_in` and `high_in` map to values between 0 and 1.

`J = imadjust(I,[low_in high_in],[low_out high_out])` maps intensity values in `I` to new values in `J` such that values between `low_in` and `high_in` map to values between `low_out` and `high_out`.

`J = imadjust(I,[low_in high_in],[low_out high_out],gamma)` maps intensity values in `I` to new values in `J`, where `gamma` specifies the shape of the curve describing the relationship between the values in `I` and `J`.

`J = imadjust(RGB,[low_in high_in],___)` maps the values in truecolor image `RGB` to new values in `J`. You can apply the same mapping or unique mappings for each color channel.

`newmap = imadjust(cmap,[low_in high_in],___)` maps the values in colormap `cmap` to new values in `newmap`. You can apply the same mapping or unique mappings for each color channel.

Examples

Adjust Contrast of Grayscale Image

Read a low-contrast grayscale image into the workspace and display it.

```
I = imread('pout.tif');
imshow(I)
```

Adjust the contrast of the image so that 1% of the data is saturated at low and high intensities, and display it.

```
J = imadjust(I);
figure
imshow(J)
```

Adjust Contrast of Grayscale Image Specifying Contrast Limits

Read a low-contrast grayscale image into the workspace and display it.

```
I = imread('pout.tif');
imshow(I);
```

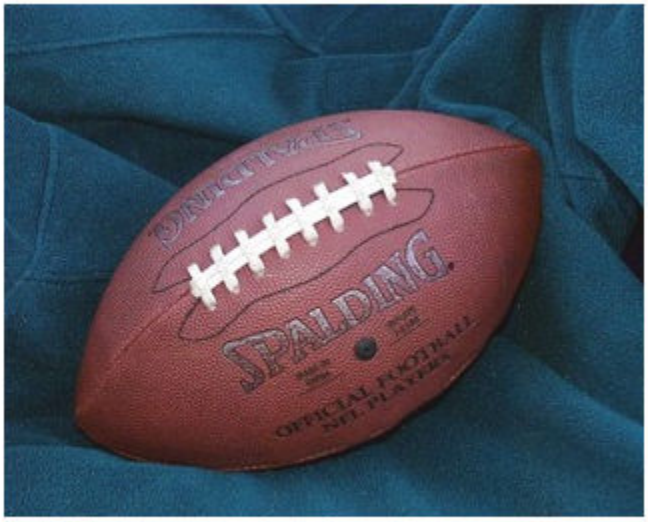
Adjust the contrast of the image, specifying contrast limits.

```
K = imadjust(I,[0.3 0.7],[]);
figure
imshow(K)
```

Adjust Contrast of Color Image

Read an RGB image into the workspace and display it.

```
RGB = imread('football.jpg');
imshow(RGB)
```



Adjust the contrast of the RGB image, specifying contrast limits.

```
RGB2 = imadjust(RGB,[.2 .3 0; .6 .7 1],[]);
figure
imshow(RGB2)
```

Standard Deviation Based Image Stretching

Read an image into the workspace, and display it.

```
I = imread('pout.tif');
imshow(I)
```

Calculate the standard deviation and the image mean for stretching.

```
n = 2;
Idouble = im2double(I);
avg = mean2(Idouble);
sigma = std2(Idouble);
```

Adjust the contrast based on the standard deviation.

```
J = imadjust(I,[avg-n*sigma avg+n*sigma],[]);
```

Display the adjusted image.

```
imshow(J)
```

Input Arguments

`I` — Grayscale image

m-by-*n* numeric matrix

Grayscale image, specified as an *m*-by-*n* numeric matrix.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`RGB` — Truecolor image

m-by-*n*-by-3 numeric array

Truecolor image, specified as an *m*-by-*n*-by-3 numeric array.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`cmap` — Colormap

c-by-3 numeric matrix

Colormap, specified as a *c*-by-3 numeric matrix with values in the range [0, 1]. Each row is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap.

Data Types: `double`

`[low_in high_in]` — Contrast limits for input image

`[0 1]` (default) | 2-element numeric vector | 2-by-3 numeric matrix

Contrast limits for the input image, specified in one of the following forms:

Input Type	Value	Description
Grayscale image	1-by-2 vector of the form `[low_in high_in]`	Specifies the contrast limits in the input grayscale image that you want to map to values in the output image. Values must be in the range `[0 1.0]`. The value `low_in` must be less than the value `high_in`.
RGB image or colormap	2-by-3 matrix of the form `[low_RGB_triplet; high_RGB_triplet]`	Specifies the contrast limits in the input RGB image or colormap that you want to map to values in the output image or colormap. Each row in the array is an RGB color triplet. Values must be in the range `[0 1]`. The value `low_RGB_triplet` must be less than the value `high_RGB_triplet`.
RGB image or colormap	1-by-2 vector of the form `[low_in high_in]`	Specifies the contrast limits in the input RGB image that you want to map to values in the output image. Each value must be in the range `[0 1.0]`. The value `low_in` must be less than the value `high_in`. If you specify a 1-by-2 vector with an RGB image or colormap, then `imadjust` applies the same adjustment to each color plane or channel.
All types	`[]`	If you specify an empty matrix (`[]`), then `imadjust` uses the default limits `[0 1]`.

`imadjust` clips value below `low_in` and above `high_in`: Values below `low_in` map to `low_out` and values above `high_in` map to `high_out`.

Data Types: `single` | `double`

`[low_out high_out]` — Contrast limits for output image
`[0 1]` (default) | 2-element numeric vector | 2-by-3 numeric matrix

Contrast limits for the output image, specified in one of the following forms:

Input Type	Value	Description
Grayscale image	1-by-2 vector of the form `[low_out high_out]`	Specifies the contrast limits of the output grayscale image. Each value must be in the range `[0 1]`.
RGB image or colormap	2-by-3 matrix of the form `[low_RGB_triplet; high_RGB_triplet]`	Specifies the contrast limits of the output RGB image or colormap. Each row in the array is an RGB color triplet. Values must be in the range `[0 1]`.
RGB image or colormap	1-by-2 vector of the form `[low_out high_out]`	Specifies the contrast limits in the output image. Each value must be in the range `[0 1]`. If you specify a 1-by-2 vector with an RGB image or colormap, then `imadjust` applies the same adjustment to each plane or channel.
All types	`[]`	If you specify an empty matrix (`[]`), then `imadjust` uses the default limits `[0 1]`.

If `high_out` is less than `low_out`, then `imadjust` reverses the output image, as in a photographic negative.

Data Types: `single` | `double`

`gamma` — Shape of curve describing relationship of input and output values
1 (default) | nonnegative scalar | 1-by-3 numeric vector

Shape of curve describing the relationship of input and output values, specified as a nonnegative scalar or a 1-by-3 numeric vector.

- If `gamma` is less than 1, then `imadjust` weights the mapping toward higher (brighter) output values.
- If `gamma` is greater than 1, then `imadjust` weights the mapping toward lower (darker) output values.
- If `gamma` is a 1-by-3 vector, then `imadjust` applies a unique gamma to each color component or channel.
- If you omit the argument, then `gamma` defaults to `1` (linear mapping).

Data Types: `double`

Output Arguments

`J` — Adjusted image
grayscale image | RGB image

Adjusted image, returned as a grayscale or RGB image. `J` has the same size and class as the input grayscale image `I` or truecolor image `RGB`.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`newmap` — Adjusted colormap
c-by-3 numeric matrix

Adjusted colormap, returned as an *c*-by-3 numeric matrix of the same class as the input colormap, `map`.

Data Types: `single` | `double`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imadjust` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imadjust` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, `imadjust` does not support indexed images.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, `imadjust` does not support indexed images.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`brighten` | `histeq` | `stretchlim`

Introduced before R2006a

imadjustn

Adjust intensity values in *N*-D volumetric image

Syntax

```
J = imadjustn(V)
J = imadjustn(V,[low_in high_in])
J = imadjustn(V,[low_in high_in].[low_out high_out])
J = imadjustn(V,[low_in high_in],[low_out high_out],gamma)
```

Description

`J = imadjustn(V)` maps the values in the *N*-D volumetric intensity image `V` to new values in `J`. `imadjustn` increases the contrast of the output volumetric image `J`.

By default, `imadjustn` saturates the bottom 1% and the top 1% of all pixel values. This syntax is equivalent to `imadjustn(V,stretchlim(V(:)))`.

`J = imadjustn(V,[low_in high_in])` maps the values in `V` to new values in the range [0, 1]. Values below `low_in` map to `0` and values above `high_in` map to `1`.

`J = imadjustn(V,[low_in high_in].[low_out high_out])` maps the values in `V` to new values in `J` such that values between `low_in` and `high_in` map to values between `low_out` and `high_out`. Values below `low_in` are clipped to `low_out` and values above `high_in` are clipped to `high_out`.

If `high_out` is less than `low_out`, then `imadjustn` reverses the output image volume, as in a photographic negative.

`J = imadjustn(V,[low_in high_in],[low_out high_out],gamma)` maps the values in `V` to `J` using a nonlinear gamma curve.

Examples

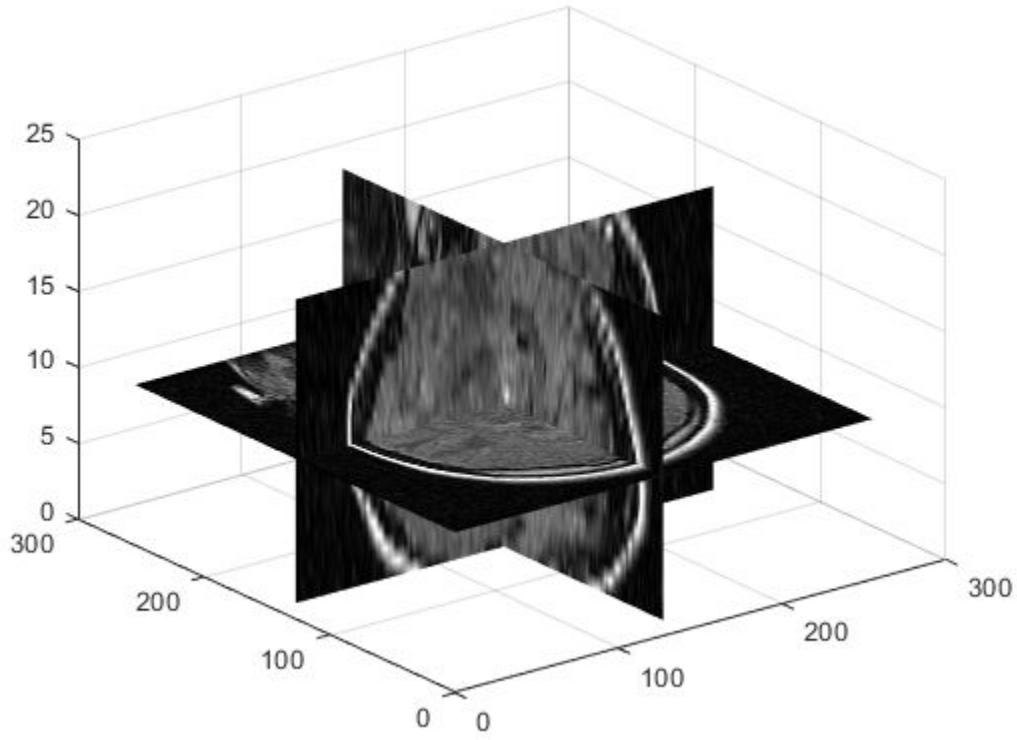
Scale Intensity of 3-D Volume of MRI Data

Load a 3-D image into the workspace, then save the image as data type `double`.

```
load mristack;
V1 = im2double(mristack);
```

Display cross-sections of the image.

```
figure
slice(V1,size(V1,2)/2,size(V1,1)/2,size(V1,3)/2)
colormap gray
shading interp
```

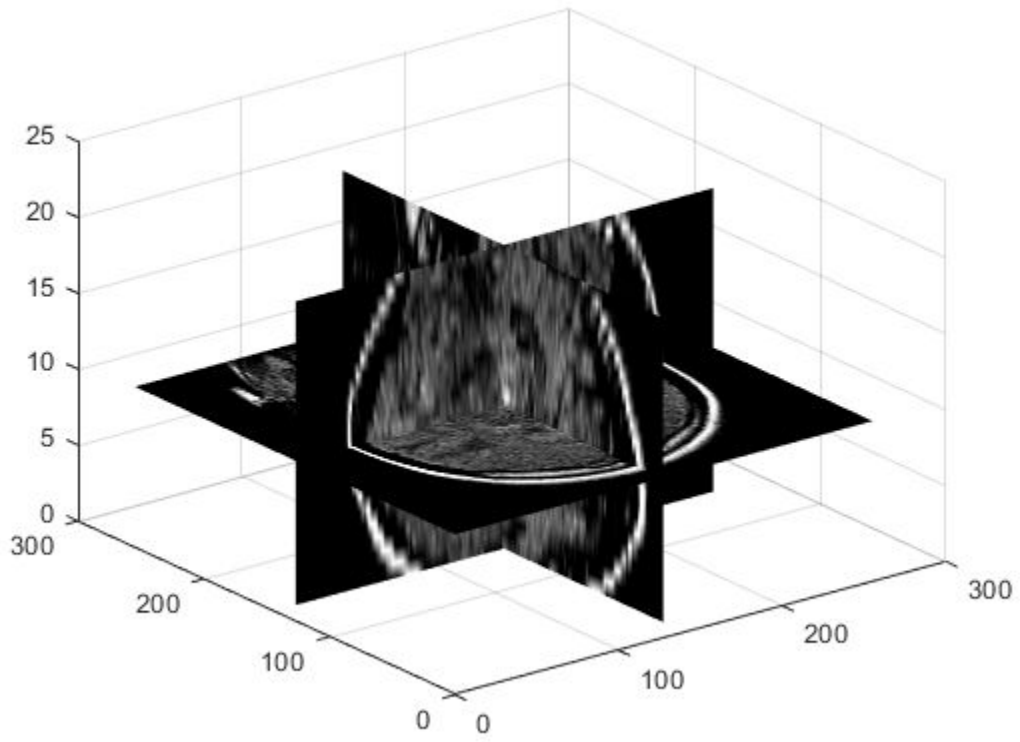


Adjust the image intensity values. `imadjustn` maps input values between 0.2 and 0.8 to the default output range of [0, 1]. `imadjustn` clips input values below 0.2 and above 0.8.

```
V2 = imadjustn(V1,[0.2 0.8],[]);
```

Display cross-sections of the contrast-adjusted image.

```
figure
slice(V2,size(V2,2)/2,size(V2,1)/2,size(V2,3)/2)
colormap gray
shading interp
```



Input Arguments

`V` — Volumetric intensity image

N-D numeric array

Volumetric intensity image, specified as an *N*-D numeric array.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`[low_in high_in]` — Range of values in input image

`[0 1]` (default) | 2-element vector

Range of values in the input image, specified as a 2-element vector of the form `[low_in high_in]`, with values in the range [0, 1]. Before adjusting intensity values, `imadjustn` converts the input image to class `double` (using `im2double`), rescaling values to the range [0, 1]. `low_in` and `high_in` correspond to the specified input range after conversion to `double`.

You can use an empty matrix (`[]`) for `[low_in high_in]` to specify the default of `[0 1]`.

Data Types: `double`

`[low_out high_out]` — Range of values in output image

`[0 1]` (default) | 2-element vector

Range of values in output image, specified as a 2-element vector of the form `[low_out high_out]`, with values in the range [0, 1]. Before adjusting intensity values, `imadjustn` converts the input

image to class `double` (using `im2double`), rescaling values to the range [0,1]. `low_out` and `high_out` correspond to the specified output range after conversion to `double`. After adjusting intensity values, `imadjustn` converts the image to the data type of the input image.

You can use an empty matrix (`[]`) for `[low_out high_out]` to specify the default of `[0 1]`.

Data Types: `double`

`gamma` — Shape of gamma curve

1 (default) | numeric scalar

Shape of gamma curve describing relationship between values in `V` and `J`, specified as a numeric scalar.

- If you omit the argument, then `gamma` defaults to 1 and performs a linear mapping.
- If the value is less than 1, then `imadjustn` weights the mapping toward higher (brighter) output values.
- If the value is greater than 1, then `imadjustn` weights the mapping toward lower (darker) output values.

Data Types: `double`

Output Arguments

`J` — Volume with adjusted intensity values

N-D volumetric intensity image

Volume with adjusted intensity values, returned as an *N*-D volumetric intensity image. The output volume has the same class as the input image.

See Also

`histeq` | `stretchlim` | `decorrstretch` | `imhistmatchn`

Introduced in R2017b

ImageAdapter class

Interface for image I/O

Description

`ImageAdapter` is an abstract class that defines custom region-based reading and writing of images in arbitrary image file formats. You can use classes that inherit from the `ImageAdapter` interface with the `blockproc` function to perform file-based block processing.

To write an Image Adapter class for a particular file format, you must be able to:

- Query the size of the file on disk
- Read a rectangular block of data from the file

To use this class, you must inherit from the `ImageAdapter` class. Type the following syntax as the first line of your class definition file:

```
classdef MyAdapter < ImageAdapter
    ...
end
```

Classes that inherit from `ImageAdapter` must implement the `readRegion` and `close` methods to support basic region-based reading of images. The optional `writeRegion` method allows for incremental, region-based writing of images. Image Adapter classes that do not implement the `writeRegion` method are read-only.

The `ImageAdapter` class is a `handle` class.

Class Attributes

`Abstract`	`true`
`HandleCompatible`	`true`

For information on class attributes, see “Class Attributes”.

Properties

`ImageSize` — Image size

`[]` (default) | 2-element vector of positive integers | 3-element vector of positive integers

Image size, specified as a 2-element vector of positive integers `[m n]`, or a 3-element vector of positive integers `[m n p]`, where `m` is the number of rows, `n` is the number of columns, and `p` is the number of channels of the image.

When you construct a new class that inherits from `ImageAdapter`, set the `ImageSize` property in your class constructor.

Example: `[1920 1080]`

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

`Colormap` — Colormap

`[ ]` (default) | `c`-by-3 numeric matrix

Colormap for indexed images, specified as a `c`-by-3 numeric matrix with values in the range [0, 1]. Each row of the matrix is a 3-element RGB triplet that specifies the red, green, and blue components of a single color.

When you construct a new class that inherits from `ImageAdapter`, set the `Colormap` property in your class constructor.

Attributes:

`GetAccess`	`public`
`SetAccess`	`protected`

Methods

Public Methods

close	Close ImageAdapter object
readRegion	Read region of image
writeRegion	Write block of data to region of image

See Also

`blockproc`

Topics

“Compute Statistics for Large Images”
“Abstract Classes and Class Members”
“Perform Block Processing on Image Files in Unsupported Formats”

Introduced in R2010a

close

Class: `ImageAdapter`

Close `ImageAdapter` object

Syntax

```
close(adapter)
```

Description

`close(adapter)` closes the `ImageAdapter` object and performs any necessary clean-up, such as closing file handles.

Input Arguments

adapter — Image adapter

`ImageAdapter`

Image adapter, specified as an instance of an `ImageAdapter`.

Attributes

`Abstract`	`true`

To learn about attributes of methods, see Method Attributes.

Tips

- When you construct a class that inherits from the `ImageAdapter` class, you must implement this method.
- `blockproc` does not call the `close` method. If your image adapter opens file handles or requires other class clean-up responsibilities, then you must call the `close` method explicitly in your image processing pipeline.

See Also

`ImageAdapter`

Introduced in R2010a

readRegion

Class: `ImageAdapter`

Read region of image

Syntax

```
data = readRegion(adapter,region_start, region_size)
```

Description

`data = readRegion(adapter,region_start, region_size)` reads an image region of size `region_size` with top-left pixel at coordinate `region_start`.

Input Arguments

`adapter` — Image adapter

`ImageAdapter`

Image adapter, specified as an instance of an `ImageAdapter`.

`region_start` — Top-left coordinates of region

2-element vector of positive integers

Top-left coordinates of the region to read, specified as a 2-element vector of positive integers of the form `[row column]`.

`region_size` — Size of region

2-element vector of positive integers

Size of the region to read, specified as a 2-element vector of positive integers of the form `[numrows numcolumns]`.

Output Arguments

`data` — Image data

numeric array

Image data, returned as a numeric array of size `region_size`.

Attributes

`Abstract`	`true`

To learn about attributes of methods, see Method Attributes.

Tips

- When you construct a class that inherits from the `ImageAdapter` class, you must implement this method.

See Also

`ImageAdapter`

Topics

“Compute Statistics for Large Images”
“Perform Block Processing on Image Files in Unsupported Formats”

Introduced in R2010a

writeRegion

Class: `ImageAdapter`

Write block of data to region of image

Syntax

```
writeRegion(adapter,region_start,region_data)
```

Description

`writeRegion(adapter,region_start,region_data)` writes a contiguous block of data `region_data` to the region of the image with top-left pixel at coordinate `region_start`.

Input Arguments

`adapter` — Image adapter

`ImageAdapter`

Image adapter, specified as an instance of an `ImageAdapter`.

`region_start` — Top-left coordinates of region

2-element vector of positive integers

Top-left coordinates of the region to write, specified as a 2-element vector of positive integers of the form `[row column]`.

`region_data` — Image data

numeric array

Image data, specified as a numeric array.

Tips

- When you construct a class that inherits from the `ImageAdapter` class, you can optionally implement this method to enable incremental, region-based writing of images. Image adapter classes that do not implement the `writeRegion` method are read-only.

See Also

`ImageAdapter`

Introduced in R2010a

imageinfo

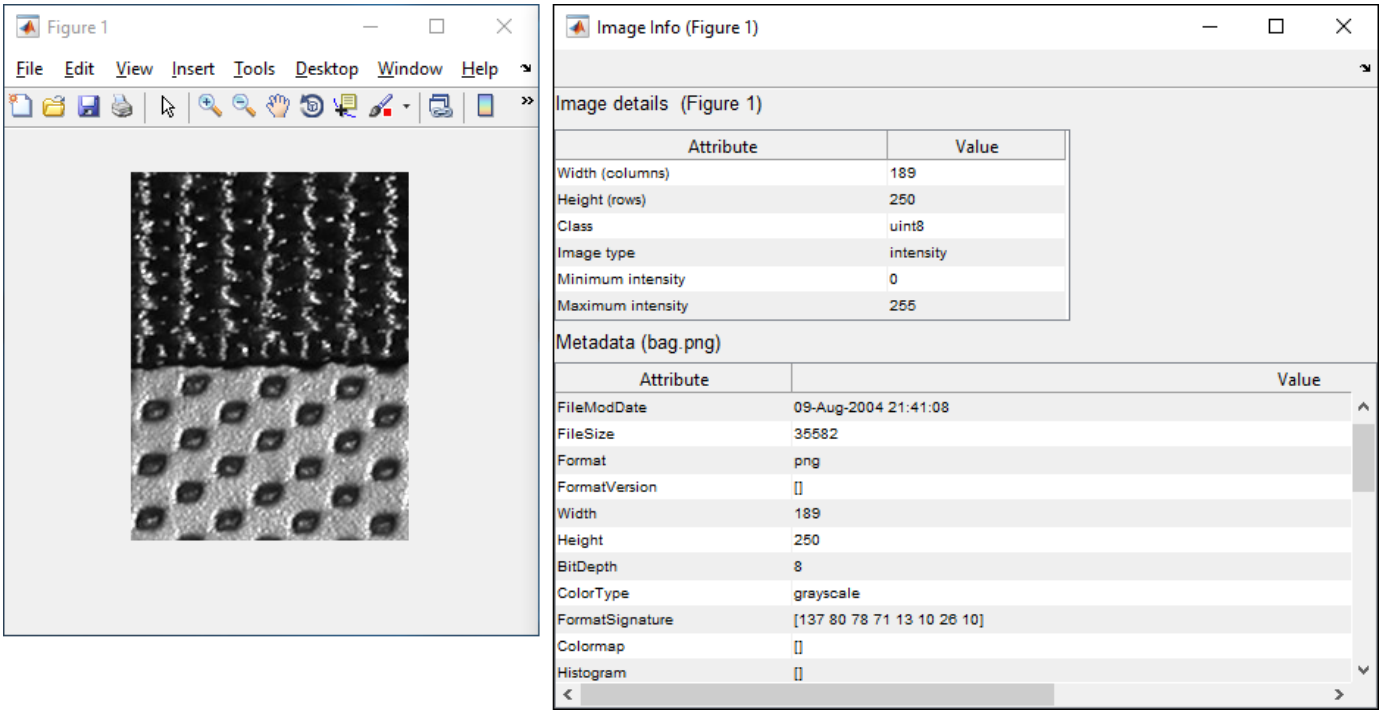
Image Information tool

Syntax

```
imageinfo
imageinfo(h)
imageinfo(filename)
imageinfo(info)
imageinfo(himage,filename)
imageinfo(himage,info)
htool = imageinfo( ___ )
```

Description

Use the `imageinfo` function to create an Image Information tool. The tool displays information about the basic attributes and metadata of the target image in a separate figure.



`imageinfo` creates an Image Information tool associated with the image in the current figure. The tool displays information about the basic attributes of the target image in a separate figure.

`imageinfo(h)` creates an Image Information tool associated with `h`, where `h` is a handle to a figure, axes, or image object.

`imageinfo(filename)` creates an Image Information tool containing image metadata from the graphics file `filename`. The image does not have to be displayed in a figure window.

`imageinfo(info)` creates an Image Information tool containing the image metadata in the structure `info`.

`imageinfo(himage,filename)` creates an Image Information tool containing information about the basic attributes of the image specified by the handle `himage` and the image metadata from the graphics file `filename`.

`imageinfo(himage,info)` creates an Image Information tool containing information about the basic attributes of the image specified by the handle `himage` and the image metadata in the structure `info`.

`htool = imageinfo( ___ )` returns a handle to the Image Information tool figure.

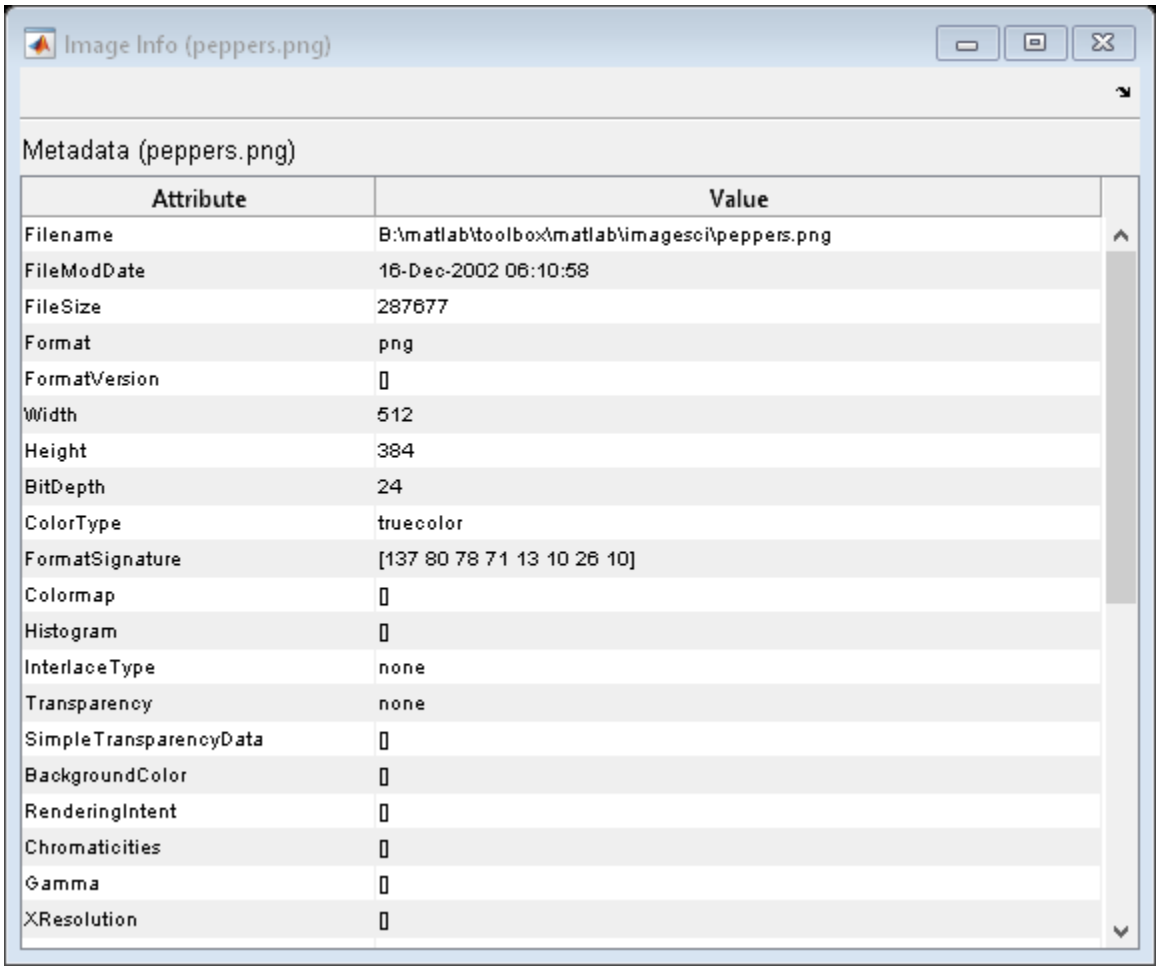
Examples

Open Image Information Tool

There are several ways to open an Image Information tool. This example demonstrates three different ways to open this tool.

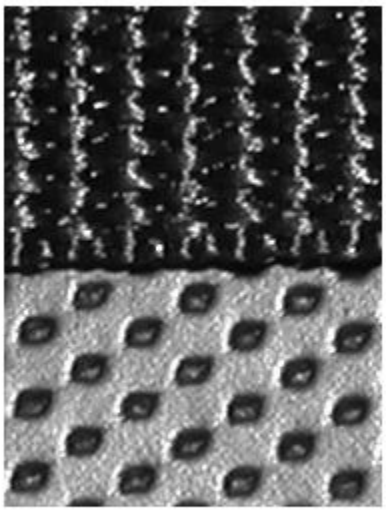
Open an Image Information tool containing metadata from an image file. It is not necessary to display the image.

```
imageinfo('peppers.png')
```



Display an image in a figure window.

```
h = imshow('bag.png');
```

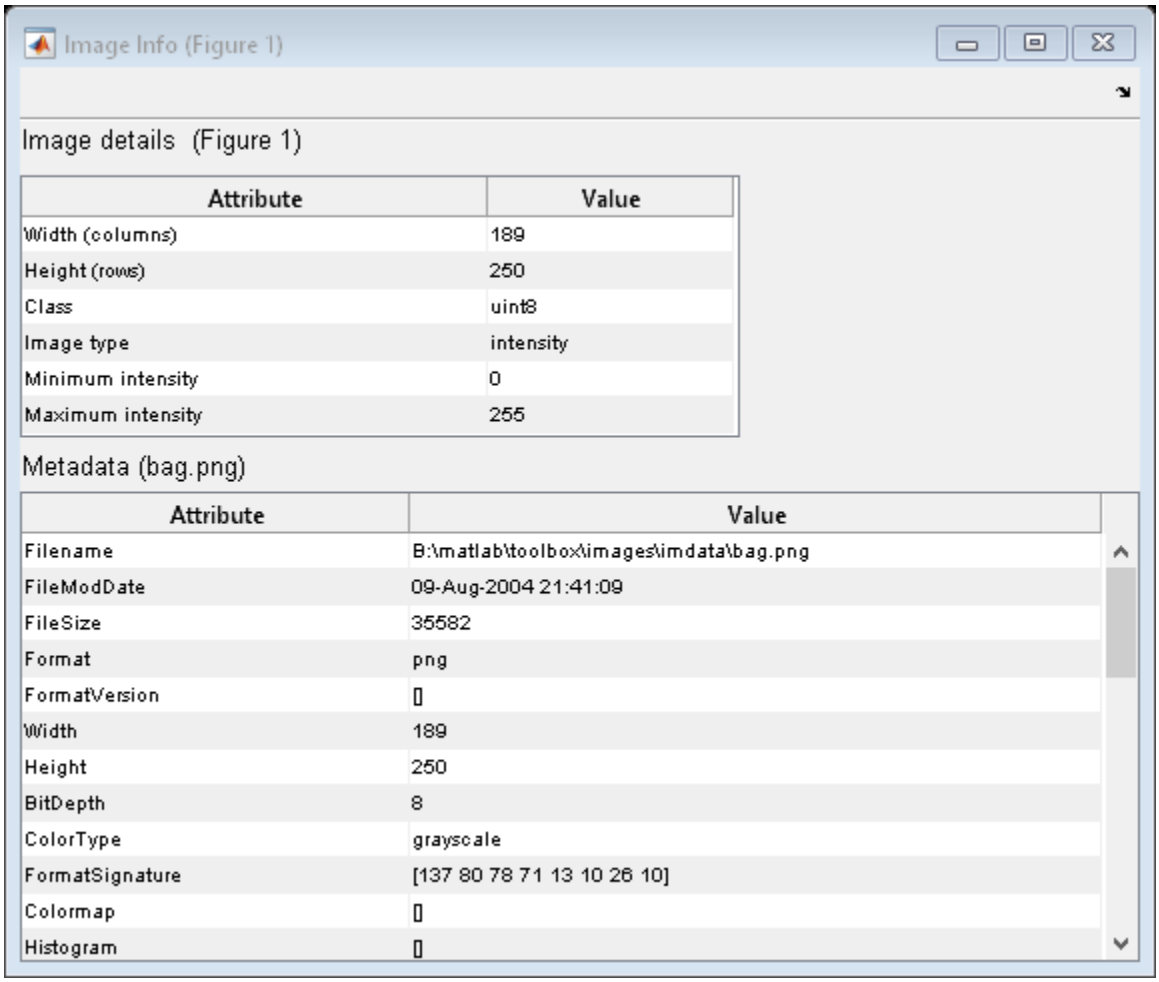


Get the image metadata.

```
info = imfinfo('bag.png');
```

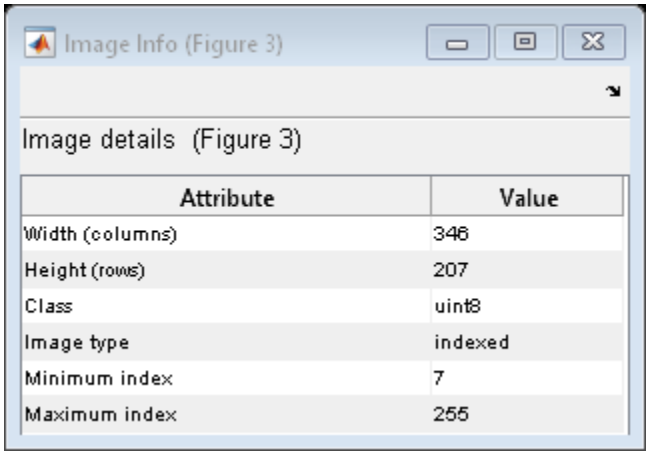
Open an Image Information tool associated with the figure that also contains the image metadata.

```
imageinfo(h,info)
```



Display a new image, then open an Image Information tool associated with the image.

```
imshow('canoe.tif')
imageinfo
```



Input Arguments

`h` — Handle to graphics object

handle

Handle to a figure, axes, uipanel, or image graphics object, specified as a handle. If `h` is an axes or figure handle, then `imageinfo` uses the first image returned by `findobj(h,'Type','image')`.

`filename` — File name

character vector | string scalar

File name, specified as a character vector. `filename` can be any file type that has been registered with an information function in the file formats registry, `imformats`, so its information can be read by `imfinfo`. `filename` can also be a DICOM, NITF, Interfile, or Analyze file.

`info` — Image metadata

structure

Image metadata, specified as a structure returned by the functions `imfinfo`, `dicominfo`, `nitfinfo`, `interfileinfo`, or `analyze75info`. `info` can also be a user-created structure.

`himage` — Handle to image object

handle

Handle to an image graphics object, specified as a handle.

Output Arguments

`htool` — Handle to Image Information tool figure

handle

Handle to Image Information tool figure, returned as a handle.

Tips

- The table lists the basic image attribute information included in the Image Information tool display. Note that the tool contains either four or six fields, depending on the type of image.

Attribute Name	Value
Width (columns)	Number of columns in the image
Height (rows)	Number of rows in the image
Class	Data type used by the image, such as `'uint8'`. **Note** For `single` or `int16` images, `imageinfo` returns a `'Class'` value of `'double'`, because the image object converts the `CData` of these images to `double`.
Image type	One of the image types identified by the Image Processing Toolbox software: `'intensity'` `'truecolor'`, `'binary'`, or `'indexed'`.

Attribute Name	Value
Minimum intensity or index	For grayscale images, this value represents the lowest intensity value of any pixel. For indexed images, this value represents the lowest index value into a colormap. This field is not included for `'binary'` or `'truecolor'` images.
Maximum intensity or index	For grayscale images, this value represents the highest intensity value of any pixel. For indexed images, this value represents the highest index value into a colormap. This field is not included for `'binary'` or `'truecolor'` images.

- `imageinfo` gets information about image attributes by querying the image object's `CData`. The image object converts the `CData` for `single` or `int16` images to class `double`. In these cases, `imageinfo(H)` displays a `'Class'` attribute of `'double'`, even though the image is of class `single` or `int16`. For example,

```
h = imshow(ones(10,'int16'));
class(get(h,'CData'))
```

See Also

Image Viewer | `dicominfo` | `imattributes` | `imfinfo` | `imformats` | `nitfinfo`

Introduced before R2006a

imagemodel

Image model object

Description

An image model object stores information about an image such as class, type, display range, width, height, minimum intensity value and maximum intensity value.

The image model object supports functions that you can use to access this information, get information about the pixels in an image, and perform special text formatting. An `imagemodel` object works by querying the target image `CData`.

Creation

Syntax

```
imgmodel = imagemodel(himage)
```

Description

`imgmodel = imagemodel(himage)` creates an image model object associated with a target image `himage`.

If `himage` is an array of image objects, then `imgmodel` is an array of image model objects.

Input Arguments

`himage` — Target image
handle | array of handles

Target image, specified as a handle or array of handles to image objects.

Object Functions

getClassType	Get class of image from image model
getDisplayRange	Get display range of image from image model
getImageHeight	Get height of image from image model
getImageType	Get type of image from image model
getImageWidth	Get width of image from image model
getMaxIntensity	Get maximum value of image from image model
getMinIntensity	Get minimum value of image from image model
getNumberFormatFcn	Get function handle that converts numeric value into character vector
getPixelInfoString	Get pixel value as character vector
getPixelRegionFormatFcn	Get function handle that formats pixel value into character vector
getPixelValue	Get pixel value as numeric array
getDefaultPixelInfoString	Get default pixel value as character vector

getDefaultPixelRegionString	Get type of information displayed in Pixel Region tool as character vector
getScreenPixelRGBValue	Get screen value of specified pixel in image model
getimagemodel	Image model object from image object

Examples

Create an Image Model from Image Objects

Create an image model associated with a single image object.

```
h = imshow('peppers.png');
```

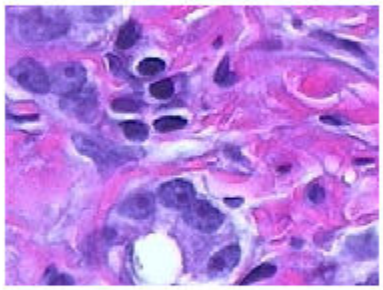
```
im = imagemodel(h)
```

```
im =

IMAGEMODEL object accessing an image with these properties:

       ClassType: 'uint8'
    DisplayRange: []
```

```
     ImageHeight: 384
       ImageType: 'truecolor'
      ImageWidth: 512
    MinIntensity: []
    MaxIntensity: []
```

Create an image model for an array of image object handles.

```
figure
subplot(1,2,1)
h1 = imshow('hestain.png');
subplot(1,2,2)
h2 = imshow('coins.png');
```



```
im = imagemodel([h1 h2])
```

```
im =

1x2 array of IMAGEMODEL objects.
```

Get RGB Pixel Values From Image Model

Pixel values obtained from an `imagemodel` object can be returned in several formats suitable for display in different interactive image processing tools.

Create an image model associated with a color image.

```
h = imshow('flamingos.jpg');
```

```
im = imagemodel(h)
```

```
im =

IMAGEMODEL object accessing an image with these properties:

       ClassType: 'uint8'
    DisplayRange: []
     ImageHeight: 972
       ImageType: 'truecolor'
      ImageWidth: 1296
    MinIntensity: []
    MaxIntensity: []
```

Select a pixel by specifying row and column coordinates. This pixel has (row, column) coordinates (100, 200).

```
r = 100;
c = 200;
```

Get the numeric value of the pixel using the `getPixelValue` function.

```
pxValue = getPixelValue(im,r,c)
```

```
pxValue = 1x3 uint8 row vector

   104    95    54
```

Get the default pixel information string using the `getDefaultPixelInfoString` function. This string depends on the type of image but does not use the pixel values. The pixel information string is suitable for use with the Pixel Information tool.

```
defaultPxInfoStr = getDefaultPixelInfoString(im)
```

```
defaultPxInfoStr =
'[R G B]'
```

Using the same string format, get the pixel information string for the specified pixel by using the `getPixelInfoString` function.

```
pxInfoStr = getPixelInfoString(im,r,c)
```

```
pxInfoStr =
'[104 95 54]'
```

Get the default pixel region string using the `getDefaultPixelRegionString` function. This string depends on the type of image but does not use the pixel values. The pixel region string is suitable for use with the Pixel Region tool.

```
defaultPxRegStr = getDefaultPixelRegionString(im)
```

```
defaultPxRegStr =
    'R:000
     G:000
     B:000'
```

There are two steps to get the pixel region string for the specified pixel in the same string format. First, get a function `formatFcn` that formats numeric pixel values by using the `getPixelRegionFormatFcn` function. Then, specify the row and column coordinate of the pixel as input arguments to `formatFcn` to get the formatted string.

```
formatFcn = getPixelRegionFormatFcn(im);
pxRegStr = formatFcn(r,c)
```

```
pxRegStr = 1x1 cell array
    {'R:104...'}
```

See Also

`getimagemodel` | `imattributes`

Introduced before R2006a

getClassType

Get class of image from image model

Syntax

```
imgclass = getClassType(imgmodel)
```

Description

`imgclass = getClassType(imgmodel)` returns the class associated with the image model, `imgmodel`.

Input Arguments

`imgmodel` — Image model

scalar `imagemodel` object

Image model, specified as a scalar `imagemodel` object.

Output Arguments

`imgclass` — Class of image

character vector

Class of the image `CData`, returned as a character vector such as `'uint8'`.

See Also

`imageinfo` | `getDisplayRange` | `getMaxIntensity` | `getMinIntensity`

Introduced before R2006a

getDefaultPixelInfoString

Get default pixel value as character vector

Syntax

```
pixval = getDefaultPixelInfoString(imgmodel)
```

Description

`pixval = getDefaultPixelInfoString(imgmodel)` returns a default pixel value character vector matching the information displayed in the Pixel Information tool on page 1-1710, based on the type of image in `imgmodel`. This character vector can be used in place of actual pixel information values.

Examples

Get RGB Pixel Values From Image Model

Pixel values obtained from an `imagemodel` object can be returned in several formats suitable for display in different interactive image processing tools.

Create an image model associated with a color image.

```
h = imshow('flamingos.jpg');
```

```
im = imagemodel(h)
```

```
im =

IMAGEMODEL object accessing an image with these properties:

       ClassType: 'uint8'
    DisplayRange: []
     ImageHeight: 972
       ImageType: 'truecolor'
      ImageWidth: 1296
    MinIntensity: []
    MaxIntensity: []
```

Select a pixel by specifying row and column coordinates. This pixel has (row, column) coordinates (100, 200).

```
r = 100;
c = 200;
```

Get the numeric value of the pixel using the `getPixelValue` function.

```
pxValue = getPixelValue(im,r,c)
```

```
pxValue = 1x3 uint8 row vector

   104    95    54
```

Get the default pixel information string using the `getDefaultPixelInfoString` function. This string depends on the type of image but does not use the pixel values. The pixel information string is suitable for use with the Pixel Information tool.

```
defaultPxInfoStr = getDefaultPixelInfoString(im)
```

```
defaultPxInfoStr =
'[R G B]'
```

Using the same string format, get the pixel information string for the specified pixel by using the `getPixelInfoString` function.

```
pxInfoStr = getPixelInfoString(im,r,c)
```

```
pxInfoStr =
'[104 95 54]'
```

Get the default pixel region string using the `getDefaultPixelRegionString` function. This string depends on the type of image but does not use the pixel values. The pixel region string is suitable for use with the Pixel Region tool.

```
defaultPxRegStr = getDefaultPixelRegionString(im)
```

```
defaultPxRegStr =
    'R:000
     G:000
     B:000'
```

There are two steps to get the pixel region string for the specified pixel in the same string format. First, get a function `formatFcn` that formats numeric pixel values by using the `getPixelRegionFormatFcn` function. Then, specify the row and column coordinate of the pixel as input arguments to `formatFcn` to get the formatted string.

```
formatFcn = getPixelRegionFormatFcn(im);
pxRegStr = formatFcn(r,c)
```

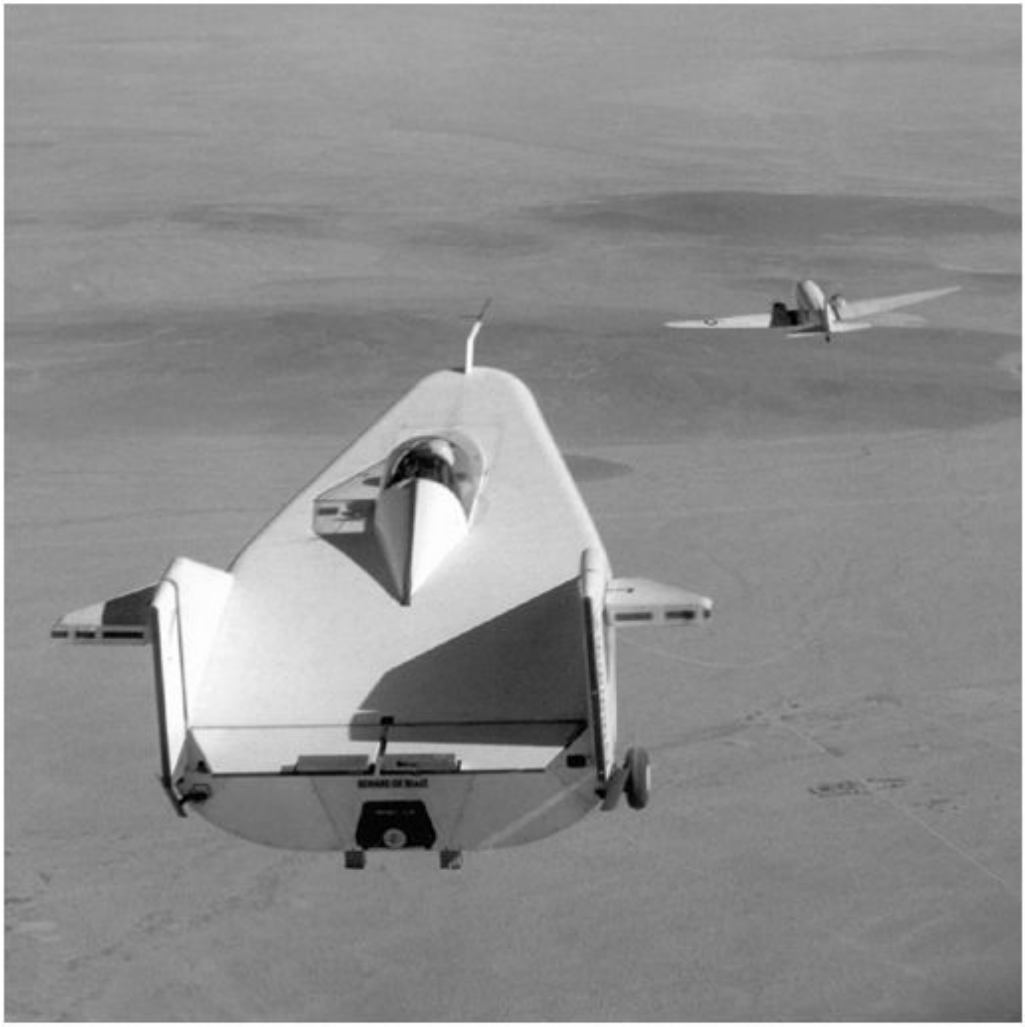
```
pxRegStr = 1x1 cell array
    {'R:104...'}
```

Get Grayscale Pixel Value From Image Model

Pixel values obtained from an `imagemodel` object can be returned in several formats suitable for display in different interactive image processing tools.

Create an image model associated with a grayscale image.

```
h = imshow('liftingbody.png');
```



```
im = imagemodel(h)
```

```
im =

IMAGEMODEL object accessing an image with these properties:

       ClassType: 'uint8'
    DisplayRange: [0 255]
     ImageHeight: 512
       ImageType: 'intensity'
      ImageWidth: 512
    MinIntensity: 0
```

```
    MaxIntensity: 255
```

Select a pixel by specifying the row and column coordinates. This pixel has (row, column) coordinates (50, 250).

```
r = 50;
c = 250;
```

Get the numeric value of the pixel using the `getPixelValue` function.

```
pxValue = getPixelValue(im,r,c)
```

```
pxValue = uint8
    151
```

Convert the numeric pixel value to a string. First, get a function `formatFcn` that formats numeric pixel values by using the `getNumberFormatFcn`. Then, specify the numeric value of the pixel as the input argument to `formatFcn` to get the formatted string.

```
formatFcn = getNumberFormatFcn(im);
pxValueStr = formatFcn(pxValue)
```

```
pxValueStr =
'151'
```

Get the default pixel information string using the `getDefaultPixelInfoString` function. This string depends on the type of image but does not use the pixel values. The pixel information string is suitable for use with the Pixel Information tool.

```
defaultPxInfoStr = getDefaultPixelInfoString(im)
```

```
defaultPxInfoStr =
'Intensity'
```

Using the same string format, get the pixel information string for the specified pixel by using the `getPixelInfoString` function.

```
pxInfoStr = getPixelInfoString(im,r,c)
```

```
pxInfoStr =
'151'
```

Get the default pixel region string using the `getDefaultPixelRegionString` function. This string depends on the type of image but does not use the pixel values. The pixel region string is suitable for use with the Pixel Region tool.

```
defaultPxRegStr = getDefaultPixelRegionString(im)
```

```
defaultPxRegStr =
'000'
```

There are two steps to get the pixel region string for the specified pixel in the same string format. First, get a function `formatFcn` that formats numeric pixel values by using the `getPixelRegionFormatFcn` function. Then, specify the row and column coordinate of the pixel as input arguments to `formatFcn` to get the formatted string.

```
formatFcn = getPixelRegionFormatFcn(im);
pxRegStr = formatFcn(r,c)
```

```
pxRegStr = 1x1 cell array
    {'151'}
```

Input Arguments

`imgmodel` — Image model

scalar `imagemodel` object

Image model, specified as a scalar `imagemodel` object.

Output Arguments

`pixval` — Default pixel value

`'Intensity'` | `'[R G B]'` | `'BW'` | `'<Index> [R G B]'`

Default pixel value, returned as one of the following.

Image Type	Default Pixel Value
Grayscale	`'Intensity'`
Truecolor (RGB)	`'[R G B]'`
Binary	`'BW'`
Indexed	`'<Index> [R G B]'`

Data Types: `char`

See Also

`getPixelInfoString` | `getImageType` | `getDefaultPixelRegionString` | `impixelinfo` | `impixelinfoval`

Introduced before R2006a

getDefaultPixelRegionString

Get type of information displayed in Pixel Region tool as character vector

Syntax

```
pixval = getDefaultPixelRegionString(imgmodel)
```

Description

`pixval = getDefaultPixelRegionString(imgmodel)` returns a default pixel value character vector matching the information displayed in the Pixel Region tool on page 1-1715, based on the type of image in `imgmodel`. This character vector can be used in place of actual pixel information values.

Examples

Get RGB Pixel Values From Image Model

Pixel values obtained from an `imagemodel` object can be returned in several formats suitable for display in different interactive image processing tools.

Create an image model associated with a color image.

```
h = imshow('flamingos.jpg');
```

```
im = imagemodel(h)
```

```
im =

IMAGEMODEL object accessing an image with these properties:

       ClassType: 'uint8'
    DisplayRange: []
     ImageHeight: 972
       ImageType: 'truecolor'
      ImageWidth: 1296
    MinIntensity: []
    MaxIntensity: []
```

Select a pixel by specifying row and column coordinates. This pixel has (row, column) coordinates (100, 200).

```
r = 100;
c = 200;
```

Get the numeric value of the pixel using the `getPixelValue` function.

```
pxValue = getPixelValue(im,r,c)
```

```
pxValue = 1x3 uint8 row vector

   104    95    54
```

Get the default pixel information string using the `getDefaultPixelInfoString` function. This string depends on the type of image but does not use the pixel values. The pixel information string is suitable for use with the Pixel Information tool.

```
defaultPxInfoStr = getDefaultPixelInfoString(im)
```

```
defaultPxInfoStr =
'[R G B]'
```

Using the same string format, get the pixel information string for the specified pixel by using the `getPixelInfoString` function.

```
pxInfoStr = getPixelInfoString(im,r,c)
```

```
pxInfoStr =
'[104 95 54]'
```

Get the default pixel region string using the `getDefaultPixelRegionString` function. This string depends on the type of image but does not use the pixel values. The pixel region string is suitable for use with the Pixel Region tool.

```
defaultPxRegStr = getDefaultPixelRegionString(im)
```

```
defaultPxRegStr =
    'R:000
     G:000
     B:000'
```

There are two steps to get the pixel region string for the specified pixel in the same string format. First, get a function `formatFcn` that formats numeric pixel values by using the `getPixelRegionFormatFcn` function. Then, specify the row and column coordinate of the pixel as input arguments to `formatFcn` to get the formatted string.

```
formatFcn = getPixelRegionFormatFcn(im);
pxRegStr = formatFcn(r,c)
```

```
pxRegStr = 1x1 cell array
    {'R:104...'}
```

Input Arguments

`imgmodel` — Image model

scalar `imagemodel` object

Image model, specified as a scalar `imagemodel` object.

Output Arguments

`pixval` — Default pixel value

`'000'` | `'R:000 G:000 B:000'` | `'0'` | `'<000> R:0.00 G:0.00 B:0.00'`

Default pixel value, returned as one of the following.

Image Type	Default Pixel Value
Grayscale	`'000'`
Truecolor (RGB)	`'R:000 G:000 B:000'`
Binary	`'0'`
Indexed	`'<000> R:0.00 G:0.00 B:0.00'`

Data Types: `char`

See Also

`getPixelRegionFormatFcn` | `getImageType` | `getDefaultPixelInfoString` | `impixelregion`

Introduced before R2006a

getDisplayRange

Get display range of image from image model

Syntax

```
disp_range = getDisplayRange(imgmodel)
```

Description

`disp_range = getDisplayRange(imgmodel)`, returns the display range associated with a grayscale image in `imgmodel`.

Input Arguments

`imgmodel` — Image model

scalar `imagemodel` object

Image model, specified as a scalar `imagemodel` object.

Output Arguments

`disp_range` — Image display range

2-element numeric vector | `[]`

Image display range, returned as a 2-element numeric vector of the form `[min max]` for grayscale images. For other image types, the value returned is an empty array, `[]`.

Data Types: `double`

See Also

`getMaxIntensity` | `getMinIntensity` | `getClassType` | `imdisplayrange`

Introduced before R2006a

getImageHeight

Get height of image from image model

Syntax

```
imgheight = getImageHeight(imgmodel)
```

Description

`imgheight = getImageHeight(imgmodel)` returns the number of rows of the image in image model `imgmodel`.

Input Arguments

`imgmodel` — Image model

scalar `imagemodel` object

Image model, specified as a scalar `imagemodel` object.

Output Arguments

`imgheight` — Image height

positive integer

Image height in rows of pixels, returned as positive integer.

Data Types: `double`

See Also

`getImageWidth` | `imageinfo`

Introduced before R2006a

getImageType

Get type of image from image model

Syntax

```
imgtype = getImageType(imgmodel)
```

Description

`imgtype = getImageType(imgmodel)` returns the type of image in `imgmodel`.

Input Arguments

`imgmodel` — Image model

scalar `imagemodel` object

Image model, specified as a scalar `imagemodel` object.

Output Arguments

`imgtype` — Image type

`'intensity'` | `'truecolor'` | `'binary'` | `'indexed'`

Image type, returned as `'intensity'`, `'truecolor'`, `'binary'`, or `'indexed'`.

See Also

`imageinfo`

Topics

“Image Types in the Toolbox”

Introduced before R2006a

getImageWidth

Get width of image from image model

Syntax

```
imgwidth = getImageWidth(imgmodel)
```

Description

`imgwidth = getImageWidth(imgmodel)` returns the number of columns of the image in image model `imgmodel`.

Input Arguments

`imgmodel` — Image model

scalar `imagemodel` object

Image model, specified as a scalar `imagemodel` object.

Output Arguments

`imgwidth` — Image width

positive integer

Image width in columns of pixels, returned as positive integer.

Data Types: `double`

See Also

`getImageHeight` | `imageinfo`

Introduced before R2006a

getMaxIntensity

Get maximum value of image from image model

Syntax

```
maxval = getMaxIntensity(imgmodel)
```

Description

`maxval = getMaxIntensity(imgmodel)` returns the maximum value of the image in `imgmodel`.

Input Arguments

`imgmodel` — Image model

scalar `imagemodel` object

Image model, specified as a scalar `imagemodel` object.

Output Arguments

`maxval` — Maximum image value

numeric scalar | `[]`

Maximum image value, returned as a numeric scalar. For a grayscale image, the value returned is the maximum intensity, calculated as `max(Image(:))`. For an indexed image, the value returned is the maximum index. For any other image type, the value returned is an empty array, `[]`. The class of `maxval` depends on the class of the target image.

See Also

`getMinIntensity` | `getDisplayRange` | `getClassType` | `imdisplayrange`

Topics

“Image Types in the Toolbox”

Introduced before R2006a

getMinIntensity

Get minimum value of image from image model

Syntax

```
minval = getMinIntensity(imgmodel)
```

Description

`minval = getMinIntensity(imgmodel)` returns the minimum value of the image in `imgmodel`.

Input Arguments

`imgmodel` — Image model

scalar `imagemodel` object

Image model, specified as a scalar `imagemodel` object.

Output Arguments

`minval` — Minimum image value

numeric scalar | `[]`

Minimum image value, returned as a numeric scalar. For a grayscale image, the value returned is the minimum intensity, calculated as `min(Image(:))`. For an indexed image, the value returned is the minimum index. For any other image type, the value returned is an empty array, `[]`. The class of `minval` depends on the class of the target image.

See Also

`getMaxIntensity` | `getDisplayRange` | `getClassType` | `imdisplayrange`

Topics

“Image Types in the Toolbox”

Introduced before R2006a

getNumberFormatFcn

Get function handle that converts numeric value into character vector

Syntax

```
fun = getNumberFormatFcn(imgmodel)
```

Description

`fun = getNumberFormatFcn(imgmodel)` returns a handle to a function that converts a single numeric value into a character vector for image model `imgmodel`.

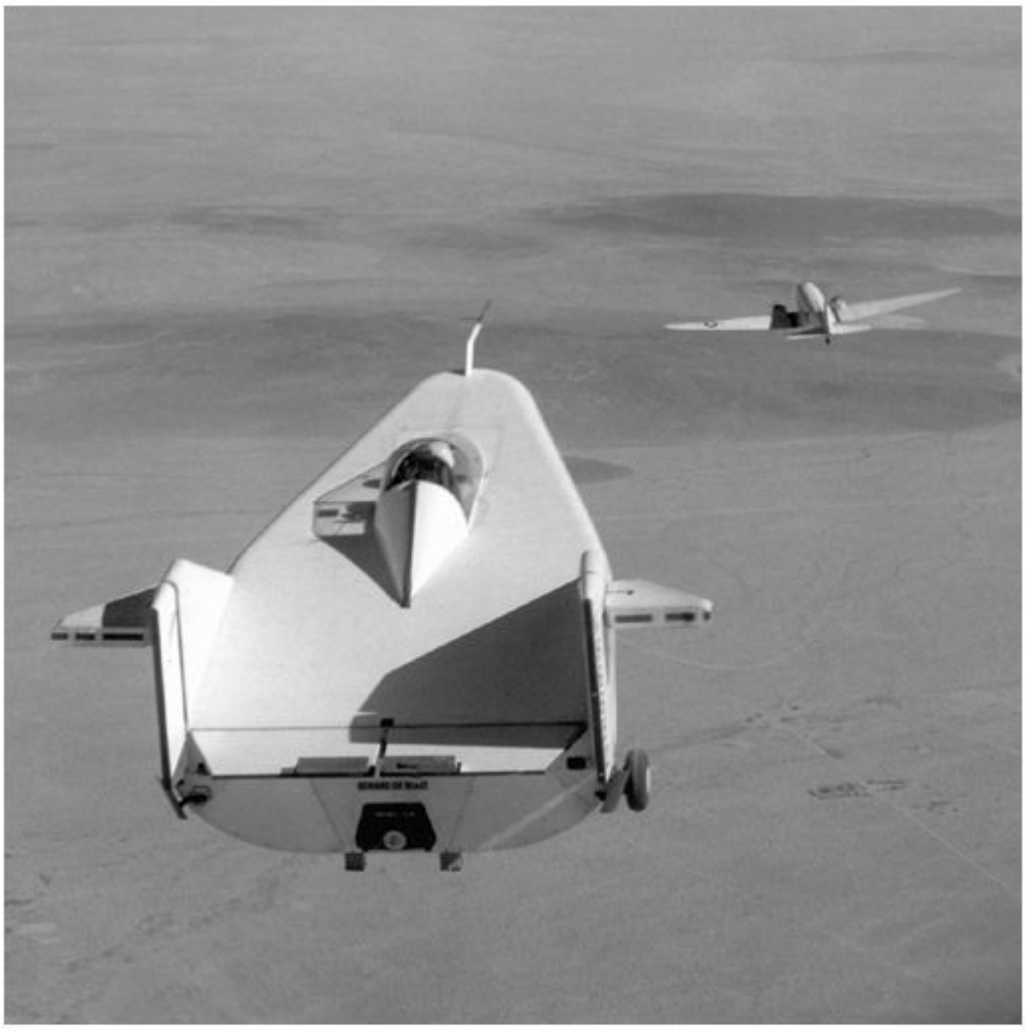
Examples

Get Grayscale Pixel Value From Image Model

Pixel values obtained from an `imagemodel` object can be returned in several formats suitable for display in different interactive image processing tools.

Create an image model associated with a grayscale image.

```
h = imshow('liftingbody.png');
```



```
im = imagemodel(h)
```

```
im =

IMAGEMODEL object accessing an image with these properties:

       ClassType: 'uint8'
    DisplayRange: [0 255]
     ImageHeight: 512
       ImageType: 'intensity'
      ImageWidth: 512
    MinIntensity: 0
    MaxIntensity: 255
```

Select a pixel by specifying the row and column coordinates. This pixel has (row, column) coordinates (50, 250).

```
r = 50;
c = 250;
```

Get the numeric value of the pixel using the `getPixelValue` function.

```
pxValue = getPixelValue(im,r,c)
```

```
pxValue = uint8
    151
```

Convert the numeric pixel value to a string. First, get a function `formatFcn` that formats numeric pixel values by using the `getNumberFormatFcn`. Then, specify the numeric value of the pixel as the input argument to `formatFcn` to get the formatted string.

```
formatFcn = getNumberFormatFcn(im);
pxValueStr = formatFcn(pxValue)
```

```
pxValueStr =
'151'
```

Get the default pixel information string using the `getDefaultPixelInfoString` function. This string depends on the type of image but does not use the pixel values. The pixel information string is suitable for use with the Pixel Information tool.

```
defaultPxInfoStr = getDefaultPixelInfoString(im)
```

```
defaultPxInfoStr =
'Intensity'
```

Using the same string format, get the pixel information string for the specified pixel by using the `getPixelInfoString` function.

```
pxInfoStr = getPixelInfoString(im,r,c)
```

```
pxInfoStr =
'151'
```

Get the default pixel region string using the `getDefaultPixelRegionString` function. This string depends on the type of image but does not use the pixel values. The pixel region string is suitable for use with the Pixel Region tool.

```
defaultPxRegStr = getDefaultPixelRegionString(im)
```

```
defaultPxRegStr =
'000'
```

There are two steps to get the pixel region string for the specified pixel in the same string format. First, get a function `formatFcn` that formats numeric pixel values by using the `getPixelRegionFormatFcn` function. Then, specify the row and column coordinate of the pixel as input arguments to `formatFcn` to get the formatted string.

```
formatFcn = getPixelRegionFormatFcn(im);
pxRegStr = formatFcn(r,c)
```

```
pxRegStr = 1x1 cell array
    {'151'}
```

Input Arguments

`imgmodel` — Image model

scalar `imagemodel` object

Image model, specified as a scalar `imagemodel` object.

Output Arguments

`fun` — Number format function

function handle

Number format function that converts numeric values into character vectors, returned as a function handle. `fun` accepts one input argument, a numeric scalar or logical scalar. `fun` returns the number as a character vector.

Data Types: `function_handle`

See Also

`getPixelValue`

Introduced before R2006a

getPixelInfoString

Get pixel value as character vector

Syntax

```
pixval = getPixelInfoString(imgmodel,r,c)
```

Description

`pixval = getPixelInfoString(imgmodel,r,c)` returns as a character vector the value of a single pixel with (row, column) coordinate (`r`, `c`) in image `imgmodel`. The format of the character vector matches the information displayed in the Pixel Information tool on page 1-1710.

Examples

Get RGB Pixel Values From Image Model

Pixel values obtained from an `imagemodel` object can be returned in several formats suitable for display in different interactive image processing tools.

Create an image model associated with a color image.

```
h = imshow('flamingos.jpg');
```

```
im = imagemodel(h)
```

```
im =

IMAGEMODEL object accessing an image with these properties:

       ClassType: 'uint8'
    DisplayRange: []
     ImageHeight: 972
       ImageType: 'truecolor'
      ImageWidth: 1296
    MinIntensity: []
    MaxIntensity: []
```

Select a pixel by specifying row and column coordinates. This pixel has (row, column) coordinates (100, 200).

```
r = 100;
c = 200;
```

Get the numeric value of the pixel using the `getPixelValue` function.

```
pxValue = getPixelValue(im,r,c)
```

```
pxValue = 1x3 uint8 row vector

   104    95    54
```

Get the default pixel information string using the `getDefaultPixelInfoString` function. This string depends on the type of image but does not use the pixel values. The pixel information string is suitable for use with the Pixel Information tool.

```
defaultPxInfoStr = getDefaultPixelInfoString(im)
```

```
defaultPxInfoStr =
'[R G B]'
```

Using the same string format, get the pixel information string for the specified pixel by using the `getPixelInfoString` function.

```
pxInfoStr = getPixelInfoString(im,r,c)
```

```
pxInfoStr =
'[104 95 54]'
```

Get the default pixel region string using the `getDefaultPixelRegionString` function. This string depends on the type of image but does not use the pixel values. The pixel region string is suitable for use with the Pixel Region tool.

```
defaultPxRegStr = getDefaultPixelRegionString(im)
```

```
defaultPxRegStr =
    'R:000
     G:000
     B:000'
```

There are two steps to get the pixel region string for the specified pixel in the same string format. First, get a function `formatFcn` that formats numeric pixel values by using the `getPixelRegionFormatFcn` function. Then, specify the row and column coordinate of the pixel as input arguments to `formatFcn` to get the formatted string.

```
formatFcn = getPixelRegionFormatFcn(im);
pxRegStr = formatFcn(r,c)
```

```
pxRegStr = 1x1 cell array
    {'R:104...'}
```

Input Arguments

`imgmodel` — Image model

scalar `imagemodel` object

Image model, specified as a scalar `imagemodel` object.

`r` — Row coordinate
positive integer

Row coordinate of pixel, specified as a positive integer.

`c` — Column coordinate
positive integer

Column coordinate of pixel, specified as a positive integer.

Output Arguments

`pixval` — Pixel value
character vector

Pixel value, returned as a character vector. The table shows the character vector returned for a black pixel for each image type.

Image Type	Sample Pixel Value
Grayscale	`'000'`
Truecolor (RGB)	`'[0 0 0]'`
Binary	`'0'`
Indexed	`'<000> [0 0 0]'`

Data Types: `char`

See Also

`getPixelRegionFormatFcn` | `getPixelValue` | `impixelinfo` | `impixelinfoval`

Introduced before R2006a

getPixelRegionFormatFcn

Get function handle that formats pixel value into character vector

Syntax

```
fun = getPixelRegionFormatFcn(imgmodel)
```

Description

`fun = getPixelRegionFormatFcn(imgmodel)` returns a function that formats one or more pixel values in image model `imgmodel` as character vectors. The format of the character vectors matches the information displayed in the Pixel Region tool on page 1-1715.

Examples

Get RGB Pixel Values From Image Model

Pixel values obtained from an `imagemodel` object can be returned in several formats suitable for display in different interactive image processing tools.

Create an image model associated with a color image.

```
h = imshow('flamingos.jpg');
```

```
im = imagemodel(h)
```

```
im =

IMAGEMODEL object accessing an image with these properties:

       ClassType: 'uint8'
    DisplayRange: []
     ImageHeight: 972
       ImageType: 'truecolor'
      ImageWidth: 1296
    MinIntensity: []
    MaxIntensity: []
```

Select a pixel by specifying row and column coordinates. This pixel has (row, column) coordinates (100, 200).

```
r = 100;
c = 200;
```

Get the numeric value of the pixel using the `getPixelValue` function.

```
pxValue = getPixelValue(im,r,c)
```

```
pxValue = 1x3 uint8 row vector

   104    95    54
```

Get the default pixel information string using the `getDefaultPixelInfoString` function. This string depends on the type of image but does not use the pixel values. The pixel information string is suitable for use with the Pixel Information tool.

```
defaultPxInfoStr = getDefaultPixelInfoString(im)
```

```
defaultPxInfoStr =
'[R G B]'
```

Using the same string format, get the pixel information string for the specified pixel by using the `getPixelInfoString` function.

```
pxInfoStr = getPixelInfoString(im,r,c)
```

```
pxInfoStr =
'[104 95 54]'
```

Get the default pixel region string using the `getDefaultPixelRegionString` function. This string depends on the type of image but does not use the pixel values. The pixel region string is suitable for use with the Pixel Region tool.

```
defaultPxRegStr = getDefaultPixelRegionString(im)
```

```
defaultPxRegStr =
    'R:000
     G:000
     B:000'
```

There are two steps to get the pixel region string for the specified pixel in the same string format. First, get a function `formatFcn` that formats numeric pixel values by using the `getPixelRegionFormatFcn` function. Then, specify the row and column coordinate of the pixel as input arguments to `formatFcn` to get the formatted string.

```
formatFcn = getPixelRegionFormatFcn(im);
pxRegStr = formatFcn(r,c)
```

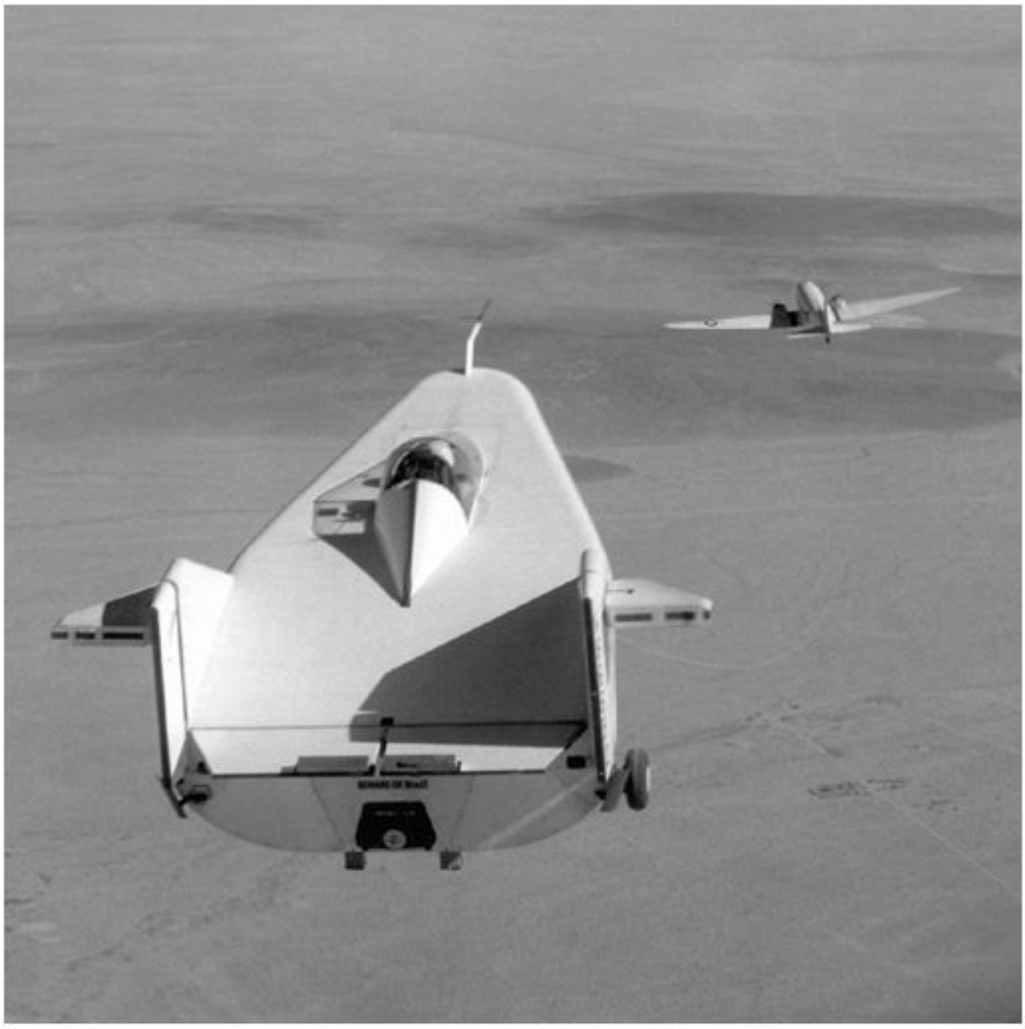
```
pxRegStr = 1x1 cell array
    {'R:104...'}
```

Get Multiple Grayscale Pixel Values From Image Model

Pixel values obtained from an `imagemodel` object can be returned in several formats suitable for display in different interactive image processing tools.

Create an image model associated with a grayscale image.

```
h = imshow('liftingbody.png');
```



```
im = imagemodel(h);
```

Specify the row and column coordinates of multiple pixels as vectors.

```
r = [50 400 500];
c = [250 300 500];
```

Get the numeric value of the pixel using the `getPixelValue` function.

```
pxValue = getPixelValue(im,r,c)
```

```
pxValue = 1x3 uint8 row vector
```

```
   151    74   104
```

There are two steps to get the pixel region strings for the pixels. First, get a function `formatFcn` that formats numeric pixel values by using the `getPixelRegionFormatFcn` function. Then, specify the row and column coordinate of the pixels as input arguments to `formatFcn` to get the formatted strings.

```
formatFcn = getPixelRegionFormatFcn(im);
pxRegStr = formatFcn(r,c)
```

```
pxRegStr = 3x1 cell
    {'151'}
    {'74' }
    {'104'}
```

Input Arguments

`imgmodel` — Image model

scalar `imagemodel` object

Image model, specified as a scalar `imagemodel` object.

Output Arguments

`fun` — Pixel value format function

function handle

Pixel value format function, returned as a function handle. The function `fun` has two input arguments, which are the row and column coordinates of pixels in the target image. For grayscale, indexed, and binary images, `fun` can accept row vectors specifying multiple pixels. For RGB images, `fun` only accepts a single pixel. `fun` returns pixel values as a cell array of character vectors, formatted according to the input image type.

Image Type	Sample Format of Pixel Value
Grayscale	`'000'`
Truecolor (RGB)	`'R:000 G:000 B:000'`
Binary	`'0'`
Indexed	`'<000> R:0.00 G:0.00 B:0.00'`

Data Types: `function_handle`

See Also

`getPixelInfoString` | `getPixelValue` | `getDefaultPixelRegionString` | `impixelregion`

Introduced before R2006a

getPixelValue

Get pixel value as numeric array

Syntax

```
pixval = getPixelValue(imgmodel,r,c)
```

Description

`pixval = getPixelValue(imgmodel,r,c)` returns the numeric value of one or more pixels with (row, column) coordinate (`r`, `c`) in image model `imgmodel`.

Examples

Get RGB Pixel Values From Image Model

Pixel values obtained from an `imagemodel` object can be returned in several formats suitable for display in different interactive image processing tools.

Create an image model associated with a color image.

```
h = imshow('flamingos.jpg');
```

```
im = imagemodel(h)
```

```
im =

IMAGEMODEL object accessing an image with these properties:

       ClassType: 'uint8'
    DisplayRange: []
     ImageHeight: 972
       ImageType: 'truecolor'
      ImageWidth: 1296
    MinIntensity: []
    MaxIntensity: []
```

Select a pixel by specifying row and column coordinates. This pixel has (row, column) coordinates (100, 200).

```
r = 100;
c = 200;
```

Get the numeric value of the pixel using the `getPixelValue` function.

```
pxValue = getPixelValue(im,r,c)
```

```
pxValue = 1x3 uint8 row vector

   104    95    54
```

Get the default pixel information string using the `getDefaultPixelInfoString` function. This string depends on the type of image but does not use the pixel values. The pixel information string is suitable for use with the Pixel Information tool.

```
defaultPxInfoStr = getDefaultPixelInfoString(im)
```

```
defaultPxInfoStr =
'[R G B]'
```

Using the same string format, get the pixel information string for the specified pixel by using the `getPixelInfoString` function.

```
pxInfoStr = getPixelInfoString(im,r,c)
```

```
pxInfoStr =
'[104 95 54]'
```

Get the default pixel region string using the `getDefaultPixelRegionString` function. This string depends on the type of image but does not use the pixel values. The pixel region string is suitable for use with the Pixel Region tool.

```
defaultPxRegStr = getDefaultPixelRegionString(im)
```

```
defaultPxRegStr =
    'R:000
     G:000
     B:000'
```

There are two steps to get the pixel region string for the specified pixel in the same string format. First, get a function `formatFcn` that formats numeric pixel values by using the `getPixelRegionFormatFcn` function. Then, specify the row and column coordinate of the pixel as input arguments to `formatFcn` to get the formatted string.

```
formatFcn = getPixelRegionFormatFcn(im);
pxRegStr = formatFcn(r,c)
```

```
pxRegStr = 1x1 cell array
    {'R:104...'}
```

Input Arguments

`imgmodel` — Image model

scalar `imagemodel` object

Image model, specified as a scalar `imagemodel` object.

`r` — Row coordinate
positive integer | vector of positive integers

Row coordinate of pixel, specified as a positive integer or vector of positive integers.

`c` — Column coordinate
positive integer | vector of positive integers

Column coordinate of pixel, specified as a positive integer or vector of positive integers.

Output Arguments

`pixval` — Pixel value
numeric array | logical array

Pixel value, returned as one of the following.

Input Type	Return Format
p grayscale pixels	*p*-element numeric row vector
p RGB pixels	Numeric row vector of length *p**3. The first *p* elements are the red value for each pixel. The next *p* elements are the green value for each pixel. The last *p* elements are the blue value for each pixel.
p binary pixels	*p*-element logical row vector
p indexed pixels	*p*-by-3 numeric array. Each row specifies a pixel. The columns specify the red, green, and blue components of the pixel value.

See Also
`getPixelRegionFormatFcn` | `getPixelInfoString` | `impixel` | `drawpoint`

Introduced before R2006a

getScreenPixelRGBValue

Get screen value of specified pixel in image model

Syntax

```
pixval = getScreenPixelRGBValue(imgmodel,r,c)
```

Description

`pixval = getScreenPixelRGBValue(imgmodel,r,c)` returns the screen value of one or more pixels with (row, column) coordinate (`r`, `c`) in image `imgmodel`.

Input Arguments

`imgmodel` — Image model

scalar `imagemodel` object

Image model, specified as a scalar `imagemodel` object.

`r` — Row coordinate

positive integer | vector of positive integers

Row coordinate of pixel, specified as a positive integer or vector of positive integers.

`c` — Column coordinate

positive integer | vector of positive integers

Column coordinate of pixel, specified as a positive integer or vector of positive integers.

Output Arguments

`pixval` — Screen pixel value

numeric scalar | numeric vector | numeric array

Pixel value, returned as one of the following.

Input Type	Return Format
p grayscale pixels	*p*-by-3 numeric array. Each row specifies a pixel. The columns specify the red, green, and blue components of the pixel value.
p RGB pixels	Numeric row vector of length *p**3. The first *p* elements are the red value for each pixel. The next *p* elements are the green value for each pixel. The last *p* elements are the blue value for each pixel.
p binary pixels	*p*-by-3 numeric array. Each row specifies a pixel. The columns specify the red, green, and blue components of the pixel value.
p indexed pixels	*p*-by-3 numeric array. Each row specifies a pixel. The columns specify the red, green, and blue components of the pixel value.

Data Types: `double`

See Also

`getPixelValue`

Introduced before R2006a

images.blocked.Adapter class

Package: `images.blocked`

Adapter interface for `blockedImage` objects

Description

The `images.blocked.Adapter` class specifies the interface for block-based reading and writing of array data. Classes that inherit from this interface can be used with `blockedImage` objects, enabling block-based stream processing of array data.

The `images.blocked.Adapter` class is a `handle` class.

Class Attributes

`Abstract`	`true`

For information on class attributes, see “Class Attributes”.

Creation

To implement this class, you must:

- Inherit from the `images.blocked.Adapter` class. Type the following syntax as the first line of your class definition file:

  ```
  classdef MyAdapter < images.blocked.Adapter
      ...
  end
  ```
- Define three required methods for reading image data from disk: `openToRead`, `getInfo`, and `getIOBlock`.
- Optionally, define methods that enable additional reading and writing capabilities. The table lists the complete set of capabilities offered by `Adapter` methods.
- Optionally, for single-file destinations, define an `Extension` property that specifies the file extension to use when automatically creating a destination location. The property must be a string, such as "jpg". For adapters that store data in a folder, do not add this property or specify the value of the property as empty (`[]`).

Capability	Methods to Implement
Read data (Required)	`openToRead` – Open source for reading `getInfo` – Gather information about the source `getIOBlock` – Get specified I/O block
Write data (Optional)	`openToWrite` – Create and open destination for writing `setIOBlock` – Set specified I/O block

Capability	Methods to Implement
Perform clean up tasks (Optional)	`close` – Perform clean up tasks such as closing file handles
Enable parallel block processing (Optional)	`openInParallelToAppend` – Use the adapter in parallel mode with the `apply` object function
Resume writing after interruption (Optional)	`alreadyWritten` – Enable the resume option in the `apply` object function

Examples

Define Adapter to Read 3-D TIFF Files

This example shows how to define and use a custom adapter that reads 3-D TIFF image data as a single volumetric image.

Create a `.m` class definition file that contains the code implementing your custom adapter. You must save this file in your working folder or in a folder that is on the MATLAB® path. The name of the `.m` file must be the same as the name of your object. For example, if you want your adapter to have the name My3DTIFFAdapter, then the name of the `.m` file must be `My3DTIFFAdapter.m`. The `.m` class definition file must contain the following steps:

- Step 1: Inherit from the images.blocked.Adapter class.
- Step 2: Define required methods.

In addition to these steps, define any other properties or methods that you need to process and analyze your data.

```
%% STEP 1: INHERIT FROM ADAPTER CLASS
classdef My3DTIFFAdapter < images.blocked.Adapter

    properties
        File (1,1) string
        Info (1,1) struct
    end

%% STEP 2: DEFINE REQUIRED METHODS
    methods

        % Define the openToRead method
        function openToRead(obj,source)
            obj.File = source;
        end

        % Define the getInfo method
        function info = getInfo(obj)
            % Read raw file info
            finfo = imfinfo(obj.File);

            % Make sure all slices are the same size.
            assert(all(finfo(1).Width==[finfo.Width]), ...
                'All slices do not have the same width.');
            assert(all(finfo(1).Height==[finfo.Height]), ...
                'All slices do not have the same height.');
```

```
            obj.Info.Size = [finfo(1).Height, finfo(1).Width, numel(finfo)];

            % The first two dims of the smallest unit of data that can be
            % read depends on the type of TIFF file - stripped or tiled.
            % The third dim is always 1 - indicating that the smallest unit
            % that can be read in the third dimensions is 1 (one slice).
            if isempty(finfo(1).TileWidth)
                % This is a stripped TIFF file
                obj.Info.IOBlockSize = [finfo(1).RowsPerStrip, finfo(1).Width, 1];
            else
                % This is a tiled TIFF file
                obj.Info.IOBlockSize = [finfo(1).TileLength, finfo(1).TileWidth, 1];
            end

            % This is usually the same for a data set and can be hardcoded.
            assert(finfo(1).BitsPerSample==8 && finfo(1).BitDepth==8)
            obj.Info.Datatype = "uint8";
            obj.Info.InitialValue = cast(0,obj.Info.Datatype);
            info = obj.Info;
        end

        % Define the getIOBlock method
        function block = getIOBlock(obj,ioblockSub,level)
            assert(level==1)

            % Convert ioblockSub (which is in terms of IOBlockSize) into a
            % 'PixelRegion' coordinate.
            regionStart = (ioblockSub-1).*obj.Info.IOBlockSize + 1;
            regionEnd = (ioblockSub).*obj.Info.IOBlockSize;
            rows = [regionStart(1), regionEnd(1)];
            cols = [regionStart(2), regionEnd(2)];
            slices = [regionStart(3), regionEnd(3)];

            block = tiffreadVolume(obj.File, ...
                'PixelRegion',{rows,cols,slices});
        end

    end
end
```

Your custom adapter is now ready. Use `My3DTIFFAdapter` to create an adapter object that reads files with 3-D TIFF image data.

Test Custom 3-D TIFF Adapter

Create a blocked image that reads data using the custom adapter, `My3DTIFFAdapter`. This adapter is attached to the example as a supporting file. Display the size of the blocked image.

```
filename = 'mri.tif';
a = My3DTIFFAdapter

a =
  My3DTIFFAdapter with properties:

    File: ""
```

```
    Info: [1×1 struct]
```

```
bim = blockedImage(filename,'Adapter',My3DTIFFAdapter);
bimSize = bim.Size
```

```
bimSize = 1×3

   128   128    27
```

For comparison, create a blocked image that reads the data using the default adapter. Display the size of the blocked image.

```
bimDefault = blockedImage(filename);
bimDefaultSize = bimDefault.Size
```

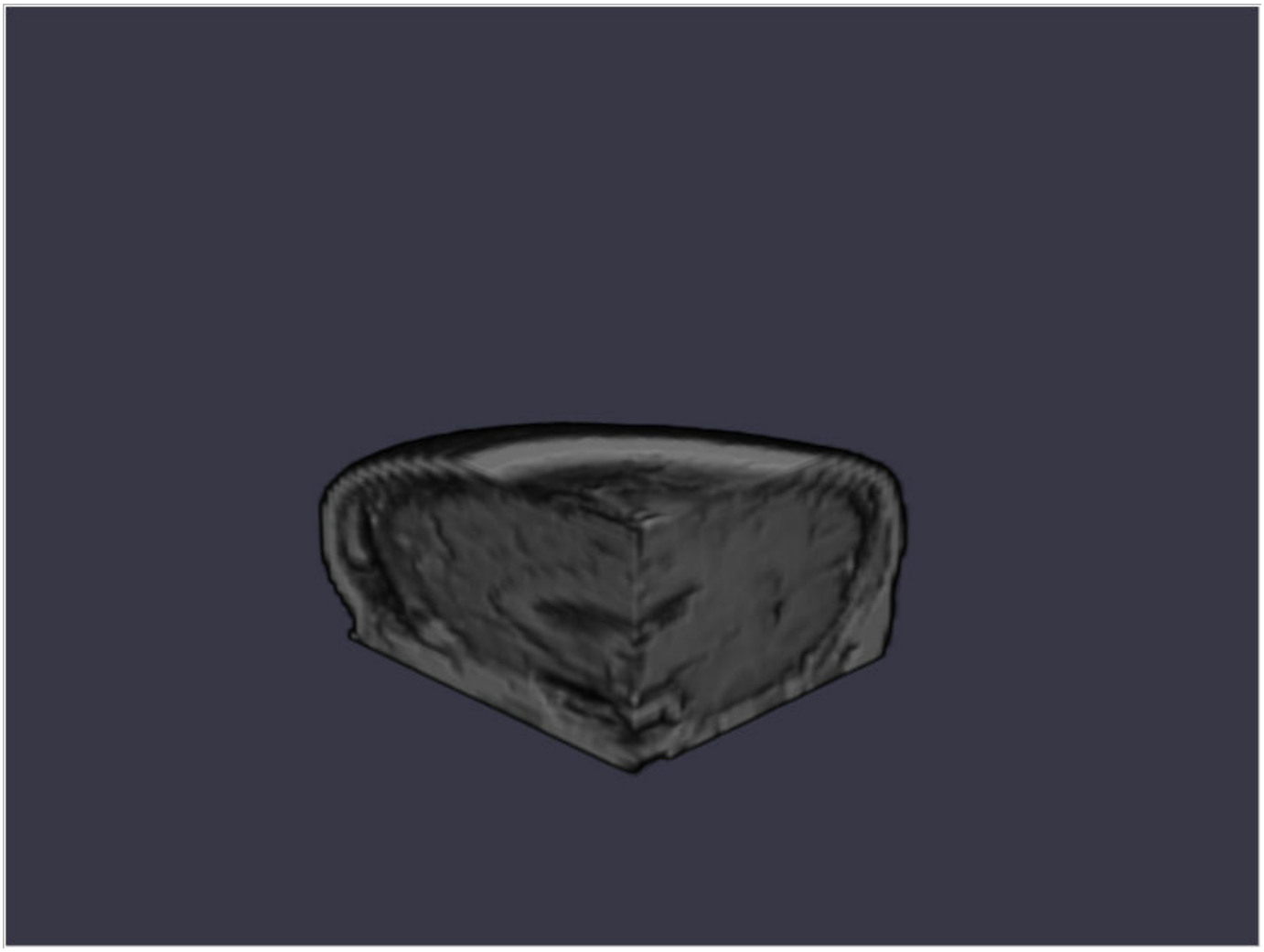
```
bimDefaultSize = 27×2

   128   128
   128   128
   128   128
   128   128
   128   128
   128   128
   128   128
   128   128
   128   128
   128   128
      ⋮
```

Test the custom adapter by reading one quadrant of the volumetric data.

```
regionStart = [1 1 1];
regionEnd = [bimSize(1:2)/2 bimSize(3)];
vol = getRegion(bim,regionStart,regionEnd);
```

Display the data.

```
p = uipanel(figure);
volshow(vol,'Parent',p);
```



Tips

- The toolbox includes several built-in adapters that subclass from the `Adapter` class. All these adapters support both read and write operations. All of the adapters that work on a per-block basis, such as `GenericImageBlocks`, can be used with the parallel mode of the `apply` object function.

Adapter	Description
`BINBlocks`	Store each block as a binary file in a folder
`GenericImage`	Store blocks in a single image
`GenericImageBlocks`	Store each block as an image file in a folder
`H5`	Store blocks in a single HDF5 image
`H5Blocks`	Store each block as an HDF5 file in a folder
`InMemory`	Store blocks in a variable in main memory

Adapter	Description
`JPEGBlocks`	Store each block as a JPEG file in a folder
`MATBlocks`	Store each block as a MAT file in a folder
`PNGBlocks`	Store each block as a PNG file in a folder
`TIFF`	Store blocks in a single TIFF file

See Also

`blockedImage`

Introduced in R2021a

alreadyWritten

Package: `images.blocked`

List of blocks already written

Syntax

```
ioblocksubs = alreadyWritten(obj,level)
```

Description

`ioblocksubs = alreadyWritten(obj,level)` returns a list of block subscripts that have data written to them. This list is used to skip processing existing output in `blockedImage/apply` calls when the parameter `Resume` is set to `true`.

Input Arguments

`obj` — Adapter object

`images.blocked.Adapter` object

Adapter object, specified as an instance of an adapter class that is subclassed from the `images.blocked.Adapter` class.

`level` — Image resolution level

integer-valued scalar

Image resolution level, specified as an integer-valued scalar.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`ioblocksubs` — List of written block

array of block subscripts

List of written blocks, returned as an array of subscripts.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

See Also

`images.blocked.Adapter`

Introduced in R2021a

close

Package: `images.blocked`

Close adapter

Syntax

```
close(obj)
```

Description

`close(obj)` closes and releases resources acquired during `openToRead`, `openToWrite`, and `openInParallelToAppend` methods. Use this method to flush data, close file handles and perform other clean up actions.

`close` may get called more than once.

Input Arguments

`obj` — Adapter object

`images.blocked.Adapter` object

Adapter object, specified as an instance of an adapter class that is subclassed from the `images.blocked.Adapter` class.

See Also

`images.blocked.Adapter`

Introduced in R2021a

getInfo

Package: `images.blocked`

Gather information about source

Syntax

```
info = getInfo(obj)
```

Description

`info = getInfo(obj)` gathers and returns `info`, a structure containing information about the source.

Input Arguments

`obj` — Adapter object

`images.blocked.Adapter` object

Adapter object, specified as an instance of an adapter class that is subclassed from the `images.blocked.Adapter` class.

Output Arguments

`info` — Structure containing information about source

`struct`

Structure containing information about source, returned as a structure with these fields.

Field	Description
`Size`	An *L*-by-*N* integer-valued array representing image size(s); For this field and all subsequent fields,*L* is the number of levels in `Source`. For a single resolution level image *L* is 1. *N* is the number of dimensions in the image.
`IOBlockSize`	An *L*-by-*N* integer-valued array representing the smallest unit of data that can be read from the source.
`Datatype`	An *L*-by-*N* string array containing the MATLAB datatype for each level.
`InitialValue`	A scalar value of type specified by `Datatype`, indicating the initial data value for each level. If the types and values differ for a multiresolution array, this can be a cell array .

You can optionally return these additional fields.

Optional Field	Description
`UserData`	A scalar `struct` containing additional metadata about the source. This field can be empty.
`WorldStart`	An *L*-by-*N* numeric array specifying the starting edge location of the image in world coordinates.*L* is the number of levels in `Source`. *N* is the number of dimensions in the image.
`WorldEnd`	An *L*-by-*N* numeric array specifying the ending edge location of the image in world coordinates.*L* is the number of levels in `Source`. *N* is the number of dimensions in the image.

See Also

`images.blocked.Adapter`

Introduced in R2021a

getIOBlock

Package: `images.blocked`

Read specified I/O block

Syntax

```
block = getIOBlock(obj,ioblocksub,level)
```

Description

`block = getIOBlock(obj,ioblocksub,level)` reads the block specified by the block subscript `ioblocksub` from the specified resolution level `level`. The return value `block` is empty if there is no data for the corresponding block. `blockedImage` uses the `InitialValue` property to create a block for such block subscripts.

Input Arguments

`obj` — Adapter object

`images.blocked.Adapter` object

Adapter object, specified as an instance of an adapter class that is subclassed from the `images.blocked.Adapter` class.

`ioblocksub` — Block subscripts

integer-valued scalar

Block subscripts, specified as an integer-valued scalar. These subscripts span the grid made by `Size./IOBlockSize`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`level` — Image resolution level

integer-valued scalar

Image resolution level, specified as an integer-valued scalar. For single-resolution level files, the `level` parameter is always 1.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`block` — Data read

numeric array

Data read, returned as a numeric array.

See Also

`images.blocked.Adapter`

Introduced in R2021a

openInParallelToAppend

Package: `images.blocked`

Open destination on parallel worker to append blocks

Syntax

```
openInParallelToAppend(obj, destination)
```

Description

`openInParallelToAppend(obj, destination)` opens the location specified by `destination` on a parallel worker in preparation for appending blocks. This function is only invoked when the `UseParallel` parameter of `blockedImage/apply` is set to `true`. `openToWrite` is guaranteed to have been called once on a corresponding `destination` value earlier on a separate instance of the adapter in the main MATLAB session. A copy of the adapter is made for each parallel worker and `openInParallelToAppend` is called once before subsequent `setIOBlock`. The adapter uses the `AlternateFileSystemRoots` property of the blocked image to resolve `destination` on the worker.

An adapter which implements this method must be able to support multiple adapter instances (one on each parallel worker) appending blocks to the same destination simultaneously. This is usually handled by writing independent files for each block in a single destination folder.

Input Arguments

`obj` — Adapter object

`images.blocked.Adapter` object

Adapter object, specified as an instance of an adapter class that is subclassed from the `images.blocked.Adapter` class.

`destination` — Location

`string` scalar | `char` vector

Location, specified as a `string` scalar or `char` vector.

Data Types: `char` | `string`

See Also

`images.blocked.Adapter`

Introduced in R2021a

openToRead

Package: `images.blocked`

Open source for read access

Syntax

```
openToRead(obj,source)
```

Description

`openToRead(obj,source)` opens the `source`, specified as a scalar string, for read access. This method issues an error if the adapter does not support `source`.

Input Arguments

`obj` — Adapter object

`images.blocked.Adapter` object

Adapter object, specified as an instance of an adapter class that is subclassed from the `images.blocked.Adapter` class.

`source` — Location to read from

`string` scalar | `char` vector

Location to read from, specified as a `string` scalar or `char` vector.

Data Types: `char` | `string`

Tips

- `openToRead` gets called when a previously saved `blockedImage` is loaded from a MAT file.

See Also

`images.blocked.Adapter`

Introduced in R2021a

openToWrite

Package: `images.blocked`

Create and open destination for writing

Syntax

```
openToWrite(obj,destination,info,currentlevel)
```

Description

`openToWrite(obj,destination,info,currentlevel)` opens the location specified by the `string` scalar `destination` for writing. `info` is a scalar structure that describes the object. `currentlevel` is an integer-valued scalar indicating the current level for which data will be written.

Use this method to prepare the destination for writing. For example, open file handle or create destination folder, or write file header or meta data to the destination. When writing multiresolution (pyramid) images, `openToWrite` is called for each level before the corresponding level's `setIOBlock` is called.

Input Arguments

`obj` — Adapter object

`images.blocked.Adapter` object

Adapter object, specified as an instance of an adapter class that is subclassed from the `images.blocked.Adapter` class.

`destination` — Location to write to

`string` scalar | `char` vector

Location to write to, specified as a `string` scalar or `char` vector.

Data Types: `char` | `string`

`info` — Structure containing information about source

`struct`

Structure containing information about source, returned as a structure with these fields.

Field	Description
`Size`	An *L*-by-*N* integer-valued array representing image size(s); For this field and all subsequent fields,*L* is the number of levels in `Source`. For a single resolution level image *L* is 1. *N* is the number of dimensions in the image.

Field	Description
`IOBlockSize`	An *L*-by-*N* integer-valued array representing the smallest unit of data that can be read from the source.
`Datatype`	An *L*-by-*N* string array containing the MATLAB datatype for each level.
`InitialValue`	A scalar value of type specified by `Datatype`, indicating the initial data value for each level. If the types and values differ for a multiresolution array, this can be a cell array .

You can optionally return these additional fields.

Optional Field	Description
`UserData`	A scalar `struct` containing additional metadata about the source. This field can be empty.
`WorldStart`	An *L*-by-*N* numeric array specifying the starting edge location of the image in world coordinates.*L* is the number of levels in `Source`. *N* is the number of dimensions in the image.
`WorldEnd`	An *L*-by-*N* numeric array specifying the ending edge location of the image in world coordinates.*L* is the number of levels in `Source`. *N* is the number of dimensions in the image.

`currentlevel` — Image resolution level
integer-valued scalar

Image resolution level, specified as an integer-valued scalar.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

See Also

`images.blocked.Adapter`

Introduced in R2021a

setIOBlock

Package: `images.blocked`

Write specified block

Syntax

```
setIOBlock(obj,ioblocksub, level,block)
```

Description

`setIOBlock(obj,ioblocksub, level,block)` writes the data, `block`, to the specified block subscript, `ioblocksub`, at the specified multi-resolution level, `level`.

Input Arguments

`obj` — Adapter object

`images.blocked.Adapter` object

Adapter object, specified as an instance of an adapter class that is subclassed from the `images.blocked.Adapter` class.

`ioblocksub` — Block subscript

numeric array

Block subscript, specified as a numeric array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`level` — Image resolution level

integer-valued scalar

Image resolution level, specified as an integer-valued scalar. For single-resolution images, `level` is always 1.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`block` — Block of data

numeric array

Block of data, specified as a numeric array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

See Also

`images.blocked.Adapter`

Introduced in R2021a

images.dicom.decodeUID

Get information about DICOM unique identifier

Syntax

```
details = images.dicom.decodeUID(uid)
```

Description

`details = images.dicom.decodeUID(uid)` returns information about the DICOM unique identifier contained in `uid`. `details` contains the name and type of the UID. If `uid` corresponds to a transfer syntax, then `details` also contains the endianness and the DICOM value representation (VR) necessary for reading the image pixels, and information about compression.

The `images.dicom.decodeUID` function can interpret IDs defined in the PS 3.6-1999 specification, with some additions from PS 3.6-2009.

Examples

Decode DICOM UID

Read metadata from a DICOM file and extract a UID field.

```
info = dicominfo("CT-MONO2-16-ankle.dcm");
uid = info.SOPClassUID;
```

Decode the UID.

```
uid_info = images.dicom.decodeUID(uid)
```

```
uid_info = struct with fields:
    Value: '1.2.840.10008.5.1.4.1.1.7'
     Name: 'Secondary Capture Image Storage'
     Type: 'SOP Class'
```

Input Arguments

uid — DICOM unique identifier

character vector | string scalar

DICOM unique identifier, specified as a string or character vector.

Example: `"1.2.840.10008.5.1.4.1.1.7"`

Data Types: `char` | `string`

Output Arguments

`details` — Information from UID

structure

Information from UID, returned as a structure with these fields:

Field	Description
`Value`	Value of the UID, returned as a character vector. If `uid` is not a valid UID, then the value of this field is an empty character vector, `''`.
`Name`	Name of the UID, returned as a character vector.
`Type`	Type of the UID, returned as a character vector.

For transfer syntaxes, the structure contains these additional fields:

Field	Description
`Compressed`	Compression, returned as a logical `true` or `false`.
`Endian`	Endianness, returned as `'ieee-be'` or `'ieee-le'`.
`PixelEndian`	Pixel endianness, returned as `'ieee-be'` or `'ieee-le'`.
`VR`	Value representation (VR), returned as `'Implicit'` or `'Explicit'`.
`LossyCompression`	Compression type is lossy, returned as a logical `true` or `false`.
`CompressionType`	Compression type, returned as a character vector.

See Also

`dicominfo` | `dicomuid`

Introduced in R2017b

images.dicom.parseDICOMDIR

Extract metadata from DICOMDIR file

Syntax

```
info = images.dicom.parseDICOMDIR(filename)
```

Description

`info = images.dicom.parseDICOMDIR(filename)` extracts the metadata from the DICOM directory file (DICOMDIR) named in `filename`, returning the information in the structure `info`. If `filename` is not a DICOMDIR file, then the function returns an error.

A DICOMDIR is a special DICOM file that serves as a directory to a collection of DICOM files stored on removable media, such as CD- and DVD-ROMs. When devices write DICOM files to removable media, they typically write a DICOMDIR file on the disk to serve as a list of the disk contents.

Examples

Extract Metadata from DICOMDIR File

Read information about the contents of a DICOM folder into the workspace.

```
detailsStruct = images.dicom.parseDICOMDIR("DICOMDIR");
```

Input Arguments

`filename` — Name of DICOMDIR file

string scalar | character vector

Name of the DICOMDIR file, specified as a string scalar or character vector. `filename` can specify a full path name or a relative path name to the file.

Data Types: `char` | `string`

Output Arguments

`info` — Metadata from DICOMDIR file

structure

Metadata from the DICOMDIR file, returned as a structure.

See Also

`dicominfo`

Introduced in R2017b

images.roi.CircleMovingEventData class

Package: `images.roi`

Event data passed when circle ROI is moving

Description

The `images.roi.CircleMovingEventData` class is the class passed to listeners when a `Circle` ROI is moving. When the ROI class triggers an event using the `notify` handle class method, MATLAB assigns values to the properties of an `images.roi.CircleMovingEventData` object and passes that object to the listener callback function (the event handler).

The `images.roi.CircleMovingEventData` class is a `handle` class.

Class Attributes

`ConstructOnLoad`	`true`
`HandleCompatible`	`true`

For information on class attributes, see “Class Attributes”.

Creation

The `notify` handle class method creates an `images.roi.CircleMovingEventData` object when called to trigger an event. `images.roi.CircleMovingEventData` accepts no input arguments.

Properties

`Source` — Event source

object

Event source, specified as a handle to the object that triggered the event.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`
`GetObservable`	`true`
`SetObservable`	`true`

`EventName` — Name of event

character vector

Name of the event, specified as a character vector.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

Data Types: `char`

`PreviousCenter` — Position before ROI moved

two-element numeric vector

Position before ROI moved, specified as a two-element numeric vector of the form `[x y]`.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

`CurrentCenter` — Position after ROI moved

two-element numeric vector

Position after ROI moved, specified as a two-element numeric vector of the form `[x y]`.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

`PreviousRadius` — Radius before change in size

numeric scalar

Radius before change in size, specified as a numeric scalar.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

`CurrentRadius` — Radius after change in size

numeric scalar

Radius after change in size, specified as a numeric scalar.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

Examples

Access Event Data

This callback function gets the event source object handle and the event name and other properties from the `images.roi.CircleMovingEventData` object passed to it when the event is triggered.

```
function myCallbk(s,evtData)
   eventSource = evtData.Source;
   eventName = evtData.EventName;
   previousCenter = evtData.PreviousCenter;
   currentCenter = evtData.CurrentCenter;
   previousRadius = evtData.PreviousRadius;
   currentRadius = evtData.CurrentRadius;
end
```

See Also

`drawcircle` | `Circle`

Introduced in R2018b

images.roi.CuboidMovingEventData class

Package: `images.roi`

Event data passed when cuboid ROI is moving

Description

The `images.roi.CuboidMovingEventData` class is the class passed to listeners when a `Cuboid` ROI is moving. When the ROI class triggers an event using the `notify` handle class method, MATLAB assigns values to the properties of an `images.roi.CuboidMovingEventData` object and passes that object to the listener callback function (the event handler).

The `images.roi.CuboidMovingEventData` class is a `handle` class.

Class Attributes

`ConstructOnLoad`	`true`
`HandleCompatible`	`true`

For information on class attributes, see “Class Attributes”.

Creation

The `notify` handle class method creates an `images.roi.CuboidMovingEventData` object when called to trigger an event. `images.roi.CuboidMovingEventData` accepts no input arguments.

Properties

`Source` — Event source

object

Event source, specified as a handle to the object that triggered the event.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`
`GetObservable`	`true`
`SetObservable`	`true`

`EventName` — Name of event

character vector

Name of the event, specified as a character vector.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

`PreviousPosition` — Position before ROI moved

1-by-6 numeric array

Position before ROI moved, specified as a 1-by-6 numeric array of the form `[x y z w h d]`.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

`CurrentPosition` — Position after ROI moved

1-by-6 numeric array

Position after ROI moved, specified as a 1-by-6 numeric array of the form `[x y z w h d]`.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

`PreviousRotationAngle` — Orientation before ROI rotated

1-by-3 numeric array

Orientation before ROI rotated, specified as a 1-by-3 numeric array, measured in degrees.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

`CurrentRotationAngle` — Orientation after ROI rotated

1-by-3 numeric array

Orientation after ROI rotated, specified as a 1-by-3 numeric array, measured in degrees.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

Examples

Access Event Data

This callback function gets the event source object handle, the event name, and other properties from the `images.roi.CuboidMovingEventData` object passed to it when the event is triggered.

```
function myCallbk(s,evtData)
   eventSource = evtData.Source;
   eventName = evtData.EventName;
   previousCenter = evtData.PreviousCenter;
   currentCenter = evtData.CurrentCenter;
   previousRotationAngle = evtData.PreviousRotationAngle;
   currentRotationAngle = evtData.CurrentRotationAngle;
end
```

See Also

`Cuboid` | `drawcuboid`

Introduced in R2019a

images.roi.EllipseMovingEventData class

Package: `images.roi`

Event data passed when ellipse ROI is moving

Description

The `images.roi.EllipseMovingEventData` class is the class passed to listeners when an `Ellipse` ROI is moving. When the ROI class triggers an event using the `notify` handle class method, MATLAB assigns values to the properties of an `images.roi.EllipseMovingEventData` object and passes that object to the listener callback function (the event handler).

The `images.roi.EllipseMovingEventData` class is a `handle` class.

Class Attributes

`ConstructOnLoad`	`true`
`HandleCompatible`	`true`

For information on class attributes, see “Class Attributes”.

Creation

The `notify` handle class method creates an `images.roi.EllipseMovingEventData` object when called to trigger an event. `images.roi.EllipseMovingEventData` accepts no input arguments.

Properties

`Source` — Event source

object

Event source, specified as a handle to the object that triggered the event.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`
`GetObservable`	`true`
`SetObservable`	`true`

`EventName` — Name of event

character vector

Name of event, specified as a character vector.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

Data Types: `char`

`PreviousCenter` — Position before ROI moved

two-element numeric vector

Position before ROI moved, specified as a two-element numeric vector of the form `[m n]`.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

`CurrentCenter` — Position after ROI moved

two-element numeric vector

Position after ROI moved, specified as a two-element numeric vector of the form `[m n]`.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

`PreviousSemiAxes` — Lengths of semiaxes before ROI was reshaped

two-element numeric vector

Lengths of semiaxes before ROI was reshaped, specified as a two-element numeric vector.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

`CurrentSemiAxes` — Lengths of semiaxes after ROI was reshaped

two-element numeric vector

Lengths of semiaxes after ROI was reshaped, specified as a two-element numeric vector.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

`PreviousRotationAngle` — Orientation of ROI before rotation

numeric scalar

Orientation of ROI before rotation, specified as a numeric scalar, measured in degrees.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`
`GetObservable`	`true`
`SetObservable`	`true`

`CurrentRotationAngle` — Orientation of ROI after rotation

numeric scalar

Position after ROI moved, specified as a numeric scalar, measured in degrees.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`
`GetObservable`	`true`
`SetObservable`	`true`

Examples

Access Event Data

This callback function gets the event source object handle and the event name and other properties from the `images.roi.EllipseMovingEventData` object passed to it when the event is triggered.

```
function myCallbk(s,evtData)
   eventSource = evtData.Source;
   eventName = evtData.EventName;
   previousCenter = evtData.PreviousCenter;
   currentCenter = evtData.CurrentCenter;
   previousSemiAxes = evtData.PreviousRadius;
   currentSemiAxes = evtData.CurrentRadius;
   previousRotationAngle = evtData.PreviousRotationAngle;
   currentRotationAngle = evtData.CurrentRotationAngle;
end
```

See Also

`Ellipse` | `drawellipse`

Introduced in R2018b

images.roi.RectangleMovingEventData class

Package: `images.roi`

Event data passed when rectangle ROI is moving

Description

The `images.roi.RectangleMovingEventData` class is the class passed to listeners when a `Rectangle` ROI is moving. When the ROI class triggers an event using the `notify` handle class method, MATLAB assigns values to the properties of an `images.roi.RectangleMovingEventData` object and passes that object to the listener callback function (the event handler).

The `images.roi.RectangleMovingEventData` class is a `handle` class.

Class Attributes

`ConstructOnLoad`	`true`
`HandleCompatible`	`true`

For information on class attributes, see “Class Attributes”.

Creation

The `notify` handle class method creates an `images.roi.RectangleMovingEventData` object when called to trigger an event. `images.roi.RectangleMovingEventData` accepts no input arguments.

Properties

`Source` — Event source

object

Event source, specified as a handle to the object that triggered the event.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`
`GetObservable`	`true`
`SetObservable`	`true`

`EventName` — Name of event

character vector

Name of the event, specified as a character vector.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

`PreviousPosition` — Position before ROI moved

two-element numeric vector

Position before ROI moved, specified as a two-element numeric vector of the form `[m n]`.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

`CurrentPosition` — Position after ROI moved

two-element numeric vector

Position after ROI moved, specified as a two-element numeric vector of the form `[m n]`.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

`PreviousRotationAngle` — Orientation before ROI moved

numeric scalar

Orientation before ROI moved, specified as a numeric scalar, measured in degrees.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

`CurrentRotationAngle` — Orientation after ROI moved

numeric scalar

Orientation after ROI moved, specified as a numeric scalar, measured in degrees.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

Examples

Access Event Data

This callback function gets the event source object handle, the event name, and other properties from the `images.roi.RectangleMovingEventData` object passed to it when the event is triggered.

```
function myCallbk(s,evtData)
   eventSource = evtData.Source;
   eventName = evtData.EventName;
   previousCenter = evtData.PreviousCenter;
   currentCenter = evtData.CurrentCenter;
   previousRotationAngle = evtData.PreviousRotationAngle;
   currentRotationAngle = evtData.CurrentRotationAngle;
end
```

See Also

`drawrectangle` | `Rectangle`

Introduced in R2018b

images.roi.ROIClickedEventData class

Package: `images.roi`

Event data passed when ROI is clicked

Description

The `images.roi.ROIClickedEventData` class is the class passed to listeners when a region-of-interest (ROI) is clicked. When the ROI class triggers an event using the `notify` handle class method, MATLAB assigns values to the properties of an `images.roi.ROIClickedEventData` object and passes that object to the listener callback function (the event handler).

The `images.roi.ROIClickedEventData` class is a `handle` class.

Class Attributes

`ConstructOnLoad`	`true`
`HandleCompatible`	`true`

For information on class attributes, see “Class Attributes”.

Creation

The `notify` handle class method creates an `images.roi.ROIClickedEventData` object when called to trigger an event. `images.roi.ROIClickedEventData` does not accept input arguments.

Properties

`Source` — Event source

object

Event source object, specified as a handle to the object that triggered the event.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`
`GetObservable`	`true`
`SetObservable`	`true`

`EventName` — Name of event

character vector

Name of the event, specified as a character vector.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

SelectionType — Type of selection

character vector

Type of selection, specified as one of the following character vectors.

SelectionType Value	Description
'left'	Left mouse-click
'right'	Right mouse-click
'double'	Double-click
'shift'	Shift-left mouse-click
'ctrl'	Control-left mouse-click

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

SelectedPart — Part of ROI that was clicked

character vector

Part of the ROI that was clicked, specified as one of the character vectors in this table.

SelectedPart Value	Description
'edge'	Clicked edge of ROI.
'face'	Clicked face of ROI.
'label'	Clicked ROI label.
'marker'	Clicked marker used to reshape the ROI.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

CurrentSelected — ROI is currently selected

logical scalar

ROI is currently selected, specified as a logical scalar. Returns `1` when the ROI is selected, otherwise, `0`. To deselect an ROI, use **Ctrl-click**.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`
`GetObservable`	`true`
`SetObservable`	`true`

`PreviousSelected` — ROI was previously selected

logical scalar

ROI was previously selected, specified as a logical scalar. Returns `1` when the ROI was already selected and `0` when the ROI was not previously selected.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`
`GetObservable`	`true`
`SetObservable`	`true`

Examples

Access Event Data

This callback function gets the event source object handle, the event name, and other properties from the `images.roi.ROIClickedEventData` object passed to it when the event is triggered.

```
function myCallbk(s,evtData)
   eventSource = evtData.Source;
   eventName = evtData.EventName;
   selectionType = evtData.SelectionType;
   selectedPart = evtData.SelectedPart;
   currselected = evtData.CurrentSelected;
   prevselected = evtData.PreviousSelected;
end
```

See Also

`addlistener` | `notify` | `images.roi.ROIMovingEventData` | `AssistedFreehand` | `Circle` | `Crosshair` | `Cuboid` | `Ellipse` | `Freehand` | `Line` | `Point` | `Polyline` | `Rectangle` | `Polygon`

Topics

“Use Wait Function After Drawing ROI”

Introduced in R2018b

images.roi.ROIMovingEventData class

Package: `images.roi`

Event data passed when ROI is moving

Description

The `images.roi.ROIMovingEventData` class is the class passed to listeners when a region-of-interest (ROI) is moving. When the ROI class triggers an event using the `notify` handle class method, MATLAB assigns values to the properties of an `images.roi.ROIMovingEventData` object and passes that object to the listener callback function (the event handler).

The `images.roi.ROIMovingEventData` class is a `handle` class.

Class Attributes

`ConstructOnLoad`	`true`
`HandleCompatible`	`true`

For information on class attributes, see “Class Attributes”.

Creation

The `notify` handle class method creates an `images.roi.ROIMovingEventData` object when called to trigger an event. `images.roi.ROIMovingEventData` does not accept input arguments. Subclasses of `event.EventData` cannot pass arguments to the superclass constructor.

Properties

`Source` — Event source

object

Event source, specified as a handle to the object that triggered the event.

Attributes:

`GetAccess`	`public`
`SetAccess`	`private`
`GetObservable`	`true`
`SetObservable`	`true`

`EventName` — Name of event

character vector

Name of the event, specified as a character vector.

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

Data Types: char

PreviousPosition — Position before ROI moved

two-element numeric vector

Position before the ROI moved, specified as a two-element numeric vector of the form [x y].

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

Data Types: char

CurrentPosition — Position after ROI moved

two-element numeric vector

Position after the ROI moved, specified as a two-element numeric vector of the form [x y].

Attributes:

GetAccess	public
SetAccess	private
GetObservable	true
SetObservable	true

Data Types: char

Examples

Access Event Data

This callback function gets the event source object handle, the event name, and other properties from the images.roi.ROIMovingEventData object passed to it when the event is triggered.

```
function myCallbk(s,evtData)
   eventSource = evtData.Source;
   eventName = evtData.EventName;
   previousPosition = evtData.PreviousPosition;
   currentPosition = evtData.CurrentPosition;
end
```

See Also

AssistedFreehand | Circle | Crosshair | Cuboid | Ellipse | Freehand | Line | Point | Polyline | Rectangle | Polygon

Introduced in R2018b

images.stack.browser.CrosshairMovingEventData class

Package: `images.stack.browser`

Event data passed when Crosshair ROI is moving

Description

The `images.stack.browser.CrosshairMovingEventData` class is the class passed to listeners when the crosshair in an `orthosliceViewer` object is moved interactively. The `orthosliceViewer` object triggers an event using the `notify` handle class method. MATLAB assigns values to the properties of an `images.stack.browser.CrosshairMovingEventData` object and passes that object to the listener callback function (the event handler). Programmatic positioning of the crosshair does not trigger this event.

The `images.stack.browser.CrosshairMovingEventData` class is a `handle` class.

Creation

The `notify` handle class method creates an `images.stack.browser.CrosshairMovingEventData` object when called to trigger an event.

Properties

`Source` — Event source

object

Event source, specified as a handle to the object that triggered the event.

`EventName` — Name of event

character vector

Name of the event, specified as a character vector.

`PreviousPosition` — Position before the crosshair moved

three-element numeric vector

Position before the crosshair moved, specified as a three-element numeric vector of the form `[x y z]`.

`CurrentPosition` — Position after crosshair moved

three-element numeric vector

Position after crosshair moved, specified as a three-element numeric vector of the form `[x y z]`.

Examples

Set Up Listener for Orthoslice Viewer Crosshair Events

Load a stack of images.

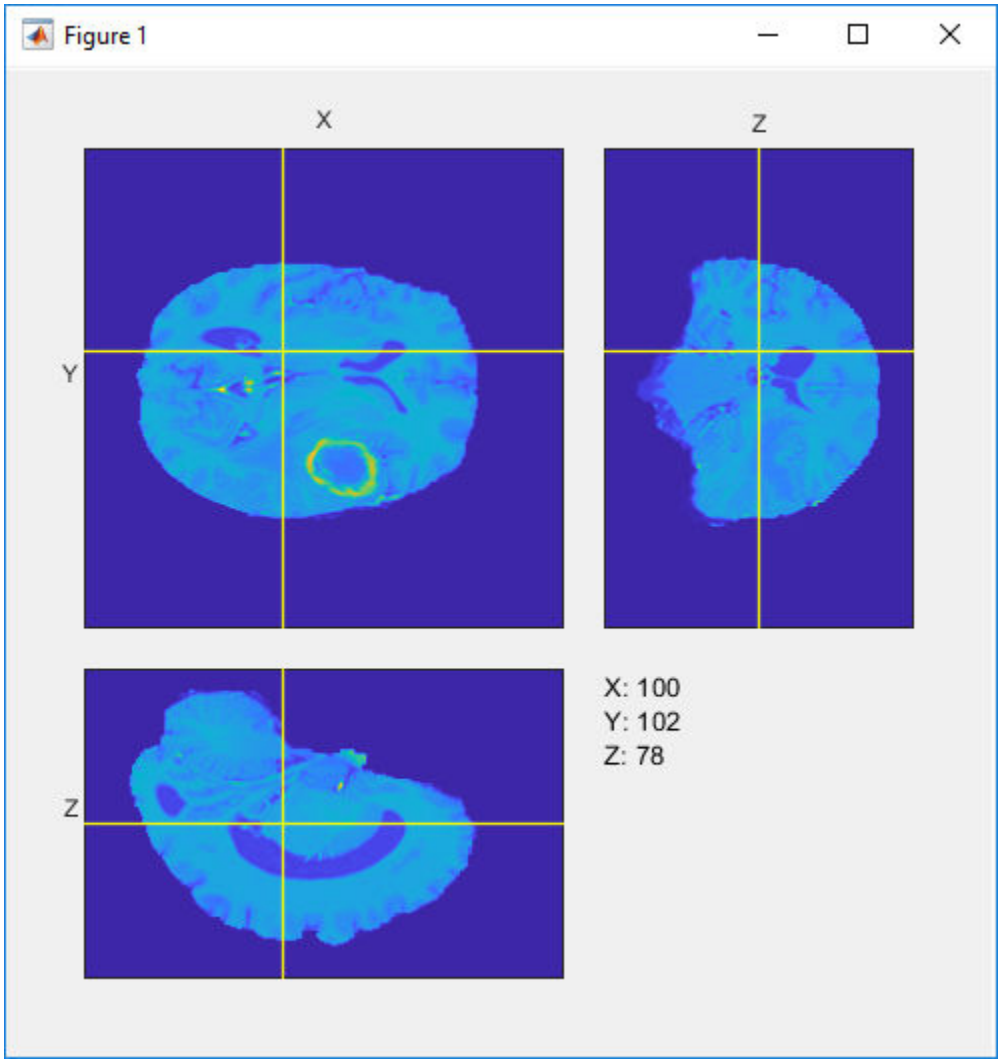
```
load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled','images','vol_001.mat'));
```

Create a custom colormap for viewing slices.

```
cmap = parula(256);
```

View the image stack in the Orthoslice Viewer.

```
os = orthosliceViewer(vol,'Colormap',cmap);
```



Set up listeners for the two Orthoslice Viewer crosshair moving events. When you move the crosshair, the Orthoslice Viewer sends notifications of these events and executes the callback function you specify.

```
addlistener(os,'CrosshairMoving',@allevents);
addlistener(os,'CrosshairMoved',@allevents);
```

The `allevents` callback function displays the name of each event with the previous position and the current position of the crosshair.

```
function allevents(src,evt)
evname = evt.EventName;
    switch(evname)
        case{'CrosshairMoved'}
            disp(['Crosshair moved previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['Crosshair moved current position: ' mat2str(evt.CurrentPosition)]);
        case{'CrosshairMoving'}
            disp(['Crosshair moving previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['Crosshair moving current position: ' mat2str(evt.CurrentPosition)]);
    end
 end
```

See Also

`Crosshair` | `orthosliceViewer`

Introduced in R2019b

images.stack.browser.SliderMovingEventData class

Package: `images.stack.browser`

Event data passed when slider in Slice Viewer is moving

Description

The `images.stack.browser.SliderMovingEventData` class is the class passed to listeners when the slider in a `sliceViewer` object moves. The `sliceViewer` object triggers an event using the `notify` handle class method. MATLAB assigns values to the properties of an `images.stack.browser.SliderMovingEventData` class and passes that class to the listener callback function (the event handler). Programmatic positioning of the slider does not trigger this event.

The `images.stack.browser.SliderMovingEventData` class is a `handle` class.

Creation

The `notify` handle class method creates an `images.stack.browser.SliderMovingEventData` object when called to trigger an event.

Properties

`Source` — Event source
object

Event source, specified as a handle to the object that triggered the event.

`EventName` — Name of event
character vector

Name of the event, specified as a character vector.

`CurrentValue` — Image frame indicated by slider position
numeric scalar

Image frame indicated by slider position, specified as a numeric scalar.

Examples

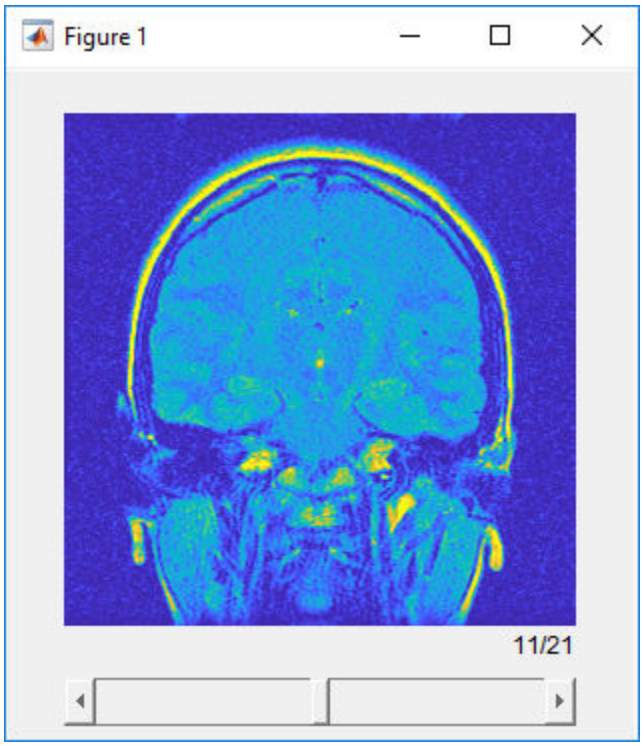
Set Up Listener for Slice Viewer Slider Events

Load a stack of images into the workspace.

```
load mristack
```

View the data in the slice viewer, specifying a custom colormap for viewing the slices. The slice viewer opens the stack of images and displays the one in the middle. Use the slider to view a different slice.

```
cmap = parula(256);
s = sliceViewer(mristack,'Colormap',cmap);
```



Set up listeners for the two `sliceViewer` object slider events: when the slider is moving and when the slider has been moved. When you move the slider, the slice viewer sends notifications of these events and executes the specified callback function.

```
addlistener(s,'SliderValueChanging',@allevents);
addlistener(s,'SliderValueChanged',@allevents);
```

Use this `allevents` callback function to display the name of each event and the current position of the slider.

```
function allevents(src,evt)
    evname = evt.EventName;
    switch(evname)
        case{'SliderValueChanging'}
            disp(['Slider value changing event: ' mat2str(evt.CurrentValue)]);
        case{'SliderValueChanged'}
            disp(['Slider value changed event: ' mat2str(evt.CurrentValue)]);
    end
end
```

See Also

`sliceViewer`

Introduced in R2019b

imapplymatrix

Linear combination of color channels

Syntax

```
Y = imapplymatrix(M,X)
Y = imapplymatrix(M,X,C)
Y = imapplymatrix( ___ ,output_type)
```

Description

`Y = imapplymatrix(M,X)` computes the linear combination of the rows of `M` with the color channels of `X`.

`Y = imapplymatrix(M,X,C)` computes the linear combination of the rows of `M` with the color channels of `X`, adding the corresponding constant value `C` to each combination.

`Y = imapplymatrix( ___ ,output_type)` returns the result of the linear combination in an array of type `output_type`.

Examples

Compute Linear Combination of Color Channels

This example shows how to create a grayscale image by computing the linear combination of three colors channels.

Read a truecolor image into the workspace.

```
RGB = imread('peppers.png');
```

Create a coefficient matrix

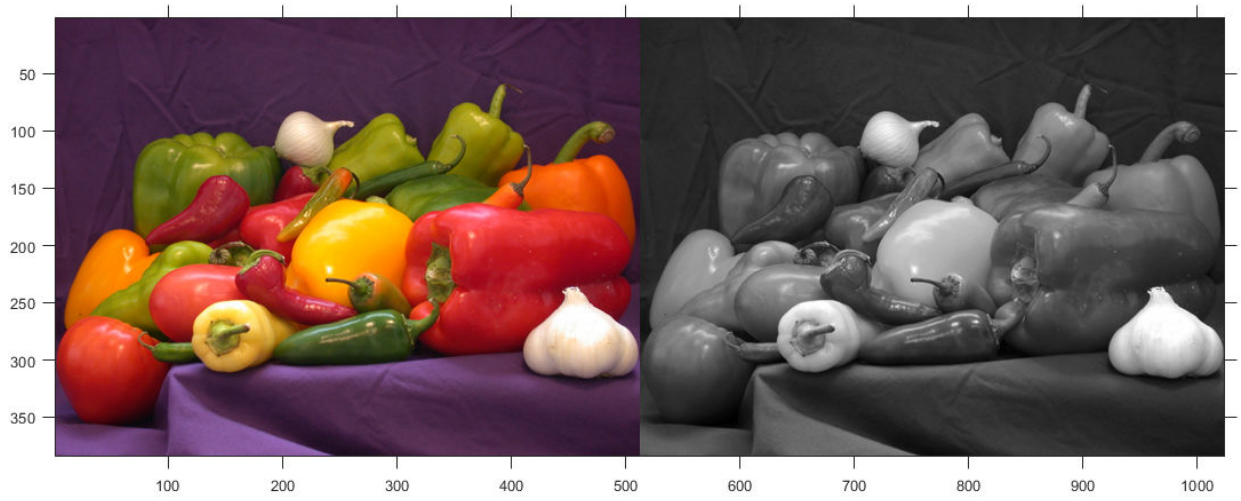
```
M = [0.30, 0.59, 0.11];
```

Compute the linear combination of the RGB channels using the coefficient matrix.

```
gray = imapplymatrix(M, RGB);
```

Display the original image and the grayscale conversion.

```
imshowpair(RGB,gray,'montage')
```



Input Arguments

`M` — Weighting coefficients for each color channel

q-by-*p* numeric array

Weighting coefficients for each color channel, specified as a *q*-by-*p* numeric array. *p* is the length of the third dimension of `X`. In other words, *p*`=size(X,3)`. *q* is in the range [1,*p*].

`X` — Input image

m-by-*n*-by-*p* numeric array

Input image, specified as an *m*-by-*n*-by-*p* numeric array.

`C` — Constant to add to each channel

q-element numeric vector

Constant to add to each channel during the linear combination, specified as *q*-element numeric vector, where *q* is the number of rows in `M`.

Data Types: `double`

`output_type` — Output data type

`'double'` | `'single'` | `'uint8'` | `'uint16'` | `'uint32'` | `'int8'` | `'int16'` | `'int32'`

Output data type, specified as one of the following: `'double'`, `'single'`, `'uint8'`, `'uint16'`, `'uint32'`, `'int8'`, `'int16'`, or `'int32'`.

Data Types: `char` | `string`

Output Arguments

`Y` — Output image

numeric array

Output image comprised of the linear combination of the rows of `M` with the color channels of `X`, returned as a numeric array. If `output_type` is not specified, the data type of `Y` is the same as the data type of `X`.

See Also

`imlincomb` | `immultiply`

Introduced in R2011b

imattributes

Information about image attributes

Syntax

```
attrs = imattributes
attrs = imattributes(img)
attrs = imattributes(imgmodel)
```

Description

`attrs = imattributes` returns information about an image in the current figure. If the current figure does not contain an image, then `imattributes` returns an empty array.

`attrs = imattributes(img)` returns information about the image specified by image object `img`. The `imattributes` function gets the image attributes by querying the image object's `CData`.

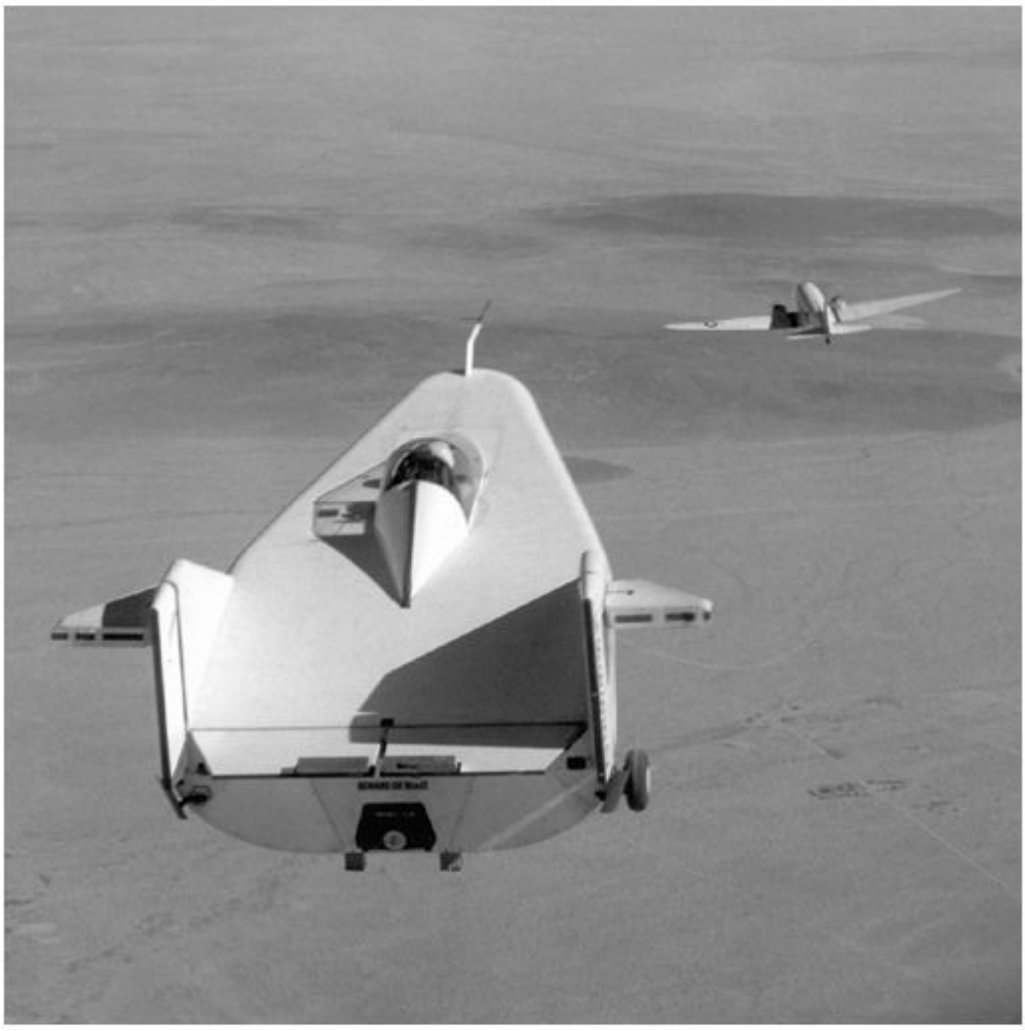
`attrs = imattributes(imgmodel)` returns information about the image represented by the image model object, `imgmodel`.

Examples

Retrieve Attributes of Grayscale Image

Read a grayscale image into the workspace.

```
h = imshow('liftingbody.png');
```



Get the image attributes.

```
attrs = imattributes(h)
```

```
attrs = 6x2 cell
    {'Width (columns)'  }    {'512'      }
    {'Height (rows)'    }    {'512'      }
    {'Class'            }    {'uint8'    }
    {'Image type'       }    {'intensity'}
    {'Minimum intensity'}    {'0'        }
    {'Maximum intensity'}    {'255'      }
```

Retrieve Attributes of Truecolor Image

```
h = imshow('gantrycrane.png');
```

```
im = imagemodel(h);
attrs = imattributes(im)
```

```
attrs = 4x2 cell
    {'Width (columns)'}    {'400'      }
    {'Height (rows)'  }    {'264'      }
    {'Class'          }    {'uint8'    }
    {'Image type'     }    {'truecolor'}
```

Input Arguments

`img` — Image

`image` object

Image, specified as an `image` object.

`imgmodel` — Image model

`imagemodel` object

Image model, specified as an `imagemodel` object.

Output Arguments

`attrs` — Image attributes

cell array of character vectors

Image attributes, returned as a cell array of character vectors. The cell array has size 4-by-2 for binary and truecolor images and size 6-by-2 for grayscale (intensity) and indexed images. The first column of the cell array contains the name of the attribute. The second column contains the value of the attribute.

The table lists these attributes in the order they appear in the cell array.

Attribute Name	Value
`'Width (columns)'`	Number of columns in the image.
`'Height (rows)'`	Number of rows in the image.
`'Class'`	Data type used by the image, such as `uint8`.
`'Image type'`	One of the image types identified by the Image Processing Toolbox software: `'intensity'`, `'truecolor'`, `'binary'`, or `'indexed'`.
`'Minimum intensity'`	• For intensity images, this value represents the lowest intensity value of any pixel. • For indexed images, this value represents the lowest index value into a colormap. This attribute is not included for `'binary'` or `'truecolor'` images.
`'Maximum intensity'`	• For intensity images, this value represents the highest intensity value of any pixel. • For indexed images, this value represents the highest index value into a colormap. This attribute is not included for `'binary'` or `'truecolor'` images.

See Also

`imagemodel`

Introduced before R2006a

imbilatfilt

Bilateral filtering of images with Gaussian kernels

Syntax

```
J = imbilatfilt(I)
J = imbilatfilt(I,degreeOfSmoothing)
J = imbilatfilt(I,degreeOfSmoothing,spatialSigma)
J = imbilatfilt( ___ ,Name,Value)
```

Description

`J = imbilatfilt(I)` applies an edge-preserving Gaussian bilateral filter to the grayscale or RGB image, `I`.

`J = imbilatfilt(I,degreeOfSmoothing)` specifies the amount of smoothing. When `degreeOfSmoothing` is a small value, `imbilatfilt` smooths neighborhoods with small variance (uniform areas) but does not smooth neighborhoods with large variance, such as strong edges. When the value of `degreeOfSmoothing` increases, `imbilatfilt` smooths both uniform areas and neighborhoods with larger variance.

`J = imbilatfilt(I,degreeOfSmoothing,spatialSigma)` also specifies the standard deviation, `spatialSigma`, of the spatial Gaussian smoothing kernel. Larger values of `spatialSigma` increase the contribution of more distant neighboring pixels, effectively increasing the neighborhood size.

`J = imbilatfilt( ___ ,Name,Value)` uses name-value pairs to change the behavior of the bilateral filter.

Examples

Smooth Grayscale Image Using Bilateral Filtering

Read and display a grayscale image. Observe the horizontal striation artifact in the sky region.

```
I = imread('cameraman.tif');
imshow(I)
```

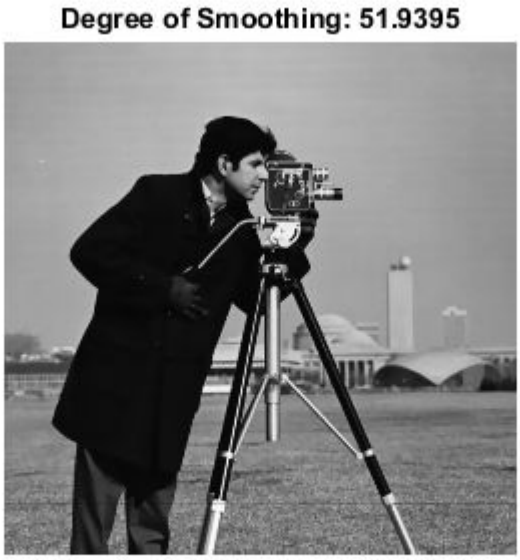
Inspect a patch of the image from the sky region. Compute the variance of the patch, which approximates the variance of the noise.

```
patch = imcrop(I,[170, 35, 50 50]);
imshow(patch)
```

```
patchVar = std2(patch)^2;
```

Filter the image using bilateral filtering. Set the degree of smoothing to be larger than the variance of the noise.

```
DoS = 2*patchVar;
J = imbilatfilt(I,DoS);
imshow(J)
title(['Degree of Smoothing: ',num2str(DoS)])
```



The striation artifact is reduced, but not eliminated. To improve the smoothing, increase the value of `spatialSigma` to 2 so that distant neighboring pixels contribute more to the Gaussian smoothing kernel. This effectively increases the spatial extent of the bilateral filter.

```
K = imbilatfilt(I,DoS,2);
imshow(K)
title(['Degree of Smoothing: ',num2str(DoS),', Spatial Sigma: 2'])
```

The striation artifact in the sky is successfully removed. The sharpness of strong edges such as the silhouette of the man, and textured regions such as the grass in the foreground of the image, have been preserved.

Smooth Color Image Using Bilateral Filtering

Read an RGB image.

```
imRGB = imread("coloredChips.png");
imshow(imRGB)
```

Convert the image to the L*a*b* color space, so that the bilateral filter smooths perceptually similar colors.

```
imLAB = rgb2lab(imRGB);
```

Extract a patch that contains no sharp edges. Compute the variance in the Euclidean distance from the origin, in the L*a*b* color space.

```
patch = imcrop(imLAB,[34,71,60,55]);
patchSq = patch.^2;
```

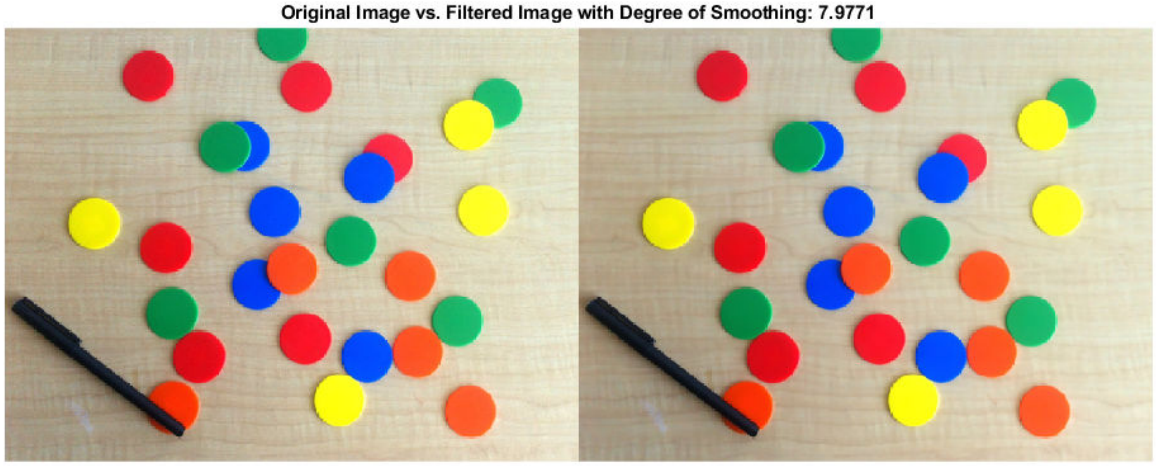
```
edist = sqrt(sum(patchSq,3));
patchVar = std2(edist).^2;
```

Filter the image in the L*a*b* color space using bilateral filtering. Set the `DegreeOfSmoothing` value to be higher than the variance of the patch.

```
DoS = 2*patchVar;
smoothedLAB = imbilatfilt(imLAB,DoS);
```

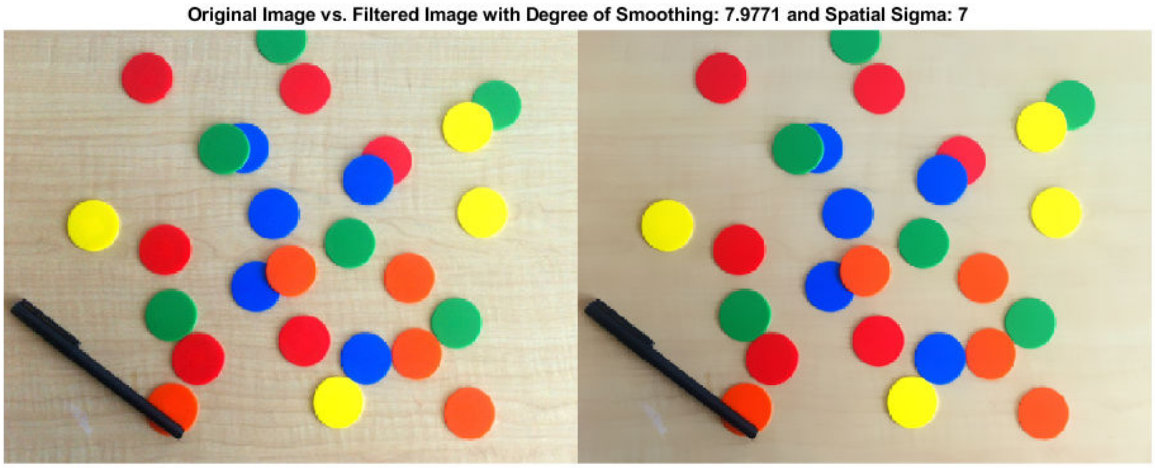
Convert the image back to the RGB color space, and display the smoothed image.

```
smoothedRBG = lab2rgb(smoothedLAB,"Out","uint8");
montage({imRGB,smoothedRBG})
title("Original Image vs. Filtered Image with Degree of Smoothing: " + num2str(DoS))
```



The colors of the chips and black pen appear more uniform, but the horizontal grains in the table are still visible. Increase the spatial extent of the filter so that the effective neighborhood of the filter spans the space between the horizontal grains (this distance is approximately seven pixels). Also increase the `DegreeOfSmoothing` to smooth these regions more aggressively.

```
DoS2 = 4*patchVar;
smoothedLAB2 = imbilatfilt(imLAB,DoS2,7);
smoothedRBG2 = lab2rgb(smoothedLAB2,"Out","uint8");
montage({imRGB,smoothedRBG2})
title("Original Image vs. Filtered Image with Degree of Smoothing: " + num2str(DoS) + " and Spati
```



The color of the wooden table is more uniform with the larger neighborhood and larger degree of smoothing. The edge sharpness of the chips and pen is preserved.

Input Arguments

`I` — Image to filter

2-D grayscale image | 2-D color image

Image to filter, specified as a 2-D grayscale image of size *m*-by-*n* or a 2-D color image of size *m*-by-*n*-by-3.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`degreeOfSmoothing` — Degree of smoothing

positive number

Degree of smoothing, specified as a positive number. The default value of `degreeOfSmoothing` depends on the data type of image `I`, and is calculated as `0.01*diff(getrangefromclass(I)).^2`. For example, the default degree of smoothing is `650.25` for images of data type `uint8`, and the default is `0.01` for images of data type `double` with pixel values in the range [0, 1].

`spatialSigma` — Standard deviation of spatial Gaussian smoothing kernel

1 (default) | positive number

Standard deviation of spatial Gaussian smoothing kernel, specified as a positive number.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `imbilatfilt(I,NeighborhoodSize=7)` performs bilateral filtering on image `I` using a 7-by-7 pixel neighborhood

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `imbilatfilt(I,"NeighborhoodSize",7)` performs bilateral filtering on image `I` using a 7-by-7 pixel neighborhood

`NeighborhoodSize` — Neighborhood size
odd-valued positive integer

Neighborhood size, specified as an odd-valued positive integer. By default, the neighborhood size is `2*ceil(2*SpatialSigma)+1` pixels

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Padding` — Padding
`"replicate"` (default) | `"symmetric"` | numeric scalar

Padding, specified as one of these values.

Value	Description
`"replicate"`	Input array values outside the bounds of the array are assumed to equal the nearest array border value.
`"symmetric"`	Input array values outside the bounds of the array are computed by mirror-reflecting the array across the array border.
numeric scalar, *x*	Input image values outside the bounds of the image are assigned the value *x*.

Example: `Padding="symmetric"`

Example: `Padding=128`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

Output Arguments

`J` — Filtered image
numeric array

Filtered image, returned as a numeric array of the same size and data type as the input image, `I`.

Tips

- The value of `degreeOfSmoothing` corresponds to the variance of the Range Gaussian kernel of the bilateral filter [1]. The Range Gaussian is applied on the Euclidean distance of a pixel value from the values of its neighbors.
- To smooth perceptually close colors of an RGB image, convert the image to the CIE L*a*b* space using `rgb2lab` before applying the bilateral filter. To view the results, convert the filtered image to RGB using `lab2rgb`.
- Increasing `spatialSigma` increases `NeighborhoodSize`, which increases the filter execution time. You can specify a smaller `NeighborhoodSize` to trade accuracy for faster execution time.

References

[1] Tomasi, C., and R. Manduchi. "Bilateral Filtering for Gray and Color Images". *Proceedings of the 1998 IEEE® International Conference on Computer Vision*. Bombay, India. Jan 1998, pp. 836–846.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imbilatfilt` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- When generating code, all character vector input arguments must be compile-time constants.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

When generating CUDA® code, all character vector input arguments must be compile-time constants.

See Also

`imdiffusefilt` | `imgaussfilt` | `imguidedfilter` | `imfilter` | `nlfilter` | `locallapfilt` | `imnlmfilt`

Introduced in R2018a

imbinarize

Binarize 2-D grayscale image or 3-D volume by thresholding

Syntax

```
BW = imbinarize(I)
BW = imbinarize(I,method)
BW = imbinarize(I,T)
BW = imbinarize(I,'adaptive',Name,Value)
```

Description

`BW = imbinarize(I)` creates a binary image from 2-D or 3-D grayscale image `I` by replacing all values above a globally determined threshold with `1`s and setting all other values to `0`s. By default, `imbinarize` uses Otsu's method, which chooses the threshold value to minimize the intraclass variance of the thresholded black and white pixels [1]. `imbinarize` uses a 256-bin image histogram to compute Otsu's threshold. To use a different histogram, see `otsuthresh`.

`BW = imbinarize(I,method)` creates a binary image from image `I` using the thresholding method specified by `method`: `'global'` or `'adaptive'`.

`BW = imbinarize(I,T)` creates a binary image from image `I` using the threshold value `T`. `T` can be a global image threshold, specified as a scalar luminance value, or a locally adaptive threshold, specified as a matrix of luminance values.

`BW = imbinarize(I,'adaptive',Name,Value)` creates a binary image from image `I` using name-value pairs to control aspects of adaptive thresholding.

Examples

Binarize Image Using Global Threshold

Read grayscale image into the workspace.

```
I = imread('coins.png');
```

Convert the image into a binary image.

```
BW = imbinarize(I);
```

Display the original image next to the binary version.

```
figure
imshowpair(I,BW,'montage')
```

Binarize Image Using Locally Adaptive Thresholding

Read grayscale image into workspace.

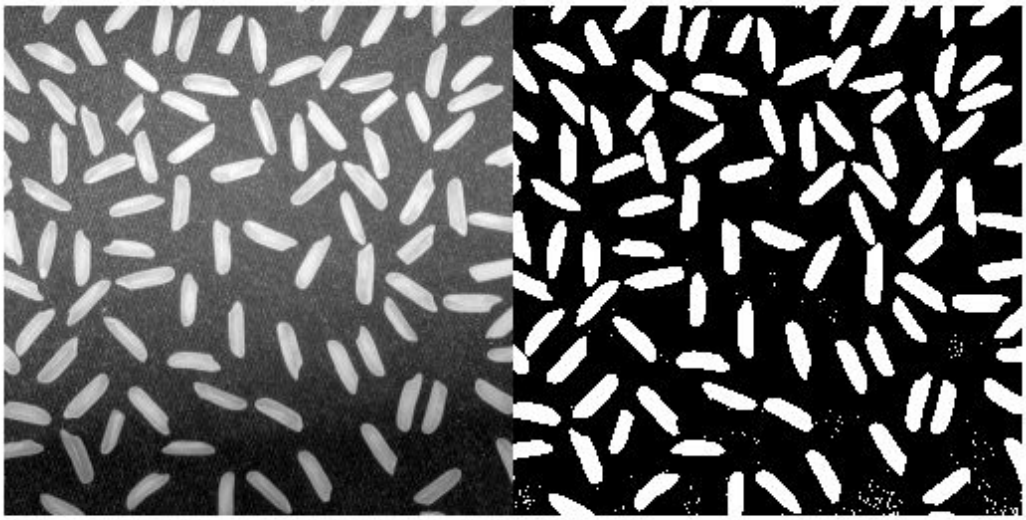
```
I = imread('rice.png');
```

Convert grayscale image to binary image.

```
BW = imbinarize(I, 'adaptive');
```

Display original image along side binary version.

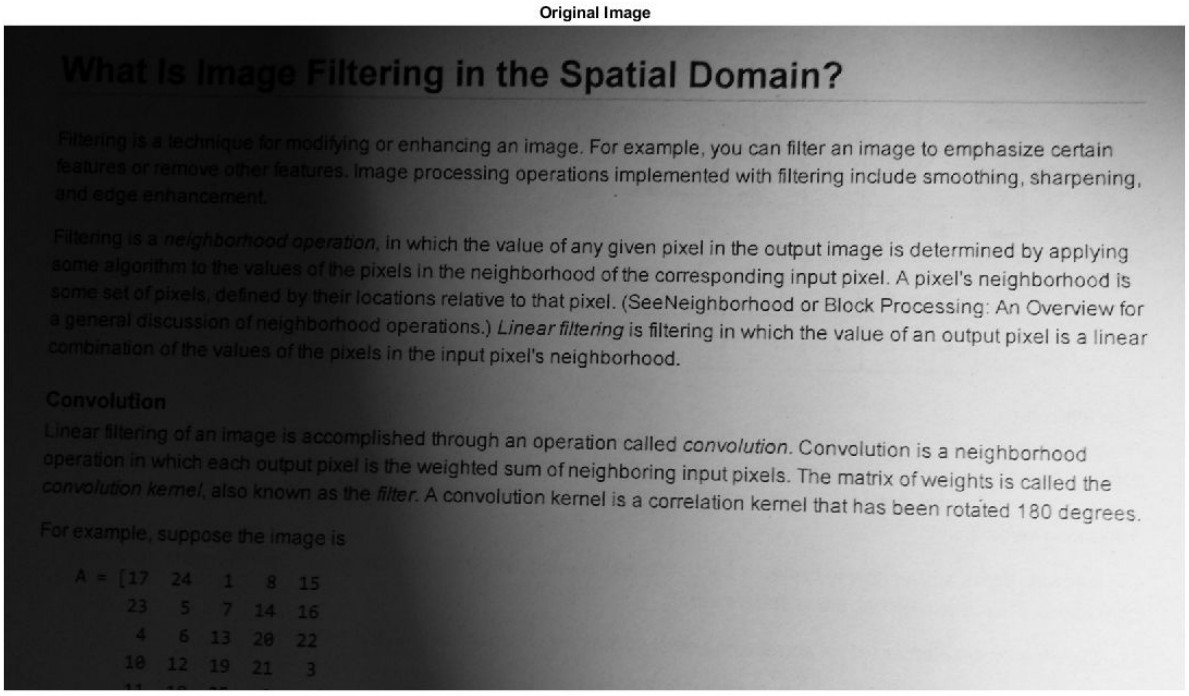
```
figure
imshowpair(I,BW,'montage')
```



Binarize Images with Darker Foreground Than Background

Read a grayscale image into the workspace and display it.

```
I = imread('printedtext.png');
figure
imshow(I)
title('Original Image')
```

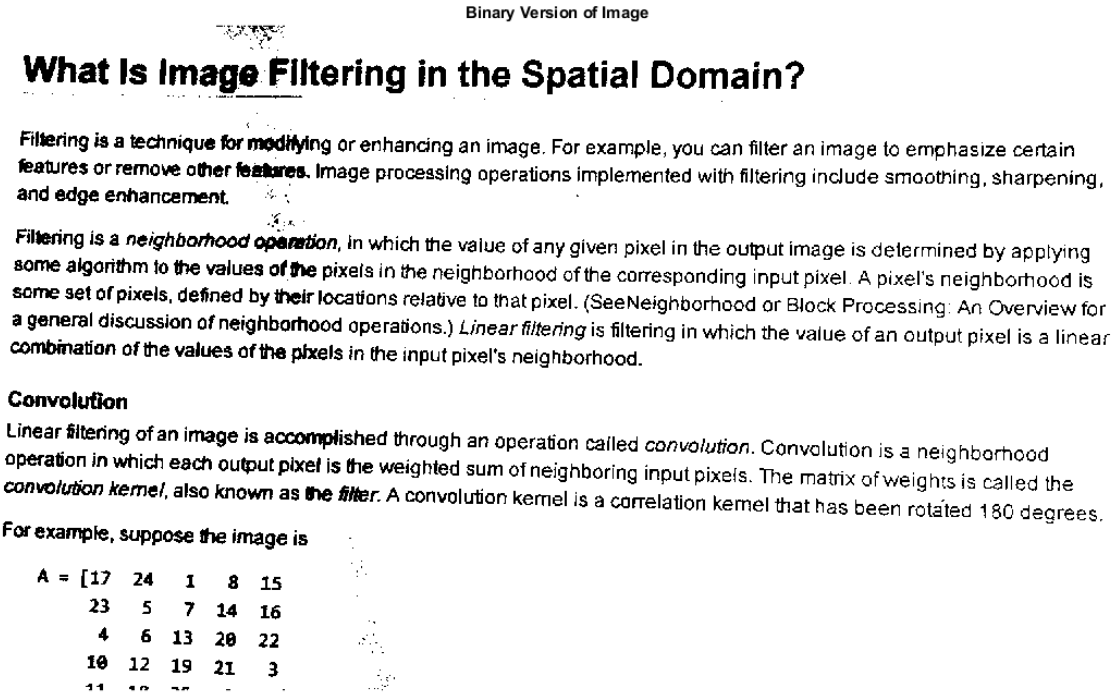


Convert the image to a binary image using adaptive thresholding. Use the `ForegroundPolarity` parameter to indicate that the foreground is darker than the background.

```
BW = imbinarize(I,'adaptive','ForegroundPolarity','dark','Sensitivity',0.4);
```

Display the binary version of the image.

```
figure
imshow(BW)
title('Binary Version of Image')
```



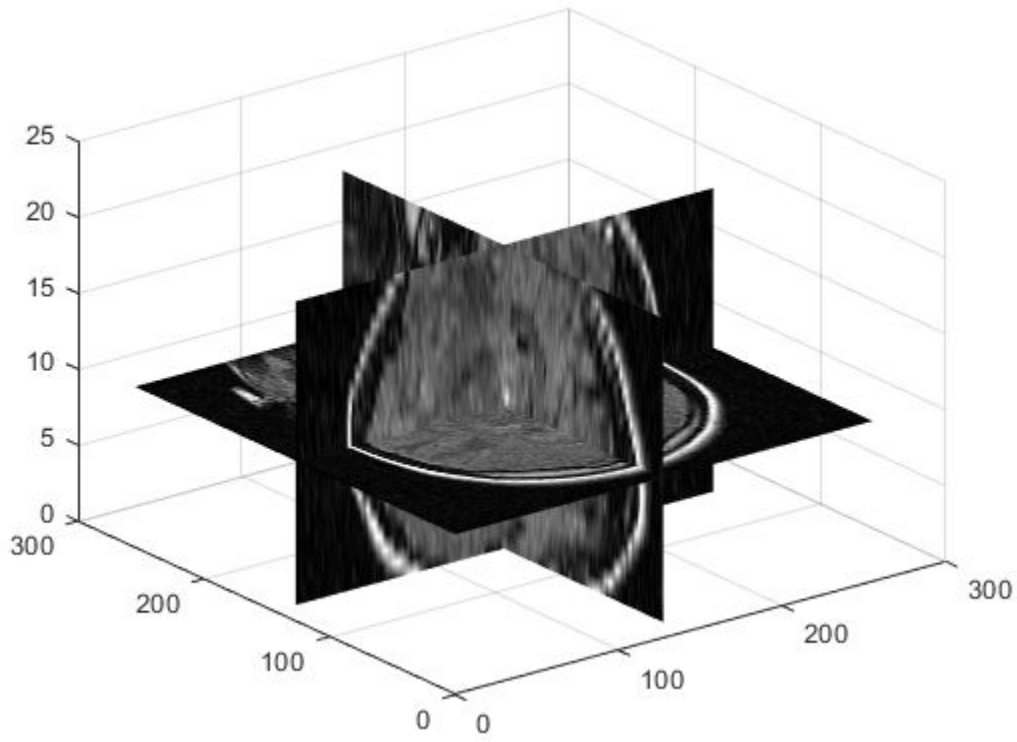
Binarize 3-D Volume Using Global Thresholding

Load 3-D grayscale intensity data into the workspace.

```
load mristack;
V = mristack;
```

View the 3-D volume.

```
figure
slice(double(V),size(V,2)/2,size(V,1)/2,size(V,3)/2)
colormap gray
shading interp
```

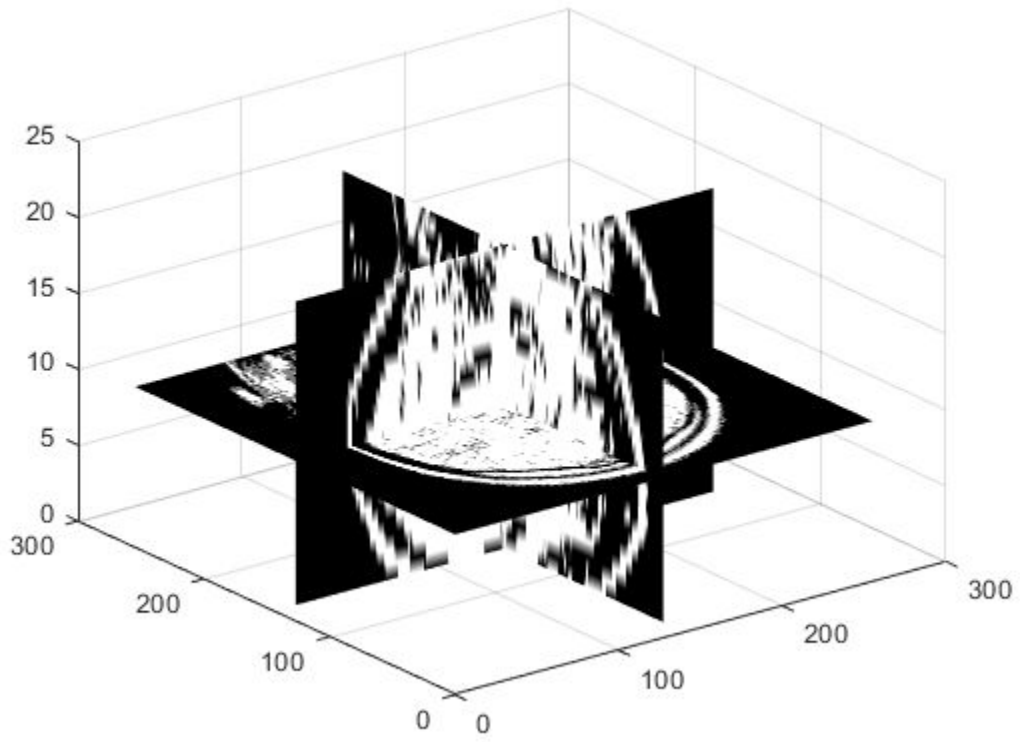


Convert the intensity volume into a 3-D binary volume.

```
J = imbinarize(V);
```

View the 3-D binary volume.

```
figure
slice(double(J),size(J,2)/2,size(J,1)/2,size(J,3)/2)
colormap gray
shading interp
```



Input Arguments

`I` — Input image

2-D grayscale image | 3-D grayscale volume

Input image, specified as a 2-D grayscale image or a 3-D grayscale volume. `imbinarize` expects pixel values of data type `double` and `single` to be in the range [0, 1]. You can use the `rescale` function to adjust pixel values to the expected range.

Note `imbinarize` interprets an RGB image as a volumetric grayscale image and does not binarize each channel separately. To produce a binary image from an RGB image, first convert the image to a grayscale image using `rgb2gray`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`method` — Method used to binarize image

`'global'` (default) | `'adaptive'`

Method used to binarize image, specified as one of the following values.

Values	Meaning
`'global'`	Calculate global image threshold using Otsu's method. See `graythresh` for more information about Otsu's method.
`'adaptive'`	Calculate locally adaptive image threshold chosen using local first-order image statistics around each pixel. See `adaptthresh` for details. If the image contains `Inf`s or `NaN`s, the behavior of `imbinarize` for the `'adaptive'` method is undefined. Propagation of `Inf`s or `NaN`s might not be localized to the neighborhood around `Inf` and `NaN` pixels.

Data Types: `char` | `string`

T — Threshold
numeric scalar | numeric array

Threshold luminance value, specified as a numeric scalar or numeric array with values in the range [0, 1].

- If `T` is a numeric scalar, then `imbinarize` interprets `T` as a global image threshold. Use `graythresh` or `otsuthresh` to compute a global image threshold.
- If `T` is a numeric array, then `imbinarize` interprets `T` as a locally adaptive threshold. Use `adaptthresh` to compute a locally adaptive threshold.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `BW = imbinarize(I,'adaptive','Sensitivity',0.4);`

`Sensitivity` — Sensitivity factor for adaptive thresholding
`0.50` (default) | number in the range [0, 1]

Sensitivity factor for adaptive thresholding, specified as the comma-separated pair consisting of `'Sensitivity'` and a number in the range [0, 1]. A high sensitivity value leads to thresholding more pixels as foreground, at the risk of including some background pixels.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ForegroundPolarity` — Determine which pixels are considered foreground pixels
`'bright'` (default) | `'dark'`

Determine which pixels are considered foreground pixels for adaptive thresholding, specified as the comma-separated pair consisting of `'ForegroundPolarity'` and one of the following values.

Value	Meaning
`'bright'`	The foreground is brighter than the background.
`'dark'`	The foreground is darker than the background

Data Types: `char` | `string`

Output Arguments

BW — Output binary image

logical matrix | logical array

Output binary image, returned as a logical matrix or logical array of the same size as `I`.

Data Types: `logical`

Tips

- To produce a binary image from an indexed image, first convert the image to a grayscale image using `ind2gray`.

Algorithms

The `'adaptive'` method binarizes the image using a locally adaptive threshold. `imbinarize` computes a threshold for each pixel using the local mean intensity around the neighborhood of the pixel. This technique is also called Bradley's method [2]. The `'adaptive'` method also uses a neighborhood size of approximately 1/8th of the size of the image (computed as `2*floor(size(I)/16)+1`). To use a different first order local statistic or a different neighborhood size, see `adaptthresh`.

References

[1] Otsu, N., "A Threshold Selection Method from Gray-Level Histograms." *IEEE Transactions on Systems, Man, and Cybernetics*. Vol. 9, No. 1, 1979, pp. 62–66.

[2] Bradley, D., G. Roth, "Adapting Thresholding Using the Integral Image," *Journal of Graphics Tools*. Vol. 12, No. 2, 2007, pp.13–21.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imbinarize` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imbinarize` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, all character vector input arguments must be compile-time constants.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, all character vector input arguments must be compile-time constants.

See Also

graythresh | otsuthresh | adaptthresh | **Image Segmenter**

Introduced in R2016a

imbothat

Bottom-hat filtering

Syntax

```
J = imbothat(I,SE)
J = imbothat(I,nhood)
```

Description

`J = imbothat(I,SE)` performs morphological bottom-hat filtering on the grayscale or binary image `I` using the structuring element `SE`. Bottom-hat filtering computes the morphological closing of the image (using `imclose`) and then subtracts the original image from the result.

`J = imbothat(I,nhood)` bottom-hat filters the image `I`, where `nhood` is a matrix of `0`s and `1`s that specifies the structuring element neighborhood.

This syntax is equivalent to `imbothat(I,strel(nhood))`.

Examples

Enhance Contrast Using Bottom-hat and Top-hat Filtering

Read image into the workspace and display it.

```
I = imread('pout.tif');
imshow(I)
```

Create a disk-shaped structuring element.

```
se = strel('disk',3);
```

Add the original image `I` to the top-hat filtered image, and then subtract the bottom-hat filtered image.

```
J = imsubtract(imadd(I,imtophat(I,se)),imbothat(I,se));
figure
imshow(J)
```

Input Arguments

`I` — Input image

grayscale image | binary image

Input image, specified as a grayscale image or binary image of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`SE` — Structuring element

`strel` object | `offsetstrel` object

Structuring element, specified as a single `strel` object or `offsetstrel` object. If the image `I` is data type `logical`, the structuring element must be flat.

`nhood` — Structuring element neighborhood

matrix of `0`s and `1`s

Structuring element neighborhood, specified as a matrix of `0`s and `1`s.

Example: `[0 1 0; 1 1 1; 0 1 0]`

Output Arguments

`J` — Bottom-hat filtered image

grayscale image | binary image

Bottom-hat filtered image, returned as a grayscale image or binary image. `J` has the same data type as input image `I`.

Tips

- If the dimensionality of the image `I` is greater than the dimensionality of the structuring element, then the `imbothat` function applies the same morphological closing to all planes along the higher dimensions.

  You can use this behavior to perform bottom-hat filtering on RGB images. Specify a 2-D structuring element for RGB images to operate on each color channel separately.
- When you specify a structuring element neighborhood, `imbothat` determines the center element of `nhood` by `floor((size(nhood)+1)/2)`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imbothat` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imbothat` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- The input image `I` must be 2-D or 3-D.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- `gpuArray` input must be of type `uint8` or `logical`.
- The structuring element `SE` must be flat and 2-D.

For more information, see “Image Processing on a GPU”.

See Also

Functions

`imclose` | `imdilate` | `imerode` | `imopen` | `imtophat`

Objects

`strel` | `offsetstrel`

Introduced before R2006a

imboxfilt

2-D box filtering of images

Syntax

```
B = imboxfilt(A)
B = imboxfilt(A,filterSize)
B = imboxfilt( ___ ,Name,Value)
```

Description

`B = imboxfilt(A)` filters image `A` with a 2-D, 3-by-3 box filter. A box filter is also called a mean filter.

`B = imboxfilt(A,filterSize)` filters image `A` with a 2-D box filter with size specified by `filterSize`.

`B = imboxfilt( ___ ,Name,Value)` uses name-value pair arguments to control aspects of the filtering.

Examples

Compute Mean Filter Over Specified Neighborhood

Read image into the workspace.

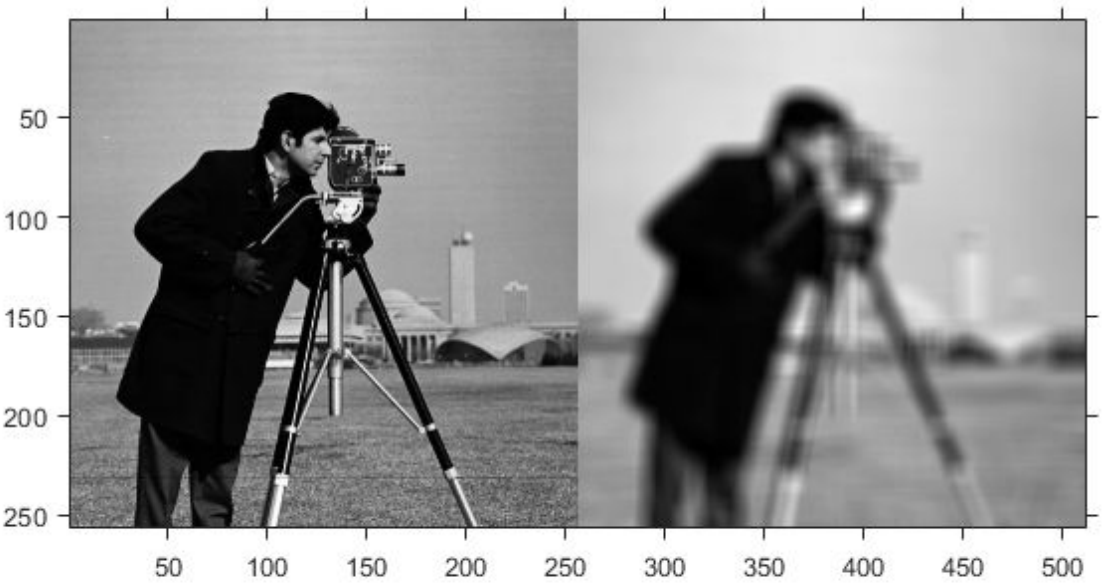
```
A = imread('cameraman.tif');
```

Perform the mean filtering using an 11-by-11 filter.

```
localMean = imboxfilt(A,11);
```

Display the original image and the filtered image, side-by-side.

```
imshowpair(A,localMean,'montage')
```



Compute Local Area Sums Over Specified Neighborhood

Read image into the workspace.

```
A = imread('cameraman.tif');
```

Change the data type of the image to `double` to avoid integer overflow.

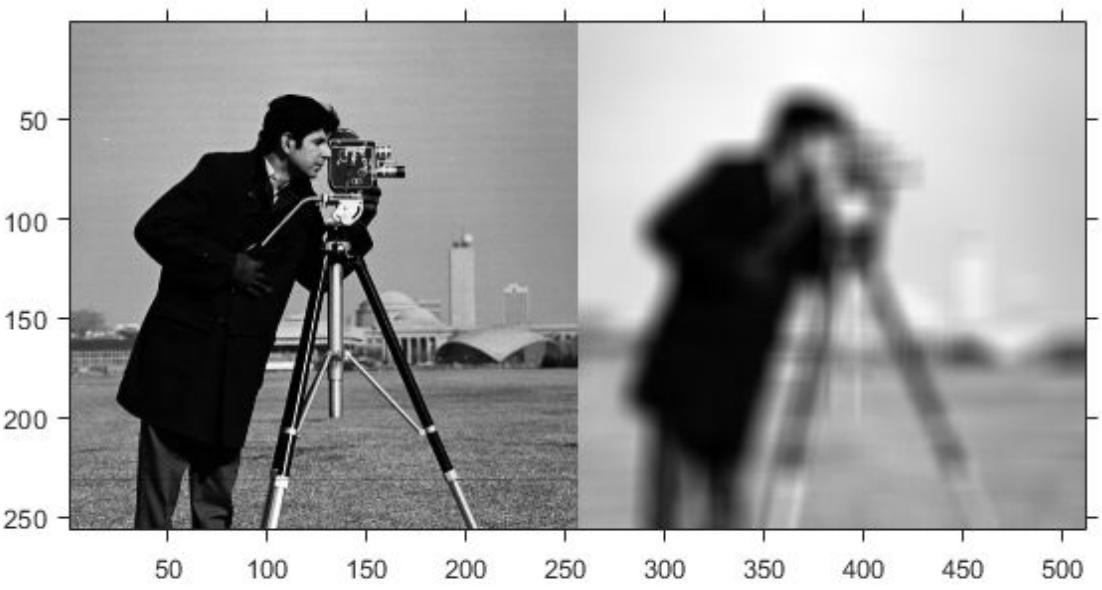
```
A = double(A);
```

Filter image, calculating local area sums, using a 15-by-15 box filter. To calculate local area sums, rather than the mean, set the `NormalizationFactor` parameter to 1.

```
localSums = imboxfilt(A, 15, 'NormalizationFactor',1);
```

Display the original image and the filtered image, side-by-side.

```
imshowpair(A,localSums,'montage')
```



Input Arguments

A — Image to be filtered

numeric array

Image to be filtered, specified as a numeric array of any dimension. If the input image has more than two dimensions (`ndims(I)>2`), such as for an RGB image, then `imboxfilt` performs box filtering of all 2-D planes along the higher dimensions.

If `A` contains `Infs` or `NaNs`, then the behavior of `imboxfilt` is undefined. This can happen when integral image based filtering is used. To restrict the propagation of `Infs` and `NaNs` in the output, consider using `imfilter` instead.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`filterSize` — Size of box filter

3 (default) | positive, odd integer | 2-element vector of positive, odd integers

Size of box filter, specified as a positive odd integer or 2-element vector of positive, odd integers. If `filterSize` is scalar, then the box filter is square.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `B = imboxfilt(A,5,'Padding','circular');`

`Padding` — Padding pattern
`'replicate'` (default) | `'circular'` | `'symmetric'` | numeric scalar

Padding pattern, specified as one of the following values or a numeric scalar. If you specify a scalar value, input image pixels outside the bounds of the image are implicitly assumed to have the scalar value.

Value	Description
`'circular'`	Input image values outside the bounds of the image are computed by implicitly assuming the input image is periodic.
`'replicate'`	Input image values outside the bounds of the image are assumed equal to the nearest image border value.
`'symmetric'`	Input image values outside the bounds of the image are computed by mirror-reflecting the array across the array border.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

`NormalizationFactor` — Normalization factor applied to box filter
`1/filterSize.^2`, if scalar, and `1/prod(filterSize)`, if vector (default) | numeric scalar

Normalization factor applied to box filter, specified as a numeric scalar.

The default `'NormalizationFactor'` has the effect of a mean filter — the pixels in the output image are the local means of the image over the neighborhood determined by `filterSize`. To get local area sums, set `'NormalizationFactor'` to `1`. To avoid overflow in such circumstances, consider using double precision images by converting the input image to class `double`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`B` — Filtered image
numeric array

Filtered image, returned as a numeric array of the same size as the input image `A`.

Algorithms

`imboxfilt` performs filtering using either convolution-based filtering or integral image filtering, using an internal heuristic to determine which filtering approach to use.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imboxfilt` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imboxfilt` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, all character vector input arguments must be compile-time constants.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, all character vector input arguments must be compile-time constants.

See Also

`imboxfilt3` | `imfilter` | `integralBoxFilter`

Introduced in R2015b

imboxfilt3

3-D box filtering of 3-D images

Syntax

```
B = imboxfilt3(A)
B = imboxfilt3(A,filterSize)
B = imboxfilt3( ___ ,Name,Value)
```

Description

`B = imboxfilt3(A)` filters the 3-D image `A` with a 3-D box filter, 3-by-3-by-3 in size.

`B = imboxfilt3(A,filterSize)` filters 3-D image `A` with a 3-D box filter of size `filterSize`.

`B = imboxfilt3( ___ ,Name,Value)` uses name-value pair arguments to control aspects of the filtering.

Examples

Compute Mean Filter in MRI Volume

Load 3-D image data into the workspace.

```
volData = load('mri');
vol = squeeze(volData.D);
```

Filter the image with a 3-D box filter.

```
localMean = imboxfilt3(vol,[5 5 3]);
```

Input Arguments

`A` — Image to be filtered

3-D numeric array

Image to be filtered, specified as a 3-D numeric array.

If `A` contains `Infs` or `NaNs`, the behavior of `imboxfilt3` is undefined. This can happen when integral image based filtering is used. To restrict the propagation of `Infs` and `NaNs` in the output, consider using `imfilter` instead.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`filterSize` — Size of box filter

3 (default) | positive, odd integer | 3-element vector of positive, odd integers

Size of box filter, specified as a positive odd integer or 3-element vector of positive, odd integers. If `filterSize` is scalar, then the filter is a cube.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Example: B = imboxfilt3(A,5,'padding','circular');

Padding — Padding pattern

'replicate' (default) | 'circular' | 'symmetric' | numeric scalar

Padding pattern, specified as one of the following values or a numeric scalar. If you specify a scalar value, input image pixels outside the bounds of the image are implicitly assumed to have the scalar value.

Value	Description
'circular'	Input image values outside the bounds of the image are computed by implicitly assuming the input image is periodic.
'replicate'	Input image values outside the bounds of the image are assumed equal to the nearest image border value.
'symmetric'	Input image values outside the bounds of the image are computed by mirror-reflecting the array across the array border.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | char | string

NormalizationFactor — Normalization factor applied to box filter

1/filterSize.^3, if scalar, and 1/prod(filterSize), if vector (default) | numeric scalar

Normalization factor applied to box filter, specified as a numeric scalar.

The default 'NormalizationFactor' has the effect of a mean filter — the pixels in the output image are the local means of the image. To get local area sums, set 'NormalizationFactor' to 1. To avoid overflow in such circumstances, consider using double precision images by converting the input image to class double.

Example: 'NormalizationFactor',1

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

Output Arguments

B — Filtered image

3-D numeric array

Filtered image, returned as a 3-D numeric array.

Algorithms

`imboxfilt` performs filtering using either convolution-based filtering or integral image filtering, using an internal heuristic to determine which filtering approach to use.

See Also

`imboxfilt` | `imfilter` | `integralBoxFilter3`

Introduced in R2015b

imclearborder

Suppress light structures connected to image border

Syntax

```
J = imclearborder(I)
J = imclearborder(I,conn)
```

Description

`J = imclearborder(I)` suppresses structures in image `I` that are lighter than their surroundings and that are connected to the image border. Use this function to clear the image border. For grayscale images, `imclearborder` tends to reduce the overall intensity level in addition to suppressing border structures. The output image, `J`, is grayscale or binary, depending on the input.

`J = imclearborder(I,conn)` specifies the pixel connectivity, `conn`.

Examples

Impact of Connectivity on Clearing the Border

Create a simple binary image.

```
BW = [0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0
      1     0     0     1     1     1     0     0     0
      0     1     0     1     1     1     0     0     0
      0     0     0     1     1     1     0     0     0
      0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0];
```

Clear pixels on the border of the image using 4-connectivity. Note that `imclearborder` does not clear the pixel at (5,2) because, with 4-connectivity, it is not considered connected to the border pixel at (4,1).

```
BWc1 = imclearborder(BW,4)
```

```
BWc1 = 9×9

     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     1     1     1     0     0     0
     0     1     0     1     1     1     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
```

Now clear pixels on the border of the image using 8-connectivity. `imclearborder` clears the pixel at (5,2) because, with 8-connectivity, it is considered connected to the border pixel (4,1).

```
BWc2 = imclearborder(BW,8)
```

```
BWc2 = 9×9

     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
```

Input Arguments

`I` — Grayscale or binary image

numeric array | logical array

Grayscale or binary image, specified as a numeric or logical array.

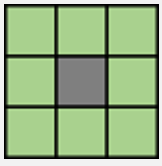
Example: `I = imread('pout.tif');`

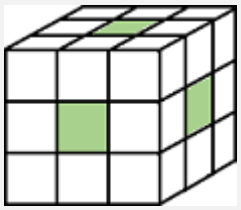
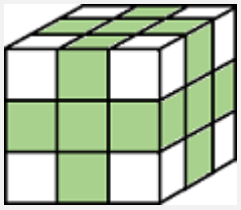
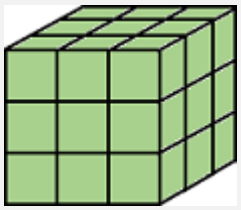
Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`conn` — Pixel connectivity

4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 matrix of `0`s and `1`s

Pixel connectivity, specified as one of the values in this table. The default connectivity is `8` for 2-D images, and `26` for 3-D images.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. The neighborhood of a pixel are the adjacent pixels in the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. The neighborhood of a pixel are the adjacent pixels in the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		

Value	Meaning	
6	Pixels are connected if their faces touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down	 Current pixel is shown in gray.
18	Pixels are connected if their faces or edges touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `imclearborder` uses the default value `conndef(ndims(I),'maximal')`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Note A pixel on the edge of the input image might not be considered to be a border pixel if you specify a nondefault connectivity. For example, if `conn = [0 0 0; 1 1 1; 0 0 0]`, elements on the first and last row are not considered to be border pixels because, according to that connectivity definition, they are not connected to the region outside the image.

Data Types: `double` | `logical`

Output Arguments

J — Processed image

numeric array | logical array

Processed grayscale or binary image, returned as numeric or logical array, depending on the input image you specify.

Algorithms

`imclearborder` uses morphological reconstruction where:

- Mask image is the input image.
- Marker image is zero everywhere except along the border, where it equals the mask image.

References

[1] Soille, P., *Morphological Image Analysis: Principles and Applications*, Springer, 1999, pp. 164-165.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imclearborder` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imclearborder` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- Supports only up to 3-D inputs.
- The optional second input argument, `conn`, must be a compile-time constant.

See Also

`conndef`

Introduced before R2006a

imclose

Morphologically close image

Syntax

```
J = imclose(I,SE)
J = imclose(I,nhood)
```

Description

`J = imclose(I,SE)` performs morphological closing on the grayscale or binary image `I`, using the structuring element `SE`. The morphological close operation is a dilation followed by an erosion, using the same structuring element for both operations.

`J = imclose(I,nhood)` closes the image `I`, where `nhood` is a matrix of `0`s and `1`s that specifies the structuring element neighborhood.

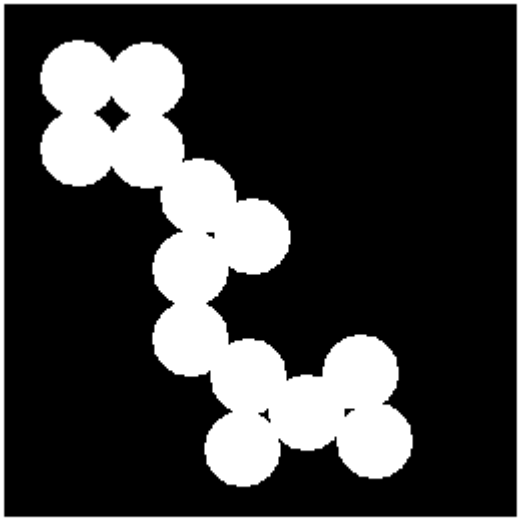
This syntax is equivalent to `imclose(I,strel(nhood))`.

Examples

Use Morphological Closing to Fill Gaps in an Image

Read a binary image into the workspace and display it.

```
originalBW = imread('circles.png');
imshow(originalBW);
```



Create a disk-shaped structuring element. Use a disk structuring element to preserve the circular nature of the object. Specify a radius of 10 pixels so that the largest gap gets filled.

```
se = strel('disk',10);
```

Perform a morphological close operation on the image.

```
closeBW = imclose(originalBW,se);
figure, imshow(closeBW)
```

Input Arguments

`I` — Input image

grayscale image | binary image

Input image, specified as a grayscale image or binary image of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`SE` — Structuring element

`strel` object | `offsetstrel` object

Structuring element, specified as a single `strel` object or `offsetstrel` object. If the image `I` is data type `logical`, then the structuring element must be flat.

`nhood` — Structuring element neighborhood

matrix of `0`s and `1`s

Structuring element neighborhood, specified as a matrix of `0`s and `1`s.

Example: `[0 1 0; 1 1 1; 0 1 0]`

Output Arguments

J — Closed image

grayscale image | binary image

Closed image, returned as a grayscale image or binary image. `J` has the same data type as input image `I`.

Tips

- If the dimensionality of the image `I` is greater than the dimensionality of the structuring element, then the `imclose` function applies the same morphological closing to all planes along the higher dimensions.

  You can use this behavior to perform morphological closing on RGB images. Specify a 2-D structuring element for RGB images to operate on each color channel separately.
- When you specify a structuring element neighborhood, `imclose` determines the center element of `nhood` by `floor((size(nhood)+1)/2)`.

Compatibility Considerations

`imclose` pads image border

Behavior changed in R2017a

Starting in R2017a, `imclose` pads the input image border by half the size of the structuring element. Padding the image removes border artifacts when there are foreground pixels near the boundary of the input image.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imclose` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imclose` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- The input image `I` must be 2-D or 3-D.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input image `I` must be 2-D or 3-D.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- `gpuArray` input must be of type `uint8` or `logical`.
- The structuring element `SE` must be flat and 2-D.

For more information, see “Image Processing on a GPU”.

See Also

Functions
`imopen` | `imdilate` | `imerode`

Objects
`strel` | `offsetstrel`

Introduced before R2006a

imcolordiff

Color difference based on CIE94 or CIE2000 standard

Syntax

```
dE = imcolordiff(I1,I2)
dE = imcolordiff(I1,I2,Name,Value)
```

Description

`dE = imcolordiff(I1,I2)` calculates the color difference between two RGB images or color maps using the CIE94 standard.

`dE = imcolordiff(I1,I2,Name,Value)` specifies additional aspects of the computation, such as the input color space and the CIE standard, using one or more name-value pair arguments.

Examples

Calculate Color Difference of Images Using CIE94 Standard

Read a color image into the workspace.

```
I1 = imread('peppers.png');
imshow(I1)
```

Alter the local color contrast in the image.

```
I2 = localcontrast(I1);
imshow(I2)
```

Calculate the color difference of the images using the default color standard, CIE94.

```
dE = imcolordiff(I1,I2);
```

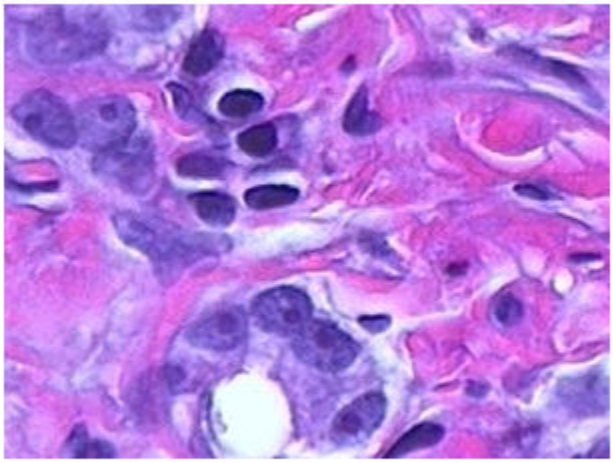
Display the color difference as an image. Scale the display range to use the full range of pixel values in dE.

```
imshow(dE,[])
```

Calculate Color Difference of L*a*b* Images using CIE94 Standard

Read and display an image of tissue stained with hemotoxylin and eosin (H&E).

```
he = imread('hestain.png');
imshow(he)
```



Convert the image to the L*a*b* color space.

```
lab = rgb2lab(he);
```

Make a copy of the image, then increase the signal of the a* channel. Red tones in the image become more saturated while the image overall brightness and the blue tones are unchanged.

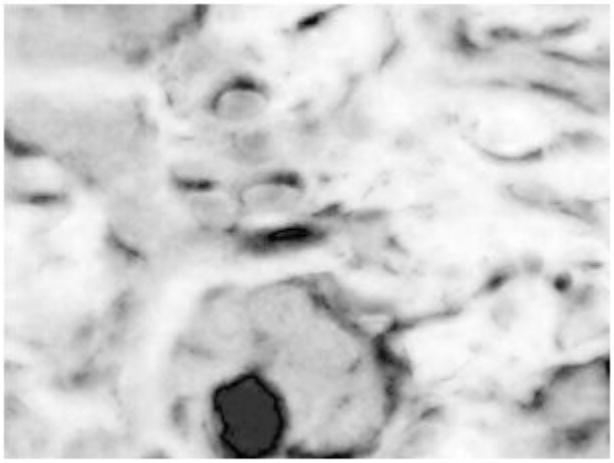
```
lab2 = lab;
scaleFactor = 1.1;
lab2(:,:,2) = scaleFactor*lab(:,:,2);
```

Calculate the color difference of the original and enhanced image in the L*a*b* color space.

```
dE = imcolordiff(lab,lab2,'isInputLab',true);
```

Display the color difference as an image. Scale the display range to match the range of pixel values in `dE`. Bright regions indicate the greatest color difference and correspond with the pink regions of tissue.

```
imshow(dE,[])
```



Calculate Color Difference of Two Colors using CIEDE2000 Standard

Specify two RGB color values.

```
pureRed = uint8([255,0,0]);
darkRed = uint8([255,10,50]);
```

Calculate the color difference of the colors using the CIEDE2000 standard.

```
dE = imcolordiff(pureRed,darkRed,"Standard","CIEDE2000")
```

```
dE = single
    7.4449
```

Calculate Color Difference using Textile Weighting Factors

Read and display an RGB image of fabric.

```
fabric = imread('fabric.png');
imshow(fabric)
```

Simulate a second image of fabric by altering the local color contrast in the image.

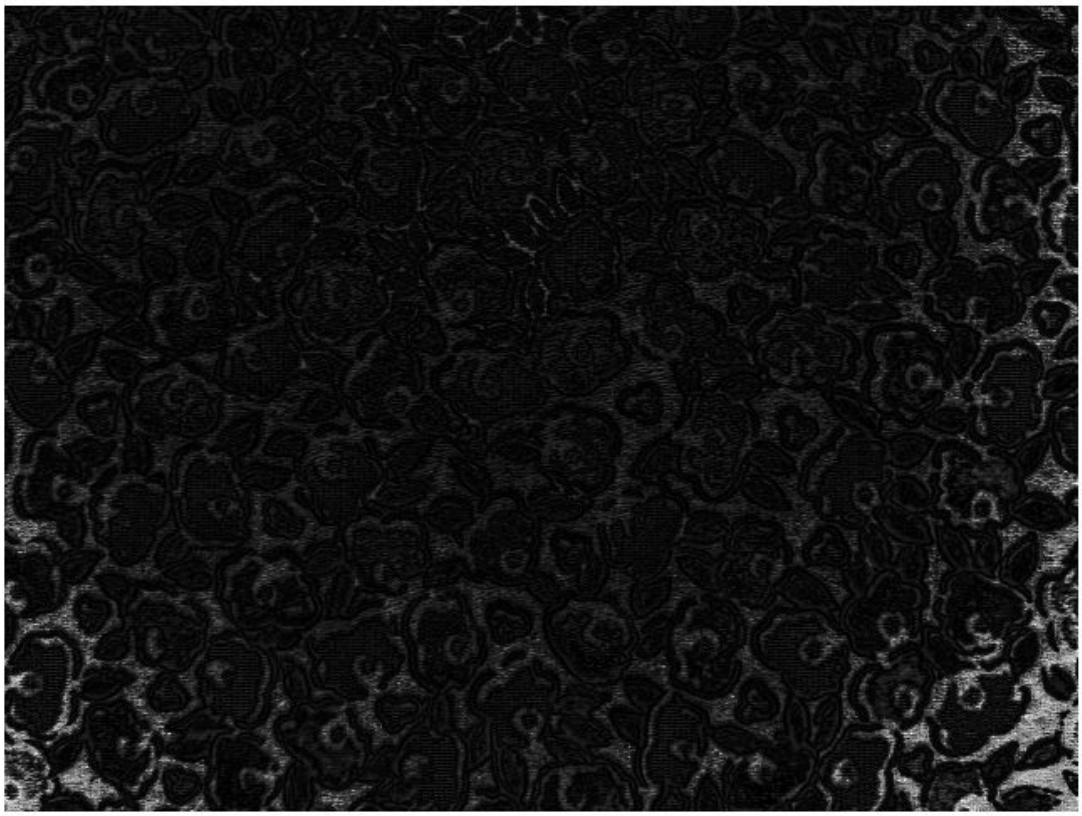
```
fabric2 = localcontrast(fabric);
imshow(fabric2)
```

Calculate the color difference of the two images using the CIEDE2000 standard. Specify a luminance coefficient and K1 and K2 weighting factors appropriate for textiles.

```
dE = imcolordiff(fabric,fabric2,'Standard','CIEDE2000', ...
    'kL',2,'K1',0.048,'K2',0.014);
```

Display the color difference. Scale the display range to the full range of pixel values in `dE`.

```
imshow(dE,[])
```



Input Arguments

`I1` — First set of color data

m-by-*n*-by-3 numeric array | *c*-by-3 numeric matrix

First set of color data, specified as an *m*-by-*n*-by-3 numeric array representing an image or a *c*-by-3 numeric matrix representing a set of *c* colors. `I1` and `I2` must be the same size with values in the same color space.

By default, the `imcolordiff` function interprets the color data as RGB color values. To calculate the color difference in the L*a*b* color space, specify the '`isInputLab`' argument as `true`. L*a*b* color values can be of data type `single` or `double` only.

Data Types: `single` | `double` | `uint8` | `uint16`

`I2` — Second set of color data

m-by-*n*-by-3 numeric array | *c*-by-3 numeric matrix

Second set of color data, specified as an *m*-by-*n*-by-3 numeric array representing an image or a *c*-by-3 numeric matrix representing a set of *c* colors. `I1` and `I2` must be the same size with values in the same color space.

By default, `imcolordiff` interprets the color data as RGB color values. To calculate the difference of colors in the L*a*b* color space, specify the '`isInputLab`' argument as `true`. L*a*b* color values can be of data type `single` or `double` only.

Data Types: `single` | `double` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Standard',"CIEDE2000"` calculates the color difference between two RGB images using the CIEDE2000 standard.

`Standard` — CIE standard

`"CIE94"` (default) | `"CIEDE2000"`

CIE standard used to compute the color difference value, specified as the comma-separated pair consisting of `'Standard'` and one of these values:

Value	Description
`"CIE94"`	The CIE94 standard. This standard improves the perceptual non-uniformities of the CIE76 standard implemented in the `deltaE` function.
`"CIEDE2000"`	The CIEDE2000 standard. This standard further improves the perceptual uniformity through five additional corrections: a hue rotation term, compensation for neutral colors, and compensation for lightness, chroma, and hue.

Data Types: `char` | `string`

`isInputLab` — Color values are in L*a*b* color space

`false` or `0` (default) | `true` or `1`

Color values are in the L*a*b* color space, specified as the comma-separated pair consisting of `'isInputLab'` and a numeric or logical `0` (`false`) or `1` (`true`).

`kL` — Luminance coefficient

`1` (default) | numeric scalar

Luminance coefficient, specified as the comma-separated pair consisting of `'kL'` and a numeric scalar. The luminance coefficient is typically `1` for applications in graphic arts and `2` for applications in textiles.

`K1` — K1 weighting factor

`0.045` (default) | numeric scalar

K1 weighting factor, specified as the comma-separated pair consisting of `'K1'` and a numeric scalar. The K1 weighting factor is typically `0.045` for applications in graphic arts and `0.048` for applications in textiles.

K2 — K2 weighting factor

`0.015` (default) | numeric scalar

K2 weighting factor, specified as the comma-separated pair consisting of `'K2'` and a numeric scalar. The K2 weighting factor is typically `0.015` for applications in graphic arts and `0.014` for applications in textiles.

Output Arguments

dE — Color difference

m-by-*n* matrix | *c*-element column vector

Color difference (delta E), returned as one of the following.

- An *m*-by-*n* matrix when the input color data `I1` and `I2` represent images
- A *c*-element column vector when `I1` and `I2` represent a set of *c* colors

If `I1` or `I2` is of data type `double`, then `dE` is of data type `double`. Otherwise, `dE` is of data type `single`.

Data Types: `single` | `double`

Tips

- To calculate color differences following the CIE76 standard, use the `deltaE` function. This function is faster than the `imcolordiff` function, but less precise.

References

[1] Sharma, Gaurav, Wencheng Wu, and Edul N. Dalal, "The CIEDE2000 Color-Difference Formula: Implementation Notes, Supplementary Test Data, and Mathematical Observations". *Color Research and Application* 30, no. 1 (February 2005): 21–30. https://doi.org/10.1002/col.20070.

See Also

`deltaE` | `colorangle` | `measureColor`

Topics

“Understanding Color Spaces and Color Space Conversion”

Introduced in R2020b

imcolormaptool

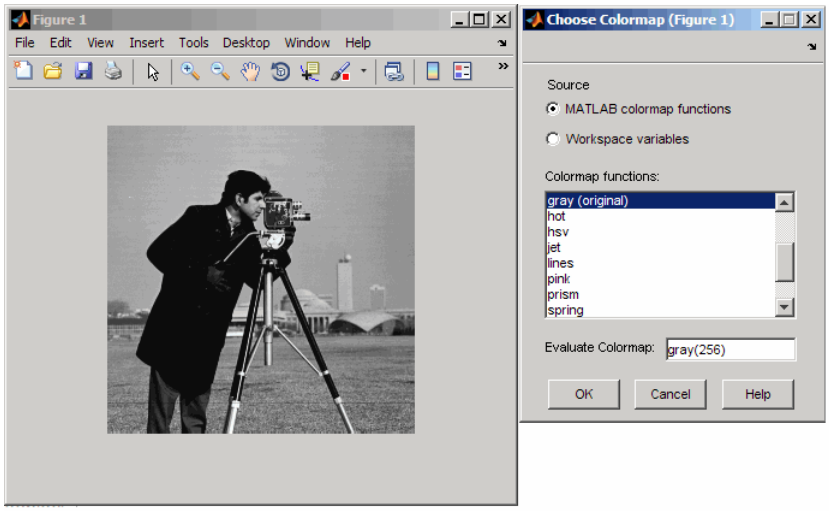
Choose Colormap tool

Syntax

```
imcolormaptool
imcolormaptool(h)
htool = imcolormaptool( ___ )
```

Description

Use the `imcolormaptool` function to create a Choose Colormap tool. The Choose Colormap tool is an interactive colormap selection tool that allows you to change the colormap of a figure by selecting a colormap from a list of MATLAB colormap functions or workspace variables, or by entering a custom MATLAB expression.



Choose Colormap Tool

`imcolormaptool` launches the Choose Colormap tool in a separate figure, which is associated with the current figure.

`imcolormaptool(h)` launches the Choose Colormap tool using `h` as the target figure. `h` must contain either a grayscale or an indexed image.

`htool = imcolormaptool( ___ )` returns a handle to the Choose Colormap tool figure, `htool`.

Examples

Open Choose Colormap Tool

```
h = figure;
imshow("cameraman.tif")
imcolormaptool(h);
```

Input Arguments

h — Graphics object

figure | axes

Graphics object, specified as a figure or axes.

Output Arguments

htool — Handle to Choose Colormap tool figure

handle

Handle to Choose Colormap tool figure, returned as a handle.

See Also

Image Viewer | `colormap` | `imshow`

Introduced in R2009a

imcomplement

Complement image

Syntax

```
J = imcomplement(I)
```

Description

`J = imcomplement(I)` computes the complement on page 1-1426 of the image `I` and returns the result in `J`.

Examples

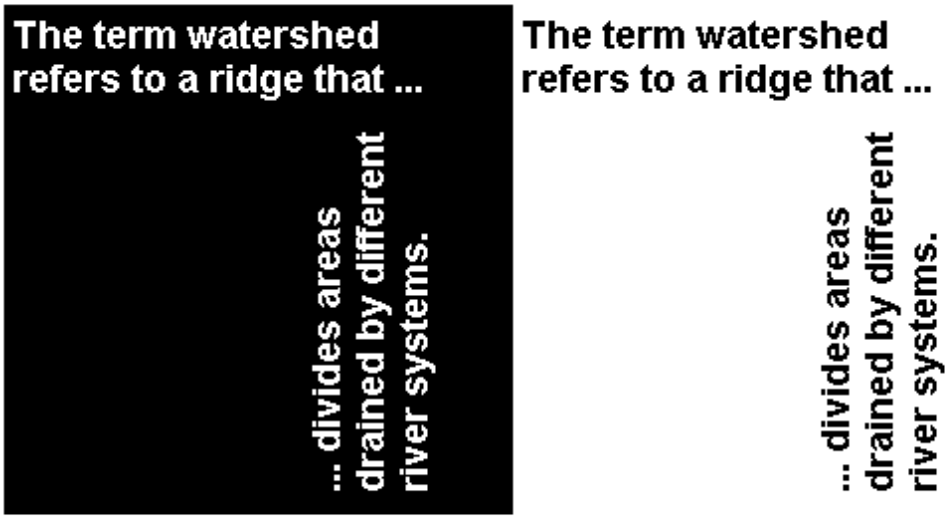
Create the Complement of a uint8 Array

```
X = uint8([ 255 10 75; 44 225 100]);
X2 = imcomplement(X)
```

```
X2 = 2x3 uint8 matrix

     0   245   180
   211    30   155
```

Reverse Black and White in a Binary Image

```
bw = imread('text.png');
bw2 = imcomplement(bw);
imshowpair(bw,bw2,'montage')
```



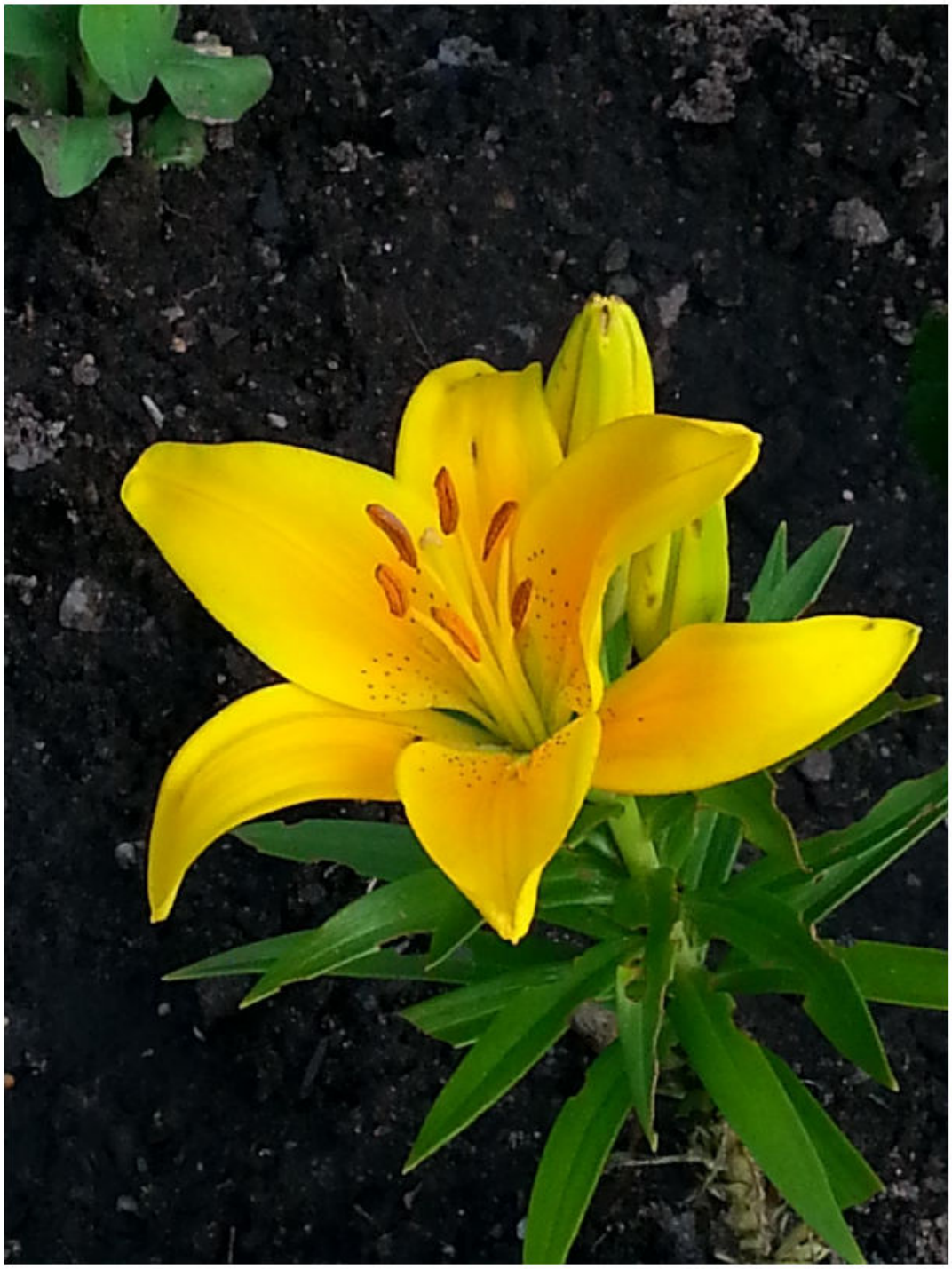
Create the Complement of an Intensity Image

```
I = imread('cameraman.tif');
J = imcomplement(I);
imshowpair(I,J,'montage')
```

Create the Complement of a Color Image

Read a color image into the workspace.

```
rgb = imread('yellowlily.jpg');
imshow(rgb)
```



Display the complement of the image.

```
c = imcomplement(rgb);
imshow(c)
```

Each color channel of the resulting image is the complement of the corresponding color channel in the original image. Regions that were dark, such as dirt, become light. In the original image, the leaves appear green, and petals appear yellow because of a mixture of red and green signals. In the complement image, the leaves appear purple because the red and blue signals are larger than the green signal. The flower petals appear blue because the blue signal is larger than the red and green channels.

Input Arguments

`I` — Input image

RGB image | grayscale image | binary image

Input image, specified as an RGB, grayscale, or binary image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

Output Arguments

`J` — Image complement

RGB image | grayscale image | binary image

Image complement, specified as an RGB, grayscale, or binary image. `J` has the same size and class as the input image, `I`.

More About

Image Complement

In the complement of a binary image, zeros become ones and ones become zeros. Black and white are reversed.

In the complement of a grayscale or color image, each pixel value is subtracted from the maximum pixel value supported by the class (or 1.0 for double-precision images). The difference is used as the pixel value in the output image. In the output image, dark areas become lighter and light areas become darker. For color images, reds become cyan, greens become magenta, blues become yellow, and vice versa.

Tips

- If `I` is a grayscale or RGB image of class `double`, then you can use the expression `1-I` instead of this function.
- If `I` is a binary image, then you can use the expression `~I` instead of this function.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imcomplement` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- `imcomplement` does not support `int64` and `uint64` data types.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- `imcomplement` does not support `int64` and `uint64` data types.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also
`imabsdiff` | `imadd` | `imdivide` | `imlincomb` | `immultiply` | `imsubtract`

Introduced before R2006a

imcontour

Create contour plot of image data

Syntax

```
imcontour(I)
imcontour(I,levels)
imcontour(I,V)
imcontour(x,y,___)
imcontour(___,LineSpec)
[C,h] = imcontour(___)
```

Description

`imcontour(I)` draws a contour plot of the grayscale image `I`, choosing the number of levels and the values of levels automatically. `imcontour` automatically sets up the axes so their orientation and aspect ratio match the image.

`imcontour(I,levels)` specifies the number, `levels`, of equally spaced contour levels in the plot.

`imcontour(I,V)` draws contour lines at the data values specified in vector `V`. The number of contour levels is equal to `length(V)`.

`imcontour(x,y,___)` uses the vectors `x` and `y` to specify the image *x*- and *y* coordinates.

`imcontour(___,LineSpec)` draws the contours using the line type and color specified by `LineSpec`. Marker symbols are ignored.

`[C,h] = imcontour(___)` returns the contour matrix, `C`, and the contour patches, `h`, that are drawn onto the current axes.

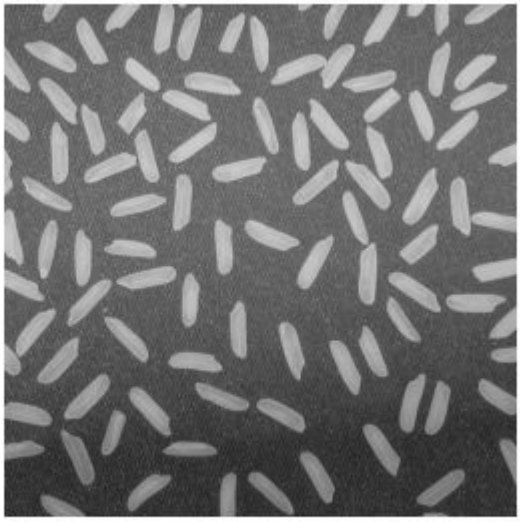
Examples

Create Contour Plot of Image Data

This example shows how to create a contour plot of an image.

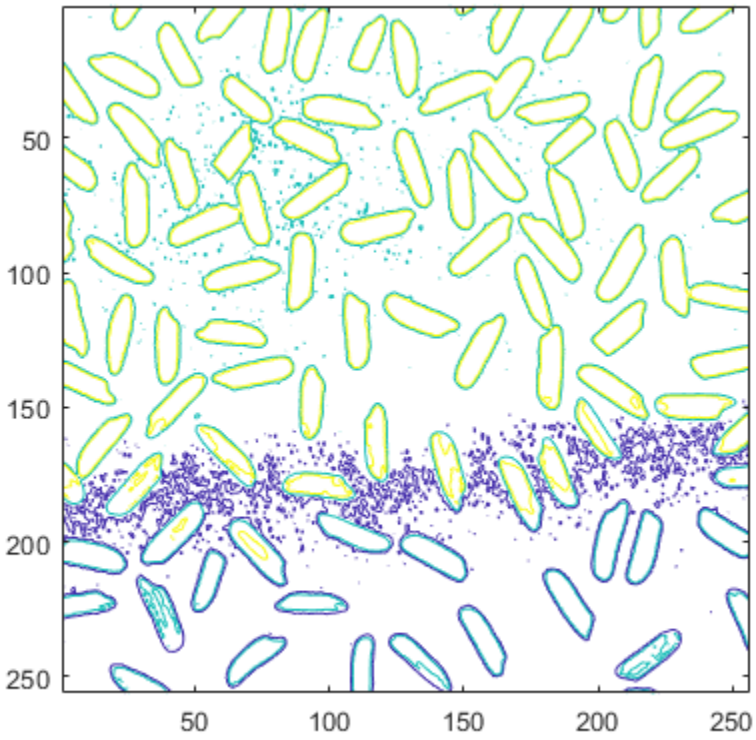
Read grayscale image and display it. The example uses an example image of grains of rice.

```
I = imread('rice.png');
imshow(I)
```



Create a contour plot of the image using `imcontour` .

```
figure;
imcontour(I,3)
```



Input Arguments

`I` — Grayscale image

m-by-*n* matrix

Grayscale image, specified as an *m*-by-*n* matrix.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`levels` — Number of contour levels

numeric scalar

Number of contour levels, specified as a numeric scalar.

`V` — Value of contour levels

numeric vector

Value of contour levels, specified as a numeric vector with length greater than or equal to two. Use `V = [v v]` to compute a single contour at level `v`.

`x` — Image *x* values

2-element numeric vector | *n*-element numeric vector

Image *x* values, specified as one of the following:

- 2-element numeric vector of the form `[xmin xmax]` — Image extent in the *x* direction.
- *n*-element numeric vector — *x*-coordinate of each column.

y — Image *y* values

2-element numeric vector | *m*-element numeric vector

Image *y* values, specified as one of the following:

- 2-element numeric vector of the form `[ymin ymax]` — Image extent in the *y* direction.
- *m*-element numeric vector — *y*-coordinate of each row.

`LineSpec` — Line style and color

character vector | string scalar

Line style and color, specified as a character vector or string scalar containing a line style specifier, a color specifier, or both.

Example: `"--r"` specifies red dashed lines

These two tables list the line style and color options.

Line Style	Description	Resulting Line
`"-"`	Solid line	
`"--"`	Dashed line	
`":"`	Dotted line	
`"-."`	Dash-dotted line	

Color Specifier	Description	Appearance
`r`	red	
`g`	green	
`b`	blue	
`c`	cyan	
`m`	magenta	
`k`	yellow	
`k`	black	
`w`	white	

Output Arguments

`C` — Contour matrix

numeric matrix

Contour matrix, returned as a matrix with two rows. The matrix is defined according to the `ContourMatrix` property of the `Contour` object, `h`.

h — Contour patches
`Contour` object

Contour patches, returned as a `Contour` object.

See Also

Functions
`contour` | `clabel`

Properties
Contour

Introduced before R2006a

imcontrast

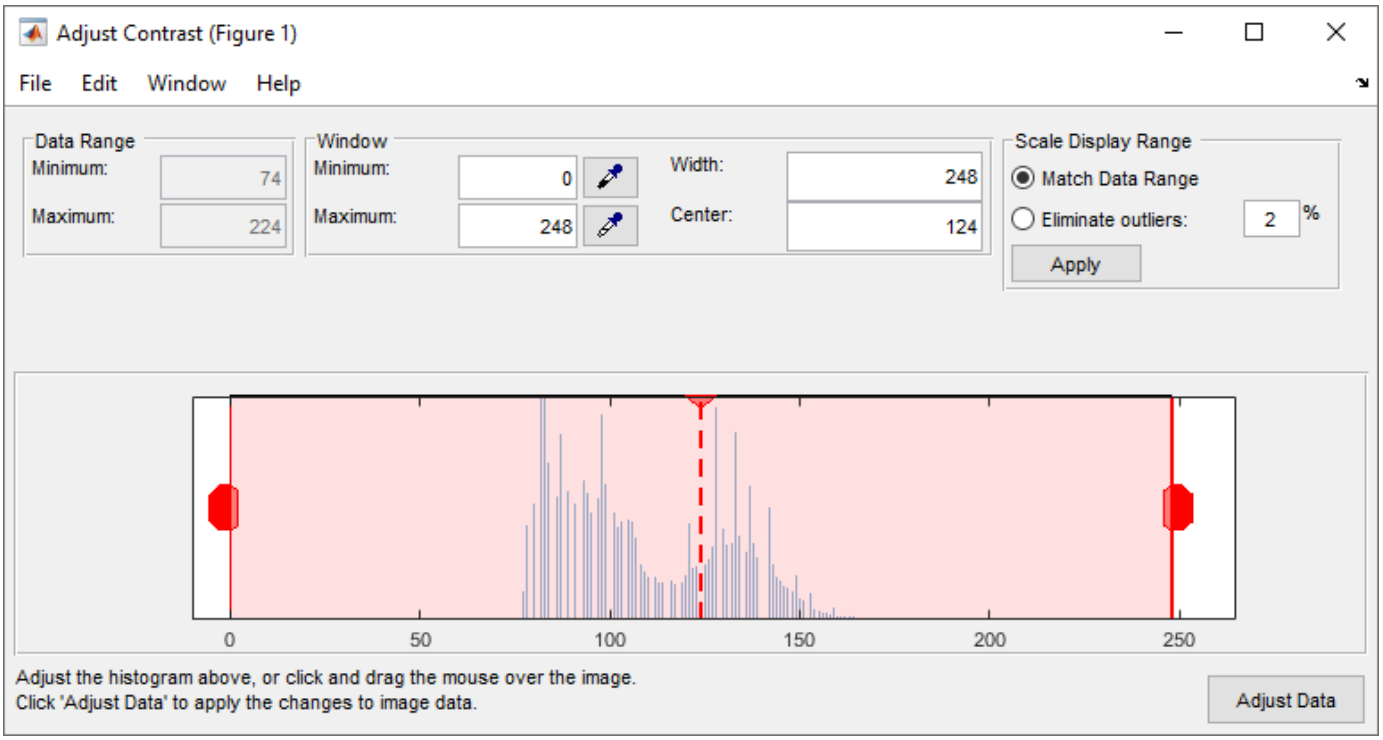
Adjust Contrast tool

Syntax

```
imcontrast
imcontrast(h)
htool = imcontrast( ___ )
```

Description

Use the `imcontrast` function to create an Adjust Contrast tool. The Adjust Contrast tool is an interactive contrast and brightness adjustment tool that you can use to adjust the black-to-white mapping used to display a grayscale image. For more information about using the tool, see “Tips” on page 1-1434.



`imcontrast` creates an Adjust Contrast tool in a separate figure that is associated with the grayscale image in the current figure, called the target image.

`imcontrast(h)` creates the Adjust Contrast tool associated with the image specified by the handle `h`.

`htool = imcontrast( ___ )` returns the handle `htool` to the Adjust Contrast tool figure.

Examples

Adjust the Contrast of the Current Image

Read an image into the workspace. Adjust the contrast of the current image.

```
imshow('pout.tif')
imcontrast
```

Adjust Image Contrast, Specifying Figure Handle

Read an image into the workspace and define the handle of the figure as `h1`. Open a second figure window and define the handle of that figure as `h2`. Adjust the contrast of the first figure by specifying `h1` in the call to `imcontrast`.

```
h1 = figure;
imshow('pout.tif');
h2 = figure;
imshow('coins.png');
imcontrast(h1)
```

Input Arguments

h — Handle to graphics object

handle

Handle to a figure, axes, uipanel, or image graphics object, specified as a handle. If `h` is an axes or figure handle, then `imcontrast` uses the first image returned by `findobj(H,'Type','image')`.

Output Arguments

htool — Handle to Adjust Contrast tool figure

handle

Handle to Adjust Contrast tool figure, returned as a handle.

Tips

The Adjust Contrast tool presents a scaled histogram of pixel values (overly represented pixel values are truncated for clarity). Dragging on the left red bar in the histogram display changes the minimum value. The minimum value, and any pixel value less than the minimum, display as black. Dragging on the right red bar in the histogram changes the maximum value. The maximum value, and any value greater than the maximum, display as white. Values in between the red bars display as intermediate shades of gray.

Together the minimum and maximum values create a "window". Stretching the window reduces contrast. Shrinking the window increases contrast. Changing the center of the window changes the brightness of the image. It is possible to manually enter the minimum, maximum, width, and center values for the window. Changing one value automatically updates the other values and the image.

- Click and drag the mouse within the target image to interactively change the window values. Dragging the mouse horizontally from left to right changes the window width. Dragging the mouse vertically up and down changes the window center. Holding down the **Ctrl** key before clicking and dragging the mouse accelerates the rate of change; holding down the **Shift** key before clicking and dragging the mouse slows the rate of change. Keys must be pressed before clicking and dragging.
- When you use the tool, `imcontrast` adjusts the contrast of the displayed image by modifying the axes `CLim` property. To modify the actual pixel values in the target image, click the **Adjust Data** button. This button is unavailable until you make a change to the contrast of the image.
- The Adjust Contrast tool can handle grayscale images of class `double` and `single` with data ranges beyond the default display range, which is `[0 1]`. For these images, `imcontrast` sets the histogram limits to fit the image data range, with padding at the upper and lower bounds.

See Also

Image Viewer | `imadjust` | `stretchlim`

Topics

“Adjust Image Contrast in Image Viewer App”

Introduced before R2006a

imcrop

Crop image

Syntax

```
Icropped = imcrop
Icropped = imcrop(I)
Xcropped = imcrop(X,cmap)
___ = imcrop(h)

Icropped = imcrop(I,rect)
Xcropped = imcrop(X,cmap,rect)
___ = imcrop(xref,yref,___)

[___,rectout] = imcrop(___)
[xrefout,yrefout,___] = imcrop(___)
imcrop(___)
```

Description

Crop Image Interactively

Note The interactive syntaxes do not support categorical images. For categorical images, you must specify the crop region, `rect`.

`Icropped = imcrop` creates an interactive Crop Image tool associated with the grayscale, truecolor, or binary image displayed in the current figure. `imcrop` returns the cropped image, `Icropped`.

With this syntax and the other interactive syntaxes, the Crop Image tool blocks the MATLAB command line until you complete the operation. For more information about using the Crop Image tool, see “Interactive Behavior” on page 1-1446.

`Icropped = imcrop(I)` displays the grayscale, truecolor, or binary image `I` in a figure window and creates an interactive Crop Image tool associated with the image.

`Xcropped = imcrop(X,cmap)` displays the indexed image `X` in a figure using the colormap `cmap`, and creates an interactive Crop Image tool associated with that image. `imcrop` returns the cropped indexed image, `Xcropped`, which also has the colormap `cmap`.

`___ = imcrop(h)` creates an interactive Crop Image tool associated with the image specified by the handle `h`.

Crop Image by Specifying Crop Region

`Icropped = imcrop(I,rect)` crops the image `I` according to the position and dimensions specified in the crop rectangle `rect`. The cropped image includes all pixels in the input image that are completely *or partially* enclosed by the rectangle.

The actual size of the output image does not always correspond exactly with the width and height specified by `rect`. For example, suppose `rect` is `[20 20 40 30]`, using the default spatial coordinate system. The upper left corner of the specified rectangle is the center of the pixel with spatial (*x*,*y*) coordinates (20,20). The lower right corner of the rectangle is the center of the pixel with spatial (*x*,*y*) coordinates (60,50). The resulting output image has size 31-by-41 pixels, not 30-by-40 pixels.

`Xcropped = imcrop(X,cmap,rect)` crops the indexed image `X` with colormap `cmap` according to the position and dimensions specified in the crop rectangle `rect`. `imcrop` returns the cropped indexed image, `Xcropped`, which also has the colormap `cmap`.

`___ = imcrop(xref,yref, ___ )` crops the input image using the world coordinate system defined by `xref` and `yref`. After the `xref` and `yref` input arguments, you can specify the arguments of any syntax that includes an input image `I` or `X`.

Specify Additional Output Options

`[ ___ ,rectout] = imcrop( ___ )` also returns the position of the crop rectangle in `rectout`. You can use the input arguments of any other syntax.

`[xrefout,yrefout, ___ ] = imcrop( ___ )` also returns the image limits of the input image in `xrefout` and `yrefout`.

`imcrop( ___ )` without output arguments displays the cropped image in a new figure window. This syntax does not support categorical images.

Examples

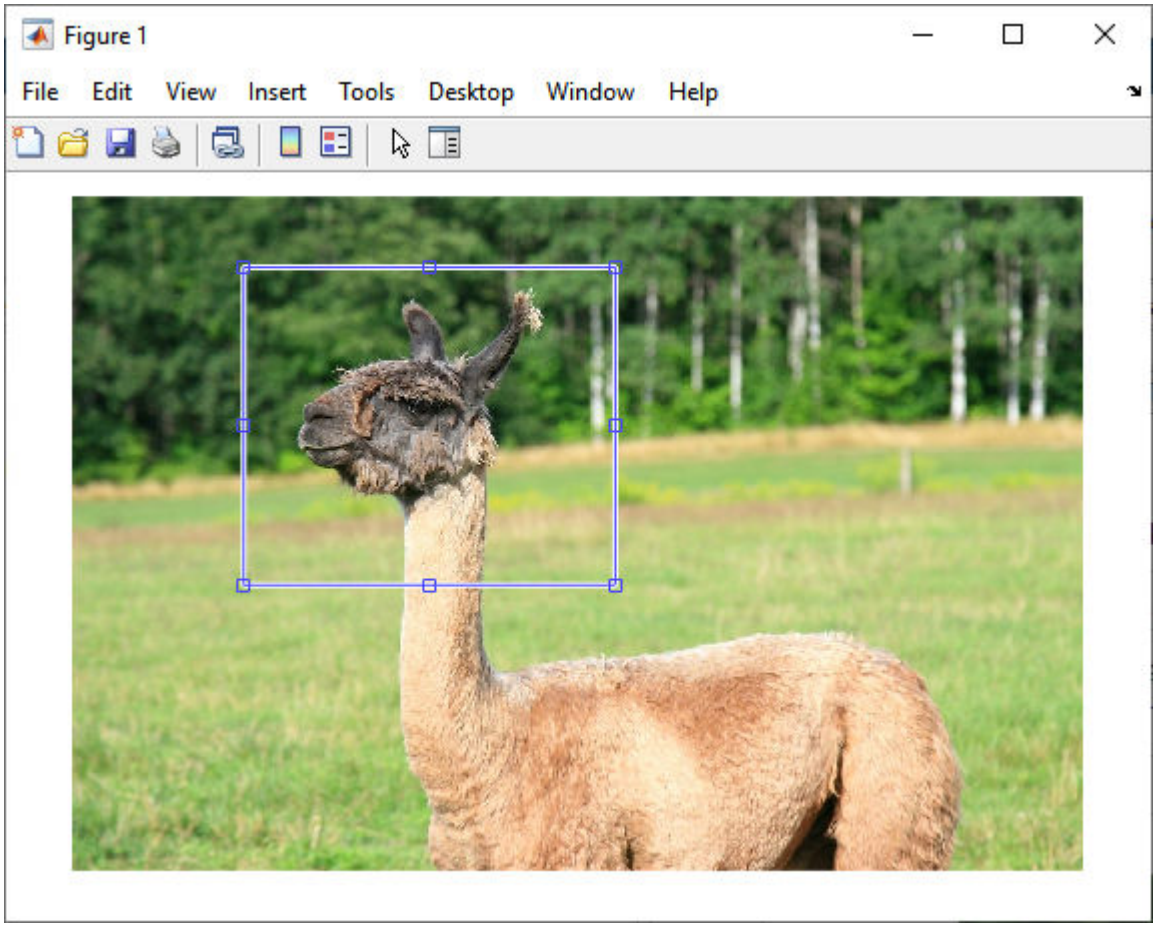
Crop Image Using Interactive Crop Image Tool

Read image into the workspace.

```
I = imread('llama.jpg');
```

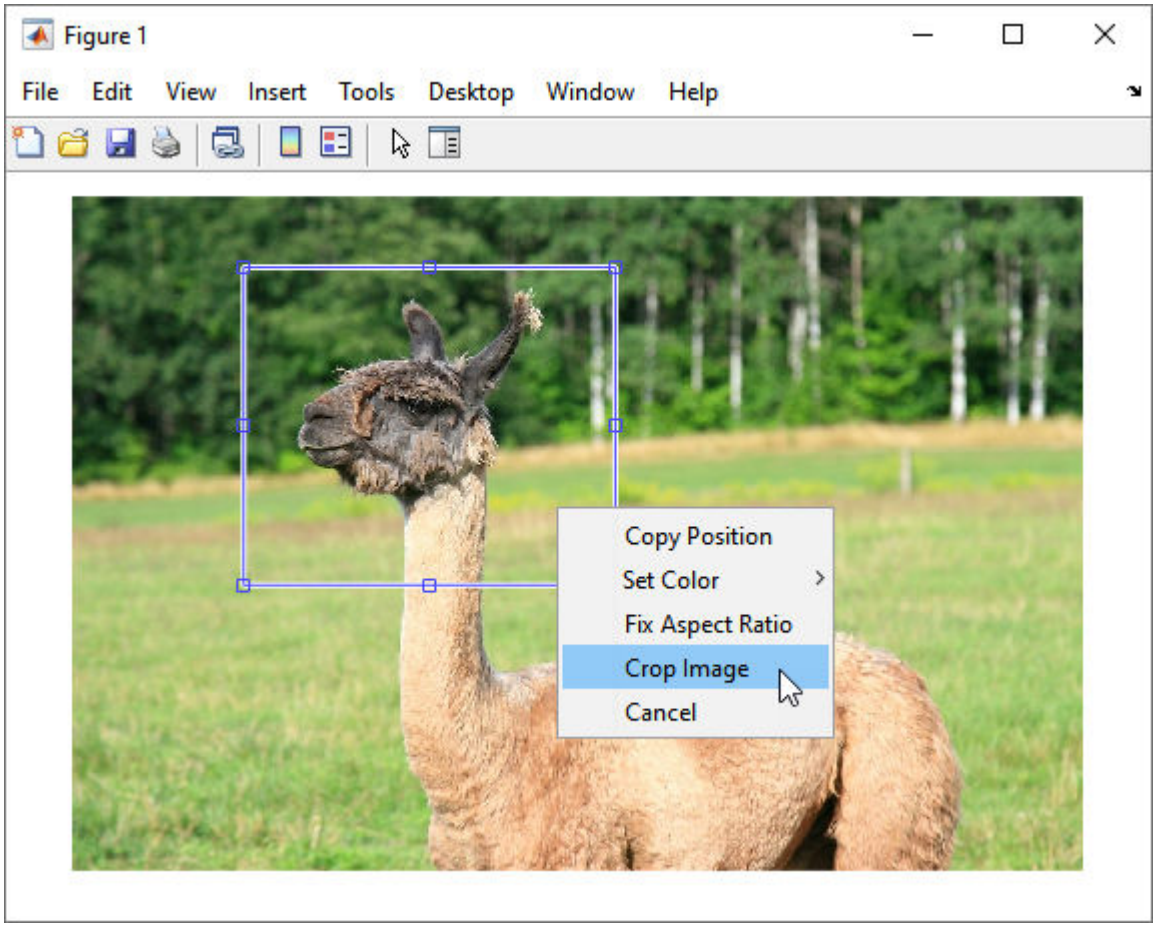
Open the Crop Image tool associated with this image. Specify a variable in which to store the cropped image. The example includes the optional return value `rect` in which `imcrop` returns the four-element position vector of the rectangle you draw.

```
[J,rect] = imcrop(I);
```

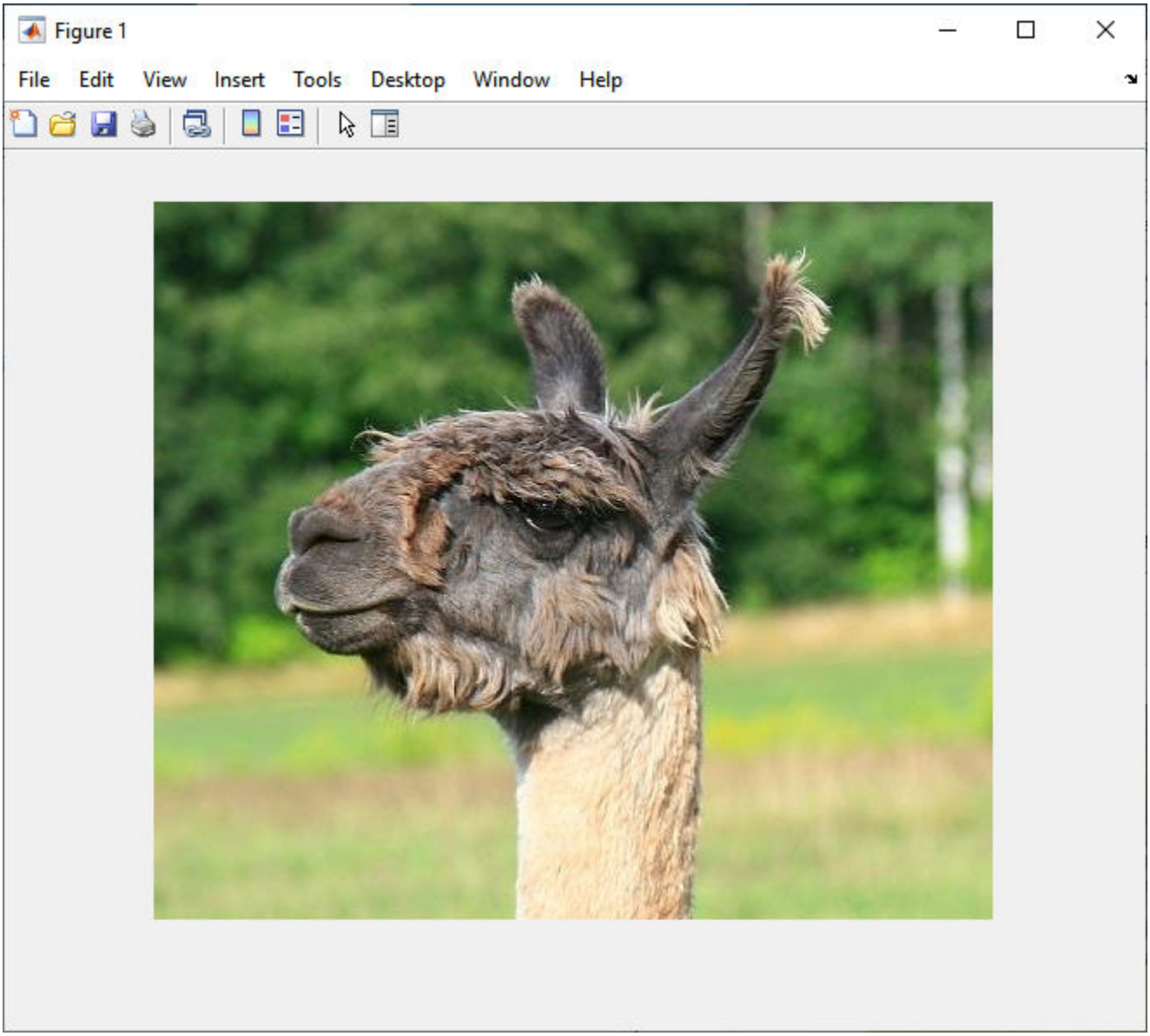


When you move the cursor over the image, it changes to a cross-hairs. The Crop Image tool blocks the command line until you complete the operation.

Using the mouse, draw a rectangle over the portion of the image that you want to crop. Perform the crop operation by double-clicking in the crop rectangle or selecting Crop Image on the context menu.



The cropped image appears in the figure window.



The Crop Image tool returns the cropped area in the return variable, `J`. The variable `rect` is the four-element position vector describing the crop rectangle you specified. Get information about the returned variables.

```
whos
```

```
  Name        Size                 Bytes  Class     Attributes

  I         876x1314x3           3453192  uint8
  J         413x483x3             598437  uint8
  rect        1x4                     32  double
```

Crop Image By Specifying Crop Rectangle

Read image into the workspace.

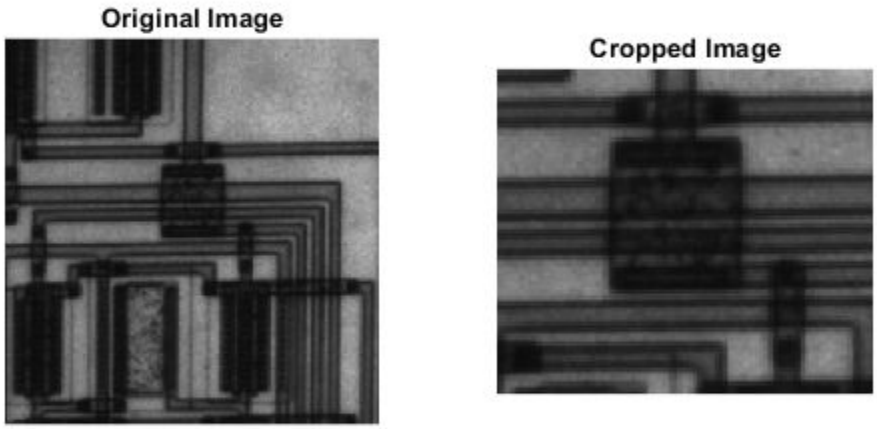
```
I = imread('circuit.tif');
```

Crop image, specifying crop rectangle.

```
I2 = imcrop(I,[75 68 130 112]);
```

Display original image and cropped image.

```
subplot(1,2,1)
imshow(I)
title('Original Image')
subplot(1,2,2)
imshow(I2)
title('Cropped Image')
```



Center Crop Image Using Spatial Referencing Rectangle

Read and display an image.

```
I = imread('parkavenue.jpg');
imshow(I)
```

Specify a target window size as a two-element vector of the form [*width*, *height*].

```
targetSize = [300 600];
```

Create a `Rectangle` object that specifies the spatial extent of the crop window.

```
r = centerCropWindow2d(size(I),targetSize);
```

Crop the image to the spatial extents. Display the cropped region.

```
J = imcrop(I,r);
imshow(J)
```

Crop Indexed Image Specifying Crop Rectangle

Load indexed image with its associated map into the workspace.

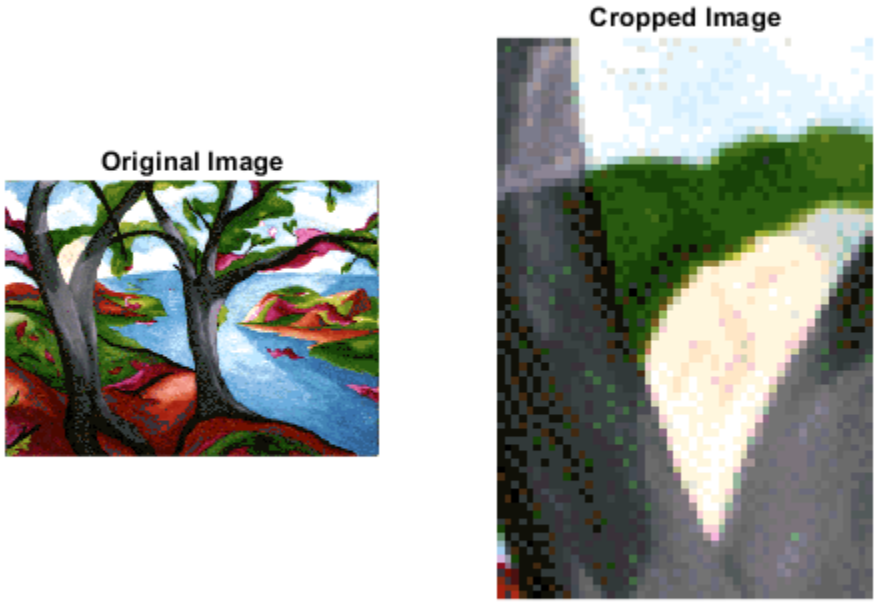
```
load trees
```

Crop indexed image, specifying crop rectangle.

```
X2 = imcrop(X,map,[30 30 50 75]);
```

Display original image and cropped image.

```
subplot(1,2,1)
imshow(X,map)
title('Original Image')
subplot(1,2,2)
imshow(X2,map)
title('Cropped Image')
```



Input Arguments

`I` — Image to be cropped

numeric matrix | numeric array | logical matrix | categorical matrix

Image to be cropped, specified as one of the following.

- *m*-by-*n* numeric matrix representing a grayscale image
- *m*-by-*n*-by-3 numeric array representing a truecolor image
- *m*-by-*n* logical matrix representing a binary mask.
- *m*-by-*n* categorical matrix representing a label image.

> **Note** For categorical input, you must specify a crop rectangle, `rect`. The interactive syntaxes do not support categorical input.

When you use an interactive syntax, `imcrop` calls the `imshow` function and accepts whatever image classes `imshow` accepts.

Data Types: `single` | `double` | `int8` | `int16` | `uint8` | `uint16` | `logical` | `categorical`

`X` — Indexed image to be cropped

matrix of integers

Indexed image to be cropped, specified as a matrix of integers.

Data Types: `single` | `double` | `int8` | `int16` | `uint8` | `uint16` | `logical`

`cmap` — Colormap
c-by-3 numeric matrix

Colormap associated with the indexed image `X`, specified as a *c*-by-3 numeric matrix. Each row is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap. Values with data type `single` or `double` must be in the range [0, 1].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`rect` — Size and position of crop rectangle
4-element numeric vector | `Rectangle` object

Size and position of the crop rectangle in spatial coordinates, specified as a 4-element numeric vector of the form `[xmin ymin width height]` or an `images.spatialref.Rectangle` object.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`h` — Input image
handle

Input image, specified as a handle to a figure, axes, uipanel, or image graphics object. If `h` is an axes or figure handle, then `imcrop` uses the first image returned by `findobj(H,'Type','image')`.

`xref` — Image limits in world coordinates along *x*-dimension
2-element numeric vector

Image limits in world coordinates along the *x*-dimension, specified as a 2-element numeric vector of the form [*xmin xmax*] where *xmax* is greater than *xmin*. The value of `xref` sets the image `XData`.

`yref` — Image limits in world coordinates along *y*-dimension
2-element numeric vector

Image limits in world coordinates along the *y*-dimension, specified as a 2-element numeric vector of the form [*ymin ymax*] where *ymax* is greater than *ymin*. The value of `yref` sets the image `YData`.

Output Arguments

`Icropped` — Cropped image
numeric array | numeric matrix | logical matrix | categorical matrix

Cropped image, returned as a numeric array, numeric matrix, logical matrix, or categorical matrix.

- If you specify an input image `I`, then the output image has the same data type as the input image.
- If you do not specify an input image, then the output image generally has the same data type as the input image. However, if the input image has data type `int16` or `single`, then the output image has data type `double`.

`Xcropped` — Cropped indexed image
numeric matrix

Cropped indexed image, returned as a numeric matrix.

`rectout` — Size and position of crop rectangle
4-element numeric vector

Size and position of the crop rectangle, returned as a 4-element numeric vector of the form `[xmin ymin width height]`.

`xrefout` — Image limits in world coordinates along *x*-dimension
2-element numeric vector

Image limits in world coordinates along the *x*-dimension, returned as a 2-element numeric vector of the form [*xmin xmax*]. If you specify image limits in a world coordinate system using `xref`, then `xrefout` is equal to `xref`. Otherwise, `xrefout` is equal to the original image `XData`.

`yrefout` — Image limits in world coordinates along *y*-dimension
2-element numeric vector

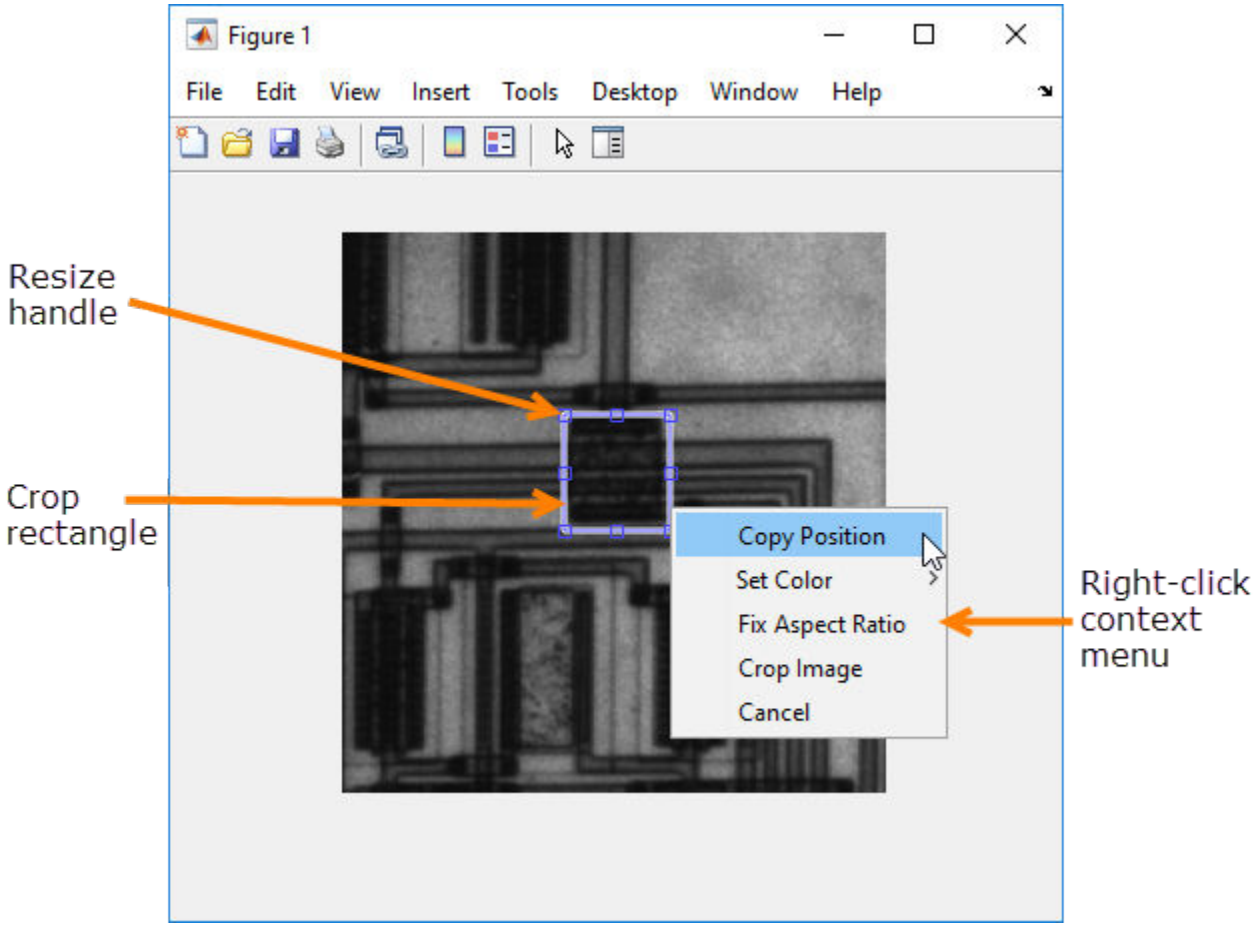
Image limits in world coordinates along the *y*-dimension, returned as a 2-element numeric vector of the form [*ymin ymax*]. If you specify image limits in a world coordinate system using `yin`, then `yrefout` is equal to `yin`. Otherwise, `yrefout` is equal to the original image `YData`.

More About

Interactive Behavior

The Crop Image tool is a moveable, resizeable rectangle that you can position over the image and perform the crop operation interactively using the mouse.

When the Crop Image tool is active in a figure, the pointer changes to cross hairs  when you move it over the target image. Using the mouse, you specify the crop rectangle by clicking and dragging the mouse. You can move or resize the crop rectangle using the mouse. When you are finished sizing and positioning the crop rectangle, create the cropped image by double-clicking the left mouse button. You can also choose **Crop Image** from the context menu. The figure illustrates the Crop Image tool with the context menu displayed.



Interactive Behavior	Description
Delete the Crop Image tool.	Press **Backspace**, **Escape** or **Delete**, or right-click inside the crop rectangle and select **Cancel** from the context menu. Note: If you delete the ROI, the function returns empty values.
Resize the Crop Image tool.	Select any of the resize handles on the crop rectangle. The pointer changes to a double-headed arrow ↔. Click and drag the mouse to resize the crop rectangle.
Move the Crop Image tool.	Move the pointer inside the boundary of the crop rectangle. The pointer changes to a fleur shape ✥. Click and drag the mouse to move the rectangle over the image.
Change the color used to display the crop rectangle.	Right-click inside the boundary of the crop rectangle and select **Set Color** from the context menu.
Crop the image.	Double-click the left mouse button or right-click inside the boundary of the crop rectangle and select **Crop Image** from the context menu.

Interactive Behavior	Description
Retrieve the coordinates of the crop rectangle.	Right-click inside the boundary of the crop rectangle and select **Copy Position** from the context menu. `imcrop` copies a 4-element position vector (`[xmin ymin width height]`) to the clipboard.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imcrop` supports the generation of C code (requires MATLAB Coder). For more information, see "Code Generation for Image Processing".
- The interactive syntaxes are not supported, including:
  - `Icropped = imcrop`
  - `Icropped = imcrop(I)`
  - `Xcropped = imcrop(X,cmap)`
  - `Icropped = imcrop(h)`
- Indexed images are not supported, including the non-interactive syntax `Xcropped = imcrop(X,cmap,rect);`

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The interactive syntaxes are not supported, including:
  - `Icropped = imcrop`
  - `Icropped = imcrop(I)`
  - `Xcropped = imcrop(X,cmap)`
  - `Icropped = imcrop(h)`
- Indexed images are not supported, including the non-interactive syntax `Xcropped = imcrop(X,cmap,rect);`

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

Usage notes and limitations:

- The interactive syntaxes are not supported, including:
  - `Icropped = imcrop`
  - `Icropped = imcrop(I)`
  - `Xcropped = imcrop(X,cmap)`
  - `Icropped = imcrop(h)`

For more information, see “Run MATLAB Functions in Thread-Based Environment”.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- The interactive syntaxes are not supported, including:
  - `Icropped = imcrop`
  - `Icropped = imcrop(I)`
  - `Xcropped = imcrop(X,cmap)`
  - `Icropped = imcrop(h)`
- Categorical images are not supported.

For more information, see “Image Processing on a GPU”.

See Also

`zoom` | `imcrop3` | `drawrectangle` | `images.spatialref.Rectangle`

Topics
“Image Types in the Toolbox”
“Define World Coordinate System of Image”

Introduced before R2006a

imcrop3

Crop 3-D image

Syntax

```
Vout = imcrop3(V,cuboid)
```

Description

`Vout = imcrop3(V,cuboid)` crops the image volume `V` according to `cuboid`, which specifies the size and position of the cropping window in spatial coordinates.

Examples

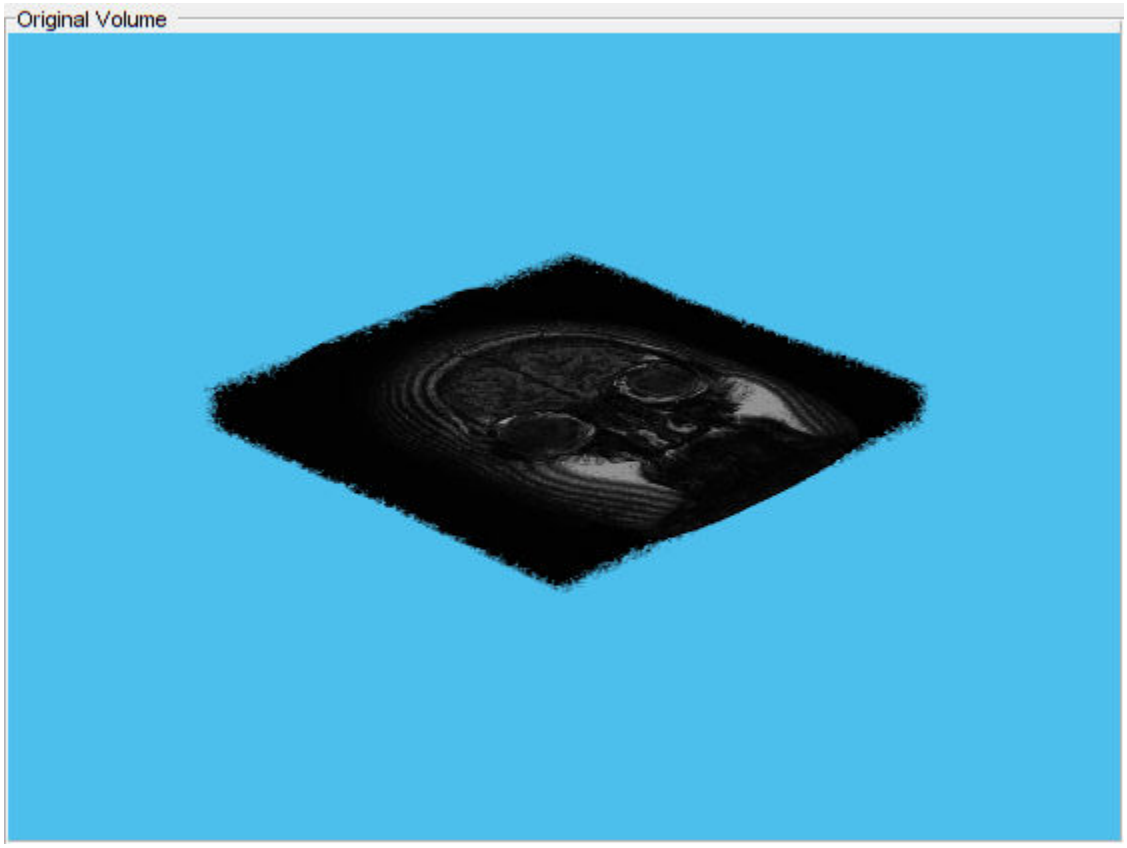
Crop 3-D Volume

Load a 3-D volume into the workspace.

```
D = load('mristack');
V = D.mristack;
```

Display the image.

```
fullViewPnl = uipanel(figure,'Title','Original Volume');
volshow(V,'Parent',fullViewPnl);
```

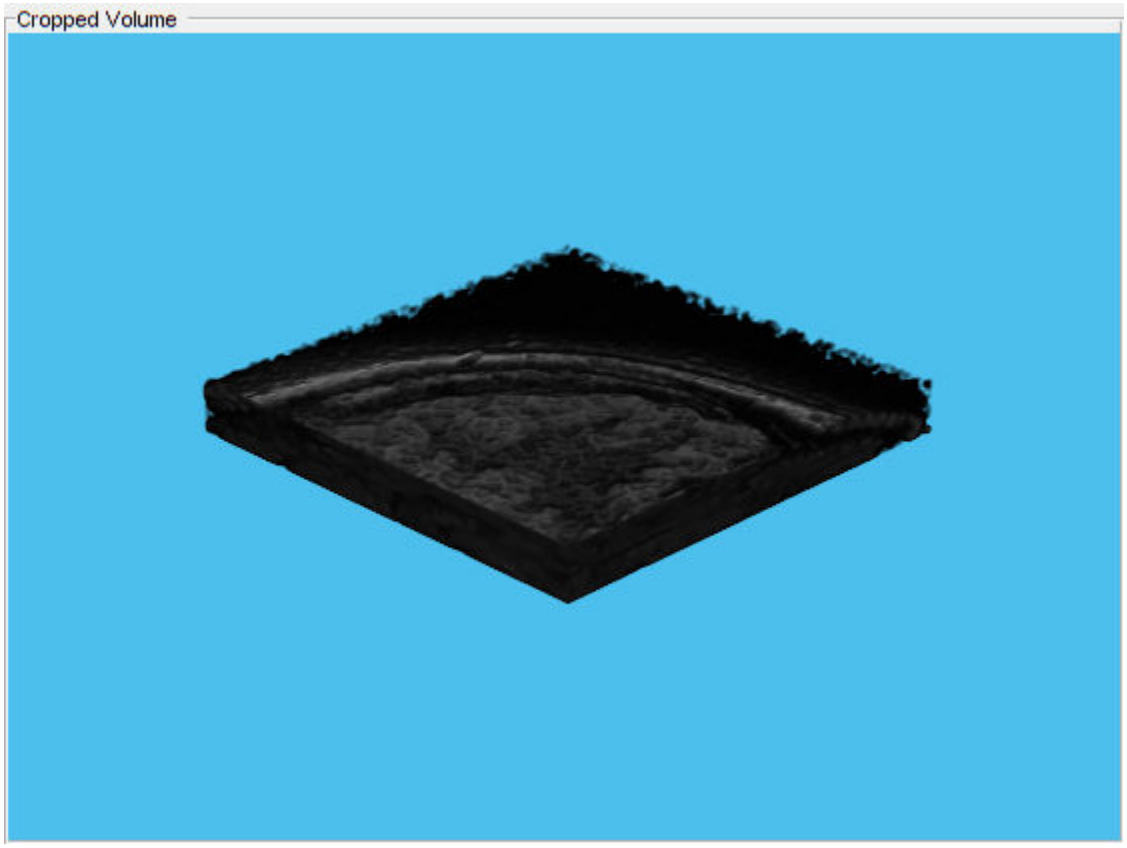


Crop the volume using `imcrop3`, specifying the size and position of the cuboidal crop region.

```
Vout = imcrop3(V,[30 40 10 100 100 10]);
```

Display the cropped image.

```
fullViewPnl = uipanel(figure,'Title','Cropped Volume');
volshow(Vout,'Parent',fullViewPnl);
```



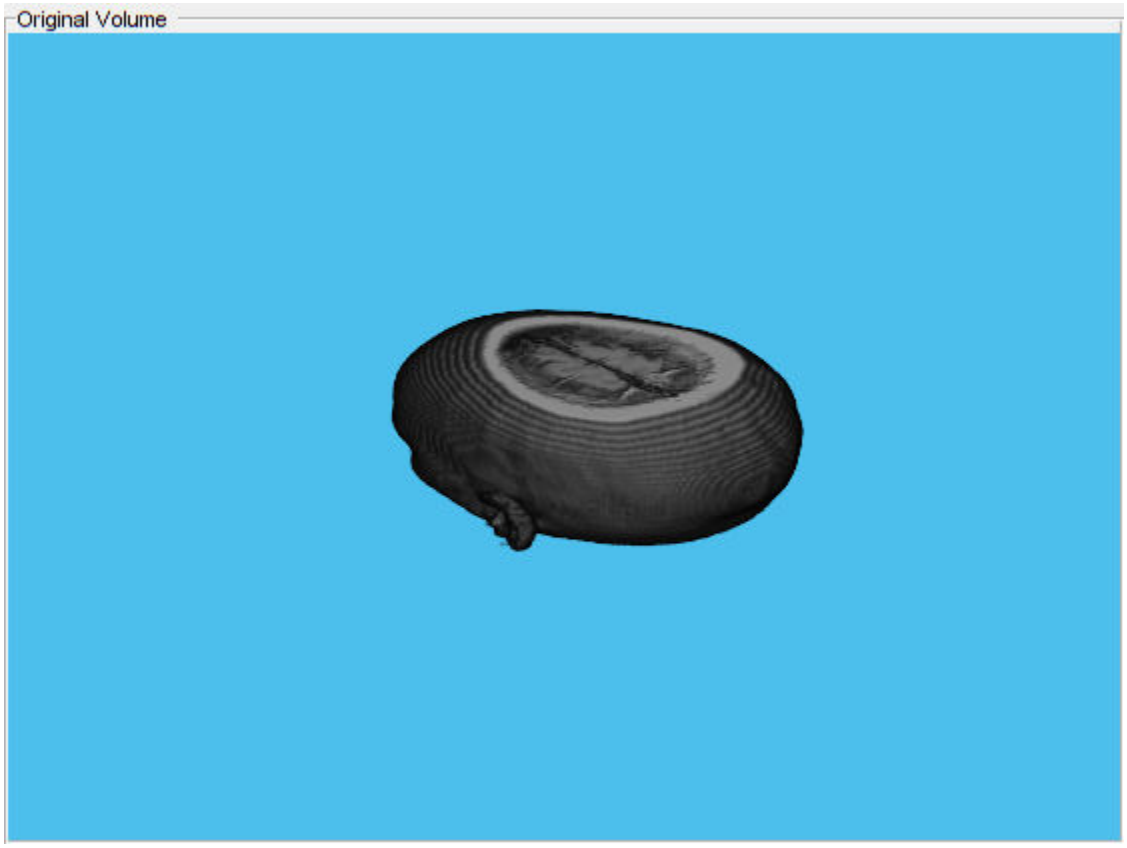
Crop 3-D Image Volume Using Fixed Off-Center Spatial Extent

Load a 3-D MRI image. Use the `squeeze` function to remove any singleton dimensions.

```
S = load('mri.mat','D');
volumeData = squeeze(S.D);
```

Display the image.

```
fullViewPnl = uipanel(figure,'Title','Original Volume');
volshow(volumeData,'Parent',fullViewPnl);
```



Create a `Cuboid` object and specify the cropping window size in all three dimensions.

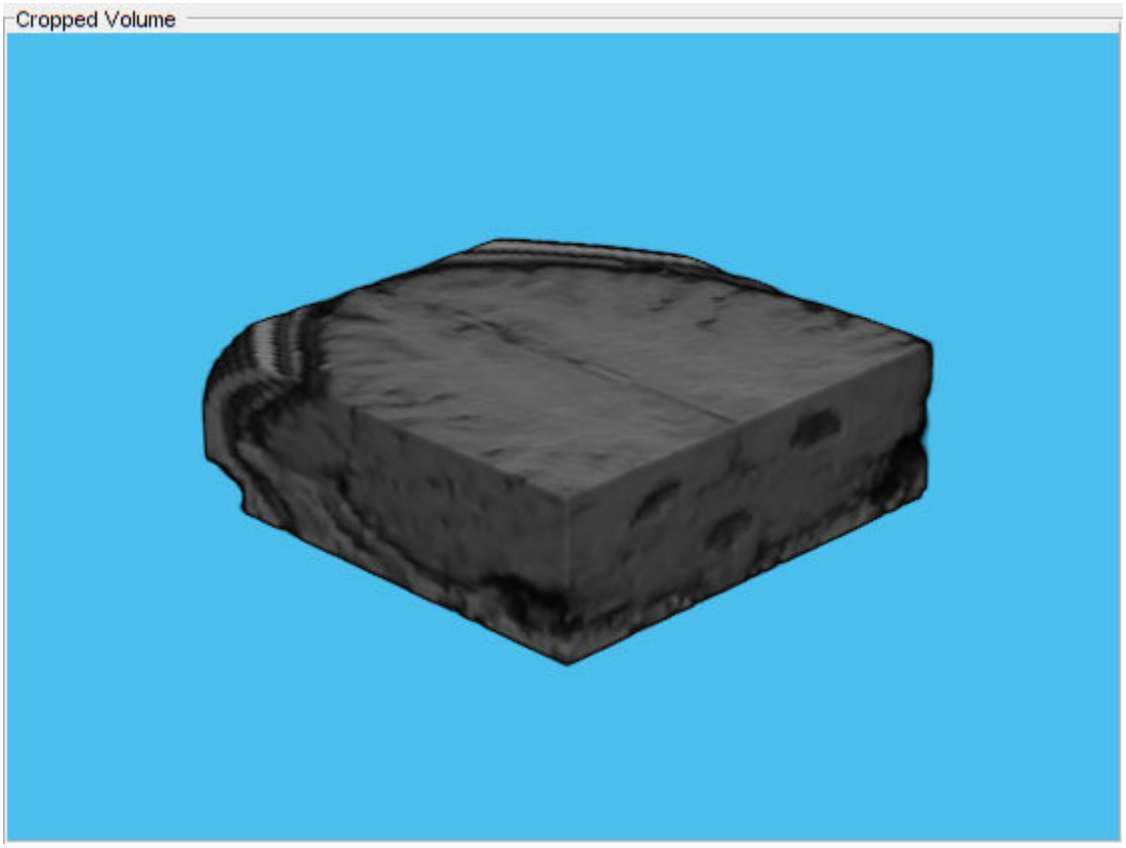
```
c = images.spatialref.Cuboid([30,90],[30,90],[1,20]);
```

Crop the image based on the `Cuboid` dimensions.

```
croppedVolume = imcrop3(volumeData,c);
```

Display the cropped image.

```
fullViewPnl = uipanel(figure,'Title','Cropped Volume');
volshow(croppedVolume,'Parent',fullViewPnl);
```



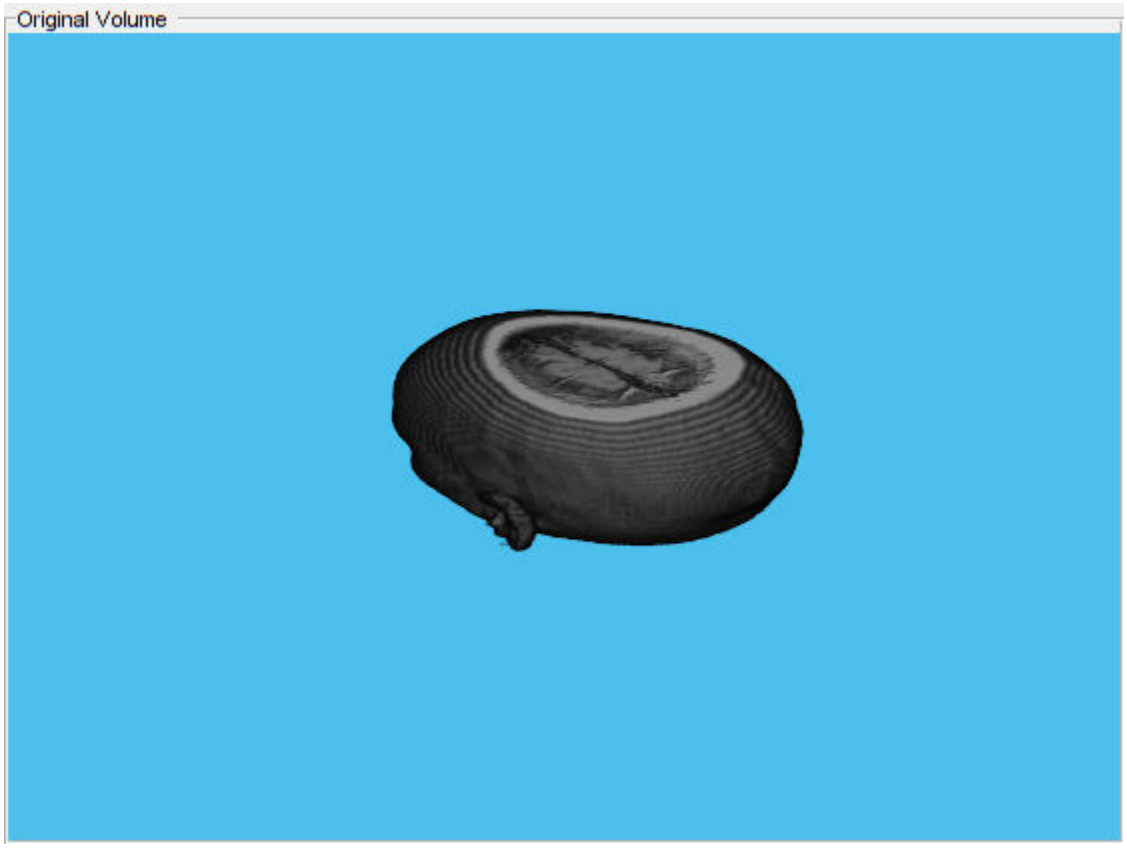
Center Crop 3-D Image to Target Size

Load a 3-D MRI image. Use the `squeeze` function to remove any singleton dimensions.

```
load mri;
D = squeeze(D);
```

Display the image.

```
fullViewPnl = uipanel(figure,'Title','Original Volume');
volshow(D,'Parent',fullViewPnl);
```



Specify the target size of the cropping window.

```
targetSize = [64 64 10];
```

Create a center cropping window that crops the specified image from its center.

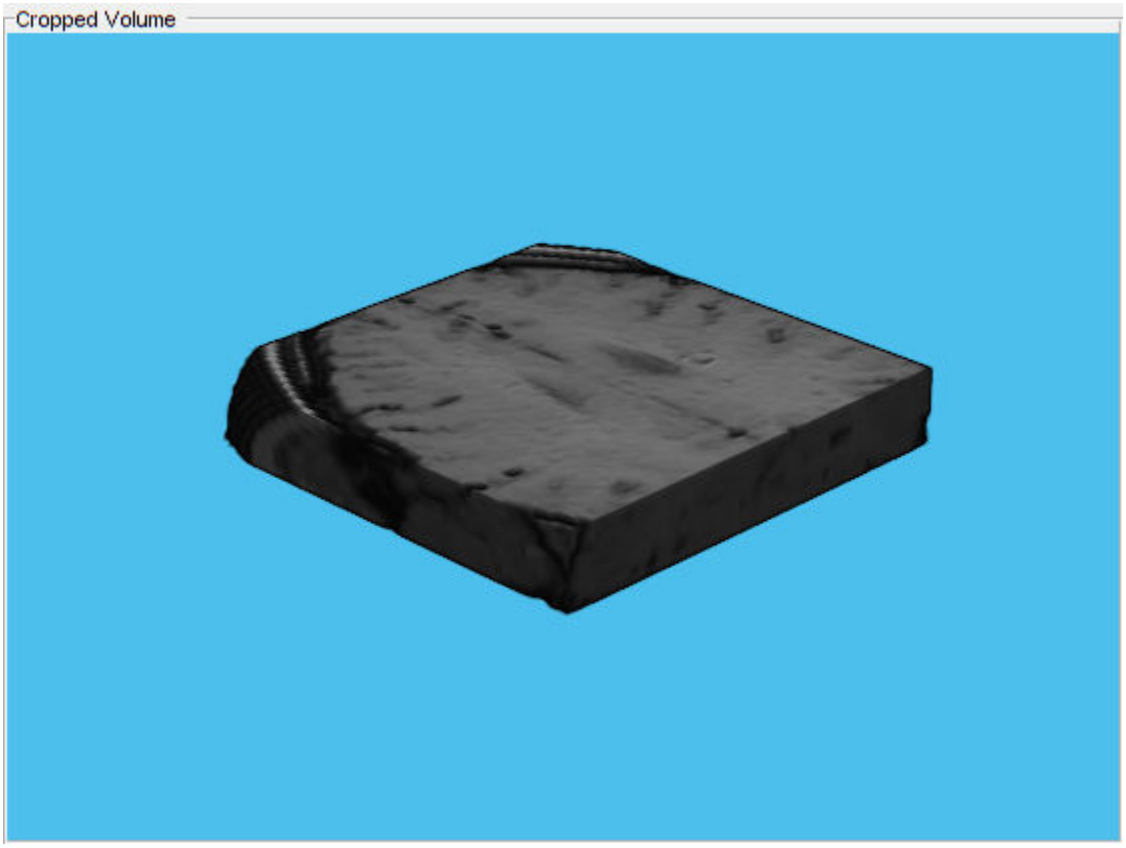
```
win = centerCropWindow3d(size(D),targetSize);
```

Crop the image using the center cropping window.

```
Dcrop = imcrop3(D,win);
```

Display the cropped image in a display panel.

```
fullViewPnl = uipanel(figure,'Title','Cropped Volume');
volshow(Dcrop,'Parent',fullViewPnl);
```



Input Arguments

`V` — Volume to be cropped

numeric array | logical array | categorical array

Volume to be cropped, specified as a numeric, logical, or categorical array. `V` can be a 3-D array that represents a single channel 3-D volume or a 4-D array that represents a multichannel 3-D volume. If `V` represents a multichannel 3-D volume, then `imcrop3` crops the first three dimensions only.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `categorical`

`cuboid` — Size and position of crop volume

6-element numeric vector | `Cuboid` object

Size and position of the crop volume in spatial coordinates, specified as a 6-element vector of the form `[xmin ymin zmin width height depth]` or a `images.spatialref.Cuboid` object.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`Vout` — Cropped volume

logical, numeric, or categorical array

Cropped volume, returned as a logical, numeric, or categorical array of the same class as the input volume `V`.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

`images.spatialref.Cuboid` | `imcrop`

Introduced in R2019b

imdiffuseest

Estimate parameters for anisotropic diffusion filtering

Syntax

```
[gradientThreshold,numberOfIterations] = imdiffuseest(I)
[gradientThreshold,numberOfIterations] = imdiffuseest(I,Name,Value)
```

Description

`[gradientThreshold,numberOfIterations] = imdiffuseest(I)` estimates the gradient threshold and number of iterations required to filter the grayscale image `I` using anisotropic diffusion.

`[gradientThreshold,numberOfIterations] = imdiffuseest(I,Name,Value)` uses name-value pairs to change the behavior of the anisotropic diffusion algorithm.

Examples

Estimate Parameters for Anisotropic Diffusion Filtering

Read a grayscale image, then apply strong Gaussian noise to it. Display the noisy image.

```
I = imread('pout.tif');
Inoisy = imnoise(I,'gaussian',0,0.005);
imshow(Inoisy)
title('Noisy Image')
```

Estimate the gradient threshold and number of iterations needed to perform anisotropic diffusion filtering of the image.

```
[gradThresh,numIter] = imdiffuseest(Inoisy)
```

```
gradThresh = 1x5 uint8 row vector

   64   50   39   34   29


numIter = 5
```

Filter the noisy image by using anisotropic diffusion with the estimated parameters.

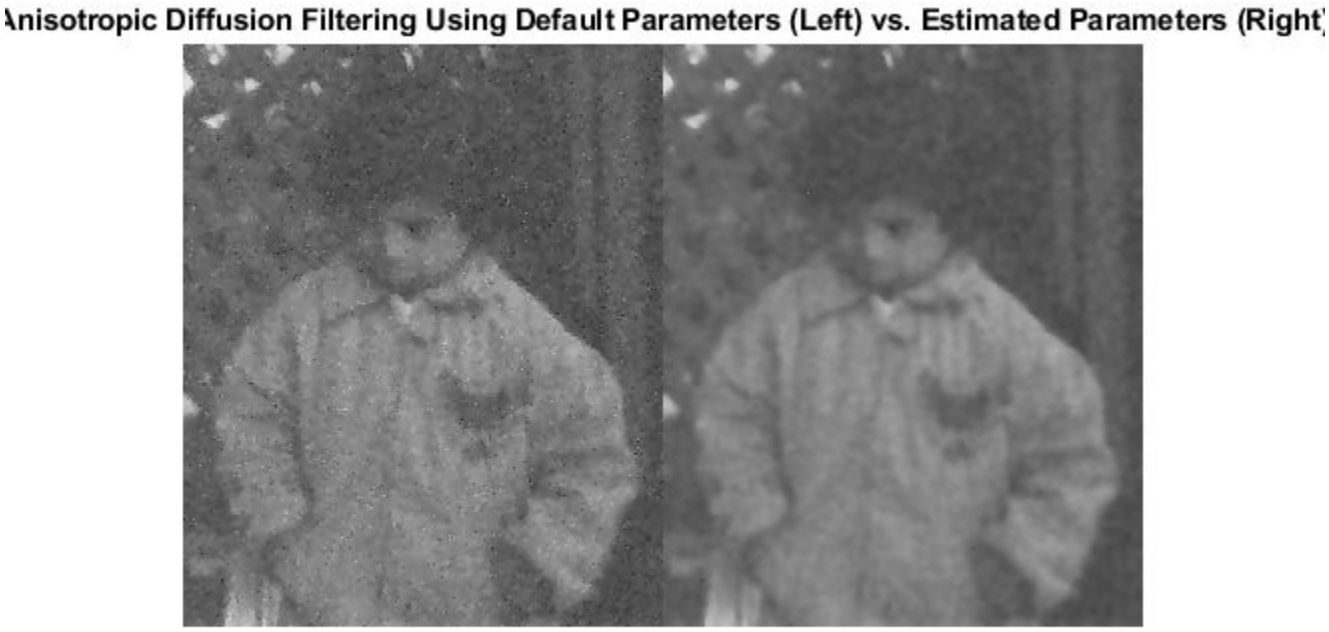
```
Idiffuseest = imdiffusefilt(Inoisy,'GradientThreshold', ...
     gradThresh,'NumberOfIterations',numIter);
```

For comparison, also filter the noisy image by using anisotropic diffusion with the default parameters. The default gradient threshold is 25.5 because the data type of the image is `uint8`, and the default number of iterations is 5.

```
Idiffusedef = imdiffusefilt(Inoisy);
```

Visually compare the two filtered images.

```
 montage({Idiffusedef,Idiffuseest},'ThumbnailSize',[])
 title(['Anisotropic Diffusion Filtering Using ' ...
     'Default Parameters (Left) vs. Estimated Parameters (Right)'])
```



Some noise remains in the image that was filtered using default parameters. The noise is almost completely absent from the image that was filtered using estimated parameters. The sharpness of edges in both images, especially high-contrast edges such as the trellis and white collar, is preserved.

Input Arguments

`I` — Image

2-D grayscale image

Image to be filtered, specified as a 2-D grayscale image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `imdiffuseest(I,'Connectivity','minimal')` estimates parameters required for anisotropic diffusion on image `I`, using minimal connectivity.

`Connectivity` — Connectivity

`'maximal'` (default) | `'minimal'`

Connectivity of a pixel to its neighbors, specified as the comma-separated pair consisting of `'Connectivity'` and `'maximal'` or `'minimal'`. Maximal connectivity considers eight nearest neighbors and minimal connectivity considers four nearest neighbors.

`ConductionMethod` — Conduction method
`'exponential'` (default) | `'quadratic'`

Conduction method, specified as the comma-separated pair consisting of `'ConductionMethod'` and `'exponential'` or `'quadratic'`. Exponential diffusion favors high-contrast edges over low-contrast edges. Quadratic diffusion favors wide regions over smaller regions.

Output Arguments

`gradientThreshold` — Gradient threshold
numeric vector

Gradient threshold, returned as a numeric vector of the same data type as the input image, `I`. The length of the vector is equal to `numberOfIterations`.

`numberOfIterations` — Number of iterations
positive integer

Number of iterations to use in the diffusion process, returned as a positive integer.

References

[1] Perona, P., and J. Malik. "Scale-space and edge detection using anisotropic diffusion." *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 12, No. 7, July 1990, pp. 629–639.

[2] Tsiotsios, C., and M. Petrou. "On the choice of the parameters for anisotropic diffusion in image processing." *Pattern Recognition*. Vol. 46, No. 5, May 2013, pp. 1369–1381.

See Also
`imdiffusefilt`

Introduced in R2018a

imdiffusefilt

Anisotropic diffusion filtering of images

Syntax

```
J = imdiffusefilt(I)
J = imdiffusefilt(I,Name,Value)
```

Description

`J = imdiffusefilt(I)` applies anisotropic diffusion filtering to image `I` and returns the result in `J`.

`J = imdiffusefilt(I,Name,Value)` uses name-value pairs to change the behavior of the anisotropic diffusion algorithm.

Examples

Perform Edge-Preserving Smoothing Using Anisotropic Diffusion

Read an image into the workspace and display it.

```
I = imread('cameraman.tif');
imshow(I)
title('Original Image')
```

Smooth the image using anisotropic diffusion. For comparison, also smooth the image using Gaussian blurring. Adjust the standard deviation `sigma` of the Gaussian smoothing kernel so that textured regions, such as the grass, are smoothed a similar amount for both methods.

```
Idiffusion = imdiffusefilt(I);
sigma = 1.2;
Igaussian = imgaussfilt(I,sigma);
```

Display the results.

```
montage({Idiffusion,Igaussian},'ThumbnailSize',[])
title('Smoothing Using Anisotropic Diffusion (Left) vs. Gaussian Blurring (Right)')
```

Anisotropic diffusion preserves the sharpness of edges better than Gaussian blurring.

Perform Edge-Aware Noise Reduction Using Anisotropic Diffusion

Read a grayscale image, then apply strong Gaussian noise to it. Display the noisy image.

```
I = imread('pout.tif');
noisyImage = imnoise(I,'gaussian',0,0.005);
imshow(noisyImage)
title('Noisy Image')
```

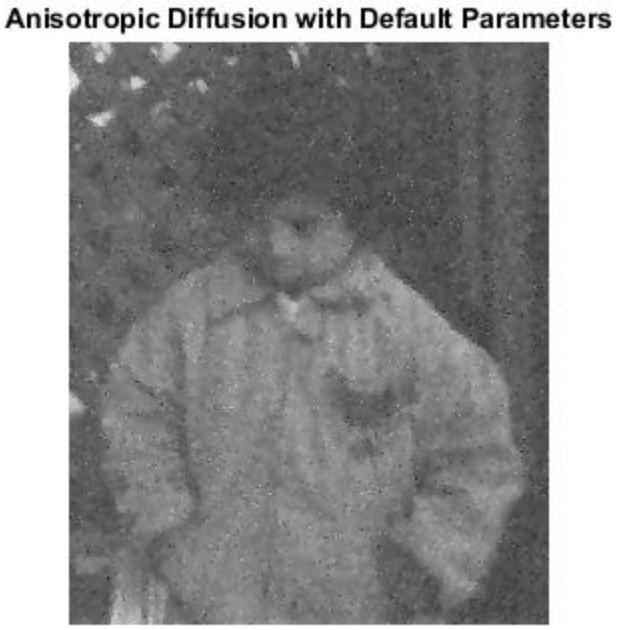
Compute the structural similarity index (SSIM) to measure the quality of the noisy image. The closer the SSIM value is to 1, the better the image agrees with the noiseless reference image.

```
n = ssim(I,noisyImage);
disp(['The SSIM value of the noisy image is ',num2str(n),'.'])
```

```
The SSIM value of the noisy image is 0.26556.
```

Reduce the noise using anisotropic diffusion. First, try the default parameters for the anisotropic diffusion filter, and display the result.

```
B = imdiffusefilt(noisyImage);
imshow(B)
title('Anisotropic Diffusion with Default Parameters')
```




```
nB = ssim(I,B);
disp(['The SSIM value using default anisotropic diffusion is ',num2str(nB),'.'])
```

```
The SSIM value using default anisotropic diffusion is 0.65665.
```

The image is still degraded by noise, so refine the filter. Choose the quadratic conduction method because the image is characterized more by wide homogenous regions than by high-contrast edges. Estimate the optimal gradient threshold and number of iterations by using the `imdiffuseest` function. Display the resulting image.

```
[gradThresh,numIter] = imdiffuseest(noisyImage,'ConductionMethod','quadratic');
C = imdiffusefilt(noisyImage,'ConductionMethod','quadratic', ...
    'GradientThreshold',gradThresh,'NumberOfIterations',numIter);
imshow(C)
title('Anisotropic Diffusion with Estimated Parameters')
```


```
nC = ssim(I,C);
disp(['The SSIM value using quadratic anisotropic diffusion is ',num2str(nC),'.'])
```

```
The SSIM value using quadratic anisotropic diffusion is 0.88135.
```

Noise is less apparent in the resulting image. The SSIM value, which is closer to 1, confirms that the quality of the image has improved.

Perform 3-D Edge-Aware Noise Reduction

Load a noisy 3-D grayscale MRI volume.

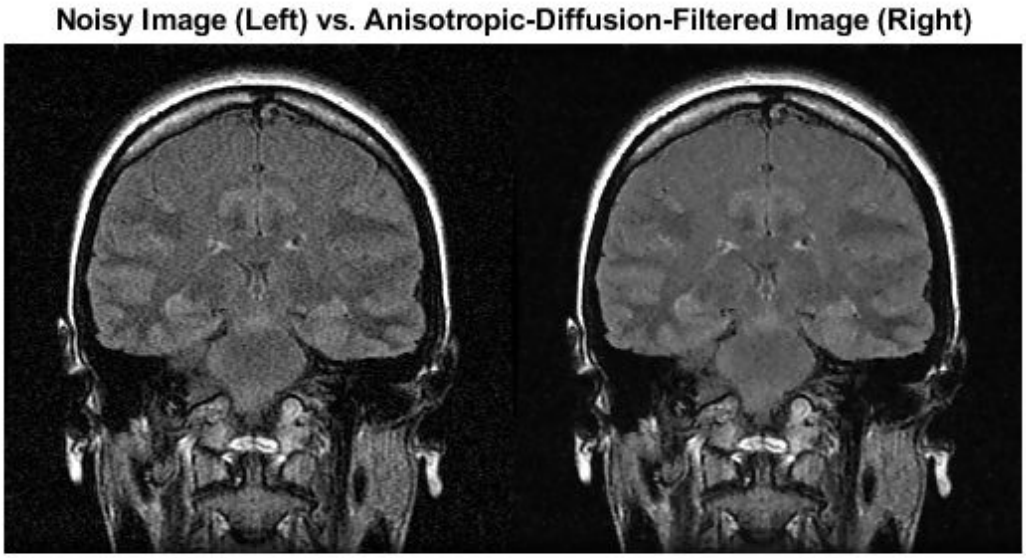
```
load mristack
```

Perform edge-aware noise reduction on the volume using anisotropic diffusion. To prevent over-smoothing the low-contrast features in the brain, decrease the number of iterations from the default number, 5. The tradeoff is that less noise is removed.

```
diffusedImage = imdiffusefilt(mristack,'NumberOfIterations',3);
```

To compare the noisy image and the filtered image in detail, display the tenth slice of both.

```
imshowpair(mristack(:,:,10),diffusedImage(:,:,10),'montage')
title('Noisy Image (Left) vs. Anisotropic-Diffusion-Filtered Image (Right)')
```



Calculate the Naturalness Image Quality Evaluator (NIQE) score averaged over all slices in the volume. The NIQE score provides a quantitative measure of image quality that does not require a reference image. Lower NIQE scores reflect better perceptual image quality.

```
nframes = size(mristack,3);
m = 0;
d = 0;
for i = 1:nframes
    m = m + niqe(mristack(:,:,i));
    d = d + niqe(diffusedImage(:,:,i));
end
mAvg = m/nframes;
dAvg = d/nframes;
disp(['The NIQE score of the noisy volume is ',num2str(mAvg),'.'])
```

```
The NIQE score of the noisy volume is 5.7794.
```

```
disp(['The NIQE score using anisotropic diffusion is ',num2str(dAvg),'.'])
```

```
The NIQE score using anisotropic diffusion is 4.1391.
```

The NIQE score is consistent with the observation of reduced noise in the filtered image.

Input Arguments

`I` — Image to filter

2-D grayscale image | 3-D grayscale volume

Image to filter, specified as a 2-D grayscale image of size *m*-by-*n* or a 3-D grayscale volume of size *m*-by-*n*-by-*k*.

Note To apply anisotropic diffusion filtering to a color image, use `imdiffusefilt` on each color channel independently.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `imdiffusefilt(I,'NumberOfIterations',4,'Connectivity','minimal')` performs anisotropic diffusion on image `I`, using 4 iterations and minimal connectivity.

`GradientThreshold` — Gradient threshold

numeric scalar | numeric vector

Gradient threshold, specified as the comma-separated pair consisting of `'GradientThreshold'` and a numeric scalar or a numeric vector of length `NumberOfIterations`. The value of `GradientThreshold` controls the conduction process by classifying gradient values as an actual edge or as noise. Increasing the value of `GradientThreshold` smooths the image more. The default value is 10% of the dynamic range of the image. You can use the `imdiffuseest` function to estimate a suitable value of `GradientThreshold`.

`NumberOfIterations` — Number of iterations

5 (default) | positive integer

Number of iterations to use in the diffusion process, specified as the comma-separated pair consisting of `'NumberOfIterations'` and a positive integer. You can use the `imdiffuseest` function to estimate a suitable value of `NumberOfIterations`.

`Connectivity` — Connectivity

`'maximal'` (default) | `'minimal'`

Connectivity of a pixel to its neighbors, specified as the comma-separated pair consisting of `'Connectivity'` and one of these values:

- `'maximal'` — Considers 8 nearest neighbors for 2-D images, and 26 nearest neighbors for 3-D images
- `'minimal'` — Considers 4 nearest neighbors for 2-D images, and 6 nearest neighbors for 3-D images

`ConductionMethod` — Conduction method

`'exponential'` (default) | `'quadratic'`

Conduction method, specified as the comma-separated pair consisting of `'ConductionMethod'` and `'exponential'` or `'quadratic'`. Exponential diffusion favors high-contrast edges over low-contrast edges. Quadratic diffusion favors wide regions over smaller regions.

Output Arguments

J — Diffusion-filtered image

numeric array

Diffusion-filtered image, returned as a numeric array of the same size and data type as the input image, `I`.

References

[1] Perona, P., and J. Malik. "Scale-space and edge detection using anisotropic diffusion." *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 12, No. 7, July 1990, pp. 629–639.

[2] Gerig, G., O. Kubler, R. Kikinis, and F. A. Jolesz. "Nonlinear anisotropic filtering of MRI data." *IEEE Transactions on Medical Imaging*. Vol. 11, No. 2, June 1992, pp. 221–232.

See Also

`imdiffuseest` | `imfilter` | `imgaussfilt` | `imguidedfilter` | `locallapfilt` | `imnlmfilt`

Introduced in R2018a

imdilate

Dilate image

Syntax

```
J = imdilate(I,SE)
J = imdilate(I,nhood)
J = imdilate( ___ ,packopt)
J = imdilate( ___ ,shape)
```

Description

`J = imdilate(I,SE)` dilates the grayscale, binary, or packed binary image `I` using the structuring element `SE`.

`J = imdilate(I,nhood)` dilates the image `I`, where `nhood` is a matrix of `0`s and `1`s that specifies the structuring element neighborhood.

This syntax is equivalent to `imdilate(I,strel(nhood))`.

`J = imdilate( ___ ,packopt)` specifies whether `I` is a packed binary image.

`J = imdilate( ___ ,shape)` specifies the size of the output image.

Examples

Dilate Image with Vertical Line Structuring Element

Read a binary image into the workspace.

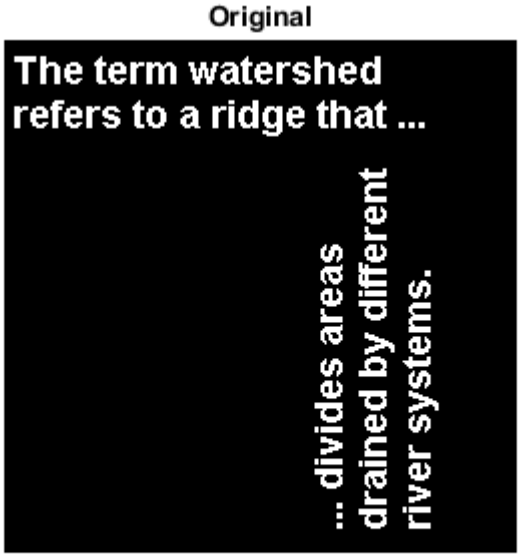
```
BW = imread('text.png');
```

Create a vertical line shaped structuring element.

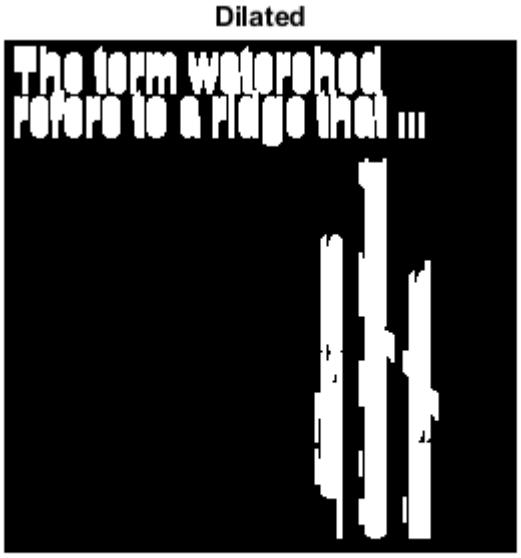
```
se = strel('line',11,90);
```

Dilate the image with a vertical line structuring element and compare the results.

```
BW2 = imdilate(BW,se);
imshow(BW), title('Original')
```




```
figure, imshow(BW2), title('Dilated')
```



Dilate Grayscale Image with Rolling Ball

Read a grayscale image into the workspace.

```
originalI = imread('cameraman.tif');
```

Create a nonflat ball-shaped structuring element.

```
se = offsetstrel('ball',5,5);
```

Dilate the image.

```
dilatedI = imdilate(originalI,se);
```

Display the original image and the dilated image.

```
imshowpair(originalI,dilatedI,'montage')
```

Determine Domain of Composition of Structuring Elements

Create two flat, line-shaped structuring elements, one at 0 degrees and the other at 90 degrees.

```
se1 = strel('line',3,0)
```

```
se1 =
strel is a line shaped structuring element with properties:

      Neighborhood: [1 1 1]
    Dimensionality: 2
```

```
se2 = strel('line',3,90)
```

```
se2 =
strel is a line shaped structuring element with properties:

      Neighborhood: [3x1 logical]
    Dimensionality: 2
```

Dilate the scalar value 1 with both structuring elements in sequence, using the `'full'` option.

```
composition = imdilate(1,[se1 se2],'full')
```

```
composition = 3×3

     1     1     1
     1     1     1
     1     1     1
```

Dilate Points in 3D Space Using Spherical Structuring Elements

Create a logical 3D volume with two points.

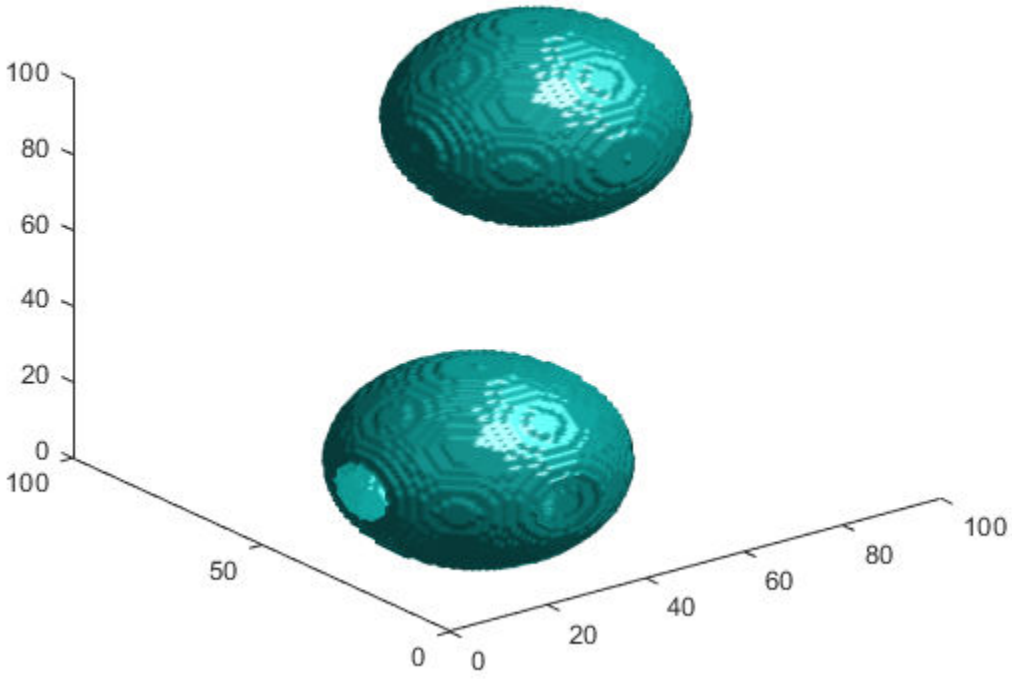
```
BW = false(100,100,100);
BW(25,25,25) = true;
BW(75,75,75) = true;
```

Dilate the 3D volume using a spherical structuring element.

```
se = strel('sphere',25);
dilatedBW = imdilate(BW,se);
```

Visualize the dilated image volume.

```
figure
isosurface(dilatedBW, 0.5)
```



Input Arguments

I — Input image

grayscale image | binary image | packed binary image

Input image, specified as a grayscale image, binary image, or packed binary image of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

SE — Structuring element

`strel` object | `offsetstrel` object | array of `strel` objects | array of `offsetstrel` objects

Structuring element, specified as a scalar `strel` object or `offsetstrel` object. `SE` can also be an array of `strel` object or `offsetstrel` objects, in which case `imdilate` performs multiple dilations of the input image, using each structuring element in succession.

`imdilate` performs grayscale dilation for all images except images of data type `logical`. In this case, the structuring element must be flat and `imdilate` performs binary dilation.

nhood — Structuring element neighborhood

matrix of `0`s and `1`s

Structuring element neighborhood, specified as a matrix of `0`s and `1`s.

Example: `[0 1 0; 1 1 1; 0 1 0]`

packopt — Indicator of packed binary image

`'notpacked'` (default) | `'packed'`

Indicator of packed binary image, specified as one of the following.

Value	Description
`'notpacked'`	`I` is treated as a normal array.
`'ispacked'`	`I` is treated as a packed binary image as produced by `bwpack`. `I` must be a 2-D `uint32` array and `SE` must be a flat 2-D structuring element. The value of `shape` must be `'same'`.

Data Types: `char` | `string`

shape — Size of output image

`'same'` (default) | `'full'`

Size of the output image, specified as one of the following.

Value	Description
`'same'`	The output image is the same size as the input image. If the value of `packopt` is `'ispacked'`, then `shape` must be `'same'`.
`'full'`	Compute the full dilation.

Data Types: `char` | `string`

Output Arguments

J — Dilated image

grayscale image | binary image | packed binary image

Dilated image, returned as a grayscale image, binary image, or packed binary image. If the input image `I` is packed binary, then `J` is also packed binary. `J` has the same data type as `I`.

More About

Binary Dilation

The *binary dilation* of A by B, denoted $A \oplus B$, is defined as the set operation:

$$A \oplus B = \left\{ z \middle| \left(\widehat{B}\right)_z \cap A \neq \varnothing \right\},$$

where $\widehat{B}$ is the reflection of the structuring element B. In other words, it is the set of pixel locations z, where the reflected structuring element overlaps with foreground pixels in A when translated to z. Note that some applications use a definition of dilation in which the structuring element is not reflected.

For more information about binary dilation, see [1] on page 1-1476.

Grayscale Dilation

In the general form of *grayscale dilation*, the structuring element has a height. The grayscale dilation of $A(x, y)$ by $B(x, y)$ is defined as:

$$(A \oplus B)(x, y) = \max\{A(x - x', y - y') + B(x', y') | (x', y') \in D_B\},$$

where D_B is the domain of the structuring element B and $A(x, y)$ is assumed to be $-\infty$ outside the domain of the image. Note that some applications define grayscale dilation using an equation with $A(x + x', y + y')$ instead of $A(x - x', y - y')$.

To create a structuring element with nonzero height values, use the syntax `strel(nhood,height)`, where `height` gives the height values and `nhood` corresponds to the structuring element domain, D_B.

Most commonly, grayscale dilation is performed with a flat structuring element ($B(x,y) = 0$). Grayscale dilation using such a structuring element is equivalent to a local-maximum operator:

$$(A \oplus B)(x, y) = \max\{A(x - x', y - y') | (x', y') \in D_B\}\ .$$

All of the `strel` syntaxes except for `strel(nhood,height)`, `strel('arbitrary',nhood,height)`, and `strel('ball',___)` produce flat structuring elements.

Tips

- If the dimensionality of the image `I` is greater than the dimensionality of the structuring element, then the `imdilate` function applies the same morphological dilation to all planes along the higher dimensions.

  You can use this behavior to perform morphological dilation on RGB images. Specify a 2-D structuring element for RGB images to operate on each color channel separately.
- When you specify a structuring element neighborhood, `imdilate` determines the center element of `nhood` by `floor((size(nhood)+1)/2)`.
- `imdilate` automatically takes advantage of the decomposition of a structuring element object (if it exists). Also, when performing binary dilation with a structuring element object that has a decomposition, `imdilate` automatically uses binary image packing to speed up the dilation [3].

References

[1] Gonzalez, Rafael C., Richard E. Woods, and Steven L. Eddins. *Digital Image Processing Using MATLAB*. Third edition. Knoxville: Gatesmark Publishing, 2020.

[2] Haralick, Robert M., and Linda G. Shapiro. *Computer and Robot Vision*. 1st ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1992, pp. 158-205.

[3] Boomgaard, Rein van den, and Richard van Balen. “Methods for Fast Morphological Image Transforms Using Bitmapped Binary Images.” *CVGIP: Graphical Models and Image Processing* 54, no. 3 (May 1, 1992): 252–58. https://doi.org/10.1016/1049-9652(92)90055-3.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imdilate` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imdilate` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see "Types of Code Generation Support in Image Processing Toolbox".
- The input image `I` must be 2-D or 3-D.
- The structuring element `SE` must be a single element—arrays of structuring elements are not supported. To obtain the same result as that obtained using an array of structuring elements, call the function sequentially.
- When the target is `MATLAB Host Computer`, the `packopt` and `shape` arguments must be compile-time constants. When the target is any other platform, the `packopt` syntax is not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input image `I` must be 2-D or 3-D.
- Packed binary input images (`packopt` syntax) are not supported.
- For 3-D input images with more than three channels, only C/C++ code is generated.
- The structuring element `SE` must be a compile-time constant. CUDA code is generated only for 1-D or 2-D structuring elements. If the structuring element is 3-D, C/C++ code is generated. Code generation is not supported for structuring elements with more than three dimensions.
- For non-flat structuring elements, only C/C++ code is generated.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- `gpuArray` input must be of type `uint8` or `logical`.
- The structuring element `SE` must be flat and 2-D.
- The `packopt` argument is not supported on the GPU.

For more information, see "Image Processing on a GPU".

See Also

Functions

`bwpack` | `bwunpack` | `conv2` | `filter2` | `imclose` | `imerode` | `imopen`

Objects

`strel` | `offsetstrel`

Introduced before R2006a

imdisplayrange

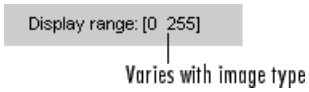
Display Range tool

Syntax

```
imdisplayrange
imdisplayrange(h)
imdisplayrange(hparent,himage)
htool = imdisplayrange( ___ )
```

Description

Use the `imdisplayrange` function to create a Display Range tool. The Display Range tool shows the display range of the grayscale image or images in the figure.



`imdisplayrange` creates a Display Range tool in the current figure.

`imdisplayrange(h)` creates a Display Range tool in the figure specified by the handle `h`.

`imdisplayrange(hparent,himage)` creates a Display Range tool in `hparent` that shows the display range of `himage`.

`htool = imdisplayrange( ___ )` returns a handle to the Display Range tool uipanel.

Examples

Create Display Range Tool

Display an image and include the Display Range tool.

```
imshow('bag.png');
imdisplayrange;
```

Import a 16-bit DICOM image and display it with its default range and scaled range in the same figure.

```
dcm = dicomread('CT-MONO2-16-ankle.dcm');
subplot(1,2,1), imshow(dcm);
subplot(1,2,2), imshow(dcm,[]);
imdisplayrange;
```

Input Arguments

h — Handle to graphics object

handle

Handle to a figure, axes, uipanel, or image graphics object, specified as a handle. Axes, uipanel, or figure objects must contain at least one image object.

`hparent` — Handle to figure or uipanel object
handle

Handle to a figure or uipanel object that contains the Display Range tool, specified as a handle.

`himage` — Handle to images
handle | array of handles

Handle to one or more images, specified as a handle or an array of image handles.

Output Arguments

`htool` — Handle to Display Range tool
handle

Handle to Display Range tool uipanel, returned as a handle.

Tips

- The Display Range tool is a uipanel object, positioned in the lower-right corner of the figure. It contains the label **Display range:** followed by the display range values for the image.
- For an indexed, truecolor, or binary image, the display range is not applicable and is set to empty (`[]`).
- The Display Range tool can work with multiple images in a figure. When the pointer is not in an image in a figure, the Display Range tool displays `[black white]`.

See Also

Image Viewer

Introduced before R2006a

imdivide

Divide one image into another or divide image by constant

Syntax

```
Z = imdivide(X,Y)
```

Description

`Z = imdivide(X,Y)` divides each element in the array `X` by the corresponding element in array `Y` and returns the result in the corresponding element of the output array `Z`.

Examples

Divide Two uint8 Arrays

This example shows how to divide two `uint8` arrays.

```
X = uint8([ 255 0 75; 44 225 100]);
Y = uint8([ 50 50 50; 50 50 50 ]);
```

Divide each element in `X` by the corresponding element in `Y`. Note that fractional values greater than or equal to 0.5 are rounded up to the nearest integer.

```
Z = imdivide(X,Y)
```

```
Z = 2x3 uint8 matrix

   5   0   2
   1   5   2
```

Divide each element in `Y` by the corresponding element in `X`. Note that when dividing by zero, the output is truncated to the range of the integer type.

```
W = imdivide(Y,X)
```

```
W = 2x3 uint8 matrix

     0   255     1
     1     0     1
```

Divide Image Background

Read a grayscale image into the workspace.

```
I = imread('rice.png');
```

Estimate the background.

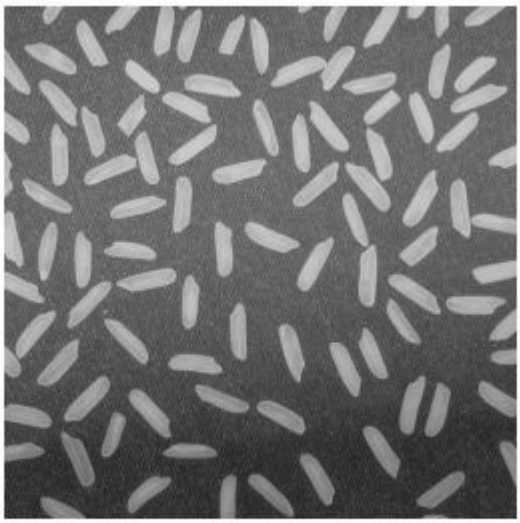
```
background = imopen(I,strel('disk',15));
```

Divide out the background from the image.

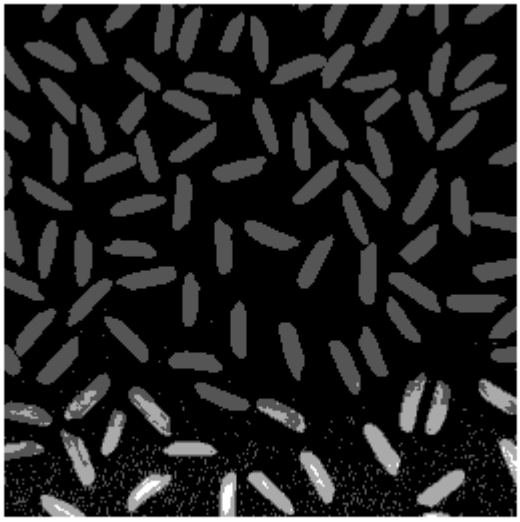
```
J = imdivide(I,background);
```

Display the original image and the processed image.

```
imshow(I)
```



```
figure
imshow(J,[])
```



Divide an Image by a Constant Factor

Read an image into the workspace.

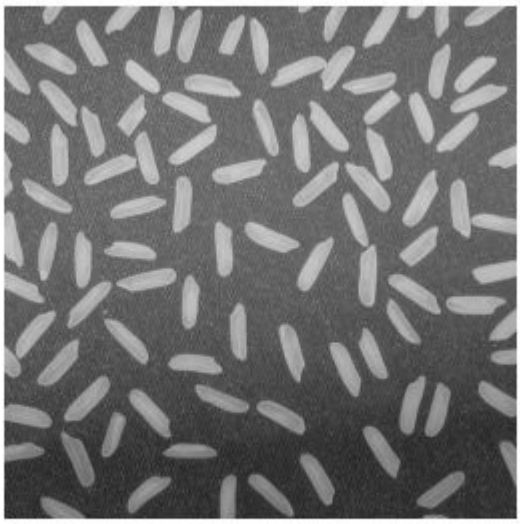
```
I = imread('rice.png');
```

Divide each value of the image by a constant factor of 2.

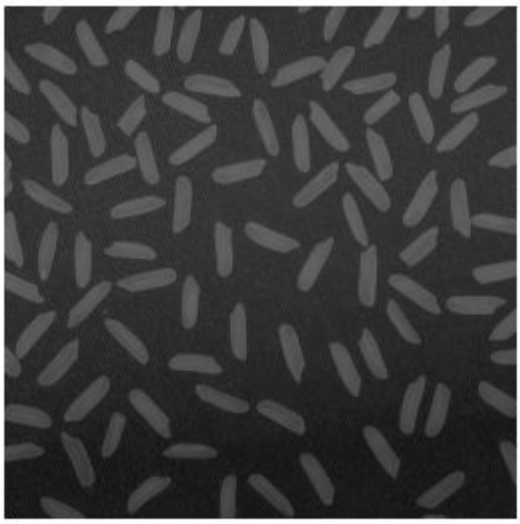
```
J = imdivide(I,2);
```

Display the original image and the processed image.

```
imshow(I)
```



```
figure
imshow(J)
```



Input Arguments

X — First array

numeric array | logical array

First array, specified as a numeric array or logical array of any dimension.

Y — Second array

numeric scalar | numeric array | logical array

Second array (divisor) to be divided from `X`, specified as a numeric or logical array of the same size and class as `X`, or a numeric scalar of type `double`.

Output Arguments

Z — Quotient

numeric array

Quotient, returned as a numeric array of the same size as `X`. `Z` is the same class as `X` unless `X` is logical, in which case `Z` is data type `double`. If `X` is an integer array, elements of the output that exceed the range of the integer type are truncated, and fractional values are rounded.

See Also

`imabsdiff` | `imadd` | `imcomplement` | `imlincomb` | `immultiply` | `imsubtract`

Introduced before R2006a

imerase

Remove image pixels within rectangular region of interest

Syntax

```
Ierased = imerase(I,rect)
Ierased = imerase(I,rect,'FillValues',fillValues)
```

Description

`Ierased = imerase(I,rect)` remove pixels of image `I` within the rectangular region defined by `rect` and returns the image with the erased region, `Ierased`.

`Ierased = imerase(I,rect,'FillValues',fillValues)` also specifies the fill value to apply to the erased pixels.

Examples

Erase Pixels from Random Window

Read and display an image.

```
I = imread("peppers.png");
imshow(I)
```

Select a rectangular region of size 50-by-100 pixels from a random location in the image.

```
rect = randomWindow2d(size(I),[50 100]);
```

Erase the pixels from within the rectangular region.

```
J = imerase(I,rect);
```

Display the erased image. The erased pixels have the value `0`.

```
imshow(J)
```

Erase Pixels from Specified Window

Read and display an image.

```
I = imread("car1.jpg");
imshow(I)
```

Specify the size and position of the erase rectangle as a 4-element vector of the form [*xmin ymin width height*].

```
rect = [1040 1525 250 200];
```

Erase the pixels from within the rectangular region, and fill the erased pixels with the color green.

```
J = imerase(I,rect,"FillValues",[0 255 0]);
```

Display the erased image.

```
imshow(J)
```

Fill Erased Region with Random Colors

Read and display a color image.

```
I = imread('flamingos.jpg');
imshow(I)
```

Select a random square window from the image. The area of the window is between 2% and 13% of the area of the entire image.

```
win = randomWindow2d(size(I),"Scale",[0.02 0.13],"DimensionRatio",[1 1;1 1]);
```

Determine the height and width of the erase region.

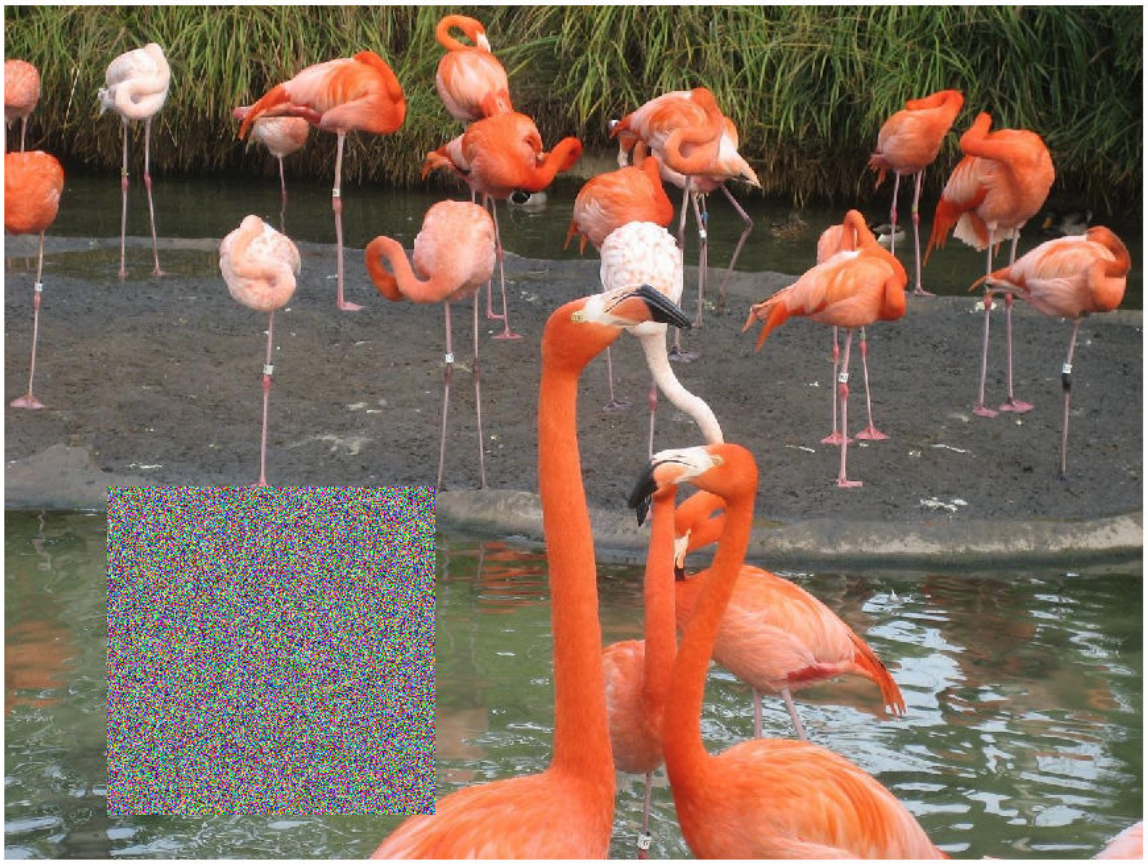
```
hwin = diff(win.YLimits)+1;
wwin = diff(win.XLimits)+1;
```

Erase the pixels within the erase region. Fill each pixel with a random color.

```
J = imerase(I,win,"FillValues",randi([1 255],[hwin wwin 3]));
```

Display the erased image.

```
imshow(J)
```



Input Arguments

`I` — Image with region to be erased

numeric matrix | numeric array

Image with a region to be erased, specified as a numeric matrix representing a grayscale image or a numeric array with three channels representing a color image.

`rect` — Size and position of erase rectangle

4-element numeric vector | `Rectangle` object

Size and position of the erase rectangle, specified as a 4-element numeric vector of the form [*xmin ymin width height*] or a `Rectangle` object.

`fillValues` — Fill value

`0` (default) | numeric scalar | 3-element numeric vector | numeric matrix | numeric array

Fill value to apply to erased pixels, specified as one of these values.

Fill Value	Result
numeric scalar	Fill erased pixels of a grayscale or RGB image with the specified gray value.
3-element numeric vector	Fill erased pixels of an RGB image with the specified color.
numeric matrix	Fill each erased pixel of a grayscale or RGB image with the corresponding gray value in `fillValue`. The matrix specified by `fillValue` must have the same height and width as the erase rectangle, `rect`.
numeric array with 3 planes	Fill each erased pixel of an RGB image with the color in the corresponding pixel of `fillValue`. The array specified by `fillValue` must have the same height and width as the erase rectangle, `rect`.

Output Arguments

`Ierased` — Image with erased region

numeric matrix | numeric array

Image with erased region, returned as a numeric matrix or numeric array of the same size as the input image, `I`.

See Also

`Rectangle` | `imcrop`

Introduced in R2021a

imerode

Erode image

Syntax

```
J = imerode(I,SE)
J = imerode(I,nhood)
J = imerode( ___ ,packopt,m)
J = imerode( ___ ,shape)
```

Description

`J = imerode(I,SE)` erodes the grayscale, binary, or packed binary image `I` using the structuring element `SE`.

`J = imerode(I,nhood)` erodes the image `I`, where `nhood` is a matrix of `0`s and `1`s that specifies the structuring element neighborhood.

This syntax is equivalent to `imerode(I,strel(nhood))`.

`J = imerode( ___ ,packopt,m)` specifies whether input image `I` is a packed binary image. `m` specifies the row dimension of the original unpacked image.

`J = imerode( ___ ,shape)` specifies the size of the output image.

Examples

Erode Binary Image with Line Structuring Element

Read binary image into the workspace.

```
originalBW = imread('text.png');
```

Create a flat, line-shaped structuring element.

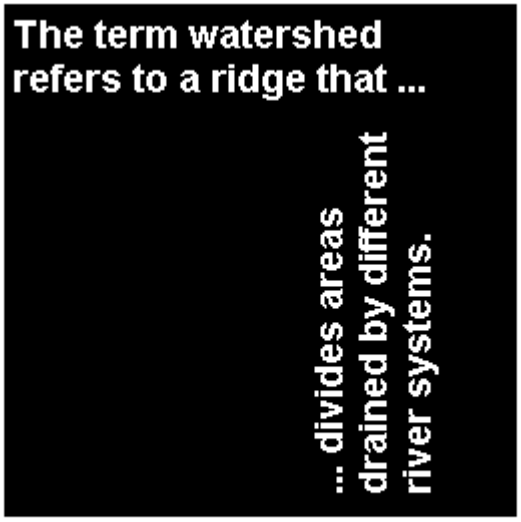
```
se = strel('line',11,90);
```

Erode the image with the structuring element.

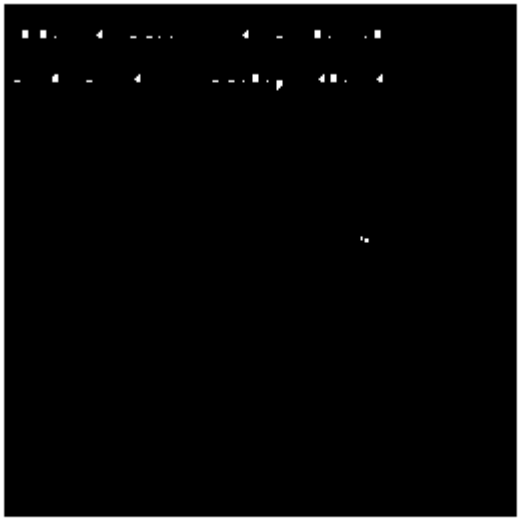
```
erodedBW = imerode(originalBW,se);
```

View the original image and the eroded image.

```
figure
imshow(originalBW)
```




```
figure
imshow(erodedBW)
```



Erode Grayscale Image with Rolling Ball

Read grayscale image into the workspace.

```
originalI = imread('cameraman.tif');
```

Create a nonflat `offsetstrel` object.

```
se = offsetstrel('ball',5,5);
```

Erode the image.

```
erodedI = imerode(originalI,se);
```

Display original image and eroded image.

```
figure
imshow(originalI)
```

```
figure
imshow(erodedI)
```

Erode MRI Stack Volume Using Cubic Structuring Element

Create a binary volume.

```
load mristack
BW = mristack < 100;
```

Create a cubic structuring element.

```
se = strel('cube',3)
```

```
se =
strel is a cube shaped structuring element with properties:

      Neighborhood: [3x3x3 logical]
    Dimensionality: 3
```

Erode the volume with a cubic structuring element.

```
erodedBW = imerode(BW, se);
```

Input Arguments

I — Input image

grayscale image | binary image | packed binary image

Input image, specified as a grayscale image, binary image, or packed binary image of any dimension.

Data Types: single | double | int8 | int16 | int32 | uint8 | uint16 | uint32 | logical

SE — Structuring element

strel object | offsetstrel object | array of strel objects | array of offsetstrel objects

Structuring element, specified as a scalar strel object or offsetstrel object. SE can also be an array of strel object or offsetstrel objects, in which case imerode performs multiple erosions of the input image, using each structuring element in succession.

imerode performs grayscale erosion for all images except images of data type logical. In this case, the structuring element must be flat and imerode performs binary erosion.

nhood — Structuring element neighborhood

matrix of 0s and 1s

Structuring element neighborhood, specified as a matrix of 0s and 1s.

Example: [0 1 0; 1 1 1; 0 1 0]

packopt — Indicator of packed binary image

'notpacked' (default) | 'packed'

Indicator of packed binary image, specified as one of the following.

Value	Description
'notpacked'	I is treated as a normal array.
'ispacked'	I is treated as a packed binary image as produced by bwpack. I must be a 2-D uint32 array and SE must be a flat 2-D structuring element. The value of shape must be 'same'.

Data Types: char | string

m — Row dimension of original unpacked image

positive integer

Row dimension of the original unpacked image, specified as a positive integer.

Data Types: double

shape — Size of output image

'same' (default) | 'full'

Size of the output image, specified as one of the following.

Value	Description
'same'	The output image is the same size as the input image. If the value of packopt is 'ispacked', then shape must be 'same'.
'full'	Compute the full erosion.

Data Types: char | string

Output Arguments

J — Eroded image

grayscale image | binary image | packed binary image

Eroded image, returned as a grayscale image, binary image, or packed binary image. If the input image `I` is packed binary, then `J` is also packed binary. `J` has the same data type as `I`.

More About

Binary Erosion

The *binary erosion* of A by B, denoted $A \ominus B$, is defined as the set operation $A \ominus B = \{z|(B_z \subseteq A\}$. In other words, it is the set of pixel locations z, where the structuring element translated to location z overlaps only with foreground pixels in A.

For more information on binary erosion, see [1] on page 1-1500.

Grayscale Erosion

In the general form of *grayscale erosion*, the structuring element has a height. The grayscale erosion of $A(x, y)$ by $B(x, y)$ is defined as:

$$(A \ominus B)(x, y) = \min \{A(x + x', y + y') - B(x', y') \mid (x', y') \in D_B\},$$

D_B is the domain of the structuring element B and $A(x,y)$ is assumed to be $+\infty$ outside the domain of the image. To create a structuring element with nonzero height values, use the syntax `strel(nhood,height)`, where `height` gives the height values and `nhood` corresponds to the structuring element domain, D_B.

Most commonly, grayscale erosion is performed with a flat structuring element ($B(x,y) = 0$). Grayscale erosion using such a structuring element is equivalent to a local-minimum operator:

$$(A \ominus B)(x, y) = \min \{A(x + x', y + y') \mid (x', y') \in D_B\}.$$

All of the `strel` syntaxes except for `strel(nhood,height)`, `strel('arbitrary',nhood,height)`, and `strel('ball', ...)` produce flat structuring elements.

Tips

- If the dimensionality of the image `I` is greater than the dimensionality of the structuring element, then the `imerode` function applies the same morphological erosion to all planes along the higher dimensions.

  You can use this behavior to perform morphological erosion on RGB images. Specify a 2-D structuring element for RGB images to operate on each color channel separately.
- When you specify a structuring element neighborhood, `imerode` determines the center element of `nhood` by `floor((size(nhood)+1)/2)`.
- `imerode` automatically takes advantage of the decomposition of a structuring element object (if it exists). Also, when performing binary erosion with a structuring element object that has a decomposition, `imerode` automatically uses binary image packing to speed up the erosion [3].

References

[1] Gonzalez, Rafael C., Richard E. Woods, and Steven L. Eddins. *Digital Image Processing Using MATLAB*. Third edition. Knoxville: Gatesmark Publishing, 2020.

[2] Haralick, Robert M., and Linda G. Shapiro. *Computer and Robot Vision*. 1st ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1992, pp. 158-205.

[3] Boomgaard, Rein van den, and Richard van Balen. "Methods for Fast Morphological Image Transforms Using Bitmapped Binary Images." *CVGIP: Graphical Models and Image Processing* 54, no. 3 (May 1, 1992): 252–58. https://doi.org/10.1016/1049-9652(92)90055-3.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imerode` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imerode` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see "Types of Code Generation Support in Image Processing Toolbox".
- The input image, `I`, must be 2-D or 3-D.
- The structuring element argument `SE` must be a single element——arrays of structuring elements are not supported. To obtain the same result as that obtained using an array of structuring elements, call the function sequentially.
- When the target is `MATLAB Host Computer`, the `packopt` and `shape` arguments must be compile-time constants. When the target is any other platform, the `packopt` syntax is not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input image `I` must be 2-D or 3-D.
- Packed binary input images (`packopt` syntax) are not supported.
- For 3-D input images with more than three channels, only C/C++ code is generated.
- The structuring element argument `SE` must be a compile-time constant. CUDA code is generated only for 1-D or 2-D structuring elements. If the structuring element is 3-D, C/C++ code is generated. Code generation is not supported for structuring elements with more than three dimensions.
- For non-flat structuring elements, only C/C++ code is generated.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- `gpuArray` input must be of type `uint8` or `logical`
- The structuring element `SE` must be flat and 2-D.
- The `packopt` argument is not supported on the GPU.

For more information, see “Image Processing on a GPU”.

See Also

Functions
`bwpack` | `bwunpack` | `conv2` | `filter2` | `imclose` | `imdilate` | `imopen`

Objects
`strel` | `offsetstrel`

Introduced before R2006a

imextendedmax

Extended-maxima transform

Syntax

```
BW = imextendedmax(I,H)
BW = imextendedmax(I,H,conn)
```

Description

`BW = imextendedmax(I,H)` returns the extended-maxima transform for `I`, which is the regional maxima of the `H`-maxima transform. Regional maxima are connected components of pixels with a constant intensity value, and whose external boundary pixels all have a lower value.

`BW = imextendedmax(I,H,conn)` computes the extended-maxima transform, where `conn` specifies the pixel connectivity.

Examples

Perform Extended-Maxima transform

Read image into workspace.

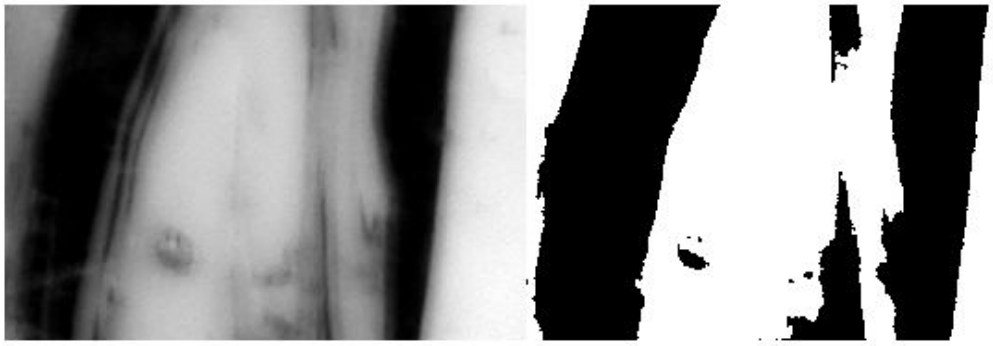
```
I = imread('glass.png');
```

Calculate the extended-maxima transform.

```
BW = imextendedmax(I,80);
```

Display original image and transformed image side-by-side.

```
imshowpair(I,BW,'montage')
```



Input Arguments

`I` — Input image

numeric array

Input image, specified as a numeric array of any dimension.

Example: `I = imread('glass.png');`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`H` — H-maxima transform

nonnegative scalar

H-maxima transform, specified as a nonnegative scalar.

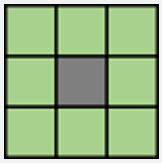
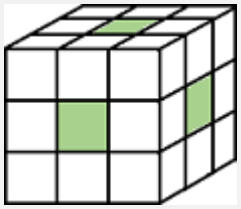
Example: `BW = imextendedmax(I,80);`

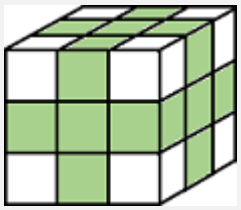
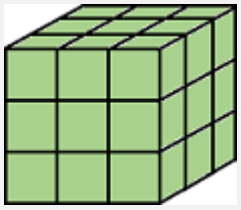
Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`conn` — Pixel connectivity

4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 matrix of `0`s and `1`s

Pixel connectivity, specified as one of the values in this table. The default connectivity is `8` for 2-D images, and `26` for 3-D images.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. The neighborhood of a pixel are the adjacent pixels in the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. The neighborhood of a pixel are the adjacent pixels in the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		
6	Pixels are connected if their faces touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down	 Current pixel is shown in gray.

Value	Meaning	
18	Pixels are connected if their faces or edges touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `imextendedmax` uses the default value `conndef(ndims(I),'maximal')`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`BW` — Transformed image

logical array

Transformed image, returned as a logical array the same size as `I`.

References

[1] Soille, P. *Morphological Image Analysis: Principles and Applications*. Springer-Verlag, 1999, pp. 170-171.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imextendedmax` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imextendedmax` generates code

that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

- When generating code, the optional third input argument, `conn`, must be a compile-time constant.

See Also

`conndef` | `imextendedmin` | `imhmax` | `imreconstruct` | `imregionalmax`

Introduced before R2006a

imextendedmin

Extended-minima transform

Syntax

```
BW = imextendedmin(I,H)
BW = imextendedmin(I,H,conn)
```

Description

`BW = imextendedmin(I,H)` computes the extended-minima transform, which is the regional minima of the `H`-minima transform. Regional minima are connected components of pixels with a constant intensity value, and whose external boundary pixels all have a higher value. `h` is a nonnegative scalar.

`BW = imextendedmin(I,H,conn)` computes the extended-minima transform, which is the regional minima of the H-minima transform.

Examples

Perform Extended-Minima transform

Read image into the workspace.

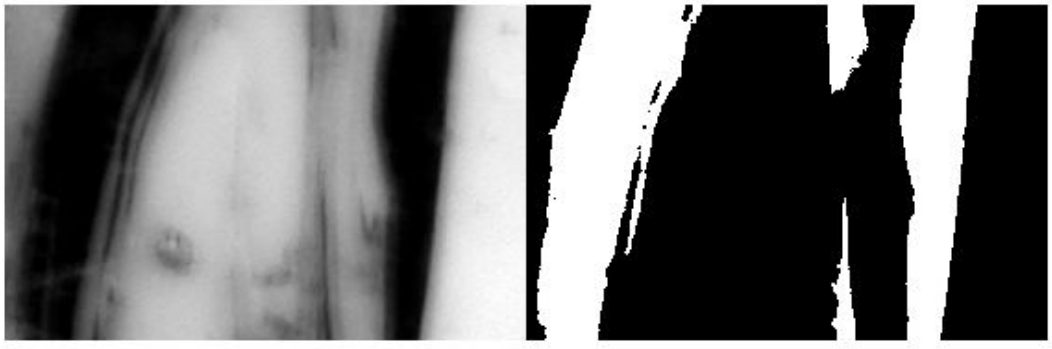
```
I = imread('glass.png');
```

Calculate the extended-minima transform.

```
BW = imextendedmin(I,50);
```

Display the original image and the transformation side-by-side.

```
imshowpair(I,BW,'montage');
```



Input Arguments

`I` — Input image

numeric array

Input array, specified as a numeric array of any dimension.

Example: `I = imread('glass.png');`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`H` — H-minima transform

nonnegative scalar

H-minima transform, specified as a nonnegative scalar.

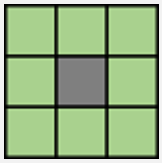
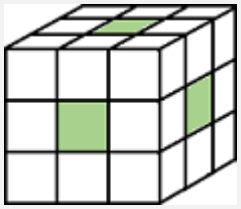
Example: `BW = imextendedmin(I,80);`

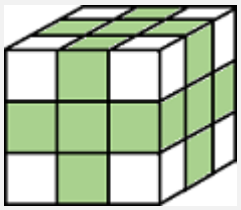
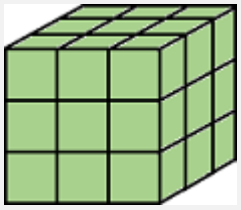
Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`conn` — Pixel connectivity

4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 matrix of `0`s and `1`s

Pixel connectivity, specified as one of the values in this table. The default connectivity is `8` for 2-D images, and `26` for 3-D images.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. The neighborhood of a pixel are the adjacent pixels in the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. The neighborhood of a pixel are the adjacent pixels in the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		
6	Pixels are connected if their faces touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down	 Current pixel is shown in gray.

Value	Meaning	
18	Pixels are connected if their faces or edges touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `imextendedmin` uses the default value `conndef(ndims(I),'maximal')`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`BW` — Transformed image

logical array

Transformed image, returned as a logical array the same size as `I`.

References

[1] Soille, P. *Morphological Image Analysis: Principles and Applications*. Springer-Verlag, 1999, pp. 170-171.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imextendedmin` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imextendedmin` generates code

that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

- When generating code, the optional third input argument, `conn`, must be a compile-time constant.

See Also

`conndef` | `imextendedmax` | `imhmin` | `imreconstruct` | `imregionalmin`

Introduced before R2006a

imfill

Fill image regions and holes

Syntax

```
BW2 = imfill(BW,locations)
BW2 = imfill(BW,locations,conn)
BW2 = imfill(BW,'holes')
BW2 = imfill(BW,conn,'holes')

I2 = imfill(I)
I2 = imfill(I,conn)

BW2 = imfill(BW)
BW2 = imfill(BW,0,conn)
[BW2, locations_out] = imfill(BW)
```

Description

`BW2 = imfill(BW,locations)` performs a flood-fill operation on background pixels of the input binary image `BW`, starting from the points specified in `locations`.

`BW2 = imfill(BW,locations,conn)` fills the area defined by `locations`, where `conn` specifies the connectivity.

`BW2 = imfill(BW,'holes')` fills holes in the input binary image `BW`. In this syntax, a hole is a set of background pixels that cannot be reached by filling in the background from the edge of the image.

`BW2 = imfill(BW,conn,'holes')` fills holes in the binary image `BW`, where `conn` specifies the connectivity.

`I2 = imfill(I)` fills holes in the grayscale image `I`. In this syntax, a hole is defined as an area of dark pixels surrounded by lighter pixels.

`I2 = imfill(I,conn)` fills holes in the grayscale image `I`, where `conn` specifies the connectivity.

`BW2 = imfill(BW)` displays the binary image `BW` on the screen and lets you define the region to fill by selecting points interactively with the mouse. To use this syntax, `BW` must be a 2-D image.

Press **Backspace** or **Delete** to remove the previously selected point. Shift-click, right-click, or double-click to select a final point and start the fill operation. Press **Return** to finish the selection without adding a point.

`BW2 = imfill(BW,0,conn)` lets you override the default connectivity as you interactively specify locations.

`[BW2, locations_out] = imfill(BW)` returns the locations of points selected interactively in `locations_out`. To use this syntax, `BW` must be a 2-D image.

Examples

Fill Image from Specified Starting Point

```
BW1 = logical([1 0 0 0 0 0 0 0
               1 1 1 1 1 0 0 0
               1 0 0 0 1 0 1 0
               1 0 0 0 1 1 1 0
               1 1 1 1 0 1 1 1
               1 0 0 1 1 0 1 0
               1 0 0 0 1 0 1 0
               1 0 0 0 1 1 1 0]);

BW2 = imfill(BW1,[3 3],8)
```

```
BW2 = 8x8 logical array

   1   0   0   0   0   0   0   0
   1   1   1   1   1   0   0   0
   1   1   1   1   1   0   1   0
   1   1   1   1   1   1   1   0
   1   1   1   1   1   1   1   1
   1   0   0   1   1   1   1   0
   1   0   0   0   1   1   1   0
   1   0   0   0   1   1   1   0
```

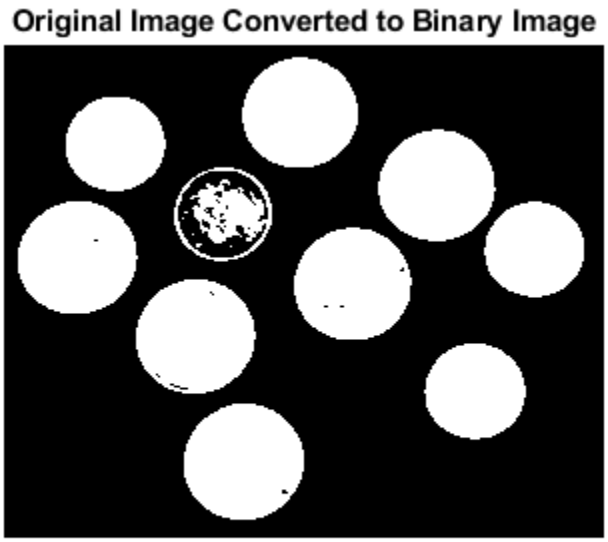
Fill Holes in a Binary Image

Read image into workspace.

```
I = imread('coins.png');
figure
imshow(I)
title('Original Image')
```

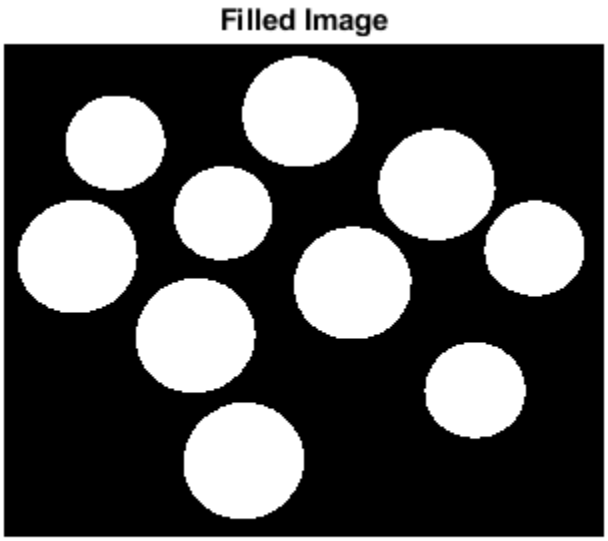
Convert image to binary image.

```
BW = imbinarize(I);
figure
imshow(BW)
title('Original Image Converted to Binary Image')
```



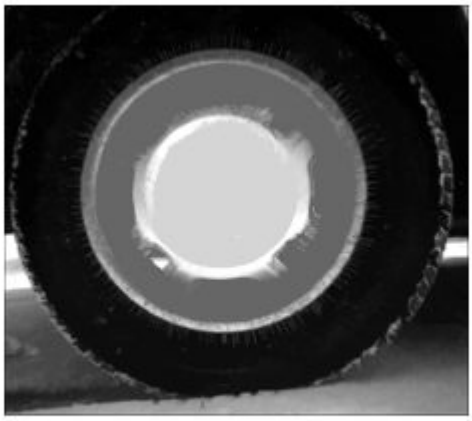
Fill holes in the binary image and display the result.

```
BW2 = imfill(BW,'holes');
figure
imshow(BW2)
title('Filled Image')
```



Fill Holes in a Grayscale Image

```
I = imread('tire.tif');
I2 = imfill(I);
figure, imshow(I), figure, imshow(I2)
```



Input Arguments

`BW` — Binary image

logical array

Binary image, specified as a logical array of any dimension.

Example: `BW = imread('text.png');`

Data Types: `logical`

`locations` — Linear indices identifying pixel locations

numeric vector of positive integers | 2-D numeric matrix of positive integers

Linear indices identifying pixel locations, specified as a numeric vector or 2-D numeric matrix of positive integers. If `locations` is a *p*-by-1 vector, then it contains the linear indices of the starting locations. If `locations` is a *p*-by-`ndims(BW)` matrix, then each row contains the array indices of one of the starting locations.

Example: `[3 3]`

Data Types: `double`

`I` — Grayscale image

numeric array

Grayscale image, specified as a numeric array of any dimension.

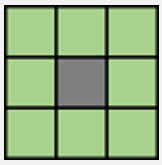
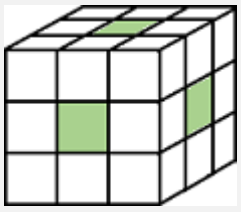
Example: `I = imread('cameraman.tif');`

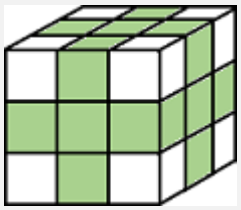
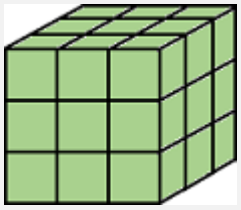
Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`conn` — Pixel connectivity

4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 matrix of `0`s and `1`s

Pixel connectivity, specified as one of the values in this table. The default connectivity is `4` for 2-D images, and `6` for 3-D images.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. The neighborhood of a pixel are the adjacent pixels in the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. The neighborhood of a pixel are the adjacent pixels in the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		
6	Pixels are connected if their faces touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down	 Current pixel is shown in gray.

Value	Meaning	
18	Pixels are connected if their faces or edges touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `imfill` uses the default value `conndef(ndims(BW),'minimal')`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Data Types: `double` | `logical`

Output Arguments

`BW2` — Filled binary image

logical array

Filled binary image, returned as logical array.

`locations_out` — Linear indices of pixel locations

numeric vector | numeric matrix

Linear indices of pixel locations, returned as a numeric vector or matrix.

`I2` — Filled grayscale image

numeric array

Filled grayscale image, returned as a numeric array.

Algorithms

`imfill` uses an algorithm based on morphological reconstruction [1].

References

[1] Soille, P., *Morphological Image Analysis: Principles and Applications*, Springer-Verlag, 1999, pp. 173–174.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imfill` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imfill` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- The optional input arguments, `conn` and `'holes'`, must be compile-time constants.
- Input argument `'holes'` is not supported, if the input is a binary image.
- `imfill` supports up to 3-D inputs only. (No N-D support.)
- The interactive syntaxes to select points are not supported. For example, the syntax `imfill(BW,0,CONN)` is not supported.
- With the `locations` input argument, once you select a format at compile time, you cannot change it at run time. However, the number of points in locations can be varied at run time.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Inputs must be 2-D, supporting only the 2-D connectivities (4 and 8).
- The interactive syntaxes to select points are not supported. For example, the syntax `imfill(BW)` is not supported.

For more information, see “Image Processing on a GPU”.

See Also

`conndef` | `bwselect` | `imreconstruct` | `regionfill`

Introduced before R2006a

imfilter

N-D filtering of multidimensional images

Syntax

```
B = imfilter(A,h)
B = imfilter(A,h,options,...)
```

Description

`B = imfilter(A,h)` filters the multidimensional array `A` with the multidimensional filter `h` and returns the result in `B`.

`B = imfilter(A,h,options,...)` performs multidimensional filtering according to one or more specified options.

Examples

Create Filter and Apply It

Read a color image into the workspace and display it.

```
originalRGB = imread('peppers.png');
imshow(originalRGB)
```

Create a motion-blur filter using the `fspecial` function.

```
h = fspecial('motion', 50, 45);
```

Apply the filter to the original image to create an image with motion blur. Note that `imfilter` is more memory efficient than some other filtering functions in that it outputs an array of the same data type as the input image array. In this example, the output is an array of `uint8`.

```
filteredRGB = imfilter(originalRGB, h);
figure, imshow(filteredRGB)
```

Filter the image again, this time specifying the replicate boundary option.

```
boundaryReplicateRGB = imfilter(originalRGB, h, 'replicate');
figure, imshow(boundaryReplicateRGB)
```

Filter Images Using imfilter with Convolution

By default, `imfilter` uses correlation because the toolbox filter design functions produce correlation kernels. Use the optional parameter to use convolution.

Create a sample matrix.

```
A = magic(5)
```

```
A = 5×5

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

Create a filter.

```
h = [-1 0 1];
```

Filter using correlation, the default.

```
imfilter(A,h)
```

```
ans = 5×5

    24   -16   -16    14    -8
     5   -16     9     9   -14
     6     9    14     9   -20
    12     9     9   -16   -21
    18    14   -16   -16    -2
```

Filter using convolution, specifying `imfilter` with the optional parameter.

```
imfilter(A,h,'conv')
```

```
ans = 5×5

   -24    16    16   -14     8
    -5    16    -9    -9    14
    -6    -9   -14    -9    20
   -12    -9    -9    16    21
   -18   -14    16    16     2
```

Convert Image Class to Avoid Negative Output Values

In this example, the output of `imfilter` has negative values when the input is of class `double`. To avoid negative values, convert the image to a different data type before calling `imfilter`. For example, when the input type is `uint8`, `imfilter` truncates output values to `0`. It might also be appropriate to convert the image to a signed integer type.

```
A = magic(5)
```

```
A = 5×5

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

Filter the image with `imfilter`.

```
h = [-1 0 1];
imfilter(A,h)
```

```
ans = 5×5

    24   -16   -16    14    -8
     5   -16     9     9   -14
     6     9    14     9   -20
    12     9     9   -16   -21
    18    14   -16   -16    -2
```

Notice that the result has negative values. To avoid negative values in the output image, convert the input image to `uint8` before performing the filtering. Since the input to `imfilter` is of class `uint8`, the output also is of class `uint8`, and `imfilter` truncates negative values to `0`.

```
A = uint8(magic(5));
imfilter(A,h)
```

```
ans = 5x5 uint8 matrix

   24    0    0   14    0
    5    0    9    9    0
    6    9   14    9    0
   12    9    9    0    0
   18   14    0    0    0
```

Input Arguments

`A` — Image to be filtered

numeric array

Image to be filtered, specified as a numeric array of dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`h` — Multidimensional filter

N-D array of data type `double`

Multidimensional filter, specified as an N-D array of data type `double`.

Data Types: `double`

`options` — Options that control the filtering operation

character vector | string scalar | numeric scalar

Options that control the filtering operation, specified as a character vector, string scalar, or numeric scalar. The following table lists all supported options.

Boundary Options

Option	Description
Padding Options	
numeric scalar, X	Input array values outside the bounds of the array are assigned the value X. When no padding option is specified, the default is 0.
'symmetric'	Input array values outside the bounds of the array are computed by mirror-reflecting the array across the array border.
'replicate'	Input array values outside the bounds of the array are assumed to equal the nearest array border value.
'circular'	Input array values outside the bounds of the array are computed by implicitly assuming the input array is periodic.
Output Size	
'same'	The output array is the same size as the input array. This is the default behavior when no output size options are specified.
'full'	The output array is the full filtered result, and so is larger than the input array.
Correlation and Convolution Options	
'corr'	imfilter performs multidimensional filtering using correlation, which is the same way that filter2 performs filtering. When no correlation or convolution option is specified, imfilter uses correlation.
'conv'	imfilter performs multidimensional filtering using convolution.

Output Arguments

B — Filtered image

numeric array

Filtered image, returned as a numeric array of the same size and class as the input image, A.

Tips

- This function may take advantage of hardware optimization for data types uint8, uint16, int16, single, and double to run faster.

Algorithms

- The imfilter function computes the value of each output pixel using double-precision, floating-point arithmetic. If the result exceeds the range of the data type, then imfilter truncates the result to the allowed range of the data type. If it is an integer data type, then imfilter rounds fractional values.
- If you specify an even-sized kernel h, then the center of the kernel is floor((size(h) + 1)/2).

  For example, the center of 4-element filter [0.25 0.75 -0.75 -0.25] is the second element, 0.75. This filter gives identical results as filtering with the 5-element filter [0 0.25 0.75 -0.75 -0.25].

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imfilter` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imfilter` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, the input image, `A`, must be 2-D or 3-D. The value of the input argument, `options`, must be a compile-time constant.
- If you specify a large kernel `h`, a kernel that contains large values, or specify an image containing large values, you can see different results between MATLAB and generated code for floating point data types. This happens because of accumulation errors due to different algorithm implementations.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, the input image, `A`, must be 2-D or 3-D. The value of the input argument, `options`, must be a compile-time constant.
- If you specify a large kernel `h`, a kernel that contains large values, or specify an image containing large values, you can see different results between MATLAB and generated code for floating point data types. This happens because of accumulation errors due to different algorithm implementations.
- With CUDA toolkit v9.0, a bug in the NVIDIA® optimization causes numerical mismatch between the results from the generated code and MATLAB. As a workaround, turn off the optimization by passing the following flags to the configuration object (`cfg`) before generating the code.

  ```
  cfg.GpuConfig.CompilerFlags = '-Xptxas -O0'
  ```

  NVIDIA is expected to fix this bug in CUDA toolkit v9.1.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- The filtering kernel `h` must be a vector or 2-D matrix of data type `double`.
- If the image is filtered using a GPU, then `imfilter` computes the value of each output pixel using either single- or double-precision floating point, depending on the data type of `A`. If `A` contains double-precision or `uint32` values, then `imfilter` uses double-precision values. For all other data types, `imfilter` uses single-precision. If `A` is an integer or logical array, then `imfilter` truncates output elements that exceed the range of the given type, and rounds fractional values.

For more information, see “Image Processing on a GPU”.

See Also

`conv2` | `convn` | `filter2` | `fspecial`

Topics

“Filter Grayscale and Truecolor (RGB) Images Using imfilter Function”
“imfilter Boundary Padding Options”
“What Is Image Filtering in the Spatial Domain?”

Introduced before R2006a

imfindcircles

Find circles using circular Hough transform

Syntax

```
centers = imfindcircles(A,radius)
[centers,radii] = imfindcircles(A,radiusRange)
[centers,radii,metric] = imfindcircles(A,radiusRange)
[___] = imfindcircles(___,Name,Value)
```

Description

`centers = imfindcircles(A,radius)` finds the circles in image `A` whose radii are approximately equal to `radius`. The output, `centers`, is a two-column matrix containing the (*x*,*y*) coordinates of the circle centers in the image.

`[centers,radii] = imfindcircles(A,radiusRange)` finds circles with radii in the range specified by `radiusRange`. The additional output argument, `radii`, contains the estimated radii corresponding to each circle center in `centers`.

`[centers,radii,metric] = imfindcircles(A,radiusRange)` also returns a column vector, `metric`, containing the magnitudes of the accumulator array peaks for each circle (in descending order). The rows of `centers` and `radii` correspond to the rows of `metric`.

`[___] = imfindcircles(___,Name,Value)` specifies additional options with one or more name-value arguments, using any of the previous syntaxes.

Examples

Detect Five Strongest Circles in an Image

This example shows how to find all circles in an image, and how to retain and display the strongest circles.

Read a grayscale image into the workspace and display it.

```
A = imread('coins.png');
imshow(A)
```

Find all the circles with radius `r` pixels in the range [15, 30].

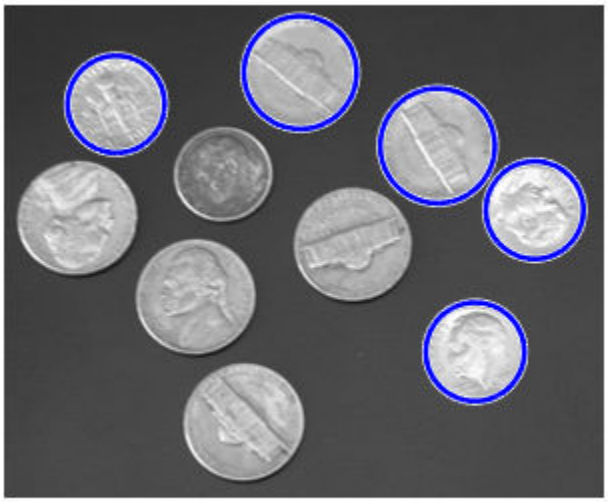
```
[centers, radii, metric] = imfindcircles(A,[15 30]);
```

Retain the five strongest circles according to the metric values.

```
centersStrong5 = centers(1:5,:);
radiiStrong5 = radii(1:5);
metricStrong5 = metric(1:5);
```

Draw the five strongest circle perimeters over the original image.

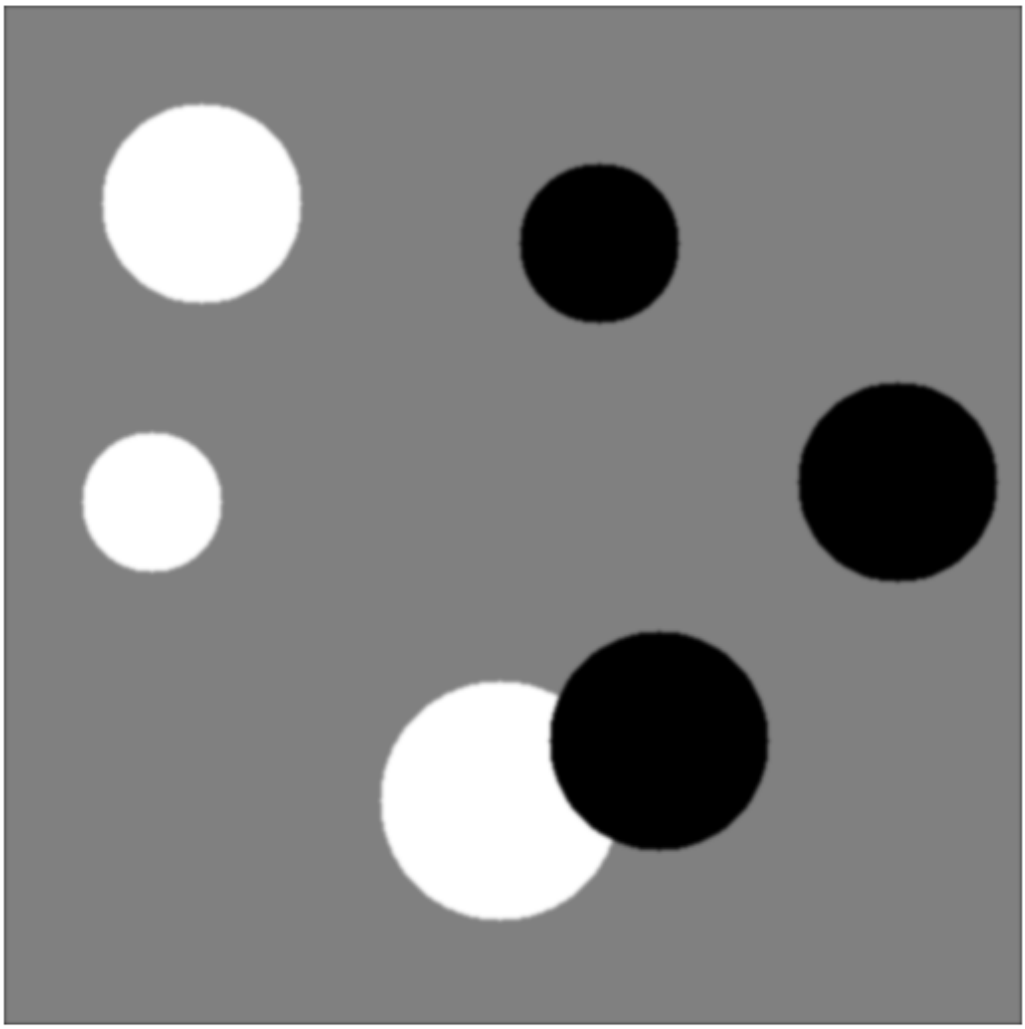
```
viscircles(centersStrong5, radiiStrong5,'EdgeColor','b');
```



Draw Lines Around Bright and Dark Circles in Image

Read the image into the workspace and display it.

```
A = imread('circlesBrightDark.png');
imshow(A)
```



Define the radius range.

```
Rmin = 30;
Rmax = 65;
```

Find all the bright circles in the image within the radius range.

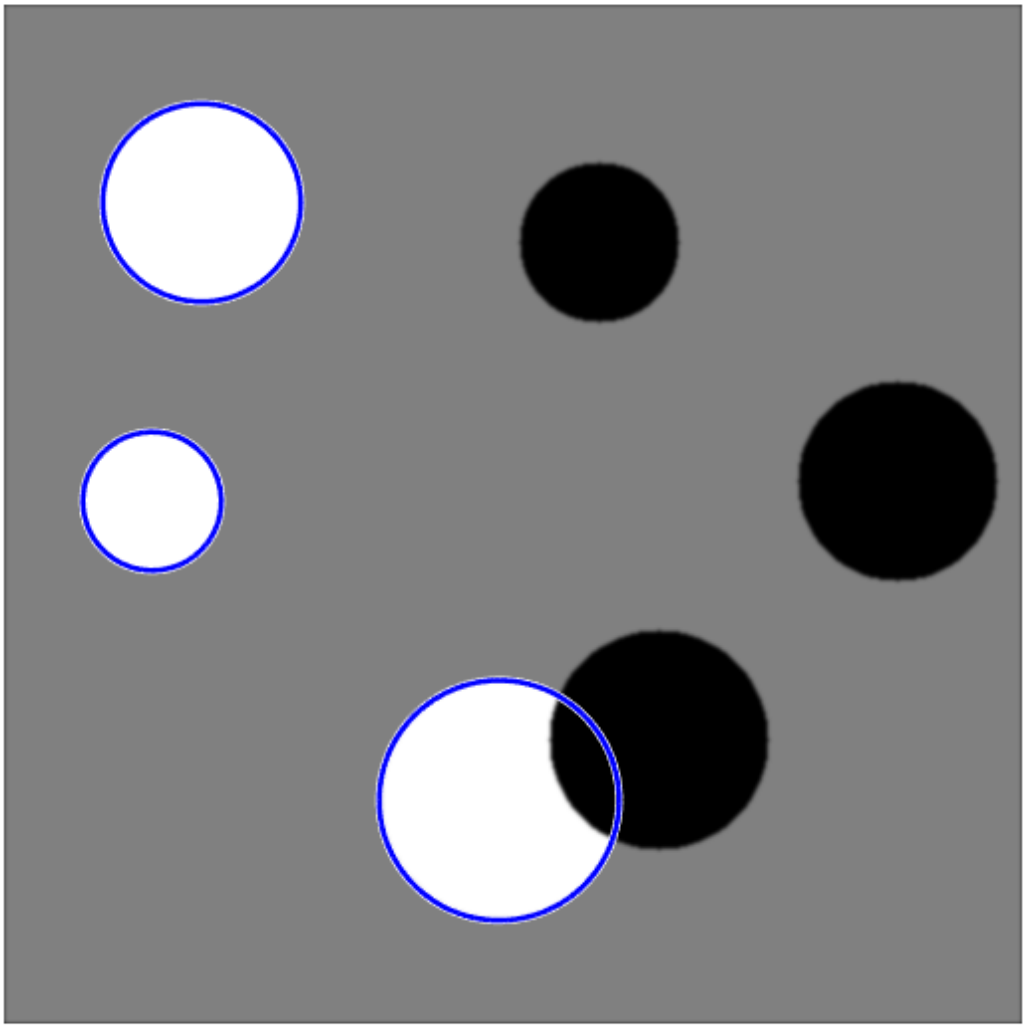
```
[centersBright, radiiBright] = imfindcircles(A,[Rmin Rmax],'ObjectPolarity','bright');
```

Find all the dark circles in the image within the radius range.

```
[centersDark, radiiDark] = imfindcircles(A,[Rmin Rmax],'ObjectPolarity','dark');
```

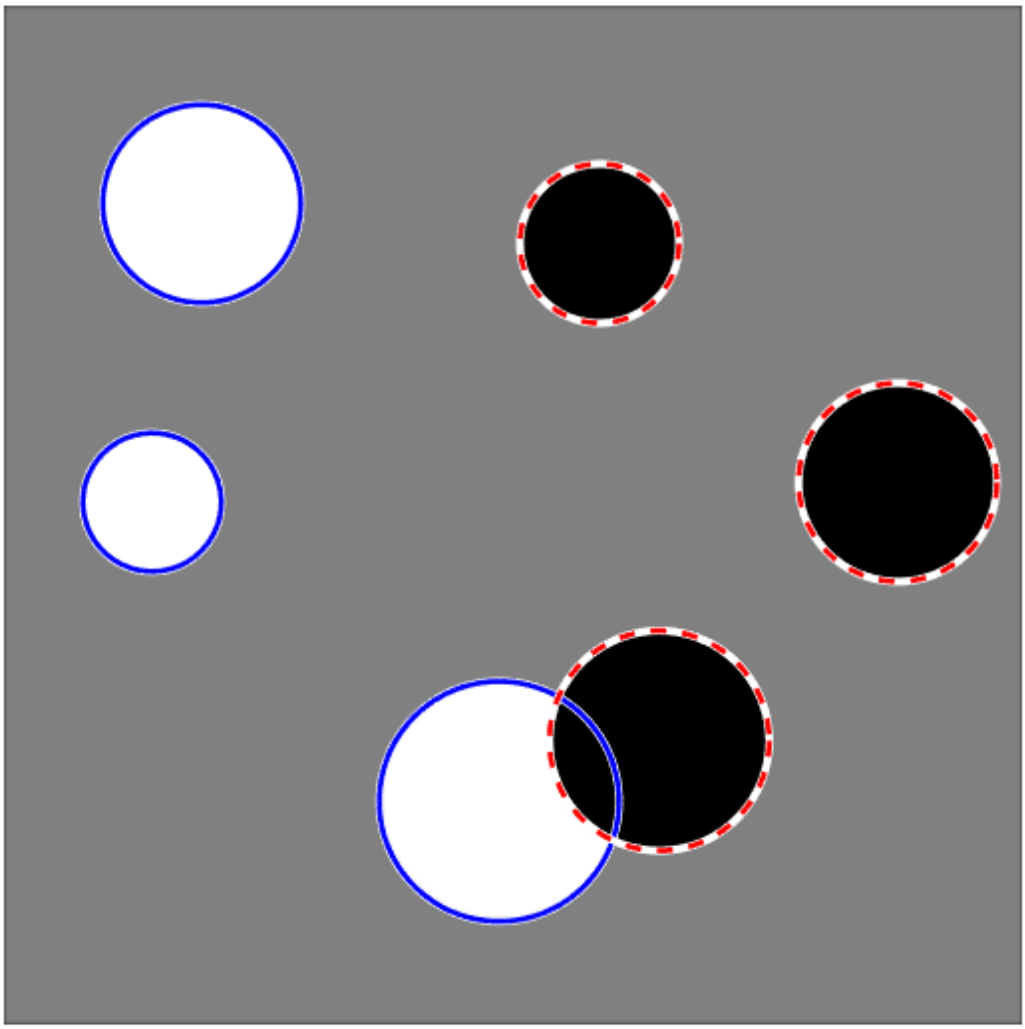
Draw blue lines around the edges of the bright circles.

```
viscircles(centersBright, radiiBright,'Color','b');
```



Draw red dashed lines around the edges of the dark circles.

```
viscircles(centersDark, radiiDark,'LineStyle','--');
```



Input Arguments

`A` — Input image

grayscale image | truecolor image | binary image

Input image in which to detect circular objects, specified as a grayscale, truecolor (RGB), or binary image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`radius` — Circle radius

positive number

Circle radius, or the approximate radius of the circular objects you want to detect, specified as a positive number.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`radiusRange` — Range of radii
2-element vector of positive integers

Range of radii for the circular objects you want to detect, specified as a 2-element vector of positive integers of the form `[rmin rmax]`, where `rmin` is less than `rmax`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `centers = imfindcircles(A,radius,ObjectPolarity=bright)` specifies bright circular objects on a dark background.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `centers = imfindcircles(A,radius,"ObjectPolarity","bright")` specifies bright circular objects on a dark background.

`ObjectPolarity` — Object polarity
`"bright"` (default) | `"dark"`

Object polarity, specified as one of the values in the table.

`"bright"`	The circular objects are brighter than the background.
`"dark"`	The circular objects are darker than the background.

`Method` — Computation method
`"PhaseCode"` (default) | `"TwoStage"`

Computation method used to compute the accumulator array, specified as one of the values in the table.

`"PhaseCode"`	Atherton and Kerbyson's [1] phase-coding method.
`"TwoStage"`	The method used in the two-stage circular Hough transform [2], [3].

Example: `"Method","PhaseCode"` specifies the Atherton and Kerbyson's phase-coding method.

`Sensitivity` — Sensitivity factor
`0.85` (default) | number in the range [0, 1]

Sensitivity factor for the circular Hough transform accumulator array, specified as a number in the range [0, 1]. As you increase the sensitivity factor, `imfindcircles` detects more circular objects, including weak and partially obscured circles. Higher sensitivity values also increase the risk of false detection.

`EdgeThreshold` — Edge gradient threshold
number in the range [0, 1]

Edge gradient threshold for determining edge pixels in the image, specified as a number in the range [0, 1]. Specify `0` to set the threshold to zero-gradient magnitude. Specify `1` to set the threshold to the maximum gradient magnitude. `imfindcircles` detects more circular objects (with both weak and strong edges) when you set the threshold to a lower value. It detects fewer circles with weak edges as you increase the value of the threshold. By default, `imfindcircles` chooses the edge gradient threshold automatically using the function `graythresh`.

Example: `"EdgeThreshold",0.5`

Output Arguments

`centers` — Coordinates of circle centers

P-by-2 matrix

Coordinates of the circle centers, returned as a *P*-by-2 matrix containing the *x*-coordinates of the circle centers in the first column and the *y*-coordinates in the second column. The number of rows, *P*, is the number of circles detected. `centers` is sorted based on the strength of the circles, from strongest to weakest.

Data Types: `double`

`radii` — Estimated radii

column vector

The estimated radii of the circle centers, returned as a column vector. The radius value at `radii(j)` corresponds to the circle centered at `centers(j,:)`.

Data Types: `double`

`metric` — Circle strengths

column vector

Circle strengths provides the relative strengths of the circle centers, returned as a column vector. The value at `metric(j)` corresponds to the circle with radius `radii(j)` centered at `centers(j,:)`.

Data Types: `double`

Tips

- The accuracy of `imfindcircles` is limited when the value of `radius` (or `rmin`) is less than or equal to 5.
- The radius estimation step is typically faster if you use the (default) `"PhaseCode"` method instead of `"TwoStage"`.
- Both computation methods, `"PhaseCode"` and `"TwoStage"` are limited in their ability to detect concentric circles. The results for concentric circles can vary depending on the input image.
- `imfindcircles` does not find circles with centers outside the domain of the image.
- `imfindcircles` converts truecolor images to grayscale using the function `rgb2gray` before processing them. Binary (`logical`) and integer type images are converted to the data type `single` using the `im2single` function before processing. To improve the accuracy of the result for binary images, `imfindcircles` also applies Gaussian smoothing using `imfilter` as a preprocessing step.

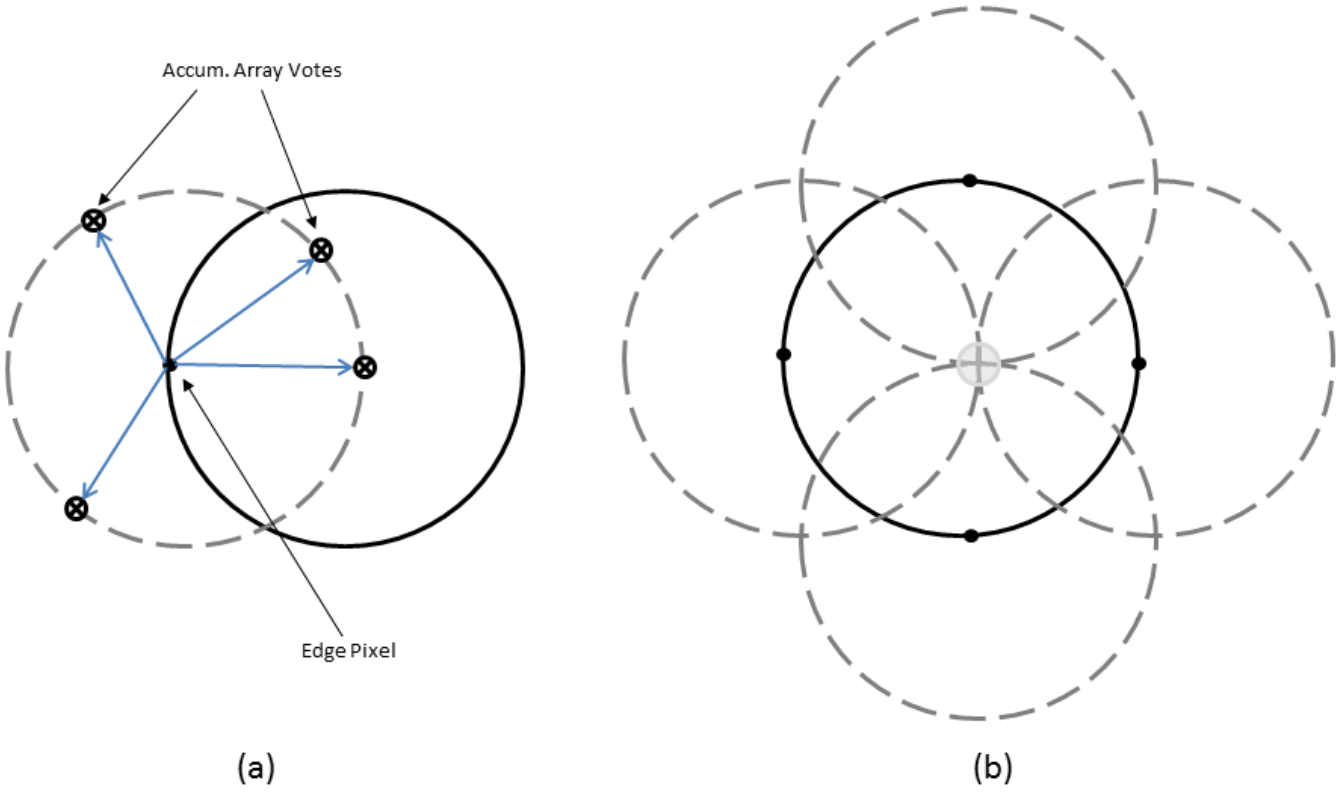
Algorithms

`imfindcircles` uses a Circular Hough Transform (CHT) based algorithm for finding circles in images. This approach is used because of its robustness in the presence of noise, occlusion and varying illumination.

The CHT is not a rigorously specified algorithm, rather there are a number of different approaches that can be taken in its implementation. However, there are three important steps which are common to all approaches.

1 Accumulator Array Computation

Foreground pixels of high gradient are designated as being candidate pixels and are allowed to cast ‘votes’ in the accumulator array. In a classical CHT implementation, the candidate pixels vote in pattern around them that forms a full circle of a fixed radius. Figure 1a shows an example of a candidate pixel lying on an actual circle (solid circle) and the classical CHT voting pattern (dashed circles) for the candidate pixel.



Classical CHT Voting Pattern

2 Center Estimation

The votes of candidate pixels belonging to an image circle tend to accumulate at the accumulator array bin corresponding to the circle’s center. Therefore, the circle centers are estimated by detecting the peaks in the accumulator array. Figure 1b shows an example of the candidate pixels (solid dots) lying on an actual circle (solid circle), and their voting patterns (dashed circles) which coincide at the center of the actual circle.

3 **Radius Estimation**

If the same accumulator array is used for more than one radius value, as is commonly done in CHT algorithms, radii of the detected circles have to be estimated as a separate step.

`imfindcircles` provides two algorithms for finding circles in images: phase-coding (default) and two-stage. Both share some common computational steps, but each has its own unique aspects as well.

The common computational features shared by both algorithms are as follows:

- **Use of 2-D Accumulator Array**

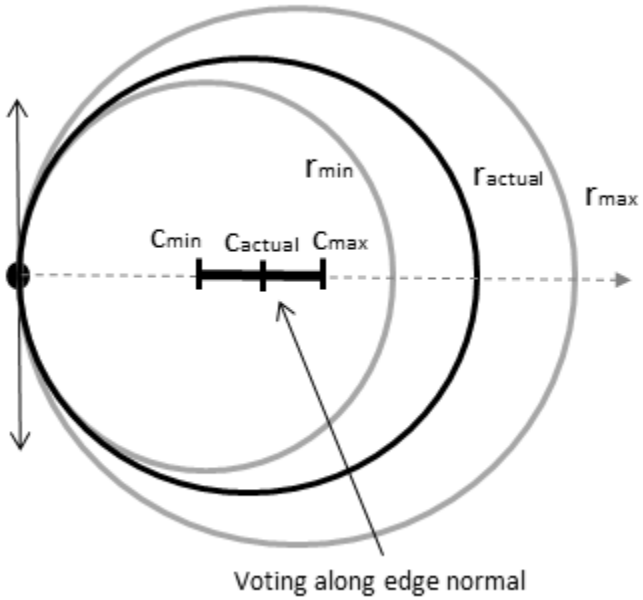
  The classical Hough Transform requires a 3-D array for storing votes for multiple radii, which results in large storage requirements and long processing times. Both the phase-coding and two-stage methods solve this problem by using a single 2-D accumulator array for all the radii. Although this approach requires an additional step of radius estimation, the overall computational load is typically lower, especially when working over a large radius range. This is a widely adopted practice in modern CHT implementations.

- **Use of Edge Pixels**

  Overall memory requirements and speed are strongly governed by the number of candidate pixels. To limit their number, the gradient magnitude of the input image is thresholded so that only pixels of high gradient are included in tallying votes.

- **Use of Edge Orientation Information**

  Performance is also optimized by restricting the number of bins available to candidate pixels. This is accomplished by using locally available edge information to only permit voting in a limited interval along direction of the gradient (Figure 2). The width of the voting interval, between points c_{min} and c_{max} in the figure, is determined by the radius range defined by r_{min} and r_{max}.



Voting Mode: Multiple Radii, Along Direction of Gradient

r_{min}	Minimum search radius
r_{max}	Maximum search radius
r_{actual}	Radius of the circle that the candidate pixel belongs to
c_{min}	Center of the circle of radius r_{min}
c_{max}	Center of the circle of radius r_{max}
c_{actual}	Center of the circle of radius r_{actual}

The two CHT methods employed by the function `imfindcircles` fundamentally differ in the manner by which the circle radii are computed.

- **Two-Stage**

  Radii are explicitly estimated using the estimated circle centers along with image information. The technique is based on computing radial histograms [2] [3].

- **Phase-Coding**

  Radii are estimated from complex values in the accumulator array, with the radius information encoded in the phase of the array entries [1]. The votes cast by the edge pixels contain information not only about the possible center locations but also about the radius of the circle associated with the center location. Unlike the two-stage method where the radius has to be estimated explicitly using radial histograms, in phase-coding the radius can be estimated by simply decoding the phase information from the estimated center location in the accumulator array.

Compatibility Considerations

`imfindcircles` uses new filter size for logical images

Behavior changed in R2019a

Staring in R2019a, the `imfindcircles` function uses a 5-by-5 filter size for smoothing logical images. `imfindcircles` may now return a different answer than in previous releases, when the filter size was 6-by-6. For example, in some instances, the function may return a different number of circles.

References

[1] T.J Atherton, D.J. Kerbyson. "Size invariant circle detection." *Image and Vision Computing*. Volume 17, Number 11, 1999, pp. 795-803.

[2] H.K Yuen, .J. Princen, J. Illingworth, and J. Kittler. "Comparative study of Hough transform methods for circle finding." *Image and Vision Computing*. Volume 8, Number 1, 1990, pp. 71–77.

[3] E.R. Davies, *Machine Vision: Theory, Algorithms, Practicalities*. Chapter 10. 3rd Edition. Morgan Kauffman Publishers, 2005.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imfindcircles` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imfindcircles` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, all character vector input parameters and values must be a compile-time constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

When the sensitivity factor is near or equal to 1, there may be minor mismatch between the results from the generated code and MATLAB simulation.

See Also

`hough` | `houghpeaks` | `houghlines` | `viscircles`

Topics

“Detect and Measure Circular Objects in an Image”

Introduced in R2012a

imflatfield

2-D image flat-field correction

Syntax

```
J = imflatfield(I,sigma)
J = imflatfield(I,sigma,mask)
J = imflatfield( ___ ,'FilterSize',filterSize)
```

Description

`J = imflatfield(I,sigma)` applies flat-field correction to the grayscale or RGB image `I`. The correction uses Gaussian smoothing with a standard deviation of `sigma` to approximate the shading component of `I`. The corrected image is returned in `J`.

`J = imflatfield(I,sigma,mask)` applies flat-field correction to image `I` only where the binary mask is `true`. Where the mask is `false`, the output image `J` contains the unmodified values of image `I`.

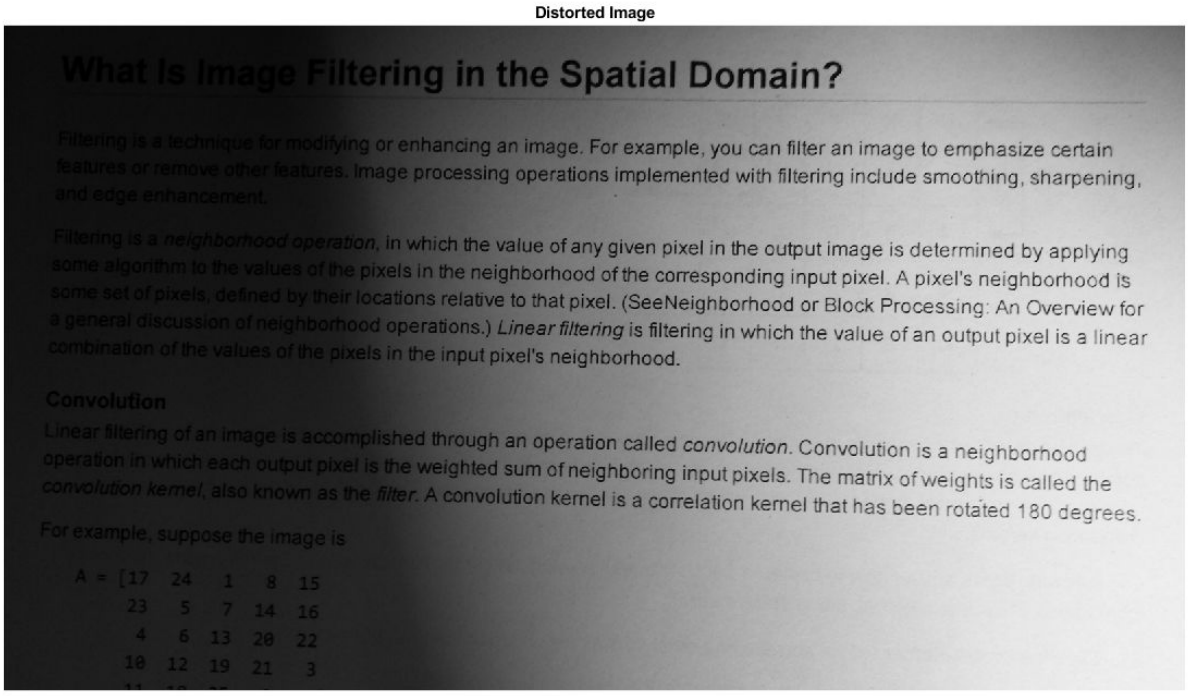
`J = imflatfield( ___ ,'FilterSize',filterSize)` specifies the size of the Gaussian smoothing filter.

Examples

Correct Shading Distortion in Grayscale Image

Load a grayscale image. This image has severe shading distortion on the left side and in the upper-right corner.

```
I = imread('printedtext.png');
imshow(I)
title('Distorted Image')
```

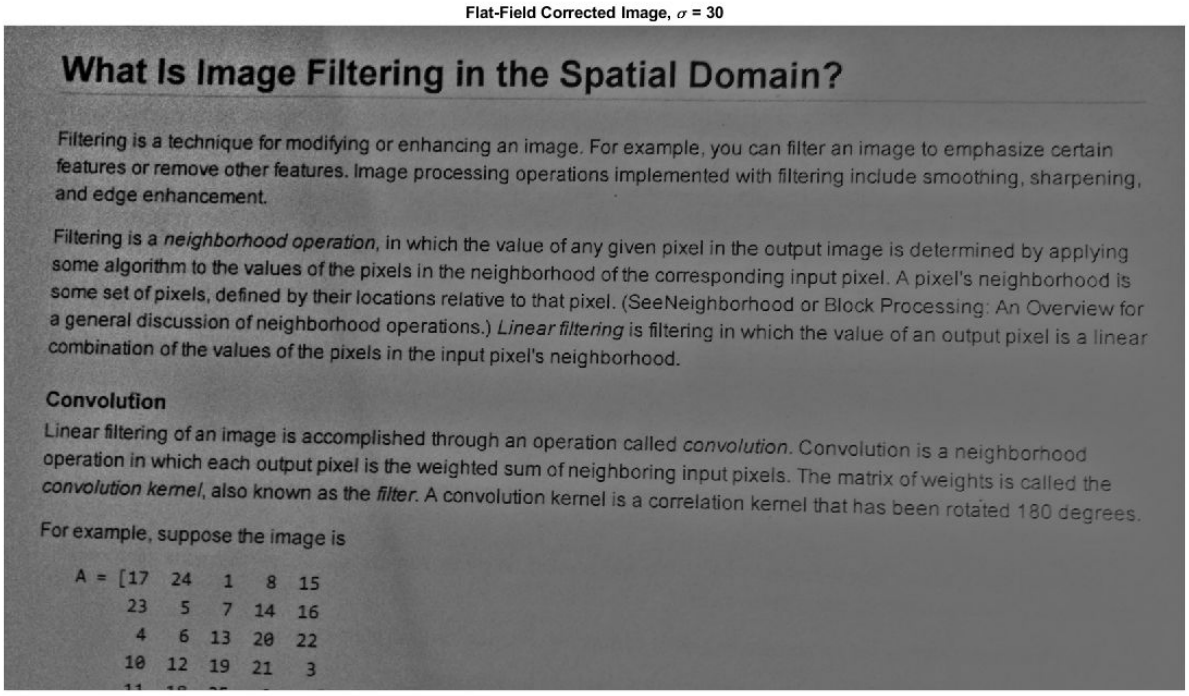


Perform the flat-field correction.

```
sigma = 30;
Iflatfield = imflatfield(I,sigma);
```

Display the result. The corrected image has more uniform brightness.

```
imshow(Iflatfield)
title(['Flat-Field Corrected Image, \sigma = ',num2str(sigma)])
```



Correct Vignetting Defect in Color Image

Load a color image that has vignetting, or darkening of the corners.

```
I = imread('fabric.png');
imshow(I)
title('Image with Vignetting')
```

Perform the flat-field correction.

```
sigma = 20;
Iflatfield = imflatfield(I,sigma);
```

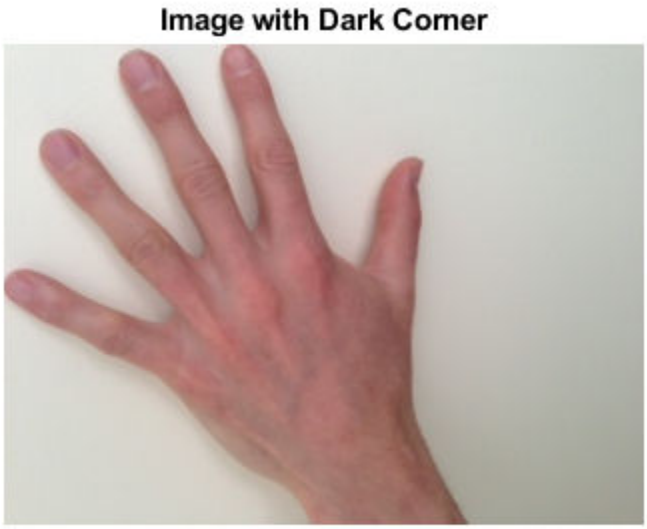
Display the result. The corrected image has more uniform brightness.

```
imshow(Iflatfield)
title(['Flat-Field Corrected Image, \sigma = ',num2str(sigma)])
```

Apply Flat-Field Correction Using Binary Mask

Load a color image. This image has a shading defect in the lower right corner.

```
I = imread('hands1.jpg');
imshow(I)
title('Image with Dark Corner')
```



Try applying flat-field correction to the entire image.

```
sigma = 25;
Iflatfield = imflatfield(I,25);
imshow(Iflatfield)
title(['Flat-Field Corrected Image, \sigma = ',num2str(sigma)])
```

The shading defect in the corner is corrected, but the center of the image is too bright and the hand has changed color. To avoid this brightening artifact, apply flat-field correction just to the background of the image.

Load the mask of this image. In the original mask, `maskHand`, the segmented hand is the region of interest (ROI). Invert the mask so that the background is the ROI. Display the mask, which shows the ROI in white.

```
maskHand = imread('hands1-mask.png');
maskBackground = ~maskHand;
imshow(maskBackground)
title('Background Mask')
```

Perform the flat-field correction on the background of the image using the mask `maskBackground`. The hand is not a region of interest in the mask, therefore flat-field correction is not applied to pixels in the hand.

```
Iflatfield2 = imflatfield(I,sigma,maskBackground);
```

Display the corrected image. The shading defect in the corner is corrected, and the hand retains its original color.

```
imshow(Iflatfield2)
title(['Flat-Field Corrected Background, \sigma = ',num2str(sigma)])
```

Input Arguments

`I` — Distorted image

2-D grayscale image | 2-D RGB image

Distorted image, specified as a 2-D grayscale image of size *m*-by-*n* or a 2-D RGB image of size *m*-by-*n*-by-3.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`sigma` — Standard deviation of Gaussian smoothing filter

positive number | 2-element vector of positive numbers

Standard deviation of the Gaussian smoothing filter, specified as a positive number or a 2-element vector of positive numbers. If you specify a scalar, then `imflatfield` uses a square Gaussian kernel.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`mask` — Binary mask

2-D numeric matrix | 2-D logical matrix

Binary mask, specified as a 2-D numeric or logical matrix of size *m*-by-*n*. For numeric input, any nonzero pixels are considered to be 1 (`true`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`filterSize` — Size of Gaussian filter

positive, odd integer | 2-element vector of positive, odd integers

Size of the Gaussian filter, specified as a scalar or 2-element vector of positive, odd integers. If you specify a scalar, then `imflatfield` uses a square filter. The default filter size is `2*ceil(2*sigma)+1`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`J` — Corrected image

2-D grayscale image | 2-D RGB image

Corrected image, returned as a 2-D grayscale or RGB image of the same size and data type as the input image, `I`.

Tips

- When `I` is an RGB image, then `imflatfield` converts the image to the HSV color space using `rgb2hsv` and applies the flat-field correction to the HSV Value channel. The image is converted back to RGB color space by using `hsv2rgb`.
- If you specify a `mask`, then `imflatfield` dilates the mask and pads the image boundaries to reduce edge artifacts during the flat-field estimation.

See Also

`rgb2hsv` | `hsv2rgb`

Introduced in R2018b

imfuse

Composite of two images

Syntax

```
C = imfuse(A,B)
[C RC] = imfuse(A,RA,B,RB)
C = imfuse( ___ ,method)
C = imfuse( ___ ,Name,Value)
```

Description

`C = imfuse(A,B)` creates a composite image from two images, `A` and `B`. If `A` and `B` are different sizes, `imfuse` pads the smaller dimensions with zeros so that both images are the same size before creating the composite. The output, `C`, is a numeric matrix containing a fused version of images `A` and `B`.

`[C RC] = imfuse(A,RA,B,RB)` creates a composite image from two images, `A` and `B`, using the spatial referencing information provided in `RA` and `RB`. The output `RC` defines the spatial referencing information for the output fused image `C`.

`C = imfuse( ___ ,method)` uses the algorithm specified by `method`.

`C = imfuse( ___ ,Name,Value)` specifies additional options with one or more name-value arguments, using any of the previous syntaxes.

Examples

Create Blended Overlay of Two Images

Load an image into the workspace. Create a copy with a rotation offset applied.

```
A = imread('cameraman.tif');
B = imrotate(A,5,'bicubic','crop');
```

Create blended overlay image, scaling the intensities of `A` and `B` jointly as a single data set. View the fused image.

```
C = imfuse(A,B,'blend','Scaling','joint');
imshow(C)
```

Save the resulting image as a `.png` file.

```
imwrite(C,'my_blend_overlay.png');
```

Create Overlay Image Using Color to Distinguish Areas of Similar Intensity

Load an image into the workspace. Create a copy and apply a rotation offset.

```
A = imread('cameraman.tif');
B = imrotate(A,5,'bicubic','crop');
```

Create a blended overlay image, using red for image `A`, green for image `B`, and yellow for areas of similar intensity between the two images. Then, display the overlay image.

```
C = imfuse(A,B,'falsecolor','Scaling','joint','ColorChannels',[1 2 0]);
imshow(C)
```

Save the resulting image as a `.png` file.

```
imwrite(C,'my_blend_red-green.png');
```

Create Overlay of Two Spatially Referenced Images

Load an image into the workspace and create a spatial referencing object associated with it.

```
A = dicomread('knee1.dcm');
RA = imref2d(size(A));
```

Create a second image by resizing image `A` and create a spatial referencing object associated with that image.

```
B = imresize(A,2);
RB = imref2d(size(B));
```

Set referencing object parameters to specify the limits of the coordinates in world coordinates.

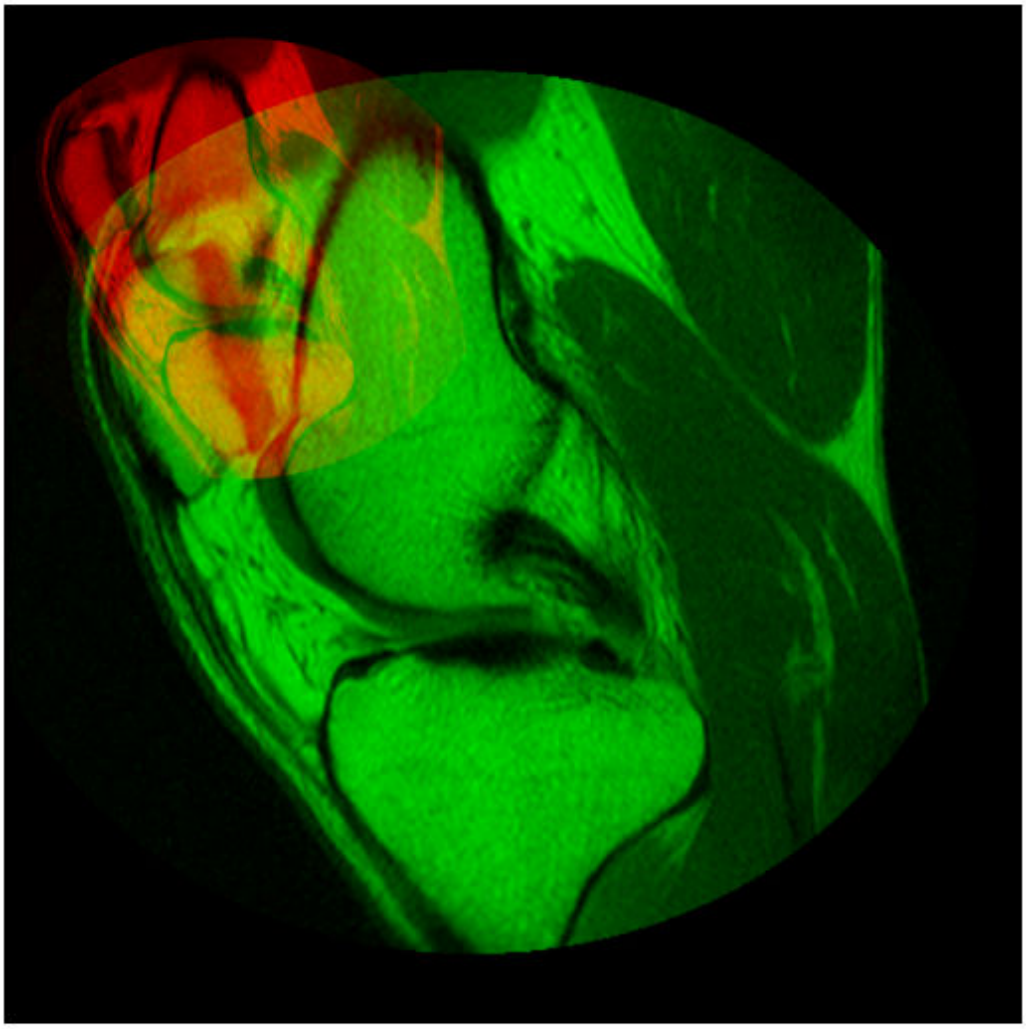
```
RB.XWorldLimits = RA.XWorldLimits;
RB.YWorldLimits = RA.YWorldLimits;
```

Create a blended overlay image using color to indicate areas of similar intensity. This example uses red for image `A`, green for image `B`, and yellow for areas of similar intensity between the two images.

```
C = imfuse(A,B,'falsecolor','Scaling','joint','ColorChannels',[1 2 0]);
```

Display the fused image. Note how the images do not appear to share many areas of similar intensity. For this example, the fused image is shrunk for easier display.

```
C = imresize(C,0.5);
imshow(C)
```

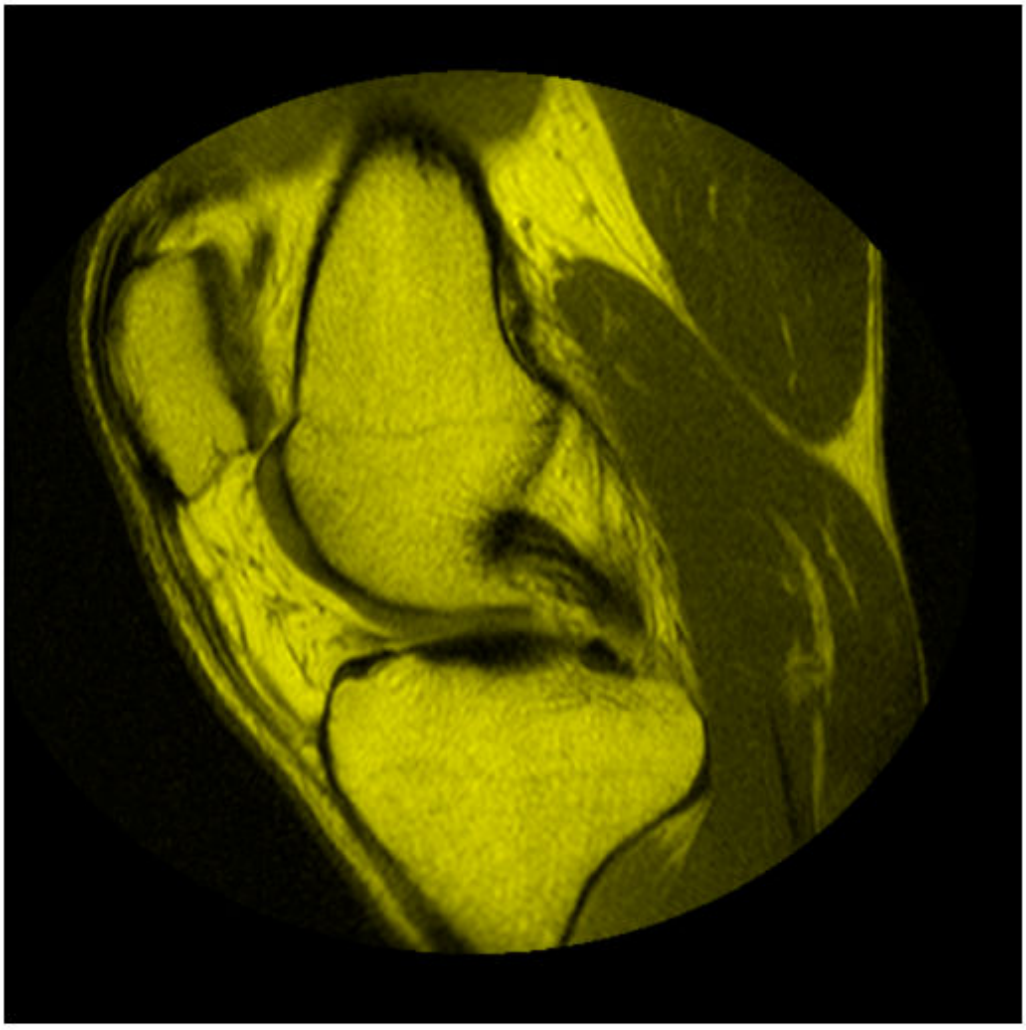


Create a new fused image, this time using the spatial referencing information in RA and RB.

```
[D,RD] = imfuse(A,RA,B,RB,'ColorChannels',[1 2 0]);
```

Display the new fused image. In this version, the image appears yellow because the images A and B have the same extent in the world coordinate system. The images actually are aligned, even though B is twice the size of A. For this example, the fused image is shrunk for easier display.

```
D = imresize(D,0.5);
imshow(D)
```



Input Arguments

A — First image to be combined into composite image

grayscale image | truecolor image | binary image

First image to be combined into a composite image, specified as a grayscale, truecolor, or binary image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

B — Second image to be combined into composite image

grayscale image | truecolor image | binary image

Second image to be combined into a composite image, specified as a grayscale, truecolor, or binary image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`RA` — Spatial referencing information associated with input image `A`
`imref2d` object

Spatial referencing information associated with the input image `A`, specified as an `imref2d` object.

`RB` — Spatial referencing information associated with input image `B`
`imref2d` object

Spatial referencing information associated with the input image `B`, specified as an `imref2d` object.

`method` — Algorithm used to combine images
`"falsecolor"` (default) | `"blend"` | `"diff"` | `"montage"`

Algorithm used to combine the images, specified as one of the following values.

Method	Description
`"falsecolor"`	Creates a composite RGB image showing `A` and `B` overlaid in different color bands. Gray regions in the composite image show where the two images have the same intensities. Magenta and green regions show where the intensities are different. This is the default method.
`"blend"`	Overlays `A` and `B` using alpha blending.
`"checkerboard"`	Creates an image with alternating rectangular regions from `A` and `B`.
`"diff"`	Creates a difference image from `A` and `B`.
`"montage"`	Puts `A` and `B` next to each other in the same image.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `C = imfuse(A,B,Scaling="joint")` scales the intensity values of `A` and `B` together as a single data set.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `C = imfuse(A,B,"Scaling","joint")` scales the intensity values of `A` and `B` together as a single data set.

`Scaling` — Intensity scaling option
`"independent"` (default) | `"joint"` | `"none"`

Intensity scaling option, specified as one of the following values:

`"independent"` — Scales the intensity values of `A` and `B` independently when `C` is created.

`"joint"`	Scales the intensity values in the images jointly as if they were together in the same image. This option is useful when you want to visualize registrations of monomodal images, where one image contains fill values that are outside the dynamic range of the other image.
`"none"`	No additional scaling.

`ColorChannels` — Output color channel for each input image
`"green-magenta"` (default) | `[R G B]` | `"red-cyan"`

Output color channel for each input image, specified as one of the following values:

`[R G B]`	A three element vector that specifies which image to assign to the red, green, and blue channels. The R, G, and B values must be 1 (for the first input image), 2 (for the second input image), and 0 (for neither image).
`"red-cyan"`	A shortcut for the vector `[1 2 2]`, which is suitable for red/cyan stereo anaglyphs.
`"green-magenta"`	A shortcut for the vector `[2 1 2]`, which is a high contrast option, ideal for people with many kinds of color blindness.

Output Arguments

`C` — Fused image
grayscale image | truecolor image | binary image

Fused image that is a composite of the input images, returned as a grayscale, truecolor, or binary image.

Data Types: `uint8`

`RC` — Spatial referencing information associated with fused image
`imref2d` object

Spatial referencing information associated with the fused image `C`, returned as an `imref2d` object.

Tips

- Use `imfuse` to create composite visualizations that you can save to a file. Use `imshowpair` to display composite visualizations to the screen.
- When you specify spatial referencing information `RA` and `RB`, `imfuse` combines the input reference objects and obtains a bounding box that contains the world limits of both images. When the total bounding box results in non-integer pixel dimensions in world coordinates, the fused image can have extra rows or columns of black pixels. The distortion occurs because `imfuse` samples the original images according to a reduced pixel extent.

See Also
`imshowpair` | `montage` | `imoverlay` | `labeloverlay`

Introduced in R2012a

imgaborfilt

Apply Gabor filter or filter bank to 2-D image

Syntax

```
[mag,phase] = imgaborfilt(A,wavelength,orientation)
[mag,phase] = imgaborfilt(A,wavelength,orientation,Name,Value)
[mag,phase] = imgaborfilt(A,gaborbank)
```

Description

`[mag,phase] = imgaborfilt(A,wavelength,orientation)` computes the magnitude and phase response of a Gabor filter for the input grayscale image `A`. `wavelength` describes the wavelength in pixels/cycle of the sinusoidal carrier. `orientation` is the orientation of the filter in degrees.

`[mag,phase] = imgaborfilt(A,wavelength,orientation,Name,Value)` applies a single Gabor filter using name-value arguments to control various aspects of filtering.

`[mag,phase] = imgaborfilt(A,gaborbank)` applies the Gabor filter bank, `gaborbank`, to the input image `A`.

Examples

Apply Single Gabor Filter to Input Image

Read an image into the workspace and convert the image to grayscale.

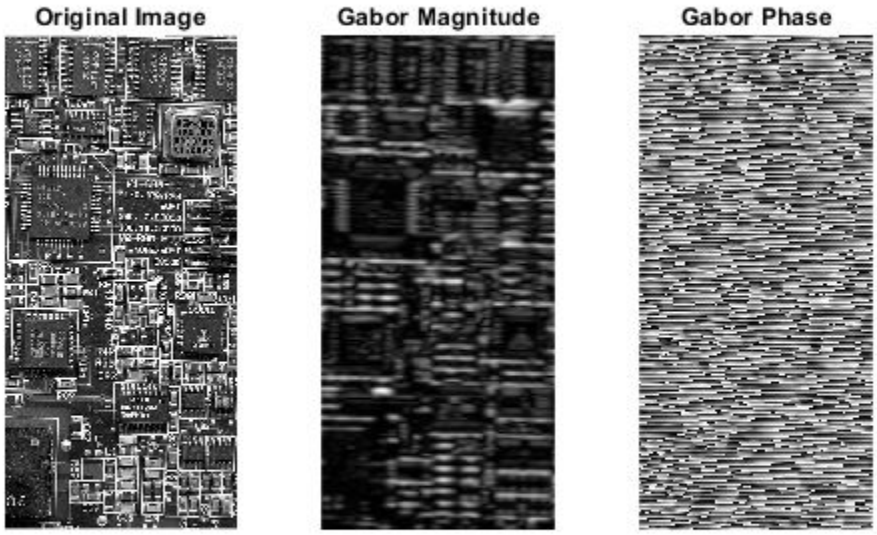
```
I = imread('board.tif');
I = im2gray(I);
```

Apply a Gabor filter to the image.

```
wavelength = 4;
orientation = 90;
[mag,phase] = imgaborfilt(I,wavelength,orientation);
```

Display the original image with plots of the magnitude and phase response calculated by the Gabor filter.

```
tiledlayout(1,3)
nexttile
imshow(I)
title('Original Image')
nexttile
imshow(mag,[])
title('Gabor Magnitude')
nexttile
imshow(phase,[])
title('Gabor Phase')
```



Apply Array of Gabor Filters to Input Image

Read image into the workspace.

```
I = imread('cameraman.tif');
```

Create array of Gabor filters, called a *filter bank*. This filter bank contains two orientations and two wavelengths.

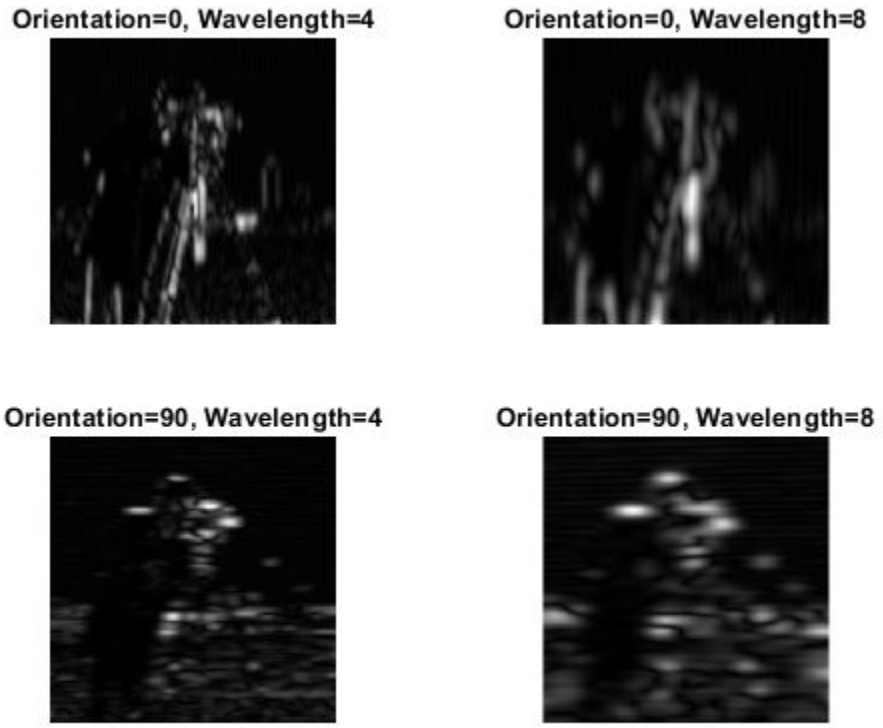
```
gaborArray = gabor([4 8],[0 90]);
```

Apply filters to input image.

```
gaborMag = imgaborfilt(I,gaborArray);
```

Display results. The figure shows the magnitude response for each filter.

```
figure
subplot(2,2,1);
for p = 1:4
    subplot(2,2,p)
    imshow(gaborMag(:,:,p),[]);
    theta = gaborArray(p).Orientation;
    lambda = gaborArray(p).Wavelength;
    title(sprintf('Orientation=%d, Wavelength=%d',theta,lambda));
end
```



Input Arguments

`A` — 2-D grayscale image

numeric matrix

2-D grayscale image, specified as a numeric matrix.

`wavelength` — Wavelength of sinusoidal carrier

number

Wavelength of the sinusoidal carrier, specified as a number greater than or equal to 2, in pixels/cycle. Typical values of `wavelength` range from 2 up to the hypotenuse length of the input image [1].

`orientation` — Orientation of filter

number

Orientation of the filter in degrees, specified as a numeric scalar in the range [0, 360]. The orientation is defined as the normal direction to the sinusoidal plane wave.

`gaborbank` — Gabor filter bank

`gabor` object | array of `gabor` object

Gabor filter bank, specified as a `gabor` object or an array of `gabor` objects.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'SpatialFrequencyBandwidth',2` specifies a spatial frequency bandwidth of two octaves

`SpatialFrequencyBandwidth` — Spatial-frequency bandwidth

`1` (default) | numeric scalar

Spatial-frequency bandwidth, specified as a numeric scalar in units of octaves. The spatial-frequency bandwidth determines the cutoff of the filter response as frequency content in the input image varies from the preferred frequency, *1/lambda*. Typical values for spatial-frequency bandwidth are in the range [0.5, 2.5].

`SpatialAspectRatio` — Ratio of semimajor and semiminor axes of Gaussian envelope

`0.5` (default) | positive number

Ratio of the semimajor and semiminor axes of Gaussian envelope (*semiminor*/*semimajor*), specified as a positive number. This argument controls the ellipticity of the Gaussian envelope. Typical values for spatial aspect ratio are in the range [0.23, 0.92].

Output Arguments

`mag` — Magnitude response

numeric matrix | numeric array

Magnitude response for the Gabor filter or filter bank, returned as a numeric matrix for a single filter or a numeric array for a filter bank. The *p*-th plane of `mag` is the magnitude response for the Gabor filter of the same index, `gaborbank`(*p*).

Data Types: `double`

`phase` — Phase response

numeric matrix | numeric array

Phase response for the Gabor filter or filter bank, returned as a numeric matrix for a single filter or a numeric array for a filter bank. The *p*-th plane of `phase` is the phase response for the Gabor filter of the same index, `gaborbank`(*p*).

Data Types: `double`

Tips

- If the image contains `Inf`s or `NaN`s, then the behavior of `imgaborfilt` is undefined because Gabor filtering is performed in the frequency domain.
- For all input data types other than `single`, `imgaborfilt` performs the computation in `double`. Input images of type `single` are filtered in type `single`. Performance optimizations may result from casting the input image to `single` prior to calling `imgaborfilt`.

References

[1] Jain, Anil K., and Farshid Farrokhnia. "Unsupervised Texture Segmentation Using Gabor Filters." *Pattern Recognition* 24, no. 12 (January 1991): 1167–86. https://doi.org/10.1016/0031-3203(91)90143-S.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imgaborfilt` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- The `wavelength`, `orientation`, `SpatialFrequencyBandwidth`, and `SpatialAspectRatio` must be compile-time constants.
- The filter bank syntax is not supported.

See Also

`gabor` | `edge` | `imfilter` | `imgradient` | `fspecial`

Topics

“Texture Segmentation Using Gabor Filters”

Introduced in R2015b

imgaussfilt

2-D Gaussian filtering of images

Syntax

```
B = imgaussfilt(A)
B = imgaussfilt(A,sigma)
B = imgaussfilt( ___ ,Name,Value)
```

Description

`B = imgaussfilt(A)` filters image `A` with a 2-D Gaussian smoothing kernel with standard deviation of 0.5, and returns the filtered image in `B`.

`B = imgaussfilt(A,sigma)` filters image `A` with a 2-D Gaussian smoothing kernel with standard deviation specified by `sigma`.

`B = imgaussfilt( ___ ,Name,Value)` uses name-value arguments to control aspects of the filtering.

Examples

Smooth Image with Gaussian Filter

Read image to be filtered.

```
I = imread('cameraman.tif');
```

Filter the image with a Gaussian filter with standard deviation of 2.

```
Iblur = imgaussfilt(I,2);
```

Display the original and filtered image in a montage.

```
montage({I,Iblur})
title('Original Image (Left) Vs. Gaussian Filtered Image (Right)')
```

Input Arguments

`A` — Image to be filtered

numeric array

Image to be filtered, specified as a numeric array of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`sigma` — Standard deviation of Gaussian distribution

`0.5` (default) | positive number | 2-element vector of positive numbers

Standard deviation of the Gaussian distribution, specified as a positive number or a 2-element vector of positive numbers. If you specify a scalar, then `imgaussfilt` uses a square Gaussian kernel.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'FilterSize',3`

`FilterSize` — Size of Gaussian filter

positive, odd integer | 2-element vector of positive, odd integers

Size of the Gaussian filter, specified as a positive, odd integer or 2-element vector of positive, odd integers. If you specify a scalar, then `imgaussfilt` uses a square filter. The default filter size is `2*ceil(2*sigma)+1`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Padding` — Image padding
`'replicate'` (default) | numeric scalar | `'circular'` | `'symmetric'`

Image padding, specified as one of the following.

Value	Description
numeric scalar	Pad image with elements of constant value.
`'circular'`	Pad with circular repetition of elements within the dimension.
`'replicate'`	Pad by repeating border elements of array.
`'symmetric'`	Pad image with mirror reflections of itself.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `string`

`FilterDomain` — Domain in which to perform filtering
`'auto'` (default) | `'spatial'` | `'frequency'`

Domain in which to perform filtering, specified as one of the following values:

Value	Description
`'auto'`	Perform convolution in the spatial or frequency domain, based on internal heuristics.
`'frequency'`	Perform convolution in the frequency domain.
`'spatial'`	Perform convolution in the spatial domain.

Data Types: `char` | `string`

Output Arguments

`B` — Filtered image
numeric array

Filtered image, returned as a numeric array of the same class and size as the input image, `A`.

Tips

- If image `A` contains elements with values `Inf` or `NaN`, then the behavior of `imgaussfilt` for frequency domain filtering is undefined. This can happen if you set the `'FilterDomain'` name-value argument to `'frequency'` or if you set it to `'auto'` and `imgaussfilt` uses frequency domain filtering. To restrict the propagation of `Inf`s and `NaN`s in the output in a manner similar to `imfilter`, consider setting the `'FilterDomain'` name-value argument to `'spatial'`.
- If you set the `'FilterDomain'` name-value argument to `'auto'`, then `imgaussfilt` uses an internal heuristic to determine whether spatial or frequency domain filtering is faster. This heuristic is machine dependent and may vary for different configurations. For optimal

performance, try both options, `'spatial'` and `'frequency'`, to determine the best filtering domain for your image and kernel size.

- If you do not specify the `'Padding'` name-value argument, then `imgaussfilt` uses `'replicate'` padding by default, which is different from the default used by `imfilter`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imgaussfilt` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imgaussfilt` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- `imgaussfilt` does not support the `FilterDomain` name-value argument specified as `'frequency'`. Filtering is always done in the spatial domain in generated code.
- When generating code, all string and character vector input arguments must be compile-time constants.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- `imgaussfilt` does not support the `FilterDomain` name-value argument specified as `'frequency'`. Filtering is always done in the spatial domain in generated code.
- When generating code, all string and character vector input arguments must be compile-time constants.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`imgaussfilt3` | `imfilter` | `fspecial`

Introduced in R2015a

imgaussfilt3

3-D Gaussian filtering of 3-D images

Syntax

```
B = imgaussfilt3(A)
B = imgaussfilt3(A,sigma)
B = imgaussfilt3( ___ ,Name,Value)
```

Description

`B = imgaussfilt3(A)` filters 3-D image `A` with a 3-D Gaussian smoothing kernel with standard deviation of 0.5, and returns the filtered image in `B`.

`B = imgaussfilt3(A,sigma)` filters 3-D image `A` with a 3-D Gaussian smoothing kernel with standard deviation specified by `sigma`.

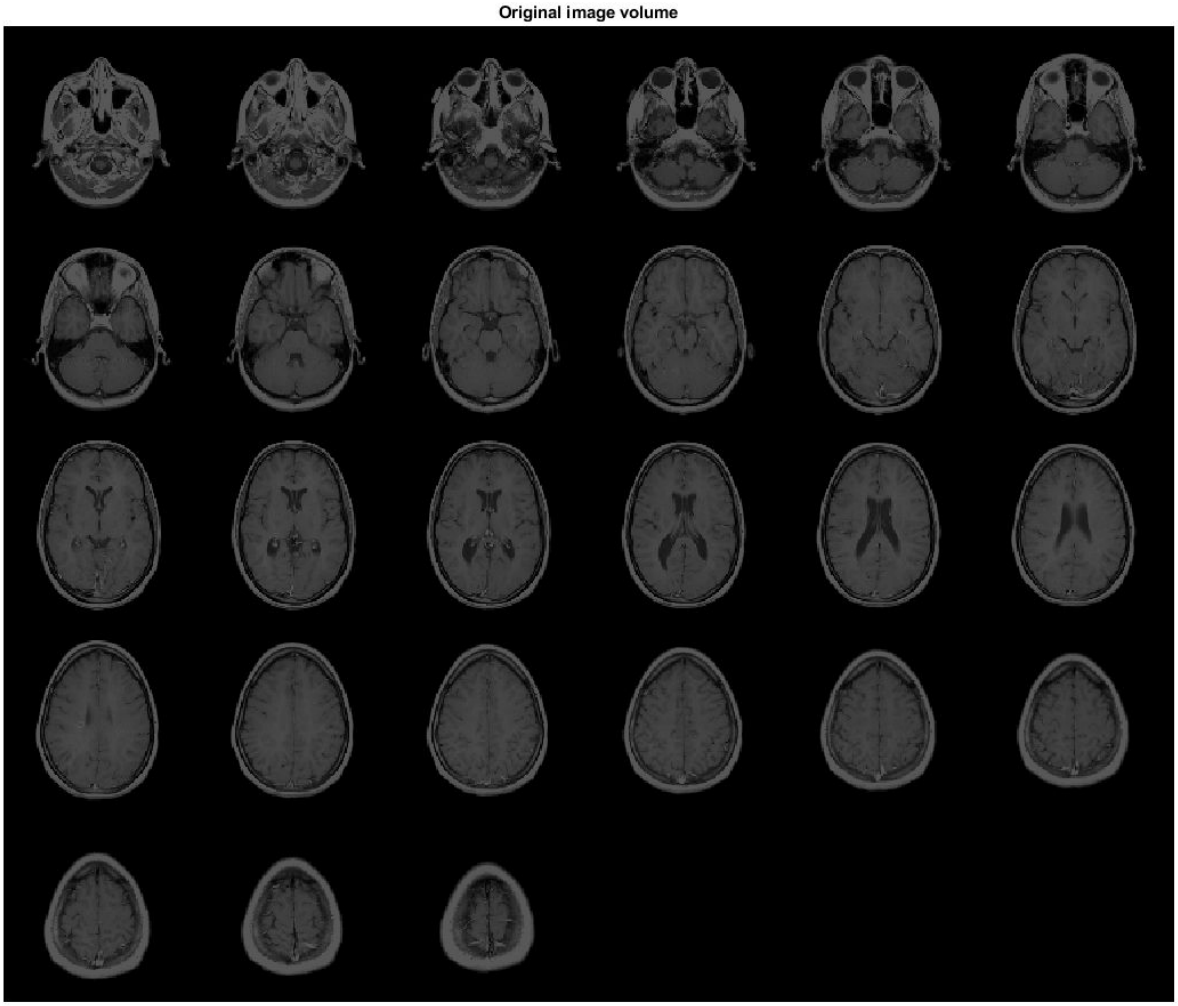
`B = imgaussfilt3( ___ ,Name,Value)` uses name-value pair arguments to control aspects of the filtering.

Examples

Smooth MRI volume with 3-D Gaussian filter

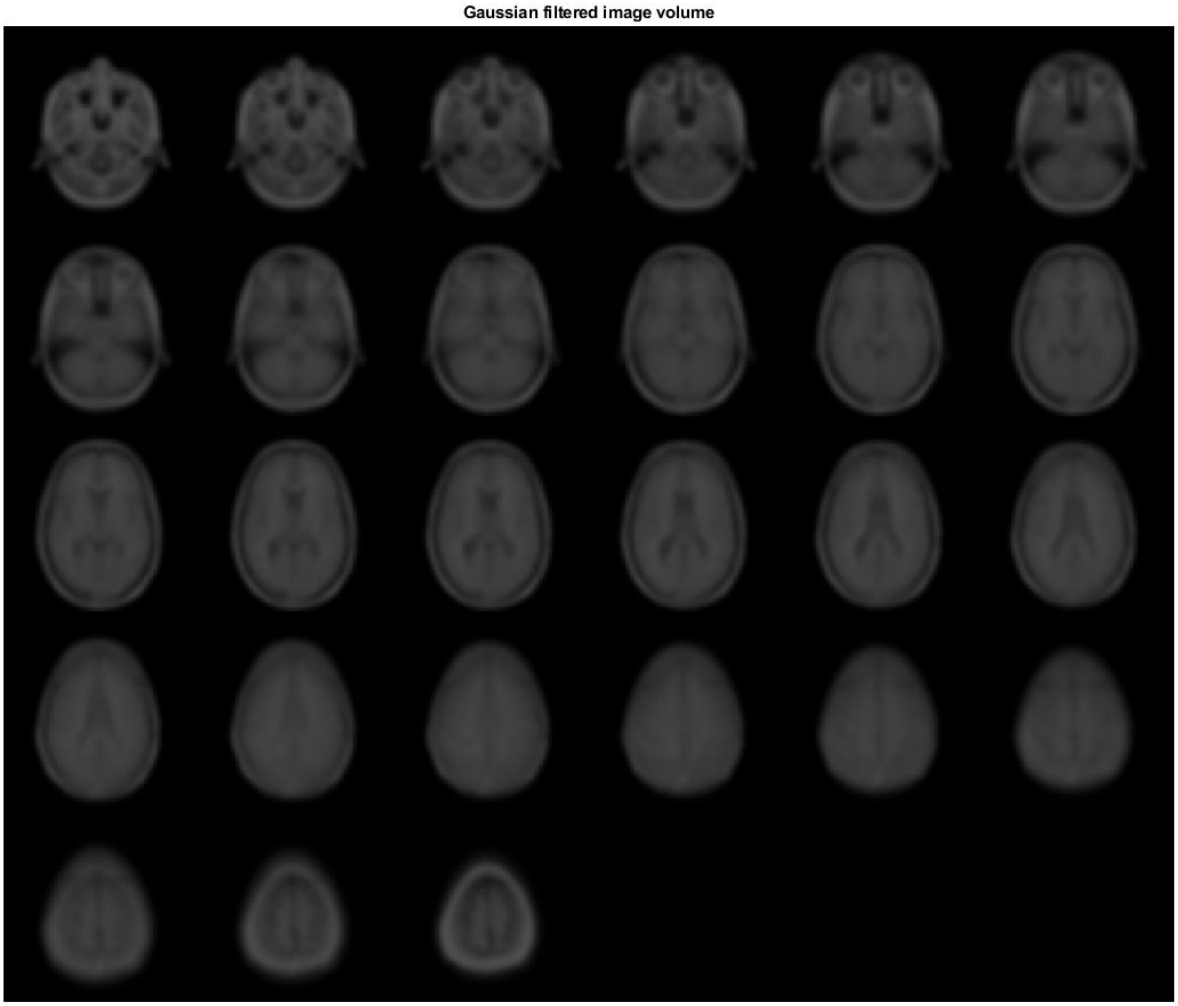
Load MRI data and display it.

```
vol = load('mri');
figure
montage(vol.D)
title('Original image volume')
```



Smooth the image with a 3-D Gaussian filter.

```
siz = vol.siz;
vol = squeeze(vol.D);
sigma = 2;

volSmooth = imgaussfilt3(vol, sigma);

figure
montage(reshape(volSmooth,siz(1),siz(2),1,siz(3)))
title('Gaussian filtered image volume')
```



Input Arguments

`A` — Image to be filtered

3-D numeric array

Image to be filtered, specified as a 3-D numeric array.

Data Types: `single` | `double` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

`sigma` — Standard deviation of the Gaussian distribution

`0.5` (default) | positive number | 3-element vector of positive numbers

Standard deviation of the Gaussian distribution, specified as positive number or a 3-element vector of positive numbers. If `sigma` is a scalar, then `imgaussfilt3` uses a cubic Gaussian kernel.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `volSmooth = imgaussfilt3(vol,sigma,'padding','circular');`

`FilterSize` — Size of the Gaussian filter

positive, odd, integer | 3-element vector of positive, odd, integers

Size of the Gaussian filter, specified as a scalar or 3-element vector of positive, odd, integers. If you specify a scalar, then `imgaussfilt3` uses a cubic filter. The default filter size is `2*ceil(2*sigma)+1`.

Example: `volSmooth = imgaussfilt3(vol,sigma,'FilterSize',5);`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Padding` — Image padding

`'replicate'` (default) | `'circular'` | `'symmetric'` | numeric scalar

Image padding, specified as one of the following.

Value	Description
numeric scalar	Pad image with elements of constant value.
`'circular'`	Pad with circular repetition of elements within the dimension.
`'replicate'`	Pad by repeating border elements of array.
`'symmetric'`	Pad image with mirror reflections of itself.

Example: `volSmooth = imgaussfilt3(vol,sigma,'padding','circular');`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `string`

`FilterDomain` — Domain in which to perform filtering

`'auto'` (default) | `'frequency'` | `'spatial'`

Domain in which to perform filtering, specified as one of the following values.

Filter Domain	Description
`'auto'`	Perform convolution in the spatial or frequency domain, based on internal heuristics.
`'frequency'`	Perform convolution in the frequency domain.
`'spatial'`	Perform convolution in the spatial domain.

Example: `volSmooth = imgaussfilt3(vol,sigma,'FilterDomain','frequency');`

Data Types: `char` | `string`

Output Arguments

B — Filtered image

numeric array

Filtered image, returned as a numeric array of the same class and size as input image.

Tips

- If image `A` contains `Inf`s or `NaN`s, then the behavior of `imgaussfilt3` for frequency domain filtering is undefined. This can happen if you set the `'FilterDomain'` parameter to `'frequency'` or if you set it to `'auto'` and `imgaussfilt3` uses frequency domain filtering. To restrict the propagation of `Inf`s and `NaN`s in the output in a manner similar to `imfilter`, consider setting the `'FilterDomain'` parameter to `'spatial'`.
- If you set the `'FilterDomain'` parameter to `'auto'`, then `imgaussfilt3` uses an internal heuristic to determine whether spatial or frequency domain filtering is faster. This heuristic is machine-dependent and may vary for different configurations. For optimal performance, try both options, `'spatial'` and `'frequency'`, to determine the best filtering domain for your image and kernel size.
- If you do not specify the `'Padding'` parameter, then `imgaussfilt3` uses `'replicate'` padding by default, which is different from the default used by `imfilter`.

Extended Capabilities

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`imgaussfilt` | `imfilter`

Introduced in R2015a

imgca

Get current axes containing image

Syntax

```
ax = imgca
ax = imgca(fig)
```

Description

`ax = imgca` returns the current axes that contains an image. The current axes can be in a regular figure window or in an Image Tool window. Note that the current axes that contains an image might not be the same as the most recently accessed axes.

If no figure contains an axes that contains an image, then `imgca` creates a new axes.

`ax = imgca(fig)` returns the current axes that contains an image in the specified figure.

Examples

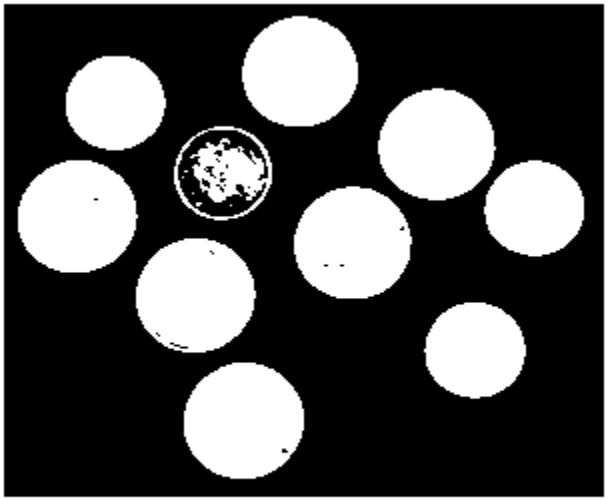
Get Axes Containing Image

Read a grayscale image into the workspace.

```
I = imread('coins.png');
imshow(I)
```

Convert the image into a binary image.

```
bw = imbinarize(I);
imshow(bw)
```

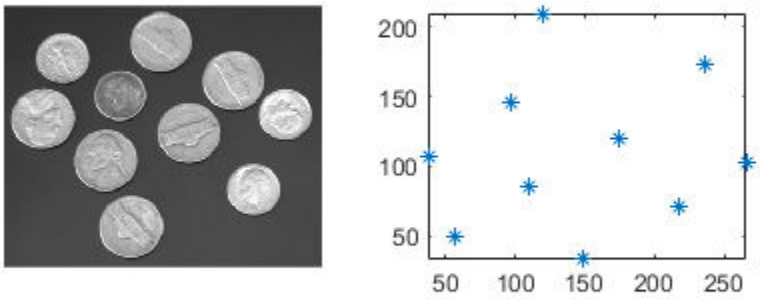


Fill holes in the binary objects, then calculate the centroids of the objects.

```
bw2 = imfill(bw,'holes');
s = regionprops(bw2, 'centroid');
centroids = cat(1,s.Centroid);
```

Display the original image and a plot of the centroids in the same figure window. Note that the current axes contains the plot of the centroids, not the displayed image.

```
subplot(1,2,1)
imshow(I)
subplot(1,2,2)
plot(centroids(:,1),centroids(:,2),'*')
axis image
```

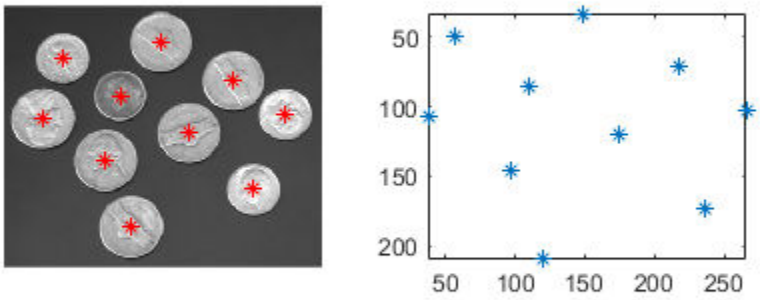


The direction of the *y*-axis is reversed for images. For equivalent comparison of the image and the plot of the centroids, reverse the *y*-axis direction of the plot. To get the most recent axes, which contains the plot of the centroids, use the `gca` function.

```
h = gca;
h.YDir = 'reverse';
```

Use `imgca` to get the most recent axes containing an image. Note that this axes is not the most recent axes. Overlay the centroids in red asterisks on the image.

```
hIm = imgca;
hold(hIm,'on')
plot(hIm,centroids(:,1),centroids(:,2),'r*')
hold(hIm,'off')
```



Input Arguments

`fig` — Figure

figure object

Figure, specified as a `figure` object.

Output Arguments

`ax` — Axes

axes object

Current axes containing an image, returned as an `axes` object.

Tips

- `imgca` can be useful in returning the axes object in the Image Tool. You cannot retrieve this axes using `gca`.

See Also

`gca` | `gcf` | `imgcf` | `imhandles`

Introduced before R2006a

imgcf

Get current figure containing image

Syntax

```
fig = imgcf
```

Description

`fig = imgcf` returns the current figure that contains an image. The figure may be a regular figure window that contains at least one image or an Image Tool window.

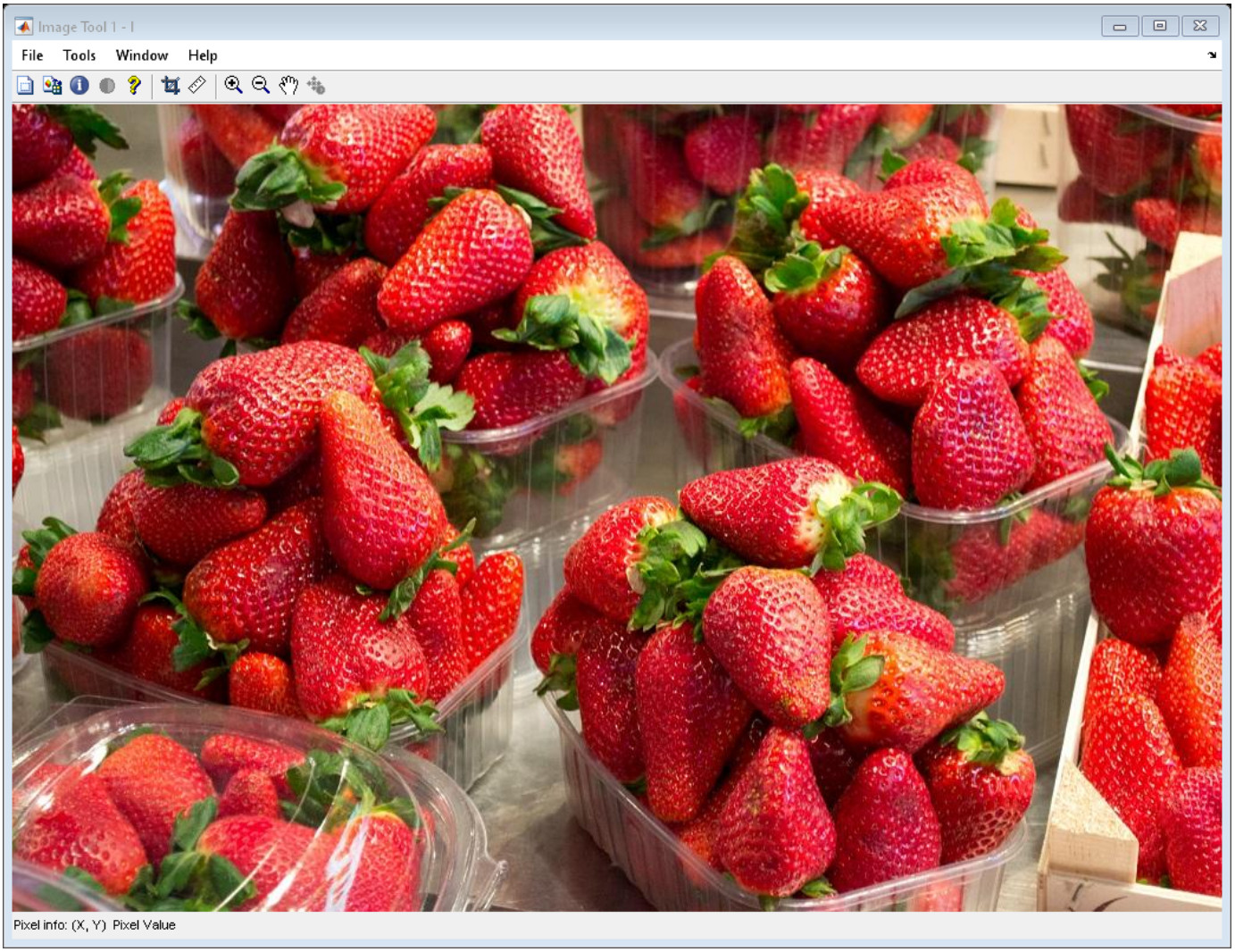
If none of the figures currently open contains an image, then `imgcf` creates a new figure.

Examples

Use Figure Handle in Image Tool Window

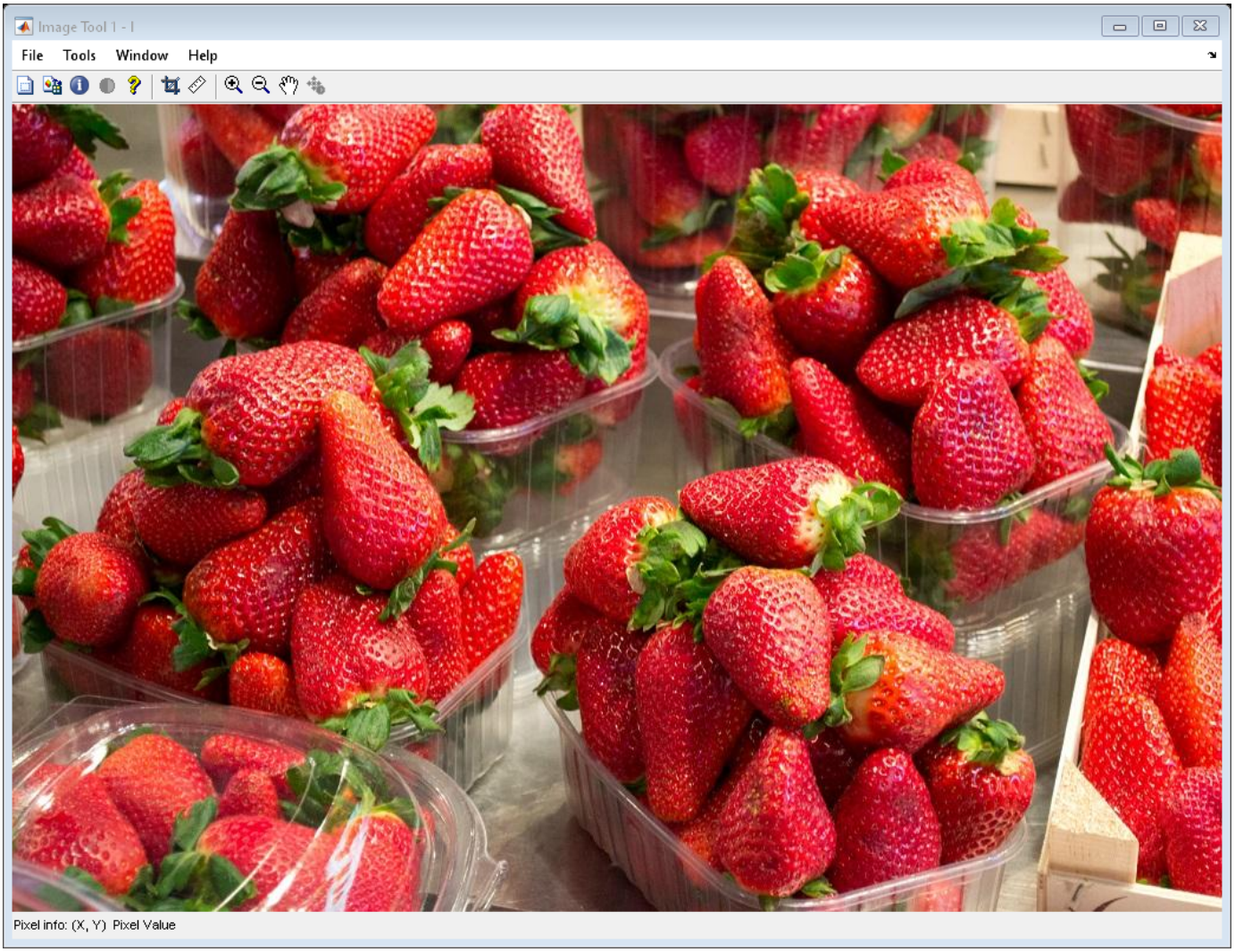
Open an image in an Image Tool window.

```
I = imread('strawberries.jpg');
imtool(I)
```



Use the handle of a figure containing an Image Tool window to center the window on the screen.

```
sz = get(groot,'ScreenSize');
pos = get(imgcf,'Position');
pos = [(sz(3)-pos(3))/2 (sz(4)-pos(4))/2 pos(3) pos(4)];
set(imgcf,'Position',pos)
```



Output Arguments

`fig` — Figure

figure object

Current figure containing an image, returned as a `figure` object.

Tips

- `imgcf` can be useful in getting the figure used by the Image Tool. You cannot retrieve the tool figure using `gcf`.

See Also

`gca` | `gcf` | `imgca` | `imhandles`

Introduced before R2006a

imgetfile

Display Open Image dialog box

Syntax

```
[filename,user_canceled] = imgetfile
[filename,user_canceled] = imgetfile(Name,Value)
```

Description

`[filename,user_canceled] = imgetfile` displays the Open Image dialog box. Use this dialog box in imaging applications to get the name of the image file a user wants to open. The Open Image dialog box includes only files that use supported image file formats (listed in `imformats`) and DICOM files. When the user selects a file and clicks **Open**, `imgetfile` returns the full path of the file in `filename` and sets the `user_canceled` return value to `false`. If the user clicks **Cancel**, `imgetfile` returns an empty character vector (`''`) in `filename` and sets the `user_canceled` return value to `true`.

Note The Open Image dialog box is modal; it blocks the MATLAB command line until the user responds.

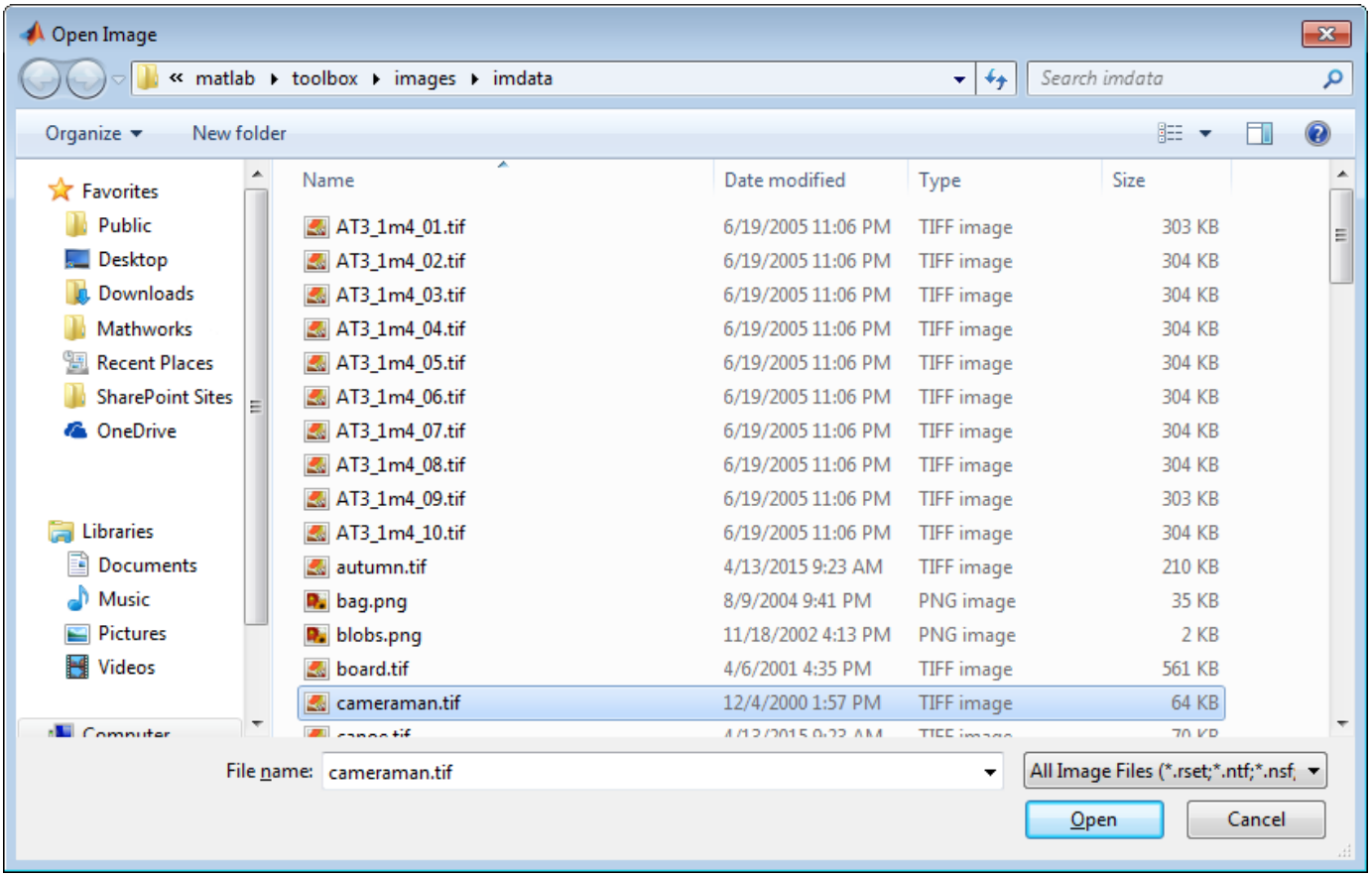
`[filename,user_canceled] = imgetfile(Name,Value)` supports name-value parameter arguments that you can use to control aspects of its behavior.

Examples

Get Name of File Selected from Specified Folder

Open the Open Image dialog box, and show the folder that contains the Image Processing Toolbox sample images.

```
sample_image_folder = fullfile(matlabroot,'toolbox/images/imdata');

[filename,user_canceled] = imgetfile('InitialPath',sample_image_folder)
```



Select an image in the list, and click **Open**. `imgetfile` returns the full path of the image file selected as a character vector. The `user_canceled` return value is set to `false`.

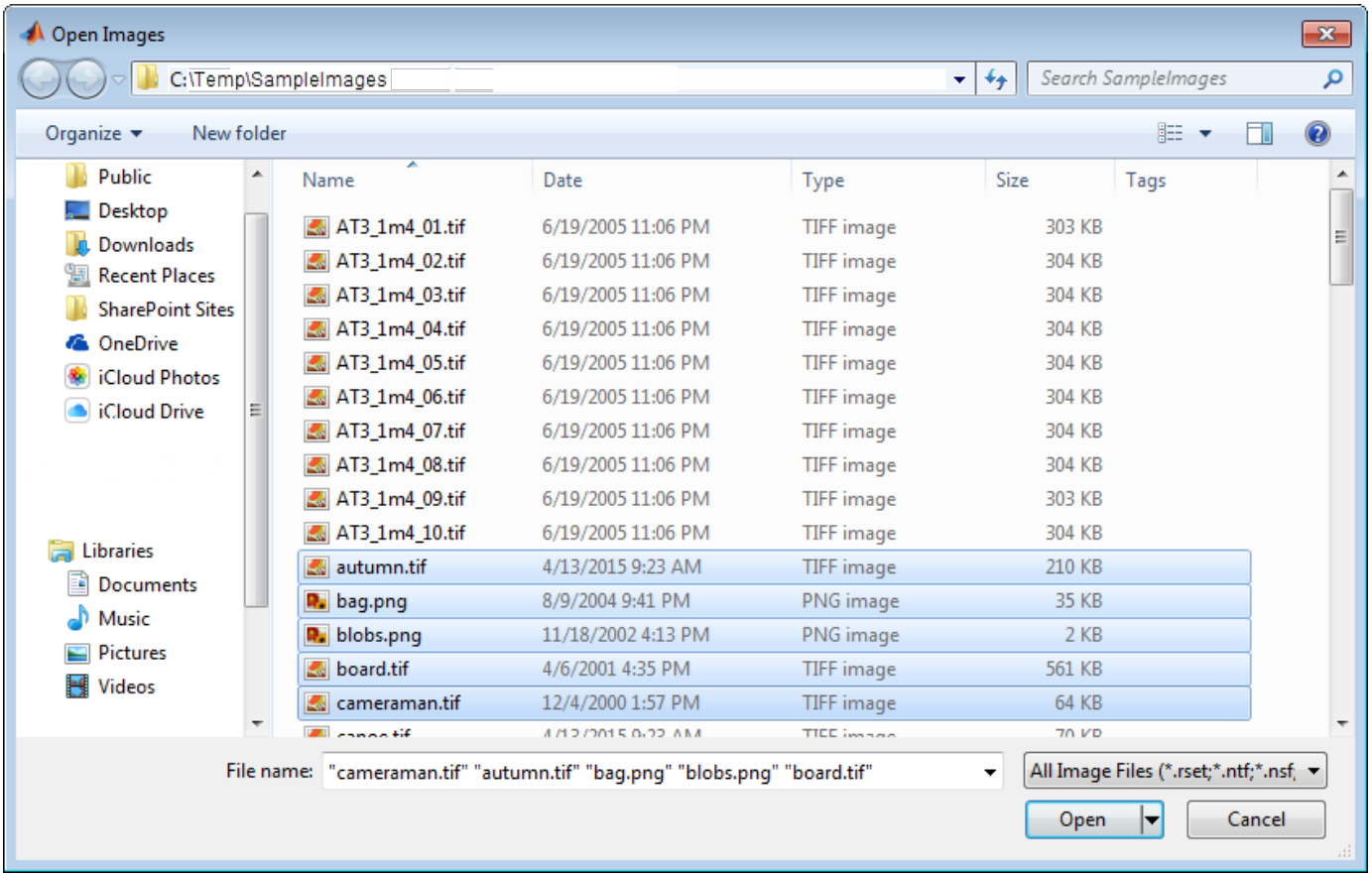
```
filename =

C:\Program Files\MATLAB\R2016b\toolbox\images\imdata\cameraman.tif


user_canceled =

  logical

   0
```

Get Names of Multiple Files from Specified Folder

Open the Open Image dialog box. This example assumes you have a folder that contains sample images on your system `C:` drive.

```
[filename,user_canceled] = imgetfile('InitialPath','C:\Temp\SampleImages','MultiSelect',true)
```

Select several images in the list using **Shift+Click** or **Ctrl+Click**.



Click **Open**. `imgetfile` returns a cell array of character vectors that contain the full path of each image file. The `user_canceled` return value is set to `false`.

```
filename =

  1×5 cell array

  Columns 1 through 3

    'C:\Temp\SampleIma…'    'C:\Temp\SampleIma…'    'C:\Temp\SampleIma…'

  Columns 4 through 5

    'C:\Temp\SampleIma…'    'C:\Temp\SampleIma…'


user_canceled =

  logical
```

```
0
```

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `[fname,user_canc] = imgetfile('InitialPath','C:\temp')`

`InitialPath` — Folder displayed when the Open Image dialog box opens
character vector | string scalar

Folder displayed when the Open Image dialog box opens, specified as a string scalar or character vector. If you do not specify an initial path, `imgetfile` opens the dialog box at the last location where an image was successfully selected.

Data Types: `char`

`MultiSelect` — Selection mode
`false` (default) | `true` | `'on'` | `'off'`

Selection mode, specified as `'on'` or `'off'`, or a Boolean value `true` or `false`. The value `true` or `'on'` turns on multiple selection, enabling a user to select more than one image in the dialog box using **Shift+click** or **Ctrl+click**. The value `false` or `'off'` turns off multiple selection. If multiple selection is on, the output parameter `filename` is a cell array of character vectors containing the full paths to the selected files.

Data Types: `logical` | `char`

Output Arguments

`filename` — Full path of image or images selected by the user
character vector | cell array of character vectors

Full path of image or images selected by the user, returned as a character vector or cell array of character vectors. If the user clicked **Cancel**, `filename` is an empty character vector (`''`).

`user_canceled` — User clicked Cancel
`false` | `true`

User clicked **Cancel**, returned as a Boolean scalar. The value is `true` if the user clicked **Cancel** or `false` if the user selected an image or images.

See Also

Image Viewer | `imformats` | `imputfile` | `uigetfile`

Introduced before R2006a

imgradient

Find gradient magnitude and direction of 2-D image

Syntax

```
[Gmag,Gdir] = imgradient(I)
[Gmag,Gdir] = imgradient(I,method)
[Gmag,Gdir] = imgradient(Gx,Gy)
```

Description

`[Gmag,Gdir] = imgradient(I)` returns the gradient magnitude, `Gmag`, and the gradient direction, `Gdir`, of the 2-D grayscale or binary image `I`.

`[Gmag,Gdir] = imgradient(I,method)` returns the gradient magnitude and direction using the specified `method`.

`[Gmag,Gdir] = imgradient(Gx,Gy)` returns the gradient magnitude and direction from the directional gradients `Gx` and `Gy` in the *x* and *y* directions, respectively.

Examples

Calculate Gradient Magnitude and Direction Using Prewitt Method

Read an image into workspace.

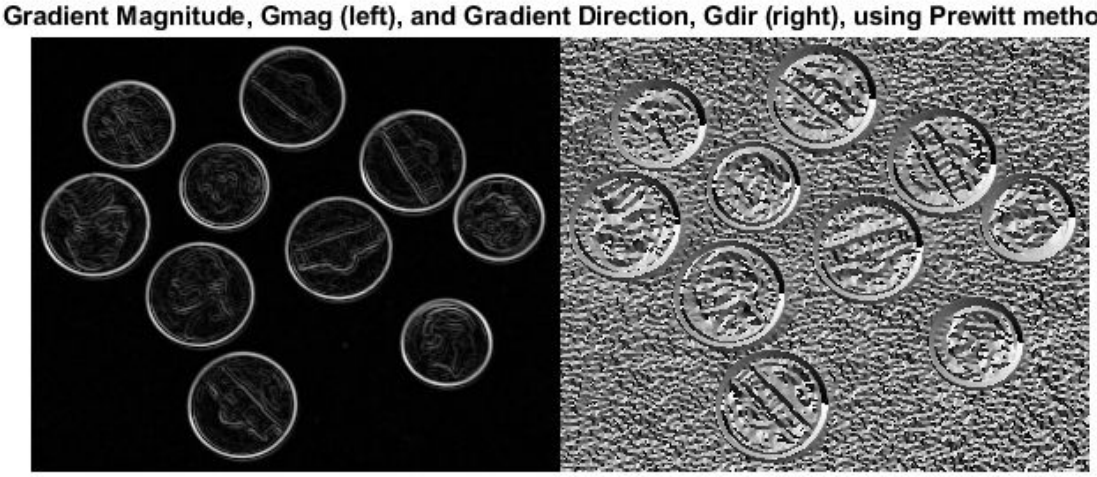
```
I = imread('coins.png');
```

Calculate the gradient magnitude and direction, specifying the Prewitt gradient operator.

```
[Gmag, Gdir] = imgradient(I,'prewitt');
```

Display the gradient magnitude and direction.

```
figure
imshowpair(Gmag, Gdir, 'montage');
title('Gradient Magnitude, Gmag (left), and Gradient Direction, Gdir (right), using Prewitt metho
```



Calculate Gradient Magnitude and Direction Using Directional Gradients

Read an image into workspace.

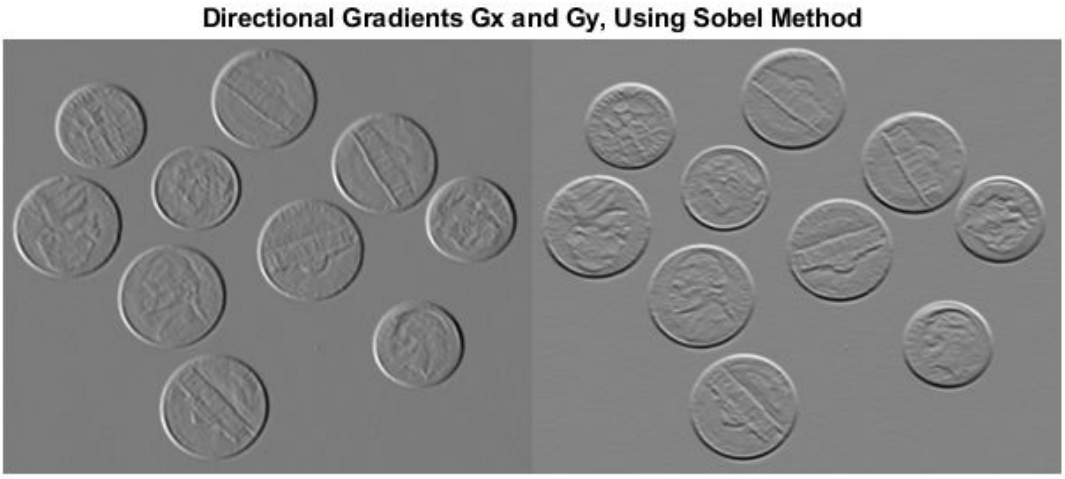
```
I = imread('coins.png');
```

Calculate the *x*- and *y*-directional gradients. By default, `imgradientxy` uses the Sobel gradient operator.

```
[Gx,Gy] = imgradientxy(I);
```

Display the directional gradients.

```
imshowpair(Gx,Gy,'montage')
title('Directional Gradients Gx and Gy, Using Sobel Method')
```

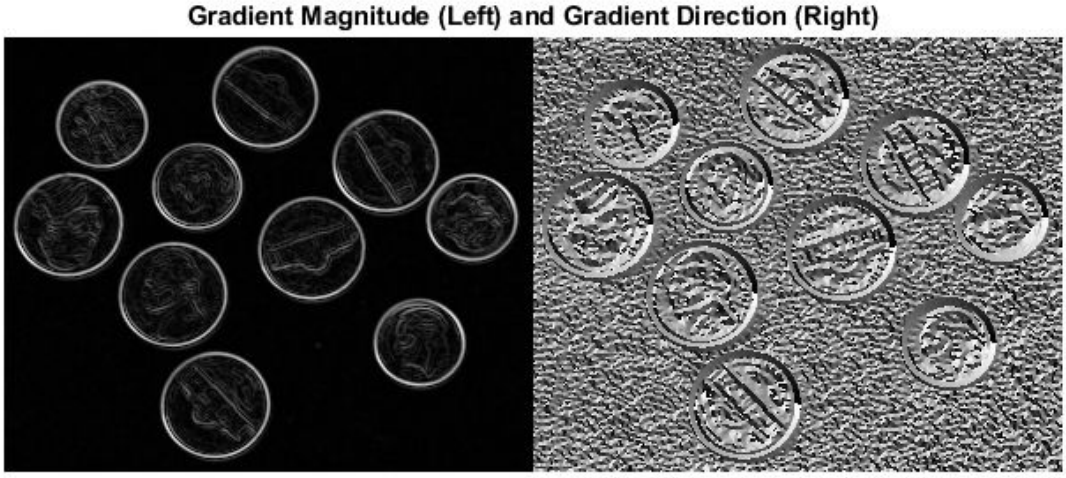


Calculate the gradient magnitude and direction using the directional gradients.

```
[Gmag,Gdir] = imgradient(Gx,Gy);
```

Display the gradient magnitude and direction.

```
imshowpair(Gmag,Gdir,'montage')
title('Gradient Magnitude (Left) and Gradient Direction (Right)')
```



Input Arguments

`I` — Input image

2-D grayscale image | 2-D binary image

Input image, specified as a 2-D grayscale or 2-D binary image.

Data Types: `single` | `double` | `int8` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`method` — Gradient operator

`'sobel'` (default) | `'prewitt'` | `'central'` | `'intermediate'` | `'roberts'`

Gradient operator, specified as one of the following values.

Method	Description

`'sobel'`	Sobel gradient operator. The gradient of a pixel is a weighted sum of pixels in the 3-by-3 neighborhood. For gradients in the vertical (*y*) direction, the weights are:
	`[ 1  2  1`
	`  0  0  0`
	` -1 -2 -1 ]`
	In the *x* direction, the weights are transposed.

`'prewitt'`	Prewitt gradient operator. The gradient of a pixel is a weighted sum of pixels in the 3-by-3 neighborhood. For gradients in the vertical (*y*) direction, the weights are:
	`[ 1  1  1`
	`  0  0  0`
	` -1 -1 -1 ]`
	In the *x* direction, the weights are transposed.

`'central'`	Central difference gradient. The gradient of a pixel is a weighted difference of neighboring pixels. In the *y* direction, `dI/dy = (I(y+1) - I(y-1))/2`.
`'intermediate'`	Intermediate difference gradient. The gradient of a pixel is the difference between an adjacent pixel and the current pixel. In the *y* direction, `dI/dy = I(y+1) - I(y)`.

`'roberts'`	Roberts gradient operator. The gradient of a pixel is the difference between diagonally adjacent pixels. For gradients in one direction, the weights are:
	`[ 1  0`
	`  0 -1 ]`
	In the orthogonal direction, the weights are flipped along the vertical axis.

Data Types: `char` | `string`

`Gx` — Horizontal gradient

numeric matrix

Horizontal gradient, specified as a numeric matrix. The horizontal (*x*) axis points in the direction of increasing column subscripts. You can use the `imgradientxy` function to calculate `Gx`.

Data Types: `single` | `double` | `int8` | `int32` | `uint8` | `uint16` | `uint32`

Gy — Vertical gradient

numeric matrix

Vertical gradient, specified as a numeric matrix of the same size as `Gx`. The vertical (*y*) axis points in the direction of increasing row subscripts. You can use the `imgradientxy` function to calculate `Gy`.

Data Types: `single` | `double` | `int8` | `int32` | `uint8` | `uint16` | `uint32`

Output Arguments

Gmag — Gradient magnitude

numeric matrix

Gradient magnitude, returned as a numeric matrix of the same size as image `I` or the directional gradients `Gx` and `Gy`. `Gmag` is of class `double`, unless the input image or directional gradients are of data type `single`, in which case it is of data type `single`.

Data Types: `double` | `single`

Gdir — Gradient direction

numeric matrix

Gradient direction, returned as a numeric matrix of the same size as gradient magnitude `Gmag`. `Gdir` contains angles in degrees within the range [-180, 180] measured counterclockwise from the positive *x*-axis. (The *x*-axis points in the direction of increasing column subscripts.) `Gdir` is of class `double`, unless the input image `I` or directional gradients are of data type `single`, in which case it is of data type `single`.

Data Types: `double` | `single`

Tips

- When applying the gradient operator at the boundaries of the image, values outside the bounds of the image are assumed to equal the nearest image border value. This is similar to the `'replicate'` boundary option in `imfilter`.

Algorithms

The algorithmic approach taken in `imgradient` for each of the listed gradient methods is to first compute directional gradients, `Gx` and `Gy`, in the *x* and *y* directions, respectively. The horizontal (*x*) axis points in the direction of increasing column subscripts. The vertical (*y*) axis points in the direction of increasing row subscripts. The gradient magnitude and direction are then computed from their orthogonal components `Gx` and `Gy`.

`imgradient` does not normalize the gradient output. If the range of the gradient output image has to match the range of the input image, consider normalizing the gradient image, depending on the `method` argument used. For example, with a Sobel kernel, the normalization factor is 1/8, for Prewitt, it is 1/6, and for Roberts it is 1/2.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imgradient` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imgradient` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- The value of `method` must be a compile time constant.
- The generated code does not always produce the same results as MATLAB for the `Gdir` output.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`imgradientxy` | `imgradientxyz` | `imgradient3` | `edge` | `fspecial`

Introduced in R2012b

imgradient3

Find gradient magnitude and direction of 3-D image

Syntax

```
[Gmag,Gazimuth,Gelevation] = imgradient3(I)
[Gmag,Gazimuth,Gelevation] = imgradient3(I,method)
[Gmag,Gazimuth,Gelevation] = imgradient3(Gx,Gy,Gz)
```

Description

`[Gmag,Gazimuth,Gelevation] = imgradient3(I)` returns the gradient magnitude, `Gmag`, gradient direction, `Gazimuth`, and gradient elevation `Gelevation` of the 3-D grayscale or binary image `I`.

`[Gmag,Gazimuth,Gelevation] = imgradient3(I,method)` calculates the gradient magnitude, direction, and elevation using the specified `method`.

`[Gmag,Gazimuth,Gelevation] = imgradient3(Gx,Gy,Gz)` calculates the gradient magnitude, direction, and elevation from the directional gradients `Gx`, `Gy`, and `Gz` in the *x*, *y*, and *z* directions, respectively.

Examples

Compute 3-D Gradient Magnitude and Direction Using Sobel Method

Read 3-D data into the workspace and prepare it for processing.

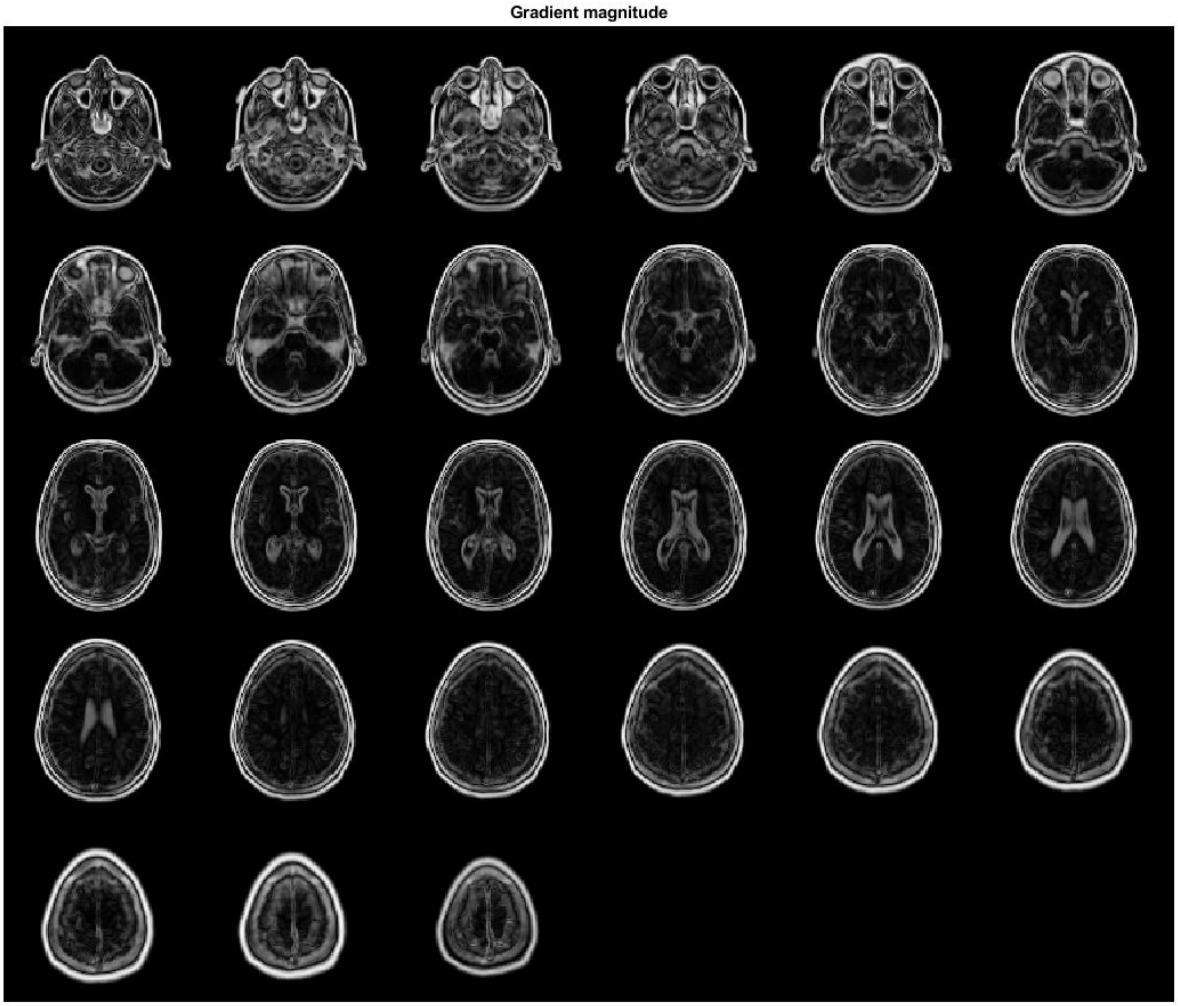
```
volData = load('mri');
sz = volData.siz;
vol = squeeze(volData.D);
```

Calculate the gradients.

```
[Gmag, Gaz, Gelev] = imgradient3(vol);
```

Visualize the gradient magnitude as a montage.

```
figure,
montage(reshape(Gmag,sz(1),sz(2),1,sz(3)),'DisplayRange',[])
title('Gradient magnitude')
```



Input Arguments

`I` — Input image

3-D grayscale image | 3-D binary image

Input image, specified as a 3-D grayscale image or 3-D binary image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`method` — Gradient operator

`'sobel'` (default) | `'prewitt'` | `'central'` | `'intermediate'`

Gradient operator, specified as one of the following values.

<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>'sobel'</td><td>Sobel gradient operator. The gradient of a pixel is a weighted sum of pixels in the 3-by-3-by-3 neighborhood. For example, in the depth (z) direction, the weights in the three planes are:
<table>
<tr><th>plane z-1</th><th>plane z</th><th>plane z+1</th></tr>
<tr><td>[1 3 1
3 6 3
1 3 1]</td><td>[0 0 0
0 0 0
0 0 0]</td><td>[-1 -3 -1
-3 -6 -3
-1 -3 -1]</td></tr>
</table></td></tr>
<tr><td>'prewitt'</td><td>Prewitt gradient operator. The gradient of a pixel is a weighted sum of pixels in the 3-by-3-by-3 neighborhood. For example, in the depth (z) direction, the weights in the three planes are:
<table>
<tr><th>plane z-1</th><th>plane z</th><th>plane z+1</th></tr>
<tr><td>[1 1 1
1 1 1
1 1 1]</td><td>[0 0 0
0 0 0
0 0 0]</td><td>[-1 -1 -1
-1 -1 -1
-1 -1 -1]</td></tr>
</table></td></tr>
<tr><td>'central'</td><td>Central difference gradient. The gradient of a pixel is a weighted difference of neighboring pixels. For example, in the depth (z) direction, <code>dI/dz = (I(z+1) - I(z-1))/2</code>.</td></tr>
<tr><td>'intermediate'</td><td>Intermediate difference gradient. The gradient of a pixel is the difference between an adjacent pixel and the current pixel. For example, in the depth (z) direction, <code>dI/dz = I(z+1) - I(z)</code>.</td></tr>
</table>

When applying the gradient operator at the boundaries of the image, `imgradient3` assumes values outside the bounds of the image equal the nearest image border value. This behavior is similar to the `'replicate'` boundary option in `imfilter`.

Data Types: `char` | `string`

Gx — Horizontal gradient

3-D numeric array

Horizontal gradient, specified as a 3-D numeric array. The horizontal (*x*) axis points in the direction of increasing column subscripts. You can use the `imgradientxyz` function to calculate `Gx`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Gy — Vertical gradient

3-D numeric array

Vertical gradient, specified as a 3-D numeric array of the same size as `Gx`. The vertical (*y*) axis points in the direction of increasing row subscripts. You can use the `imgradientxyz` function to calculate `Gy`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Gz — Depth gradient

3-D numeric array

Depth gradient, specified as a 3-D numeric array of the same size as `Gx`. The depth (*z*) axis points in the direction of increasing plane subscripts. You can use the `imgradientxyz` function to calculate `Gz`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`Gmag` — Magnitude of the gradient vector

3-D numeric array

Magnitude of the gradient vector, returned as a 3-D numeric array of the same size as image `I` or the directional gradients, `Gx`, `Gy`, and `Gz`.

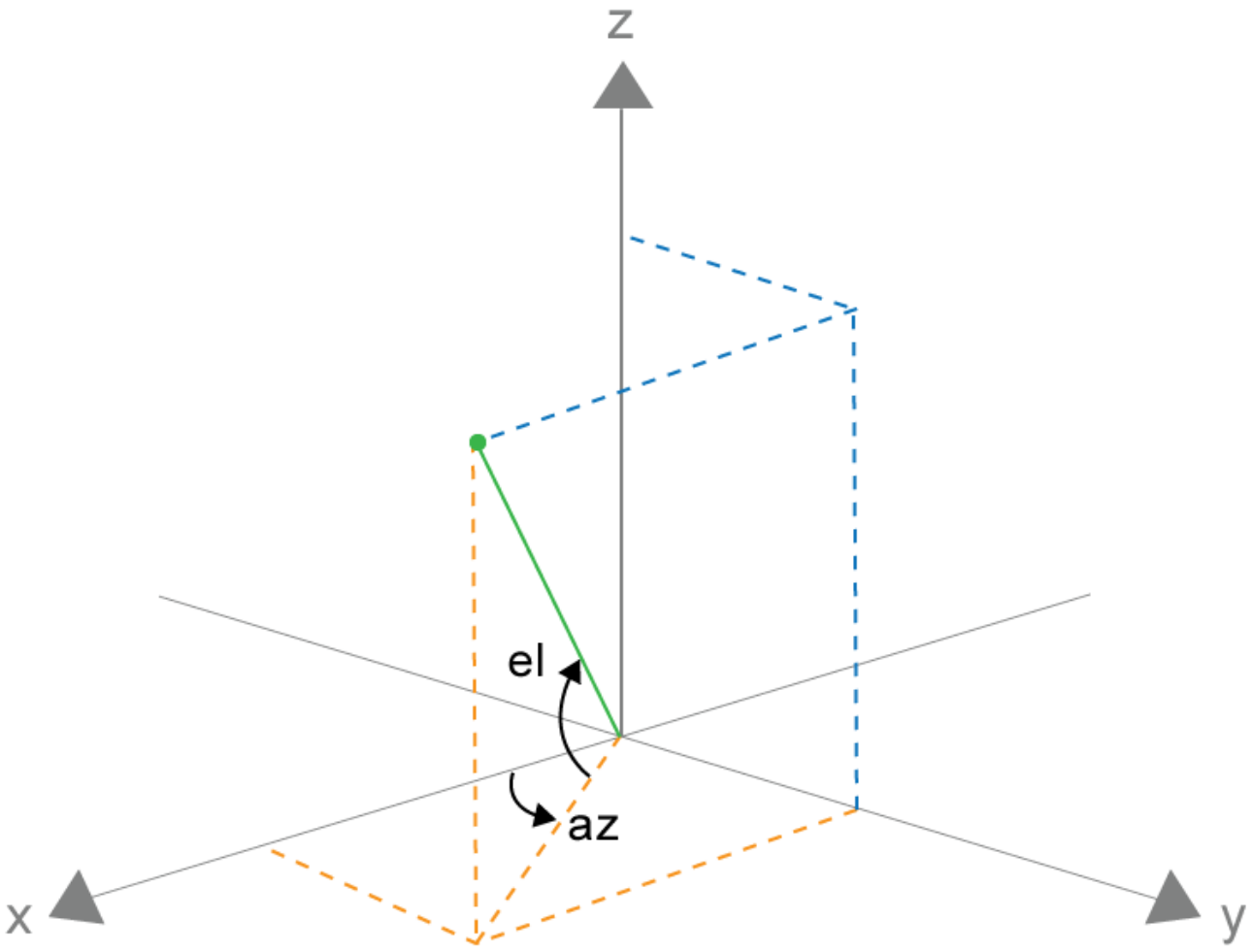
`Gmag` is of class `double`, unless the input image or any of the directional gradients are of class `single`. In this case, `Gmag` is of class `single`.

`Gazimuth` — Azimuthal angle

3-D numeric array

Azimuthal angle, returned as a 3-D numeric array of the same size as the gradient magnitude, `Gmag`. `Gazimuth` contains angles in degrees within the range [-180, 180] measured between positive *x*-axis and the projection of the point on the *x*-*y* plane.

`Gazimuth` is of class `double`, unless the input image or any of the directional gradients are of class `single`. In this case, `Gmag` is of class `single`.



Gazimuth and Gelevation

`Gelevation` — Gradient elevation
3-D numeric array

Gradient elevation, returned as a 3-D numeric array of the same size as the gradient magnitude, `Gmag`. `Gelevation` contains angles in degrees within the range [-90, 90] measured between the radial line and the *x*-*y* plane.

`Gelevation` is of class `double`, unless the input image or any of the directional gradients are of class `single`. In this case, `Gmag` is of class `single`.

Algorithms

`imgradient3` does not normalize the gradient output. If the range of the gradient output image has to match the range of the input image, consider normalizing the gradient image, depending on the `method` argument used. For example, with a Sobel kernel, the normalization factor is 1/44 and for Prewitt, the normalization factor is 1/18.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imgradient3` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- When generating code, the input argument `method` must be a compile-time constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, the input argument `method` must be a compile-time constant.

See Also

`imgradientxyz` | `imgradient` | `imgradientxy`

Introduced in R2016a

imgradientxy

Find directional gradients of 2-D image

Syntax

```
[Gx,Gy] = imgradientxy(I)
[Gx,Gy] = imgradientxy(I,method)
```

Description

`[Gx,Gy] = imgradientxy(I)` returns the directional gradients, `Gx` and `Gy` of the grayscale or binary image `I`.

`[Gx,Gy] = imgradientxy(I,method)` returns the directional gradients using the specified `method`.

Examples

Calculate Directional Gradients Using Prewitt Method

Read an image into workspace.

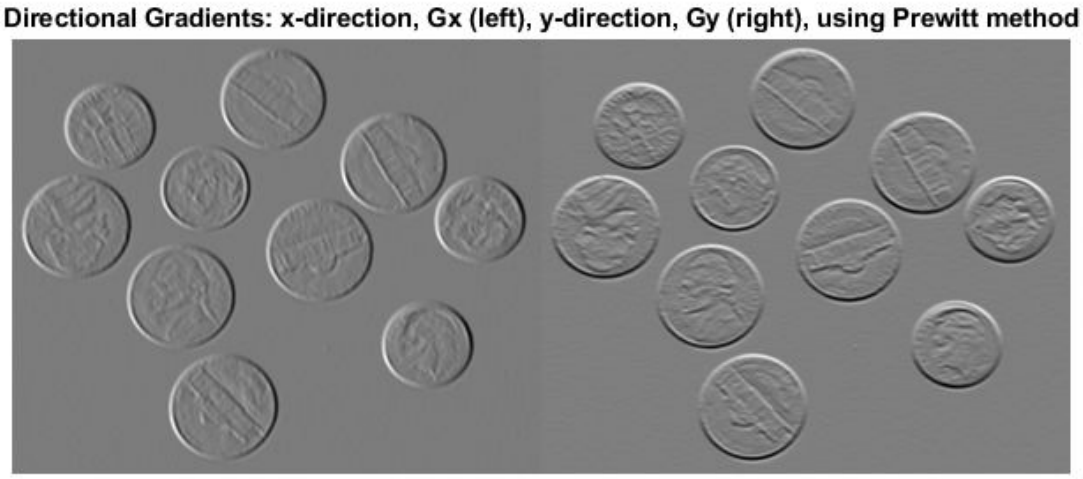
```
I = imread('coins.png');
```

Calculate the *x*- and *y*-directional gradients using the Prewitt gradient operator.

```
[Gx, Gy] = imgradientxy(I,'prewitt');
```

Display the directional gradients.

```
figure
imshowpair(Gx, Gy, 'montage');
title('Directional Gradients: x-direction, Gx (left), y-direction, Gy (right), using Prewitt metl
```



Calculate Gradient Magnitude and Direction Using Directional Gradients

Read an image into workspace.

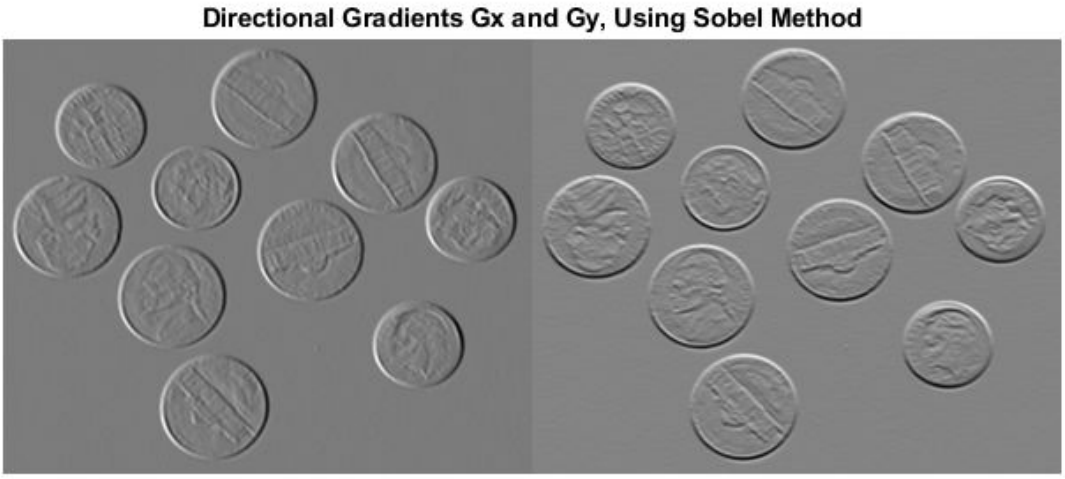
```
I = imread('coins.png');
```

Calculate the *x*- and *y*-directional gradients. By default, `imgradientxy` uses the Sobel gradient operator.

```
[Gx,Gy] = imgradientxy(I);
```

Display the directional gradients.

```
imshowpair(Gx,Gy,'montage')
title('Directional Gradients Gx and Gy, Using Sobel Method')
```

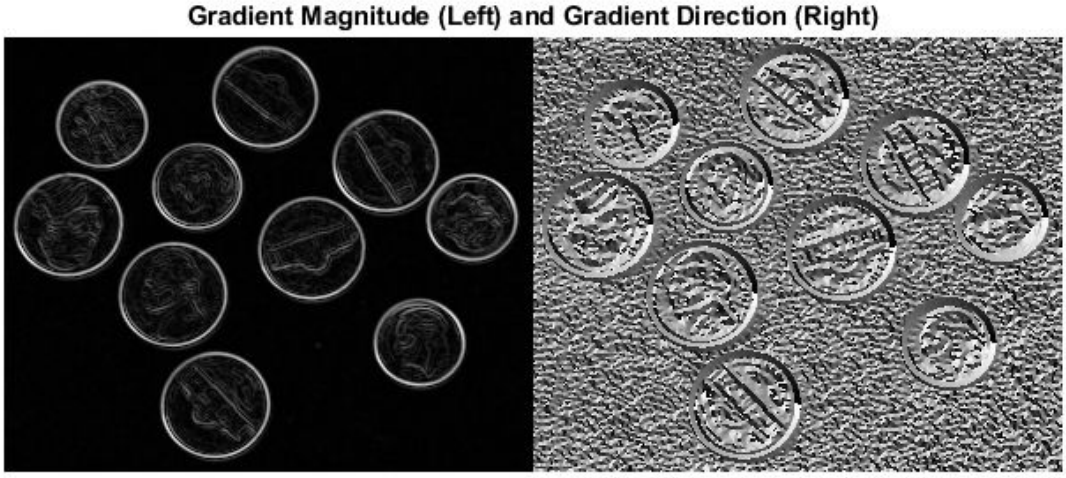


Calculate the gradient magnitude and direction using the directional gradients.

```
[Gmag,Gdir] = imgradient(Gx,Gy);
```

Display the gradient magnitude and direction.

```
imshowpair(Gmag,Gdir,'montage')
title('Gradient Magnitude (Left) and Gradient Direction (Right)')
```



Input Arguments

`I` — Input image

2-D grayscale image | 2-D binary image

Input image, specified as a 2-D grayscale image or 2-D binary image.

Data Types: `single` | `double` | `int8` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`method` — Gradient operator

`'sobel'` (default) | `'prewitt'` | `'central'` | `'intermediate'`

Gradient operator, specified as one of the following values.

Method	Description

`'sobel'`	Sobel gradient operator. The gradient of a pixel is a weighted sum of pixels in the 3-by-3 neighborhood. In the vertical (y) direction, the weights are:
	`[ 1  2  1`
	`  0  0  0`
	` -1 -2 -1 ]`
	In the x direction, the weights are transposed.

`'prewitt'`	Prewitt gradient operator. The gradient of a pixel is a weighted sum of pixels in the 3-by-3 neighborhood. In the vertical (y) direction, the weights are:
	`[ 1  1  1`
	`  0  0  0`
	` -1 -1 -1 ]`
	In the x direction, the weights are transposed.

`'central'`	Central difference gradient. The gradient of a pixel is a weighted difference of neighboring pixels. In the y direction, `dI/dy = (I(y+1) - I(y-1))/2`.
`'intermediate'`	Intermediate difference gradient. The gradient of a pixel is the difference between an adjacent pixel and the current pixel. In the y direction, `dI/dy = I(y+1) - I(y)`.

Data Types: `char` | `string`

Output Arguments

`Gx` — Horizontal gradient

numeric matrix

Horizontal gradient, returned as a numeric matrix of the same size as image `I`. The horizontal (x) axis points in the direction of increasing column subscripts. `Gx` is of data type `double`, unless the input image `I` is of data type `single`, in which case `Gx` is of data type `single`.

Data Types: `single` | `double`

`Gy` — Vertical gradient

numeric matrix

Vertical gradient, returned as a numeric matrix of the same size as image `I`. The vertical (y) axis points in the direction of increasing row subscripts. `Gy` is of data type `double`, unless the input image `I` is of data type `single`, in which case `Gy` is of data type `single`.

Data Types: `single` | `double`

Tips

- When applying the gradient operator at the boundaries of the image, values outside the bounds of the image are assumed to equal the nearest image border value.

Algorithms

The algorithmic approach is to compute directional gradients with respect to the *x*-axis and *y*-axis. The *x*-axis is defined along the columns going right and the *y*-axis is defined along the rows going down.

`imgradientxy` does not normalize the gradient output. If the range of the gradient output image has to match the range of the input image, consider normalizing the gradient image, depending on the `method` argument used. For example, with a Sobel kernel, the normalization factor is 1/8, and for Prewitt, it is 1/6.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imgradientxy` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imgradientxy` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- The value of `method` must be a compile time constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`edge` | `fspecial` | `imgradient` | `imgradient3` | `imgradientxyz`

Introduced in R2012b

imgradientxyz

Find directional gradients of 3-D image

Syntax

```
[Gx,Gy,Gz] = imgradientxyz(I)
[Gx,Gy,Gz] = imgradientxyz(I,method)
```

Description

`[Gx,Gy,Gz] = imgradientxyz(I)` returns the directional gradients `Gx`, `Gy`, and `Gz` of the 3-D grayscale or binary image `I`.

`[Gx,Gy,Gz] = imgradientxyz(I,method)` calculates the directional gradients using the specified `method`.

Examples

Compute 3-D Directional Image Gradients Using Sobel Method

Read 3-D data and prepare it for processing.

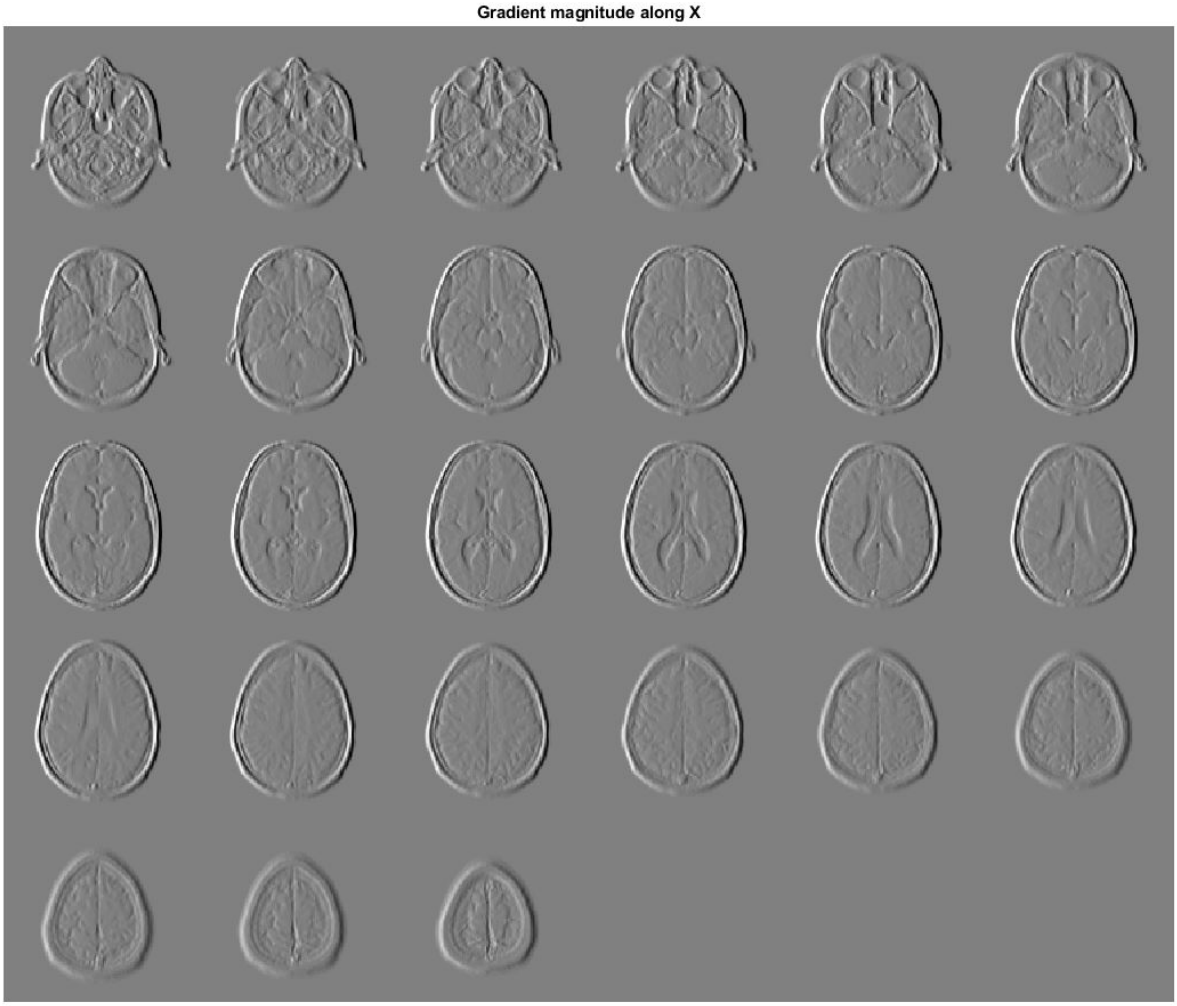
```
volData = load('mri');
sz = volData.siz;
vol = squeeze(volData.D);
```

Calculate the directional gradients.

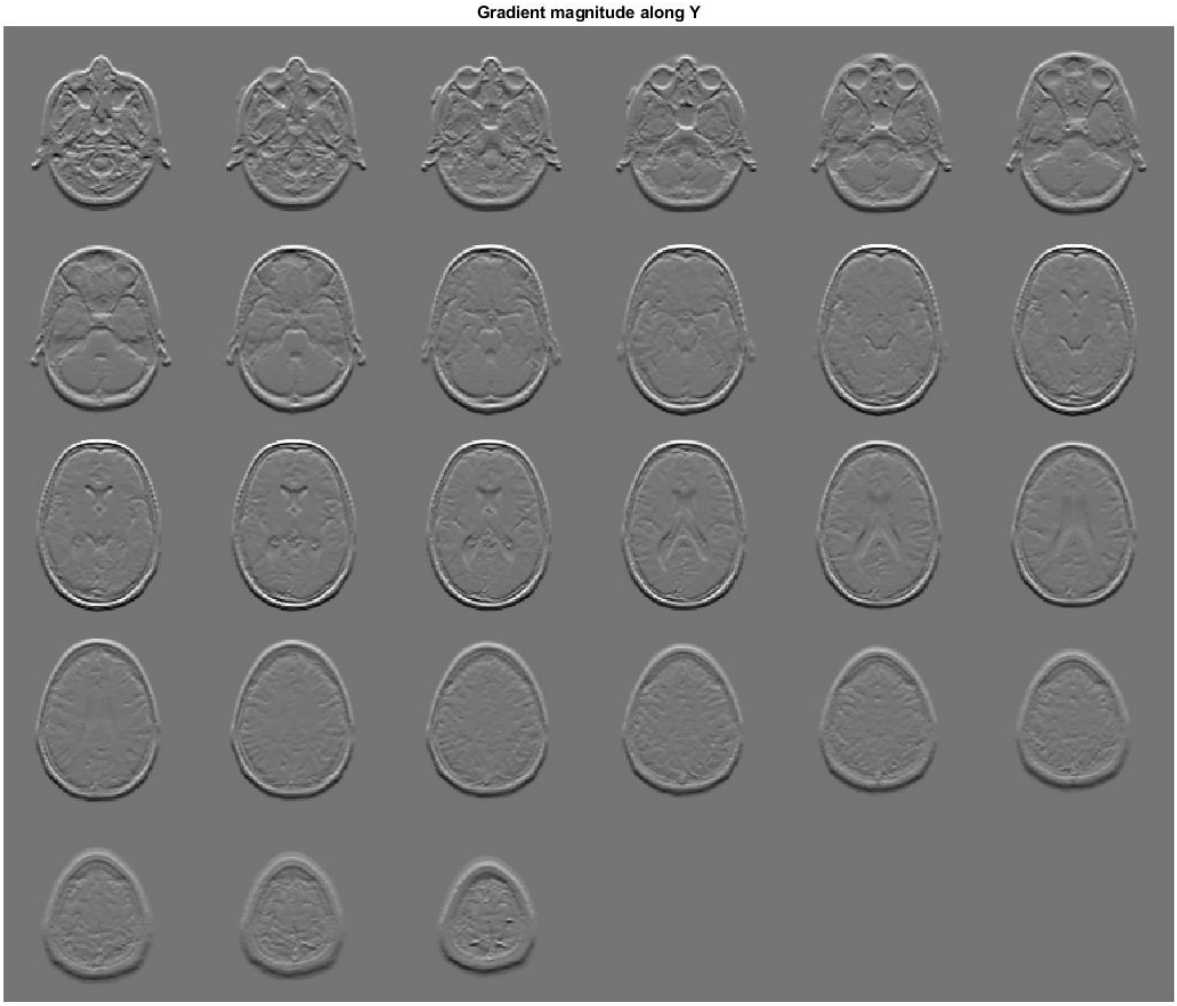
```
[Gx, Gy, Gz] = imgradientxyz(vol);
```

Visualize the directional gradients as a montage.

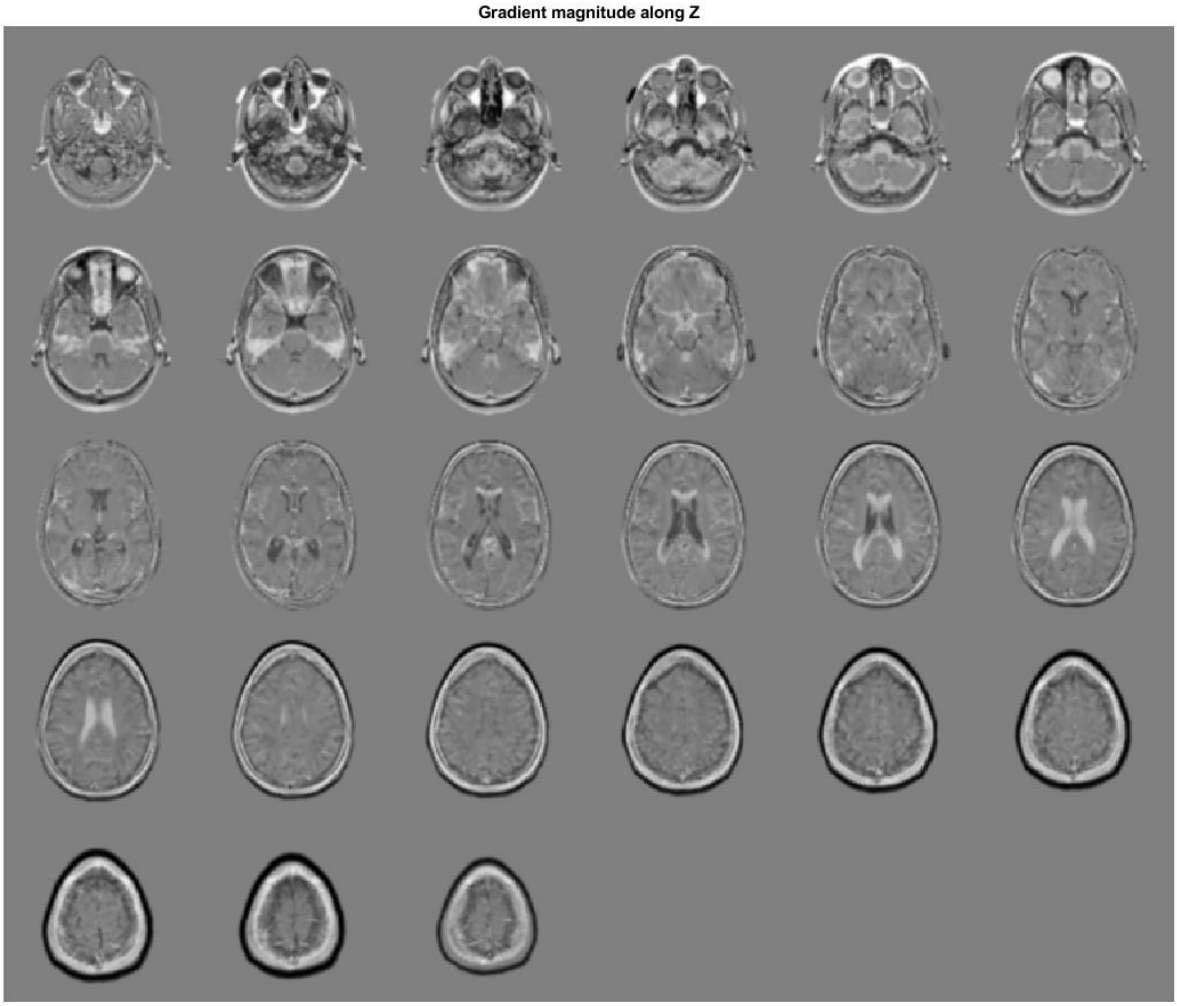
```
figure, montage(reshape(Gx,sz(1),sz(2),1,sz(3)),'DisplayRange',[])
title('Gradient magnitude along X')
```




```
figure, montage(reshape(Gy,sz(1),sz(2),1,sz(3)),'DisplayRange',[])
title('Gradient magnitude along Y')
```




```
figure, montage(reshape(Gz,sz(1),sz(2),1,sz(3)),'DisplayRange',[])
title('Gradient magnitude along Z')
```



Input Arguments

`I` — Input image

3-D grayscale image | 3-D binary image

Input image, specified as a 3-D grayscale image or 3-D binary image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`method` — Gradient operator

`'sobel'` (default) | `'prewitt'` | `'central'` | `'intermediate'`

Gradient operator, specified as one of the following values.

<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>'sobel'</td><td>Sobel gradient operator. The gradient of a pixel is a weighted sum of pixels in the 3-by-3-by-3 neighborhood. For example, in the depth (z) direction, the weights in the three planes are:
<table>
<tr><th>plane z-1</th><th>plane z</th><th>plane z+1</th></tr>
<tr><td>[1 3 1
3 6 3
1 3 1]</td><td>[0 0 0
0 0 0
0 0 0]</td><td>[-1 -3 -1
-3 -6 -3
-1 -3 -1]</td></tr>
</table></td></tr>
<tr><td>'prewitt'</td><td>Prewitt gradient operator. The gradient of a pixel is a weighted sum of pixels in the 3-by-3-by-3 neighborhood. For example, in the depth (z) direction, the weights in the three planes are:
<table>
<tr><th>plane z-1</th><th>plane z</th><th>plane z+1</th></tr>
<tr><td>[1 1 1
1 1 1
1 1 1]</td><td>[0 0 0
0 0 0
0 0 0]</td><td>[-1 -1 -1
-1 -1 -1
-1 -1 -1]</td></tr>
</table></td></tr>
<tr><td>'central'</td><td>Central difference gradient. The gradient of a pixel is a weighted difference of neighboring pixels. For example, in the depth (z) direction, <code>dI/dz = (I(z+1) - I(z-1))/2</code>.</td></tr>
<tr><td>'intermediate'</td><td>Intermediate difference gradient. The gradient of a pixel is the difference between an adjacent pixel and the current pixel. For example, in the depth (z) direction, <code>dI/dz = I(z+1) - I(z)</code>.</td></tr>
</table>

When applying the gradient operator at the boundaries of the image, `imgradientxyz` assumes values outside the bounds of the image are equal to the nearest image border value. This behavior is similar to the `'replicate'` boundary option in `imfilter`.

Data Types: `char` | `string`

Output Arguments

`Gx` — Horizontal gradient

3-D numeric array

Horizontal gradient, returned as a numeric matrix of the same size as image `I`. The horizontal (*x*) axis points in the direction of increasing column subscripts. `Gx` is of class `double`, unless the input image `I` is of class `single`, in which case `Gx` is of class `single`.

Data Types: `single` | `double`

`Gy` — Vertical gradient

3-D numeric array

Vertical gradient, returned as a numeric matrix of the same size as image `I`. The vertical (*y*) axis points in the direction of increasing row subscripts. `Gy` is of class `double`, unless the input image `I` is of class `single`, in which case `Gy` is of class `single`.

Data Types: `single` | `double`

`Gz` — Depth gradient

3-D numeric array

Depth gradient, returned as a 3-D numeric array of the same size as image `I`. The depth (*z*) axis points in the direction of increasing plane subscripts. `Gz` is of class `double`, unless the input image `I` is of class `single`, in which case `Gz` is of class `single`.

Algorithms

`imgradientxyz` does not normalize the gradient output. If the range of the gradient output image has to match the range of the input image, consider normalizing the gradient image, depending on the `method` argument used. For example, with a Sobel kernel, the normalization factor is 1/44, for Prewitt, the normalization factor is 1/18.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imgradientxyz` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- When generating code, the input argument `method` must be a compile-time constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, the input argument `method` must be a compile-time constant.

See Also

`imgradient3` | `imgradient` | `imgradientxy`

Introduced in R2016a

imguidedfilter

Guided filtering of images

Syntax

```
B = imguidedfilter(A,G)
B = imguidedfilter(A)
B = imguidedfilter( ___ ,Name,Value)
```

Description

`B = imguidedfilter(A,G)` filters binary, grayscale, or RGB image `A` using a filter guided by image `G`.

`B = imguidedfilter(A)` filters input image `A` under self-guidance, using `A` itself as the guidance image. This syntax can be used for edge-preserving smoothing of image `A`.

`B = imguidedfilter( ___ ,Name,Value)` filters the image `A` using name-value arguments to control aspects of guided filtering.

Examples

Perform Edge-Preserving Smoothing Using Guided Filtering

Read and display an image.

```
A = imread('pout.tif');
imshow(A)
```

Smooth the image using `imguidedfilter`. In this syntax, `imguidedfilter` uses the image itself as the guidance image.

```
Iguided = imguidedfilter(A);
```

For comparison, smooth the original image using a Gaussian filter defined by `imgaussfilt`. Set the standard deviation of the filter to 2.5 so that the degree of smoothing approximately matches that of the guided filter.

```
Igaussian = imgaussfilt(A,2);
```

Display the result of guided filtering and the result of Gaussian filtering. Observe that the flat regions of the two filtered images, such as the jacket and the face, have similar amounts of smoothing. However, the guided filtered image better preserves the sharpness of edges, such as around the trellis and the collar of the white shirt.

```
montage({Iguided,Igaussian})
```

Input Arguments

`A` — Image to be filtered

binary image | grayscale image | RGB image

Image to be filtered, specified as a binary, grayscale, or RGB image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`G` — Guide image

binary image | grayscale image | RGB image

Guide image, specified as a binary, grayscale, or RGB image of the same height and width as image `A`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Ismooth = imguidedfilter(A,NeighborhoodSize=[4 4]);` performs guided filtering using a neighborhood of size 4-by-4 pixels.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `Ismooth = imguidedfilter(A,"NeighborhoodSize",[4 4]);`

`NeighborhoodSize` — Size of rectangular neighborhood
`[5 5]` (default) | positive integer | 2-element vector of positive integers

Size of the rectangular neighborhood around each pixel used in guided filtering, specified as a positive integer or a 2-element vector of positive integers. If you specify a scalar value, such as `Q`, then the neighborhood is a square of size `[Q Q]`. Do not specify a value greater than the size of the image.

Example: `NeighborhoodSize=[7 7]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`DegreeOfSmoothing` — Amount of smoothing
positive number

Amount of smoothing in the output image, specified as a positive number. If you specify a small value, only neighborhoods with small variance (uniform areas) will get smoothed and neighborhoods with larger variance (such as around edges) will not be smoothed. If you specify a larger value, high variance neighborhoods, such as stronger edges, will get smoothed in addition to the relatively uniform neighborhoods. Start with the default value, check the results, and adjust the default up or down to achieve the effect you desire.

If you specify a guide image `G`, then the default value of `DegreeOfSmoothing` depends on the data type of `G`, and is calculated as `0.01*diff(getrangefromclass(G)).^2`. For example, the default degree of smoothing is `650.25` for images of data type `uint8`, and the default is `0.01` for images of data type `double` with pixel values in the range [0, 1]. If you do not specify a guide image, then the default value depends on the data type of image `A`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`B` — Filtered image
numeric array

Filtered image, returned as a numeric array of the same size and data type as `A`

Tips

- The `DegreeOfSmoothing` argument specifies a soft threshold on variance for the given neighborhood. If a pixel's neighborhood has variance much lower than the threshold, it will see some amount of smoothing. If a pixel's neighborhood has variance much higher than the threshold it will have little to no smoothing.
- Input images `A` and `G` can be of different classes. If either `A` or `G` is of class integer or logical, then `imguidedfilter` converts them to floating-point precision for internal computation.
- Input images `A` and `G` can have different number of channels.
  - If both `A` and `G` are RGB images, then `imguidedfilter` filters each channel of `A` independently using the corresponding channel of `G`.
  - If `A` is an RGB image and `G` is a single-channel image, then `imguidedfilter` filters each channel of `A` independently using the same guidance image, `G`.
  - If `A` is a single-channel image and `G` is an RGB image, then `imguidedfilter` filters `A` using the combined color statistics of all the three channels of `G`.

References

[1] Kaiming He, Jian Sun, Xiaoou Tang. *Guided Image Filtering*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 35, Issue 6, pp. 1397-1409, June 2013.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`imguidedfilter` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

See Also

`edge` | `imdiffuseest` | `imfilter` | `imgaussfilt` | `locallapfilt` | `imnlmfilt` | `imsharpen`

Topics

“Perform Flash/No-flash Denoising with Guided Filter”
“What is Guided Image Filtering?”

Introduced in R2014a

imhandles

Get all image objects

Syntax

```
himage = imhandles(hparent)
```

Description

`himage = imhandles(hparent)` returns all of the image objects whose ancestor is `hparent`. `imhandles` returns an error if the image objects do not have the same figure as their parent. `imhandles` ignores color bars.

Examples

Get Images From Parent Graphics Objects

Display an image.

```
imshow('kobi.png')
```

Return the image object in the current axes.

```
imageobj = imhandles(gca)
```

```
imageobj =
  Image with properties:

            CData: [1224x1632x3 uint8]
    CDataMapping: 'direct'

  Show all properties
```

Display two images in the same figure and use `imhandles` to get both of the image objects in the figure.

```
figure
subplot(1,2,1)
imshow('kobi.png')
subplot(1,2,2)
imshow('sherlock.jpg')
```

```
imageobjs = imhandles(gcf)

imageobjs =
  2x1 Image array:

  Image
  Image
```

Inspect the first image in the `imageobjs` array.

```
imageobjs(1)

ans =
  Image with properties:

            CData: [640x960x3 uint8]
    CDataMapping: 'direct'

  Show all properties
```

Input Arguments

`hparent` — Parent graphics object

figure | axes | uipanel | image

Parent graphics object, specified as a handle to a figure, axes, uipanel, or image graphics objects.

Output Arguments

`himage` — Image objects

image | array of images

Image objects whose ancestor is `hparent`, returned as an image or array of images.

See Also

`imgca` | `imgcf`

Introduced before R2006a

imhist

Histogram of image data

Syntax

```
[counts,binLocations] = imhist(I)
[counts,binLocations] = imhist(I,n)
[counts,binLocations] = imhist(X,cmap)

imhist( ___ )
```

Description

`[counts,binLocations] = imhist(I)` calculates the histogram for the grayscale image `I`. The `imhist` function returns the histogram counts in `counts` and the bin locations in `binLocations`. The number of bins in the histogram is determined by the image type.

`[counts,binLocations] = imhist(I,n)` specifies the number of bins, `n`, used to calculate the histogram.

`[counts,binLocations] = imhist(X,cmap)` calculates the histogram for the indexed image `X` with colormap `cmap`. The histogram has one bin for each entry in the colormap.

`imhist( ___ )` displays a plot of the histogram. If the input image is an indexed image, then the histogram shows the distribution of pixel values above a color bar of the colormap `cmap`.

Examples

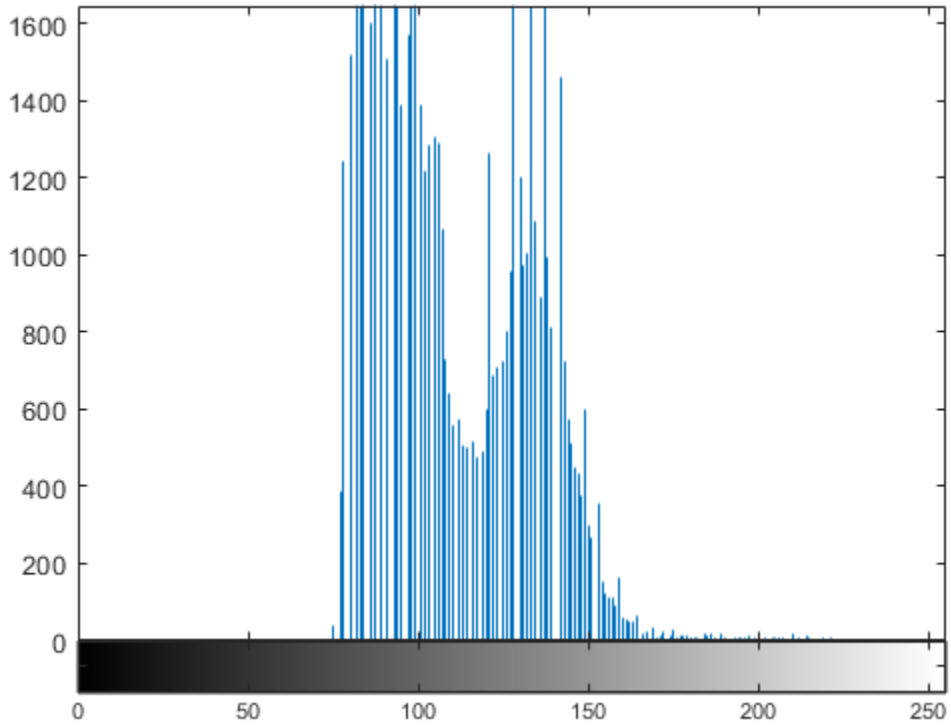
Calculate Histogram

Read a grayscale image into the workspace.

```
I = imread('pout.tif');
```

Display a histogram of the image. Since `I` is grayscale, by default the histogram will have 256 bins.

```
imhist(I)
```



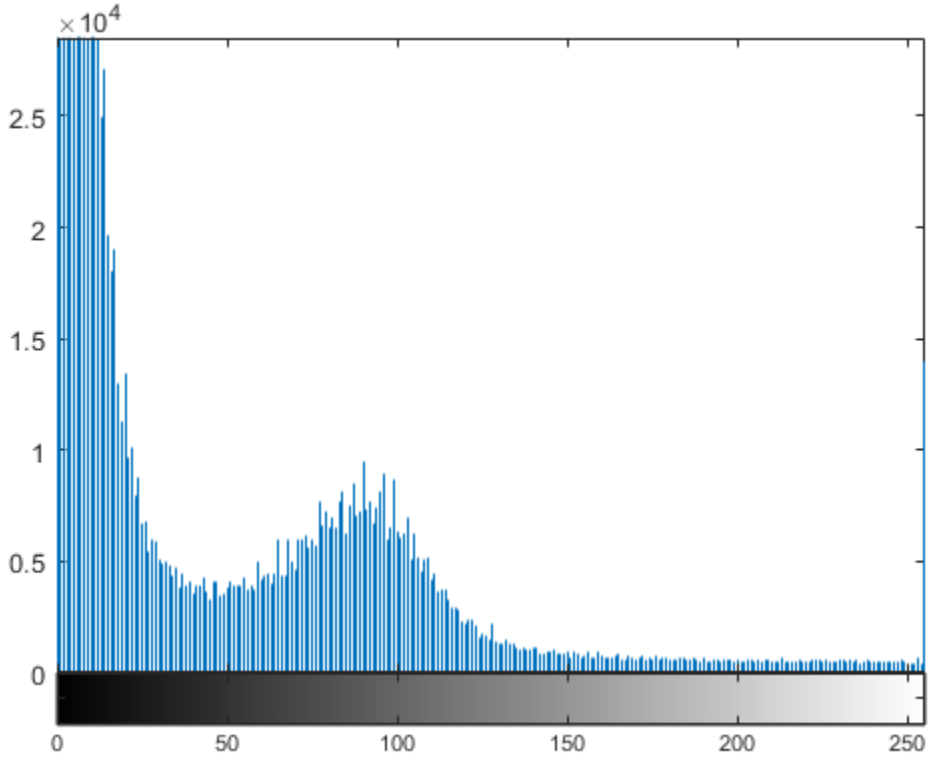
Display the Histogram of a 3-D Intensity Image

Load a 3-D dataset.

```
load mristack
```

Display the histogram of the data. Since the image is grayscale, `imhist` uses 256 bins by default.

```
imhist(mristack)
```



Input Arguments

`I` — Grayscale image

numeric array

Grayscale image, specified as a numeric array of any dimension. If the image has data type `single` or `double`, then values must be in the range [0, 1]. If `I` has values outside the range [0, 1], then you can use the `rescale` function to rescale values to the expected range.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`n` — Number of bins

positive integer

Number of bins, specified as a positive integer. If `I` is a grayscale image, then `imhist` uses a default value of 256 bins. If `I` is a binary image, then `imhist` uses two bins.

Example: `50`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`X` — Indexed image

numeric array

Indexed image, specified as a numeric array of any dimension.

Data Types: `single` | `double` | `uint8` | `uint16` | `logical`

`cmap` — Colormap
c-by-3 numeric matrix

Colormap associated with indexed image `X`, specified as a *c*-by-3 numeric matrix with values in the range [0, 1]. Each row is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap.The colormap must be at least as long as the largest index in `X`.

Data Types: `double`

Output Arguments

`counts` — Histogram counts
numeric array

Histogram counts, returned as a numeric array. If the histogram is computed for an indexed image, `X`, then the length of `counts` is the same as the length of the colormap, `cmap`.

`binLocations` — Bin locations
numeric array

Bin locations, returned as a numeric array.

Tips

- For grayscale images, the `n` bins of the histogram are each half-open intervals of width $A/(n-1)$. In particular, the pth bin is the half-open interval

  $$\frac{A(p-1.5)}{(n-1)} - B \le x < \frac{A(p-0.5)}{(n-1)} - B,$$

  where x is the intensity value. The scale factor A and offset B depend on the type of the image class as follows:

Data Type	*A*	*B*
`double`	1	0
`single`	1	0
`int8`	255	128
`int16`	65,535	32,768
`int32`	4,294,967,295	2,147,483,648
`uint8`	255	0
`uint16`	65,535	0
`uint32`	4,294,967,295	0
`logical`	1	0

- To display the histogram from `counts` and `binLocations`, use the command `stem(binLocations,counts)`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imhist` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imhist` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- If the first input is a binary image, then `n` must be a scalar constant of value 2 at compile time.
- Nonprogrammatic syntaxes are not supported. For example, the syntax `imhist(I)`, where `imhist` displays the histogram, is not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- If the first input is a binary image, then `n` must be a scalar constant of value 2 at compile time.
- Nonprogrammatic syntaxes are not supported. For example, the syntax `imhist(I)`, where `imhist` displays the histogram, is not supported.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- `imhist` does not support indexed images on a GPU.
- When you omit output arguments on a GPU, `imhist` does not display the histogram. In this case, the function returns the histogram counts in the variable `ans` and does not return the histogram bin locations.

For more information, see “Image Processing on a GPU”.

See Also

`histeq` | `histogram` | `stem`

Introduced before R2006a

imhistmatch

Adjust histogram of 2-D image to match histogram of reference image

Syntax

```
J = imhistmatch(I,ref)
J = imhistmatch(I,ref,nbins)
J = imhistmatch( ___ ,Name,Value)
[J,hgram] = imhistmatch( ___ )
```

Description

`J = imhistmatch(I,ref)` transforms the 2-D grayscale or truecolor image `I` returning output image `J` whose histogram approximately matches the histogram of the reference image `ref`.

- If both `I` and `ref` are truecolor images, then `imhistmatch` matches each color channel of `I` independently to the corresponding color channel of `ref`.
- If `I` is a truecolor RGB image and `ref` is a grayscale image, then `imhistmatch` matches each channel of `I` against the single histogram derived from `ref`.
- If `I` is a grayscale image, then `ref` must also be a grayscale image.

Images `I` and `ref` can be any of the permissible data types and need not be equal in size.

`J = imhistmatch(I,ref,nbins)` uses `nbins` equally spaced bins within the appropriate range for the given image data type. The returned image `J` has no more than `nbins` discrete levels.

- If the data type of the image is either `single` or `double`, then the histogram range is [0, 1].
- If the data type of the image is `uint8`, then the histogram range is [0, 255].
- If the data type of the image is `uint16`, then the histogram range is [0, 65535].
- If the data type of the image is `int16`, then the histogram range is [-32768, 32767].

`J = imhistmatch( ___ ,Name,Value)` uses name-value pairs to change the behavior of the histogram matching algorithm.

`[J,hgram] = imhistmatch( ___ )` returns the histogram of the reference image `ref` used for matching in `hgram`. `hgram` is a 1-by-`nbins` (when `ref` is grayscale) or a 3-by-`nbins` (when `ref` is truecolor) matrix, where `nbins` is the number of histogram bins. Each row in `hgram` stores the histogram of a single color channel of `ref`.

Examples

Match Histogram of Aerial Images

These aerial images, taken at different times, represent overlapping views of the same terrain in Concord, Massachusetts. This example demonstrates that input images `A` and `Ref` can be of different sizes and image types.

Load an RGB image and a reference grayscale image.

```
A = imread('concordaerial.png');
Ref = imread('concordorthophoto.png');
```

Get the size of A.

```
size(A)
```

```
ans = 1×3

        2036        3060           3
```

Get the size of Ref.

```
size(Ref)
```

```
ans = 1×2

        2215        2956
```

Note that image A and Ref are different in size and type. Image A is a truecolor RGB image, while image Ref is a grayscale image. Both images are of data type uint8.

Generate the histogram matched output image. The example matches each channel of A against the single histogram of Ref. Output image B takes on the characteristics of image A - it is an RGB image whose size and data type is the same as image A. The number of distinct levels present in each RGB channel of image B is the same as the number of bins in the histogram built from grayscale image Ref. In this example, the histogram of Ref and B have the default number of bins, 64.

```
B = imhistmatch(A,Ref);
```

Display the RGB image A, the reference image Ref, and the histogram matched RGB image B. The images are resized before display.

```
imshow(A)
title('RGB Image with Color Cast')
```

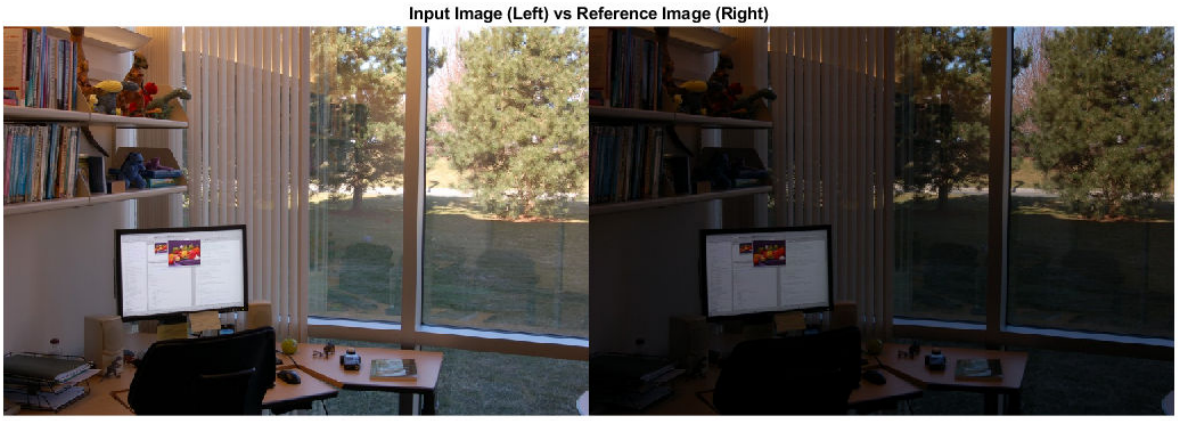

```
imshow(Ref)
title('Reference Grayscale Image')
```


```
imshow(B)
title('Histogram Matched RGB Image')
```

Histogram-Match Image Using Polynomial Method

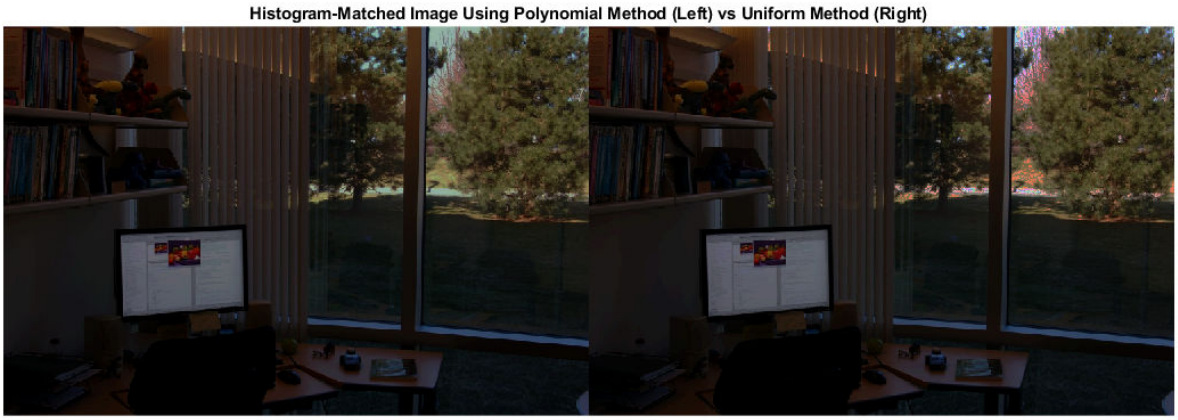
Read a color image and a reference image. To demonstrate the polynomial method, assign the reference image to be the darker of the two images.

```
I = imread('office_4.jpg');
ref = imread('office_2.jpg');
montage({I,ref})
title('Input Image (Left) vs Reference Image (Right)');
```



Use the polynomial method to adjust the intensity of image `I` so that it matches the histogram of reference image `ref`. For comparison, also adjust the intensity of image `I` using the uniform method.

```
J = imhistmatch(I,ref,'method','polynomial');
K = imhistmatch(I,ref,'method','uniform');
montage({J,K})
title('Histogram-Matched Image Using Polynomial Method (Left) vs Uniform Method (Right)');
```



The histogram-matched image using the uniform method introduces false colors in the sky and road. The histogram-matched image using the polynomial method does not exhibit this artifact.

Match Histogram with Multiple Bin Values

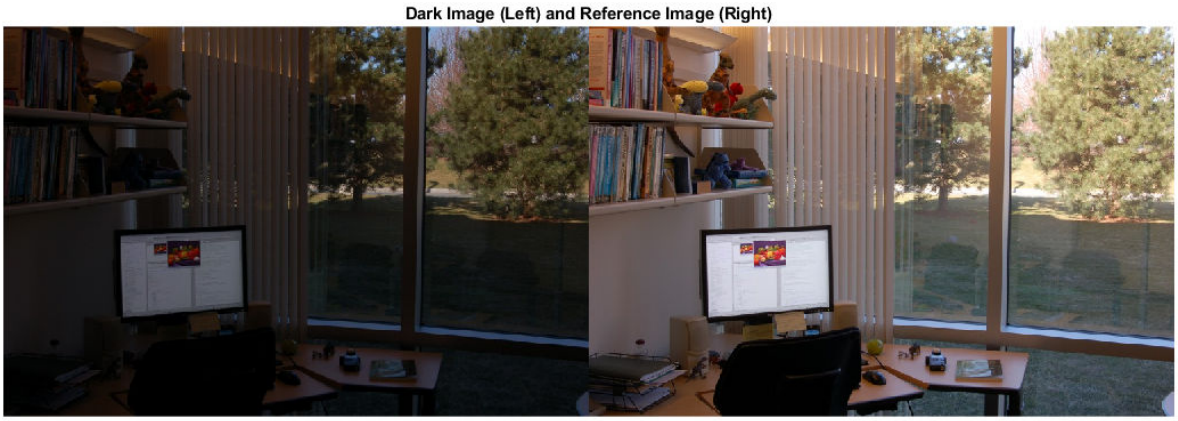
This example shows how you can vary the number of bins in the target histogram to improve histogram equalization.

Load two images of data type `uint8` into the workspace. The images were taken with a digital camera and represent two different exposures of the same scene. `A` is an underexposed image and appears dark. `ref` is a reference image with good exposure and brightness.

```
A = imread('office_2.jpg');
ref = imread('office_4.jpg');
```

Display the images in a montage.

```
montage({A,ref})
title('Dark Image (Left) and Reference Image (Right)')
```



Display the histogram of each color channel using 256 bins. You can use the helper function, `displayHistogramChannels`, that is included with the example.

```
displayHistogramChannels(A,ref)
```

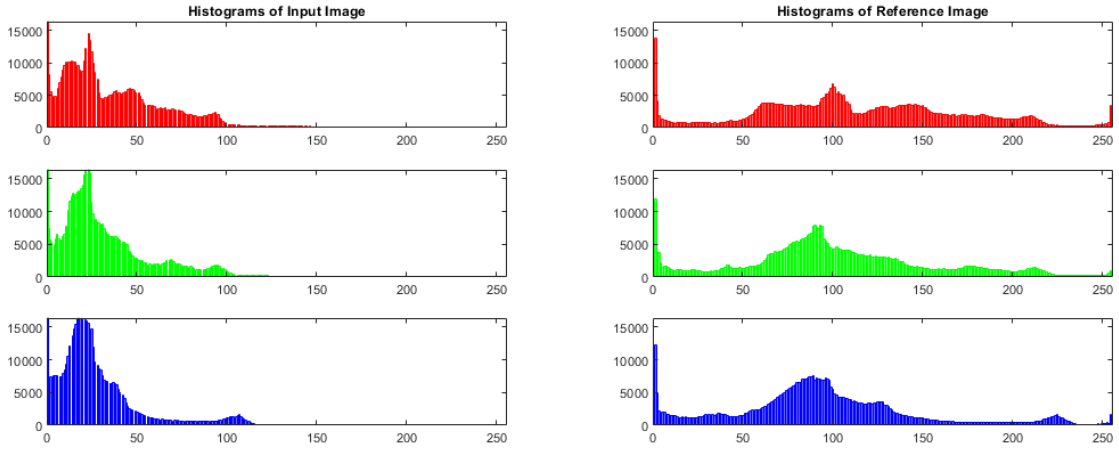


Image `A`, being the darker image, has most of its pixels in the lower bins. The reference image, `ref`, fully populates all 256 bins values in all three RGB channels.

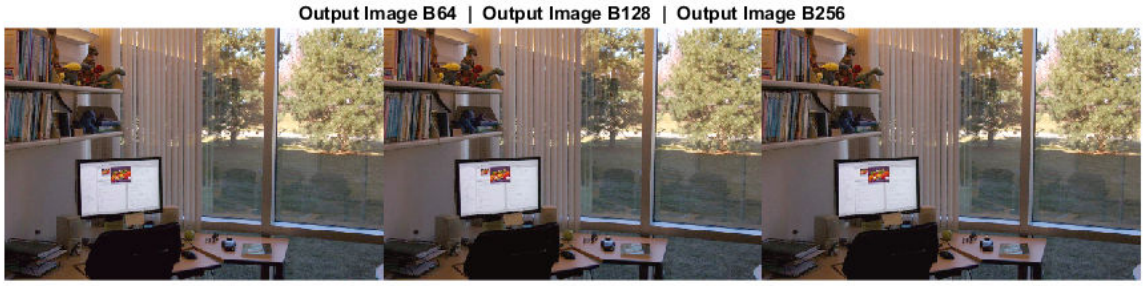
Count the number of unique 8-bit level values for each color channel of the dark and reference image. You can use the helper function, `countUniqueValues`, that is included with the example.

```
numVals = countUniqueValues(A,ref);
table(numVals(:,1),numVals(:,2),numVals(:,3), ...
  'VariableNames',["Red" "Green" "Blue"], ...
  'RowNames',["A" "ref"])
```

```
ans=2×3 table
           Red    Green    Blue
           ___    _____    ____

    A      205     193     224
    ref    256     256     256
```

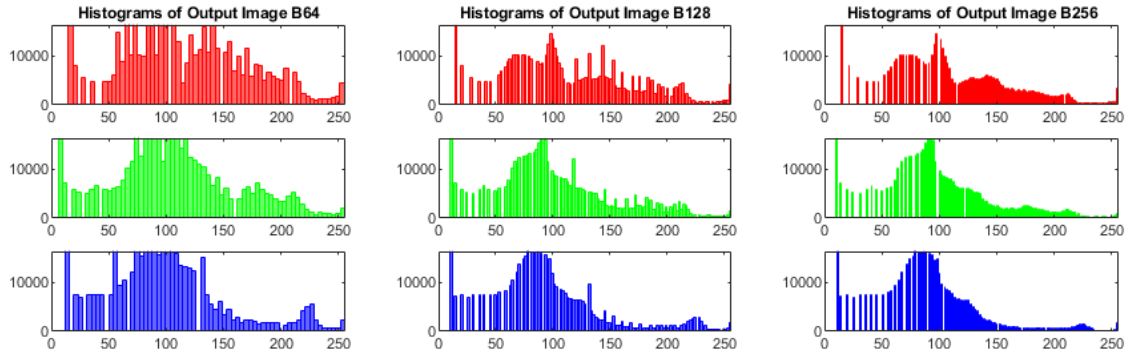
Equalize the histogram of the dark image using three different values of `nbins`: 64, 128 and 256. 64 is the default number of bins and 256 is the maximum number of bins for `uint8` pixel data.

```
[B64,hgram64]   = imhistmatch(A,ref,64);
[B128,hgram128] = imhistmatch(A,ref,128);
[B256,hgram256] = imhistmatch(A,ref,256);

figure
montage({B64,B128,B256},'Size',[1 3])
title('Output Image B64  |  Output Image B128  |  Output Image B256')
```



Display the histogram of each color channel using 256 bins. You can use the helper function, `displayThreeHistogramChannels`, that is included with the example.

```
displayThreeHistogramChannels(B64,B128,B256)
```



Count the number of unique 8-bit level values for each color channel of the three histogram equalized images. As `nbins` increases, the number of levels in each RGB channel of output image `B` also increases.

```
numVals = countUniqueValues(B64,B128,B256);
table(numVals(:,1),numVals(:,2),numVals(:,3), ...
  'VariableNames',["Red" "Green" "Blue"], ...
  'RowNames',["B64" "B128" "B256"])
```

```
ans=3×3 table
            Red    Green    Blue
            ___    _____    ____

    B64      57      60      58
    B128    101     104     104
    B256    134     135     136
```

Match Histogram of 16-Bit Grayscale MRI Image

This example shows how to perform histogram matching with different numbers of bins.

Load a 16-bit DICOM image of a knee imaged via MRI.

```
K = dicomread('knee1.dcm');   % read in original 16-bit image
LevelsK = unique(K(:));       % determine number of unique code values
disp(['image K: ',num2str(length(LevelsK)),' distinct levels']);
```

```
image K: 448 distinct levels
```

```
disp(['max level = ' num2str( max(LevelsK) )]);
```

```
max level = 473
```

```
disp(['min level = ' num2str( min(LevelsK) )]);
```

```
min level = 0
```

All 448 discrete values are at low code values, which causes the image to look dark. To rectify this, scale the image data to span the entire 16-bit range of [0, 65535].

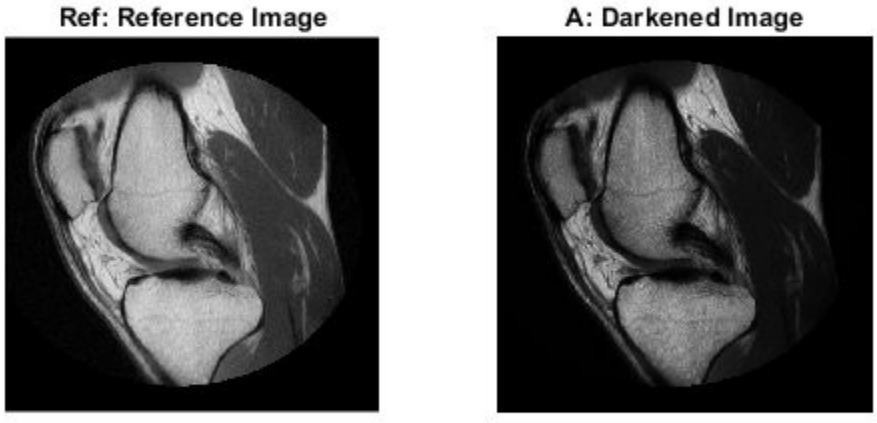
```
Kdouble = double(K);                    % cast uint16 to double
kmult = 65535/(max(max(Kdouble(:)))); % full range multiplier
Ref = uint16(kmult*Kdouble);   % full range 16-bit reference image
```

Darken the reference image `Ref` to create an image `A` that can be used in the histogram matching operation.

```
%Build concave bow-shaped curve for darkening |Ref|.
ramp = [0:65535]/65535;
ppconcave = spline([0 .1 .50  .72 .87 1],[0 .025 .25 .5 .75 1]);
Ybuf = ppval( ppconcave, ramp);
Lut16bit = uint16( round( 65535*Ybuf ) );
% Pass image |Ref| through a lookup table (LUT) to darken the image.
A = intlut(Ref,Lut16bit);
```

View the reference image `Ref` and the darkened image `A`. Note that they have the same number of discrete code values, but differ in overall brightness.

```
subplot(1,2,1)
imshow(Ref)
title('Ref: Reference Image')
subplot(1,2,2)
imshow(A)
title('A: Darkened Image');
```

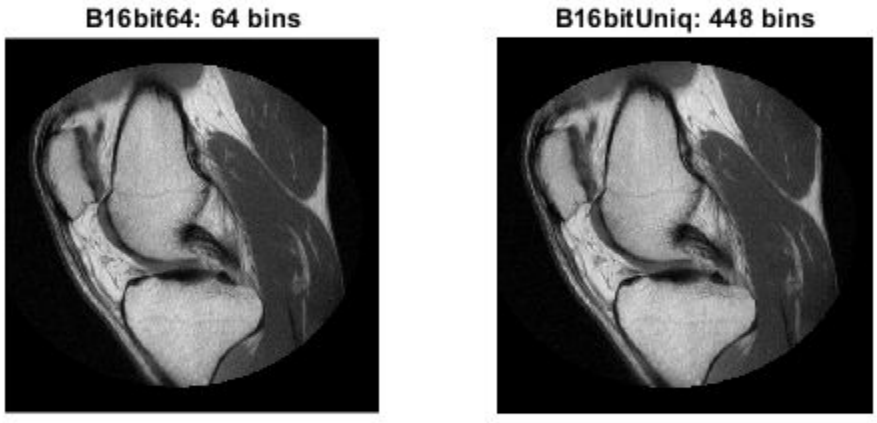


Generate histogram-matched output images using histograms with different number of bins. First use the default number of bins, 64. Then use the number of values present in image `A`, 448 bins.

```
B16bit64 = imhistmatch(A(:,:,1),Ref(:,:,1));  % default: 64 bins

N = length(LevelsK);     % number of unique 16-bit code values in image A.
B16bitUniq = imhistmatch(A(:,:,1),Ref(:,:,1),N);
```

View the results of the two histogram matching operations.

```
figure
subplot(1,2,1)
imshow(B16bit64)
title('B16bit64: 64 bins')
subplot(1,2,2)
imshow(Ref)
title(['B16bitUniq: ',num2str(N),' bins'])
```



Input Arguments

`I` — Input image

2-D truecolor image | 2-D grayscale image

Input image to be transformed, specified as a 2-D truecolor or grayscale image. The returned image will take the data type class of the input image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`ref` — Reference image whose histogram is the reference histogram

2-D truecolor image | 2-D grayscale image

Reference image whose histogram is the reference histogram, specified as a 2-D truecolor or grayscale image. The reference image provides the equally spaced `nbins` bin reference histogram which output image `J` is trying to match.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`nbins` — Number of equally spaced bins in reference histogram
64 (default) | positive integer

Number of equally spaced bins in reference histogram, specified as a positive integer. In addition to specifying the number of equally spaced bins in the histogram for image `ref`, `nbins` also represents the upper limit of the number of discrete data levels present in output image `J`.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `imhistmatch(I,ref,'Method','polynomial')` matches the histogram of image `I` to that of reference image `ref` using the polynomial mapping technique.

`Method` — Mapping technique
`'uniform'` (default) | `'polynomial'`

Mapping technique used to map the histogram of `ref` to image `I`, specified as the comma-separated pair consisting of `'Method'` and one of these values:

- `'uniform'` — Use a histogram-based intensity function and histogram equalization.
- `'polynomial'` — Calculate a cubic Hermite polynomial mapping function from the cumulative histograms of the source and reference images. The polynomial method is useful when the reference image is darker than the input image. In this situation, the polynomial method gives a smoother color transition than the uniform method.

Output Arguments

`J` — Output image
2-D truecolor RGB image | 2-D grayscale image

Output image, returned as a 2-D truecolor or grayscale image. The output image is derived from image `I` whose histogram is an approximate match to the histogram of input image `ref` built with `nbins` equally-spaced bins. Image `J` is of the same size and data type as input image `I`. Input argument `nbins` represents the upper limit of the number of discrete levels contained in image `J`.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`hgram` — Histogram counts derived from reference image `ref`
1-by-`nbins` vector | 3-by-`nbins` matrix

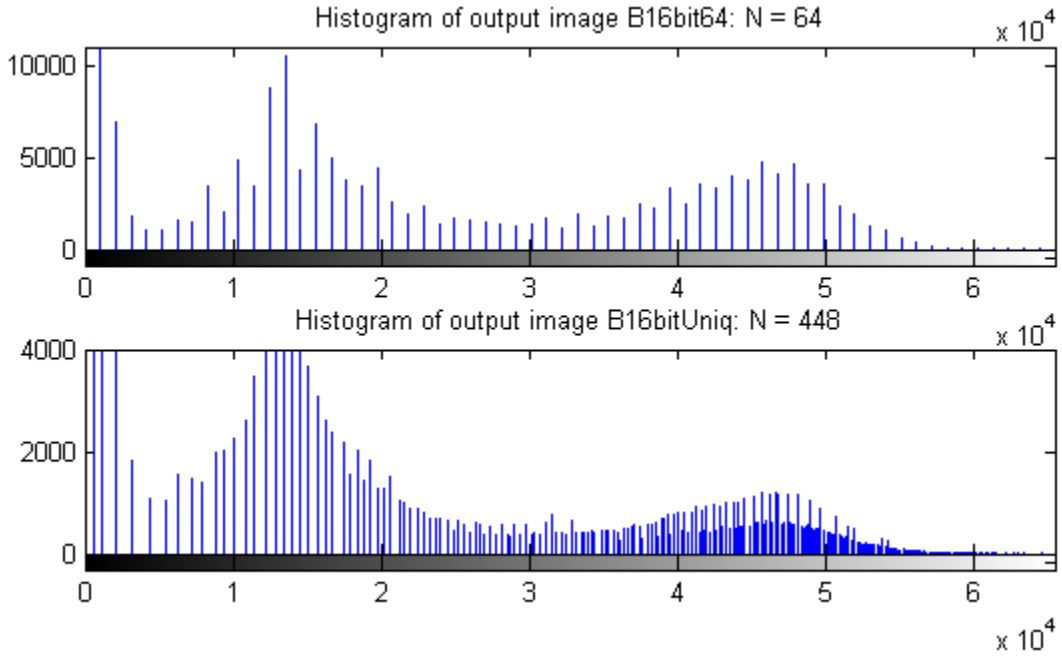
Histogram counts derived from reference image `ref`, specified as a vector or matrix. When `ref` is a truecolor image, `hgram` is a 3-by-`nbins` matrix. When `ref` is a grayscale image, `hgram` is a 1-by-`nbins` vector.

Data Types: `double`

Algorithms

The objective of `imhistmatch` is to transform image `I` such that the histogram of image `J` matches the histogram derived from image `ref`. It consists of `nbins` equally spaced bins which span the full range of the image data type. A consequence of matching histograms in this way is that `nbins` also represents the upper limit of the number of discrete data levels present in image `J`.

An important behavioral aspect of this algorithm to note is that as `nbins` increases in value, the degree of rapid fluctuations between adjacent populated peaks in the histogram of image `J` tends to increase. This can be seen in the following histogram plots taken from the 16–bit grayscale MRI example.



An optimal value for `nbins` represents a trade-off between more output levels (larger values of `nbins`) while minimizing peak fluctuations in the histogram (smaller values of `nbins`).

See Also

`histeq` | `imadjust` | `imhist` | `imhistmatchn`

Topics

“Contrast Enhancement Techniques”

Introduced in R2012b

imhistmatchn

Adjust histogram of N-D image to match histogram of reference image

Syntax

```
B = imhistmatchn(A,ref)
B = imhistmatchn(A,ref,nbins)
[B,hgram] = imhistmatchn( ___ )
```

Description

`B = imhistmatchn(A,ref)` transforms the N-D grayscale image `A` and returns output image `B` whose histogram approximately matches the histogram of the reference image `ref`. Both `A` and `ref` must be grayscale images, but they do not need to have the same data type, size, or number of dimensions.

`B = imhistmatchn(A,ref,nbins)` uses `nbins` equally spaced bins within the appropriate range for the given image data type. The returned image `B` has no more than `nbins` discrete levels.

If the data type of the image is:

- `single` or `double`, the histogram range is [0, 1].
- `uint8`, the histogram range is [0, 255].
- `uint16`, the histogram range is [0, 65535].
- `int16`, the histogram range is [-32768, 32767].

`[B,hgram] = imhistmatchn( ___ )` returns the histogram of the reference image `ref` used for matching in `hgram`. `hgram` is a 1-by-`nbins` vector, where `nbins` is the number of histogram bins.

Examples

Match Histograms of Multidimensional Images

Load an N-D grayscale image into the workspace. Also load a grayscale image to provide a reference histogram.

```
load mri D
load mristack
```

Display the original volume as slices.

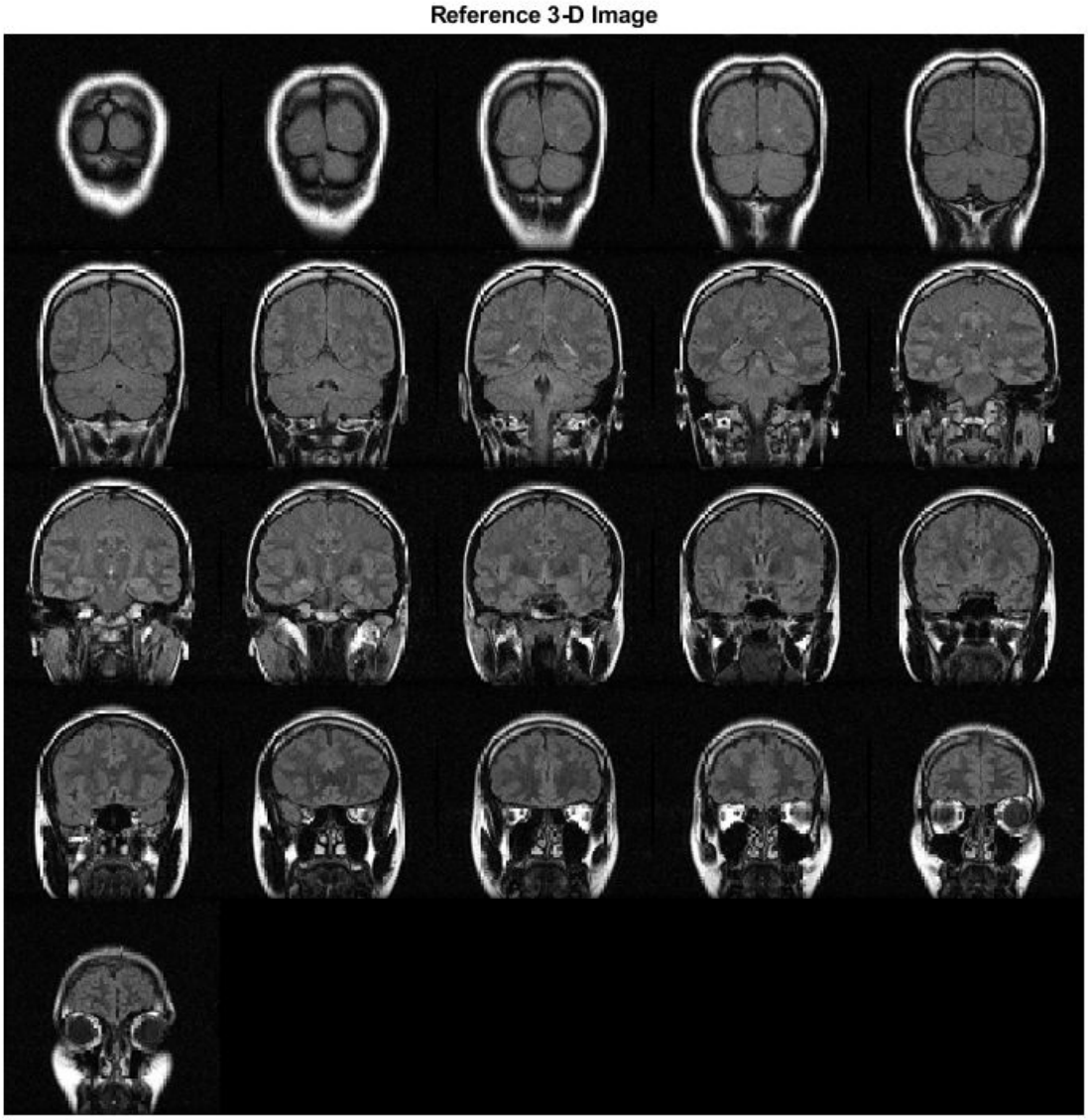
```
figure
montage(D,'DisplayRange',[])
title('Original 3-D Image')
```

Reshape the reference as a stack of grayscale slices for display.

```
ref = reshape(mristack,[256,256,1,21]);
```

Display the reference volume as slices. To display correctly on the screen, the reference volume is downsized by a factor of 0.5 using `imresize`.

```
ref_downsized = imresize(ref,0.5);
figure
montage(ref_downsized,'DisplayRange',[])
title('Reference 3-D Image')
```

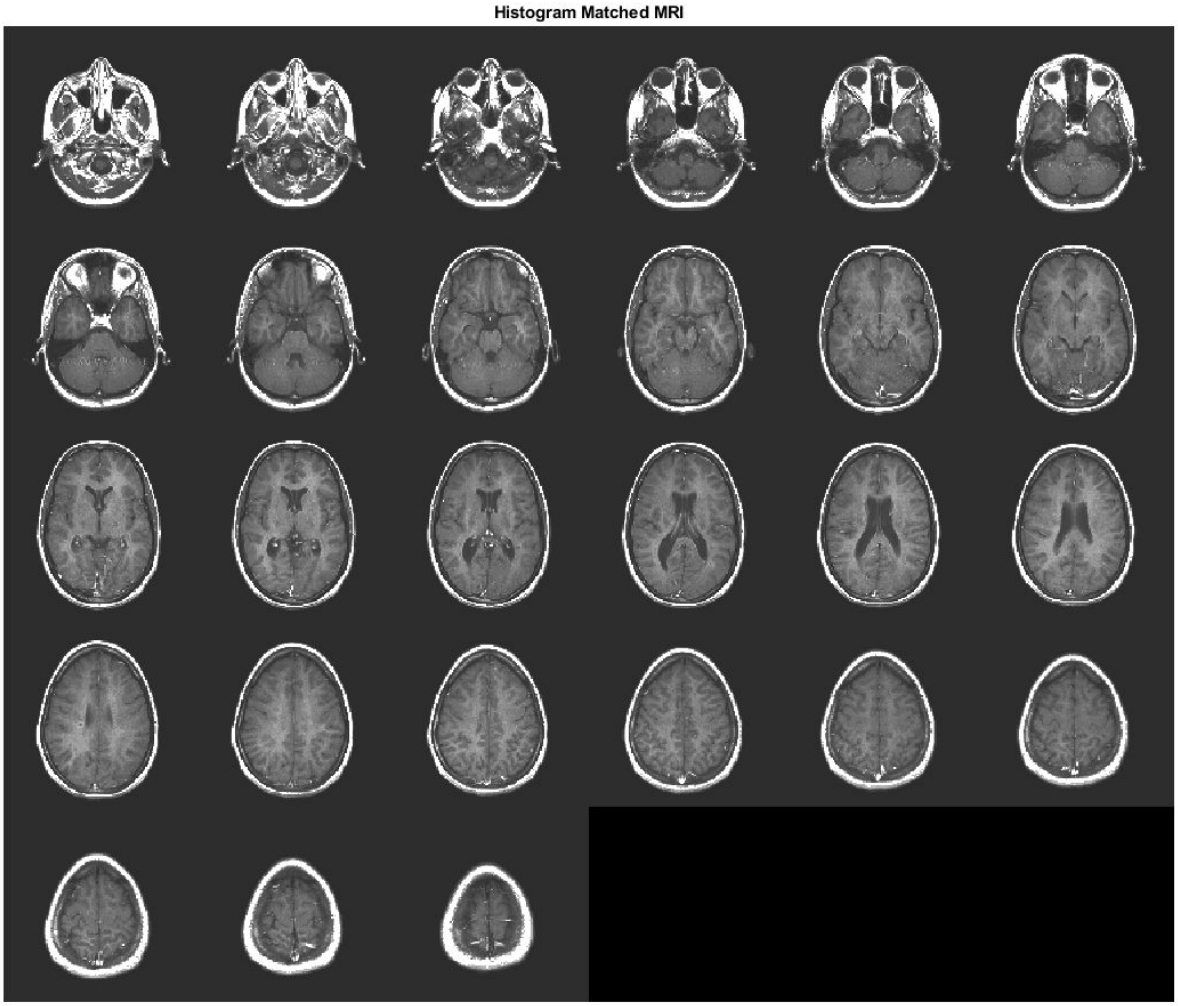


Match the histogram of `D` to the histogram of the fullsize `ref`.

```
Dmatched = imhistmatchn(D,ref);
```

Display the output. Observe that the brightness levels of the output more closely match the reference image than the original image.

```
figure
montage(Dmatched,'DisplayRange',[])
title('Histogram Matched MRI')
```



Input Arguments

`A` — Input image

N-D grayscale image

Input image to be transformed, specified as an N-D grayscale image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`ref` — Reference image whose histogram is the reference histogram

grayscale image

Reference image whose histogram is the reference histogram, specified as a grayscale image. The reference image provides the equally spaced `nbins` bin reference histogram which output image `B` is trying to match.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`nbins` — Number of equally spaced bins in reference histogram

64 (default) | positive integer

Number of equally spaced bins in reference histogram, specified as a positive integer. `nbins` also represents the upper limit of the number of discrete data levels present in output image `B`.

Data Types: `double`

Output Arguments

`B` — Output image

N-D grayscale image

Output image, returned as an N-D grayscale image. The output image is derived from image `A` whose histogram is an approximate match to the histogram of input image `ref` built with `nbins` equally spaced bins. Image `B` is of the same size and data type as input image `A`. Input argument `nbins` represents the upper limit of the number of discrete levels contained in image `B`.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`hgram` — Histogram counts derived from reference image `ref`

1-by-`nbins` vector

Histogram counts derived from reference image `ref`, returned as a 1-by-`nbins` vector.

Data Types: `double`

See Also

`imhistmatch` | `histeq` | `imadjust` | `imhist`

Introduced in R2017a

imhmax

H-maxima transform

Syntax

```
J = imhmax(I,H)
J = imhmax(I,H,conn)
```

Description

`J = imhmax(I,H)` suppresses all maxima in the intensity image `I` whose height is less than `H`. Regional maxima are connected components of pixels with a constant intensity value, and whose external boundary pixels all have a lower value.

`J = imhmax(I,H,conn)` computes the H-maxima transform, where `conn` specifies the connectivity.

Examples

Create H-Maxima Transform

Create simple sample array of zeros with several maxima.

```
a = zeros(10,10);
a(2:4,2:4) = 3;
a(6:8,6:8) = 8
```

```
a = 10×10

     0     0     0     0     0     0     0     0     0     0
     0     3     3     3     0     0     0     0     0     0
     0     3     3     3     0     0     0     0     0     0
     0     3     3     3     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     8     8     8     0     0
     0     0     0     0     0     8     8     8     0     0
     0     0     0     0     0     8     8     8     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

Calculate the maxima equal to 4 or more. Note how the area of the image set to 3 is not included.

```
b = imhmax(a,4)
```

```
b = 10×10

     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

```
     0     0     0     0     0     4     4     4     0     0
     0     0     0     0     0     4     4     4     0     0
     0     0     0     0     0     4     4     4     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

Input Arguments

`I` — Input image

numeric array

Input image, specified as a numeric array of any dimension.

Example: `I = imread('glass.png');`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`H` — H-maxima transform

nonnegative scalar

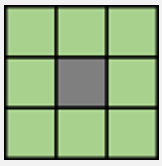
H-maxima transform, specified as a nonnegative scalar.

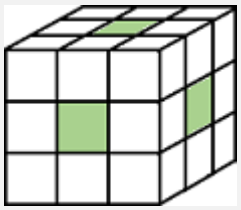
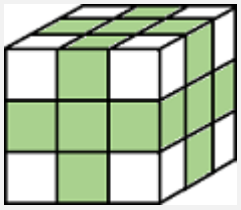
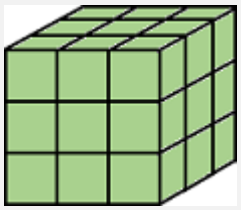
Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`conn` — Pixel connectivity

4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 matrix of `0`s and `1`s

Pixel connectivity, specified as one of the values in this table. The default connectivity is `8` for 2-D images, and `26` for 3-D images.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. The neighborhood of a pixel are the adjacent pixels in the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. The neighborhood of a pixel are the adjacent pixels in the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		

Value	Meaning	
6	Pixels are connected if their faces touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down	 Current pixel is shown in gray.
18	Pixels are connected if their faces or edges touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `imhmax` uses the default value `conndef(ndims(I),'maximal')`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`J` — Transformed image

numeric array

Transformed image, returned as a numeric array of the same size and class as `I`.

References

[1] Soille, P. *Morphological Image Analysis: Principles and Applications*. Springer-Verlag, 1999, pp. 170-171.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imhmax` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imhmax` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, the optional third input argument, `conn`, must be a compile-time constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, the optional third input argument, `conn`, must be a compile-time constant.

See Also

`conndef` | `imextendedmax` | `imhmin` | `imreconstruct` | `imregionalmax`

Introduced before R2006a

imhmin

H-minima transform

Syntax

```
J = imhmin(I,H)
J = imhmin(I,H,conn)
```

Description

`J = imhmin(I,H)` suppresses all minima in the grayscale image `I` whose depth is less than `H`. Regional minima are connected components of pixels with a constant intensity value, *t*, whose external boundary pixels all have a value greater than *t*.

`J = imhmin(I,H,conn)` computes the H-minima transform, where `conn` specifies the connectivity.

Examples

Calculate H-Minima Transform

Create a sample image with two regional minima.

```
a = 10*ones(10,10);
a(2:4,2:4) = 7;
a(6:8,6:8) = 2
```

```
a = 10×10

    10    10    10    10    10    10    10    10    10    10
    10     7     7     7    10    10    10    10    10    10
    10     7     7     7    10    10    10    10    10    10
    10     7     7     7    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10     2     2     2    10    10
    10    10    10    10    10     2     2     2    10    10
    10    10    10    10    10     2     2     2    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
```

Suppress all minima below a specified value. Note how the region with pixels valued 7 disappears in the transformed image because its depth is less than the specified h value.

```
b = imhmin(a,4)
```

```
b = 10×10

    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
```

```
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10     6     6     6    10    10
    10    10    10    10    10     6     6     6    10    10
    10    10    10    10    10     6     6     6    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
```

Input Arguments

`I` — Input image

numeric array

Input image, specified as a numeric array of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`H` — H-minima transform

nonnegative scalar

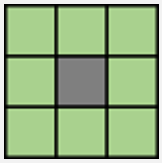
H-minima transform, specified as a nonnegative scalar.

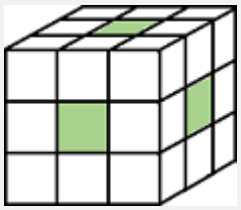
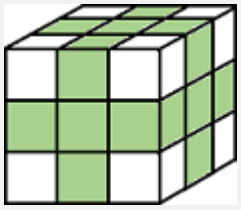
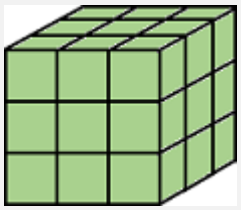
Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`conn` — Pixel connectivity

4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 matrix of `0`s and `1`s

Pixel connectivity, specified as one of the values in this table. The default connectivity is `8` for 2-D images, and `26` for 3-D images.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. The neighborhood of a pixel are the adjacent pixels in the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. The neighborhood of a pixel are the adjacent pixels in the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		

Value	Meaning	
6	Pixels are connected if their faces touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down	 Current pixel is shown in gray.
18	Pixels are connected if their faces or edges touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `imhmin` uses the default value `conndef(ndims(I),'maximal')`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`J` — Transformed image

numeric array

Transformed image, returned as a numeric array of the same size and data type as `I`.

References

[1] Soille, P. *Morphological Image Analysis: Principles and Applications*. Springer-Verlag, 1999, pp. 170-171.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imhmin` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imhmin` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, the optional third input argument, `conn`, must be a compile-time constant.

See Also

`conndef` | `imextendedmin` | `imhmax` | `imreconstruct` | `imregionalmin`

Introduced before R2006a

imimposemin

Impose minima

Syntax

```
J = imimposemin(I,BW)
J = imimposemin(I,BW,conn)
```

Description

`J = imimposemin(I,BW)` modifies the grayscale mask image `I` using morphological reconstruction so it only has regional minima wherever binary marker image `BW` is nonzero.

`J = imimposemin(I,BW,conn)` specifies the pixel connectivity for the morphological reconstruction.

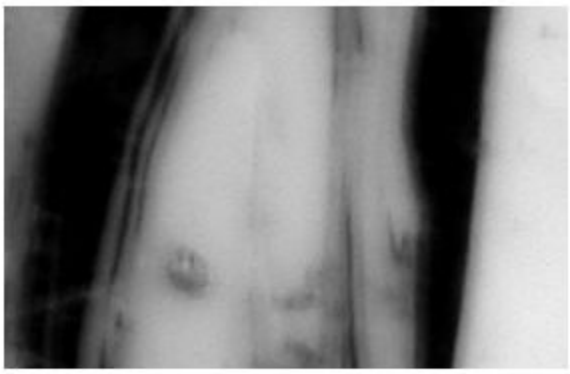
Examples

Impose Regional Minimum at One Location

This example shows how to modify an image so that one area is always a regional minimum.

Read an image and display it. This image is called the *mask* image.

```
mask = imread('glass.png');
imshow(mask)
```



Create a binary image that is the same size as the mask image and sets a small area of the binary image to 1. These pixels define the location in the mask image where a regional minimum will be imposed. The resulting image is called the *marker* image.

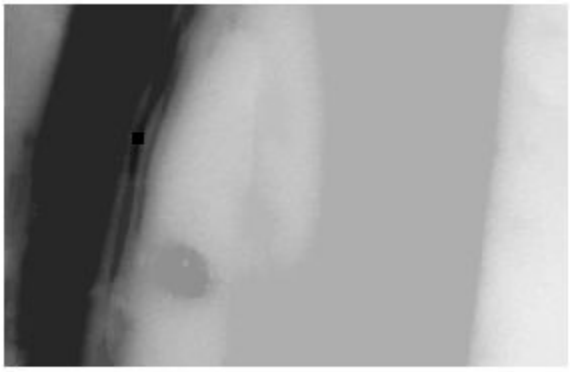
```
marker = false(size(mask));
marker(65:70,65:70) = true;
```

Superimpose the marker over the mask to show where these pixels of interest fall on the original image. The small white square marks the spot. This code is not essential to the impose minima operation.

```
J = mask;
J(marker) = 255;
figure
imshow(J)
title('Marker Image Superimposed on Mask')
```

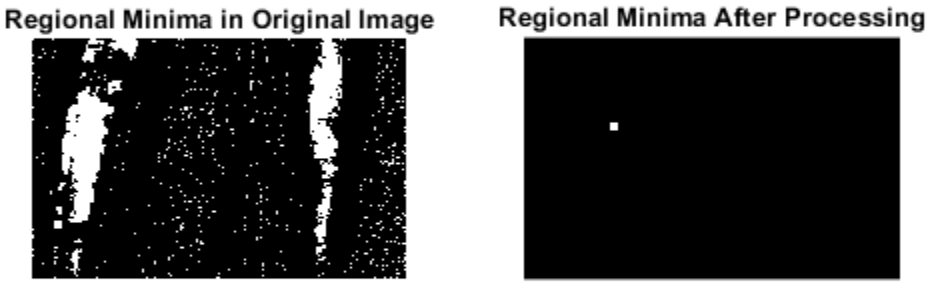
Impose the regional minimum on the input image using the `imimposemin` function. Note how all the dark areas of the original image, except the marked area, are lighter.

```
K = imimposemin(mask,marker);
figure
imshow(K)
```



To illustrate how this operation removes all minima in the original image except the imposed minimum, compare the regional minima in the original image with the regional minimum in the processed image. These calls to `imregionalmin` return binary images that specify the locations of all the regional minima in both images.

```
BW = imregionalmin(mask);
figure
subplot(1,2,1)
imshow(BW)
title('Regional Minima in Original Image')

BW2 = imregionalmin(K);
subplot(1,2,2)
imshow(BW2)
title('Regional Minima After Processing')
```



Input Arguments

`I` — Grayscale mask image

numeric array

Grayscale mask image, specified as a numeric array of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`BW` — Binary marker image

numeric array | logical array

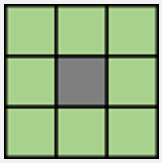
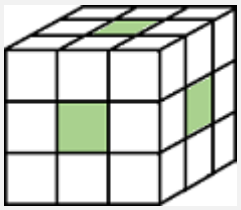
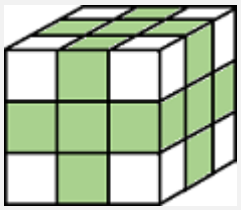
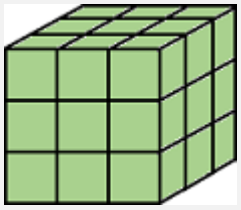
Binary marker image, specified as a numeric or logical array of the same size as the grayscale mask image `I`. For numeric input, any nonzero pixels are considered to be `1` (`true`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`conn` — Pixel connectivity

4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 matrix of `0`s and `1`s

Pixel connectivity, specified as one of the values in this table. The default connectivity is 8 for 2-D images, and 26 for 3-D images.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. The neighborhood of a pixel are the adjacent pixels in the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. The neighborhood of a pixel are the adjacent pixels in the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		
6	Pixels are connected if their faces touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down	 Current pixel is shown in gray.
18	Pixels are connected if their faces or edges touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `imimposemin` uses the default value `conndef(ndims(I),'maximal')`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Data Types: `double` | `logical`

Output Arguments

`J` — Reconstructed image

numeric array

Reconstructed image, returned as a numeric or logical array of the same size and data type as `I`.

Algorithms

`imimposemin` uses a technique based on morphological reconstruction.

See Also

`conndef` | `imreconstruct` | `imregionalmin`

Topics

“Morphological Reconstruction”

Introduced before R2006a

imlincomb

Linear combination of images

Syntax

```
Z = imlincomb(K1,A1,K2,A2,...,Kn,An)
Z = imlincomb(K1,A1,K2,A2,...,Kn,An,K)
Z = imlincomb(___,outputClass)
```

Description

`Z = imlincomb(K1,A1,K2,A2,...,Kn,An)` computes the linear combination of images, `A1`, `A2`, ... , `An`, with weights `K1`, `K2`, ... , `Kn` according to:

```
    Z = K1*A1 + K2*A2 + ... + Kn*An
```

`Z = imlincomb(K1,A1,K2,A2,...,Kn,An,K)` adds an offset, `K`, to the linear combination:

```
    Z = K1*A1 + K2*A2 + ... + Kn*An + K
```

`Z = imlincomb(___,outputClass)` specifies the output class of `Z`.

Examples

Scale an Image Using Linear Combinations

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Scale the image using a coefficient of 1.5 in the linear combination.

```
J = imlincomb(1.5,I);
```

Display the original image and the processed image.

```
imshow(I)
```

```
figure
imshow(J)
```

Form a Difference Image with Zero Value Shifted to 128

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Create a low-pass filtered copy of the image.

```
J = uint8(filter2(fspecial('gaussian'), I));
```

Find the difference image and shift the zero value to 128 using a linear combination of `I` and `J`.

```
K = imlincomb(1,I,-1,J,128); %K(r,c) = I(r,c) - J(r,c) + 128
```

Display the resulting difference image.

```
imshow(K)
```

Add Two Images and Specify Output Class Using Linear Combinations

Read two grayscale `uint8` images into the workspace.

```
I = imread('rice.png');
J = imread('cameraman.tif');
```

Add the images using a linear combination. Specify the output as type `uint16` to avoid truncating the result.

```
K = imlincomb(1,I,1,J,'uint16');
```

Display the result.

```
imshow(K,[])
```

Compare Methods for Averaging Images

This example shows the difference between nesting calls and using linear combinations when performing a series of arithmetic operations on images. To illustrate how `imlincomb` performs all the arithmetic operations before truncating the result, compare the results of calculating the average of two arrays, `X` and `Y`, using nested arithmetic functions and using `imlincomb`.

Create two arrays.

```
X = uint8([ 255 0 75; 44 225 100]);
Y = uint8([ 50 50 50; 50 50 50 ]);
```

Average the arrays using nested arithmetic functions. To calculate the average returned in `Z(1,1)`, the function `imadd` adds 255 and 50 and truncates the result to 255 before passing it to `imdivide`. The average returned in `Z(1,1)` is 128.

```
Z = imdivide(imadd(X,Y),2)
```

```
Z = 2x3 uint8 matrix

   128    25    63
    47   128    75
```

In contrast, `imlincomb` performs the addition and division in double precision and only truncates the final result. The average returned in `Z2(1,1)` is 153.

```
Z2 = imlincomb(.5,X,.5,Y)
```

```
Z2 = 2x3 uint8 matrix

   153    25    63
    47   138    75
```

Input Arguments

K1, K2, Kn — Image coefficients

numeric scalar

Image coefficients, specified as numeric scalars.

Data Types: `double`

A1, A2, An — Input images

numeric array

Input images, specified as numeric arrays of the same size and class.

K — Offset

numeric scalar

Offset, specified as a numeric scalar.

Data Types: `double`

outputClass — Output class

string scalar | character vector

Output class of Z, specified as a string scalar or character vector containing the name of a numeric class.

Example: `'uint16'`

Example: `"double"`

Output Arguments

Z — Linearly combined image

numeric array

Linearly combined image, returned as a numeric array of the same size as `A1`. If `A1` is `logical`, then `Z` is `double`, otherwise `Z` has the same class as `A1`.

Tips

- When performing a series of arithmetic operations on a pair of images, you can achieve more accurate results if you use `imlincomb` to combine the operations, rather than nesting calls to the individual arithmetic functions, such as `imadd`. When you nest calls to the arithmetic functions, and the input arrays are of an integer class, each function truncates and rounds the result before passing it to the next function, thus losing accuracy in the final result. `imlincomb` computes each element of the output `Z` individually, in double-precision floating point. If `Z` is an integer array, `imlincomb` clips elements of `Z` that exceed the range of the integer type and rounds off fractional values.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imlincomb` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imlincomb` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- You can specify up to 4 input image arguments.
- The `output_class` argument must be a compile-time constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`imadd` | `imcomplement` | `imdivide` | `immultiply` | `imsubtract`

Introduced before R2006a

imlocalbrighten

Brighten low-light image

Syntax

```
B = imlocalbrighten(A)
B = imlocalbrighten(A,amount)
B = imlocalbrighten( ___ ,'AlphaBlend',alphaBlend)
[B,D] = imlocalbrighten( ___ )
```

Description

`B = imlocalbrighten(A)` brightens low-light areas in RGB or grayscale image `A`.

`B = imlocalbrighten(A,amount)` brightens low-light areas in `A` by a specified amount.

`B = imlocalbrighten( ___ ,'AlphaBlend',alphaBlend)` also specifies whether to preserve bright areas of the input image by performing alpha blending.

`[B,D] = imlocalbrighten( ___ )` also returns the darkness estimate `D` of each pixel in the input image.

Examples

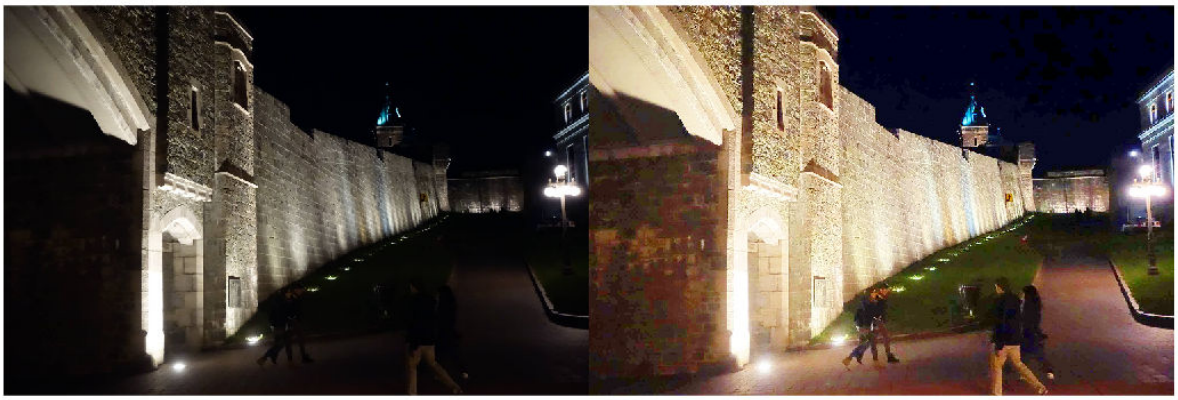
Brighten Low-Light Images

Read a low-light image into the workspace.

```
A = imread('lowlight_2.jpg');
```

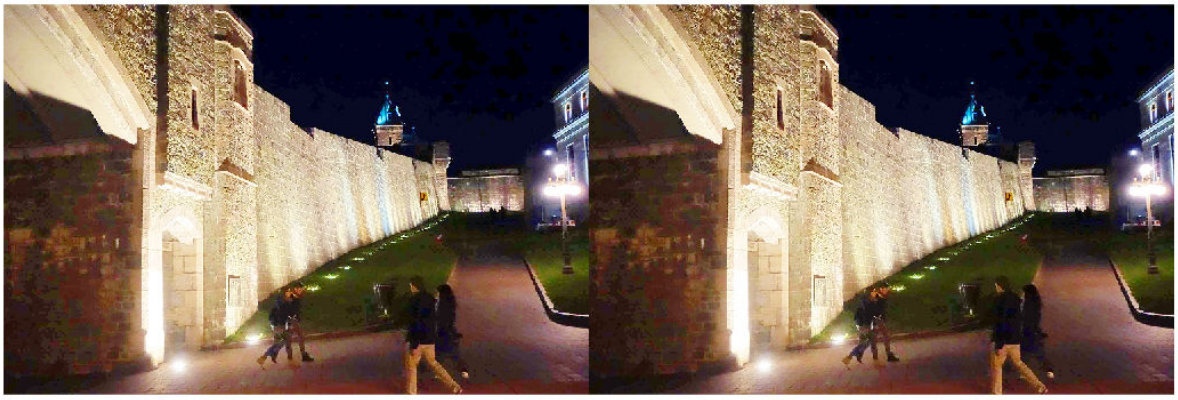
Brighten the low-light image using default parameters. Display the original and brightened image side-by-side in a montage.

```
B = imlocalbrighten(A);
montage({A,B})
```



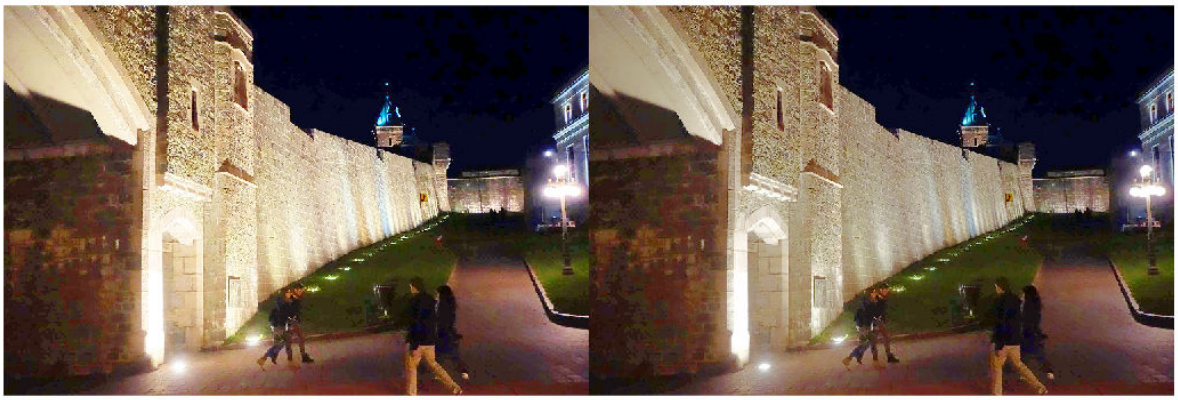
Brighten the low-light image again, this time specifying the amount of lightening to apply to the image. Display the two brightened images side-by-side in a montage.

```
B2 = imlocalbrighten(A,0.8);
montage({B,B2})
```



Use the `AlphaBlend` option to preserve content from the original image in the lightened image. View the lightened output image from the first example with the alpha blended output image. Compare the detail shown in the wall above arched entryway near the center of the image in the alpha-blended version with the original lightened image.

```
Bblend = imlocalbrighten(A,'AlphaBlend',true);
montage({B,Bblend})
```



Get the estimated darkness-per-pixel matrix return value. View the original image and the darkness estimate matrix.

```
[~,D] = imlocalbrighten(A);
montage({A,D})
```

Input Arguments

`A` — Image to be brightened

RGB image | grayscale image

Image to be brightened, specified as an RGB image or grayscale image.

Data Types: `single` | `double` | `uint8` | `uint16`

`amount` — Amount to brighten image

1 (default) | number in the range [0, 1]

Amount to brighten the image, specified as a number in the range [0, 1]. When the value is `1` (the default), `imlocalbrighten` brightens the low-light areas of `A` as much as possible. When the value is `0`, `imlocalbrighten` returns the input image unmodified.

Example: `0.2`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`alphaBlend` — Alpha blend input and enhanced image
`false` (default) | `true`

Alpha blend input and enhanced image, specified as `false` or `true`. Alpha blending combines the input image with the enhanced image to preserve brighter areas of the input image. When `true`, `imlocalbrighten` uses the estimate of darkness matrix, `D`, to preserve content of the input image proportional to the amount of light in each pixel.

Output Arguments

`B` — Brightened image
numeric array

Brightened image, returned as a numeric array of the same size and data type as the input image `A`.

`D` — Darkness estimate
numeric matrix

Darkness estimate of each pixel in the input image, returned as a numeric matrix. `D` is the same size as the first two dimensions of the input image.

Data Types: `double`

References

[1] Dong, X., G. Wang, Y. Pang, W. Li, J. Wen, W. Meng, and Y. Lu. "*Fast efficient algorithm for enhancement of low lighting video.*" Proceedings of IEEE® International Conference on Multimedia and Expo (ICME). 2011, pp. 1–6.

[2] He, Kaiming. "*Single Image Haze Removal Using Dark Channel Prior.*" Thesis, The Chinese University of Hong Kong, 2011.

[3] Dubok Park; Hyungjo Park; David K. Han; Hanseok Ko "*Single Image Dehazing with Image Entropy and Information Fidelity.*" ICIP, 2014.

See Also
`adapthisteq` | `histeq` | `imreducehaze`

Introduced in R2019b

immagbox

Magnification box for image displayed in scroll panel

Syntax

```
hbox = immagbox(hparent,himage)
```

Description

Use the `immagbox` function to add a magnification box to the same figure as an image contained in a scroll panel. A magnification box is an editable text box that contains the current magnification of the target image. When you enter a new value in the magnification box, the magnification of the target image changes. When the magnification of the target image changes for any reason, the magnification box updates the magnification value.

`hbox = immagbox(hparent,himage)` creates a magnification box for an image displayed in a scroll panel. `himage` is a handle to the target image in the scroll panel. `hparent` is a handle to the figure or uipanel object that will contain the magnification box. The function returns `hbox`, a handle to the magnification box.

Examples

Add Magnification Box to Scrollable Image

Display an image in a figure. The example suppresses the standard toolbar and menubar in the figure window because these do not work with the scroll panel.

```
hFig = figure('Toolbar','none','Menubar','none');
hIm = imshow('pears.png');
```

Create a scroll panel to contain the image.

```
hSP = imscrollpanel(hFig,hIm);
set(hSP,'Units','normalized','Position',[0 .1 1 .9])
```

Add a magnification box to the figure. Set the position of the magnification box to the lower left corner of the figure.

```
hMagBox = immagbox(hFig,hIm);
pos = get(hMagBox,'Position');
set(hMagBox,'Position',[0 0 pos(3) pos(4)])
```

```
100%
```

Get the scroll panel API so that you can control the view programmatically.

```
apiSP = iptgetapi(hSP);
```

Set the magnification of the image to 200% by using the scroll panel API function `setMagnification`. Notice how the magnification box updates.

```
apiSP.setMagnification(2)
```

Input Arguments

`hparent` — Handle to figure or uipanel object

handle

Handle to a figure or uipanel object that contains the magnification box, specified as a handle.

`himage` — Handle to target image

handle

Handle to target image, specified as a handle. The image must be displayed in a scroll panel created by `imscrollpanel`.

Output Arguments

`hbox` — Handle to magnification box

handle

Handle to magnification box, returned as a handle. A magnification box is a type of uipanel object.

More About

Magnification Box API Structure

A magnification box contains a structure of function handles, called an API. You can use the functions in this API to manipulate the magnification box. To retrieve this structure, use the `iptgetapi` function, as in the following example.

```
api = iptgetapi(hbox)
```

This table lists the magnification box API functions, in the order they appear in the structure.

Function	Description
`setMagnification`	Set the magnification of the target image in units of screen pixels per image pixel. `mag = api.setMagnification(new_mag)` `new_mag` is a scalar magnification factor.

See Also

`imscrollpanel` | `iptgetapi`

Introduced before R2006a

immovie

Make movie from multiframe image

Syntax

```
mov = immovie(X,cmap)
mov = immovie(RGB)
```

Description

`mov = immovie(X,cmap)` returns the movie structure array `mov` from the images in the multiframe indexed image `X` with colormap `cmap`.

`mov = immovie(RGB)` returns the movie structure array `mov` from the images in the multiframe truecolor image `RGB`.

Examples

Make Movie from Indexed Image Sequence

```
load mri
mov = immovie(D,map);
implay(mov)
```

Input Arguments

X — Multiframe indexed image

m-by-*n*-by-1-by-*k* numeric array

Multiframe indexed image, specified as an *m*-by-*n*-by-1-by-*k* numeric array, where *k* is the number of frames. Each frame uses the same colormap, `cmap`.

Data Types: `single` | `double` | `uint8` | `uint16` | `logical`

cmap — Colormap

c-by-3 numeric matrix

Colormap associated with multiframe indexed image `X`, specified as a *c*-by-3 numeric matrix containing the RGB values of *c* colors.

RGB — Multiframe truecolor image

m-by-*n*-by-3-by-*k* numeric array

Multiframe truecolor image, specified as an *m*-by-*n*-by-3-by-*k* numeric array, where *k* is the number of frames.

Data Types: `single` | `double` | `uint8` | `uint16`

Output Arguments

`mov` — Movie

k-by-1 array of movie frame structures

Movie, returned as an *k*-by-1 array of movie frame structures. For details about the movie frame structure, see `getframe`.

Tips

- To play the movie, use the **Video Viewer** app.
- To create a movie that can be played outside the MATLAB environment, use a `VideoWriter` object.

See Also

Video Viewer | `VideoWriter` | `getframe` | `movie` | `montage`

Introduced before R2006a

immse

Mean-squared error

Syntax

```
err = immse(X,Y)
```

Description

`err = immse(X,Y)` calculates the mean-squared error (MSE) between the arrays `X` and `Y`. A lower MSE value indicates greater similarity between `X` and `Y`.

Examples

Calculate Mean-Squared Error in Noisy Image

Read image and display it.

```
ref = imread('pout.tif');
imshow(ref)
```

Create another image by adding noise to a copy of the reference image.

```
A = imnoise(ref,'salt & pepper', 0.02);
imshow(A)
```

Calculate mean-squared error between the two images.

```
err = immse(A, ref);
fprintf('\n The mean-squared error is %0.4f\n', err);
```

```
 The mean-squared error is 353.7631
```

Input Arguments

X — Input array

numeric array

Input array, specified as a numeric array of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Y — Input array

numeric array

Input array, specified as a numeric array of the same size and data type as X.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output Arguments

`err` — Mean-squared error

positive number

Mean-squared error, returned as a positive number. The data type of `err` is `double` unless the input arguments are of data type `single`, in which case `err` is of data type `single`

Data Types: `single` | `double`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`immse` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`mean` | `median` | `psnr` | `ssim` | `sum` | `var`

Introduced in R2014b

immultiply

Multiply two images or multiply image by constant

Syntax

```
Z = immultiply(X,Y)
```

Description

`Z = immultiply(X,Y)` multiplies each element in array `X` by the corresponding element in array `Y` and returns the product in the corresponding element of the output array `Z`.

Examples

Multiply an Image by Itself

Read a grayscale image into the workspace, then convert the image to `uint8`.

```
I = imread('moon.tif');
I16 = uint16(I);
```

Multiply the image by itself. Note that `immultiply` converts the class of the image from `uint8` to `uint16` before performing the multiplication to avoid truncating the results.

```
J = immultiply(I16,I16);
```

Show the original image and the processed image.

```
imshow(I)
```

```
figure
imshow(J)
```

Scale an Image by a Constant Factor

Read an image into the workspace.

```
I = imread('moon.tif');
```

Scale each value of the image by a constant factor of 0.5.

```
J = immultiply(I,0.5);
```

Display the original image and the processed image.

```
imshow(I)
```

```
figure
imshow(J)
```

Input Arguments

X — First array
numeric array | logical array

First array, specified as a numeric array or logical array of any dimension.

Y — Second array
numeric scalar | numeric array | logical array

Second array to be multiplied with X, specified as a numeric scalar, numeric array, or logical array.

- If X is numeric, then the size and class of Y can have one of the following values:
  - Y is the same size and class as X.
  - Y is the same size as X and is logical.
  - Y is a scalar of type `double`.
- If X is logical, then Y must have the same size as X. Y can be any class.

Output Arguments

Z — Product

numeric array

Product, returned as a numeric array.

- If X is numeric, then Z has the same size and class as X.
- If X is logical, then Z has the same size and class as Y.

`immultiply` computes each element of Z individually in double-precision floating point. If X or Y is an integer array, then elements of Z exceeding the range of the integer type are truncated, and fractional values are rounded.

Tips

- If X and Y are numeric arrays of the same size and class, then you can use the expression `X.*Y` instead of `immultiply`.

See Also

`imabsdiff` | `imadd` | `imcomplement` | `imdivide` | `imlincomb` | `imsubtract`

Introduced before R2006a

imnlmfilt

Non-local means filtering of image

Syntax

```
J = imnlmfilt(I)
J = imnlmfilt(I,Name,Value)
[J,estDoS] = imnlmfilt( ___ )
```

Description

`J = imnlmfilt(I)` applies a non-local means-based filter to the grayscale or color image `I` and returns the resulting image in `J`.

`J = imnlmfilt(I,Name,Value)` uses name-value pairs to change the behavior of the non-local means filter.

`[J,estDoS] = imnlmfilt( ___ )` also returns the degree of smoothing, `estDoS`, used to estimate the denoised pixel value.

Examples

Denoise Grayscale Image Using Non-Local Means Filter

Read a grayscale image.

```
I = imread('cameraman.tif');
```

Add zero-mean white Gaussian noise with 0.0015 variance to the image using the `imnoise` function.

```
noisyImage = imnoise(I,'gaussian',0,0.0015);
```

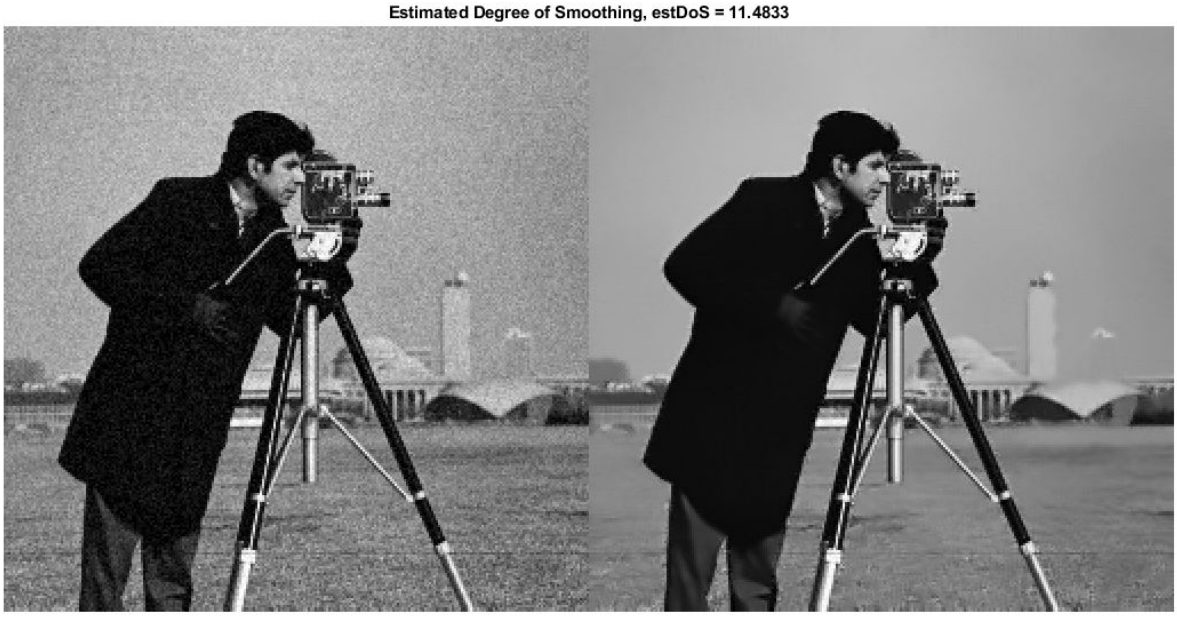
Remove noise from the image through non-local means filtering. The `imnlmfilt` function estimates the degree of smoothing based on the standard deviation of noise in the image.

```
[filteredImage,estDoS] = imnlmfilt(noisyImage);
```

Display the noisy image (left) and the non-local means filtered image (right) as a montage. Display the estimated degree of smoothing, `estDoS`, in the figure title.

The non-local means filter removes noise from the input image but preserves the sharpness of strong edges, such as the silhouette of the man and buildings. This function also smooths textured regions, such as the grass in the foreground of the image, resulting in less detail when compared to the noisy image.

```
montage({noisyImage,filteredImage})
title(['Estimated Degree of Smoothing, ', 'estDoS = ',num2str(estDoS)])
```



Denoise Color Image Using Non-Local Means Filter

Read a color image.

```
imRGB = imread('peppers.png');
```

Add white Gaussian noise with zero mean and 0.0015 variance to the image using the `imnoise` function. Display the noisy RGB image.

```
noisyRGB = imnoise(imRGB,'gaussian',0,0.0015);
imshow(noisyRGB)
```

Convert the noisy RGB image to the L*a*b color space, so that the non-local means filter smooths perceptually similar colors.

```
noisyLAB = rgb2lab(noisyRGB);
```

Extract a homogeneous L*a*b patch from the noisy background to compute the noise standard deviation.

```
roi = [210,24,52,41];
patch = imcrop(noisyLAB,roi);
```

In this L*a*b patch, compute the Euclidean distance from the origin, `edist`. Then, calculate the standard deviation of `edist` to estimate the noise.

```
patchSq = patch.^2;
edist = sqrt(sum(patchSq,3));
patchSigma = sqrt(var(edist(:)));
```

Set the `'DegreeOfSmoothing'` value to be higher than the standard deviation of the patch. Filter the noisy L*a*b* image using non-local means filtering.

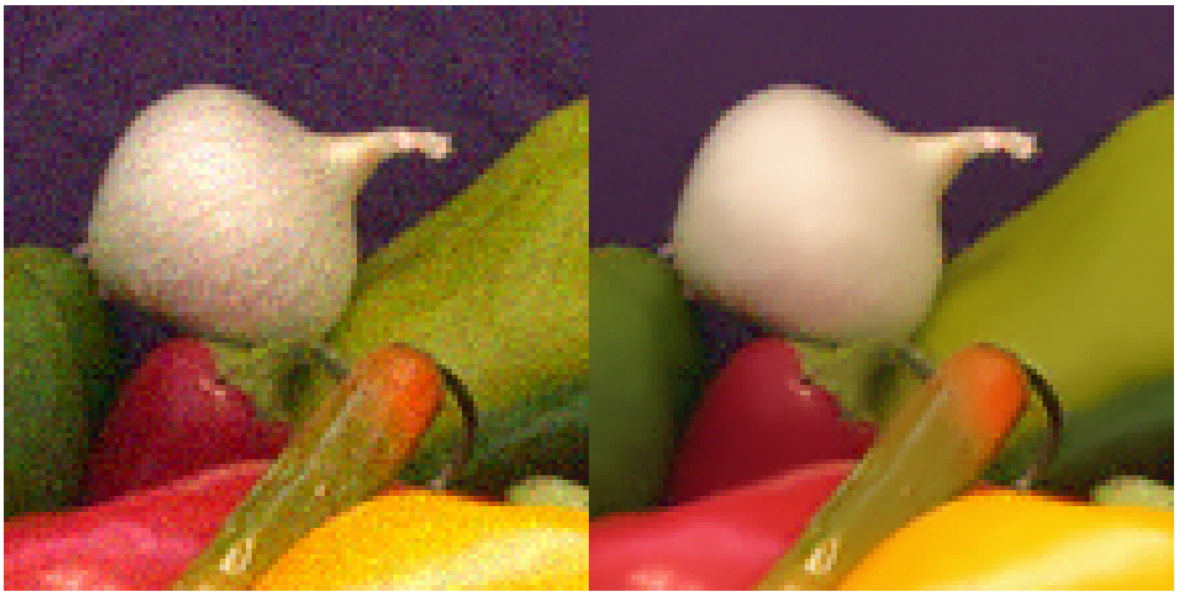
```
DoS = 1.5*patchSigma;
denoisedLAB = imnlmfilt(noisyLAB,'DegreeOfSmoothing',DoS);
```

Convert the filtered L*a*b image to the RGB color space. Display the filtered RGB image.

```
denoisedRGB = lab2rgb(denoisedLAB,'Out','uint8');
imshow(denoisedRGB)
```

Compare a patch from the noisy RGB image (left) and the same patch from the non-local means filtered RGB image (right).

```
roi2 = [178,68,110,110];
montage({imcrop(noisyRGB,roi2),imcrop(denoisedRGB,roi2)})
```



Input Arguments

`I` — Image to filter

2-D grayscale image | 2-D color image

Image to filter, specified as a 2-D grayscale image of size *m*-by-*n* or a 2-D color image of size *m*-by-*n*-by-3. The size of `I` must be greater than or equal to 21-by-21.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `J = imnlmfilt(I,'DegreeOfSmoothing',10);`

`DegreeOfSmoothing` — Degree of smoothing

positive number

Degree of smoothing, specified as the comma-separated pair consisting of `'DegreeOfSmoothing'` and a positive number. As this value increases, the smoothing in the resulting image `J` increases. If you do not specify `'DegreeOfSmoothing'`, then the default value is the standard deviation of noise estimated from the image. For more information, see “Default Degree of Smoothing” on page 1-1682.

`SearchWindowSize` — Search window size

21 (default) | odd-valued positive integer

Search window size, specified as the comma-separated pair consisting of `'SearchWindowSize'` and an odd-valued positive integer, *s*. The search for similar neighborhoods to a pixel is limited to the *s*-by-*s* region surrounding that pixel. `SearchWindowSize` affects the performance linearly in terms of time. `SearchWindowSize` cannot be larger than the size of the input image, `I`.

`ComparisonWindowSize` — Comparison window size
5 (default) | odd-valued positive integer

Comparison window size, specified as the comma-separated pair consisting of `'ComparisonWindowSize'` and an odd-valued positive integer, *c*. The `imnlmfilt` function computes similarity weights using the *c*-by-*c* neighborhood surrounding pixels. `ComparisonWindowSize` must be less than or equal to `SearchWindowSize`. For more information, see “Estimate Denoised Pixel Value” on page 1-1682.

Output Arguments

`J` — Non-local means filtered image
2-D grayscale image | 2-D color image

Non-local means filtered image, returned as a 2-D grayscale image or 2-D color image of the same size and data type as the input image, `I`.

`estDoS` — Estimated degree of smoothing
positive number

Estimated degree of smoothing, returned as a positive number. If you specify `DegreeOfSmoothing`, then `imnlmfilt` returns the same value in `estDoS`. Otherwise, `imnlmfilt` returns the default degree of smoothing estimated using “Default Degree of Smoothing” on page 1-1682.

Tips

- To smooth perceptually close colors in an RGB image, convert the image to the CIE L*a*b* color space using `rgb2lab` before applying the non-local means filter. To view the results, first convert the filtered L*a*b* image to the RGB color space using `lab2rgb`.
- If the data type of `I` is `double`, then computations are performed in data type `double`. Otherwise, computations are performed in data type `single`.

Algorithms

Default Degree of Smoothing

The default value of `'DegreeOfSmoothing'` is the standard deviation of noise estimated from the image. To estimate the standard deviation, `imnlmfilt` convolves the image with a 3-by-3 filter proposed by J. Immerkær [2]. When `I` is a color image, the default value of `'DegreeOfSmoothing'` is the standard deviations of noise averaged across the channels.

Estimate Denoised Pixel Value

The non-local means filtering algorithm estimates the denoised value of pixel *p* using these steps.

1. For a specific pixel, *q*, in the search window, calculate the weighted Euclidean distance between pixel values in the *c*-by-*c* comparison windows surrounding *p* and *q*. For color images, include all channels in the Euclidean distance calculation.

The weight is a decreasing exponential function whose rate of decay is determined by the square of `'DegreeOfSmoothing'`. When an image is noisy, `'DegreeOfSmoothing'` is large and all pixels contribute to the Euclidean distance calculation. When an image has little noise, `'DegreeOfSmoothing'` is small and only pixels with similar values contribute to the Euclidean distance calculation.

The result is a numeric scalar that indicates the similarity between the neighborhood of *p* and the neighborhood of *q*.

Note In the implementation by A. Buades et al. [1], the Euclidean distance between two comparison windows is convolved with a Gaussian kernel of size *c*-by-*c*. This convolution gives more weight to the Euclidean distance between pixel values for pixels near the center of the comparison window. The `imnlmfilt` function omits this step for computational efficiency.

---

2. Repeat this computation for each of the other pixels in the *s*-by-*s* search window, finding the weighted Euclidean distance between pixel *p* and each of those pixels. The result is an *s*-by-*s* similarity matrix that indicates similarity between the neighborhood of *p* and the other neighborhoods in the search window.
3. Normalize the similarity matrix.
4. Using the weights in the normalized similarity matrix, compute the weighted average of pixel values in the *s*-by-*s* search window around pixel *p*. The result is the denoised value of *p*.

References

[1] Buades, A., B. Coll, and J.-M. Morel. "A Non-Local Algorithm for Image Denoising." *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2, June 2005, pp. 60–65.

[2] Immerkær, J. "Fast Noise Variance Estimation." *Computer Vision and Image Understanding*. Vol. 64, Number 2, Sept. 1996, pp. 300–302.

See Also

`imbilatfilt` | `imdiffusefilt` | `imguidedfilter` | `locallapfilt`

Introduced in R2018b

imnoise

Add noise to image

Syntax

```
J = imnoise(I,'gaussian')
J = imnoise(I,'gaussian',m)
J = imnoise(I,'gaussian',m,var_gauss)
J = imnoise(I,'localvar',var_local)
J = imnoise(I,'localvar',intensity_map,var_local)
J = imnoise(I,'poisson')
J = imnoise(I,'salt & pepper')
J = imnoise(I,'salt & pepper',d)
J = imnoise(I,'speckle')
J = imnoise(I,'speckle',var_speckle)
```

Description

`J = imnoise(I,'gaussian')` adds zero-mean, Gaussian white noise with variance of 0.01 to grayscale image `I`.

`J = imnoise(I,'gaussian',m)` adds Gaussian white noise with mean `m` and variance of 0.01.

`J = imnoise(I,'gaussian',m,var_gauss)` adds Gaussian white noise with mean `m` and variance `var_gauss`.

`J = imnoise(I,'localvar',var_local)` adds zero-mean, Gaussian white noise of local variance `var_local`.

`J = imnoise(I,'localvar',intensity_map,var_local)` adds zero-mean, Gaussian white noise. The local variance of the noise, `var_local`, is a function of the image intensity values in `I`. The mapping of image intensity value to noise variance is specified by the vector `intensity_map`.

`J = imnoise(I,'poisson')` generates Poisson noise from the data instead of adding artificial noise to the data. See “Algorithms” on page 1-1687 for more information.

`J = imnoise(I,'salt & pepper')` adds salt and pepper noise, with default noise density 0.05. This affects approximately 5% of pixels.

`J = imnoise(I,'salt & pepper',d)` adds salt and pepper noise, where `d` is the noise density. This affects approximately `d*numel(I)` pixels.

`J = imnoise(I,'speckle')` adds multiplicative noise using the equation `J = I+n*I`, where `n` is uniformly distributed random noise with mean 0 and variance 0.05.

`J = imnoise(I,'speckle',var_speckle)` adds multiplicative noise with variance `var_speckle`.

Examples

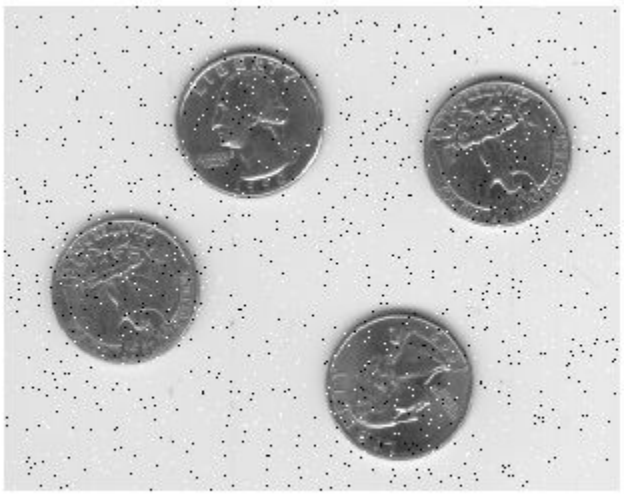
Add Noise to an Image

Read a grayscale image and display it.

```
I = imread('eight.tif');
imshow(I)
```

Add salt and pepper noise, with a noise density of 0.02, to the image. Display the result.

```
J = imnoise(I,'salt & pepper',0.02);
imshow(J)
```



Input Arguments

`I` — Grayscale image

numeric array

Grayscale image, specified as a numeric array of any dimensionality.

`imnoise` expects pixel values of data type `double` and `single` to be in the range [0, 1]. You can use the `rescale` function to adjust pixel values to the expected range. If your image is type `double` or `single` with values outside the range [0,1], then `imnoise` clips input pixel values to the range [0, 1] before adding noise.

Note For Poisson noise, `imnoise` does not support images of data type `int16`.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`m` — Mean of Gaussian noise

`0` (default) | numeric scalar

Mean of Gaussian noise, specified as a numeric scalar.

`var_gauss` — Variance of Gaussian noise

`0.01` (default) | numeric scalar

Variance of Gaussian noise, specified as a numeric scalar.

`var_local` — Local variance of Gaussian noise

numeric matrix | numeric vector

Local variance of Gaussian noise, specified as one of the following:

- A numeric matrix of the same size as `I`.
- A numeric vector the same length of `intensity_map`.

`intensity_map` — Intensity values
numeric vector

Intensity values that are mapped to Gaussian noise variance, specified as a numeric vector. The values are normalized to the range [0, 1].

You can plot the functional relationship between noise variance `var_local` and image intensity using the command `plot(intensity_map,var_local)`.

`d` — Noise density
`0.05` (default) | numeric scalar

Noise density for salt and pepper noise, specified as a numeric scalar. The noise is applied to approximately `d*numel(I)` pixels.

`var_speckle` — Variance of multiplicative noise
`0.05` (default) | numeric scalar

Variance of multiplicative noise, specified as a numeric scalar.

Output Arguments

`J` — Noisy image
numeric matrix

Noisy image, returned as a numeric matrix of the same data type as input image `I`. For images of data type `double` or `single`, the `imnoise` function clips output pixel values to the range [0, 1] after adding noise.

Algorithms

- The mean and variance parameters for `'gaussian'`, `'localvar'`, and `'speckle'` noise types are always specified as if the image were of class `double` in the range [0, 1]. If the input image is a different class, the `imnoise` function converts the image to `double`, adds noise according to the specified type and parameters, clips pixel values to the range [0, 1], and then converts the noisy image back to the same class as the input.
- The Poisson distribution depends on the data type of input image `I`:
  - If `I` is double precision, then input pixel values are interpreted as means of Poisson distributions scaled up by `1e12`. For example, if an input pixel has the value `5.5e-12`, then the corresponding output pixel will be generated from a Poisson distribution with mean of 5.5 and then scaled down by `1e12`.
  - If `I` is single precision, the scale factor used is `1e6`.
  - If `I` is `uint8` or `uint16`, then input pixel values are used directly without scaling. For example, if a pixel in a `uint8` input has the value 10, then the corresponding output pixel will be generated from a Poisson distribution with mean 10.

- To add `'salt & pepper'` noise with density `d` to an image, `imnoise` first assigns each pixel a random probability value from a standard uniform distribution on the open interval (0, 1).
  - For pixels with probability value in the range (0, `d`/2), the pixel value is set to `0`. The number of pixels that are set to `0` is approximately `d*numel(I)/2`.
  - For pixels with probability value in the range [`d`/2, `d`), the pixel value is set to the maximum value of the image data type. The number of pixels that are set to the maximum value is approximately `d*numel(I)/2`.
  - For pixels with probability value in the range [`d`, 1), the pixel value is unchanged.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imnoise` supports the generation of C code (requires MATLAB Coder). For more information, see "Code Generation for Image Processing".
- The value of string input arguments must be compile time constants.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see "Image Processing on a GPU".

See Also

`rand` | `randn`

Introduced before R2006a

imopen

Morphologically open image

Syntax

```
J = imopen(I,SE)
J = imopen(I,nhood)
```

Description

`J = imopen(I,SE)` performs morphological opening on the grayscale or binary image `I` using the structuring element `SE`. The morphological opening operation is an erosion followed by a dilation, using the same structuring element for both operations.

`J = imopen(I,nhood)` opens the image `I`, where `nhood` is a matrix of `0`s and `1`s that specifies the structuring element neighborhood.

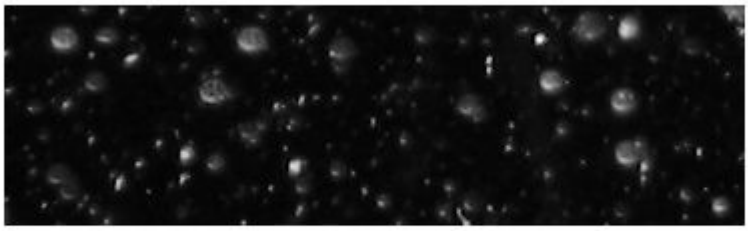
This syntax is equivalent to `imopen(I,strel(nhood))`.

Examples

Morphologically Open Image with a Disk-Shaped Structuring Element

Read the image into the workspace and display it.

```
original = imread('snowflakes.png');
imshow(original);
```



Create a disk-shaped structuring element with a radius of 5 pixels.

```
se = strel('disk',5);
```

Remove snowflakes having a radius less than 5 pixels by opening it with the disk-shaped structuring element.

```
afterOpening = imopen(original,se);
figure
imshow(afterOpening,[]);
```

Input Arguments

`I` — Input image

grayscale image | binary image

Input image, specified as a grayscale image or binary image of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`SE` — Structuring element

`strel` object | `offsetstrel` object

Structuring element, specified as a `strel` object or `offsetstrel` object. If the image `I` is of data type `logical`, the structuring element must be flat.

`nhood` — Structuring element neighborhood

matrix of `0`s and `1`s

Structuring element neighborhood, specified as a matrix of `0`s and `1`s.

Example: `[0 1 0; 1 1 1; 0 1 0]`

Output Arguments

`J` — Opened image

grayscale image | binary image

Opened image, returned as a grayscale image or binary image. `J` has the same data type as input image `I`.

Tips

- If the dimensionality of the image `I` is greater than the dimensionality of the structuring element, then the `imopen` function applies the same morphological opening to all planes along the higher dimensions.

You can use this behavior to perform morphological opening on RGB images. Specify a 2-D structuring element for RGB images to operate on each color channel separately.

- When you specify a structuring element neighborhood, `imopen` determines the center element of `nhood` by `floor((size(nhood)+1)/2)`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imopen` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imopen` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- The input image `I` must be 2-D or 3-D.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input image `I` must be 2-D or 3-D.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- `gpuArray` input must be of type `uint8` or `logical`.
- The structuring element `SE` must be flat and 2-D.

For more information, see “Image Processing on a GPU”.

See Also

Functions

`imclose` | `imdilate` | `imerode`

Objects

`strel` | `offsetstrel`

Introduced before R2006a

imoverlay

Burn binary mask into 2-D image

Syntax

```
B = imoverlay(A,BW)
B = imoverlay(A,BW,color)
```

Description

`B = imoverlay(A,BW)` fills the grayscale or RGB image `A` with a solid color where the input binary mask, `BW`, is `true`.

`B = imoverlay(A,BW,color)` specifies the color that `imoverlay` uses to fill the image.

Examples

Burn Binary Image into Grayscale Image

Read a grayscale image into the workspace.

```
A = imread('cameraman.tif');
```

Read a binary image into the workspace.

```
BW = imread('text.png');
```

Burn the binary image into the grayscale image, specifying the color to be used for the binary mask.

```
B = imoverlay(A,BW,'yellow');
```

Display the result.

```
imshow(B)
```

Burn Binary Image into RGB Image

Read an RGB image into the workspace.

```
RGB = imread('peppers.png');
```

Read a binary image into the workspace.

```
BW = imread('text.png');
```

Crop the RGB image to make it the same size as the binary mask.

```
RGB_cropped = imcrop(RGB,[64,128,255,255]);
```

Burn the binary image into the cropped RGB image, choosing the color to be used.

```
B = imoverlay(RGB_cropped,BW,'red');
```

Display the result.

```
figure
imshow(B)
```

Input Arguments

`A` — Input image

2-D grayscale image | 2-D RGB image

Input image, specified as a 2-D grayscale image or 2-D RGB image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`BW` — Mask image

2-D binary matrix | 2-D numeric matrix

Mask image, specified 2-D binary matrix or 2-D numeric matrix of the same size as the first two dimensions of image `A`. For numeric input, any nonzero pixels are considered to be `1` (`true`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`color` — Color used for overlay

`"yellow"` (default) | RGB triplet | color name | short color name

Color used for the overlay, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'r'

Example: 'green'

Example: [0 0.4470 0.7410]

Output Arguments

B — Output image

2-D RGB image

Output image, returned as a 2-D RGB image.

Data Types: uint8

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- imoverlay supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- When generating code, if you specify color as a character vector, then the value must be a compile-time constant.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, if you specify `color` as a character vector, then the value must be a compile-time constant.

See Also

`superpixels` | `boundarymask` | `labeloverlay`

Introduced in R2016a

imoverview

Overview tool for image displayed in scroll panel

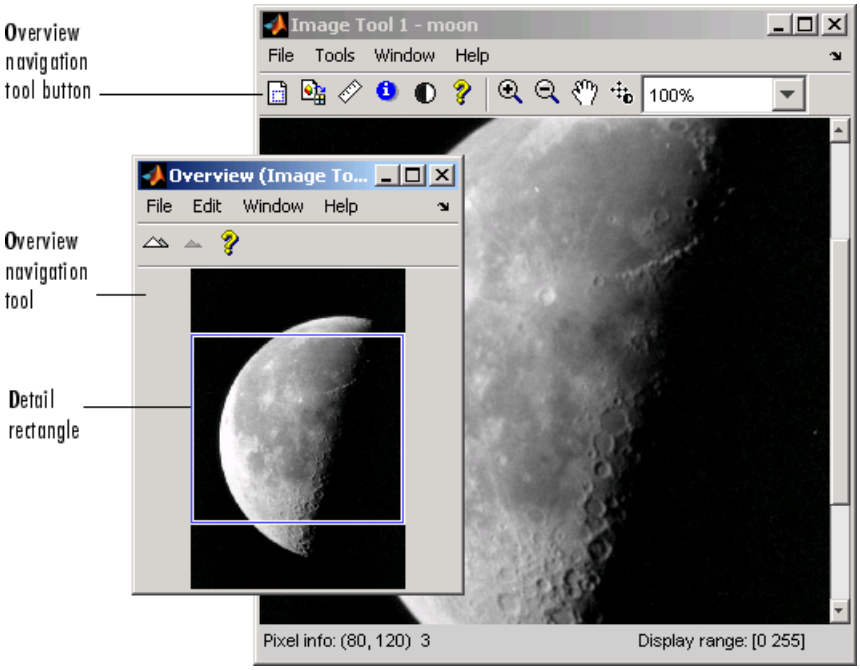
Syntax

```
imoverview(himage)
htool = imoverview( ___ )
```

Description

Use the `imoverview` function to create an Overview tool in a new figure window. The Overview tool is a navigation aid when exploring a zoomed-in version of the image.

The Overview tool displays the target image in its entirety, scaled to fit. The tool overlays a rectangle, called the detail rectangle, over the scaled version of the image. The detail rectangle shows the portion of the target image that is currently visible in the scroll panel. To view portions of the image that are not currently visible in the scroll panel, move the detail rectangle in the Overview tool.



`imoverview(himage)` creates an Overview tool associated with the image specified by the handle `himage`, called the target image.

`htool = imoverview( ___ )` returns `htool`, a handle to the Overview tool figure.

Examples

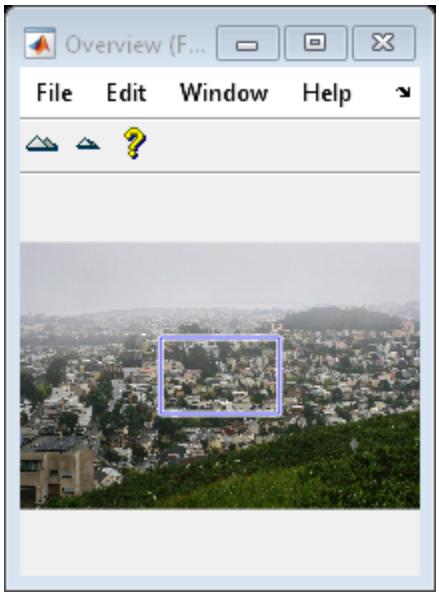
Create Overview Tool in New Figure

Display an image in a figure. Suppress the standard toolbar and menubar in the figure window because these do not work with the scroll panel.

```
hFig = figure('Toolbar','none','Menubar','none');
hIm = imshow('foggysf1.jpg');
```

Create a scroll panel to contain the image. Create an overview tool in a new figure window.

```
hSP = imscrollpanel(hFig,hIm);
imoverview(hIm)
```

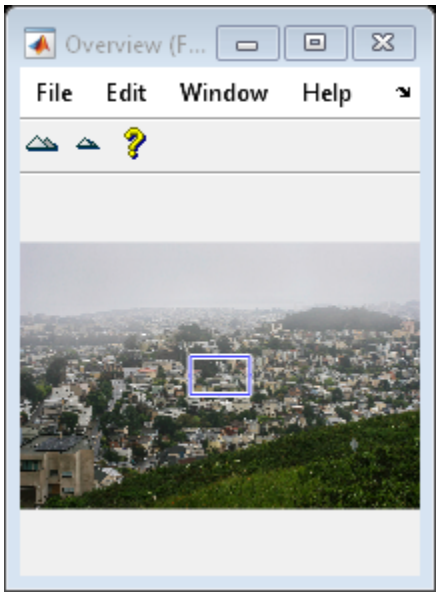


Get the scroll panel API so that you can control the view programmatically.

```
api = iptgetapi(hSP);
```

Set the magnification of the image to 200% by using the scroll panel API function `setMagnification`. Notice how the detail rectangle of the overview tool shrinks because a smaller portion of the image is displayed.

```
api = iptgetapi(hSP);
api.setMagnification(2);
```


Overview (F...
File
Edit
Window
Help

Input Arguments

`himage` — Handle to image

handle

Handle to image, specified as a handle. The image must be displayed in a scroll panel created by `imscrollpanel`.

Output Arguments

`htool` — Handle to Overview tool

handle

Handle to Overview tool figure, returned as a handle.

Tips

- To create an Overview tool that can be embedded in an existing figure or uipanel object, use `imoverviewpanel`.

See Also

`imoverviewpanel` | `imscrollpanel`

Introduced before R2006a

imoverviewpanel

Overview tool panel for image displayed in scroll panel

Syntax

```
hpanel = imoverviewpanel(hparent,himage)
```

Description

Use the `imoverviewpanel` function to add an Overview tool to the same figure as an image contained in a scroll panel. The Overview tool is a navigation aid when exploring a zoomed-in version of the image.

The Overview tool displays the target image in its entirety, scaled to fit. The tool overlays a rectangle, called the detail rectangle, over the scaled version of the image. The detail rectangle shows the portion of the target image that is currently visible in the scroll panel. To view portions of the image that are not currently visible in the scroll panel, move the detail rectangle in the Overview tool.

`hpanel = imoverviewpanel(hparent,himage)` creates an Overview tool for an image displayed in a scroll panel. `himage` is a handle to the target image in the scroll panel. `hparent` is the handle to the figure or uipanel object that will contain the Overview tool. `hpanel` is the handle to the Overview tool.

Examples

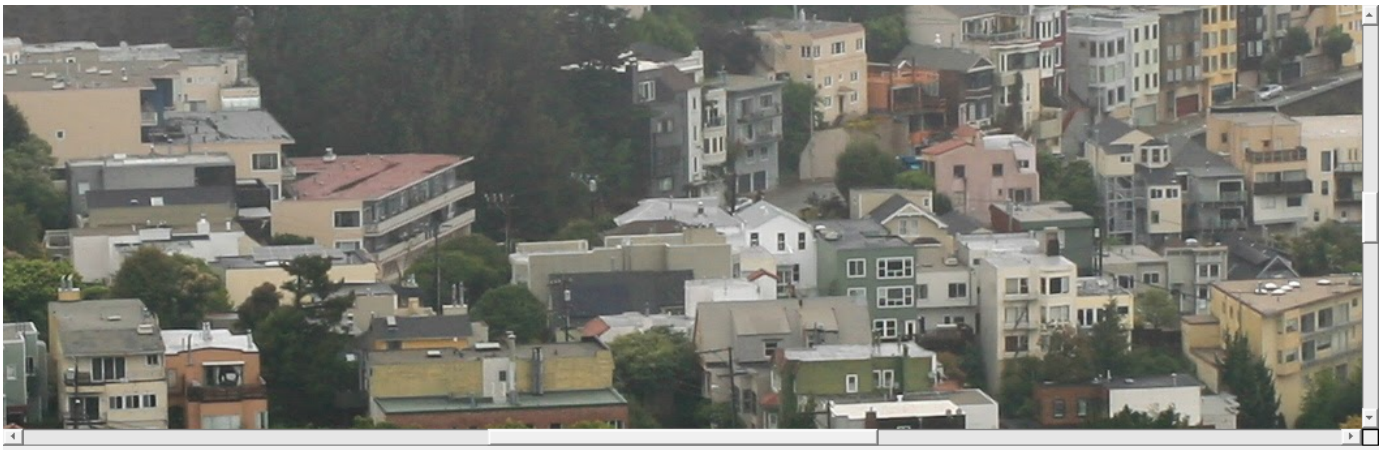
Add Overview Tool to Scrollable Image

Display an image in a figure. The example suppresses the standard toolbar and menubar in the figure window because these do not work with the scroll panel.

```
hFig = figure('Toolbar','none','Menubar','none');
hIm = imshow('foggysf1.jpg');
```

Create a scroll panel to contain the image. Set the size and position of the scroll panel so that the image occupies the top half of the figure window.

```
hSP = imscrollpanel(hFig,hIm);
set(hSP,'Units','normalized','Position',[0 .5 1 .5])
```



Add an overview tool to the figure. Set the size and position of the overview tool to occupy the bottom half of the figure window.

To explore details of the displayed image, try dragging the detail rectangle over the overview tool.

```
hOvPanel = imoverviewpanel(hFig,hIm);
set(hOvPanel,'Units','Normalized','Position',[0 0 1 .5])
```

Input Arguments

`hparent` — Handle to figure or uipanel object

handle

Handle to a figure or uipanel object that contains the Overview tool, specified as a handle.

`himage` — Handle to target image

handle

Handle to target image, specified as a handle. The image must be displayed in a scroll panel created by `imscrollpanel`.

Output Arguments

`hpanel` — Handle to Overview tool

handle

Handle to Overview tool, returned as a handle. An Overview tool is a type of uipanel object.

Tips

- To create an Overview tool in a separate figure window, use `imoverview`. When created using `imoverview`, the Overview tool includes zoom-in and zoom-out buttons.

See Also

`imoverview` | `imscrollpanel`

Introduced before R2006a

impixel

Pixel color values

Syntax

```
P = impixel
P = impixel(I)
P = impixel(X,cmap)

P = impixel(I,xi,yi)
P = impixel(X,cmap,xi,yi)
P = impixel(xref,yref,I,xi,yi)
P = impixel(xref,yref,X,cmap,xi,yi)

[xi2,yi2,P] = impixel( ___ )
```

Description

Select Pixels Interactively

`P = impixel` lets you select pixels interactively from the image in the current axes. When you finish selecting pixels, `impixel` returns the pixel values in `p`.

Use normal button clicks to select pixels. Press **Backspace** or **Delete** to remove the previously selected pixel. To add a final pixel and finish pixel selection in one step, press shift-click, or right-click or double-click. To finish selecting pixels without adding a final pixel, press **Return**. With this syntax and the other interactive syntaxes, the pixel selection tool blocks the MATLAB command line until you complete the operation.

`P = impixel(I)` displays the grayscale, RGB, or binary image `I` in a figure window and waits for you to select pixels in the image using the mouse.

`P = impixel(X,cmap)` displays the indexed image `X` with colormap `cmap` in a figure window and waits for you to select pixels in the image using the mouse.

Select Pixels by Specifying Coordinates

`P = impixel(I,xi,yi)` returns the values of pixels in grayscale, truecolor, or binary image `I`. The pixels have (*x*, *y*) coordinates `xi` and `yi`.

`P = impixel(X,cmap,xi,yi)` returns the values of pixels in indexed image `X` with colormap `cmap`. The pixels have (*x*, *y*) coordinates `xi` and `yi`.

`P = impixel(xref,yref,I,xi,yi)` returns the values of pixels in image `I` using the world coordinate system defined by `xref` and `yref`. The pixel vertices have (*x*, *y*) coordinates `xi` and `yi` in this coordinate system.

`P = impixel(xref,yref,X,cmap,xi,yi)` returns the values of pixels in the indexed image `X` with colormap `cmap`, using the world coordinate system defined by `xref` and `yref`. The pixel vertices have (*x*, *y*) coordinates `xi` and `yi` in this coordinate system.

Additionally Return Selected Pixel Coordinates

`[xi2,yi2,P] = impixel( ___ )` additionally returns the (*x*, *y*) coordinates of the selected pixels, `xi2` and `yi2`. You can use the input arguments of any other syntax.

Examples

Return Individual Pixel Values from Image

Read a truecolor image into the workspace.

```
RGB = imread('peppers.png');
```

Determine the column `c` and row `r` indices of the pixels to extract.

```
c = [1 12 146 410];
r = [1 104 156 129];
```

Return the data at the selected pixel locations.

```
pixels = impixel(RGB,c,r)
```

```
pixels = 4×3

    62    29    64
    62    34    63
   166    54    60
    59    28    47
```

Input Arguments

`I` — Image

numeric matrix | numeric array | logical matrix

Image, specified as one of the following.

- *m*-by-*n* numeric matrix representing a grayscale image
- *m*-by-*n*-by-3 numeric array representing a truecolor image
- *m*-by-*n* logical matrix representing a binary mask.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`X` — Indexed image

matrix of integers

Indexed image, specified as a matrix of integers.

Data Types: `single` | `double` | `uint8` | `uint16` | `logical`

`cmap` — Colormap

c-by-3 numeric matrix

Colormap associated with the indexed image `X`, specified as a *c*-by-3 numeric matrix. Each row is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap. Values with data type `single` or `double` must be in the range [0, 1].

Data Types: `single` | `double` | `uint8`

`xi` — *x*-coordinate of pixels to sample
numeric vector

x-coordinate of pixels to sample, specified as a numeric vector of the same length and data type as `yi`. If you specify image limits in a world coordinate system using `xref`, then `xi` is in this coordinate system. Otherwise, `xi` is in the default spatial coordinate system.

Data Types: `single` | `double`

`yi` — *y*-coordinate of pixels to sample
numeric vector

y-coordinate of pixels to sample, specified as a numeric vector of the same length and data type as `xi`. If you specify image limits in a world coordinate system using `yref`, then `yi` is in this coordinate system. Otherwise, `yi` is in the default spatial coordinate system.

Data Types: `single` | `double`

`xref` — Image limits in world coordinates along *x*-dimension
2-element numeric vector

Image limits in world coordinates along the *x*-dimension, specified as a 2-element numeric vector of the form [*xmin xmax*]. The value of `xref` sets the image `XData`. The data type of `xref` and `yref` must match.

Data Types: `single` | `double`

`yref` — Image limits in world coordinates along *y*-dimension
2-element numeric vector

Image limits in world coordinates along the *y*-dimension, specified as a 2-element numeric vector of the form [*ymin ymax*]. The value of `yref` sets the image `YData`. The data type of `xref` and `yref` must match.

Data Types: `single` | `double`

Output Arguments

`P` — Sampled pixel values
p-by-3 matrix

Sampled pixel values, returned as a *p*-by-3 matrix. `impixel` always returns pixel values as RGB triplets, regardless of the image type. The values in each row of the matrix depends on the image type.

Image Type	Result
RGB	Returns the actual RGB data for the pixel. The values are data type `double`.

Image Type	Result
Grayscale	Returns the intensity value as an RGB triplet, where R=G=B. The values are data type `double`.
Indexed	Returns the RGB triplet stored in the row of the colormap that the pixel value points to. The values have the same data type as the colormap, `cmap`.
Binary	Returns the intensity value as an RGB triplet, where R=G=B. The values are data type `double`.

`xi2` — *x*-coordinates of sampled pixels

numeric vector

x-coordinates of sampled pixels, returned as a numeric vector.

- If you select pixels interactively using the mouse, then `xi2` is interpreted as column indices.
- If you specify pixel coordinates to sample when you call `impixel`, then `xi2` is interpreted as *x*-coordinates in the same coordinate system as `xi`.

`yi2` — *y*-coordinates of sampled pixels

numeric vector

y-coordinates of sampled pixels, returned as a numeric vector.

- If you select pixels interactively using the mouse, then `yi2` is interpreted as row indices.
- If you specify pixel coordinates to sample when you call `impixel`, then `yi2` is interpreted as *y*-coordinates in the same coordinate system as `yi`.

See Also

`improfile` | `getpts`

Topics

“Image Types in the Toolbox”
“Define World Coordinate System of Image”

Introduced before R2006a

impixelinfo

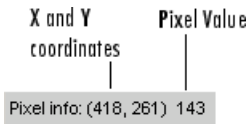
Pixel Information tool

Syntax

```
impixelinfo
impixelinfo(h)
impixelinfo(hparent,himage)
htool = impixelinfo( ___ )
```

Description

Use the `impixelinfo` function to create a Pixel Information tool. The Pixel Information tool displays information about the pixel in an image that the pointer is positioned over. If the figure contains multiple images, the tool displays pixel information for all the images. For more information about the tool, see “Pixel Information Tool” on page 1-1711.



`impixelinfo` creates a Pixel Information tool in the current figure.

`impixelinfo(h)` creates a Pixel Information tool in the figure specified by the handle `h`.

`impixelinfo(hparent,himage)` creates a Pixel Information tool in `hparent` that provides information about the pixels in `himage`.

`htool = impixelinfo( ___ )` returns a handle to the Pixel Information tool uipanel.

Examples

Add Pixel Information Tool to Figure

Display an image and add a Pixel Information tool to the figure. The example shows how you can change the position of the tool in the figure using properties of the tool uipanel object.

```
h = imshow("hestain.png");
hp = impixelinfo;
set(hp,"Position",[5 1 300 20]);
```

Use the Pixel Information tool in a figure containing multiple images of different types.

```
figure
subplot(1,2,1), imshow("liftingbody.png")
```

```
subplot(1,2,2), imshow("autumn.tif")
impixelinfo;
```

Input Arguments

h — Handle to graphics object

handle

Handle to a figure, axes, uipanel, or image graphics object, specified as a handle. Axes, uipanel, or figure objects must contain at least one image object.

hparent — Handle to figure or uipanel object

handle

Handle to a figure or uipanel object that contains the Pixel Information tool, specified as a handle.

himage — Handle to images

handle | array of handles

Handle to one or more images, specified as a handle or an array of image handles.

Output Arguments

htool — Handle to Pixel Information tool

handle

Handle to Pixel Information tool uipanel, returned as a handle.

More About

Pixel Information Tool

The Pixel Information tool is a uipanel object, positioned in the lower-left corner of the figure. The tool contains the text label **Pixel info:** followed by the pixel information. Before you move the pointer over the image, the tool contains the default pixel information text `(X,Y) Pixel Value`. Once you move the pointer over the image, the information displayed varies by image type, as shown in the following table. If you move the pointer off the image, the pixel information tool displays the default pixel information label for that image type.

Image Type	Pixel Information	Example
Intensity	(X,Y) Intensity	(13,30) 82
Indexed	(X,Y) <index> [R G B]	(2,6) <4> [0.29 0.05 0.32]
Binary	(X,Y) BW	(12,1) 0
Truecolor	(X,Y) [R G B]	(19,10) [15 255 10]
Floating point image with `CDataMapping` property set to `direct`	(X,Y) value <index> [R G B]	(19,10) 82 <4> [15 255 10]

Tips

- If you want to display the pixel information without the "Pixel Info" label, then use the `impixelinfoval` function.
- To copy the pixel information label to the clipboard, right-click while the pointer is positioned over a pixel. In the context menu displayed, choose **Copy pixel info**.

See Also

Image Viewer | `impixelinfoval`

Topics

“Get Started with Image Viewer App”

Introduced before R2006a

impixelinfoval

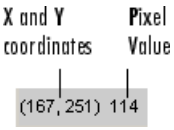
Pixel Information tool without text label

Syntax

```
htool = impixelinfoval(hparent,himage)
```

Description

Use the `impixelinfoval` function to create a Pixel Information tool without the **Pixel info:** text label. The Pixel Information tool displays information about the pixel in an image that the pointer is positioned over. If the figure contains multiple images, the tool displays pixel information for all the images. The information displayed depends on the image type. See `impixelinfo` for more details about using the Pixel Information tool.



`htool = impixelinfoval(hparent,himage)` creates a Pixel Information tool in `hparent` that provides information about the pixels in `himage`.

Examples

Add Pixel Information Tool Without Text Label

Add a Pixel Information tool to a figure, excluding the text label. Note how you can change the style and size of the font used to display the value in the tool using standard graphics object properties.

```
ankle = dicomread('CT-MONO2-16-ankle.dcm');
h = imshow(ankle,[]);
hText = impixelinfoval(gcf,h);
set(hText,'FontWeight','bold')
set(hText,'FontSize',10)
```

Input Arguments

`hparent` — Handle to figure or uipanel object

handle

Handle to a figure or uipanel object that contains the Pixel Information tool, specified as a handle.

`himage` — Handle to images

handle | array of handles

Handle to one or more images, specified as a handle or an array of image handles.

Output Arguments

`htool` — Handle to Pixel Information tool

handle

Handle to Pixel Information tool uipanel, returned as a handle.

See Also

`impixelinfo`

Introduced before R2006a

impixelregion

Pixel Region tool

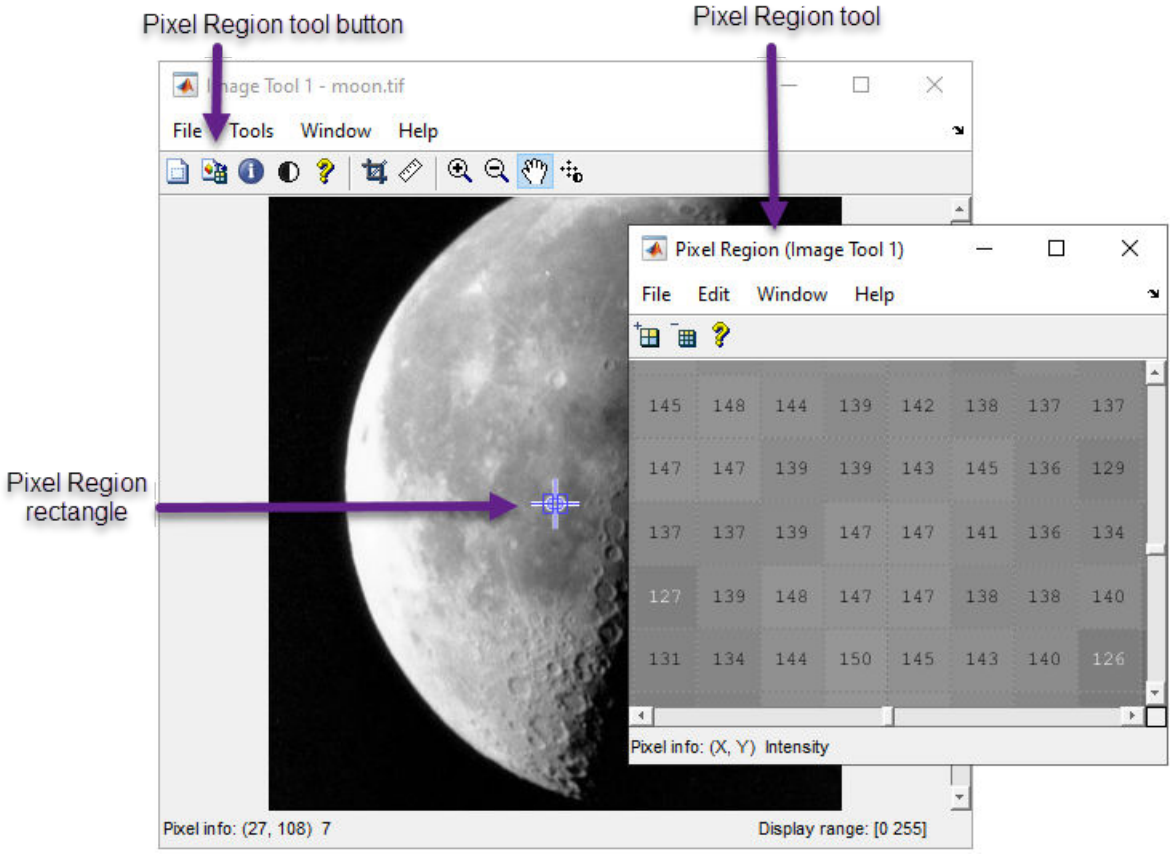
Syntax

```
impixelregion
impixelregion(h)
htool = impixelregion( ___ )
```

Description

Use the `impixelregion` function to create a Pixel Region tool in a new figure window. The Pixel Region tool is an aid to explore pixel values of images.

The Pixel Region tool displays an extreme close-up view of a small region of pixels in the target image. The tool overlays a rectangle, called the pixel region rectangle, over the target image. To view pixels in a different region, click and drag the rectangle over the target image, or scroll the Pixel Region tool. You can resize the pixel region rectangle to change the resolution of pixels in the Pixel Region tool. If the size of the pixels allows, the tool superimposes the numeric value of the pixel over each pixel.



`impixelregion` creates a Pixel Region tool associated with the image displayed in the current figure, called the target image.

`impixelregion(h)` creates a Pixel Region tool in the figure specified by the handle `h`.

`htool = impixelregion( ___ )` returns `htool`, a handle to the Pixel Region tool figure.

Examples

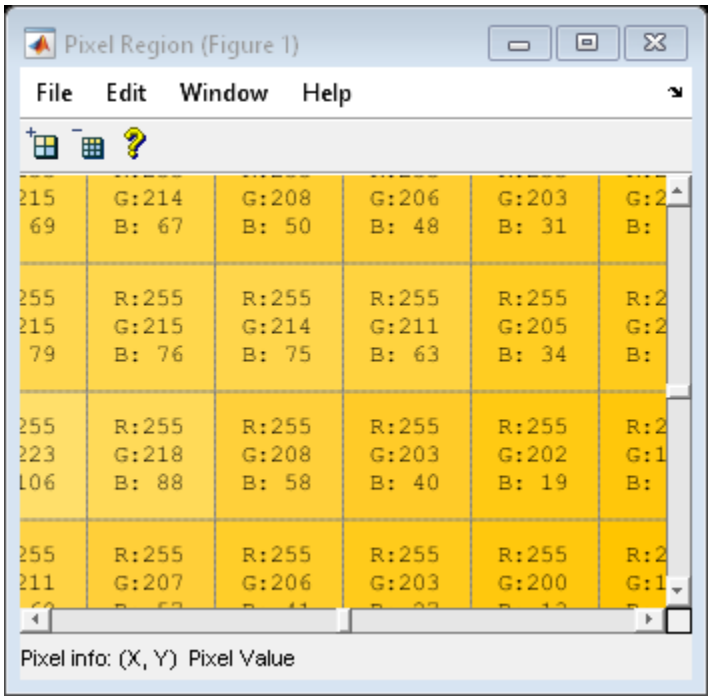
Create Pixel Region Tool in New Figure

Display an image.

```
imshow('peppers.png')
```

Create an pixel region tool in a new figure window. The tool associates with the image in the current figure.

```
impixelregion
```



Input Arguments

h — Handle to graphics object

handle

Handle to a figure, axes, uipanel, or image graphics object, specified as a handle. If `h` is an axes or figure handle, `impixelregion` uses the first image returned by `findobj(H,'Type','image')`.

Output Arguments

htool — Handle to Pixel Region tool

handle

Handle to Pixel Region tool figure, returned as a handle.

Tips

- To get a closer view of the pixels displayed in the tool, use the zoom buttons on the Pixel Region tool toolbar.
- To get the current position of the pixel region rectangle, right-click on the rectangle and select **Copy Position** from the context menu. The Pixel Region tool copies a four-element position vector to the clipboard. To change the color of the pixel region rectangle, right-click and select **Set Color**.
- To create a Pixel Region tool that can be embedded in an existing figure window or uipanel, use `impixelregionpanel`.

See Also

Image Viewer | `impixelinfo` | `impixelregionpanel`

Introduced before R2006a

impixelregionpanel

Pixel Region tool panel

Syntax

```
hpanel = impixelregionpanel(hparent,himage)
```

Description

Use the `impixelregionpanel` function to add a Pixel Region tool to the same figure as an image. The Pixel Region tool is an aid to explore pixel values of images.

The Pixel Region tool displays an extreme close-up view of a small region of pixels in the target image. The tool overlays a rectangle, called the pixel region rectangle, over the target image. To view pixels in a different region, click and drag the rectangle over the target image, or scroll the Pixel Region tool. You can resize the pixel region rectangle to change the resolution of pixels in the Pixel Region tool. If the size of the pixels allows, the tool superimposes the numeric value of the pixel over each pixel.

`hpanel = impixelregionpanel(hparent,himage)` creates a Pixel Region tool in a figure window. `himage` is a handle to the target image whose pixels are to be displayed. `hparent` is the handle to the figure or uipanel object that will contain the Pixel Region tool. `hpanel` is the handle to the Pixel Region tool.

Examples

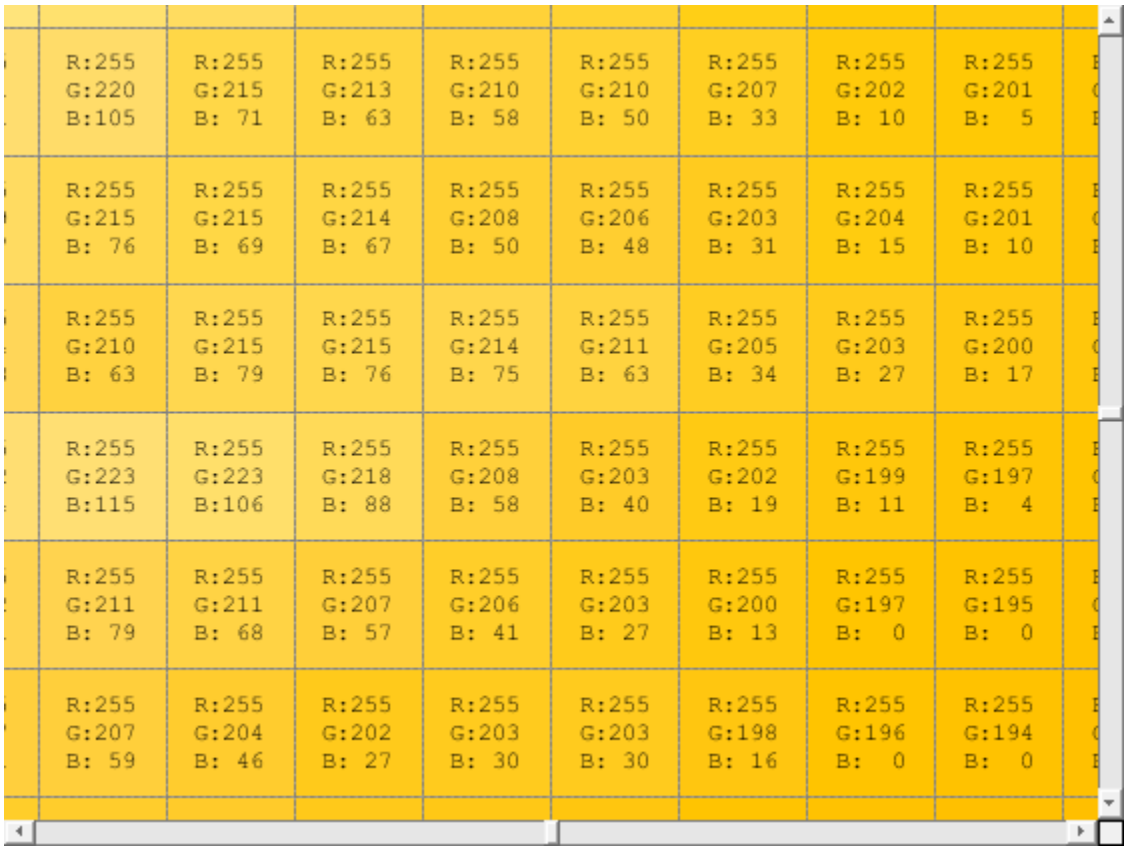
Add Pixel Region Tool to Figure

Display an image in a figure. This example displays the image in a subplot to create space in the figure window for the pixel region tool.

```
hFig = figure;
subplot(1,2,1)
hIm = imshow('peppers.png');
```

Create a pixel region tool in the same figure as the image. The pixel region tool covers the entire figure window.

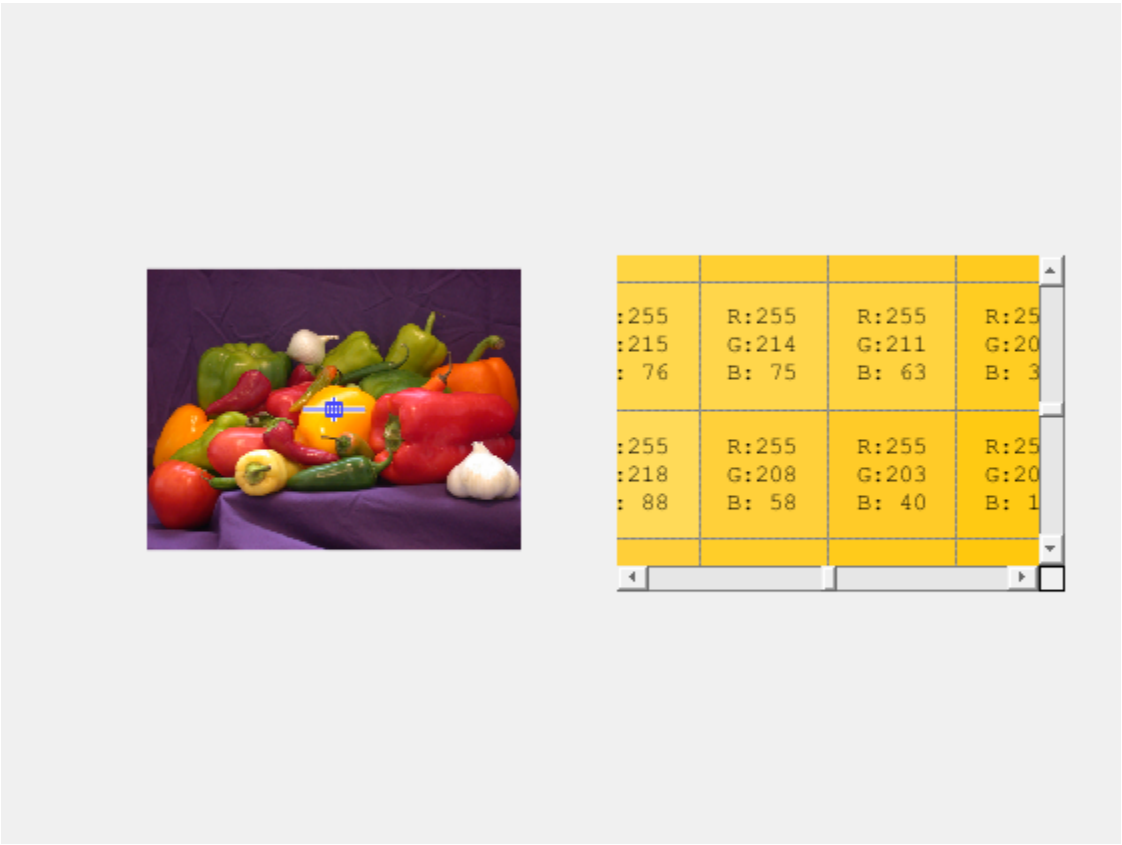
```
hpanel = impixelregionpanel(hFig,hIm);
```



Reduce the dimensions of the pixel region tool to 40% of the height and width of the figure. Specify the position of the bottom left corner of the tool so that the tool occupies the space in the figure to the right of the image.

To explore pixel values across the image, try dragging and resizing the pixel region rectangle over the target image and scrolling the pixel region tool.

```
set(hpanel,'Position',[0.55 0.3 .4 .4])
```



Input Arguments

`hparent` — Handle to figure or uipanel object

handle

Handle to a figure or uipanel object that will contain the Pixel Region tool, specified as a handle.

`himage` — Handle to target image

handle

Handle to target image, specified as a handle.

Output Arguments

`hpanel` — Handle to Pixel Region tool

handle

Handle to Pixel Region tool, returned as a handle. A Pixel Region tool is a type of uipanel object.

Tips

- To create a Pixel Region tool in a separate figure window, use `impixelregion`.

See Also

`impixelregion` | `imscrollpanel`

Introduced before R2006a

improfile

Pixel-value cross-sections along line segments

Syntax

```
c = improfile
c = improfile(n)

c = improfile(I,xi,yi)
c = improfile(xref,yref,I,xi,yi)
c = improfile( ___ ,n)

c = improfile( ___ ,method)
[cx,cy,c] = improfile( ___ )
[cx,cy,c,xi2,yi2] = improfile( ___ )
improfile( ___ )
```

Description

Select Line Segments Interactively

`c = improfile` lets you select line segments interactively from the image in the current axes. When you finish selecting line segments, `improfile` returns sampled pixel values along the line segments in `c`.

With this syntax, you specify the line or path using the mouse, by clicking points in the image. Press **Backspace** or **Delete** to remove the previously selected point. To finish selecting points, adding a final point, press shift-click, right-click, or double-click. To finish selecting points without adding a final point, press **Return**.

`c = improfile(n)` returns `n` sampled pixel values from line segments that you select interactively.

Select Line Segments by Specifying Endpoints

`c = improfile(I,xi,yi)` returns sampled pixel values along line segments in image `I`. The endpoints of the line segments have (*x*, *y*) coordinates `xi` and `yi`.

`c = improfile(xref,yref,I,xi,yi)` returns pixel values in the world coordinate system defined by `xref` and `yref`. The line segment endpoints have (*x*, *y*) coordinates `xi` and `yi` in this coordinate system.

`c = improfile( ___ ,n)` returns `n` sampled pixel values along the line segments.

Specify Interpolation Method or Output Options

`c = improfile( ___ ,method)` specifies the interpolation method for pixel coordinates. Before the `method` input argument, you can specify the input arguments of any other syntax.

`[cx,cy,c] = improfile( ___ )` additionally returns the (*x*, *y*) coordinates of the sampled pixels, `cx` and `cy`. You can use the input arguments of any other syntax.

`[cx,cy,c,xi2,yi2] = improfile( ___ )` additionally returns the (*x*, *y*) coordinates of the line segment endpoints, `xi` and `yi`.

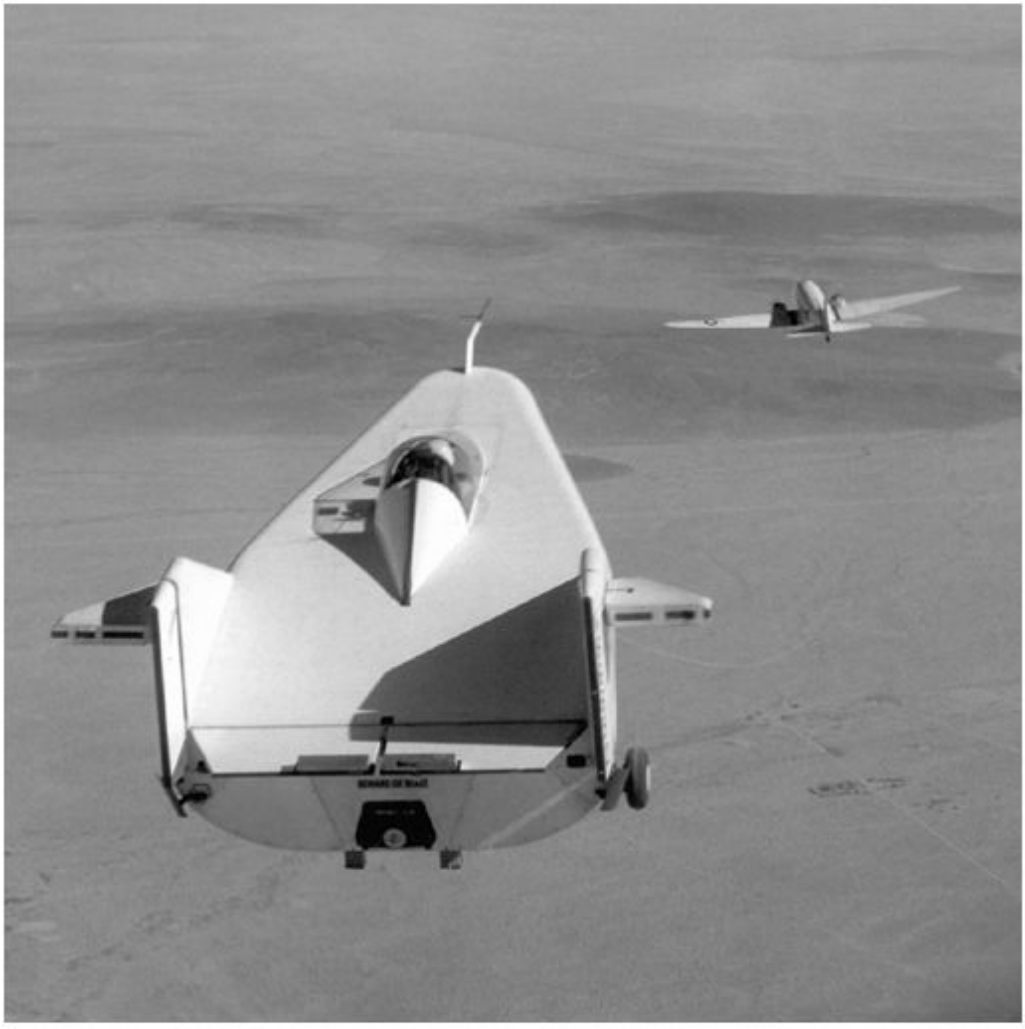
`improfile( ___ )` without output arguments displays a plot of pixel values along the line segments. If you select a single line segment, then `improfile` creates a two-dimensional plot of intensity values versus the distance along the line segment. If you select two or more line segments, then `improfile` creates a three-dimensional plot of the intensity values versus their *x*- and *y*-coordinates.

Examples

Plot Multisegment Line from Image

Read an image into the workspace, and display it.

```
I = imread('liftingbody.png');
imshow(I)
```

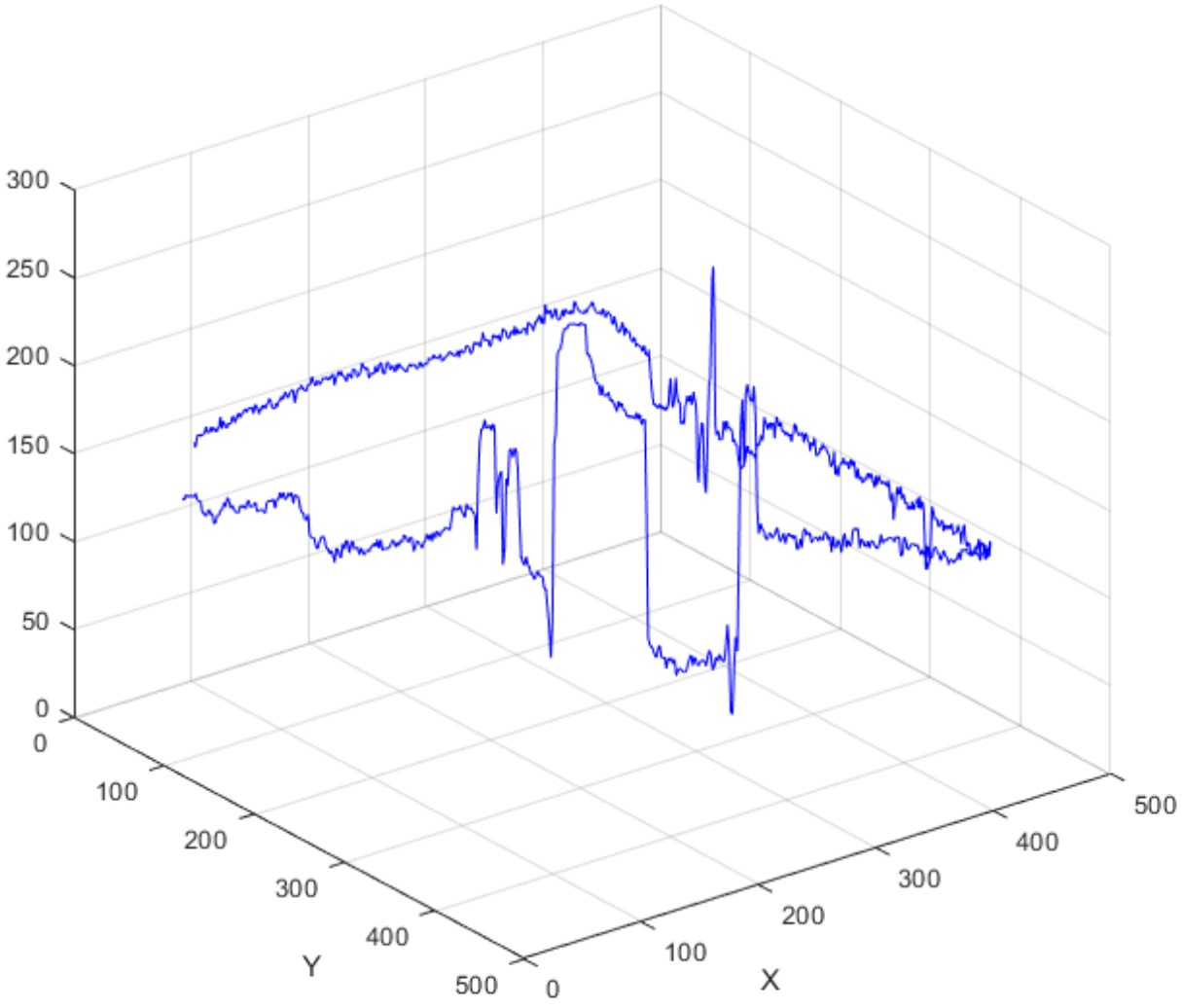


Specify *x*- and *y*-coordinates that define connected line segments.

```
x = [19 427 416 77];
y = [96 462 37 33];
```

Display a 3-D plot of the pixel values of these line segments.

```
improfile(I,x,y),grid on;
```



Input Arguments

`n` — Number of points

positive integer

Number of points to along the path to sample, specified as a positive integer. If you do not provide this argument, then `improfile` chooses a value for `n` that is roughly equal to the number of pixels that the path traverses.

Data Types: `double`

`I` — Input image

RGB image | grayscale image | binary image

Input image, specified as an RGB image, grayscale image, or binary image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`xi` — *x*-coordinate of line segment endpoints
numeric vector

x-coordinate of line segment endpoints, specified as a numeric vector of the same length as `yi`. If you specify image limits in a world coordinate system using `xref`, then `xi` is in this coordinate system. Otherwise, `xi` is in the default spatial coordinate system.

Data Types: `double`

`yi` — *y*-coordinate of line segment endpoints
numeric vector

y-coordinate of line segment endpoints, specified as a numeric vector of the same length as `xi`. If you specify image limits in a world coordinate system using `yref`, then `yi` is in this coordinate system. Otherwise, `yi` is in the default spatial coordinate system.

Data Types: `double`

`xref` — Image limits in world coordinates along *x*-dimension
2-element numeric vector

Image limits in world coordinates along the *x*-dimension, specified as a 2-element numeric vector of the form [*xmin xmax*]. The value of `xref` sets the image `XData`. The data type of `xref` and `yref` must match.

Data Types: `single` | `double`

`yref` — Image limits in world coordinates along *y*-dimension
2-element numeric vector

Image limits in world coordinates along the *y*-dimension, specified as a 2-element numeric vector of the form [*ymin ymax*]. The value of `yref` sets the image `YData`. The data type of `xref` and `yref` must match.

Data Types: `single` | `double`

`method` — Interpolation method
`'nearest'` (default) | `'bilinear'` | `'bicubic'`

Interpolation method, specified as `'nearest'` for nearest-neighbor interpolation, `'bilinear'`, or `'bicubic'`.

Data Types: `char` | `string`

Output Arguments

`c` — Sampled pixel values
n-by-1 numeric vector | n-by-1-by-3 numeric array

Sampled pixel values, returned as an n-by-1 numeric vector when `I` is a grayscale or binary image, or an n-by-1-by-3 numeric array when `I` is an RGB image.

Data Types: `double`

`cx` — *x*-coordinate of sampled pixels
n-by-1 numeric vector

x-coordinate of sampled pixels, returned as an `n`-by-1 numeric vector.

Data Types: `double`

`cy` — *y*-coordinate of sampled pixels
n-by-1 numeric vector

y-coordinate of sampled pixels, returned as an `n`-by-1 numeric vector.

Data Types: `double`

`xi2` — *x*-coordinate of line segment endpoints
numeric vector

x-coordinate of line segment endpoints, returned as a numeric vector. If you specify line segment endpoints using `xi`, then `xi2` is equal to `xi`.

Data Types: `double`

`yi2` — *y*-coordinate of line segment endpoints
numeric vector

y-coordinate of line segment endpoints, returned as a numeric vector. If you specify line segment endpoints using `yi`, then `yi2` is equal to `yi`.

Data Types: `double`

See Also

`impixel` | `interp2`

Topics
“Image Types in the Toolbox”
“Define World Coordinate System of Image”

Introduced before R2006a

imputfile

Display Save Image dialog box

Syntax

```
[filename,ext,user_canceled] = imputfile
```

Description

`[filename,ext,user_canceled] = imputfile` displays the **Save Image** dialog box which you can use to specify the full path and format of a file. Using the dialog box, you can navigate to folders in a file system and select a particular file or specify the name of a new file. `imputfile` limits the types of files displayed in the dialog box to the image file format selected in the Files of Type menu.

If you click **Save**, then `imputfile` returns the full path to the file in `filename` and the file extension associated with the file format selected from the **Files of Type** menu in `ext`. `imputfile` automatically adds the file name extension (such as `.jpg`) to the file name.

If you click **Cancel** or close the **Save Image** dialog box, then `imputfile` closes and returns control to MATLAB, sets `user_canceled` to `True` (1), and sets `filename` and `ext` to empty character vectors (`''`). Otherwise, `user_canceled` is `False` (0).

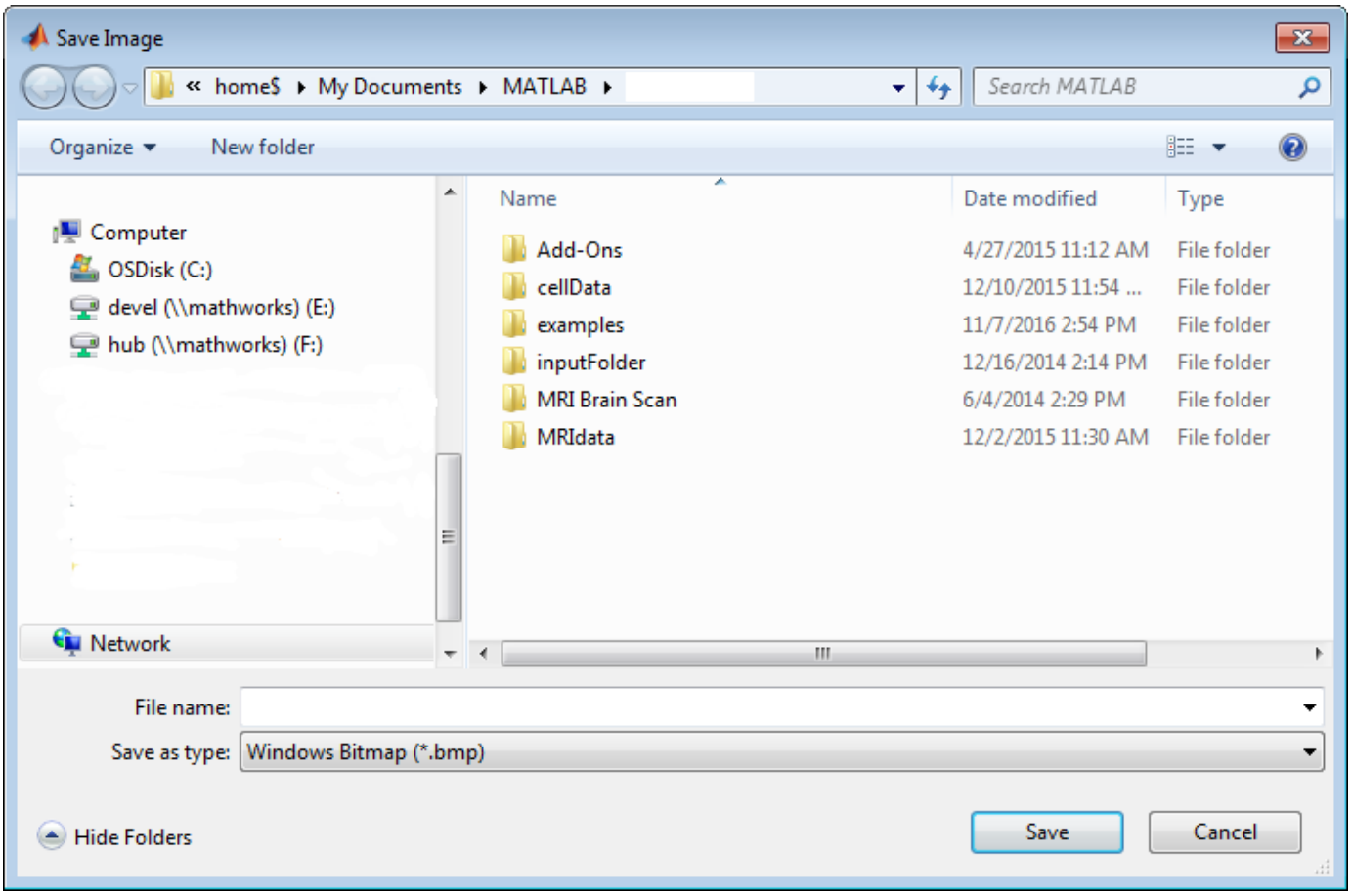
Note The **Save Image** dialog box is modal; it blocks the MATLAB command line until you click **Save** or cancel the operation.

Examples

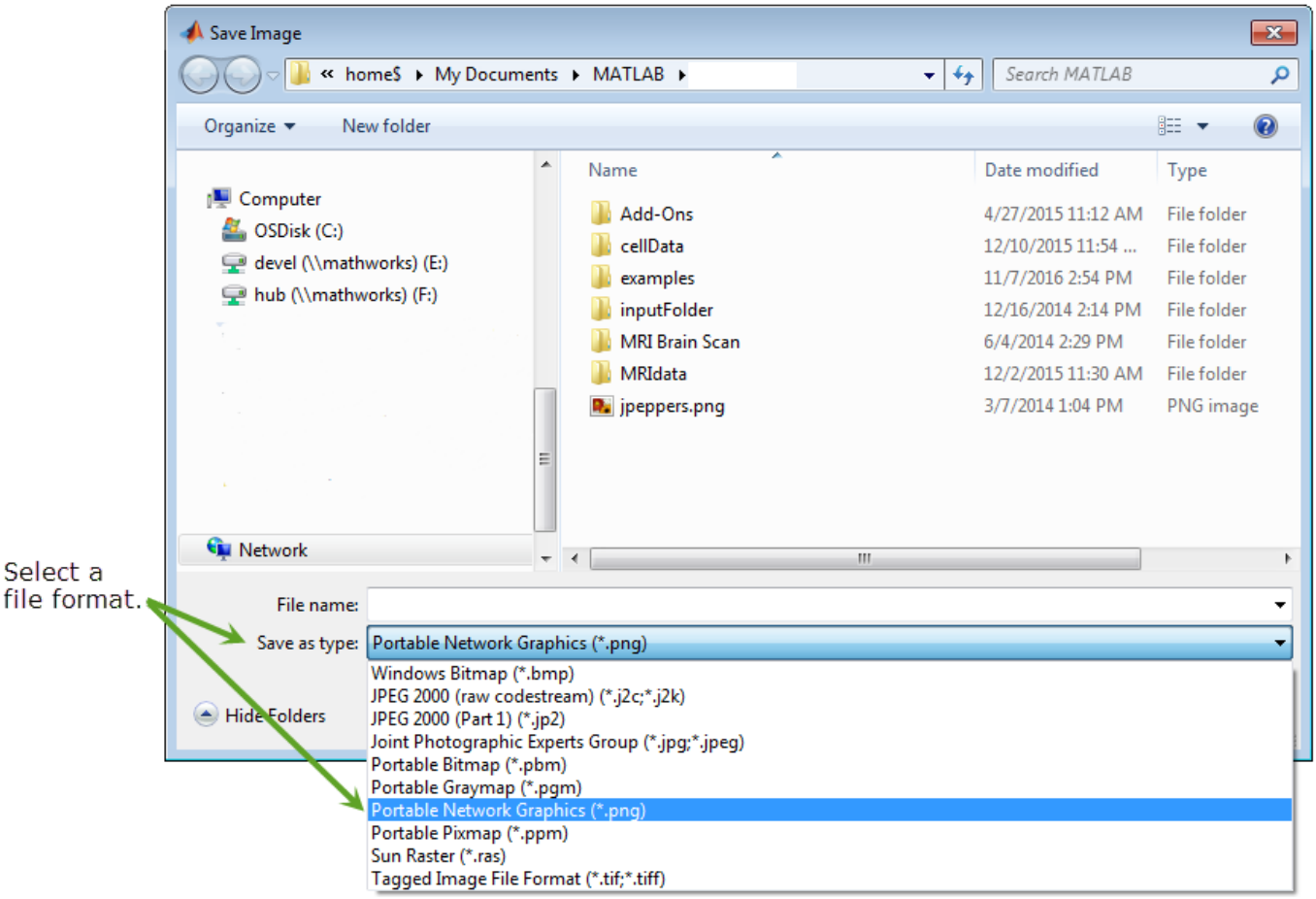
Get User-Specified File Name

Open the **Save Image** dialog box. This dialog box is modal—control in the command window is suspended until you respond to the **Save Image** dialog box.

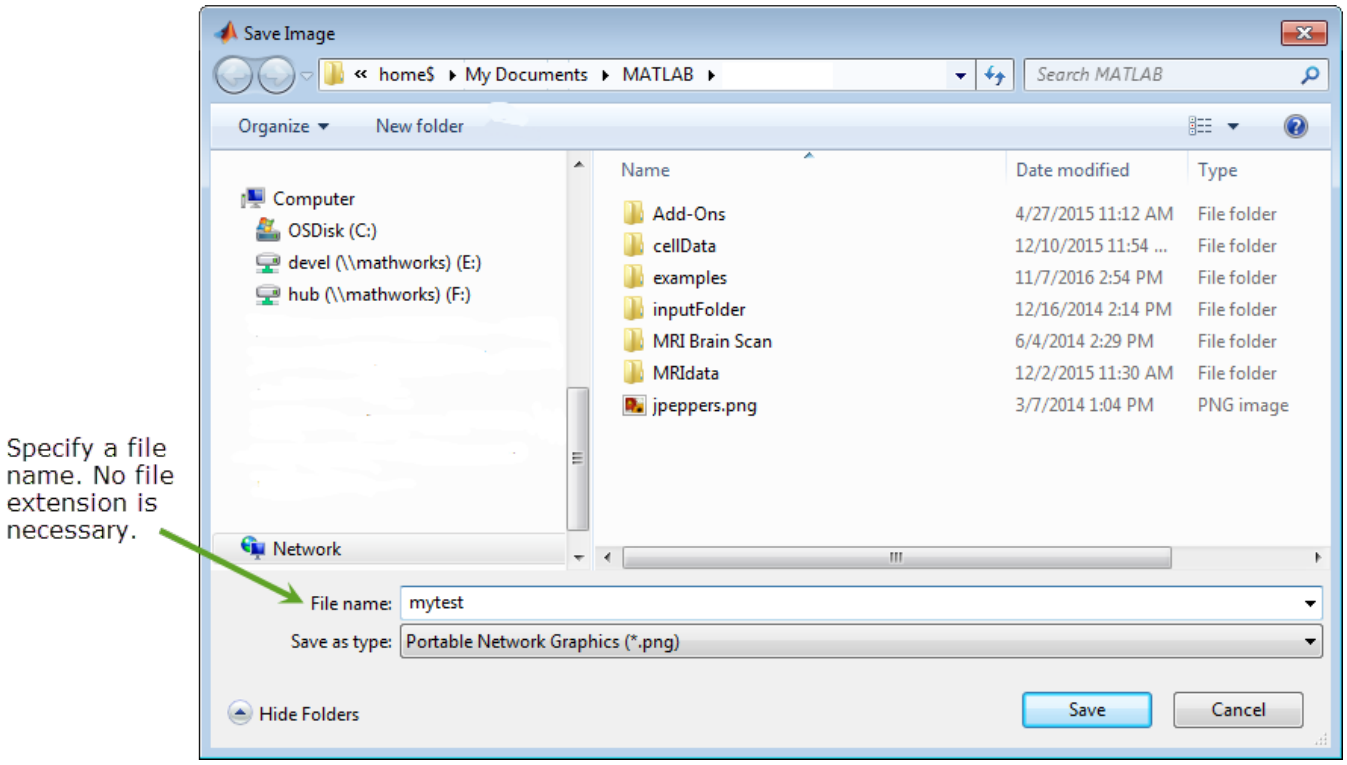
```
[fn, ext, ucancel] = imputfile
```



To view only images in Portable Network Graphics format, select the format from the **Save as type** menu.



Specify a new file name and click **Save**. `imputfile` returns the full path of the file name you specified, the file extension, and the Boolean value `false`, meaning that you did not click **Cancel**. Note that `imputfile` automatically adds the file extension of the format you selected to the file name.




```
fn =

  1×37 char array

    '\\home$\Documents\MATLAB\mytest.png'


ext =

  1×3 char array

    'png'


ucancel =

  logical

   0
```

Output Arguments

`filename` — Name of file selected

character array

Name of file selected, returned as a character array.

`ext` — File extension of a supported file format

character array

File extension of a supported file format, returned as a character array.

`user_canceled` — Flag indicating if user chose to cancel dialog
logical

Flag indicating if user chose to cancel dialog, returned as `true` or `false`.

Data Types: `logical`

See Also

Image Viewer | `imformats` | `imgetfile` | `imsave`

Introduced in R2007b

impyramid

Image pyramid reduction and expansion

Syntax

```
B = impyramid(A,direction)
```

Description

`B = impyramid(A,direction)` computes a Gaussian pyramid reduction or expansion of `A` by one level. `direction` determines whether `impyramid` performs a reduction or an expansion.

Examples

Compute Four-level Multiresolution Pyramid of Image

Read image into the workspace.

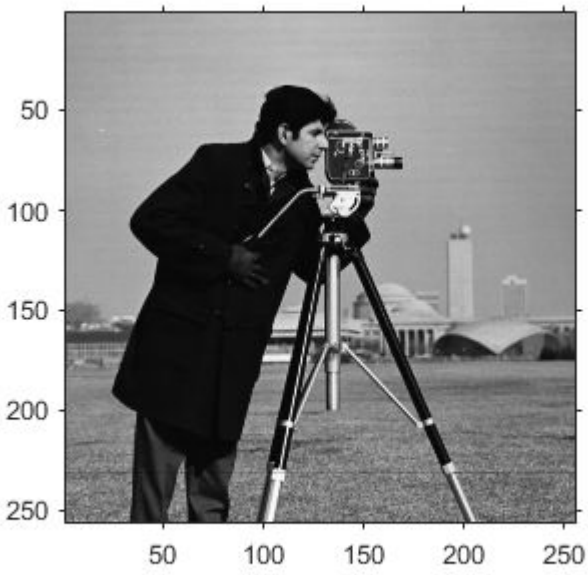
```
I = imread('cameraman.tif');
```

Perform a series of reductions. The first call reduces the original image. The other calls to `impyramid` use the previously reduced image.

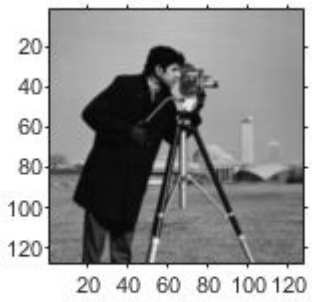
```
I1 = impyramid(I, 'reduce');
I2 = impyramid(I1, 'reduce');
I3 = impyramid(I2, 'reduce');
```

View the original image and the reduced versions.

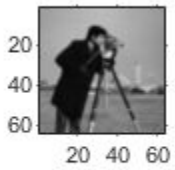
```
figure, imshow(I)
```




```
figure, imshow(I1)
```




```
figure, imshow(I2)
```



```
figure, imshow(I3)
```

Input Arguments

`A` — Image to be reduced or expanded

numeric or logical array

Image to reduced or expanded, specified as a numeric or logical array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`direction` — Reduction or expansion

`'reduce'` | `'expand'`

Reduction or expansion, specified as one of the following values:

Value	Description
`'reduce'`	Return an image, smaller than the original image.
`'expand'`	Return an image that is larger than the original image.

Data Types: `char` | `string`

Output Arguments

`B` — Reduced or expanded image

numeric or logical array

Reduced or expanded image, returned as a numeric or logical array, the same class as `A`.

Algorithms

If `A` is *m*-by-*n* and `direction` is `'reduce'`, the size of `B` is `ceil(M/2)`-by-`ceil(N/2)`. If `direction` is `'expand'`, the size of `B` is `(2*M-1)`-by-`(2*N-1)`.

Reduction and expansion take place only in the first two dimensions. For example, if `A` is 100-by-100-by-3 and `direction` is `'reduce'`, then `B` is 50-by-50-by-3.

`impyramid` uses the kernel specified on page 533 of the Burt and Adelson paper on page 1-1738:

$w = \left[\frac{1}{4} - \frac{a}{2}, \frac{1}{4}, a, \frac{1}{4}, \frac{1}{4} - \frac{a}{2}\right]$, where $a = 0.375$. The parameter a is set to `0.375` so that the equivalent weighting function is close to a Gaussian shape. In addition, the weights can be readily applied using fixed-point arithmetic.

References

[1] Burt and Adelson, "The Laplacian Pyramid as a Compact Image Code," *IEEE Transactions on Communications*, Vol. COM-31, no. 4, April 1983, pp. 532-540.

[2] Burt, "Fast Filter Transforms for Image Processing," *Computer Graphics and Image Processing*, Vol. 16, 1981, pp. 20-51

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `impyramid` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- `direction` must be a compile-time constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- `direction` must be a compile-time constant.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

`imresize`

Introduced in R2007b

imquantize

Quantize image using specified quantization levels and output values

Syntax

```
quant_A = imquantize(A,levels)
quant_A = imquantize( ___ ,values)
[quant_A,index] = imquantize( ___ )
```

Description

`quant_A = imquantize(A,levels)` quantizes image `A` using specified quantization values contained in the `N` element vector `levels`. Output image `quant_A` is the same size as `A` and contains `N + 1` discrete integer values in the range `1` to `N + 1` which are determined by the following criteria:

- If *A(k) ≤ levels(1)*, then *quant_A(k) = 1*.
- If *levels(m-1) < A(k) ≤ levels(m)* , then *quant_A(k) = m*.
- If *A(k) > levels(N)*, then *quant_A(k) = N + 1*.

Note that `imquantize` assigns values to the two implicitly defined end intervals:

- *A(k) ≤ levels(1)*
- *A(k) > levels(N)*

`quant_A = imquantize( ___ ,values)` adds the `N + 1` element vector `values` where `N = length(levels)`. Each of the `N + 1` elements of `values` specify the quantization value for one of the `N + 1` discrete pixel values in `quant_A`.

- If *A(k) ≤ levels(1)*, then *quant_A(k) = values(1)*.
- If *levels(m-1) < A(k) ≤ levels(m)* , then *quant_A(k) = values(m)*.
- If *A(k) > levels(N)*, then *quant_A(k) = values(N + 1)*.

`[quant_A,index] = imquantize( ___ )` returns an array *index* such that:

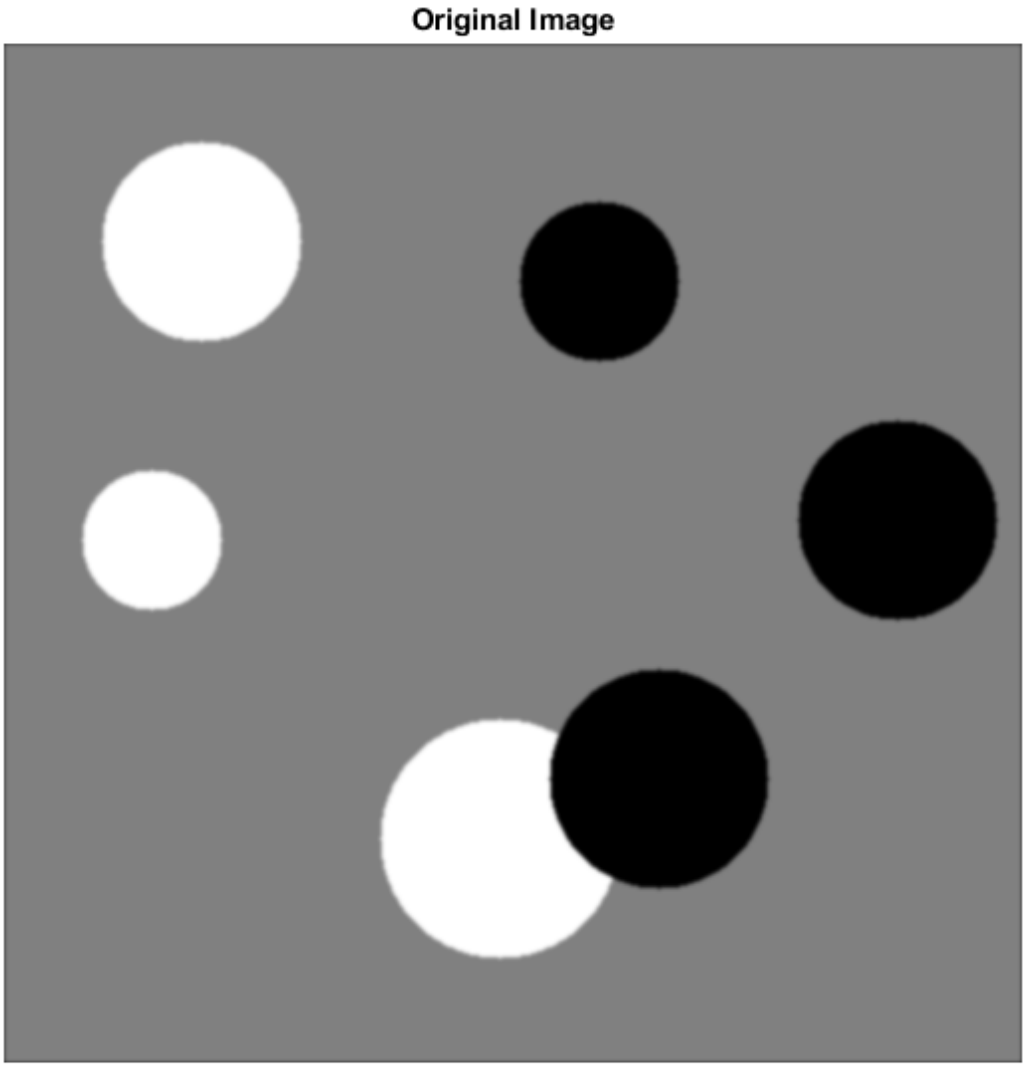
`quant_A = values(index)`

Examples

Segment Image into Three Levels Using Two Thresholds

Read image and display it.

```
I = imread('circlesBrightDark.png');
imshow(I)
axis off
title('Original Image')
```



Calculate two threshold levels.

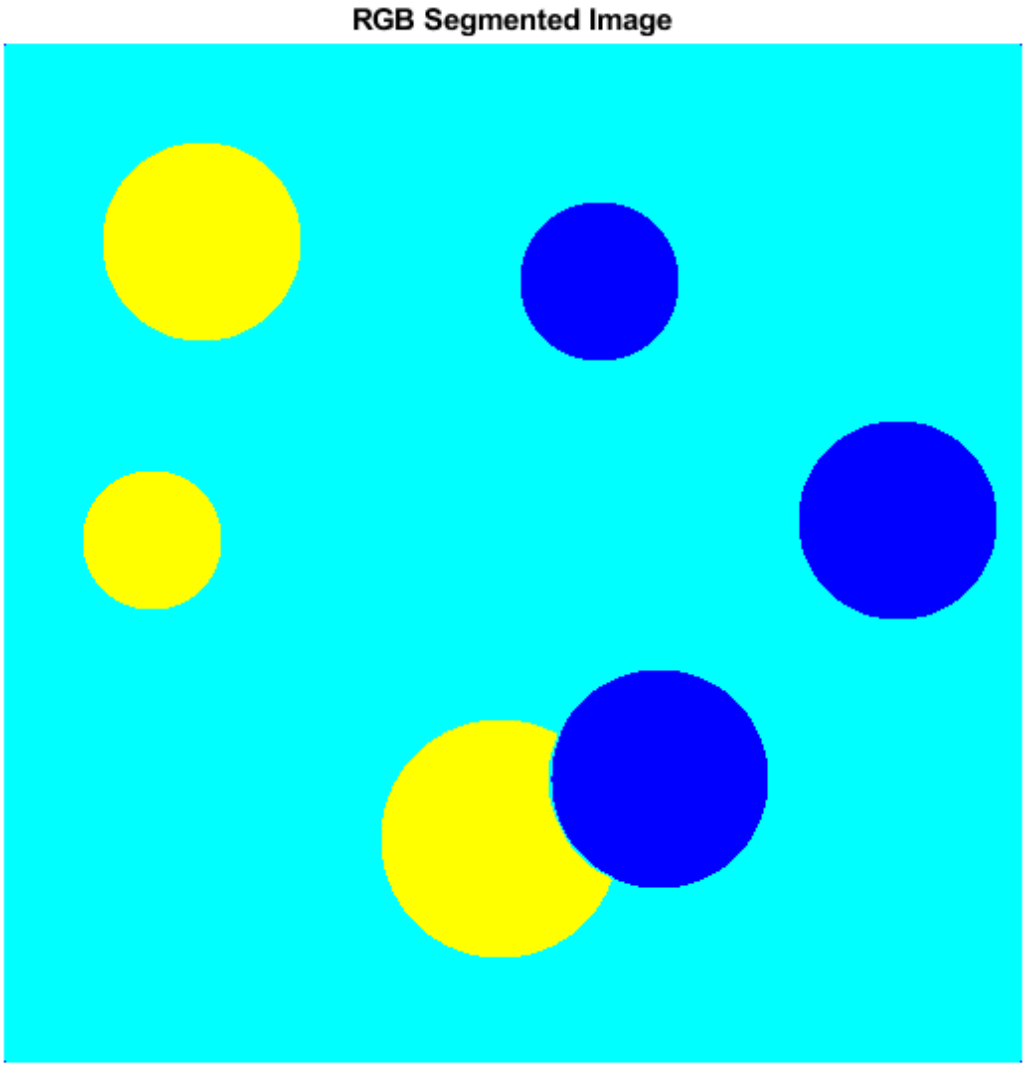
```
thresh = multithresh(I,2);
```

Segment the image into three levels using `imquantize` .

```
seg_I = imquantize(I,thresh);
```

Convert segmented image into color image using `label2rgb` and display it.

```
RGB = label2rgb(seg_I);
figure;
imshow(RGB)
axis off
title('RGB Segmented Image')
```



Compare Thresholding Entire Image Versus Plane-by-Plane Thresholding

Read truecolor (RGB) image and display it.

```
I = imread('peppers.png');
imshow(I)
axis off
title('RGB Image');
```

Generate thresholds for seven levels from the entire RGB image.

```
threshRGB = multithresh(I,7);
```

Generate thresholds for each plane of the RGB image.

```
threshForPlanes = zeros(3,7);

for i = 1:3
    threshForPlanes(i,:) = multithresh(I(:,:,i),7);
end
```

Process the entire image with the set of threshold values computed from entire image.

```
value = [0 threshRGB(2:end) 255];
quantRGB = imquantize(I, threshRGB, value);
```

Process each RGB plane separately using the threshold vector computed from the given plane. Quantize each RGB plane using threshold vector generated for that plane.

```
quantPlane = zeros( size(I) );

for i = 1:3
    value = [0 threshForPlanes(i,2:end) 255];
```

```
    quantPlane(:,:,i) = imquantize(I(:,:,i),threshForPlanes(i,:),value);
end

quantPlane = uint8(quantPlane);
```

Display both posterized images and note the visual differences in the two thresholding schemes.

```
imshowpair(quantRGB,quantPlane,'montage')
axis off
title('Full RGB Image Quantization        Plane-by-Plane Quantization')
```

To compare the results, calculate the number of unique RGB pixel vectors in each output image. Note that the plane-by-plane thresholding scheme yields about 23% more colors than the full RGB image scheme.

```
dim = size( quantRGB );
quantRGBmx3   = reshape(quantRGB,   prod(dim(1:2)), 3);
quantPlanemx3 = reshape(quantPlane, prod(dim(1:2)), 3);

colorsRGB   = unique(quantRGBmx3,   'rows' );
colorsPlane = unique(quantPlanemx3, 'rows' );

disp(['Unique colors in RGB image            : ' int2str(length(colorsRGB))]);
```

```
Unique colors in RGB image            : 188
```

```
disp(['Unique colors in Plane-by-Plane image : ' int2str(length(colorsPlane))]);
```

```
Unique colors in Plane-by-Plane image : 231
```

Threshold Grayscale Image from 256 to 8 Levels

Reduce the number of discrete levels in an image from 256 to 8. This example uses two different methods for assigning values to each of the eight output levels.

Read image and display it.

```
I = imread('coins.png');
imshow(I)
axis off
title('Grayscale Image')
```

Split the image into eight levels by obtaining seven thresholds from the `multithresh` function.

```
thresh = multithresh(I,7);
```

Construct the `valuesMax` vector such that the maximum value in each quantization interval is assigned to the eight levels of the output image.

```
valuesMax = [thresh max(I(:))]
```

```
valuesMax = 1x8 uint8 row vector

    65    88   119   149   169   189   215   255
```

```
[quant8_I_max, index] = imquantize(I,thresh,valuesMax);
```

Similarly, construct the `valuesMin` vector such that the minimum value in each quantization interval is assigned to the eight levels of the output image. Instead of calling `imquantize` again with the vector `valuesMin`, use the output argument index to assign those values to the output image.

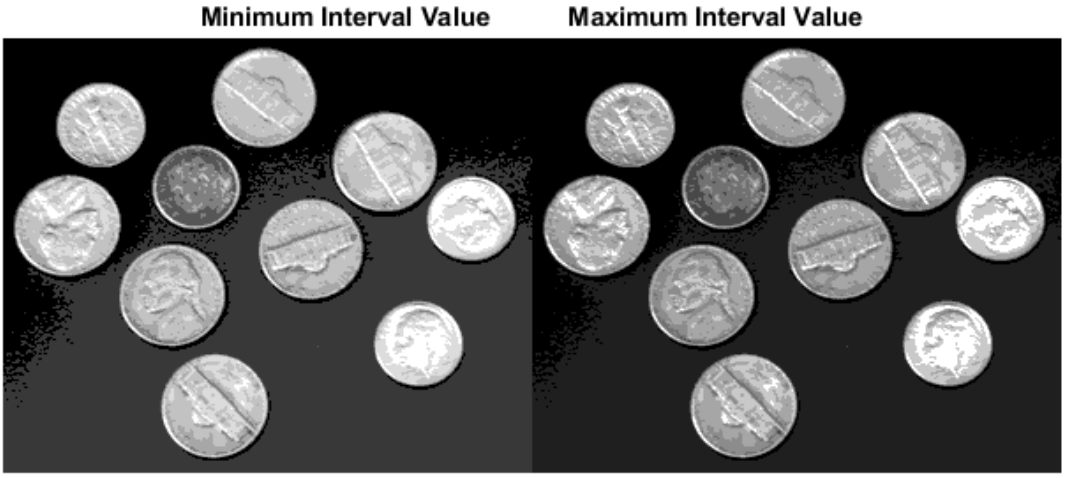
```
valuesMin = [min(I(:)) thresh]
```

```
valuesMin = 1x8 uint8 row vector

    23    65    88   119   149   169   189   215
```

```
quant8_I_min = valuesMin(index);
```

Display both eight-level output images side by side.

```
imshowpair(quant8_I_min,quant8_I_max,'montage')
title('Minimum Interval Value           Maximum Interval Value')
```



Input Arguments

`A` — Input image

image

Input image, specified as a numeric array of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`levels` — Quantization levels

vector

Quantization levels, specified as an `N` element vector. Values of the discrete quantization levels must be in monotonically increasing order.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`values` — Quantization values

vector

Quantization values, specified as an `N+1` element vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`quant_A` — Quantized output image

image

Quantized output image, returned as a numeric array the same size as *A*. If input argument *values* is specified, then *quant_A* is the same data type as *values*. If *values* is not specified, then *quant_A* is of class `double`.

`index` — Mapping array
array

Mapping array, returned as an array the same size as input image *A*. It contains integer indices which access *values* to construct the output image: `quant_A = values(index)`. If input argument *values* is not defined, then `index = quant_A`.

Data Types: `double`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

`imquantize` supports the generation of C code (requires MATLAB Coder). For more information, see "Code Generation for Image Processing".

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Thread-Based Environment
Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see "Run MATLAB Functions in Thread-Based Environment".

See Also
`multithresh` | `label2rgb` | `rgb2ind`

Introduced in R2012b

imreconstruct

Morphological reconstruction

Syntax

```
J = imreconstruct(marker,mask)
J = imreconstruct(marker,mask,conn)
```

Description

`J = imreconstruct(marker,mask)` performs morphological reconstruction of the image `marker` under the image `mask`, and returns the reconstruction in `J`. The elements of `marker` must be less than or equal to the corresponding elements of `mask`. If the values in `marker` are greater than corresponding elements in `mask`, then `imreconstruct` clips the values to the `mask` level before starting the procedure.

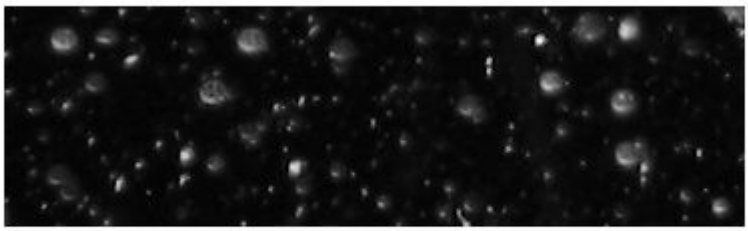
`J = imreconstruct(marker,mask,conn)` performs morphological reconstruction with the specified connectivity, `conn`.

Examples

Perform Opening-by-Reconstruction to Identify High Intensity Objects

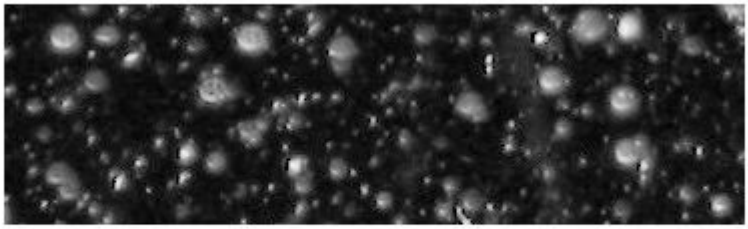
Read and display a grayscale image.

```
I = imread('snowflakes.png');
imshow(I)
```



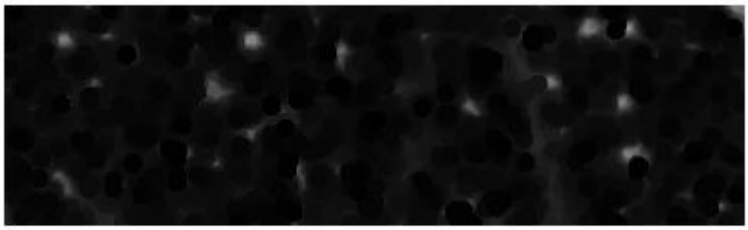
Adjust the contrast of the image to create the mask image and display results.

```
mask = adapthisteq(I);
imshow(mask)
```



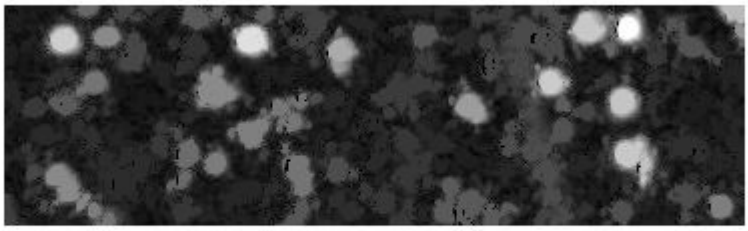
Create a marker image that identifies high-intensity objects in the image using morphological erosion and display results.

```
se = strel('disk',5);
marker = imerode(mask,se);
imshow(marker)
```



Perform morphological opening on the mask image, using the marker image to identify high-intensity objects in the mask. Display the result.

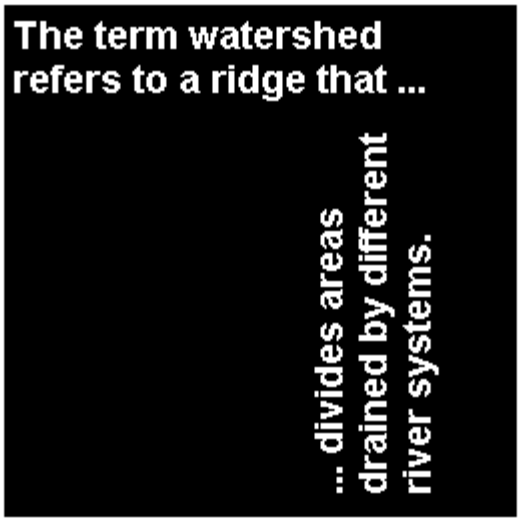
```
obr = imreconstruct(marker,mask);
imshow(obr,[])
```



Use Reconstruction to Segment an Image

Read a logical image into workspace and display it. This is the mask image.

```
mask = imread('text.png');
figure
imshow(mask)
```

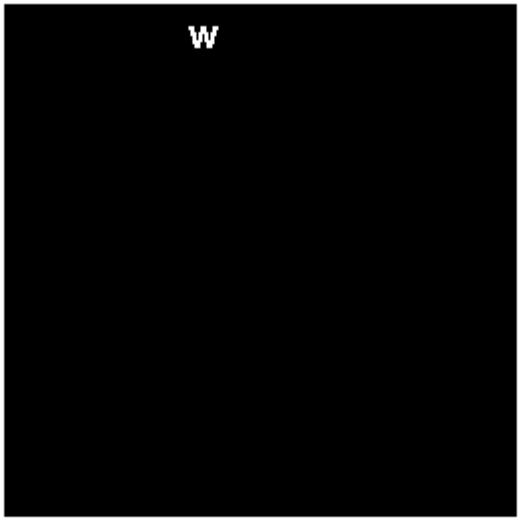


Create a marker image that identifies the object in the image you want to extract through segmentation. For this example, identify the "w" in the word "watershed".

```
marker = false(size(mask));
marker(13,94) = true;
```

Perform segmentation of the mask image using the marker image.

```
im = imreconstruct(marker,mask);
figure
imshow(im)
```



Input Arguments

`marker` — Input image

numeric array | logical array

Input image, specified as a numeric or logical array.

Example: `se = strel('disk',5); marker = imerode(mask,se);`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`mask` — Mask image

numeric array | logical array

Mask image, specified as a numeric or logical array of the same size and data type as `marker`.

Example: `mask = imread('text.png');`

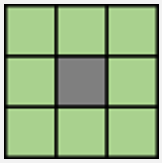
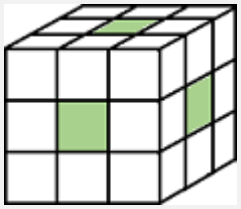
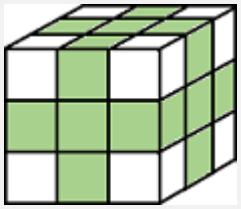
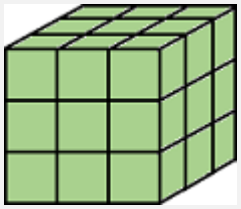
Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`conn` — Pixel connectivity

4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 matrix of `0`s and `1`s

Pixel connectivity, specified as one of the values in this table. The default connectivity is `8` for 2-D images, and `26` for 3-D images.

Value	Meaning
Two-Dimensional Connectivities	

Value	Meaning	
4	Pixels are connected if their edges touch. The neighborhood of a pixel are the adjacent pixels in the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. The neighborhood of a pixel are the adjacent pixels in the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		
6	Pixels are connected if their faces touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down	 Current pixel is shown in gray.
18	Pixels are connected if their faces or edges touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `imreconstruct` uses the default value `conndef(ndims(marker),'maximal')`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Data Types: `double` | `logical`

Output Arguments

`J` — Reconstructed image

numeric array | logical array

Reconstructed image, returned as a numeric or logical array, depending on the input image, that is the same size as the input image.

Tips

- Morphological reconstruction is the algorithmic basis for several other Image Processing Toolbox functions, including `imclearborder`, `imextendedmax`, `imextendedmin`, `imfill`, `imhmax`, `imhmin`, and `imimposemin`.
- **Performance note**: This function may take advantage of hardware optimization for data types `logical`, `uint8`, `uint16`, `single`, and `double` to run faster. Hardware optimization requires `marker` and `mask` to be 2-D images and `conn` to be either `4` or `8`.

Algorithms

`imreconstruct` uses the fast hybrid grayscale reconstruction algorithm described in [1].

References

[1] Vincent, L., "Morphological Grayscale Reconstruction in Image Analysis: Applications and Efficient Algorithms," *IEEE Transactions on Image Processing*, Vol. 2, No. 2, April, 1993, pp. 176-201.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imreconstruct` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imreconstruct` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, the optional third input argument, `conn`, must be a compile-time constant, and can only take the value `4` or `8`.
- `imreconstruct` does not support `uint64` and `int64` data types for code generation.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, the optional third input argument, `conn`, must be a compile-time constant, and can only take the value `4` or `8`.
- `imreconstruct` does not support `uint64` and `int64` data types for code generation.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Input images must be 2-D numeric or logical matrices. `imreconstruct` does not support RGB images and 3-D images on a GPU.
- Only the 2-D connectivities (4 and 8) are supported.

For more information, see “Image Processing on a GPU”.

See Also

`imclearborder` | `imextendedmax` | `imextendedmin` | `imfill` | `imhmax` | `imhmin` | `imimposemin`

Topics

“Morphological Reconstruction”

Introduced before R2006a

imreducehaze

Reduce atmospheric haze

Syntax

```
[J,T,L] = imreducehaze(I)
[___] = imreducehaze(I,amount)
[___] = imreducehaze(___,Name,Value)
```

Description

`[J,T,L] = imreducehaze(I)` reduces atmospheric haze in color or grayscale image `I`. The function returns the dehazed image `J`, an estimate `T` of the haze thickness at each pixel, and the estimated atmospheric light `L`.

`[___] = imreducehaze(I,amount)` additionally specifies the amount of haze to remove.

`[___] = imreducehaze(___,Name,Value)` changes the behavior of the dehazing algorithm using name-value arguments.

Examples

Reduce Haze Using Default Parameters

Read a hazy image into the workspace.

```
A = imread('foggysf1.jpg');
```

Reduce the haze and display the result next to the original image in a montage.

```
B = imreducehaze(A);
montage({A,B})
title("Hazy Image (Left) vs. Reduced Haze Image (Right)")
```

Reduce Haze Using approxdcp Contrast Stretching

Read a hazy image into the workspace.

```
A = imread('foggysf2.jpg');
```

Reduce 90% of the haze using the `approxdcp` method.

```
B = imreducehaze(A,0.9,'method','approxdcp');
```

Display in a montage the original hazy image and the image with reduced haze.

```
montage({A,B})
```

Estimate Haze Thickness and Image Depth

Read and display a hazy image.

```
A = imread('foggyroad.jpg');
imshow(A)
title('Hazy Image')
```

Reduce haze in the image using default parameter values. Return an estimate of the haze thickness.

```
[~,T] = imreducehaze(A);
```

Display the haze thickness measurement.

```
imshow(T)
title('Haze Thickness')
```

The haze thickness `T` provides a rough approximation of the depth `D` of the scene, defined up to an unknown multiplication factor. Add `eps` to avoid `log(0)`.

```
D = -log(1-T+eps);
```

Display the estimated depth in false color.

```
imshow(D,[])
title('Depth Estimate')
colormap hot
```

Input Arguments

`I` — Hazy image

RGB image | grayscale image

Hazy image, specified as an RGB or grayscale image.

Data Types: `single` | `double` | `uint8` | `uint16`

`amount` — Amount of haze to remove

`1` (default) | number in the range [0, 1]

Amount of haze to remove, specified as a number in the range [0, 1]. When the value is `1`, `imreducehaze` reduces the maximum amount of haze. When the value is `0`, `imreducehaze` does not reduce haze and the input image is unchanged. Larger values can cause more severe color distortion.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `B = imreducehaze(A,0.9,method="approxdcp");` reduces haze using the approximate dark channel prior method.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `B = imreducehaze(A,0.9,"method","approxdcp");`

`Method` — Technique used to reduce haze

`"simpledcp"` (default) | `"approxdcp"`

Technique used to reduce haze, specified as one of these values:

- `"simpledcp"` — Simple dark channel prior method [2]. This technique uses a per-pixel dark channel to estimate haze and quadtree decomposition to estimate the atmospheric light.
- `"approxdcp"` — Approximate dark channel prior method [1]. This technique uses both per-pixel and spatial blocks when computing the dark channel and does not use quadtree decomposition.

For more information, see Algorithms on page 1-1760.

Data Types: `char` | `string`

`AtmosphericLight` — Maximum value to be treated as haze

1-by-3 numeric vector | numeric scalar

Maximum value to be treated as haze, specified as a 1-by-3 numeric vector for RGB images or a numeric scalar for grayscale images. Values must be in the range [0, 1]. Atmospheric light values greater than 0.5 tend to give better results.

If you do not specify `AtmosphericLight`, then the `imreduzehaze` function estimates a value depending on the value of `Method`.

Data Types: `double`

`ContrastEnhancement` — Contrast enhancement technique

`"global"` (default) | `"boost"` | `"none"`

Contrast enhancement technique, specified as `"global"`, `"boost"`, or `"none"`.

Data Types: `char` | `string`

`BoostAmount` — Amount of per-pixel gain

`0.1` (default) | number in the range (0, 1]

Amount of per-pixel gain to apply as postprocessing, specified as a positive number in the range (0, 1]. This argument is only supported if `ContrastEnhancement` is specified as `"boost"`.

Data Types: `double`

Output Arguments

J — Dehazed image

numeric array

Dehazed image, returned as numeric array of the same size as the input hazy image `I`.

T — Haze thickness

numeric array

Haze thickness estimated at each pixel, returned as a numeric array.

L — Estimated atmospheric light

numeric array

Estimated atmospheric light, returned as a numeric array. `L` represents the value of the brightest nonspecular haze.

Algorithms

The model to describe a hazy image `I` is

```
I(x) = J(x)T(x) + L(1-T(x))
```

`I` is the observed intensity, `J` is the scene radiance, `L` is atmospheric light, and `T` is a transmission map describing the portion of light that reaches the camera.

Dehazing algorithms recover the scene radiance (dehazed image) `J` from an estimation of the transmission map and atmospheric light according to:

```
J(x) = (I(x)-A)/(max(t(x),t0)) + A
```

`imreducehaze` uses two different dehazing algorithms, `simpledcp` and `approxdcp`. These methods both rely on a dark channel prior, which is based on the observation that nonhazy images of outdoor scenes usually contain some pixels that have low signal in one or more color channels. The methods differ in how they estimate the dark channel prior and atmospheric light.

The dehazing algorithms in `imreducehaze` follow five steps:

1. Estimate the atmospheric light `L` using a dark channel prior.
2. Estimate the transmission map `T`.
3. Refine the estimated transmission map.
4. Restore the image.
5. Perform optional contrast enhancement.

References

[1] He, Kaiming. "Single Image Haze Removal Using Dark Channel Prior." *Thesis, The Chinese University of Hong Kong*. 2011.

[2] Dubok, et al. "Single Image Dehazing with Image Entropy and Information Fidelity." *ICIP*. 2014, pp. 4037–4041.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imreducehaze` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- The value of string input arguments must be compile time constants.

See Also

`imadjust` | `stretchlim`

Topics

“Low-Light Image Enhancement”
“Quadtree Decomposition”

Introduced in R2017b

imref2d

Reference 2-D image to world coordinates

Description

An `imref2d` object stores the relationship between the intrinsic coordinates anchored to the rows and columns of a 2-D image and the spatial location of the same row and column locations in a world coordinate system.

The image is sampled regularly in the planar world-*x* and world-*y* coordinate system such that intrinsic-*x* values align with world-*x* values, and intrinsic-*y* values align with world-*y* values. The resolution in each dimension can be different.

Creation

You can create an `imref2d` object in these ways.

- `affineOutputView` — Store the spatial extent of an image that is warped by a 2-D affine geometric transformation.
- The `imref2d` function described here

Syntax

```
R = imref2d
R = imref2d(imageSize)
R = imref2d(imageSize,pixelExtentInWorldX,pixelExtentInWorldY)
R = imref2d(imageSize,xWorldLimits,yWorldLimits)
```

Description

`R = imref2d` creates an `imref2d` object with default property settings.

`R = imref2d(imageSize)` sets the optional `ImageSize on page 1-0` property.

`R = imref2d(imageSize,pixelExtentInWorldX,pixelExtentInWorldY)` sets the optional `ImageSize on page 1-0`, `PixelExtentInWorldX on page 1-0`, and `PixelExtentInWorldY on page 1-0` properties.

`R = imref2d(imageSize,xWorldLimits,yWorldLimits)` sets the optional `ImageSize on page 1-0`, `XWorldLimits on page 1-0`, and `YWorldLimits on page 1-0` properties.

Properties

`ImageExtentInWorldX` — Span of image in the *x*-dimension in the world coordinate system
numeric scalar

Span of image in the *x*-dimension in the world coordinate system, specified as a numeric scalar. The `imref2d` object sets this value as `PixelExtentInX * ImageSize(2)`.

Data Types: `double`

`ImageExtentInWorldY` — Span of image in the *y*-dimension in the world coordinate system
numeric scalar

Span of image in the *y*-dimension in the world coordinate system, specified as a numeric scalar. The `imref2d` object sets this value as `PixelExtentInY * ImageSize(1)`.

Data Types: `double`

`ImageSize` — Number of elements in each spatial dimension
2-element positive row vector

Number of elements in each spatial dimension, specified as a 2-element positive row vector. `ImageSize` is the same form as that returned by the `size` function.

Data Types: `double`

`PixelExtentInWorldX` — Size of a single pixel in the *x*-dimension measured in the world coordinate system
positive number

Size of a single pixel in the *x*-dimension measured in the world coordinate system, specified as a positive number.

Data Types: `double`

`PixelExtentInWorldY` — Size of a single pixel in the *y*-dimension measured in the world coordinate system
positive number

Size of a single pixel in the *y*-dimension measured in the world coordinate system, specified as a positive number.

Data Types: `double`

`XWorldLimits` — Limits of image in world *x*-dimension
2-element numeric row vector

Limits of image in world *x*-dimension, specified as a 2-element row numeric vector `[xMin xMax]`.

Data Types: `double`

`YWorldLimits` — Limits of image in world *y*-dimension
2-element numeric row vector

Limits of image in world *y*-dimension, specified as a 2-element numeric row vector `[yMin yMax]`.

Data Types: `double`

`XIntrinsicLimits` — Limits of image in intrinsic units in the *x*-dimension
2-element row vector

Limits of image in intrinsic units in the *x*-dimension, specified as a 2-element row vector `[xMin xMax]`. For an *m*-by-*n* image (or an *m*-by-*n*-by-*p* image), `XIntrinsicLimits` equals `[0.5, n+0.5]`.

Data Types: `double`

`YIntrinsicLimits` — Limits of image in intrinsic units in the *y*-dimension
2-element row vector

Limits of image in intrinsic units in the *y*-dimension, specified as a 2-element row vector `[yMin yMax]`. For an *m*-by-*n* image (or an *m*-by-*n*-by-*p* image), `YIntrinsicLimits` equals `[0.5, m+0.5]`.

Data Types: `double`

Object Functions

contains	Determine if image contains points in world coordinate system
intrinsicToWorld	Convert from intrinsic to world coordinates
sizesMatch	Determine if object and image are size-compatible
worldToIntrinsic	Convert from world to intrinsic coordinates
worldToSubscript	Convert world coordinates to row and column subscripts

Examples

Create 2-D Spatial Referencing Object Knowing Image Size and World Limits

Read a 2-D grayscale image into the workspace.

```
A = imread('pout.tif');
```

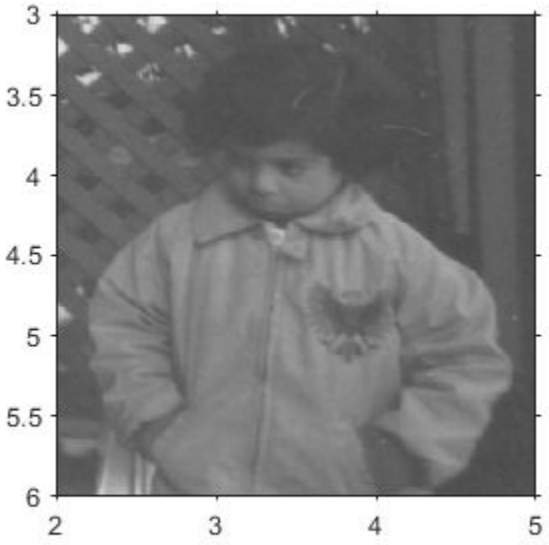
Create an `imref2d` object, specifying the size and world limits of the image associated with the object.

```
xWorldLimits = [2 5];
yWorldLimits = [3 6];
RA = imref2d(size(A),xWorldLimits,yWorldLimits)
```

```
RA =
  imref2d with properties:

          XWorldLimits: [2 5]
          YWorldLimits: [3 6]
             ImageSize: [291 240]
    PixelExtentInWorldX: 0.0125
    PixelExtentInWorldY: 0.0103
    ImageExtentInWorldX: 3
    ImageExtentInWorldY: 3
       XIntrinsicLimits: [0.5000 240.5000]
       YIntrinsicLimits: [0.5000 291.5000]
```

Display the image, specifying the spatial referencing object. The axes coordinates reflect the world coordinates.

```
figure
imshow(A,RA);
```



Create 2-D Spatial Referencing Object Knowing Image Size and Resolution

Read a 2-D grayscale image into the workspace.

```
m = dicominfo('knee1.dcm');
A = dicomread(m);
```

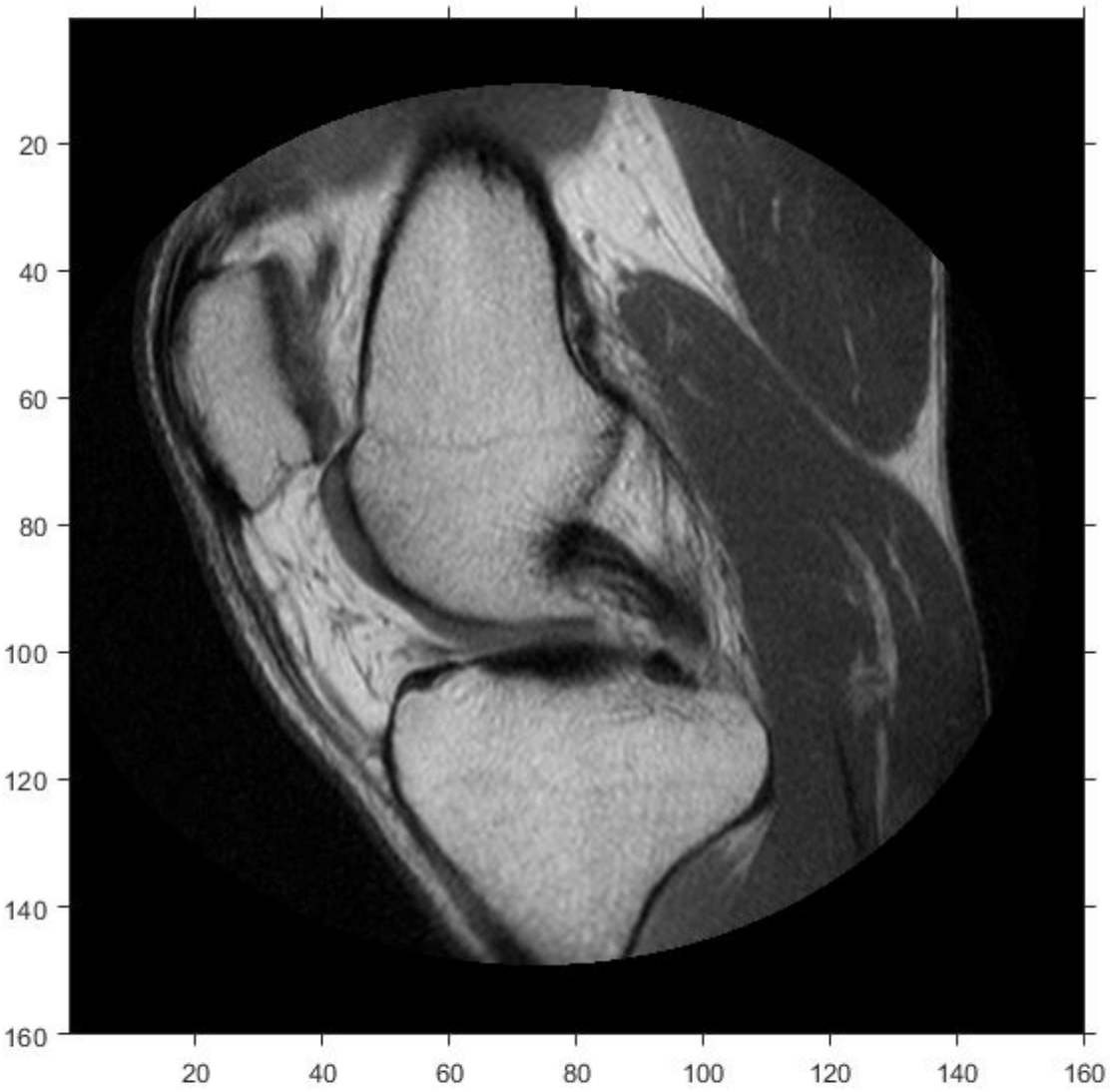
Create an `imref2d` object, specifying the size and the resolution of the pixels. The DICOM file contains a metadata field `PixelSpacing` that specifies the image resolution in each dimension in millimeters per pixel.

```
RA = imref2d(size(A),m.PixelSpacing(2),m.PixelSpacing(1))
```

```
RA =
  imref2d with properties:

          XWorldLimits: [0.1562 160.1562]
          YWorldLimits: [0.1562 160.1562]
             ImageSize: [512 512]
    PixelExtentInWorldX: 0.3125
    PixelExtentInWorldY: 0.3125
    ImageExtentInWorldX: 160
    ImageExtentInWorldY: 160
       XIntrinsicLimits: [0.5000 512.5000]
       YIntrinsicLimits: [0.5000 512.5000]
```

Display the image, specifying the spatial referencing object. The axes coordinates reflect the world coordinates.

```
figure
imshow(A,RA,'DisplayRange',[0 512])
```



Compare the width of the image in world coordinates and intrinsic coordinates. This image width in intrinsic coordinates, with units of pixels, is:

```
RA.ImageSize(1)
```

```
ans = 512
```

The image width in world coordinates, with units of millimeters, is:

```
RA.ImageExtentInWorldX
```

```
ans = 160
```

More About

Intrinsic Coordinate System

The intrinsic coordinate values (x,y) of the center point of any pixel are identical to the values of the column and row subscripts for that pixel. For example, the center point of the pixel in row 5, column 3 has intrinsic coordinates x = 3.0, y = 5.0.

The order of coordinate specification (3.0,5.0) is reversed in intrinsic coordinates relative to pixel subscripts (5,3). Intrinsic coordinates are defined on a continuous plane, while the subscript locations are discrete locations with integer values.

Tips

- You can create an `imref2d` object for an RGB image. If you create the object specifying the `ImageSize on page 1-0` property as a three-element vector (such as that returned by the `size` function), only the first two elements are used to set `ImageSize`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imref2d` supports the generation of C code (requires MATLAB Coder). For more information, see "Code Generation for Image Processing".
- When generating code, you can only specify singular objects—arrays of objects are not supported.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see "Run MATLAB Functions in Thread-Based Environment".

See Also

`imref3d` | `imwarp` | `imshow`

Topics

"Specify Fill Values in Geometric Transformation Output"

Introduced in R2013a

imref3d

Reference 3-D image to world coordinates

Description

An `imref3d` object stores the relationship between the intrinsic coordinates anchored to the columns, rows, and planes of a 3-D image and the spatial location of the same column, row, and plane locations in a world coordinate system.

The image is sampled regularly in the planar world-*x*, world-*y*, and world-*z* coordinates of the coordinate system such that intrinsic-*x*, -*y* and -*z* values align with world-*x*, -*y*, and -*z* values, respectively. The resolution in each dimension can be different.

Creation

You can create an `imref3d` object in these ways.

- `affineOutputView` — Store the spatial extent of an image that is warped by a 3-D affine geometric transformation.
- The `imref2d` function described here

Syntax

```
R = imref3d
R = imref3d(imageSize)
R =
imref3d(imageSize,pixelExtentInWorldX,pixelExtentInWorldY,pixelExtentInWorldZ
)
R = imref3d(imageSize,xWorldLimits,yWorldLimits,zWorldLimits)
```

Description

`R = imref3d` creates an `imref3d` object with default property settings.

`R = imref3d(imageSize)` sets the optional `ImageSize on page 1-0` property.

`R =
imref3d(imageSize,pixelExtentInWorldX,pixelExtentInWorldY,pixelExtentInWorldZ
)` sets the optional `ImageSize on page 1-0` , `PixelExtentInWorldX on page 1-0` , `PixelExtentInWorldY on page 1-0` , and `PixelExtentInWorldZ on page 1-0` properties.

`R = imref3d(imageSize,xWorldLimits,yWorldLimits,zWorldLimits)` sets the optional `ImageSize on page 1-0` , `XWorldLimits on page 1-0` , `YWorldLimits on page 1-0` , and `ZWorldLimits on page 1-0` properties.

Properties

`ImageExtentInWorldX` — Span of image in the *x*-dimension in the world coordinate system
numeric scalar

Span of image in the *x*-dimension in the world coordinate system, specified as a numeric scalar. The `imref3d` object calculates this value as `PixelExtentInX * ImageSize(2)`.

Data Types: `double`

`ImageExtentInWorldY` — Span of image in the *y*-dimension in the world coordinate system
numeric scalar

Span of image in the *y*-dimension in the world coordinate system, specified as a numeric scalar. The `imref3d` object calculates this value as `PixelExtentInY * ImageSize(1)`.

Data Types: `double`

`ImageExtentInWorldZ` — Span of image in the *z*-dimension in the world coordinate system
numeric scalar

Span of image in the *z*-dimension in the world coordinate system, specified as a numeric scalar. The `imref3d` object calculates this value as `PixelExtentInZ * ImageSize(3)`.

Data Types: `double`

`ImageSize` — Number of elements in each spatial dimension
3-element positive row vector

Number of elements in each spatial dimension, specified as a 3-element positive row vector. `ImageSize` is the same form as that returned by the `size` function.

Data Types: `double`

`PixelExtentInWorldX` — Size of a single pixel in the *x*-dimension measured in the world coordinate system
positive number

Size of a single pixel in the *x*-dimension measured in the world coordinate system, specified as a positive number.

Data Types: `double`

`PixelExtentInWorldY` — Size of a single pixel in the *y*-dimension measured in the world coordinate system
positive number

Size of a single pixel in the *y*-dimension measured in the world coordinate system, specified as a positive number.

Data Types: `double`

`PixelExtentInWorldZ` — Size of a single pixel in the *z*-dimension measured in the world coordinate system
positive number

Size of a single pixel in the *z*-dimension measured in the world coordinate system, specified as a positive number.

Data Types: `double`

`XWorldLimits` — Limits of image in world *x*-dimension
2-element numeric row vector

Limits of image in world *x*, specified as a 2-element row vector, `[xMin xMax]`.

Data Types: `double`

`YWorldLimits` — Limits of image in world *y*-dimension
2-element numeric row vector

Limits of image in world *y*, specified as a 2-element row vector, `[yMin yMax]`.

Data Types: `double`

`ZWorldLimits` — Limits of image in world *z*-dimension
2-element numeric row vector

Limits of image in world *z*, specified as a 2-element row vector, `[zMin zMax]`.

Data Types: `double`

`XIntrinsicLimits` — Limits of image in intrinsic units in the *x*-dimension
2-element row vector

Limits of image in intrinsic units in the *x*-dimension, specified as a 2-element row vector `[xMin xMax]`. For an *m*-by-*n*-by-*p* image, it equals `[0.5, n+0.5]`.

Data Types: `double`

`YIntrinsicLimits` — Limits of image in intrinsic units in the *y*-dimension
2-element row vector

Limits of image in intrinsic units in the *y*-dimension, specified as a 2-element row vector `[yMin yMax]`. For an *m*-by-*n*-by-*p* image, it equals `[0.5, m+0.5]`.

Data Types: `double`

`ZIntrinsicLimits` — Limits of image in intrinsic units in the *z*-dimension
2-element row vector

Limits of image in intrinsic units in the *z*-dimension, specified as a 2-element row vector `[zMin zMax]`. For an *m*-by-*n*-by-*p* image, it equals `[0.5, p+0.5]`.

Data Types: `double`

Object Functions

contains	Determine if image contains points in world coordinate system
intrinsicToWorld	Convert from intrinsic to world coordinates
sizesMatch	Determine if object and image are size-compatible
worldToIntrinsic	Convert from world to intrinsic coordinates
worldToSubscript	Convert world coordinates to row and column subscripts

Examples

Create `imref3d` Object Knowing Image Size and Resolution in Each Dimension

Read image.

```
m = analyze75info('brainMRI.hdr');
A = analyze75read(m);
```

Create an `imref3d` object associated with the image, specifying the size of the pixels. The `PixelDimensions` field of the metadata of the file specifies the resolution in each dimension in millimeters/pixel.

```
RA = imref3d(size(A),m.PixelDimensions(2),m.PixelDimensions(1),m.PixelDimensions(3));
```

```
RA =

  imref3d with properties:

          XWorldLimits: [0.5000 128.5000]
          YWorldLimits: [0.5000 128.5000]
          ZWorldLimits: [0.5000 27.5000]
             ImageSize: [128 128 27]
    PixelExtentInWorldX: 1
    PixelExtentInWorldY: 1
    PixelExtentInWorldZ: 1
    ImageExtentInWorldX: 128
    ImageExtentInWorldY: 128
    ImageExtentInWorldZ: 27
       XIntrinsicLimits: [0.5000 128.5000]
       YIntrinsicLimits: [0.5000 128.5000]
       ZIntrinsicLimits: [0.5000 27.5000]
```

Examine the extent of the image in each dimension in millimeters.

```
RA.ImageExtentInWorldX
RA.ImageExtentInWorldY
RA.ImageExtentInWorldZ
```

```
ans =

   128

ans =

   128

ans =

    27
```

More About

Intrinsic Coordinate System

The intrinsic coordinate values (x,y,z) of the center point of any pixel are identical to the values of the column, row, and plane subscripts for that pixel. For example, the center point of the pixel in row 5, column 3, plane 4 has intrinsic coordinates $x = 3.0$, $y = 5.0$, $z = 4.0$.

The order of the coordinate specification (3.0,5.0,4.0) is reversed in intrinsic coordinates relative to pixel subscripts (5,3,4). Intrinsic coordinates are defined on a continuous plane, while the subscript locations are discrete locations with integer values.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imref3d` supports the generation of C code (requires MATLAB Coder). For more information, see "Code Generation for Image Processing".
- When generating code, you can only specify singular objects—arrays of objects are not supported.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see "Run MATLAB Functions in Thread-Based Environment".

See Also

`imref2d` | `imregister`

Introduced in R2013a

imregionalmax

Regional maxima

Syntax

```
BW = imregionalmax(I)
BW = imregionalmax(I,conn)
```

Description

`BW = imregionalmax(I)` returns the binary image `BW` that identifies the regional maxima in grayscale image `I`. Regional maxima are connected components of pixels with a constant intensity value, surrounded by pixels with a lower value.

`BW = imregionalmax(I,conn)` specifies the pixel connectivity, `conn`.

Examples

Find Regional Maxima in Simple Sample Image

Create a simple sample image with several regional maxima.

```
A = 10*ones(10,10);
A(2:4,2:4) = 22;
A(6:8,6:8) = 33;
A(2,7) = 44;
A(3,8) = 45;
A(4,9) = 44
```

```
A = 10×10

    10    10    10    10    10    10    10    10    10    10
    10    22    22    22    10    10    44    10    10    10
    10    22    22    22    10    10    10    45    10    10
    10    22    22    22    10    10    10    10    44    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    33    33    33    10    10
    10    10    10    10    10    33    33    33    10    10
    10    10    10    10    10    33    33    33    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
```

Find the regional maxima. Note that the result includes the regional maxima at (3,8).

```
regmax = imregionalmax(A)
```

```
regmax = 10x10 logical array

   0   0   0   0   0   0   0   0   0   0
   0   1   1   1   0   0   0   0   0   0
```

```
0   1   1   1   0   0   0   1   0   0
0   1   1   1   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   1   1   1   0   0
0   0   0   0   0   1   1   1   0   0
0   0   0   0   0   1   1   1   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
```

Input Arguments

`I` — Grayscale image

numeric array

Grayscale image, specified as a numeric array of any dimension.

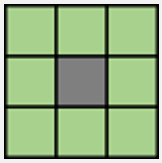
Example: `I = imread('cameraman.tif');`

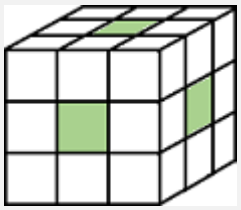
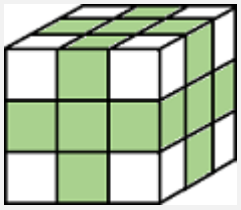
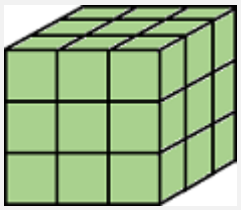
Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`conn` — Pixel connectivity

4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 matrix of `0`s and `1`s

Pixel connectivity, specified as one of the values in this table. The default connectivity is `8` for 2-D images, and `26` for 3-D images.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. The neighborhood of a pixel are the adjacent pixels in the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. The neighborhood of a pixel are the adjacent pixels in the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		

Value	Meaning	
6	Pixels are connected if their faces touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down	 Current pixel is shown in gray.
18	Pixels are connected if their faces or edges touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `imregionalmax` uses the default value `conndef(ndims(I),'maximal')`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`BW` — Locations of regional maxima

logical array

Locations of regional maxima, returned as a logical array of the same size as `I`. Pixels with the value `1` indicate regional maxima; all other pixels are set to `0`.

Data Types: `logical`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imregionalmax` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imregionalmax` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, the optional second input argument, `conn`, must be a compile-time constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Inputs must be 2-D, supporting only the 2-D connectivities (4 and 8).

For more information, see “Image Processing on a GPU”.

See Also

`conndef` | `imextendedmax` | `imhmax` | `imreconstruct` | `imregionalmin`

Introduced before R2006a

imregionalmin

Regional minima

Syntax

```
BW = imregionalmin(I)
BW = imregionalmin(I,conn)
```

Description

`BW = imregionalmin(I)` returns the binary image `BW` that identifies the regional minima in grayscale image `I`. Regional minima are connected components of pixels with a constant intensity value, surrounded by pixels with a higher value.

`BW = imregionalmin(I,conn)` specifies the desired connectivity, `conn`.

Examples

Find Regional Minima in Simple Sample Image

Create a simple sample array with several regional minima.

```
A = 10*ones(10,10);
A(2:4,2:4) = 3;
A(6:8,6:8) = 8
```

```
A = 10×10

    10    10    10    10    10    10    10    10    10    10
    10     3     3     3    10    10    10    10    10    10
    10     3     3     3    10    10    10    10    10    10
    10     3     3     3    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10     8     8     8    10    10
    10    10    10    10    10     8     8     8    10    10
    10    10    10    10    10     8     8     8    10    10
    10    10    10    10    10    10    10    10    10    10
    10    10    10    10    10    10    10    10    10    10
```

Calculate the regional minima. The function returns a binary image, the same size as the input image, in which pixels with the value `1` represent the regional minima. `imregionalmin` sets all other pixels in to `0`.

```
regmin = imregionalmin(A)
```

```
regmin = 10x10 logical array

   0   0   0   0   0   0   0   0   0   0
   0   1   1   1   0   0   0   0   0   0
   0   1   1   1   0   0   0   0   0   0
```

```
0   1   1   1   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   1   1   1   0   0
0   0   0   0   0   1   1   1   0   0
0   0   0   0   0   1   1   1   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
```

Input Arguments

`I` — Grayscale image

numeric array

Grayscale image, specified as a numeric array of any dimension.

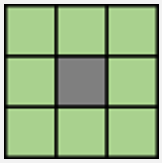
Example: `I = imread('cameraman.tif');`

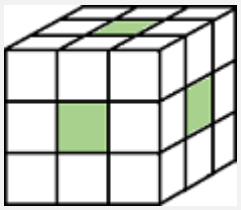
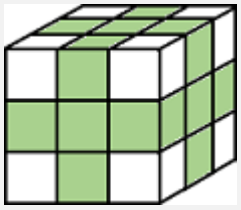
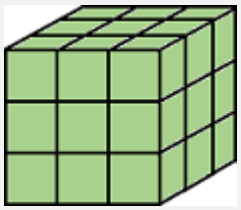
Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`conn` — Pixel connectivity

4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 matrix of `0`s and `1`s

Pixel connectivity, specified as one of the values in this table. The default connectivity is `8` for 2-D images, and `26` for 3-D images.

Value	Meaning	
Two-Dimensional Connectivities		
4	Pixels are connected if their edges touch. The neighborhood of a pixel are the adjacent pixels in the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. The neighborhood of a pixel are the adjacent pixels in the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		

Value	Meaning	
6	Pixels are connected if their faces touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down	 Current pixel is shown in gray.
18	Pixels are connected if their faces or edges touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `imregionalmin` uses the default value `conndef(ndims(I),'maximal')`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`BW` — Locations of regional minima

logical array

Locations of regional minima, returned as a logical array of the same size as `I`. Pixels with the value `1` indicate regional maxima; all other pixels are set to `0`.

Data Types: `logical`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imregionalmin` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imregionalmin` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, the optional second input argument, `conn`, must be a compile-time constant.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Inputs must be 2-D, supporting only the 2-D connectivities (4 and 8).

For more information, see “Image Processing on a GPU”.

See Also

`conndef` | `imextendedmin` | `imhmin` | `imimposemin` | `imreconstruct` | `imregionalmax`

Introduced before R2006a

imregconfig

Configurations for intensity-based registration

Syntax

```
[optimizer,metric] = imregconfig(modality)
```

Description

`[optimizer,metric] = imregconfig(modality)` creates `optimizer` and `metric` configurations that you pass to `imregister` to perform intensity-based image registration, where `modality` specifies the image capture modality.

Examples

Create Optimizer and Metric to Register Monomodal Images

Load the images into the workspace and display them. These images are monomodal because they have similar brightness and contrast.

```
fixed  = imread('pout.tif');
moving = imrotate(fixed, 5, 'bilinear', 'crop');
imshowpair(fixed, moving,'Scaling','joint')
```

Create the optimizer and metric, setting the modality to `'monomodal'`.

```
[optimizer, metric]  = imregconfig('monomodal')

optimizer =
  registration.optimizer.RegularStepGradientDescent

  Properties:
    GradientMagnitudeTolerance: 1.000000e-04
              MinimumStepLength: 1.000000e-05
              MaximumStepLength: 6.250000e-02
              MaximumIterations: 100
               RelaxationFactor: 5.000000e-01

metric =
  registration.metric.MeanSquares

  This class has no properties.
```

Pass the optimizer and metric to `imregister` to perform the registration.

```
movingRegistered = imregister(moving,fixed,'rigid',optimizer, metric);
```

View the registered images

```
figure
imshowpair(fixed, movingRegistered,'Scaling','joint')
```

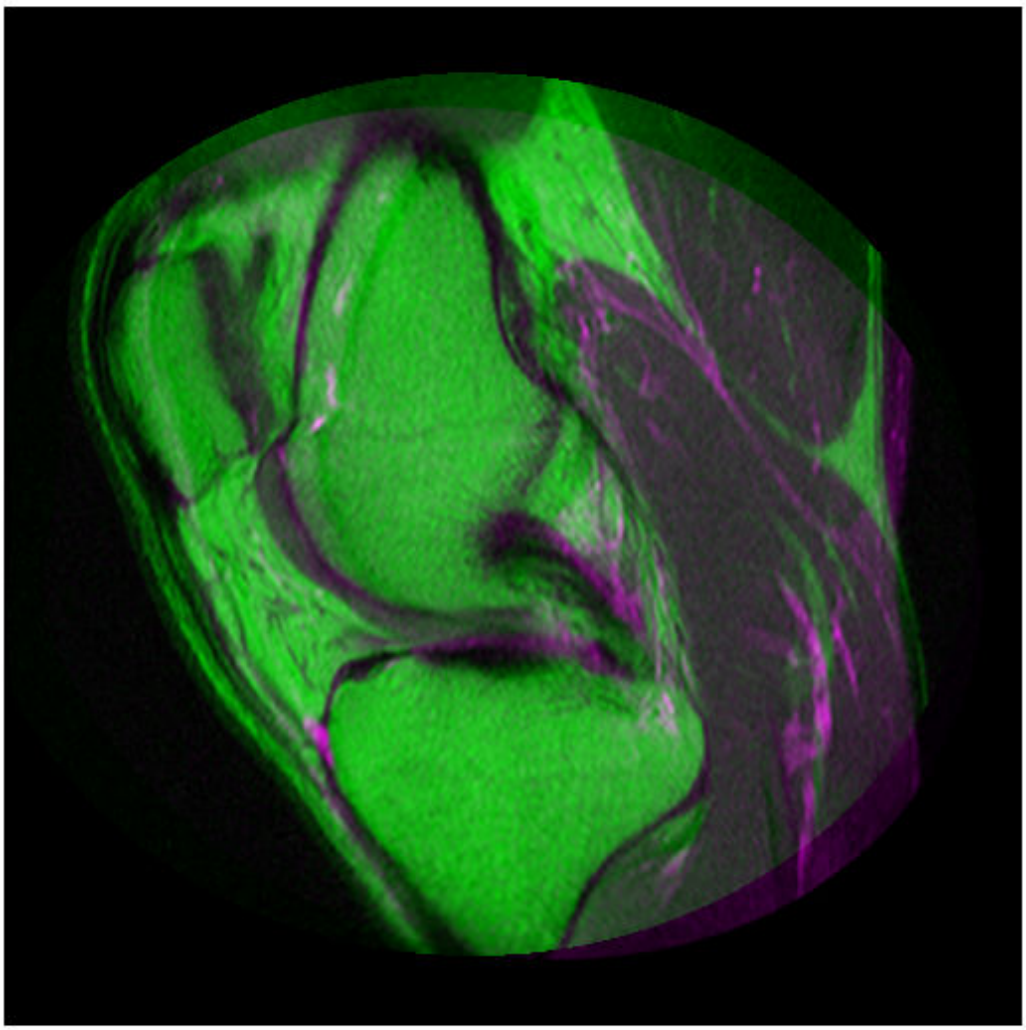
Register Multimodal MRI Images with Optimizer

Read two images. This example uses two magnetic resonance (MRI) images of a knee. The fixed image is a spin echo image, while the moving image is a spin echo image with inversion recovery. The two sagittal slices were acquired at the same time but are slightly out of alignment.

```
fixed = dicomread('knee1.dcm');
moving = dicomread('knee2.dcm');
```

View the misaligned images.

```
imshowpair(fixed, moving,'Scaling','joint')
```



Create the optimizer and metric, setting the modality to `'multimodal'` since the images come from different sensors.

```
[optimizer, metric] = imregconfig('multimodal')

optimizer =
  registration.optimizer.OnePlusOneEvolutionary

  Properties:
         GrowthFactor: 1.050000e+00
              Epsilon: 1.500000e-06
        InitialRadius: 6.250000e-03
    MaximumIterations: 100

metric =
  registration.metric.MattesMutualInformation

  Properties:
    NumberOfSpatialSamples: 500
     NumberOfHistogramBins: 50
               UseAllPixels: 1
```

Tune the properties of the optimizer to get the problem to converge on a global maxima and to allow for more iterations.

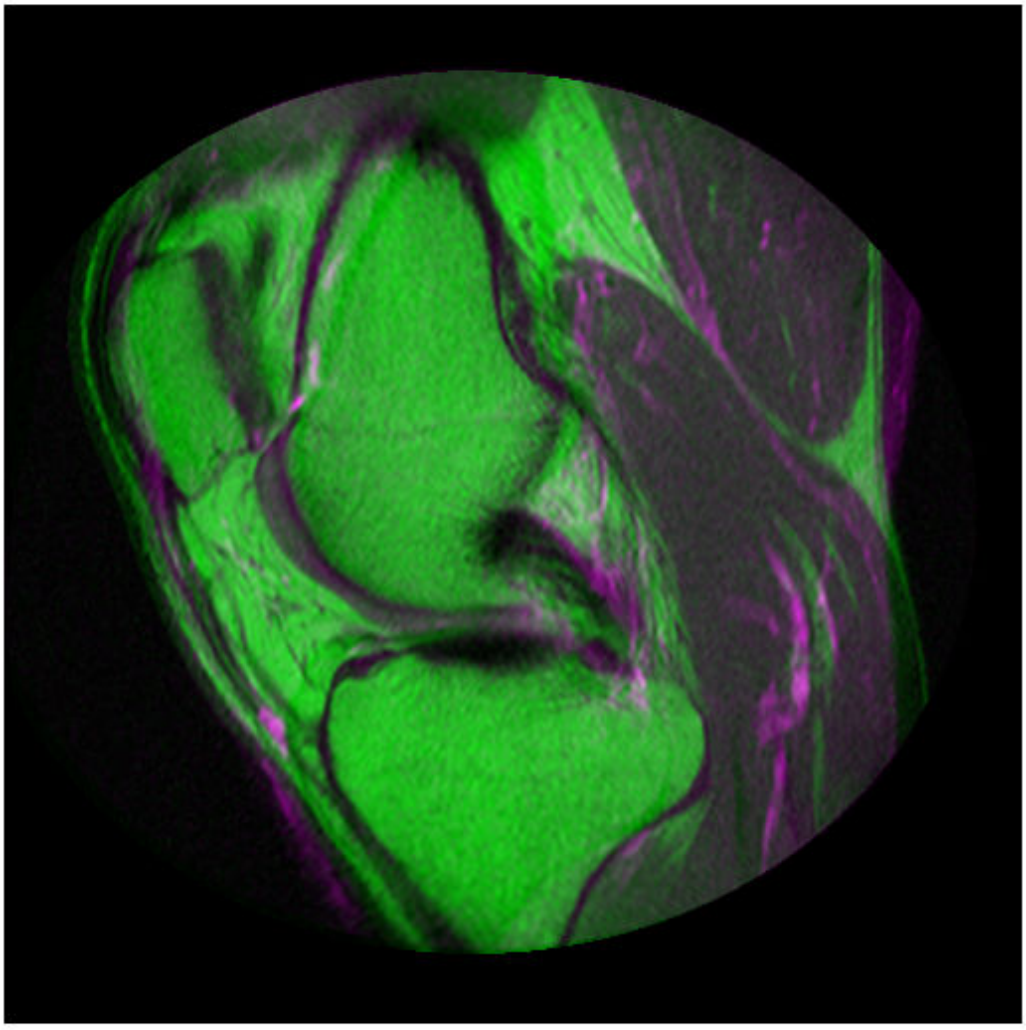
```
optimizer.InitialRadius = 0.009;
optimizer.Epsilon = 1.5e-4;
optimizer.GrowthFactor = 1.01;
optimizer.MaximumIterations = 300;
```

Perform the registration.

```
movingRegistered = imregister(moving, fixed, 'affine', optimizer, metric);
```

View the registered images.

```
figure
imshowpair(fixed, movingRegistered,'Scaling','joint')
```



Input Arguments

`modality` — Image capture modality

`"monomodal"` | `"multimodal"`

Image capture modality, specified as one of these values.

Modality	Description
`"monomodal"`	Monomodal images have similar brightness and contrast. The images are captured on the same type of scanner or sensor.

Modality	Description
`"multimodal"`	Multimodal images have different brightness and contrast. The images can come from two different types of devices, such as two camera models or two types of medical imaging modalities (like CT and MRI). The images can also come from a single device, such as a camera using different exposure settings, or an MRI scanner using different imaging sequences.

Data Types: `char` | `string`

Output Arguments

`optimizer` — Optimization configuration
`RegularStepGradientDescent` or `OnePlusOneEvolutionary` optimizer object

Optimization configuration, returned as a `RegularStepGradientDescent` or `OnePlusOneEvolutionary` optimizer object.

`metric` — Metric configuration
`MeanSquares` or `MattesMutualInformation` metric object

Metric configuration describes the image similarity metric to be optimized during registration, returned as a `MeanSquares` or `MattesMutualInformation` metric object.

Tips

- `imregconfig` returns `optimizer` and `metric` with default settings to provide a basic registration configuration. If you adjust the optimizer or metric properties, then the registration results can improve. For example, if you increase the number of iterations in the optimizer, reduce the optimizer step size, or change the number of samples in a stochastic metric, the registration improves to a point, at the expense of performance.

Extended Capabilities

Thread-Based Environment
Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Apps
Registration Estimator

Functions
`imshowpair` | `imregister`

Objects
`MattesMutualInformation` | `MeanSquares` | `RegularStepGradientDescent` | `OnePlusOneEvolutionary`

Topics
“Create an Optimizer and Metric for Intensity-Based Image Registration”
“Intensity-Based Automatic Image Registration”

Introduced in R2012a

imregcorr

Estimate geometric transformation that aligns two 2-D images using phase correlation

Syntax

```
tform = imregcorr(moving,fixed)
tform = imregcorr(moving,Rmoving,fixed,Rfixed)
tform = imregcorr( ___ ,transformType)
tform = imregcorr( ___ ,'Window',window)
[tform,peakcorr] = imregcorr( ___ )
```

Description

`tform = imregcorr(moving,fixed)` estimates the geometric transformation that aligns an image, `moving`, with a reference image, `fixed`. The function returns a geometric transformation object, `tform`, that maps pixels in `moving` to pixels in `fixed`.

`tform = imregcorr(moving,Rmoving,fixed,Rfixed)` estimates the geometric transformation that aligns an image, `moving`, with a reference image, `fixed`. `Rmoving` and `Rfixed` are spatial referencing objects that contain spatial information about the `moving` and `fixed` images, respectively. The transformation object returned, `tform`, defines the point mapping in the world coordinate system.

`tform = imregcorr( ___ ,transformType)` also specifies the type of transformation, `transformType`.

`tform = imregcorr( ___ ,'Window',window)` also specifies whether to perform windowing in the frequency domain. To increase the stability of registration results, specify `window` as `true`. However, if the common features in your images are oriented along the edges, then setting `window` to `false` can sometimes provide superior registration results.

`[tform,peakcorr] = imregcorr( ___ )` also returns the peak correlation, `peakcorr`, of the phase difference between the two images.

Examples

Register Images Using Phase Correlation

Read a reference image into the workspace.

```
fixed  = imread('cameraman.tif');
```

Create a synthetic moving image by scaling and rotating the fixed image.

```
theta = 20;
S = 2.3;
tform = affine2d([S.*cosd(theta) -S.*sind(theta) 0; ...
                  S.*sind(theta)  S.*cosd(theta) 0; ...
                  0 0 1]);
```

```
moving = imwarp(fixed,tform);
moving = moving + uint8(10*rand(size(moving)));
```

Display the fixed and the moving image alongside each other.

```
imshowpair(fixed,moving,'montage')
```

Estimate the transformation needed to align the images using `imregcorr`.

```
tformEstimate = imregcorr(moving,fixed);
```

Apply estimated geometric transform to the moving image. This example uses the `'OutputView'` parameter to obtain a registered image the same size and with the same world limits as the reference image.

```
Rfixed = imref2d(size(fixed));
movingReg = imwarp(moving,tformEstimate,'OutputView',Rfixed);
```

View the original image and the registered image side-by-side to check the registration. Then view the registered image overlaid on the original using the `'falsecolor'` option to highlight any areas where the images differ.

```
imshowpair(fixed,movingReg,'montage')
```

```
imshowpair(fixed,movingReg,'falsecolor');
```

Input Arguments

`moving` — Image to be registered
grayscale image | binary image | RGB image

Image to be registered, specified as a grayscale, binary, or RGB image. If you specify an RGB image, `imregcorr` converts it to a grayscale image using `rgb2gray` before processing.

Note The aspect ratio of `moving` affects the output transform `tform`. For best results, use a square image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`fixed` — Reference image in the target orientation
grayscale image | binary image | RGB image

Reference image in the target orientation, specified as a grayscale, binary, or RGB image. If you specify an RGB image, `imregcorr` converts it to a grayscale image using `rgb2gray` before processing.

Note The aspect ratio of `fixed` affects the output transform `tform`. For best results, use a square image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`transformType` — Type of transformation to estimate
`'similarity'` (default) | `'rigid'` | `'translation'`

Type of transformation to estimate, specified as one of the following values.

Value	Description
`'translation'`	Translation
`'rigid'`	Translation and rotation
`'similarity'`	Translation, rotation, and scaling When using the `'similarity'` option, the phase correlation algorithm is only scale invariant within some range of scale difference between the fixed and moving images. `imregcorr` limits the search space to scale differences within the range [1/4, 4]. `imregcorr` does not detect scale differences less than 1/4 or greater than 4.

Data Types: `char` | `string`

`Rmoving` — Spatial referencing information associated with the image to be registered
`imref2d` object

Spatial referencing information associated with the image to be registered, specified as an `imref2d` object.

`Rfixed` — Spatial referencing information associated with the reference (fixed) image
`imref2d` object

Spatial referencing information associated with the reference (fixed) image, specified as an `imref2d` object.

`window` — Use windowing to suppress spectral leakage effects
`true` or `1` (default) | `false` or `0`

Use windowing to suppress spectral leakage effects in the frequency domain, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `true`, the `imregcorr` function performs windowing using a Blackman filter.

Output Arguments

`tform` — Geometric transformation
`affine2d` object

Geometric transformation, returned as an `affine2d` object.

`peakcorr` — Peak correlation of phase difference
numeric scalar

Peak correlation value of the phase difference between the two images, returned as a numeric scalar.

Tips

- If your image is of type `double`, you can achieve performance improvements by casting the image to `single` with `im2single` before registration. Input images of type `double` cause the algorithm to compute FFTs in `double`.

References

[1] Reddy, B. S. and Chatterji, B. N., *An FFT-Based Technique for Translation, Rotation, and Scale-Invariant Image Registration*, IEEE Transactions on Image Processing, Vol. 5, No. 8, August 1996

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

`imregcorr` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

Thread-Based Environment
Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Apps
Registration Estimator

Functions
`imwarp` | `imshowpair` | `imregister` | `imregtform`

Introduced in R2014a

imregdemons

Estimate displacement field that aligns two 2-D or 3-D images

Syntax

```
[D,moving_reg] = imregdemons(moving,fixed)
[D,moving_reg] = imregdemons(moving,fixed,N)
[D,moving_reg] = imregdemons(___,Name,Value)
```

Description

`[D,moving_reg] = imregdemons(moving,fixed)` estimates the displacement field `D` that aligns the image to be registered, `moving`, with the reference image, `fixed`. The displacement vectors at each pixel location map locations from the `fixed` image grid to a corresponding location in the `moving` image. `moving_reg` is a warped version of the `moving` image that is warped according to the displacement field `D` and resampled using linear interpolation.

`[D,moving_reg] = imregdemons(moving,fixed,N)` specifies the number of iterations to be computed. This function does not use a convergence criterion and therefore is always guaranteed to run for the specified or default number of iterations.

`[D,moving_reg] = imregdemons(___,Name,Value)` registers the moving image using name-value pairs to control aspects of weight computation.

Examples

Register Two Images with Local Distortions

This example shows how to solve a registration problem in which the same hand has been photographed in two different poses. The misalignment of the images varies locally throughout each image. This is therefore a non-rigid registration problem.

Read the two images into the workspace.

```
fixed  = imread('hands1.jpg');
moving = imread('hands2.jpg');
```

Convert the images to grayscale for processing.

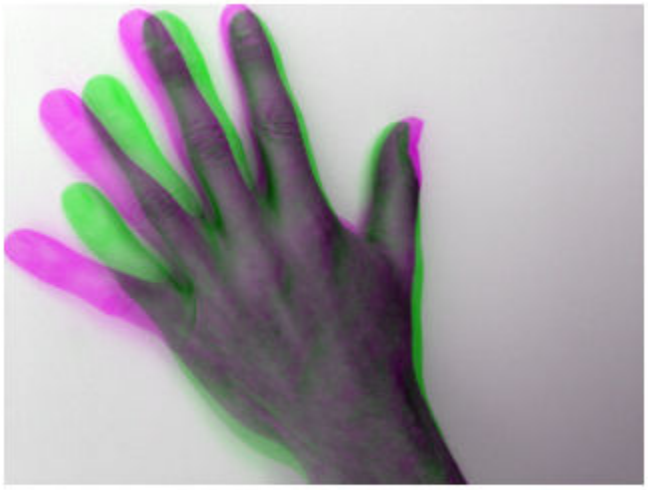
```
fixed  = im2gray(fixed);
moving = im2gray(moving);
```

Observe the initial misalignment. Fingers are in different poses.

```
imshowpair(fixed,moving,'montage')
```

Overlay the two images to make it easy to see where the images differ. The differences are highlighted in green and magenta.

```
imshowpair(fixed,moving)
```



Correct illumination differences between the `moving` and `fixed` images using histogram matching. This is a common pre-processing step.

```
moving = imhistmatch(moving,fixed);
```

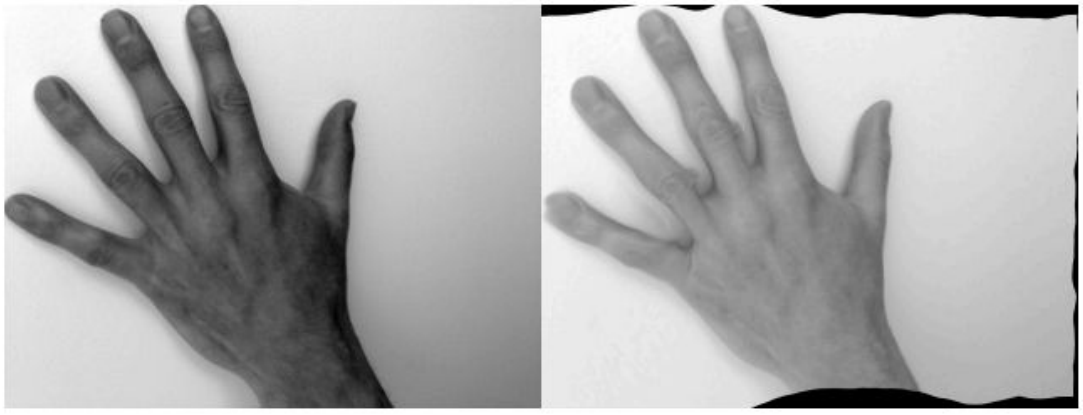
Estimate the transformation needed to bring the two images into alignment.

```
[~,movingReg] = imregdemons(moving,fixed,[500 400 200],...
    'AccumulatedFieldSmoothing',1.3);
```

Display the results of the registration. In the first figure, the images are overlaid to show the alignment.

```
imshowpair(fixed,movingReg)
```

```
imshowpair(fixed,movingReg,'montage')
```



Input Arguments

`moving` — Image to be registered

2-D grayscale image | 3-D grayscale image

Image to be registered, specified as a 2-D or 3-D grayscale image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`fixed` — Reference image in the target orientation

3-D grayscale image | 2-D grayscale image

Reference image in the target orientation, specified as a 2-D or 3-D grayscale image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`N` — Number of iterations

`100` (default) | positive integer scalar or vector

Number of iterations, specified as a positive integer scalar or vector.

When you specify a vector, `N` is the number of iterations per pyramid level (resolution level). For example, if there are 3 pyramid levels, then you can specify the vector `[100,50,25]`, where `imregdemons` performs 100 iterations at the lowest resolution level, 50 iterations at the next pyramid level, and 25 iterations at the last iteration level — the level with full resolution. Because it takes less time to process the lower resolution levels, running more iterations at low resolution and fewer iterations at the higher resolutions of the pyramid can help performance.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'AccumulatedFieldSmoothing',1.5` applies Gaussian smoothing with a standard deviation of 1.5 at each iteration

`AccumulatedFieldSmoothing` — Smoothing applied at each iteration

`1.0` (default) | positive scalar

Smoothing applied at each iteration, specified as the comma-separated pair consisting of `'AccumulatedFieldSmoothing'` and a numeric value. This parameter controls the amount of diffusion-like regularization. `imregdemons` applies the standard deviation of the Gaussian smoothing to regularize the accumulated field at each iteration. Larger values result in smoother output displacement fields. Smaller values result in more localized deformation in the output displacement field. Values typically are in the range [0.5, 3.0]. When you specify multiple `PyramidLevels`, the standard deviation used in the Gaussian smoothing remains the same at each pyramid level.

Data Types: `double`

`PyramidLevels` — Number of multi-resolution image pyramid levels to use

3 (default) | positive integer

Number of multi-resolution image pyramid levels to use, specified as the comma-separated pair consisting of `'PyramidLevels'` and a positive integer.

Data Types: `double`

`DisplayWaitbar` — Display wait bar to indicate progress
`true` (default) | `false`

Display wait bar to indicate progress, specified as the comma-separated pair consisting of `'DisplayWaitbar'` and the value `true` or `false`. When set to `true`, `imregdemons` displays a wait bar to indicate progress for long-running operations. To prevent `imregdemons` from displaying a wait bar, set `DisplayWaitbar` to `false`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

`D` — Displacement field
numeric array

Displacement field, specified as a numeric array. Displacement values are in units of pixels.

- If `fixed` is a 2-D grayscale image of size *m*-by-*n*, then the displacement field array is *m*-by-*n*-by-2. `D(:,:,1)` contains displacements along the *x*-axis and `D(:,:,2)` contains displacements along the *y*-axis.
- If `fixed` is a 3-D grayscale image of size *m*-by-*n*-by-*p*, then the displacement field array is *m*-by-*n*-by-*p*-by-3. `D(:,:,:,1)` contains displacements along the *x*-axis, `D(:,:,:,2)` contains displacements along the *y*-axis, and `D(:,:,:,3)` contains displacements along the *z*-axis.

Data Types: `double`

`moving_reg` — Aligned image
2-D or 3-D grayscale image

Registered image, returned as a 2-D or 3-D grayscale image. The image is warped according to the displacement field `D` and resampled using linear interpolation.

Tips

- To transform an image using the displacement field `D`, use `imwarp`.

References

[1] Thirion, J.-P. "Image matching as a diffusion process: an analogy with Maxwell's demons". *Medical Image Analysis*. Vol. 2, Number 3, 1998, pp. 243–260.

[2] Vercauteren, T., X. Pennec, A. Perchant, N. Ayache, "Diffeomorphic Demons: Efficient Non-parametric Image Registration", *NeuroImage*. Vol. 45, Number 1, Supplement 1, March 2009, pp. 61–72.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- The `DisplayWaitbar` name-value pair argument is not supported on the GPU.

For more information, see “Image Processing on a GPU”.

See Also

Apps

Registration Estimator

Functions

`imregcorr` | `imregister` | `imregtform` | `imshowpair` | `imwarp`

Introduced in R2014b

imregister

Intensity-based image registration

Syntax

```
moving_reg = imregister(moving,fixed,transformType,optimizer,metric)
[moving_reg,R_reg] = imregister(moving,Rmoving,fixed,Rfixed,transformType,
optimizer,metric)
___ = imregister(___,Name,Value)
```

Description

`moving_reg = imregister(moving,fixed,transformType,optimizer,metric)` transforms the 2-D or 3-D grayscale image, `moving`, so that it is registered with the reference image, `fixed`. `transformType` defines the type of transformation to perform. `metric` defines the quantitative measure of similarity between the images to optimize. `optimizer` describes the method for optimizing the metric. The function returns the registered image, `moving_reg`.

`[moving_reg,R_reg] = imregister(moving,Rmoving,fixed,Rfixed,transformType, optimizer,metric)` transforms the spatially referenced image `moving` so that it is registered with the spatially referenced image `fixed`. `Rmoving` and `Rfixed` are spatial referencing objects that describe the world coordinate limits and the resolution of `moving` and `fixed`.

`___ = imregister(___,Name,Value)` specifies additional options with one or more name-value pair arguments.

Examples

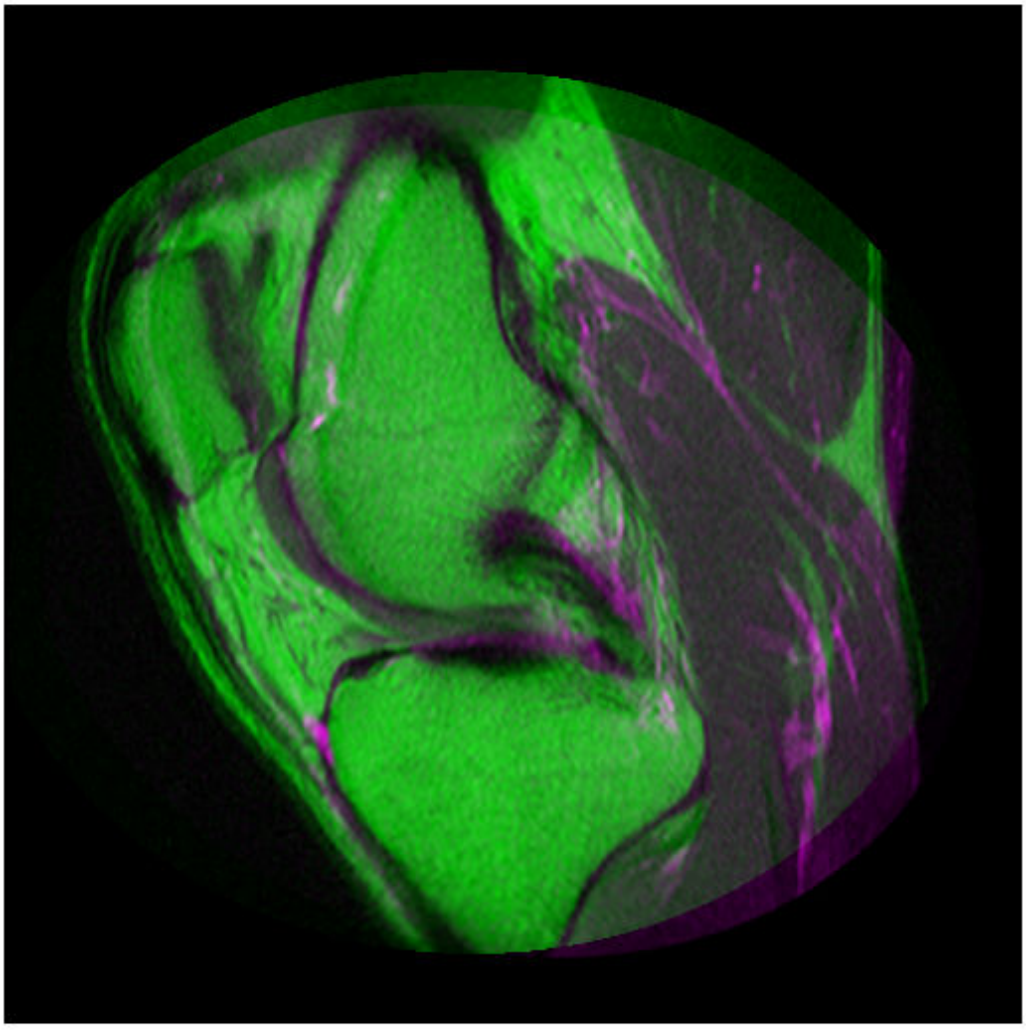
Register Multimodal MRI Images with Optimizer

Read two images. This example uses two magnetic resonance (MRI) images of a knee. The fixed image is a spin echo image, while the moving image is a spin echo image with inversion recovery. The two sagittal slices were acquired at the same time but are slightly out of alignment.

```
fixed = dicomread('knee1.dcm');
moving = dicomread('knee2.dcm');
```

View the misaligned images.

```
imshowpair(fixed, moving,'Scaling','joint')
```



Create the optimizer and metric, setting the modality to `'multimodal'` since the images come from different sensors.

```
[optimizer, metric] = imregconfig('multimodal')
```

```
optimizer =
  registration.optimizer.OnePlusOneEvolutionary

  Properties:
         GrowthFactor: 1.050000e+00
              Epsilon: 1.500000e-06
        InitialRadius: 6.250000e-03
    MaximumIterations: 100
```

```
metric =
  registration.metric.MattesMutualInformation

  Properties:
    NumberOfSpatialSamples: 500
     NumberOfHistogramBins: 50
              UseAllPixels: 1
```

Tune the properties of the optimizer to get the problem to converge on a global maxima and to allow for more iterations.

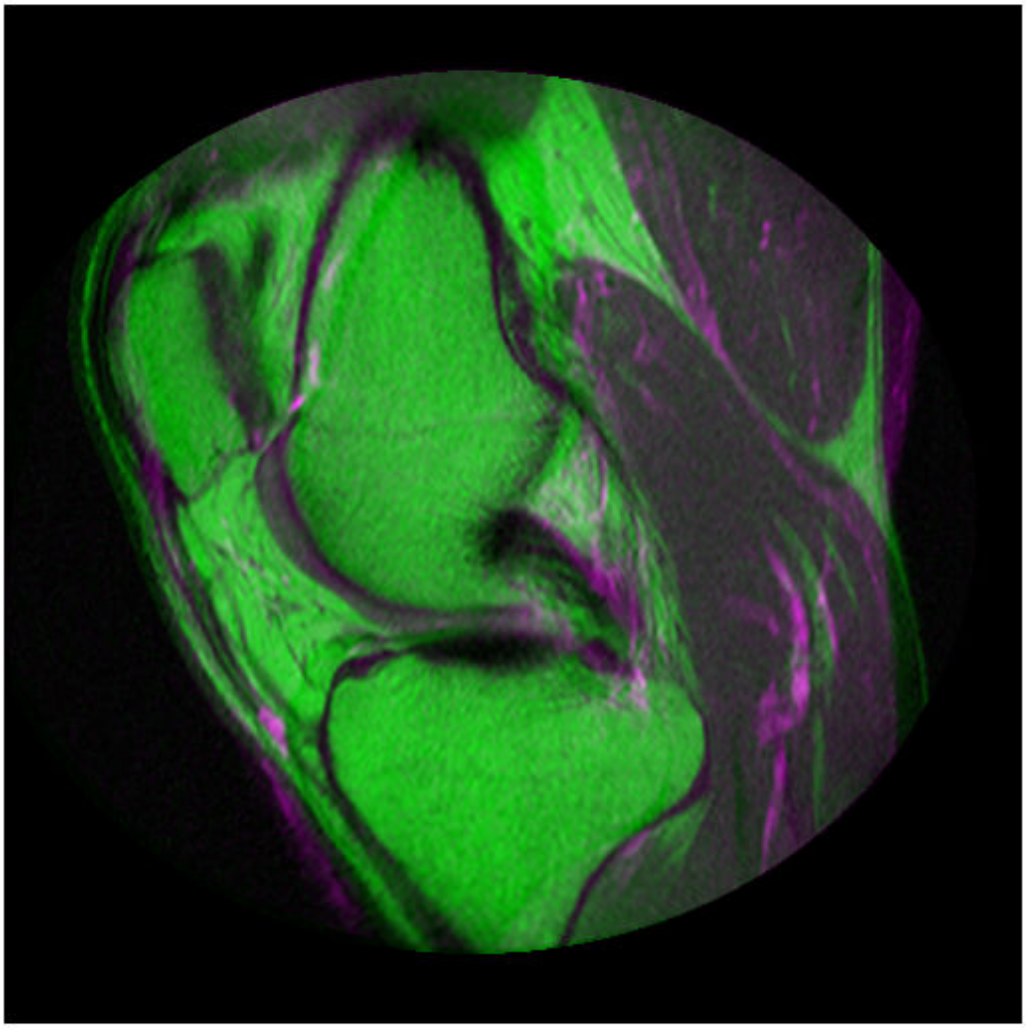
```
optimizer.InitialRadius = 0.009;
optimizer.Epsilon = 1.5e-4;
optimizer.GrowthFactor = 1.01;
optimizer.MaximumIterations = 300;
```

Perform the registration.

```
movingRegistered = imregister(moving, fixed, 'affine', optimizer, metric);
```

View the registered images.

```
figure
imshowpair(fixed, movingRegistered,'Scaling','joint')
```



Input Arguments

`moving` — Image to be registered
numeric matrix | 3-D numeric array

Image to be registered, specified as numeric matrix representing a 2-D grayscale image or a 3-D numeric array representing a 3-D grayscale volume.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`Rmoving` — Spatial referencing information associated with image to be registered
`imref2d` object | `imref3d` object

Spatial referencing information associated with the image to be registered, specified as an `imref2d` object or `imref3d` object.

`fixed` — Reference image
numeric matrix | 3-D numeric array

Reference image in the target orientation, specified as numeric matrix representing a 2-D grayscale image or a 3-D numeric array representing a 3-D grayscale volume. The reference image must have the same dimensionality as the image to be registered, `moving`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`Rfixed` — Spatial referencing information associated with reference image
`imref2d` object | `imref3d` object

Spatial referencing information associated with the reference (fixed) image, specified as an `imref2d` object or `imref3d` object.

`transformType` — Geometric transformation to be applied to image to be registered
`'translation'` | `'rigid'` | `'similarity'` | `'affine'`

Geometric transformation to be applied to the moving image, specified as one of the following values:

Value	Description
`'translation'`	(*x*,*y*) translation in 2-D, or (*x*,*y*,*z*) translation in 3-D.
`'rigid'`	Rigid transformation consisting of translation and rotation.
`'similarity'`	Nonreflective similarity transformation consisting of translation, rotation, and scale.
`'affine'`	Affine transformation consisting of translation, rotation, scale, and shear.

The `'similarity'` and `'affine'` transformation types always involve nonreflective transformations.

Data Types: `char` | `string`

`optimizer` — Method for optimizing similarity metric
`RegularStepGradientDescent` or `OnePlusOneEvolutionary` optimizer object

Method for optimizing the similarity metric, specified as a `RegularStepGradientDescent` or `OnePlusOneEvolutionary` optimizer object.

`metric` — Image similarity metric to be optimized during registration
`MeanSquares` or `MattesMutualInformation` metric object

Image similarity metric to be optimized during registration, specified as a `MeanSquares` or `MattesMutualInformation` metric object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'DisplayOptimization',1` enables the verbose optimization mode.

`DisplayOptimization` — Verbose optimization flag
`false` (default) | `true`

Verbose optimization flag, specified as the comma-separated pair consisting of `'DisplayOptimization'`, and the logical value `true` or `false`. Controls whether `imregister` displays optimization information in the command window during the registration process.

Data Types: `logical`

`InitialTransformation` — Starting geometric transformation
`affine2d` object | `affine3d` object

Starting geometric transformation, specified as the comma-separated pair consisting of `'InitialTransformation'` and an `affine2d` object or `affine3d` object.

`PyramidLevels` — Number of pyramid levels used during registration process
3 (default) | positive integer

Number of pyramid levels used during the registration process, specified as the comma-separated pair consisting of `'PyramidLevels'` and a positive integer.

Example: `'PyramidLevels',4` sets the number of pyramid levels to 4.

Data Types: `double`

Output Arguments

`moving_reg` — Registered image
numeric matrix | 3-D numeric array

Registered image, returned as a 2-D numeric matrix representing a 2-D grayscale image or a 3-D numeric array representing a 3-D grayscale volume. Any fill pixels introduced that do not correspond to locations in the original image are `0`.

`R_reg` — Spatial referencing information associated with registered image
`imref2d` object | `imref3d` object

Spatial referencing information associated with the registered image, returned as an `imref2d` object or `imref3d` object.

Tips

- Both `imregtform` and `imregister` use the same underlying registration algorithm. `imregister` performs the additional step of resampling `moving` to produce the registered output image from the geometric transformation estimate calculated by `imregtform`. Use `imregtform` when you want access to the geometric transformation that relates `moving` to `fixed`. Use `imregister` when you want a registered output image.
- Create an `optimizer` and `metric` with the `imregconfig` function before calling `imregister`. Getting good results from optimization-based image registration usually requires modifying optimizer or metric settings for the pair of images being registered. The `imregconfig` function provides a default configuration that should only be considered a starting point. For example, if you increase the number of iterations in the optimizer, reduce the optimizer step size, or change

the number of samples in a stochastic metric, the registration improves to a point, at the expense of performance. See the output of `imregconfig` for more information on the different parameters that you can modify.

- If the spatial scaling of your images differs by more than 10%, resize them with `imresize` before registering them.
- Use `imshowpair` or `imfuse` to visualize the results of registration.
- You can use `imregister` in an automated workflow to register several images.
- When you have spatial referencing information about the image to be registered, specify the information to `imregister` using spatial referencing objects. This helps `imregister` converge to better results more quickly because scale differences can be taken into account.

See Also

Apps
Registration Estimator

Functions
`imregconfig` | `imregcorr` | `imregtform` | `imwarp` | `imshowpair` | `imfuse`

Objects
`imref2d` | `imref3d`

Topics
“Create an Optimizer and Metric for Intensity-Based Image Registration”
“Intensity-Based Automatic Image Registration”

Introduced in R2012a

imregmtb

Register 2-D images using median threshold bitmaps

Syntax

```
[R1,R2,...,Rn,shift] = imregmtb(M1,M2,...,Mn,F)
```

Description

`[R1,R2,...,Rn,shift] = imregmtb(M1,M2,...,Mn,F)` registers an arbitrary number of moving images `M1,M2,...,Mn` with respect to the fixed (reference) image, `F`, using the median threshold bitmap technique. The registered images are returned in `R1,R2,...,Rn`, and the estimated displacement of the registered images is returned in `shift`.

The median threshold bitmap technique is effective for registering images captured with variable exposures. `imregmtb` considers only translations, not rotations or other types of geometric transformations.

Examples

Register Images with Jitter Using Median Threshold Bitmaps

Read a series of images with different exposures.

```
I1 = imread('office_1.jpg');
I2 = imread('office_2.jpg');
I3 = imread('office_3.jpg');
I4 = imread('office_4.jpg');
I5 = imread('office_5.jpg');
I6 = imread('office_6.jpg');
```

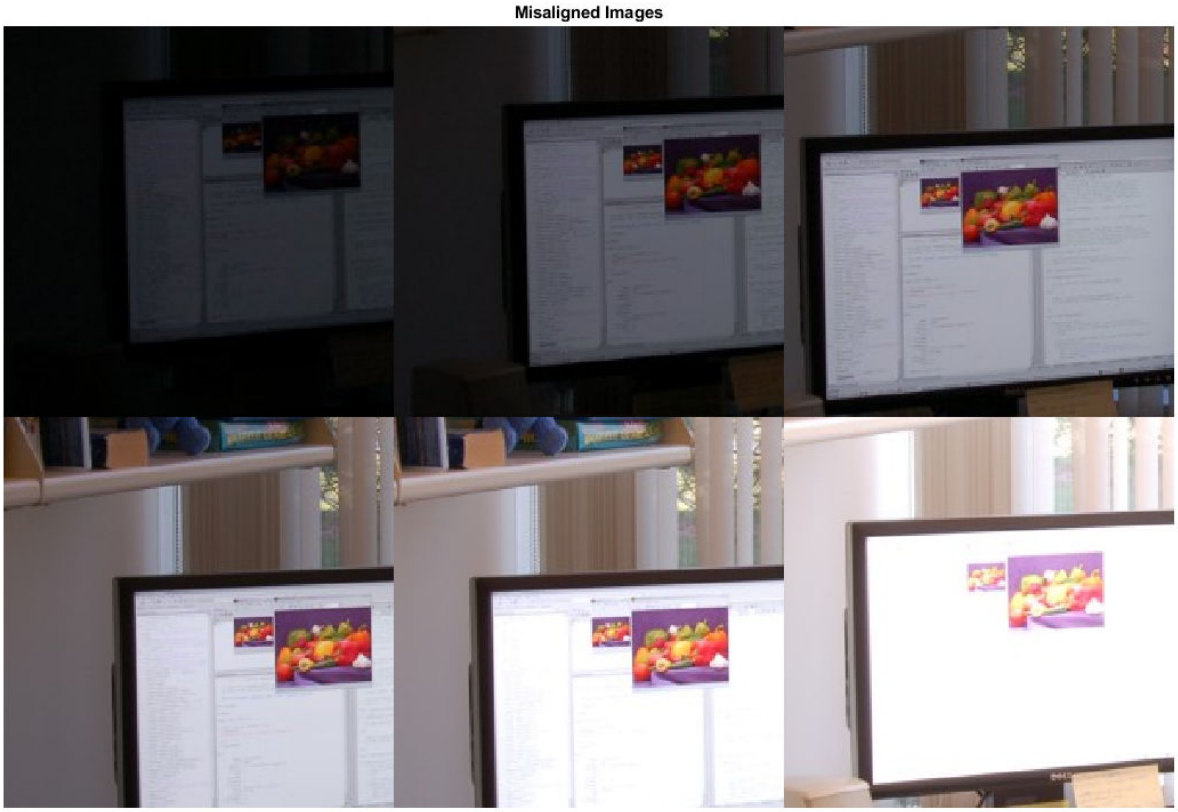
The images were captured from a fixed camera, and there are no moving objects in the scene. For this example, simulate camera motion, or jitter, by translating each image horizontally and vertically by a random amount in the range [–30, 30] pixels. Store the translation values for all five moving images in the 5-by-2 matrix `t`. Designate the sixth image, `I6`, as the fixed (or reference) image. Do not apply jitter to this image.

```
t = randi([-30 30],5,2);
I1 = imtranslate(I1,t(1,:));
I2 = imtranslate(I2,t(2,:));
I3 = imtranslate(I3,t(3,:));
I4 = imtranslate(I4,t(4,:));
I5 = imtranslate(I5,t(5,:));
```

To compare the image positions, display a region of interest (ROI) from the center of each image. The vector `roi` specifies the *x*- and *y*-coordinate of the top left corner, and the width and height of the ROI.

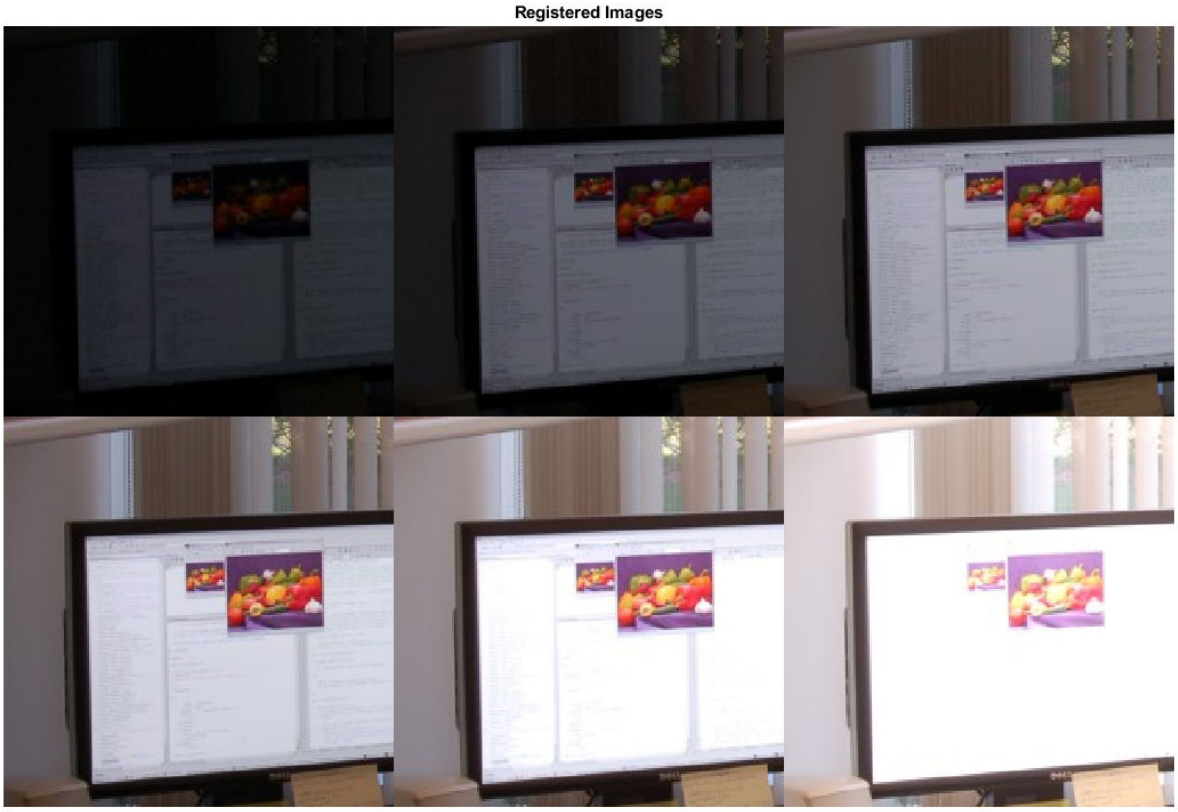
```
roi = [140 260 200 200];
montage({imcrop(I1,roi),imcrop(I2,roi),imcrop(I3,roi), ...
```

```
    imcrop(I4,roi),imcrop(I5,roi),imcrop(I6,roi)})
title('Misaligned Images')
```



Register the spatially shifted images using median threshold bitmaps. Display an ROI from the center of each image.

```
[R1,R2,R3,R4,R5,shift] = imregmtb(I1,I2,I3,I4,I5,I6);
montage({imcrop(R1,roi),imcrop(R2,roi),imcrop(R3,roi), ...
    imcrop(R4,roi),imcrop(R5,roi),imcrop(I6,roi)})
title('Registered Images')
```



The images look well-aligned.

Examine the estimated displacement, `shift`, of each moving image with respect to the fixed image. `shift` represents the estimated transformation that must be applied to the moving image to align it with the fixed image.

```
shift
```

```
shift = 5×2

   -26    25
   -25    14
    23    -3
   -25   -28
    -8   -28
```

Compare the estimated displacement to the actual displacement. Recall that the transformation `t` was applied to the fixed image to simulate the jitter of each moving image. Therefore, the transformation `-t` is analogous to the transformation returned by `shift`.

```
-t
```

```
ans = 5×2

   -19    25
```

```
   -25    14
    23    -3
   -25   -28
    -8   -28
```

The `imregmtb` function does a good job estimating the displacement of each frame.

Input Arguments

`M1,M2,...,Mn` — Moving images

grayscale images | RGB images

Moving images, specified as a series of grayscale images or RGB images with identical or variable exposures. The images must have the same size and data type.

Data Types: `single` | `double` | `uint8` | `uint16`

`F` — Fixed image

grayscale image | RGB image

Fixed image, specified as a grayscale image or RGB image. `F` must have the same size and data type as the moving images, `M1,M2,...,Mn`.

Data Types: `single` | `double` | `uint8` | `uint16`

Output Arguments

`R1,R2,...,Rn` — Registered images

grayscale images | RGB images

Registered images, returned as a series of grayscale images or RGB images. The registered images have the same size and data type as the moving images, `M1,M2,...,Mn`.

`shift` — Estimated displacement

n-by-2 numeric matrix

Estimated displacement in the horizontal and vertical direction of the *n* registered images, returned as an *n*-by-2 numeric matrix.

References

[1] Reinhard, E., W. Heidrich, P. Debevec, S. Pattanaik, G. Ward, K. Myszkowski. *High Dynamic Range Imaging, Second Edition*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2010, pp. 155–170.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

`blendexposure` | `imtranslate` | `imregister` | `imregcorr`

Introduced in R2018a

imregtform

Estimate geometric transformation that aligns two 2-D or 3-D images

Syntax

```
tform = imregtform(moving,fixed,transformType,optimizer,metric)
tform = imregtform(moving,Rmoving,fixed,Rfixed,transformType,optimizer,
metric)
tform = imregtform( ___ ,Name,Value)
```

Description

`tform = imregtform(moving,fixed,transformType,optimizer,metric)` estimates the geometric transformation that aligns the moving image `moving` with the fixed image `fixed`. `transformType` is a string scalar or character vector that defines the type of transformation to estimate. `optimizer` is an object that describes the method for optimizing the metric. `metric` is an object that defines the quantitative measure of similarity between the images to optimize. The output `tform` is a geometric transformation object that maps `moving` to `fixed`.

`tform = imregtform(moving,Rmoving,fixed,Rfixed,transformType,optimizer,metric)` estimates the geometric transformation where `Rmoving` and `Rfixed` specify the spatial referencing objects associated with the `moving` and `fixed` images. The output `tform` is a geometric transformation object in units defined by the spatial referencing objects `Rmoving` and `Rfixed`.

`tform = imregtform( ___ ,Name,Value)` estimates the geometric transformation using name-value pairs to control aspects of the operation.

Examples

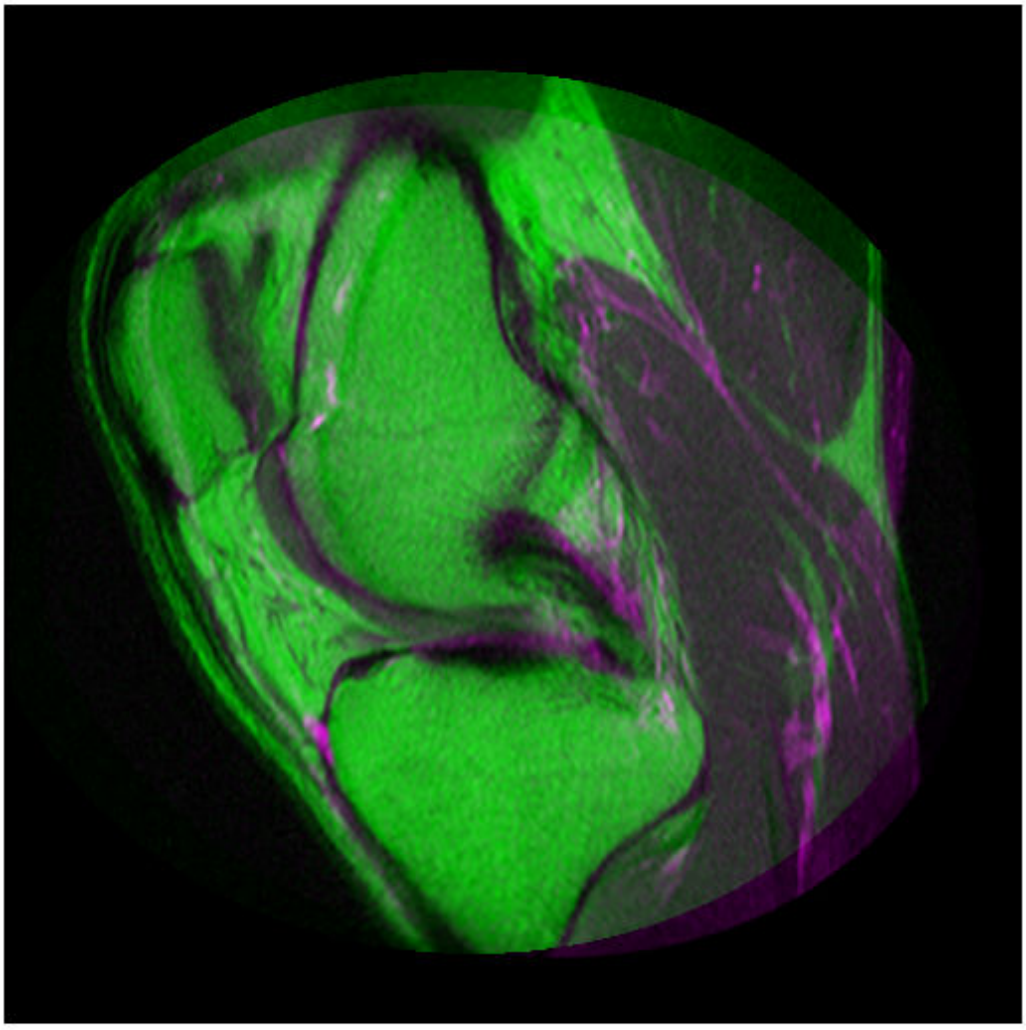
Estimate Transformation Needed for Image Registration

Read two images. This example uses two magnetic resonance (MRI) images of a knee. The fixed image is a spin echo image, while the moving image is a spin echo image with inversion recovery. The two sagittal slices were acquired at the same time but are slightly out of alignment.

```
fixed = dicomread('knee1.dcm');
moving = dicomread('knee2.dcm');
```

View the misaligned images.

```
imshowpair(fixed, moving,'Scaling','joint')
```



Create the optimizer and metric, setting the modality to `'multimodal'` since the images come from different sensors.

```
[optimizer, metric] = imregconfig('multimodal')
```

```
optimizer =
  registration.optimizer.OnePlusOneEvolutionary

  Properties:
       GrowthFactor: 1.050000e+00
            Epsilon: 1.500000e-06
      InitialRadius: 6.250000e-03
    MaximumIterations: 100
```

```
metric =
  registration.metric.MattesMutualInformation

  Properties:
    NumberOfSpatialSamples: 500
     NumberOfHistogramBins: 50
              UseAllPixels: 1
```

Tune the properties of the optimizer to get the problem to converge on a global maxima and to allow for more iterations.

```
optimizer.InitialRadius = 0.009;
optimizer.Epsilon = 1.5e-4;
optimizer.GrowthFactor = 1.01;
optimizer.MaximumIterations = 300;
```

Find the geometric transformation that maps the image to be registered (`moving`) to the reference image (`fixed`).

```
tform = imregtform(moving, fixed, 'affine', optimizer, metric)
```

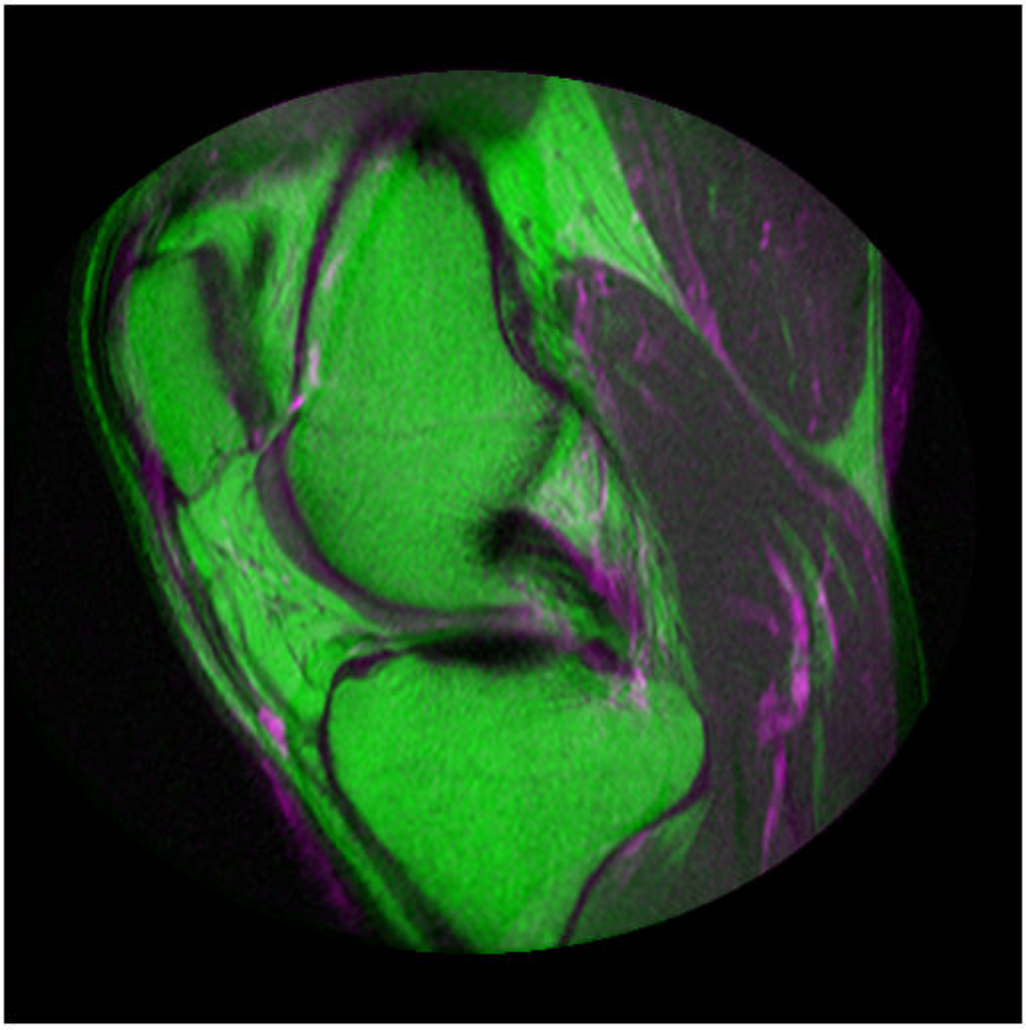
```
tform =
  affine2d with properties:

                T: [3x3 double]
    Dimensionality: 2
```

Apply the transformation to the image being registered (`moving`) using the `imwarp` function. The example uses the `'OutputView'` parameter to preserve world limits and resolution of the reference image when forming the transformed image.

```
movingRegistered = imwarp(moving,tform,'OutputView',imref2d(size(fixed)));
```

View the registered images.

```
figure
imshowpair(fixed, movingRegistered,'Scaling','joint')
```



Input Arguments

`moving` — Image to be registered
2-D or 3-D grayscale image

Image to be registered, specified as a 2-D or 3-D grayscale image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`Rmoving` — Spatial referencing information associated with the image to be registered
`imref2d` or `imref3d` object

Spatial referencing information associated with the image to be registered, specified as an `imref2d` or `imref3d` object.

`fixed` — Reference image in the target orientation
2-D or 3-D grayscale image

Reference image in the target orientation, specified as a 2-D or 3-D grayscale image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`Rfixed` — Spatial referencing information associated with the reference (fixed) image
`imref2d` or `imref3d` object

Spatial referencing information associated with the reference (fixed) image, specified as an `imref2d` or `imref3d` object.

`transformType` — Geometric transformation to be applied to the image to be registered
`'translation'` | `'rigid'` | `'similarity'` | `'affine'`

Geometric transformation to be applied to the image to be registered, specified as one of the following values:

Value	Description
`'translation'`	*(x,y)* translation.
`'rigid'`	Rigid transformation consisting of translation and rotation.
`'similarity'`	Nonreflective similarity transformation consisting of translation, rotation, and scale.
`'affine'`	Affine transformation consisting of translation, rotation, scale, and shear.

The `'similarity'` and `'affine'` transformation types always involve nonreflective transformations.

Data Types: `char` | `string`

`optimizer` — Method for optimizing the similarity metric
`RegularStepGradientDescent` or `OnePlusOneEvolutionary` optimizer object

Method for optimizing the similarity metric, specified as a `RegularStepGradientDescent` or `OnePlusOneEvolutionary` optimizer object.

`metric` — Image similarity metric to be optimized during registration
`MeanSquares` or `MattesMutualInformation` metric object

Image similarity metric to be optimized during registration, specified as a `MeanSquares` or `MattesMutualInformation` metric object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'DisplayOptimization',1` enables verbose optimization mode.

`DisplayOptimization` — Verbose optimization flag
`false` (default) | `true`

Verbose optimization flag, specified as the comma-separated pair consisting of `'DisplayOptimization'`, and the logical value `true` or `false`. Controls whether `imregister` displays optimization information in the command window during the registration process.

Data Types: `logical`

`InitialTransformation` — Starting geometric transformation
`affine2d` or `affine3d` object

Starting geometric transformation, specified as the comma-separated pair consisting of `'InitialTransformation'` and an `affine2d` or `affine3d` object.

`PyramidLevels` — Number of multi-level image pyramid levels used during the registration process
3 (default) | positive integer

Number of pyramid levels used during the registration process, specified as the comma-separated pair consisting of `'PyramidLevels'` and a positive integer.

Example: `'PyramidLevels',4` sets the number of pyramid levels to 4.

Output Arguments

`tform` — Geometric transformation
`affine2d` or `affine3d` object

Geometric transformation, returned as an `affine2d` or `affine3d` object. If the input matrices are 3-D, `imregtform` returns an `affine3d` object.

Tips

- When you have spatial referencing information available, it is important to provide this information to `imregtform`, using spatial referencing objects. This information helps `imregtform` converge to better results more quickly because scale differences can be considered.
- Both `imregtform` and `imregister` use the same underlying registration algorithm. `imregister` performs the additional step of resampling `moving` to produce the registered output image from the geometric transformation estimate calculated by `imregtform`. Use `imregtform` when you want access to the geometric transformation that relates `moving` to `fixed`. Use `imregister` when you want a registered output image.
- Getting good results from optimization-based image registration usually requires modifying optimizer and/or metric settings for the pair of images being registered. The `imregconfig` function provides a default configuration that should only be considered a starting point. See the output of the `imregconfig` for more information on the different parameters that can be modified.

See Also

Apps
Registration Estimator

Functions
`imregconfig` | `imregister` | `imshowpair` | `imwarp`

Objects
`affine2d` | `affine3d`

Topics
“Create an Optimizer and Metric for Intensity-Based Image Registration”

Introduced in R2013a

imresize3

Resize 3-D volumetric intensity image

Syntax

```
B = imresize3(V,scale)
B = imresize3(V,[numrows numcols numplanes])
B = imresize3( ___ ,method)
B = imresize3( ___ ,Name,Value)
```

Description

`B = imresize3(V,scale)` returns the volume `B` that is `scale` times the size of 3-D numeric or categorical volume `V`.

`B = imresize3(V,[numrows numcols numplanes])` returns the volume `B` that has the number of rows, columns, and planes specified by the 3-element vector `[numrows numcols numplanes]`.

`B = imresize3( ___ ,method)` returns the volume `B`, where `method` specifies the interpolation method used.

`B = imresize3( ___ ,Name,Value)` returns a resized volume where `Name,Value` pairs control aspects of the operation.

Examples

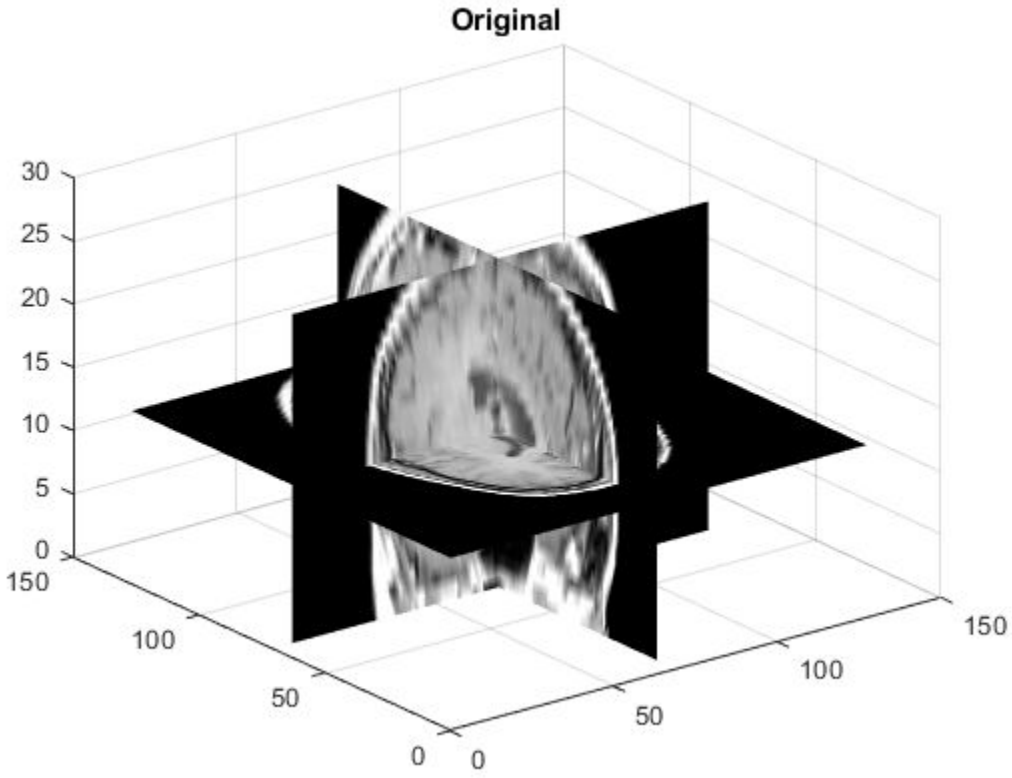
Resize 3-D Volumetric Image

Read MRI volume into the workspace.

```
s = load('mri');
mriVolumeOriginal = squeeze(s.D);
sizeO = size(mriVolumeOriginal);
```

Visualize the volume.

```
figure;
slice(double(mriVolumeOriginal),sizeO(2)/2,sizeO(1)/2,sizeO(3)/2);
shading interp, colormap gray;
title('Original');
```

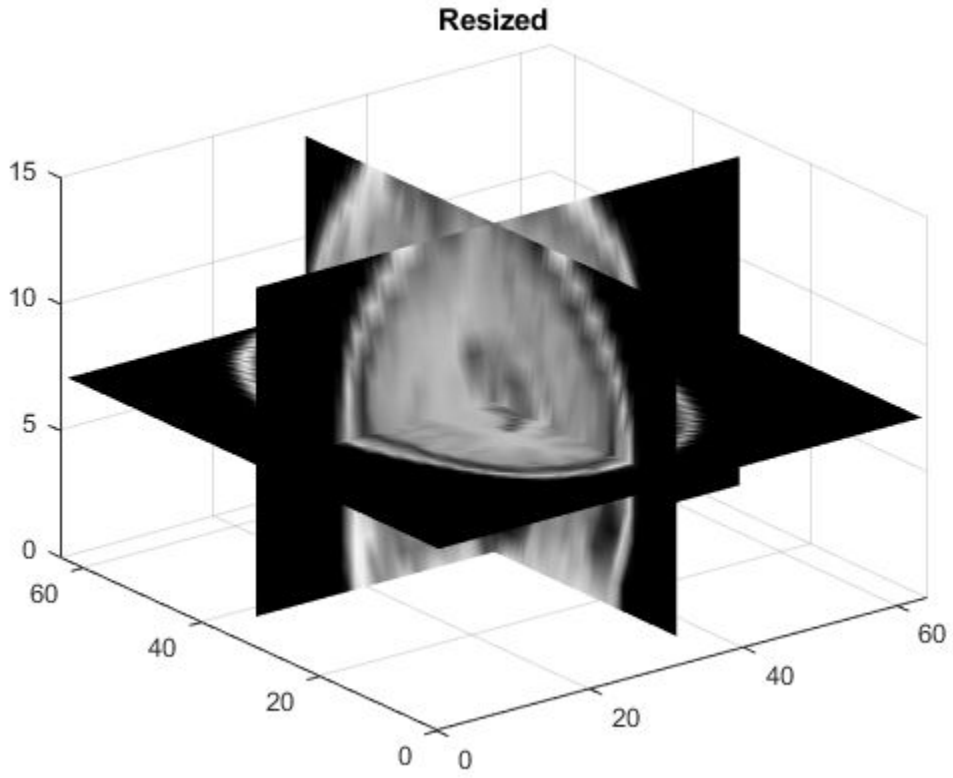


Resize the volume, reducing the size of all dimensions by one-half. This example uses the default interpolation method and antialiasing.

```
mriVolumeResized = imresize3(mriVolumeOriginal, 0.5);
sizeR = size(mriVolumeResized);
```

Visualize the resized volume.

```
figure;
slice(double(mriVolumeResized),sizeR(2)/2,sizeR(1)/2,sizeR(3)/2);
shading interp, colormap gray;
title('Resized');
```



Input Arguments

`V` — Volume to be resized

3-D numeric array | 3-D categorical array

Volume to be resized, specified as a 3-D numeric array or 3-D categorical array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `categorical`

`scale` — Scale factor

numeric scalar

Scale factor, specified as a numeric scalar.

- If `scale` is less than 1, then the output image is smaller than the input volume.
- If `scale` is greater than 1, then the output image is larger than the input volume.

`imresize3` applies the scale factor to each dimension in the volume. To apply a different resize factor to each dimension, use the `Scale` name-value pair argument.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`[numrows numcols numplanes]` — Size of output volume

3-element vector of positive integers

Size of output volume, specified as a 3-element vector of positive integers in the form `[rows columns planes]`. If you specify one numeric value and the other two values as `NaN`s, then `imresize3` computes the other two elements automatically to preserve the aspect ratio.

Data Types: `single` | `double`

`method` — Interpolation method

`'nearest'` | `'linear'` | `'box'` | `'triangle'` | `'lanczos2'` | `'lanczos3'`

Interpolation method, specified as one of the values in the following table that identifies a general method or a named interpolation kernel.

Method	Description
`'nearest'`	Nearest-neighbor interpolation. Nearest-neighbor interpolation is the only interpolation method supported for categorical images and it is the default method for images of this type.
`'linear'`	Linear interpolation
`'cubic'`	Cubic interpolation. Cubic interpolation is the default for numeric volumes. **Note** Cubic interpolation can produce pixel values outside the original range.
Interpolation Kernel	**Description**
`'box'`	Box-shaped kernel. The box-shaped kernel is the only interpolation kernel supported for categorical images.
`'triangle'`	Triangular kernel (equivalent to `'linear'`)
`'lanczos2'`	Lanczos-2 kernel
`'lanczos3'`	Lanczos-3 kernel

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Antialiasing',false`

`Antialiasing` — Perform antialiasing when shrinking a volume

`true` | `false`

Perform antialiasing when shrinking a volume, specified as the comma-separated pair consisting of `'Antialiasing'` and `true` or `false`.

- If `method` is `'nearest'`, then the default value of `'Antialiasing'` is `false`.

- If the interpolation `method` is the `'box'` interpolation kernel and the input volume is categorical, then the default value of `'Antialiasing'` is `false`.
- For all other interpolation methods, the default value of `'Antialiasing'` is `true`.

Data Types: `logical`

`Method` — Interpolation method
`'cubic'` (default) | character vector

Interpolation method, specified as the comma-separated pair consisting of `'Method'` and string scalar or character vector. For details, see `method`.

Data Types: `char` | `string`

`OutputSize` — Size of output volume
3-element vector of positive integers

Size of the output volume, specified as the comma-separated pair consisting of `'OutputSize'` and a 3-element vector of positive integers of the form `[rows cols planes]`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Scale` — Resize scale factor
positive number | 3-element vector of positive numbers

Resize scale factor, specified as the comma-separated pair consisting of `'Scale'` and a positive number or 3-element vector of positive numbers. If you specify a scalar, then `imresize3` applies the same scale factor to each dimension in the volume. If you specify a 3-element vector, then `imresize3` applies a different scale value to each dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`B` — Resized volume
array

Resized volume, returned as an array of the same class as the input volume, `V`.

Extended Capabilities

Thread-Based Environment
Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also
`imresize` | `imrotate` | `imrotate3` | `imwarp`

Introduced in R2017a

imroi class

(Not recommended) Region-of-interest (ROI) abstract base class

Note The `imroi` abstract class and its inherited classes are not recommended. Use new ROI objects instead. For more information, see “Compatibility Considerations”.

Description

The `imroi` class is an abstract base class that specifies the Image Processing Toolbox interface for working with regions of interest (ROIs). You can use classes that inherit from the `imroi` interface to create interactive ROIs over an image.

The `imroi` class is a `handle` class.

Class Attributes

`Abstract`	`true`

For information on class attributes, see “Class Attributes”.

Creation

The `imroi` class is abstract, and creating an instance of the `imroi` class is not allowed. To learn how to create an ROI object from a concrete subclass of `imroi`, see `imellipse`, `imfreehand`, `imline`, `impoint`, `impoly`, or `imrect`.

Properties

`Deletable` — ROI can be deleted

`true` (default) | `false`

ROI can be deleted, specified as `true` or `false`.

Attributes:

`GetAccess`	`public`
`SetAccess`	`public`

Data Types: `logical`

Methods

Public Methods

addNewPositionCallback	Add new-position callback to ROI object
createMask	(Not recommended) Create mask within image
getColor	Get color used to draw ROI object

getPosition	Return current position of ROI object
getPositionConstraintFcn	Return function handle to current position constraint function
removeNewPositionCallback	Remove new-position callback from ROI object
resume	(Not recommended) Resume execution of MATLAB command line
setColor	(Not recommended) Set color used to draw ROI object
setConstrainedPosition	Set ROI object to new position
setPositionConstraintFcn	Set position constraint function of ROI object
wait	(Not recommended) Block MATLAB command line until ROI creation is finished

Specialized Operators and Functions

These methods specialize standard MATLAB operators and functions for objects in this class.

`delete`

Compatibility Considerations

`imroi` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

For more information on updating code using instances of `imroi` objects to the new ROI objects, see “ROI Migration”.

See Also

`AssistedFreehand` | `Circle` | `Crosshair` | `Cuboid` | `Ellipse` | `Freehand` | `Line` | `Point` | `Polygon` | `Polyline` | `Rectangle`

Topics

“Create ROI Shapes”
“ROI Migration”

Introduced in R2008a

imdistline

Distance tool

Description

An `imdistline` object encapsulates a Distance tool, which consists of an interactive line over an image, paired with a text label that displays the distance between the line endpoints.

You can adjust the size and position of the line by using the mouse. The line also has a context menu that controls aspects of its appearance and behavior. For more information, see “Usage” on page 1-1841.

Creation

Syntax

```
h = imdistline
h = imdistline(hparent)
h = imdistline( ___ ,x,y)
```

Description

`h = imdistline` creates a Distance tool on the current axes. The function returns `h`, a handle to an `imdistline` object.

`h = imdistline(hparent)` creates a draggable Distance tool on the object specified by `hparent`.

`h = imdistline( ___ ,x,y)` creates a Distance tool with endpoints at the positions specified by `x` and `y`.

Input Arguments

`hparent` — Handle to parent object
handle

Handle to parent object, specified as a handle. The parent is typically an axes object, but can also be any other object that can be the parent of an `hggroup` object.

`x` — *x*-coordinates of endpoints
2-element numeric vector

x-coordinates of endpoints, specified as a 2-element numeric vector.

Example: `h = imdistline(gca,[10 100],[20 40]);` sets the first endpoint at the (x, y) coordinate (10, 20) and the second endpoint at the coordinate (100, 40).

`y` — *y*-coordinates of endpoints
2-element numeric vector

y-coordinates of endpoints, specified as a 2-element numeric vector.

Example: `h = imdistline(gca,[10 100],[20 40]);` sets the first endpoint at the (x, y) coordinate (10, 20) and the second endpoint at the coordinate (100, 40).

Properties

`Deletable` — ROI can be deleted

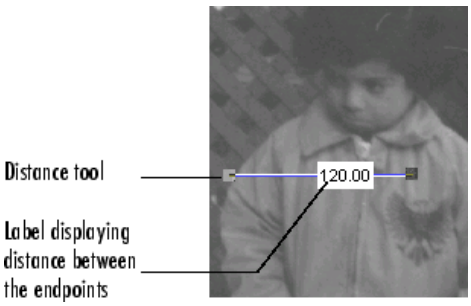
`true` (default) | `false`

ROI can be deleted, specified as `true` or `false`.

Data Types: `logical`

Usage

To move the Distance tool, position the pointer over the line, the shape changes to the fleur,  . Click and drag the line using the mouse. To resize the Distance tool, move the pointer over either of the endpoints of the line, the shape changes to the pointing finger,  . Click and drag the endpoint of the line using the mouse.



The line also supports a context menu that allows you to control various aspects of its functioning and appearance. Right-click the line to access the context menu.

Distance Tool Behavior	Context Menu Item
Export endpoint and distance data to the workspace	Select **Export to Workspace** from the context menu.
Toggle the distance label on/off.	Select **Show Distance Label** from the context menu.
Specify horizontal and vertical drag constraints	Select **Constrain Drag** from the context menu.
Change the color used to display the line.	Select **Set Color** from the context menu.
Delete the Distance tool object	Select **Delete** from the context menu.

Object Functions

addNewPositionCallback	Add new-position callback to ROI object
createMask	(Not recommended) Create mask within image
delete	Delete handle object

getAngleFromHorizontal	Return angle between Distance tool and horizontal axis
getColor	Get color used to draw ROI object
getDistance	Return distance between endpoints of Distance tool
getLabelHandle	Return handle to text label of Distance tool
getLabelTextFormatter	Return format of text label of Distance tool
getLabelVisible	Return visibility of text label of Distance tool
getPosition	Return current position of ROI object
getPositionConstraintFcn	Return function handle to current position constraint function
removeNewPositionCallback	Remove new-position callback from ROI object
resume	(Not recommended) Resume execution of MATLAB command line
setColor	(Not recommended) Set color used to draw ROI object
setConstrainedPosition	Set ROI object to new position
setLabelTextFormatter	Set format used to display text label of Distance tool
setLabelVisible	Set visibility of text label of Distance tool
setPosition	(Not recommended) Move ROI object to new position
setPositionConstraintFcn	Set position constraint function of ROI object
wait	(Not recommended) Block MATLAB command line until ROI creation is finished

Examples

Insert Distance Tool into an Image

Insert a Distance tool into an image. Use `makeConstrainToRectFcn` to specify a drag constraint function that prevents the Distance tool from being dragged outside the extent of the image. Right-click the Distance tool and explore the context menu options.

```
imshow('pout.tif')
h = imdistline;
fcn = makeConstrainToRectFcn('imline',get(gca,'XLim'),get(gca,'YLim'));
setDragConstraintFcn(h,fcn);
```

Position Endpoints of Distance Tool

Position endpoints of the Distance tool at the specified locations.

```
imshow('pout.tif')
h = imdistline(gca,[10 100],[10 100]);
```

Delete the Distance tool.

```
delete(h)
```

Use Distance Tool with Spatial Referencing

Use the Distance tool with `XData` and `YData` of associated image in non-pixel units. This example requires the `boston.tif` image from the Mapping Toolbox™ software, which includes material copyrighted by GeoEye™, all rights reserved.

```
start_row = 1478;
end_row = 2246;
meters_per_pixel = 1;
```

```
rows = [start_row meters_per_pixel end_row];
start_col = 349;
end_col = 1117;
cols = [start_col meters_per_pixel end_col];
img  = imread('boston.tif','PixelRegion',{rows,cols});
figure
hImg = imshow(img);
title('1 meter per pixel')
```

Specify the initial position of distance tool on Harvard Bridge.

```
hline = imdistline(gca,[271 471],[108 650]);
setLabelTextFormatter(hline,'%02.0f meters');
```

Repeat the process but work with a 2 meter per pixel sampled image. Verify that the same distance is obtained.

```
meters_per_pixel = 2;
rows = [start_row meters_per_pixel end_row];
cols = [start_col meters_per_pixel end_col];
img  = imread('boston.tif','PixelRegion',{rows,cols});
figure
hImg = imshow(img);
title('2 meters per pixel')
```

Convert `XData` and `YData` to meters using conversion factor.

```
XDataInMeters = get(hImg,'XData')*meters_per_pixel;
YDataInMeters = get(hImg,'YData')*meters_per_pixel;
```

Set `XData` and `YData` of the image to reflect desired units.

```
set(hImg,'XData',XDataInMeters,'YData',YDataInMeters);
set(gca,'XLim',XDataInMeters,'YLim',YDataInMeters);
```

Specify the initial position of distance tool on Harvard Bridge.

```
hline = imdistline(gca,[271 471],[108 650]);
setLabelTextFormatter(hline,'%02.0f meters');
```

Tips

- If you use `imdistline` with an axes that contains an image object, and do not specify a drag constraint function, then you can drag the line outside the extent of the image. When used with an axes created by the `plot` function, the axes limits automatically expand to accommodate the movement of the line.
- You can also use the `Line` ROI object to create an interactive customizable distance tool. For an example, see “Measure Distances in an Image”.

See Also

`Line` | `drawline`

Introduced before R2006a

imellipse

(Not recommended) Create draggable ellipse

Note `imellipse` is not recommended. Use the `Ellipse` ROI object instead. You can also use the ROI creation convenience function `drawellipse`. If you used `imellipse` to create a circular ROI, use the `Circle` ROI object instead. For more information, see “Compatibility Considerations”.

Description

An `imellipse` object encapsulates an interactive ellipse over an image.

You can adjust the size and position of the ellipse by using the mouse. The ellipse also has a context menu that controls aspects of its appearance and behavior. For more information, see “Usage” on page 1-1831.

Creation

Syntax

```
h = imellipse
h = imellipse(hparent)
h = imellipse(hparent,position)
h = imellipse(___,"PositionConstraintFcn",fcn)
```

Description

`h = imellipse` begins interactive placement of an ellipse on the current axes, and returns an `imellipse` object.

`h = imellipse(hparent)` begins interactive placement of an ellipse on the object specified by `hparent`.

`h = imellipse(hparent,position)` creates a draggable ellipse at the position `position` on the object specified by `hparent`.

`h = imellipse(___,"PositionConstraintFcn",fcn)` also specifies where the ellipse can be dragged using a position constraint function, `fcn`.

Input Arguments

`hparent` — Handle to parent object

handle

Handle to parent object, specified as a handle. The parent is typically an axes object, but can also be any other object that can be the parent of an `hggroup` object.

`position` — Position of ellipse

4-element vector

Position of the ellipse as defined by a bounding rectangle, specified as a 4-element vector of the form `[xmin ymin width height]`. The initial size of the bounding rectangle is `width`-by-`height`, and the upper-left corner of the rectangle is at the (x,y) coordinate (`xmin`,`ymin`).

`fcn` — Position constraint function
function handle

Position constraint function, specified as a function handle. `fcn` is called whenever the mouse is dragged. You can use this function to control where the ellipse can be dragged. See the help for the `setPositionConstraintFcn` function for information about valid function handles.

Properties

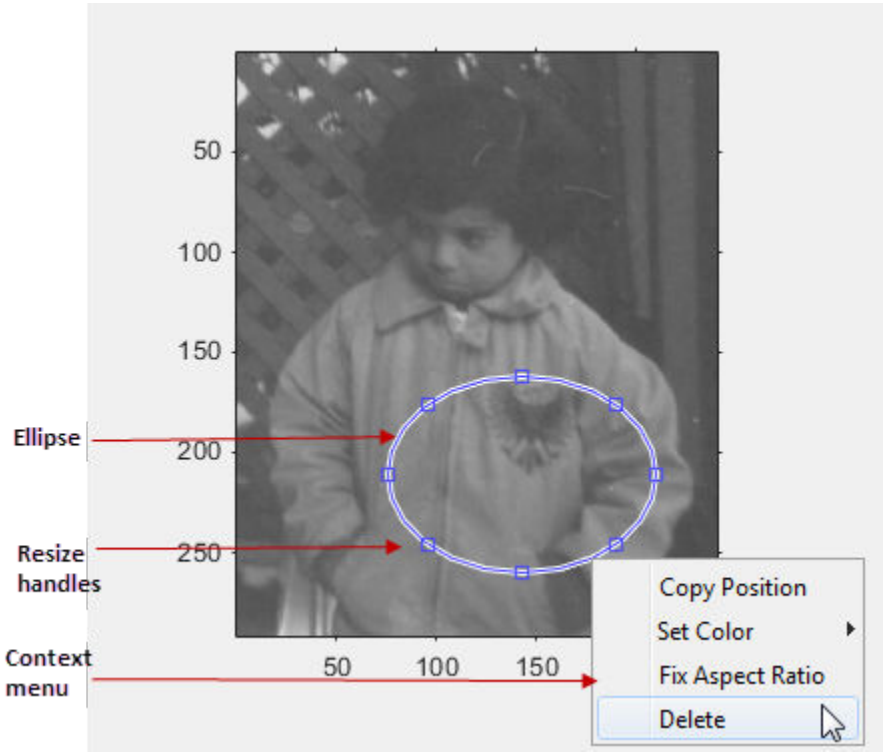
`Deletable` — ROI can be deleted
`true` (default) | `false`

ROI can be deleted, specified as `true` or `false`.

Data Types: `logical`

Usage

When you call `imellipse` with an interactive syntax, the pointer changes to a cross hairs when over an image. Click and drag the mouse to specify the size and position of the ellipse. The ellipse also supports a context menu that you can use to control aspects of its appearance and behavior. Right-click on the ellipse to access this context menu.



The table lists the interactive behavior supported by `imellipse`.

Interactive Behavior	Description
Moving the entire ellipse.	Move the pointer inside the ellipse. The pointer changes to a fleur shape  . Click and drag the mouse to move the ellipse.
Resizing the ellipse.	Move the pointer over a resizing handle on the ellipse. The pointer changes to a double-ended arrow shape  . Click and drag the mouse to resize the ellipse.
Changing the color used to display the ellipse.	Move the pointer inside the ellipse. Right-click and select **Set Color** from the context menu.
Retrieving the current position of the ellipse.	Move the pointer inside the ellipse. Right-click and select **Copy Position** from the context menu. `imellipse` copies a four-element position vector `[xmin ymin width height]` to the clipboard.
Preserving the current aspect ratio of the ellipse during resizing.	Move the pointer inside the ellipse. Right-click and select **Fix Aspect Ratio** from the context menu.
Deleting the ellipse	Move the pointer inside the ellipse. Right-click and select **Delete** from the context menu. To remove this option from the context menu, set the `Deletable` property to false: `h = imellipse(); h.Deletable = false;`

Object Functions

Each `imellipse` object supports a number of methods. Type `methods imellipse` to see a complete list.

addNewPositionCallback	Add new-position callback to ROI object
createMask	(Not recommended) Create mask within image
delete	Delete handle object
getColor	Get color used to draw ROI object
getPosition	Return current position of ROI object
getPositionConstraintFcn	Return function handle to current position constraint function
getVertices	Return vertices on perimeter of ellipse ROI object
removeNewPositionCallback	Remove new-position callback from ROI object
resume	(Not recommended) Resume execution of MATLAB command line
setColor	(Not recommended) Set color used to draw ROI object
setConstrainedPosition	Set ROI object to new position
setFixedAspectRatioMode	Preserve aspect ratio when resizing ROI object
setPosition	(Not recommended) Move ROI object to new position
setPositionConstraintFcn	Set position constraint function of ROI object
setResizable	Set resize behavior of ROI object
wait	(Not recommended) Block MATLAB command line until ROI creation is finished

Examples

Update Title when Ellipse Moves

Create an ellipse.

```
imshow("coins.png")
h = imellipse(gca,[10 10 100 100]);
```

Use callbacks to display the updated position in the title of the figure. The example illustrates using the `makeConstrainToRectFcn` to keep the ellipse inside the original `XLim` and `YLim` ranges.

```
addNewPositionCallback(h,@(p) title(mat2str(p,3)));
fcn = makeConstrainToRectFcn("imellipse",get(gca,"XLim"),get(gca,"YLim"));
setPositionConstraintFcn(h,fcn);
```

Click and Drag to Place Ellipse

Interactively place an ellipse by clicking and dragging. Use `wait` to block the MATLAB command line. Double-click on the ellipse to resume execution of the MATLAB command line.

```
imshow("coins.png")
h = imellipse;
position = wait(h);
```

Tips

If you use `imellipse` with an axes that contains an image object, and do not specify a position constraint function, users can drag the ellipse outside the extent of the image and lose the ellipse. When used with an axes created by the `plot` function, the axes limits automatically expand to accommodate the movement of the ellipse.

Compatibility Considerations

`imellipse` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

Replace use of the `imellipse` object with the new `Ellipse` ROI object. You can also use the ROI convenience function `drawellipse`.

Update ROI Creation Code

Update all instances of `imellipse`.

Discouraged Usage	Recommended Replacement

This example creates an ellipse ROI.
 	
`imshow("cameraman.tif");`
`h = imellipse(gca,[10 10 100 150]);`
	Here is roughly equivalent code, replacing the old ROI object with the new ROI object. Note that, with the previous ROIs, you defined an ellipse by the size of the bounding rectangle, `[x,y,width,height]`. With the new ROIs, you define an ellipse by specifying the center point of the ellipse and the length of the semi-major and semi-minor axes.
	
	`imshow("cameraman.tif");`
	`h = drawellipse(gca,"Center",[65 90],"Semi",[50 75]);`

Other ROI Code Updates

Update code that uses any of the object functions of the `imellipse` ROI object. In many cases, you can replace the call to an `imellipse` object function by simply accessing or setting the value of an `Ellipse` ROI object property. For example, replace calls to `getColor` or `setColor` with use of the `Color` property. In some cases, you must replace the `imellipse` object function with an object function of the new `Ellipse` ROI. Each of the individual `imellipse` ROI object methods include information about migrating to the new `Ellipse` ROI object. For a migration overview, see “ROI Migration”.

See Also

`Ellipse` | `drawellipse`

Topics

“Create ROI Shapes”
“ROI Migration”

Introduced in R2007b

imfreehand

(Not recommended) Create draggable freehand region

Note `imfreehand` is not recommended. Use the new `Freehand` ROI object instead. You can also use the new ROI creation convenience function `drawfreehand`. Another option is the `AssistedFreehand` object, which enables you to hand-draw a shape that automatically follows the edges in the underlying image. For more information, see “Compatibility Considerations”.

Description

An `imfreehand` object encapsulates an interactive freehand region over an image.

You can add vertices and adjust the size and position of the polygon by using the mouse. The polygon also has a context menu that controls aspects of its appearance and behavior. For more information, see “Usage” on page 1-1836.

Creation

Syntax

```
h = imfreehand
h = imfreehand(hparent)
h = imfreehand(___,Name,Value)
```

Description

`h = imfreehand` begins interactive placement of a freehand region on the current axes, and returns an `imfreehand` object.

`h = imfreehand(hparent)` begins interactive placement of a freehand region on the object specified by `hparent`.

`h = imfreehand(___,Name,Value)` specifies name-value pairs that control the behavior of the freehand region.

Input Arguments

`hparent` — Handle to parent object

handle

Handle to parent object, specified as a handle. The parent is typically an axes object, but can also be any other object that can be the parent of an `hggroup` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`PositionConstraintFcn` — Position constraint function
function handle

Position constraint function, specified as a function handle. `fcn` is called whenever the mouse is dragged. You can use this function to control where the freehand region can be dragged. See the help for the `setPositionConstraintFcn` function for information about valid function handles.

`Closed` — Freehand region is closed
`true` (default) | `false`

Freehand region is closed, specified as `true` or `false`. When set to `true` (the default), `imfreehand` draws a straight line to connect the endpoints of the freehand line to create a closed region. If set to `false`, `imfreehand` leaves the region open.

Data Types: `logical`

Properties

`Deletable` — ROI can be deleted
`true` (default) | `false`

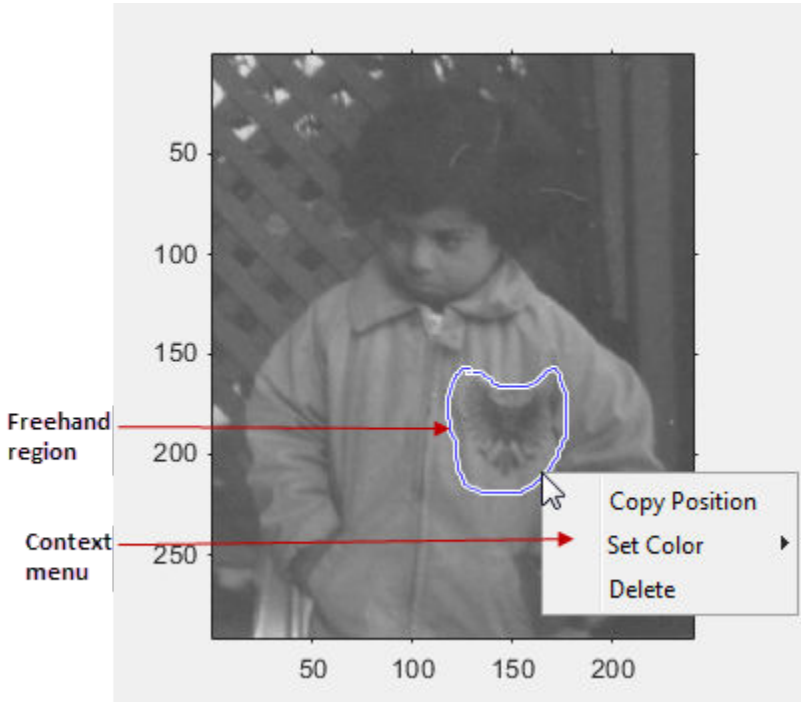
ROI can be deleted, specified as `true` or `false`.

Data Types: `logical`

Usage

When you call `imfreehand` with an interactive syntax, the pointer changes to a cross hairs  when positioned over an image. Click and drag the mouse to draw the freehand region and adjust the position of the region. By default, `imfreehand` draws a straight line connecting the last point you drew with the first point, but you can control this behavior using the `Closed` parameter.

The freehand region also supports a context menu that you can use to control aspects of its appearance and behavior.



The table lists the interactive features supported by `imfreehand`.

Interactive Behavior	Description
Moving the region.	Move the pointer inside the freehand region. The pointer changes to a fleur shape . Click and hold the left mouse button to move the region.
Changing the color used to draw the region.	Move the pointer inside the freehand region. Right-click and select **Set Color** from the context menu.
Retrieving the current position of the freehand region.	Move the pointer inside the freehand region. Right-click and select **Copy Position** from the context menu. `imfreehand` copies an *n*-by-2 array of coordinates on the boundary of the ROI to the clipboard.
Deleting the region	Move the pointer inside the region. Right-click and select **Delete** from the context menu. To remove this option from the context menu, set the `Deletable` property to false: `h = imfreehand(); h.Deletable = false;`

Object Functions

Each `imfreehand` object supports a number of methods. Type `methods imfreehand` to see a complete list.

addNewPositionCallback	Add new-position callback to ROI object
createMask	(Not recommended) Create mask within image
delete	Delete handle object
getColor	Get color used to draw ROI object
getPosition	Return current position of ROI object

getPositionConstraintFcn	Return function handle to current position constraint function
removeNewPositionCallback	Remove new-position callback from ROI object
resume	(Not recommended) Resume execution of MATLAB command line
setClosed	Set closure behavior of ROI object
setColor	(Not recommended) Set color used to draw ROI object
setConstrainedPosition	Set ROI object to new position
setPositionConstraintFcn	Set position constraint function of ROI object
wait	(Not recommended) Block MATLAB command line until ROI creation is finished

Examples

Click and Drag to Place Freehand Region

Interactively place a closed freehand region of interest by clicking and dragging over an image.

```
imshow("pout.tif")
h = imfreehand;
```

Interactively move the freehand region by clicking and dragging. Use the `wait` function to block the MATLAB command line. Double-click on the freehand region to resume execution of the MATLAB command line.

```
position = wait(h);
```

Tips

- If you use `imfreehand` with an axes that contains an image object, and do not specify a position constraint function, users can drag the freehand region outside the extent of the image and lose the freehand region. When used with an axes created by the `plot` function, the axes limits automatically expand to accommodate the movement of the freehand region.
- To cancel the interactive placement, press the Esc key. `imfreehand` returns an empty object.

Compatibility Considerations

`imfreehand` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

Replace use of the `imfreehand` ROI object with the `Freehand` or `AssistedFreehand` object. You can also use the ROI convenience functions `drawfreehand` or `drawassisted`.

Update ROI Creation Code

Update all instances of `imfreehand`.

Discouraged Usage	Recommended Replacement

This example enables drawing of a free-hand ROI.
 	
`imshow("cameraman.tif");`
`h = imfreehand`
	Here is equivalent code, replacing the old ROI object with the new ROI object. This example uses the ROI creation convenience function.
	
	`imshow("cameraman.tif");`
	`h = drawfreehand`

Other ROI Code Updates

Update code that uses any of the object functions of the `imfreehand` ROI object. In many cases, you can replace the call to an `imfreehand` object function by simply accessing or setting the value of a `Freehand` ROI object property. For example, replace calls to `getColor` or `setColor` with use of the `Color` property. In some cases, you must replace the `imfreehand` object function with an object function of the new `Freehand` ROI. Each of the individual `imfreehand` ROI object functions include information about migrating to the new `Freehand` ROI object. For a migration overview, see “ROI Migration”.

See Also

`Freehand` | `AssistedFreehand` | `drawassisted` | `drawfreehand`

Topics

“Create ROI Shapes”
“ROI Migration”

Introduced in R2007b

imline

(Not recommended) Create draggable, resizable line

Note `imline` is not recommended. Use the new `Line` ROI object instead. You can also use the ROI creation convenience function `drawline`. For more information, see "Compatibility Considerations".

Description

An `imline` object encapsulates an interactive line over an image.

You can adjust the size and position of the line by using the mouse. The line also has a context menu that controls aspects of its appearance and behavior. For more information, see "Usage" on page 1-1841.

Creation

Syntax

```
h = imline
h = imline(hparent)
h = imline(hparent,position)
h = imline(hparent,x,y)
h = imline( ___ ,"PositionConstraintFcn",fcn)
```

Description

`h = imline` begins interactive placement of a line on the current axes, and returns an `imline` object.

`h = imline(hparent)` begins interactive placement of a line on the object specified by `hparent`.

`h = imline(hparent,position)` creates a draggable, resizeable line, with coordinates defined by `position`.

`h = imline(hparent,x,y)` creates a draggable, resizeable line, with *x*- and *y*-coordinates of the endpoints defined by `x` and `y`.

`h = imline( ___ ,"PositionConstraintFcn",fcn)` also specifies where the line can be dragged using a position constraint function, `fcn`.

Input Arguments

`hparent` — Handle to parent object
handle

Handle to parent object, specified as a handle. The parent is typically an axes object, but can also be any other object that can be the parent of an `hggroup` object.

`position` — Position of line endpoints
2-element vector

Position of line endpoints, specified as a 2-by-2 array of the form `[x1 y1; x2 y2]`.

`x` — *x*-coordinates of line endpoints
2-element vector

x-coordinates of line endpoints, specified as a 2-element vector of the form `x = [x1 x2]`.

`y` — *y*-coordinates of line endpoints
2-element vector

y-coordinates of line endpoints, specified as a 2-element vector of the form `y = [y1 y2]`.

`fcn` — Position constraint function
function handle

Position constraint function, specified as a function handle. `fcn` is called whenever the mouse is dragged. You can use this function to control where the ellipse can be dragged. See the help for the `setPositionConstraintFcn` function for information about valid function handles.

Properties

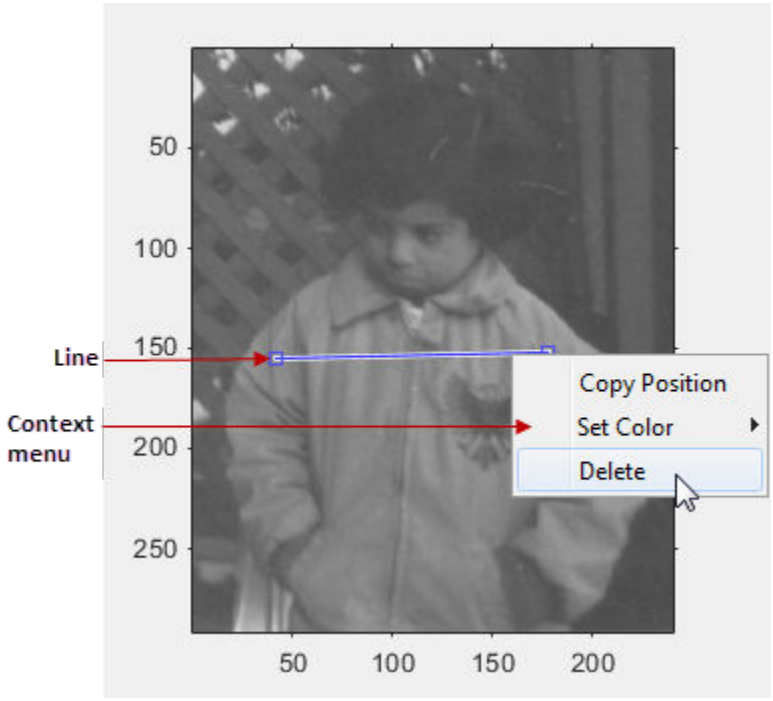
`Deletable` — ROI can be deleted
`true` (default) | `false`

ROI can be deleted, specified as `true` or `false`.

Data Types: `logical`

Usage

When you call `imline` with an interactive syntax, the pointer changes to a cross hairs  when over the image. Click and drag the mouse to specify the position and length of the line. The line supports a context menu that you can use to control aspects of its appearance and behavior.



The table describes the interactive behavior supported by `imline`.

Interactive Behavior	Description
Moving the line.	Move the pointer over the line. The pointer changes to a fleur shape  . Click and drag the mouse to move the line.
Moving the endpoints of the line.	Move the pointer over either end of the line. The pointer changes to the pointing finger,  . Click and drag the mouse to resize the line.
Changing the color used to display the line.	Move the pointer over the line. Right-click and select **Set Color** from the context menu.
Retrieving the coordinates of the endpoints of the line.	Move the pointer over the line. Right-click and select **Copy Position** from the context menu. `imline` copies a 2-by-2 array to the clipboard specifying the coordinates of the endpoints of the line in the form `[X1 Y1; X2 Y2]`.
Deleting the line	Move the pointer on top of the line. Right-click and select **Delete** from the context menu. To remove this option from the context menu, set the `Deletable` property to false: `h = imline(); h.Deletable = false;`

Object Functions

Each `imline` object supports a number of functions. Type `methods imline` to see a complete list.

addNewPositionCallback	Add new-position callback to ROI object
createMask	(Not recommended) Create mask within image
delete	Delete handle object
getColor	Get color used to draw ROI object

getPosition	Return current position of ROI object
getPositionConstraintFcn	Return function handle to current position constraint function
removeNewPositionCallback	Remove new-position callback from ROI object
resume	(Not recommended) Resume execution of MATLAB command line
setColor	(Not recommended) Set color used to draw ROI object
setConstrainedPosition	Set ROI object to new position
setPosition	(Not recommended) Move ROI object to new position
setPositionConstraintFcn	Set position constraint function of ROI object
wait	(Not recommended) Block MATLAB command line until ROI creation is finished

Examples

Update Title when Line Moves

Display a line in a custom color.

```
imshow("pout.tif")
h = imline(gca,[10 100],[100 100]);
setColor(h,[0 1 0]);
```

Use the `addNewPositionCallback` function. Move the line, note that the 2-by-2 position vector of the line is displayed in the title above the image. Explore the context menu of the line by right clicking on the line.

```
id = addNewPositionCallback(h,@(pos) title(mat2str(pos,3)));
```

After observing the callback behavior, remove the callback using the `removeNewPositionCallback` function.

```
removeNewPositionCallback(h,id);
```

Click and Drag to Place Line

Interactively place a line by clicking and dragging. Use `wait` to block the MATLAB command line. Double-click on the line to `resume` execution of the MATLAB command line.

```
imshow("pout.tif")
h = imline;
position = wait(h);
```

Tips

- If you use `imline` with an axes that contains an image object, and do not specify a position constraint function, users can drag the line outside the extent of the image and lose the line. When used with an axes created by the `plot` function, the axis limits automatically expand to accommodate the movement of the line.
- Use `imdistline` to create an interactive line with a text box that displays the distance between line endpoints.

Compatibility Considerations

`imline` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

Replace use of the `imline` ROI object with the new `Line` ROI object. You can also use the ROI creation convenience function `drawline`.

Update ROI Creation Code

Update all instances of `imline`.

Discouraged Usage	Recommended Replacement

This example creates a line ROI.
 	
`imshow("cameraman.tif");`
`h = imline(gca,[10 10; 100 150]);`
	Here is equivalent code, replacing the old ROI with the equivalent new ROI object. This example uses the ROI creation convenience function. Note that you must specify the size and position information as name-value arguments.
	
	`imshow("cameraman.tif");`
	`h = drawline(gca,"Position",[10 10; 100 150]);`

Other ROI Code Updates

Update instances where your code uses one of the object functions of `imline`. In many cases, you replace the call to an object function by simply accessing or setting the value of a `Line` ROI object property. For example, replace calls to `getColor` or `setColor` with use of the `Color` property. In some cases, you must replace the `imline` object function with an object function of the new `Line` ROI. Each of the individual `imline` ROI object functions include information about migrating to the new `Line` ROI object. For a migration overview, see “ROI Migration”.

See Also

`imdistline` | `Line` | `drawline`

Topics

“Create ROI Shapes”
“ROI Migration”

Introduced before R2006a

impoint

(Not recommended) Create draggable point

Note `impoint` is not recommended. Use the new `Point` ROI object instead. You can also use the new ROI creation convenience function `drawpoint`. For more information, see "Compatibility Considerations".

Description

An `impoint` object encapsulates an interactive point over an image.

You can adjust the position of the point by using the mouse. The point also has a context menu that controls aspects of its appearance and behavior. For more information, see "Usage" on page 1-1846.

Creation

Syntax

```
h = impoint
h = impoint(hparent)
h = impoint(hparent,position)
h = impoint(hparent,x,y)
h = impoint( ___ ,"PositionConstraintFcn",fcn)
```

Description

`h = impoint` begins interactive placement of a point on the current axes, and returns an `impoint` object.

`h = impoint(hparent)` begins interactive placement of a point on the object specified by `hparent`.

`h = impoint(hparent,position)` creates a draggable point with coordinates defined by `position`.

`h = impoint(hparent,x,y)` creates a draggable point with (*x*, *y*) coordinates defined by `x` and `y`.

`h = impoint( ___ ,"PositionConstraintFcn",fcn)` also specifies where the point can be dragged using a position constraint function, `fcn`.

Input Arguments

`hparent` — Handle to parent object

handle

Handle to parent object, specified as a handle. The parent is typically an axes object, but can also be any other object that can be the parent of an `hggroup` object.

`position` — Position of point
2-element vector

Position of point, specified as a 2-element vector of the form `[x y]`.

`x` — *x*-coordinate of point
numeric scalar

x-coordinate of the point, specified as a numeric scalar.

`y` — *y*-coordinate of point
numeric scalar

y-coordinate of the point, specified as a numeric scalar.

`fcn` — Position constraint function
function handle

Position constraint function, specified as a function handle. `fcn` is called whenever the mouse is dragged. You can use this function to control where the ellipse can be dragged. See the help for the `setPositionConstraintFcn` function for information about valid function handles.

Properties

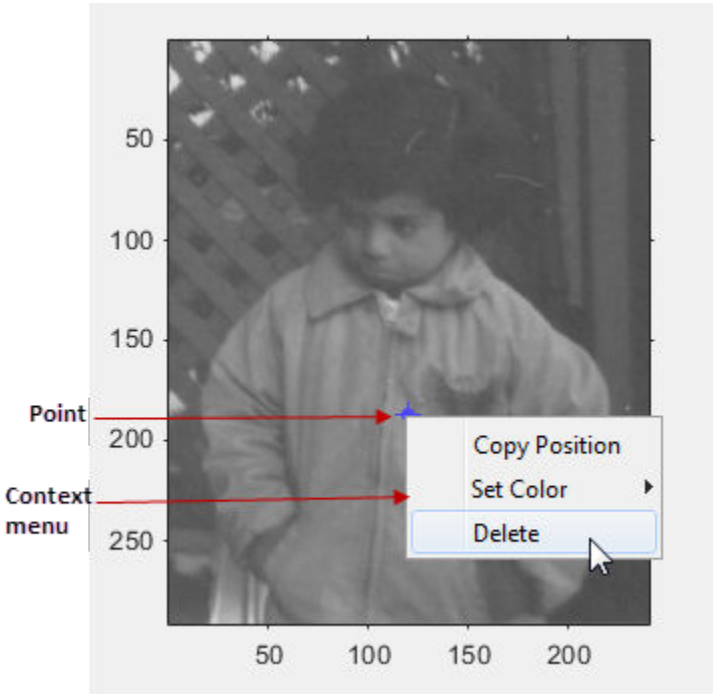
`Deletable` — ROI can be deleted
`true` (default) | `false`

ROI can be deleted, specified as `true` or `false`.

Data Types: `logical`

Usage

When you call `impoint` with an interactive syntax, the pointer changes to a cross hairs  when over the image. Click and drag the mouse to specify the position of the point. The point supports a context menu that you can use to control aspects of its appearance and behavior.



The table describes the interactive behavior supported by `impoint`.

Interactive Behavior	Description
Moving the point.	Move the mouse pointer over the point. The mouse pointer changes to a fleur shape  . Click and drag the mouse to move the point.
Changing the color used to display the point.	Move the mouse pointer over the point. Right-click and select **Set Color** from the context menu and specify the color you want to use.
Retrieving the coordinates of the point.	Move the mouse pointer over the point. Right-click and select **Copy Position** from the context menu to copy a 1-by-2 array to the clipboard specifying the coordinates of the point `[X Y]`.
Deleting the point	Move the pointer on top of the point. Right-click and select **Delete** from the context menu. To remove this option from the context menu, set the `Deletable` property to false: `h = impoint(); h.Deletable = false;`

Object Functions

Each `impoint` object supports a number of functions. Type `methods impoint` to see a complete list.

addNewPositionCallback	Add new-position callback to ROI object
createMask	(Not recommended) Create mask within image
delete	Delete handle object
getColor	Get color used to draw ROI object
getPosition	Return current position of ROI object
getPositionConstraintFcn	Return function handle to current position constraint function
removeNewPositionCallback	Remove new-position callback from ROI object

resume	(Not recommended) Resume execution of MATLAB command line
setColor	(Not recommended) Set color used to draw ROI object
setConstrainedPosition	Set ROI object to new position
setPosition	(Not recommended) Move ROI object to new position
setPositionConstraintFcn	Set position constraint function of ROI object
setString	Set text label for point ROI object
wait	(Not recommended) Block MATLAB command line until ROI creation is finished

Examples

Enforce Boundary Constraint when Point Moves

Use `impoint` functions to set custom color, set a label, enforce a boundary constraint, and update position in title as point moves.

```
imshow("rice.png")
h = impoint(gca,100,200);
```

Update the title with the new position by using `addNewPositionCallback`.

```
addNewPositionCallback(h,@(h) title(sprintf('(%1.0f,%1.0f)',h(1),h(2))));
```

Construct a boundary constraint function by using `makeConstrainToRectFcn`.

```
fcn = makeConstrainToRectFcn("impoint",get(gca,"XLim"),get(gca,"YLim"));
```

Enforce the boundary constraint function using `setPositionConstraintFcn`.

```
setPositionConstraintFcn(h,fcn);
setColor(h,"r");
setString(h,"Point label");
```

Click and Drag to Move Point

Interactively place a point. Use `wait` to block the MATLAB command line. Double-click on the point to `resume` execution of the MATLAB command line

```
imshow("pout.tif")
h = impoint(gca,[]);
position = wait(h);
```

Tips

If you use `impoint` with an axes that contains an image object, and do not specify a drag constraint function, then users can drag the point outside the extent of the image and lose the point. When used with an axes created by the `plot` function, the axes limits automatically expand to accommodate the movement of the point.

Compatibility Considerations

`impoint` is not recommended
Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

Replace use of the `impoint` ROI object with the new `Point` ROI object. You can also use the ROI creation convenience function `drawpoint`.

Update ROI Creation Code

Update all instances of `impoint`.

Discouraged Usage	Recommended Replacement

This example creates a point ROI.
 	
`imshow("cameraman.tif");`
`h = impoint(gca,[25 50]);`
	Here is equivalent code, replacing the old ROI object with the new ROI object. This example uses the ROI creation convenience function. Note that you must specify the size and position information as a name-value argument.
	
	`imshow("cameraman.tif");`
	`h = drawpoint(gca,"Position",[25 50]);`

Other ROI Code Updates

Update code that uses any of the object functions of the `impoint` ROI object. In many cases, you can replace the call to an `impoint` object function by simply accessing or setting the value of a `Point` ROI object property. For example, replace calls to `getColor` or `setColor` with use of the `Color` property. In some cases, you must replace the `impoint` object function with an object function of the new `Point` ROI. Each of the individual `impoint` ROI object functions include information about migrating to the new `Point` ROI object. For a migration overview, see “ROI Migration”.

See Also

`Point` | `drawpoint`

Topics

“Create ROI Shapes”
“ROI Migration”

Introduced before R2006a

impoly

(Not recommended) Create draggable, resizable polygon

Note `impoly` is not recommended. Use the new `Polygon` object instead. You can also use the new ROI creation convenience function `drawpolygon`. Another option is the `Polyline` object, which enables you to create an open polygon, or polyline shape. For more information, see "Compatibility Considerations".

Description

An `impoly` object encapsulates an interactive polygon over an image.

You can add vertices and adjust the size and position of the polygon by using the mouse. The polygon also has a context menu that controls aspects of its appearance and behavior. For more information, see "Usage" on page 1-1851.

Creation

Syntax

```
h = impoly
h = impoly(hparent)
h = impoly(hparent,position)
h = impoly( ___ ,Name,Value)
```

Description

`h = impoly` begins interactive placement of a polygon on the current axes, and returns an `impoly` object.

`h = impoly(hparent)` begins interactive placement of a polygon on the object specified by `hparent`.

`h = impoly(hparent,position)` creates a draggable, resizeable polygon with vertices at coordinates defined by `position`.

`h = impoly( ___ ,Name,Value)` specifies name-value pairs that control the behavior of the polygon.

Input Arguments

`hparent` — Handle to parent object
handle

Handle to parent object, specified as a handle. The parent is typically an axes object, but can also be any other object that can be the parent of an `hggroup` object.

`position` — Position of polygon vertices

n-by-2 matrix

Position of polygon vertices, specified as an *n*-by-2 matrix. The two columns define the *x*- and *y*-coordinate, respectively, of each of the *n* vertices.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`PositionConstraintFcn` — Position constraint function

function handle

Position constraint function, specified as a function handle. `fcn` is called whenever the mouse is dragged. You can use this function to control where the polygon can be dragged. See the help for the `setPositionConstraintFcn` function for information about valid function handles.

`Closed` — Polygon is closed

`true` (default) | `false`

Polygon is closed, specified as `true` or `false`. When set to `true` (the default), `impoly` creates a closed polygon, that is, it draws a straight line between the last vertex specified and the first vertex specified to create a closed region. When `Closed` is `false`, `impoly` does not connect the last vertex with the first vertex, creating an open polygon (or polyline).

Data Types: `logical`

Properties

`Deletable` — ROI can be deleted

`true` (default) | `false`

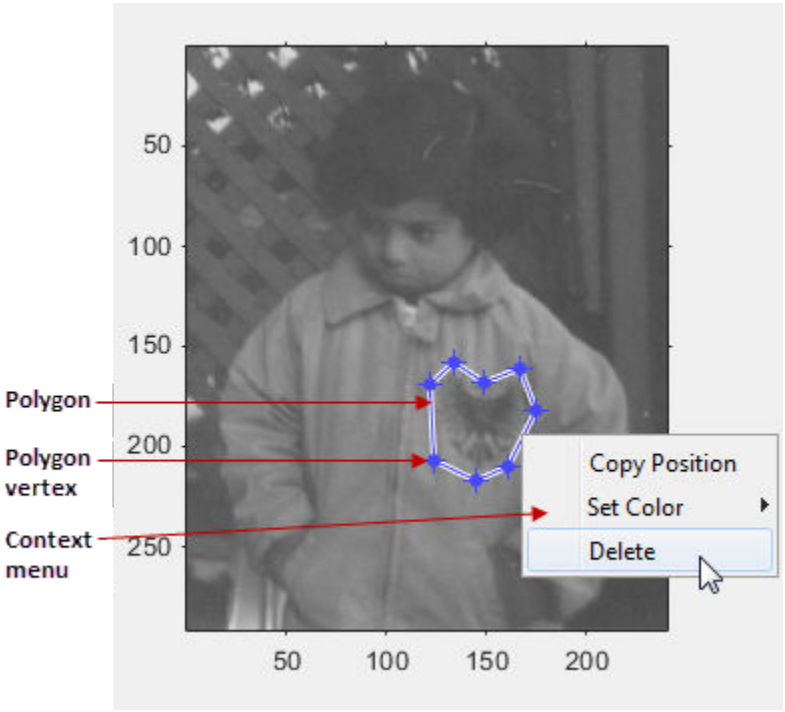
ROI can be deleted, specified as `true` or `false`.

Data Types: `logical`

Usage

When you call `impoly` with an interactive syntax, the pointer changes to a cross hairs  when over the image. Click and drag the mouse to define the vertices of the polygon and adjust the size, shape, and position of the polygon. By default, `impoly` draws a straight line connecting the last point you drew with the first point, but you can control this behavior using the `Closed` parameter.

The polygon also supports a context menu that you can use to control aspects of its appearance and behavior. The choices in the context menu vary whether you position the pointer on an edge of the polygon (or anywhere inside the region) or on one of the vertices. The figure shows the context menu when the pointer is on the polygon but not on a vertex.



The table lists the interactive behaviors supported by `impoly`.

Interactive Behavior	Description
Closing the polygon.	Use any of the following mechanisms: • Move the pointer over the initial vertex of the polygon that you selected. The pointer changes to a circle ○. Click either mouse button. • Double-click the left mouse button. This action creates a vertex at the point under the mouse and draws a straight line connecting this vertex with the initial vertex. • Click the right mouse button. This action draws a line connecting the last vertex selected with the initial vertex; it does not create a new vertex.
Adding a new vertex.	Move the pointer over an edge of the polygon. Press and hold the **A** key. The shape of the pointer changes ✦. Click the left mouse button to create a new vertex at that position on the line.
Moving a vertex. (Reshaping the polygon.)	Move the pointer over a vertex. The pointer changes to a circle ○. Click and drag the vertex to its new position.
Deleting a vertex.	Move the pointer over a vertex. The shape changes to a circle ○. Right-click and select **Delete Vertex** from the vertex context menu. This action deletes the vertex and adjusts the shape of the polygon, drawing a new straight line between the two vertices that were neighbors of the deleted vertex.

Interactive Behavior	Description
Deleting the polygon	Move the pointer inside the polygon or on one of the lines that define the polygon, not on a vertex. Right-click and select **Delete** from the context menu. To remove this option from the context menu, set the `Deletable` property to false: `h = impoly(); h.Deletable = false;`
Moving the polygon.	Move the pointer inside the polygon. The pointer changes to a fleur shape  . Click and drag the mouse to move the polygon.
Changing the color of the polygon	Move the pointer inside the polygon. Right-click and select **Set Color** from the context menu.
Retrieving the coordinates of the vertices	Move the pointer inside the polygon. Right-click and select **Copy Position** from the context menu. `impoly` copies an *n*-by-2 array containing the *x*- and *y*-coordinates of each vertex to the clipboard. *n* is the number of vertices you specified.

Object Functions

Each `impoly` object supports a number of methods. Type `methods impoly` to see a complete list.

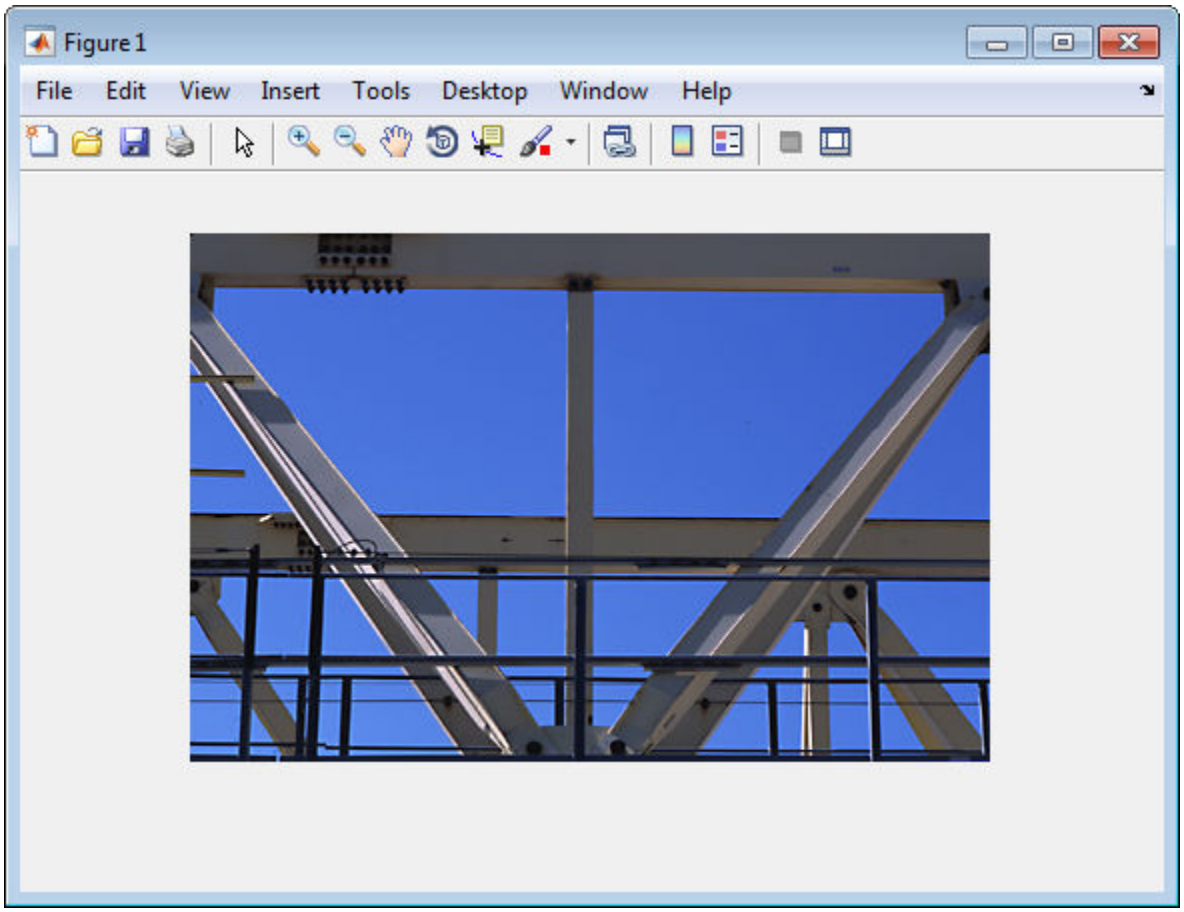
addNewPositionCallback	Add new-position callback to ROI object
createMask	(Not recommended) Create mask within image
delete	Delete handle object
getColor	Get color used to draw ROI object
getPosition	Return current position of ROI object
getPositionConstraintFcn	Return function handle to current position constraint function
removeNewPositionCallback	Remove new-position callback from ROI object
resume	(Not recommended) Resume execution of MATLAB command line
setClosed	Set closure behavior of ROI object
setColor	(Not recommended) Set color used to draw ROI object
setConstrainedPosition	Set ROI object to new position
setPosition	(Not recommended) Move ROI object to new position
setPositionConstraintFcn	Set position constraint function of ROI object
setVerticesDraggable	Set vertex behavior of ROI object
wait	(Not recommended) Block MATLAB command line until ROI creation is finished

Examples

Draw Polygon on Image and Specify Position Constraint Function

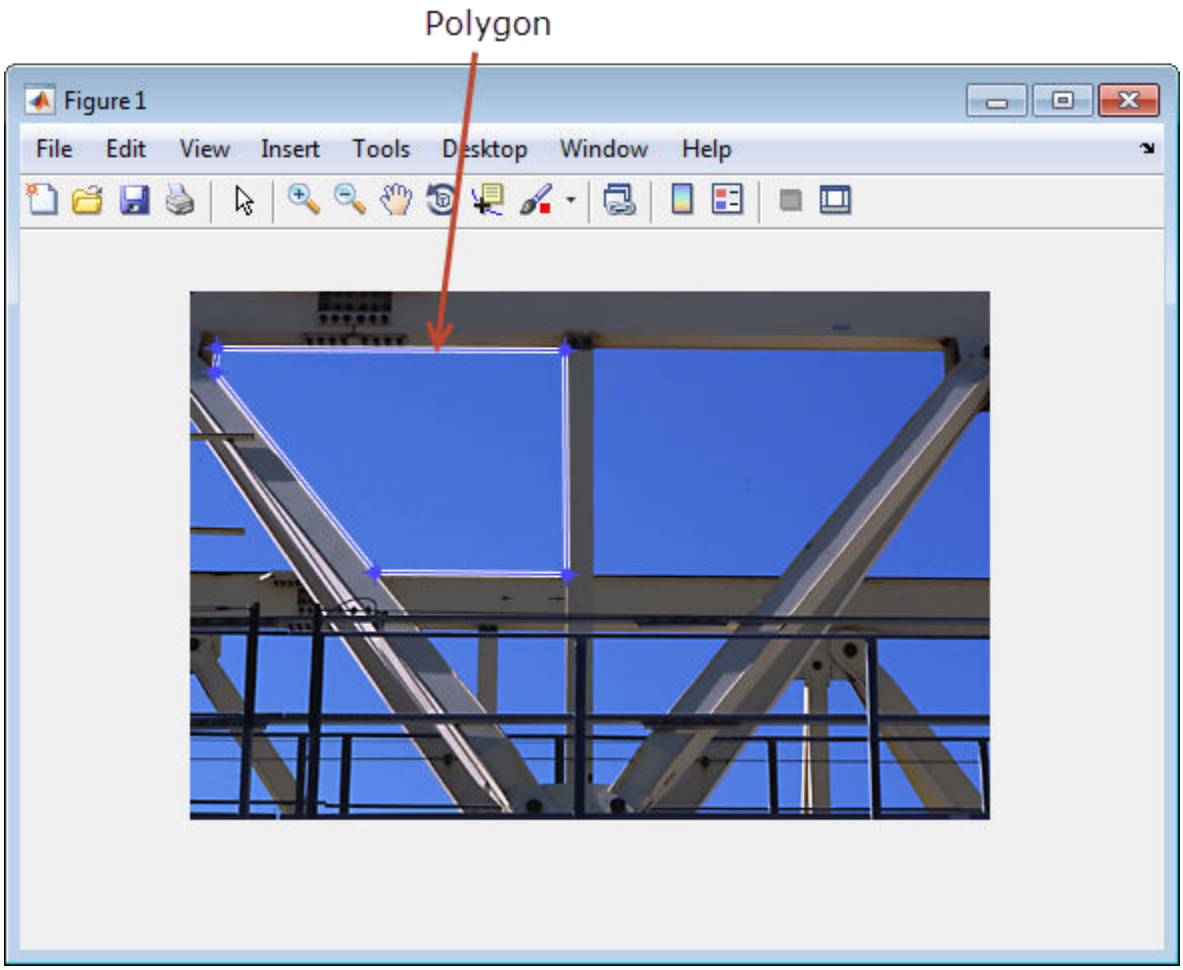
Display an image.

```
imshow("gantrycrane.png")
```



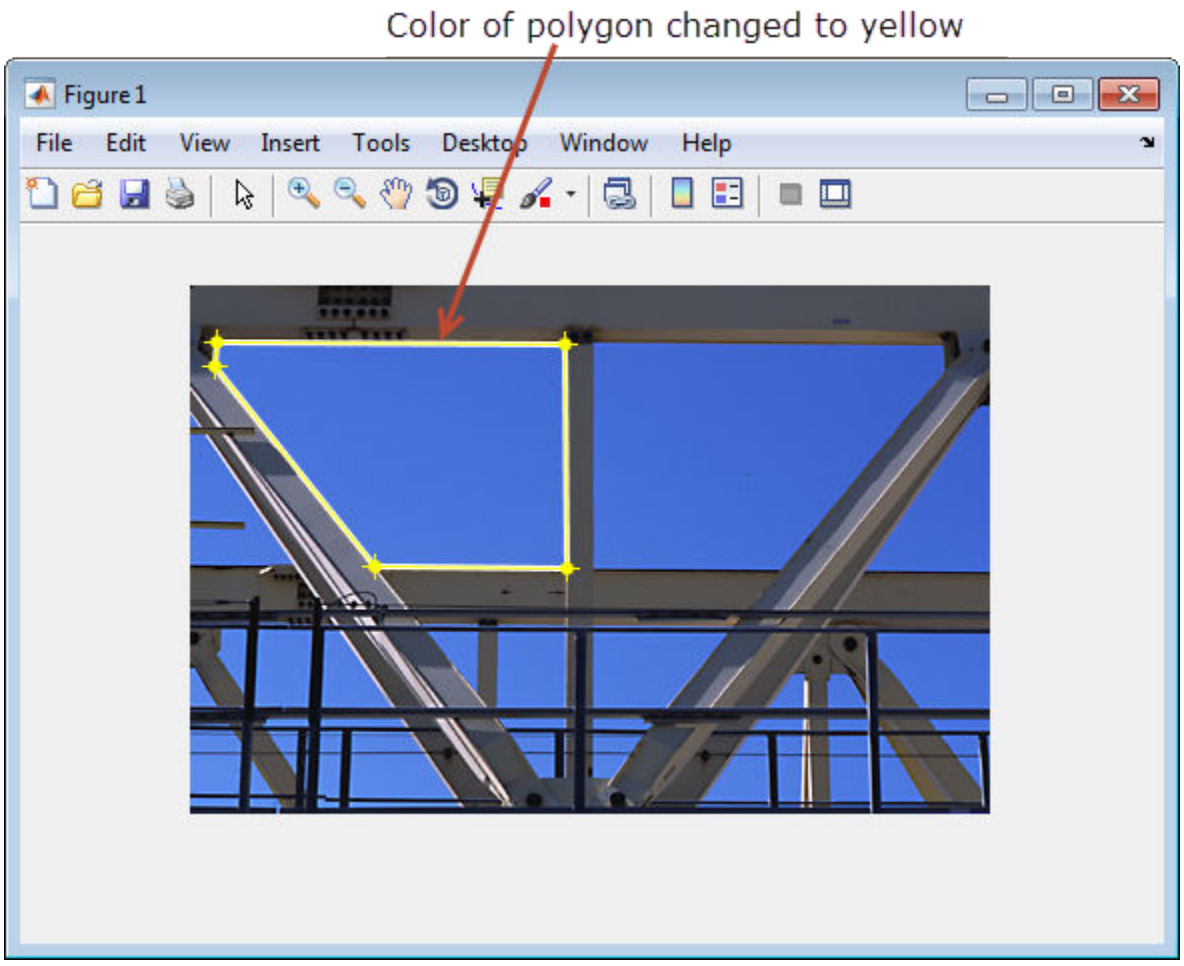
Draw a polygon on the image, specifying the location of five vertices.

```
h = impoly(gca,[188,30; 189,142; 93,141; 13,41; 14,29]);
```



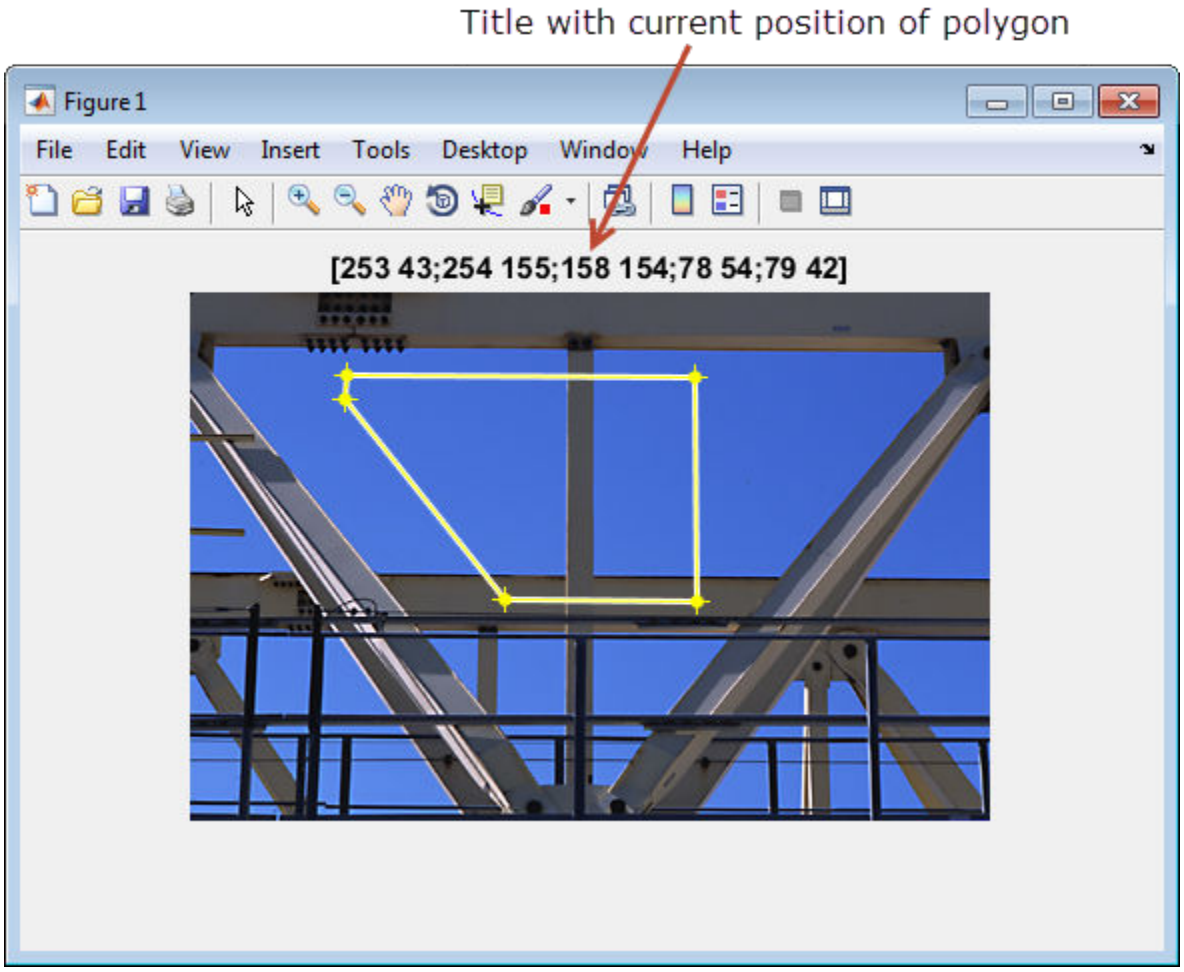
Set the color of the polygon to yellow.

```
setColor(h,"yellow");
```



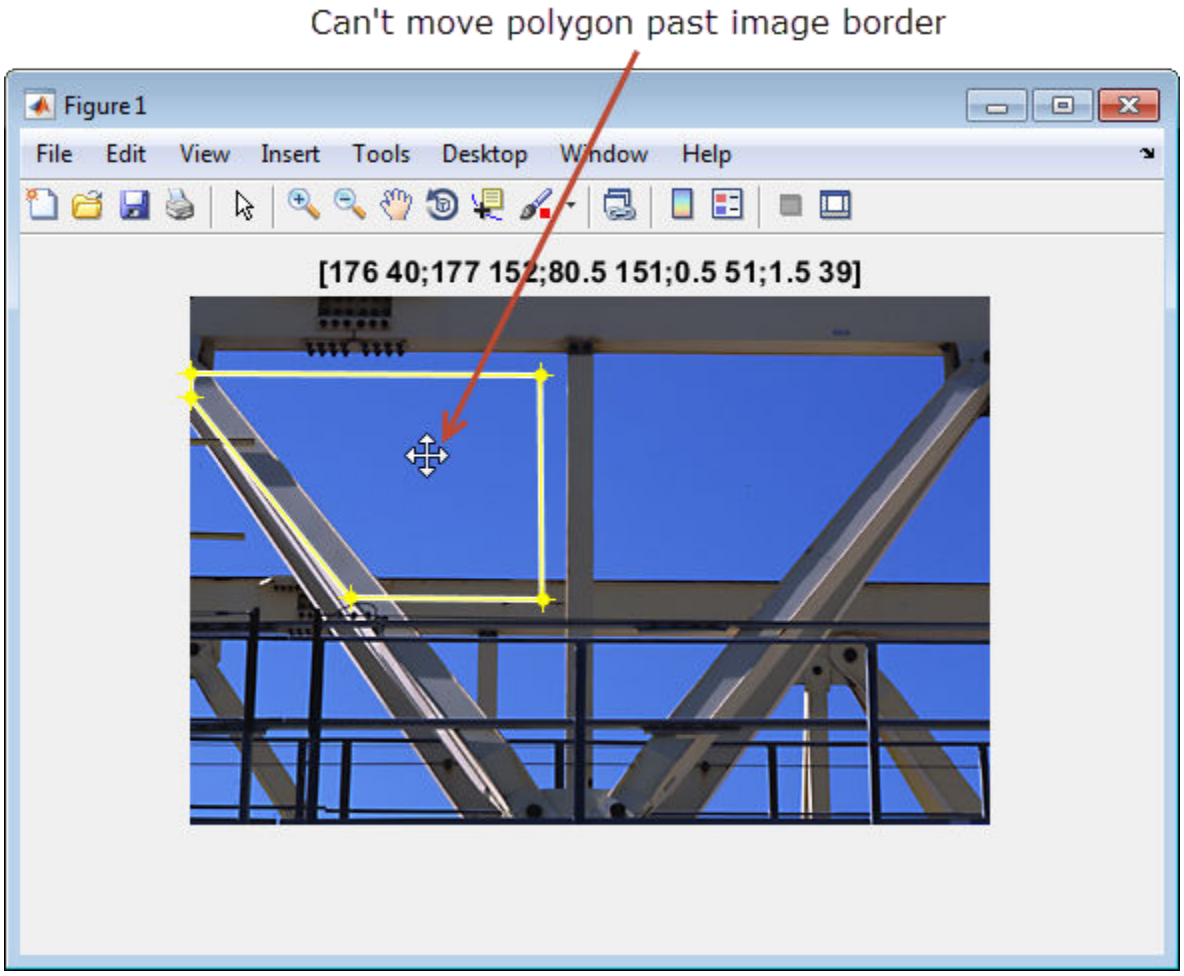
Define a function for the new position callback. This function displays the current position of the polygon whenever it is moved.

```
addNewPositionCallback(h, @(p) title(mat2str(p,3)));
```



Create the function that constrains the movement of the polygon by using `makeConstrainToRectFcn`, specifying the boundary of the image as the limits. Enforce the boundary constraint function using `setPositionConstraintFcn`.

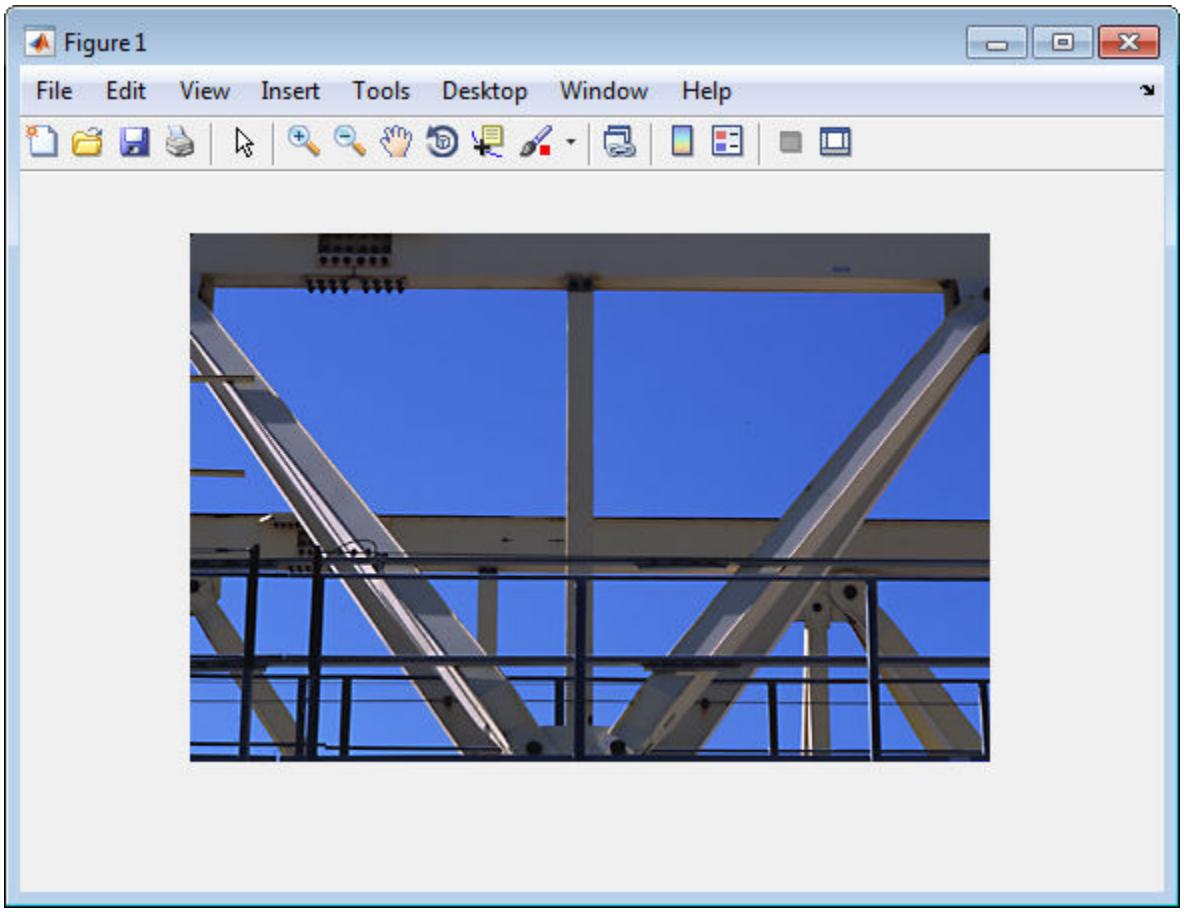
```
fcn = makeConstrainToRectFcn("impoly",get(gca,"XLim"),get(gca,"YLim"));
setPositionConstraintFcn(h,fcn);
```



Interactively Create a Polygon by Clicking to Specify Vertex Locations

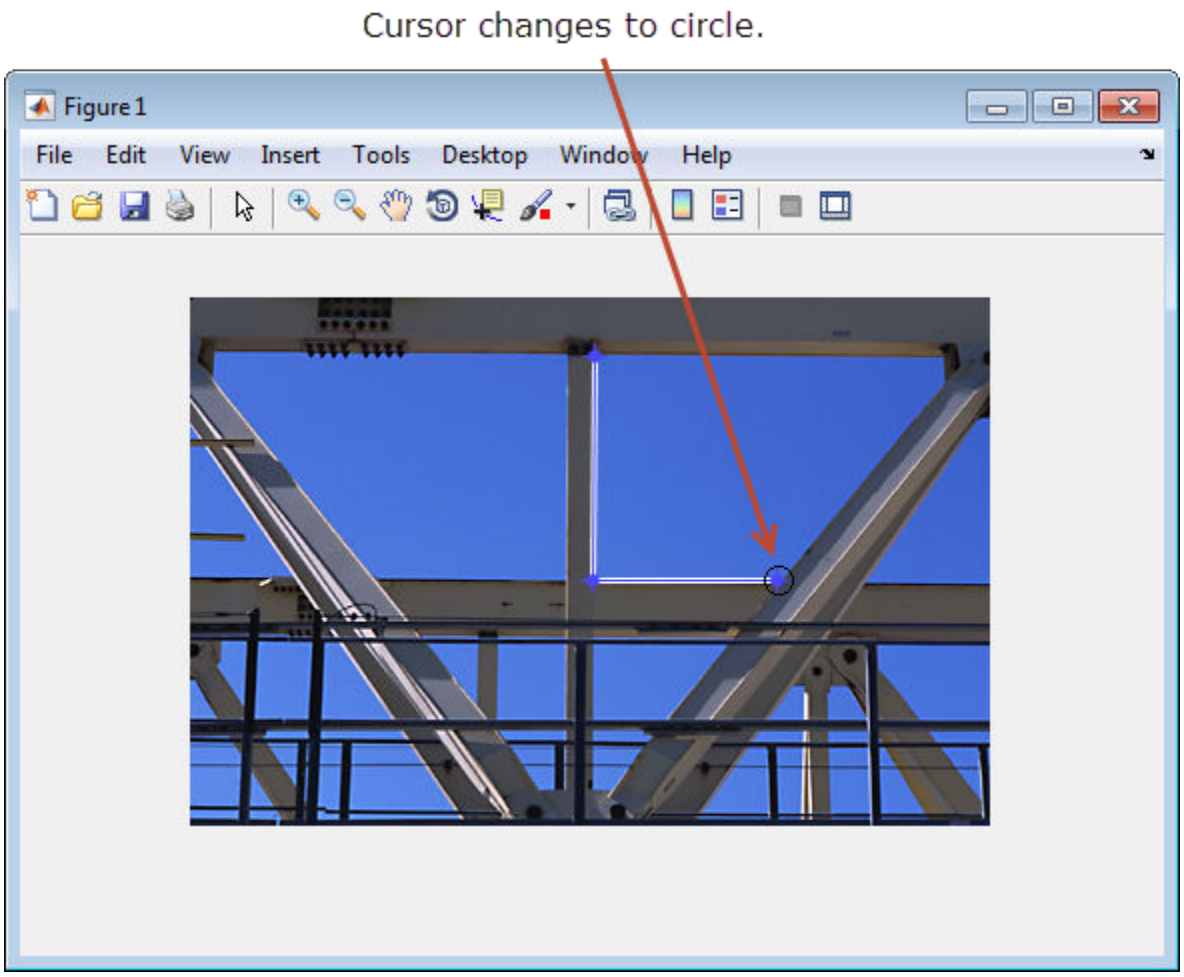
Display image.

```
figure
imshow("gantrycrane.png");
```

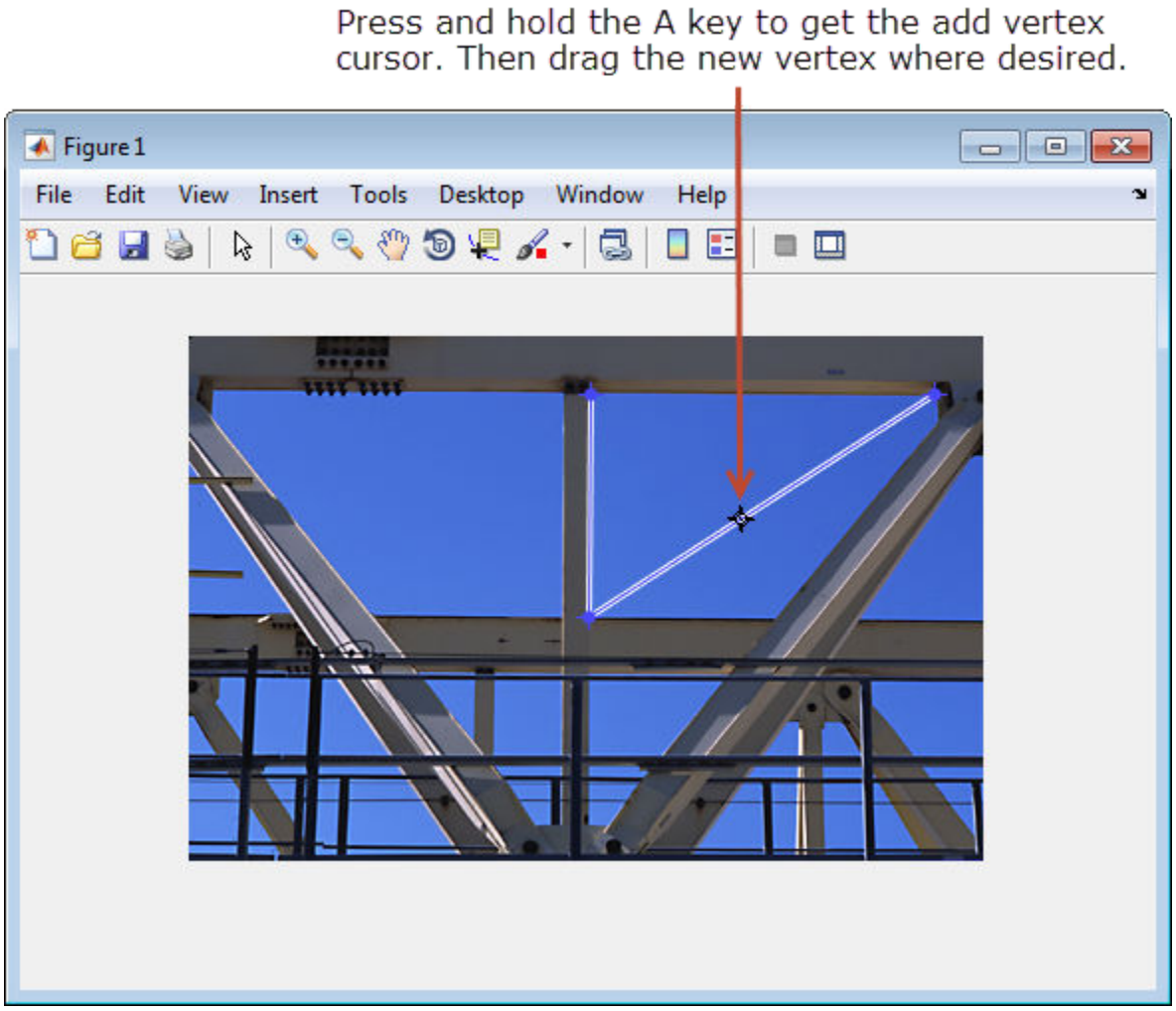


Create a polygon, specifying several vertices, but leave it unfinished so that you can finish it interactively. The example sets `Closed` to `false` so that the polygon is left open. When you move the cursor over one of the endpoints of the polygon, the cursor shape changes to a circle.

```
h = impoly(gca,[203,30; 202,142; 294,142],"Closed",false);
```



Complete the polygon. Grab one of the ends of the existing lines. Extend the line by dragging it to another corner of the shape you want to create. Then, while positioning the cursor over the line, press and hold the **A** key to add a vertex to the line. Once you create the vertex you can drag it anywhere you want to create the shape you want. Continue dragging the line and adding vertices as you want. For more information, see “Usage” on page 1-1851.



Tips

If you use `impoly` with an axes that contains an image object, and do not specify a position constraint function, users can drag the polygon outside the extent of the image and lose the polygon. When used with an axes created by the `plot` function, the axes limits automatically expand when the polygon is dragged outside the extent of the axes.

Compatibility Considerations

`impoly` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

Replace use of the `impoly` object with the `Polygon` or `Polyline` object. You can also use the `drawpolygon` or `drawpolyline` function to create an these objects.

Update ROI Creation Code

Update all instances of `impoly`.

Discouraged Usage	Recommended Replacement

This example creates a closed polygonal ROI.
 	
`imshow("cameraman.tif");`
`h = impoly(gca,[10 10; 10 100; 80 100]);`
	Here is equivalent code, replacing the old ROI object with the new ROI object. This example uses the ROI creation convenience function. Note that you must specify the position information as a name-value argument.
	
	`imshow("cameraman.tif");`
	`h = drawpolygon(gca,"Position",[10 10; 10 100; 80 100`

Other ROI Code Updates

Update code that uses any of the object methods of the `impoly` ROI object. In many cases, you can replace the call to an `impoly` object method by simply accessing or setting the value of a `Polygon` ROI object property. For example, replace calls to `getColor` or `setColor` with use of the `Color` property. In some case, you must replace the `impoly` object method with an object method of the new ROI. The documentation for each `impoly` ROI object method includes information about migrating to the new ROI object. For a migration overview, see “ROI Migration”.

See Also

`Polygon` | `Polyline` | `drawpolygon` | `drawpolyline`

Topics

“Create ROI Shapes”
“ROI Migration”

Introduced in R2007b

imrect

(Not recommended) Create draggable rectangle

Note `imrect` is not recommended. Use the new `Rectangle` ROI object instead. You can also use the new ROI convenience function `drawrectangle`. For more information, see "Compatibility Considerations".

Description

An `imrect` object encapsulates an interactive rectangle over an image.

You can adjust the size and position of the rectangle by using the mouse. The rectangle also has a context menu that controls aspects of its appearance and behavior. For more information, see "Usage" on page 1-1864.

Creation

Syntax

```
h = imrect
h = imrect(hparent)
h = imrect(hparent,position)
h = imrect( ___ ,"PositionConstraintFcn",fcn)
```

Description

`h = imrect` begins interactive placement of a rectangle on the current axes, and returns an `imrect` object.

`h = imrect(hparent)` begins interactive placement of a rectangle on the object specified by `hparent`.

`h = imrect(hparent,position)` creates a draggable rectangle at the position `position` on the object specified by `hparent`.

`h = imrect( ___ ,"PositionConstraintFcn",fcn)` also specifies where the rectangle can be dragged using a position constraint function, `fcn`.

Input Arguments

`hparent` — Handle to parent object

handle

Handle to parent object, specified as a handle. The parent is typically an axes object, but can also be any other object that can be the parent of an `hggroup` object.

`position` — Position of rectangle

4-element vector

Position of the rectangle, specified as a 4-element vector of the form `[xmin ymin width height]`. The initial size of the rectangle is `width`-by-`height`, and the upper-left corner of the rectangle is at the (x,y) coordinate (`xmin`,`ymin`).

`fcn` — Position constraint function
function handle

Position constraint function, specified as a function handle. `fcn` is called whenever the mouse is dragged. You can use this function to control where the ellipse can be dragged. See the help for the `setPositionConstraintFcn` function for information about valid function handles.

Properties

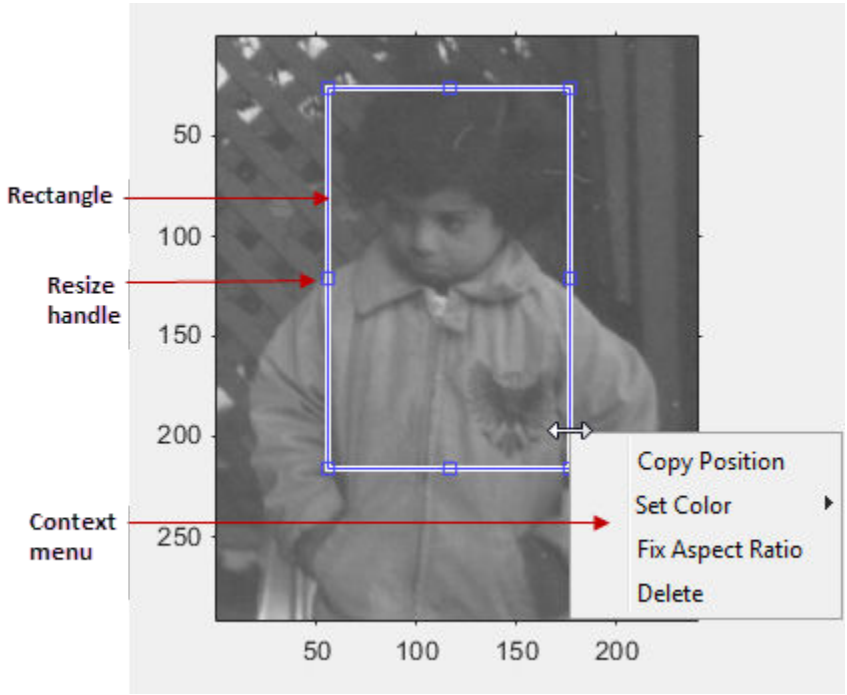
`Deletable` — ROI can be deleted
`true` (default) | `false`

ROI can be deleted, specified as `true` or `false`.

Data Types: `logical`

Usage

When you call `imrect` with an interactive syntax, the pointer changes to a cross hairs when over the image. You can create the rectangle and adjust its size and position using the mouse. The rectangle also supports a context menu that you can use to control aspects of its appearance and behavior. Right-click on the rectangle to access this context menu.



The table lists the interactive behaviors supported by `imrect`.

Interactive Behavior	Description
Moving the rectangle.	Move the pointer inside the rectangle. The pointer changes to a fleur shape  . Click and drag the mouse to move the rectangle.
Resizing the rectangle.	Move the pointer over any of the edges or corners of the rectangle, the shape changes to a double-ended arrow,  . Click and drag the edge or corner using the mouse.
Changing the color of the rectangle.	Move the pointer inside the rectangle. Right-click and select **Set Color** from the context menu.
Retrieving the coordinates of the current position	Move the pointer inside the polygon. Right-click and select **Copy Position** from the context menu. `imrect` copies a four-element position vector to the clipboard.
Preserve the current aspect ratio of the rectangle during interactive resizing.	Move the pointer inside the rectangle. Right-click and select **Fix Aspect Ratio** from the context menu.
Deleting the rectangle	Move the pointer inside the rectangle or on an edge of the rectangle. Right-click and select **Delete** from the context menu. To remove this option from the context menu, set the `Deletable` property to false: `h = imrect(); h.Deletable = false;`

When you use `setResizeable` to make the rectangle non-resizable, the **Fix Aspect Ratio** context menu item is not provided.

Object Functions

Each `imrect` object supports a number of functions. Type `methods imrect` to see a complete list.

addNewPositionCallback	Add new-position callback to ROI object
createMask	(Not recommended) Create mask within image
delete	Delete handle object
getColor	Get color used to draw ROI object
getPosition	Return current position of ROI object
getPositionConstraintFcn	Return function handle to current position constraint function
removeNewPositionCallback	Remove new-position callback from ROI object
resume	(Not recommended) Resume execution of MATLAB command line
setColor	(Not recommended) Set color used to draw ROI object
setConstrainedPosition	Set ROI object to new position
setFixedAspectRatioMode	Preserve aspect ratio when resizing ROI object
setPosition	(Not recommended) Move ROI object to new position
setPositionConstraintFcn	Set position constraint function of ROI object
setResizable	Set resize behavior of ROI object
wait	(Not recommended) Block MATLAB command line until ROI creation is finished

Examples

Update Title when Rectangle Moves

Display a rectangle ROI over an image. Display the position of the rectangle in the title. The title updates when you move the rectangle. Try dragging one side of the rectangle outside the boundary of the image.

```
imshow("cameraman.tif")
h = imrect(gca,[10 10 100 100]);
addNewPositionCallback(h,@(p) title(mat2str(p,3)));
```

Specify a position constraint function using `makeConstrainToRectFcn` to keep the rectangle inside the original `XLim` and `YLim` ranges of the image.

```
fcn = makeConstrainToRectFcn("imrect",get(gca,"XLim"),get(gca,"YLim"));
setPositionConstraintFcn(h,fcn);
```

Now drag the rectangle using the mouse. Observe that the rectangle can no longer extend past the image boundary.

Click and Drag to Place Rectangle

Interactively place a rectangle by clicking and dragging. Use `wait` to block the MATLAB command line. Double-click on the rectangle to resume execution of the MATLAB command line.

```
imshow("pout.tif");
h = imrect;
position = wait(h);
```

Tips

If you use `imrect` with an axes that contains an image object, and do not specify a position constraint function, users can drag the rectangle outside the extent of the image. When used with an axes created by the `plot` function, the axes limits automatically expand to accommodate the movement of the rectangle.

Compatibility Considerations

`imrect` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

Replace use of the `imrect` ROI object with the new `Rectangle` ROI object. You can also use the ROI creation convenience function `drawrectangle`.

Update ROI Creation Code

Update all instances of `imrect`.

Discouraged Usage	Recommended Replacement

This example creates a rectangular ROI.
 	
`imshow("cameraman.tif");`
`h = imrect(gca,[10 10 100 150]);`
	Here is equivalent code, replacing the old ROI object with the new ROI object. This example uses the ROI creation convenience function. Note that you must specify the size and position information as a name-value argument.
	
	`imshow("cameraman.tif");`
	`h = drawrectangle(gca,"Position",[10 10 100 150]);`

Other ROI Code Updates

Update code that uses any of the object functions of the `imrect` ROI object. In many cases, you can replace the call to an `imrect` object function by simply accessing or setting the value of a `Rectangle` ROI object property. For example, replace calls to `getColor` or `setColor` with use of the `Color` property. In some cases, you must replace the `imrect` object function with an object function of the new `Rectangle` ROI. Each of the individual `imrect` ROI object functions include information about migrating to the new `Rectangle` ROI object. For a migration overview, see “ROI Migration”.

See Also

`Rectangle` | `drawrectangle`

Topics

“Create ROI Shapes”
“ROI Migration”

Introduced before R2006a

addNewPositionCallback

Add new-position callback to ROI object

Note `addNewPositionCallback` is not recommended. With the new ROIs, use the `addlistener` object function instead. For more information, see “Compatibility Considerations”.

Syntax

```
id = addNewPositionCallback(h,fcn)
```

Description

`id = addNewPositionCallback(h,fcn)` adds the function handle `fcn` to the list of new-position callback functions of the ROI object `h`. Whenever the ROI object changes its position, each function in the list is called with the syntax:

```
fcn(pos)
```

`pos` is of the form returned by the object's `getPosition` method. The return value, `id`, is used only with `removeNewPositionCallback`.

Examples

Display Updated Position in Title

Create a rectangle ROI object. Display the position of the rectangle in the title. The title updates when you move the rectangle.

```
imshow("cameraman.tif")
h = imrect(gca, [10 10 100 100]);
addNewPositionCallback(h,@(p) title(mat2str(p,3)));
```

Now drag the rectangle using the mouse to observe the callback behavior.

Input Arguments

h — ROI object

`imellipse` | `imfreehand` | `imline` | `impoint` | `impoly` | `imrect`

ROI object, specified as an `imellipse`, `imfreehand`, `imline`, `impoint`, `impoly`, or `imrect` object.

fcn — Function handle

handle

Function handle, specified as a handle. The function must accept a numeric array as input. The array must have the same form as returned when calling `getPosition` on the object. For more information, see “Create Function Handle”.

Output Arguments

`id` — Identifier of new-position callback function

`struct`

Identifier of new-position callback function, returned as a `struct`.

Compatibility Considerations

`addNewPositionCallback` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

With the new ROIs, the `Position` property contains the current location of the ROI. To receive notification when this value changes, set up a listener using the `addlistener` object function. To listen for position information, set up a listener for the `"MovingROI"` or `"ROIMoved"` events.

Update Code

Update all instances of `addNewPositionCallback`.

Discouraged Usage	Recommended Replacement

This example uses the `addNewPositionCallback` method to specify a callback function to execute when the ROI changes position. In this example, the callback function displays the current position in a title.
 	
`imshow("cameraman.tif")`
`h = imrect(gca,[10 10 100 100]);`
 	
`addNewPositionCallback(h,@(pos) myCallback(pos));`
 	
`function myCallback(pos)`
 	
`    title(mat2str(pos,3));`
 	
`end`
	Here is equivalent code, replacing the `addNewPositionCallback` object function with the `addlistener` object function. This example listens for the `"MovingROI"` event.
	
	`imshow("cameraman.tif")`
	`h = drawrectangle(gca,"Position",[10 10 100 100]);`
	
	`addlistener(h,"MovingROI",@(src,evt) myCallback(evt));`
	`function myCallback(evt)`
	
	`    title(mat2str(evt.CurrentPosition,3));`
	
	`end`

See Also

`imroi` | `removeNewPositionCallback` | `makeConstrainToRectFcn` | `setPositionConstraintFcn` | `getPositionConstraintFcn` | `getPosition`

Topics

“ROI Migration”
“Create Function Handle”
“Anonymous Functions”

“Parameterizing Functions”

Introduced in R2008a

createMask

(Not recommended) Create mask within image

Note `createMask` is not recommended. Use the `createMask` object function associated with the new ROI objects instead, described in “Compatibility Considerations”.

Syntax

```
BW = createMask(h)
BW = createMask(h,himage)
```

Description

`BW = createMask(h)` returns a mask, or binary image, with `1`s inside the ROI object `h` and `0`s everywhere else. The input image must be contained within the same axes as the ROI object.

`BW = createMask(h,himage)` returns a mask the same size as the image `himage`, with `1`s inside the ROI object `h` and `0`s everywhere else. This syntax is required when the axes that contains the ROI holds more than one image.

Examples

Create Binary Mask from Ellipse

Create an ellipse ROI.

```
imshow("coins.png")
e = imellipse;
```

Use the mouse to reshape and reposition the ellipse. Then, create a binary mask from the ROI. Pixels inside the ROI have the value `1`, and pixels outside the ROI have the value `0`. Display the mask in a new figure.

```
BW = createMask(e);
figure
imshow(BW)
```

Input Arguments

h — ROI object

`imellipse` | `imline` | `impoint` | `impoly` | `imrect`

ROI object, specified as an `imellipse`, `imline`, `impoint`, `impoly`, or `imrect` object.

himage — Handle to image

handle

Handle to one image, specified as a handle.

Output Arguments

BW — Mask

binary matrix

Mask, returned as a binary matrix. The mask is the same size as the input image contained in the same axes as `h`, or the image `himage`.

Compatibility Considerations

createMask is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

To create a binary mask image using the new ROIs, use the `createMask` object function associated with the new ROIs.

Update Code

Update all instances of `createMask`.

Discouraged Usage	Recommended Replacement

This example uses the `createMask` method to create a binary mask image from an ROI.
 	
`imshow("cameraman.tif")`
`h = imrect(gca, [10 10 100 100]);`
`bw = createMask(h);`
`imshow(bw);`
	Here is equivalent code, creating a binary mask image using one of the new ROI objects. Call the `createMask` object function associated with the new ROIs as you did with the previous ROIs.
	
	`imshow("cameraman.tif")`
	`h = drawrectangle(gca,"Position",[10 10 100 100]);`
	`bw = createMask(h);`
	`imshow(bw);`

See Also

`createMask` | `roifilt2` | `regionfill`

Topics

“ROI Migration”
“Specify ROI as Binary Mask”

Introduced in R2008a

getAngleFromHorizontal

Return angle between Distance tool and horizontal axis

Note `getAngleFromHorizontal` is not recommended. Use one of the ROI classes instead, described in “Create ROI Shapes”.

Syntax

```
angle = getAngleFromHorizontal(h)
```

Description

`angle = getAngleFromHorizontal(h)` returns the angle, in degrees, between the line defined by the Distance tool, `h`, and the horizontal axis.

Input Arguments

h — Distance tool

`imdistline`

Distance tool, specified as an `imdistline` object.

Output Arguments

angle — Angle

numeric scalar

Angle, returned as a numeric scalar in the range [0, 180] degrees.

Data Types: `double`

Algorithms

To understand how `imdistline` calculates the angle returned by `getAngleToHorizontal`, draw an imaginary horizontal vector from the bottom endpoint of the distance line, extending to the right. The value returned by `getAngleToHorizontal` is the angle from this horizontal vector to the distance line, which can range from 0 to 180 degrees.

See Also

`getDistance`

Introduced before R2006a

getColor

Get color used to draw ROI object

Note `getColor` is not recommended. Using the new ROIs, retrieve the color of the ROI by accessing the value of `Color` property instead. For more information, see “Compatibility Considerations”.

Syntax

```
color = getColor(h)
```

Description

`color = getColor(h)` gets the color used to draw the ROI object `h`.

Input Arguments

h — ROI object

`imellipse` | `imfreehand` | `imline` | `impoint` | `impoly` | `imrect`

ROI object, specified as an `imellipse`, `imfreehand`, `imline`, `impoint`, `impoly`, or `imrect` object.

Output Arguments

color — RGB color value

3-element numeric vector

RGB color value, returned as a 3-element numeric vector.

Compatibility Considerations

getColor is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

To retrieve the color of the new ROIs, access the value of the `Color` property of the ROI.

Update Code

Update all instances of `getColor`.

Discouraged Usage	Recommended Replacement

This example creates an ROI and uses `getColor` to retrieve the color of the ROI.
 	
`imshow('cameraman.tif');`
`h = imrect(gca,[10 10 100 100]);`
`rgb = getColor(h)`
	Here is equivalent code, replacing the old ROI with the new ROI object and then accessing the value of the `Color` property of the ROI.
	
	`imshow('cameraman.tif');`
	`h = drawrectangle(gca,'Position',[10 10 100 100]);`
	`rgb = h.Color`

See Also

`imroi` | `setColor`

Topics

“ROI Migration”

Introduced before R2006a

getDistance

Return distance between endpoints of Distance tool

Note `getDistance` is not recommended. Use one of the ROI classes instead, described in “Create ROI Shapes”.

Syntax

```
dist = getDistance(h)
```

Description

`dist = getDistance(h)` returns the distance between the endpoints of the Distance tool, `h`.

Input Arguments

h — Distance tool

`imdistline`

Distance tool, specified as an `imdistline` object.

Output Arguments

dist — Distance between endpoints

numeric scalar

Distance between endpoints, returned as a numeric scalar. The data units of the distance are determined by the `XData` and `YData` properties of the underlying image. By default, the distance is measured in pixels.

See Also

`getPosition` | `getAngleFromHorizontal`

Introduced before R2006a

getLabelHandle

Return handle to text label of Distance tool

Note `getLabelHandle` is not recommended. Use one of the ROI classes instead, described in "Create ROI Shapes".

Syntax

```
hlabel = getLabelHandle(h)
```

Description

`hlabel = getLabelHandle(h)` returns a handle to the text label of the Distance tool, `h`.

Input Arguments

h — Distance tool

`imdistline`

Distance tool, specified as an `imdistline` object.

Output Arguments

hlabel — Handle to text label

handle

Handle to text label, returned as a handle to a Text object.

See Also

Text

Introduced before R2006a

getLabelTextFormatter

Return format of text label of Distance tool

Note `getLabelTextFormatter` is not recommended. Use one of the ROI classes instead, described in “Create ROI Shapes”.

Syntax

```
str = getLabelTextFormatter(h)
```

Description

`str = getLabelTextFormatter(h)` returns a character array specifying the format used to display the label text of the Distance tool, `h`.

Input Arguments

h — Distance tool

`imdistline`

Distance tool, specified as an `imdistline` object.

Output Arguments

str — Text format

character array

Text format of Distance tool label, returned as a character array in a format expected by `sprintf`.

See Also

`getLabelVisible` | `setLabelTextFormatter` | `setLabelVisible`

Introduced before R2006a

getLabelVisible

Return visibility of text label of Distance tool

Note `getLabelVisible` is not recommended. Use one of the ROI classes instead, described in "Create ROI Shapes".

Syntax

```
visible = getLabelVisible(h)
```

Description

`visible = getLabelVisible(h)` returns the visibility of the text label of the Distance tool, `h`.

Input Arguments

h — Distance tool

`imdistline`

Distance tool, specified as an `imdistline` object.

Output Arguments

visible — Label visibility

`'on'` | `'off'`

Label visibility, returned as `'on'` or `'off'`.

See Also

`setLabelTextFormatter` | `getLabelTextFormatter` | `setLabelVisible`

Introduced before R2006a

getPosition

Return current position of ROI object

Note `getPosition` is not recommended. Using the new ROIs, retrieve the position of the ROI by accessing the value of `Position` property instead. For more information, see “Compatibility Considerations”.

Syntax

```
pos = getPosition(h)
```

Description

`pos = getPosition(h)` returns the current position of the ROI object, `h`.

Input Arguments

h — ROI object

`imellipse` | `imfreehand` | `imline` | `impoint` | `impoly` | `imrect`

ROI object, specified as an `imellipse`, `imfreehand`, `imline`, `impoint`, `impoly`, or `imrect` object.

Output Arguments

pos — Position of ROI object

numeric array

Position of the ROI object, returned as a numeric array. The shape of the array depends on the type of ROI object.

ROI Object	Returned position
`imellipse`	4-element vector of the form `[xmin ymin width height]`, representing the size and position of a bounding box around the ellipse. The initial size of the bounding box is `width`-by-`height` pixels. The upper-left corner of the box is at the (x,y) coordinate (`xmin`,`ymin`).
`imfreehand`	*n*-by-2 matrix. The two columns define the *x*- and *y*-coordinates, respectively, of the *n* points along the boundary of the freehand region.
`imline`	2-by-2 matrix of the form `[x1 y1; x2 y2]`, representing the position of the two endpoints of the line.
`impoint`	1-by-2 vector of the form `[x y]`.
`impoly`	*n*-by-2 matrix. The two columns define the *x*- and *y*-coordinates, respectively, of each of the *n* vertices.

ROI Object	Returned position
`imrect`	4-element vector of the form `[xmin ymin width height]`. The initial size of the rectangle is `width`-by-`height` pixels. The upper-left corner of the rectangle is at the (x,y) coordinate (`xmin`,`ymin`).

Compatibility Considerations

`getPosition` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

To retrieve the current position of the ROI, access the value of the `Position` property of the ROI.

Update Code

Update all instances of `getPosition`.

Discouraged Usage	Recommended Replacement

This example creates an ROI and uses `getPosition` to retrieve the current location of the ROI.
 	
`imshow('cameraman.tif');`
`h = imrect(gca,[10 10 100 100]);`
`pos = getPosition(h)`
	Here is equivalent code, replacing the old ROI with a new ROI object and then accessing the value of the `Position` property of the ROI.
	
	`imshow('cameraman.tif');`
	`h = drawrectangle(gca,'Position',[10 10 100 100]);`
	`pos = h.Position`

See Also

`imroi` | `setPosition` | `getPositionConstraintFcn`

Topics

“ROI Migration”

Introduced in R2008a

getPositionConstraintFcn

Return function handle to current position constraint function

Note `getPositionConstraintFcn` is not recommended. With the new ROIs, use the `DrawingArea` property instead. For more information, see “Compatibility Considerations”.

Syntax

```
fcn = getPositionConstraintFcn(h)
```

Description

`fcn = getPositionConstraintFcn(h)` returns a function handle `fcn` to the current position constraint function of the ROI object `h`.

Input Arguments

h — ROI object

`imellipse` | `imline` | `impoint` | `impoly` | `imrect`

ROI object, specified as an `imellipse`, `imline`, `impoint`, `impoly`, or `imrect` object.

Output Arguments

fcn — Function handle

handle

Function handle, returned as a handle. For more information, see “Create Function Handle”.

Compatibility Considerations

getPositionConstraintFcn is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

Using the new ROIs, you do not need to create and register a position constraint function. You use the `DrawingArea` property of the ROI to control where you can draw and move an ROI. Therefore, there is no need for a function to retrieve the current constraint function.

Update Code

Update all instances of `getPositionConstraintFcn`.

Discouraged Usage	Recommended Replacement

This example creates a point ROI and uses the `setPositionConstraintFcn` method to confine ROI creation and movement to within the boundaries of the underlying image.
 	
`imshow('cell.tif')`
`h = impoint(gca,20,60);`
`% Create a position constraint function.`
`x = get(gca,'XLim');`
`y = get(gca,'YLim');`
`fcn = makeConstrainToRectFcn('impoint',x,y);`
`% Register the constraint function with the ROI`
`setPositionContraintFcn(h,fcn);`
`% Retrieve the handle of the current position constraint function.`
`fcn = getPositionContraintFcn(h);`
	With the new ROIs, there is no equivalent to getting a handle to the current position constraint function. The new ROIs use the `DrawingArea` property to specify the limits of the area where you can create and move an ROI. For example, this code creates a 10-pixel margin inside the image boundary where you cannot draw or move the ROI.
	
	`imshow('cell.tif')`
	`h = drawpoint(gca,'Position',[20 60])`
	`[height width] = size(I); %Get image dimensions`
	`h.DrawingArea = [10,10,(width-20),(height-20)];`

See Also

`imroi` | `makeConstrainToRectFcn` | `setPositionConstraintFcn` | `setConstrainedPosition` | `getPosition`

Topics

“ROI Migration”

Introduced in R2008a

getVertices

Return vertices on perimeter of ellipse ROI object

Note `getVertices` is not recommended. Using the new `Ellipse` ROI, access the value of the `Vertices` property instead. For more information, see “Compatibility Considerations”.

Syntax

```
v = getVertices(h)
```

Description

`v = getVertices(h)` returns a set of vertices that lie along the perimeter of an ellipse ROI object.

Input Arguments

h — ROI object

`imellipse`

ROI object, specified as an `imellipse` object.

Output Arguments

v — Vertices

n-by-2 matrix

Vertices, returned as an *n*-by-2 matrix. The two columns define the *x*- and *y*-coordinates, respectively, of each of the *n* vertices.

Compatibility Considerations

`getVertices` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

To retrieve the vertices of a new `Ellipse` ROI, access the value of the `Vertices` property of the ROI.

Update Code

Update all instances of `getVertices`.

Discouraged Usage	Recommended Replacement

This example creates an Ellipse ROI and uses `getVertices` to retrieve the vertices of the ROI.
 	
`imshow('cameraman.tif')`
`h = imellipse(gca,[10 10 100 100]);`
`v = getVertices(h)`
	Here is equivalent code, creating a new `Ellipse` ROI and then accessing the value of the `Ellipse` ROIs `Vertices` property.
	
	`imshow('cameraman.tif')`
	`h = drawellipse(gca,'Center',[60,35],'SemiAxes',[50,2`
	`v = h.Vertices`

See Also

`wait` | `getPosition` | `getColor`

Topics

“ROI Migration”

Introduced in R2007b

removeNewPositionCallback

Remove new-position callback from ROI object

Note `removeNewPositionCallback` is not recommended. With the new ROIs, use the `addlistener` object function instead. For more information, see “Compatibility Considerations”.

Syntax

```
removeNewPositionCallback(h,id)
```

Description

`removeNewPositionCallback(h,id)` removes the corresponding function from the new-position callback list of the ROI object, `h`.

Examples

Add and Remove New Position Callback

Create a line ROI object. Display the position of the line in the title. Use `addNewPositionCallback` to update the title each time you move the line.

```
imshow("pout.tif")
h = imline(gca,[10 100],[100 100]);
id = addNewPositionCallback(h,@(pos) title(mat2str(pos,3)));
```

Move the line to observe the callback behavior.

After observing the callback behavior, remove the callback. The title no longer changes when you move the line.

```
removeNewPositionCallback(h,id);
```

Input Arguments

h — ROI object

`imellipse` | `imline` | `impoint` | `impoly` | `imrect`

ROI object, specified as an `imellipse`, `imline`, `impoint`, `impoly`, or `imrect` object.

id — Identifier of new-position callback function

`struct`

Identifier of new-position callback function, specified as a `struct`.

Compatibility Considerations

`removeNewPositionCallback` is not recommended
Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

With the new ROIs, the `Position` property contains the current location of the ROI. To receive notification when this value changes, set up a listener using the `addlistener` object function. To remove this callback, delete the listener object.

Update Code

Update all instances of `removeNewPositionCallback`.

Discouraged Usage	Recommended Replacement

This example uses the `addNewPositionCallback` method to specify a callback function to execute when the ROI changes position. The code then uses `removeNewPositionCallback` to remove the callback.
 	
`imshow("cameraman.tif")`
`h = imrect(gca, [10 10 100 100]);`
`% Add callback that updates the title with position`
`id = addNewPositionCallback(h,@(p) title(mat2str(p,3)));`
`% Remove position callback. Title no longer updates`
`removeNewPositionCallback(h,id);`
	Here is equivalent code, creating a new ROI object and replacing the `addNewPositionCallback` object function with the `addlistener` object function. This example listens for the `"MovingROI"` event. To remove the listener, use `delete(el)`.
	
	`imshow("cameraman.tif")`
	`h = drawrectangle(gca,"Position",[10 10 100 100]);`
	`% Set up a listener for ROI moving events.`
	`el = addlistener(h,"MovingROI",@mymovecb)`
	`% Callback to update title with current position`
	`function mymovecb(src,evt)`
	`    currpos = evt.CurrentPosition;`
	`    title(mat2str(currpos,3))`
	`end`

See Also

`imroi` | `addNewPositionCallback` | `makeConstrainToRectFcn` | `setPositionConstraintFcn` | `getPositionConstraintFcn`

Topics
“ROI Migration”

Introduced in R2008a

resume

(Not recommended) Resume execution of MATLAB command line

Note `resume` is not recommended. The new ROI objects do not support a `resume` object function. For more information, see “Compatibility Considerations”.

Syntax

```
resume(h)
```

Description

`resume(h)` resumes execution of the MATLAB command line.

The `resume` function is useful when you need to exit `wait` from a callback function. When called after a call to `wait`, `resume` causes `wait` to return the ROI position or coordinates of vertices.

Input Arguments

h — ROI object

`imellipse` | `imline` | `impoint` | `impoly` | `imrect`

ROI object, specified as an `imellipse`, `imline`, `impoint`, `impoly`, or `imrect` object.

Compatibility Considerations

`resume` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

The new ROI objects do not support the `resume` object function. The `wait` object function associated with the new ROIs no longer returns position information, so there is no need for a `resume` function to trigger this return.

Update Code

Update all instances of `resume`.

Discouraged Usage	Recommended Replacement

`imshow('cameraman.tif');`
`h = imrect;`
`pos = wait(h);`
`% Call resume to return control`
`% to the command line`
`resume(h)`
	With the new ROIs, there is no equivalent to the `resume` object function. To migrate this code, remove the `wait` return value and delete calls to `resume` in callback functions.

See Also

Topics
“ROI Migration”
“Create ROI Shapes”
“Use Wait Function After Drawing ROI”

Introduced in R2008a

setClosed

Set closure behavior of ROI object

Note `setClosed` is not recommended. With the new ROIs, use the `Closed` property instead. For more information, see “Compatibility Considerations”.

Syntax

```
setClosed(h,TF)
```

Description

`setClosed(h,TF)` sets whether the ROI object, `h`, is closed after the last point is selected.

Input Arguments

h — ROI object

`imfreehand` | `impoly`

ROI object, specified as an `imfreehand` or `impoly` object.

TF — ROI object is closed

`true` | `false`

ROI object is closed, specified as `true` or `false`. When set to `true`, a straight line connect the endpoints of the ROI object to create a closed region. If set to `false`, the endpoints are not connected and the region is open.

Data Types: `logical`

Compatibility Considerations

setClosed is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

In the existing ROIs, you can control whether a hand-drawn shape or a polygonal shape are closed or open. By default, when you double-click to draw the final vertex of one of these shapes, the ROI draws a line between the last vertex and the first vertex to close the shape. Using the `setClosed` method, you can create an open hand-drawn shape or polyline. With the new ROIs, the `Freehand` and `AssistedFreehand` shapes support a `Closed` property that you can use to specify whether the shape is closed or open. The new ROIs support both a polygon (closed) and a polyline (open) shape, so there is no need for these ROIs to support a `Closed` property.

Update Code

Update all instances of `setClosed`.

Discouraged Usage	Recommended Replacement

This example creates a polygonal ROI and uses the `setClosed` method to turn the closed polygon into an open polyline.
 	
`imshow('cameraman.tif');`
`h = impoly(gca,[10 10; 50 10; 20 100]);`
`setClosed(h,false);`
	To create an open polyline ROI, replace the call to the `impoly` with `drawpolyline` (or the`Polyline` object). Use `drawpolygon` (or the `Polygon` object) to create a closed polygonal shape.
	
	`imshow('cameraman.tif');`
	`h = drawpolyline(gca,'Position',[10 10; 50 10; 20 100`

This example creates a hand-drawn ROI and uses the `setClosed`method to turn the closed shape into an open shape.
 	
`imshow('cameraman.tif');`
`h = imfreehand;`
`setClosed(h,false);`
	To create an open, hand-drawn ROI shape, replace the call to `imfreehand` with `drawfreehand` or `drawassisted`. You can also create `Freehand` or `AssistedFreehand` objects. Use the `Closed` property to turn the closed shape into an open shape.
	
	`imshow('cameraman.tif');`
	`h = drawfreehand;`
	`h.Closed = false;`

See Also

Introduced in R2007b

setColor

(Not recommended) Set color used to draw ROI object

Note `setColor` is not recommended. With the new ROIs, set the value of the `Color` property instead. For more information, see “Compatibility Considerations”.

Syntax

```
setColor(h,color)
```

Description

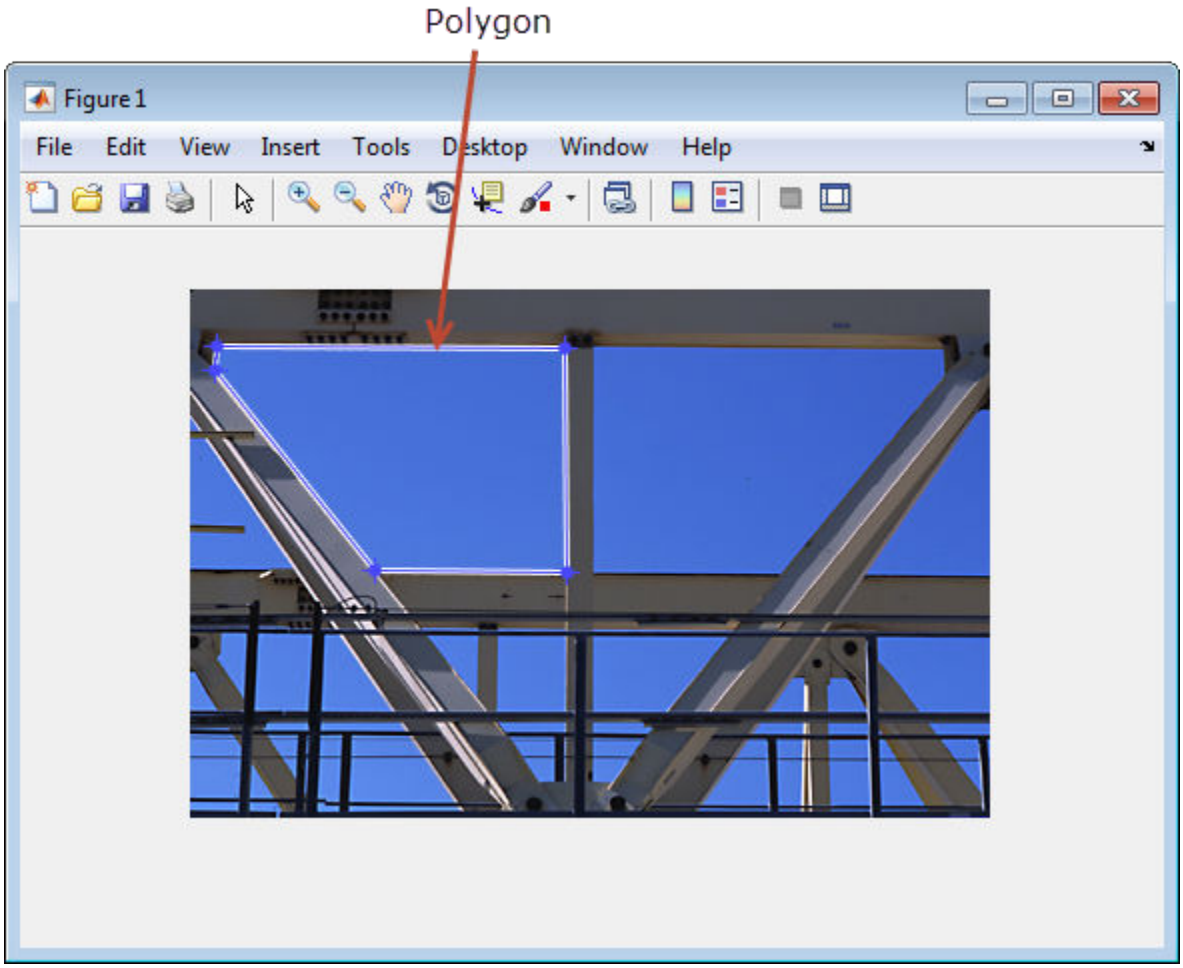
`setColor(h,color)` sets the color used to draw the ROI object `h`.

Examples

Set Color of Polygon ROI

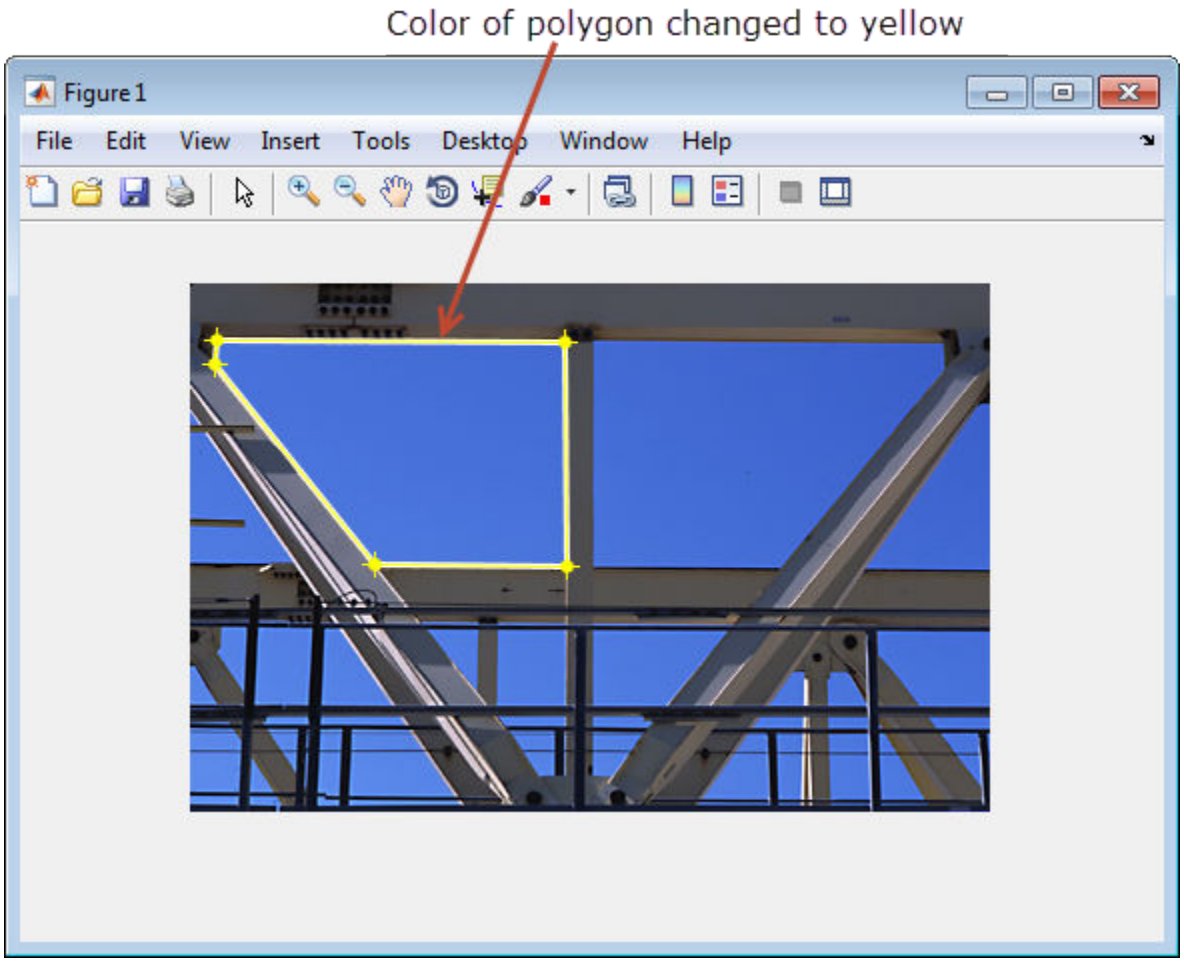
Display an image. Draw a polygon on the image, specifying the location of five vertices.

```
imshow('gantrycrane.png')
h = impoly(gca,[188,30; 189,142; 93,141; 13,41; 14,29]);
```



Set the color of the polygon to yellow.

```
setColor(h,'yellow');
```



Input Arguments

h — ROI object

`imellipse` | `imfreehand` | `imline` | `impoint` | `impoly` | `imrect`

ROI object, specified as an `imellipse`, `imfreehand`, `imline`, `impoint`, `impoly`, or `imrect` object.

`color` — ROI color

RGB triplet | color name | short color name

ROI color, specified as the comma-separated pair consisting of `'Color'` and an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'Color','r'`

Example: `'Color','green'`

Example: `'Color',[0 0.4470 0.7410]`

Compatibility Considerations

`setColor` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

To set the color of the new ROIs, set the value of the `Color` property.

Update Code

Update all instances of `setColor`.

Discouraged Usage	Recommended Replacement

This example creates an ROI and uses `setColor` to specify the color of the ROI.
 	
`imshow('cameraman.tif');`
`h = imrect(gca,[10 10 100 100]);`
`setColor(h,'yellow');`
	Replace the ROI with the equivalent new ROI object. Then, delete the call to `setColor` and set the value of the `Color` property of the ROI.
	
	`imshow('cameraman.tif');`
	`h = drawrectangle(gca,'Position',[10 10 100 100]);`
	`h.Color = 'yellow'`

See Also

`getColor` | `imroi`

Introduced in R2008a

setConstrainedPosition

Set ROI object to new position

Note `setConstrainedPosition` is not recommended. For information about setting position constraints, see “Compatibility Considerations”.

Syntax

```
setConstrainedPosition(h,pos)
```

Description

`setConstrainedPosition(h,pos)` sets the ROI object `h` to a new position. The candidate position, `pos`, is subject to the position constraint function specified by `setPositionConstraintFcn`.

Input Arguments

h — ROI object

`imellipse` | `imline` | `impoint` | `impoly` | `imrect`

ROI object, specified as an `imellipse`, `imline`, `impoint`, `impoly`, or `imrect` object.

pos — Candidate position of ROI object

numeric array

Candidate position of the ROI object, specified as a numeric array. The shape of the array depends on the type of ROI object, and is consistent with the form returned by the `setPosition` function.

ROI Object	Position
`imellipse`	4-element vector of the form `[xmin ymin width height]`, representing the new size and position of a bounding box around the ellipse. The new size of the bounding box is `width`-by-`height` pixels. The upper-left corner of the box is at the new (x,y) coordinate (`xmin`,`ymin`).
`imline`	2-by-2 matrix of the form `[x1 y1; x2 y2]`, representing the new position of the two endpoints of the line.
`impoint`	1-by-2 vector of the form `[x y]`.
`impoly`	*n*-by-2 matrix. The two columns define the new *x*- and *y*-coordinates, respectively, of each of the *n* vertices.
`imrect`	4-element vector of the form `[xmin ymin width height]`. The new size of the rectangle is `width`-by-`height` pixels. The upper-left corner of the rectangle is at the new (x,y) coordinate (`xmin`,`ymin`).

Compatibility Considerations

`setConstrainedPosition` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

With the existing ROIs, you use `makeConstrainToRectFcn` to create a function to specify the limits of the area in which you can draw or move an ROI. You then register this function with the ROI. When you use the `setPosition` object function, the ROI moves to wherever you specify, even if it is outside of the constrained limits. If you use `setConstrainedPosition`, the ROI honors the limits set by the constrained position function.

With the new ROIs, you use the `DrawingArea` property of the ROI to specify the area in which you can draw or move an ROI. When you set the location using the `Position` property, it does not honor limits set by the `DrawingArea` property.

Update Code

Update all instances of `setConstrainedPosition`.

Discouraged Usage	Recommended Replacement

This example uses the `makeConstrainToRectFcn` function to create a 20-pixel border in which you cannot create or move an ROI. The example then tries to specify a location for the ROI that lies outside these limits. Using the `setConstrainedPosition` object function, the ROI does not honor locations specified outside the limits. Using the `setPosition` object function you can specify locations outside the limits.
 	
`imshow('cell.tif')`
`h = impoint(gca,20,60);`
`% Make a function that constrains movement of the point`
`x = get(gca,'XLim');`
`y = get(gca,'YLim');`
`fcn = makeConstrainToRectFcn('impoint',x,y);;`
`% Apply the constraint function to the ROI.`
`setPositionConstraintFcn(h,fcn);`
`% Try to specify a Position value outside the limits.`
`setConstrainedPosition(h,[1 51]);`
`% Note how ROI does not honor value outside of limits.`
	With the new ROIs, use the `DrawingArea` property to limit the area in which you can draw or move an ROI. This example creates a 20-pixel margin inside the image boundary. There is no way to make the ROI honor constraints when specifying a location in the `Position` property.
	
	`I = imread('cell.tif');`
	`imshow(I)`
	`h = drawpoint(gca,'Position',[20 60])`
	`[height width] = size(I); %Get image dimensions`
	`h.DrawingArea = [20,20,(width-40),(height-40)];`

See Also

`imroi` | `getPositionConstraintFcn` | `setPositionConstraintFcn` | `setPosition` | `getPosition`

Introduced in R2008a

setFixedAspectRatioMode

Preserve aspect ratio when resizing ROI object

Note `setFixedAspectRatioMode` is not recommended. With the new ROIs, set the value of the `FixedAspectRatio` property instead. For more information, see “Compatibility Considerations”.

Syntax

```
setFixedAspectRatioMode(h,TF)
```

Description

`setFixedAspectRatioMode(h,TF)` sets whether the aspect ratio of the ROI object is preserved during interactive resizing.

Examples

Fix Aspect Ratio of Ellipse

Create an ellipse ROI object.

```
imshow("coins.png")
h = imellipse(gca,[10 10 100 100]);
```

Specify a position constraint function using `makeConstrainToRectFcn` to keep the ellipse inside the boundary of the image.

```
fcn = makeConstrainToRectFcn("imellipse",get(gca,"XLim"),get(gca,"YLim"));
setPositionConstraintFcn(h,fcn);
```

Try resizing and reshaping the ellipse.

Now, fix the aspect ratio of the ellipse.

```
setFixedAspectRatioMode(h,true);
```

Try resizing the ellipse. The aspect ratio of the ellipse does not change.

Input Arguments

h — ROI object

`imellipse` | `imrect`

ROI object, specified as an `imellipse` or `imrect` object.

TF — Fix aspect ratio

`true` | `false`

Fix the aspect ratio when resizing ROI object, specified as `true` or `false`.

Data Types: `logical`

Compatibility Considerations

`setFixedAspectRatioMode` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

To control whether an `Ellipse` or `Rectangle` ROI maintains the aspect ratio when being resized, use the `FixedAspectRatio` property of the ROI.

Update Code

Update all instances of `setFixedAspectRatioMode` method.

Discouraged Usage	Recommended Replacement

This example uses the `setFixedAspectRatioMode` method to preserve the aspect ratio of a rectangle when resizing.
 	
`imshow("cameraman.tif")`
`h = imrect(gca,[10 10 100 100]);`
`setFixedAspectRatioMode(h,true);`
	Create a Rectangle ROI, using the new ROI objects, and replace use of the `setFixedAspectRatioMode` method with setting the value of the `FixedAspectRatio` property. To preserve the aspect ration when resizing, set the property to `true`.
	
	`imshow("camerman.tif")`
	`h = drawrectangle(gca,"Position",[10 10 100 100]);`
	`h.FixedAspectRatio = true;`

See Also

`setResizable`

Introduced before R2006a

setLabelTextFormatter

Set format used to display text label of Distance tool

Note `setLabelTextFormatter` is not recommended. Use one of the ROI classes instead, described in “Create ROI Shapes”.

Syntax

```
setLabelTextFormatter(h,str)
```

Description

`setLabelTextFormatter(h,str)` sets the format used to display the label text of the Distance tool, `h`.

Examples

Format Label of Distance Tool

Display an image and create a Distance tool.

```
imshow("pout.tif")
hline = imdistline(gca,[71 171],[108 150]);
```

Modify the format of the label to indicate that distance is measured in pixels.

```
setLabelTextFormatter(hline,"%02.0f pixels");
```

Input Arguments

h — Distance tool

`imdistline` object

Distance tool, specified as an `imdistline` object.

str — Text format

string scalar | character vector

Text format of Distance tool label, specified as a string scalar or character vector in a format expected by `sprintf`.

See Also

`getLabelVisible` | `getLabelTextFormatter` | `setLabelVisible`

Introduced before R2006a

setLabelVisible

Set visibility of text label of Distance tool

Note `setLabelVisible` is not recommended. Use one of the ROI classes instead, described in "Create ROI Shapes".

Syntax

```
setLabelVisible(h,TF)
```

Description

`setLabelVisible(h,TF)` sets the visibility of the text label of the Distance tool, `h`.

Input Arguments

h — Distance tool

`imdistline`

Distance tool, specified as an `imdistline` object.

TF — Text label is visible

`true` | `false`

Text label is visible, specified as `true` or `false`.

Data Types: `logical`

See Also

`getLabelVisible` | `setLabelTextFormatter` | `getLabelTextFormatter`

Introduced before R2006a

setPosition

(Not recommended) Move ROI object to new position

Note `setPosition` is not recommended. With the new ROIs, set the value of the `Position` property instead. For more information, see “Compatibility Considerations”.

Syntax

```
setPosition(h,pos)
setPosition(h,x,y)
```

Description

`setPosition(h,pos)` moves the position of the ROI object, `h`, to the location specified by `pos`.

`setPosition(h,x,y)` specifies the new *x*- and *y*-coordinates of points of a line or point ROI object.

Input Arguments

h — ROI object

`imellipse` | `imline` | `impoint` | `impoly` | `imrect`

ROI object, specified as an `imellipse`, `imline`, `impoint`, `impoly`, or `imrect` object.

pos — New position of ROI object

numeric array

New position of the ROI object, specified as a numeric array. The shape of the array depends on the type of ROI object.

ROI Object	Position
`imellipse`	4-element vector of the form `[xmin ymin width height]`, representing the new size and position of a bounding box around the ellipse. The new size of the bounding box is `width`-by-`height` pixels. The upper-left corner of the box is at the new (x,y) coordinate (`xmin`,`ymin`).
`imline`	2-by-2 matrix of the form `[x1 y1; x2 y2]`, representing the new position of the two endpoints of the line.
`impoint`	1-by-2 vector of the form `[x y]`.
`impoly`	*n*-by-2 matrix. The two columns define the new *x*- and *y*-coordinates, respectively, of each of the *n* vertices.
`imrect`	4-element vector of the form `[xmin ymin width height]`. The new size of the rectangle is `width`-by-`height` pixels. The upper-left corner of the rectangle is at the new (x,y) coordinate (`xmin`,`ymin`).

x — New *x*-coordinate of points
2-element vector | numeric scalar

New *x*-coordinate of points.

- If `h` is an `imline` object, then `x` is a 2-element vector that represents the *x*-coordinates of the two line endpoints.
- If `h` is an `impoint` object, then `x` is a numeric scalar that represents the *x*-coordinate of the single point.

y — New *y*-coordinate of points
2-element vector | numeric scalar

New *y*-coordinate of points.

- If `h` is an `imline` object, then `y` is a 2-element vector that represents the *y*-coordinates of the two line endpoints.
- If `h` is an `impoint` object, then `y` is a numeric scalar that represents the *y*-coordinate of the single point.

Compatibility Considerations

`setPosition` is not recommended
Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

To specify the current location of the ROI, assign a value to the `Position` property of the ROI.

Update Code

Update all instances of `setPosition` method.

Discouraged Usage	Recommended Replacement

This example sets the current location of the ROI using the `setPosition` object function.
 	
`imshow('cameraman.tif');`
`h = imrect(gca,[10 10 100 100]);`
`setPosition(h,[20 20 200 200]);;`
	Create one of the new ROI objects and replace use of the `setPosition` object function with assigning a value to the `Position` property of the ROI.
	
	`imshow('cameraman.tif');`
	`h = drawrectangle(gca,'Position',[10 10 100 100]);`
	`h.Position = [20 20 200 200];`

See Also
`Ellipse` | `Line` | `Point` | `Polygon` | `Polyline` | `Rectangle`

Topics
“Create ROI Shapes”

“ROI Migration”

Introduced in R2008a

setPositionConstraintFcn

Set position constraint function of ROI object

Note `setPositionConstraintFcn` is not recommended. With the new ROIs, use the `DrawingArea` property instead. For more information, see “Compatibility Considerations”.

Syntax

```
setPositionConstraintFcn(h,fcn)
```

Description

`setPositionConstraintFcn(h,fcn)` sets the position constraint function of the ROI object `h` to be the specified function handle, `fcn`. Whenever the object is moved because of a mouse drag, the constraint function is called using the syntax:

```
constrained_position = fcn(pos)
```

Examples

Update Title when Rectangle Moves

Display a rectangle ROI over an image.

```
imshow("cameraman.tif")
h = imrect(gca,[10 10 100 100]);
```

Display the position of the rectangle in the title.

```
addNewPositionCallback(h,@(p) title(mat2str(p,3)));
```

The title updates when you move the rectangle. Try dragging one side of the rectangle outside the boundary of the image.

Specify a position constraint function using `makeConstrainToRectFcn` to keep the rectangle inside the original `XLim` and `YLim` ranges.

```
fcn = makeConstrainToRectFcn("imrect",get(gca,"XLim"),get(gca,"YLim"));
setPositionConstraintFcn(h,fcn);
```

Now drag the rectangle using the mouse. Observe that the rectangle can no longer extend past the image boundary.

Input Arguments

h — ROI object

`imellipse` | `imline` | `impoint` | `impoly` | `imrect`

ROI object, specified as an `imellipse`, `imline`, `impoint`, `impoly`, or `imrect` object.

`fcn` — Function handle
handle

Function handle, specified as a handle. You can use the `makeConstrainToRectFcn` to create this function. The function must accept a numeric array as input, and it must return a numeric array as output. Both arrays must have the same form as returned when calling `getPosition` on the object. For more information, see “Create Function Handle”.

Compatibility Considerations

`setPositionConstraintFcn` is not recommended
Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

In the existing ROIs, you can create a function that controls where you can draw or move an ROI. You then register the position constraint function with the ROI. To specify the area when you can draw or move an ROI, use the `DrawingArea` property.

Update Code

Update all instances of `setPositionConstraintFcn` used with the freehand or polygonal ROI.

Discouraged Usage	Recommended Replacement

This example creates a point ROI and uses the `setPositionConstraintFcn` method to confine ROI creation and movement to within the boundaries of the underlying image.
 	
`imshow("cell.tif")`
`h = impoint(gca,20,60);`
`% Make a function that constrains movement of the point.`
`x = get(gca,"XLim");`
`y = get(gca,"YLim");`
`fcn = makeConstrainToRectFcn("impoint",x,y);`
`% Apply the constraint function to the ROI.`
`setPositionConstraintFcn(h,fcn);`
	Create one of the new ROI objects and use the `DrawingArea` property to specify the limits where you can create or move an ROI. For example, this code creates a 10 pixel margin inside the image boundary.
	
	`imshow("cell.tif")`
	`h = drawpoint(gca,"Position",[20 60])`
	`[height width] = size(I); %Get image dimensions`
	`h.DrawingArea = [10,10,(width-20),(height-20)];`

See Also

`imroi` | `addNewPositionCallback` | `makeConstrainToRectFcn` | `getPositionConstraintFcn` | `setConstrainedPosition` | `getPosition`

Topics
“Create Function Handle”
“Anonymous Functions”
“Parameterizing Functions”

Introduced in R2008a

setResizable

Set resize behavior of ROI object

Note `setResizable` is not recommended. With the new ROIs, use the `InteractionsAllowed` property instead. For more information, see “Compatibility Considerations”.

Syntax

```
setResizable(h,TF)
```

Description

`setResizable(h,TF)` sets whether the ROI object may be resized interactively.

Examples

Fix Size of Ellipse

Create an ellipse ROI object.

```
imshow("coins.png")
h = imellipse(gca,[10 10 100 100]);
```

Specify a position constraint function using `makeConstrainToRectFcn` to keep the ellipse inside the boundary of the image.

```
fcn = makeConstrainToRectFcn("imellipse",get(gca,"XLim"),get(gca,"YLim"));
setPositionConstraintFcn(h,fcn);
```

Click and drag with the mouse to try resizing, reshaping, and moving the ellipse.

Now, disable resizing the ellipse.

```
setResizable(h,false);
```

Click and drag the ellipse again. You can move it, but not change the size or shape of it.

Input Arguments

h — ROI object

`imellipse` | `imrect`

ROI object, specified as an `imellipse` or `imrect` object.

TF — Enable resizing of ROI object

`true` | `false`

Enable resizing of ROI object, specified as `true` or `false`.

Data Types: `logical`

Compatibility Considerations

`setResizable` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

To control whether an Ellipse or Rectangle ROI is resizable, use the `InteractionsAllowed` property of the ROI.

Update Code

Update all instances of `setResizable` used with the Ellipse ROI and the Rectangle ROI.

Discouraged Usage	Recommended Replacement

This example uses the `setResizable` method to turn off the ability to resize a `Rectangle` ROI.
 	
`imshow('cameraman.tif')`
`h = imrect(gca,[10 10 100 100]);`
`setResizable(h,false);`
	Create a `Rectangle` ROI, using the new ROI objects, and replace use of the `setResizable` method with setting the value of the `InteractionsAllowed` property. To turn off resizing, set the value of the property to `'none'`.
	
	`imshow('camerman.tif')`
	`h = drawrectangle(gca,'Position',[10 10 100 100]);`
	`h.InteractionsAllowed = 'none';`

See Also

`setFixedAspectRatioMode`

Introduced before R2006a

setString

Set text label for point ROI object

Note `setString` is not recommended. With the new `Point` ROI, set the value of the `Label` property instead. For more information, see “Compatibility Considerations”.

Syntax

```
setString(h,text)
```

Description

`setString(h,text)` places a text label, `text`, to the lower right of the point ROI object, `h`.

Examples

Set Label of Point ROI

Display an image and create a point ROI.

```
imshow("rice.png")
h = impoint(gca,100,200);
```

Set the label of the ROI.

```
setString(h,"My point label");
```

Input Arguments

h — Point ROI object

`impoint`

Point ROI object, specified as an `impoint` object.

text — Text label

string scalar | character vector

Text label, specified as a string scalar or character vector.

Data Types: `string` | `char`

Compatibility Considerations

setString is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support

events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

To specify a label for a point ROI, assign a value to the `Label` property of the ROI. Use the `LabelVisible` property to control the visibility of the label.

Update Code

Update all instances of `setString` method.

Discouraged Usage	Recommended Replacement

This example specifies a label for the point ROI using the `setString` method.
 	
`imshow("cameraman.tif");`
`h = impoint(gca,[100 100]);`
`setString(h,"My Label");`
	Create a point ROI using the new ROI objects and replace use of the `getString` method with setting the value of the `Label` property of the ROI. You can control the visibility of the label using the `LabelVisible` property. All the new ROI objects support a `Label` property.
	
	`imshow("cameraman.tif");`
	`h = drawpoint(gca,"Position",[100 100]);`
	`h.Label = "My Label";`

See Also

Introduced before R2006a

setVerticesDraggable

Set vertex behavior of ROI object

Note `setVerticesDraggable` is not recommended. With the new `Polygon` ROI, set the value of the `InteractionsAllowed` property instead. For more information, see "Compatibility Considerations".

Syntax

```
setVerticesDraggable(h,TF)
```

Description

`setVerticesDraggable(h,TF)` sets whether the vertices of the ROI object, `h`, can be dragged after placement.

Input Arguments

h — ROI object

`impoly`

ROI object, specified as an `impoly` object.

TF — Polygon ROI vertices are draggable

`true` | `false`

Polygon ROI vertices are draggable, specified as `true` or `false`.

Data Types: `logical`

Compatibility Considerations

`setVerticesDraggable` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see "Create ROI Shapes".

To specify whether the vertices of a polygon are draggable, assign a value to the `InteractionsAllowed` property of the `Polygon` ROI.

Update Code

Update all instances of `setVerticesDraggable` method.

Discouraged Usage	Recommended Replacement

This example uses `setVerticesDraggable` to turn off the capability of moving the vertices of a polygon to reshape it.
 	
`imshow('cameraman.tif');`
`h = impoly(gca,[188,30; 189,142; 93,141]);`
`setVerticesDraggable(h,false);`
	Here is the equivalent code, creating a `Polygon` ROI and replacing use of `setVerticesDraggable` with setting the value of the `InteractionsAllowed` property to a value that doesn't provide reshaping abilities (`'none'`).
	
	`imshow('cameraman.tif');`
	`h = drawpolygon(gca,'Position',[188,30; 189,142; 93,1`
	`h.InteractionsAllowed = 'none';`

See Also

`getVertices`

Topics

“ROI Migration”

Introduced in R2007b

wait

(Not recommended) Block MATLAB command line until ROI creation is finished

Note `wait` is not recommended. Use the `wait` method associated with the new ROIs instead. For more information, see “Compatibility Considerations”.

Syntax

```
pos = wait(h)
v = wait(he)
```

Description

`pos = wait(h)` blocks execution of the MATLAB command line until you finish positioning the ROI object `h`. Indicate completion by double-clicking on the ROI object. The function returns the position, `pos`, of the ROI object.

`v = wait(he)` blocks execution of the MATLAB command line until you finish positioning the ellipse ROI object `he`. Indicate completion by double-clicking on the ROI object. The function returns the coordinates of vertices, `v`, along the perimeter of the ellipse.

Examples

Click and Drag to Place Rectangle

Interactively place a rectangle by clicking and dragging. Use `wait` to block the MATLAB command line. Double-click on the rectangle to resume execution of the MATLAB command line.

```
imshow('pout.tif')
h = imrect;
position = wait(h)
```

Click and Drag to Place Ellipse

Interactively place an ellipse by clicking and dragging. Use `wait` to block the MATLAB command line. Double-click on the ellipse to resume execution of the MATLAB command line.

```
imshow('coins.png')
h = imellipse;
position = wait(h)
```

Input Arguments

h — ROI object

`imfreehand` | `imline` | `impoint` | `impoly` | `imrect`

ROI object, specified as an `imfreehand`, `imline`, `impoint`, `impoly`, or `imrect` object.

he — Ellipse ROI object

`imellipse`

Ellipse ROI object, specified as an `imellipse` object.

Output Arguments

pos — Position of ROI object

numeric array

Position of the ROI object, returned as a numeric array. The shape of the array depends on the type of ROI object, and is consistent with the output of `getPosition`.

ROI Object	Returned position
`imfreehand`	*n*-by-2 matrix. The two columns define the *x*- and *y*-coordinates, respectively, of the *n* points along the boundary of the freehand region.
`imline`	2-by-2 matrix of the form `[x1 y1; x2 y2]`, representing the position of the two endpoints of the line.
`impoint`	1-by-2 vector of the form `[x y]`.
`impoly`	*n*-by-2 matrix. The two columns define the *x*- and *y*-coordinates, respectively, of each of the *n* vertices.
`imrect`	4-element vector of the form `[xmin ymin width height]`. The initial size of the rectangle is `width`-by-`height` pixels. The upper-left corner of the rectangle is at the (x,y) coordinate (`xmin`,`ymin`).

v — Vertices of ellipse ROI object

n-by-2 matrix

Vertices of ellipse ROI object, returned as an *n*-by-2 matrix. The two columns define the *x*- and *y*-coordinates, respectively, of each of the *n* vertices. The form of the matrix is consistent with the output of `getVertices`.

Compatibility Considerations

`wait` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

In 19b, all the new ROI objects support a `wait` object function, as the old ROI objects did. Use the `wait` function to block the MATLAB command line after creating an ROI. For example, you can use `wait` to block the command line until you have finished positioning the ROI.

By default, the new `wait` function returns control to the command line after you double-click the ROI. However, using events, you can implement a custom `wait` function that resumes execution of the command line after several types of actions, such as clicking the ROI while pressing the Shift key or clicking a specific part of the ROI such as the label. For an example, see “Use Wait Function After Drawing ROI”.

Update Code

Update all instances of `wait`.

Discouraged Usage	Recommended Replacement

This example creates a Rectangle ROI and then pauses the MATLAB command line. You can move the ROI during this pause. When you are done, double-click the mouse. Control returns to the command line and the `wait` function returns position information to the workspace in the variable `pos`.
`imshow('cameraman.tif')`
`h = imrect(gca,[10 10 100 100]);`
`pos = wait(h);`
`% When you double-click, wait returns.`
`% View the value of the pos variable.`
`pos`
	To migrate use of the `wait` function, create the ROI using the new ROI objects. Remove the `wait` return value—the new `wait` object function does not return a value. Instead, the ROI updates the values properties changed during the pause, such as the `Position` property.
	`imshow('cameraman.tif');`
	`h = drawrectangle(gca,'Position',[10 10 100 100]);`
	`wait(h);`
	`% When you double-click, control returns to the comma`
	`% View the value of the Position property.`
	`h.Position`

`he = drawellipse;` `pos = wait(he);`	The new `wait` object function does not support a separate syntax for obtaining the position of an Ellipse ROI. Use the `wait(h)` syntax instead.

See Also

Topics
“Create ROI Shapes”
“Use Wait Function After Drawing ROI”

Introduced in R2008a

imrotate

Rotate image

Syntax

```
J = imrotate(I,angle)
J = imrotate(I,angle,method)
J = imrotate(I,angle,method,bbox)
```

Description

`J = imrotate(I,angle)` rotates image `I` by `angle` degrees in a counterclockwise direction around its center point. To rotate the image clockwise, specify a negative value for `angle`. `imrotate` makes the output image `J` large enough to contain the entire rotated image. By default, `imrotate` uses nearest neighbor interpolation, setting the values of pixels in `J` that are outside the rotated image to `0`.

`J = imrotate(I,angle,method)` rotates image `I` using the interpolation method specified by `method`.

`J = imrotate(I,angle,method,bbox)` also uses the `bbox` argument to define the size of the output image. You can crop the output to the same size as the input image or return the entire rotated image.

Examples

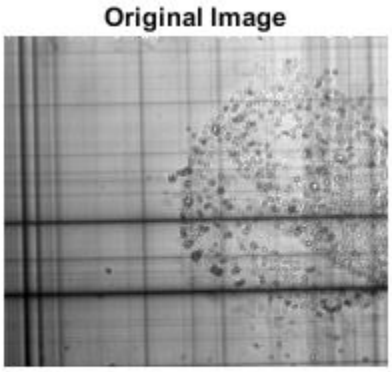
Rotate Image Clockwise for Better Horizontal Alignment

Read an image into the workspace, and convert it to a grayscale image.

```
I = fitsread('solarspectra.fts');
I = rescale(I);
```

Display the original image.

```
figure
imshow(I)
title('Original Image')
```

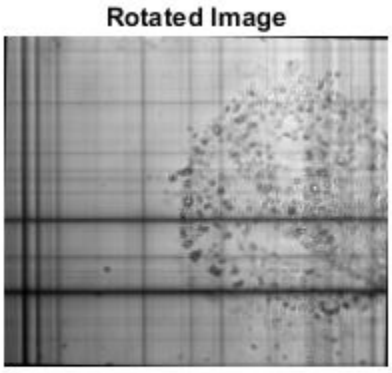


Rotate the image 1 degree clockwise to bring it into better horizontal alignment. The example specified bilinear interpolation and requests that the result be cropped to be the same size as the original image.

```
J = imrotate(I,-1,'bilinear','crop');
```

Display the rotated image.

```
figure
imshow(J)
title('Rotated Image')
```



Input Arguments

I — Image to be rotated

numeric array | logical array | categorical array

Image to be rotated, specified as a numeric array, logical array, or categorical array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `categorical`

`angle` — Amount of rotation in degrees
numeric scalar

Amount of rotation in degrees, specified as a numeric scalar.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`method` — Interpolation method
`'nearest'` (default) | `'bilinear'` | `'bicubic'`

Interpolation method, specified as one of the following values:

Value	Description
`'nearest'`	Nearest-neighbor interpolation. The output pixel is assigned the value of the pixel that the point falls within. No other pixels are considered. Nearest-neighbor interpolation is the only method supported for categorical images.
`'bilinear'`	Bilinear interpolation. The output pixel value is a weighted average of pixels in the nearest 2-by-2 neighborhood.
`'bicubic'`	Bicubic interpolation. The output pixel value is a weighted average of pixels in the nearest 4-by-4 neighborhood. **Note** Bicubic interpolation can produce pixel values outside the original range.

Data Types: `char` | `string`

`bbox` — Bounding box defining size of output image
`'loose'` (default) | `'crop'`

Bounding box that defines the size of output image, specified as either of the following values:

Value	Description
`'crop'`	Make output image `J` the same size as the input image `I`, cropping the rotated image to fit.
`'loose'`	Make output image `J` large enough to contain the entire rotated image. `J` is larger than `I`.

Data Types: `char` | `string`

Output Arguments

`J` — Rotated image
numeric array | logical array | categorical array

Rotated image, returned as a numeric, logical, or categorical array of the same data type as the input image, `I`.

Tips

- This function changed in version 9.3 (R2015b). Previous versions of the Image Processing Toolbox use different spatial conventions. If you need the same results produced by the previous implementation, use the function `imrotate_old`.
- In some instances, this function takes advantage of hardware optimization for data types `uint8`, `uint16`, `single`, and `double` to run faster.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imrotate` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imrotate` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see "Types of Code Generation Support in Image Processing Toolbox".
- Input images of data type categorical are not supported.
- The `method` and `bbox` arguments must be compile-time constants.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- Input images of data type categorical are not supported.
- The `method` and `bbox` arguments must be compile-time constants.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see "Run MATLAB Functions in Thread-Based Environment".

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- `gpuArray` input must be of type `uint8`, `uint16`, `single`, or `logical`.
- The `'bicubic'` interpolation mode used in the GPU implementation of this function differs from the default (CPU) bicubic mode. The GPU and CPU versions of this function are expected to give slightly different results.

For more information, see "Image Processing on a GPU".

See Also

`imcrop` | `imrotate3` | `imresize` | `imtransform` | `tformarray`

Topics
“Rotate an Image”
“Find Image Rotation and Scale”

Introduced before R2006a

imrotate3

Rotate 3-D volumetric grayscale image

Syntax

```
B = imrotate3(V,angle,W)
B = imrotate3(V,angle,W,method)
B = imrotate3(V,angle,W,method,bbox)
B = imrotate3( ___ ,'FillValues',fillValues)
```

Description

`B = imrotate3(V,angle,W)` rotates the 3-D volume `V` by `angle` degrees counterclockwise around an axis passing through the origin `[0 0 0]`. `W` is a 1-by-3 vector which specifies the direction of the axis of rotation in 3-D space. By default, `imrotate3` sets the values of voxels in `B` that are outside the boundaries of the rotated volume to `0`.

`B = imrotate3(V,angle,W,method)` also specifies the interpolation method.

`B = imrotate3(V,angle,W,method,bbox)` also specifies the size of the output volume, `bbox`. If you specify `'crop'`, then `imrotate3` makes the output volume the same size as the input volume. If you specify `'loose'`, then `imrotate3` makes the output volume large enough to include the entirety of the rotated volume.

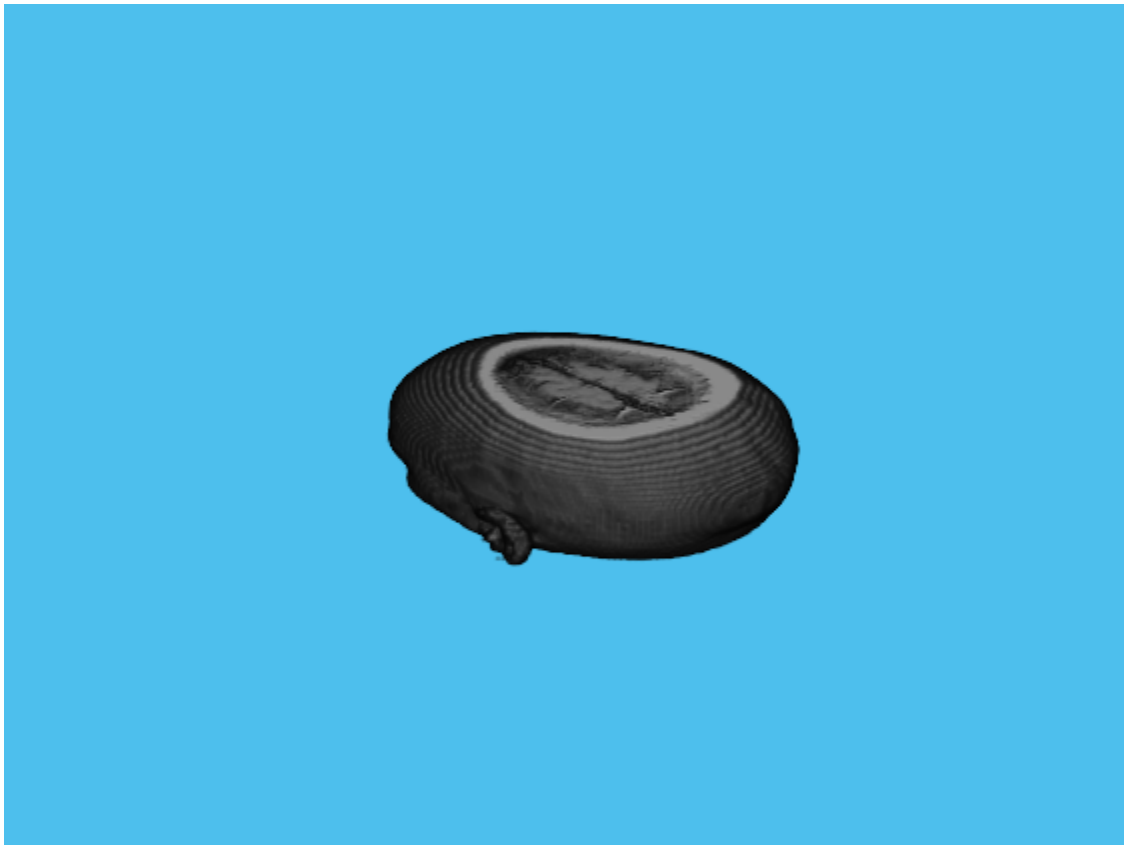
`B = imrotate3( ___ ,'FillValues',fillValues)` sets the fill values used for output pixels without a corresponding pixel in the input image.

Examples

Rotate 3-D Volume

Load a 3-D volumetric grayscale image into the workspace, and display it.

```
s = load('mri');
mriVolume = squeeze(s.D);
volshow(mriVolume);
```

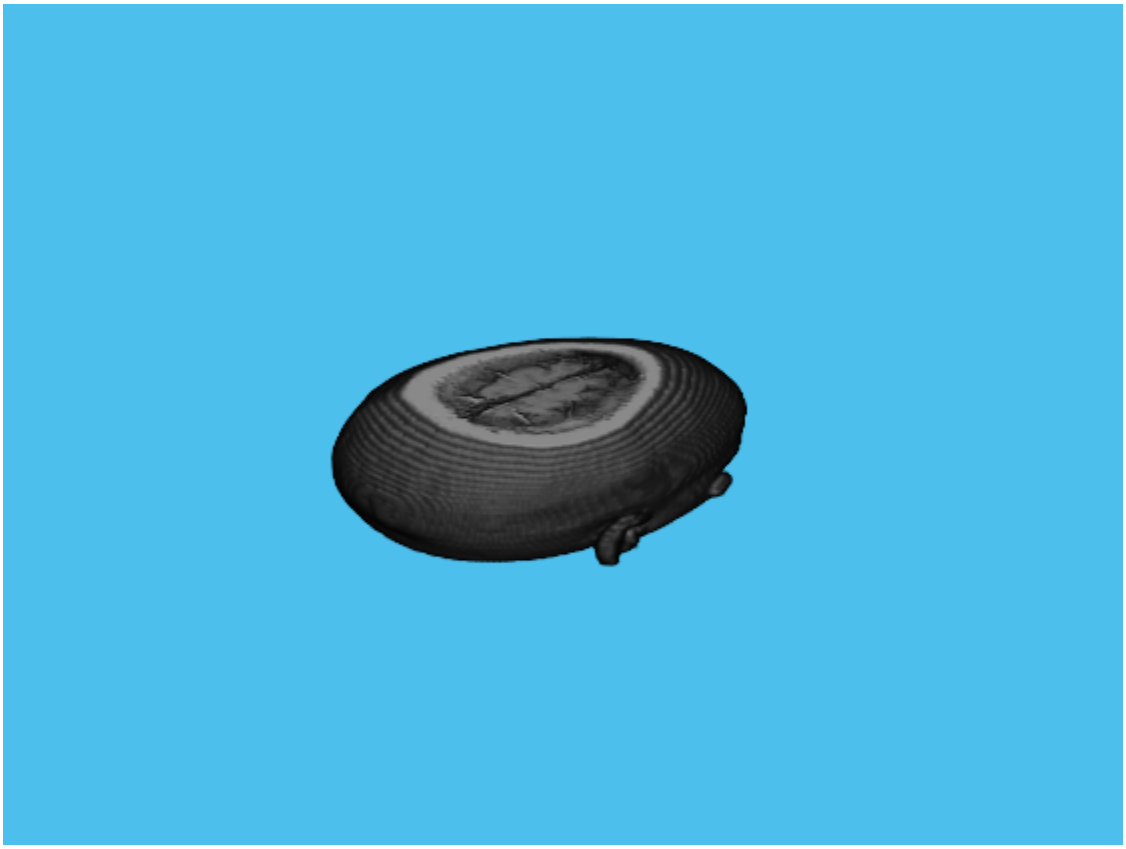


Rotate the volume 90 degrees around the Z axis.

```
B = imrotate3(mriVolume,90,[0 0 1],'nearest','loose','FillValues',0);
```

Display the rotated output volume. You can also explore the volume in the Volume Viewer app.

```
volshow(B);
```



Input Arguments

`V` — Volume to be rotated

3-D numeric array | 3-D logical array | 3-D categorical array

Volume to be rotated, specified as a 3-D numeric array, 3-D logical array, or 3-D categorical array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `categorical`

`angle` — Rotation angle in degrees

numeric scalar

Rotation angle in degrees, specified as numeric scalar. To rotate the volume clockwise, specify a negative value for `angle`. `imrotate3` makes the output volume `B` large enough to contain the entire rotated 3-D volume.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`W` — Direction of the axis of rotation

1-by-3 numeric vector

Direction of the axis of rotation in 3-D space in Cartesian coordinates, specified as a 1-by-3 numeric vector.

If you want to specify the direction of the axis of rotation in spherical coordinates, use `sph2cart` to convert values to Cartesian coordinates before passing it to `imrotate3`.

Example: `[0 0 1]` rotates the volume around the Z axis

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`method` — Interpolation method

`'nearest'` | `'linear'` | `'cubic'`

Interpolation method, specified as one of the following values.

Method	Description
`'nearest'`	Nearest neighbor interpolation. The output voxel is assigned the value of the voxel that the point falls within. No other voxels are considered. Nearest-neighbor interpolation is the only method supported for categorical images and it is the default method for images of this type.
`'linear'`	Trilinear interpolation. Trilinear interpolation is the default method for numeric and logical images.
`'cubic'`	Tricubic interpolation **Note** Tricubic interpolation can produce pixel values outside the original range.

Data Types: `char` | `string`

`bbox` — Size of the output volume

`'loose'` (default) | `'crop'`

Size of the output volume, specified as either of the following values.

Method	Description
`'crop'`	Make the output volume the same size as the input volume, cropping the rotated volume to fit.
`'loose'`	Make the output volume large enough to contain the entire rotated volume. Usually, the rotated volume is larger than the input volume.

Data Types: `char` | `string`

`fillValues` — Fill values

numeric scalar | string scalar | character vector | `missing`

Fill values used for output pixels outside the input image, specified as one of the following values. `imrotate3` uses fill values for output pixels when the corresponding inverse transformed location in the input image is completely outside the input image boundaries.

Image Type	Format of Fill Values
Numeric image or logical image	• Numeric scalar. The default fill value of numeric and logical images is `0`.
Categorical image	• Valid category in the image, specified as a string scalar or character vector. • `missing`, which corresponds to the `<undefined>` category. This is the default fill value for categorical images. For more information, see `missing`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `string`

Output Arguments

`B` — Rotated volume

numeric array | logical array | categorical array

Rotated volume, returned as a numeric, logical, or categorical array of the same class as the input volume, `V`.

Tips

- `imrotate3` assumes that the input volume `V` is centered on the origin `[0 0 0]`. If your volume is not centered on the origin, then use `imtranslate` to translate the volume to `[0 0 0]` before using `imrotate3`. You can translate the output volume `B` back to the original position with the opposite translation vector.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

`imrotate` | `imresize3` | `imtranslate` | `imwarp` | **Volume Viewer**

Introduced in R2017a

imsave

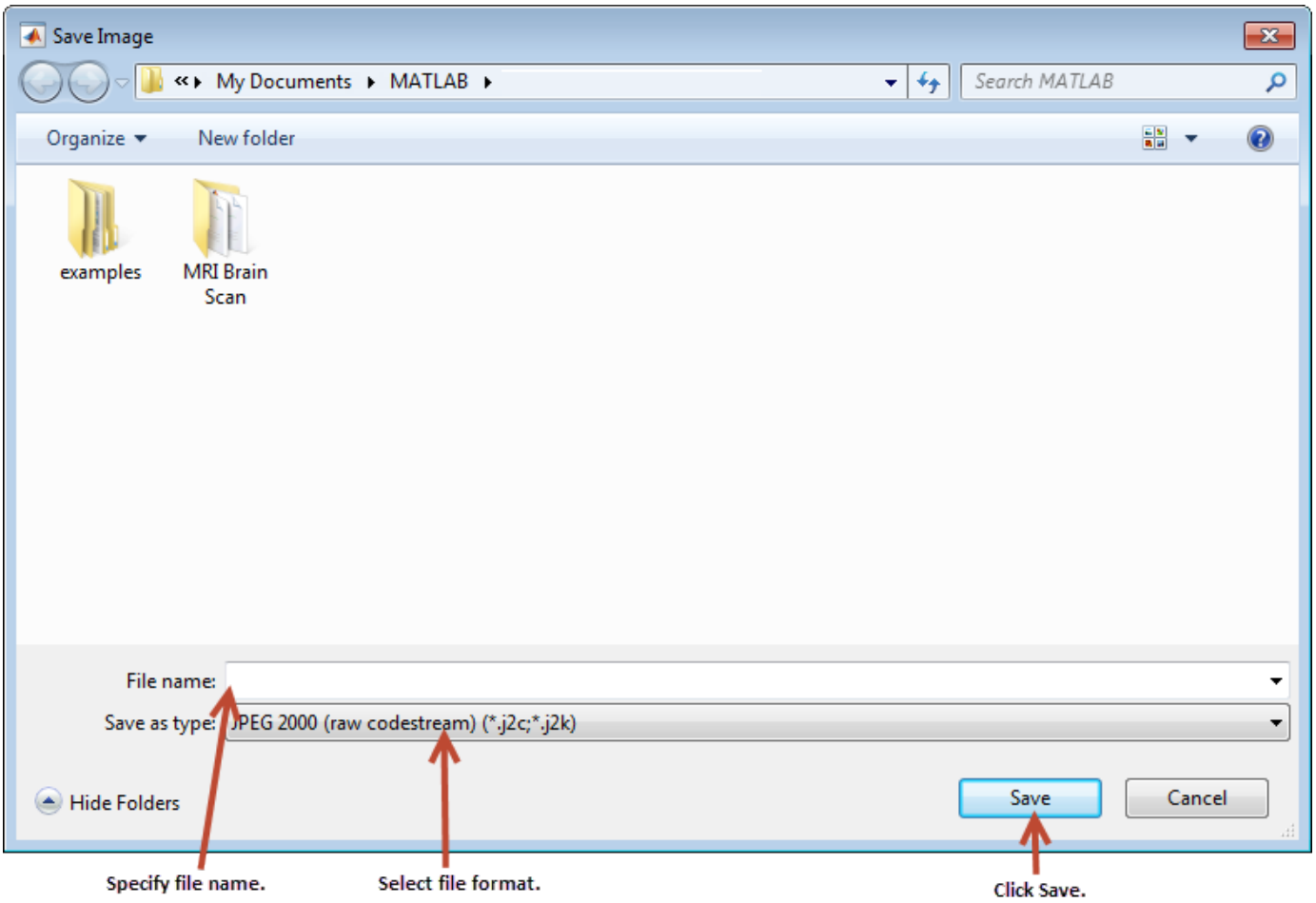
Save Image Tool

Syntax

```
imsave
imsave(h)
[filename,user_canceled] = imsave( ___ )
```

Description

Use the `imsave` function to create a Save Image tool that displays an interactive file chooser dialog box. Use this dialog box to navigate your file system to determine where to save the image file and specify the name of the file. Choose the graphics file format you want to use from among the image file formats listed in the **Files of Type** menu. For more information about using the tool, see “Tips” on page 1-1933.



`imsave` creates a Save Image tool in a separate figure that is associated with the image in the current figure, called the target image.

`imsave(h)` creates a Save Image tool associated with the image specified by the handle `h`.

`[filename,user_canceled] = imsave( ___ )` returns the full path to the file selected in `filename` and indicates whether you canceled the save operation.

Examples

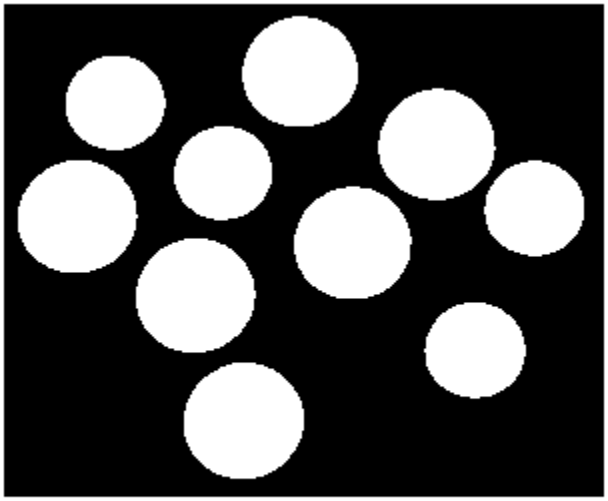
Save Displayed Image

Read a grayscale image into the workspace. Display the image.

```
I = imread('coins.png');
imshow(I)
```

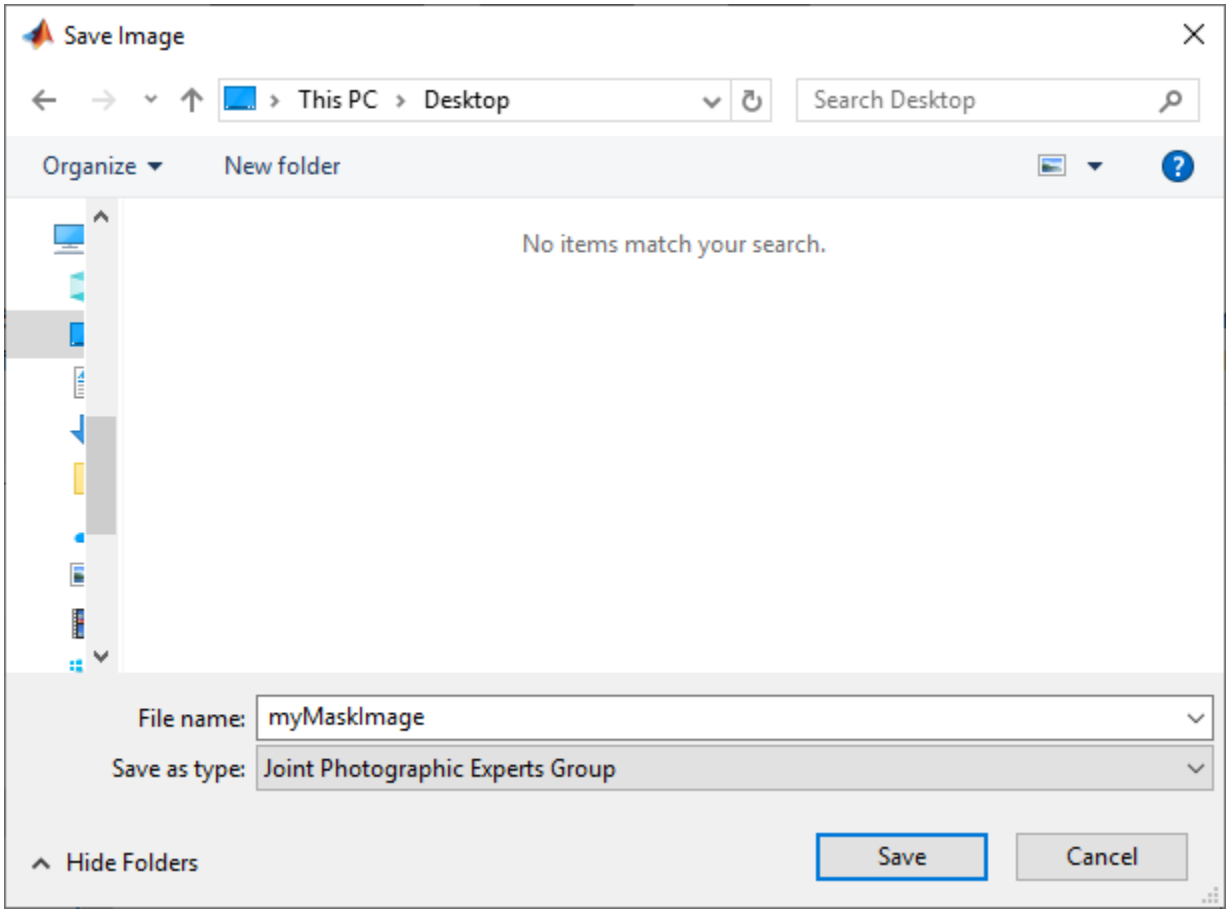
Process the image. This example creates a binary mask in which the background is black and the coins are white.

```
bw = imbinarize(I);
bw = imfill(bw,'holes');
imshow(bw,[])
```



Save the binary image to file by using the Save Image tool. You can navigate to the desired directory and specify the file name and file format. This example saves the image to the Desktop with the file name `myMaskImage` in the JPEG file format.

```
imsave
```



Input Arguments

h — Handle to graphics object

handle

Handle to a figure, axes, uipanel, or image graphics object, specified as a handle. If `h` is an axes or figure handle, then `imsave` uses the first image returned by `findobj(H,'Type','image')`.

Output Arguments

filename — Full path to file

character vector | `''`

Full path to file, returned as a character vector. If you cancel the save operation, then `filename` is an empty character array, `''`.

user_canceled — User canceled operation

`false` (default) | `true`

User canceled operation, returned as `false` or `true`. If you press the **Cancel** button or close the save window, then `imsave` sets `user_canceled` to `true`; otherwise, `false`.

Tips

- In contrast to the **Save as** option in the figure **File** menu, the Save Image tool saves only the image displayed in the figure. The **Save as** option in the figure window File menu saves the entire figure window, not just the image.
- `imsave` uses `imwrite` to save the image, using default options.
- If you specify a file name that already exists, then `imsave` displays a warning message. Select **Yes** to use the file name or **No** to return to the dialog to select another file name. If you select **Yes**, then the Save Image tool attempts to overwrite the target file.
- The Save Image tool is modal; it blocks the MATLAB command line until you respond.

See Also

`imformats` | `imgetfile` | `imputfile` | `imwrite`

Introduced in R2007b

imscrollpanel

Scroll panel for interactive image navigation

Syntax

```
hpanel = imscrollpanel(hparent,himage)
```

Description

Use the `imscrollpanel` function to add a scroll panel to an image. If the size or magnification makes an image too large to display in a figure on the screen, then the scroll panel displays a portion of the image at 100% magnification (one screen pixel represents one image pixel). The scroll panel adds horizontal and vertical scroll bars to enable navigation around the image.

`hpanel = imscrollpanel(hparent,himage)` creates a scroll panel containing the target image (the image to be navigated). `himage` is a handle to the target image. `hparent` is a handle to the figure or uipanel that will contain the scroll panel. The function returns `hpanel`, a handle to the scroll panel.

Examples

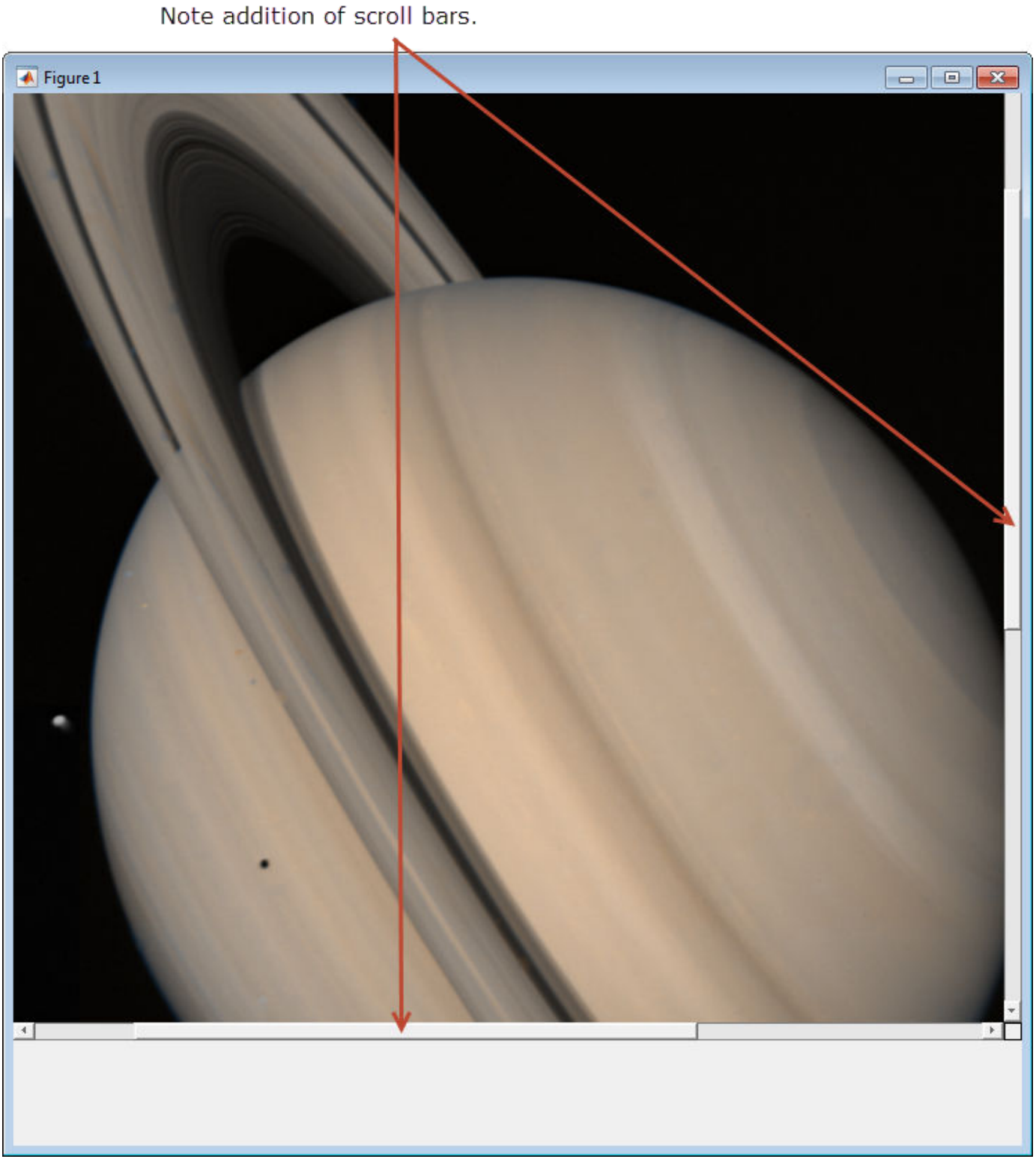
Create Scroll Panel with Magnification Box and Overview Tool

Display an image in a figure. The example suppresses the standard toolbar and menubar in the figure window because these do not work with the scroll panel.

```
hFig = figure('Toolbar','none',...
              'Menubar','none');
hIm = imshow('saturn.png');
```

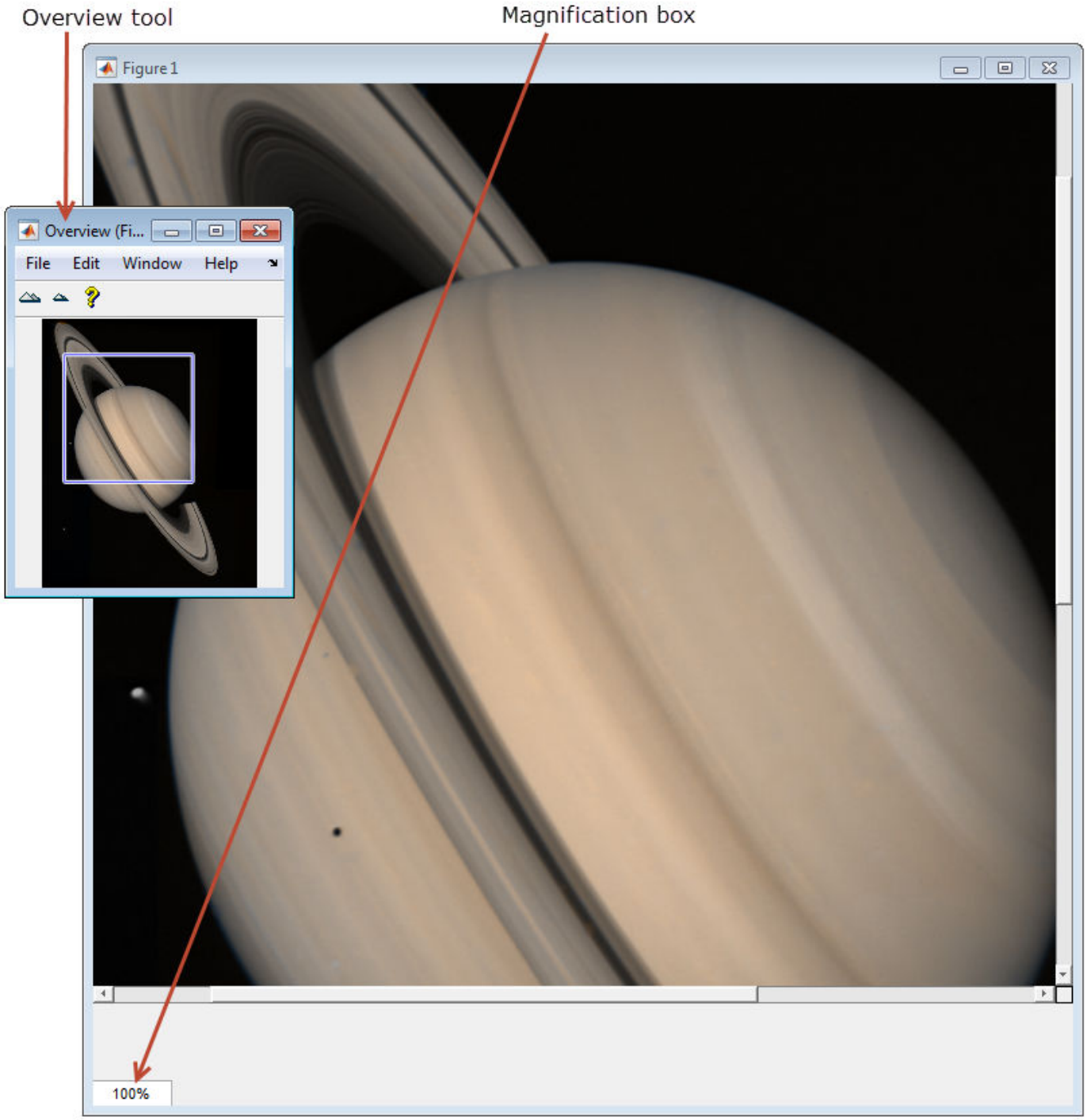
Create a scroll panel to contain the image.

```
hSP = imscrollpanel(hFig,hIm);
set(hSP,'Units','normalized','Position',[0 .1 1 .9])
```



Add a Magnification Box and an Overview tool to the figure.

```
hMagBox = immagbox(hFig,hIm);
pos = get(hMagBox,'Position');
set(hMagBox,'Position',[0 0 pos(3) pos(4)])
imoverview(hIm)
```



Get the scroll panel API so that you can control the view programmatically.

```
api = iptgetapi(hSP);
```

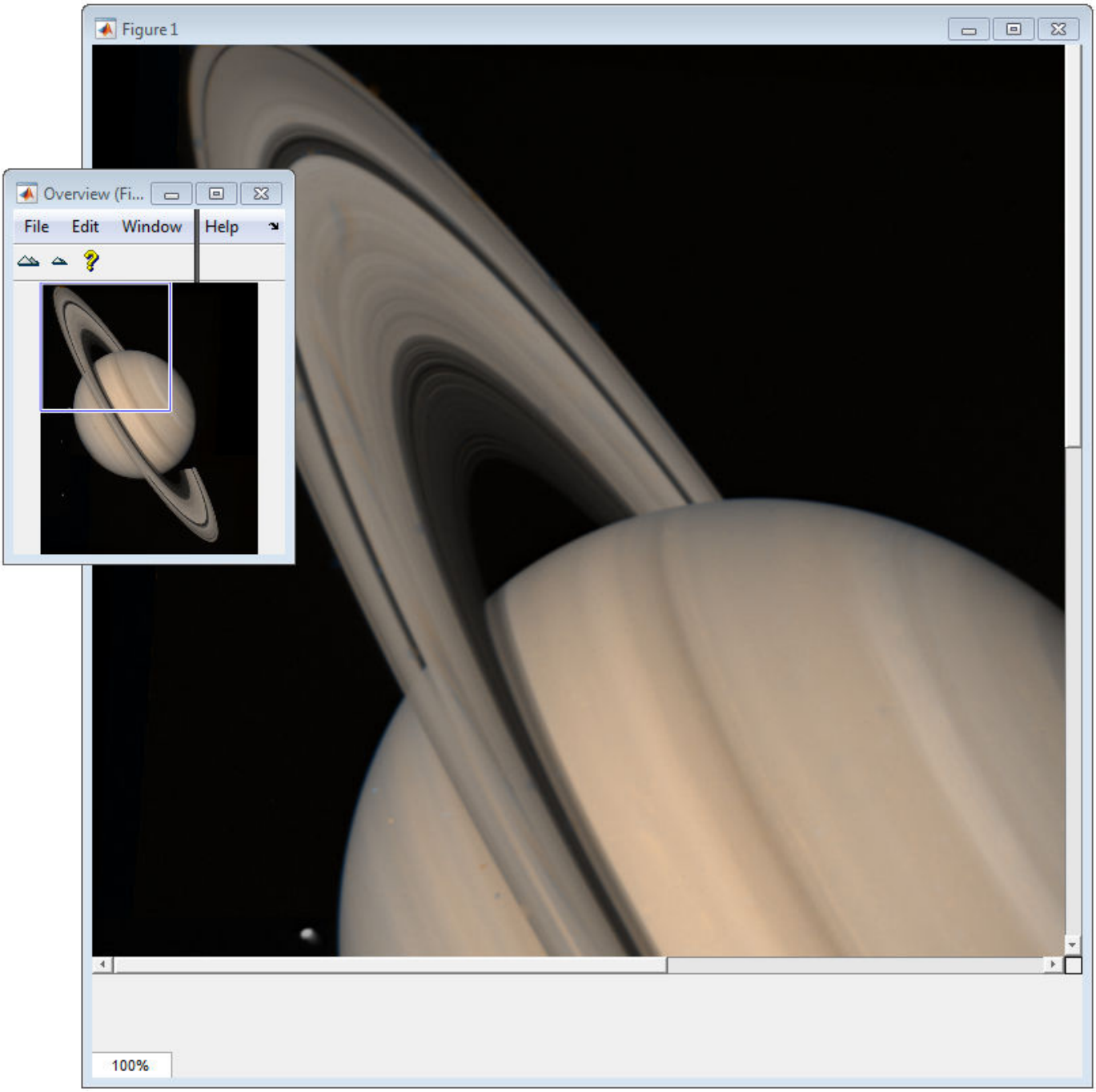
Get the current magnification and position.

```
mag = api.getMagnification()
r = api.getVisibleImageRect()
```

```
mag =

     1

r =

  125.0072  201.5646  716.0000  709.0000
```

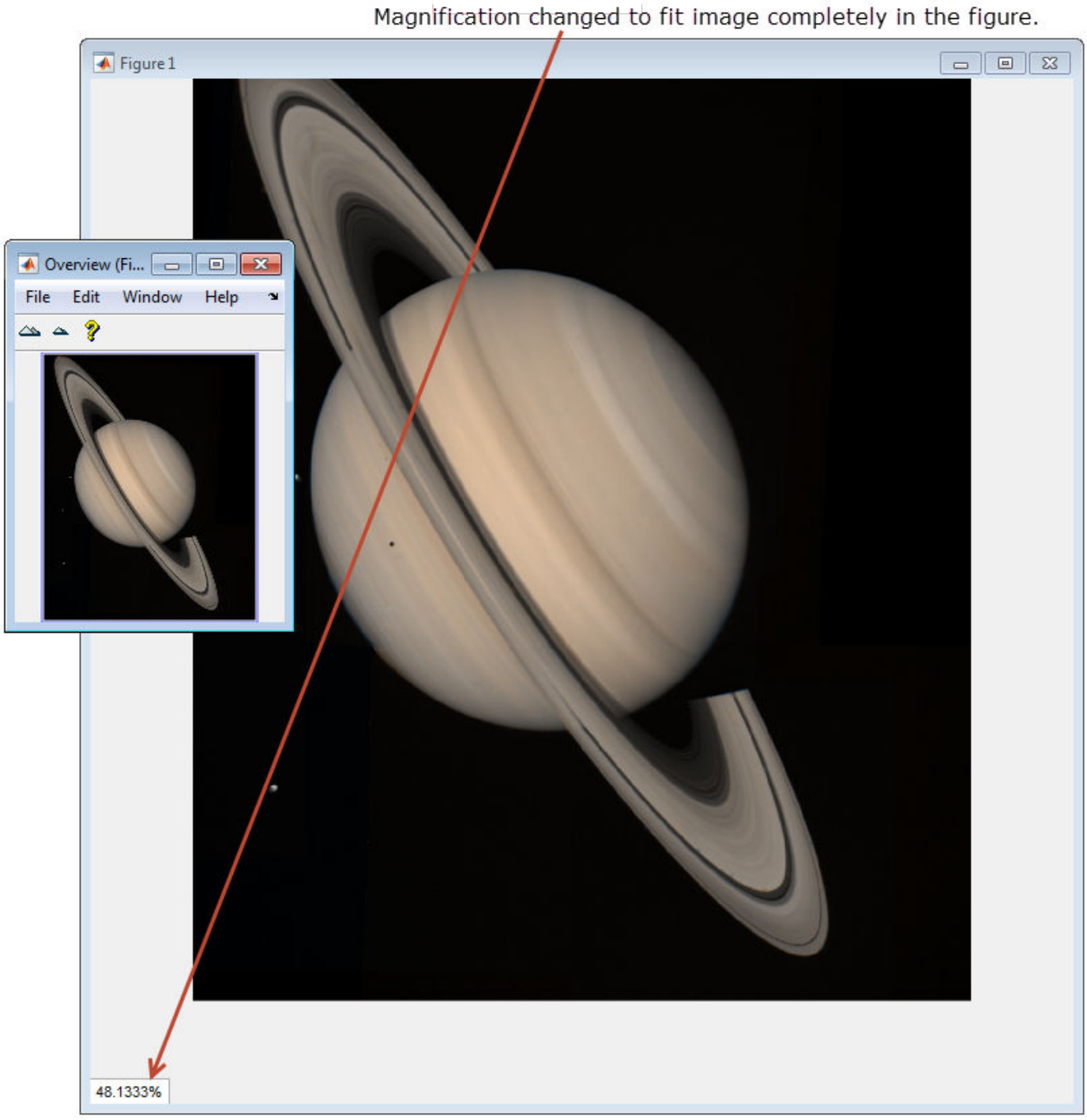
Use the scroll panel object API function `setVisibleLocation` to view the top left corner of the image.

```
api.setVisibleLocation(0.5,0.5)
```



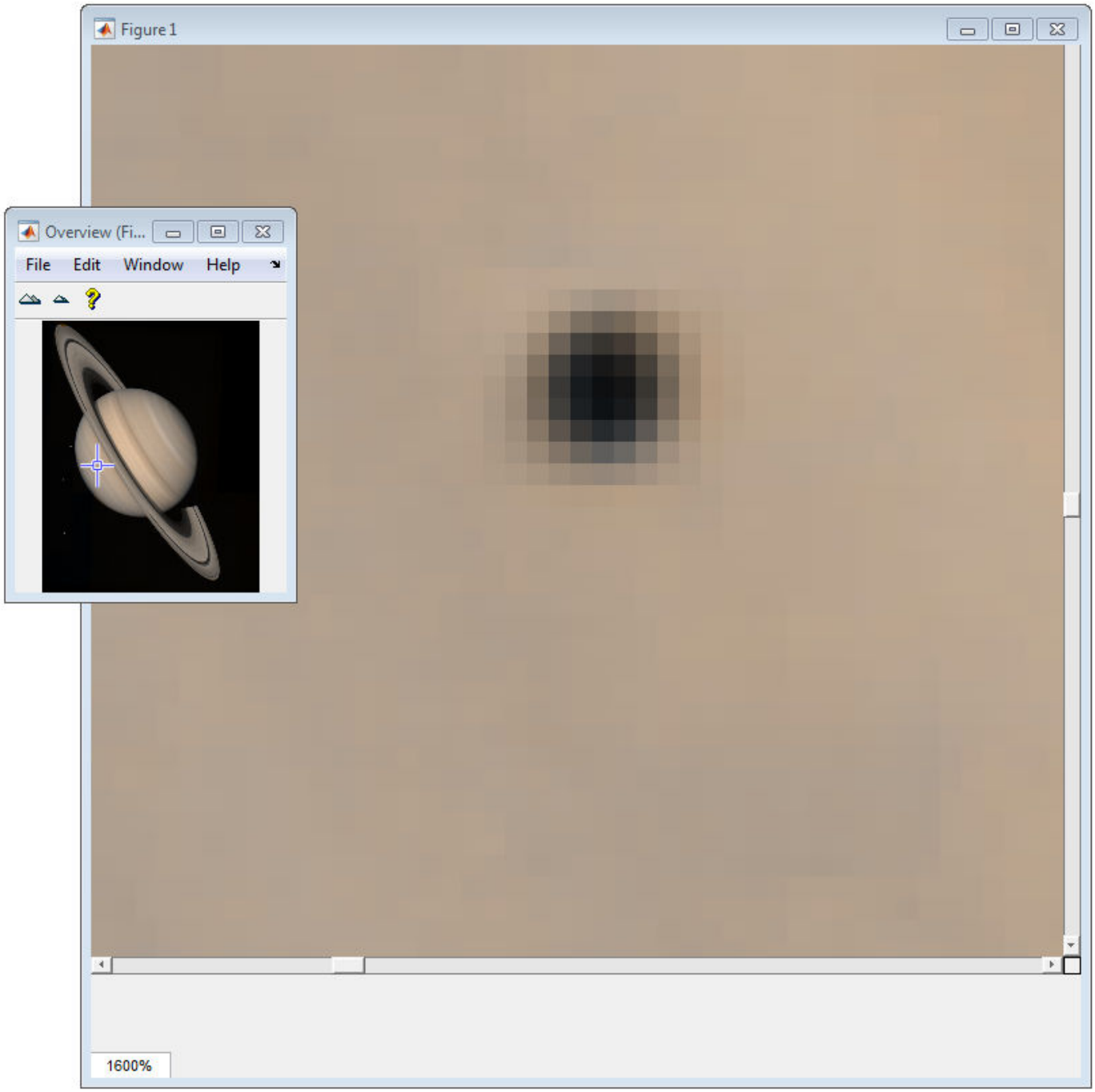
Change the magnification of the image so that the image fits entirely in the scroll panel. In the following figure, note that the scroll bars are no longer visible.

```
api.setMagnification(api.findFitMag())
```



Zoom in to 1600% on the dark spot.

```
api.setMagnificationAndCenter(16,306,800)
```



Input Arguments

`hparent` — Handle to figure or uipanel object

handle

Handle to a figure or uipanel object that contains the scroll panel, specified as a handle.

`himage` — Handle to target image

handle

Handle to target image, specified as a handle.

Output Arguments

`hpanel` — Handle to scroll panel

handle

Handle to scroll panel, returned as a handle. A scroll panel is a type of uipanel object.

More About

Scroll Panel API Structure

A scroll panel contains a structure of function handles, called an API. You can use the functions in this API to manipulate the scroll panel. To retrieve this structure, use the `iptgetapi` function, as in the following example.

```
api = iptgetapi(hpanel)
```

This table lists the scroll panel API functions, in the order they appear in the structure.

Function	Description
`setMagnification`	Set the magnification of the target image in units of screen pixels per image pixel. `mag = api.setMagnification(new_mag)` `new_mag` is a scalar magnification factor.
`getMagnification`	Return the current magnification factor of the target image in units of screen pixels per image pixel. `mag = api.getMagnification()` Multiply `mag` by 100 to convert to percentage. For example if `mag` is 2, then the magnification is 200%.
`setMagnificationAndCenter`	Change the magnification and make the point with (*x*,*y*) coordinate (`cx`,`cy`) in the target image appear in the center of the scroll panel. This operation is equivalent to a simultaneous zoom and recenter. `api.setMagnificationAndCenter(mag,cx,cy)`
`findFitMag`	Return the magnification factor that would make the target image just fit in the scroll panel. `mag = api.findFitMag()`

Function	Description

`setVisibleLocation`	Move the target image so that the specified location is visible, and update the scroll bars.
	
	`api.setVisibleLocation(xmin, ymin)`
	`api.setVisibleLocation([xmin ymin])`

`getVisibleLocation`	Return the location of the currently visible portion of the target image. `loc = api.getVisibleLocation()` `loc` is a vector `[xmin ymin]`.
`getVisibleImageRect`	Return the current visible portion of the image. `r = api.getVisibleImageRect()` `r` is a rectangle `[xmin ymin width height]`.

`addNewMagnificationCallback`	Add the function handle `fcn` to the list of new-magnification callback functions.
	
	`id = api.addNewMagnificationCallback(fcn)`
	
	Whenever the scroll panel magnification changes, each function in the list is called with the syntax:
	
	`fcn(mag)`
	
	`mag` is a scalar magnification factor.
	
	The return value, `id`, is used only with `removeNewMagnificationCallback`.

`removeNewMagnificationCallback`	Remove the corresponding function from the new-magnification callback list. `api.removeNewMagnificationCallback(id)` `id` is the identifier returned by `addNewMagnificationCallback`.

`addNewLocationCallback`	Add the function handle `fcn` to the list of new-location callback functions.
	
	`id = api.addNewLocationCallback(fcn)`
	
	Whenever the scroll panel location changes, each function in the list is called with the syntax:
	
	`fcn(loc)`
	
	`loc` is `[xmin ymin]`.
	
	The return value, `id`, is used only with `removeNewLocationCallback`.

Function	Description
`removeNewLocationCallback`	Remove the corresponding function from the new-location callback list. `api.removeNewLocationCallback(id)` `id` is the identifier returned by `addNewLocationCallback`.

`replaceImage`	`api.replaceImage(...,PARAM1,VAL1,PARAM2,VAL2,...)` replaces the image displayed in the scroll panel.
	
	`api.replaceImage(I)`
	`api.replaceImage(BW)`
	`api.replaceImage(RGB)`
	`api.replaceImage(I,MAP)`
	`api.replaceImage(filename)`
	
	By default, the new image data is displayed centered, at 100% magnification. The image handle is unchanged.
	
	The parameters you can specify include many of the parameters supported by `imshow`, including `'Colormap'`, `'DisplayRange'`, and `'InitialMagnification'`. In addition, you can use the `'PreserveView'` parameter to preserve the current magnification and centering of the image during replacement. Specify the logical scalar `True` to preserve current centering and magnification.

Tips

- `imscrollpanel` changes the object hierarchy of the target image. Instead of the familiar figure→axes→image object hierarchy, `imscrollpanel` inserts several uipanel and uicontrol objects between the figure and the axes object.
- Scrollbar navigation as provided by `imscrollpanel` is incompatible with the default MATLAB figure navigation buttons (pan, zoom in, zoom out). The corresponding menu items and toolbar buttons should be removed in a custom GUI that includes a scrollable uipanel created by `imscrollpanel`.
- When you run `imscrollpanel`, it appears to take over the entire figure because, by default, an uipanel object has `'Units'` set to `'normalized'` and `'Position'` set to `[0 0 1 1]`. If you want to see other children of `hparent` while using your new scroll panel, you must manually set the `'Position'` property of `hpanel`.

See Also

Image Viewer | `immagbox` | `imoverview` | `imoverviewpanel` | `iptgetapi`

Topics

“Add Navigation Aids”

Introduced before R2006a

imsegfmm

Binary image segmentation using fast marching method

Syntax

```
BW = imsegfmm(W,mask,thresh)
BW = imsegfmm(W,C,R,thresh)
BW = imsegfmm(W,C,R,P,thresh)
[BW,D] = imsegfmm( ___ )
```

Description

`BW = imsegfmm(W,mask,thresh)` returns a segmented image `BW`, which is computed using the fast marching method. The array `W` specifies weights for each pixel. `mask` is a logical array that specifies seed locations. `thresh` specifies the threshold level.

`BW = imsegfmm(W,C,R,thresh)` returns a segmented image, with seed locations specified by the vectors `C` and `R`, which contain column and row indices. `C` and `R` must contain values which are valid pixel indices in `W`.

`BW = imsegfmm(W,C,R,P,thresh)` returns a segmented image, with seed locations specified by the vectors `C`, `R`, and `P`, which contain column, row, and plane indices. `C`, `R`, and `P` must contain values which are valid pixel indices in `W`.

`[BW,D] = imsegfmm( ___ )` returns the normalized geodesic distance map `D` computed using the fast marching method. `BW` is a thresholded version of `D`, where all the pixels that have normalized geodesic distance values less than `thresh` are considered foreground pixels and set to `true`. `D` can be thresholded at different levels to obtain different segmentation results.

Examples

Segment Image Using Fast Marching Method Algorithm

This example shows how to segment an object in an image using Fast Marching Method based on differences in grayscale intensity as compared to the seed locations.

Read image.

```
I = imread('cameraman.tif');
imshow(I)
title('Original Image')
```

Create mask and specify seed location. You can also use `roipoly` to create the mask interactively.

```
mask = false(size(I));
mask(170,70) = true;
```

Compute the weight array based on grayscale intensity differences.

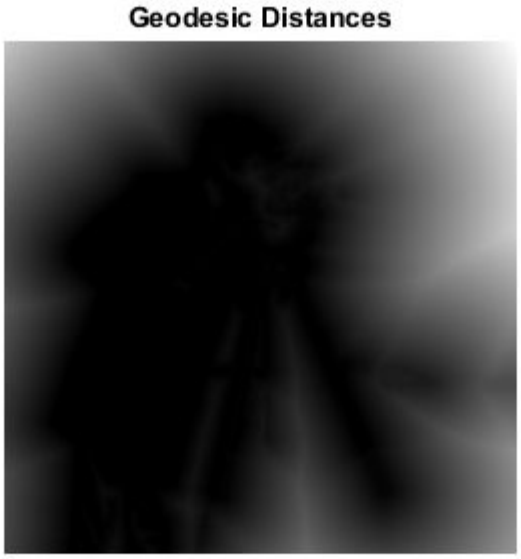
```
W = graydiffweight(I, mask, 'GrayDifferenceCutoff', 25);
```

Segment the image using the weights.

```
thresh = 0.01;
[BW, D] = imsegfmm(W, mask, thresh);
figure
imshow(BW)
title('Segmented Image')
```

You can threshold the geodesic distance matrix `D` using different thresholds to get different segmentation results.

```
figure
imshow(D)
title('Geodesic Distances')
```



Segment Object in Volume Based on Intensity Differences

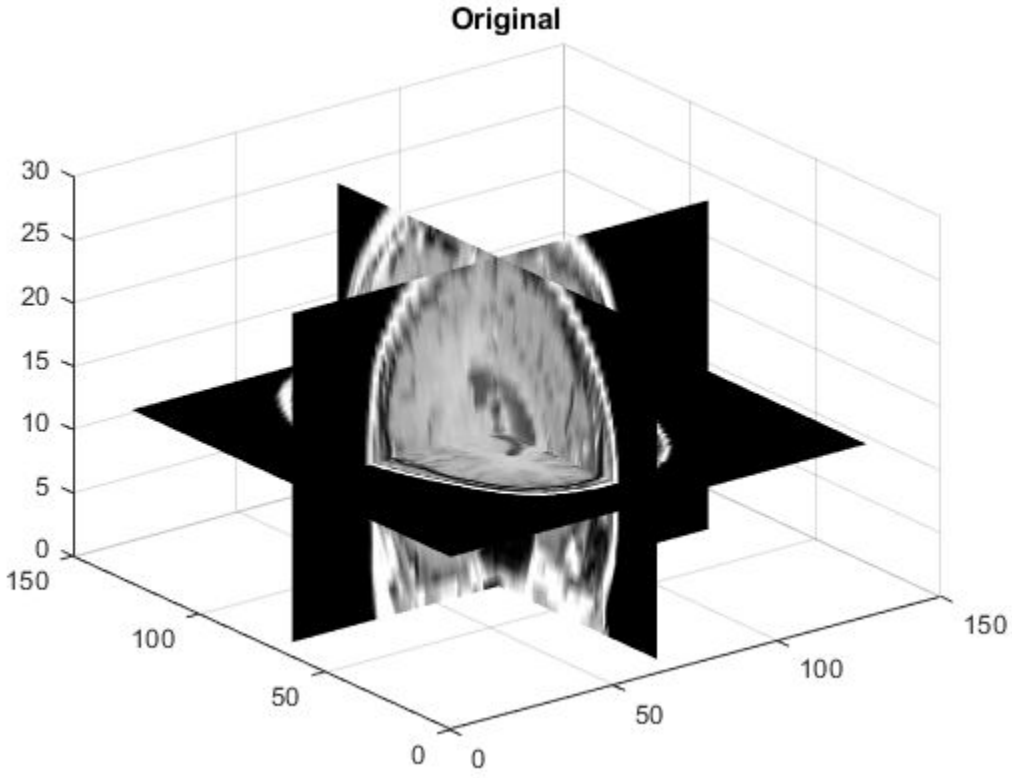
This example segments the brain from MRI data of the human head.

Load the MRI data.

```
load("mri")
V = squeeze(D);
```

Visualize the data.

```
sizeO = size(V);
figure
slice(double(V),sizeO(2)/2,sizeO(1)/2,sizeO(3)/2);
shading interp
colormap("gray")
title("Original")
```



Set the seed locations.

```
seedR = 75;
seedC = 60;
seedP = 10;
```

Compute weights based on grayscale intensity differences.

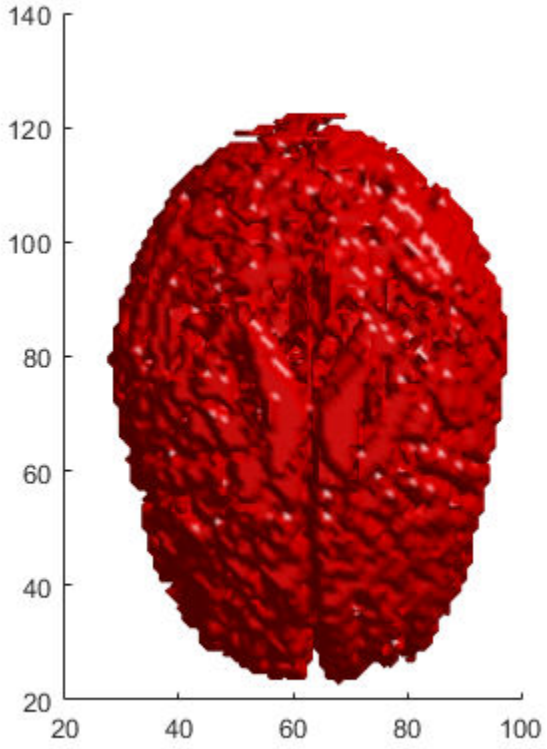
```
W = graydiffweight(V,seedC,seedR,seedP,"GrayDifferenceCutoff",25);
```

Segment the image using the weights.

```
thresh = 0.002;
BW = imsegfmm(W,seedC,seedR,seedP,thresh);
```

Visualize the segmented image using an isosurface.

```
figure
p = patch(isosurface(double(BW)));
p.FaceColor = "red";
p.EdgeColor = "none";
daspect([1 1 27/64]);
camlight
lighting phong
```



Input Arguments

`W` — Weight array

non-negative numeric array

Weight array, specified as a non-negative numeric array. You can compute the weight array by using the `graydiffweight` or `gradientweight` functions. Large values in `W` identify the foreground (object) and small values identify the background.

Data Types: `single` | `double` | `uint8` | `int8` | `int16` | `uint16` | `int32` | `uint32`

`mask` — Seed locations mask
logical array

Seed locations mask, specified as a logical array of the same size as `W`. Locations where `mask` is `true` are seed locations. If you use `graydiffweight` to create the weight matrix `W`, it is recommended that you use the same value of `mask` with `imsegfmm` that you used with `graydiffweight`.

Data Types: `logical`

`thresh` — Threshold
number in the range [0, 1]

Threshold level used to obtain the binary image, specified as a number in the range [0, 1]. Low values typically result in large foreground regions (logical true) in `BW`, and high values produce small foreground regions.

Example: `0.5`

Data Types: `double`

`C` — Column index of reference pixels
numeric vector

Column index of reference pixels, specified as a numeric vector.

Example: `[50 75 93]`

Data Types: `double`

`R` — Row index of reference pixels
numeric vector

Row index of reference pixels, specified as a numeric vector.

Example: `[48 71 89]`

Data Types: `double`

`P` — Plane index of reference pixels
numeric vector

Plane index of reference pixels, specified as a numeric vector.

Example: `[2 4 7]`

Data Types: `double`

Output Arguments

`BW` — Segmented image
logical array

Segmented image, returned as a logical array of the same size as `W`.

Data Types: `logical`

D — Normalized geodesic distance map
numeric array

Normalized geodesic distance map, returned as a numeric array of the same size as `W`. If `W` is of class `single`, then `D` is of class `single`. Otherwise, `D` is of class `double`.

Data Types: `double` | `single`

Tips

- `imsegfmm` uses double-precision floating point operations for internal computations for all classes except class `single`. If `W` is of class `single`, `imsegfmm` uses single-precision floating point operations internally.
- `imsegfmm` sets pixels with `0` or `NaN` weight values to `Inf` in the geodesic distance image `D`. These pixels are part of the background (logical false) in the segmented image `BW`.

References

[1] Sethian, J. A. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*, Cambridge University Press, 2nd Edition, 1999.

See Also

`activecontour` | `graydist` | `graydiffweight` | `gradientweight` | **Image Segmenter**

Introduced in R2014b

imseggeodesic

Segment image into two or three regions using geodesic distance-based color segmentation

Syntax

```
L = imseggeodesic(RGB,BW1,BW2)
L = imseggeodesic(RGB,BW1,BW2,BW3)
[L,P] = imseggeodesic(___)
[L,P] = imseggeodesic(___,Name,Value)
```

Description

`L = imseggeodesic(RGB,BW1,BW2)` segments the color image `RGB`, returning a segmented binary image with labels `L`. `BW1` and `BW2` are binary images that specify the location of the initial seed regions, called scribbles, for the two regions (foreground and background).

`imseggeodesic` uses the scribbles specified in `BW1` and `BW2` as representative samples for computing the statistics for their respective regions, which it then uses in segmentation. The scribbles specified by `BW1` and `BW2` (regions that are logical true) should not overlap. The underlying algorithm uses the statistics estimated over the regions marked by the scribbles for segmentation. The greater the number of pixels marked by scribbles, the more accurate the estimation of the region statistics, which typically leads to more accurate segmentation. Therefore, it is a good practice to provide as many scribbles as possible. Typically, provide at least a few hundred pixels as scribbles for each region.

`L = imseggeodesic(RGB,BW1,BW2,BW3)` segments the color image `RGB`, returning a segmented image with three segments (trinary segmentation) with the region labels specified by label matrix `L`. `BW1`, `BW2`, and `BW3` are binary images that specify the location of the initial seed regions or scribbles for the three regions. The scribbles specified by `BW1`, `BW2`, and `BW3` (regions that are logical true) should not overlap.

`[L,P] = imseggeodesic(___)` also returns the probability for each pixel belonging to each of the labels in matrix `P`.

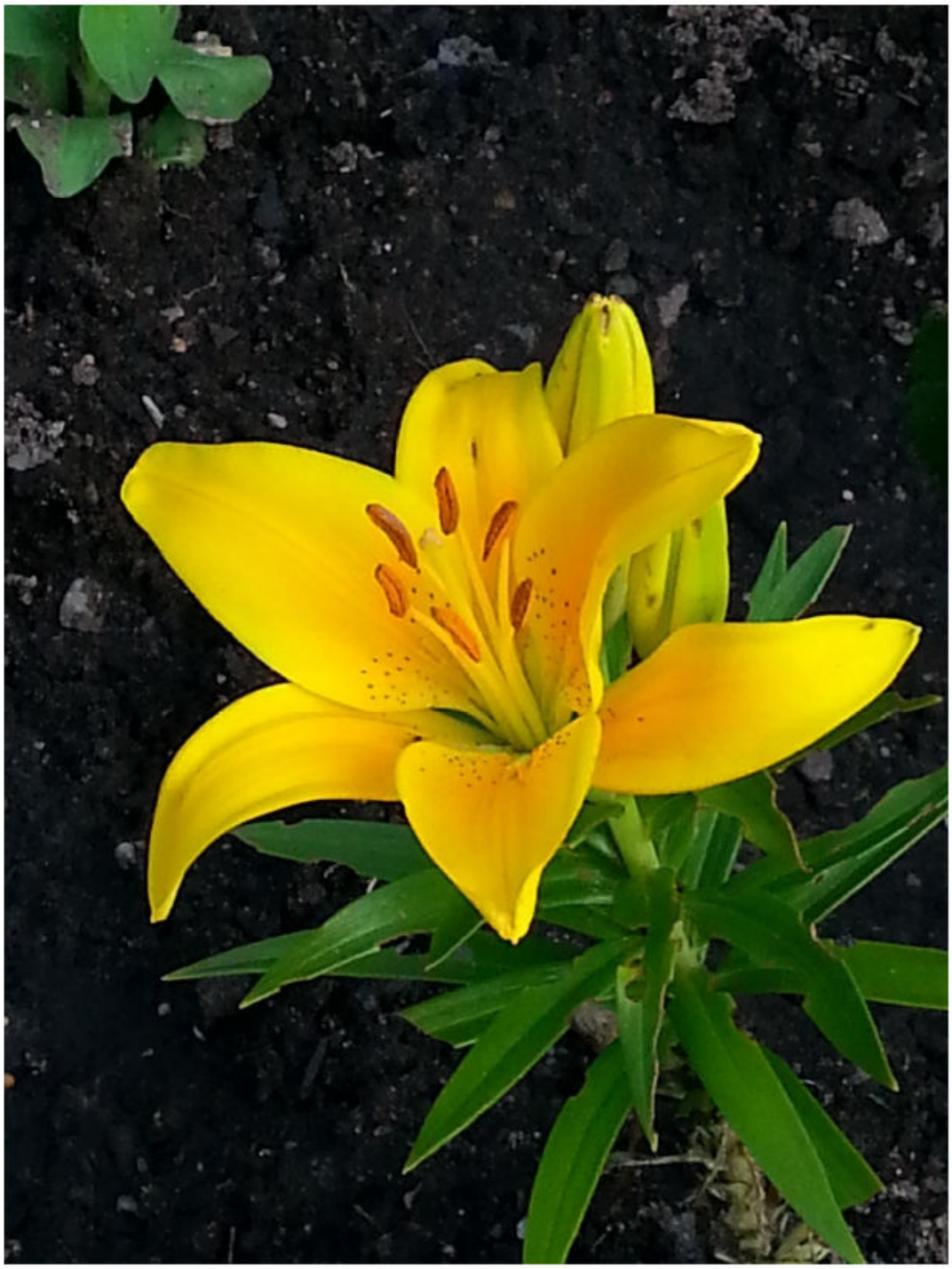
`[L,P] = imseggeodesic(___,Name,Value)` uses name-value pairs to control aspects of segmentation.

Examples

Segment Image into Two Regions Using Color Information

Read and display an image.

```
RGB = imread('yellowlily.jpg');
imshow(RGB)
```

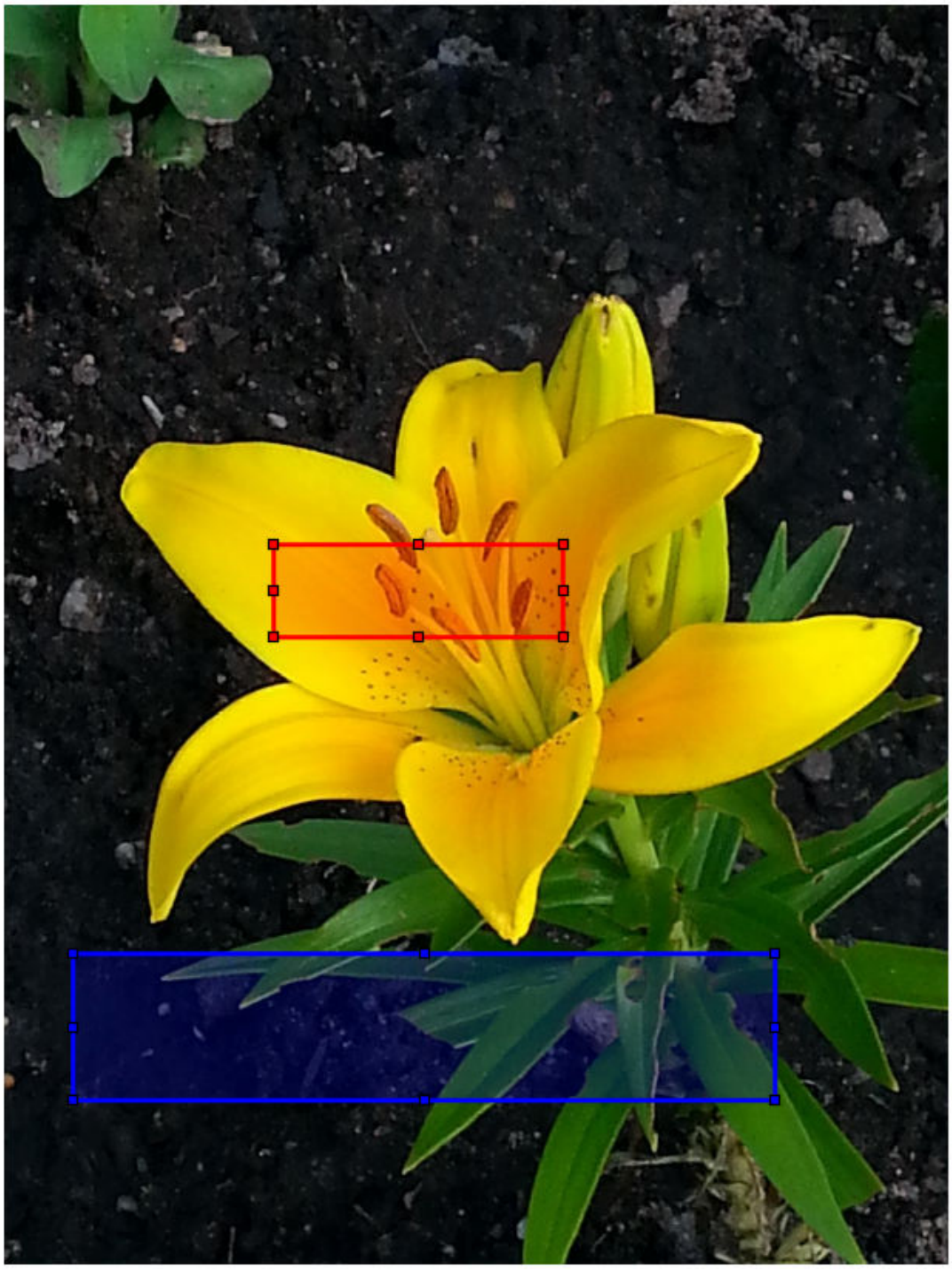


The goal is to segment the petals of the flower. Specify the initial seed region as a rectangular ROI by using the `drawrectangle` function. Display the ROI in red. The 'Position' name-value pair argument specifies the upper left coordinate, width, and height of the ROI as the 4-element vector `[xmin, ymin, width, height]`. If you want to draw the rectangle interactively, then omit the 'Position' name-value pair argument.

```
roiObject = drawrectangle(gca,'Position',[350 700 375 120],'Color','r');
```

Specify the initial seed regions for the background as a rectangular ROI. Display the ROI in blue.

```
roiBackground = drawrectangle(gca,'Position',[90 1230 910 190],'Color','b');
```



Create a mask for each ROI in which the ROI is `true` and other pixels are `false`.

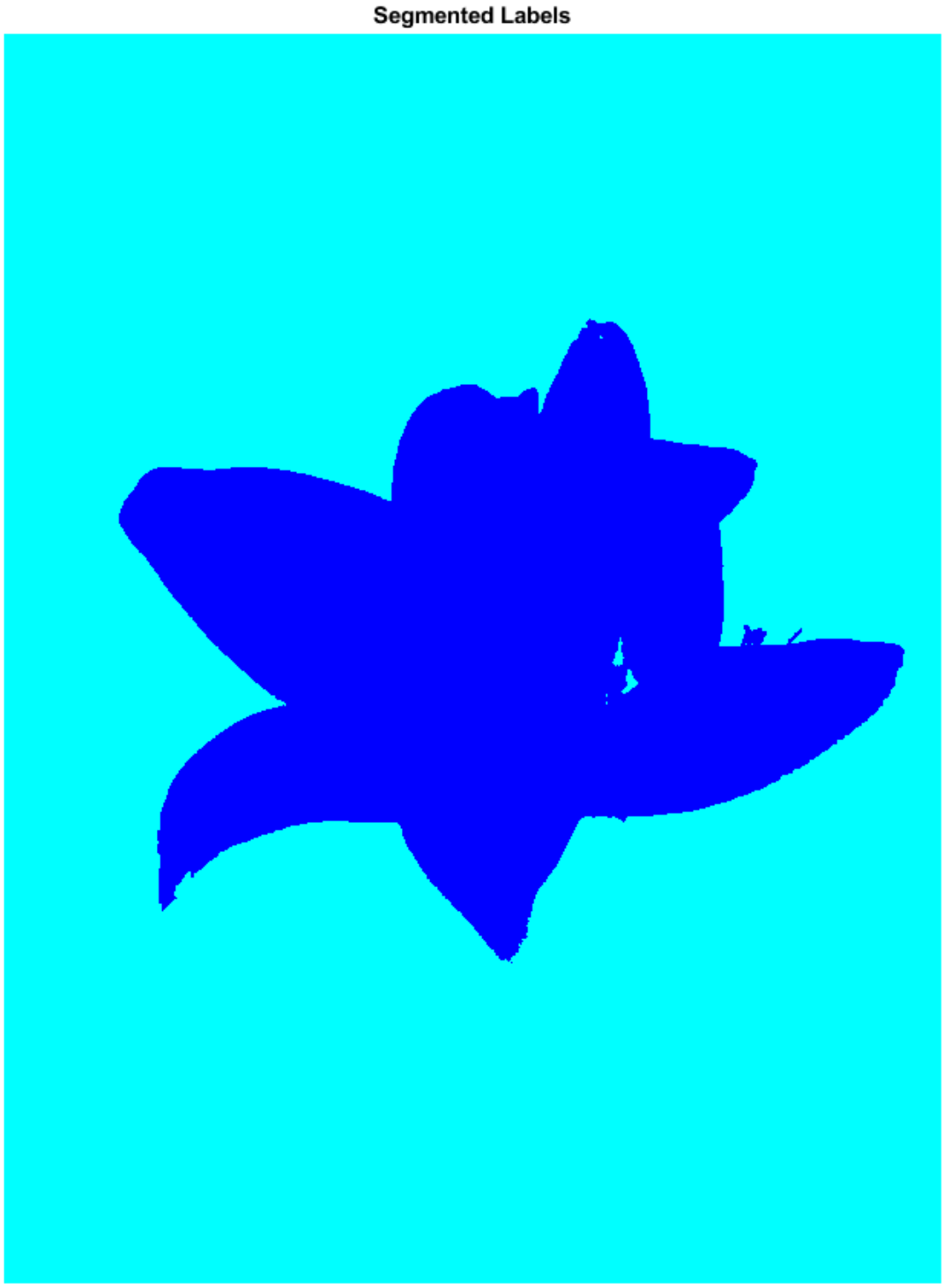
```
maskObject = createMask(roiObject);
maskBackground = createMask(roiBackground);
```

Segment the image.

```
[L,P] = imseggeodesic(RGB,maskObject,maskBackground);
```

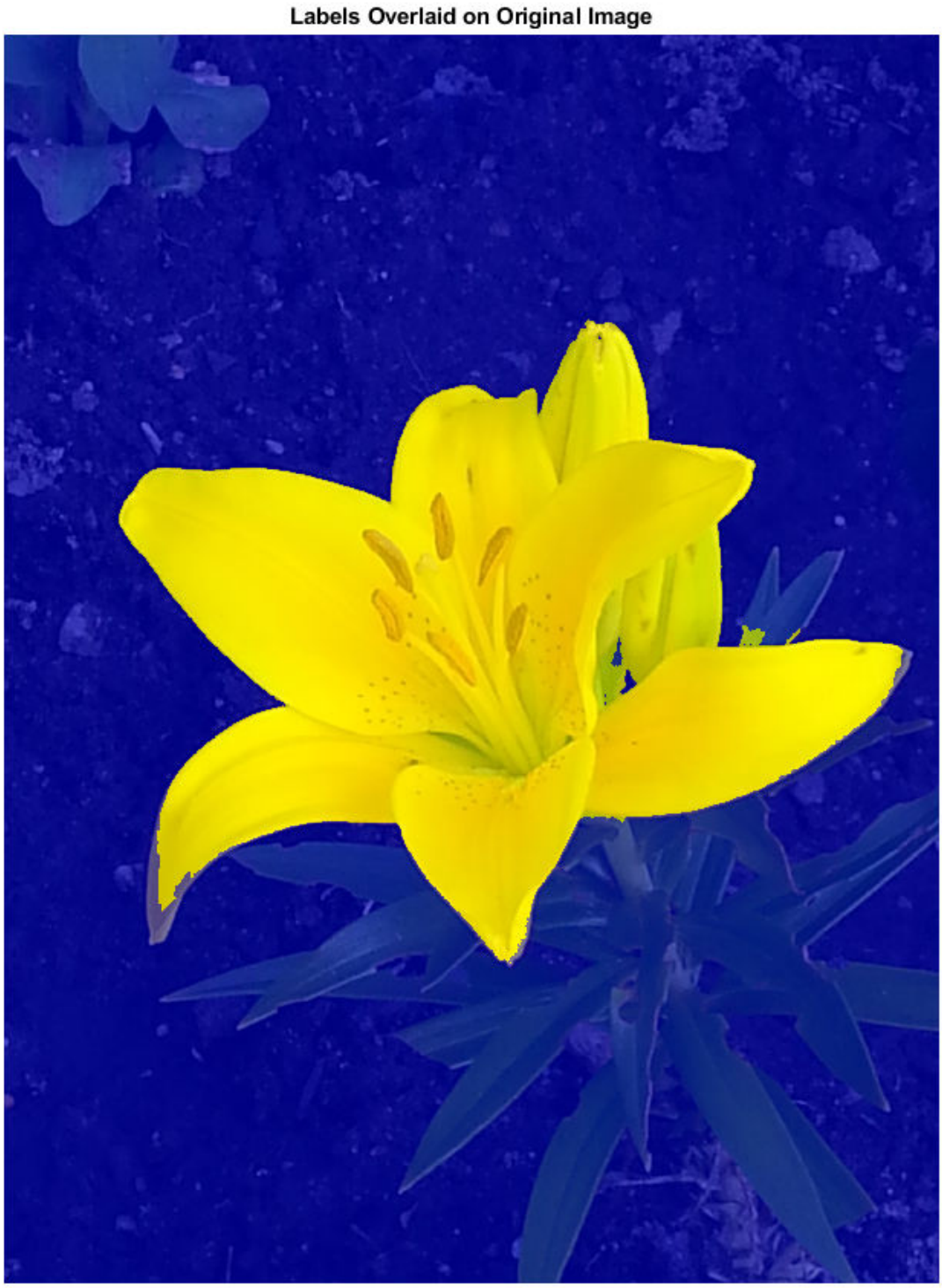
Display the segmented labels.

```
imshow(label2rgb(L))
title('Segmented Labels')
```



Display the segmented labels over the original image.

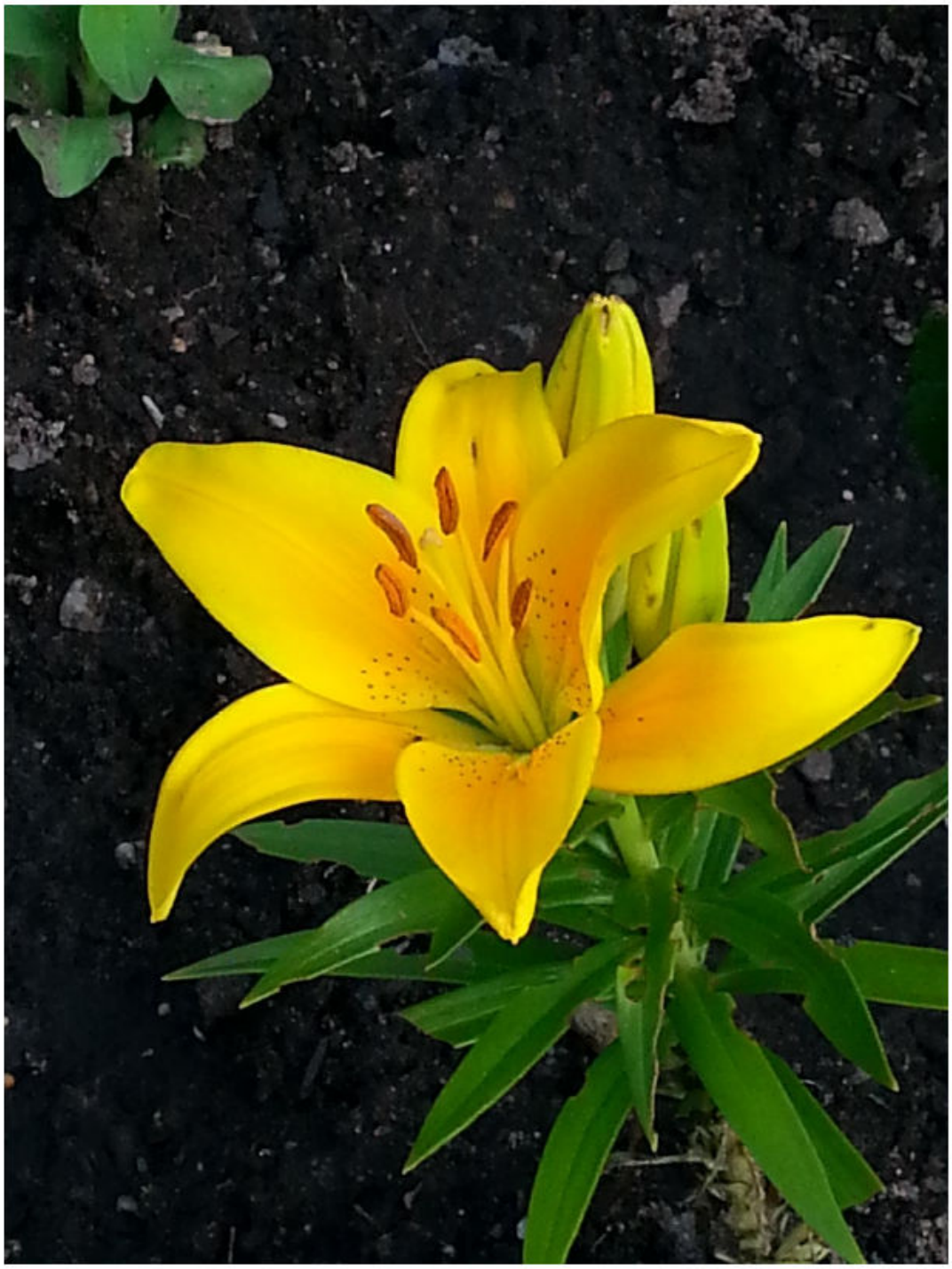
```
imshow(labeloverlay(RGB,L))
title('Labels Overlaid on Original Image')
```



Segment Image into Three Regions Using Color Information

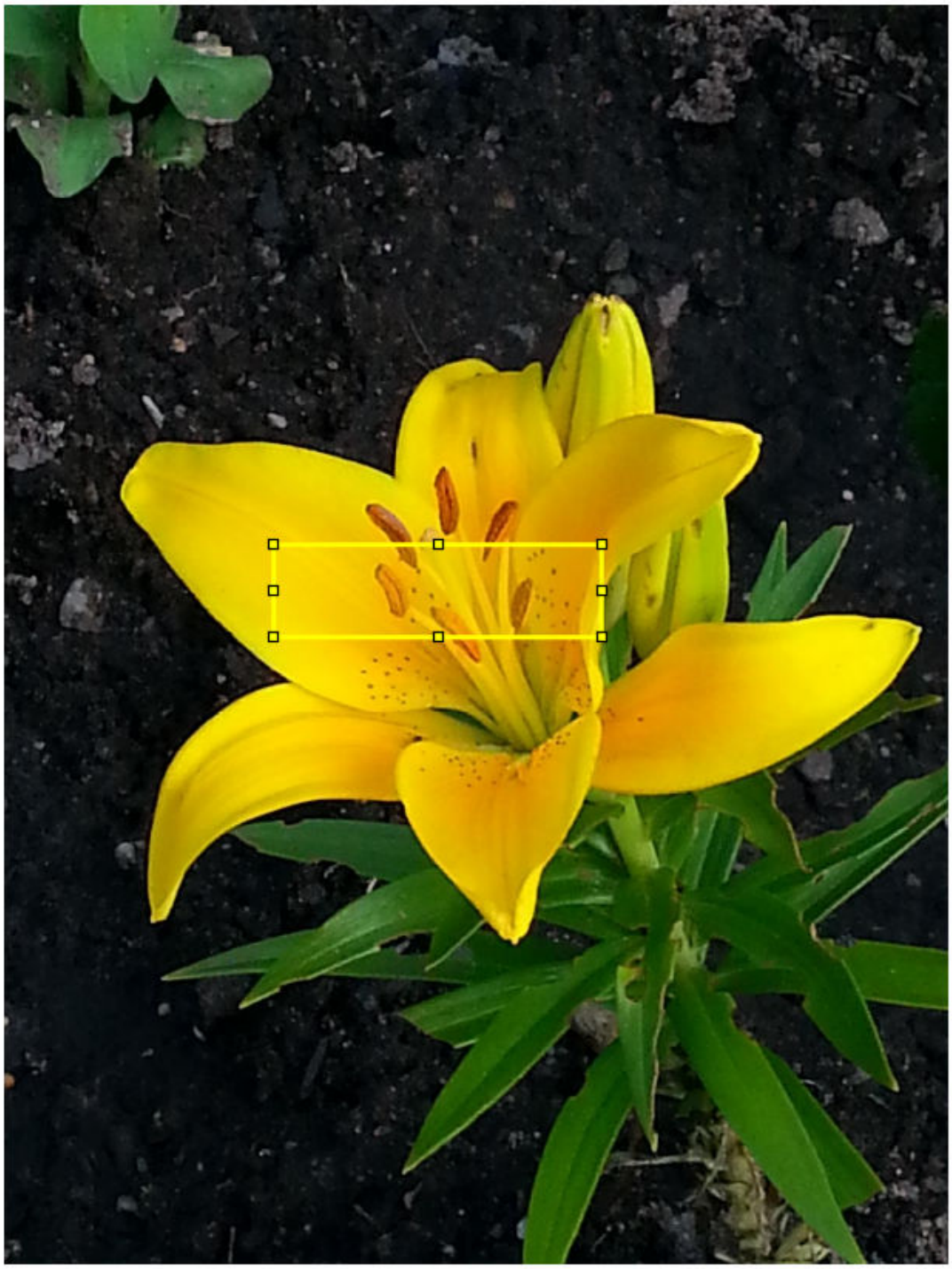
Read and display an image.

```
RGB = imread('yellowlily.jpg');
imshow(RGB)
```



The first region consists of the yellow flower petals. Specify the initial seed region as a rectangular ROI by using the `drawrectangle` function. Draw the ROI in yellow. The 'Position' name-value pair argument specifies the upper left coordinate, width, and height of the ROI as the 4-element vector `[xmin, ymin, width, height]`. If you want to draw the rectangle interactively, then omit the 'Position' name-value pair argument.

```
r1 = drawrectangle(gca,'Position',[350 700 425 120],'Color','y');
```

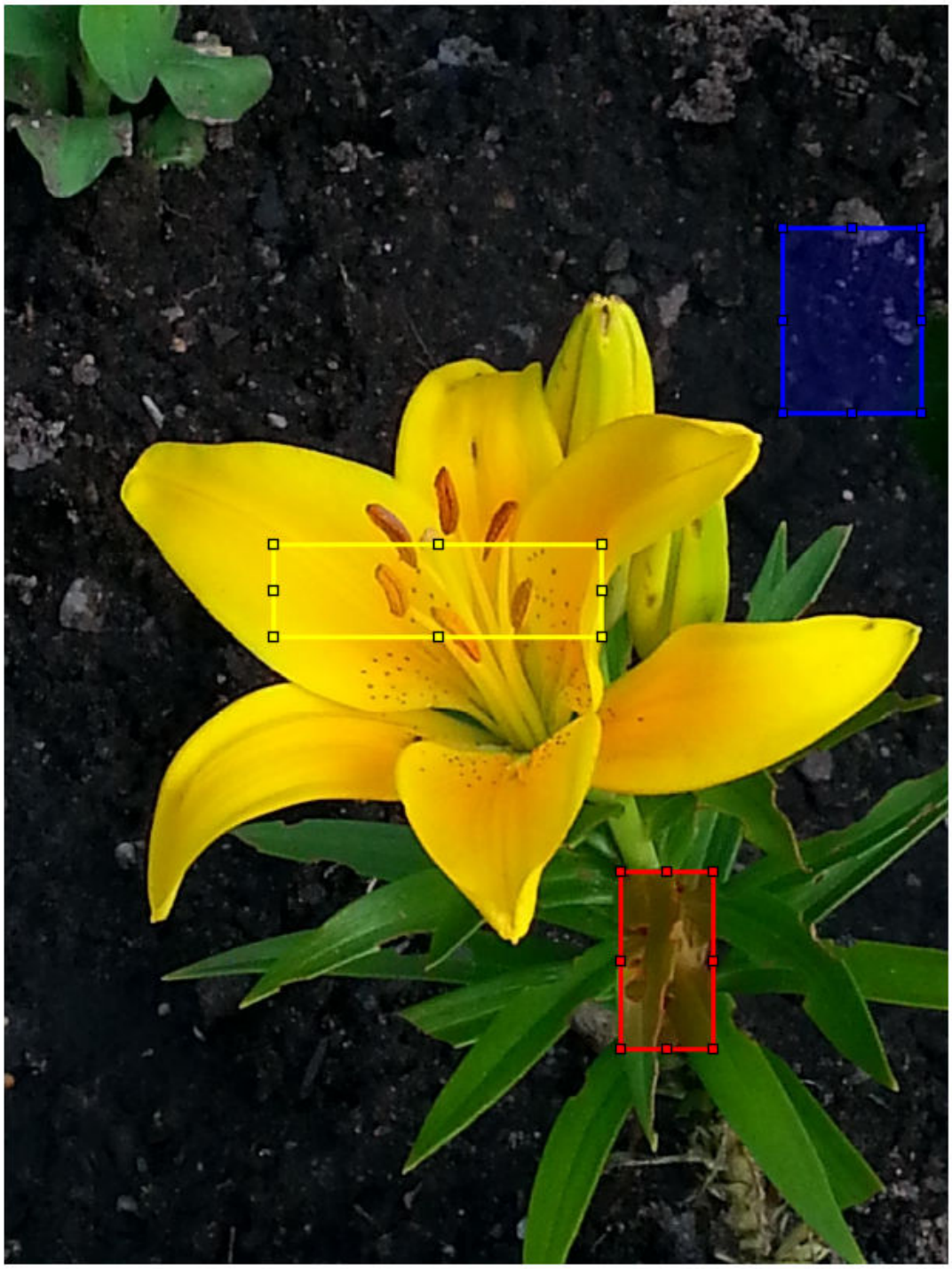


The second region consists of the green leaves. Specify the seed region as a rectangular ROI and draw the ROI in red.

```
r2 = drawrectangle(gca,'Position',[800 1124 120 230],'Color','r');
```

The third region is background, which is the dirt in this image. Specify the seed region as a rectangular ROI and draw the ROI in blue.

```
r3 = drawrectangle(gca,'Position',[1010 290 180 240],'Color','b');
```



Create a mask for each ROI in which the ROI is `true` and other pixels are `false`.

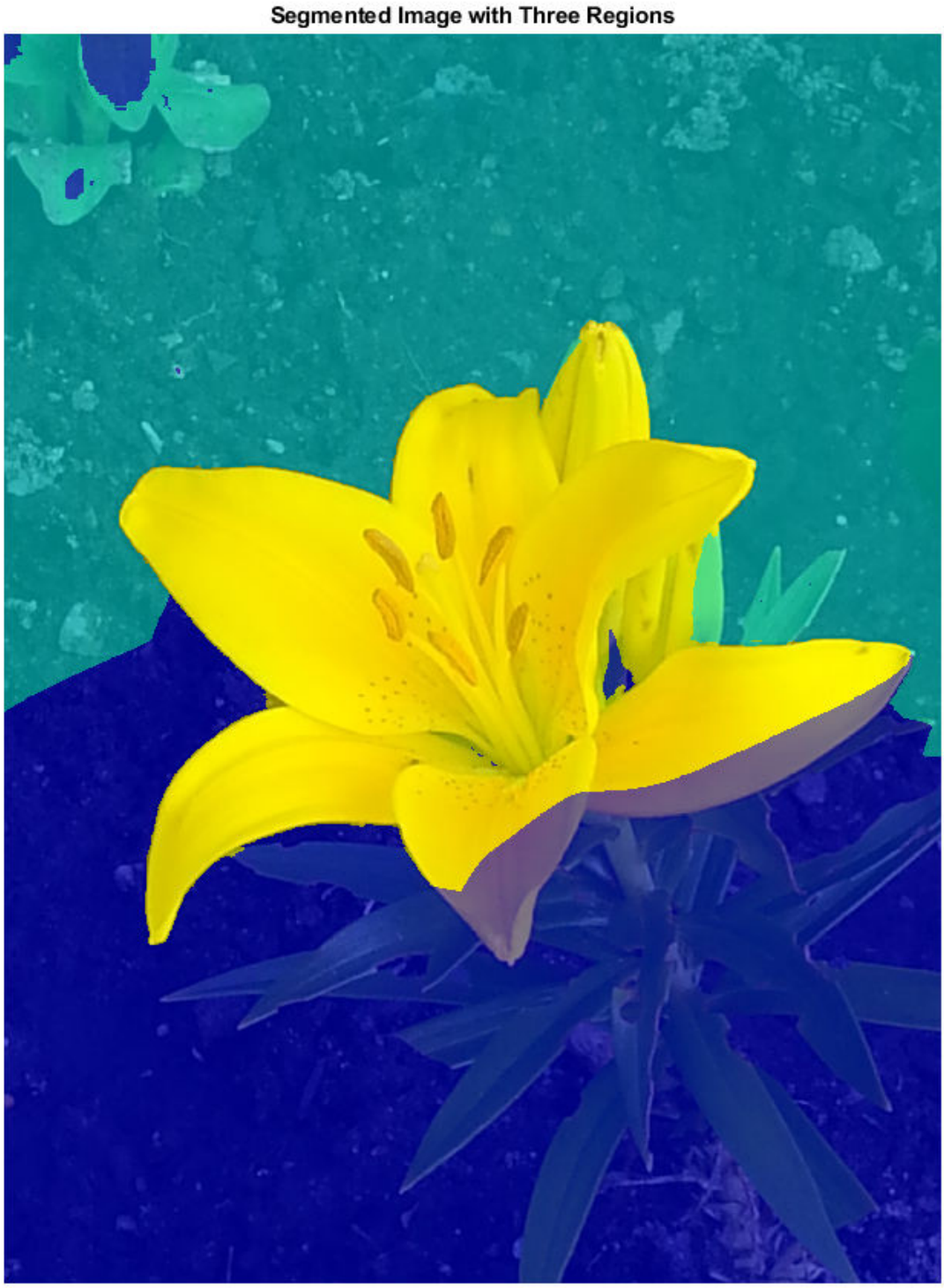
```
mask1 = createMask(r1);
mask2 = createMask(r2);
mask3 = createMask(r3);
```

Segment the image.

```
[L,P] = imseggeodesic(RGB,mask1,mask2,mask3,'AdaptiveChannelWeighting',true);
```

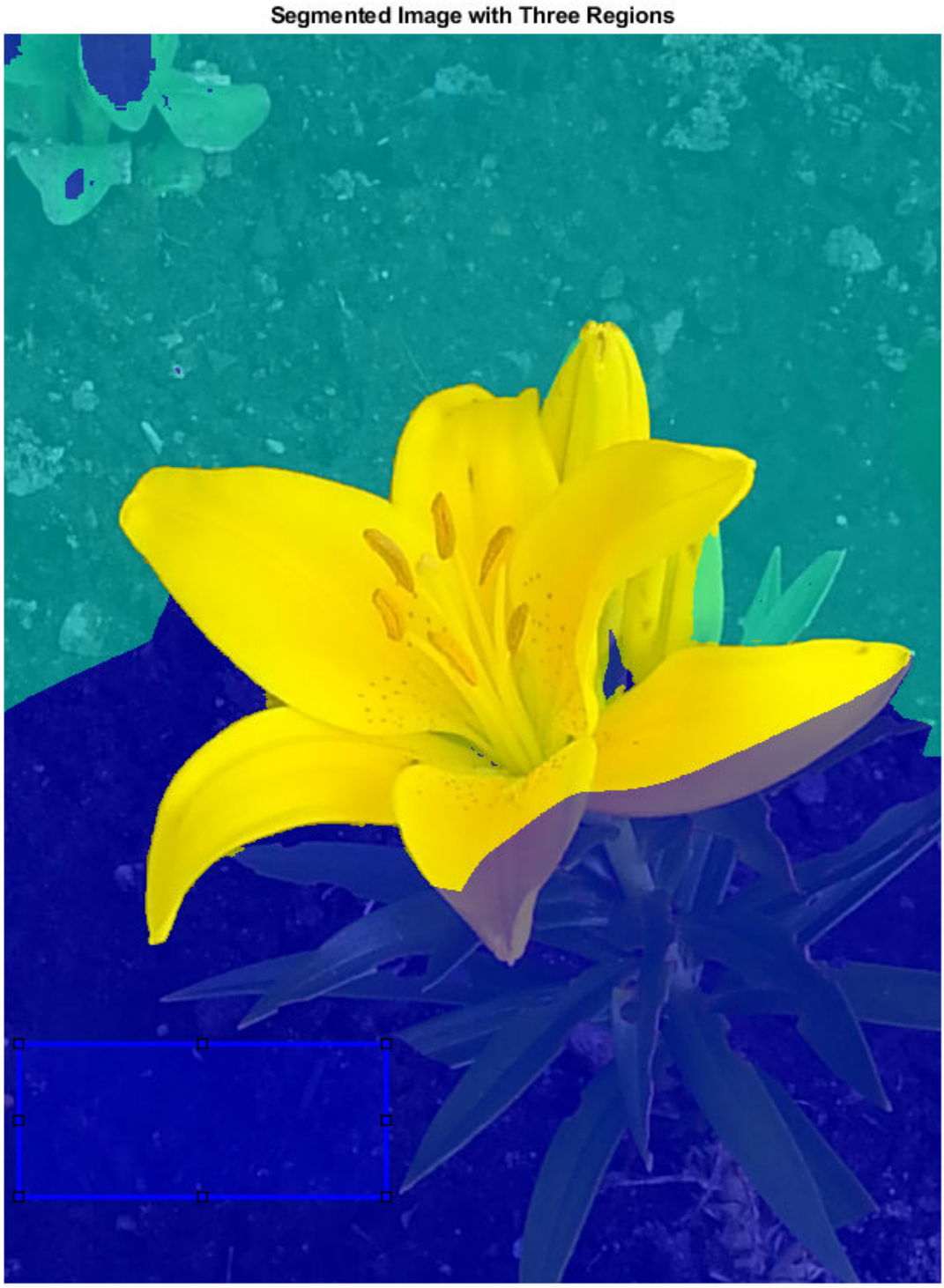
Display the segmented labels over the original image.

```
imshow(labeloverlay(RGB,L))
title('Segmented Image with Three Regions')
```


Segmented Image with Three Regions

The lower right corner of the image is misclassified as region 2 (leaves). Add another background ROI.

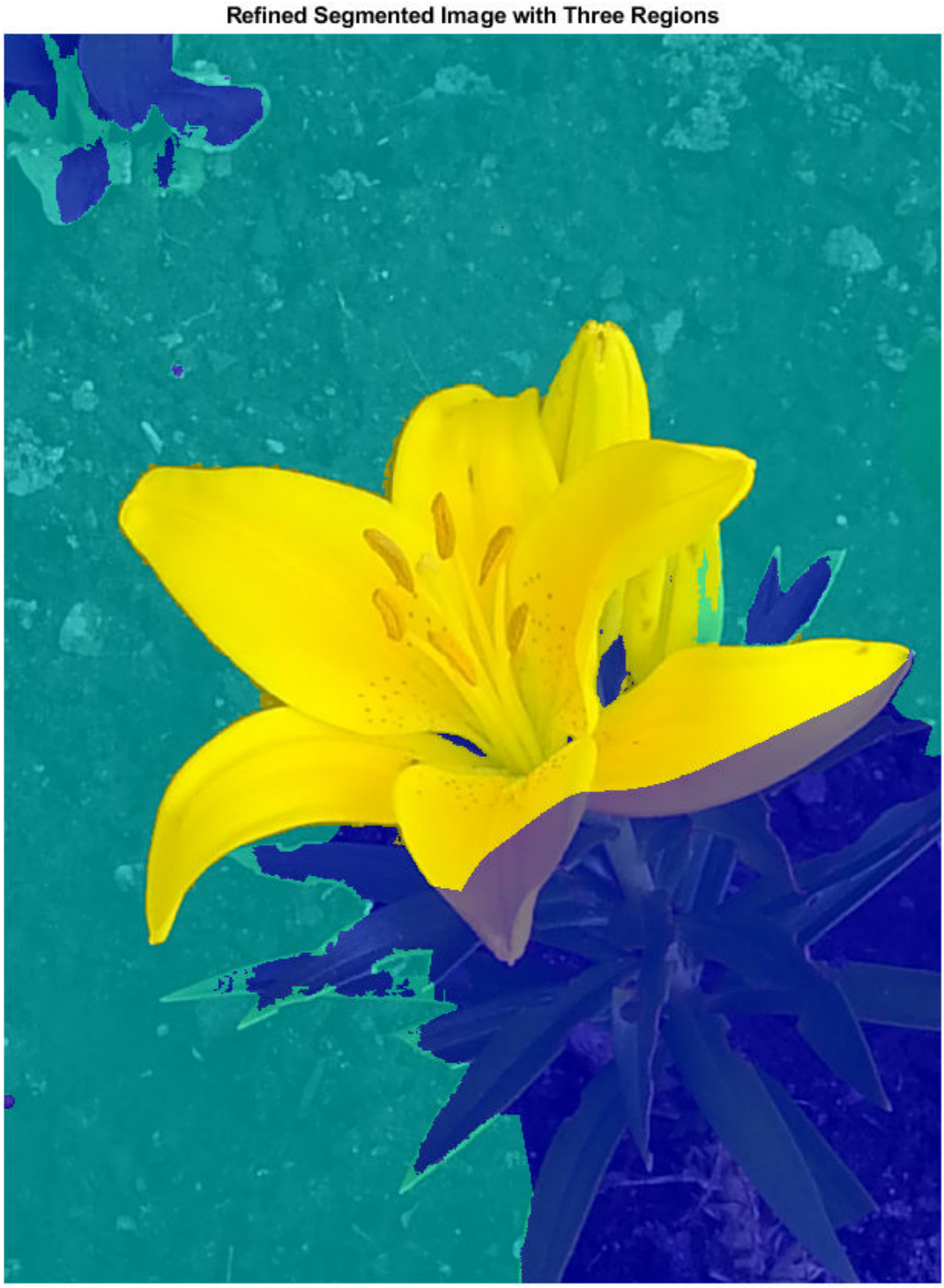
```
r4 = drawrectangle(gca,'Position',[20 1320 480 200],'Color','b');
```




```
mask4 = createMask(r4);
maskBackground = mask3 + mask4;
```

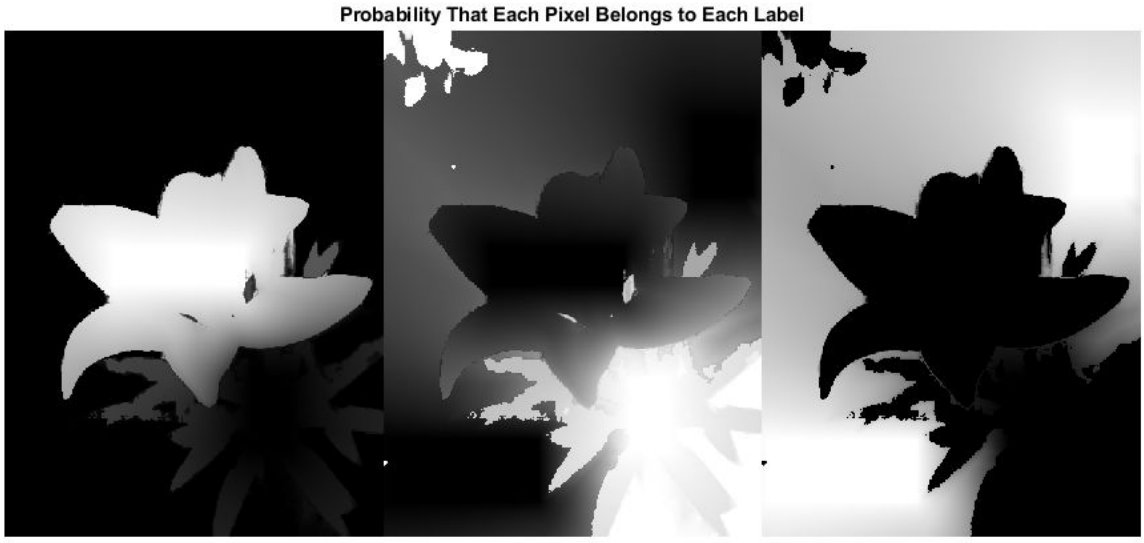
Segment the image then display the segmented labels over the original image.

```
[L,P] = imseggeodesic(RGB,mask1,mask2,maskBackground,'AdaptiveChannelWeighting',true);
imshow(labeloverlay(RGB,L))
title('Refined Segmented Image with Three Regions')
```



Display the probability that each pixel is belongs to each label.

```
montage(P,'Size',[1 3])
title('Probability That Each Pixel Belongs to Each Label')
```



Input Arguments

RGB — Image to be segmented

RGB image

Image to be segmented, specified as an RGB image. `imseggeodesic` converts the input RGB image to the YCbCr color space before performing the segmentation.

Example: `RGB = imread('peppers.png');`

Data Types: `double` | `uint8` | `uint16`

BW1 — Scribble image for first region

logical matrix

Scribble image for the first region, specified as a logical matrix. `BW1` must have the same number of rows and columns as the input image `RGB`. To create the scribbles interactively, first draw an ROI using functions such as `drawcircle`, `drawfreehand`, `drawpolygon`, or `drawrectangle`. Then, create a mask from the ROI using `createMask`.

Data Types: `logical`

BW2 — Scribble image for second region

logical matrix

Scribble image for the second region, specified as a logical matrix. `BW2` must have the same number of rows and columns as the input image `RGB`. To create the scribbles interactively, first draw an ROI using functions such as `drawcircle`, `drawfreehand`, `drawpolygon`, or `drawrectangle`. Then, create a mask from the ROI using `createMask`.

Data Types: `logical`

BW3 — Scribble image for third region
logical matrix

Scribble image for the third region, specified as a logical matrix. `BW3` must have the same number of rows and columns as the input image `RGB`. To create the scribbles interactively, first draw an ROI using functions such as `drawcircle`, `drawfreehand`, `drawpolygon`, or `drawrectangle`. Then, create a mask from the ROI using `createMask`.

Data Types: `logical`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `[L,P] = imseggeodesic(RGB,BW1,BW2,BW3,'AdaptiveChannelWeighting',true);`

AdaptiveChannelWeighting — Use adaptive channel weighting
`false` (default) | `true`

Use adaptive channel weighting, specified as a logical scalar. When `true`, `imseggeodesic` weights the channels proportional to the amount of discriminatory information they have that is useful for segmentation (based on the scribbles provided as input). When `false` (the default), `imseggeodesic` weights all the channels equally.

Data Types: `logical`

Output Arguments

L — Label matrix
matrix of nonnegative integers

Label matrix, returned as a matrix of nonnegative integers. Pixels labeled `0` are the background and pixels labeled `1` identify a segmented region. Pixels labeled `2` identify another segmented region in trinary segmentation.

Data Types: `double`

P — Probability a pixel belongs to a labeled region
M-by-*N*-by-2 matrix (binary segmentation) | *M*-by-*N*-by-3 matrix (trinary segmentation)

Probability a pixel belongs to a labeled region, specified as an *M*-by-*N*-by-2 matrix for binary segmentation or an *M*-by-*N*-by-3 matrix for trinary segmentation. *M* and *N* are the number of rows and columns in the input image. `P(i,j,k)` specifies the probability of pixel at location `(i,j)` belonging to label `k`.

Data Types: `double`

Tips

- The scribbles for the two (or three) regions should not overlap each other. Each scribble matrix (`BW1`, `BW2`, and `BW3`) should be nonempty, that is, there should be at least one pixel (although the more the better) marked as logical `true` in each of the scribbles.

Algorithms

`imseggeodesic` uses a geodesic distance-based color segmentation algorithm (similar to [1] on page 1-1975).

References

[1] A. Protiere and G. Sapiro, *Interactive Image Segmentation via Adaptive Weighted Distances*, IEEE Transactions on Image Processing. Volume 16, Issue 4, 2007.

See Also

`activecontour` | **Color Thresholder** | `imsegfmm` | `rgb2ycbcr`

Introduced in R2015a

imsegkmeans

K-means clustering based image segmentation

Syntax

```
L = imsegkmeans(I,k)
[L,centers] = imsegkmeans(I,k)
L = imsegkmeans(I,k,Name,Value)
```

Description

`L = imsegkmeans(I,k)` segments image `I` into `k` clusters by performing k-means clustering and returns the segmented labeled output in `L`.

`[L,centers] = imsegkmeans(I,k)` also returns the cluster centroid locations, `centers`.

`L = imsegkmeans(I,k,Name,Value)` uses name-value arguments to control aspects of the k-means clustering algorithm.

Examples

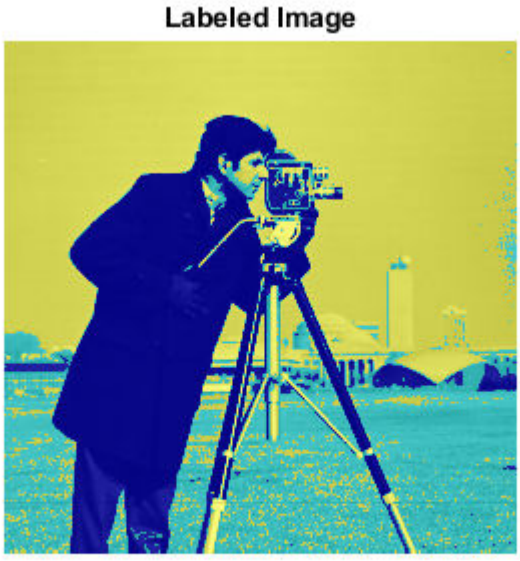
Segment Grayscale Image using k-Means Clustering

Read an image into the workspace.

```
I = imread("cameraman.tif");
imshow(I)
title("Original Image")
```

Segment the image into three regions using k-means clustering.

```
[L,Centers] = imsegkmeans(I,3);
B = labeloverlay(I,L);
imshow(B)
title("Labeled Image")
```



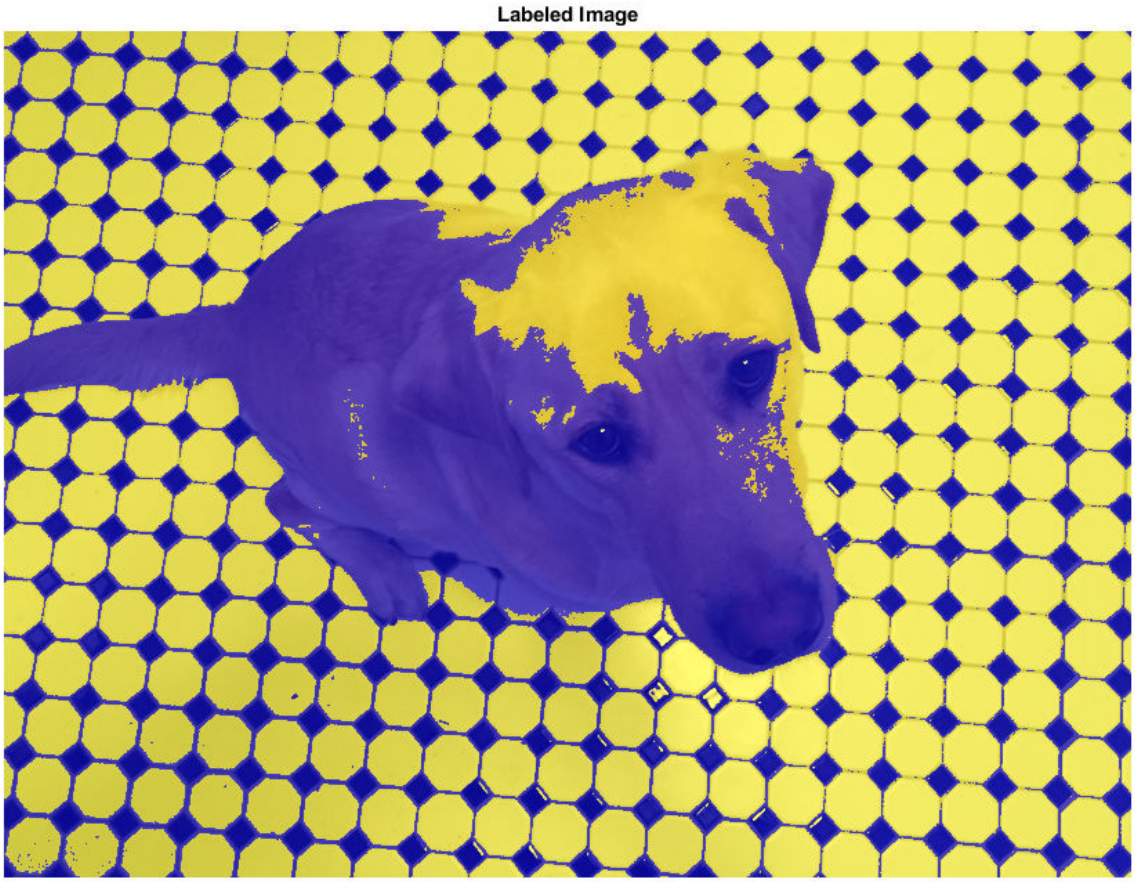
Improve k-Means Segmentation Using Texture and Spatial Information

Read an image into the workspace. Reduce the image size to make the example run more quickly.

```
RGB = imread("kobi.png");
RGB = imresize(RGB,0.5);
imshow(RGB)
```

Segment the image into two regions using k-means clustering.

```
L = imsegkmeans(RGB,2);
B = labeloverlay(RGB,L);
imshow(B)
title("Labeled Image")
```



Several pixels are mislabeled. The rest of the example shows how to improve the k-means segmentation by supplementing the information about each pixel.

Supplement the image with information about the texture in the neighborhood of each pixel. To obtain the texture information, filter a grayscale version of the image with a set of Gabor filters.

Create a set of 24 Gabor filters, covering 6 wavelengths and 4 orientations.

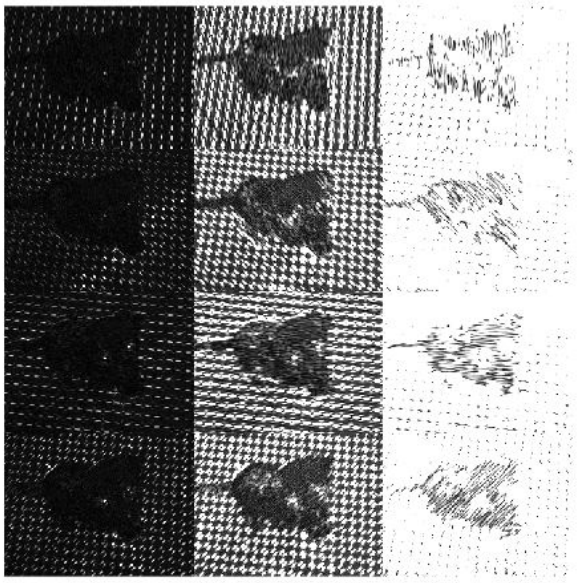
```
wavelength = 2.^(0:5) * 3;
orientation = 0:45:135;
g = gabor(wavelength,orientation);
```

Convert the image to grayscale.

```
I = im2gray(im2single(RGB));
```

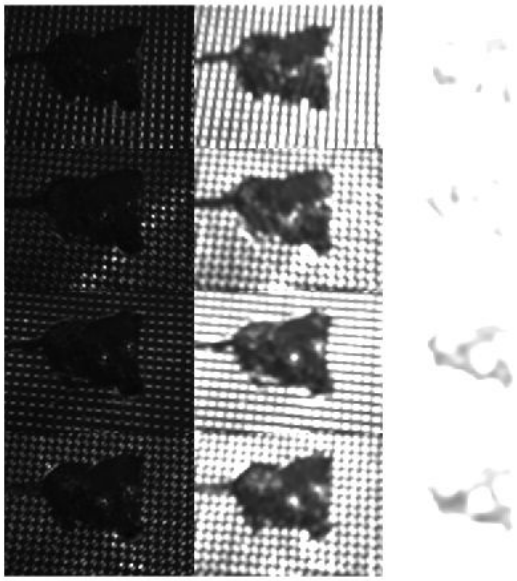
Filter the grayscale image using the Gabor filters. Display the 24 filtered images in a montage.

```
gabormag = imgaborfilt(I,g);
montage(gabormag,"Size",[4 6])
```



Smooth each filtered image to remove local variations. Display the smoothed images in a montage.

```
for i = 1:length(g)
    sigma = 0.5*g(i).Wavelength;
    gabormag(:,:,i) = imgaussfilt(gabormag(:,:,i),3*sigma);
end
montage(gabormag,"Size",[4 6])
```



Supplement the information about each pixel with spatial location information. This additional information allows the k-means clustering algorithm to prefer groupings that are close together spatially.

Get the *x* and *y* coordinates of all pixels in the input image.

```
nrows = size(RGB,1);
ncols = size(RGB,2);
[X,Y] = meshgrid(1:ncols,1:nrows);
```

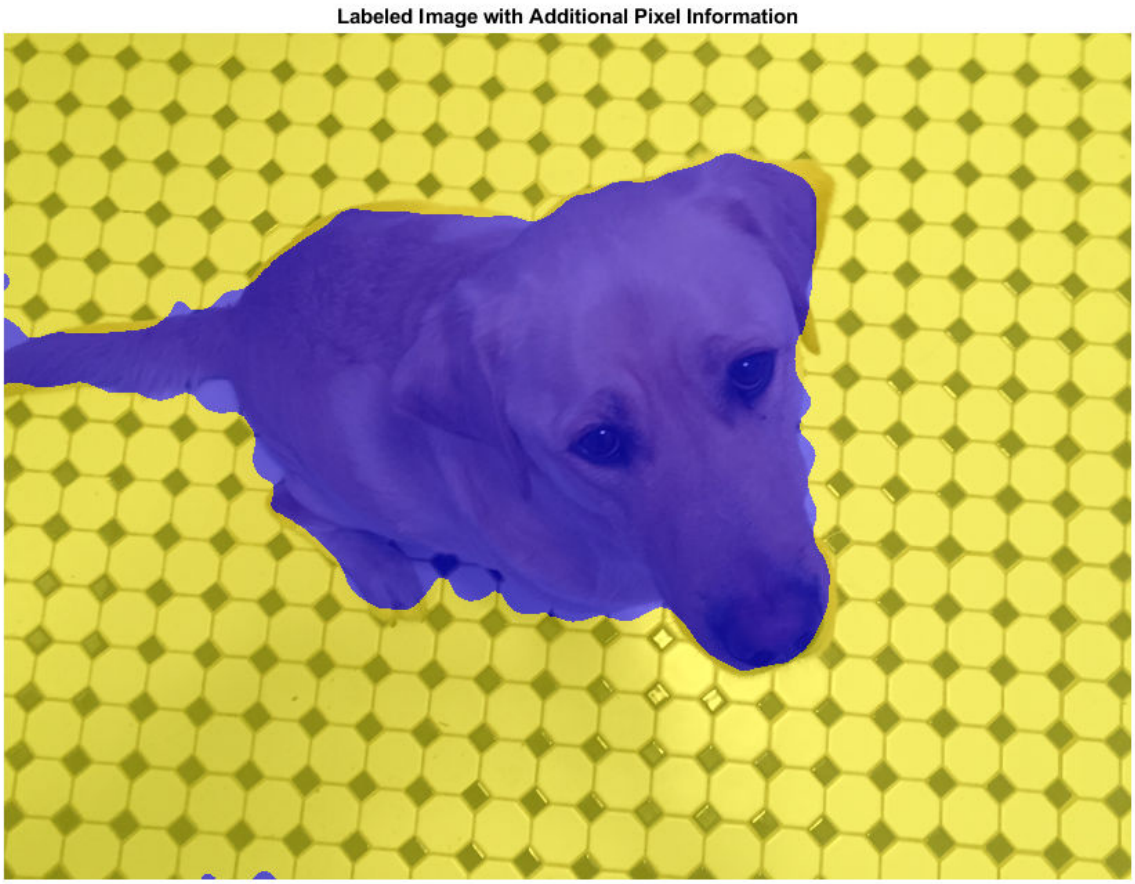
Concatenate the intensity information, neighborhood texture information, and spatial information about each pixel.

For this example, the feature set includes intensity image `I` instead of the original color image, `RGB`. The color information is omitted from the feature set because the yellow color of the dog's fur is similar to the yellow hue of the tiles. The color channels do not provide enough distinct information about the dog and the background to make a clean segmentation.

```
featureSet = cat(3,I,gabormag,X,Y);
```

Segment the image into two regions using k-means clustering with the supplemented feature set.

```
L2 = imsegkmeans(featureSet,2,"NormalizeInput",true);
C = labeloverlay(RGB,L2);
imshow(C)
title("Labeled Image with Additional Pixel Information")
```



Compress Color Image Using k-Means Segmentation

Read an image into the workspace.

```
I = imread("peppers.png");
imshow(I)
title("Original Image")
```

Segment the image into 50 regions by using k-means clustering. Return the label matrix `L` and the cluster centroid locations `C`. The cluster centroid locations are the RGB values of each of the 50 colors.

```
[L,C] = imsegkmeans(I,50);
```

Convert the label matrix into an RGB image. Specify the cluster centroid locations, `C`, as the colormap for the new image.

```
J = label2rgb(L,im2double(C));
```

Display the quantized image.

```
imshow(J)
title("Color Quantized Image")
```

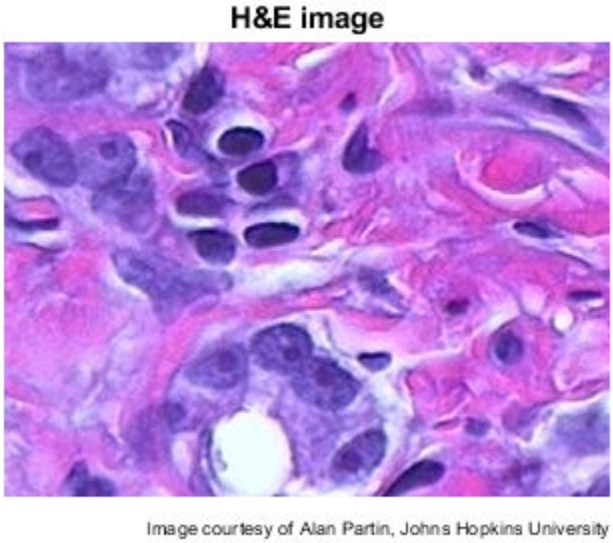
Write the original and compressed images to file. The quantized image file is approximate one quarter the size of the original image file.

```
imwrite(I,"peppersOriginal.png");
imwrite(J,"peppersQuantized.png");
```

Improve K-Means Segmentation Using Alternative Color Space

Read an image into the workspace. The image shows tissue stained with hemotoxylin and eosin (H&E). This staining method helps pathologists distinguish between tissue types that are stained blue-purple and pink.

```
he = imread("hestain.png");
imshow(he), title("H&E image");
text(size(he,2),size(he,1)+15, ...
     "Image courtesy of Alan Partin, Johns Hopkins University", ...
     "FontSize",7,"HorizontalAlignment","right");
```



Convert the image to the L*a*b* color space using the `rgb2lab` function. The L*a*b* color space separates image luminosity and color. This makes it easier to segment regions by color, independent of lightness.

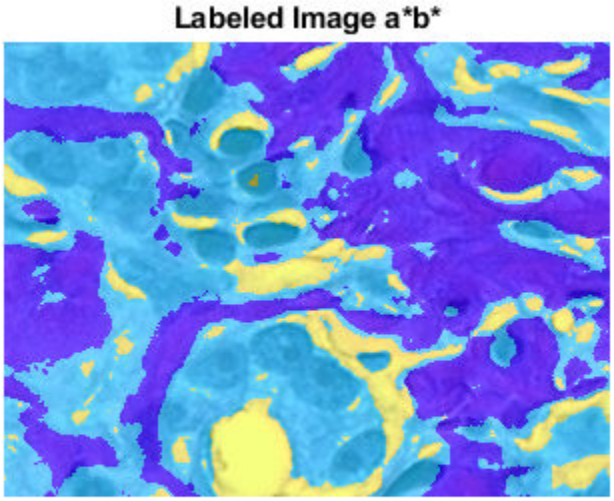
```
lab_he = rgb2lab(he);
```

To segment the image using only color information, limit the image to the a* and b* values in `lab_he`. Convert the image to data type `single` for use with `imsegkmeans`. Use the `imsegkmeans` function to segment the image into three regions.

```
ab = lab_he(:,:,2:3);
ab = im2single(ab);
numColors = 3;
L2 = imsegkmeans(ab,numColors);
```

Display the label image as an overlay on the original image. The label image separates the white, blue-purple, and pink stained tissue regions.

```
B2 = labeloverlay(he,L2);
imshow(B2)
title("Labeled Image a*b*")
```



Input Arguments

`I` — Image to segment

2-D grayscale image | 2-D color image | 2-D multispectral image

Image to segment, specified as a 2-D grayscale image, 2-D color image, or 2-D multispectral image. If the original image is of data type `double`, convert the image to data type `single` by using the `im2single` function.

Data Types: `single` | `int8` | `int16` | `uint8` | `uint16`

`k` — Number of clusters

positive integer

Number of clusters to create, specified as a positive integer.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `imsegkmeans(I,k,NumAttempts=5)` repeats the clustering process five times.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `imsegkmeans(I,k,"NumAttempts",5)` repeats the clustering process five times.

`NormalizeInput` — Normalize input data

`true` or `1` (default) | `false` or `0`

Normalize input data to zero mean and unit variance, specified as a numeric or logical `1` (`true`) or `0` (`false`). If you specify `true`, then `imsegkmeans` normalizes each channel of the input individually.

`NumAttempts` — Number of times to repeat the clustering process
3 (default) | positive integer

Number of times to repeat the clustering process using new initial cluster centroid positions, specified as a positive integer.

`MaxIterations` — Maximum number of iterations
`100` (default) | positive integer

Maximum number of iterations, specified as a positive integer.

`Threshold` — Accuracy threshold
`1e-4` (default) | positive number

Accuracy threshold, specified as a positive number. The algorithm stops when each of the cluster centers move less than the threshold value in consecutive iterations.

Output Arguments

`L` — Label matrix
matrix of positive integers

Label matrix, specified as a matrix of positive integers. Pixels with label 1 belong to the first cluster, label 2 belong to the second cluster, and so on for each of the `k` clusters. `L` has the same first two dimensions as image `I`. The data type of `L` depends on the number of clusters.

Data Type of L	Number of Clusters
`uint8`	`k <= 255`
`uint16`	`256 <= k <= 65535`
`uint32`	`65536 <= k <= 2^32-1`
`double`	`2^32 <= k`

`centers` — Cluster centroid locations
numeric matrix

Cluster centroid locations, returned as a numeric matrix of size *k*-by-*c*, where *k* is the number of clusters and *c* is the number of channels. `centers` is the same data type as the image `I`.

Tips

- The function yields reproducible results. The output does not vary across multiple runs given the same input arguments.
- The `imsegkmeans` function accepts input images in all supported color spaces. Using a different color space generates different results. If you do not receive satisfactory results for an input image, consider trying an alternative color space. For more information about color spaces in MATLAB, see “Understanding Color Spaces and Color Space Conversion”.
- To perform k-means clustering on images of data type `double`, convert the image to data type `single` by using the `im2single` function. For applications requiring input data of type `double`, see the `kmeans` function.

References

[1] Arthur, David, and Sergei Vassilvitskii. “K-Means++: The Advantages of Careful Seeding.” In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1027–35. SODA ’07. USA: Society for Industrial and Applied Mathematics, 2007.

See Also

Apps
Image Segmenter

Functions
`imsegkmeans3` | `gabor` | `imgaborfilt` | `labeloverlay` | `label2rgb` | `superpixels` | `lazysnapping` | `watershed` | `labelmatrix`

Topics
“Color-Based Segmentation Using K-Means Clustering”

Introduced in R2018b

imsegkmeans3

K-means clustering based volume segmentation

Syntax

```
L = imsegkmeans3(V,k)
[L,centers] = imsegkmeans3(V,k)
L = imsegkmeans3(V,k,Name,Value)
```

Description

`L = imsegkmeans3(V,k)` segments volume `V` into `k` clusters by performing k-means clustering and returns the segmented labeled output in `L`.

`[L,centers] = imsegkmeans3(V,k)` also returns the cluster centroid locations, `centers`.

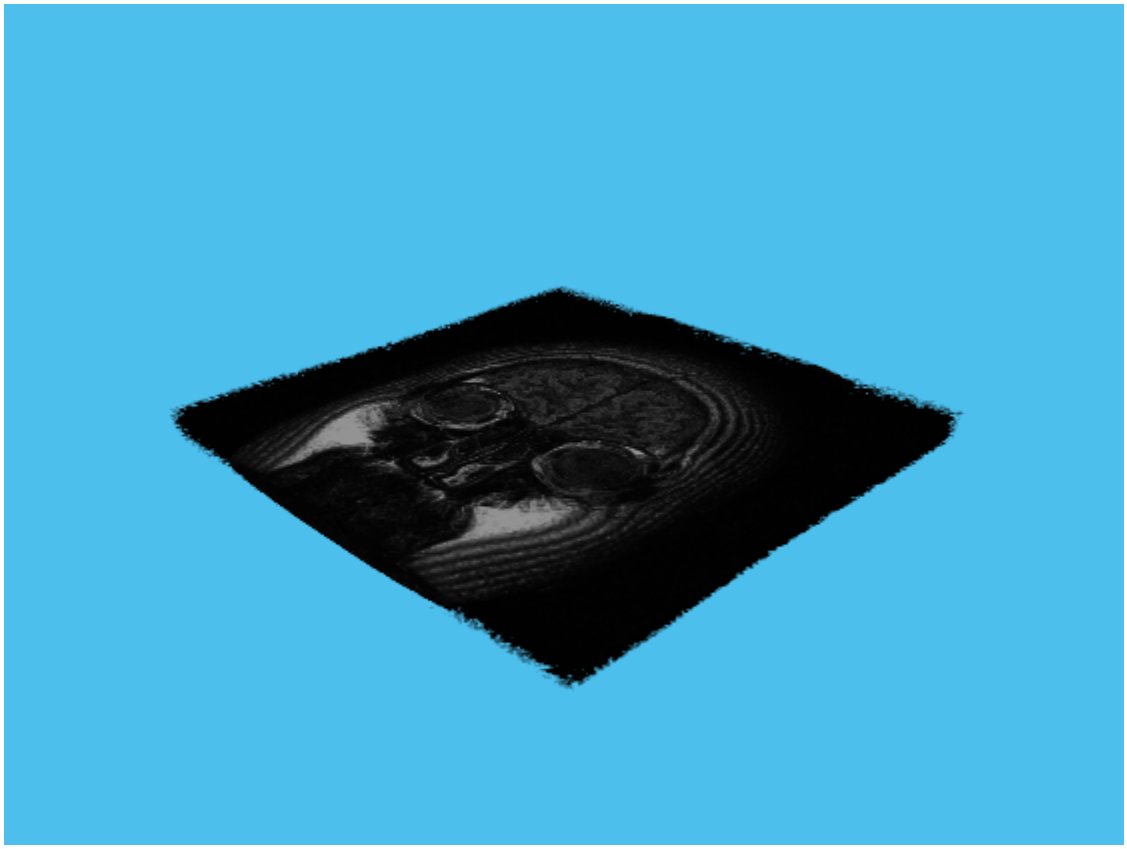
`L = imsegkmeans3(V,k,Name,Value)` uses name-value pairs to control aspects of the k-means clustering algorithm.

Examples

Segment Volume Using k-Means Clustering

Load a 3-D grayscale MRI volume and display it using `volshow`.

```
load mristack
volshow(mristack);
```

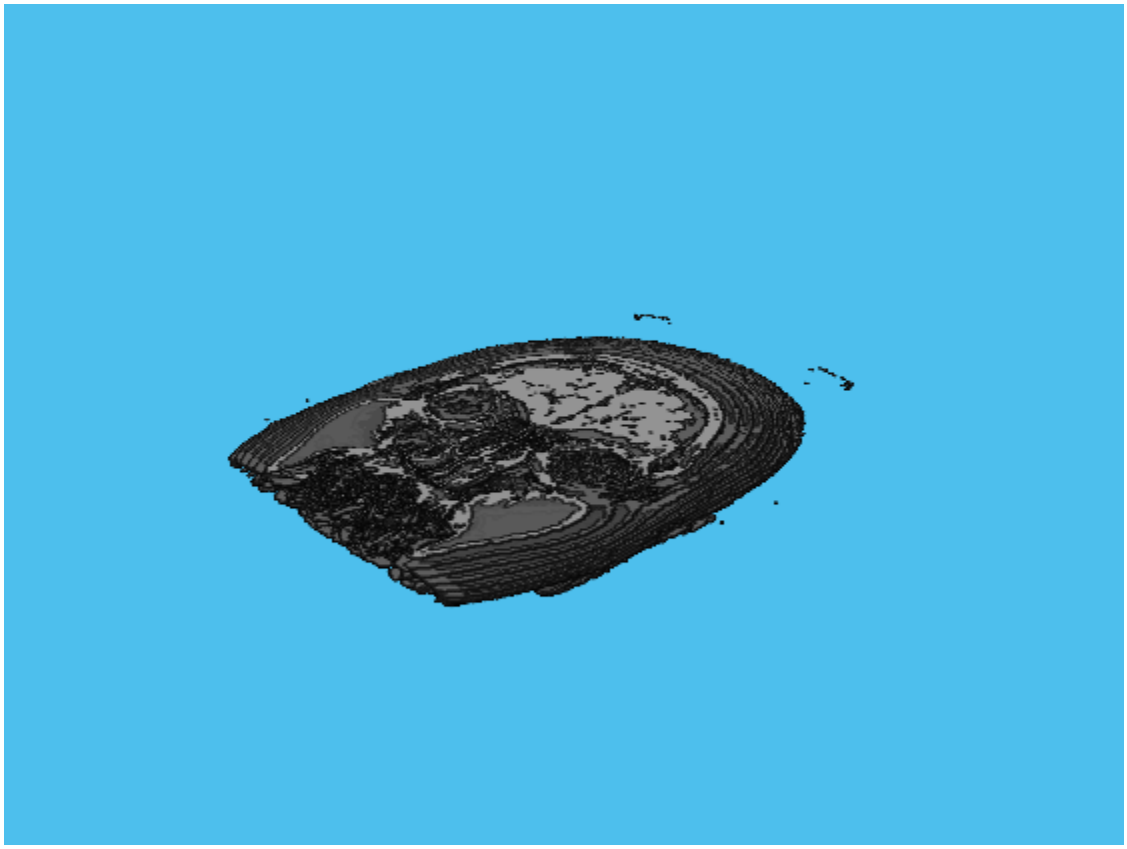


Segment the volume into three clusters.

```
L = imsegkmeans3(mristack,3);
```

Display the segmented volume using `volshow`. To explore slices of the segmented volume, use the Volume Viewer app.

```
figure
volshow(L);
```



Input Arguments

`V` — Volume to segment

3-D grayscale volume | 3-D multispectral volume

Volume to segment, specified as a 3-D grayscale volume of size *m*-by-*n*-by-*p* or a 3-D multispectral volume of size *m*-by-*n*-by-*p*-by-*c*, where *p* is the number of planes and *c* is number of channels.

Note `imsegkmeans3` treats 2-D color images like 3-D volumes of size *m*-by-*n*-by-3. If you want 2-D behavior, then use the `imsegkmeans` function.

Data Types: `single` | `int8` | `int16` | `uint8` | `uint16`

`k` — Number of clusters

positive integer

Number of clusters to create, specified as a positive integer.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'NumAttempts',5`

`NormalizeInput` — Normalize input data
`true` or `1` (default) | `false` or `0`

Normalize input data to zero mean and unit variance, specified as the comma-separated pair consisting of `'NormalizeInput'` and a numeric or logical `1` (`true`) or `0` (`false`). If you specify `true`, then `imsegkmeans3` normalizes each channel of the input individually.

`NumAttempts` — Number of times to repeat the clustering process
`3` (default) | positive integer

Number of times to repeat the clustering process using new initial cluster centroid positions, specified as the comma-separated pair consisting of `'NumAttempts'` and a positive integer.

`MaxIterations` — Maximum number of iterations
`100` (default) | positive integer

Maximum number of iterations, specified as the comma-separated pair consisting of `'MaxIterations'` and a positive integer.

`Threshold` — Accuracy threshold
`1e-4` (default) | positive number

Accuracy threshold, specified as the comma-separated pair consisting of `'Threshold'` and a positive number. The algorithm stops when each of the cluster centers move less than the threshold value in consecutive iterations.

Output Arguments

`L` — Label matrix
matrix of positive integers

Label matrix, specified as a matrix of positive integers. Pixels with label 1 belong to the first cluster, label 2 belong to the second cluster, and so on for each of the `k` clusters. `L` has the same first three dimensions as volume `V`. The class of `L` depends on number of clusters.

Class of L	Number of Clusters
`'uint8'`	`k <= 255`
`'uint16'`	`256 <= k <= 65535`
`'uint32'`	`65536 <= k <= 2^32-1`
`'double'`	`2^32 <= k`

`centers` — Cluster centroid locations
numeric matrix

Cluster centroid locations, returned as a numeric matrix of size *k*-by-*c*, where *k* is the number of clusters and *c* is the number of channels. `centers` is the same class as the image `I`.

Tips

- The function yields reproducible results. The output will not vary in multiple runs given the same input arguments.

References

[1] Arthur, David, and Sergei Vassilvitskii. “K-Means++: The Advantages of Careful Seeding.” In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1027–35. SODA ’07. USA: Society for Industrial and Applied Mathematics, 2007.

See Also

Apps
Volume Viewer

Functions
`imsegkmeans` | `superpixels3` | `watershed` | `lazysnapping`

Introduced in R2018b

imsharpen

Sharpen image using unsharp masking

Syntax

```
B = imsharpen(A)
B = imsharpen(A,Name,Value)
```

Description

`B = imsharpen(A)` sharpens the grayscale or truecolor (RGB) image `A` by using the unsharp masking on page 1-1998 method.

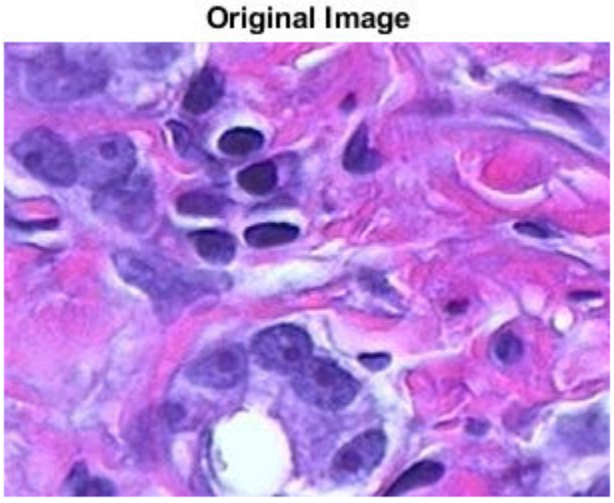
`B = imsharpen(A,Name,Value)` uses name-value arguments to control aspects of the unsharp masking.

Examples

Sharpen Image

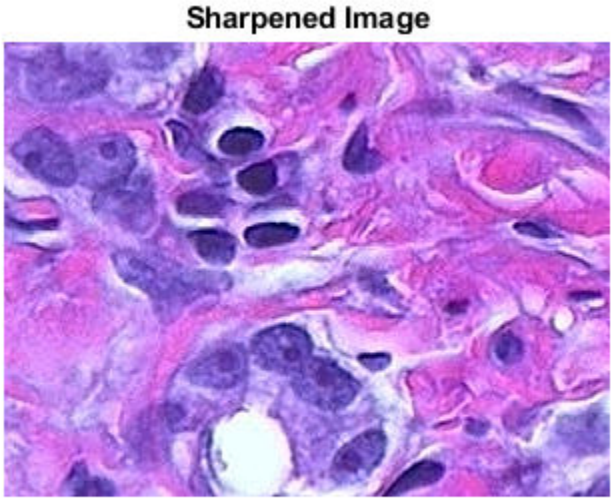
Read an image into the workspace and display it.

```
a = imread('hestain.png');
imshow(a)
title('Original Image');
```



Sharpen the image using the `imsharpen` function and display it.

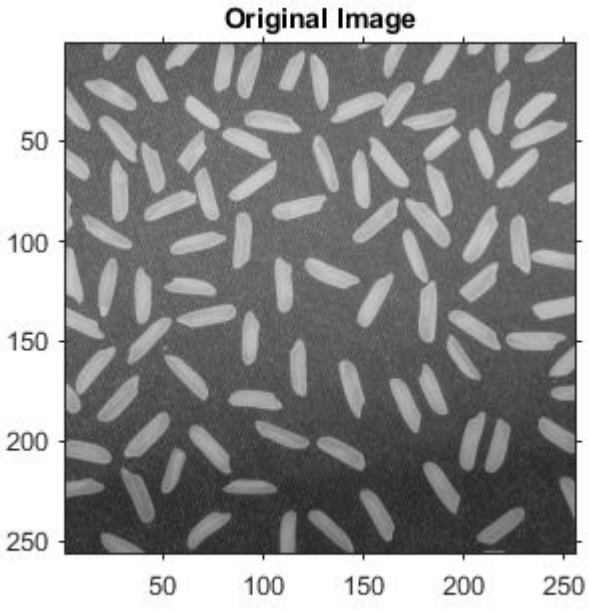
```
b = imsharpen(a);
figure, imshow(b)
title('Sharpened Image');
```



Control the Amount of Sharpening at the Edges

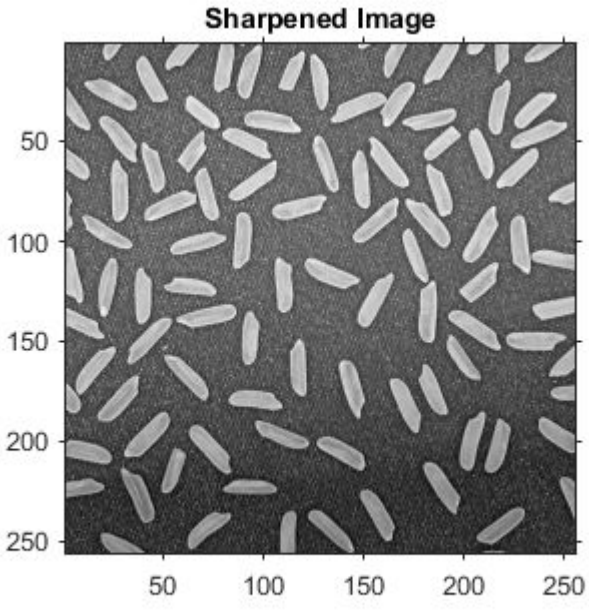
Read an image into the workspace and display it.

```
a = imread('rice.png');
imshow(a), title('Original Image');
```



Sharpen image, specifying the `radius` and `amount` parameters.

```
b = imsharpen(a,'Radius',2,'Amount',1);
figure, imshow(b)
title('Sharpened Image');
```



Input Arguments

`A` — Image to be sharpened

grayscale image | RGB image

Image to be sharpened, specified as a grayscale or RGB image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `B = imsharpen(A,Radius=1.5);` performs sharpening using a Gaussian lowpass filter with standard deviation `1.5`.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `B = imsharpen(A,"Radius",1.5);`

`Radius` — Standard deviation of Gaussian lowpass filter

`1` (default) | positive number

Standard deviation of the Gaussian lowpass filter, specified as a positive number. This argument controls the size of the region around the edge pixels that is affected by sharpening. A large value sharpens wider regions around the edges, whereas a small value sharpens narrower regions around edges.

Example: `Radius=1.5`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Amount` — Strength of sharpening effect

`0.8` (default) | number

Strength of the sharpening effect, specified as a number. A larger value leads to larger increase in the contrast of the sharpened pixels. Typical values for this parameter are within the range [0, 2], although values greater than 2 are allowed. Very large values for this argument may create undesirable effects in the output image.

Example: `Amount=1.2`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Threshold` — Minimum contrast required for a pixel to be considered an edge pixel

`0` (default) | number in the range [0, 1]

Minimum contrast required for a pixel to be considered an edge pixel, specified as a number in the range [0, 1]. Larger values (closer to 1) allow sharpening only in high-contrast regions, such as strong edges, while leaving low-contrast regions unaffected. Smaller values (closer to 0) additionally allow sharpening in relatively smoother regions of the image. This argument is useful in avoiding sharpening noise in the output image.

Example: `Threshold=0.7`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

B — Sharpened image

numeric array

Sharpened image, returned as a numeric array of the same size and data type as the input image A.

More About

Sharpening

Sharpness is the contrast between different colors. A quick transition from black to white looks sharp. A gradual transition from black to gray to white looks blurry. Sharpening images increases the contrast along the edges where different colors meet.

Unsharp masking

The unsharp masking technique comes from a publishing industry process in which an image is sharpened by subtracting a blurred (unsharp) version of the image from itself. Do not be confused by the name of this filter: an unsharp filter is an operator used to sharpen an image.

Tips

- If A is a truecolor (RGB) image, then `imsharpen` converts the image to the *L*a*b** color space, applies sharpening to the *L** channel only, and then converts the image back to the RGB color space before returning it as the output image B.

Compatibility Considerations

`imsharpen` function uses new color space conversion operations for RGB images

Behavior changed in R2022a

Starting in R2022a, the `imsharpen` uses different color space conversion operations to sharpen RGB images. In R2021b and earlier, the `imsharpen` function performed the color space conversions using the `makecform` and `applycform` functions. Starting in R2022a, the `imsharpen` function performs the color space conversions using the `rgb2lab` and `lab2rgb` functions.

The new operations yield different results for sharpened RGB images. If you need to reproduce the old behavior, then you can replace the call to `imsharpen` with a call to the `images.compatibility.imsharpen.r2021b.imsharpen` function instead. You do not need to change the input arguments.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`imsharpen` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

See Also

`fspecial` | `imadjust` | `imcontrast`

Topics

“Sharpen Region of Interest in an Image”

Introduced in R2013a

imshow

Display image

Syntax

```
imshow(I)
imshow(I,[low high])
imshow(I,[])
imshow(RGB)
imshow(BW)
imshow(X,map)
imshow(filename)
imshow(___,Name,Value)

himage = imshow( ___ )

imshow(I,RI)
imshow(X,RX,map)
```

Description

`imshow(I)` displays the grayscale image `I` in a figure. `imshow` uses the default display range for the image data type and optimizes figure, axes, and image object properties for image display.

`imshow(I,[low high])` displays the grayscale image `I`, specifying the display range as a two-element vector, `[low high]`. For more information, see the `DisplayRange` argument.

`imshow(I,[])` displays the grayscale image `I`, scaling the display based on the range of pixel values in `I`. `imshow` uses `[min(I(:)) max(I(:))]` as the display range. `imshow` displays the minimum value in `I` as black and the maximum value as white. For more information, see the `DisplayRange` argument.

`imshow(RGB)` displays the truecolor image `RGB` in a figure.

`imshow(BW)` displays the binary image `BW` in a figure. For binary images, `imshow` displays pixels with the value `0` (zero) as black and `1` as white.

`imshow(X,map)` displays the indexed image `X` with the colormap `map`.

`imshow(filename)` displays the image stored in the graphics file specified by `filename`.

`imshow(___,Name,Value)` displays an image, using name-value pairs to control aspects of the operation.

`himage = imshow( ___ )` returns the image object created by `imshow`.

`imshow(I,RI)` displays the image `I` with associated 2-D spatial referencing object `RI`.

`imshow(X,RX,map)` displays the indexed image `X` with associated 2-D spatial referencing object `RX` and colormap `map`.

Examples

Display RGB, Grayscale, Binary, or Indexed Image

Display an RGB (truecolor), grayscale, binary, or indexed image using `imshow`.

Display an RGB Image

Read a sample RGB image, `peppers.png`, into the MATLAB workspace.

```
rgbImage = imread("peppers.png");
```

Display the RGB image using `imshow`.

```
imshow(rgbImage)
```

Display a Grayscale Image

Convert the RGB image to a grayscale image by using the `rgb2gray` function.

```
grayImage = rgb2gray(rgbImage);
```

Display the grayscale image using `imshow`.

```
imshow(grayImage)
```

Display a Binary Image

Convert the grayscale image to a binary image by using thresholding.

```
meanVal = mean(grayImage,"all");
binaryImage = grayImage >= meanVal;
```

Display the binary image using `imshow`.

```
imshow(binaryImage)
```

Display an Indexed Image

Read a sample indexed image, `corn.tif`, into the MATLAB workspace.

```
[corn_indexed,map] = imread('corn.tif');
```

Display the indexed image using `imshow`.

```
imshow(corn_indexed,map)
```

Display Image from File

Display an image stored in a file.

```
imshow('peppers.png');
```

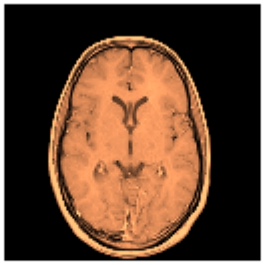
Change Colormap of Displayed Image

Load a sample grayscale volumetric image, `mri.mat`, into the variable `D` in the workspace. Remove the singleton dimension of the volume using the `squeeze` function.

```
load("mri.mat");
vol = squeeze(D);
```

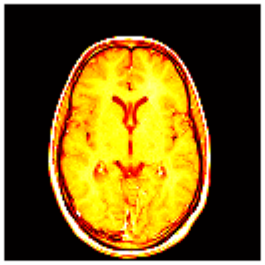
Select a slice from the middle of the volume. Display the slice using the `copper` colormap and scaling the display range to the range of pixel values.

```
sliceZ = vol(:,:,13);
imshow(sliceZ,[],Colormap=copper)
```



Change the colormap for the image using the `colormap` function.

```
colormap(hot)
```



Scale Display Range of Image

Read a truecolor (RGB) image into the workspace. The data type of the image is `uint8`.

```
RGB = imread('peppers.png');
```

Extract the green channel of the image. The green channel is the second color plane.

```
G = RGB(:,:,2);
imshow(G)
```

Create a filter that detects horizontal edges in the image.

```
filt = [-1 -1 -1;0 0 0;1 1 1];
```

Filter the green channel of the image using the `filter2` function. The result is an image of data type `double`, with a minimum value of -422 and a maximum value of 656. Pixels with a large magnitude in the filtered image indicate strong edges.

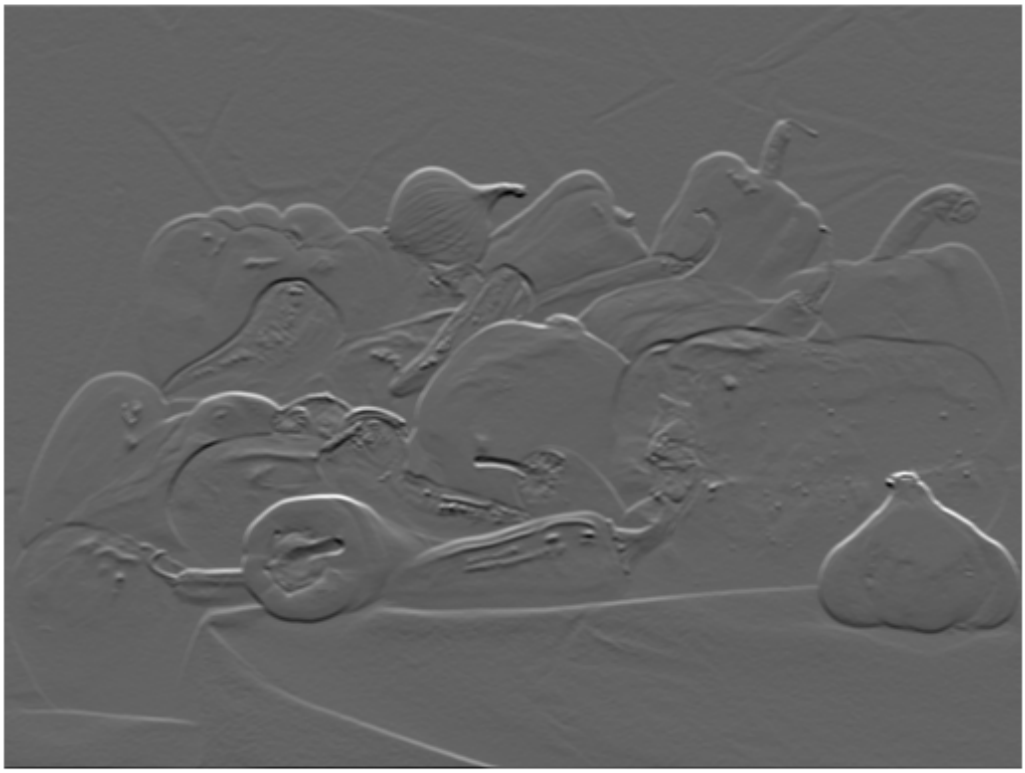
```
edgeG = filter2(filt,G);
```

Display the filtered image using `imshow` with the default display range. For images of data type `double`, the default display range is [0, 1]. The image appears black and white because the filtered pixel values exceed the range [0, 1].

```
imshow(edgeG)
```

Display the filtered image and scale the display range to the pixel values in the image. The image displays with the full range of grayscale values.

```
imshow(edgeG,[])
```



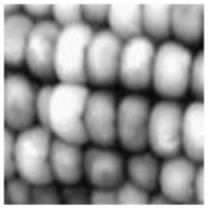
Magnify Image using Nearest Neighbor and Bilinear Interpolation

Read the grayscale image from the `corn.tif` file into the workspace. The grayscale version of the image is the second image in the file.

```
corn_gray = imread('corn.tif',2);
```

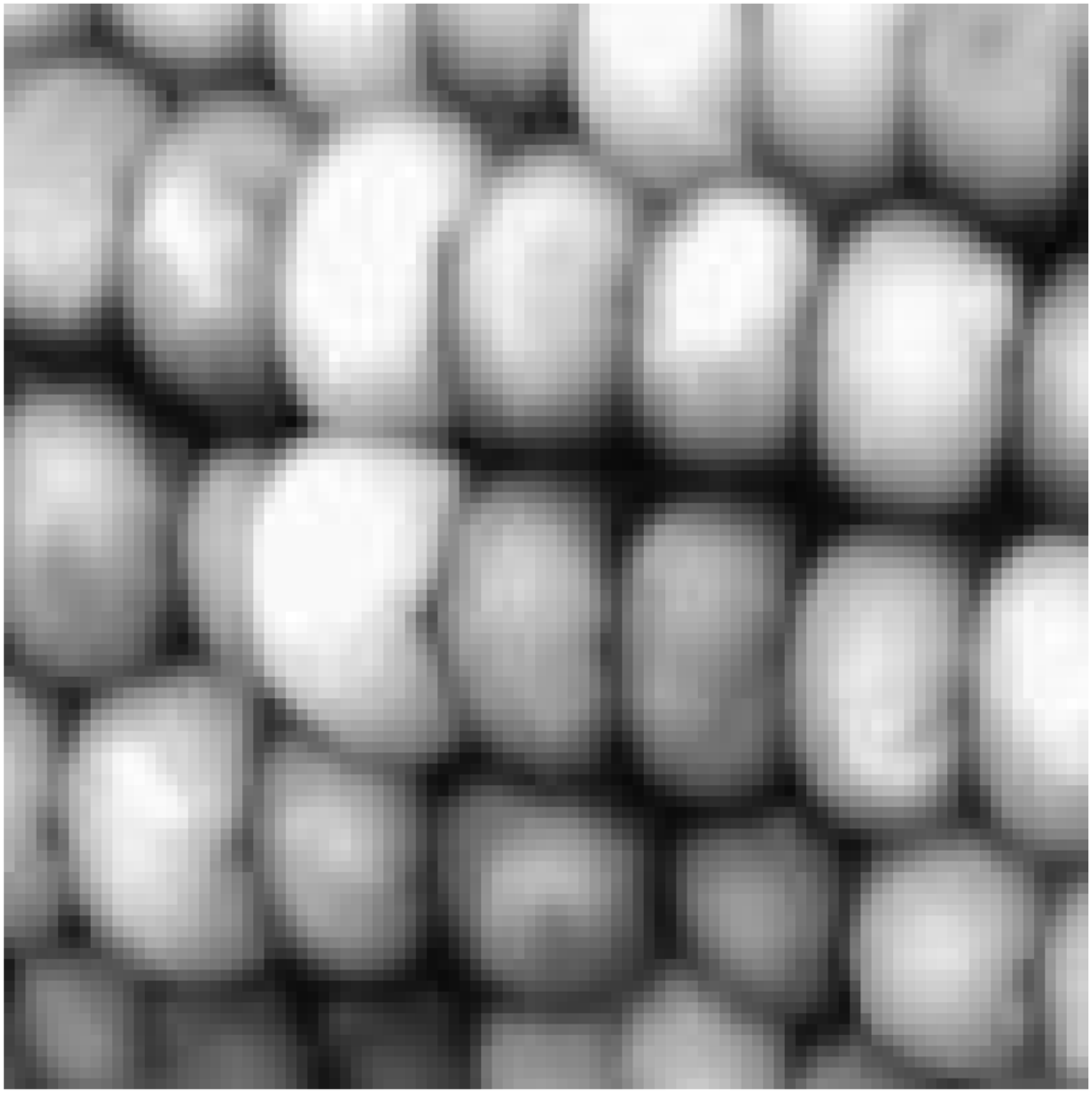
Select a small portion of the image. Display the detail image at 100% magnification using `imshow`.

```
corn_detail = corn_gray(1:100,1:100);
imshow(corn_detail)
```



Display the image at 1000% magnification by using the `'InitialMagnification'` name-value pair argument. By default, `inshow` performs nearest neighbor interpolation of pixel values. The image has blocking artifacts.

```
imshow(corn_detail,'InitialMagnification',1000)
```



Display the image at 1000% magnification, specifying the bilinear interpolation technique. The image appears smoother.

```
imshow(corn_detail,'InitialMagnification',1000,'Interpolation',"bilinear")
```

Display Image Using Associated Spatial Referencing Object

Read image into the workspace.

```
I = imread('pout.tif');
```

Display the image. Note the axes limits reflect the size of the image.

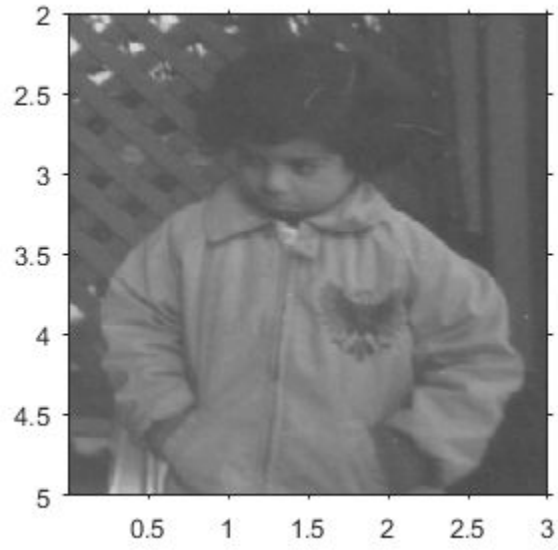
```
figure
imshow(I)
```

Create a spatial referencing object associated with the image. Use the referencing object to set the x- and y-axes limits in the world coordinate system.

```
RI = imref2d(size(I));
RI.XWorldLimits = [0 3];
RI.YWorldLimits = [2 5];
```

Display the image, specifying the spatial referencing object. Note the change to the x- and y-axes limits.

```
figure
imshow(I,RI)
```



Input Arguments

`I` — Input grayscale image

matrix

Input grayscale image, specified as a matrix. A grayscale image can be any numeric data type.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`RGB` — Input truecolor image

m-by-*n*-by-3 array

Input truecolor image, specified as an *m*-by-*n*-by-3 array.

If you specify a truecolor image of data type `single` or `double`, then values should be in the range [0, 1]. If pixel values are outside this range, then you can use the `rescale` function to scale pixel values to the range [0, 1]. The `'DisplayRange'` argument has no effect when the input image is truecolor.

Data Types: `single` | `double` | `uint8` | `uint16`

`BW` — Input binary image

matrix

Input binary image, specified as a matrix.

Data Types: `logical`

`X` — Indexed image

2-D matrix of positive integers

Indexed image, specified as a 2-D matrix of positive integers. The values in `X` are indices into the colormap specified by `map`.

Data Types: `single` | `double` | `uint8` | `logical`

`map` — Colormap
c-by-3 matrix

Colormap associated with indexed image `X`, specified as a *c*-by-3 matrix. Each row of `map` is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap. When `map` is of data type `single` or `double`, the values of the matrix are in the range [0, 1].

Data Types: `single` | `double` | `uint8`

`filename` — File name
character vector

File name, specified as a character vector. The image must be readable by the `imread` function. The `imshow` function displays the image, but does not store the image data in the MATLAB workspace. If the file contains multiple images, then `imshow` displays the first image in the file.

Example: `'peppers.png'`

Data Types: `char`

`[low high]` — Grayscale image display range
two-element vector

Grayscale image display range, specified as a two-element vector. For more information, see the `'DisplayRange'` name-value pair argument.

Example: `[50 250]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`RI` — 2-D spatial referencing object associated with input image
`imref2d` object

2-D spatial referencing object associated with input image, specified as an `imref2d` object.

`RX` — 2-D spatial referencing object associated with indexed image
`imref2d` object

2-D spatial referencing object associated with an indexed image, specified as a `imref2d` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `imshow('board.tif','Border','tight')`

`Border` — Figure window border space
`'loose'` (default) | `'tight'`

Figure window border space, specified as the comma-separated pair consisting of `'Border'` and either `'tight'` or `'loose'`. When set to `'loose'`, the figure window includes space around the

image in the figure. When set to `'tight'`, the figure window does not include any space around the image in the figure.

If the image is very small or if the figure contains other objects besides an image and its axes, `imshow` might use a border regardless of how this parameter is set.

Data Types: `char`

`Colormap` — Colormap
c-by-3 matrix

Colormap of the axes, specified as the comma-separated pair consisting of `'Colormap'` and a *c*-by-3 matrix with values in the range [0, 1]. Each row of the matrix is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap. Use this argument to view grayscale images in false color. If you specify an empty colormap (`[]`), then the `imshow` function ignores this argument.

Example: `cmap = copper; imshow('board.tif','Colormap',cmap)`

Data Types: `double`

`DisplayRange` — Grayscale image display range
two-element vector | `[]`

Display range of a grayscale image, specified as a two-element vector of the form `[low high]`. The `imshow` function displays the value `low` (and any value less than `low`) as black, and it displays the value `high` (and any value greater than `high`) as white. Values between `low` and `high` are displayed as intermediate shades of gray, using the default number of gray levels.

If you specify an empty matrix (`[]`), then `imshow` uses a display range of `[min(I(:)) max(I(:))]`. In other words, the minimum value in `I` is black, and the maximum value is white.

If you do not specify a display range, then `imshow` selects a default display range based on the image data type.

- If `I` is an integer type, then `DisplayRange` defaults to the minimum and maximum representable values for that integer class. For example, the default display range for `uint16` arrays is [0, 65535].
- If `I` is data type `single` or `double`, then the default display range is [0, 1].

Note Including the parameter name is optional, except when the image is specified by a file name. The syntax `imshow(I,[low high])` is equivalent to `imshow(I,'DisplayRange',[low high])`. If you call `imshow` with a file name, then you must specify the `'DisplayRange'` parameter.

Example: `'DisplayRange',[10 250]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`InitialMagnification` — Initial magnification of image display
`100` (default) | numeric scalar | `'fit'`

Initial magnification of the image display, specified as the comma-separated pair consisting of `'InitialMagnification'` and a numeric scalar or `'fit'`. If set to `100`, then `imshow` displays the image at 100% magnification (one screen pixel for each image pixel). If set to `'fit'`, then `imshow` scales the entire image to fit in the window.

Initially, `imshow` attempts to display the entire image at the specified magnification. If the magnification value is so large that the image is too big to display on the screen, `imshow` displays the image at the largest magnification that fits on the screen.

If the image is displayed in a figure with its `'WindowStyle'` property set to `'docked'`, then `imshow` displays the image at the largest magnification that fits in the figure.

Note: If you specify the axes position, `imshow` ignores any initial magnification you might have specified and defaults to the `'fit'` behavior.

When you use `imshow` with the `'Reduce'` parameter, the initial magnification must be `'fit'`.

In MATLAB Online™, `'InitialMagnification'` is set to `'fit'` and cannot be changed.

Example: `'InitialMagnification',80`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char`

`Interpolation` — Interpolation technique
`'nearest'` (default) | `'bilinear'`

Interpolation method, specified as `'nearest'` or `'bilinear'`. MATLAB uses interpolation to display a scaled version of the image on your screen. The value you choose does not affect the image data. Choose an interpolation method based on your image content and the effect you want to achieve:

- `'nearest'` — Nearest neighbor interpolation. The value of a pixel located at (*x*, *y*) is the value of the pixel that is closest to (*x*, *y*) in the original image. This method is best when there are a small number of pixel values that represent distinct categories, or when you want to see individual pixels in a highly zoomed-in view.
- `'bilinear'` — Bilinear interpolation. The value of a pixel located at (*x*, *y*) is a weighted average of the surrounding pixels in the original image. To minimize display artifacts, the `imshow` function performs antialiasing when you shrink the image. This method is best in almost all other situations.

`Parent` — Parent axes of image object
`Axes` object | `UIAxes` object

Parent axes of image object, specified as the comma-separated pair consisting of `'Parent'` and an `Axes` object or a `UIAxes` object. Use the `'Parent'` name-value argument to build a UI that gives you control of the `Figure` and `Axes` properties.

`Reduce` — Indicator for subsampling
`true` | `false` | `1` | `0`

Indicator for subsampling image, specified as the comma-separated pair consisting of `'Reduce'` and either `true`, `false`, `1`, or `0`. This argument is valid only when you use it with the name of a TIFF file. Use the `Reduce` argument to display overviews of very large images.

Data Types: `logical`

`XData` — X-axis limits of nondefault coordinate system
two-element vector

X-axis limits of nondefault coordinate system, specified as the comma-separated pair consisting of `'XData'` and a two-element vector. This argument establishes a nondefault spatial coordinate system

by specifying the image `XData`. The value can have more than two elements, but `imshow` uses only the first and last elements.

Example: `'XData',[100 200]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`YData` — Y-axis limits of nondefault coordinate system
two-element vector

Y-axis limits of nondefault coordinate system, specified as the comma-separated pair consisting of `'YData'` and a two-element vector. The value can have more than two elements, but `imshow` uses only the first and last elements.

Example: `'YData',[100 200]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`himage` — Image created by `imshow`
image object

Image created by the `imshow` function, specified as an image object.

Tips

- To change the colormap after you create the image, use the `colormap` command.
- You can display multiple images with different colormaps in the same figure using `imshow` with the `tiledlayout` and `nexttile` functions.
- You can create an axes on top of the axes created by `imshow` by using the `hold on` command after calling `imshow`.
- You can use the **Image Viewer** app as an integrated environment for displaying images and performing common image processing tasks.
- You can set Image Processing Toolbox preferences that modify the behavior of `imshow` by using the `iptsetpref` function.
- The `imshow` function is not supported when you start MATLAB with the `-nojvm` option.

Extended Capabilities

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- This function accepts GPU arrays, but does not run on a GPU.

For more information, see “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox).

Distributed Arrays
Partition large arrays across the combined memory of your cluster using Parallel Computing Toolbox™.

Usage notes and limitations:

- This function operates on distributed arrays, but executes in the client MATLAB.

For more information, see “Run MATLAB Functions with Distributed Arrays” (Parallel Computing Toolbox).

See Also

`imread` | `image` | `imagesc` | `imwrite` | `imfinfo` | `iptsetpref` | `colormap`

Topics

“Image Types in the Toolbox”

Introduced before R2006a

imshowpair

Compare differences between images

Syntax

```
obj = imshowpair(A,B)
obj = imshowpair(A,RA,B,RB)
obj = imshowpair( ___ ,method)
obj = imshowpair( ___ ,Name,Value)
```

Description

`obj = imshowpair(A,B)` creates a composite RGB image showing `A` and `B` overlaid in different color bands. To choose another type of visualization of the two images, use the `method` argument. If `A` and `B` are different sizes, `imshowpair` pads the smaller dimensions with zeros on the bottom and right edges so that the two images are the same size. By default, `imshowpair` scales the intensity values of `A` and `B` independently from each other. `imshowpair` returns `obj`, an image object.

`obj = imshowpair(A,RA,B,RB)` displays the differences between images `A` and `B`, using the spatial referencing information provided in `RA` and `RB`. `RA` and `RB` are spatial referencing objects.

`obj = imshowpair( ___ ,method)` uses the visualization method specified by `method`.

`obj = imshowpair( ___ ,Name,Value)` specifies additional options with one or more `Name,Value` pair arguments, using any of the previous syntaxes.

Examples

Display Two Images That Differ by Rotation Offset

Display a pair of grayscale images with two different visualization methods, `'diff'` and `'blend'`.

Load an image into the workspace. Create a copy with a rotation offset applied.

```
A = imread('cameraman.tif');
B = imrotate(A,5,'bicubic','crop');
```

Display the difference of `A` and `B`.

```
imshowpair(A,B,'diff')
```

Display a blended overlay of A and B.

```
figure
imshowpair(A,B,'blend','Scaling','joint')
```

Display Two Spatially Referenced Images with Different Brightness Ranges

Read an image. Create a copy and apply rotation and a brightness adjustment.

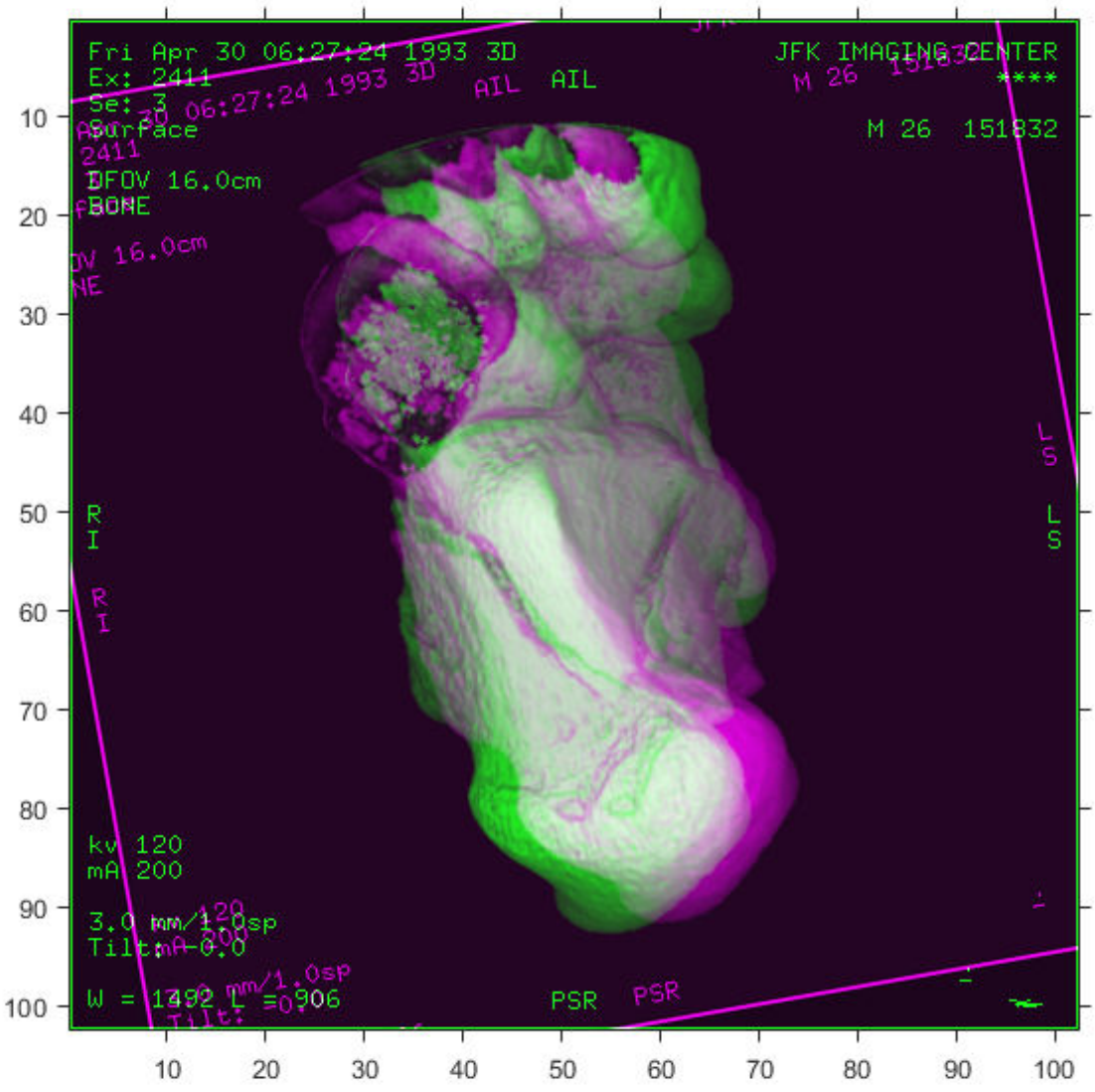
```
A = dicomread('CT-MONO2-16-ankle.dcm');
B = imrotate(A,10,'bicubic','crop');
B = B * 0.2;
```

In this example, we know that the resolution of images `A` and `B` is 0.2mm. Provide this information using two spatial referencing objects.

```
RA = imref2d(size(A),0.2,0.2);
RB = imref2d(size(B),0.2,0.2);
```

Display the images with the default method (`'falsecolor'`) and apply brightness scaling independently to each image. Specify the axes that will be the parent of the image object created by `imshowpair`.

```
figure;
hAx = axes;
imshowpair(A,RA,B,RB,'Scaling','independent','Parent',hAx);
```



Input Arguments

A — Image to be displayed

grayscale image | truecolor image | binary image

Image to be displayed, specified as a grayscale, truecolor, or binary image.

B — Image to be displayed

grayscale image | truecolor image | binary image

Image to be displayed, specified as a grayscale, truecolor, or binary image.

RA — Spatial referencing information about an input image
spatial referencing object

Spatial referencing information about an input image, specified as spatial referencing object, of class `imref2d`.

RB — Spatial referencing information about an input image
spatial referencing object

Spatial referencing information about an input image, specified as spatial referencing object, of class `imref2d`.

`method` — Visualization method to display combined images
`'falsecolor'` (default) | `'blend'` | `'diff'` | `'montage'`

Visualization method to display combined images, specified as one of the following values.

Value	Description
`'falsecolor'`	Creates a composite RGB image showing `A` and `B` overlaid in different color bands. Gray regions in the composite image show where the two images have the same intensities. Magenta and green regions show where the intensities are different. This is the default method.
`'blend'`	Overlays `A` and `B` using alpha blending.
`'checkerboard'`	Creates an image with alternating rectangular regions from `A` and `B`.
`'diff'`	Creates a difference image from `A` and `B`.
`'montage'`	Places `A` and `B` next to each other in the same image.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Scaling','joint'` scales the intensity values of `A` and `B` together as a single data set.

`ColorChannels` — Output color channel for each input image
`'green-magenta'` (default) | `[R G B]` | `'red-cyan'`

Output color channel for each input image, specified as one of the following values:

`[R G B]`	A three element vector that specifies which image to assign to the red, green, and blue channels. The R, G, and B values must be 1 (for the first input image), 2 (for the second input image), and 0 (for neither image).
`'red-cyan'`	A shortcut for the vector `[1 2 2]`, which is suitable for red/cyan stereo anaglyphs.

`'green-magenta'`	A shortcut for the vector `[2 1 2]`, which is a high contrast option, ideal for people with many kinds of color blindness.

`Interpolation` — Interpolation technique
`'nearest'` (default) | `'bilinear'`

Interpolation technique used when scaling an image, specified as the comma-separated pair consisting of `'Interpolation'` and one of the following values.

Value	Description
`'nearest'`	Nearest neighbor interpolation (default)
`'bilinear'`	Bilinear interpolation

`Parent` — Parent of image object
axes object

Parent of image object created by `imshowpair`, specified as an axes object.

`Scaling` — Intensity scaling option
`'independent'` (default) | `'joint'` | `'none'`

Intensity scaling option, specified as one of the following values.

`'independent'`	Scales the intensity values of `A` and `B` independently from each other.
`'joint'`	Scales the intensity values in the images jointly as if they were together in the same image. This option is useful when you want to visualize registrations of monomodal images, where one image contains fill values that are outside the dynamic range of the other image.
`'none'`	No additional scaling.

Data Types: `char` | `string`

Output Arguments

`obj` — Visualization of two images
image object

Visualization of two images, returned as an image object.

Tips

- Use `imfuse` to create composite visualizations that you can save to a file. Use `imshowpair` to display composite visualizations to the screen.

See Also
`imfuse` | `imregister` | `imshow` | `imtransform` | `montage`

Introduced in R2012a

imsplit

Split multichannel image into its individual channels

Syntax

```
[c1,c2,c3,...,ck] = imsplit(I)
```

Description

`[c1,c2,c3,...,ck] = imsplit(I)` returns a set of *k* images representing the individual channels in the *k*-channel image `I`.

Examples

Split RGB Image into Its Component Channels

Read an RGB image into the workspace and display the image.

```
I = imread('peppers.png');
imshow(I)
```

Split the image into its component red, green, and blue channels.

```
[r,g,b] = imsplit(I);
```

Display the three color channels as a montage. Red peppers have a signal predominantly in the red channel. Yellow and green peppers have a signal in both the red and green channels. White objects, such as the garlic in the foreground, have a strong signal in all three channels.

```
montage({r,g,b},'Size',[1 3])
```

Split Image in HSV Colorspace into Its Component Channels

Read an RGB image into the workspace and display the image.

```
rgbImage = imread('peppers.png');
imshow(rgbImage)
```

Convert the RGB image to the HSV colorspace by using the `rgb2hsv` function.

```
hsvImage = rgb2hsv(rgbImage);
```

Split the HSV image into its component hue, saturation, and value channels.

```
[h,s,v] = imsplit(hsvImage);
```

Display the three channels as a montage.

```
montage({h,s,v},'Size',[1 3])
```

Input Arguments

`I` — Input image

m-by-*n*-by-*k* array

Input image, specified as an *m*-by-*n*-by-*k* numeric array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

`c1,c2,c3,...,ck` — Output images

numeric matrix for each channel

Output images, returned as *k* individual numeric matrices, where *k* is the number of channels in the input image. The output images are the same class as the input image.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`imsplit` supports the generation of C code (requires MATLAB Coder). For more information, see "Code Generation for Image Processing".

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see "Run MATLAB Functions in Thread-Based Environment".

See Also

`cat` | `im2gray`

Topics
“Display Separated Color Channels of RGB Image”
“Convert Between RGB and HSV Color Spaces”

Introduced in R2018b

imsubtract

Subtract one image from another or subtract constant from image

Syntax

```
Z = imsubtract(X,Y)
```

Description

`Z = imsubtract(X,Y)` subtracts each element in array `Y` from the corresponding element in array `X` and returns the difference in the corresponding element of the output array `Z`.

Examples

Subtract Two uint8 Arrays

This example shows how to subtract two `uint8` arrays. Note that negative results are rounded to 0.

```
X = uint8([ 255 0 75; 44 225 100]);
Y = uint8([ 50 50 50; 50 50 50 ]);
Z = imsubtract(X,Y)
```

```
Z = 2x3 uint8 matrix

   205     0    25
     0   175    50
```

Subtract Image Background

Read a grayscale image into the workspace.

```
I = imread('rice.png');
```

Estimate the background.

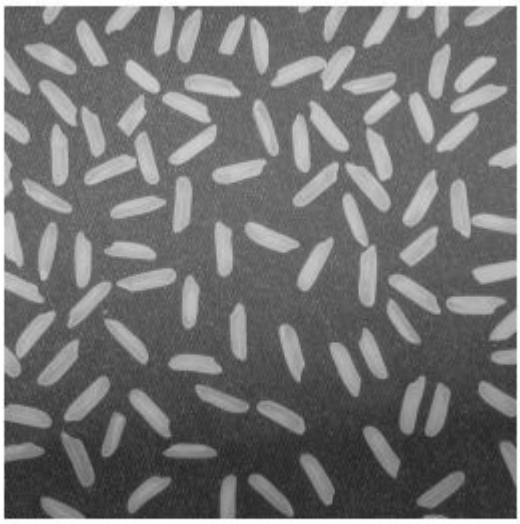
```
background = imopen(I,strel('disk',15));
```

Subtract the background from the image.

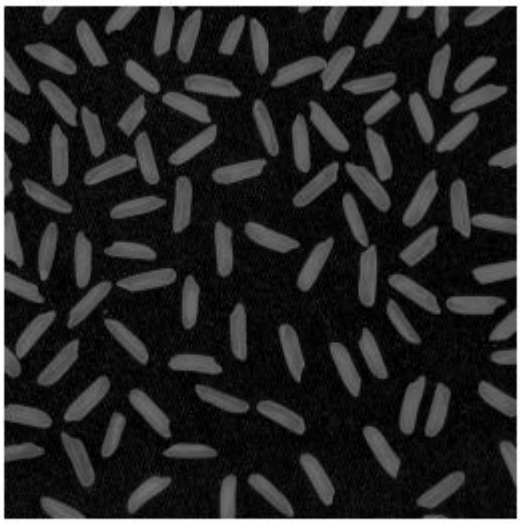
```
J = imsubtract(I,background);
```

Display the original image and the processed image.

```
imshow(I)
```



```
figure
imshow(J)
```



Subtract a Constant from an Image

Read an image into the workspace.

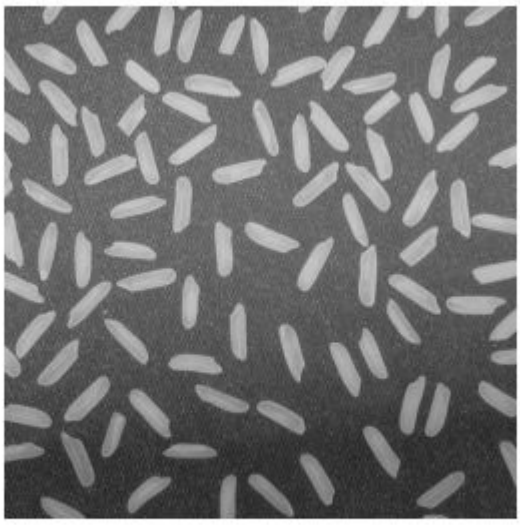
```
I = imread('rice.png');
```

Subtract a constant value from the image.

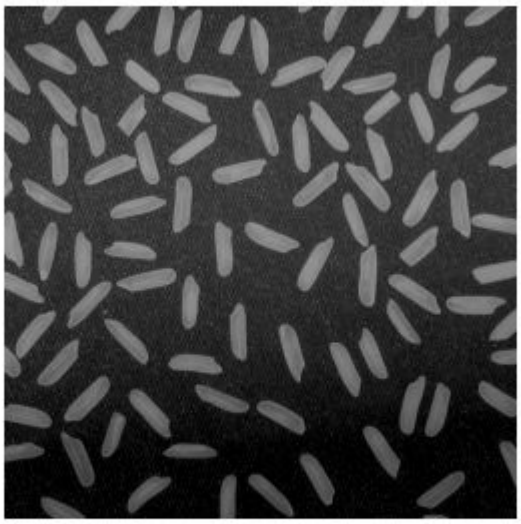
```
J = imsubtract(I,50);
```

Display the original image and the result.

```
imshow(I)
```



```
figure
imshow(J)
```



Input Arguments

X — First array

numeric array | logical array

First array (minuend), specified as a numeric array or logical array of any dimension.

Y — Second array

numeric scalar | numeric array | logical array

Second array (subtrahend) to be subtracted from `X`, specified as a numeric array or logical array of the same size and class as `X`, or a numeric scalar of type `double`.

Output Arguments

Z — Difference

numeric array

Difference, returned as a numeric array of the same size as `X`. `Z` is the same class as `X` unless `X` is logical, in which case `Z` is data type `double`. If `X` is an integer array, then elements of the output that exceed the range of the integer type are truncated, and fractional values are rounded.

See Also

`imabsdiff` | `imadd` | `imcomplement` | `imdivide` | `imlincomb` | `immultiply`

Introduced before R2006a

imtophat

Top-hat filtering

Syntax

```
J = imtophat(I,SE)
J = imtophat(I,nhood)
```

Description

`J = imtophat(I,SE)` performs morphological top-hat filtering on the grayscale or binary image `I` using the structuring element `SE`. Top-hat filtering computes the morphological opening of the image (using `imopen`) and then subtracts the result from the original image.

`J = imtophat(I,nhood)` top-hat filters the image `I`, where `nhood` is a matrix of `0`s and `1`s that specifies the structuring element neighborhood.

This syntax is equivalent to `imtophat(I,strel(nhood))`.

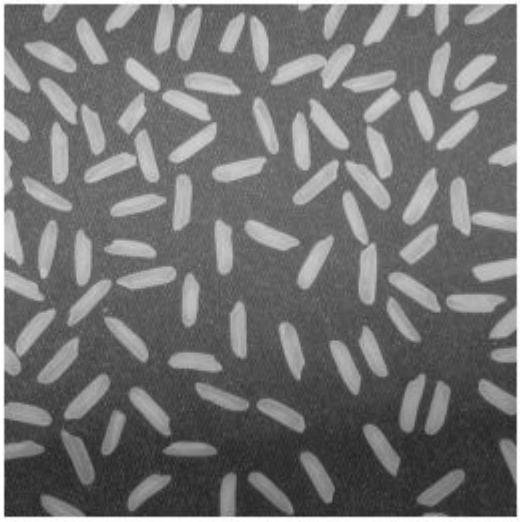
Examples

Use Top-hat Filtering to Correct Uneven Illumination

This example shows how to use top-hat filtering with a disk-shaped structuring element to remove uneven background illumination from an image with a dark background.

Read an image and display it.

```
original = imread('rice.png');
imshow(original)
```

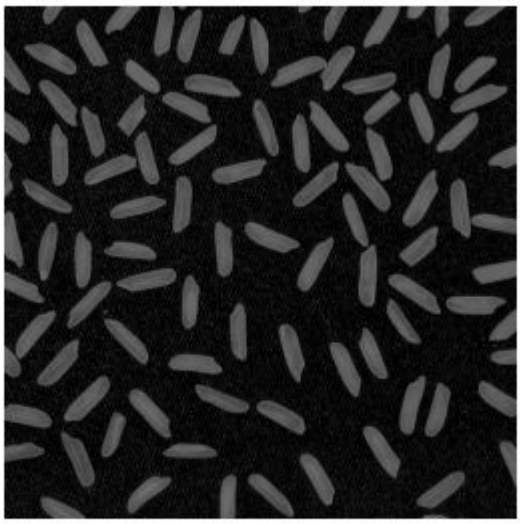


Create the structuring element.

```
se = strel('disk',12);
```

Perform the top-hat filtering and display the image.

```
tophatFiltered = imtophat(original,se);
figure
imshow(tophatFiltered)
```



Use `imadjust` to improve the visibility of the result.

```
contrastAdjusted = imadjust(tophatFiltered);
figure
imshow(contrastAdjusted)
```

Input Arguments

`I` — Input image

grayscale image | binary image

Input image, specified as a grayscale image or binary image of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`SE` — Structuring element

`strel` object | `offsetstrel` object

Structuring element, specified as a single `strel` object or `offsetstrel` object. If the image `I` is data type `logical`, the structuring element must be flat.

`nhood` — Structuring element neighborhood

matrix of `0`s and `1`s

Structuring element neighborhood, specified as a matrix of `0`s and `1`s.

Example: `[0 1 0; 1 1 1; 0 1 0]`

Output Arguments

`J` — Top-hat filtered image

grayscale image | binary image

Top-hat filtered image, returned as a grayscale image or binary image. `J` has the same data type as input image `I`.

Tips

- If the dimensionality of the image `I` is greater than the dimensionality of the structuring element, then the `imtophat` function applies the same morphological opening to all planes along the higher dimensions.

  You can use this behavior to perform top-hat filtering on RGB images. Specify a 2-D structuring element for RGB images to operate on each color channel separately.
- When you specify a structuring element neighborhood, `imtophat` determines the center element of `nhood` by `floor((size(nhood)+1)/2)`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imtophat` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imtophat` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

- The input image `I` must be 2-D or 3-D.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The image input `I` must be 2-D or 3-D.
- The structuring element `SE` must be a compile-time constant.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- `gpuArray` input must be of type `uint8` or `logical`.
- The structuring element `SE` must be flat and 2-D.

For more information, see “Image Processing on a GPU”.

See Also

Functions
`imclose` | `imdilate` | `imerode` | `imopen` | `imbothat`

Objects
`strel` | `offsetstrel`

Introduced before R2006a

imtransform

(Not recommended) Apply 2-D spatial transformation to image

Note `imtransform` is not recommended. Use `imwarp` instead for 2-D and 3-D transformations. Use `tformarray` for higher dimensional transformations.

Syntax

```
B = imtransform(A,tform)
B = imtransform(A,tform,interp)
B = imtransform( ___ ,Name,Value)
[B,xdata,ydata] = imtransform( ___ )
```

Description

`B = imtransform(A,tform)` transforms image `A` according to the 2-D spatial transformation defined by `tform`, and returns the transformed image, `B`.

If `A` is a color image, then `imtransform` applies the same 2-D transformation to each color channel. Likewise, if `A` is a volume or image sequence with three or more dimensions, then `imtransform` applies the same 2-D transformation to all 2-D planes along the higher dimensions.

`B = imtransform(A,tform,interp)` specifies the form of interpolation to use.

`B = imtransform( ___ ,Name,Value)` uses name-value pairs to control various aspects of the spatial transformation.

`[B,xdata,ydata] = imtransform( ___ )` also returns the extent of the output image `B` in the output X-Y space. By default, `imtransform` calculates `xdata` and `ydata` automatically so that `B` contains the entire transformed image `A`. However, you can override this automatic calculation by specifying values for the `XData` and `YData` name-value pair input arguments.

Examples

Simple Transformation

Apply a horizontal shear to a grayscale image.

```
I = imread('cameraman.tif');
tform = maketform('affine',[1 0 0; .5 1 0; 0 0 1]);
J = imtransform(I,tform);
imshow(J)
```

Projective Transformation

Map a square to a quadrilateral with a projective transformation. Set up an input coordinate system so that the input image fills the unit square with vertices (0 0), (1 0), (1 1), (0 1).

```
I = imread('cameraman.tif');
udata = [0 1];  vdata = [0 1];
```

Transform to a quadrilateral with vertices (-4 2), (-8 3), (-3 -5), (6 3).

```
tform = maketform('projective',[ 0 0;  1  0;  1  1; 0 1],...
                               [-4 2; -8 -3; -3 -5; 6 3]);
```

Fill with gray and use bicubic interpolation. Make the output size the same as the input size.

```
[B,xdata,ydata] = imtransform(I,tform,'bicubic', ...
                              'udata',udata,...
                              'vdata',vdata,...
                              'size',size(I),...
                              'fill',128);
subplot(1,2,1); imshow(I,'XData',udata,'YData',vdata)
subplot(1,2,2); imshow(B,'XData',xdata,'YData',ydata)
```

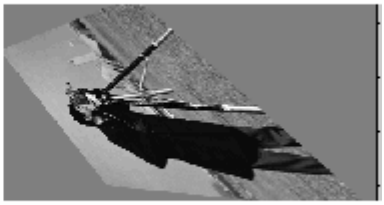


Image Registration

Read an aerial photo into the MATLAB workspace and view it.

```
unregistered = imread('westconcordaerial.png');
figure
imshow(unregistered)
```

Read an orthophoto into the MATLAB workspace and view it.

```
figure
imshow('westconcordorthophoto.png')
```

Load control points that were previously picked.

```
load westconcordpoints
```

Create a transformation structure for a projective transformation using the points.

```
t_concord = cp2tform(movingPoints,fixedPoints,'projective');
```

Get the width and height of the orthophoto, perform the transformation, and view the result.

```
info = imfinfo('westconcordorthophoto.png');

registered = imtransform(unregistered,t_concord,...
    'XData',[1 info.Width],'YData',[1 info.Height]);
figure
imshow(registered)
```

Input Arguments

A — Image to be transformed

numeric array | logical array

Image to be transformed, specified as a numeric or logical array of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

tform — Transformation structure

`TFORM` structure

Transformation structure, specified as a `TFORM` structure, such as returned by `maketform`. The first dimension of the transformation is the horizontal or *x*-coordinate, and the second dimension is the vertical or *y*-coordinate. This convention is the reverse of the array subscripting convention in MATLAB.

interp — Interpolation method

`'bilinear'` (default) | `'nearest'` | `'bicubic'` | `resampler` structure

Interpolation method, specified as one of these values.

Interpolation Method	Description
`'bilinear'`	Linear interpolation
`'nearest'`	Nearest-neighbor interpolation—the output pixel is assigned the value of the pixel that the point falls within. No other pixels are considered.
`'bicubic'`	Cubic interpolation
`resampler` structure	`resampler` structure returned by `makeresampler`. This option allows more control over how `imtransform` performs resampling.

Data Types: `char`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'FillValues',128`

`UData, VData` — Spatial extent in U-V input space

2-element numeric vector

Spatial extent of input image `A` in the U-V input space, specified as 2-element numeric vectors. The values of `UData` and `VData` represent coordinates in the world coordinate system. The two elements of `UData` give the *u*-coordinates (horizontal) of the first and last columns of `A`, respectively. The two elements of `VData` give the *v*-coordinates (vertical) of the first and last rows of `A`.

By default, the spatial extent of `A` in the U-V space is the same as the image extent in intrinsic coordinates. In other words, the default value of `UData` is `[1 size(A,2)]` and the default value of `VData` is `[1 size(A,1)]`.

`XData, YData` — Spatial extent in X-Y output space

2-element numeric vector

Spatial extent of transformed image `B` in the X-Y input space, specified as 2-element numeric vectors. The values of `XData` and `YData` represent coordinates in the world coordinate system. The two elements of `XData` give the *x*-coordinates (horizontal) of the first and last columns of `B`, respectively. The two elements of `YData` give the *y*-coordinates (vertical) of the first and last rows of `B`.

If you do not specify `XData` and `YData`, then `imtransform` estimates values that contain the entire transformed output image. To determine these values, `imtransform` uses the `findbounds` function.

`XYScale` — Size of pixels in X-Y output space

numeric scalar | 2-element numeric vector

Size of pixels in X-Y output space, specified as a numeric scalar or 2-element numeric vector. If `XYScale` is a scalar, then output pixels are square and `XYScale` specifies the side length. Otherwise, the two elements of `XYScale` specify the width and height of each output pixel in X-Y space, respectively.

The default value of `XYScale` depends on whether you specify `Size`:

- If you specify `Size`, then `imtransform` calculates `XYScale` from `Size`, `XData`, and `YData`.
- If you do not specify `Size`, then `imtransform` uses the scale of the input pixels for `XYScale`, except in cases where an excessively large output image would result.

Note In cases where preserving the scale of the input image would result in an excessively large output image, the `imtransform` function automatically increases the value of `XYScale`. To ensure that the output pixel scale matches the input pixel scale, specify the `XYScale` parameter. For example, call `imtransform` as shown in the following syntax:

```
B = imtransform(A,T,'XYScale',1)
```

`Size` — Size of transformed image

2-element vector of positive integers

Size of transformed image `B`, specified as a 2-element vector of positive integers. The two elements of `Size` specify the number of rows and columns of the output image `B`, respectively. For higher dimensions, `imtransform` takes the size of `B` directly from the size of input image `A`. Thus, `size(B,k)` equals `size(A,k)` for `k > 2`.

If you do not specify `Size`, then `imtransform` derives this value from `XData`, `YData`, and `XYScale`.

`FillValues` — Fill value

numeric scalar | numeric array

Fill value used for output pixels outside the input image boundaries, specified as the comma-separated pair consisting of `'FillValues'` and a numeric scalar or numeric array. Fill values are used for output pixels when the corresponding inverse transformed location in the input image is completely outside the input image boundaries.

- If the input image `A` is 2-D, then `FillValues` must be a scalar.
- If `A` is 3-D or N-D, then `FillValues` can be an array whose size satisfies the following constraint: `size(FillValues,k)` must equal either `size(A,k+2)` or `1`.

  For example, if `A` is a `uint8` RGB image that is 200-by-200-by-3, then possibilities for `'FillValues'` include the following values.

Value	Fill
`0`	Fill with black
`[0;0;0]`	Fill with black
`255`	Fill with white
`[255;255;255]`	Fill with white
`[0;0;255]`	Fill with blue
`[255;255;0]`	Fill with yellow

  For a second example, if `A` is 4-D with size 200-by-200-by-3-by-10, then you can specify `'FillValues'` as a scalar, 1-by-10 vector, 3-by-1 vector, or 3-by-10 matrix.

Output Arguments

`B` — Transformed image

numeric array | logical array

Transformed image, returned as a numeric or logical array of the same dimensionality as the input image `A`.

`xdata` — Horizontal extent in X-Y output space

2-element numeric vector

Horizontal extent of the transformed image `B` in X-Y output space, returned as a 2-element numeric vector. The two elements of `xdata` give the *x*-coordinates (horizontal) of the first and last columns of `B` in the world coordinate system, respectively.

Note The first element of `xdata` always equals the first element of the `XData` argument, if specified. However, sometimes the second element of `xdata` does not exactly equal the second element of `XData`. The values differ either because of the need for an integer number of rows and columns, or because you specified values for `XData`, `YData`, `XYScale`, and `Size` that are not entirely consistent.

`ydata` — Vertical extent in X-Y output space

2-element numeric vector

Vertical extent of the transformed image `B` in X-Y output space, returned as a 2-element numeric vector. The two elements of `ydata` give the *y*-coordinates (vertical) of the first and last rows of `B` in the world coordinate system, respectively.

Note The first element of `ydata` always equals the first element of the `YData` argument, if specified. However, sometimes the second element of `ydata` does not exactly equal the second element of `YData`. The values differ either because of the need for an integer number of rows and columns, or because you specified values for `XData`, `YData`, `XYScale`, and `Size` that are not entirely consistent.

Tips

- **Image Registration.** The `imtransform` function automatically shifts the origin of your output image to make as much of the transformed image visible as possible. If you use `imtransform` to do image registration, the syntax `B = imtransform(A,tform)` can produce unexpected results. To control the spatial location of the output image, set `XData` and `YData` explicitly.
- **Pure Translation.** Calling the `imtransform` function with a purely translational transformation results in an output image that is exactly like the input image unless you specify `XData` and `YData` values in your call to `imtransform`. For example, if you want the output to be the same size as the input revealing the translation relative to the input image, call `imtransform` as shown in the following syntax:

  ```
  B = imtransform(A,T,'XData',[1 size(A,2)],...
     'YData',[1 size(A,1)])
  ```

  For more information about this topic, see “Perform Simple 2-D Translation Transformation”.
- **Transformation Speed.** If you do not specify the output-space location for `B` using `XData` and `YData`, then `imtransform` estimates the location automatically using the function `findbounds`. You can use `findbounds` as a quick forward-mapping option for some commonly used transformations, such as affine or projective. For transformations that do not have a forward mapping, such as polynomial transformations computed by `fitgeotrans`, `findbounds` can take much longer. If you can specify `XData` and `YData` directly for such transformations, then `imtransform` may run noticeably faster.
- **Clipping.** The automatic estimate of `XData` and `YData` using `findbounds` sometimes clips the output image. To avoid clipping, set `XData` and `YData` directly.

See Also

`checkerboard` | `imresize` | `imrotate` | `maketform` | `makeresampler` | `tformarray`

Topics

“Perform Simple 2-D Translation Transformation”
“Padding and Shearing an Image Simultaneously”

“Exploring Slices from a 3-Dimensional MRI Data Set”

Introduced before R2006a

imtranslate

Translate image

Syntax

```
B = imtranslate(A,translation)
[B,RB] = imtranslate(A,RA,translation)
___ = imtranslate( ___ ,method)
___ = imtranslate( ___ ,Name,Value)
```

Description

`B = imtranslate(A,translation)` translates image `A` by the 2-D or 3-D translation vector specified in `translation`.

If `A` has more than two dimensions and `translation` is a 2-element vector, then `imtranslate` applies the 2-D translation to each plane of `A`.

`[B,RB] = imtranslate(A,RA,translation)` translates the spatially referenced image `A` with its associated spatial referencing object `RA`. The translation vector, `translation`, is in the world coordinate system. The function returns the translated spatially referenced image `B`, with its associated spatial referencing object, `RB`.

`___ = imtranslate( ___ ,method)` translates image `A`, using the interpolation method specified by `method`.

`___ = imtranslate( ___ ,Name,Value)` translates the input image using name-value pairs to control various aspects of the translation.

Examples

Translate 2-D Image

Read image into the workspace.

```
I = imread('pout.tif');
```

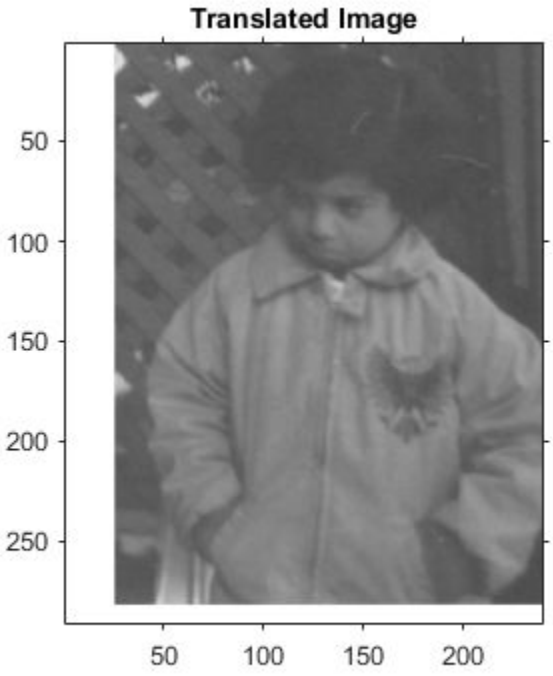
Translate the image.

```
J = imtranslate(I,[25.3, -10.1],'FillValues',255);
```

Display the original image and the translated image.

```
figure
imshow(I);
title('Original Image');
set(gca,'Visible','on');
```


```
figure
imshow(J);
title('Translated Image');
set(gca,'Visible','on');
```



Translate 2-D Image and View Entire Translated Image

Read image into the workspace.

```
I = imread('pout.tif');
```

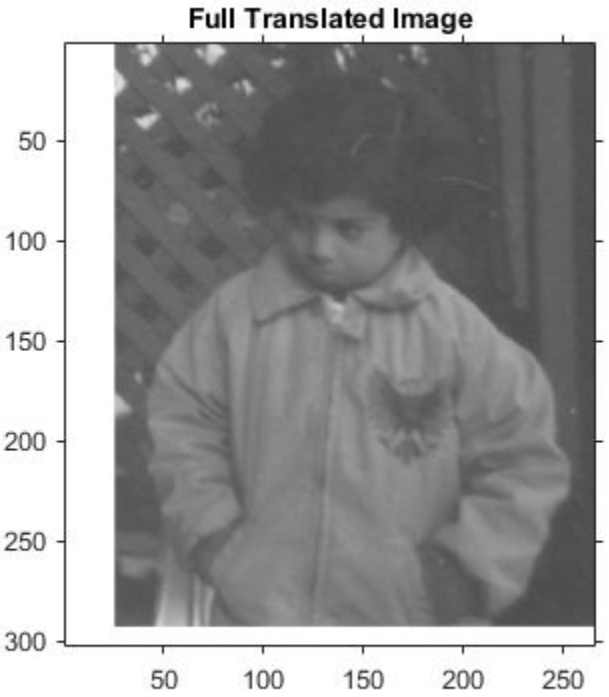
Translate the image. Use the `OutputView` parameter to specify that you want the entire translated image to be visible.

```
J = imtranslate(I,[25.3, -10.1],'FillValues',255,'OutputView','full');
```

Display the original image and the translated image.

```
figure
imshow(I);
title('Original Image');
set(gca,'Visible','on');
```

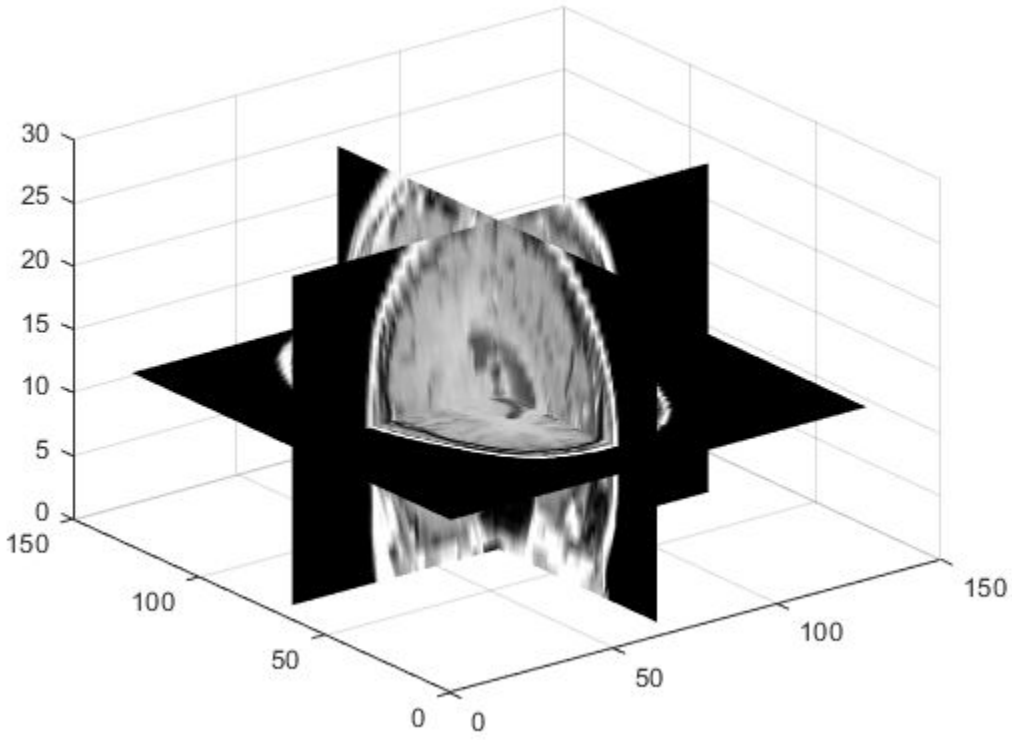

```
figure
imshow(J);
title('Full Translated Image');
set(gca,'Visible','on');
```



Translate 3-D MRI Dataset

Load MRI data into the workspace and display it.

```
s = load('mri');
mriVolume = squeeze(s.D);
sizeIn = size(mriVolume);
hFigOriginal = figure;
hAxOriginal  = axes;
slice(double(mriVolume),sizeIn(2)/2,sizeIn(1)/2,sizeIn(3)/2);
grid on, shading interp, colormap gray
```

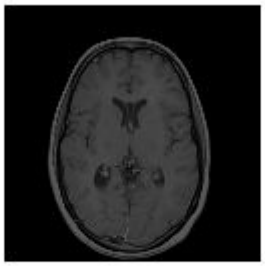


Apply a translation in the X,Y direction.

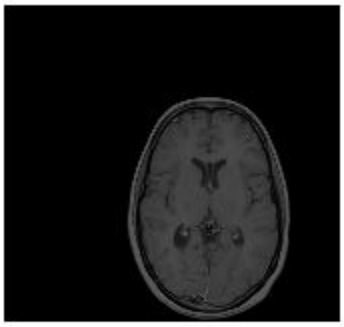
```
mriVolumeTranslated = imtranslate(mriVolume,[40,30,0],'OutputView','full');
```

Visualize the translation by viewing an axial slice plane taken through center of the volume. Note the shift in the X and Y directions.

```
sliceIndex = round(sizeIn(3)/2);
axialSliceOriginal   = mriVolume(:,:,sliceIndex);
axialSliceTranslated = mriVolumeTranslated(:,:,sliceIndex);

imshow(axialSliceOriginal);
```



```
imshow(axialSliceTranslated);
```



Input Arguments

A — Image to be translated

numeric array | logical array | categorical array

Image to be translated, specified as a numeric array, logical array, or categorical array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical` | `categorical`

RA — Spatial referencing information associated with the input image A

`imref2d` or `imref3d` spatial referencing object

Spatial referencing information associated with the input image `A`, specified as an `imref2d` or `imref3d` spatial referencing object.

translation — Translation vector

2-element numeric vector | 3-element numeric vector

Translation vector, specified as a 2-element numeric vector `[Tx Ty]` or a 3-element numeric vector `[Tx Ty Tz]`. Values can be fractional.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`method` — Interpolation method

`'linear'` (default) | `'nearest'` | `'cubic'`

Interpolation method, specified by one of the following values.

Value	Description
`'nearest'`	Nearest-neighbor interpolation. The output pixel is assigned the value of the pixel that the point falls within. No other pixels are considered. Nearest-neighbor interpolation is the only method supported for categorical images and it is the default method for images of this type.
`'bilinear'`	Linear interpolation. Linear interpolation is the default method for numeric and logical images.
`'bicubic'`	Cubic interpolation. **Note** Cubic interpolation can produce pixel values outside the original range.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `mriVolumeTranslated = imtranslate(mriVolume, [40,30,0],'OutputView','full');`

`OutputView` — Output world limits

`'same'` (default) | `'full'`

Output world limits, specified as the comma-separated pair consisting of `'OutputView'` and one of the following values.

Value	Description
`'same'`	Output world limits are the same as the input image.
`'full'`	Output world limits are the bounding rectangle that includes both the input image and the translated output image.

Data Types: `char` | `string`

`FillValues` — Fill values

numeric scalar | numeric array | string scalar | character vector | `missing`

Fill values used for output pixels outside the input image, specified as the comma-separated pair consisting of `'FillValues'` and one of the following values. `imtranslate` uses fill values for output pixels when the corresponding inverse transformed location in the input image is completely outside the input image boundaries.

The default fill value of numeric and logical images is `0`. The default fill value of categorical images is `missing`, which corresponds to the `<undefined>` category.

Image Type	Translation Dimension	Format of Fill Values
2-D grayscale or logical image	2-D	• Numeric scalar
2-D color image or 2-D multispectral image	2-D	• Numeric scalar • *c*-element numeric vector specifying a fill value for each of the *c* channels. The number of channels, *c*, is 3 for color images.
Series of *p* 2-D images	2-D	• Numeric scalar • *c*-by-*p* numeric matrix. The number of channels, *c*, is 1 for grayscale images and 3 for color images.
N-D image	2-D	• Numeric scalar • Numeric array whose size matches dimensions 3-to-*N* of the input image `A`. For example, if `A` is 200-by-200-by-10-by-3, then `FillValues` can be a 10-by-3 array.
3-D grayscale or logical image	3-D	• Numeric scalar
Categorical image	2-D or 3-D	• Valid category in the image, specified as a string scalar or character vector. • `missing`, which corresponds to the `<undefined>` category. For more information, see `missing`.

Example: `255` fills a `uint8` image with white pixels

Example: `1` fills a `double` image with white pixels

Example: `[0 1 0]` fills a `double` color image with green pixels

Example: `[0 1 0; 0 1 1]'`, for a series of two `double` color images, fills the first image with green pixels and the second image with cyan pixels

Example: `"vehicle"` fills a categorical image with the "vehicle" category

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `string` | `char`

Output Arguments

B — Translated image

numeric array | logical array | categorical array

Translated image, returned as a numeric, logical, or categorical array of the same data type as the input image, `A`.

`RB` — Spatial referencing information associated with the output image
`imref2d` object | `imref3d` object

Spatial referencing information associated with the output image, returned as an `imref2d` or `imref3d` spatial referencing object.

Tips

- `imtranslate` is optimized for integrally valued `translation` vectors.
- When `'OutputView'` is `'full'` and `translation` is a fractional number of pixels, then `imtranslate` expands the world limits of the output spatial referencing object to the nearest full pixel increment. `imtranslate` does this so that it contains both the original and translated images at the same resolution as the input image. The additional image extent in each is added on one side of the image, in the direction that the translation vector points. For example, when `translation` is fractional and positive in both X and Y, then `imtranslate` expands the maximum of `XWorldLimits` and `YWorldLimits` to enclose the `'full'` bounding rectangle at the resolution of the input image.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imtranslate` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imtranslate` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- Input images of data type categorical are not supported.
- The function supports only 2-D translation vectors, `translation`. 3-D translations are not supported.

Thread-Based Environment
Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also
`imref2d` | `imref3d` | `imresize` | `imrotate` | `imwarp`

Topics
“Translate an Image Using imtranslate Function”

Introduced in R2014a

imwarp

Apply geometric transformation to image

Syntax

```
B = imwarp(A,tform)
B = imwarp(A,D)
[B,RB] = imwarp(A,RA,tform)
[___] = imwarp(___,interp)
[___] = imwarp(___,Name,Value)
```

Description

`B = imwarp(A,tform)` transforms the numeric, logical, or categorical image `A` according to the geometric transformation `tform`. The function returns the transformed image in `B`.

`B = imwarp(A,D)` transforms image `A` according to the displacement field `D`.

`[B,RB] = imwarp(A,RA,tform)` transforms a spatially referenced image specified by the image data `A` and its associated spatial referencing object `RA`. The outputs are a spatially referenced image specified by the image data `B` and its associated spatial referencing object `RB`.

`[___] = imwarp(___,interp)` specifies the type of interpolation to use.

`[___] = imwarp(___,Name,Value)` specifies name-value arguments to control various aspects of the geometric transformation.

Tip If the input transformation `tform` does not define a forward transform, then use the `OutputView` name-value argument to accelerate the transformation.

Examples

Apply Horizontal Shear to Image

Read grayscale image into workspace and display it.

```
I = imread('cameraman.tif');
imshow(I)
```

Create a 2-D geometric transformation object.

```
tform = affine2d([1 0 0; .5 1 0; 0 0 1])
```

```
tform =
  affine2d with properties:

                T: [3x3 double]
    Dimensionality: 2
```

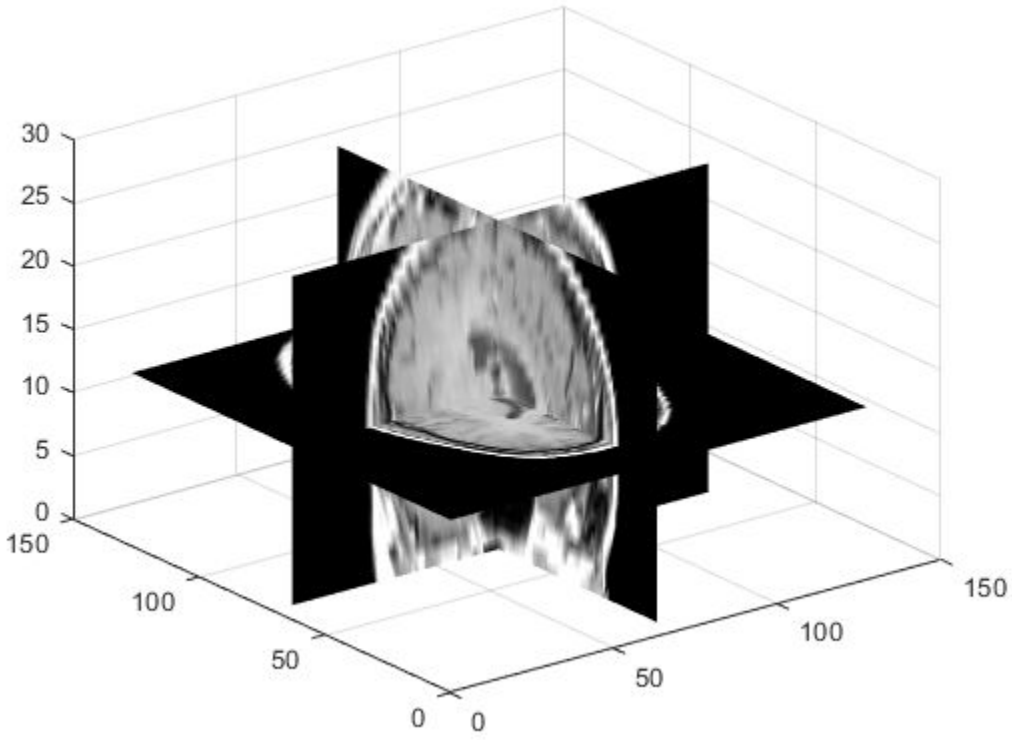
Apply the transformation to the image.

```
J = imwarp(I,tform);
figure
imshow(J)
```

Apply Rotation Transformation to 3-D MRI Dataset

Read 3-D MRI data into the workspace and visualize it.

```
s = load('mri');
mriVolume = squeeze(s.D);
sizeIn = size(mriVolume);
hFigOriginal = figure;
hAxOriginal  = axes;
slice(double(mriVolume),sizeIn(2)/2,sizeIn(1)/2,sizeIn(3)/2);
grid on, shading interp, colormap gray
```



Create a 3-D geometric transformation object. First create a transformation matrix that rotates the image around the *y*-axis. Then create an `affine3d` object from the transformation matrix.

```
theta = pi/8;
t = [cos(theta)  0      -sin(theta)   0
     0             1              0     0
     sin(theta)    0       cos(theta)   0
     0             0              0     1];
tform = affine3d(t)
```

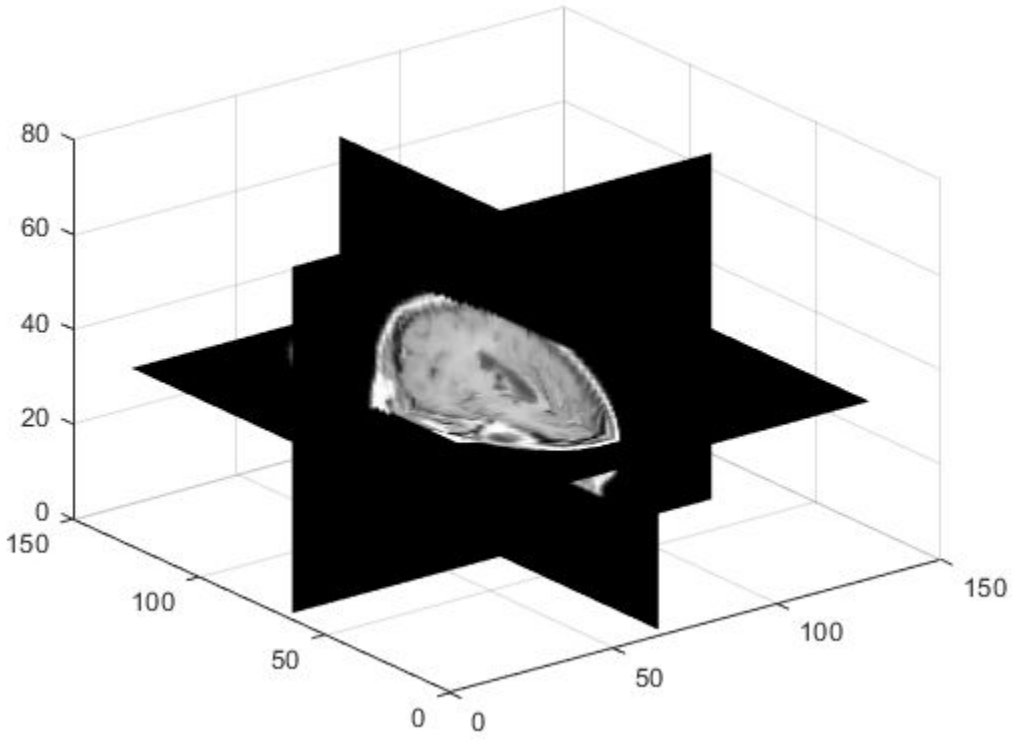
```
tform =
  affine3d with properties:

                T: [4x4 double]
    Dimensionality: 3
```

Apply the transformation to the image.

```
mriVolumeRotated = imwarp(mriVolume,tform);
```

Visualize three slice planes through the center of the transformed volumes.

```
sizeOut = size(mriVolumeRotated);
hFigRotated = figure;
hAxRotated  = axes;
slice(double(mriVolumeRotated),sizeOut(2)/2,sizeOut(1)/2,sizeOut(3)/2)
grid on, shading interp, colormap gray
```

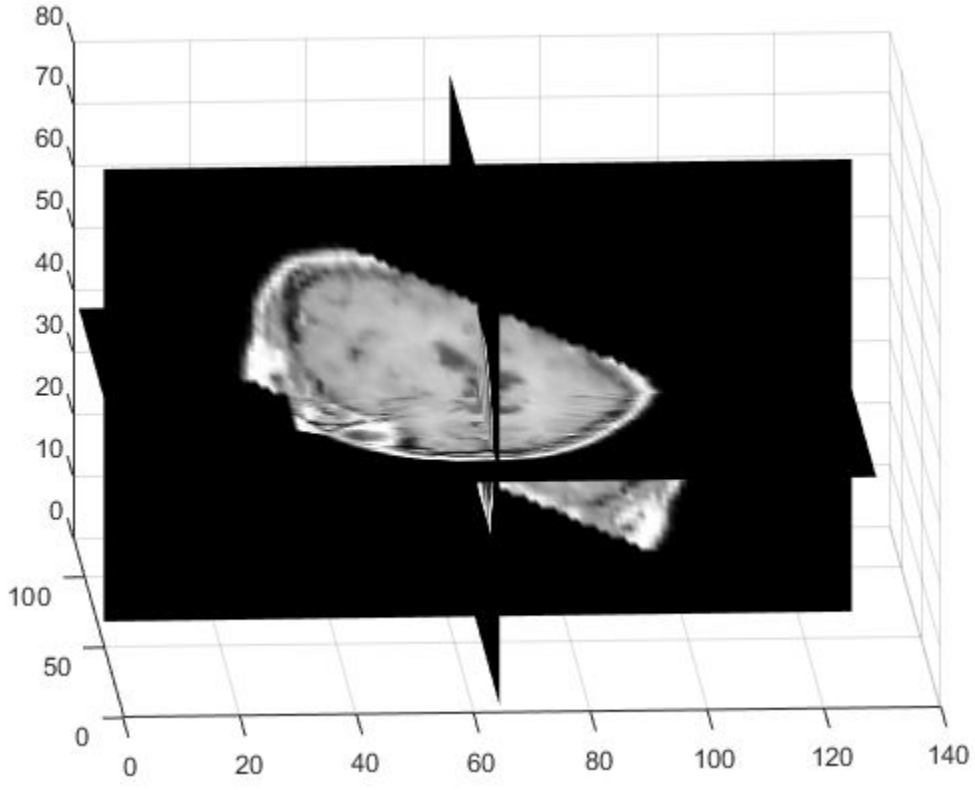


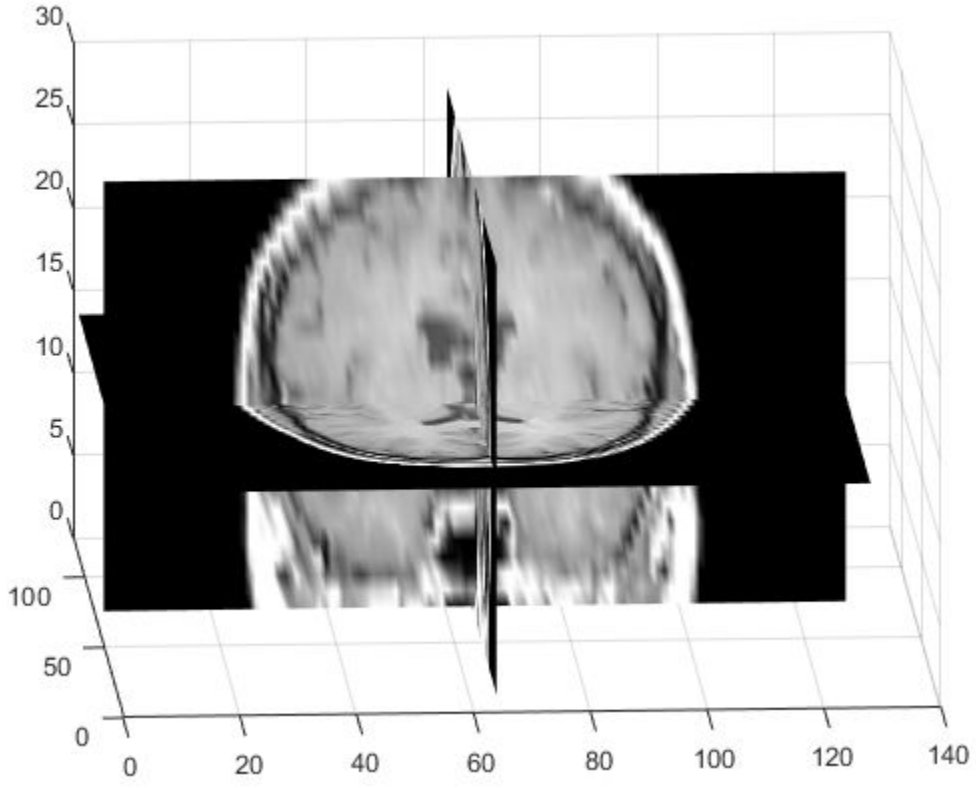
Link the views of both axes together.

```
linkprop([hAxOriginal,hAxRotated],'View');
```

Set the view to see the effect of rotation.

```
set(hAxRotated,'View',[-3.5 20.0])
```


80
70
60
50
40
30
20
10
0
100
50
0
0
20
40
60
80
100
120
140



Warp Image Using Different Output View Styles

Read and display an image. To see the spatial extents of the image, make the axes visible.

```
A = imread('kobi.png');
iptsetpref('ImshowAxesVisible','on')
imshow(A)
```

Create a 2-D affine transformation. This example creates a randomized transformation that consists of scale by a factor in the range [1.2, 2.4], rotation by an angle in the range [-45, 45] degrees, and horizontal translation by a distance in the range [100, 200] pixels.

```
tform = randomAffine2d('Scale',[1.2,2.4],'XTranslation',[100 200],'Rotation',[-45,45]);
```

Create three different output views for the image and transformation.

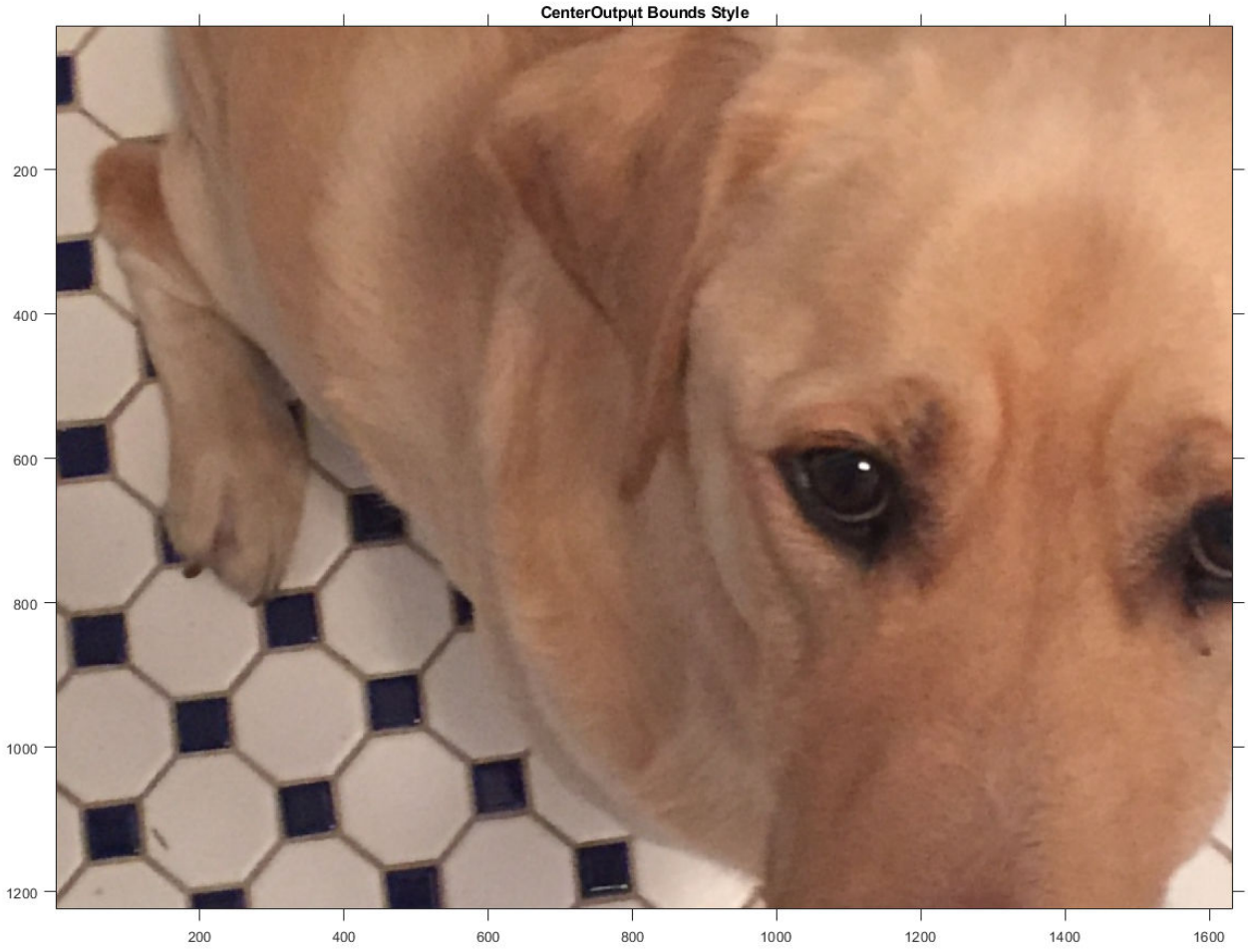
```
centerOutput = affineOutputView(size(A),tform,'BoundsStyle','CenterOutput');
followOutput = affineOutputView(size(A),tform,'BoundsStyle','FollowOutput');
sameAsInput = affineOutputView(size(A),tform,'BoundsStyle','SameAsInput');
```

Apply the transformation to the input image using each of the different output view styles.

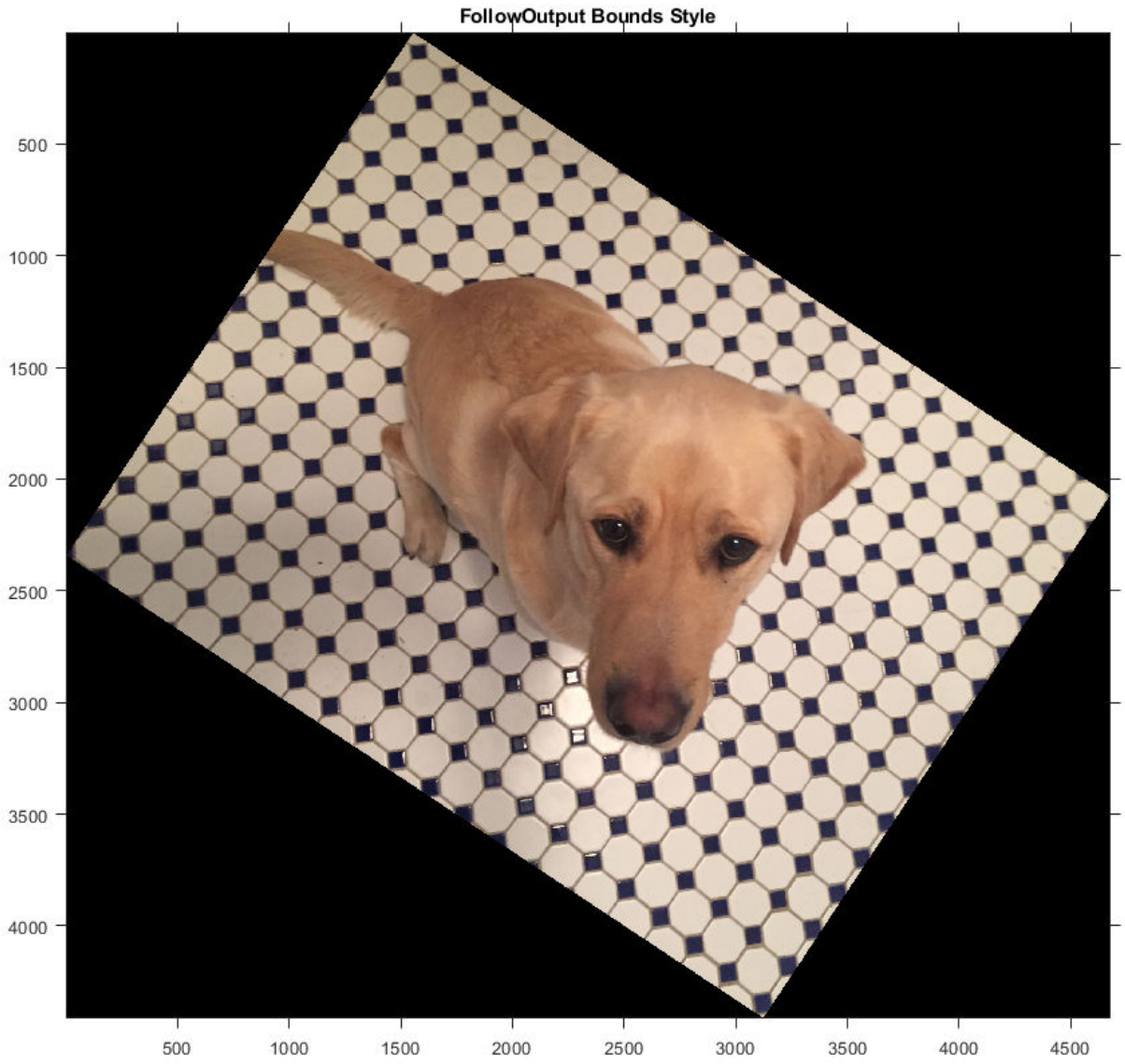
```
BCenterOutput = imwarp(A,tform,'OutputView',centerOutput);
BFollowOutput = imwarp(A,tform,'OutputView',followOutput);
BSameAsInput = imwarp(A,tform,'OutputView',sameAsInput);
```

Display the resulting images.

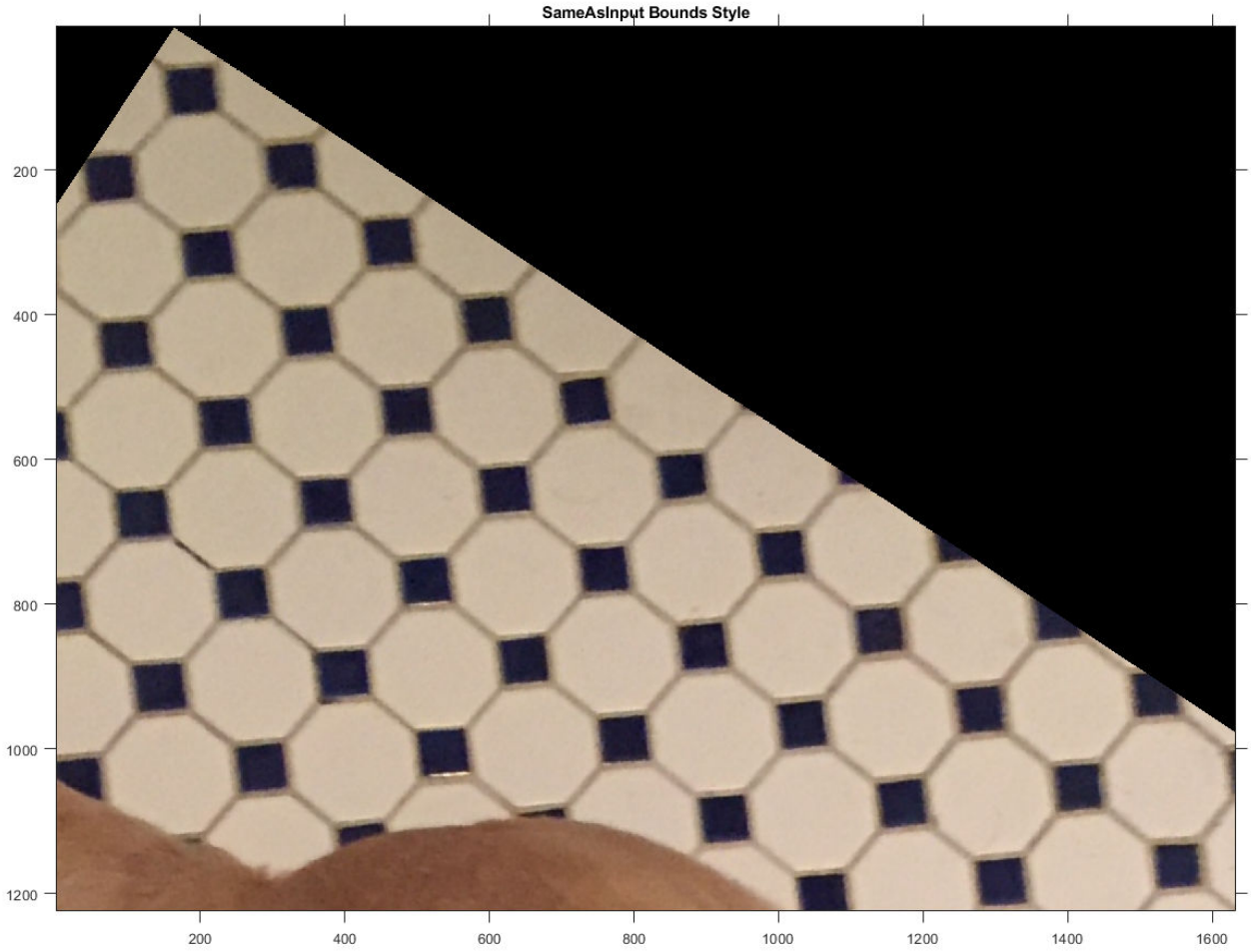
```
imshow(BCenterOutput)
title('CenterOutput Bounds Style');
```




```
imshow(BFollowOutput)
title('FollowOutput Bounds Style');
```




```
imshow(BSameAsInput)
title('SameAsInput Bounds Style');
```




```
iptsetpref('ImshowAxesVisible','off')
```

Input Arguments

A — Image to be transformed

numeric array | logical array | categorical array

Image to be transformed, specified as a numeric, logical, or categorical array of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `categorical`

tform — Geometric transformation

`rigid2d` object | `affine2d` object | `projective2d` object | `rigid3d` object | `affine3d` object

Geometric transformation to apply, specified as a `rigid2d`, `affine2d`, `projective2d`, `rigid3d`, or `affine3d` object.

- If `tform` is 2-D and `A` has more than two dimensions, such as for an RGB image, then `imwarp` applies the same 2-D transformation to all 2-D planes along the higher dimensions.

- If `tform` is 3-D, then `A` must be a 3-D image volume.

`D` — Displacement field
numeric array

Displacement field, specified as numeric array. The displacement field defines the grid size and location of the output image. Displacement values are in units of pixels. `imwarp` assumes that `D` is referenced to the default intrinsic coordinate system. To estimate the displacement field, use `imregdemons`.

- If `A` is a 2-D grayscale image of size *m*-by-*n*, then specify the displacement field as an *m*-by-*n*-by-2 array. `D(:,:,1)` contains displacements along the *x*-axis. `imwarp` adds these values to column and row locations in `D` to produce remapped locations in `A`. Similarly, `D(:,:,2)` contains displacements along the *y*-axis.
- If `A` is a 2-D RGB or multispectral image of size *m*-by-*n*-by-*c* and you specify `D` as an *m*-by-*n*-by-2 array, then `imwarp` operates on each 2-D color channel independently. `D(:,:,1)` contains displacements along the *x*-axis for all of the color channels. Similarly, `D(:,:,2)` contains displacements along the *y*-axis.
- If `A` is a 3-D grayscale image of size *m*-by-*n*-by-*p*, then specify the displacement field array as an *m*-by-*n*-by-*p*-by-3 array. `D(:,:,:,1)` contains displacements along the *x*-axis. `imwarp` adds these values to column, row, and depth locations in `D` to produce remapped locations in `A`. Similarly, `D(:,:,:,2)` and `D(:,:,:,3)` contain displacements along the *y*- and *z*-axis.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`RA` — Spatial referencing information of image to be transformed
`imref2d` object | `imref3d` object

Spatial referencing information of the image to be transformed, specified as an `imref2d` object for a 2-D transformation or an `imref3d` object for a 3-D transformation.

`interp` — Type of interpolation used
`"nearest"` | `"linear"` | `"cubic"`

Type of interpolation used, specified as one of these values.

Interpolation Method	Description
`"nearest"`	Nearest-neighbor interpolation. The output pixel is assigned the value of the pixel that the point falls within. No other pixels are considered. Nearest-neighbor interpolation is the only method supported for categorical images and it is the default method for images of this type.
`"linear"`	Linear interpolation. This is the default interpolation method for numeric and logical images.
`"cubic"`	Cubic interpolation

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `B = imwarp(A,tform,FillValues=255)` uses a fill value of 255

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `B = imwarp(A,tform,"FillValues",255)` uses a fill value of 255

`OutputView` — Size and location of output image
`imref2d` object | `imref3d` object

Size and location of output image in the world coordinate system, specified as an `imref2d` or `imref3d` spatial referencing object. The object has properties that define the size of the output image and the location of the output image in the world coordinate system.

You can create an output view by using the `affineOutputView` function. To replicate the default output view calculated by `imwarp`, use the default bounds style (`"CenterOutput"`) of `affineOutputView`.

You cannot specify `OutputView` when you specify an input displacement field `D`.

`FillValues` — Fill value
numeric scalar | numeric array | string scalar | character vector | `missing`

Fill values used for output pixels outside the input image, specified as one of the values in the table. `imwarp` uses fill values for output pixels when the corresponding inverse transformed location in the input image is completely outside the input image boundaries.

The default fill value of numeric and logical images is `0`. The default fill value of categorical images is `missing`, which corresponds to the `<undefined>` category.

Image Type	Transformation Dimensionality	Format of Fill Values
2-D grayscale or logical image	2-D	• Numeric scalar
2-D color image or 2-D multispectral image	2-D	• Numeric scalar • *c*-element numeric vector specifying a fill value for each of the *c* channels. The number of channels, *c*, is 3 for color images.
Series of *p* 2-D images	2-D	• Numeric scalar • *c*-by-*p* numeric matrix. The number of channels, *c*, is 1 for grayscale images and 3 for color images.
N-D image	2-D	• Numeric scalar • Numeric array whose size matches dimensions 3-to-*N* of the input image `A`. For example, if `A` is 200-by-200-by-10-by-3, then `FillValues` can be a 10-by-3 array.

Image Type	Transformation Dimensionality	Format of Fill Values
3-D grayscale or logical image	3-D	• Numeric scalar
Categorical image	2-D or 3-D	• Valid category in the image, specified as a string scalar or character vector. • `missing`, which corresponds to the `<undefined>` category. For more information, see `missing`.

Example: `255` fills a `uint8` image with white pixels

Example: `1` fills a `double` image with white pixels

Example: `[0 1 0]` fills a `double` color image with green pixels

Example: `[0 1 0; 0 1 1]'`, for a series of two `double` color images, fills the first image with green pixels and the second image with cyan pixels

Example: `"vehicle"` fills a categorical image with the "vehicle" category

`SmoothEdges` — Pad image to create smooth edges
`false` (default) | `true`

Pad image to create smooth edges, specified as `true` or `false`. When set to `true`, `imwarp` create a smoother edge in the output image by padding the input image with values specified by `FillValues`. When set to `false`, `imwarp` does not pad the image. Choosing `false` (not padding) the input image can result in a sharper edge in the output image. This sharper edge can be useful to minimize seam distortions when registering two images side by side.

Output Arguments

`B` — Transformed image
numeric array | logical array | categorical array

Transformed image, returned as a numeric, logical, or categorical array of the same data type as the input image `A`.

`RB` — Spatial referencing information of transformed image
`imref2d` object | `imref3d` object

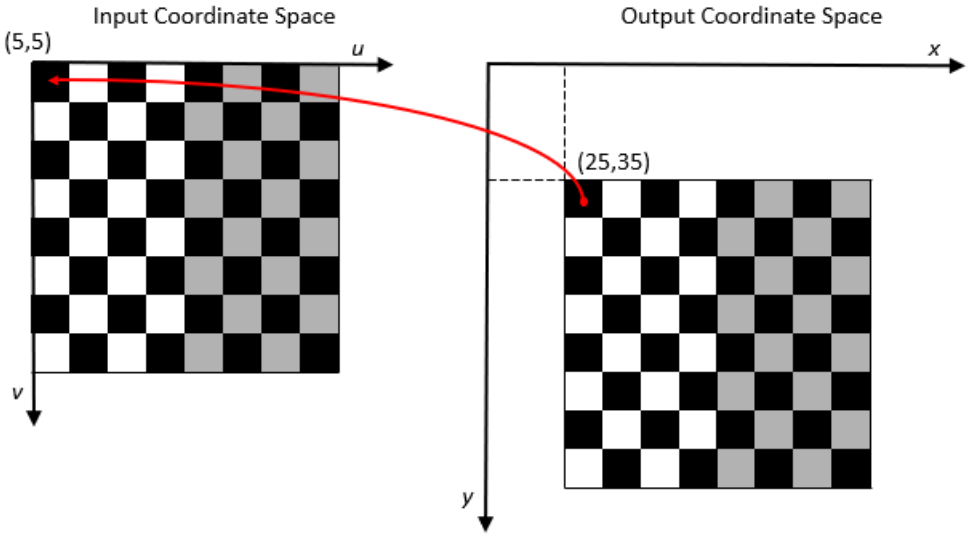
Spatial referencing information of the transformed image, returned as an `imref2d` or `imref3d` spatial referencing object.

Algorithms

`imwarp` determines the value of pixels in the output image by mapping locations in the output image to the corresponding locations in the input image (an inverse mapping). When the center of a pixel in the output image does not map to the center of a pixel in the input image, `imwarp` interpolates within the input image to calculate the output pixel value.

The figure illustrates a translation transformation of a checkerboard image, in which each square is 10-by-10 pixels. By convention, the axes in input space are labeled *u* and *v* and the axes in output

space are labeled *x* and *y*. Using the inverse transformation, the pixel with (*x*,*y*) coordinates (25,35) in the output coordinate space is mapped to the (*u*,*v*) coordinates (5,5) in the input coordinate space.



`imwarp` performs the mapping using world coordinates. For more information, see “Image Coordinate Systems”.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `imwarp` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `imwarp` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- Input images of data type categorical are not supported.
- The geometric transformation object input, `tform`, must be a `rigid2d`, `affine2d`, or `projective2d` object.
- The interpolation method and optional parameter names must be constants.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- Input images of data type categorical are not supported.
- The geometric transformation object input, `tform`, must be a `rigid2d`, `affine2d`, or `projective2d` object.
- The interpolation method and optional parameter names must be constants.

- The spatial referencing information output, `RB`, is not supported.

Thread-Based Environment
Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Only `"nearest"` and `"linear"` interpolation types are supported.
- Only `false` is supported for the value of the `SmoothEdges` argument.

For more information, see “Image Processing on a GPU”.

See Also

Apps
Registration Estimator

Functions
`affineOutputView` | `imregister` | `imregtform` | `imregdemons` | `imtranslate` | `randomWindow2d` | `centerCropWindow2d`

Objects
`affine2d` | `affine3d` | `rigid2d` | `rigid3d` | `projective2d` | `geometricTransform2d` | `geometricTransform3d`

Topics
“2-D and 3-D Geometric Transformation Process Overview”

Introduced in R2013a

ind2gray

Convert indexed image to grayscale image

Syntax

```
I = ind2gray(X,cmap)
```

Description

`I = ind2gray(X,cmap)` converts the indexed image `X` with colormap `cmap` to a grayscale image, `I`. The `ind2gray` function removes the hue and saturation information from the input image while retaining the luminance.

Examples

Convert Indexed Image to Grayscale

Load an indexed image into the workspace.

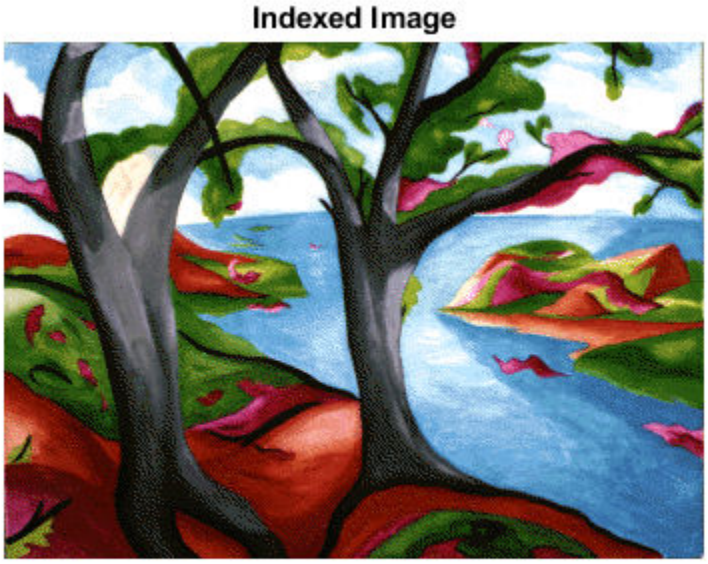
```
[X, map] = imread('trees.tif');
```

Convert the image to grayscale using `ind2gray`.

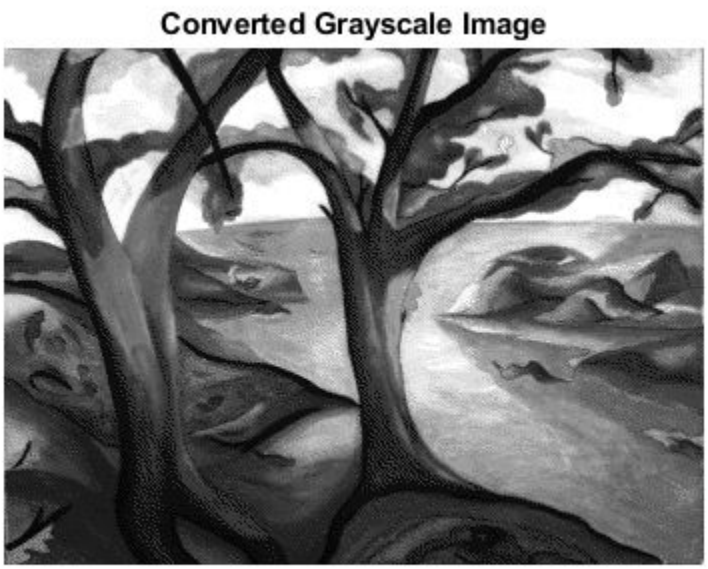
```
I = ind2gray(X,map);
```

Display the indexed image and the converted grayscale image.

```
imshow(X,map)
title('Indexed Image')
```




```
figure
imshow(I)
title('Converted Grayscale Image')
```



Input Arguments

X — Indexed image

numeric array

Indexed image, specified as a numeric array of any size and dimensionality.

Data Types: `single` | `double` | `uint8` | `uint16`

cmap — Colormap

c-by-3 numeric matrix

Colormap associated with indexed image `X`, specified as a *c*-by-3 numeric matrix with values in the range [0, 1]. Each row is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap.

Data Types: `double`

Output Arguments

I — Grayscale image

numeric array

Grayscale image, specified as a numeric array. `I` has the same size, dimensionality and class as `X`.

Algorithms

`ind2gray` converts the colormap to NTSC coordinates using `rgb2ntsc`, and sets the hue and saturation components (*I* and *Q*) to zero, creating a gray colormap. `ind2gray` then replaces the indices in the image `X` with the corresponding grayscale intensity values in the gray colormap.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Image Viewer | `gray2ind` | `imshow` | `mat2gray` | `im2gray` | `rgb2ntsc`

Introduced before R2006a

inpaintCoherent

Restore specific image regions using coherence transport based image inpainting

Syntax

```
J = inpaintCoherent(I,mask)
J = inpaintCoherent(I,mask,Name,Value)
```

Description

`J = inpaintCoherent(I,mask)` restores specific regions in the input image using the coherence transport based inpainting method. `mask` is a logical image that denotes the target regions in the image to be filled through inpainting.

`J = inpaintCoherent(I,mask,Name,Value)` specifies additional inpainting options using one or more name-value arguments.

Examples

Remove Overlayed Text From Image Through Inpainting

Read an image to be inpainted into the workspace. This image contains text overlays to be removed.

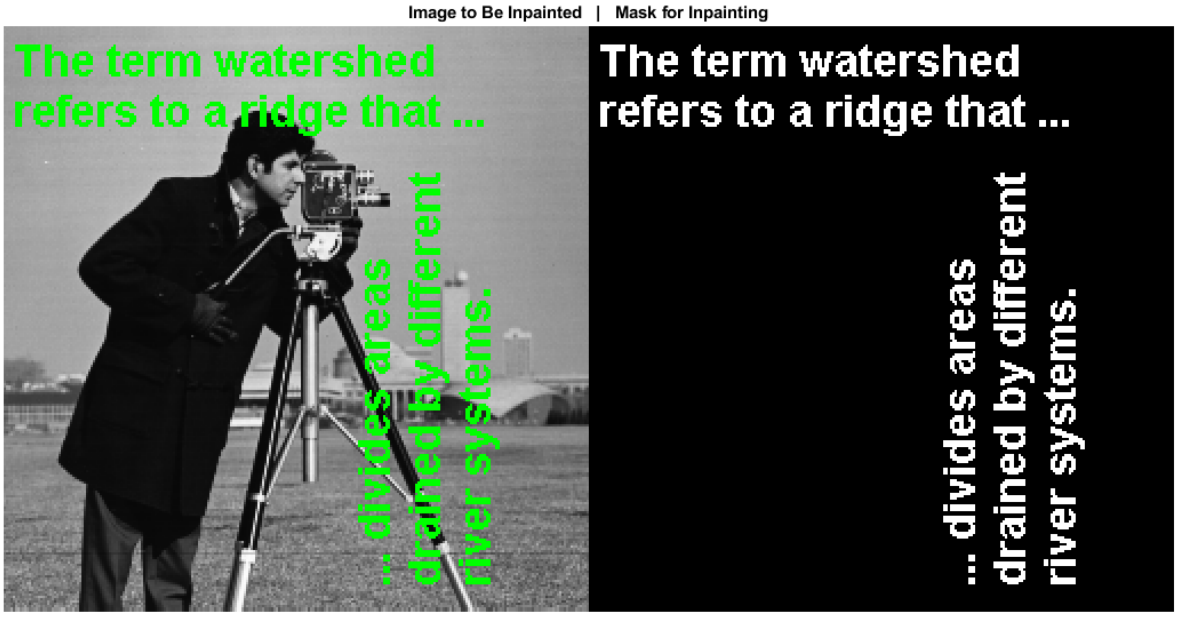
```
I = imread('overlayimage.png');
```

Read the mask image into the workspace. This mask image contains the overlayed text regions present in the image to be inpainted.

```
mask = imread('text.png');
```

Display the image to be inpainted and its corresponding mask image.

```
montage({I,mask});
title(['Image to Be Inpainted','   |   ','Mask for Inpainting'])
```



Inpaint the original image by removing the text overlays.

```
J = inpaintCoherent(I,mask);
```

Display the original image and the inpainted image.

```
montage({I,J});
title(['Image to Be Inpainted','    |    ','Inpainted Image'])
```

Remove Objects in Image Regions Through Inpainting

Read an image to be inpainted into the workspace.

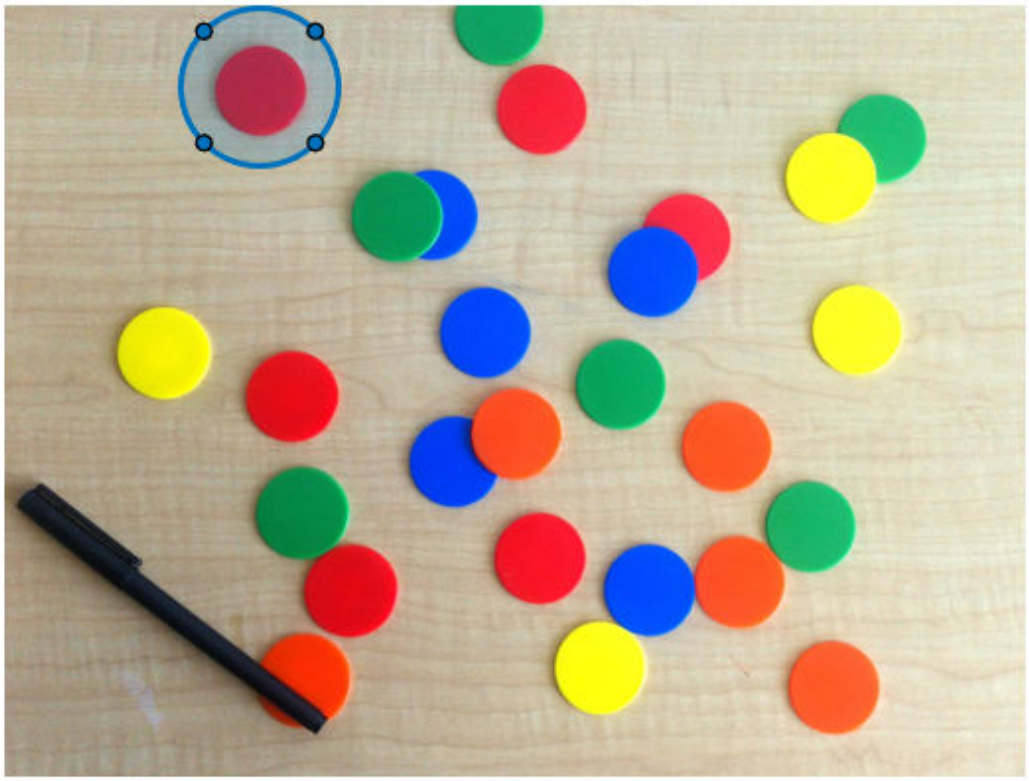
```
I = imread('coloredChips.png');
```

Display the image.

```
figure
imshow(I,[])
```

Use the `drawcircle` function to select a circular region of interest (ROI) for inpainting. Use the `Center` and `Radius` name-value pairs to specify the location of an ROI.

```
h = drawcircle('Center',[130,42],'Radius',40);
```



Select Multiple ROIs for Inpainting

You can also select multiple ROIs iteratively.

Set the number of regions to be inpainted to 6.

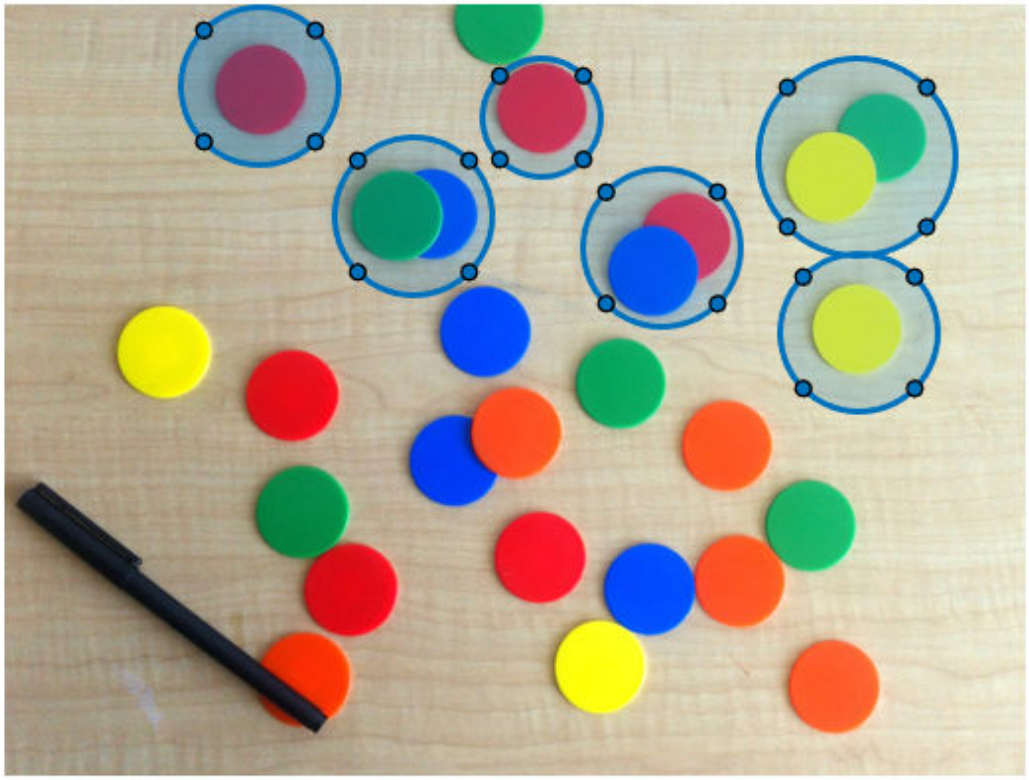
```
numRegion = 6;
```

Specify the center and radii for each region.

```
roiCenter = [130 42;433 78;208 108;334 124;434 167;273 58];
roiRadius = [40 50 40 40 40 30];
```

Select multiple circular ROIs iteratively by specifying the `drawcircle` `Center` and `Radius` name-value pairs.

```
roi = cell([numRegion,1]);
for i = 1:numRegion
    c = roiCenter(i,:);
    r = roiRadius(i);
    h = drawcircle('Center',c,'Radius',r);
    roi{i} = h;
end
```



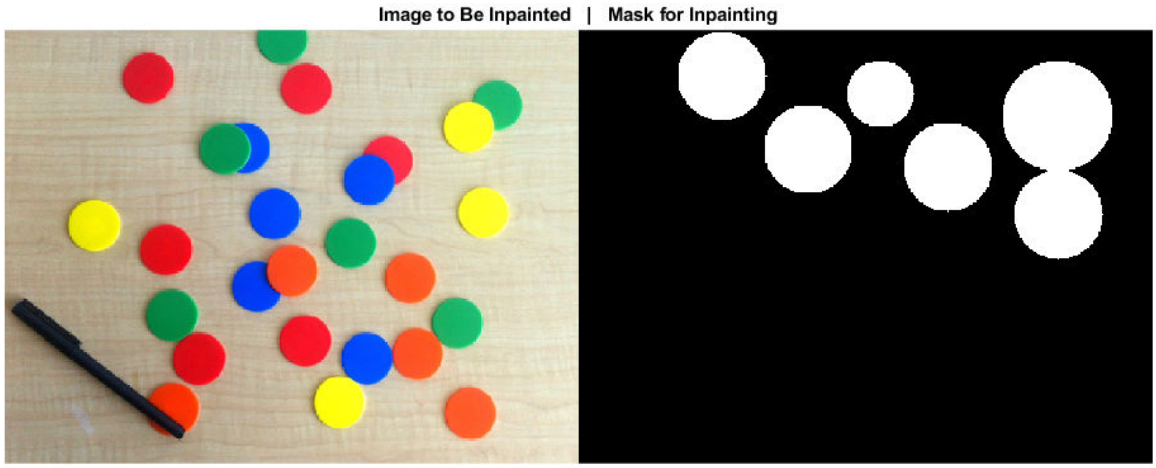
Use the `createMask` function to generate a mask from the selected ROIs.

```
mask = zeros(size(I,1),size(I,2));
for i = 1:numRegion
    newmask = createMask(roi{i});
```

```
    mask = xor(mask,newmask);
end
```

Display the image to be inpainted and its corresponding mask image.

```
montage({I,mask});
title(['Image to Be Inpainted','    |    ','Mask for Inpainting'])
```

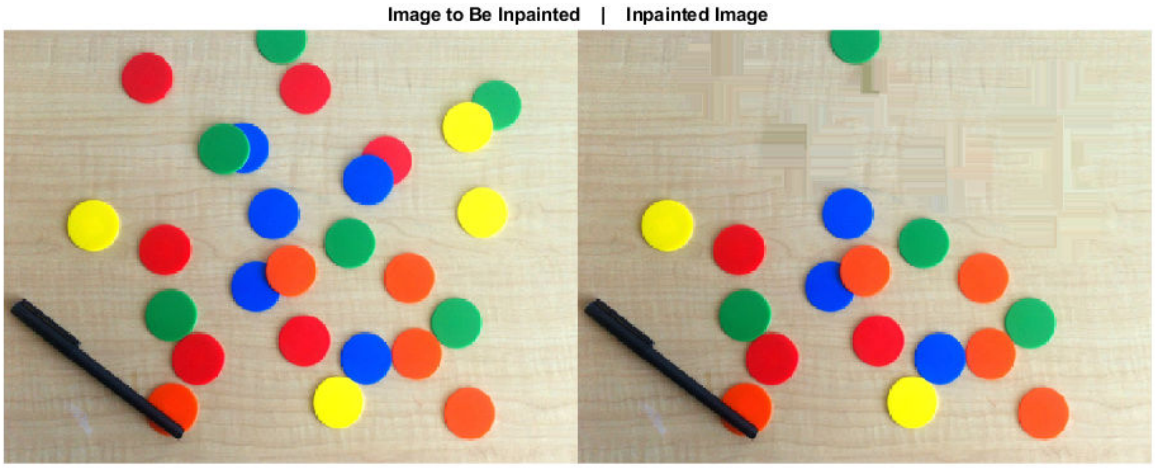


Remove objects in the ROIs through inpainting. Specify a standard deviation of 0.5 and an inpainting radius of 1.

```
J = inpaintCoherent(I,mask,'SmoothingFactor',0.5,'Radius',1);
```

Display the original image and the inpainted image.

```
montage({I,J});
title(['Image to Be Inpainted','    |    ','Inpainted Image']);
```



Input Arguments

`I` — Image to inpaint

grayscale image | RGB color image

Image to inpaint, specified as a grayscale image of size *m*-by-*n* or an RGB color image of size *m*-by-*n*-by-3.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`mask` — Spatial mask of target regions

2-D binary image

Spatial mask of target regions, specified as a 2-D binary image of size *m*-by-*n*, where *m* and *n* are the dimensions of input image `I`. The nonzero pixels in `mask` constitute the target regions to be filled through inpainting.

Note

- You can generate the mask using any of these functions: `drawcircle`, `drawpolygon`, `drawrectangle`, `drawassisted`, or `drawfreehand`. Alternatively, you can use the segmentation tools in **Image Segmenter** app.

Data Types: `logical`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `J = inpaintCoherent(I,mask,'Radius',7)`

`SmoothingFactor` — Standard deviation of Gaussian filter
2 (default) | positive number

Standard deviation of Gaussian filter, specified as the comma-separated pair consisting of `'SmoothingFactor'` and a positive number. This value is used to compute the scales of the Gaussian filters while estimating the coherence direction.

`Radius` — Inpainting radius
5 (default) | positive integer

Inpainting radius, specified as the comma-separated pair consisting of `'Radius'` and a positive integer. The inpainting radius denotes the radius of the circular neighborhood region centered on the pixel to be inpainted.

Output Arguments

`J` — Inpainted image
grayscale image | RGB color image

Inpainted image, returned as a grayscale image or RGB color image of the same size and data type as input image `I`.

Tips

- The inpainting results depend on the name-value pair specification. You can modify the values of `'Radius'` and `'SmoothingFactor'` for varied results.
- Each ROI in the binary mask image must be sufficiently large to enclose the corresponding region in the image to be inpainted.

Algorithms

The coherence transport based inpainting method is a pixel-based approach for removing objects and filling regions in images [1]. Inpainting is performed inwards starting from the boundary pixels of the target region. The inpainting value for a pixel is estimated from its coherent neighboring pixels with known values. The steps involved are summarized as follows:

1. Identify target regions from the input image to be filled or inpainted. Generate a binary mask of the size same as the input image. The nonzero pixels in the mask image must contain the target regions to be inpainted.

   The order in which the pixels in the target region are inpainted is calculated from their Euclidean distance to the boundary of the target region.
2. The inpainting value for a pixel in the target region is the weighted average of known pixel values within its inpainting radius. The known pixels along the coherence direction are assigned higher weight value than the incoherent neighboring pixels. The coherence direction is estimated by using a structure tensor.

References

[1] F. Bornemann and T. März. "Fast Image Inpainting Based on Coherence Transport." *Journal of Mathematical Imaging and Vision.* Vol. 28, 2007, pp. 259–278.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`inpaintCoherent` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

See Also

`imfill` | `regionfill` | `inpaintExemplar` | `roifilt2`

Introduced in R2019a

inpaintExemplar

Restore specific image regions using exemplar-based image inpainting

Syntax

```
J = inpaintExemplar(I,mask)
J = inpaintExemplar(I,mask,Name,Value)
```

Description

`J = inpaintExemplar(I,mask)` fills specific regions in the input image using the exemplar-based inpainting method. `mask` is a logical image that denotes the target regions in the image to be filled using inpainting.

`J = inpaintExemplar(I,mask,Name,Value)` specifies additional inpainting options using one or more name-value arguments.

Examples

Remove Objects in Image Using Inpainting

Read an image into the workspace.

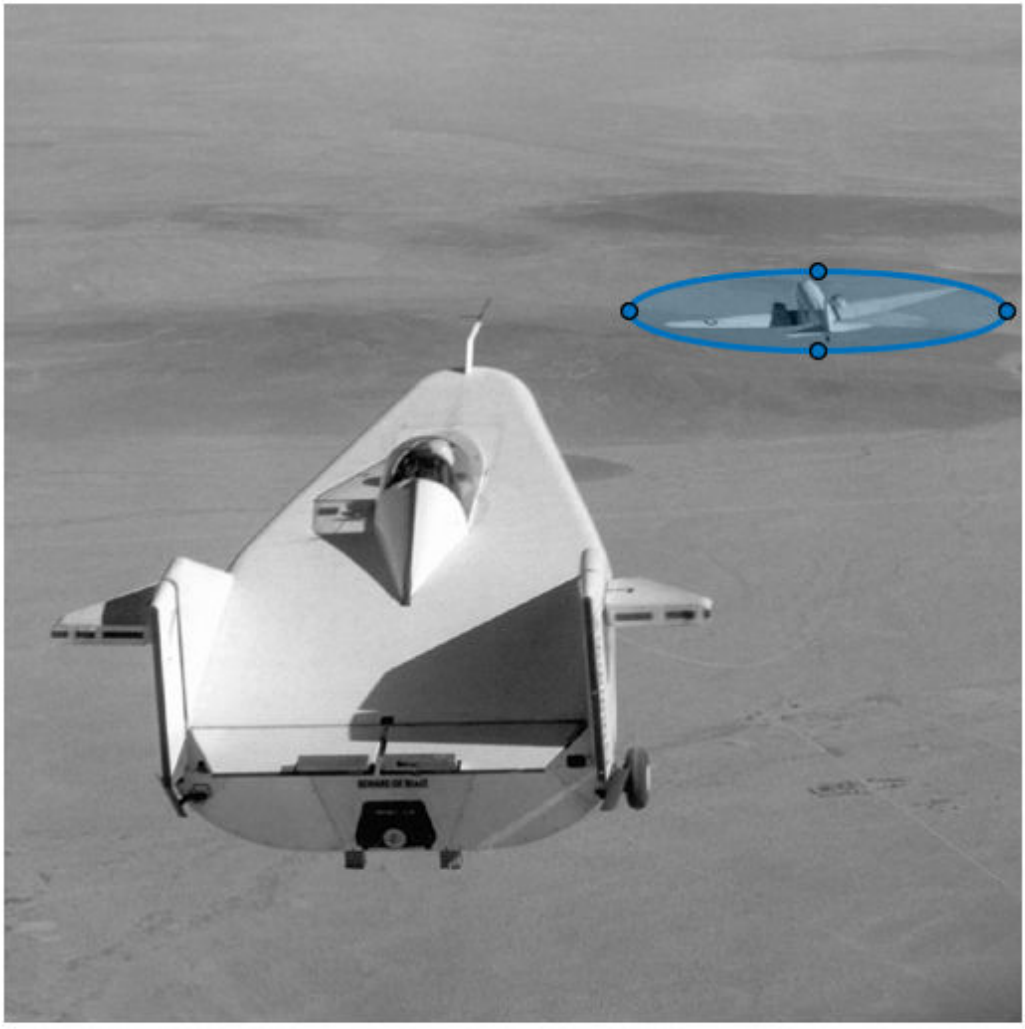
```
I = imread('liftingbody.png');
```

Display the image.

```
figure
imshow(I,[])
```

Use the `drawellipse` function to select an elliptical region of interest (ROI) for inpainting. Use the `'Center'` and `'SemiAxes'` name-value pairs to specify the location of an ROI.

```
h = drawellipse('Center',[410 155],'SemiAxes',[95 20]);
```

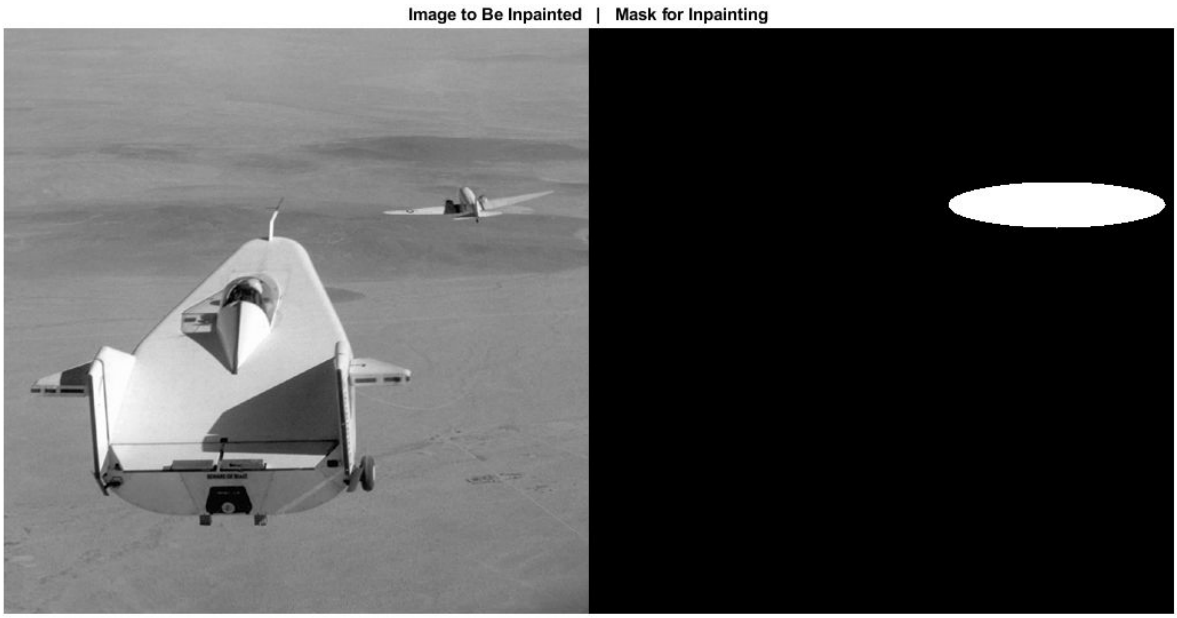


Use the `createMask` function to generate a mask from the selected ROIs.

```
mask = createMask(h);
```

Display the image to be inpainted and its corresponding mask image.

```
montage({I,mask});
title(['Image to Be Inpainted','    |    ','Mask for Inpainting'])
```

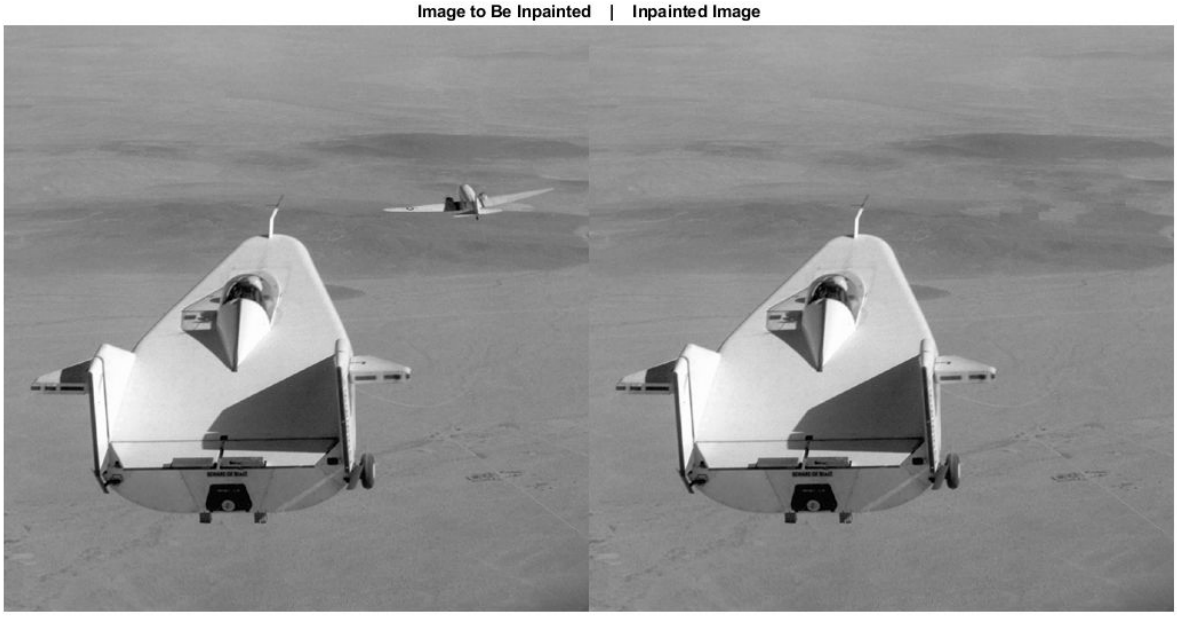


Remove objects in the ROI by using inpainting.

```
J = inpaintExemplar(I,mask);
```

Display the original image and the inpainted image.

```
montage({I,J});
title(['Image to Be Inpainted','    |    ','Inpainted Image']);
```



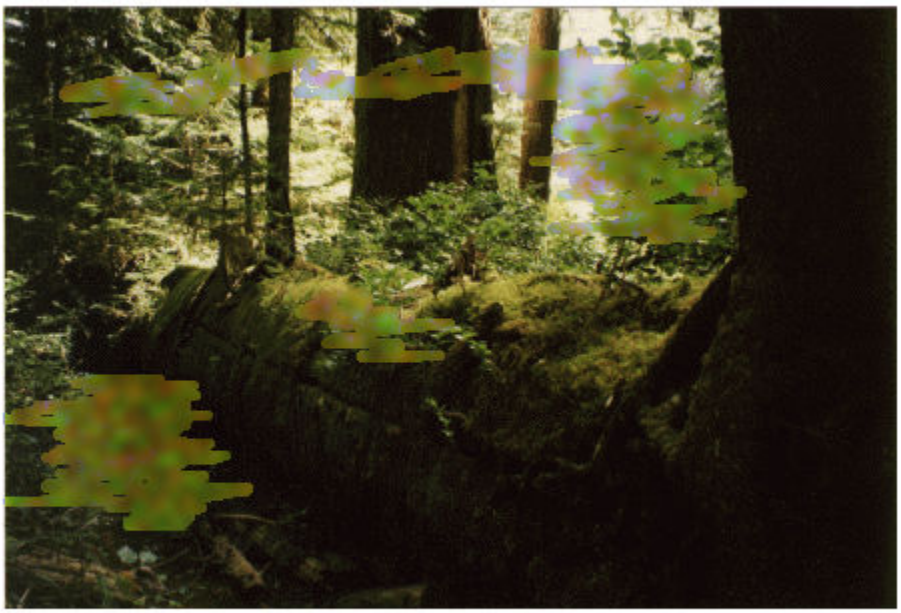
Restore Distorted Image Regions Using Inpainting

Read an image into the workspace.

```
I = imread('forestdistorted.png');
```

Display the image. The image comprises distorted regions to be restored using inpainting.

```
figure
imshow(I,[])
```

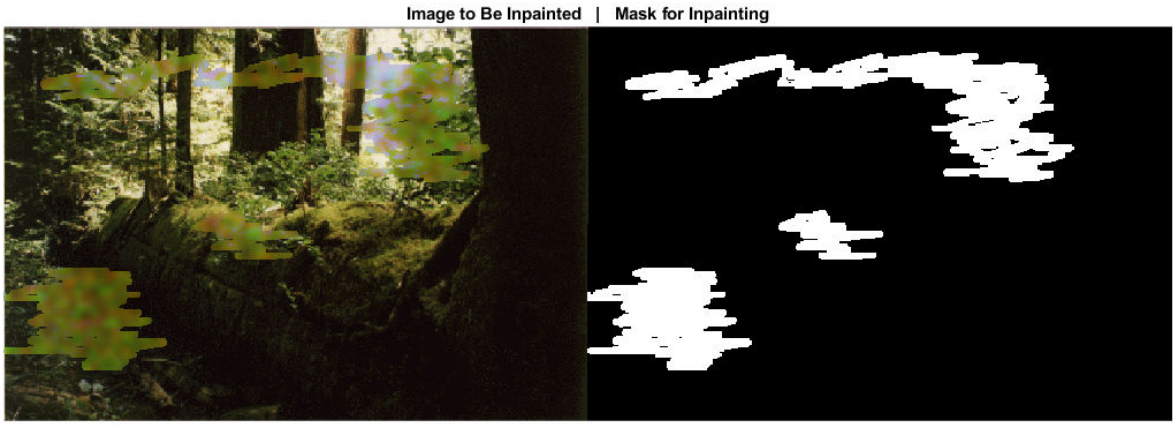


Read a binary mask image containing the distorted image regions into the workspace.

```
mask = imread('imagemask.png');
```

Display the image to be inpainted and its corresponding mask image.

```
montage({I,mask});
title(['Image to Be Inpainted','    |    ','Mask for Inpainting'])
```

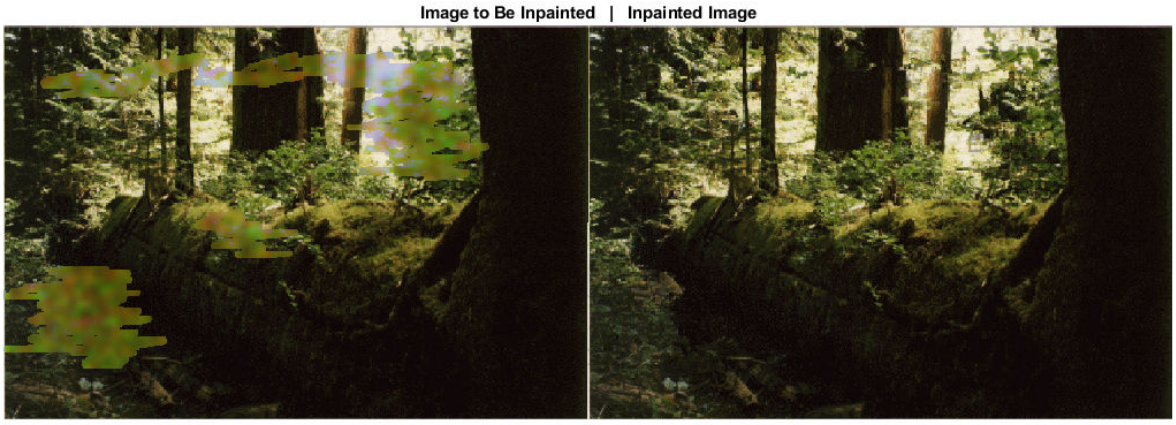


Inpaint the original image to restore the distorted image region. Specify the fill order and the patch size for inpainting as `tensor` and `7`, respectively.

```
J = inpaintExemplar(I,mask,'FillOrder','tensor','PatchSize',7);
```

Display the original image and the inpainted image.

```
montage({I,J});
title(['Image to Be Inpainted','   |   ','Inpainted Image'])
```



Input Arguments

`I` — Image to be inpainted

2-D grayscale image | RGB image

Image to be inpainted, specified as a 2-D grayscale image or an RGB image of size *m*-by-*n*.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`mask` — Spatial mask of target regions

2-D binary image

Spatial mask of target regions, specified as a 2-D binary image of the same size as the input image `I`. The nonzero pixels in `mask` specify the target regions to be filled using inpainting.

Data Types: `logical`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `J = inpaintExemplar(I,mask,'FillOrder','gradient')`

`FillOrder` — Filling order

`'gradient'` (default) | `'tensor'`

Filling order, specified as the comma-separated pair consisting of `'FillOrder'` and `'gradient'` or `'tensor'`. The filling order denotes the priority function to be used for calculating the patch priority. The patch priority value specifies the order of filling for the image patches in target regions.

Data Types: `char` | `string`

`PatchSize` — Size of image patch

`[9 9]` (default) | scalar | vector

Size of the image patch, specified as the comma-separated pair consisting of `'PatchSize'` and one of these options.

- A scalar, *s* — The image patch is a square region of size *s*-by-*s*.
- A vector of form [*p s*] — The image patch is a square or rectangular region of size *p*-by-*s*.

The default image patch size is 9-by-9. An image patch refers to the image region considered for patch matching and inpainting.

Note

- The size of the image patch must be at least 3-by-3 and always less than the size of the input image.

Data Types: `double`

Output Arguments

`J` — Inpainted image

2-D grayscale image | RGB image

Inpainted image, returned as a 2-D grayscale image or an RGB image of the same size and data type as input image `I`.

Algorithms

The exemplar-based image inpainting algorithm is a patch-based approach that restores target regions in the input image by using these steps.

1. Identify target regions from the input image.
2. Generate a binary mask of the same size as the input image. The nonzero pixels in the mask image must correspond to the target regions to be inpainted.
3. Identify the source region. All regions, excluding the target regions, in the input image comprise the source region. That is, *source region = input image − target regions*.
4. For every patch of size *p*-by-*s* centered on a boundary pixel in the target region, compute the patch priority by using the gradient or tensor method.
5. Find the patch with the maximum priority. This patch constitutes the target patch to be inpainted.
6. Given the target patch, search for the best-matching patch in the source region by using the sum of square difference (SSD).
7. Copy image data from the best-matching patch to the target patch.
8. Update the input image, mask, and patch priority value.
9. Repeat steps 4–8 until the target regions are inpainted.

References

[1] Criminisi, A., P. Perez, and K. Toyama. "Region Filling and Object Removal by Exemplar-Based Image Inpainting." *IEEE Transactions on Image Processing*. Vol. 13, No. 9, 2004, pp. 1200–1212.

[2] Le Meur, O., M. Ebdelli, and C. Guillemot. "Hierarchical Super-Resolution-Based-Inpainting." *IEEE Transactions on Image Processing*. Vol. 22, No. 10, 2013, pp. 3779–3790.

See Also

`imfill` | `regionfill` | `inpaintCoherent` | `roifilt2`

Topics

“Interactive Image Inpainting Using Exemplar Matching”

Introduced in R2019b

integralBoxFilter

2-D box filtering of integral images

Syntax

```
B = integralBoxFilter(A)
B = integralBoxFilter(A,filterSize)
B = integralBoxFilter( ___ ,Name,Value)
```

Description

`B = integralBoxFilter(A)` filters the integral image `A` with a 3-by-3 box filter. Returns the filtered image, `B`.

`B = integralBoxFilter(A,filterSize)` filters the integral image `A` with a 2-D box filter with size specified by `filterSize`.

`B = integralBoxFilter( ___ ,Name,Value)` uses name-value pairs to control various aspects of the filtering.

Examples

Filter Integral Image

Read image into the workspace.

```
A = imread('cameraman.tif');
```

Pad the image by the radius of the filter neighborhood. This example uses an 11-by-11 filter.

```
filterSize = [11 11];
padSize = (filterSize-1)/2;
Apad = padarray(A, padSize, 'replicate','both');
```

Compute the integral image of the padded input image.

```
intA = integralImage(Apad);
```

Filter the integral image.

```
B = integralBoxFilter(intA, filterSize);
```

Display original image and filtered image.

```
figure
imshow(A)
title('Original Image')
```


```
figure
imshow(B,[])
title('Filtered Image')
```

Filter Image with Horizontal and Vertical Motion Blur

Read image into the workspace.

```
A = imread('cameraman.tif');
```

Pad the image by radius of the filter neighborhood, calculated `(11-1)/2`.

```
padSize = [5 5];
Apad = padarray(A, padSize, 'replicate', 'both');
```

Calculate the integral image of the padded input.

```
intA = integralImage(Apad);
```

Filter the integral image with a vertical [11 1] filter.

```
Bvert = integralBoxFilter(intA, [11 1]);
```

Crop the output to retain input image size and display it.

```
Bvert = Bvert(:,6:end-5);
```

Filter the integral image with a horizontal [1 11] filter.

```
Bhorz = integralBoxFilter(intA, [1 11]);
```

Crop the output to retain input image size.

```
Bhorz = Bhorz(6:end-5,:);
```

Display the original image and the filtered images.

```
figure,
imshow(A)
title('Original Image')
```

```
figure,
imshow(Bvert,[])
title('Filtered with Vertical Filter')
```


```
figure,
imshow(Bhorz,[])
title('Filtered with Horizontal Filter')
```

Input Arguments

`A` — Integral image to be filtered

numeric array

Integral image to be filtered, specified as a numeric array of any dimension.

The integral image must be upright — `integralBoxFilter` does not support rotated integral images. The first row and column of the integral image is assumed to be zero-padded, as returned by `integralImage`.

Data Types: `double`

`filterSize` — Size of box filter

3 (default) | positive, odd integer | 2-element vector of positive, odd integers

Size of box filter, specified as a positive, odd integer or 2-element vector of positive, odd integers. If `filterSize` is scalar, then `integralBoxFilter` uses a square box filter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `B = integralBoxFilter(A,5,'NormalizationFactor',1);`

`NormalizationFactor` — Normalization factor applied to box filter

`1/filterSize.^2`, if scalar, and `1/prod(filterSize)`, if vector (default) | numeric scalar

Normalization factor applied to box filter, specified as a numeric scalar.

The default `'NormalizationFactor'` has the effect of a mean filter — the pixels in the output image are the local means of the image. To get local area sums, set `'NormalizationFactor'` to `1`. To avoid overflow in such circumstances, consider using double precision images by converting the input image to class `double`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`B` — Filtered image

numeric array

Filtered image, returned as a numeric array. `integralBoxFilter` returns only the parts of the filtering that are computed without padding.

Data Types: `double`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `integralBoxFilter` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- The `'NormalizationFactor'` parameter must be a compile-time constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The `'NormalizationFactor'` parameter must be a compile-time constant.

See Also

`imboxfilt` | `integralImage`

Topics

“Integral Image”

Introduced in R2015b

integralBoxFilter3

3-D box filtering of 3-D integral images

Syntax

```
B = integralBoxFilter3(A)
B = integralBoxFilter3(A,filterSize)
B = integralBoxFilter3( ___ ,Name,Value)
```

Description

`B = integralBoxFilter3(A)` filters integral image `A` with a 3-by-3-by-3 box filter.

`B = integralBoxFilter3(A,filterSize)` filters integral image `A` with a 3-D box filter with size specified by `filterSize`.

`B = integralBoxFilter3( ___ ,Name,Value)` uses name-value pairs to control various aspects of the filtering.

Examples

Filter 3-D MRI Volume with Box Filter

Load 3-D MRI data.

```
volData = load('mri');
vol = squeeze(volData.D);
```

Pad the image volume by the radius of the filter neighborhood.

```
filterSize = [5 5 3];
padSize = (filterSize-1)/2;
volPad = padarray(vol, padSize, 'replicate', 'both');
```

Calculate the 3-D integral image of the padded input.

```
intVol = integralImage3(volPad);
```

Filter the 3-D integral image with a [5 5 3] filter.

```
volFilt = integralBoxFilter3(intVol, filterSize);
```

Input Arguments

A — Integral image to be filtered

3-D numeric array

Integral image to be filtered, specified as a 3-D numeric array.

`integralBoxFilter3` expects the input integral image, `A`, to be an upright integral image computed using `integralImage3`. `integralBoxFilter3` does not support rotated integral images. The first row, column and plane of the integral image is assumed to be padded, as returned by `integralImage3`.

Data Types: `double`

`filterSize` — Size of box filter
3 (default) | positive, odd integer | 3-element vector of positive, odd integers

Size of box filter, specified as a positive odd integer or 3-element vector of positive, odd integers. If `filterSize` is scalar, then `integralBoxFilter3` uses a cube box filter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `B = integralBoxFilter3(A,5,'NormalizationFactor',1);`

`NormalizationFactor` — Normalization factor applied to box filter
`1/filterSize.^3`, if scalar, and `1/prod(filterSize)`, if vector (default) | numeric scalar

Normalization factor applied to box filter, specified as a numeric scalar.

The default `'NormalizationFactor'` has the effect of a mean filter—the pixels in the output image are the local means of the image. To get local area sums, set `'NormalizationFactor'` to `1`. To avoid overflow in such circumstances, consider using double precision images by converting the input image to class `double`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`B` — Filtered image
3-D numeric array

Filtered image, returned as a 3-D numeric array. `integralBoxFilter3` returns only the parts of the filtering that are computed without padding.

Data Types: `double`

See Also
`imboxfilt3` | `integralImage3`

Topics
“Integral Image”

Introduced in R2015b

integralImage

Calculate 2-D integral image

Syntax

```
J = integralImage(I)
J = integralImage(I,orientation)
```

Description

In an integral image, each pixel represents the cumulative sum of a corresponding input pixel with all pixels above and to the left of the input pixel.

An integral image enables you to rapidly calculate summations over image subregions. Subregion summations can be computed in constant time as a linear combination of only four pixels in the integral image, regardless of the size of the subregion. Use of integral images was popularized by the Viola-Jones algorithm [1].

`J = integralImage(I)` calculates the integral image from image `I`. The function zero-pads the top and left side of the output integral image, `J`.

`J = integralImage(I,orientation)` calculates the integral image with the orientation specified by `orientation`.

Examples

Create Integral Image

Create a simple sample matrix.

```
I = magic(5)
```

```
I = 5×5

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

Calculate the integral image of the sample matrix. These steps show how the first few values in the original matrix map to values in the integral image. Note that the pixel with (row, column) coordinate (r, c) in the original image corresponds to the pixel with coordinate (r+1, c+1) in the integral image.

- The first row and column in the integral image are all `0`s.
- The pixel in the original matrix at coordinate (1, 1) with value 17 is unchanged in the integral image because there are no other pixels in the summation. Therefore, the pixel in the integral image at coordinate (2, 2) has the value 17.

- The pixel in the original matrix at coordinate (1, 2) maps to the pixel (2, 3) in the integral image. The value is the summation of the original pixel value (24), the pixels above it (0), and the pixels to its left (17): 24 + 17 + 0 = 41.
- The pixel in the original matrix at coordinate (1, 3) maps to the pixel (2, 4) in the integral image. The value is the summation of the original pixel value (1), the pixel above it (0), and the pixels to its left (which have already been summed to 41). Thus the value at pixel (2,4) in the integral image is 1 + 41 + 0 = 42.

```
J = integralImage(I)
```

```
J = 6×6

     0     0     0     0     0     0
     0    17    41    42    50    65
     0    40    69    77    99   130
     0    44    79   100   142   195
     0    54   101   141   204   260
     0    65   130   195   260   325
```

Calculate Subregion Sum Using Integral Image

Read a grayscale image into the workspace. Display the image.

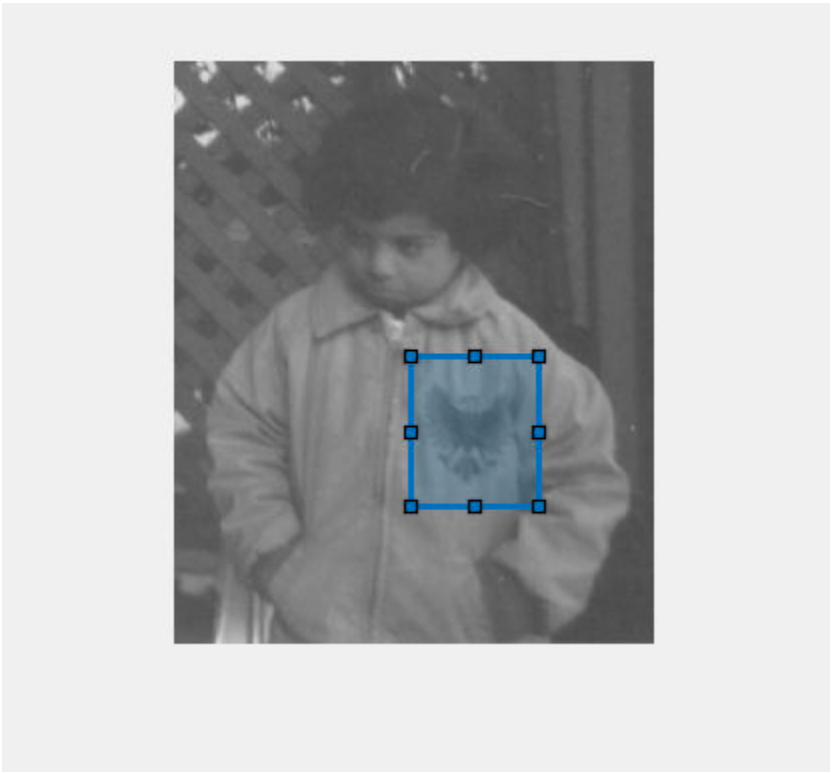
```
I = imread('pout.tif');
imshow(I)
```

Compute the integral image.

```
J = integralImage(I);
```

Use the `drawrectangle` tool to select a rectangular subregion. The tool returns a `Rectangle` object.

```
d = drawrectangle;
```



The `Vertices` property of the `Rectangle` object stores the coordinates of vertices as a 4-by-2 matrix. Vertices are ordered starting with the top-left and continuing in a clockwise direction. Split the matrix into two vectors containing the row and column coordinates. Because the integral image is zero-padded on the top and left side, increment the row and column coordinates by 1 to retrieve the corresponding elements of the integral array.

```
r = floor(d.Vertices(:,2)) + 1;
c = floor(d.Vertices(:,1)) + 1;
```

Calculate the sum of all pixels in the rectangular subregion by combining four pixels of the integral image.

```
regionSum = J(r(1),c(1)) - J(r(2),c(2)) + J(r(3),c(3)) - J(r(4),c(4))
```

```
regionSum = 613092
```

Compute Subregion Integral with Rotated Orientation

Create a simple sample matrix.

```
I = magic(5)
```

```
I = 5×5

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

Create an integral image with a rotated orientation.

```
J = integralImage(I,'rotated')
```

```
J = 6×7

     0     0     0     0     0     0     0
     0    17    24     1     8    15     0
    17    64    47    40    38    39    15
    64    74    91   104   105    76    39
    74   105   149   188   183   130    76
   105   170   232   272   236   195   130
```

Define a rotated rectangular subregion. This example specifies a subregion with top corner at coordinate (1,3) in the original image. The subregion has a rotated height of 1 and width of 2.

```
r = 1;
c = 3;
h = 1;
w = 2;
```

Get the value of the four corner pixels of the subregion in the integral image.

```
regionBottom = J(r+w+h,c-h+w+1);
regionTop = J(r,c+1);
regionLeft = J(r+h,c-h+1);
regionRight = J(r+w,c+w+1);
regionCorners = [regionBottom regionTop regionLeft regionRight]
```

```
regionCorners = 1×4

   105     0    24    39
```

Calculate the sum of pixels in the subregion by summing the four corner pixel values.

```
regionSum = regionBottom + regionTop - regionLeft - regionRight
```

```
regionSum = 42
```

Input Arguments

I — Image

numeric array

Image, specified as a numeric array of any dimension. If the input image has more than two dimensions (`ndims(I)>2`), such as for an RGB image, then `integralImage` computes the integral image for all 2-D planes along the higher dimensions.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`orientation` — Image orientation
`'upright'` (default) | `'rotated'`

Image orientation, specified as `'upright'` or `'rotated'`. If you set the orientation to `'rotated'`, then `integralImage` returns the integral image for computing sums over rectangles rotated by 45 degrees.

Data Types: `char` | `string`

Output Arguments

`J` — Integral image
numeric matrix

Integral image, returned as a numeric matrix. The function zero-pads the integral image according to the `orientation` of the image. Such sizing facilitates the computation of pixel sums along image boundaries. The integral image, `J`, is essentially a padded version of the value `cumsum(cumsum(I,2))`.

Image Orientation	Size of Integral Image
Upright integral image	Zero-padded on top and left. `size(J) = size(I)+1`
Rotated integral image	Zero-padded at the top, left, and right. `size(J) = size(I)+[1 2]`

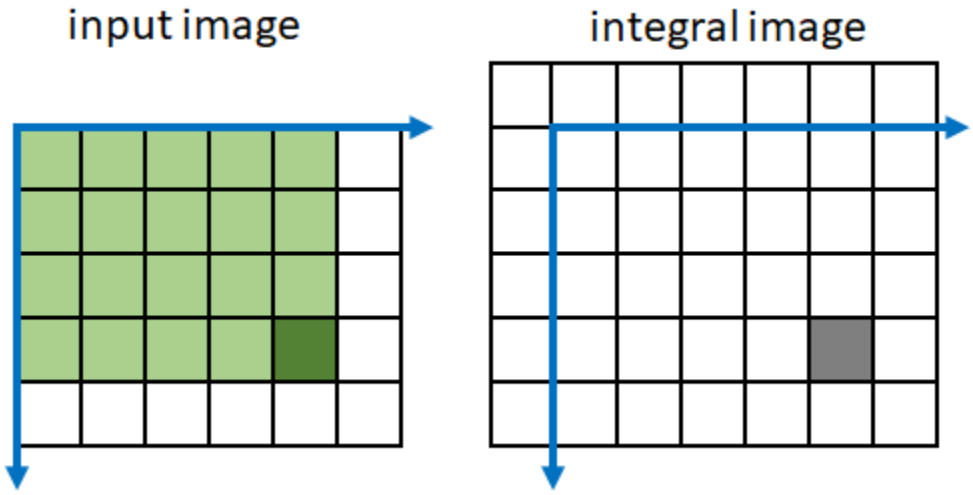
Data Types: `double`

Algorithms

Integral Image Summation

Every pixel in an integral image represents the summation of the corresponding input pixel value with all input pixels above and to the left of the input pixel. Because `integralImage` zero-pads the resulting integral image, a pixel with (row, column) coordinate (`m`, `n`) in the original image maps to the pixel with coordinate (`m`+1, `n`+1) in the integral image.

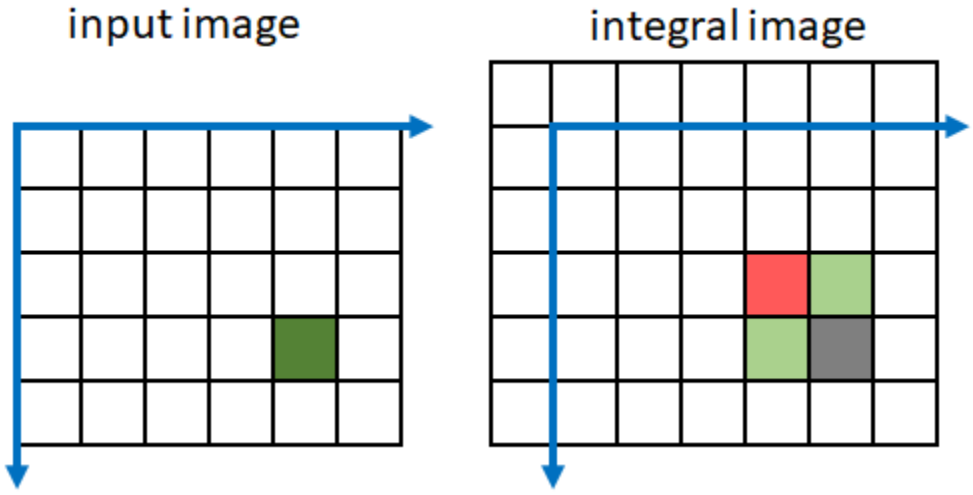
In the figure, the current pixel in the input image is the dark green pixel at coordinate (4, 5). All pixels in the input image above and to the left of the input pixel are colored in light green. The summation of the green pixel values is returned in the integral image pixel with coordinate (5, 6), colored in gray.



`integralImage` performs a faster computation of the integral image by summing pixel values in both the input image and the integral image. Pixel (`m`, `n`) in integral image `J` is a linear combination of only four pixels: one from the input image and three previously-calculated pixels from the integral image.

```
J(m,n) = J(m,n-1) + J(m-1,n) + I(m-1,n-1) - J(m-1,n-1)
```

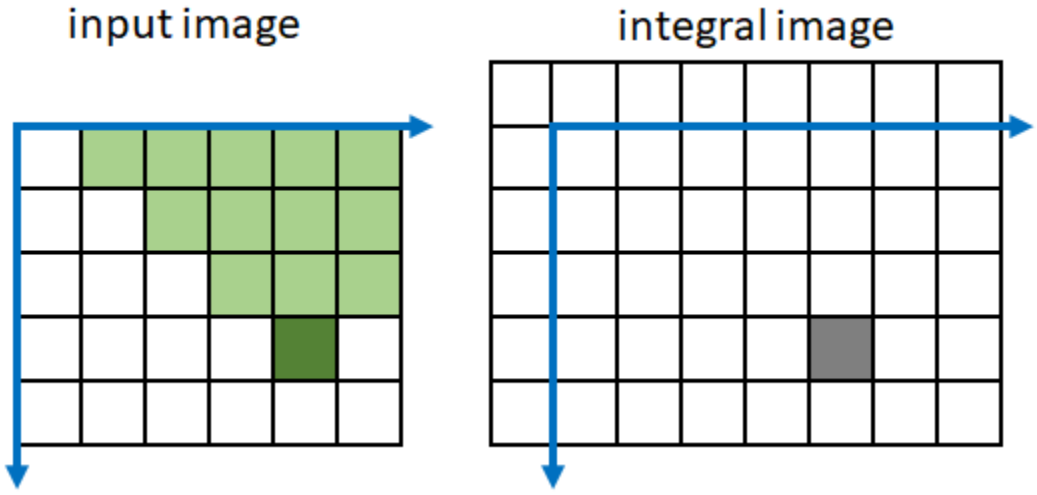
This figure shows which pixels are included in the sum when calculating the integral image at the gray pixel. Green pixels add to the sum and red pixels subtract from the sum.



Rotated Integral Image Summation

If you specify the image `orientation` as `'rotated'`, then pixels in an integral image represent the summation of a corresponding input pixel value with all input pixels that are diagonally above the input pixel. `integralImage` performs the summation along diagonal lines. This approach is less computationally intensive than rotating the image and calculating the integral image in rectilinear directions.

In the figure, the current pixel in the input image is the dark green pixel at coordinate (4, 5). All pixels in the input image diagonally above the input pixel are colored in light green. The summation of the green pixel values is returned in the integral image pixel with coordinate (5, 6), colored in gray.



`integralImage` performs a faster computation of the rotated integral image by summing pixel values in both the input image and the integral image. Pixel (`m`, `n`) in integral image `J` is a linear combination of only five pixels: two from the input image and three previously-calculated pixels from the integral image:

```
J(m,n) = J(m-1,n-1) + J(m-1,n+1) - J(m-2,n) + I(m-1,n-1) + I(m-2,n-1)
```

This figure shows which pixels are included in the sum when calculating the integral image at the gray pixel. Green pixels add to the sum and red pixels subtract from the sum.

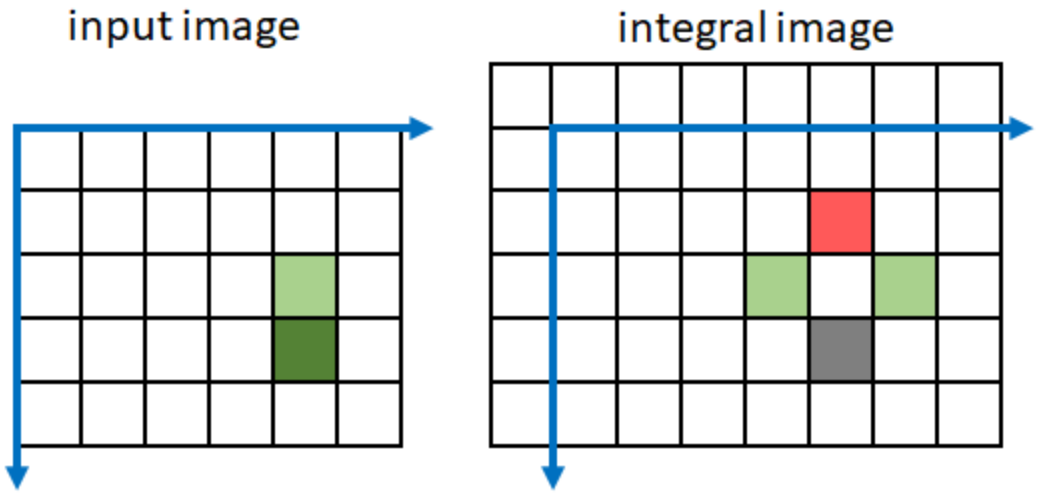
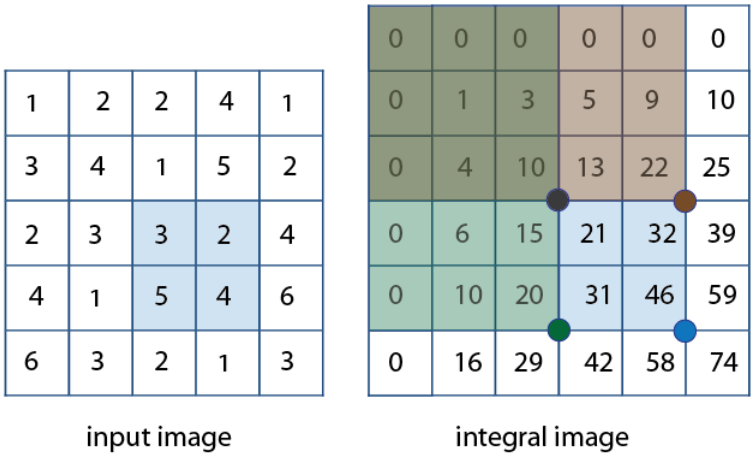


Image Subregion Summation

A subregion in an upright orientation with top-left coordinate (`m`,`n`), height `h`, and width `w` in the original image has the summation:

```
regionSum = J(m–1,n–1) + J(m+h–1,n+w–1) – J(m+h–1,n–1) – J(m-1,n+w-1)
```

For example, in the input image below, the summation of the blue shaded region is: 46 – 22 – 20 + 10 = 14. The calculation subtracts the regions above and to the left of the shaded region. The area of overlap is added back to compensate for the double subtraction.



A subregion in an rotated orientation uses a different definition of height and width [2]. The summation of the region is:

```
regionSum = J(m+h+w,n-h+w+1) + J(m,n+1) - J(m+h,n-h+1) - J(m+w,n+w+1)
```

References

[1] Viola, P., and M. J. Jones. "Rapid Object Detection using a Boosted Cascade of Simple Features". *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2001. Vol. 1, pp. 511–518.

[2] Lienhart, R., and J. Maydt. "An Extended Set of Haar-like Features for Rapid Object Detection". *Proceedings of the 2002 IEEE International Conference on Image Processing*. Sept. 2002. Vol. 1, pp. 900–903.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`integralImage` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

See Also

`cumsum` | `integralBoxFilter`

Topics

“Apply Multiple Filters to Integral Image”
“Integral Image”

Introduced in R2015b

integralImage3

Calculate 3-D integral image

Syntax

```
J = integralImage3(I)
```

Description

`J = integralImage3(I)` calculates the integral image, `J`, from grayscale volumetric image `I`.

Examples

Compute Integral Image of 3-D Input Image

Create a 3-D input image.

```
I = reshape(1:125,5,5,5);
```

Define a 3-by-3-by-3 sub-volume as `[startRow, startCol, startPlane, endRow, endCol, endPlane]`.

```
[sR, sC, sP, eR, eC, eP] = deal(2, 2, 2, 4, 4, 4);
```

Create an integral image from the input image and compute the sum over a 3-by-3-by-3 sub-volume of I.

```
J = integralImage3(I);
regionSum = J(eR+1,eC+1,eP+1) - J(eR+1,eC+1,sP) - J(eR+1,sC,eP+1) ...
        - J(sR,eC+1,eP+1) + J(sR,sC,eP+1) + J(sR,eC+1,sP) ...
        + J(eR+1,sC,sP) -J(sR,sC,sP)
```

```
regionSum = 1701
```

Verify that the sum of pixels is accurate.

```
sum(sum(sum(I(sR:eR, sC:eC, sP:eP))))
```

```
ans = 1701
```

Input Arguments

I — Grayscale volume

3-D numeric array

Grayscale volume, specified as a 3-D numeric array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output Arguments

J — Integral image

numeric array

Integral image, returned as a numeric array. The function zero-pads the top, left and along the first plane, resulting in `size(J) = size(I) + 1`. side of the integral image. The class of the output is `double`. The resulting size of the output integral image equals: `size(J) = size(I) + 1`. Such sizing facilitates easy computation of pixel sums along all image boundaries. The integral image, `J`, is essentially a padded version of the value `cumsum(cumsum(cumsum(I),2),3)`.

Data Types: `double`

More About

Integral Image

In an integral image, every pixel is the summation of the pixels above and to the left of it. Using an integral image, you can rapidly calculate summations over image subregions. Use of integral images was popularized by the Viola-Jones algorithm. Integral images facilitate summation of pixels and can be performed in constant time, regardless of the neighborhood size.

See Also

`integralBoxFilter3` | `integralImage`

Topics

“Integral Image”

Introduced in R2015b

interfileinfo

Read metadata from Interfile file

Syntax

```
info = interfileinfo(filename)
```

Description

`info = interfileinfo(filename)` returns a structure whose fields contain information about an image in the Interfile header file specified by `filename`.

The Interfile file format was developed for the exchange of nuclear medicine data. In Interfile 3.3, metadata is stored in a header file, separate from the image data. The two files have the same name with different file extensions. The header file has the file extension `.hdr` and the image file has the file extension `.img`.

Input Arguments

`filename` — Name of Interfile header file

character vector | string scalar

Name of Interfile header file, specified as a character vector or string scalar. The file must be in the current directory or in a directory on the MATLAB path.

Data Types: `char` | `string`

Output Arguments

`info` — Metadata of Interfile file

struct

Metadata of Interfile file, returned as a structure.

References

[1] Todd-Pokropek, A, Cradduck, T.D., and Deconinck, F. *A File Format for the Exchange of Nuclear Medicine Image Data: a specification of Interfile Version 3.3.* Nucl Med Commun 13(9): 673-99, 1992.

See Also

`interfileread`

Topics

“Interfile Files”

Introduced before R2006a

interfileread

Read images in Interfile format

Syntax

```
I = interfileread(filename)
I = interfileread(filename,window)
```

Description

`I = interfileread(filename)` reads the images in the first energy window of the Interfile image file specified by `filename`.

The Interfile file format was developed for the exchange of nuclear medicine data. In Interfile 3.3, metadata is stored in a header file, separate from the image data. The two files have the same name with different file extensions. The header file has the file extension `.hdr` and the image file has the file extension `.img`.

`I = interfileread(filename,window)` reads the Interfile image data in the energy window specified by `window`.

Input Arguments

`filename` — Name of Interfile image file

character vector | string scalar

Name of Interfile image file, specified as a character vector or string scalar. The file must be in the current directory or in a directory on the MATLAB path.

Data Types: `char` | `string`

`window` — Energy window

numeric scalar

Energy window, specified as a numeric scalar. The images in the energy window must have the same size.

Output Arguments

`I` — Interfile image

numeric matrix | numeric array

Interfile image, returned as a 2-D numeric matrix for a single image or a 3-D numeric array for multiple images.

References

[1] Todd-Pokropek, A, Cradduck, T.D., and Deconinck, F., *A File Format for the Exchange of Nuclear Medicine Image Data: a specification of Interfile Version 3.3.* Nucl Med Commun 13(9): 673-99, 1992.

See Also

`interfileinfo`

Topics

“Interfile Files”

Introduced before R2006a

intlut

Convert integer values using lookup table

Syntax

```
B = intlut(A,lut)
```

Description

`B = intlut(A,lut)` converts values in array `A` based on lookup table `lut` and returns these new values in array `B`.

Examples

Convert Integer Values using Lookup Table

Create an array of integers.

```
A = uint8([1 2 3 4; 5 6 7 8; 9 10 0 1])
```

```
A = 3x4 uint8 matrix

    1    2    3    4
    5    6    7    8
    9   10    0    1
```

Create a lookup table. In this example, the lookup table is created by following the vector [2 4 8 16] with repeated copies of the vector [0 150 200 250].

```
LUT = [2 4 8 16 repmat(uint8([0 150 200 255]),1,63)];
```

Convert the values of A by referring to the lookup table. Note that the first index of the lookup table is 0.

```
B = intlut(A, LUT)
```

```
B = 3x4 uint8 matrix

     4     8    16     0
   150   200   255     0
   150   200     2     4
```

Input Arguments

A — Input matrix

array of integers

Input matrix, specified as an array of integers.

Data Types: int16 | uint8 | uint16

lut — Lookup table
vector of integers

Lookup table, specified as a vector of integers.

- If A has data type uint8, then lut must be a uint8 vector with 256 elements.
- If A has data type uint16 or int16, then lut must be a vector with 65536 elements of the same class as A.

Data Types: int16 | uint8 | uint16

Output Arguments

B — Converted matrix
array of integers

Converted matrix, returned as an array of integers. B has the same size and data type as A.

Data Types: int16 | uint8 | uint16

Algorithms

- When A has data type uint8 or uint16, an offset of 1 is applied when indexing into the lookup table. For example, if an element of A has the value *alpha*, then the corresponding element in B has the value lut(*alpha*+1).
- When A has data type int16, an additional offset of 32768 is applied to the lookup table index. For example, if an element of A has the value *alpha*, then the corresponding element in B has the value lut(*alpha*+32768+1).

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- intlut supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic MATLAB Host Computer target platform, intlut generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also
ind2gray | rgb2ind

Introduced before R2006a

intrinsicToWorld

Convert from intrinsic to world coordinates

Syntax

```
[xWorld, yWorld] = intrinsicToWorld(R,xIntrinsic,yIntrinsic)
[xWorld, yWorld, zWorld] = intrinsicToWorld(R,xIntrinsic,yIntrinsic,
zIntrinsic)
```

Description

`[xWorld, yWorld] = intrinsicToWorld(R,xIntrinsic,yIntrinsic)` maps points from the 2-D intrinsic system (`xIntrinsic`,`yIntrinsic`) to the 2-D world system (`xWorld`,`yWorld`) based on the relationship defined by 2-D spatial referencing object `R`.

If the *k*th input coordinates (`xIntrinsic`(*k*),`yIntrinsic`(*k*)) fall outside the image bounds in the intrinsic coordinate system, `intrinsicToWorld` extrapolates `xWorld`(*k*) and `yWorld`(*k*) outside the image bounds in the world coordinate system.

`[xWorld, yWorld, zWorld] = intrinsicToWorld(R,xIntrinsic,yIntrinsic, zIntrinsic)` maps points from the intrinsic coordinate system to the world coordinate system using 3-D spatial referencing object `R`.

Examples

Convert 2-D Intrinsic Coordinates to World Coordinates

Read a 2-D grayscale image into the workspace.

```
m = dicominfo('knee1.dcm');
A = dicomread(m);
```

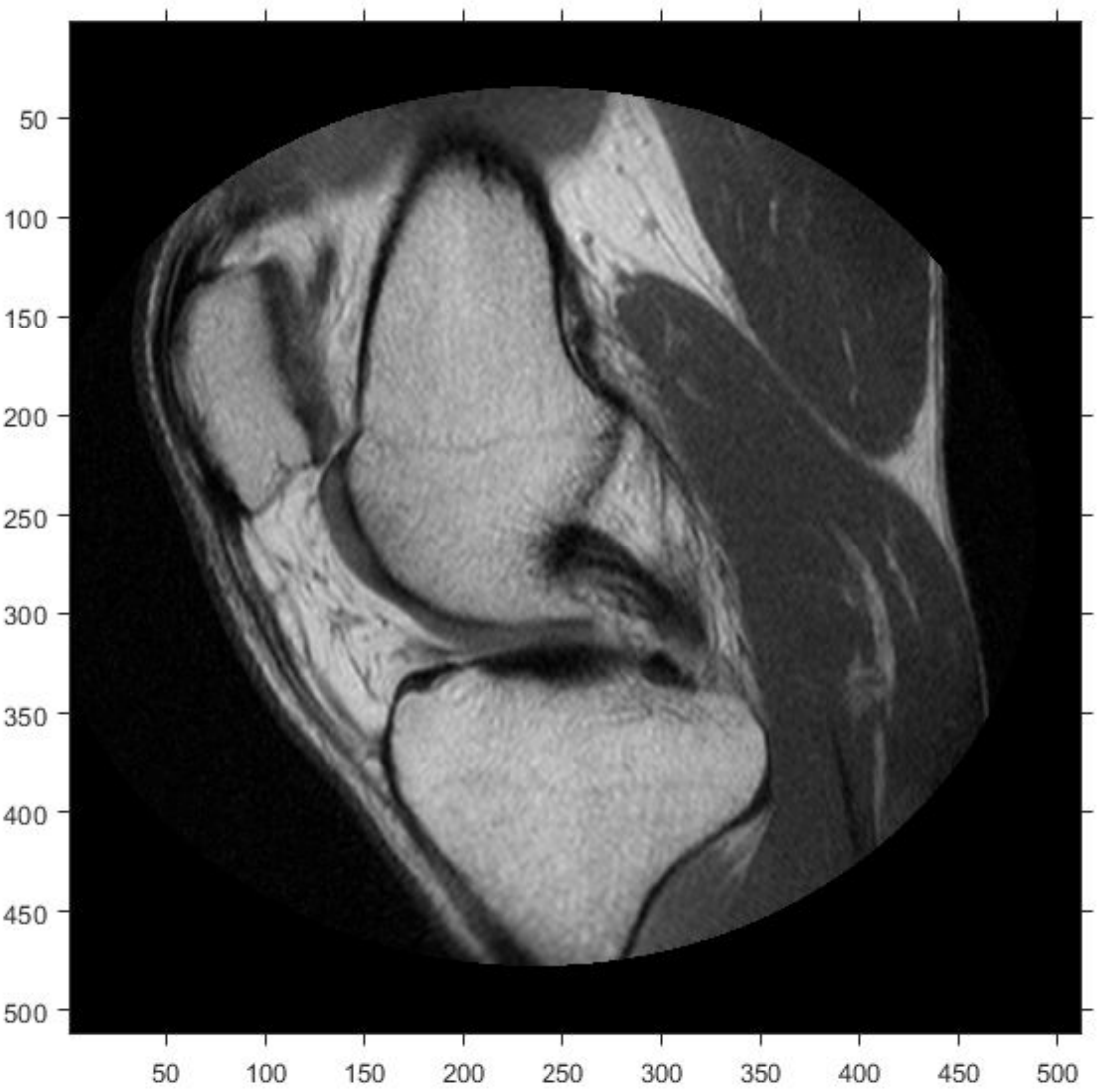
Create an `imref2d` object, specifying the size and the resolution of the pixels. The DICOM file contains a metadata field `PixelSpacing` that specifies the image resolution in each dimension in millimeters per pixel.

```
RA = imref2d(size(A),m.PixelSpacing(2),m.PixelSpacing(1))
```

```
RA =
  imref2d with properties:

           XWorldLimits: [0.1562 160.1562]
           YWorldLimits: [0.1562 160.1562]
              ImageSize: [512 512]
    PixelExtentInWorldX: 0.3125
    PixelExtentInWorldY: 0.3125
    ImageExtentInWorldX: 160
    ImageExtentInWorldY: 160
       XIntrinsicLimits: [0.5000 512.5000]
       YIntrinsicLimits: [0.5000 512.5000]
```

Display the image, omitting the spatial referencing object. The axes coordinates reflect the intrinsic coordinates. Notice that the coordinate (0,0) is in the upper left corner.

```
figure
imshow(A,'DisplayRange',[0 512])
axis on
```



Suppose you want to calculate the approximate position and width of the knee in millimeters. Select the endpoints of a line segment that runs horizontally across the knee at the level of the kneecap. For example, use the (*x*,*y*) points (34,172) and (442,172).

```
xIntrinsic = [34 442];
yIntrinsic = [172 172];
```

Convert these points from intrinsic coordinates to world coordinates.

```
[xWorld,yWorld] = intrinsicToWorld(RA,xIntrinsic,yIntrinsic)

xWorld = 1×2

   10.6250  138.1250


yWorld = 1×2

   53.7500   53.7500
```

The world coordinates of the two points are (10.625,53.75) and (138.125,53.75), in units of millimeters. The approximate width of the knee in millimeters is:

```
width = xWorld(2) - xWorld(1)

width = 127.5000
```

Convert 3-D Intrinsic Coordinates to World Coordinates

Read a 3-D volume into the workspace. This image consists of 27 frames of 128-by-128 pixel images.

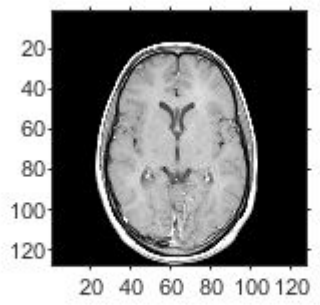
```
load mri;
D = squeeze(D);
D = ind2gray(D,map);
```

Create an `imref3d` spatial referencing object associated with the volume. For illustrative purposes, provide a pixel resolution in each dimension. The resolution is in millimeters per pixel.

```
R = imref3d(size(D),2,2,4)

R =
  imref3d with properties:

           XWorldLimits: [1 257]
           YWorldLimits: [1 257]
           ZWorldLimits: [2 110]
              ImageSize: [128 128 27]
    PixelExtentInWorldX: 2
    PixelExtentInWorldY: 2
    PixelExtentInWorldZ: 4
    ImageExtentInWorldX: 256
    ImageExtentInWorldY: 256
    ImageExtentInWorldZ: 108
       XIntrinsicLimits: [0.5000 128.5000]
       YIntrinsicLimits: [0.5000 128.5000]
       ZIntrinsicLimits: [0.5000 27.5000]
```

Display the middle slice of the volume, omitting the spatial referencing object. The axes coordinates reflect the intrinsic coordinates. Notice that the coordinate (0,0) is in the upper left corner of this plane. z=0 is right below the first slice, and the z-axis is positive in the upward direction, towards the crown of the head.

```
figure
imshow(D(:,:,13))
axis on
```



Suppose you want to determine the position, in millimeters, of features within this slice. Select four sample points, and store their intrinsic coordinates in vectors. For example, the first point has intrinsic coordinates (54,46,13). The intrinsic *z*-coordinate is the same for all points within this slice.

```
xI = [54 71 57 70];
yI = [46 48 79 80];
zI = [13 13 13 13];
```

Convert the intrinsic coordinates to world coordinates using `intrinsicToWorld`.

```
[xW,yW,zW] = intrinsicToWorld(R,xI,yI,zI)
```

```
xW = 1×4

   108   142   114   140


yW = 1×4

    92    96   158   160


zW = 1×4

    52    52    52    52
```

The resulting vectors are the world *x*-, *y*-, and *z*-coordinates, in millimeters, of the selected points. The first point, for example, is offset from the origin by 108mm in the *x*-direction, 92 mm in the *y*-direction, and 52 mm in the z-direction.

Input Arguments

R — Spatial referencing object

`imref2d` or `imref3d` object

Spatial referencing object, specified as an `imref2d` or `imref3d` object.

`xIntrinsic` — Coordinates along the *x*-dimension in the intrinsic coordinate system
numeric scalar or vector

Coordinates along the *x*-dimension in the intrinsic coordinate system, specified as a numeric scalar or vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`yIntrinsic` — Coordinates along the *y*-dimension in the intrinsic coordinate system
numeric scalar or vector

Coordinates along the *y*-dimension in the intrinsic coordinate system, specified as a numeric scalar or vector. `yIntrinsic` is the same length as `xIntrinsic`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`zIntrinsic` — Coordinates along the *z*-dimension in the intrinsic coordinate system
numeric scalar or vector

Coordinates along the *z*-dimension in the intrinsic coordinate system, specified as a numeric scalar or vector. `zIntrinsic` is the same length as `xIntrinsic`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`xWorld` — Coordinates along the *x*-dimension in the world coordinate system
numeric scalar or vector

Coordinates along the *x*-dimension in the world coordinate system, returned as a numeric scalar or vector. `xWorld` is the same length as `xIntrinsic`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`yWorld` — Coordinates along the *y*-dimension in the world coordinate system
numeric scalar or vector

Coordinates along the *y*-dimension in the world coordinate system, returned as a numeric scalar or vector. `yWorld` is the same length as `xIntrinsic`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`zWorld` — Coordinates along the *z*-dimension in the world coordinate system
numeric scalar or vector

Coordinates along the *z*-dimension in the world coordinate system, returned as a numeric scalar or vector. `zWorld` is the same length as `xIntrinsic`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

See Also
`imref2d` | `imref3d` | `worldToIntrinsic`

Introduced in R2013a

invert

Invert geometric transformation

Syntax

```
invtform = invert(tform)
```

Description

`invtform = invert(tform)` returns the inverse of the geometric transformation `tform`.

Examples

Invert 2-D Rotation

Read and display an image.

```
I = imread('pout.tif');
imshow(I)
```

Create an `affine2d` object that defines a 30 degree clockwise rotation around the origin. View the transformation matrix stored in the `T` property.

```
theta = 30;
tform = affine2d([cosd(theta) sind(theta) 0; -sind(theta) cosd(theta) 0; 0 0 1]);
tform.T
```

```
ans = 3×3

    0.8660    0.5000         0
   -0.5000    0.8660         0
         0         0    1.0000
```

Apply the forward geometric transformation, `tform`, to the image. Display the rotated image.

```
J = imwarp(I,tform);
imshow(J)
```

Invert the geometric transformation. The result is a new `affine2d` object that represents a 30 degree rotation in the counterclockwise direction.

```
invtform = invert(tform);
invtform.T
```

```
ans = 3×3

    0.8660   -0.5000         0
```

```
    0.5000    0.8660         0
         0         0    1.0000
```

Apply the inverse geometric transformation, `invtform`, to the rotated image `J`. The final image, `K`, has the same size, shape, and orientation as the original image. Black padding around the image results from the two transformations.

```
K = imwarp(J,invtform);
imshow(K)
```

Input Arguments

`tform` — Geometric transformation

`affine2d` object | `affine3d` object | `rigid2d` object | `rigid3d` object | `projective2d` object

Geometric transformation, specified as an `affine2d`, `affine3d`, `rigid2d`, `rigid3d`, or `projective2d` geometric transformation object.

Output Arguments

`invtform` — Inverse geometric transformation

geometric transformation object

Inverse geometric transformation, returned as a geometric transformation object. `invtform` is the same type of object as `tform`.

See Also

`transformPointsForward` | `transformPointsInverse`

Introduced in R2013a

iptaddcallback

Add function handle to callback list

Syntax

```
iptaddcallback(obj,callback,@fun)
ID = iptaddcallback(obj,callback,@fun)
```

Description

`iptaddcallback(obj,callback,@fun)` adds the function `fun` to the list of functions to be called when the callback of graphics object `obj` executes.

`iptaddcallback` can be useful when you need to notify more than one tool about the same callback event for a single object.

`ID = iptaddcallback(obj,callback,@fun)` also returns an identifier, `ID`, for the callback function `fun`.

Examples

Add Two Callback Functions to Figure

Create a figure and register two callback functions. Whenever MATLAB detects mouse motion over the figure, function handles `f1` and `f2` are called in the order in which they were added to the list.

```
figobj = figure;
f1 = @(varargin) disp('Callback 1');
f2 = @(varargin) disp('Callback 2');
iptaddcallback(figobj,'WindowButtonMotionFcn',f1);
iptaddcallback(figobj,'WindowButtonMotionFcn',f2);
```

Input Arguments

`obj` — Graphics object

figure | axes | uipanel | image

Graphics object, specified as a handle to a figure, axes, uipanel, or image graphics objects.

`callback` — Callback property

character vector

Callback property of the graphics object `obj`, specified as a character vector. For a list of callbacks for graphics objects, see Figure Properties, Axes Properties, Panel Properties, and Image Properties.

Data Types: `char`

`fun` — Callback function

function handle

Callback function, specified as a function handle. For more information, see “Create Function Handle”.

Data Types: `function_handle`

Output Arguments

`ID` — Callback identifier

positive integer

Callback identifier for function `fun`, returned as a positive integer.

Tips

- Callback functions that have already been added to an object using the `set` command continue to work after you call `iptaddcallback`. The first time you call `iptaddcallback` for a given object and callback, the function checks to see if a different callback function is already installed. If a callback is already installed, then `iptaddcallback` replaces that callback function with the `iptaddcallback` callback processor, and then adds the preexisting callback function to the `iptaddcallback` list.

See Also

`iptremovecallback`

Topics

“Callbacks — Programmed Response to User Action”
“Overview Events and Listeners”

Introduced before R2006a

iptcheckconn

Check validity of connectivity argument

Syntax

```
iptcheckconn(conn,func_name,var_name,arg_pos)
```

Description

`iptcheckconn(conn,func_name,var_name,arg_pos)` checks if `conn` is a valid pixel connectivity and issues a formatted error message if the connectivity is invalid.

- If the connectivity is valid, then `iptcheckconn` returns nothing. Valid connectivities are one of these scalar values: 1, 4, 6, 8, 18, or 26. A connectivity can also be a 3-by-3-by- ... -by-3 array of `0`s and `1`s. The central element of a connectivity array must be nonzero and the array must be symmetric about its center.
- If the connectivity is invalid, then `iptcheckconn` issues a formatted error message that includes information about the function name (`func_name`), the variable name (`var_name`), and the argument position (`arg_pos`). These values are used only to create the error message, not to check whether the pixel connectivity is valid.

Examples

Check Validity of 4-by-4 Matrix

Create a 4-by-4 array and pass it as the connectivity argument.

```
iptcheckconn(eye(4),'myfun','myvar',2)
```

`eye(4)` is not a valid pixel connectivity so `iptcheckconn` returns an error message:

```
Function MYFUN expected input number 2, myvar, to be a valid connectivity specifier. A nonscalar
connectivity specifier must be 3-by-3-by- ... -by-3.
```

Input Arguments

`conn` — Pixel connectivity

numeric scalar | numeric array

Pixel connectivity to check, specified as a numeric scalar or array.

Data Types: `double` | `logical`

`func_name` — Function name

character vector | string scalar

Function name to include in an error message when `conn` is an invalid pixel connectivity, specified as a character vector or string scalar.

Data Types: `char` | `string`

`var_name` — Variable name
character vector | string scalar

Variable name to include in an error message when `conn` is an invalid pixel connectivity, specified as a character vector or string scalar.

Data Types: `char` | `string`

`arg_pos` — Argument position
positive integer

Argument position to include in an error message when `conn` is an invalid pixel connectivity, specified as a numeric scalar.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `iptcheckconn` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- When generating code, all input arguments must be compile-time constants.

See Also
`conndef`

Topics
“Pixel Connectivity”

Introduced before R2006a

iptcheckhandle

Check validity of handle

Syntax

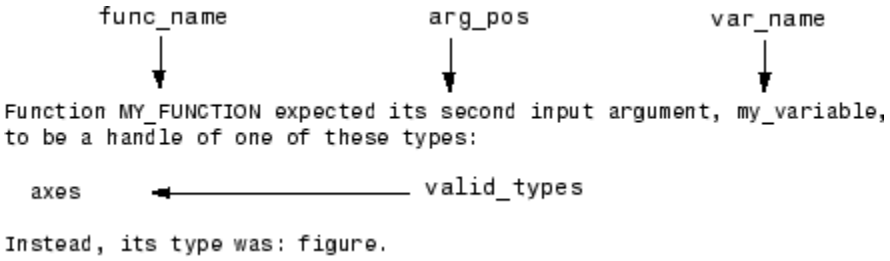
```
iptcheckhandle(obj,valid_types,func_name,var_name,arg_pos)
```

Description

`iptcheckhandle(obj,valid_types,func_name,var_name,arg_pos)` checks if object `obj` is a valid graphics object and issues a formatted error message if the handle is invalid.

- If the object is a valid graphics object as specified by `valid_types`, then `iptcheckhandle` returns nothing.
- If the connectivity argument is invalid, then `iptcheckhandle` issues a formatted error message that includes information about the function name (`func_name`), the variable name (`var_name`), and the argument position (`arg_pos`). These values are used only to create the error message, not to check whether the graphics object handle is valid.

The figure shows the format of the error message and indicates which parts you can customize using `iptcheckhandle` arguments.



Examples

Trigger Error When Graphics Object Is Not Axes

To trigger the error message, create a figure that does not contain an axes object and then check for a valid axes handle.

```
fig = figure; % create figure without an axes
iptcheckhandle(fig,{'axes'},'my_function','my_variable',2)
```

`fig` is not an axes handle so `iptcheckhandle` returns an error message:

```
Function MY_FUNCTION expected its second input argument, my_variable,
to be one of these types:

 axes

Instead, its type was: figure.
```

Input Arguments

`obj` — Object
handle

Object, specified as a handle.

`valid_types` — Valid types of graphics objects
cell array of character vectors

Valid types of graphics objects, specified as a cell array containing one or more of these character vectors.

- `'axes'`
- `'figure'`
- `'hggroup'`
- `'image'`
- `'uipanel'`

Example: `{'uipanel','figure'}` specifies that a valid graphics object must be either a panel container or a figure.

`func_name` — Function name
character vector | string scalar

Function name to include in an error message when `obj` is an invalid graphics object, specified as a character vector or string scalar.

Data Types: `char` | `string`

`var_name` — Variable name
character vector | string scalar

Variable name to include in an error message when `obj` is an invalid graphics object, specified as a character vector or string scalar.

Data Types: `char` | `string`

`arg_pos` — Argument position
positive integer

Argument position to include in an error message when `obj` is an invalid graphics object, specified as a positive integer.

See Also
`validateattributes` | `iptcheckmap` | `narginchk` | `validatestring` | `iptnum2ordinal`

Introduced before R2006a

iptcheckinput

Check validity of array

Note `iptcheckinput` will be removed in a future release. Use `validateattributes` instead.

Syntax

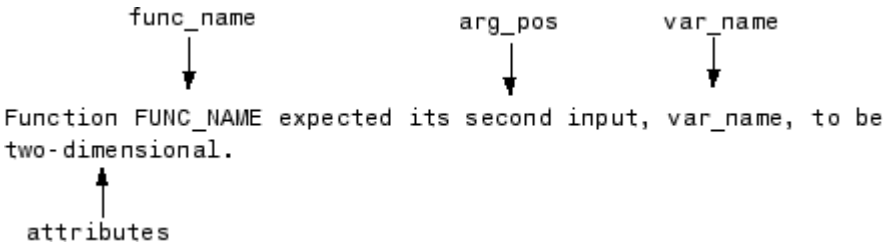
```
iptcheckinput(A,valid_classes,valid_attributes, func_name,var_name,arg_pos)
```

Description

`iptcheckinput(A,valid_classes,valid_attributes, func_name,var_name,arg_pos)` checks the validity of the input array `A` and issues a formatted error message if the array is invalid.

- If the array has valid class and attributes as specified by `valid_classes` and `valid_attributes`, then `iptcheckinput` returns nothing.
- If the class or attributes are invalid, then `iptcheckinput` issues a formatted error message that includes information about the function name (`func_name`), the variable name (`var_name`), and the argument position (`arg_pos`). These values are used only to create the error message, not to check whether the array is valid.

The figure shows the format of the error message and indicates which parts you can customize using `iptcheckinput` arguments.



Examples

Trigger Error When Array Is Not 2-D

To trigger this error message, create a 3-D array and then check for the attribute `'2d'`.

```
A = [ 1 2 3; 4 5 6 ];
B = [ 7 8 9; 10 11 12];
C = cat(3,A,B);
iptcheckinput(C,{'numeric'},{'2d'},'func_name','var_name',2)
```

`C` is not 2-D so `iptcheckinput` returns an error message:

```
Function FUNC_NAME expected its second input, var_name, to be two-dimensional.
```

Input Arguments

A — Input array

array | ...

Input array, specified as an array.

valid_classes — Valid classes

cell array of character vectors

Valid classes of array A, specified as a cell array of character vectors. The tables list common classes for image processing applications.

Numeric Classes

int8	uint8	single
int16	uint16	double
int32	uint32	
int64	uint64	

Other Common Classes

categorical	char	cell
function_handle	logical	string
struct	table	

You can use 'numeric' as an abbreviation for the set of classes uint8, uint16, uint32, int8, int16, int32, single, and double.

Example: {'logical' 'cell'} specifies that a valid array must be a logical array or a cell array.

valid_attributes — Valid attributes

{} | cell array of character vectors

Valid attributes of array A, specified as a cell array of character vectors. The table lists the supported attributes in alphabetical order.

2d	column	even	finite
integer	nonempty	nonnan	nonnegative
nonsparse	nonzero	odd	positive
real	row	scalar	twod
vector			

If you specify valid_attributes as the empty cell array {}, then iptcheckinput does not check the attributes of A.

Example: {'real' 'nonempty' 'finite'} specifies that a valid array must be real and nonempty and contain only finite values.

func_name — Function name

character vector | string scalar

Function name to include in an error message when `A` is an invalid array, specified as a character vector or string scalar.

Data Types: `char` | `string`

`var_name` — Variable name
character vector | string scalar

Variable name to include in an error message when `A` is an invalid array, specified as a character vector or string scalar.

Data Types: `char` | `string`

`arg_pos` — Argument position
positive integer

Argument position to include in an error message when `A` is an invalid array, specified as a positive integer.

See Also

`validateattributes` | `iptcheckhandle` | `iptcheckmap` | `narginchk` | `validatestring` | `iptnum2ordinal`

Introduced before R2006a

iptcheckmap

Check validity of colormap

Syntax

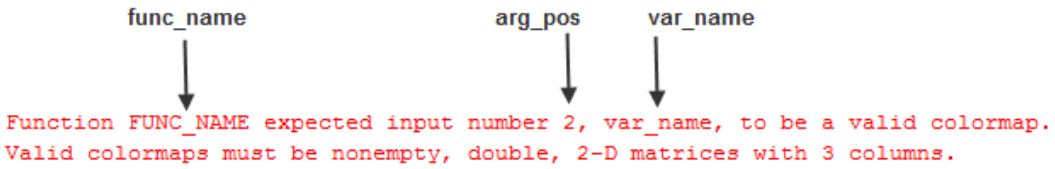
```
iptcheckmap(map,func_name,var_name,arg_pos)
```

Description

`iptcheckmap(map,func_name,var_name,arg_pos)` checks the validity of the MATLAB colormap `map` and issues a formatted error message if the colormap is invalid.

- If the colormap is valid, then `iptcheckmap` returns nothing.
- If the colormap is invalid, then `iptcheckmap` issues a formatted error message that includes information about the function name (`func_name`), the variable name (`var_name`), and the argument position (`arg_pos`). These values are used only to create the error message, not to check whether the array is valid.

The figure shows the format of the error message and indicates which parts you can customize using `iptcheckmap` arguments.



Examples

Trigger Error For Invalid Colormap

```
bad_map = ones(10);
iptcheckmap(bad_map,'func_name','var_name',2)
```

```
Function FUNC_NAME expected input number 2, var_name, to be a valid colormap. Valid
colormaps must be nonempty, double, 2-D matrices with 3 columns.
```

Input Arguments

`map` — Colormap

numeric array

Colormap, specified as a numeric array.

`func_name` — Function name

character vector | string scalar

Function name to include in an error message when `map` is an invalid colormap, specified as a character vector or string scalar.

Data Types: `char` | `string`

`var_name` — Variable name
character vector | string scalar

Variable name to include in an error message when `map` is an invalid colormap, specified as a character vector or string scalar.

Data Types: `char` | `string`

`arg_pos` — Argument position
positive integer

Argument position to include in an error message when `map` is an invalid colormap, specified as a positive integer.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

`iptcheckmap` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

See Also

`iptcheckhandle` | `validateattributes` | `narginchk` | `validatestring` | `iptnum2ordinal`

Introduced before R2006a

iptchecknargin

Check number of input arguments

Note `iptchecknargin` will be removed in a future release. Use `narginchk` instead.

Syntax

```
iptchecknargin(low,high,num_inputs,func_name)
```

Description

`iptchecknargin(low,high,num_inputs,func_name)` checks whether `num_inputs` is a valid number of input arguments and issues a formatted error message if the number of input arguments is invalid.

- If the number of input arguments is in the range [`low` `high`], then `iptchecknargin` returns nothing.
- If the number of input argument is less than `low` or greater than `high`, then `iptchecknargin` issues a formatted error message that includes information about the function name (`func_name`). This value is used only to create the error message, not to check whether the number of input arguments is valid.

Examples

Trigger Error For Invalid Number of Input Arguments

Create a function called `test_function` that accepts any number of input arguments. Within the function, call `iptchecknargin` to check that the number of arguments passed to the function is within the range [1, 3]. Save the function with the file name `test_function.m`.

```
function test_function(varargin)
    iptchecknargin(1,3,nargin,mfilename);
end
```

Trigger the error message by executing the function at the MATLAB command line, specifying more than the expected number of arguments.

```
test_function(eye(3),5,pi,7)
```

Input Arguments

`low` — Smallest valid number of input arguments

nonnegative integer

Smallest valid number of input arguments, specified as a nonnegative integer.

`high` — Largest valid number of input arguments

nonnegative integer | `Inf`

Largest valid number of input arguments, specified as a nonnegative integer or `Inf`.

`num_inputs` — Number of actual input arguments
nonnegative integer | `nargin`

Number of actual input arguments, specified as a nonnegative integer. You can also specify `nargin` to use the number of function input arguments to the currently executing function.

`func_name` — Function name
character vector | string scalar | `mfilename`

Function name to include in an error message when `num_inputs` is an invalid number of input arguments, specified as a character vector or string scalar. You can also specify `mfilename` to use the name of the currently executing function.

Data Types: `char` | `string`

See Also

`narginchk` | `iptcheckhandle` | `validateattributes` | `iptcheckmap` | `validatestring` | `iptnum2ordinal`

Introduced before R2006a

iptcheckstrs

Check validity of option

Note `iptcheckstrs` will be removed in a future release. Use `validatestring` instead.

Syntax

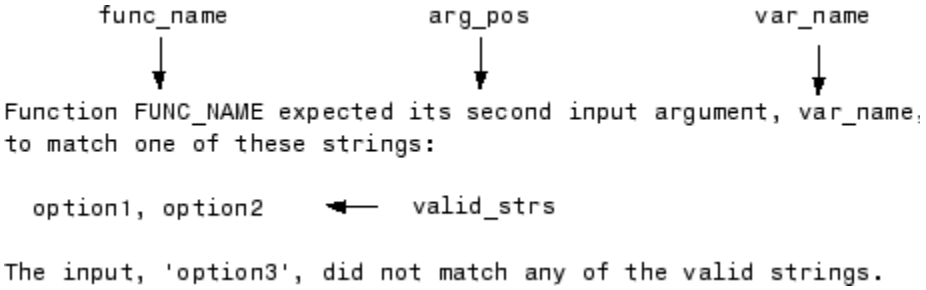
```
param = iptcheckstrs(str,valid_strs,func_name,var_name,arg_pos)
```

Description

`param = iptcheckstrs(str,valid_strs,func_name,var_name,arg_pos)` checks whether `str` is a valid parameter name and issues a formatted error message if the parameter name is invalid.

- If there is a case-insensitive, nonambiguous match between `str` and a valid parameter name in `valid_strs`, then `iptcheckstrs` returns the valid parameter name in `param`.
- If there is no match or the match is ambiguous, then `iptcheckstrs` issues a formatted error message that includes information about the function name (`func_name`), the variable name (`var_name`), and the argument position (`arg_pos`). These values are used only to create the error message, not to check whether the parameter is valid.

The figure shows the format of the error message and indicates which parts you can customize using `iptcheckstrs` arguments.



Examples

Trigger Error For Invalid Parameter Name

Define a cell array of character vectors that contains valid parameter names. To trigger an error message, pass in a character vector that is not in the cell array.

```
valid_params = {'option1','option2'};
iptcheckstrs('option3',valid_params,'func_name','var_name',2)
```

```
Function FUNC_NAME expected its second input argument, var_name,
to match one of these:  option1, option2

The input, 'option3', did not match any of the valid strings.
```

Return Valid Parameter Name

Define a cell array of character vectors that contains valid parameter names. Check the validity of a parameter name that differs only by case from a character vector in the cell array.

```
valid_params = {'option1','option2'};
iptcheckstrs('OPTION2',valid_params,'func_name','var_name',2)
```

```
param =

    'option2'
```

Input Arguments

`str` — Parameter name

character vector

Parameter name to check, specified as a character vector.

`valid_strs` — Valid parameter names

cell array of character vectors

Valid parameter names, specified as a cell array of character vectors.

`func_name` — Function name

character vector | string scalar

Function name to include in an error message when `h` is an invalid graphics object handle, specified as a character vector or string scalar.

Data Types: `char` | `string`

`var_name` — Variable name

character vector | string scalar

Variable name to include in an error message when `h` is an invalid graphics object handle, specified as a character vector or string scalar.

Data Types: `char` | `string`

`arg_pos` — Argument position

positive integer

Argument position to include in an error message when `h` is an invalid graphics object handle, specified as a positive integer.

Output Arguments

`param` — Validated parameter name

character vector

Validated parameter name, returned as a character vector.

See Also

validatestring | iptcheckhandle | validateattributes | iptcheckmap | narginchk | iptnum2ordinal

Introduced before R2006a

iptdemos

Index of Image Processing Toolbox examples

Syntax

```
iptdemos
```

Description

`iptdemos` displays the HTML page that lists all the Image Processing Toolbox examples. `iptdemos` displays the page in the MATLAB Help browser.

See Also

`ipticondir`

Introduced before R2006a

iptgetapi

Get Application Programmer Interface (API) for handle

Syntax

```
API = iptgetapi(h)
```

Description

`API = iptgetapi(h)` returns the API structure of an interactive modular tool with handle `h`.

Examples

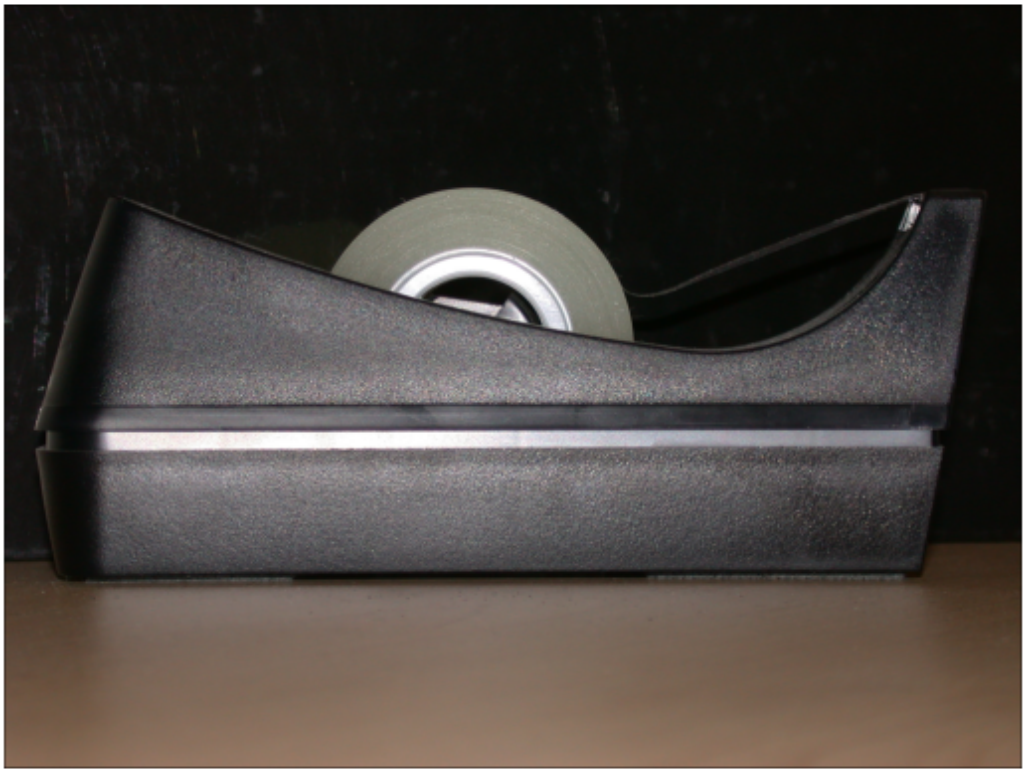
Use `imscrollpanel` API to Adjust Image Magnification

Display an image in a figure window.

```
hFig = figure('Toolbar','none','Menubar','none');
hIm = imshow('tape.png');
```

Add a Scroll Panel tool to the figure.

```
hSP = imscrollpanel(hFig,hIm);
```



Get the API associated with the Scroll Panel tool.

```
api = iptgetapi(hSP);
```

Use the API to magnify the image by 200%.

```
api.setMagnification(2)
```

Input Arguments

h — Handle to interactive modular tool

handle

Handle to an interactive modular tool such as an `imdistline`, `imline`, `immagbox`, or `imscrollpanel`.

Output Arguments

API — Handle API

struct | [ ]

Handle API, returned as a struct whose fields are the function handles that belong to the interactive modular tool `h`. If `h` is not a handle to an interactive modular tool, then `API` is returned as an empty array, `[]`.

See Also

`imline` | `immagbox` | `imdistline` | `imscrollpanel`

Introduced before R2006a

iptGetPointerBehavior

Retrieve pointer behavior from graphics object

Syntax

```
pointerBehavior = iptGetPointerBehavior(obj)
```

Description

`pointerBehavior = iptGetPointerBehavior(obj)` returns the pointer behavior structure associated with the graphics object `obj`. A pointer behavior structure contains function handles that interact with a figure's pointer manager (see `iptPointerManager`) to control what happens when the figure's mouse pointer moves over and then exits the object.

Input Arguments

`obj` — Graphics object

figure | axes | uipanel | image

Graphics object, specified as a handle to a figure, axes, uipanel, or image graphics objects.

Output Arguments

`pointerBehavior` — Pointer behavior

struct | `[]`

Pointer behavior, returned as a structure with three fields.

Field	When Called
`enterFcn`	Called when the mouse pointer moves over the object.
`traverseFcn`	Called once when the mouse pointer moves over the object, and called again each time the mouse moves within the object.
`exitFcn`	Called when the mouse pointer leaves the object.

If `obj` does not contain a pointer behavior structure, then `iptGetPointerBehavior` returns `[]`.

See Also

`iptPointerManager` | `iptSetPointerBehavior`

Introduced in R2006a

iptgetpref

Get values of Image Processing Toolbox preferences

Syntax

```
prefs = iptgetpref
value = iptgetpref(prefname)
```

Description

`prefs = iptgetpref` returns a structure containing all of the Image Processing Toolbox preferences with their current values.

You can also use the Image Processing Toolbox Preferences dialog box to get the preferences. To access the dialog box, click **Preferences** on the **Home** tab in the MATLAB desktop, or call the `iptprefs` function.

`value = iptgetpref(prefname)` returns the value of the Image Processing Toolbox preference specified by `prefname`.

Examples

Get Value of Single Image Processing Toolbox Preference

Get the value of the `'ImshowAxesVisible'` preference.

```
value = iptgetpref('ImshowAxesVisible')
```

```
value =

off
```

Input Arguments

`prefname` — Name of an Image Processing Toolbox preference

character vector | string scalar

Name of an Image Processing Toolbox preference, specified as one of the following.

- `'ImshowBorder'`
- `'ImshowAxesVisible'`
- `'ImshowInitialMagnification'`
- `'ImtoolStartWithOverview'`
- `'ImtoolInitialMagnification'`
- `'UseIPPL'`
- `'VolumeViewerUseHardware'`

Data Types: `char` | `string`

Output Arguments

`prefs` — Value of all Image Processing Toolbox preferences

structure

Value of all Image Processing Toolbox preferences, returned as a structure. Each field in the structure has the name of an Image Processing Toolbox preference.

Data Types: `struct`

`value` — Value of single Image Processing Toolbox preference

character vector | numeric scalar | logical scalar

Value of the Image Processing Toolbox preference `prefname`, returned as a character vector, numeric scalar, or logical scalar.

Data Types: `char` | `double` | `logical`

Tips

- You can also use the Image Processing Toolbox Preferences dialog box to get the preferences. To access the dialog box, click **Preferences** on the **Home** tab in the MATLAB desktop, or call the `iptprefs` function.

See Also

`imshow` | `iptprefs` | `iptsetpref`

Introduced before R2006a

ipticondir

Directories containing Image Processing Toolbox and MATLAB icons

Syntax

```
[DirI,DirM] = ipticondir
```

Description

`[DirI,DirM] = ipticondir` returns the names of the directories containing Image Processing Toolbox icons and MATLAB icons.

Examples

List Icons in Image Processing Toolbox

Get the directories containing the Image Processing Toolbox and MATLAB icons.

```
[iptdir,MATLABdir] = ipticondir
```

List the contents of the directory containing Image Processing Toolbox icons.

```
dir(iptdir)
```

Output Arguments

`DirI` — Directory containing Image Processing Toolbox icons

character vector

Directory containing Image Processing Toolbox icons, returned as a character vector.

Data Types: `char`

`DirM` — Directory containing MATLAB icons

character vector

Directory containing MATLAB icons, returned as a character vector.

Data Types: `char`

See Also

Image Viewer

Introduced before R2006a

iptnum2ordinal

Convert positive integer to ordinal character vector

Syntax

```
ordstr = iptnum2ordinal(number)
```

Description

`ordstr = iptnum2ordinal(number)` converts the positive integer `number` to the ordinal character vector `ordstr`.

Examples

Convert Integers to Ordinal Numbers

Convert the number 4 to an ordinal number. The ordinal number is spelled out in entirety.

```
str = iptnum2ordinal(4)
```

```
str =

    'fourth'
```

Convert the number 23 to an ordinal number. The ordinal number consists of a numeral and the ordinal suffix `'rd'`.

```
str = iptnum2ordinal(23)
```

```
str =

    '23rd'
```

Input Arguments

`number` — Positive integer

numeric scalar

Positive integer, specified as a numeric scalar.

Output Arguments

`number` — Ordinal number

character vector

Ordinal number, returned as a character vector.

- Numbers less than or equal to twenty are spelled out.
- Numbers greater than twenty consist of a numeral and an ordinal suffix: `'st'` (for "first"), `'nd'` (for "second"), `'rd'` (for "third"), or `'th'`.

Data Types: `char`

Introduced before R2006a

iptPointerManager

Create pointer manager in figure

Syntax

```
iptPointerManager(hFigure)
iptPointerManager(hFigure,'disable')
iptPointerManager(hFigure,'enable')
```

Description

`iptPointerManager(hFigure)` creates a pointer manager in the specified figure, `hFigure`. If the figure contains a pointer behavior structure on page 1-2154, then the pointer manager controls the pointer behavior for graphics objects in the figure.

Use `iptSetPointerBehavior` to associate a pointer behavior structure with a particular object and to define specific actions that occur when the mouse pointer moves over and then leaves the object.

`iptPointerManager(hFigure,'disable')` disables the figure's pointer manager.

`iptPointerManager(hFigure,'enable')` enables and updates the figure's pointer manager.

Examples

Create Pointer Manager in Figure with Line Object

Plot a line. Create a pointer manager in the figure. Then, associate a pointer behavior structure with the line object in the figure that changes the mouse pointer into a fleur whenever the pointer is over it.

```
h = plot(1:10);
iptPointerManager(gcf);
enterFcn = @(hFigure,currentPoint)set(hFigure,'Pointer','fleur');
iptSetPointerBehavior(h,enterFcn);
```

Input Arguments

`hFigure` — Figure

figure

Figure, specified as a `figure` object.

More About

Pointer Behavior Structure

A pointer behavior structure has three fields that specify the behavior of the pointer when the mouse moves over and then exits an object in the figure.

To define the specific actions of the pointer, set the value of these fields to function handles. If you set a field to `[]`, then no action is taken. When the pointer manager calls the function handles, it passes two arguments: the figure object and the current position of the pointer.

Field	When Called
`enterFcn`	Called when the mouse pointer moves over the object.
`traverseFcn`	Called once when the mouse pointer moves over the object, and called again each time the mouse moves within the object.
`exitFcn`	Called when the mouse pointer leaves the object.

Tips

- If a figure already contains a pointer manager, then `iptPointerManager(hFigure)` does not create a new pointer manager. The syntax has the same behavior as `iptPointerManager(hFigure,'enable')`.
- `iptPointerManager` considers not just the object the pointer is over, but all objects in the figure. `iptPointerManager` searches the graphics objects hierarchy to find the first object that contains a pointer behavior structure. The `iptPointerManager` then executes that object's pointer behavior function. For more information, see “Graphics Object Hierarchy”.

  For example, you could set the pointer to be a fleur and associate that pointer with the axes. Then, when you slide the pointer into the figure window, it will initially be the default pointer, then change to a fleur when you cross into the axes, and remain a fleur when you slide over the objects parented to the axes.
- If you specify a pointer behavior using `iptSetPointerBehavior` and then change the figure pointer without using `iptSetPointerBehavior`, then the `iptPointerManager` may not update to reflect the new behavior. Some ways to change the figure pointer without using `iptSetPointerBehavior` include using ROI objects such as `Polygon`, another graphics object, another custom UI, or code that modifies the pointer from within a callback.

See Also

`iptGetPointerBehavior` | `iptSetPointerBehavior`

Topics

“Graphics Object Hierarchy”

Introduced in R2006a

iptprefs

Display Image Processing Toolbox Preferences dialog box

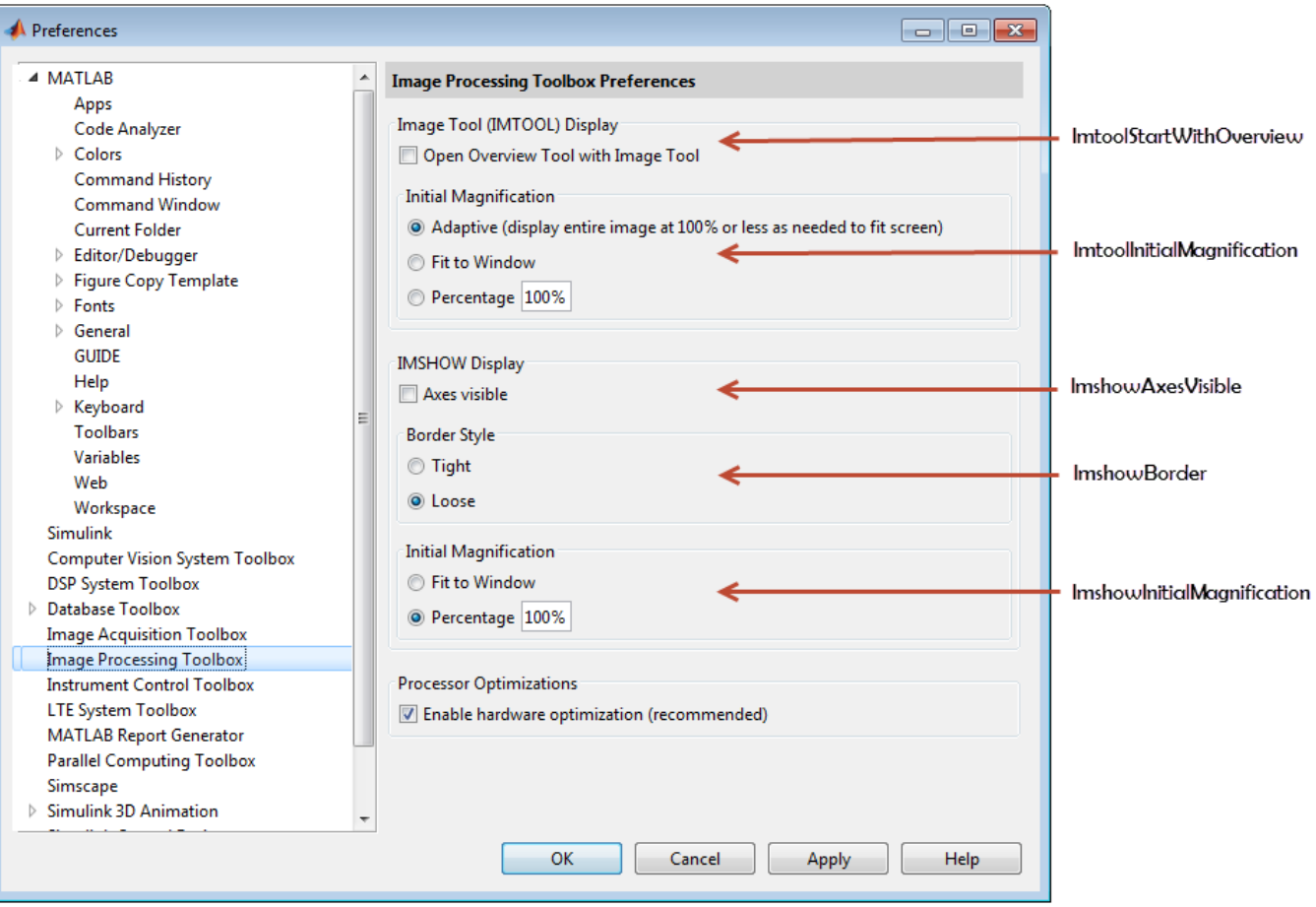
Syntax

```
iptprefs
```

Description

`iptprefs` opens the Image Processing Toolbox Preferences dialog box, part of the MATLAB Preferences dialog box. You can also open this dialog box by clicking **Preferences** on the Home tab, in the Environment section.

The Image Processing Toolbox Preferences dialog box contains display preferences for the `imshow` function, **Image Viewer** app, and **Volume Viewer** app, and provides an option for enabling hardware optimizations. For a list of all supported preferences with information about how to set them at the command line, see `iptsetpref`. The figure shows how the preferences relate to options in the Preferences dialog box.



mage Processing Toolbox Preferences Dialog Box

See Also

Image Viewer | **Volume Viewer** | `imshow` | `iptgetpref` | `iptsetpref`

Introduced in R2009a

iptremovecallback

Delete function handle from callback list

Syntax

```
iptremovecallback(obj,callback,ID)
```

Description

`iptremovecallback(obj,callback,ID)` deletes the callback with identifier `ID` from the list of callbacks for graphics object `obj`.

Examples

Add and Remove Callbacks from Figure

Add three callbacks to a figure and try them interactively. Whenever MATLAB detects mouse motion over the figure, functions `f1`, `f2`, and `f3` are called in that order.

```
h = figure;
f1 = @(varargin) disp('Callback 1');
f2 = @(varargin) disp('Callback 2');
f3 = @(varargin) disp('Callback 3');
id1 = iptaddcallback(h, 'WindowButtonMotionFcn', f1);
id2 = iptaddcallback(h, 'WindowButtonMotionFcn', f2);
id3 = iptaddcallback(h, 'WindowButtonMotionFcn', f3);
```

Remove the callback `f2`. Move the mouse over the figure again. Whenever MATLAB detects mouse motion over the figure, only functions `f1` and `f3` are called.

```
iptremovecallback(h,'WindowButtonMotionFcn',id2);
```

Input Arguments

`obj` — Graphics object

figure | axes | uipanel | image

Graphics object, specified as a handle to a figure, axes, uipanel, or image graphics objects.

`callback` — Callback property

character vector

Callback property of the graphics object `obj`, specified as a character vector. For a list of callbacks for graphics objects, see Figure Properties, Axes Properties, Panel Properties, and Image Properties.

Data Types: `char`

`ID` — Callback identifier

positive integer

Callback identifier for function `fun`, specified as a positive integer. This identifier is returned by `iptaddcallback` when you add a function to the callback list.

See Also

`iptaddcallback`

Introduced before R2006a

iptSetPointerBehavior

Store pointer behavior structure in graphics object

Syntax

```
iptSetPointerBehavior(obj,pointerBehavior)
iptSetPointerBehavior(obj,[])
iptSetPointerBehavior(obj,fun)
```

Description

`iptSetPointerBehavior(obj,pointerBehavior)` stores the specified pointer behavior structure in the specified graphics object, `obj`. If `obj` is an array of objects, then `iptSetPointerBehavior` stores the same structure in each object.

If the figure has a pointer manager installed, then the pointer manager calls these functions when the mouse moves over and then exits an object in the figure. See `iptPointerManager`.

`iptSetPointerBehavior(obj,[])` clears the pointer behavior from the graphics object or objects.

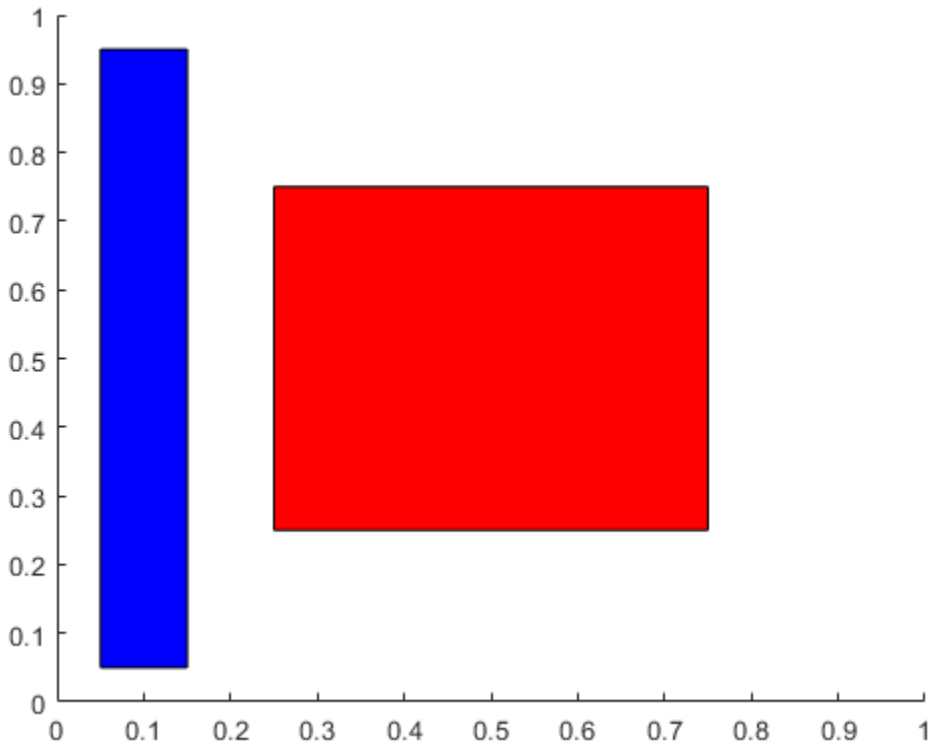
`iptSetPointerBehavior(obj,fun)` creates a pointer behavior structure, setting the `enterFcn` field to the specified function `fun`, and setting the `traverseFcn` and `exitFcn` fields to `[]`. This syntax is provided as a convenience because, for many common uses, only the `enterFcn` field is necessary.

Examples

Change Figure Properties When Pointer Is Over Graphics Objects

Show a figure with two rectangular patch graphics objects.

```
patchobj1 = patch([.25 .75 .75 .25 .25],...
               [.25 .25 .75 .75 .25], 'r');
patchobj2 = patch([.05 .15 .15 .05 .05],...
               [.05 .05 .95 .95 .05], 'b');
xlim([0 1])
ylim([0 1])
```



Specify the pointer behavior by creating a structure with three fields, `enterFcn`, `exitFcn`, and `traverseFcn`.

Whenever the pointer crosses over a specified object, change the mouse pointer to a fleur and change the title of the figure. Specify this behavior using the `enterFcn` field.

```
pb.enterFcn = @(fig,currentPoint) set(fig, ...
    'Name','Over Patch', ...
    'Pointer','fleur');
```

When the pointer moves off the object, restore the original pointer and the figure title. Specify this behavior using the `exitFcn` field.

```
pb.exitFcn = @(fig,currentPoint) set(fig, ...
    'Name','', ...
    'Pointer','arrow');
```

Do not change the figure as the pointer traverses the object. Set the `traverseFcn` field as `[]`.

```
pb.traverseFcn = [];
```

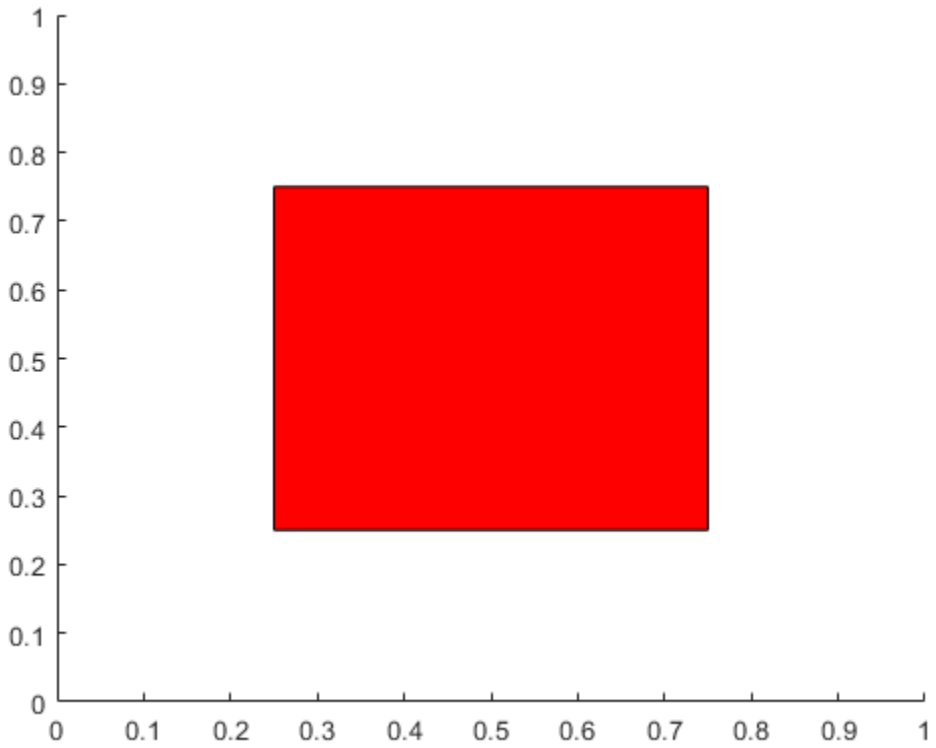
Create a pointer manager in the current figure. Then, associate the pointer behavior structure `pb` with both patch objects. Move the mouse around the figure to see the pointer behavior change.

```
iptSetPointerBehavior([patchobj1,patchobj2],pb);
iptPointerManager(gcf)
```

Change Appearance of Pointer While Traversing Figure

Show a figure with a rectangular patch graphics object. Increase the x- and y-limits of the image to add some white space around the patch.

```
patchobj = patch([.25 .75 .75 .25 .25],...
                 [.25 .25 .75 .75 .25], 'r');
xlim([0 1])
ylim([0 1])
```



Specify the pointer behavior by creating a structure named `pb` with three fields.

- The `enterFcn` and `exitFcn` fields are set to `[]` so the pointer takes no action when it moves across the boundary of a graphics object.
- The `traverseFcn` field is set as a handle to the function `overMe`, which is defined as a helper function at the end of this example. As the pointer moves over the graphics object, the helper function changes the pointer symbol depending on the location of the pointer.

```
pb.enterFcn = [];
pb.exitFcn = [];
pb.traverseFcn = @overMe;
```

Create a pointer manager in the current figure. Then, associate the pointer behavior structure `pb` with the Patch graphics object `patchobj`. Move the mouse around the figure to see changes in the pointer behavior.

```
iptPointerManager(gcf);
iptSetPointerBehavior(patchobj,pb);
```

Helper Function

```
function overMe(hFigure,currentPoint)
%overMe Set figure pointer depending on pointer location.
%   overMe(hFigure,currentPoint) sets the hFigure mouse pointer to be
%   either 'topr', 'topl', 'botr', 'botl', depending on whether
%   currentPoint is in the top right, top left, bottom right, or bottom
%   left of the hFigure's current axes.

hAxes = get(hFigure,'CurrentAxes');

% Get the axes position in pixel units.
oldUnits = get(hAxes,'Units');
set(hAxes,'Units','pixels');
axesPosition = get(hAxes,'Position');
set(hAxes,'Units',oldUnits);

x_middle = axesPosition(1) + 0.5*axesPosition(3);
y_middle = axesPosition(2) + 0.5*axesPosition(4);

x = currentPoint(1,1);
y = currentPoint(1,2);

if (x > x_middle)
    if (y > y_middle)
        pointer = 'topr';
    else
        pointer = 'botr';
    end
else
    if (y > y_middle)
        pointer = 'topl';
    else
        pointer = 'botl';
    end
end

set(hFigure,'Pointer',pointer);
end
```

Input Arguments

`obj` — Graphics object

figure | axes | uipanel | image

Graphics object, specified as a handle to a figure, axes, uipanel, or image graphics objects. `obj` can also be an array of graphics objects.

`pointerBehavior` — Pointer behavior

structure

Pointer behavior, specified as a structure with three fields.

To define the specific actions of the pointer, set the value of these fields to function handles. If you set a field to `[]`, then no action is taken. When the pointer manager calls the function handles, it passes two arguments: the figure object and the current position of the pointer.

Field	When Called
`enterFcn`	Called when the mouse pointer moves over the object.
`traverseFcn`	Called once when the mouse pointer moves over the object, and called again each time the mouse moves within the object.
`exitFcn`	Called when the mouse pointer leaves the object.

`fun` — Pointer behavior when pointer moves over object
function handle

Pointer behavior when pointer moves over object, specified as a function handle.

Data Types: `function_handle`

Tips

- If you specify a pointer behavior using `iptSetPointerBehavior` and then change the figure pointer without using `iptSetPointerBehavior`, then the `iptPointerManager` may not update to reflect the new behavior. Some ways to change the figure pointer without using `iptSetPointerBehavior` include using ROI objects such as `Polygon`, another graphics object, another custom UI, or code that modifies the pointer from within a callback.

See Also

`iptGetPointerBehavior` | `iptPointerManager`

Introduced in R2006a

iptsetpref

Set Image Processing Toolbox preferences or display valid values

Syntax

```
iptsetpref(prefname)
iptsetpref(prefname,prefvalue)
```

Description

`iptsetpref(prefname)` displays the valid values for the Image Processing Toolbox preference specified by `prefname`.

`iptsetpref(prefname,prefvalue)` sets the Image Processing Toolbox preference specified by the `prefname` to the value specified by `prefvalue`. The setting persists until you change it.

You can also use the Image Processing Toolbox Preferences dialog box to set the preferences. To access the dialog box, click **Preferences** on the **Home** tab in the MATLAB desktop, or call the `iptprefs` function.

Examples

Set Image Processing Toolbox Preference

```
iptsetpref('ImshowBorder','tight')
```

Input Arguments

`prefname` — Name of Image Processing Toolbox preference

character vector

Name of an Image Processing Toolbox preference, specified as one of the following character vectors.

The following table details the available preferences and their syntaxes. Note that preference names are case insensitive and you can abbreviate them. The default value appears enclosed in braces (`{}`).

Image Processing Toolbox Preferences

Preference Name	Description
`'ImshowAxesVisible'`	Controls whether `imshow` displays images with the axes box and tick labels. Possible values: `'on'` — Include axes box and tick labels. `{'off'}` — Do not include axes box and tick labels.
`'ImshowBorder'`	Controls whether `imshow` includes a border around the image in the figure window. Possible values: `{'loose'}` — Include a border between the image and the edges of the figure window, thus leaving room for axes labels, titles, etc. `'tight'` — Adjust the figure size so that the image entirely fills the figure. **Note** There still can be a border if the image is very small, or if there are other objects besides the image and its axes in the figure. You can override this preference by specifying the `'Border'` parameter when you call `imshow`.
`'ImshowInitialMagnification'`	Controls the initial magnification of the image displayed by `imshow`. Possible values: Any numeric value — `imshow` interprets numeric values as a percentage. The default value is `100`. A magnification of 100% means that there should be one screen pixel for every image pixel. `'fit'` — Scale the image so that it fits into the window in its entirety. You can override this preference by specifying the `'InitialMagnification'` parameter when you call `imshow`, or by calling the `truesize` function manually after displaying the image.

Preference Name	Description
`'ImtoolInitialMagnification'`	Controls the initial magnification of the image displayed by the **Image Viewer** app. Possible values: `{'adaptive'}` — Display the entire image. If the image is too large to display on the screen at 100% magnification, display the image at the largest magnification that fits on the screen. This is the default. Any numeric value — Specify the magnification as a percentage. A magnification of 100% means that there should be one screen pixel for every image pixel. `'fit'` — Scale the image so that it fits into the window in its entirety. You can override this preference by specifying the `'InitialMagnification'` parameter when you open the **Image Viewer** app using the `imtool` function.
`'ImtoolStartWithOverview'`	Controls whether the Overview tool opens automatically when you open an image using the **Image Viewer** app. Possible values: `true` — Overview tool opens when you open an image. `{false}` — Overview tool does not open when you open an image. This is the default behavior.
`'VolumeViewerUseHardware'`	Controls whether the **Volume Viewer** app uses OpenGL shaders on the local graphics hardware to optimize volume rendering. Possible values: `{true}` — Enable hardware optimization. `false` — Disable hardware optimization. **Note** Setting this preference to `false` has the side effect of removing certain functionality from the app and will drastically slow down rendering performance. This preference should only be set to false in technical support scenarios to resolve problems with graphics drivers.

Preference Name	Description
'UseIPPL'	Controls whether some toolbox functions use hardware optimization or not. Possible values: {true} — Enable hardware optimization false — Disable hardware optimization **Note** Setting this preference value clears all loaded MEX-files.

Data Types: char

prefvalue — Value you want to assign to an Image Processing Toolbox preference
character vector | numeric scalar | logical scalar

Value you want to assign to an Image Processing Toolbox preference, specified as one of the values listed in the table in prefname.

Example: iptsetpref('ImshowBorder','tight')

See Also

Image Viewer | **Volume Viewer** | imshow | iptgetpref | iptprefs

Introduced before R2006a

iptwindowalign

Align figure windows

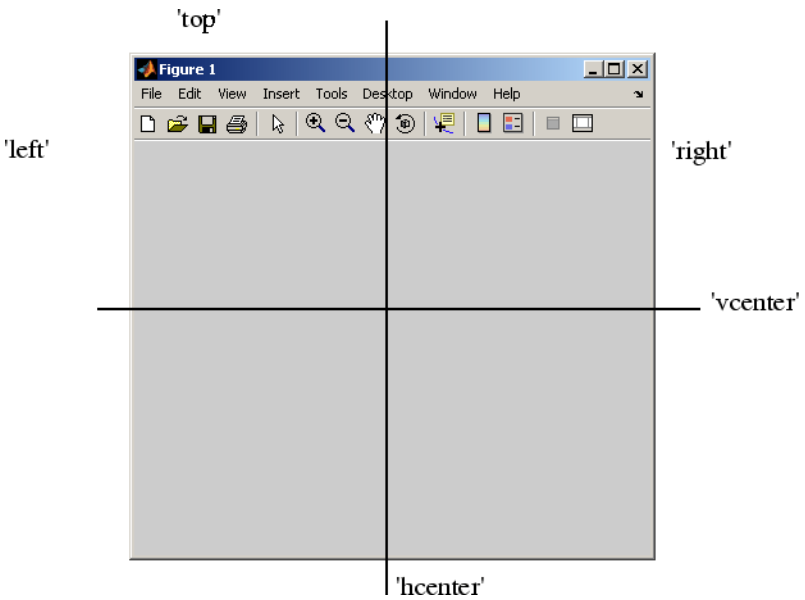
Syntax

```
iptwindowalign(fixed_fig,fixed_edge, moving_fig,moving_edge)
```

Description

`iptwindowalign(fixed_fig,fixed_edge, moving_fig,moving_edge)` aligns the edge `moving_edge` of figure `moving_fig` with the edge `fixed_edge` of figure `fixed_fig`.

You can align two figure windows along the top, bottom, left, or right edges. You can also center the figures horizontally or vertically. The figure shows the possible alignments.



Examples

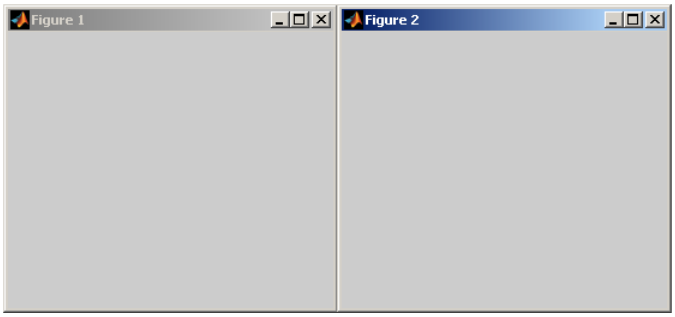
Align Two Figure Windows

To illustrate some possible figure window alignments, first create two figures: `fig1` and `fig2`. Initially, `fig2` overlays `fig1` on the screen.

```
fig1 = figure;
fig2 = figure;
```

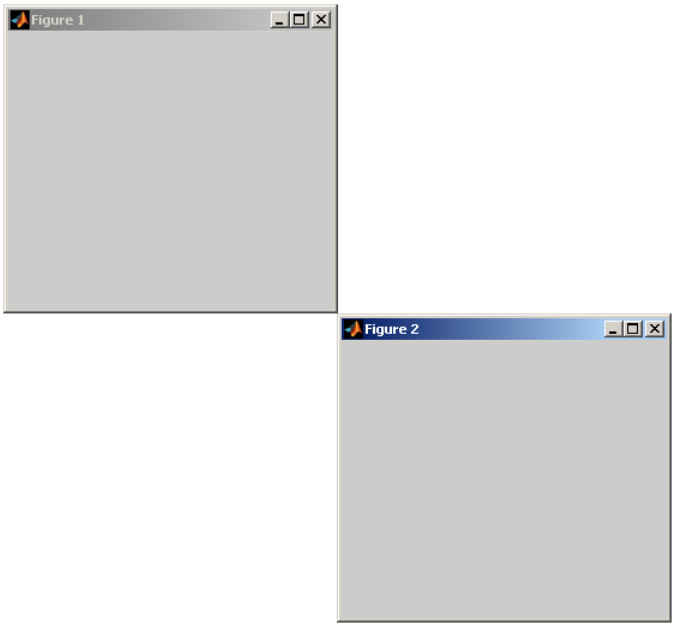
Use `iptwindowalign` to move `fig2` so its left edge is aligned with the right edge of `fig1`.

```
iptwindowalign(fig1,'right',fig2,'left');
```



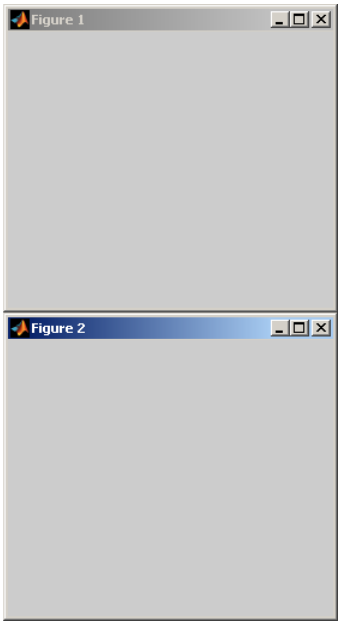
Now move `fig2` so its top edge is aligned with the bottom edge of `fig1`.

```
iptwindowalign(fig1,'bottom',fig2,'top');
```



Now move `fig2` so the two figures are centered horizontally.

```
iptwindowalign(fig1,'hcenter',fig2,'hcenter');
```



Input Arguments

`fixed_fig` — Fixed figure window

handle

Fixed figure window, specified as a handle to a figure.

`fixed_edge` — Alignment of fixed figure window

`'left'` | `'right'` | `'hcenter'` | `'top'` | `'bottom'` | `'vcenter'`

Alignment of the fixed figure window, specified as `'left'`, `'right'`, `'hcenter'`, `'top'`, `'bottom'`, or `'vcenter'`. To center the figures horizontally, use `'hcenter'`. To center the figures vertically, use `'vcenter'`.

`moving_fig` — fixed

numeric array

Moving figure window, specified as a handle to a figure.

`moving_edge` — Alignment of moving figure window

`'left'` | `'right'` | `'hcenter'` | `'top'` | `'bottom'` | `'vcenter'`

Alignment of the moving figure window, specified as `'left'`, `'right'`, `'hcenter'`, `'top'`, `'bottom'`, or `'vcenter'`. To center the figures vertically, use `'vcenter'`.

Tips

- The two specified edges must be consistent in terms of their direction. For example, you cannot specify `'left'` for `fixed_edge` and `'bottom'` for `moving_edge`.
- `iptwindowalign` constrains the position adjustment of `moving_fig` to keep the figure entirely visible on the screen.

- `iptwindowalign` has no effect if either figure window is docked.

See Also

Image Viewer

Introduced before R2006a

iradon

Inverse Radon transform

Syntax

```
I = iradon(R,theta)
I = iradon(R,theta,interp,filter,frequency_scaling,output_size)
[I,H] = iradon( ___ )
```

Description

`I = iradon(R,theta)` reconstructs the image `I` from projection data in `R` captured at projection angles `theta`.

`I = iradon(R,theta,interp,filter,frequency_scaling,output_size)` specifies parameters to use in the inverse Radon transform. You can specify any combination of the last four arguments. `iradon` uses default values for arguments that you omit.

`[I,H] = iradon( ___ )` also returns the frequency response of the filter, `H`.

Examples

Compare Filtered and Unfiltered Backprojection

Create an image of the phantom. Display the image.

```
P = phantom(128);
imshow(P)
title('Original image')
```

Perform a Radon transform of the image.

```
R = radon(P,0:179);
```

Perform filtered backprojection.

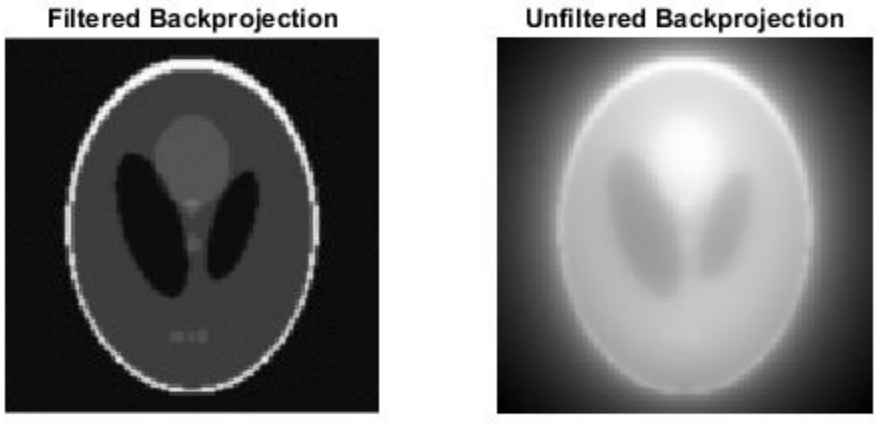
```
I1 = iradon(R,0:179);
```

Perform unfiltered backprojection.

```
I2 = iradon(R,0:179,'linear','none');
```

Display the reconstructed images.

```
figure
subplot(1,2,1)
imshow(I1,[])
title('Filtered Backprojection')
subplot(1,2,2)
imshow(I2,[])
title('Unfiltered Backprojection')
```



Examine Backprojection at a Single Angle

Create an image of the phantom.

```
P = phantom(128);
```

Perform a Radon transform of the image, then get the projection vector corresponding to a projection at a 45 degree angle.

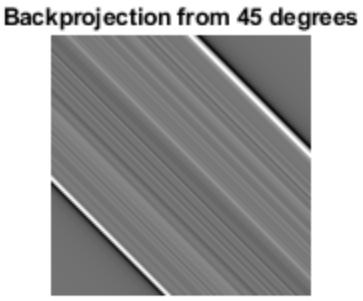
```
R = radon(P,0:179);
r45 = R(:,46);
```

Perform the inverse Radon transform of this single projection vector. The `iradon` syntax does not allow you to do this directly, because if `theta` is a scalar it is treated as an increment. You can accomplish the task by passing in two copies of the projection vector and then dividing the result by 2.

```
I = iradon([r45 r45], [45 45])/2;
```

Display the result.

```
imshow(I, [])
title('Backprojection from 45 degrees')
```



Input Arguments

R — Parallel beam projection data

numeric column vector | numeric matrix

Parallel beam projection data, specified as one of the following.

- If `theta` is a scalar, then specify `R` as a numeric column vector containing the Radon transform for `theta` degrees.
- If `theta` is a vector, then specify `R` as a 2-D matrix in which each column is the Radon transform for one of the angles in `theta`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

theta — Projection angles

numeric vector | numeric scalar | `[]`

Projection angles (in degrees), specified as one of the following.

Value	Description
numeric vector	Projection angles. There must be equal spacing between the angles.

Value	Description
numeric scalar	Incremental angle between projections. Projections are taken at angles `m*theta`, where `m = 0,1,2,...,size(R,2)-1`.
`[]`	Automatically set the incremental angle between projections to `180/size(R,2)`

Data Types: `double`

`interp` — Type of interpolation
`'linear'` (default) | `'nearest'` | `'spline'` | `'pchip'` | `'v5cubic'`

Type of interpolation to use in the back projection, specified as one of these values, listed in order of increasing accuracy and computational complexity.

Value	Description
`'nearest'`	Nearest-neighbor interpolation
`'linear'`	Linear interpolation
`'spline'`	Spline interpolation
`'pchip'`	Shape-preserving piecewise cubic interpolation
`'v5cubic'`	Cubic convolution used in MATLAB 5

Data Types: `char` | `string`

`filter` — Filter
`'Ram-Lak'` (default) | `'Shepp-Logan'` | `'Cosine'` | `'Hamming'` | `'Hann'` | `'None'`

Filter to use for frequency domain filtering, specified as one of these values.

Value	Description
`'Ram-Lak'`	Cropped Ram-Lak or ramp filter. The frequency response of this filter is | `f` |. Because this filter is sensitive to noise in the projections, one of the filters listed below might be preferable. These filters multiply the Ram-Lak filter by a window that de-emphasizes high frequencies.
`'Shepp-Logan'`	Multiplies the Ram-Lak filter by a `sinc` function
`'Cosine'`	Multiplies the Ram-Lak filter by a `cosine` function
`'Hamming'`	Multiplies the Ram-Lak filter by a Hamming window
`'Hann'`	Multiplies the Ram-Lak filter by a Hann window
`'None'`	No filtering. `iradon` returns unfiltered backprojection data.

Data Types: `char` | `string`

`frequency_scaling` — Scale factor
1 (default) | positive number in the range (0, 1]

Scale factor for rescaling the frequency axis, specified as a positive number in the range (0, 1]. If `frequency_scaling` is less than 1, then the filter is compressed to fit into the frequency range `[0,frequency_scaling]`, in normalized frequencies; all frequencies above `frequency_scaling` are set to `0`.

`output_size` — Number of rows and columns in the reconstructed image
positive integer

Number of rows and columns in the reconstructed image, specified as a positive integer. If `output_size` is not specified, the size is determined from the length of the projections according to:

```
output_size = 2*floor(size(R,1)/(2*sqrt(2)))
```

If you specify `output_size`, then `iradon` reconstructs a smaller or larger portion of the image but does not change the scaling of the data. If the projections were calculated with the `radon` function, then the reconstructed image might not be the same size as the original image.

Output Arguments

`I` — Grayscale image
numeric matrix

Grayscale image, returned as a numeric matrix. If input projection data `R` is data type `single`, then `I` is `single`; otherwise `I` is `double`.

Data Types: `single` | `double`

`H` — Frequency response
numeric vector

Frequency response of the filter, returned as a numeric vector.

Data Types: `double`

Algorithms

`iradon` assumes that the center of rotation is the center point of the projections, which is defined as `ceil(size(R,1)/2)`.

`iradon` uses the filtered back projection algorithm to perform the inverse Radon transform. The filter is designed directly in the frequency domain and then multiplied by the FFT of the projections. The projections are zero-padded to a power of 2 before filtering to prevent spatial domain aliasing and to speed up the FFT.

References

[1] Kak, A. C., and M. Slaney, *Principles of Computerized Tomographic Imaging*, New York, NY, IEEE Press, 1988.

Extended Capabilities

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- The GPU implementation of this function supports only nearest-neighbor and linear interpolation methods.

For more information, see “Image Processing on a GPU”.

See Also

`fan2para` | `fanbeam` | `ifanbeam` | `para2fan` | `phantom` | `radon`

Introduced before R2006a

isflat

True for flat structuring element

Note `isflat` will be removed in a future release. See `strel` for the current list of methods.

Syntax

```
TF = isflat(SE)
```

Description

`TF = isflat(SE)` returns `true` (`1`) if the structuring element `SE` is flat; otherwise the function returns `false` (`0`).

Input Arguments

SE — Structuring element

`strel` object | `offsetstrel` object | array of `strel` objects | array of `offsetstrel` objects

Structuring element, specified as a `strel` object, an array of `strel` objects, an `offsetstrel` object, or an array of `offsetstrel` objects.

Output Arguments

TF — Structuring element is flat

logical scalar | logical array

Structuring element `SE` is flat, returned as a logical scalar or logical array of the same size as `SE`. If the structuring element consists of `offsetstrel` objects, then `TF` is `false` regardless of the neighborhood.

Data Types: `logical`

See Also

Topics

“Structuring Elements”

Introduced before R2006a

isicc

Check for valid ICC profile data

Syntax

```
tf = isicc(profile)
```

Description

`tf = isicc(profile)` checks if the input profile is a valid International Color Consortium (ICC) profile data. The function returns a logical value that indicates if the input is a valid ICC profile.

Examples

Check if ICC Profile Data is Valid

Read in an ICC profile data into the workspace.

```
profile = iccread('sRGB.icm');
```

Check if `profile` is a valid ICC profile data. The function returns logical 1 (true).

```
tf = isicc(profile)
```

```
tf = logical
   1
```

Create a new ICC profile data without the header and copyright fields. Use `rmfield` to remove the '`Header`' and '`Copyright`' fields from the ICC profile data structure.

```
newProfile = rmfield(profile,{'Header','Copyright'});
```

Inspect the new profile data to verify that the '`Header`' and '`Copyright`' fields are removed.

```
newProfile
```

```
newProfile = struct with fields:
             TagTable: {17x3 cell}
          Description: [1x1 struct]
      MediaWhitePoint: [0.9505 1 1.0891]
      MediaBlackPoint: [0 0 0]
        DeviceMfgDesc: [1x1 struct]
      DeviceModelDesc: [1x1 struct]
      ViewingCondDesc: [1x1 struct]
    ViewingConditions: [1x1 struct]
            Luminance: [76.0365 80 87.1246]
          Measurement: [1x1 struct]
           Technology: 'Cathode Ray Tube Display'
              MatTRC: [1x1 struct]
          PrivateTags: {}
```

```
             Filename: 'sRGB.icm'
```

Check if `newProfile` is a valid ICC profile data. The function returns logical 0 (false).

```
tf = isicc(newProfile)
```

```
tf = logical
   0
```

Input Arguments

`profile` — ICC profile data

structure array

ICC profile data, specified as a structure array, represents an ICC profile in the data format returned by `iccread`. The ICC profile data must contain all the tags and fields required by the ICC profile specification.

Data Types: `struct`

Output Arguments

`tf` — Valid ICC profile

1 (true) | 0 (false)

Valid ICC profile, returned as logical 1 (`true`) when the input is a valid ICC profile data, and logical 0 (`false`) otherwise.

Algorithms

`isicc` checks if `profile` has a complete set of the tags required for an ICC profile. `profile` must contain a `Header` field, which in turn must contain a `Version` field and a `DeviceClass` field. These fields along with others, are used to determine the set of required tags according to the ICC profile specification. The required tags for ICC profile specifications related to Version 2 (ICC.1:2001-04) and Version 4 (ICC.1:2001-12) are available at https://www.color.org.

See Also

`applycform` | `iccread` | `iccwrite` | `makecform`

Introduced before R2006a

isnitf

Check if file is National Imagery Transmission Format (NITF) file

Syntax

```
[tf,NITFversion] = isnitf(filename)
```

Description

`[tf,NITFversion] = isnitf(filename)` returns `true` if the file specified by `filename` is a National Imagery Transmission Format (NITF) file. `isnitf` also returns the NITF version of valid NITF files in `NITFversion`.

Input Arguments

`filename` — Name of file

character vector | string scalar

Name of file, specified as a character vector or string scalar.

Data Types: `char` | `string`

Output Arguments

`tf` — File is NITF file

`true` | `false`

File is an NITF file, returned as `true` or `false`.

`NITFversion` — NITF version

character vector | `'UNK'`

NITF version of the file, returned as a character vector such as `'2.1'`. If the file is not an NITF file, then `NITFversion` is returned as `'UNK'`.

See Also

`nitfinfo` | `nitfread`

Introduced in R2007b

isRigid

Determine if transformation is rigid transformation

Syntax

```
TF = isRigid(tform)
```

Description

`TF = isRigid(tform)` determines whether or not the affine transformation specified by `tform` is a rigid transformation.

Examples

Check If 2-D Transformation Is Rigid

Create an `affine2d` object that defines a pure translation.

```
A = [ 1  0  0
      0  1  0
     40 40  1 ];

tform = affine2d(A)
```

```
tform =

  affine2d with properties:

                T: [3x3 double]
    Dimensionality: 2
```

Test if it is a rigid transformation.

```
tf = isRigid(tform)
```

```
tf =

     1
```

Check If 3-D Transformation Is Rigid

Create an `affine3d` object that defines a different scale factor in each dimension.

```
Sx = 1.2;
Sy = 1.6;
Sz = 2.4;
tform = affine3d([Sx 0 0 0; 0 Sy 0 0; 0 0 Sz 0; 0 0 0 1])
```

```
tform =

  affine3d with properties:
```

```
                T: [4x4 double]
    Dimensionality: 3
```

Check if the transformation is rigid.

```
TF = isRigid(tform)
```

```
TF =

     0
```

Input Arguments

`tform` — Geometric transformation

`affine2d` or `affine3d` geometric transformation object

Geometric transformation, specified as an `affine2d` or `affine3d` geometric transformation object.

Output Arguments

TF — Flag indicating rigid transformation

scalar

Flag indicating rigid transformation, returned as a logical scalar. `TF` is `True` when `tform` is a rigid transformation.

Data Types: `logical`

More About

Rigid Transformation

A rigid transformation includes only rotation and translation. It does not include reflection, and it does not modify the size or shape of an input object.

See Also

`isSimilarity` | `isTranslation`

Introduced in R2013a

isrset

Check if file is valid R-Set file

Syntax

```
[tf,supported] = isrset(filename)
```

Description

`[tf,supported] = isrset(filename)` checks if the specified file is a valid reduced resolution dataset (R-Set) file.

Logical scalar `tf` indicates if the file is an R-Set file. Logical scalar `supported` confirms if input is an R-Set file compatible with the current version of the Image Processing Toolbox. If the function returns true for both `tf` and `supported`, the specified file is a valid R-Set file.

Examples

Check If File Is Valid R-Set File

Load a file into the workspace.

```
filename = 'MandiRset';
```

Check if the file is a valid R-Set file. Confirm if both outputs are `true`.

```
[tf,supported] = isrset(filename)
```

```
tf = logical
   1


supported = logical
   1
```

Input Arguments

`filename` — Name of the file

character vector | string scalar

Name of the file, specified as a character vector or string scalar.

Data Types: `char` | `string`

Output Arguments

`tf` — R-Set file type validation

logical scalar

R-Set file type validation, returned as a logical scalar.

- `1(true)` — The input file is an R-Set file.
- `0(false)` — The input file is not an R-Set file.

Data Types: `logical`

`supported` — Version support validation
logical scalar

Version support validation, returned as a logical scalar.

- `1(true)` — The specified file is an R-Set file created using a version of the `rsetwrite` function that is compatible with the version of the Image Processing Toolbox used to read the R-Set file.
- `0(false)` — The specified file is either:
  - Not an R-Set file
  - An R-Set file created using a version of the `rsetwrite` function that is not compatible with the version of the Image Processing Toolbox used to read the R-Set file.

Data Types: `logical`

See Also

`rsetwrite` | `openrset`

Introduced in R2009a

isSimilarity

Determine if transformation is similarity transformation

Syntax

```
TF = isSimilarity(tform)
```

Description

`TF = isSimilarity(tform)` determines whether or not the affine transformation specified by `tform` is a similarity transformation.

Examples

Check if 2-D transformation is a similarity transformation

Create an `affine2d` object that defines a pure translation.

```
A = [ 1  0  0
      0  1  0
     40 40  1 ];

tform = affine2d(A)
```

```
tform =

  affine2d with properties:

                T: [3x3 double]
    Dimensionality: 2
```

Check if transformation is a similarity transformation.

```
tf = isSimilarity(tform)
```

```
tf =

     1
```

Check if 3-D transformation is a similarity transformation

Create an `affine3d` object that defines a different scale factor in each dimension.

```
Sx = 1.2;
Sy = 1.6;
Sz = 2.4;
tform = affine3d([Sx 0 0 0; 0 Sy 0 0; 0 0 Sz 0; 0 0 0 1])
```

```
tform =

  affine3d with properties:
```

```
                T: [4x4 double]
    Dimensionality: 3
```

Check if the transformation is a similarity transformation.

```
TF = isSimilarity(tform)
```

```
TF =

     0
```

Input Arguments

`tform` — Geometric transformation

`affine2d` or `affine3d` geometric transformation object

Geometric transformation, specified as an `affine2d` or `affine3d` geometric transformation object.

Output Arguments

TF — Flag indicating similarity transformation

scalar

Flag indicating similarity transformation, returned as a logical scalar. `TF` is `True` when `tform` is a similarity transformation.

Data Types: `logical`

More About

Similarity Transformation

A similarity transformation includes only rotation, translation, isotropic scaling, and reflection. A similarity transformation does not modify the shape of an input object. Straight lines remain straight, and parallel lines remain parallel.

Note `isSimilarity` returns `True` if the transformation includes reflection. Some toolbox functions, such as `imregister`, support only non-reflective similarity. Other functions, such as `fitgeotrans`, support reflection.

See Also

`isRigid` | `isTranslation`

Introduced in R2013a

isTranslation

Determine if transformation is pure translation

Syntax

```
TF = isTranslation(tform)
```

Description

`TF = isTranslation(tform)` determines whether or not the rigid or affine transformation specified by `tform` is a pure translation.

Examples

Check If 2-D Transformation Is a Pure Translation

Create an `affine2d` object that defines a pure translation.

```
A = [ 1  0  0
      0  1  0
     40 40  1 ];

tform = affine2d(A)
```

```
tform =

  affine2d with properties:

                T: [3x3 double]
    Dimensionality: 2
```

Check if the transformation is a pure translation.

```
tf = isTranslation(tform)
```

```
tf =

     1
```

Check If 3-D Transformation Is a Pure Translation

Create an `affine3d` object that defines a different scale factor in each dimension.

```
Sx = 1.2;
Sy = 1.6;
Sz = 2.4;
tform = affine3d([Sx 0 0 0; 0 Sy 0 0; 0 0 Sz 0; 0 0 0 1]);
```

```
tf =

  affine3d with properties:
```

```
                T: [4x4 double]
    Dimensionality: 3
```

Check if the transformation is a pure translation. Since `tform` scales the object,

```
tf = isTranslation(tform)
```

```
tf =

     0
```

As expected, the transformation is not a pure translation since scaling changes the size and shape of an input volume.

Input Arguments

`tform` — Geometric transformation

`affine2d` object | `affine3d` object | `rigid2d` object

Geometric transformation, specified as an `affine2d`, `affine3d`, or `rigid2d` geometric transformation object.

Output Arguments

TF — Flag indicating pure translation transformation

scalar

Flag indicating pure translation transformation, returned as a logical scalar. TF is `True` when `tform` represents a pure translation.

Data Types: `logical`

More About

Translation Transformation

A translation transformation shifts an image without modifying the image size, shape, or orientation. A 2-D translation is represented by a matrix T of the form:

```
[1 0 0;
 0 1 0;
 e f 1];
```

A 3-D translation is represented by a matrix of the form:

```
[1 0 0 0;
 0 1 0 0;
 0 0 1 0;
 j k l 1];
```

See Also

`isRigid` | `isSimilarity`

Introduced in R2013a

jaccard

Jaccard similarity coefficient for image segmentation

Syntax

```
similarity = jaccard(BW1,BW2)
similarity = jaccard(L1,L2)
similarity = jaccard(C1,C2)
```

Description

`similarity = jaccard(BW1,BW2)` computes the intersection of binary images `BW1` and `BW2` divided by the union of `BW1` and `BW2`, also known as the Jaccard index. The images can be binary images, label images, or categorical images.

`similarity = jaccard(L1,L2)` computes the Jaccard index for each label in label images `L1` and `L2`.

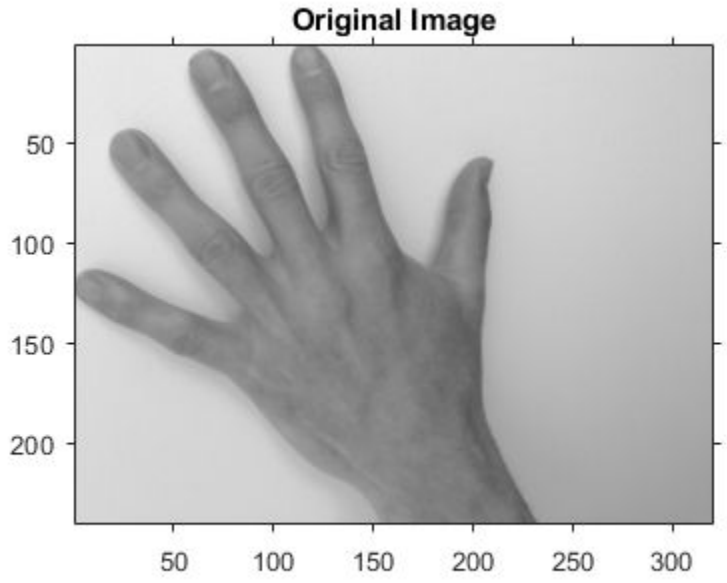
`similarity = jaccard(C1,C2)` computes the Jaccard index for each category in categorical images `C1` and `C2`.

Examples

Compute Jaccard Similarity Coefficient for Binary Segmentation

Read an image containing an object to segment. Convert the image to grayscale, and display the result.

```
A = imread('hands1.jpg');
I = im2gray(A);
figure
imshow(I)
title('Original Image')
```



Use the active contours (snakes) method to segment the hand.

```
mask = false(size(I));
mask(25:end-25,25:end-25) = true;
BW = activecontour(I, mask, 300);
```

Read in the ground truth against which to compare the segmentation.

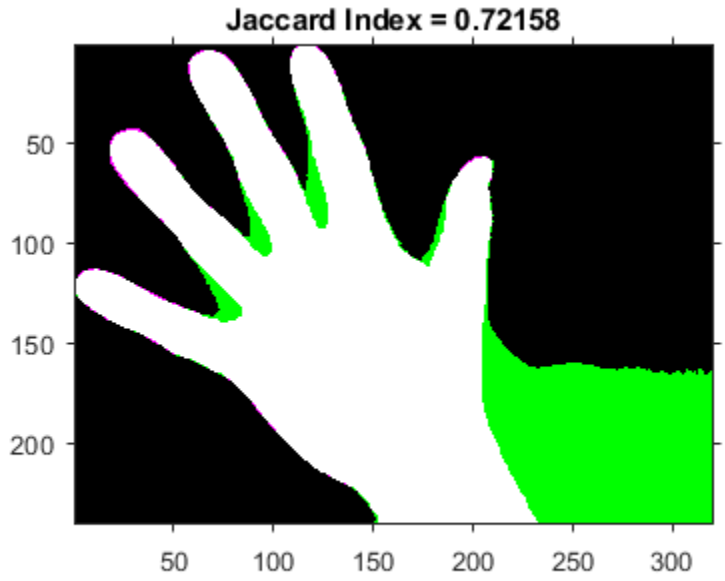
```
BW_groundTruth = imread('hands1-mask.png');
```

Compute the Jaccard index of this segmentation.

```
similarity = jaccard(BW, BW_groundTruth);
```

Display the masks on top of each other. Colors indicate differences in the masks.

```
figure
imshowpair(BW, BW_groundTruth)
title(['Jaccard Index = ' num2str(similarity)])
```



Compute Jaccard Similarity Coefficient for Multi-Region Segmentation

This example shows how to segment an image into multiple regions. The example then computes the Jaccard similarity coefficient for each region.

Read in an image with several regions to segment.

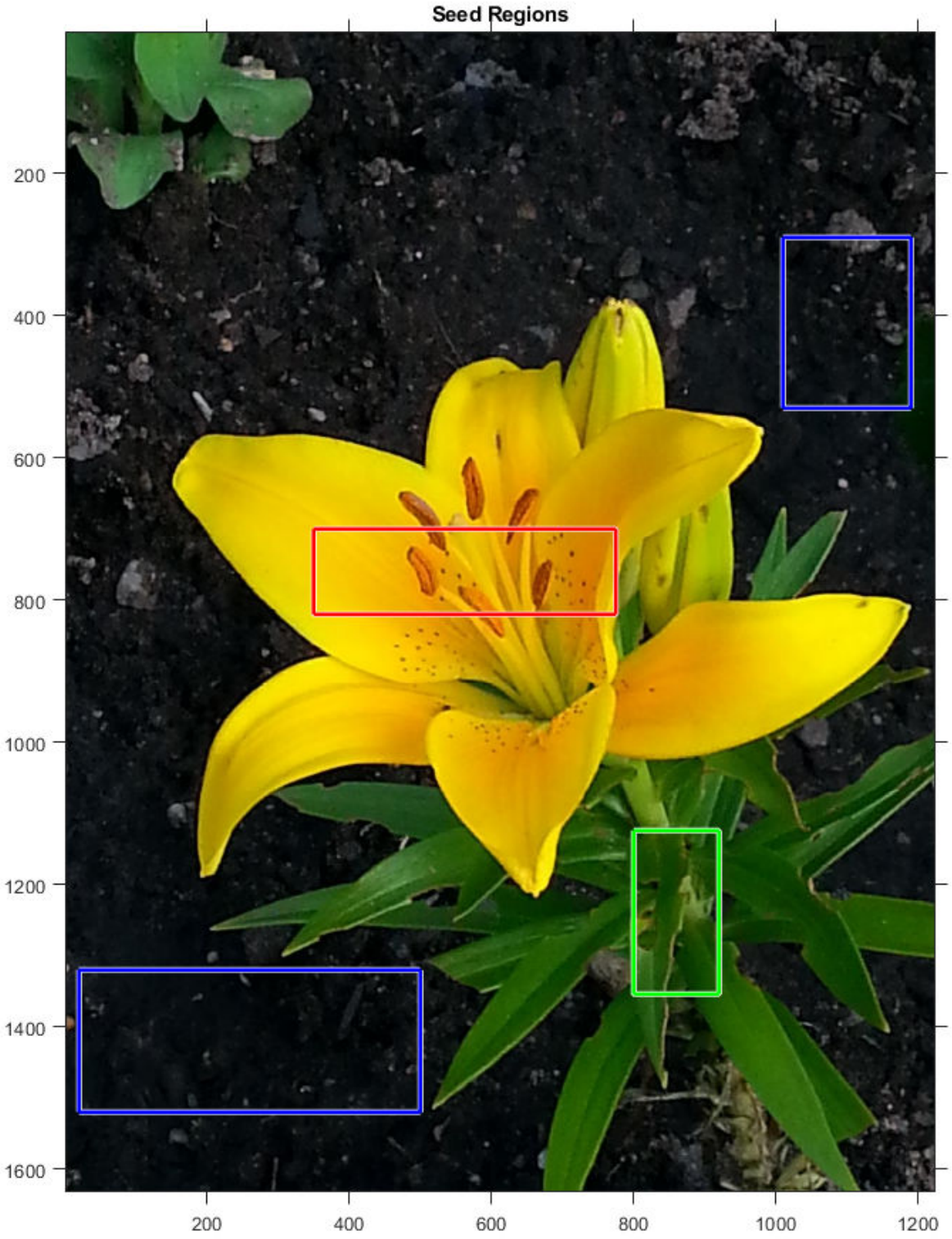
```
RGB = imread('yellowlily.jpg');
```

Create scribbles for three regions that distinguish their typical color characteristics. The first region classifies the yellow flower. The second region classifies the green stem and leaves. The last region classifies the brown dirt in two separate patches of the image. Regions are specified by a 4-element vector, whose elements indicate the x- and y-coordinate of the upper left corner of the ROI, the width of the ROI, and the height of the ROI.

```
region1 = [350 700 425 120]; % [x y w h] format
BW1 = false(size(RGB,1),size(RGB,2));
BW1(region1(2):region1(2)+region1(4),region1(1):region1(1)+region1(3)) = true;

region2 = [800 1124 120 230];
BW2 = false(size(RGB,1),size(RGB,2));
BW2(region2(2):region2(2)+region2(4),region2(1):region2(1)+region2(3)) = true;

region3 = [20 1320 480 200; 1010 290 180 240];
BW3 = false(size(RGB,1),size(RGB,2));
BW3(region3(1,2):region3(1,2)+region3(1,4),region3(1,1):region3(1,1)+region3(1,3)) = true;
BW3(region3(2,2):region3(2,2)+region3(2,4),region3(2,1):region3(2,1)+region3(2,3)) = true;
```

Display the seed regions on top of the image.

```
figure
imshow(RGB)
hold on
visboundaries(BW1,'Color','r');
visboundaries(BW2,'Color','g');
visboundaries(BW3,'Color','b');
title('Seed Regions')
```



Segment the image into three regions using geodesic distance-based color segmentation.

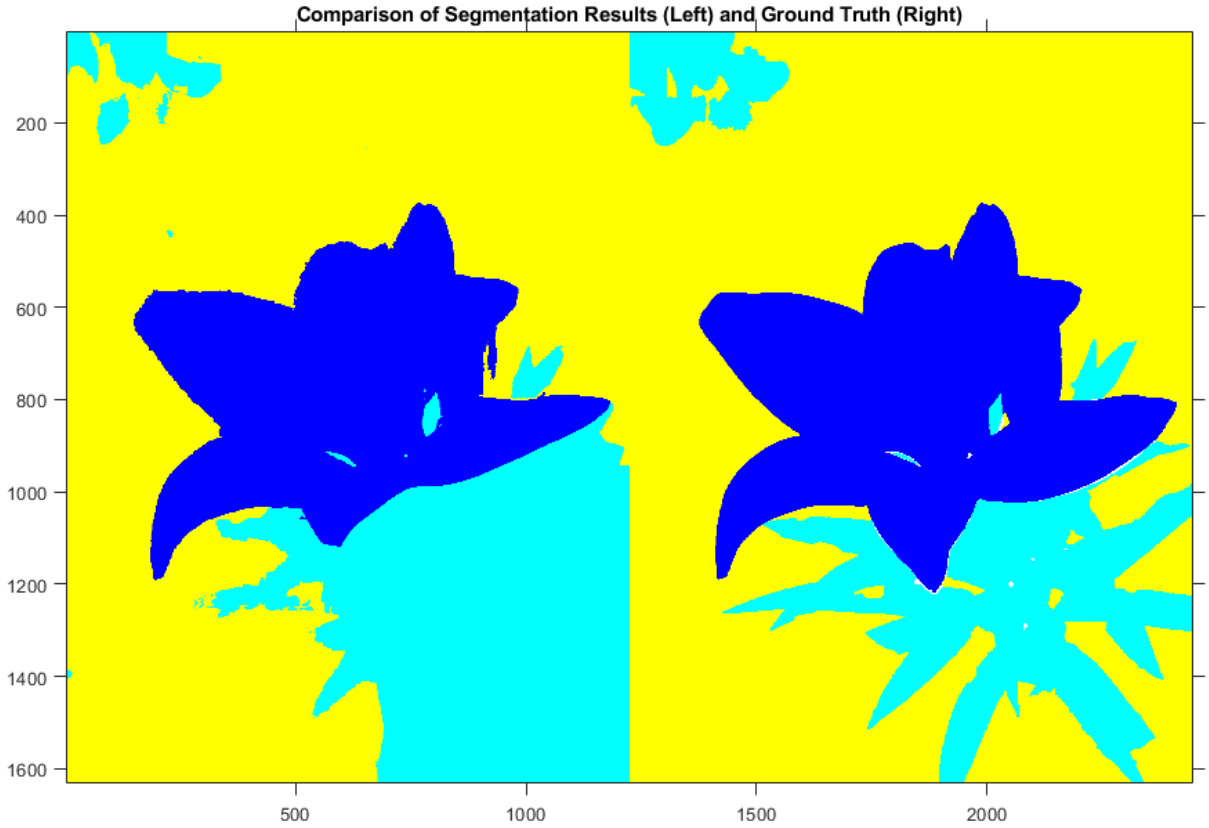
```
L = imseggeodesic(RGB,BW1,BW2,BW3,'AdaptiveChannelWeighting',true);
```

Load a ground truth segmentation of the image.

```
L_groundTruth = double(imread('yellowlily-segmented.png'));
```

Visually compare the segmentation results with the ground truth.

```
figure
imshowpair(label2rgb(L),label2rgb(L_groundTruth),'montage')
title('Comparison of Segmentation Results (Left) and Ground Truth (Right)')
```



Compute the Jaccard similarity index (IoU) for each segmented region.

```
similarity = jaccard(L, L_groundTruth)
```

```
similarity = 3×1

    0.8861
    0.5683
    0.8414
```

The Jaccard similarity index is noticeably smaller for the second region. This result is consistent with the visual comparison of the segmentation results, which erroneously classifies the dirt in the lower right corner of the image as leaves.

Input Arguments

`BW1` — First binary image

logical array

First binary image, specified as a logical array of any dimension.

Data Types: `logical`

`BW2` — Second binary image

logical array

Second binary image, specified as a logical array of the same size as `BW1`.

Data Types: `logical`

`L1` — First label image

array of nonnegative integers

First label image, specified as an array of nonnegative integers, of any dimension.

Data Types: `double`

`L2` — Second label image

array of nonnegative integers

Second label image, specified as an array of nonnegative integers, of the same size as `L1`.

Data Types: `double`

`C1` — First categorical image

`categorical` array

First categorical image, specified as a `categorical` array of any dimension.

Data Types: `category`

`C2` — Second categorical image

`categorical` array

Second categorical image, specified as a `categorical` array of the same size as `C1`.

Data Types: `category`

Output Arguments

`similarity` — Jaccard similarity coefficient

numeric scalar | numeric vector

Jaccard similarity coefficient, returned as a numeric scalar or numeric vector with values in the range [0, 1]. A `similarity` of 1 means that the segmentations in the two images are a perfect match. If the input arrays are:

- binary images, `similarity` is a scalar.
- label images, `similarity` is a vector, where the first coefficient is the Jaccard index for label 1, the second coefficient is the Jaccard index for label 2, and so on.

- categorical images, `similarity` is a vector, where the first coefficient is the Jaccard index for the first category, the second coefficient is the Jaccard index for the second category, and so on.

Data Types: `double`

More About

Jaccard Similarity Coefficient

The Jaccard similarity coefficient of two sets A and B (also known as intersection over union or IoU) is expressed as:

$$\texttt{jaccard}(A,B) = |\,\texttt{intersection}(A,B)\,| \,/\, |\,\texttt{union}(A,B)\,|$$

where $|A|$ represents the cardinal of set A. The Jaccard index can also be expressed in terms of true positives (TP), false positives (FP) and false negatives (FN) as:

$$\texttt{jaccard}(A,B) = TP \,/\, (TP + FP + FN)$$

The Jaccard index is related to the Dice index according to:

$$\texttt{jaccard}(A,B) = \texttt{dice}(A,B) \,/\, (2 - \texttt{dice}(A,B))$$

See Also

`dice` | `bfscore`

Introduced in R2017b

jitterColorHSV

Randomly alter color of pixels

Syntax

```
J = jitterColorHSV(I,Name,Value)
```

Description

`J = jitterColorHSV(I,Name,Value)` adjusts the color of RGB image `I` with a randomly selected value of hue, saturation, brightness, and contrast from the HSV color space on page 1-2204. Specify the range of each type of adjustment using name-value pair arguments.

Examples

Randomly Adjust Image Color

Read and display an image.

```
I = imread('kobi.png');
imshow(I)
```

Randomly adjust the hue, saturation, brightness, and contrast of the image. To demonstrate the randomness of the adjustment, repeat the operation on the original image three times.

```
J1 = jitterColorHSV(I,'Contrast',0.4,'Hue',0.1,'Saturation',0.2,'Brightness',0.3);
J2 = jitterColorHSV(I,'Contrast',0.4,'Hue',0.1,'Saturation',0.2,'Brightness',0.3);
J3 = jitterColorHSV(I,'Contrast',0.4,'Hue',0.1,'Saturation',0.2,'Brightness',0.3);
```

Display the adjusted images in a montage.

```
montage({J1,J2,J3},'Size',[1 3])
```

Input Arguments

`I` — RGB image

m-by-*n*-by-3 numeric array

RGB image with original pixel values, specified as an *m*-by-*n*-by-3 numeric array.

Data Types: `single` | `double` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `J = jitterColorHSV(I,'Hue',0.1)`

`Hue` — Hue offset

`0` (default) | numeric scalar | 2-element numeric vector

Hue offset, specified as the comma-separated pair consisting of `'Hue'` and one of the following values. `jitterColorHSV` converts input RGB image `I` to the HSV color space before adding a random value to the hue channel of the image. `jitterColorHSV` circularly wraps the modified hue to the range [0, 1] before converting the jittered HSV image back to the RGB color space.

Value	Meaning
Numeric scalar in the range [0, 1]	Add random amount of hue from the uniform distribution `[-Hue Hue]`
2-element numeric vector with elements in the range [-1, 1]	Add a random amount of hue from a continuous uniform distribution within the specified interval. The second element must be larger than or equal to the first element.

Data Types: `single` | `double`

`Saturation` — Saturation offset

`0` (default) | numeric scalar | 2-element numeric vector

Saturation offset, specified as the comma-separated pair consisting of `'Saturation'` and one of the following values. `jitterColorHSV` converts input RGB image `I` to the HSV color space before adding a random value to the saturation channel of the image. `jitterColorHSV` clips the modified saturation to the range [0, 1] before converting the jittered HSV image back to the RGB color space.

Value	Meaning
Numeric scalar in the range [0, 1]	Add random amount of saturation from the uniform distribution `[-Saturation Saturation]`
2-element numeric vector with elements in the range [-1, 1]	Add a random amount of saturation from a continuous uniform distribution within the specified interval. The second element must be larger than or equal to the first element.

Data Types: `single` | `double`

`Brightness` — Brightness offset

`0` (default) | numeric scalar | 2-element numeric vector

Brightness offset, specified as the comma-separated pair consisting of `'Brightness'` and one of the following values. `jitterColorHSV` converts input RGB image `I` to the HSV color space before adding a random value to the brightness (value) channel of the image. `jitterColorHSV` clips the modified brightness to the range [0, 1] before converting the jittered HSV image back to the RGB color space.

Value	Meaning
Numeric scalar in the range [0, 1]	Add random amount of brightness from the uniform distribution `[-Brightness Brightness]`
2-element numeric vector with elements in the range [-1, 1]	Add a random amount of brightness from a continuous uniform distribution within the specified interval. The second element must be larger than or equal to the first element.

Data Types: `single` | `double`

`Contrast` — Contrast scale factor

`0` (default) | positive number | 2-element numeric vector

Contrast scale factor, specified as the comma-separated pair consisting of `'Contrast'` and one of the following values. `jitterColorHSV` converts input RGB image `I` to the HSV color space before scaling the brightness (value) channel of the image by a random factor. `jitterColorHSV` clips the modified brightness to the range [0, 1] before converting the jittered HSV image back to the RGB color space.

Value	Meaning
Positive number	Scale the brightness by a random factor from the uniform distribution `[1-Contrast 1+Contrast]`

Value	Meaning
2-element numeric vector of positive numbers	Scale the brightness by a random factor from the uniform distribution within the specified interval. The second element must be larger than or equal to the first element.

Data Types: `single` | `double`

Output Arguments

J — Jittered RGB image

numeric array

Jittered RGB image, returned as a numeric array of the same size and data type as the input image, `I`.

Data Types: `single` | `double` | `uint8` | `uint16`

More About

HSV Color Space

The HSV color space defines the hue, saturation, and value (brightness) for each pixel, respectively, as described in the table.

Attribute	Description
Hue	Value from `0` to `1` that corresponds to the color's position on a color wheel. As hue increases from `0` to `1`, the color transitions from red to orange, yellow, green, cyan, blue, magenta, and finally back to red.
Saturation	Amount of hue or departure from neutral. `0` indicates a grayscale image and `1` indicates maximum saturation.
Value	Maximum value among the red, green, and blue components of a specific color.

See Also

`rgb2hsv` | `hsv2rgb` | `randomAffine2d` | `randomWindow2d` | `centerCropWindow2d`

Topics

"Convert Between RGB and HSV Color Spaces"

Introduced in R2019b

lab2double

Convert L*a*b* color values to `double`

Syntax

```
labD = lab2double(lab)
```

Description

`labD = lab2double(lab)` converts L*a*b* color values to type `double`.

Examples

Convert L*a*b* Color Values to double

This example shows how to convert `uint8` L*a*b* values to `double`.

Create a `uint8` vector specifying the color white in L*a*b* colorspace.

```
w = uint8([255 128 128]);
```

Convert the L*a*b* color value to `double`.

```
lab2double(w)
```

```
ans = 1×3

   100     0     0
```

Input Arguments

`lab` — Color values to convert

m-by-3 numeric matrix | *m*-by-*n*-by-3 numeric array

Color values to convert, specified as a *m*-by-3 numeric matrix of color values (one color per row), or an *m*-by-*n*-by-3 numeric array.

Data Types: `uint8` | `uint16`

Output Arguments

`labD` — Converted color values

numeric array

Converted color values, returned as a numeric array of same size as the input.

Data Types: `double`

Algorithms

The function converts the L*a*b* color values to type `double`. The Image Processing Toolbox software follows the convention that double-precision L*a*b* arrays contain 1976 CIE L*a*b* values. The L*a*b* arrays that are `uint8` or `uint16` follow the convention in the ICC profile specification (ICC.1:2001-4, `www.color.org`) for representing L*a*b* values as unsigned 8-bit or 16-bit integers. The ICC encoding convention is illustrated by these tables.

Value (L*)	uint8 Value	uint16 Value
0.0	0	0
100.0	255	65280
100.0 + (25500/65280)	None	65535

Value (a* or b*)	uint8 Value	uint16 Value
-128.0	0	0
0.0	128	32768
127.0	255	65280
127.0 + (255/256)	None	65535

See Also

`applycform` | `lab2uint8` | `lab2uint16` | `makecform` | `whitepoint` | `xyz2double` | `xyz2uint16`

Introduced in R2006a

lab2rgb

Convert CIE 1976 L*a*b* to RGB

Syntax

```
rgb = lab2rgb(lab)
rgb = lab2rgb(lab,Name,Value)
```

Description

`rgb = lab2rgb(lab)` converts CIE 1976 L*a*b* values to sRGB values.

`rgb = lab2rgb(lab,Name,Value)` specifies additional conversion options, such as the color space of the RGB image, using one or more name-value pair arguments.

Examples

Convert L*a*b* Color to RGB

Convert a color value in the L*a*b* color space to standard RGB color space.

```
lab2rgb([70 5 10])
```

```
ans = 1×3

    0.7359    0.6566    0.6010
```

Convert L*a*b* Color to Adobe RGB

Convert a color value in L*a*b* color space to the Adobe RGB (1998) color space.

```
lab2rgb([70 5 10],'ColorSpace','adobe-rgb-1998')
```

```
ans = 1×3

    0.7086    0.6507    0.5978
```

Convert L*a*b* Color to RGB Specifying Whitepoint

Convert an L*a*b* color value to standard RGB specifying the D50 whitepoint.

```
lab2rgb([70 5 10],'WhitePoint','d50')
```

```
ans = 1×3
```

```
    0.7282    0.6573    0.6007
```

Convert L*a*b* Color to 8-bit-encoded RGB Color

Convert an L*a*b* color value to an 8-bit encoded RGB color value.

```
lab2rgb([70 5 10],'OutputType','uint8')
```

```
ans = 1x3 uint8 row vector

   188   167   153
```

Input Arguments

`lab` — L*a*b* color values

numeric array

L*a*b* color values to convert, specified as a numeric array in one of these formats.

- *c*-by-3 colormap. Each row specifies one L*a*b* color value.
- *m*-by-*n*-by-3 image
- *m*-by-*n*-by-3-by-*p* stack of images

Attribute	Description
*L**	Luminance or brightness of the image. Values are in the range [0, 100], where 0 specifies black and 100 specifies white. As *L** increases, colors become brighter.
*a**	Amount of red or green tones in the image. A large positive *a** value corresponds to red/magenta. A large negative *a** value corresponds to green. Although there is no single range for *a**, values commonly fall in the range [-100, 100] or [-128, 127).
*b**	Amount of yellow or blue tones in the image. A large positive *b** value corresponds to yellow. A large negative *b** value corresponds to blue. Although there is no single range for *b**, values commonly fall in the range [-100, 100] or [-128, 127).

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `lab2rgb([70 5 10],'WhitePoint','d50')`

`ColorSpace` — Color space of the output RGB values

`'srgb'` (default) | `'adobe-rgb-1998'` | `'linear-rgb'`

Color space of the output RGB values, specified as the comma-separated pair consisting of `'ColorSpace'` and `'srgb'`, `'adobe-rgb-1998'`, or `'linear-rgb'`. If you specify `'linear-rgb'`, then `lab2rgb` returns linearized sRGB values.

Data Types: `char`

`WhitePoint` — Reference white point

`'d65'` (default) | `'a'` | `'c'` | `'e'` | `'d50'` | `'d55'` | `'icc'` | 1-by-3 vector

Reference white point, specified as the comma-separated pair consisting of `'WhitePoint'` and a 1-by-3 vector or one of the CIE standard illuminants listed in the table.

Value	White Point
`'a'`	CIE standard illuminant A, `[1.0985, 1.0000, 0.3558]`. Simulates typical, domestic, tungsten-filament lighting with correlated color temperature of 2856 K.
`'c'`	CIE standard illuminant C, `[0.9807, 1.0000, 1.1822]`. Simulates average or north sky daylight with correlated color temperature of 6774 K. Deprecated by CIE.
`'e'`	Equal-energy radiator, `[1.000, 1.000, 1.000]`. Useful as a theoretical reference.
`'d50'`	CIE standard illuminant D50, `[0.9642, 1.0000, 0.8251]`. Simulates warm daylight at sunrise or sunset with correlated color temperature of 5003 K. Also known as horizon light.
`'d55'`	CIE standard illuminant D55, `[0.9568, 1.0000, 0.9214]`. Simulates mid-morning or mid-afternoon daylight with correlated color temperature of 5500 K.
`'d65'`	CIE standard illuminant D65, `[0.9504, 1.0000, 1.0888]`. Simulates noon daylight with correlated color temperature of 6504 K.
`'icc'`	Profile Connection Space (PCS) illuminant used in ICC profiles. Approximation of `[0.9642, 1.000, 0.8249]` using fixed-point, signed, 32-bit numbers with 16 fractional bits. Actual value: `[31595,32768, 27030]/32768`.

Data Types: `single` | `double` | `char`

`OutputType` — Data type of returned RGB values

`'double'` | `'single'` | `'uint8'` | `'uint16'`

Data type of returned RGB values, specified as one of the following values: `'double'`, `'single'`, `'uint8'`, or `'uint16'`. If you do not specify `OutputType`, the output type is the same type as the input.

Data Types: `char`

Output Arguments

`rgb` — Converted RGB color values

numeric array

Converted RGB color values, returned as a numeric array of the same shape as the input. The output data type is the same as the input data type unless you specify the `OutputType` parameter.

Tips

- If you specify the output RGB color space as `'linear-rgb'`, then the output values are linearized sRGB values. If instead you want the output color space to be linearized Adobe RGB (1998), then you can use the `rgb2lin` function.

  For example, to convert CIE 1976 L*a*b* image `LAB` to linearized Adobe RGB (1998) color space, perform the conversion in two steps:

  ```
  RGBadobe = lab2rgb(LAB,'ColorSpace','adobe-rgb-1998');
  RGBlinadobe = rgb2lin(RGBadobe,'ColorSpace','adobe-rgb-1998');
  ```

- `lab2rgb` can return color values that are out of the RGB gamut. A converted RGB color is out of gamut when any of its component values is less than 0 or greater than 1. For more information, see “Determine If L*a*b* Value Is in RGB Gamut”.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `lab2rgb` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- When generating code, all character vector input arguments must be compile-time constants.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, all character vector input arguments must be compile-time constants.

See Also

`rgb2lab` | `xyz2rgb` | `lab2xyz`

Topics

“Understanding Color Spaces and Color Space Conversion”
“Device-Independent Color Spaces”
“Determine If L*a*b* Value Is in RGB Gamut”

Introduced in R2014b

lab2uint16

Convert L*a*b color values to `uint16`

Syntax

```
lab16 = lab2uint16(lab)
```

Description

`lab16 = lab2uint16(lab)` converts L*a*b* color values to type `uint16`.

Examples

Convert L*a*b* Color Values to uint16

This example shows how to convert L*a*b* color values from `double` to `uint16`.

Create a `double` vector specifying the color white in L*a*b* colorspace.

```
w = [100 0 0];
```

Convert the L*a*b* color value to `uint16`.

```
lab2uint16(w)
```

```
ans = 1x3 uint16 row vector

   65280   32768   32768
```

Input Arguments

`lab` — Color values to convert

m-by-3 numeric matrix | *m*-by-*n*-by-3 numeric array

Color values to convert, specified as a *m*-by-3 numeric matrix of color values (one color per row), or an *m*-by-*n*-by-3 numeric array.

Data Types: `double` | `uint8`

Output Arguments

`lab16` — Converted color values

numeric array

Converted color values, returned as a numeric array of same size as the input.

Data Types: `uint16`

Algorithms

The function converts the L*a*b* color values to type `uint16`. The Image Processing Toolbox software follows the convention that double-precision L*a*b* arrays contain 1976 CIE L*a*b* values. The L*a*b* arrays that are `uint8` or `uint16` follow the convention in the ICC profile specification (ICC.1:2001-4, `www.color.org`) for representing L*a*b* values as unsigned 8-bit or 16-bit integers. The ICC encoding convention is illustrated by these tables.

Value (L*)	uint8 Value	uint16 Value
0.0	0	0
100.0	255	65280
100.0 + (25500/65280)	None	65535

Value (a* or b*)	uint8 Value	uint16 Value
-128.0	0	0
0.0	128	32768
127.0	255	65280
127.0 + (255/256)	None	65535

See Also

`applycform` | `lab2double` | `lab2uint8` | `makecform` | `whitepoint` | `xyz2double` | `xyz2uint16`

Introduced before R2006a

lab2uint8

Convert L*a*b color values to `uint8`

Syntax

```
lab8 = lab2uint8(lab)
```

Description

`lab8 = lab2uint8(lab)` converts L*a*b* color values to type `uint8`.

Examples

Convert L*a*b* Color Values to uint8

This example shows how to convert L*a*b* color values from `double` to `uint8`.

Create a `double` vector specifying the color white in L*a*b* colorspace.

```
w = [100 0 0];
```

Convert the L*a*b* color value to `uint8`.

```
lab2uint8(w)
```

```
ans = 1x3 uint8 row vector

   255   128   128
```

Input Arguments

`lab` — Color values to convert

m-by-3 numeric matrix | *m*-by-*n*-by-3 numeric array

Color values to convert, specified as a *m*-by-3 numeric matrix of color values (one color per row), or an *m*-by-*n*-by-3 numeric array.

Data Types: `double` | `uint16`

Output Arguments

`lab8` — Converted color values

numeric array

Converted color values, returned as a numeric array of same size as the input.

Data Types: `uint8`

Algorithms

The function converts the L*a*b* color values to type `uint8`. The Image Processing Toolbox software follows the convention that double-precision L*a*b* arrays contain 1976 CIE L*a*b* values. The L*a*b* arrays that are `uint8` or `uint16` follow the convention in the ICC profile specification (ICC.1:2001-4, `www.color.org`) for representing L*a*b* values as unsigned 8-bit or 16-bit integers. The ICC encoding convention is illustrated by these tables.

Value (L*)	uint8 Value	uint16 Value
0.0	0	0
100.0	255	65280
100.0 + (25500/65280)	None	65535

Value (a* or b*)	uint8 Value	uint16 Value
-128.0	0	0
0.0	128	32768
127.0	255	65280
127.0 + (255/256)	None	65535

See Also

`applycform` | `lab2double` | `lab2uint16` | `makecform` | `whitepoint` | `xyz2double` | `xyz2uint16`

Introduced before R2006a

lab2xyz

Convert CIE 1976 L*a*b* to CIE 1931 XYZ

Syntax

```
xyz = lab2xyz(lab)
xyz = lab2xyz(lab,'WhitePoint',whitePoint)
```

Description

`xyz = lab2xyz(lab)` converts CIE 1976 L*a*b* values to CIE 1931 XYZ values (2° observer).

`xyz = lab2xyz(lab,'WhitePoint',whitePoint)` specifies the reference white point of the illuminant.

Examples

Convert L*a*b* Color to XYZ

Convert an L*a*b* color value to XYZ using the default reference white point, D65.

```
lab2xyz([50 10 -5])
```

```
ans = 1×3

    0.1942    0.1842    0.2282
```

Convert L*a*b* Color to XYZ Specifying Whitepoint

Convert an L*a*b* color value to XYZ specifying the D50 whitepoint.

```
lab2xyz([50 10 -5],'WhitePoint','d50')
```

```
ans = 1×3

    0.1970    0.1842    0.1729
```

Input Arguments

`lab` — L*a*b* color values

numeric array

Color values to convert, specified as a numeric array in one of these formats.

- *c*-by-3 colormap. Each row specifies one L*a*b* color value.
- *m*-by-*n*-by-3 image
- *m*-by-*n*-by-3-by-*p* stack of images

Attribute	Description
*L**	Luminance or brightness of the image. Values are in the range [0, 100], where 0 specifies black and 100 specifies white. As *L** increases, colors become brighter.
*a**	Amount of red or green tones in the image. A large positive *a** value corresponds to red/magenta. A large negative *a** value corresponds to green. Although there is no single range for *a**, values commonly fall in the range [-100, 100] or [-128, 127).
*b**	Amount of yellow or blue tones in the image. A large positive *b** value corresponds to yellow. A large negative *b** value corresponds to blue. Although there is no single range for *b**, values commonly fall in the range [-100, 100] or [-128, 127).

Data Types: `single` | `double`

`whitePoint` — Reference white point
`'d65'` (default) | `'a'` | `'c'` | `'e'` | `'d50'` | `'d55'` | `'icc'` | 1-by-3 vector

Reference white point, specified as a 1-by-3 vector or one of the CIE standard illuminants listed in the table.

Value	White Point
`'a'`	CIE standard illuminant A, `[1.0985, 1.0000, 0.3558]`. Simulates typical, domestic, tungsten-filament lighting with correlated color temperature of 2856 K.
`'c'`	CIE standard illuminant C, `[0.9807, 1.0000, 1.1822]`. Simulates average or north sky daylight with correlated color temperature of 6774 K. Deprecated by CIE.
`'e'`	Equal-energy radiator, `[1.000, 1.000, 1.000]`. Useful as a theoretical reference.
`'d50'`	CIE standard illuminant D50, `[0.9642, 1.0000, 0.8251]`. Simulates warm daylight at sunrise or sunset with correlated color temperature of 5003 K. Also known as horizon light.
`'d55'`	CIE standard illuminant D55, `[0.9568, 1.0000, 0.9214]`. Simulates mid-morning or mid-afternoon daylight with correlated color temperature of 5500 K.
`'d65'`	CIE standard illuminant D65, `[0.9504, 1.0000, 1.0888]`. Simulates noon daylight with correlated color temperature of 6504 K.
`'icc'`	Profile Connection Space (PCS) illuminant used in ICC profiles. Approximation of `[0.9642, 1.000, 0.8249]` using fixed-point, signed, 32-bit numbers with 16 fractional bits. Actual value: `[31595,32768, 27030]/32768`.

Data Types: `single` | `double` | `char`

Output Arguments

xyz — Converted XYZ color values

numeric array

Converted XYZ color values, returned as a numeric array of the same shape and type as the input.

See Also

`rgb2xyz` | `xyz2lab` | `lab2rgb`

Introduced in R2014b

label2idx

Convert label matrix to cell array of linear indices

Syntax

```
pixelIndexList = label2idx(L)
```

Description

`pixelIndexList = label2idx(L)` converts the regions described by the label matrix `L` into linear indices `pixelIndexList`.

Examples

Calculate Pixel Index List for Small Label Matrix

Create a small sample matrix containing three regions.

```
BW = logical([1 1 1 0 0 0 0 0
              1 1 1 0 1 1 0 0
              1 1 1 0 1 1 0 0
              1 1 1 0 0 0 0 0
              1 1 1 0 0 0 1 0
              1 1 1 0 0 0 1 0
              1 1 1 0 0 1 1 0
              1 1 1 0 0 0 0 0]);
```

Create a label matrix from this sample image.

```
L = bwlabel(BW)
```

```
L = 8×8

     1     1     1     0     0     0     0     0
     1     1     1     0     2     2     0     0
     1     1     1     0     2     2     0     0
     1     1     1     0     0     0     0     0
     1     1     1     0     0     0     3     0
     1     1     1     0     0     0     3     0
     1     1     1     0     0     3     3     0
     1     1     1     0     0     0     0     0
```

Get a linear index list of all the pixels in each region. The function returns a cell array with an element for each region it finds in the label matrix.

```
pixelIndexList = label2idx(L)
```

```
pixelIndexList=1×3 cell array
    {24x1 double}    {4x1 double}    {4x1 double}
```

Examine one of the pixel index lists returned. For example, look at the second cell in the returned cell array. It contains the linear indices for all the pixels in the region labeled "2". The upper left corner of the region is pixel BW(2,5), which is the 34th pixel in linear indexing.

```
pixelIndexList{2}
```

```
ans = 4×1

    34
    35
    42
    43
```

Input Arguments

L — Label matrix

numeric array

Label matrix, specified as a numeric array of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

`pixelIndexList` — Linear indices of pixels in regions

1-by-*n* cell array

Linear indices of pixels in regions, returned as a 1-by-*n* cell array. Each element of the output, `pixelIndexList{n}`, is a vector that contains all the linear indices in `L` where `L` is equal to `n`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`label2idx` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`labelmatrix` | `superpixels` | `label2rgb`

Introduced in R2016a

label2rgb

Convert label matrix into RGB image

Syntax

```
RGB = label2rgb(L)
RGB = label2rgb(L,cmap)
RGB = label2rgb(L,cmap,zerocolor)
RGB = label2rgb(L,cmap,zerocolor,order)
RGB = label2rgb( ___ ,'OutputFormat',outputFormat)
```

Description

`RGB = label2rgb(L)` converts a label image, `L` into an RGB color image for the purpose of visualizing the labeled regions. The `label2rgb` function determines the color to assign to each object based on the number of objects in the label matrix. The `label2rgb` function picks colors from the entire range of the colormap.

`RGB = label2rgb(L,cmap)` specifies the colormap `cmap` to be used in the RGB image.

`RGB = label2rgb(L,cmap,zerocolor)` specifies the RGB color of the background elements (pixels labeled `0`).

`RGB = label2rgb(L,cmap,zerocolor,order)` controls how `label2rgb` assigns colors to regions in the label matrix.

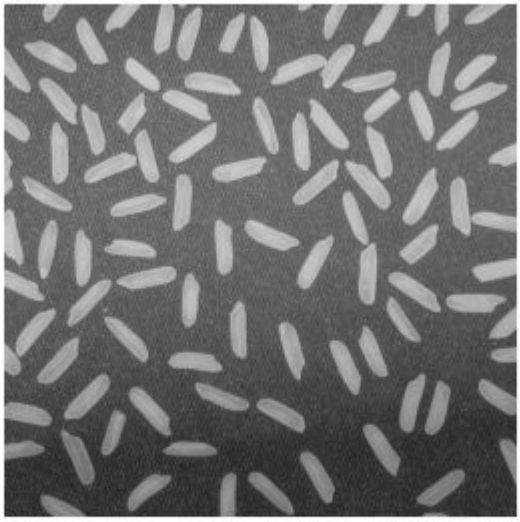
`RGB = label2rgb( ___ ,'OutputFormat',outputFormat)` enables you to specify that the function return a list of unique colors instead of an RGB image.

Examples

Use Color to Highlight Elements in Label Matrix

Read an image and display it.

```
I = imread('rice.png');
imshow(I)
```



Create a label matrix from the image.

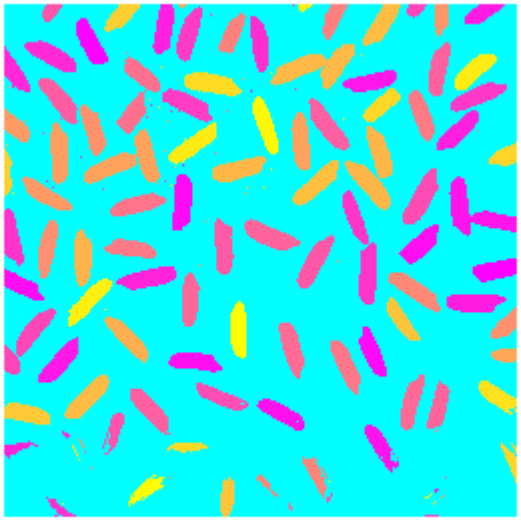
```
BW = imbinarize(I);
CC = bwconncomp(BW);
L = labelmatrix(CC);
```

Convert the label matrix into RGB image, using default settings.

```
RGB = label2rgb(L);
figure
imshow(RGB)
```

Convert the label matrix into an RGB image, specifying optional parameters. This example uses the `'spring'` colormap, sets background pixels to the color cyan, and randomizes how colors are assigned to the labels.

```
RGB2 = label2rgb(L,'spring','c','shuffle');
figure
imshow(RGB2)
```



Input Arguments

`L` — Label image

matrix of nonnegative integers | categorical matrix

Label image of contiguous regions, specified as one of the following.

- A matrix of nonnegative integers. Pixels labeled `0` are the background. Pixels labeled `1` make up one object; pixels labeled `2` make up a second object; and so on. You can get a numeric label image from labeling functions such as `watershed` or `labelmatrix`.
- A categorical matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `categorical`

`cmap` — Colormap

`'jet'` (default) | *c*-by-3 matrix | colormap function | handle

Colormap to be used in the generated color image `RGB`, specified as one of the following.

Value	Description
c-by-3 matrix of data type `double`	Colormap matrix specifying *c* colors, each as an RGB triple. *c* must be greater than or equal to the number of labels, *numlabels*, in label matrix `L`. You can compute the number of labels as `numlabels = max(L(:))`. If *c* is greater than *numlabels*, then `label2rgb` creates the `RGB` image using only the first *numlabels* rows in the matrix.
colormap function	Name of a MATLAB colormap function, such as `'jet'` or `'gray'`. See `colormap` for a list of supported colormaps.
colormap handle	Handle of a colormap function, such as `@jet` or `@gray`.

`zerocolor` — Fill color

`[1 1 1]` (white) (default) | 3-element vector | `'b'` | `'c'` | `'g'`

Fill color, specified as a 3-element vector representing an RGB triple or one of the following color abbreviations for numeric label images. `label2rgb` applies the fill color to the label `0` for numeric label images or the label `<undefined>` for categorical label images.

Value	Color
`'b'`	Blue
`'c'`	Cyan
`'g'`	Green
`'k'`	Black

Value	Color
'm'	Magenta
'r'	Red
'w'	White
'y'	Yellow

order — Color order
'noshuffle' (default) | 'shuffle'

Color order, specified as 'noshuffle' or 'shuffle'. The 'noshuffle' order arranges colormap colors to label matrix regions in numerical order. The 'shuffle' order assigns colormap colors pseudorandomly.

outputFormat — Output format
'image' (default) | 'triplets'

Output format of the RGB data returned in RGB, specified as one of the following.

- 'image' — Return an RGB image. If the size of the input label matrix L is M-by-N, then the size of the output RGB image is M-by-N-by-3.
- 'triplets' — Return a list of RGB colors. The size of the output is a C-by-3 matrix containing an RGB triplet for each of the C labels in the input label matrix.

Output Arguments

RGB — RGB data
numeric matrix

RGB data, returned as an numeric matrix.

Data Types: uint8

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- label2rgb supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- Input label images of data type categorical are not supported.
- When generating code, for best results when using the standard syntax RGB = label2rgb(L,cmap,zerocolor,order):
  - Submit at least two input arguments: the label matrix, L, and the colormap matrix, cmap.
  - cmap must be a *c*-by-3 matrix of data type double. You cannot specify the name of a MATLAB colormap function or a function handle of a colormap function.
  - If you set the background color zerocolor to the same color as one of the regions, then label2rgb will not issue a warning.

- If you supply a value for `order`, then it must be `'noshuffle'`.

Thread-Based Environment
Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

`bwconncomp` | `bwlabel` | `bwlabeln` | `colormap` | `ismember` | `labelmatrix` | `watershed`

Introduced before R2006a

labelmatrix

Create label matrix from `bwconncomp` structure

Syntax

```
L = labelmatrix(CC)
```

Description

A label matrix labels objects or connected components in a binary image with unique integer values. Use a label matrix to visualize distinct objects or connected components.

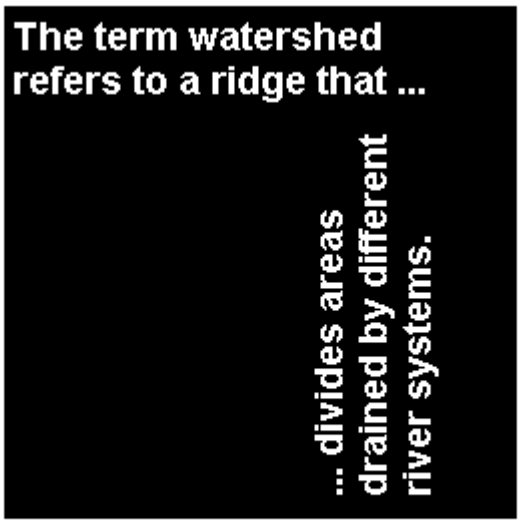
`L = labelmatrix(CC)` creates a label matrix, `L`, from the connected components structure `CC` returned by `bwconncomp`.

Examples

Find and Display Connected Components

Read a binary image into the workspace. Display the image.

```
BW = imread('text.png');
imshow(BW)
```



Calculate the connected components using `bwconncomp`.

```
CC = bwconncomp(BW);
```

Create a label matrix using `labelmatrix`. Each label has a unique numeric index.

```
L = labelmatrix(CC);
```

Find the maximum value of the label matrix. This value indicates the number of detected objects, in this case, 88.

```
numObjects = max(L(:))
```

```
numObjects = uint8
    88
```

Display the label matrix as an image. Because the maximum label value is much smaller than the maximum value of the `uint8` data type, increase the display range of the image to make the labels more distinct.

```
imshow(L,[])
```

It is challenging to see the objects labeled with small label values. Further, it is challenging to differentiate objects with comparable label values. To make it easier to differentiate the different connected components, display the label matrix as an RGB image using `label2rgb` and shuffle the color order of the labels.

```
imshow(label2rgb(L,'jet','k','shuffle'));
```

Input Arguments

`CC` — Connected components

struct

Connected components, specified as a structure with four fields.

Field	Description
`Connectivity`	Connectivity of the connected components (objects)
`ImageSize`	Size of the binary image
`NumObjects`	Number of connected components (objects) in the binary image.
`PixelIdxList`	1-by-`NumObjects` cell array where the *k*-th element in the cell array is a vector containing the linear indices of the pixels in the *k*-th object.

Output Arguments

`L` — Label matrix

matrix of nonnegative integers

Label matrix of contiguous regions, returned as matrix of nonnegative integers. The pixels labeled `0` are the background. The pixels labeled `1` make up one object; the pixels labeled `2` make up a second object; and so on.

The size of `L` is determined by the value of the `CC.ImageSize` field. The class of `L` depends upon the number of contiguous regions. `labelmatrix` uses the smallest class that can represent the number of objects, `CC.NumObjects`, as shown in the table.

Class	Range
`'uint8'`	`CC.NumObjects` ≤ 255
`'uint16'`	256 ≤ `CC.NumObjects` ≤ 65535
`'uint32'`	65536 ≤ `CC.NumObjects` ≤ $2^{32} - 1$
`'double'`	`CC.NumObjects` ≥ 2^{32}

Data Types: `double` | `uint8` | `uint16` | `uint32`

See Also

`bwconncomp` | `label2rgb` | `regionprops`

Topics

“Correct Nonuniform Illumination and Analyze Foreground Objects”
“Label and Measure Connected Components in a Binary Image”

Introduced in R2009a

labeloverlay

Overlay label matrix regions on 2-D image

Syntax

```
B = labeloverlay(A,L)
B = labeloverlay(A,BW)
B = labeloverlay(A,C)
B = labeloverlay( ___ ,Name,Value)
```

Description

`B = labeloverlay(A,L)` fuses the input image, `A`, with a different color for each nonzero label in label matrix `L`. The `labeloverlay` function does not fuse background pixels with a color.

`B = labeloverlay(A,BW)` fuses the input image with a color where mask `BW` is `true`. The `labeloverlay` function does not fuse background pixels (labeled `false`) with a color.

`B = labeloverlay(A,C)` fuses the input image with a different color for each label in categorical matrix `C`. The `labeloverlay` function does not fuse pixels of the `<undefined>` category with a color.

`B = labeloverlay( ___ ,Name,Value)` computes the fused overlay image, `B`, using `Name,Value` pairs to control aspects of the computation.

Examples

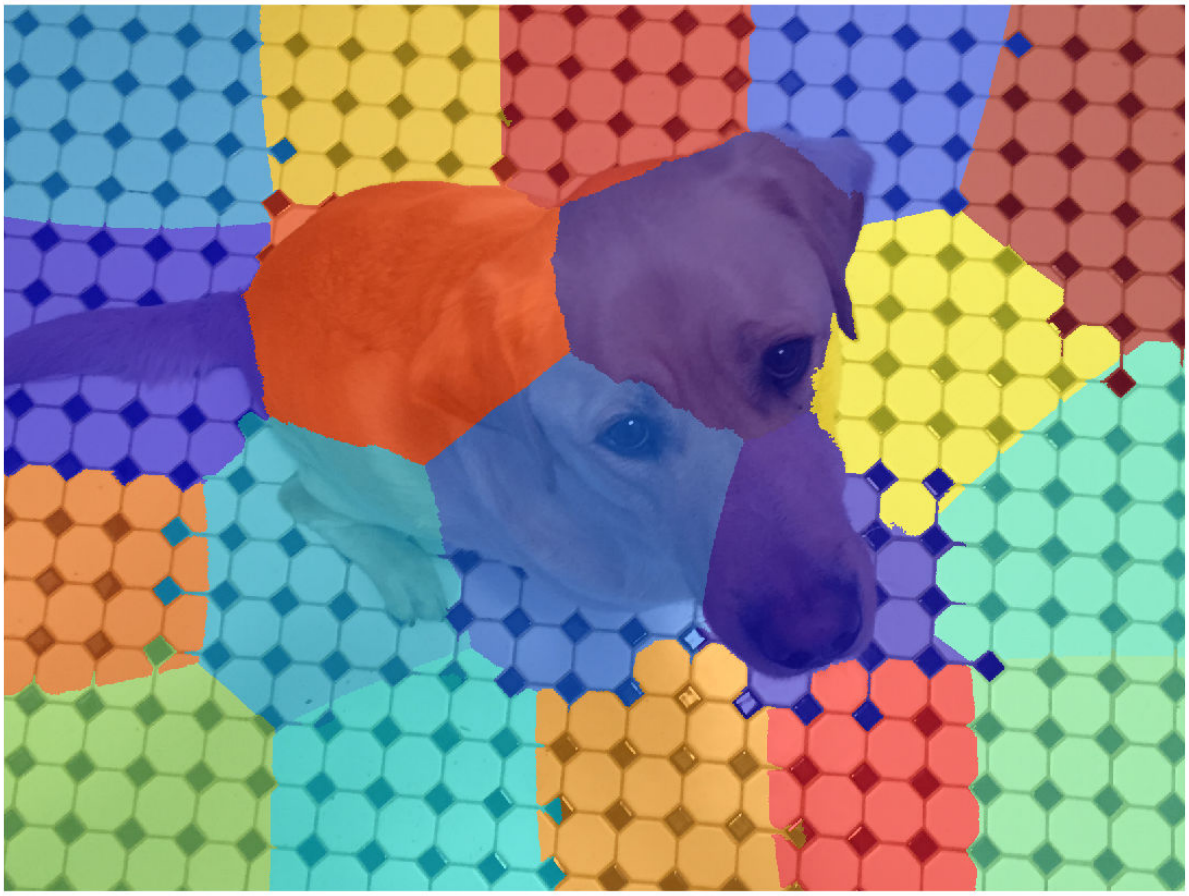
Visualize Segmentation over Color Image

Read an image, then segment it using the `superpixels` function.

```
A = imread('kobi.png');
[L,N] = superpixels(A,20);
```

Fuse the segmentation results with the original image. Display the fused image.

```
B = labeloverlay(A,L);
imshow(B)
```



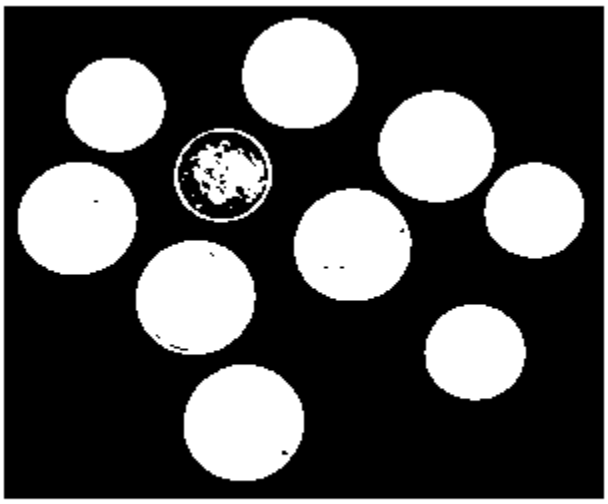
Visualize Binary Mask over Grayscale Image

Read a grayscale image and display it.

```
A = imread('coins.png');
imshow(A)
```

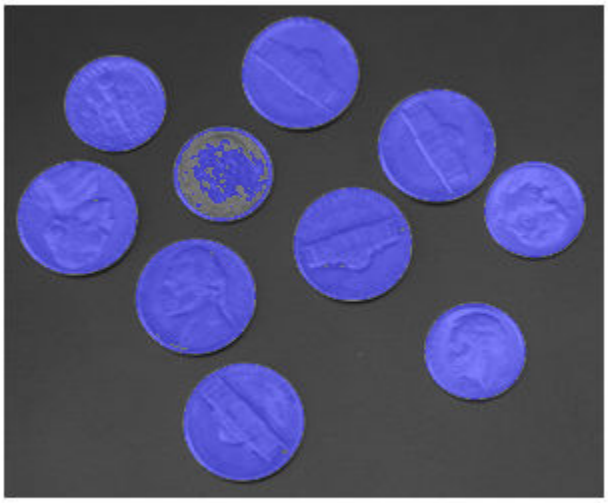
Create a mask using binary thresholding.

```
t = graythresh(A);
BW = imbinarize(A,t);
imshow(BW)
```



Fuse the mask with the original image. Display the fused image.

```
B = labeloverlay(A,BW);
imshow(B)
```



Visualize Categorical Labels over Image

Read a grayscale image.

```
A = imread('coins.png');
```

Create a mask using binary thresholding.

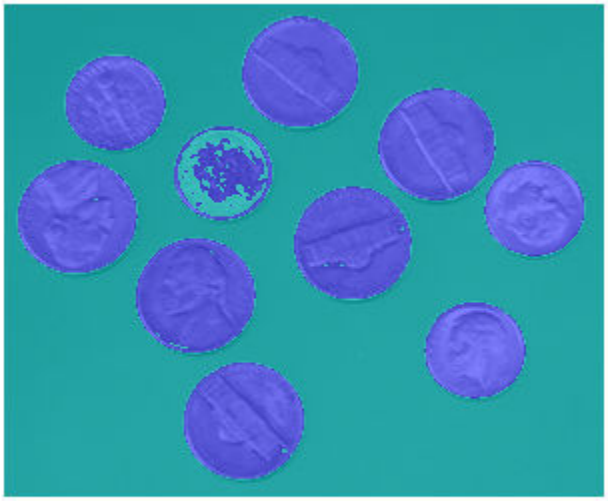
```
BW = imbinarize(A);
```

Create categorical labels based on the image contents.

```
stringArray = repmat("table",size(BW));
stringArray(BW) = "coin";
categoricalSegmentation = categorical(stringArray);
```

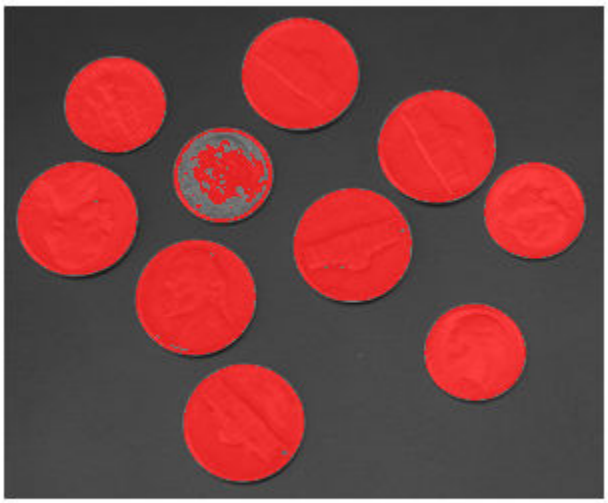
Fuse the categorical labels with the original image. Display the fused image.

```
B = labeloverlay(A,categoricalSegmentation);
imshow(B)
```



Fuse the original image with only one label from the categorical segmentation. Change the colormap, increase the opacity of the label, and display the result.

```
figure
C = labeloverlay(A,categoricalSegmentation,'IncludedLabels',"coin", ...
    'Colormap','autumn','Transparency',0.25);
imshow(C)
```



Input Arguments

`A` — Input image

2-D grayscale image | 2-D color image

Input image, specified as a 2-D grayscale or color image.

Data Types: `single` | `double` | `int8` | `int16` | `uint8` | `uint16`

`L` — Labels

matrix of nonnegative integers

Labels, specified as a matrix of nonnegative integers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`BW` — Mask

logical matrix

Mask, specified as a logical matrix.

Data Types: `logical`

`C` — Category labels

categorical matrix

Category labels, specified as a `categorical` matrix.

Data Types: `categorical`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Colormap','hot'` displays labels in colors from the `'hot'` colormap.

`Colormap` — Colormap

`'jet'` (default) | *l*-by-3 colormap | string | character vector

Colormap, specified as the comma-separated pair consisting of `'Colormap'` and one of these values:

- An *l*-by-3 colormap. RGB triplets in each row of the colormap must be normalized to the range [0, 1]. *l* is the number of labels in label matrix `L`, binary mask `BW`, or categorical matrix `C`.
- A string or character vector corresponding to one of the valid inputs to the `colormap` function. `labeloverlay` permutes the specified colormap so that adjacent labels are more distinct.

Example: `[0.2, 0.1, 0.5; 0.1, 0.5, 0.8]`

Example: `'hot'`

Data Types: `single` | `double` | `char` | `string`

`IncludedLabels` — Labels to display

integer | vector of integers | string | vector of strings

Labels to display in the fused image, specified as the comma-separated pair consisting of `'IncludedLabels'` and one of the following:

- An integer, or vector of integers, in the range `[0, max(L(:))]`. By default, `labeloverlay` displays all nonzero labels.
- A string, or vector of strings, corresponding to labels in categorical matrix `C`. By default, `labeloverlay` displays all defined categorical labels.

Any label not included in the vector is considered the background. For example, in the vector `[1,3,4]`, the value 2 would be considered the background, if it existed as a label.

Example: `[1,3,4]`

Example: `["flower","stem"]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `string`

`Transparency` — Transparency
`0.5` (default) | number in the range [0, 1]

Transparency of displayed labels, specified as the comma-separated pair consisting of `'Transparency'` and a number in the range [0, 1].

- A value of `0` makes the colored labels completely opaque.
- A value of `1` makes the colored labels completely transparent.

Data Types: `single` | `double`

Output Arguments

`B` — Fused image
numeric matrix

Fused image, returned as a numeric matrix of the same size as `A`.

Data Types: `uint8`

See Also

`superpixels` | `imoverlay` | `imshowpair`

Topics
“Getting Started with Semantic Segmentation Using Deep Learning” (Computer Vision Toolbox)

Introduced in R2017b

labelvolshow

Display labeled volume

Description

A `labelvolshow` object displays labeled volumetric data and enables you to modify the appearance of the display. You can embed the intensity volume with the labeled volume and display both volumes at once.

Creation

Syntax

```
labelvolshow(L)
labelvolshow(L,V)
labelvolshow( ___ ,Name,Value)
h = labelvolshow( ___ )
```

Description

`labelvolshow(L)` displays 3-D labeled volume `L` in a figure.

`labelvolshow(L,V)` displays 3-D labeled volume `L` and 3-D intensity volume `V` in a figure. `L` and `V` must be the same size.

`labelvolshow( ___ ,Name,Value)` uses one or more name-value pairs to set “Properties” on page 1-2238 that control visualization of the volumes. Enclose each property name in quotes.

Example: `labelvolshow(L,V,'BackgroundColor','w','VolumeThreshold',0.2)` displays 3-D labeled volume `L` and grayscale volume `V` in a figure with a white background color. All pixels of `V` that have a value less than `0.2` are fully transparent.

`h = labelvolshow( ___ )` returns a `labelvolshow` object, `h`, with properties that can be used to control visualization of the volumes. Use input arguments from any of the previous syntaxes.

Input Arguments

`L` — Labeled volume

3-D numeric array

Labeled volume, specified as a 3-D numeric array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `categorical`

`V` — Intensity volume

3-D numeric array

Intensity volume, specified as a 3-D numeric array of the same size as the labeled volume, `L`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

Properties

BackgroundColor — Background color

`[0.3 0.75 0.93]` (default) | RGB triplet | color name | short color name

Background color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'BackgroundColor','r'`

Example: `'BackgroundColor','green'`

Example: `'BackgroundColor',[0 0.4470 0.7410]`

CameraPosition — Location of camera

`[4 4 2.5]` (default) | 3-element vector

Location of the camera, or the viewpoint, specified as a 3-element vector of the form `[x y z]`. This vector defines the axes coordinates of the camera location, which is the point from which you view the axes. The camera is oriented along the view axis, which is a straight line that connects the camera position and the camera target. Changing the `CameraPosition` property changes the point

from which you view the volume. For an illustration, see “Camera Graphics Terminology”. Interactively rotating the volume modifies the value of this property.

`CameraUpVector` — Vector defining upwards direction
`[0 0 1]` (default) | 3-element vector

Vector defining upwards direction, specified as a 3-element vector of the form `[x y z]`. By default, `labelvolshow` defines the *z*-axis as the up direction (`[0 0 1]`). For an illustration, see “Camera Graphics Terminology”. Interactively rotating the volume modifies the value of this property.

`CameraTarget` — Point used as camera target
`[0 0 0]` (default) | 3-element vector

Point used as the camera target, specified as a 3-element vector of the form `[x y z]`. The camera is oriented along the view axis, which is a straight line that connects the camera position and the camera target. For an illustration, see “Camera Graphics Terminology”.

`CameraViewAngle` — Field of view
15 (default) | numeric scalar

Field of view, specified as a scalar angle in the range [0, 180). The greater the angle, the larger the field of view. Also, with bigger angles, objects appear smaller in the scene. For an illustration, see “Camera Graphics Terminology”.

`InteractionsEnabled` — Volume is interactive
`true` (default) | `false`

Volume is interactive, specified as `true` (`1`) or `false` (`0`). When `true` (default), you can zoom in and out on the labeled volume using the mouse scroll wheel, and rotate the volume by clicking and dragging. Rotation and zoom are performed about the value specified by `CameraTarget`. When this value is `false`, you cannot interact with the volume.

`LabelColor` — Label colors
numLabels-by-3 numeric matrix

Label colors, specified as a *numLabels*-by-3 numeric matrix with values in the range [0, 1]. *numLabels* is the number of labels in the labeled volume. By default, `labelvolshow` specifies the label colors using a random colormap.

`LabelOpacity` — Label opacity
numLabels-by-1 numeric vector

Label opacity, specified as a *numLabels*-by-1 numeric vector with values in the range [0, 1]. *numLabels* is the number of labels in the labeled volume. By default, labels are opaque (`1`) for all labels except label `0`. `LabelOpacity` is not supported when embedding volumes together.

`LabelsPresent` — Label values
numLabels-by-1 numeric vector

This property is read-only.

Label values, specified as a *numLabels*-by-1 numeric vector. *numLabels* is the number of labels in the labeled volume.

`LabelVisibility` — Label visibility
numLabels-by-1 logical vector

Label visibility, specified as a *numLabels*-by-1 logical vector. *numLabels* is the number of labels in the labeled volume. By default, all labels are visible (`true`) for all labels except label `0`.

`Parent` — Parent of `labelvolshow` object
`gcf` (default) | uipanel | figure

Parent of the `labelvolshow` object, specified as a handle to a `uipanel` or `figure`. If you do not specify a parent, the parent of the `labelvolshow` object is `gcf`.

`ScaleFactors` — Scale factors used to rescale volume
`[1 1 1]` (default) | 1-by-3 vector of positive numbers

Scale factors used to rescale volumes, specified as a 1-by-3 vector of positive numbers. The values in the array correspond to the scale factor applied in the *x*-, *y*-, and *z*-direction.

`ShowIntensityVolume` — Display intensity volume
`true` | `false`

Display intensity volume, specified as `true` (`1`) or `false` (`0`). When the value is `true`, the function displays both the labeled volume and the intensity volume. When the value is `false`, the function only displays the labeled volume. The default is `true` when the `labelvolshow` object contains both a labeled volume and an intensity volume. The default is `false` when the object contains only a labeled volume.

`VolumeOpacity` — Volume opacity
`0.5` (default) | number in the range [0, 1]

Volume opacity, specified as a number in the range [0, 1]. This value defines the opacity of volume data when both labeled and intensity volumes are embedded together. All of the embedded volume intensities above the `VolumeThreshold` value have the opacity of `VolumeOpacity`.

`VolumeThreshold` — Threshold of volume intensities
`0.4` (default) | number in the range [0, 1]

Threshold of volume intensities, specified as a normalized number in the range [0, 1]. All of the volume intensities below this threshold value have an opacity of `0`.

Object Functions

setVolume Set new labelvolshow object

Examples

View Labeled Volume With and Without Intensity Volume

Read a grayscale image of a brain MRI. The image is stored in the workspace variable `vol`.

```
load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled', ...
    'images','vol_001.mat'));
```

Read the corresponding labeled image into the workspace variable `label`.

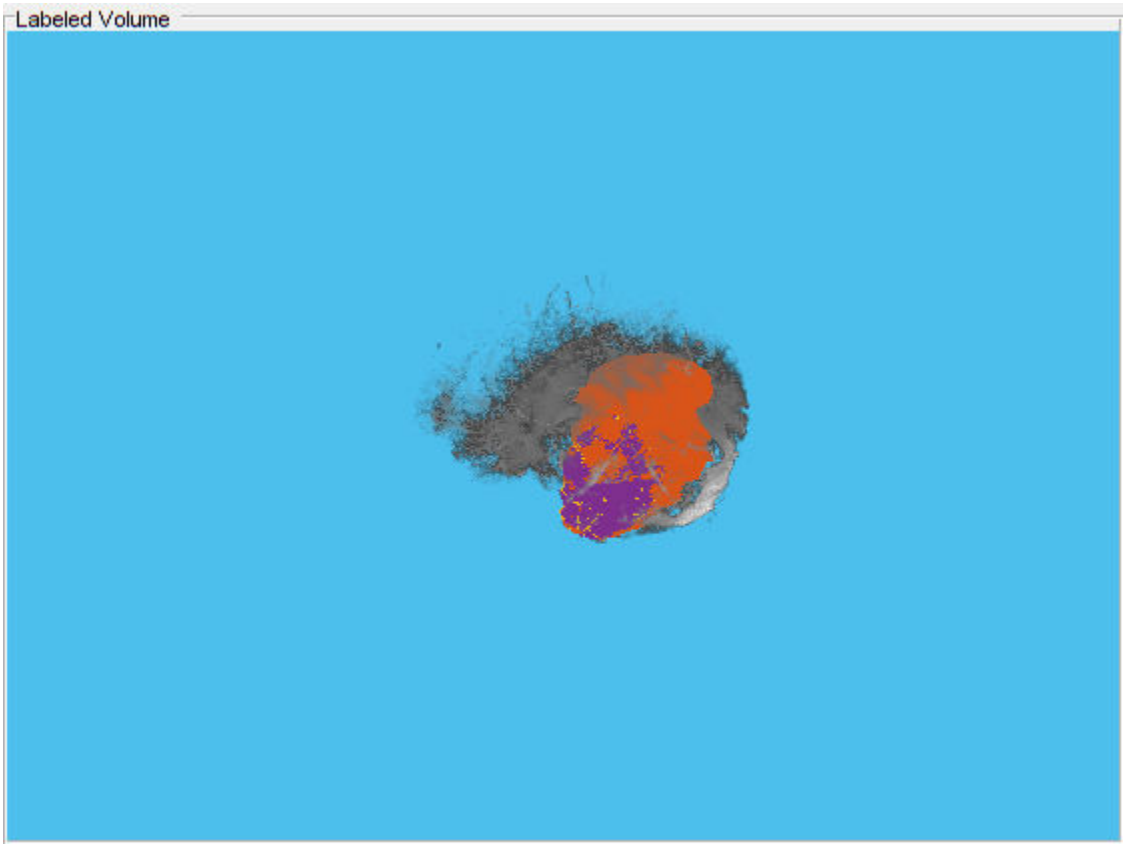
```
load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled', ...
    'labels','label_001.mat'));
```

Customize the display panel.

```
ViewPnl = uipanel(figure,'Title','Labeled Volume');
```

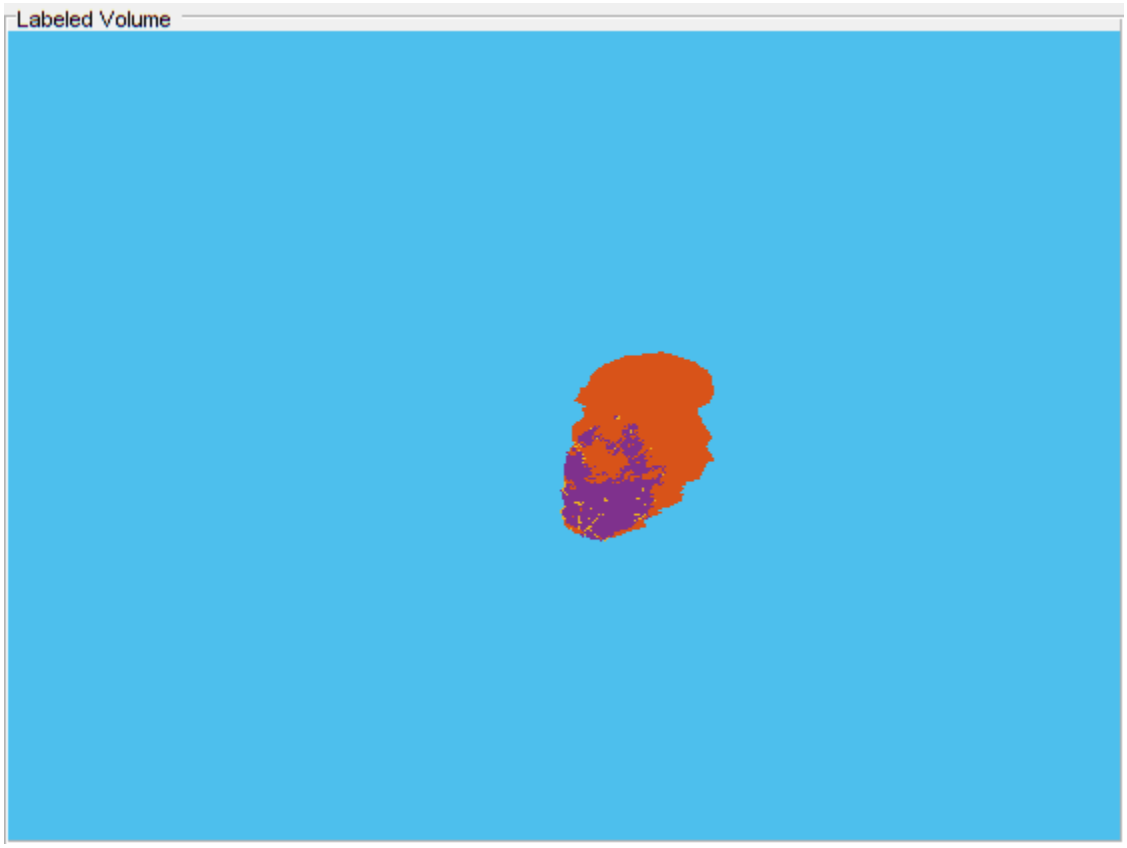
View the labeled volume and the intensity volume.

```
h = labelvolshow(label,vol,'Parent',ViewPnl);
```



Hide the intensity volume. Only the labels appear.

```
h.ShowIntensityVolume = false;
```



View Labeled Volume and Change Color and Opacity

Read a grayscale volume of a brain MRI. The image is stored in the workspace variable `vol`.

```
load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled', ...
    'images','vol_001.mat'));
```

Read the corresponding labeled volume into the workspace variable `label`. The volume has three labels, excluding the background label `0`.

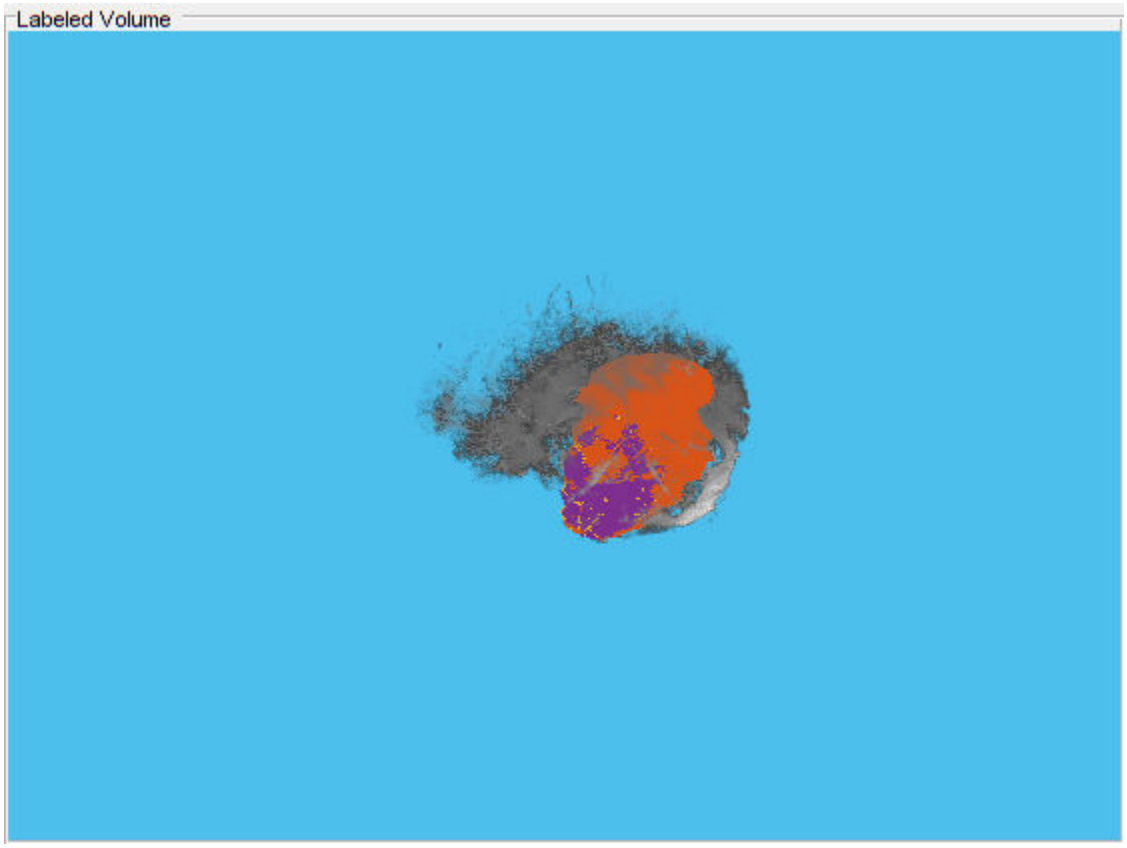
```
load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled', ...
    'labels','label_001.mat'));
```

Customize the display panel.

```
ViewPnl = uipanel(figure,'Title','Labeled Volume');
```

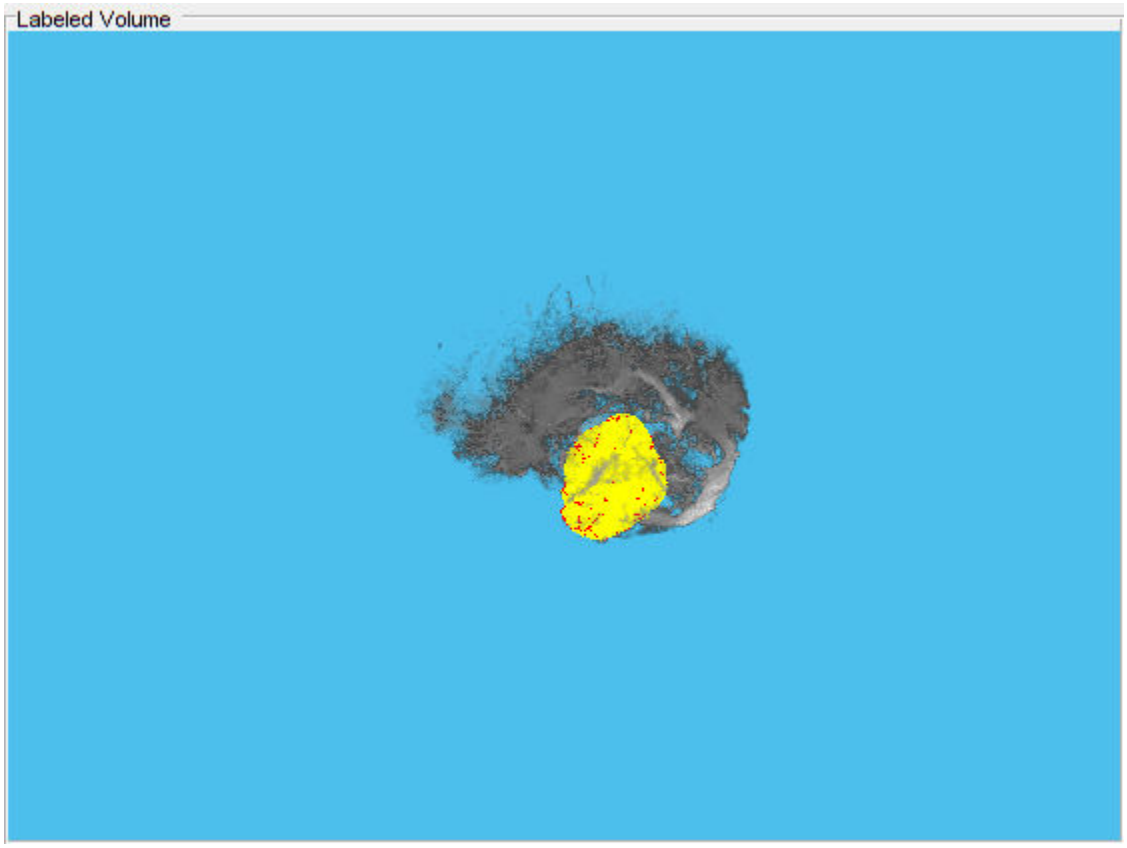
Display the labeled volume.

```
h = labelvolshow(label,vol,'Parent',ViewPnl);
```



Make the first non-background label (the second label) fully transparent. Change the color of the second non-background label to red and the third non-background label to yellow.

```
h.LabelOpacity(2) = 0;
h.LabelColor(3,:) = [1 0 0];
h.LabelColor(4,:) = [1 1 0];
```



See Also

`volshow` | **Volume Viewer** | `slice` | `modefilt`

Introduced in R2019a

setVolume

Set new `labelvolshow` object

Syntax

```
setVolume(hLabelVol,L)
setVolume(hLabelVol,L,V)
```

Description

`setVolume(hLabelVol,L)` updates the `labelvolshow` object `hLabelVol` with a new labeled volume `L`. `setVolume` preserves the current viewpoint and other visualization settings remain unchanged, but the label properties are set to their respective defaults.

`setVolume(hLabelVol,L,V)` updates the `labelvolshow` object `hLabelVol` with a new labeled volume `L` and a new intensity volume `V`.

Examples

Change Labeled Volume in labelvolshow Object

Load an intensity volume and an associated labeled volume into the workspace.

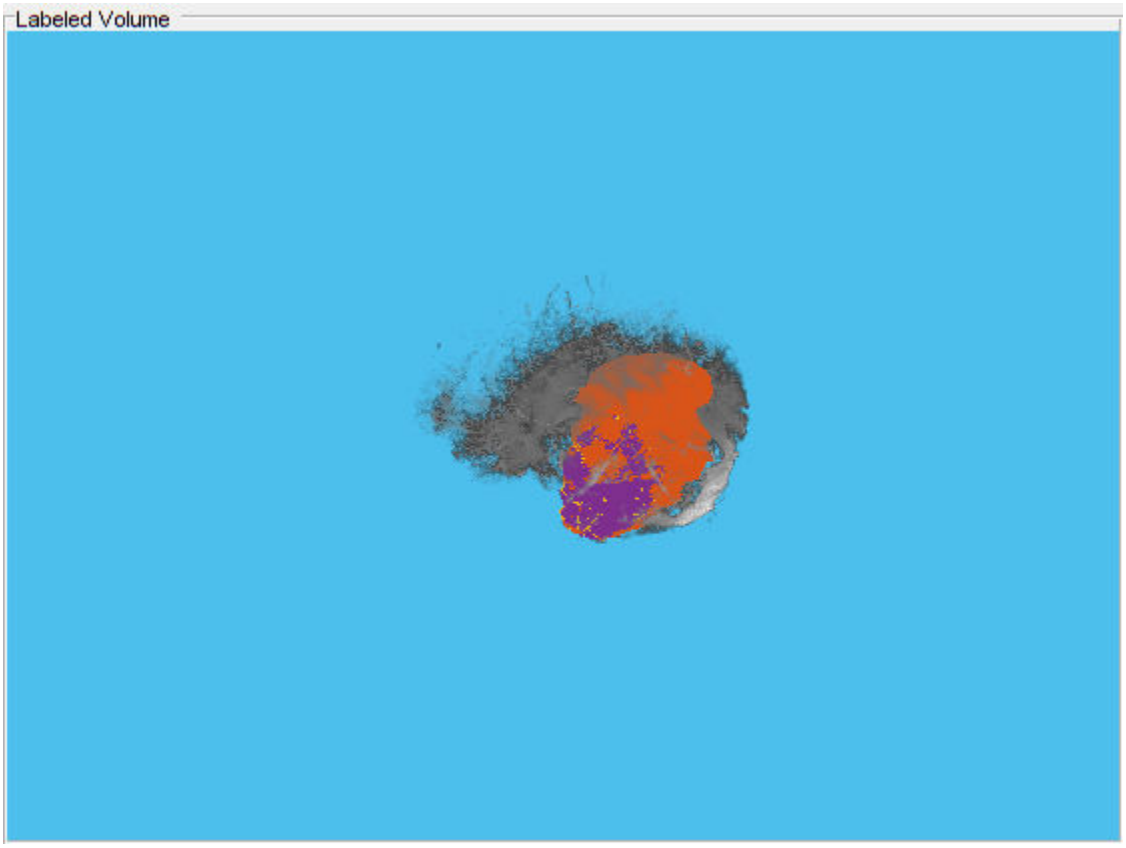
```
load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled','images','vol_001.mat'));
load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled','labels','label_001.mat'));
```

Customize the display panel.

```
ViewPnl = uipanel(figure,'Title','Labeled Volume');
```

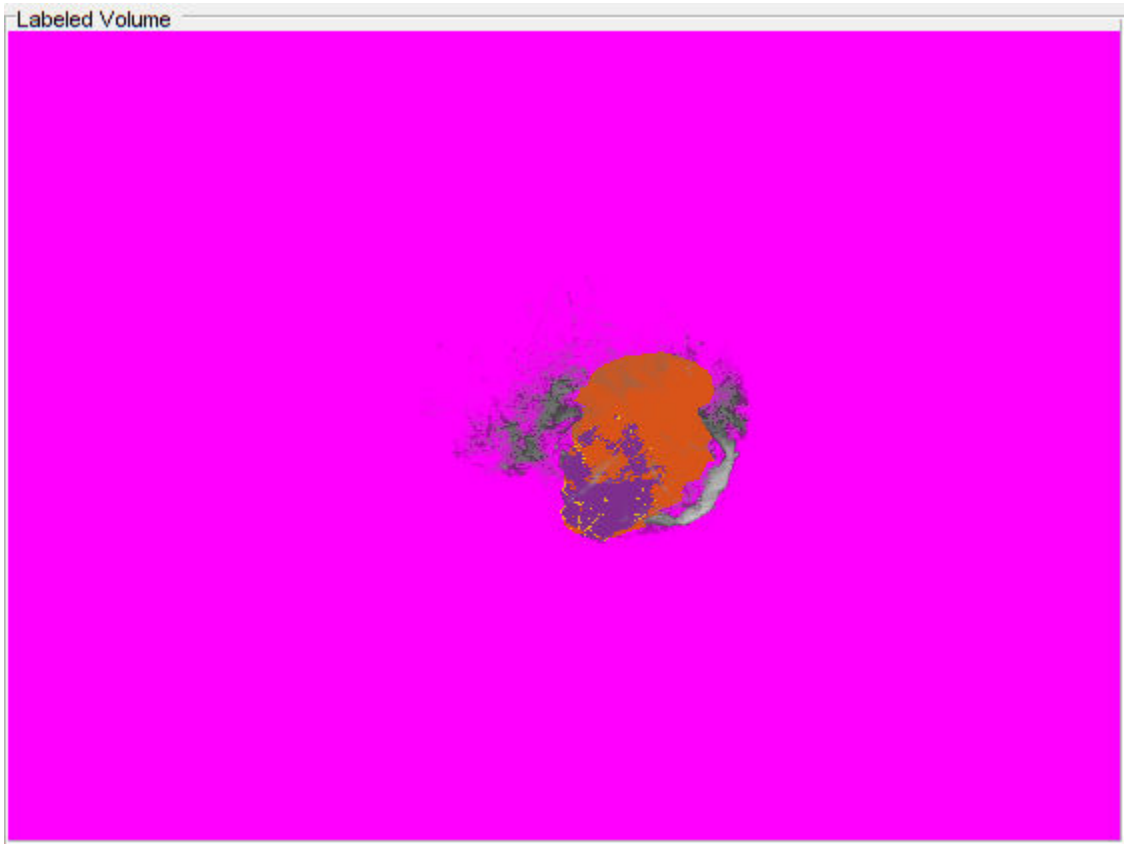
Display the labeled volume along with an intensity volume.

```
hVol = labelvolshow(label,vol,'Parent',ViewPnl);
```



Change the background color to magenta and decrease the opacity of the intensity volume.

```
hVol.VolumeOpacity = 0.2;
hVol.BackgroundColor = 'magenta';
```

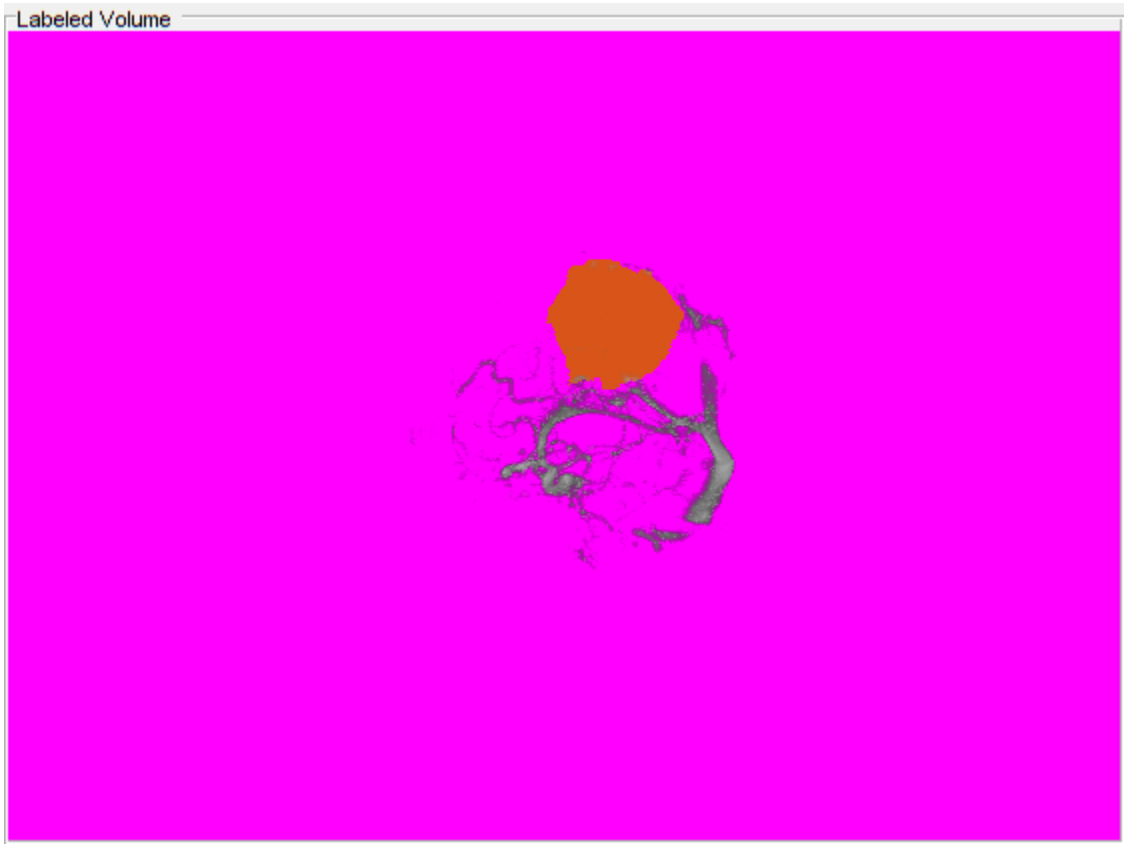


Load another intensity volume and an associated labeled volume into the workspace.

```
im = load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled','images','vol_002.mat'));
data = load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled','labels','label_002.mat'));

newIntensityVol = im.vol;
newLabelVol = data.label;
```

Change the volume in the `labelvolshow` object `hVol`. Note how `labelvolshow` preserves the rendering settings of the background color and intensity volume transparency.

```
setVolume(hVol,newLabelVol,newIntensityVol)
```



Input Arguments

`hLabelVol` — Labeled volume object
`labelvolshow` object

Labeled volume object, specified as a `labelvolshow` object.

`L` — Labeled volumetric data
3-D labeled volume

Labeled volumetric data, specified as a 3-D labeled volume.

`V` — Volumetric data
3-D grayscale volume

Volumetric data, specified as a 3-D grayscale volume.

See Also
`labelvolshow`

Introduced in R2019a

depthToSpace2dLayer

Depth to space layer

Description

A 2-D depth to space layer permutes data from the depth dimension into blocks of 2-D spatial data.

Given an input feature map of size [*H W C***height***width*] and blocks of size [*height width*], the output feature map size is [*H***height W***width C*].

This object requires Deep Learning Toolbox.

Creation

Syntax

```
layer = depthToSpace2dLayer(blockSize)
layer = depthToSpace2dLayer(blockSize,Name,Value)
```

Description

`layer = depthToSpace2dLayer(blockSize)` creates a 2-D depth to space layer, specifying the block size to rearrange the input activation. The `blockSize` input sets the BlockSize property.

`layer = depthToSpace2dLayer(blockSize,Name,Value)` uses name-value pairs to set the `Mode` and `Name` properties. You can specify multiple name-value pairs. Enclose each property name in quotes.

Example: `depthToSpace2dLayer(blockSize,"Mode","CRD")` creates a 2-D depth to space layer that orders data by column, row, and then depth.

Properties

`BlockSize` — Block size to reorder input activation

vector of two positive integers

Block size to reorder the input activation, specified as a vector of two positive integers `[h w]`, where `h` is the height and `w` is the width. When creating the layer, you can specify `BlockSize` as a scalar to use the same value for both dimensions.

Example: `[2 1]` specifies blocks of height 2 and width 1.

`Mode` — Order of rearranged dimensions

`"drc"` (default) | `"crd"`

Order of rearranged dimensions from the input data, specified as `"dcr"` or `"crd"`. When you specify `"dcr"`, the layer orders data by depth, column, and then row. When you specify `"crd"`, the layer orders data by column, row, and then depth.

Data Types: `char` | `string`

`Name` — Layer name
`''` (default) | character vector | string scalar

Layer name, specified as a character vector or a string scalar. For `Layer` array input, the `trainNetwork`, `assembleNetwork`, `layerGraph`, and `dlnetwork` functions automatically assign names to layers with name `''`.

Data Types: `char` | `string`

`NumInputs` — Number of inputs
`1` (default)

This property is read-only.

Number of inputs of the layer. This layer accepts a single input only.

Data Types: `double`

`InputNames` — Input names
`{'in'}` (default)

This property is read-only.

Input names of the layer. This layer accepts a single input only.

Data Types: `cell`

`NumOutputs` — Number of outputs
`1` (default)

This property is read-only.

Number of outputs of the layer. This layer has a single output only.

Data Types: `double`

`OutputNames` — Output names
`{'out'}` (default)

This property is read-only.

Output names of the layer. This layer has a single output only.

Data Types: `cell`

Examples

Create 2-D Depth to Space Layer

Specify the block size for reordering input activations.

```
blockSize = [2 2];
```

Create a 2-D depth to space layer that orders data by column, row, and then depth.

```
layer = depthToSpace2dLayer(blockSize,"Mode","crd","Name","depthToSpaceLayer")

layer =
  DepthToSpace2DLayer with properties:

         Name: 'depthToSpaceLayer'
    BlockSize: [2 2]
         Mode: "crd"

   Learnable Parameters
    No properties.

   State Parameters
    No properties.

  Show all properties
```

Extended Capabilities

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

To generate CUDA or C++ code by using GPU Coder™, you must first construct and train a deep neural network. Once the network is trained and evaluated, you can configure the code generator to generate code and deploy the convolutional neural network on platforms that use NVIDIA or ARM® GPU processors. For more information, see “Deep Learning with GPU Coder” (GPU Coder).

For this layer, you can generate code that takes advantage of the NVIDIA CUDA deep neural network library (cuDNN), NVIDIA TensorRT high performance inference library, or the ARM `Compute Library` for Mali GPU.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

`SpaceToDepthLayer` | `depthToSpace` | `spaceToDepth`

Topics

“List of Deep Learning Layers” (Deep Learning Toolbox)
“Deep Learning in MATLAB” (Deep Learning Toolbox)

Introduced in R2021a

resize2dLayer

2-D resize layer

Description

A 2-D resize layer resizes 2-D input by a scale factor, to a specified height and width, or to the size of a reference input feature map. Use of this layer requires Deep Learning Toolbox.

Creation

Syntax

```
layer = resize2dLayer('Scale',scale)
layer = resize2dLayer('OutputSize',outputSize)
layer = resize2dLayer('EnableReferenceInput',tf)
layer = resize2dLayer(___,Name,Value)
```

Description

`layer = resize2dLayer('Scale',scale)` creates a 2-D resize layer and sets the Scale property as the scale factor specified by `scale`.

`layer = resize2dLayer('OutputSize',outputSize)` creates a 2-D resize layer and sets the OutputSize property with the height and width specified by `outputSize`.

`layer = resize2dLayer('EnableReferenceInput',tf)` creates a 2-D resize layer and sets the EnableReferenceInput property with the boolean specified by `tf`. When you specify the value as `true`, the layer adds an additional input that accepts a reference feature map and resizes the input to the size of the reference feature map.

`layer = resize2dLayer(___,Name,Value)` sets the optional Method, GeometricTransformMode, NearestRoundingMode, and Name properties using name-value pair arguments. You can specify multiple name-value pair arguments. Enclose each property name in single quotes.

Example: `layer = resize2dLayer('OutputSize',[128 128],'Method','bilinear')` creates a 2-D resize layer that resizes input to 128-by-128 pixels using bilinear interpolation

Properties

Resize

`Scale` — Scale factor to resize input

2-element row vector of positive numbers

Scale factor to resize input, specified as 2-element row vector of positive numbers. The scale factors are for the row and column dimensions, respectively. When creating the layer, you can specify `Scale` as a scalar to use the same value for both dimensions.

`OutputSize` — Output size of resized input
2-element row vector of positive integers

Output size of resized input, specified as a 2-element row vector of positive integers of the form [*nrows ncols*]. You can specify one element as NaN, in which case the layer computes the value automatically to preserve the aspect ratio of the input.

`EnableReferenceInput` — Add reference feature map as input
`false` or `0` | `true` or `1`

Add reference feature map as input to the layer, specified as a numeric or logical `0` (`false`) or `1` (`true`). When you specify the value as `true`, the layer resizes the height and width of the input to match the height and width of the reference feature map. The resizing operation does not change the number of channels of the input.

When you enable a reference feature map, the inputs to the layer have the names `'in1'` and `'ref'`, where `'ref'` is the name of the reference feature map. Use the input names when connecting or disconnecting the layer by using `connectLayers` or `disconnectLayers`.

`Method` — Interpolation method
`'nearest'` (default) | `'bilinear'`

Interpolation method, specified as `'nearest'` for nearest neighbor interpolation or `'bilinear'` for bilinear interpolation.

`GeometricTransformMode` — Geometric transformation mode
`'half-pixel'` (default) | `'asymmetric'`

Geometric transformation mode to map points from input space to output space, specified as `'half-pixel'` or `'asymmetric'`.

`NearestRoundingMode` — Rounding mode for nearest neighbor interpolation
`'round'` (default) | `'floor'` | `'onnx-10'`

Rounding mode for nearest neighbor interpolation, specified as one of the following.

- `'round'` — use the same rounding behavior as the MATLAB `round` function.
- `'floor'` — use the same rounding behavior as the MATLAB `floor` function.
- `'onnx-10'` — reproduce the resizing behavior of the ONNX (Open Neural Network Exchange) opset 10 Resize operator.

This property is valid when the `Method` property is `'nearest'`.

Layer

`Name` — Layer name
`''` (default) | character vector | string scalar

Layer name, specified as a character vector or a string scalar. For `Layer` array input, the `trainNetwork`, `assembleNetwork`, `layerGraph`, and `dlnetwork` functions automatically assign names to layers with name `''`.

Data Types: `char` | `string`

`NumInputs` — Number of inputs
1 (default) | 2

Number of inputs of the layer, specified as 1 when the EnableReferenceInput property is `false` or 2 when the `EnableReferenceInput` property is `true`.

Data Types: `double`

`InputNames` — Input names
`{'in'}` (default) | `{'in','ref'}`

Input names of the layer, specified as `{'in'}` when the EnableReferenceInput property is `false` or `{'in','ref'}` when the `EnableReferenceInput` property is `true`.

Data Types: `cell`

`NumOutputs` — Number of outputs
1 (default)

This property is read-only.

Number of outputs of the layer. This layer has a single output only.

Data Types: `double`

`OutputNames` — Output names
`{'out'}` (default)

This property is read-only.

Output names of the layer. This layer has a single output only.

Data Types: `cell`

Examples

Create 2-D Resize Layer Specifying Scale Factor

Create a 2-D resize layer with a horizontal scale factor of 2 and a vertical scale factor of 4.

```
layer = resize2dLayer('Scale',[2 4])
```

```
layer =
  Resize2DLayer with properties:

                      Name: ''
                     Scale: [2 4]
                OutputSize: []
      EnableReferenceInput: 0
                    Method: 'nearest'
    GeometricTransformMode: 'half-pixel'
       NearestRoundingMode: 'round'

   Learnable Parameters
    No properties.
```

```
   State Parameters
    No properties.

  Show all properties
```

Create 2-D Resize Layer Specifying Output Size

Create a 2-D resize layer named 'resize224' with an output size of [224 224].

```
layer = resize2dLayer('OutputSize',[224 224],'Name','resize224')
```

```
layer =
  Resize2DLayer with properties:

                      Name: 'resize224'
                     Scale: []
                OutputSize: [224 224]
      EnableReferenceInput: 0
                    Method: 'nearest'
    GeometricTransformMode: 'half-pixel'
       NearestRoundingMode: 'round'

   Learnable Parameters
    No properties.

   State Parameters
    No properties.

  Show all properties
```

Create 2-D Resize Layer with Reference Port

Create an array of layers that includes a 2-D resize layer that accepts a reference input feature map.

```
layers = [
    imageInputLayer([32 32 3],'Name','image')
    resize2dLayer('EnableReferenceInput',true,'Name','resize')]
```

```
layers =
  2x1 Layer array with layers:

     1   'image'    Image Input   32x32x3 images with 'zerocenter' normalization
     2   'resize'   Resize        nnet.cnn.layer.Resize2DLayer
```

Create a `layerGraph`. The first input of the 2-D resize layer is automatically connected to the output of the image input layer.

```
lgraph = layerGraph(layers);
```

Connect the `'ref'` input of the 2-D resize layer to the output of a layer that provides a reference feature map by using the `connectLayers` function. This example shows a trivial connection in which the `'ref'` input is also connected to the output of the image input layer.

```
lgraph = connectLayers(lgraph,'image','resize/ref');
```

Create 2-D Resize Layer with Bilinear Interpolation

Create a 2-D resize layer named 'rescale0.5' with a uniform scale factor of 0.5. Specify the interpolation method as bilinear interpolation.

```
layer = resize2dLayer('Scale',0.5,'Method','bilinear','Name','rescale0.5')
```

```
layer =
  Resize2DLayer with properties:

                      Name: 'rescale0.5'
                     Scale: [0.5000 0.5000]
                OutputSize: []
       EnableReferenceInput: 0
                    Method: 'bilinear'
     GeometricTransformMode: 'half-pixel'
        NearestRoundingMode: 'round'

   Learnable Parameters
    No properties.

   State Parameters
    No properties.

  Show all properties
```

References

[1] *Open Neural Network Exchange*. https://github.com/onnx/.

[2] *ONNX*. https://onnx.ai/.

Extended Capabilities

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

See Also

`resize3dLayer` | `dlresize` | `averagePooling2dLayer` | `transposedConv2dLayer` | `importONNXLayers`

Topics

“Deep Learning in MATLAB” (Deep Learning Toolbox)
“Specify Layers of Convolutional Neural Network” (Deep Learning Toolbox)

“List of Deep Learning Layers” (Deep Learning Toolbox)

Introduced in R2020b

resize3dLayer

3-D resize layer

Description

A 3-D resize layer resizes 3-D input by a scale factor, to a specified height, width, and depth, or to the size of a reference input feature map. Use of this layer requires Deep Learning Toolbox.

Creation

Syntax

```
layer = resize3dLayer('Scale',scale)
layer = resize3dLayer('OutputSize',outputSize)
layer = resize3dLayer('EnableReferenceInput',tf)
layer = resize3dLayer(___,Name,Value)
```

Description

`layer = resize3dLayer('Scale',scale)` creates a 3-D resize layer and sets the Scale property as the scale factor specified by `scale`.

`layer = resize3dLayer('OutputSize',outputSize)` creates a 3-D resize layer and sets the OutputSize property with the height, width, and depth specified by `outputSize`.

`layer = resize3dLayer('EnableReferenceInput',tf)` creates a 3-D resize layer and sets the EnableReferenceInput property with the boolean specified by `tf`. When you specify the value as `true`, the layer adds an additional input that accepts a reference feature map and resizes the input to the size of the reference feature map.

`layer = resize3dLayer(___,Name,Value)` also sets the optional Method, GeometricTransformMode, NearestRoundingMode, and Name properties using name-value pair arguments. You can specify multiple name-value pair arguments. Enclose each property name in single quotes.

Example: `layer = resize3dLayer('OutputSize',[128 128 36],'Method','trilinear')` creates a 3-D resize layer that resizes input to 128-by-128-by-36 pixels using trilinear interpolation

Properties

Resize

`Scale` — Scale factor to resize input

3-element row vector of positive numbers

Scale factor to resize input, specified as 3-element row vector of positive numbers. The scale factors are for the row, column, and plane dimensions, respectively. When creating the layer, you can specify `Scale` as a scalar to use the same value for all dimensions.

`OutputSize` — Output size of resized input
3-element row vector of positive integers

Output size of resized input, specified as a 3-element row vector of positive integers of the form [*nrows ncols nplanes*]. You can specify two elements as NaN, in which case the layer computes the values automatically to preserve the aspect ratio of the input.

`EnableReferenceInput` — Add reference feature map as input
`false` or `0` | `true` or `1`

Add reference feature map as input to the layer, specified as a numeric or logical `0` (`false`) or `1` (`true`). When you specify the value as `true`, the layer resizes the height, width, and depth of the input to match the height, width, and depth of the reference feature map. The resizing operation does not change the number of channels of the input.

When you enable a reference feature map, the inputs to the layer have the names `'in1'` and `'ref'`, where `'ref'` is the name of the reference feature map. Use the input names when connecting or disconnecting the layer by using `connectLayers` or `disconnectLayers`.

`Method` — Interpolation method
`'nearest'` (default) | `'trilinear'`

Interpolation method, specified as `'nearest'` for nearest neighbor interpolation or `'trilinear'` for trilinear interpolation.

`GeometricTransformMode` — Geometric transformation mode
`'half-pixel'` (default) | `'asymmetric'`

Geometric transformation mode to map points from input space to output space, specified as `'half-pixel'` or `'asymmetric'`.

`NearestRoundingMode` — Rounding mode for nearest neighbor interpolation
`'round'` (default) | `'floor'` | `'onnx-10'`

Rounding mode for nearest neighbor interpolation, specified as one of the following.

- `'round'` — use the same rounding behavior as the MATLAB `round` function.
- `'floor'` — use the same rounding behavior as the MATLAB `floor` function.
- `'onnx-10'` — reproduce the resizing behavior of the ONNX (Open Neural Network Exchange) opset 10 Resize operator.

This property is valid when the `Method` property is `'nearest'`.

Layer

`Name` — Layer name
`''` (default) | character vector | string scalar

Layer name, specified as a character vector or a string scalar. For `Layer` array input, the `trainNetwork`, `assembleNetwork`, `layerGraph`, and `dlnetwork` functions automatically assign names to layers with name `''`.

Data Types: `char` | `string`

`NumInputs` — Number of inputs
1 (default) | 2

Number of inputs of the layer, specified as 1 when the EnableReferenceInput property is `false` or 2 when the EnableReferenceInput property is `true`.

Data Types: `double`

`InputNames` — Input names
`{'in'}` (default) | `{'in','ref'}`

Input names of the layer, specified as `{'in'}` when the EnableReferenceInput property is `false` or `{'in','ref'}` when the EnableReferenceInput property is `true`.

Data Types: `cell`

`NumOutputs` — Number of outputs
1 (default)

This property is read-only.

Number of outputs of the layer. This layer has a single output only.

Data Types: `double`

`OutputNames` — Output names
`{'out'}` (default)

This property is read-only.

Output names of the layer. This layer has a single output only.

Data Types: `cell`

Examples

Create 3-D Resize Layer Specifying Scale Factor

Create a 3-D resize layer. Specify a horizontal and vertical scale factor of 2 and a depthwise scale factor of 4.

```
layer = resize3dLayer('Scale',[2 2 4])
```

```
layer =
  Resize3DLayer with properties:

                      Name: ''
                     Scale: [2 2 4]
                OutputSize: []
      EnableReferenceInput: 0
                    Method: 'nearest'
    GeometricTransformMode: 'half-pixel'
       NearestRoundingMode: 'round'

   Learnable Parameters
    No properties.
```

```
   State Parameters
    No properties.

  Show all properties
```

Create 3-D Resize Layer Specifying Output Size

Create a 3-D resize layer named 'resize224' with an output size of [224 224 224].

```
layer = resize3dLayer('OutputSize',[224 224 224],'Name','resize224')
```

```
layer =
  Resize3DLayer with properties:

                      Name: 'resize224'
                     Scale: []
                OutputSize: [224 224 224]
      EnableReferenceInput: 0
                    Method: 'nearest'
    GeometricTransformMode: 'half-pixel'
       NearestRoundingMode: 'round'

   Learnable Parameters
    No properties.

   State Parameters
    No properties.

  Show all properties
```

Create 3-D Resize Layer with Reference Port

Create an array of layers that includes a 3-D resize layer that accepts a reference input feature map.

```
layers = [
    image3dInputLayer([32 32 32 3],'Name','image')
    resize3dLayer('EnableReferenceInput',true,'Name','resize')]
```

```
layers =
  2x1 Layer array with layers:

     1   'image'    3-D Image Input   32x32x32x3 images with 'zerocenter' normalization
     2   'resize'   Resize            nnet.cnn.layer.Resize3DLayer
```

Create a `layerGraph`. The first input of the 3-D resize layer is automatically connected to the output of the 3-D image input layer.

```
lgraph = layerGraph(layers);
```

Connect the `'ref'` input of the 3-D resize layer to the output of a layer that provides a reference feature map by using the `connectLayers` function. This example shows a trivial connection in which the `'ref'` input is also connected to the output of the 3-D image input layer.

```
lgraph = connectLayers(lgraph,'image','resize/ref');
```

Create 3-D Resize Layer with Trilinear Interpolation

Create a 3-D resize layer named 'rescale0.5' with a uniform scale factor of 0.5. Specify the interpolation method as trilinear interpolation.

```
layer = resize3dLayer('Scale',0.5,'Method','trilinear','Name','rescale0.5')
```

```
layer =
  Resize3DLayer with properties:

                      Name: 'rescale0.5'
                     Scale: [0.5000 0.5000 0.5000]
                OutputSize: []
      EnableReferenceInput: 0
                    Method: 'trilinear'
    GeometricTransformMode: 'half-pixel'
       NearestRoundingMode: 'round'

   Learnable Parameters
    No properties.

   State Parameters
    No properties.

  Show all properties
```

References

[1] *Open Neural Network Exchange*. https://github.com/onnx/.

[2] *ONNX*. https://onnx.ai/.

See Also

`resize2dLayer` | `averagePooling3dLayer` | `transposedConv3dLayer` | `dlresize` | `importONNXLayers`

Topics

“Deep Learning in MATLAB” (Deep Learning Toolbox)
“Specify Layers of Convolutional Neural Network” (Deep Learning Toolbox)
“List of Deep Learning Layers” (Deep Learning Toolbox)

Introduced in R2020b

spaceToDepthLayer

Space to depth layer

Description

A space to depth layer permutes the spatial blocks of the input into the depth dimension. Use this layer when you need to combine feature maps of different size without discarding any feature data.

Given an input feature map of size [*H W C*] and blocks of size [*height width*], the output feature map size is [`floor`(*H*/*height*) `floor`(*W*/*width*) *C***height***width*].

This object requires Deep Learning Toolbox.

Creation

Syntax

```
layer = spaceToDepthLayer(blockSize)
layer = spaceToDepthLayer(blockSize,'Name',Name)
```

Description

`layer = spaceToDepthLayer(blockSize)` creates a space to depth layer, specifying the block size to reorder the input activation. The `blockSize` input sets the BlockSize property.

`layer = spaceToDepthLayer(blockSize,'Name',Name)` creates a space to depth layer and sets the optional `Name` property.

Properties

`BlockSize` — Block size to reorder input activation

vector of two positive integers

Block size to reorder the input activation, specified as a vector of two positive integers `[h w]`, where `h` is the height and `w` is the width. When creating the layer, you can specify `BlockSize` as a scalar to use the same value for both dimensions.

Example: `[2 1]` specifies blocks of height 2 and width 1.

`Name` — Layer name

`''` (default) | character vector | string scalar

Layer name, specified as a character vector or a string scalar. For `Layer` array input, the `trainNetwork`, `assembleNetwork`, `layerGraph`, and `dlnetwork` functions automatically assign names to layers with name `''`.

Data Types: `char` | `string`

`NumInputs` — Number of inputs
1 (default)

This property is read-only.

Number of inputs of the layer. This layer accepts a single input only.

Data Types: `double`

`InputNames` — Input names
`{'in'}` (default)

This property is read-only.

Input names of the layer. This layer accepts a single input only.

Data Types: `cell`

`NumOutputs` — Number of outputs
1 (default)

This property is read-only.

Number of outputs of the layer. This layer has a single output only.

Data Types: `double`

`OutputNames` — Output names
`{'out'}` (default)

This property is read-only.

Output names of the layer. This layer has a single output only.

Data Types: `cell`

Examples

Create Space to Depth Layer

Specify the block size to reorder input activations.

```
blockSize = [2 2];
```

Create a space to depth layer named `'spacetodepth'`.

```
layer = spaceToDepthLayer(blockSize,'Name','spacetodepth')
```

```
layer =
  SpaceToDepthLayer with properties:

         Name: 'spacetodepth'
    BlockSize: [2 2]

   Learnable Parameters
    No properties.
```

```
  State Parameters
   No properties.

 Show all properties
```

Extended Capabilities

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

To generate CUDA or C++ code by using GPU Coder, you must first construct and train a deep neural network. Once the network is trained and evaluated, you can configure the code generator to generate code and deploy the convolutional neural network on platforms that use NVIDIA or ARM GPU processors. For more information, see “Deep Learning with GPU Coder” (GPU Coder).

For this layer, you can generate code that takes advantage of the NVIDIA CUDA deep neural network library (cuDNN), NVIDIA TensorRT high performance inference library, or the ARM `Compute Library` for Mali GPU.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

`roiMaxPooling2dLayer` | `yolov2Layers` | `DepthToSpace2DLayer` | `depthToSpace` | `spaceToDepth`

Topics

“Getting Started with Object Detection Using Deep Learning” (Computer Vision Toolbox)
“Getting Started with YOLO v2” (Computer Vision Toolbox)
“List of Deep Learning Layers” (Deep Learning Toolbox)
“Deep Learning in MATLAB” (Deep Learning Toolbox)

Introduced in R2020b

lazysnapping

Segment image into foreground and background using graph-based segmentation

Syntax

```
BW = lazysnapping(A,L,foremask,backmask)
BW = lazysnapping(A,L,foreind,backind)
BW = lazysnapping( ___ ,Name,Value)
```

Description

`BW = lazysnapping(A,L,foremask,backmask)` segments the image `A` into foreground and background regions using lazy snapping. The label matrix `L` specifies the subregions of the image. `foremask` and `backmask` are masks designating pixels in the image as foreground and background, respectively.

`BW = lazysnapping(A,L,foreind,backind)` segments the image `A` into foreground and background regions. `foreind` and `backind` specify the linear indices of the pixels in the image marked as foreground and background, respectively.

`BW = lazysnapping( ___ ,Name,Value)` segments the image or volume using name-value pairs to control aspects of the segmentation.

Examples

Perform Lazy Snapping Using Foreground and Background Masks

Read and display an image.

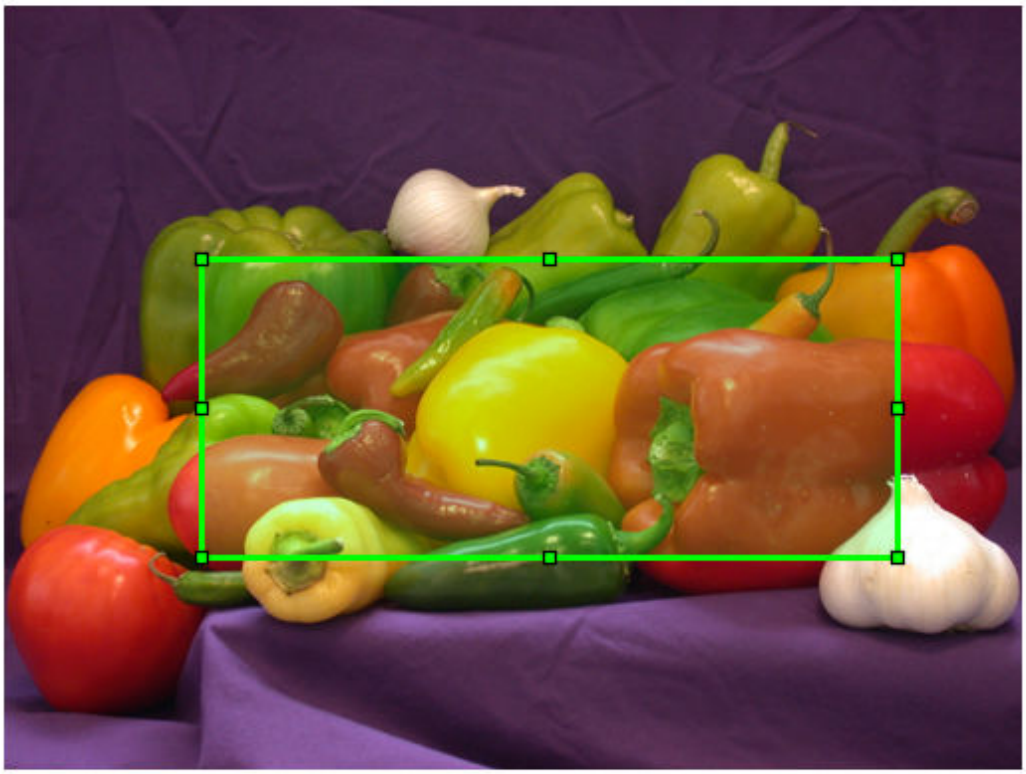
```
RGB = imread('peppers.png');
imshow(RGB)
```

Create a label matrix.

```
L = superpixels(RGB,500);
```

Specify a rectangular ROI within the foreground by using the `drawrectangle` function. The 'Position' name-value pair argument specifies the upper left coordinates, width, and height of the ROI as the 4-element vector `[xmin,ymin,width,height]`. If you want to draw the rectangle interactively, then omit the 'Position' name-value pair argument.

```
f = drawrectangle(gca,'Position',[100 128 350 150],'Color','g');
```

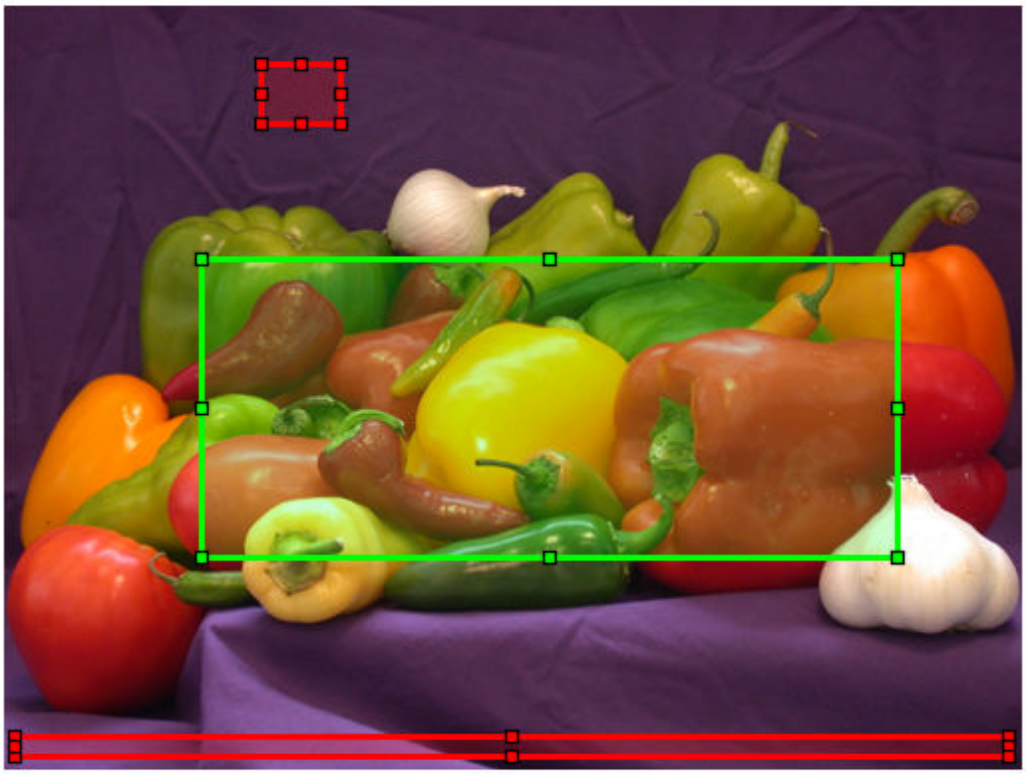


Create a mask that contains the foreground pixels.

```
foreground = createMask(f,RGB);
```

Specify background ROIs. To improve the segmentation accuracy, this example specifies two rectangular ROIs in different areas of the background.

```
b1 = drawrectangle(gca,'Position',[130 30 40 30],'Color','r');
b2 = drawrectangle(gca,'Position',[6 368 500 10],'Color','r');
```



Create a mask that contains the background pixels. This mask is the union of the two background ROIs.

```
background = createMask(b1,RGB) + createMask(b2,RGB);
```

Perform lazy snapping.

```
BW = lazysnapping(RGB,L,foreground,background);
```

Visualize the result of the segmentation by highlighting the foreground in green.

```
imshow(labeloverlay(RGB,BW,'Colormap',[0 1 0]))
```

Create a masked image in which the background is black.

```
maskedImage = RGB;
maskedImage(repmat(~BW,[1 1 3])) = 0;
imshow(maskedImage)
```

Perform Lazy Snapping Using Pixel Indices

Read and display an image.

```
RGB = imread('peppers.png');
imshow(RGB)
```

Create a label matrix.

```
L = superpixels(RGB,500);
```

Specify the x- and y-coordinates of pixels in the foreground.

```
foregroundX = [34 114 195 259 392 467 483];
foregroundY = [298 140 135 200 205 283 104];
```

Convert the coordinates to linear indices. `sub2ind` takes (row, column) coordinates so specify the input arguments with the y-coordinates before the x-coordinates.

```
foregroundInd = sub2ind(size(RGB),foregroundY,foregroundX);
```

Specify the x- and y-coordinates of pixels in the background.

```
backgroundX = [130 170];
backgroundY = [52 32];
```

Convert the coordinates to linear indices.

```
backgroundInd = sub2ind(size(RGB),backgroundY,backgroundX);
```

Perform lazy snapping.

```
BW = lazysnapping(RGB,L,foregroundInd,backgroundInd);
```

Display the segmented mask. Foreground pixels are `true` and background pixels are `false`.

```
imshow(BW)
```

Display the mask over the original image, highlighting foreground pixels in green.

```
imshow(labeloverlay(RGB,BW,'Colormap',[0 1 0]))
```

Segment Volume in Foreground and Background

Load 3-D volumetric image into the workspace.

```
D = load('mri.mat');
V  = squeeze(D.D);
```

Create a 2-D mask identifying initial foreground and background seed points.

```
seedLevel = 10;
fseed = V(:,:,seedLevel) > 75;
bseed = V(:,:,seedLevel) == 0;
figure;
imshow(fseed)
```

```
figure;
imshow(bseed)
```

Place seed points into empty 3-D mask.

```
fmask = zeros(size(V));
bmask = fmask;
fmask(:,:,seedLevel) = fseed;
bmask(:,:,seedLevel) = bseed;
```

Generate a 3-D label matrix.

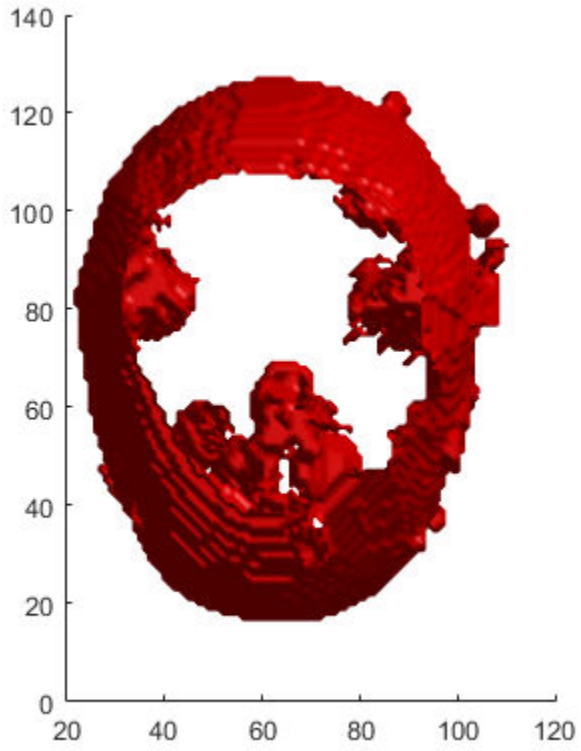
```
L = superpixels3(V,500);
```

Segment the image into foreground and background using Lazy Snapping.

```
bw = lazysnapping(V,L,fmask,bmask);
```

Display the 3-D segmented image.

```
figure;
p = patch(isosurface(double(bw)));
p.FaceColor = 'red';
p.EdgeColor = 'none';
daspect([1 1 27/128]);
camlight; lighting phong
```



Input Arguments

A — Image to segment

2-D grayscale image | 2-D truecolor image | 2-D multispectral image | 3-D grayscale volume

Image to segment, specified as a 2-D grayscale, truecolor, or multispectral image or a 3-D grayscale volume. For `double` and `single` images, `lazysnapping` assumes the range of the image to be [0, 1]. For `uint16`, `int16`, and `uint8` images, `lazysnapping` assumes the range to be the full range for the given data type. If the values do not match the expected range based on the data type, then scale the image to the expected range or adjust `EdgeWeightScaleFactor` to improve results.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

L — Label matrix

numeric array

Label matrix of the input image or volume, specified as numeric array. For 2-D grayscale images and 3-D grayscale volumes, the size of `L` must match the size of the input image `A`. For color images and multichannel images, `L` must be a 2-D array where the first two dimensions match the first two dimensions of the input image `A`.

Do not mark a given subregion of the label matrix as belonging to both the foreground mask and the background mask. If a region of the label matrix contains pixels belonging to both the foreground mask and background mask, `lazysnapping` segments the region as background.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`foremask` — Mask image that defines foreground

logical array

Mask image that defines the foreground, specified as a logical array. For 2-D grayscale images and 3-D grayscale volumes, the size of `foremask` must match the size of the input image `A`. For color images and multichannel images, `foremask` must be a 2-D array where the first two dimensions match the first two dimensions of the input image `A`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`backmask` — Mask image that defines background

logical array

Mask image that defines the background, specified as a logical array. For 2-D grayscale images and 3-D grayscale volumes, the size of `backmask` must match the size of the input image `A`. For color images and multichannel images, `backmask` must be a 2-D array where the first two dimensions match the first two dimensions of the input image `A`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`foreind` — Linear index of foreground pixels

numeric vector

Linear index of pixels in the label matrix, specified as a numeric vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`backind` — Linear index of background pixels

numeric vector

Linear index of pixels that define the background, specified as a numeric vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Connectivity',6`

`Connectivity` — Connectivity of connected components

8 for 2-D images and 26 for 3-D images (default) | 4 | 6 | 18

Connectivity of connected components, specified as the comma-separated pair consisting of `'Connectivity'` and one of the following: 4 or 8, for 2-D images, and 6, 18, or 26 for 3-D images (volumes).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`EdgeWeightScaleFactor` — Scale factor for edge weights
`500` (default) | positive number

Scale factor for edge weights between the subregions of the label matrix, specified as the comma-separated pair consisting of `'EdgeWeightScaleFactor'` and a positive number. Typical values range from [10, 1000]. Increasing this value increases the likelihood that `lazysnapping` labels neighboring subregions together as either foreground or background.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`BW` — Segmented image
logical array

Segmented image, returned as a logical array of the same size as the label matrix, `L`.

Data Types: `logical`

Tips

- The lazy snapping algorithm developed by Li et al. clusters foreground and background values using the K-means method. This implementation of the lazy snapping algorithm does not cluster similar foreground or background pixels. To improve performance, reduce the number of pixels with similar values that are identified as foreground or background.
- To obtain masks `foremask` or `backmask` interactively, you can draw an ROI on the image then create a mask from the ROI by using the `createMask` function. For more information, see “Create ROI Shapes”.
- To obtain pixel indices `foreind` or `backind` interactively, you can draw a `Polyline` ROI object by using the `drawpolyline` function. Get the *x*- and *y*-coordinates of the vertices from the `Position` property of the `Polyline`. Finally, convert the coordinates to linear indices by using the `sub2ind` function. Note that the `sub2ind` function uses (*row*, *column*) coordinates instead of (*x*, *y*) coordinates.

References

[1] Y. Li, S. Jian, C. Tang, H. Shum, *Lazy Snapping* In Proceedings from the 31st International Conference on Computer Graphics and Interactive Techniques, 2004.

See Also

Image Segmenter

Topics
“Segment Image Using Graph Cut in Image Segmenter”

Introduced in R2017a

lin2rgb

Apply gamma correction to linear RGB values

Syntax

```
B = lin2rgb(A)
B = lin2rgb(A,Name,Value)
```

Description

`B = lin2rgb(A)` applies a gamma correction to the linear RGB values in image `A` so that `B` is in the sRGB color space, which is suitable for display.

`B = lin2rgb(A,Name,Value)` applies gamma correction using name-value pairs to control additional options.

Examples

Plot Gamma Curve of sRGB and Adobe RGB

Define a range of linear values. This vector defines 257 equally spaced points between 0 and 1.

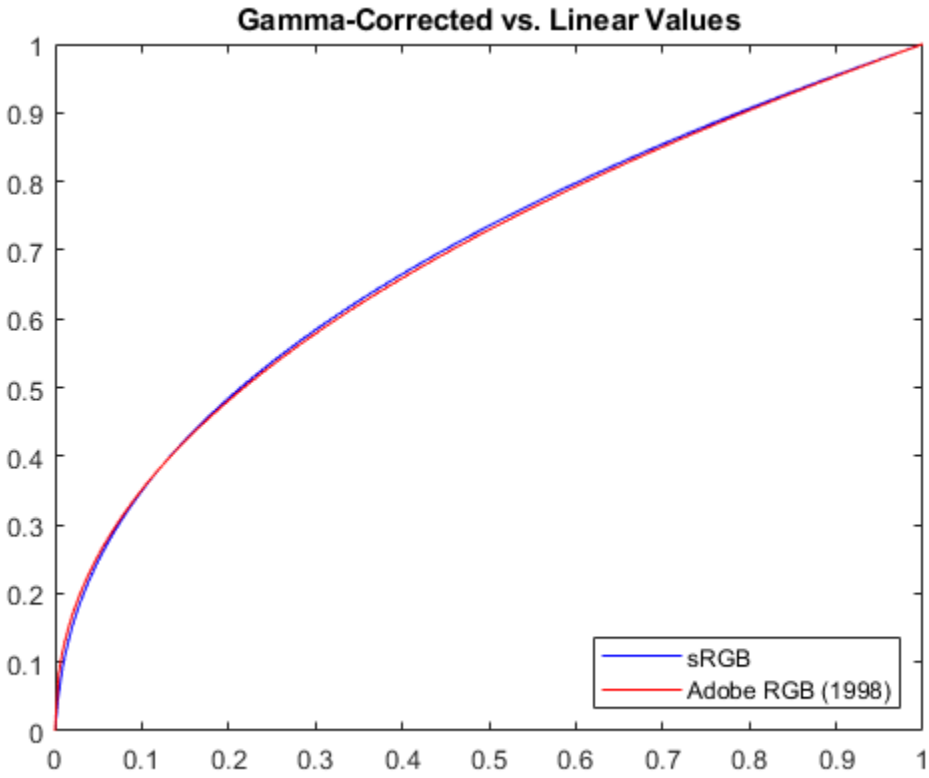
```
lin = linspace(0,1,257);
```

Apply gamma correction to the linear values based on the sRGB standard. Then apply gamma correction to the linear values based on the Adobe RGB (1998) standard.

```
sRGB = lin2rgb(lin);
adobeRGB = lin2rgb(lin,'ColorSpace','adobe-rgb-1998');
```

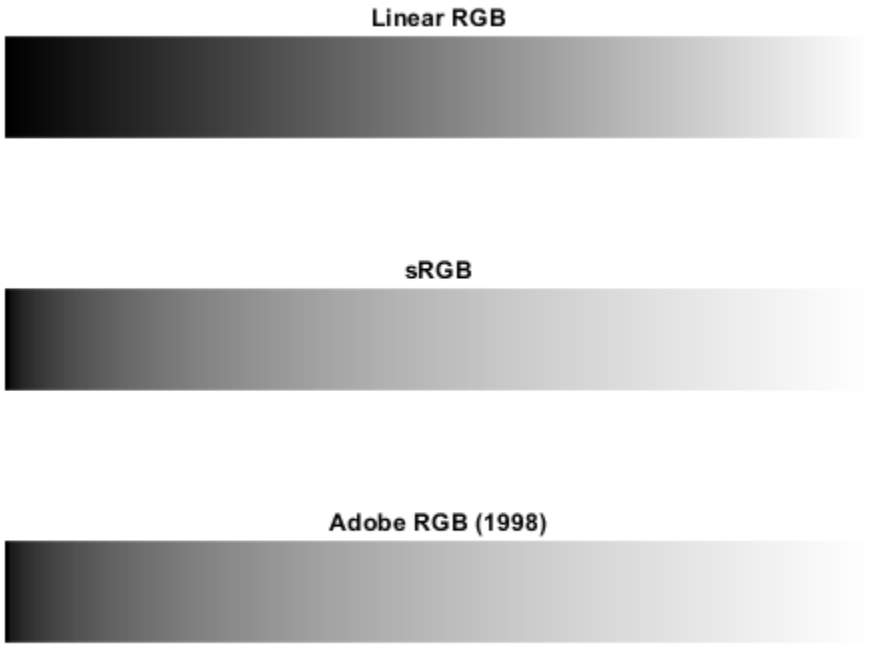
Plot the gamma-corrected curves.

```
figure
plot(lin,sRGB,'b',lin,adobeRGB,'r')
title('Gamma-Corrected vs. Linear Values')
legend('sRGB','Adobe RGB (1998)','Location','southeast')
```



For an alternative visualization, plot color bars representing each color space.

```
cb_lin = ones(30,257) .* lin;
cb_sRGB = ones(30,257) .* sRGB;
cb_adobeRGB = ones(30,257) .* adobeRGB;

figure
subplot(3,1,1); imshow(cb_lin); title('Linear RGB')
subplot(3,1,2); imshow(cb_sRGB); title('sRGB');
subplot(3,1,3); imshow(cb_adobeRGB); title('Adobe RGB (1998)');
```



The gamma-corrected color spaces get brighter more quickly than the linear color space, as expected.

Apply sRGB Gamma Correction to Linear RGB Image

Open an image file containing minimally processed linear RGB intensities.

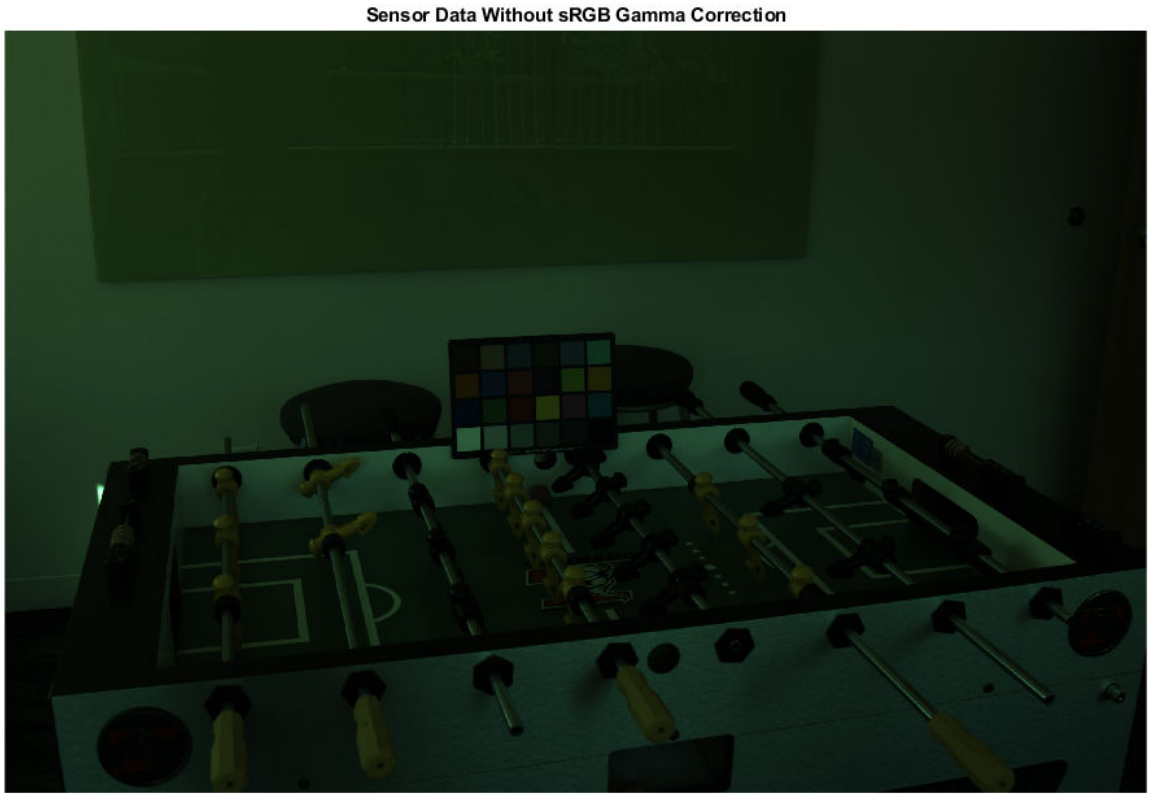
```
A = imread('foosballraw.tiff');
```

The image data is the raw sensor data after correcting the black level and scaling to 16 bits per pixel. Interpolate the intensities to reconstruct color by using the `demosaic` function. The color filter array pattern is RGGB.

```
A_demosaiced = demosaic(A,'rggb');
```

Display the image. To shrink the image so that it appears fully on the screen, set the optional initial magnification to a value less than 100.

```
figure
imshow(A_demosaiced,'InitialMagnification',25)
title('Sensor Data Without sRGB Gamma Correction')
```

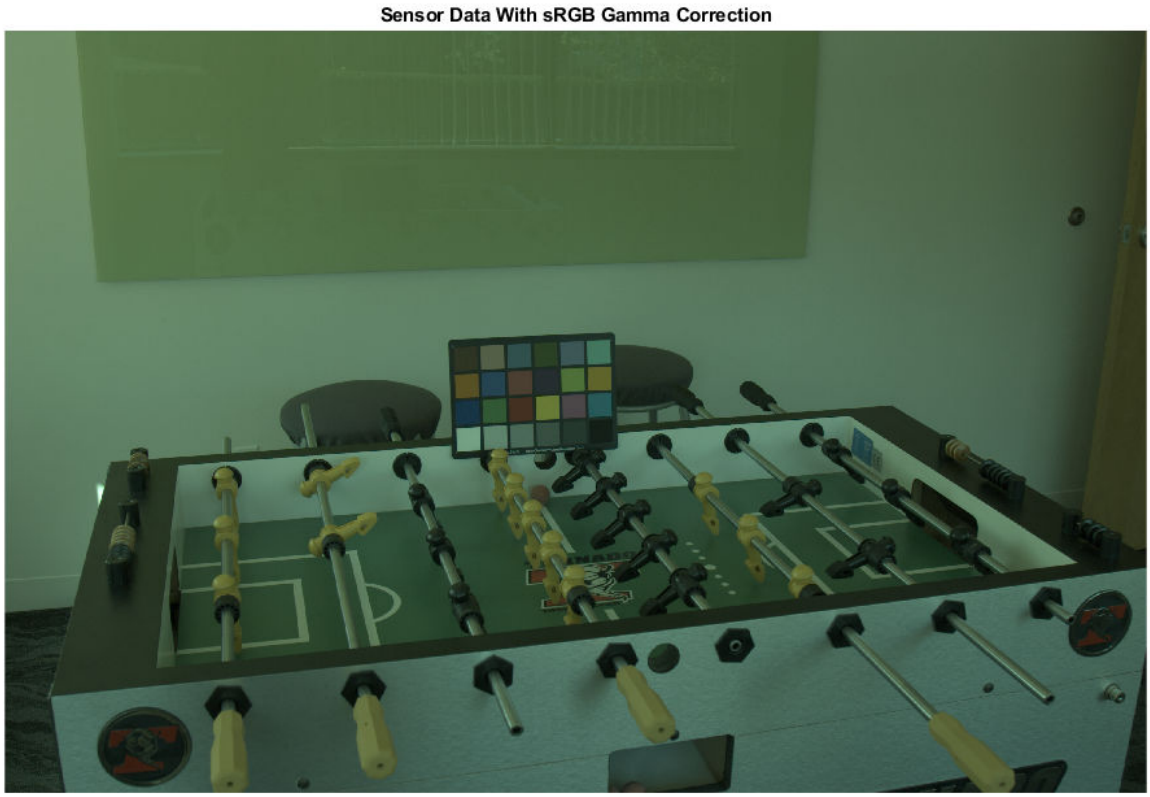


The image appears dark because it is in the linear RGB color space. Apply gamma correction to the image according to the sRGB standard, storing the values in double precision.

```
A_sRGB = lin2rgb(A_demosaiced,'OutputType','double');
```

Display the gamma-corrected image, setting the optional magnification.

```
figure
imshow(A_sRGB,'InitialMagnification',25)
title('Sensor Data With sRGB Gamma Correction');
```



The gamma-corrected image looks brighter than the linear image, as expected.

Input Arguments

A — Linear RGB color values

numeric array

Linear RGB color values, specified as a numeric array in one of the following formats.

- *c*-by-3 colormap. Each row specifies one RGB color value.
- *m*-by-*n*-by-3 image
- *m*-by-*n*-by-3-by-*p* stack of images

Data Types: `single` | `double` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `B = lin2rgb(I,'ColorSpace','adobe-rgb-1998')` applies gamma correction to an image, `I`, according to the Adobe RGB (1998) standard.

`ColorSpace` — Color space of the output image
`'srgb'` (default) | `'adobe-rgb-1998'`

Color space of the output image, specified as the comma-separated pair consisting of `'ColorSpace'` and `'srgb'` or `'adobe-rgb-1998'`.

Data Types: `char` | `string`

`OutputType` — Data type of output RGB values
`'double'` | `'single'` | `'uint8'` | `'uint16'`

Data type of the output RGB values, specified as the comma-separated pair consisting of `'OutputType'` and `'double'`, `'single'`, `'uint8'`, or `'uint16'`. By default, the output data type is the same as the data type of `A`.

Data Types: `char` | `string`

Output Arguments

`B` — Gamma-corrected RGB image
numeric array

Gamma-corrected RGB image, returned as a numeric array of the same size as the input `A`.

Algorithms

Gamma Correction Using the sRGB Standard

The gamma correction to transform linear RGB tristimulus values into sRGB tristimulus values is defined by the following parametric curve:

$f(u) = -f(-u), \qquad u < 0$

$f(u) = c \cdot u, \qquad 0 \le u < d$

$f(u) = a \cdot u^{\gamma} + b, \qquad u \ge d,$

where u represents a color value with these parameters:

$a = 1.055$

$b = –0.055$

$c = 12.92$

$d = 0.0031308$

$\gamma = 1/2.4$

Gamma Correction Using the Adobe RGB (1998) Standard

The gamma correction to transform linear RGB tristimulus values into Adobe RGB (1998) tristimulus values uses a simple power function:

$v = u^{\gamma}, \qquad u \geq 0$

$v = -(-u)^{\gamma}, \qquad u < 0,$

with

$\gamma = 1/2.19921875$

References

[1] Ebner, Marc. "Gamma Correction." *Color Constancy*. Chichester, West Sussex: John Wiley & Sons, 2007.

[2] Adobe Systems Incorporated. "Inverting the color component transfer function." *Adobe RGB (1998) Color Image Encoding*. Section 4.3.5.2, May 2005, p.12.

See Also

`rgb2lin`

Introduced in R2017b

localcontrast

Edge-aware local contrast manipulation of images

Syntax

```
B = localcontrast(A)
B = localcontrast(A,edgeThreshold,amount)
```

Description

`B = localcontrast(A)` enhances the local contrast of the grayscale or RGB image `A`.

`B = localcontrast(A,edgeThreshold,amount)` enhances or flattens the local contrast of `A` by increasing or smoothing details while leaving strong edges unchanged. `edgeThreshold` defines the minimum intensity amplitude of strong edges to leave intact. `amount` is the amount of enhancement or smoothing desired.

Examples

Increase or Reduce Local Contrast of Image

Import an RGB image.

```
A = imread('peppers.png');
```

Increase the local contrast of the input image.

```
edgeThreshold = 0.4;
amount = 0.5;
B = localcontrast(A, edgeThreshold, amount);
```

Display the results compared to the original image

```
imshowpair(A, B, 'montage')
```

Reduce the local contrast of the input image.

```
amount = -0.5;
B2 = localcontrast(A, edgeThreshold, amount);
```

Display the new results again, compared to the original image.

```
imshowpair(A, B2, 'montage')
```

Input Arguments

A — Grayscale or RGB image to be filtered

real, non-sparse, *m*-by-*n* or *m*-by-*n*-by-3 matrix

Grayscale or RGB image to be filtered, specified as a real, non-sparse, *m*-by-*n* or *m*-by-*n*-by-3 matrix.

Data Types: `single` | `int8` | `int16` | `uint8` | `uint16`

`edgeThreshold` — Amplitude of strong edges to leave intact
`0.3` (default) | numeric scalar in the range `[0,1]`

Amplitude of strong edges to leave intact, specified as a numeric scalar in the range `[0,1]`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`amount` — Amount of enhancement or smoothing desired
`0.25` (default) | numeric scalar in the range `[-1,1]`

Amount of enhancement or smoothing desired, specified as a numeric scalar in the range `[-1,1]`. Negative values specify edge-aware smoothing. Positive values specify edge-aware enhancement.

Value	Description
`0`	Leave input image unchanged.
`1`	Strongly enhance the local contrast of the input image
`-1`	Strongly smooth the details of the input image

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`B` — Filtered image
numeric array

Filtered image, returned as a numeric array the same size and class as the input image.

See Also
`imadjust` | `imcontrast` | `imsharpen` | `locallapfilt`

Introduced in R2016b

locallapfilt

Fast local Laplacian filtering of images

Syntax

```
B = locallapfilt(I,sigma,alpha)
B = locallapfilt(I,sigma,alpha,beta)
B = locallapfilt( ___ ,Name,Value)
```

Description

`B = locallapfilt(I,sigma,alpha)` filters the grayscale or RGB image `I` with an edge-aware, fast local Laplacian filter. `sigma` characterizes the amplitude of edges in `I`. `alpha` controls smoothing of details.

`B = locallapfilt(I,sigma,alpha,beta)` filters the image using `beta` to control the dynamic range of `A`.

`B = locallapfilt( ___ ,Name,Value)` uses name-value pairs to control advanced aspects of the filter.

Examples

Increase Local Contrast of RGB Image Using Local Laplacian Filtering

Import an RGB image

```
A = imread('peppers.png');
```

Set parameters of the filter to increase details smaller than 0.4.

```
sigma = 0.4;
alpha = 0.5;
```

Use fast local Laplacian filtering

```
B = locallapfilt(A, sigma, alpha);
```

Display the original and filtered images side-by-side.

```
imshowpair(A, B, 'montage')
```

Increase Local Contrast, Balancing Speed and Quality

Local Laplacian filtering is a computationally intensive algorithm. To speed up processing, `locallapfilt` approximates the algorithm by discretizing the intensity range into a number of samples defined by the '`NumIntensityLevels`' parameter. This parameter can be used to balance speed and quality.

Import an RGB image and display it.

```
A = imread('peppers.png');
figure
imshow(A)
title('Original Image')
```

Use a `sigma` value to process the details and an `alpha` value to increase the contrast, effectively enhancing the local contrast of the image.

```
sigma = 0.2;
alpha = 0.3;
```

Using fewer samples increases the execution speed, but can produce visible artifacts, especially in areas of flat contrast. Time the function using only 20 intensity levels.

```
t_speed = timeit(@() locallapfilt(A, sigma, alpha, 'NumIntensityLevels', 20))
```

```
t_speed = 0.1760
```

Now, process the image and display it.

```
B_speed = locallapfilt(A, sigma, alpha, 'NumIntensityLevels', 20);
figure
imshow(B_speed)
title(['Enhanced with 20 intensity levels in ' num2str(t_speed) ' sec'])
```

A larger number of samples yields better looking results at the expense of more processing time. Time the function using 100 intensity levels.

```
t_quality = timeit(@() locallapfilt(A, sigma, alpha, 'NumIntensityLevels', 100))
```

```
t_quality = 0.8182
```

Process the image with 100 intensity levels and display it:

```
B_quality = locallapfilt(A, sigma, alpha, 'NumIntensityLevels', 100);
figure
imshow(B_quality)
title(['Enhancement with 100 intensity levels in ' num2str(t_quality) ' sec'])
```

Try varying the number of intensity levels on your own images. Try also flattening the contrast (with `alpha` > 1). You will see that the optimal number of intensity levels is different for every image and varies with `alpha`. By default, `locallapfilt` uses a heuristic to balance speed and quality, but it cannot predict the best value for every image.

Boost Local Color Contrast Using 'ColorMode'

Import a color image, reduce its size, and display it.

```
A = imread('car2.jpg');
A = imresize(A, 0.25);
figure
imshow(A)
title('Original Image')
```

Set the parameters of the filter to dramatically increase details smaller than 0.3 (out of a normalized range of 0 to 1).

```
sigma = 0.3;
alpha = 0.1;
```

Let's compare the two different modes of color filtering. Process the image by filtering its intensity and by filtering each color channel separately:

```
B_luminance = locallapfilt(A, sigma, alpha);
B_separate  = locallapfilt(A, sigma, alpha, 'ColorMode', 'separate');
```

Display the filtered images.

```
figure
imshow(B_luminance)
title('Enhanced by boosting the local luminance contrast')
```


```
figure
imshow(B_separate)
title('Enhanced by boosting the local color contrast')
```

An equal amount of contrast enhancement has been applied to each image, but colors are more saturated when setting '`ColorMode`' to '`separate`'.

Perform Edge-Aware Noise Reduction

Import an image. Convert the image to floating point so that we can add artificial noise more easily.

```
A = imread('pout.tif');
A = im2single(A);
```

Add Gaussian noise with zero mean and 0.001 variance.

```
A_noisy = imnoise(A, 'gaussian', 0, 0.001);
psnr_noisy = psnr(A_noisy, A);
fprintf('The peak signal-to-noise ratio of the noisy image is %0.4f\n', psnr_noisy);
```

```
The peak signal-to-noise ratio of the noisy image is 30.0234
```

Set the amplitude of the details to smooth, then set the amount of smoothing to apply.

```
sigma = 0.1;
alpha = 4.0;
```

Apply the edge-aware filter.

```
B = locallapfilt(A_noisy, sigma, alpha);
psnr_denoised = psnr(B, A);
fprintf('The peak signal-to-noise ratio of the denoised image is %0.4f\n', psnr_denoised);
```

```
The peak signal-to-noise ratio of the denoised image is 32.2016
```

Note an improvement in the PSNR of the image.

Display all three images side by side. Observe that details are smoothed and sharp intensity variations along edges are unchanged.

```
figure
subplot(1,3,1), imshow(A), title('Original')
subplot(1,3,2), imshow(A_noisy), title('Noisy')
subplot(1,3,3), imshow(B), title('Denoised')
```

Smooth Image Details Without Affecting Edge Sharpness

Import the image, resize it and display it

```
A = imread('car1.jpg');
A = imresize(A, 0.25);
figure
imshow(A)
title('Original Image')
```

The car is dirty and covered in markings. Let's try to erase the dust and markings on the body. Set the amplitude of the details to smooth, and set a large amount of smoothing to apply.

```
sigma = 0.2;
alpha = 5.0;
```

When smoothing (`alpha` > 1), the filter produces high quality results with a small number of intensity levels. Set a small number of intensity levels to process the image faster.

```
numLevels = 16;
```

Apply the filter.

```
B = locallapfilt(A, sigma, alpha, 'NumIntensityLevels', numLevels);
```

Display the "clean" car.

```
figure
imshow(B)
title('After smoothing details')
```

Input Arguments

`I` — Image to filter

2-D grayscale image | 2-D truecolor image

Image to filter, specified as a 2-D grayscale image or 2-D truecolor image.

Data Types: `single` | `int8` | `int16` | `uint8` | `uint16`

`sigma` — Amplitude of edges

non-negative number

Amplitude of edges, specified as a non-negative number. `sigma` should be in the range [0, 1] for integer images and for single images defined over the range [0, 1]. For single images defined over a different range [`a`, `b`], `sigma` should also be in the range [`a`, `b`].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`alpha` — Smoothing of details

positive number

Smoothing of details, specified as a positive number. Typical values of `alpha` are in the range [0.01, 10].

Value	Description
`alpha` less than `1`	Increases the details of the input image, effectively enhancing the local contrast of the image without affecting edges or introducing halos.
`alpha` greater than `1`	Smooths details in the input image while preserving crisp edges
`alpha` equal to `1`	The details of the input image are left unchanged.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`beta` — Dynamic range
`1` (default) | non-negative number

Dynamic range, specified as a non-negative number. Typical values of beta are in the range [0, 5]. `beta` affects the dynamic range of `A`.

Value	Description
`beta` less than `1`	Reduces the amplitude of edges in the image, effectively compressing the dynamic range without affecting details.
`beta` greater than `1`	Expands the dynamic range of the image.
`beta` equal to `1`	Dynamic range of the image is left unchanged. This is the default value.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'ColorMode','separate'`

`ColorMode` — Method used to filter RGB images
`'luminance'` (default) | `'separate'`

Method used to filter RGB images, specified as one of the following values. This parameter has no effect on grayscale images.

Value	Description
`'luminance'`	`locallapfilt` converts the input RGB image to grayscale before filtering and reintroduces color after filtering, which changes the contrast of the input image without affecting colors.
`'separate'`	`locallapfilt` filters each color channel independently.

Data Types: `char` | `string`

`NumIntensityLevels` — Number of intensity samples

`'auto'` (default) | positive integer

Number of intensity samples in the dynamic range of the input image, specified as `'auto'` or positive integer. A higher number of samples gives results closer to exact local Laplacian filtering. A lower number increases the execution speed. Typical values are in the range `[10, 100]`. If set to `'auto'`, `locallapfilt` chooses the number of intensity levels automatically to balance quality and speed based on other parameters of the filter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

Output Arguments

`B` — Filtered image

numeric array

Filtered image, returned as a numeric array the same size and data type as the input image, `A`.

References

[1] Paris, Sylvain, Samuel W. Hasinoff, and Jan Kautz. *Local Laplacian filters: edge-aware image processing with a Laplacian pyramid*, ACM Trans. Graph. 30.4 (2011): 68.

[2] Aubry, Mathieu, et al. *Fast local laplacian filters: Theory and applications*. ACM Transactions on Graphics (TOG) 33.5 (2014): 167.

See Also

`localcontrast` | `localtonemap`

Introduced in R2016b

localtonemap

Render HDR image for viewing while enhancing local contrast

Syntax

```
rgb = localtonemap(hdr)
rgb = localtonemap(hdr,Name,Value)
```

Description

`rgb = localtonemap(hdr)` converts the high dynamic range (HDR) image `hdr` to a low dynamic range (LDR) image, `rgb`, suitable for display. `localtonemap` uses a process called tone mapping while preserving its local contrast.

`rgb = localtonemap(hdr,Name,Value)` controls various aspects of the tone mapping using name-value pair arguments.

Examples

Compress Dynamic Range of HDR Image for Viewing

Load a high dynamic range image.

```
HDR = hdrread('office.hdr');
```

Apply local tone mapping with a small amount of dynamic range compression.

```
RGB = localtonemap(HDR, 'RangeCompression', 0.1);
```

Display the resulting tone-mapped image.

```
imshow(RGB)
```

Repeat the operation but, this time, accentuate the details in the image.

```
RGB = localtonemap(HDR, ...
                   'RangeCompression', 0.1, ...
                   'EnhanceContrast', 0.5);
```

Display the resulting tone-mapped image with increased details.

```
imshow(RGB)
```

Input Arguments

`hdr` — HDR image

m-by-*n* numeric matrix | *m*-by-*n*-by-3 numeric array

HDR image, specified as an *m*-by-*n* numeric matrix or *m*-by-*n*-by-3 numeric array.

Data Types: `single`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'RangeCompression',0.5`

`RangeCompression` — Amount of compression

1 (default) | number in the range [0,1]

Amount of compression applied to the dynamic range of the HDR image, specified as a number in the range [0, 1].

Value	Description
`0`	Minimum compression, which consists in only remapping the middle 99% intensities to a dynamic range of 100:1 followed by gamma correction with an exponent of 1/2.2.
`1`	Maximum compression using local Laplacian filtering.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`EnhanceContrast` — Amount of local contrast enhancement
`0` (default) | number in the range [0, 1]

Amount of local contrast enhancement, specified as a number in the range [0, 1].

Value	Description
`0`	No change to local contrast
`1`	Maximum local contrast enhancement

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`rgb` — Tone-mapped LDR image
numeric array

Tone-mapped LDR image, returned as a numeric array of the same size as the input image `hdr`.

Algorithms

`localtonemap` uses local Laplacian filtering in logarithmic space to compress the dynamic range of HDR while preserving or enhancing its local contrast. The 99% middle intensities of the compressed image are then remapped to a fixed 100:1 dynamic range to give the output image a consistent look. `localtonemap` then applies gamma correction to produce the final image for display.

See Also
`tonemap` | `locallapfilt`

Introduced in R2016b

makecform

Create color transformation structure

Syntax

```
C = makecform(type)
C = makecform(type,'WhitePoint',WP)
C = makecform(type,'AdaptedWhitePoint',WP)
C = makecform('adapt','WhiteStart',WPS,'WhiteEnd',WPE,'AdaptModel',model)

C = makecform('srgb2cmyk','RenderingIntent',intent)
C = makecform('cmyk2srgb','RenderingIntent',intent)

C = makecform('icc',src_profile,dest_profile)
C = makecform('icc',src_profile,dest_profile,'SourceRenderingIntent',
src_intent,'DestRenderingIntent',dest_intent)

C = makecform('mattrc',MatTRC,'Direction',direction)
C = makecform('mattrc',profile,'Direction',direction)
C = makecform('mattrc',profile,'Direction',direction,'RenderingIntent',
trc_intent)
C = makecform('graytrc',profile,'Direction',direction)
C = makecform('graytrc',profile,'Direction',direction,'RenderingIntent',
trc_intent)

C = makecform('clut',profile,LUTtype)
C = makecform('named',profile,space)
```

Description

The `makecform` function supports conversions between members of the family of device-independent color spaces defined by the *Commission Internationale de l'Éclairage* (International Commission on Illumination, or CIE). `makecform` also supports conversions to and from the *sRGB* and *CMYK* color spaces. To perform a color space transformation, pass the color transformation structure created by `makecform` as an argument to the `applycform` function.

`C = makecform(type)` creates a color transformation structure `C` that defines the color space conversion specified by `type`.

`C = makecform(type,'WhitePoint',WP)` specifies the value of the reference white point, `WP`, for `'xyz2lab'` or `'lab2xyz'` conversions.

`C = makecform(type,'AdaptedWhitePoint',WP)` specifies the adapted white point, `WP`, for `'srgb2lab'`, `'lab2srgb'`, `'srgb2xyz'`, or `'xyz2srgb'` conversions.

`C = makecform('adapt','WhiteStart',WPS,'WhiteEnd',WPE,'AdaptModel',model)` creates a linear chromatic-adaptation color transformation using the chromatic-adaptation model, `model`, starting with whitepoint `WPS` and ending with whitepoint `WPE`.

`C = makecform('srgb2cmyk','RenderingIntent',intent)` and

`C = makecform('cmyk2srgb','RenderingIntent',intent)` specify the rendering intent for color transformations between *sRGB* IEC61966-2.1 and "Specifications for Web Offset Publications" (SWOP) *CMYK*.

`C = makecform('icc',src_profile,dest_profile)` creates a color transformation based on two ICC profiles, `src_profile` and `dest_profile`.

`C = makecform('icc',src_profile,dest_profile,'SourceRenderingIntent', src_intent,'DestRenderingIntent',dest_intent)` creates a color transformation based on two ICC color profiles, `src_profile` and `dest_profile`, specifying the rendering intent for the source and destination profiles.

`C = makecform('mattrc',MatTRC,'Direction',direction)` creates a color transformation based on a Matrix/Tone Reproduction Curve (MatTRC) model, in either the forward or inverse direction.

`C = makecform('mattrc',profile,'Direction',direction)` creates a color transformation based on the `'MatTRC'` field of the ICC color profile `profile`, in either the forward or inverse direction.

`C = makecform('mattrc',profile,'Direction',direction,'RenderingIntent', trc_intent)` adds the option of specifying the rendering intent.

`C = makecform('graytrc',profile,'Direction',direction)` creates a monochrome transformation based on a single-channel Tone Reproduction Curve (GrayTRC) contained in an ICC color profile.

`C = makecform('graytrc',profile,'Direction',direction,'RenderingIntent', trc_intent)` adds the option of specifying the rendering intent.

`C = makecform('clut',profile,LUTtype)` creates a color transformation based on a color lookup table of the type `LUTtype`, contained in an ICC color profile, `profile`.

`C = makecform('named',profile,space)` creates a color transformation from a named color profile (with a `'NamedColor2'` field) to coordinates in the color space `space`.

Examples

Convert sRGB Image to L*a*b*

Convert RGB image to *L*a*b**, assuming input image is *sRGB*.

```
rgb = imread('peppers.png');
cform = makecform('srgb2lab');
lab = applycform(rgb,cform);
```

Convert RGB to XYZ

Convert from a non-standard RGB color profile to the device-independent *XYZ* profile connection space. Note that the ICC input profile must include a `MatTRC` value.

```
InputProfile = iccread('myRGB.icc');
C = makecform('mattrc',InputProfile.MatTRC, ...
              'direction','forward');
```

Input Arguments

type — Color space conversion type

'cmyk2srgb' | 'srgb2cmyk' | 'lab2xyz' | 'xyz2lab' | ...

Color space conversion type, specified as one of the following character vectors. For a list of the abbreviations used by the Image Processing Toolbox software for each color space, see "More About" on page 1-2311.

Type	Description
'cmyk2srgb'	Convert from the *CMYK* color space to the *sRGB* color space.
'lab2lch'	Convert from the *L*a*b** to the *L*ch* color space.
'lab2srgb'	Use `lab2rgb` instead.
'lab2xyz'	Use `lab2xyz` instead.
'lch2lab'	Convert from the *L*ch* to the *L*a*b** color space.
'srgb2cmyk'	Convert from the *sRGB* to the *CMYK* color space.
'srgb2lab'	Use `rgb2lab` instead.
'srgb2xyz'	Use `rgb2xyz` instead.
'upvpl2xyz'	Convert from the *u′v′L* to the *XYZ* color space.
'uvl2xyz'	Convert from the *uvL* to the *XYZ* color space.
'xyl2xyz'	Convert from the *xyY* to the *XYZ* color space.
'xyz2lab'	Use `xyz2lab` instead.
'xyz2srgb'	Use `xyz2rgb` instead.
'xyz2upvpl'	Convert from the *XYZ* to the *u′v′L* color space.
'xyz2uvl'	Convert from the *XYZ* to the *uvL* color space.
'xyz2xyl'	Convert from the *XYZ* to the *xyY* color space.

Data Types: `char` | `string`

WP — White point

`[0.9642 1.0000 0.8249]` (default) | 1-by-3 numeric vector

Reference or adapted white point, specified as a 1-by-3 numeric vector of *XYZ* values, scaled so that *Y* = 1. Use the `whitepoint` function to create the `WP` vector. The default white point is the vector returned by `whitepoint('ICC')`.

To get an adaptive whitepoint value that is consistent with some published *sRGB* equations, set the value of `WP` to `[0.9504, 1.0000, 1.0888]`, which is the vector returned by `whitepoint('D65')`.

WPS, WPE — Starting or ending white point

1-by-3 numeric vector

Starting or ending white point used for a linear chromatic-adaptation transform, specified as a 1-by-3 numeric vector of *XYZ* values, scaled so that *Y* = 1. Use the `whitepoint` function to create the `WPS` or `WPE` vector.

`intent` — Rendering intent
`'Perceptual'` (default) | `'AbsoluteColorimetric'` | `'RelativeColorimetric'` | `'Saturation'`

Rendering intent, specified as `'Perceptual'`, `'AbsoluteColorimetric'`, `'RelativeColorimetric'`, or `'Saturation'`.

Rendering intents specify the style of reproduction that should be used when these profiles are combined. For most devices, the range of reproducible colors is much smaller than the range of colors represented by the PCS. Rendering intents define gamut mapping techniques. Each rendering intent has distinct aesthetic and color-accuracy trade-offs.

Value	Description
`'AbsoluteColorimetric'`	Maps all out-of-gamut colors to the nearest gamut surface while maintaining the relationship of all in-gamut colors. This absolute rendering contains color data that is relative to a perfectly reflecting diffuser.
`'Perceptual'` (default)	Employs vendor-specific gamut mapping techniques for optimizing the range of producible colors of a given device. The objective is to provide the most aesthetically pleasing result even though the relationship of the in-gamut colors might not be maintained. This media-relative rendering contains color data that is relative to the device's white point.
`'RelativeColorimetric'`	Maps all out-of-gamut colors to the nearest gamut surface while maintaining the relationship of all in-gamut colors. This media-relative rendering contains color data that is relative to the device's white point.
`'Saturation'`	Employs vendor-specific gamut mapping techniques for maximizing the saturation of device colors. This rendering is generally used for simple business graphics such as bar graphs and pie charts. This media-relative rendering contains color data that is relative to the device's white point.

`src_intent`, `dest_intent` — Source or destination rendering intent
`'Perceptual'` (default) | `'AbsoluteColorimetric'` | `'RelativeColorimetric'` | `'Saturation'`

Source or destination rendering intent for a color transformation between two ICC profiles, specified as `'Perceptual'`, `'AbsoluteColorimetric'`, `'RelativeColorimetric'`, or `'Saturation'`. For more information, see `intent`.

`trc_intent` — Rendering intent for tone reproduction curve
`'RelativeColorimetric'` (default) | `'AbsoluteColorimetric'`

Rendering intent for tone reproduction curve (MatTRC or grayTRC), specified as `'RelativeColorimetric'` or `'AbsoluteColorimetric'`. When `'AbsoluteColorimetric'` is specified, the colorimetry is referenced to a perfect diffuser, rather than to the media white point of the ICC color profile, `profile`. For more information, see `intent`.

`model` — Chromatic-adaptation model
`'Bradford'` (default) | `'vonKries'`

Chromatic-adaptation model used to create a linear chromatic-adaptation transform, specified as `'Bradford'` or `'vonKries'`.

`profile` — ICC color profile
`struct`

ICC color profile, specified as a structure as returned by `iccread`. If `profile` is a named color profile, it must have a `NamedColor2` field.

`src_profile, dest_profile` — Source or destination ICC color profile
`struct`

Source or destination ICC color profile, specified as a structure as returned by `iccread`.

`MatTRC` — Matrix/tone reproduction curve model
`struct`

Matrix/tone reproduction curve model, specified as a structure. `MatTRC` is typically obtained from the `'MatTRC'` field of an ICC profile structure returned by `iccread`, based on tags contained in an ICC color profile. The `MatTRC` model contains an *RGB*-to-*XYZ* matrix and *RGB* tone reproduction curves.

`direction` — Direction to apply tone reproduction curve model
`'forward'` | `'inverse'`

Direction to apply the tone reproduction curve model, specified as `'forward'` or `'inverse'`.

- For a multi-channel tone reproduction curve (`'mattrc'`), `'forward'` applies the model in the *RGB* to *XYZ* direction, and `'inverse'` applies the model in the *XYZ* to *RGB* direction. For more information, see section 6.3.1.2 of the International Color Consortium specification ICC.1:2001-04 or ICC.1:2001-12, available at https://www.color.org.
- For a single-channel tone reproduction curve (`'graytrc'`), `'forward'` applies the model in the device to PCS direction, and `'inverse'` applies the model in the PCS to device direction. "Device" here refers to the grayscale signal communicating with the monochrome device. "PCS" is the Profile Connection Space of the ICC profile and can be either *XYZ* or *L*a*b**, depending on the `'ConnectionSpace'` field in `profile.Header`.

`LUTtype` — Lookup table type
`'AToB0'` (default) | `'AToB1'` | `'BToA0'` | `'Gamut'` | `'Preview0'` | ...

Lookup table type, specified as one of the following values. `LUTtype` specifies which `'clut'` in the `profile` structure is to be used. Each `LUTtype` listed in the table below contains the components of an 8-bit or 16-bit LUTtag that performs a transformation between device colors and PCS colors using a particular rendering. For more information about `'clut'` transformations, see Section 6.5.7 of the International Color Consortium specification ICC.1:2001-04 (Version 2) or Section 6.5.9 of ICC.1:2001-12 (Version 4), available at https://www.color.org.

LUT Type	Description
`'AToB0'` (default)	Device to PCS: perceptual rendering intent
`'AToB1'`	Device to PCS: media-relative colorimetric rendering intent
`'AToB2'`	Device to PCS: saturation rendering intent
`'AToB3'`	Device to PCS: ICC-absolute rendering intent
`'BToA0'`	PCS to device: perceptual rendering intent
`'BToA1'`	PCS to device: media-relative colorimetric rendering intent
`'BToA2'`	PCS to device: saturation rendering intent
`'BToA3'`	PCS to device: ICC-absolute rendering intent

LUT Type	Description
`'Gamut'`	Determines which PCS colors are out of gamut for a given device
`'Preview0'`	PCS colors to the PCS colors available for soft proofing using the perceptual rendering
`'Preview1'`	PCS colors available for soft proofing using the media-relative colorimetric rendering.
`'Preview2'`	PCS colors to the PCS colors available for soft proofing using the saturation rendering.

space — Color space
`'PCS'` | `'Device'`

Color space, specified as `'PCS'` or `'Device'`. The `'PCS'` option is always available and will return *L*a*b** or *XYZ* coordinates, depending on the `'ConnectionSpace'` field in `profile.Header`. The `'Device'` option, when active, returns device coordinates, the dimension depending on the `'ColorSpace'` field in `profile.Header`. Coordinates are always returned in `'double'` format.

Output Arguments

C — Color transformation
`struct`

Color transformation structure, returned as a `struct`.

More About

Color Space Abbreviations

The Image Processing Toolbox software uses the following abbreviations to represent color spaces.

Abbreviation	Description
`xyz`	1931 CIE *XYZ* tristimulus values (2° observer)
`xyl`	1931 CIE *xyY* chromaticity values (2° observer), where `x` and `y` refer to the *xy*-coordinates of the associated CIE chromaticity diagram, and `l` refers to *Y* (luminance).
`uvl`	1960 CIE *uvY* values, where `u` and `v` refer to the *uv*-coordinates, and `l` refers to *Y* (luminance).
`upvpl`	1976 CIE *u′v′Y* values, where `up` and `vp` refer to the *u′v′*-coordinates and `l` refers to *Y* (luminance).
`lab`	1976 CIE *L*a*b** values. Note that `l` refers to *L** (CIE 1976 psychometric lightness) rather than luminance (*Y*).
`lch`	Polar transformation of CIE *L*a*b** values, where `c` = chroma and `h` = hue
`cmyk`	Standard values used by printers
`srgb`	Standard computer monitor RGB values, (`IEC 61966-2-1`)

References

[1] International Color Consortium. https://www.color.org.

See Also

`lab2rgb` | `lab2xyz` | `rgb2lab` | `rgb2xyz` | `xyz2lab` | `xyz2rgb` | `applycform` | `iccread` | `iccwrite` | `isicc` | `whitepoint`

Introduced before R2006a

makeConstrainToRectFcn

Create rectangularly bounded drag constraint function

Note `makeConstrainToRectFcn` is not recommended. With the new ROIs, use the `DrawingArea` property instead. For more information, see “Compatibility Considerations”.

Syntax

```
fcn = makeConstrainToRectFcn(roi,x,y)
```

Description

`fcn = makeConstrainToRectFcn(roi,x,y)` creates a position constraint function for draggable tools of a given ROI type. The position of the tool is constrained by rectangular boundaries described by position vectors `x` and `y`.

Examples

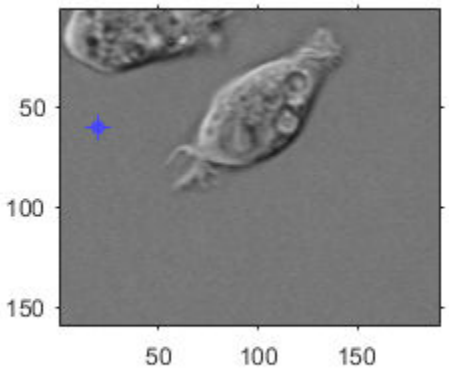
Constrain Drag of impoint to Image Limits

Display an image.

```
imshow('cell.tif')
```

Create an `impoint` object at the (*x*,*y*) coordinate (20,60). In an image, the positive *y* direction is downwards.

```
h = impoint(gca,20,60);
```



Make a function that constrains the `impoint` to the image limits.

```
x = get(gca,'XLim');
y = get(gca,'YLim');
fcn = makeConstrainToRectFcn('impoint',x,y);
```

Apply the constraint function to the `impoint`. Try dragging the point past the boundary of the image. The constraint function prevents the point from crossing the image boundary.

```
setPositionConstraintFcn(h,fcn);
```

Input Arguments

`roi` — ROI type

`'imellipse'` | `'imfreehand'` | `'imline'` | `'impoint'` | `'impoly'` | `'imrect'`

ROI type, specified as `'imellipse'`, `'imfreehand'`, `'imline'`, `'impoint'`, `'impoly'`, or `'imrect'`.

Data Types: `char` | `string`

`x` — Rectangular boundaries in the *x* direction

2-element numeric vector

Rectangular boundaries in the *x* direction, specified as a 2-element numeric vector of the form `[xmin xmax]`.

`y` — Rectangular boundaries in the *y* direction

2-element numeric vector

Rectangular boundaries in the *y* direction, specified as a 2-element numeric vector of the form `[ymin ymax]`.

Output Arguments

`fcn` — Function handle

handle

Function handle, returned as a handle. For more information, see “Create Function Handle”.

Compatibility Considerations

`makeConstrainToRectFcn` is not recommended

Not recommended starting in R2018b

Starting in R2018b, a new set of ROI objects replaces the existing set of ROI objects. The new objects provide more functional capabilities, such as face color transparency. The new classes also support events that you can use to respond to changes in your ROI such as moving or being clicked. Although there are no plans to remove the old ROI objects at this time, switch to the new ROIs to take advantage of the additional capabilities and flexibility. For more information on creating ROIs using the new ROI functions, see “Create ROI Shapes”.

With the new ROIs, you use the `DrawingArea` property of the ROI to specify the area in which you can draw or move an ROI.

Update Code

Update all instances of `makeConstrainToRectFcn`.

Discouraged Usage	Recommended Replacement

This example uses the `makeConstrainToRectFcn` function to create a function that limits the area in which you can create or move an ROI to the size of the underlying image. By default, you can move an ROI off the image area.
 	
`imshow('cell.tif')`
`h = impoint(gca,20,60);`
`% Make a function that constrains movement of the point`
`x = get(gca,'XLim');`
`y = get(gca,'YLim');`
`fcn = makeConstrainToRectFcn('impoint',x,y);`
`% Apply the constraint function to the ROI.`
`setPositionConstraintFcn(h,fcn);`
	Here is equivalent code, replacing use of `makeConstrainToRectFcn` function with the `DrawingArea` property of the ROI. By default, the new ROIs limit their creation and movement to the size of the underlying image, so there is no need to recreate that part of the example. Instead, this example creates a 10-pixel margin inside the image boundary where the ROI cannot go.
	`I = imread('cell.tif');`
	`imshow(I)`
	`h = drawpoint(gca,'Position',[20 60])`
	`[height width] = size(I); %Get image dimensions`
	`h.DrawingArea = [10,10,(width-20),(height-20)];`

See Also

`imdistline` | `imellipse` | `imfreehand` | `imline` | `impoint` | `impoly` | `imrect`

Topics

“ROI Migration”

Introduced in R2006a

makehdr

Create high dynamic range image

Syntax

```
HDR = makehdr(files)
HDR = makehdr(imds)
HDR = makehdr(___,Name,Value)
HDR = makehdr(images,Name,Value)
```

Description

`HDR = makehdr(files)` creates the single-precision, high dynamic range (HDR) image `HDR` from the set of spatially registered, low dynamic range (LDR) images in `files`.

`HDR = makehdr(imds)` creates the single-precision, high dynamic range image `HDR` from the set of spatially registered LDR images stored as `ImageDatastore` object, `imds`.

`HDR = makehdr(___,Name,Value)` uses name-value pairs to control various aspects of the image creation in addition to the input argument from any of the previous syntaxes.

Note The input image files must contain the Exchangeable Image File Format (EXIF) exposure metadata. `makehdr` uses the middle exposure between the brightest and darkest images as the base exposure for the HDR calculations. This value does not need to appear in any particular file. For more information about calculating this middle exposure value, see “Algorithms” on page 1-2324.

`HDR = makehdr(images,Name,Value)` creates the single-precision HDR image `HDR` from the set of spatially registered LDR images stored in a cell array `images`. Specify the exposure values for images in the input cell array by using the name-value pair `'ExposureValues'` or `'RelativeExposure'`.

Note When input is a cell array of LDR images, you must specify either the exposure or the relative exposure values as the second input argument. To specify the exposure values, use the name-value pair `'ExposureValues'`. To specify the relative exposure values, use the name-value pair `'RelativeExposure'`.

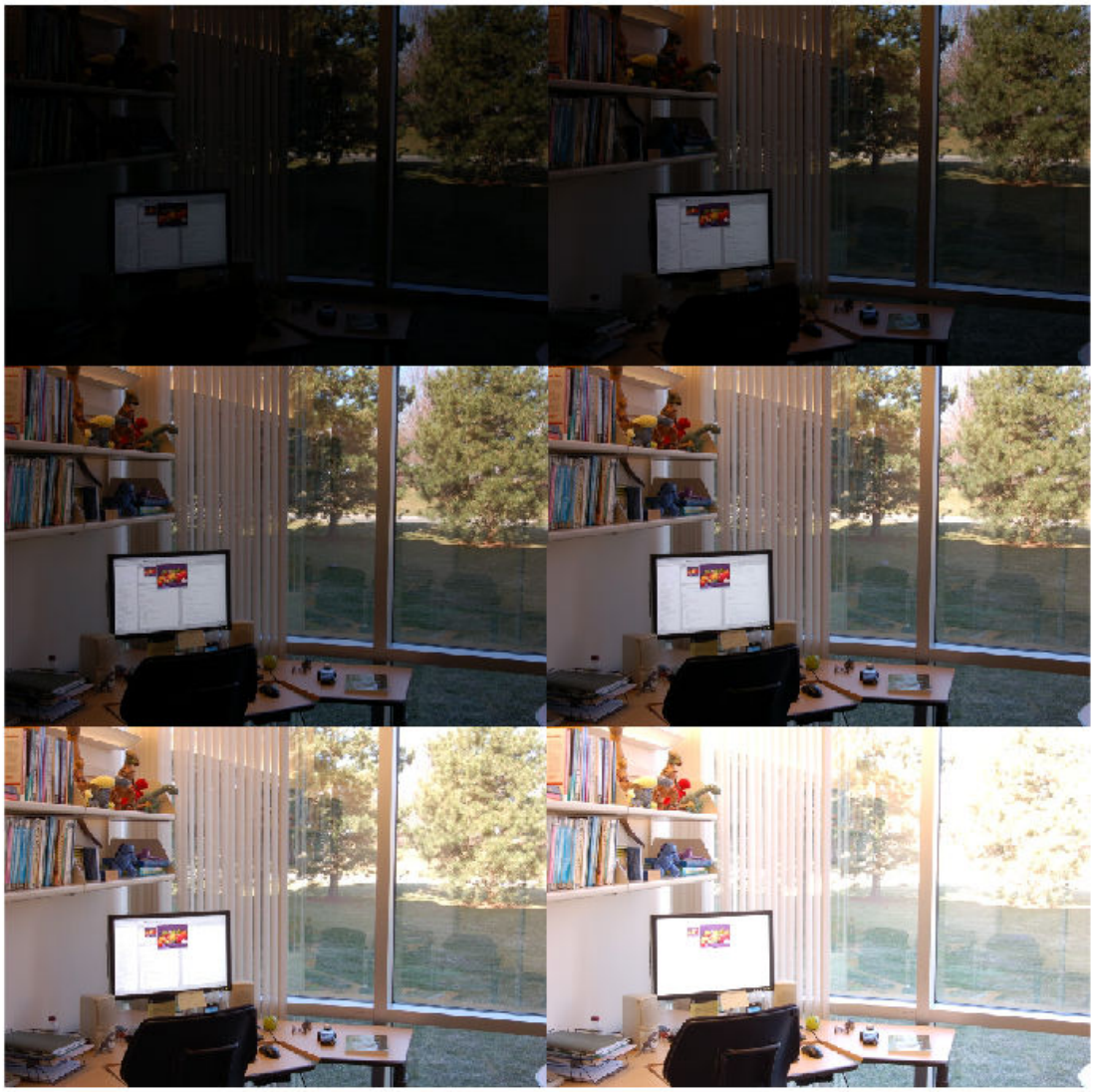
Examples

Create HDR Image from Set of LDR Images

Create a high dynamic range (HDR) image from a set of low dynamic range (LDR) images that share the same f-stop but have different exposure times.

Load six low dynamic range images into the workspace. Create a vector of their respective exposure times. Display the images as a montage.

```
files = {'office_1.jpg','office_2.jpg','office_3.jpg',...
         'office_4.jpg','office_5.jpg','office_6.jpg'};
expTimes = [0.0333 0.1000 0.3333 0.6250 1.3000 4.0000];
montage(files)
```



Combine the LDR images into an HDR image.

```
hdr = makehdr(files,'RelativeExposure',expTimes./expTimes(1));
```

Display the HDR image.

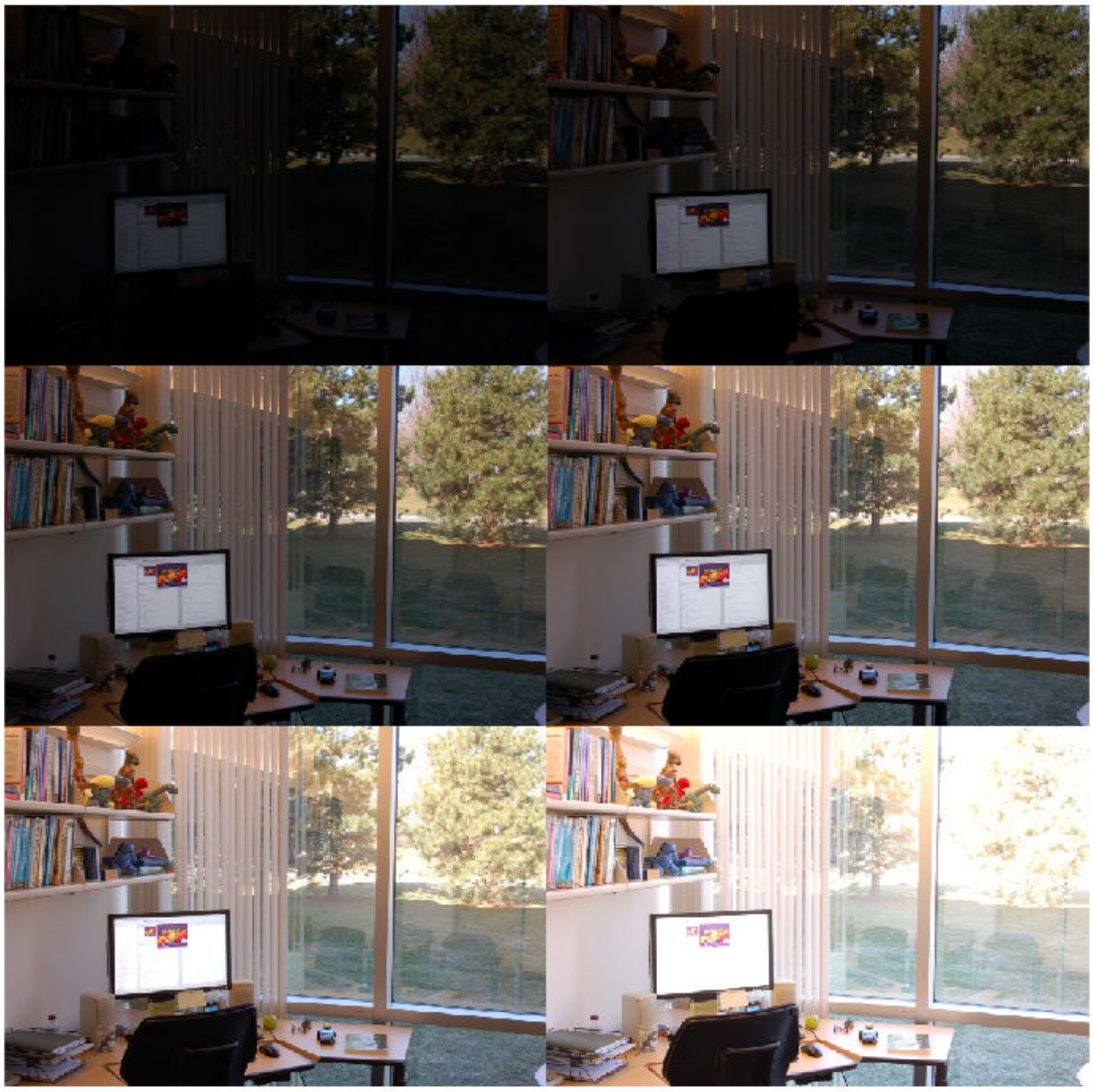
```
rgb = tonemap(hdr);
imshow(rgb)
```

Create HDR Image Using Camera Response Function

Create a high dynamic range (HDR) image from a set of six low dynamic range (LDR) images that share the same f-stop but have different exposure times. The estimated camera response function values are computed from these LDR images and used to generate an HDR image.

Read the set of six spatially registered, LDR images into the workspace. Create an `imageDatastore` object containing these images. Display the images as a montage.

```
setDir = fullfile(toolboxdir('images'),'imdata','office_*');
imds = imageDatastore(setDir);
montage(imds)
```



Estimate the camera response function from images in the datastore.

```
crf = camresponse(imds);
```

Combine the LDR images into an HDR image by using the estimated camera response function values.

```
hdr = makehdr(imds,'CameraResponse',crf);
```

Display the HDR image.

```
rgb = tonemap(hdr);
imshow(rgb)
```

Create HDR Image from Cell Array of LDR Images

Create a high dynamic range (HDR) image from a cell array of low dynamic range (LDR) images that share the same f-stop but have different exposure times.

Read six low dynamic range images into the workspace.

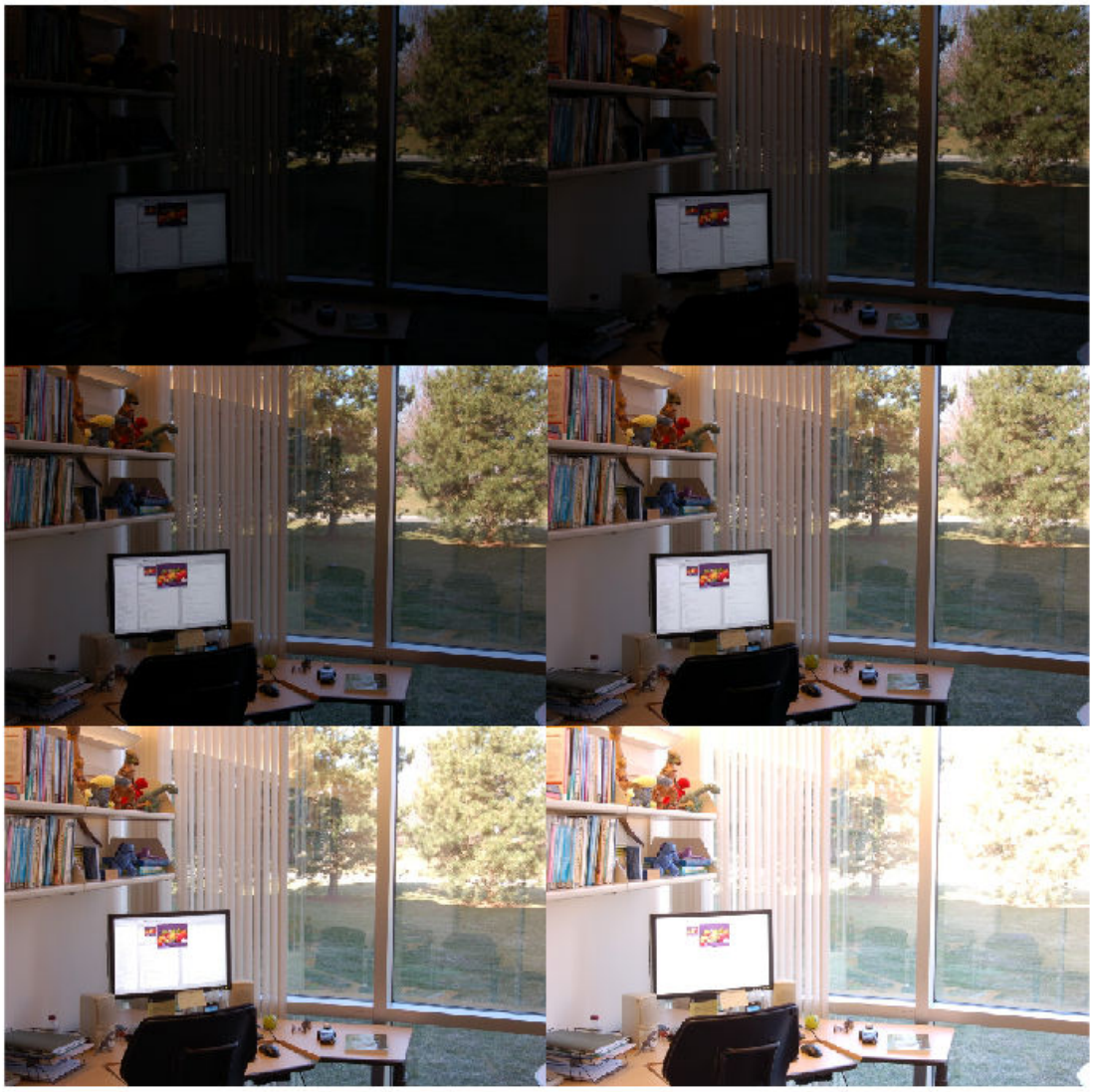
```
image1 = imread('office_1.jpg');
image2 = imread('office_2.jpg');
image3 = imread('office_3.jpg');
image4 = imread('office_4.jpg');
image5 = imread('office_5.jpg');
image6 = imread('office_6.jpg');
```

Create a cell array of LDR images in the workspace by using the cell construction operation, `{ }`.

```
images = {image1,image2,image3,image4,image5,image6};
```

Display the images as a montage.

```
montage(images)
```



Specify the exposure value for each LDR image in the input cell array.

```
exposure = [0.0333 0.1000 0.3333 0.6250 1.3000 4.0000];
```

Compute the relative exposure values with respect to the exposure value of the first LDR image in the input cell array.

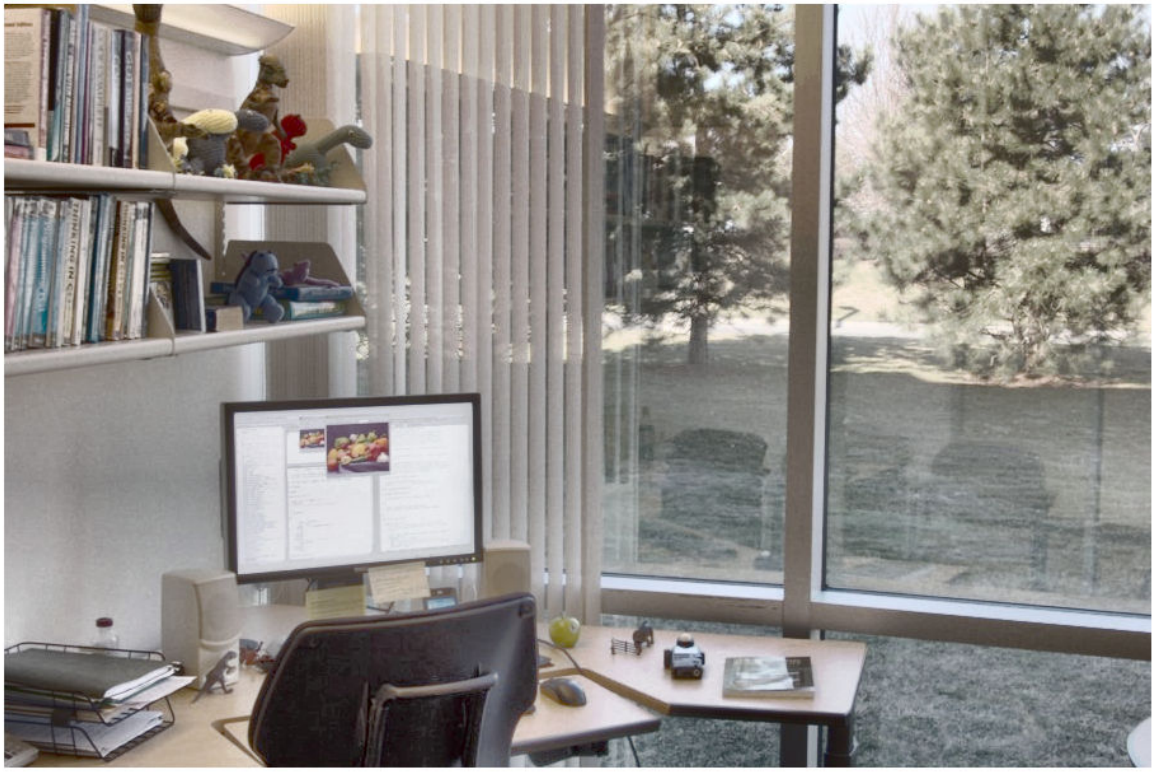
```
relExposure = exposure./exposure(1);
```

Combine the LDR images into an HDR image. Specify the relative exposure values for each image in the cell array.

```
hdr = makehdr(images,'RelativeExposure',relExposure);
```

Display the HDR image.

```
rgb = tonemap(hdr);
imshow(rgb)
```



Input Arguments

`files` — Set of spatially registered LDR images

string array | cell array of character vectors

Set of spatially registered LDR images, specified as a string array or a cell array of character vectors. These images can be color or grayscale of any bit depth. However, the preferred bit depth for LDR images is 8 or 16.

Data Types: `char` | `string` | `cell`

`imds` — Set of spatially registered LDR images

`ImageDatastore` object

Set of spatially registered LDR images, specified as an `ImageDatastore` object. These images can be color or grayscale of any bit depth. However, the preferred bit depth for LDR images is 8 or 16.

`images` — Set of spatially registered LDR images

cell array

Set of spatially registered LDR images, specified as a cell array. These images can be color or grayscale of any bit depth. However, the preferred bit depth for LDR images is 8 or 16.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `makehdr(files,'RelativeExposure',[0.1 0.3 0.4]);`

`BaseFile` — Name of file to use as base exposure

string scalar | character vector

Name of file to use as base exposure, specified as a string scalar or character vector.

Data Types: `char` | `string`

Note

- You can use only one of the `'BaseFile'`, `'ExposureValues'`, and `'RelativeExposure'` name-value pairs at a time.
- You must not specify `'BaseFile'` name-value pair, when the input is a cell array of spatially registered LDR images.

`ExposureValues` — Exposure value of each file in input set

numeric vector of positive values

Exposure value of each image in input set, specified as a numeric vector of positive values. The *k*th element in the vector corresponds to the *k*th LDR image in the input set. An increase of one exposure value (EV) corresponds to doubling the exposure. A decrease of one EV corresponds to halving the exposure. If you specify this parameter, the function overrides the EXIF exposure metadata.

Data Types: `single` | `double`

`RelativeExposure` — Relative exposure value of each file in input set

numeric vector of positive values

Relative exposure value of each image in input set, specified as a numeric vector of positive values. The *k*th element in the vector corresponds to the *k*th LDR image in the input set.

For example, an image with a relative exposure (RE) value of 0.5 has half as much exposure as an image with an RE value of 1. Similarly, an image with an RE value of 3 has three times the exposure of an image with an RE value of 1. If you specify this parameter, the function overrides the EXIF exposure metadata.

Data Types: `single` | `double`

`MinimumLimit` — Minimum correctly exposed value

positive integer

Minimum correctly exposed value, specified as a positive integer. For each LDR image, pixels with a smaller value than this minimum are considered underexposed and do not contribute to the final HDR

image. By default, this minimum value is set to 2% of the maximum intensity allowed by the image data type.

Data Types: `single` | `double`

`MaximumLimit` — Maximum correctly exposed value
positive integer

Maximum correctly exposed value, specified as a positive integer. For each LDR image, pixels with a larger value than this maximum are considered overexposed and do not contribute to the final HDR image. By default, this maximum value is set to 98% of the maximum intensity allowed by the image data type.

Data Types: `single` | `double`

`CameraResponse` — Camera response function
n-by-1 vector | *n*-by-3 matrix

Camera response function, specified as a *n*-by-1 vector for grayscale images and *n*-by-3 matrix for color images. The camera response function maps the log-exposure value (scene radiance) to the intensity levels in the input images. The value of *n* is $2^{bit\ depth}$. For example, if the bit depth of the input set of images is 8, then *n* is 256.

Note The `'MaximumLimit'` and `'MinimumLimit'` name-value pairs are ignored when `'CameraResponse'` is specified.

Data Types: `single` | `double`

Output Arguments

`HDR` — High dynamic range image
m-by-*n*-by-3 numeric array

High dynamic range image, returned as an *m*-by-*n*-by-3 numeric array.

Data Types: `single`

Algorithms

The `makehdr` function calculates the middle exposure value by using the exposure values (EVs) of the input images. The exposure value for each image is computed based on the aperture and shutter speed. The aperture and shutter speed values are stored in the EXIF metadata of that input file or is specified using the `'ExposureValues'` name-value pair. The middle EV is calculated as an average of the highest and lowest EVs and is used as the base exposure.

References

[1] Reinhard et al. *High Dynamic Range Imaging* 2006. Ch. 4.

[2] Debevec, P.E., and J. Malik. "Recovering High Dynamic Range Radiance Maps from Photographs." In *ACM SIGGRAPH 2008 classes*, Article No. 31. New York, NY: ACM, 2008.

See Also

`hdrread` | `hdrwrite` | `localtonemap` | `tonemap` | `tonemapfarbman` | `camresponse`

Topics

“Work with High Dynamic Range Images”
“Image Types in the Toolbox”

Introduced in R2008a

makelut

Create lookup table for use with `bwlookup`

Syntax

```
lut = makelut(fun,n)
```

Description

`lut = makelut(fun,n)` creates a lookup table. `fun` is a function that creates a numeric output from a binary neighborhood of size `n`-by-`n`. The function creates a lookup table by passing all possible neighborhoods to `fun`, one at a time, and storing the outputs in vector `lut`.

Use the lookup table with `bwlookup` to perform nonlinear neighborhood filtering.

Examples

Make Lookup Table for 2-by-2 Neighborhood

Create a lookup table for 2-by-2 neighborhoods. In this example, the function passed to `makelut` returns `true` if the number of `1`s in the neighborhood is 2 or greater, and returns `false` otherwise.

```
f = @(x) (sum(x(:)) >= 2);
lut = makelut(f,2)
```

```
lut = 16×1

     0
     0
     0
     1
     0
     1
     1
     1
     0
     1
     ⋮
```

Input Arguments

`fun` — Function handle

handle

Function handle, specified as a handle. The function must accept an `n`-by-`n` binary matrix of `1`s and `0`s as input and return a scalar.

For more information about function handles, see “Create Function Handle”.

n — Neighborhood size

2 | 3

Neighborhood size for the lookup table, specified as 2 or 3.

Output Arguments

lut — Lookup table

16-element numeric vector | 512-element numeric vector

Lookup table, returned as a 16-element numeric vector when n is 2, or a 512-element numeric vector when n is 3.

Data Types: double

See Also

bwlookup

Introduced before R2006a

makeresampler

Create resampling structure

Syntax

```
R = makeresampler(interpolant,padMethod)
R = makeresampler(Name,Value)
```

Description

`R = makeresampler(interpolant,padMethod)` creates a separable resampler structure for use with `tformarray`. The `interpolant` argument specifies the interpolating kernel that the separable resampler uses. The `padMethod` argument controls how the resampler interpolates or assigns values to output elements that map close to or outside the edge of the input array.

`R = makeresampler(Name,Value)` creates a user-written resampler using name-value arguments.

Examples

Use Separable Resampler to Stretch an Image in the Y Direction

Read an image into the workspace and display it.

```
A = imread('moon.tif');
imshow(A)
```

Create a separable resampler.

```
resamp = makeresampler({'nearest','cubic'},'fill');
```

Create a spatial transformation structure (TFORM) that defines an affine transformation.

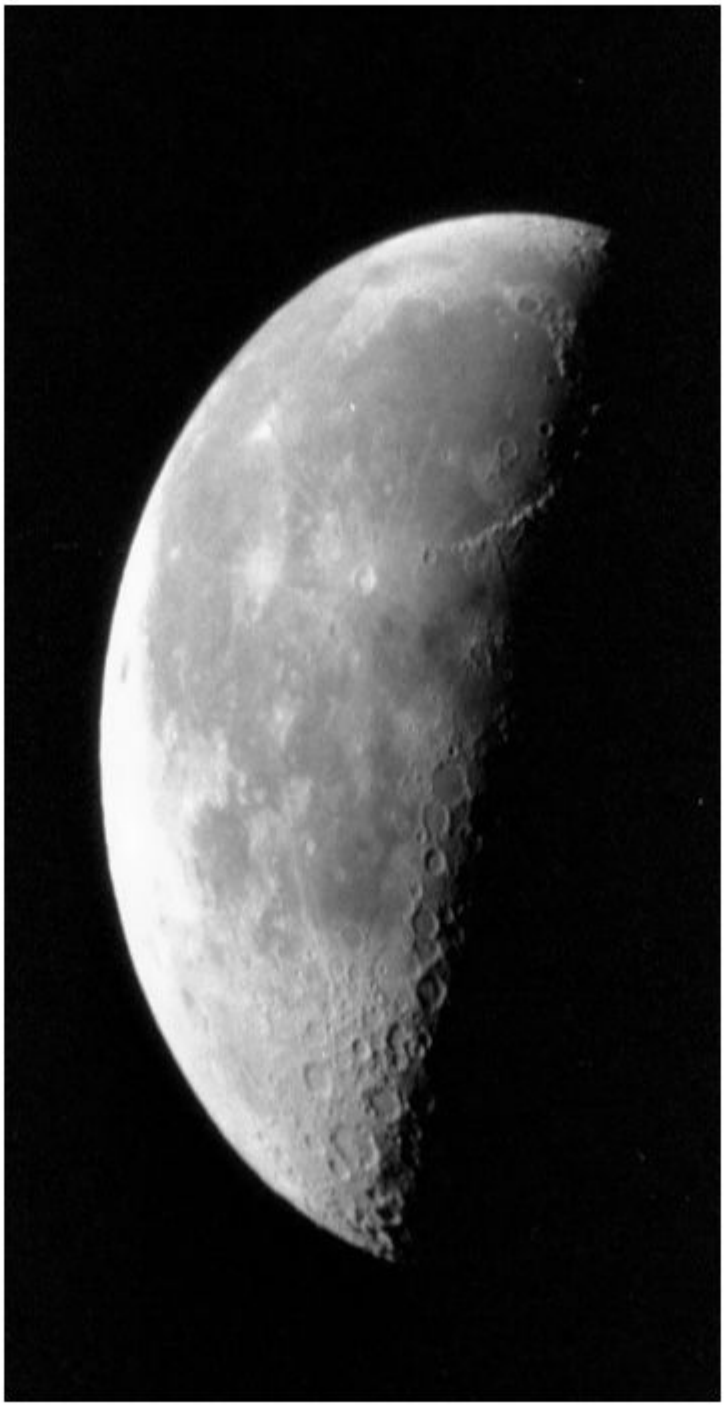
```
stretch = maketform('affine',[1 0; 0 1.3; 0 0]);
```

Apply the transformation, specifying the custom resampler.

```
B = imtransform(A,stretch,resamp);
```

Display the transformed image.

```
imshow(B)
```



Input Arguments

`interpolant` — Interpolation kernel

`"cubic"` | `"linear"` | `"nearest"` | cell array

Interpolation kernel, specified as `"nearest"`, `"linear"`, `"cubic"`, or a cell array. These kernels perform nearest-neighbor, bilinear, bicubic, and custom interpolation, respectively.

Define a custom interpolation kernel as a two-element cell array in either of these forms:

Form	Description
`{half_width, positive_half}`	`half_width` is a positive scalar designating the half width of a symmetric interpolating kernel. `positive_half` is a vector of values regularly sampling the kernel on the closed interval `[0 positive_half]`.
`{half_width, interp_fcn}`	`interp_fcn` is a function handle that returns interpolating kernel values, given an array of input values in the interval `[0 positive_half]`.

You can define the interpolation method independently along each transform dimension by specifying a cell array whose number of elements is equal to the number of transform dimensions. Each element of the cell array must be one of the prior types of interpolant kernels. For example, consider this value of `interpolant` for a 3-D interpolation kernel:

```
{"nearest","linear",{2 KERNEL_TABLE}}
```

In this example, the resampler uses nearest-neighbor interpolation along the first transform dimension, linear interpolation along the second dimension, and custom table-based interpolation along the third.

Data Types: `char` | `string` | `cell`

`padMethod` — Pad method

`"bound"` | `"circular"` | `"replicate"` | `"symmetric"` | `"fill"`

Pad method used to assign values to output elements that map outside the input array, specified as one of the these values.

Pad Method	Description
`"bound"`	Assign values from the fill value array to points that map outside the input array. Repeat border elements of the array for points that map inside the array (same as `"replicate"`). When `interpolant` is `"nearest"`, this pad method produces the same results as `"fill"`. `"bound"` is like `"fill"`, but avoids mixing fill values and input image values.
`"circular"`	Pad array with circular repetition of elements within the dimension. Same as `padarray`.
`"fill"`	Generate an output array with smooth-looking edges (except when using nearest-neighbor interpolation). For output points that map near the edge of the input array (either inside or outside), it combines input image and fill values. When `interpolant` is `"nearest"`, this pad method produces the same results as `"bound"`.

Pad Method	Description
`"replicate"`	Pad array by repeating border elements of array. Same as `padarray`.
`"symmetric"`	Pad array with mirror reflections of itself. Same as `padarray`.

For `"fill"`, `"replicate"`, `"circular"`, or `"symmetric"`, the resampling performed by `tformarray` occurs in two logical steps:

1. Pad the array `A` infinitely to fill the entire input transform space.
2. Evaluate the convolution of the padded `A` with the resampling kernel at the output points specified by the geometric map.

Each nontransform dimension is handled separately. The padding is virtual (accomplished by remapping array subscripts) for performance and memory efficiency. If you implement a custom resampler, you can implement these behaviors.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"Type","separable"` creates a separable resampler

Type — Resampler type

`"separable"` | `"custom"`

Resampler type, specified as one of the following values.

Type	Description
`"separable"`	Create a separable resampler. If you specify this value, the only other arguments that you can specify are `Interpolant` and `PadMethod`. The result is equivalent to using the `makeresampler(interpolant,padMethod)` syntax.
`"custom"`	Create a customer resampler. If you specify this value, you must specify the `NDims` and `ResampleFcn` arguments and, optionally, the `CustomData` argument.

Data Types: `char` | `string`

PadMethod — Pad method

character vector | string scalar

See the `padMethod` argument for more information.

Data Types: `char` | `string`

Interpolant — Interpolation kernel

character vector | string scalar | cell array

See the `interpolant` argument for more information.

Data Types: `char` | `string` | `cell`

`NDims` — Dimensionality custom resampler can handle
positive integer

Dimensionality custom resampler can handle, specified as a positive integer. Use a value of `Inf` to indicate that the custom resampler can handle any dimension. If `"Type"` is `"custom"`, you must specify `NDims`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`ResampleFcn` — Function that performs the resampling
function handle

Function that performs the resampling, specified as a function handle. You call this function with the following interface:

```
B = resample_fcn(A,M,TDIMS_A,TDIMS_B,FSIZE_A,FSIZE_B,F,R)
```

For more information about the input arguments to this function, see the help for `tformarray`. The argument `M` is an array that maps the transform subscript space of `B` to the transform subscript space of `A`. If `A` has `N` transform dimensions `(N = length(TDIMS_A))` and `B` has `P` transform dimensions `(P = length(TDIMS_B))`, then `ndims(M) = P + 1`, if `N > 1` and `P` if `N == 1`, and `size(M,P + 1) = N`.

The first `P` dimensions of `M` correspond to the output transform space, permuted according to the order in which the output transform dimensions are listed in `TDIMS_B`. (In general `TDIMS_A` and `TDIMS_B` need not be sorted in ascending order, although some resamplers can impose such a limitation.) Thus, the first `P` elements of `size(M)` determine the sizes of the transform dimensions of `B`. The input transform coordinates to which each point is mapped are arrayed across the final dimension of `M`, following the order given in `TDIMS_A`. `M` must be `double`. `FSIZE_A` and `FSIZE_B` are the full sizes of `A` and `B`, padded with `1`'s as necessary to be consistent with `TDIMS_A`, `TDIMS_B`, and `size(A)`.

Data Types: `function_handle`

`CustomData` — User-defined data
numeric array | string scalar | character vector

User-defined data, specified using a string scalar, character vector, or numeric array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `string`

Output Arguments

`R` — Resampler
structure

Resampler, returned as a structure.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

`tformarray`

Introduced before R2006a

maketform

Create N-D spatial transformation structure (TFORM)

Note The `maketform` function is not recommended for 2-D and 3-D geometric transformations. For more information, see “Compatibility Considerations”.

Syntax

```
T = maketform('affine',A)
T = maketform('projective',P)

T = maketform('custom',ndims_in,ndims_out,forward_fcn,inverse_fcn,tdata)

T = maketform('box',tsize,outCornerStart,outCornerEnd)
T = maketform('box',inCorners,outCorners)

T = maketform('composite',T1,T2,...,TL)
T = maketform('composite',[T1,T2,...,TL])
```

Description

Create N-D Affine and Projective Transformations from Matrices

`T = maketform('affine',A)` creates a spatial transformation structure `T` for an N-dimensional affine transformation specified as matrix `A`. The transformation structure `T` has both forward and inverse transformations.

A spatial transformation structure (called a `TFORM` structure) can be used with the `tformarray`, `tformfwd`, and `tforminv` functions.

`T = maketform('projective',P)` creates a `TFORM` structure for an N-dimensional projective transformation specified as matrix `P`. `T` has both forward and inverse transformations.

Create Transformation from Forward or Inverse Functions

`T = maketform('custom',ndims_in,ndims_out,forward_fcn,inverse_fcn,tdata)` creates a custom `TFORM` structure `T` based on user-provided function handles and parameters. `ndims_in` and `ndims_out` are the numbers of input and output dimensions. `forward_fcn` and `inverse_fcn` are function handles to forward and inverse functions. The `tdata` argument can be any MATLAB array and is typically used to store parameters of the custom transformation. It is accessible to `forward_fcn` and `inverse_fcn` via the `tdata` field of `T`.

Create Transformation for Spatial Referencing

`T = maketform('box',tsize,outCornerStart,outCornerEnd)` creates an N-dimensional affine `TFORM` structure `T` that maps an input box defined by the coordinates of a corner, `ones(1,N)`, and size `tsize`, to an output box defined by the opposite corners `outCornerStart` and `outCornerEnd`. The `'box'` `TFORM` structure is typically used to register the row and column subscripts of an image or array to some world coordinate system.

`T = maketform('box',inCorners,outCorners)` creates an N-dimensional affine `TFORM` structure `T`. The transformation maps an input box defined by the opposite corners `inCorners(1,:)` and `inCorners(2,:)` to an output box defined by the opposite corners `outCorners(1,:)` and `outCorners(2,:)`.

Create Composite Transformation

`T = maketform('composite',T1,T2,...,TL)` creates a `TFORM` structure `T` that is a composite of transformations `T1, T2, ..., TL` specified as comma-separated `TFORM` structures. The forward and inverse functions of `T` are the functional compositions of the forward and inverse functions of the component transformations `T1, T2, ..., TL`.

`T = maketform('composite',[T1,T2,...,TL])` builds a `TFORM` structure `T` that is a composite of transformations `T1, T2, ..., TL` specified in a vector. The forward and inverse functions of `T` are the functional compositions of the forward and inverse functions of the component transformations `T1, T2, ..., TL`.

Examples

Make TFORM and Apply Transformation to Image

Create a `TFORM` structure that defines an affine transformation.

```
T = maketform('affine',[.5 0 0; .5 2 0; 0 0 1])
```

```
T =

  struct with fields:

       ndims_in: 2
      ndims_out: 2
    forward_fcn: @fwd_affine
    inverse_fcn: @inv_affine
          tdata: [1×1 struct]
```

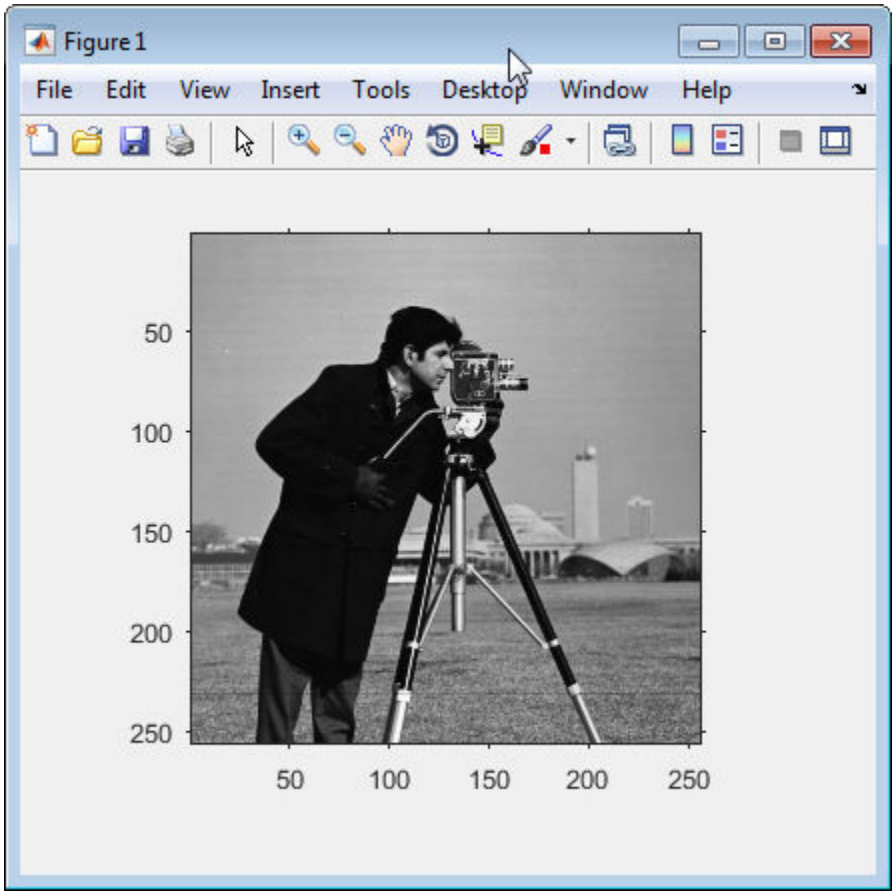
Apply the forward transformation.

```
tformfwd([10 20],T)
```

```
ans =

    15    40
```

Read an image into the workspace and display it.

```
I = imread('cameraman.tif');
imshow(I)
```

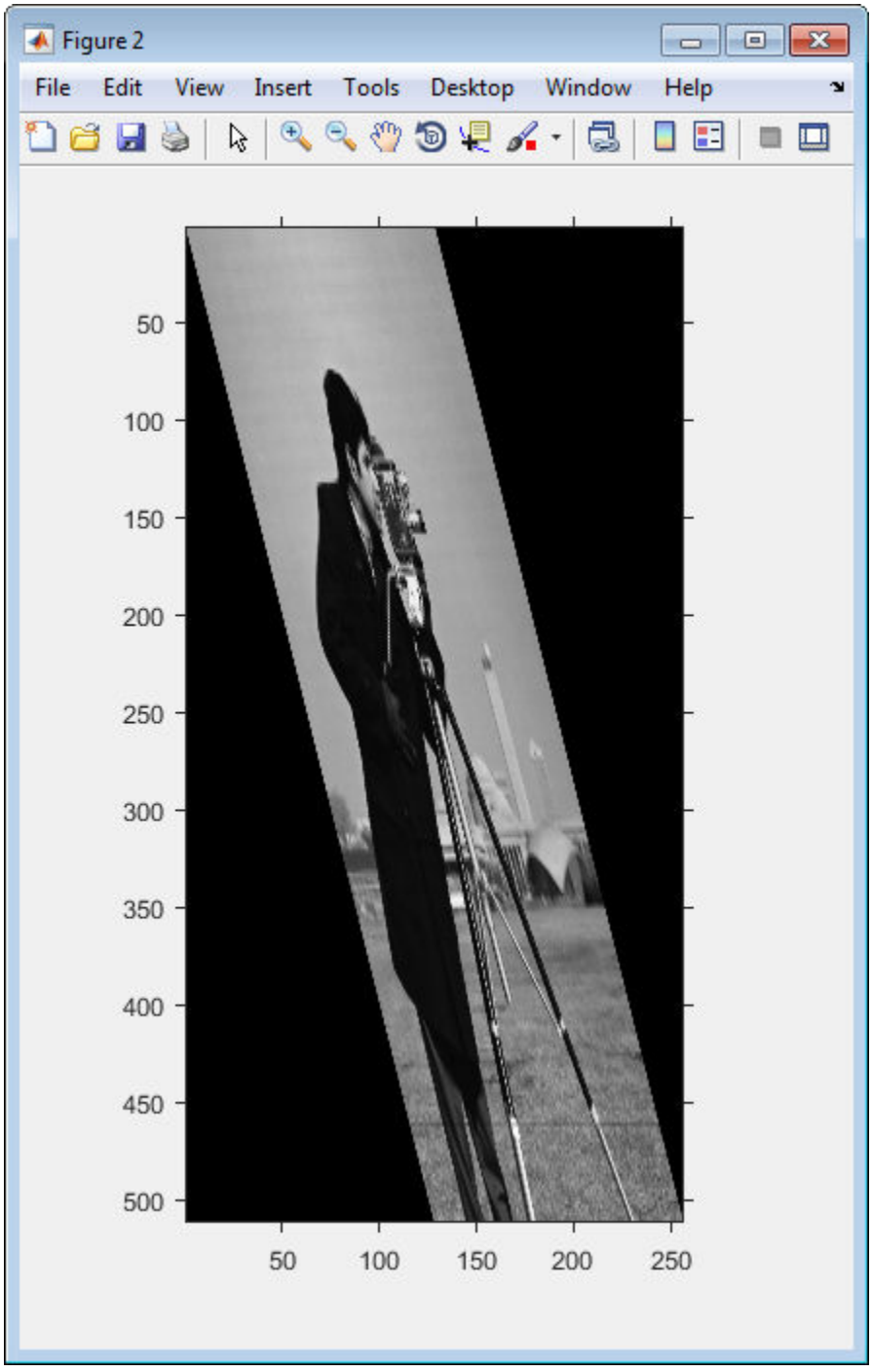


Apply the transformation to the image.

```
I2 = imtransform(I,T);
```

Display the original image and the transformed image.

```
imshow(I2)
```



Input Arguments

A — Affine transformation

(*N*+1)-by-(*N*+1) matrix | (*N*+1)-by-*N* matrix

Affine transformation, specified as an (*N*+1)-by-(*N*+1) matrix or an (*N*+1)-by-*N* matrix, where *N* is the dimensionality of the affine transformation. The matrix must be nonsingular and real.

If `A` is (*N*+1)-by-(*N*+1), the last column of `A` must be `[zeros(N,1);1]`. Otherwise, `A` is augmented automatically, such that its last column is `[zeros(N,1);1]`. The matrix `A` defines a forward transformation such that `tformfwd(U,T)`, where `U` is a 1-by-*N* vector, returns a 1-by-*N* vector `X`, such that `X = U * A(1:N,1:N) + A(N+1,1:N)`.

Data Types: `double`

`P` — Projective transformation
(*N*+1)-by-(*N*+1) matrix

Projective transformation, specified as an (*N*+1)-by-(*N*+1) matrix, where *N* is the dimensionality of the projective transformation. The matrix must be nonsingular and real. `P(N+1,N+1)` cannot be 0.

The matrix `P` defines a forward transformation such that `tformfwd(U,T)`, where `U` is a 1-by-*N* vector, returns a 1-by-*N* vector `X`, such that `X = W(1:N)/W(N+1)`, where `W = [U 1] * P`.

Data Types: `double`

`ndims_in` — Number of input dimensions
positive integer

Number of input dimensions, specified as a positive integer.

Data Types: `double`

`ndims_out` — Number of output dimensions
positive integer

Number of output dimensions, specified as a positive integer.

Data Types: `double`

`forward_fcn` — Forward function
function handle | `[]`

Forward function, specified as a function handle that supports the syntax `X = forward_fcn(U,T)`. `U` is a *numpts*-by-`ndims_in` matrix whose rows are points in the transformation input space and `X` is a *numpts*-by-`ndims_out` matrix whose rows are points in the transformation output space.

`forward_fcn` can be empty.

Data Types: `function_handle`

`inverse_fcn` — Inverse function
function handle

Inverse function, specified as a function handle that supports the syntax `U = inverse_fcn(X,T)`. `U` is a *numpts*-by-`ndims_in` matrix whose rows are points in the transformation input space and `X` is a *numpts*-by-`ndims_out` matrix whose rows are points in the transformation output space.

`inverse_fcn` can be empty. However, to use the `TFORM` struct `T` with the `tformarray` function, you must define `inverse_fcn`.

Data Types: `function_handle`

`tdata` — Parameters of custom transformation
array

Parameters of custom transformation, specified as an array.

Data Types: `double`

`tsize` — Size of input box

N-element vector of positive integers

Size of input box, specified as an N-element vector of positive integers.

Data Types: `double`

`outCornerStart` — Starting corner coordinates in output space

N-element vector

Starting corner coordinates in the output space, specified as an N-element vector. `outCornerStart`(*k*) and `outCornerEnd`(*k*) must be different unless `tsize`(*k*) is 1, in which case the affine scale factor along the *k*-th dimension is assumed to be 1.0.

Data Types: `double`

`outCornerEnd` — Opposite corner coordinates in output space

N-element vector

Opposite corner coordinates in the output space, specified as an N-element vector. `outCornerStart`(*k*) and `outCornerEnd`(*k*) must be different unless `tsize`(*k*) is 1, in which case the affine scale factor along the *k*-th dimension is assumed to be 1.0.

Data Types: `double`

`inCorners` — Corner coordinates in input space

N-by-2 numeric matrix

Corner coordinates in the input space, specified as an N-by-2 numeric matrix. The first column represents the coordinates of one corner and the second column represents the coordinates of the opposite corner. `inCorners`(1,*k*) and `inCorners`(2,*k*) must be different unless `outCorners`(1,*k*) and `outCorners`(2,*k*) are the same.

Data Types: `double`

`outCorners` — Corner coordinates in output space

N-by-2 numeric matrix

Corner coordinates in the output space, specified as an N-by-2 numeric matrix. The first column represents the coordinates of one corner and the second column represents the coordinates of the opposite corner. `outCorners`(1,*k*) and `outCorners`(2,*k*) must be different unless `inCorners`(1,*k*) and `inCorners`(2,*k*) are the same.

Data Types: `double`

`T1,T2,...,TL` — Component transformations

`TFORM` structures

Component transformations, specified as `TFORM` structures.

The inputs `T1, T2, ..., TL` are ordered just as they would be when using the standard notation for function composition: `T = T1` ∘ `T2` ∘ `...` ∘ `TL`. Composition is associative, but not commutative. This means that to apply `T` to the input `U`, you must apply `TL` first and `T1` last. Thus if `L = 3`, for

example, then `tformfwd(U,T)` is the same as `tformfwd(tformfwd(tformfwd(U,T3),T2),T1)`. The components `T1` through `TL` must be compatible in terms of the numbers of input and output dimensions.

`T` has a defined forward transformation function only if all the component transformations have defined forward transform functions. `T` has a defined inverse transformation function only if all the component transformations have defined inverse transform functions.

Data Types: `function_handle`

Output Arguments

T — Multidimensional spatial transformation

`TFORM` structure

Multidimensional spatial transformation, returned as a `TFORM` structure.

Compatibility Considerations

`maketform` is not recommended for 2-D and 3-D geometric transformations

Not recommended starting in R2018b

The `maketform` function is not recommended for 2-D and 3-D geometric transformations. Instead, create a 2-D or 3-D geometric transformation in these ways:

- Define a 2-D affine or projective transformation from a 3-by-3 matrix using the `affine2d` or `projective2d` objects.
- Define a 3-D affine transformation from a 4-by-4 matrix using the `affine3d` object.
- Define a 2-D affine, projective, polynomial, or other nonlinear transformation from pairs of control points using the `fitgeotrans` function.
- Define a 2-D or 3-D geometric transformation from point-wise mapping functions using the `geometricTransform2d` or `geometricTransform3d` object.

For more information about 2-D and 3-D geometric transformation objects, see “2-D and 3-D Geometric Transformation Process Overview”.

This table shows a few syntaxes of `maketform` with the recommended replacement code.

Discouraged Usage	Recommended Replacement

Create a 2-D affine transformation from a 3-by-3 matrix `A`.
 	
`A = [0.5 0 0; 0.5 2 0; 0 0 1];`
`T = maketform('affine',A);`
	Create an `affine2d` object.
	
	`A = [0.5 0 0; 0.5 2 0; 0 0 1];`
	`T = affine2d(A);`

Create a 2-D affine transformation that maps each row of `U` to the corresponding row of `X`. The `U` and `X` arguments define the corners of triangles as a 3-by-2 matrix.
 	
`U = [21.6 64.2; 71.1 70.3; 28.7 48.3];`
`X = [10.7 30.6; 40.5 50.6; 20.6 10.7];`
`T = maketform("affine",U,X);`
	Create an `affine2d` object using the `fitgeotrans` function.
	
	`U = [21.6 64.2; 71.1 70.3; 28.7 48.3];`
	`X = [10.7 30.6; 40.5 50.6; 20.6 10.7];`
	`T = fitgeotrans(U,X,"affine");`

Discouraged Usage	Recommended Replacement

Create a 2-D projective transformation that maps each row of `U` to the corresponding row of `X`. The `U` and `X` arguments define the corners of quadrangles as a 4-by-2 matrix.
 	
`U = [121 94; 319 79; 128 292; 352 281];`
`X = [165 113; 354 130; 143 285; 354 312];`
`T = maketform("projective",U,X);`
	Create a `projective2d` object using the `fitgeotrans` function.
	
	`U = [121 94; 319 79; 128 292; 352 281];`
	`X = [165 113; 354 130; 143 285; 354 312];`
	`T = fitgeotrans(U,X,"projective");`

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

`tformfwd` | `tforminv` | `fliptform` | `tformarray`

Topics

“N-Dimensional Spatial Transformations”

Introduced before R2006a

mat2gray

Convert matrix to grayscale image

Syntax

```
I = mat2gray(A,[amin amax])
I = mat2gray(A)
```

Description

`I = mat2gray(A,[amin amax])` converts the matrix `A` to a grayscale image `I` that contains values in the range 0 (black) to 1 (white). `amin` and `amax` are the values in `A` that correspond to 0 and 1 in `I`. Values less than `amin` are clipped to 0, and values greater than `amax` are clipped to 1.

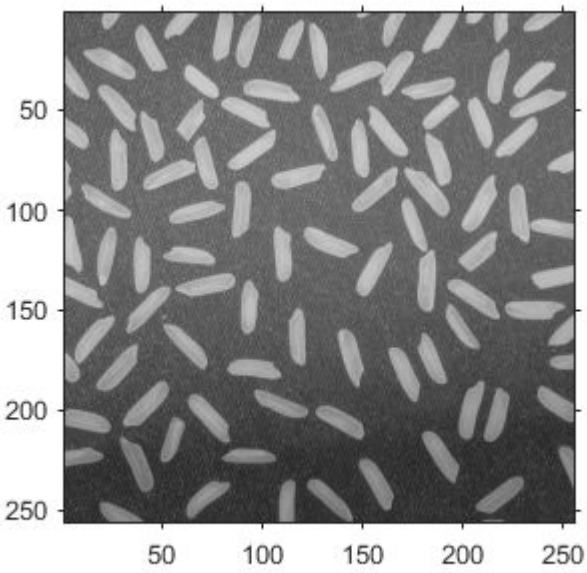
`I = mat2gray(A)` sets the values of `amin` and `amax` to the minimum and maximum values in `A`.

Examples

Convert a Matrix into an Image

Read an image and display it.

```
I = imread('rice.png');
figure
imshow(I)
```



Perform an operation that returns a numeric matrix. This operation looks for edges.

```
J = filter2(fspecial('sobel'),I);
min_matrix = min(J(:))
```

```
min_matrix = -779
```

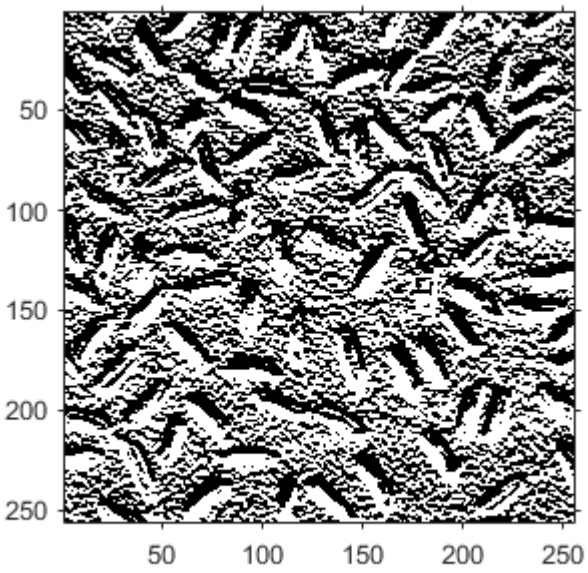
```
max_matrix = max(J(:))
```

```
max_matrix = 560
```

Note that the matrix has data type `double` with values outside of the range [0,1], including negative values.

Display the result of the operation. Because the data range of the matrix is outside the default display range of `imshow`, every pixel with a positive value displays as white, and every pixel with a negative or zero value displays as black. It is challenging to see the edges of the grains of rice.

```
figure
imshow(J)
```



Convert the matrix into an image. Display the maximum and minimum values of the image.

```
K = mat2gray(J);
min_image = min(K(:))
```

```
min_image = 0
```

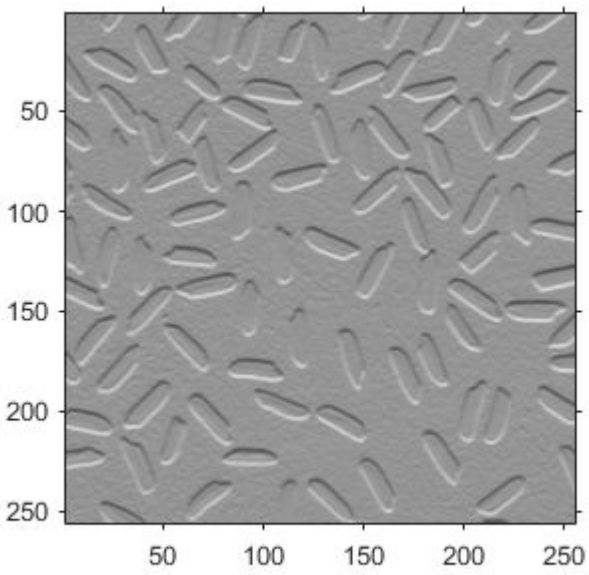
```
max_image = max(K(:))
```

```
max_image = 1
```

Note that values are still data type `double`, but that all values are in the range [0, 1].

Display the result of the conversion. Pixels show a range of grayscale colors, which makes the location of the edges more apparent.

```
figure
imshow(K)
```



Input Arguments

`A` — Input image

numeric matrix

Input image, specified as a numeric matrix.

`[amin amax]` — Input black and white values

2-element numeric vector

Input black and white values, specified as a 2-element numeric vector.

- Values in input image `A` that are less than or equal to `amin` are mapped to the value 0 in the intensity image, `I`.
- Values in `A` that are greater than or equal to `amax` are mapped to the value 1 in `I`.

Output Arguments

`I` — Output intensity image

numeric matrix

Output intensity image, returned as a numeric matrix with values in the range [0, 1].

Data Types: `double`

Extended Capabilities

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`rescale` | `gray2ind` | `ind2gray` | `im2gray`

Introduced before R2006a

maxhessiannorm

Maximum of Frobenius norm of Hessian of matrix

Syntax

```
C = maxhessiannorm(I)
C = maxhessiannorm(I,thickness)
```

Description

`C = maxhessiannorm(I)` returns the maximum of Frobenius norm of the Hessian of grayscale image `I`.

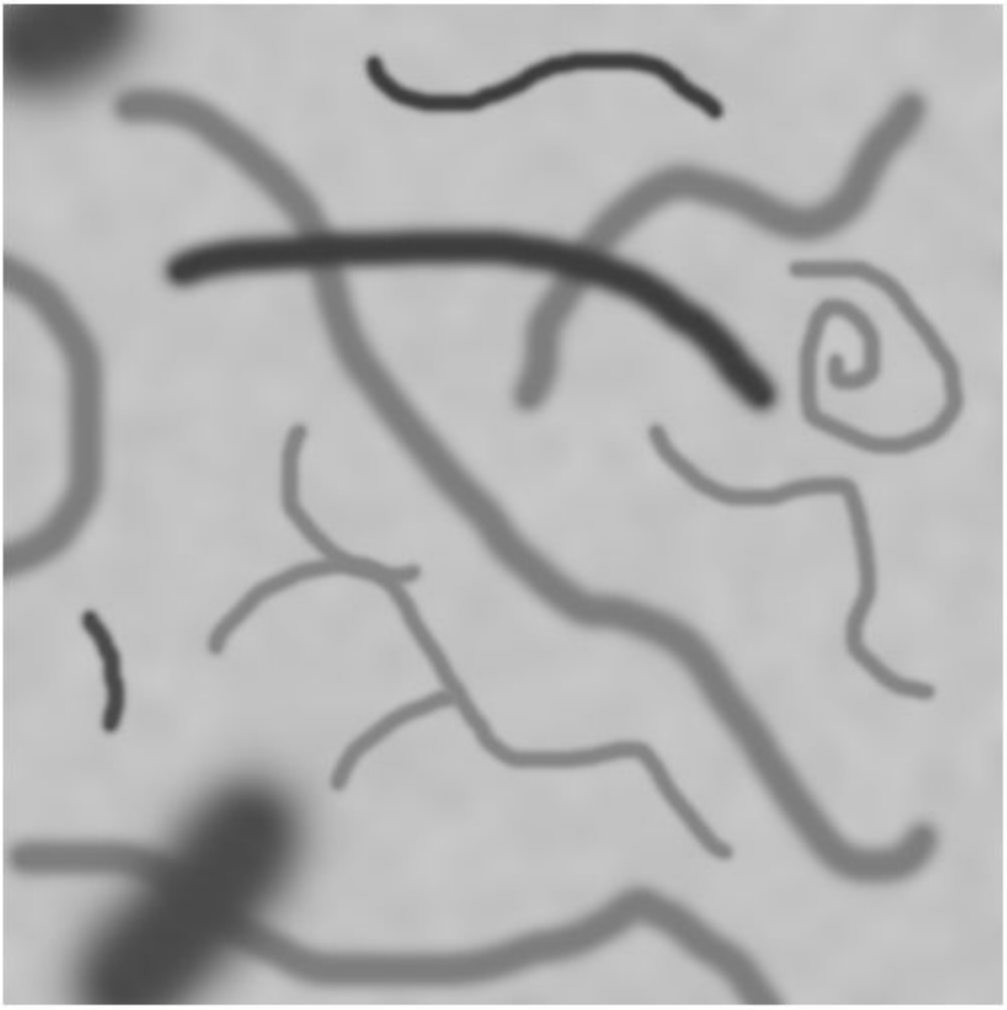
`C = maxhessiannorm(I,thickness)` also specifies the thickness of tubular structures.

Examples

Find Threads Using Maximum of Frobenius Norm of Image Hessian

Read and display an image that contains tubular threads of varying thicknesses.

```
I = imread('threads.png');
imshow(I)
```

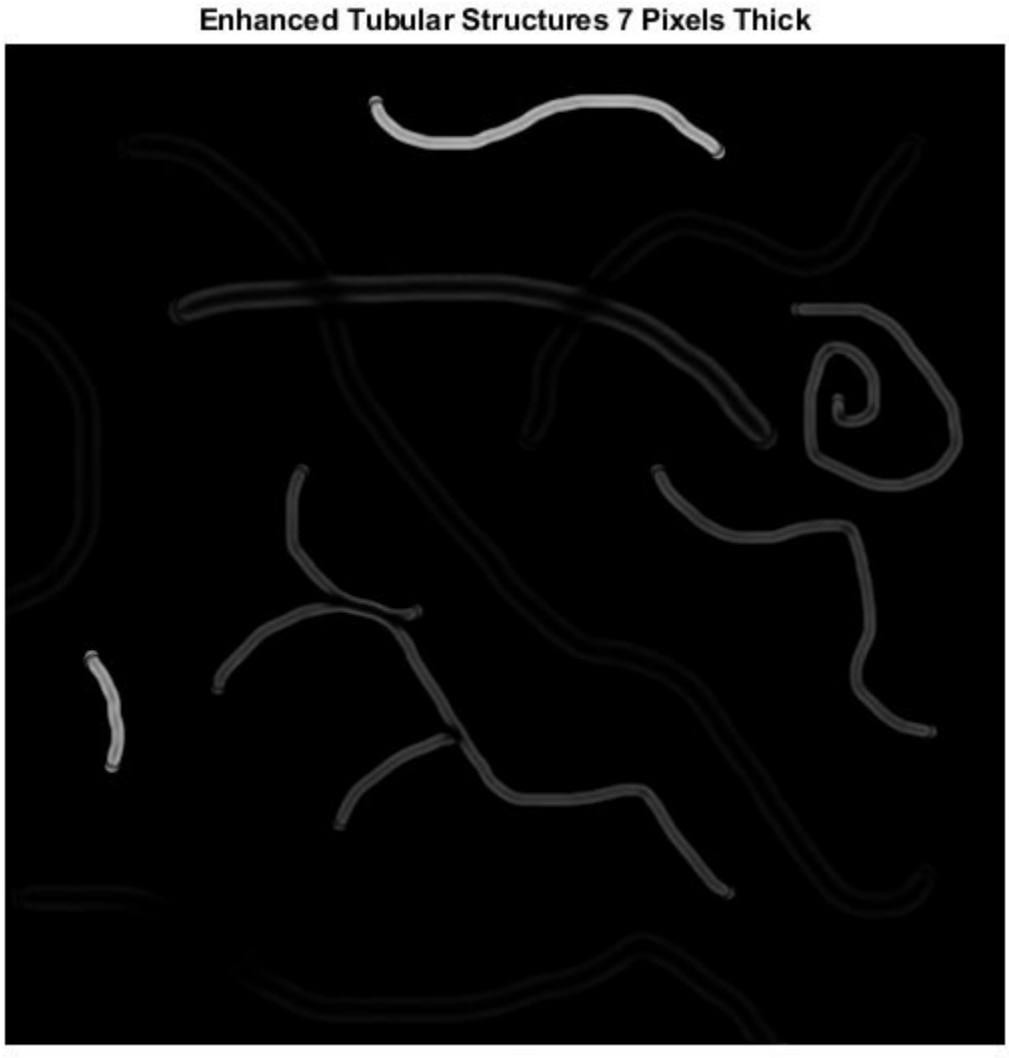


Calculate the maximum of the Frobenius norm of the Hessian of the image, with tubular thickness set to seven pixels.

```
C = maxhessiannorm(I,7);
```

Create an enhanced version of the image highlighting threads seven pixels thick. Use a structure sensitivity threshold equal to half of the maximum of the Frobenius norm of the Hessian. In the image, threads show up dark against a light background, so specify the object polarity as `'dark'`. Display the enhanced image.

```
J = fibermetric(I,7,'ObjectPolarity','dark','StructureSensitivity',0.5*C);
imshow(J)
title('Enhanced Tubular Structures 7 Pixels Thick')
```

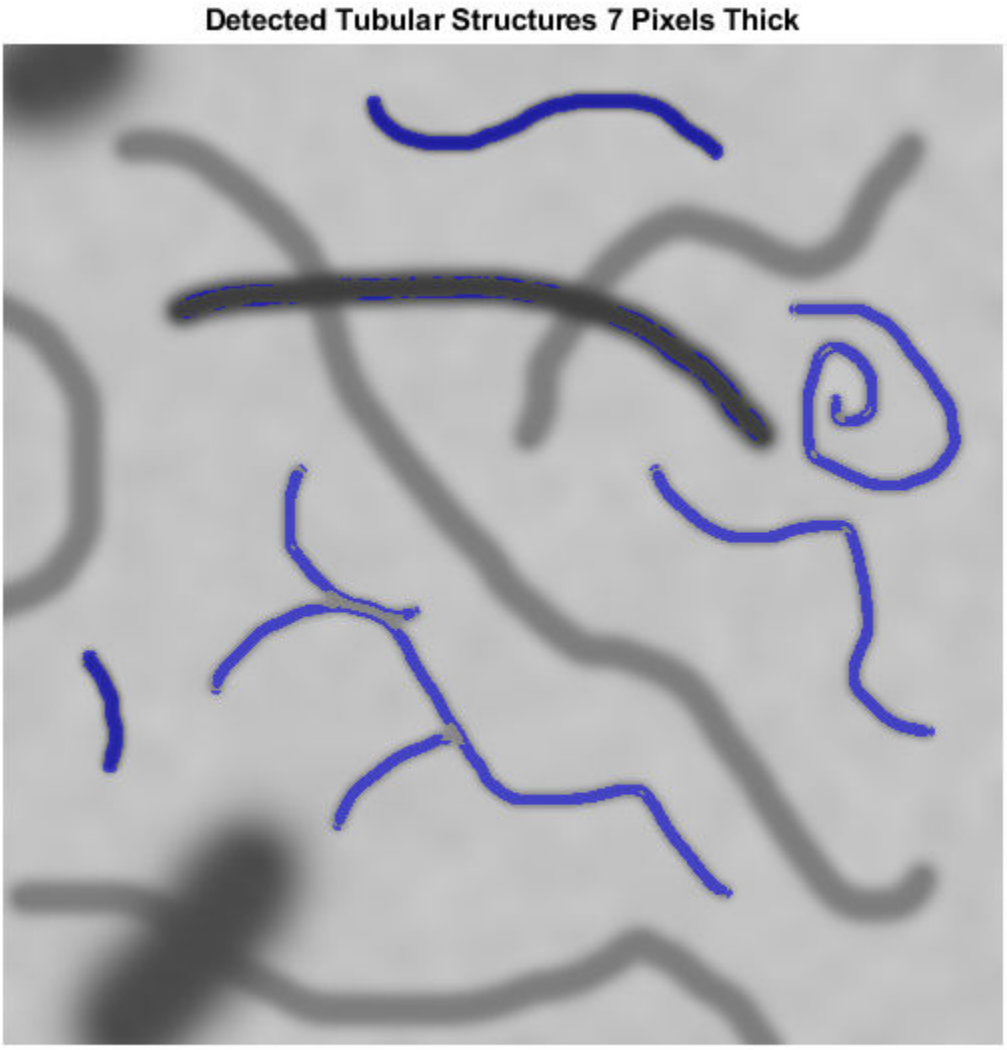


Threshold the enhanced image to create a binary mask containing only the threads with the specified thickness.

```
BW = imbinarize(J);
```

Display the mask over the original image using the `labeloverlay` function. The overlay has a blue tint where the mask is `true`, meaning those threads have the specified thickness.

```
maskl = labeloverlay(I,BW);
imshow(maskl)
title('Detected Tubular Structures 7 Pixels Thick')
```



Input Arguments

`I` — Image with elongated or tubular structures

2-D grayscale image

Image with elongated or tubular structures, specified as 2-D grayscale image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`thickness` — Thickness of tubular structures

4 (default) | positive integer

Thickness of tubular structures in pixels, specified as a positive integer.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`C` — Maximum Hessian norm

numeric scalar

Maximum of the Frobenius norm of the Hessian of grayscale image `I`, returned as a numeric scalar.

Data Types: `double`

Tips

- `maxhessiannorm` is a helper function to `fibermetric`, which changed default behavior in R2018b. If you want to reproduce the prior default behavior, then specify `StructureSensitivity` as `0.5*maxhessiannorm(I)`.

References

[1] Frangi, Alejandro F., et al. *Multiscale vessel enhancement filtering*. Medical Image Computing and Computer-Assisted Intervention — MICCAI'98. Springer Berlin Heidelberg, 1998. pp. 130–137.

See Also

`edge` | `imgradient` | `fibermetric`

Introduced in R2018b

mean2

Average or mean of matrix elements

Syntax

```
B = mean2(A)
```

Description

`B = mean2(A)` computes the mean of all values in array `A`.

Examples

Compute Mean of an Image

Read an image into the workspace.

```
I = imread('liftingbody.png');
```

Compute the mean.

```
meanval = mean2(I)
```

```
meanval = 140.2991
```

Input Arguments

A — Input data

numeric array | logical array

Input data, specified as a numerical or logical array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

B — Mean

numeric scalar

Mean of input data, returned as a numeric scalar. If the data type of `A` is `single`, then the data type of `B` is also single. Otherwise, the data type of `B` is `double`.

Data Types: `single` | `double`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`mean2` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`std2` | `corr2` | `mean` | `std`

Introduced before R2006a

measureChromaticAberration

Measure chromatic aberration at slanted edges using Imatest eSFR chart

Syntax

```
aberrationTable = measureChromaticAberration(chart)
aberrationTable = measureChromaticAberration(chart,Name,Value)
```

Description

`aberrationTable = measureChromaticAberration(chart)` measures the chromatic aberration at all slanted edge regions of interest (ROIs) of an Imatest eSFR chart [1].

`aberrationTable = measureChromaticAberration(chart,Name,Value)` measures the chromatic aberration with additional parameters to specify a subset of ROIs to measure.

Examples

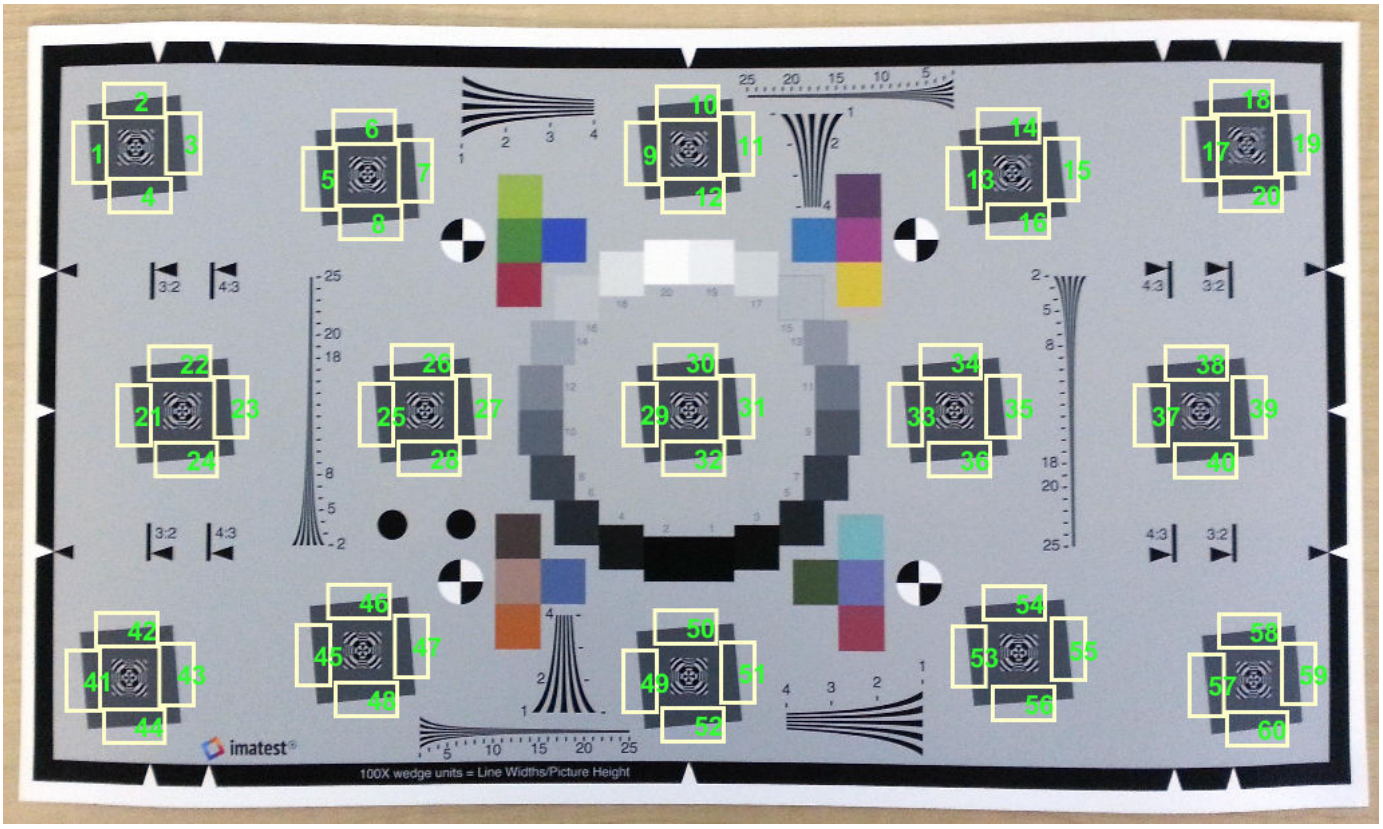
Measure Chromatic Aberration of Slanted Edges on eSFR Chart

Read an image of an eSFR chart into the workspace.

```
I = imread('eSFRTestImage.jpg');
```

Create an `esfrChart` object, then display the chart with ROI annotations. The 60 slanted edge ROIs are labeled with green numbers.

```
chart = esfrChart(I);
displayChart(chart,'displayColorROIs',false,...
    'displayGrayROIs',false,'displayRegistrationPoints',false)
```



Measure the chromatic aberration in all slanted edge ROIs. Examine the contents of the returned table, `chTable`, for a single ROI.

```
chTable = measureChromaticAberration(chart);
ROIIndex = 3;
chTable(3,:)
```

```
ans=1×5 table
    ROI    aberration    percentAberration     edgeProfile      normalizedEdgeProfile
    ___    __________    _________________    _____________    _____________________

     3       1.5261          0.11843         {336x4 table}        {336x4 table}
```

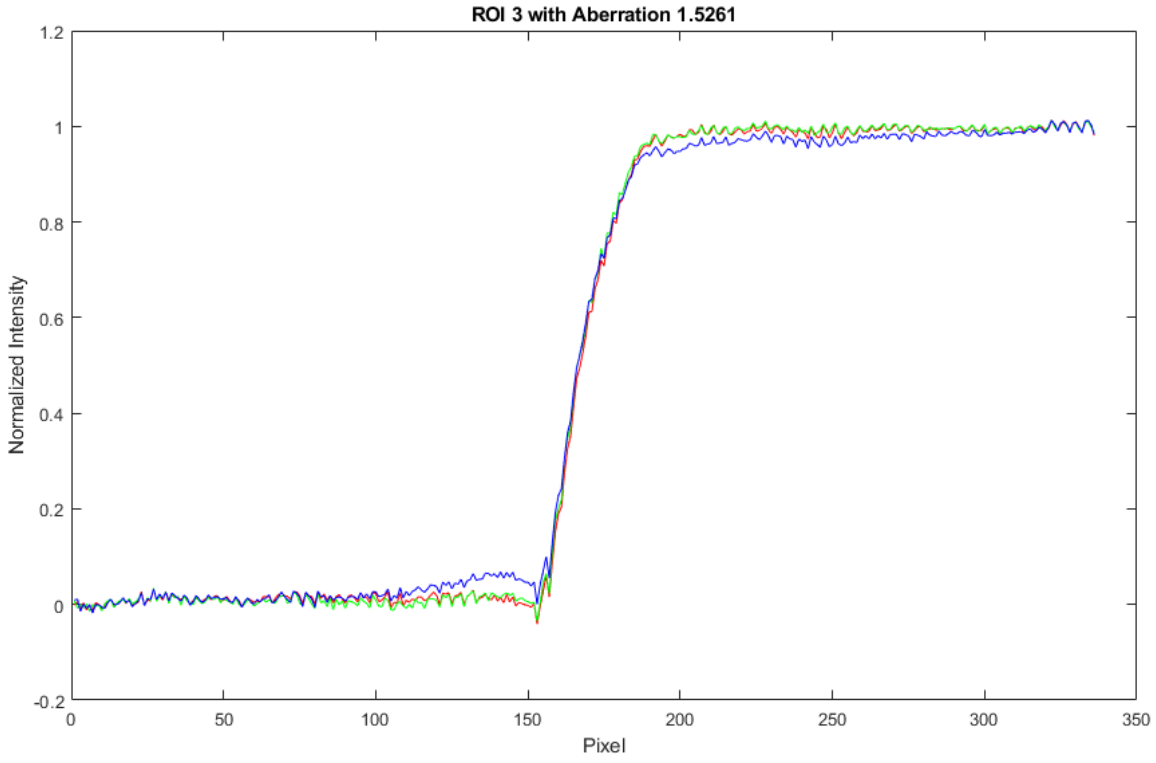
Store the normalized edge profile in a separate variable, `edgeProfile`, for clarity. Examine the normalized color intensity of the first and last pixel of `edgeProfile`.

```
edgeProfile = chTable.normalizedEdgeProfile{ROIIndex};
edgeProfile([1 end],:)
```

```
ans=2×4 table
    normalizedEdgeProfile_R    normalizedEdgeProfile_G    normalizedEdgeProfile_B    normalizedEd
    _______________________    _______________________    _______________________    ____________

          -0.0014365                  0.0072757                  0.0089823                 0.005
            0.98138                     0.9884                     0.98325                   0.9
```

Plot the normalized intensity for the ROI.

```
npix = length(edgeProfile.normalizedEdgeProfile_R);
plot(1:npix,edgeProfile.normalizedEdgeProfile_R,'r', ...
    1:npix,edgeProfile.normalizedEdgeProfile_G,'g', ...
    1:npix,edgeProfile.normalizedEdgeProfile_B,'b')
xlabel('Pixel')
ylabel('Normalized Intensity')
title(['ROI ' num2str(ROIIndex) ' with Aberration ' num2str(chTable.aberration(ROIIndex))])
```



The blue channel has a higher intensity than the red and green channels immediately before the edge, and a lower intensity than the red and green channels immediately after the edge. This difference in intensity contributes to the measured value of chromatic aberration.

The measured values of `aberration` and `percentAberration` for this edge are relatively small. Visual inspection of the image confirms that the sides of the edge do not have a strong color tint.

Input Arguments

`chart` — eSFR chart

`esfrChart` object

eSFR chart, specified as an `esfrChart` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'ROIIndex',2` measures the chromatic aberration only of ROI 2.

`ROIIndex` — ROI indices

`1:60` (default) | scalar | vector

ROI indices to include in measurements, specified as the comma-separated pair consisting of `'ROIIndex'` and a scalar or vector of integers in the range [1, 60]. The indices match the ROI numbers displayed by `displayChart`.

Note `measureChromaticAberration` uses the intersection of ROIs specified by `'ROIIndex'` and `'ROIOrientation'`.

Example: `29:32`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`ROIOrientation` — ROI orientation

`'both'` (default) | `'vertical'` | `'horizontal'`

ROI orientation, specified as the comma-separated pair consisting of `'ROIOrientation'` and `'both'`, `'vertical'`, or `'horizontal'`. The `measureChromaticAberration` function performs measurements only on ROIs with the specified orientation.

Note `measureChromaticAberration` uses the intersection of ROIs specified by `'ROIIndex'` and `'ROIOrientation'`.

Example: `'vertical'`

Data Types: `char` | `string`

Output Arguments

`aberrationTable` — Chromatic aberration measurements

m-by-5 table

Chromatic aberration measurements, returned as an *m*-by-5 table. *m* is the number of sampled ROIs.

The five columns represent these variables:

Variable	Description
`ROI`	Index of the sampled ROI. The value of `ROI` is an integer in the range [1, 60].

Variable	Description
`aberration`	Chromatic aberration, measured as the area between the maximum and the minimum red, green, and blue edge intensity profiles. The measured chromatic aberration indicates perceptual chromatic aberration. `aberration` is a scalar of type `double`.
`percentAberration`	Aberration, expressed as a percentage of the distance in pixels between the center of the image and the center of the ROI.
`edgeProfile`	Intensity profile of each color channel across the edge in the ROI. `edgeProfile` is an *s*-by-4 table, where *s* is the number of samples across the edge. The four columns represent the red, green, blue, and luminance values, averaged along the edge. Luminance (*Y*) is a linear combination of the red (*R*), green (*G*), and blue (*B*) channels according to: $Y = 0.213R + 0.715G + 0.072B$ **Note** The sampling rate for the chromatic aberration measurement is about four times the sampling rate of the image.
`normalizedEdgeProfile`	Intensity profile, normalized between [0, 1] using 5% of the front end and tail end of data. `normalizedEdgeProfile` is an *s*-by-4 table with a similar structure to `edgeProfile`.

Tips

- Chromatic aberration is best measured at slanted edges that are:
  - Roughly orthogonal to the line connecting the center of the image and the center of the ROI
  - Farthest from the center of the image

  Because chromatic aberration increases radially from the center of the image, measurements at slanted edges near the center of the image can be ignored.
- The absolute chromatic aberration reported in the `aberration` field is measured in the horizontal or vertical direction. However, chromatic aberration is a radial phenomenon, and radial measurements are more accurate.

References

[1] Imatest. "Esfr". https://www.imatest.com/mathworks/esfr/.

See Also

`measureSharpness` | `displayChart`

Topics

“Anatomy of Imatest Extended eSFR Chart”
“Evaluate Quality Metrics on eSFR Test Chart”

Introduced in R2017b

measureColor

Measure color reproduction using test chart

Syntax

```
colorTable = measureColor(chart)
[colorTable,colorCorrectionMatrix] = measureColor(chart)
```

Description

`colorTable = measureColor(chart)` measures the color values at all color regions of interest (ROIs) of an Imatest eSFR chart [1] or a Calibrite ColorChecker Classic chart [2].

`[colorTable,colorCorrectionMatrix] = measureColor(chart)` also returns a color correction matrix computed using a linear least squares fit.

Examples

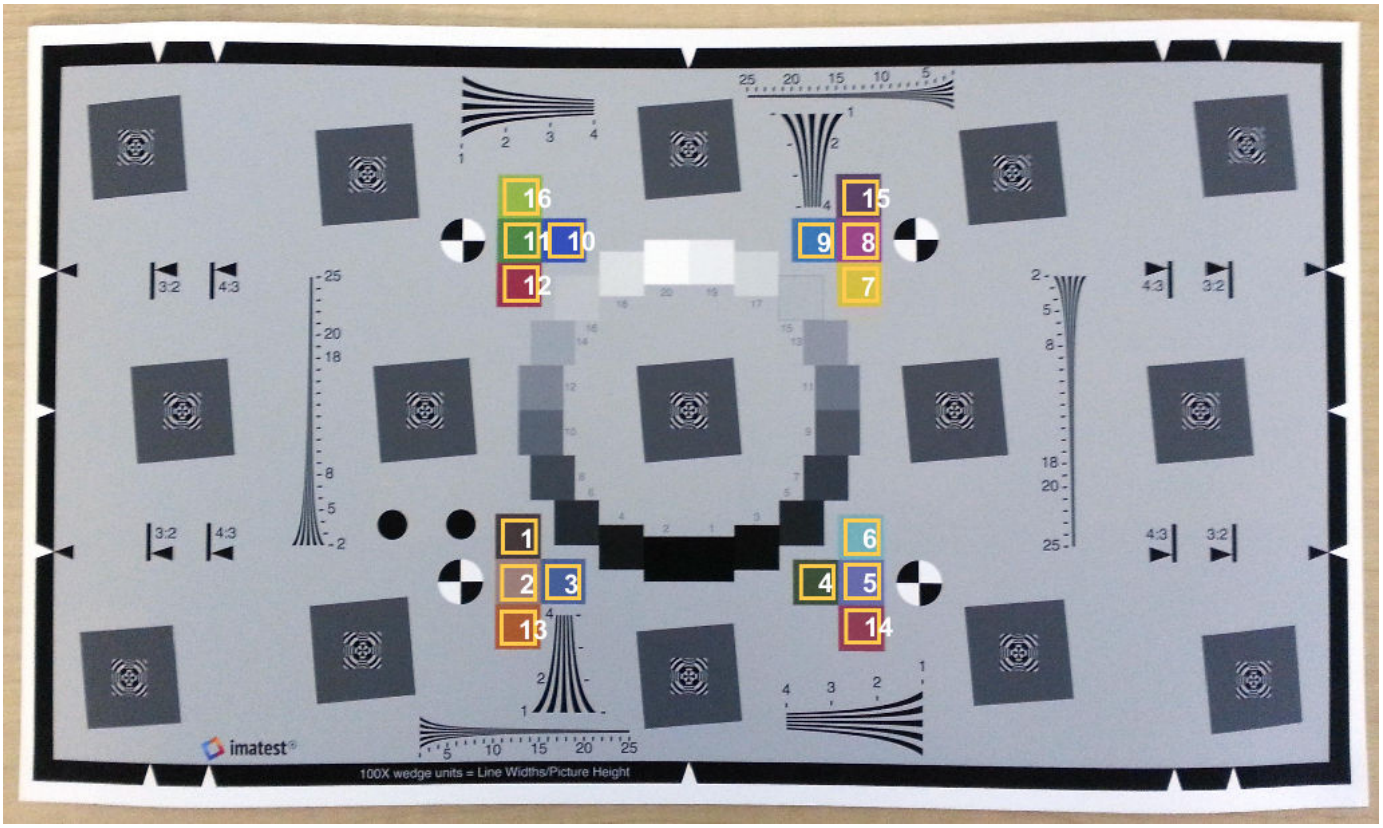
Measure Color Accuracy of eSFR Chart

Read an image of an eSFR chart into the workspace.

```
I = imread('eSFRTestImage.jpg');
```

Create an `esfrChart` object, then display the chart with ROI annotations. The 16 color patch ROIs are labeled with white numbers.

```
chart = esfrChart(I);
displayChart(chart,'displayEdgeROIs',false, ...
    'displayGrayROIs',false,'displayRegistrationPoints',false)
```



Measure the color in all color patch ROIs.

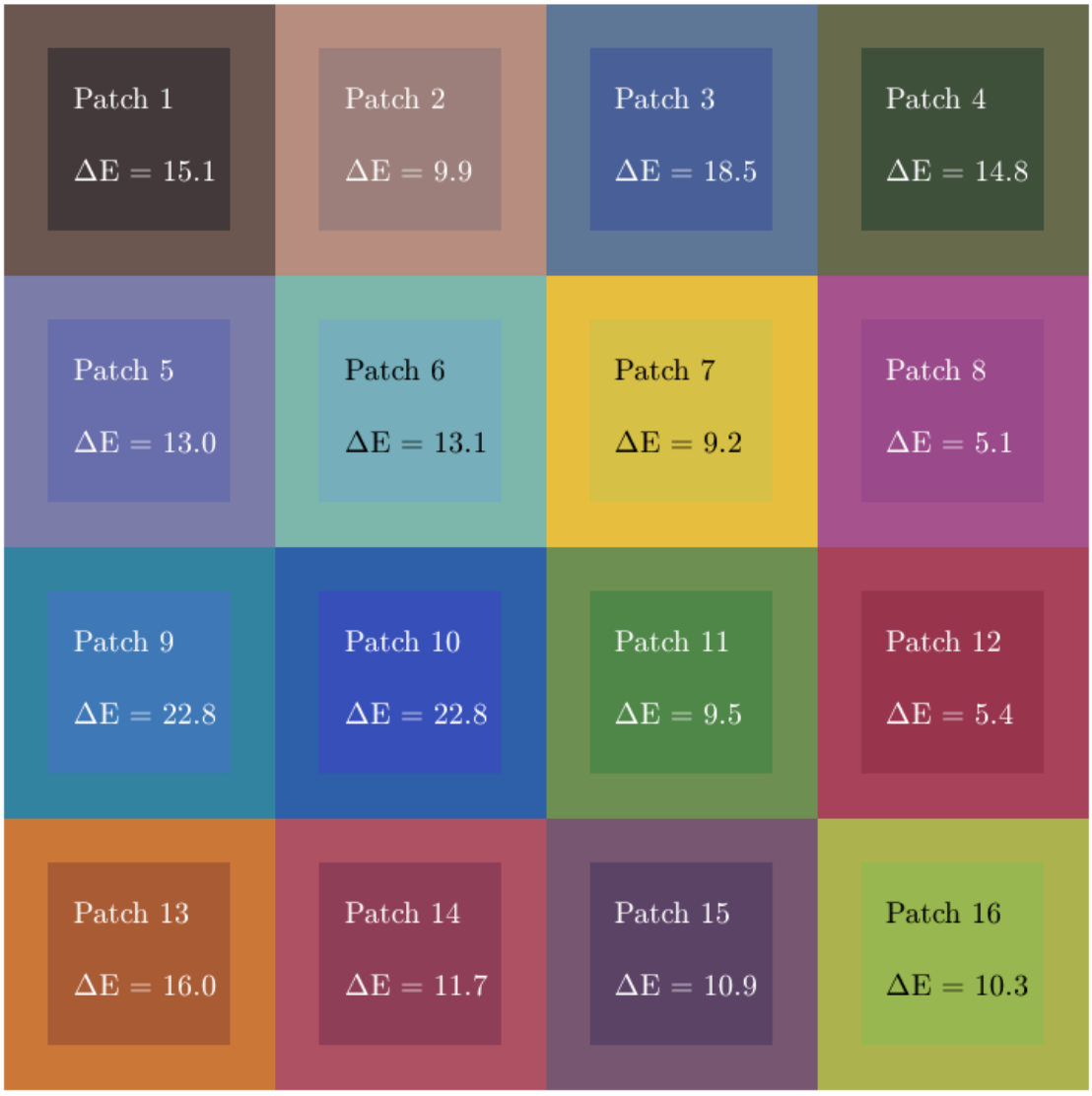
```
colorTable = measureColor(chart)
```

```
colorTable=16×8 table
    ROI    Measured_R    Measured_G    Measured_B    Reference_L    Reference_a    Reference_b
    ___    __________    __________    __________    ___________    ___________    ___________

     1         67            57            58          38.586          7.541          7.0812
     2        156           127           122          62.182         13.225          13.826
     3         73            95           152          49.369        -0.51463        -20.062
     4         62            79            58          43.926         -6.8587         17.278
     5        104           109           171          53.415          9.457         -22.822
     6        118           175           187           69.95         -20.889        -0.21752
     7        214           192            69          78.643          1.8052         67.091
     8        154            73           138          46.853         41.998         -17.056
     9         62           120           182           51.05         -15.166        -22.416
    10         55            80           185          40.811          8.7346        -44.265
    11         79           135            72          55.716         -23.419         28.839
    12        152            53            77          42.759         44.167          7.9536
    13        169            91            52          58.211          27.58          47.578
    14        142            63            87          47.012          39.15          8.5453
    15         91            67           102          40.591         17.951          -9.525
    16        152           183            80          70.505        -16.318          49.811
```

Display the color accuracy measurements. Each square color patch is the measured color, and the thick surrounding border is the reference color for that ROI. Each color accuracy measurement is

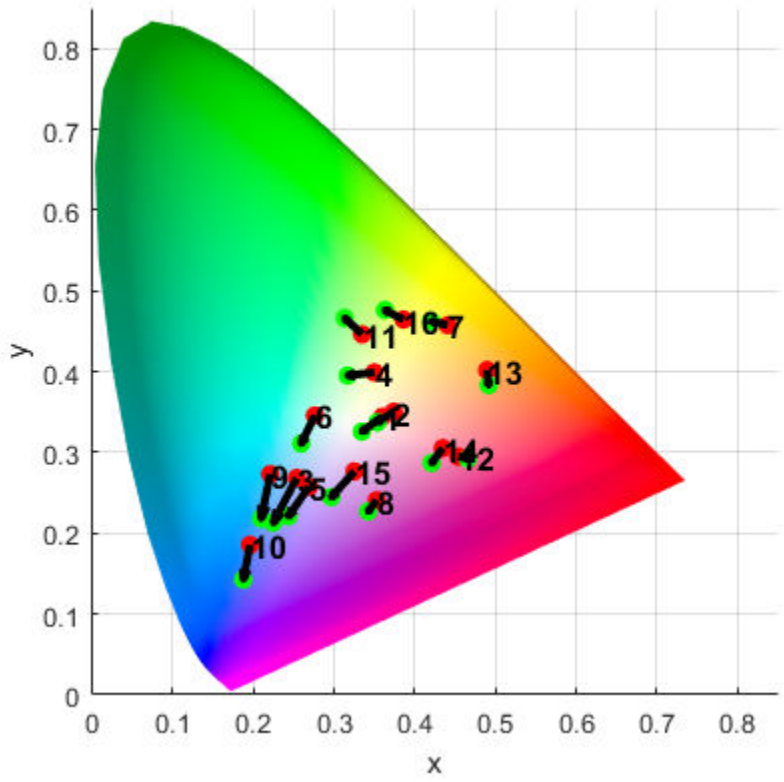
displayed as `Delta_E`, the Euclidean distance between measured and reference colors in the CIE 1976 L*a*b* color space. More accurate colors have a smaller `Delta_E`.

```
figure
displayColorPatch(colorTable)
```



For an alternative representation of the color accuracy measurements, plot the measured and reference colors in the CIE 1976 L*a*b* color space on a chromaticity diagram. Red circles indicate the reference color. Green circles indicate the measured color of each color patch. The chromaticity diagram does not portray the brightness of color.

```
figure
plotChromaticity(colorTable)
```



ROIs with a shorter distance between the reference and measurement points have smaller differences in chromaticity, which can contribute to a smaller value of `Delta_E`. However, brightness also contributes to the value of `Delta_E`. For example, even though the reference and measurement points for ROI 13 are near each other on the chromaticity diagram, they have a large `Delta_E` because of their large difference in brightness.

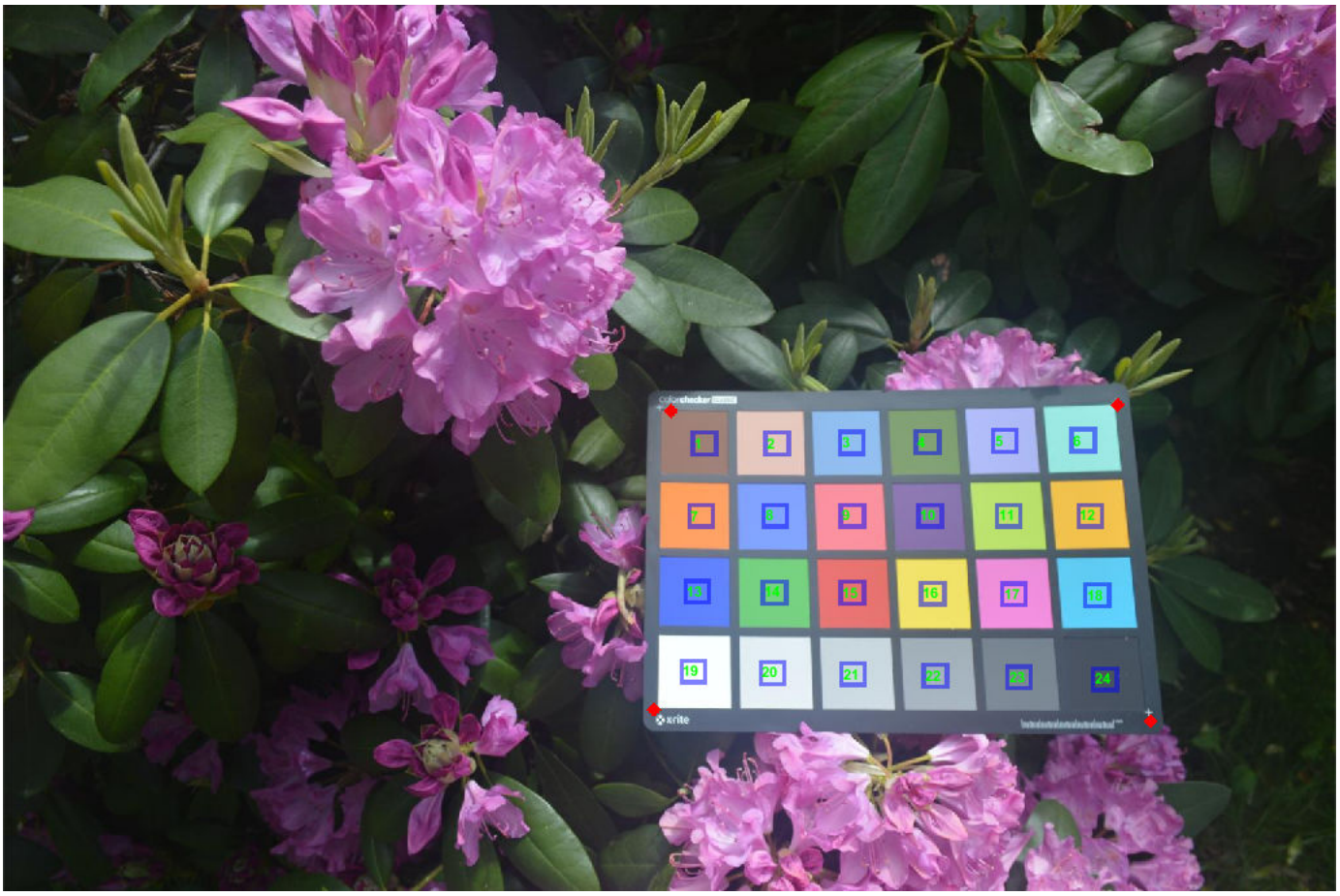
Measure Color of ColorChecker Chart

Read an image of a ColorChecker® chart into the workspace.

```
I = imread("colorCheckerTestImage.jpg");
```

Create a `colorChecker` object, then display the chart with ROI annotations.

```
chart = colorChecker(I);
displayChart(chart)
```



Measure the color in each color patch ROI.

```
colorTable = measureColor(chart)
```

```
colorTable=24×9 table
    ROI          Color          Measured_R    Measured_G    Measured_B    Reference_L    Reference
    ___    ________________     __________    __________    __________    ___________    _________

     1     {'DarkSkin'    }        160           129           120          37.54          14.37
     2     {'LightSkin'   }        229           200           191          64.66          19.27
     3     {'BlueSky'     }        146           191           241          49.32          -3.82
     4     {'Foliage'     }        130           161           117          43.46         -12.74
     5     {'BlueFlower'  }        175           187           248          54.94           9.61
     6     {'BluishGreen' }        155           232           226          70.48         -32.26
     7     {'Orange'      }        255           161            99          62.73          35.83
     8     {'PurplishBlue'}        130           164           254          39.43          10.75
     9     {'ModerateRed' }        252           146           160          50.57          48.64
    10     {'Purple'      }        139           118           175           30.1          22.54
    11     {'YellowGreen' }        187           226           110          71.77         -24.13
    12     {'OrangeYellow'}        241           194            76          71.51          18.24
    13     {'Blue'        }         96           131           255          28.37          15.42
    14     {'Green'       }        118           209           130          54.38         -39.72
    15     {'Red'         }        234           116           114          42.43          51.05
    16     {'Yellow'      }        241           227           105           81.8           2.67
```

Input Arguments

`chart` — Test chart

`esfrChart` object | `colorChecker` object

Test chart, specified as an `esfrChart` object or a `colorChecker` object.

Output Arguments

`colorTable` — Color values

p-by-8 table

Color values in each color patch, returned as a *p*-by-8 table, where *p* is the number of color patches on the test chart, `chart`.

The eight columns represent these variables:

Variable	Description
`ROI`	Index of the sampled ROI. The value of `ROI` is an integer in the range [1, 16]. The indices match the ROI numbers displayed by `displayChart`.
`Measured_R`	Mean value of red channel pixels in the ROI. `Measured_R` is a scalar of the same data type as `chart.Image`, which can be of type `single`, `double`, `uint8`, or `uint16`.
`Measured_G`	Mean value of green channel pixels in the ROI. `Measured_G` is a scalar of the same data type as `chart.Image`.
`Measured_B`	Mean value of blue channel pixels in the ROI. `Measured_B` is a scalar of the same data type as `chart.Image`.
`Reference_L`	Reference L* value of the ROI. `Reference_L` is a scalar of type `double`.
`Reference_a`	Reference a* value of the ROI. `Reference_a` is a scalar of type `double`.
`Reference_b`	Reference b* value of the ROI. `Reference_b` is a scalar of type `double`.
`Delta_E`	Euclidean color distance between the measured and reference color values in the L*a*b* color space, as outlined in CIE 1976. `Delta_E` is a scalar of type `double`.

`colorCorrectionMatrix` — Color correction coefficients

4-by-3 matrix

Color correction coefficients, returned as a 4-by-3 matrix. `colorCorrectionMatrix` represents an affine transformation that you can use to color-correct images that are captured under similar lighting conditions as the test chart image.

Data Types: `double`

Tips

- The `measureColor` function assumes that measured RGB values are in the sRGB color space. The `measureColor` converts the values to the L*a*b* color space to calculate the color error, `Delta_E`.
- When the `chart` is an `esfrChart` object, the white point of the reference L*a*b* values is the CIE standard illuminant D65.
- When the `chart` is a `colorChecker` object, the white point of the reference L*a*b* values is the CIE standard illuminant D50. The reference L*a*b* values are for the "After November 2014" version of the ColorChecker chart.

References

[1] Imatest. "Esfr". https://www.imatest.com/mathworks/esfr/.

[2] Calibrite. "ColorChecker Classic". https://calibrite.com/us/product/colorchecker-classic/.

See Also

`displayColorPatch` | `plotChromaticity` | `measureIlluminant` | `deltaE` | `imcolordiff`

Topics

“Calculate CIE94 Color Difference of Colors on Test Chart”
“Correct Colors Using Color Correction Matrix”
“Evaluate Quality Metrics on eSFR Test Chart”

Introduced in R2017b

measureIlluminant

Measure scene illuminant using test chart

Syntax

```
illuminant = measureIlluminant(chart)
```

Description

`illuminant = measureIlluminant(chart)` measures the scene illuminant using the gray regions of interest (ROIs) of an Imatest eSFR chart [1] or a Calibrite ColorChecker Classic chart [2].

Examples

Measure Illuminant of eSFR Chart

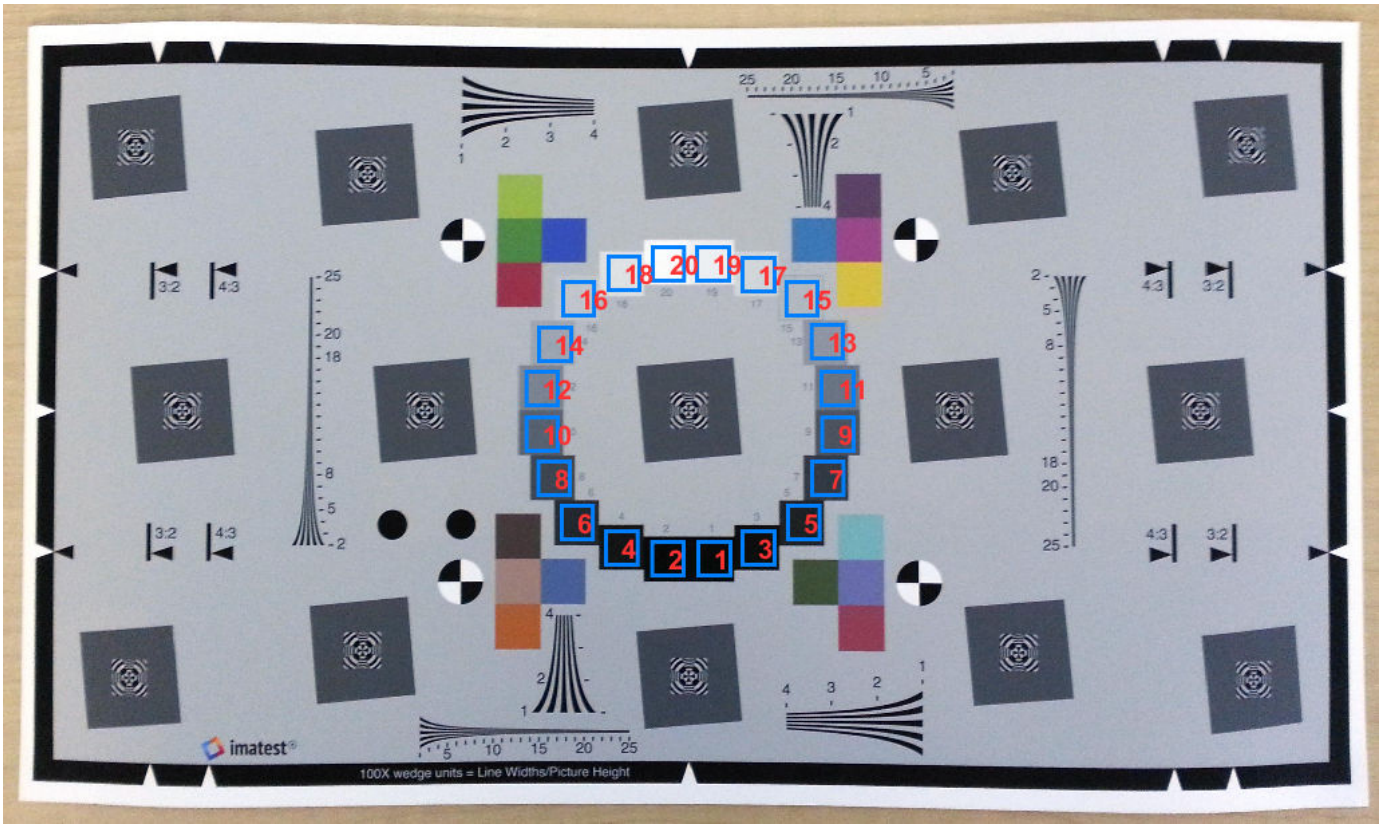
This example shows how to measure the illuminant of an eSFR chart using the gray patch ROIs. The example then white balances the image of the eSFR chart.

Read an image of an eSFR chart into the workspace.

```
I = imread('eSFRTestImage.jpg');
```

Create an `esfrChart` object. Display the chart, highlighting the 20 gray patches.

```
chart = esfrChart(I);
displayChart(chart,'displayEdgeROIs',false, ...
    'displayColorROIs',false,'displayRegistrationPoints',false)
```



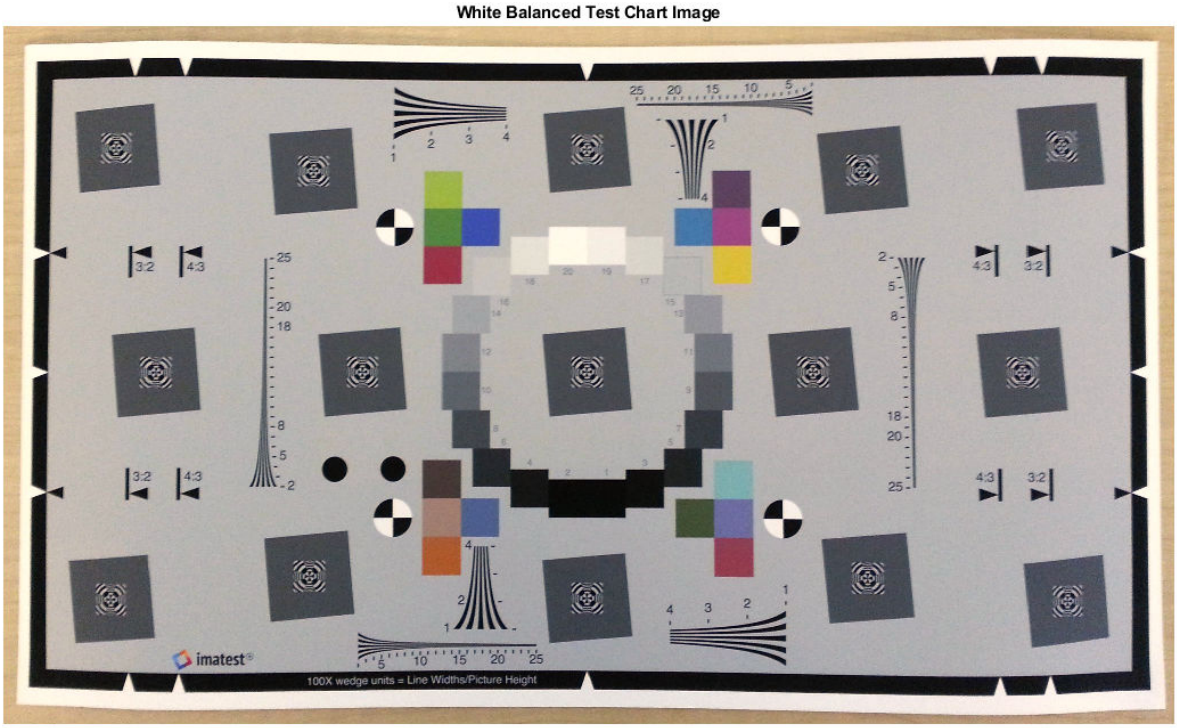
Estimate the illuminant using the gray patch ROIs. The illuminant has a stronger blue component than the red and green. This result is consistent with the image of the test chart, which has a blue tint.

```
illum = measureIlluminant(chart)
```

```
illum = 1×3

  110.9147  116.0008  123.2339
```

White balance the chart image and display the result. The white balanced image has less of a blue tint, especially in the middle gray patches and over the background of the image.

```
J = chromadapt(I,illum);
imshow(J)
title('White Balanced Test Chart Image')
```



You can use the estimated illuminant to white balance other images acquired under similar lighting conditions.

Input Arguments

`chart` — Test chart

`esfrChart` object | `colorChecker` object

Test chart, specified as an `esfrChart` object or a `colorChecker` object.

Output Arguments

`illuminant` — Scene illuminant

3-element row vector

Scene illuminant, returned as a 3-element row vector.

Data Types: `double`

Tips

- To white-balance an image, use the `chromadapt` function.
- It is recommended to measure the scene illuminant using linear image data. If you need to linearize your image data, then you can use the `rgb2lin` function.

References

[1] Imatest. "Esfr". https://www.imatest.com/mathworks/esfr/.

[2] Calibrite. "ColorChecker Classic". https://calibrite.com/us/product/colorchecker-classic/.

See Also

`measureColor` | `chromadapt`

Topics

“Comparison of Auto White Balance Algorithms”
“Anatomy of Imatest Extended eSFR Chart”
“Evaluate Quality Metrics on eSFR Test Chart”

Introduced in R2017b

measureNoise

Measure noise using Imatest eSFR chart

Syntax

```
noiseTable = measureNoise(chart)
```

Description

`noiseTable = measureNoise(chart)` measures the noise levels using the gray regions of interest (ROIs) of an Imatest eSFR chart [1].

Examples

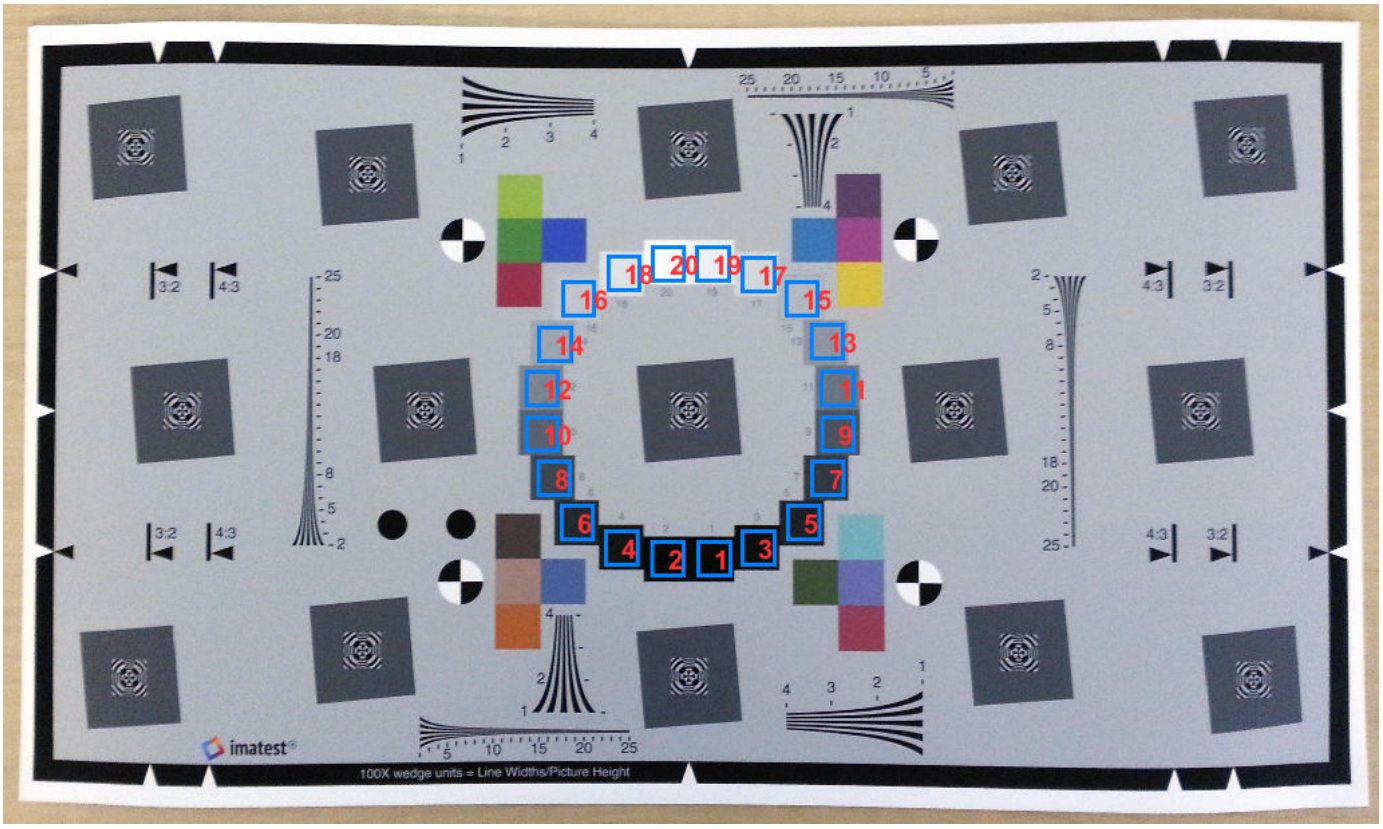
Measure Noise of eSFR Chart

Read an image of an eSFR chart into the workspace.

```
I = imread('eSFRTestImage.jpg');
```

Create an `esfrChart` object, then display the chart with ROI annotations. The 20 gray patch ROIs are labeled with red numbers.

```
chart = esfrChart(I);
displayChart(chart,'displayColorROIs',false, ...
    'displayEdgeROIs',false,'displayRegistrationPoints',false)
```



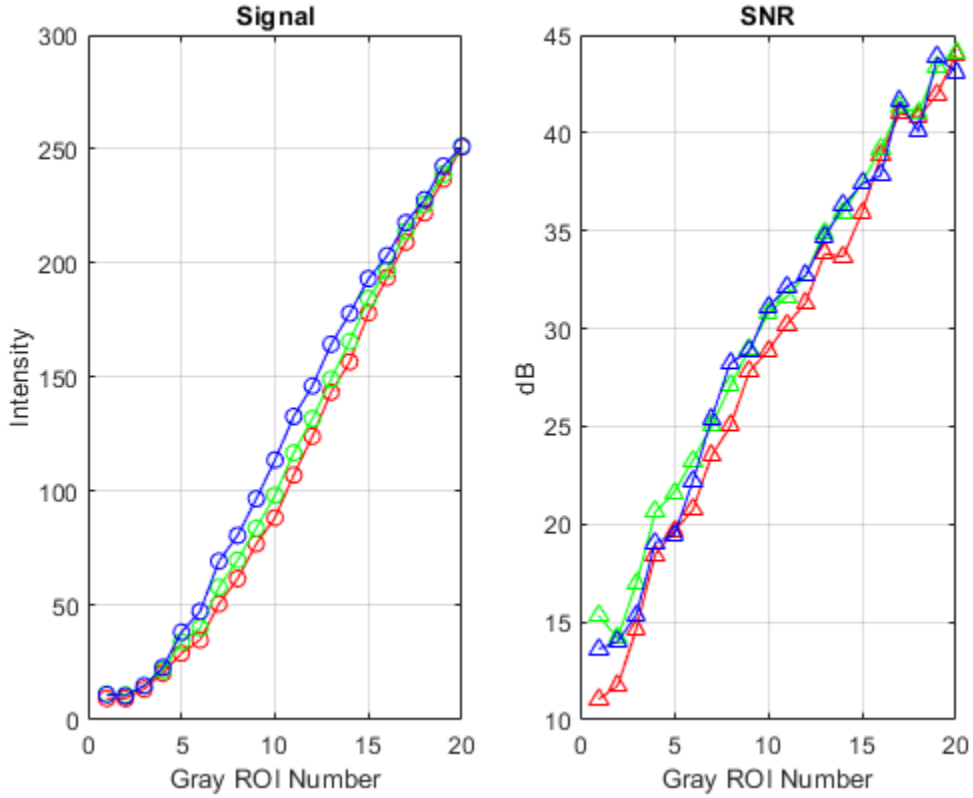
Measure the noise in all gray patch ROIs.

```
noiseTable = measureNoise(chart)
```

```
noiseTable=20×22 table
    ROI    MeanIntensity_R    MeanIntensity_G    MeanIntensity_B    RMSNoise_R    RMSNoise_G    R
    ___    _______________    _______________    _______________    __________    __________    _

     1         9.4147             11.349             11.099           2.6335        1.9417
     2         9.2873             10.896             10.503            2.405        2.1309
     3         13.488              14.95             15.017           2.4966        2.1156
     4         20.411             21.689             22.946           2.4395        2.0206
     5         29.189             34.144             38.442           3.0436        2.8317
     6         35.009             40.337             47.544           3.2201        2.7705
     7         50.768             58.206             69.539           3.3931        3.2661
     8         61.871              69.98             80.779           3.4734        3.0966
     9         77.115             83.999             96.869           3.1467        2.9973
    10         88.552             98.426             113.87           3.1846        2.8538
    11         107.25             116.97             132.94           3.3128        3.0561
    12         124.23             131.96             146.27           3.3817        3.0611
    13         143.52              149.3             164.52            2.922        2.6763
    14         156.87             165.76             178.05           3.2507        2.6489
    15         178.25             184.59              193.3           2.8498         2.474
    16         193.81             196.97             203.42           2.2181        2.1638
     ⋮
```

Display a graph of the mean signal and the signal to noise ratio (SNR) of the three color channels over the 20 gray patch ROIs.

```
figure
subplot(1,2,1)
plot(noiseTable.ROI,noiseTable.MeanIntensity_R,'r-o', ...
     noiseTable.ROI,noiseTable.MeanIntensity_G,'g-o', ...
     noiseTable.ROI,noiseTable.MeanIntensity_B,'b-o')
title('Signal')
ylabel('Intensity')
xlabel('Gray ROI Number')
grid on
subplot(1,2,2)
plot(noiseTable.ROI,noiseTable.SNR_R,'r-^', ...
     noiseTable.ROI,noiseTable.SNR_G,'g-^', ...
     noiseTable.ROI,noiseTable.SNR_B,'b-^')
title('SNR')
ylabel('dB')
xlabel('Gray ROI Number')
grid on
```



Input Arguments

chart — eSFR chart

`esfrChart` object

eSFR chart, specified as an `esfrChart` object.

Output Arguments

`noiseTable` — Noise values

20-by-22 table

Noise values of each gray patch, returned as a 20-by-22 table. The 20 rows correspond to the 20 gray patches on the eSFR chart. The 22 columns represent the variables shown in the table. Each variable is a scalar of type `double`.

Variable	Description
`ROI`	Index of the sampled ROI. The value of `ROI` is an integer in the range [1, 20]. The indices match the ROI numbers displayed by `displayChart`.
`MeanIntensity_R`	Mean value of red channel pixels in the ROI.
`MeanIntensity_G`	Mean value of green channel pixels in the ROI.
`MeanIntensity_B`	Mean value of blue channel pixels in the ROI.
`RMSNoise_R`	Root mean square (RMS) noise of red channel pixels in the ROI.
`RMSNoise_G`	RMS noise of green channel pixels in the ROI.
`RMSNoise_B`	RMS noise of blue channel pixels in the ROI.
`PercentNoise_R`	RMS noise of red pixels, expressed as a percentage of the maximum of the original chart image data type.
`PercentNoise_G`	RMS noise of green pixels, expressed as a percentage of the maximum of the original chart image data type.
`PercentNoise_B`	RMS noise of blue pixels, expressed as a percentage of the maximum of the original chart image data type.
`SignalToNoiseRatio_R`	Ratio of signal (`MeanIntensity_R`) to noise (`RMSNoise_R`) in the red channel.
`SignalToNoiseRatio_G`	Ratio of signal (`MeanIntensity_G`) to noise (`RMSNoise_G`) in the green channel.
`SignalToNoiseRatio_B`	Ratio of signal (`MeanIntensity_B`) to noise (`RMSNoise_B`) in the blue channel.
`SNR_R`	Signal-to-noise ratio (SNR) of the red channel, in dB. `SNR_R = 20*log(MeanIntensity_R/RMSNoise_R)`.
`SNR_G`	SNR of the green channel, in dB. `SNR_G = 20*log(MeanIntensity_G/RMSNoise_G)`.
`SNR_B`	SNR of the blue channel, in dB. `SNR_B = 20*log(MeanIntensity_B/RMSNoise_B)`.
`PSNR_R`	Peak SNR of the red channel, in dB.
`PSNR_G`	Peak SNR of the green channel, in dB.
`PSNR_B`	Peak SNR of the blue channel, in dB.
`RMSNoise_Y`	RMS noise of luminance (Y) channel pixels in the ROI.
`RMSNoise_Cb`	RMS noise of chrominance (Cb) channel pixels in the ROI.

Variable	Description
`RMSNoise_Cr`	RMS noise of chrominance (Cr) channel pixels in the ROI.

Tips

- To linearize data for noise measurements, first undo the gamma correction of an sRGB test chart image by using the `rgb2lin` function. Then, create an `esfrChart` object from the linear image, and input the `esfrChart` object to the `measureNoise` function.

References

[1] Imatest. "Esfr". https://www.imatest.com/mathworks/esfr/.

See Also

`measureIlluminant` | `displayChart`

Topics

“Anatomy of Imatest Extended eSFR Chart”
“Evaluate Quality Metrics on eSFR Test Chart”

Introduced in R2017b

measureSharpness

Measure spatial frequency response using Imatest eSFR chart

Syntax

```
sharpnessTable = measureSharpness(chart)
sharpnessTable = measureSharpness(chart,Name,Value)
[sharpnessTable,aggregateSharpnessTable] = measureSharpness( ___ )
```

Description

`sharpnessTable = measureSharpness(chart)` measures the spatial frequency response (SFR) at all slanted edge regions of interest (ROIs) of an Imatest eSFR chart [1]. The returned sharpness table includes the frequency for each ROI at which the response drops to 50% of the initial and peak values.

`sharpnessTable = measureSharpness(chart,Name,Value)` measures the SFR at all specified slanted edge ROIs, specifying additional parameters.

`[sharpnessTable,aggregateSharpnessTable] = measureSharpness( ___ )` also returns the average SFR of vertical and horizontal ROIs, using the input arguments of either of the previous syntaxes.

Examples

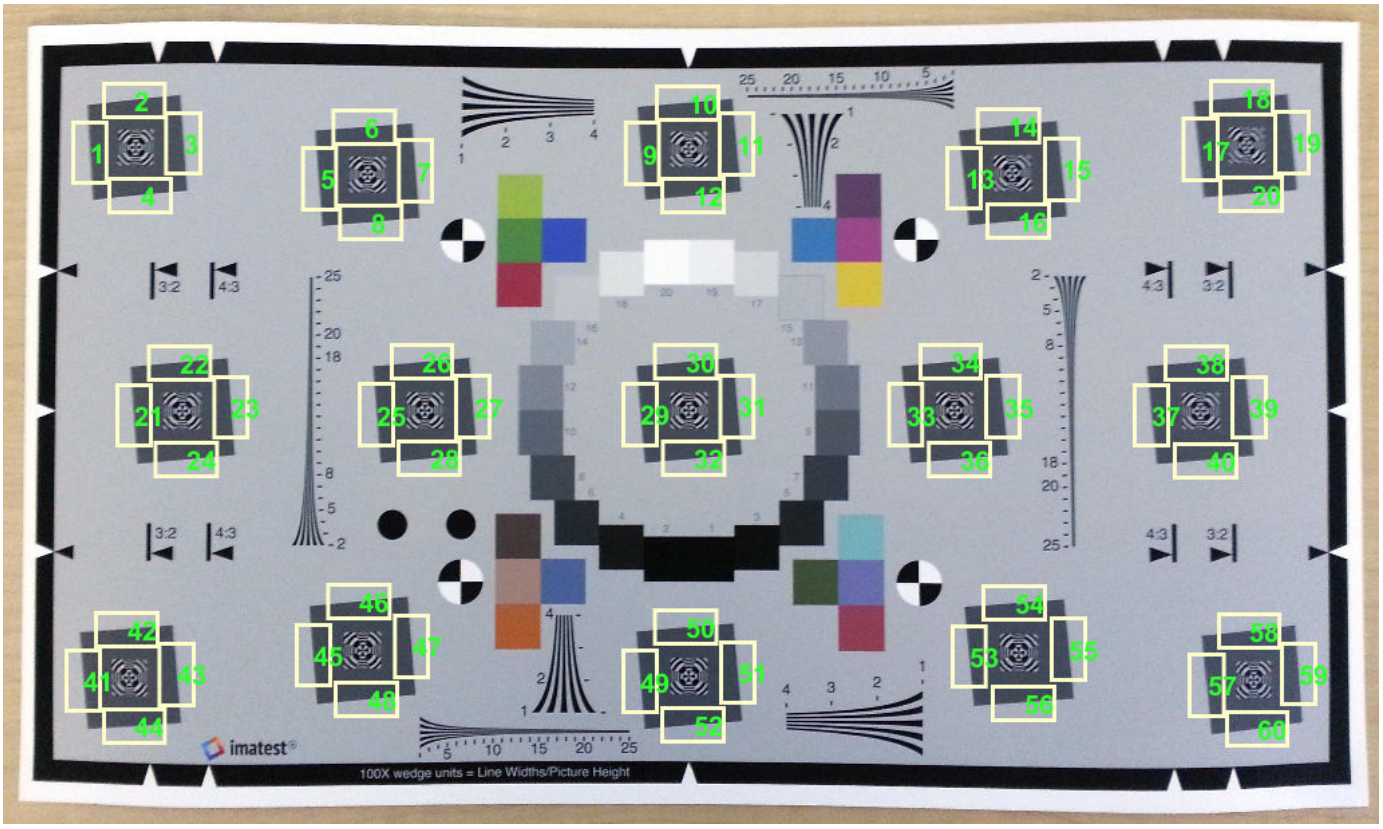
Measure Sharpness of Slanted Edges on eSFR Chart

Read an image of an eSFR chart into the workspace.

```
I = imread('eSFRTestImage.jpg');
```

Create an `esfrChart` object, then display the chart with ROI annotations. The 60 slanted edge ROIs are labeled with green numbers.

```
chart = esfrChart(I);
displayChart(chart,'displayColorROIs',false,...
    'displayGrayROIs',false,'displayRegistrationPoints',false)
```



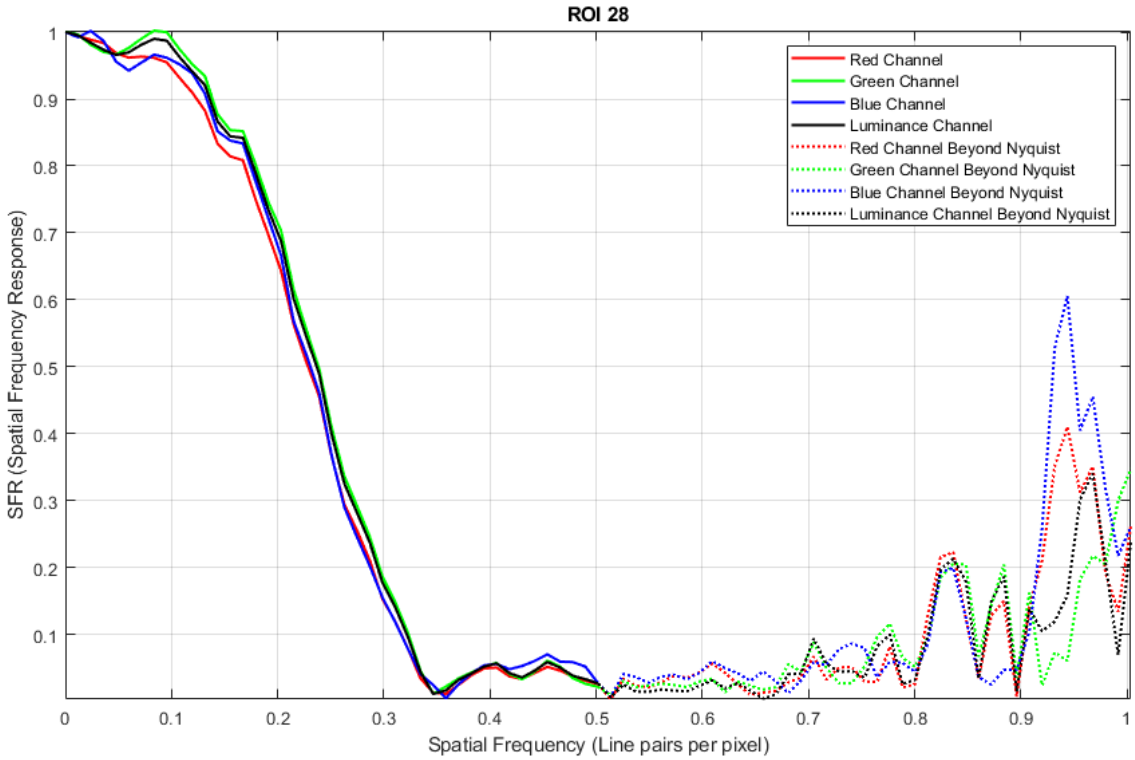
Measure the edge sharpness in ROIs 25-28, and return the measurements in `sharpnessTable`. Include measurements of the MTF70 and MTF30 by specifying the `'percentResponse'` name-value pair argument.

```
sharpnessTable = measureSharpness(chart,'ROIIndex',25:28,'PercentResponse',[70 30])
```

```
sharpnessTable=4×9 table
    ROI    slopeAngle    confidenceFlag        SFR          comment                          MTF
    ___    __________    ______________    ___________    ___________    ____________________

    25       4.2268          true          {85x5 table}    {0x0 double}    0.061637    0.059828
    26       5.0814          true          {85x5 table}    {0x0 double}     0.18553     0.18604
    27       4.7787          true          {85x5 table}    {0x0 double}    0.069499    0.06935
    28       4.7966          true          {85x5 table}    {0x0 double}     0.19057    0.20361
```

Select the fourth row in the sharpness table, which corresponds to ROI 28. Display the SFR plot of the ROI.

```
idx = 4;
plotSFR(sharpnessTable(idx,:))
```



Print the MTF70 and MTF30 measurements of the ROI. Compare the measurements against the plot.

The MTF70 measurement of the red and blue color channels are slightly smaller than 0.2, while the MTF70 measurement of the green and luminance channels are slightly larger than 0.2. These measurements agree with a visual inspection of the SFR plot, on which an SFR value of 0.7 occurs at spatial frequencies around 0.2 line pairs per pixel.

```
mtf70 = sharpnessTable.MTF70(idx,:)
```

```
mtf70 = 1×4

    0.1906    0.2036    0.1959    0.2001
```

The MTF30 measurement of the blue color channel is noticeably smaller than the MTF30 measurement of the other color channels. This measurement agrees with a visual inspection of the SFR plot, on which the SFR curve of the blue channel drops off more quickly than the other channels.

```
mtf30 = sharpnessTable.MTF30(idx,:)
```

```
mtf30 = 1×4

    0.2619    0.2726    0.2613    0.2697
```

Input Arguments

`chart` — eSFR chart

`esfrChart` object

eSFR chart, specified as an `esfrChart` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'ROIIndex',2` measures the sharpness only of ROI 2.

`ROIIndex` — ROI indices

`1:60` (default) | scalar | vector

ROI indices to include in measurements, specified as the comma-separated pair consisting of `'ROIIndex'` and a scalar or vector of integers in the range [1, 60]. The indices match the ROI numbers displayed by `displayChart`.

Note `measureSharpness` uses the intersection of ROIs specified by `'ROIIndex'` and `'ROIOrientation'`.

Example: `29:32`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`ROIOrientation` — ROI orientation

`'both'` (default) | `'vertical'` | `'horizontal'`

ROI orientation, specified as the comma-separated pair consisting of `'ROIOrientation'` and `'both'`, `'vertical'`, or `'horizontal'`. The `measureSharpness` function performs measurements only on ROIs with the specified orientation.

Note `measureSharpness` uses the intersection of ROIs specified by `'ROIIndex'` and `'ROIOrientation'`.

Example: `'vertical'`

Data Types: `char` | `string`

`PercentResponse` — Value of frequency response

`50` (default) | scalar | vector

Value of frequency response at which to report the corresponding spatial frequency, specified as the comma-separated pair consisting of `'PercentResponse'` and a scalar or vector of integers in the range [1, 100].

Each value of `PercentResponse` adds two columns to the `sharpnessTable` and `aggregateSharpnessTable` output arguments. The columns indicate the frequency at which the

SFR drops to the specified percent of the initial and peak values. For example, when `PercentResponse` has the value 50, both output tables have the columns `MTF50` and `MTF50P`. These columns indicate the frequency at which the SFR drops to 50% of the initial value and peak value, respectively.

Example: `30`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `string`

Output Arguments

`sharpnessTable` — SFR measurements of edges

m-by-*n* table

SFR measurements of edges, returned as an *m*-by-*n* table. *m* is the number of sampled ROIs. *n* changes values depending on `PercentResponse`. The first five columns are always present and represent these variables:

Variable	Description
`ROI`	Index of the sampled ROI. The value of `ROI` is an integer in the range [1, 60].
`slopeAngle`	Angle between the slanted edge and pure vertical or horizontal, depending on the ROI orientation. The angle is measured in degrees, and it is returned as a scalar of type `double`.
`confidenceFlag`	Boolean flag that indicates whether the sharpness measurement is reliable. `confidenceFlag` is `true` when the measurement is reliable. `confidenceFlag` is `false` when the measurement is unreliable due to the following conditions: • `slopeAngle` is less than 3.5 degrees or more than 15 degrees. • The contrast within the ROI is less than 20%. The contrast of a slanted edge ROI is defined as `100 * (IHigh - ILow)/(IHigh + ILow)`, where `IHigh` and `ILow` are the estimated average intensities of the high and low intensity regions across the edge. The contrast is computed for only the red channel.
`SFR`	Spatial frequency response of the edge in the ROI. `SFR` is an *f*-by-5 table. The five columns represent the frequency value and the red, green, blue, and luminance values corresponding to that frequency. *f* is the number of frequency samples of the MTF. Luminance (*Y*) is a linear combination of the red (*R*), green (*G*), and blue (*B*) channels according to: $Y = 0.213R + 0.715G + 0.072B$
`comment`	When `confidenceFlag` is `false`, then `comment` describes the reason the measurement is unreliable. When `confidenceFlag` is `true`, then `comment` is the empty vector, `[]`.

Each value of `PercentResponse` adds two columns that indicate the frequency at which the SFR drops to the specified percent of the initial and peak value. The format of each entry in the column is

a 1-by-4 vector. The four elements correspond to the red, green, blue, and luminance channels, respectively.

`aggregateSharpnessTable` — Average SFR measurements of vertical and horizontal edges
table with one or two rows

Average SFR measurements of vertical and horizontal edges, returned as a table with one or two rows. `aggregateSharpnessTable` has one row when all sampled ROIs have the same orientation. It has two rows when the sampled ROIs have mixed orientation. `aggregateSharpnessTable` has three fewer columns than `sharpnessTable`.

The first two columns of `aggregateSharpnessTable` are always present and represent these variables:

Variable	Description
`Orientation`	Orientation of the averaged SFRs. The value of `Orientation` is either `'horizontal'` or `'vertical'`.
`SFR`	Averaged spatial frequency response of all edges in included ROIs with the orientation specified by `Orientation`. `SFR` is an *s*-by-5 table. The five columns represent the frequency value, and the averaged red, green, blue, and luminance values corresponding to that frequency. *s* is the number of frequency samples of the MTF. Luminance (*Y*) is computed as a linear combination of the red (*R*), green (*G*), and blue (*B*) channels according to: $Y = 0.213R + 0.715G + 0.072B$

Each value of `PercentResponse` adds two columns that indicate the frequency at which the SFR drops to the specified percent of the initial and peak value. The format of each entry in the column is a 1-by-4 vector. The four elements correspond to the red, green, blue, and luminance channels, averaged among all sampled ROIs with the same orientation.

Tips

- Slanted edges on a properly oriented chart are at an angle of 5 degrees from the horizontal or vertical. Sharpness measurements are not accurate when the edge orientation deviates significantly from 5 degrees.
- Sharpness is higher toward the center of the imaged region and decreases toward the periphery. Horizontal sharpness is usually higher than vertical sharpness.

Algorithms

The SFR measurement algorithm is based on work by Peter Burns [2] [3]. First, `measureSharpness` determines the edge position with sub-pixel resolution for each *scan line*, or row or column of pixels perpendicular to the edge, in the ROI. For example, each row of pixels is a scan line for a near-vertical edge. Next, `measureSharpness` aligns and averages the scan lines to create an oversampled edge intensity profile. The function takes the derivative of the intensity profile and applies a windowing function. The returned SFR measurement is the absolute value of the Fourier transform of the windowed derivative.

References

[1] Imatest. "Esfr". https://www.imatest.com/mathworks/esfr/.

[2] Burns, Peter. "Slanted-Edge MTF for Digital Camera and Scanner Analysis." *Society for Imaging Science and Technology; Proceedings of the Image Processing, Image Quality, Image Capture Systems Conference*. Portland, Oregon, March 2000, pp. 135–138.

[3] Burns, Peter. "sfrmat3: SFR evaluation for digital cameras and scanners." URL: http://losburns.com/imaging/software/SFRedge/sfrmat3_post/index.html.

See Also

`plotSFR` | `displayChart`

Topics

“Anatomy of Imatest Extended eSFR Chart”
“Evaluate Quality Metrics on eSFR Test Chart”
“Fourier Transform”

Introduced in R2017b

medfilt2

2-D median filtering

Syntax

```
J = medfilt2(I)
J = medfilt2(I,[m n])
J = medfilt2( ___ ,padopt)
```

Description

`J = medfilt2(I)` performs median filtering of the image `I` in two dimensions. Each output pixel contains the median value in a 3-by-3 neighborhood around the corresponding pixel in the input image.

`J = medfilt2(I,[m n])` performs median filtering, where each output pixel contains the median value in the `m`-by-`n` neighborhood around the corresponding pixel in the input image.

`J = medfilt2( ___ ,padopt)` controls how `medfilt2` pads the image boundaries.

Examples

Remove Salt and Pepper Noise from Image

Read image into workspace and display it.

```
I = imread('eight.tif');
figure, imshow(I)
```

Add salt and pepper noise.

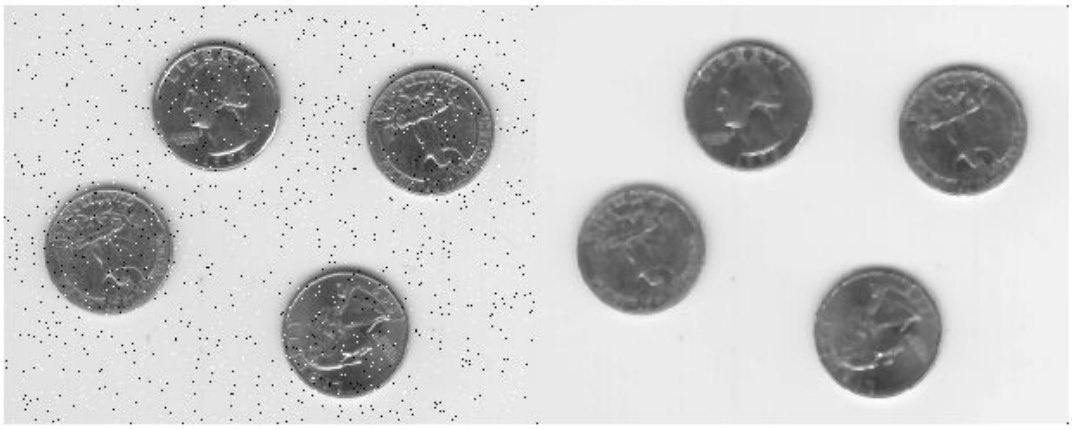
```
J = imnoise(I,'salt & pepper',0.02);
```

Use a median filter to filter out the noise.

```
K = medfilt2(J);
```

Display results, side-by-side.

```
imshowpair(J,K,'montage')
```



Input Arguments

`I` — Input image

2-D grayscale image | 2-D binary image

Input image, specified as a 2-D grayscale or binary image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`[m n]` — Neighborhood size

`[3 3]` (default) | 2-element vector

Neighborhood size, specified as a 2-element vector of positive integers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`padopt` — Padding option

`'zeros'` (default) | `'symmetric'` | `'indexed'`

Padding option, specified as one of the following values.

Value	Description
`'zeros'` (default)	Pad the image with `0`s.
`'symmetric'`	Symmetrically extend the image at the boundaries.
`'indexed'`	If the class of `I` is `double`, then pad the image with `1`s; otherwise, pad with `0`s.

Data Types: `char` | `string`

Output Arguments

`J` — Output image

numeric matrix

Output image, returned as a numeric matrix of the same class as the input image `I`.

Tips

- Median filtering is a nonlinear operation often used in image processing to reduce "salt and pepper" noise. A median filter is more effective than convolution when the goal is to simultaneously reduce noise and preserve edges. For information about performance considerations, see `ordfilt2`.
- If the input image `I` is of an integer class, then all the output values are returned as integers. If the number of pixels in the neighborhood (`m*n`) is even, then some of the median values might not be integers. In these cases, the fractional parts are discarded. Logical input is treated similarly. For example, the true median for the following 2-by-2 neighborhood in a `uint8` array is 4.5, but `medfilt2` discards the fractional part and returns 4.

  ```
  1 5
  4 8
  ```

- If you specify `padopt` as `'zeros'` or `'indexed'`, then the padding can skew the median near the image boundary. Pixels within one-half the width of the neighborhood (`[m n]/2`) of the edges can appear distorted.

Algorithms

On the CPU, `medfilt2` uses `ordfilt2` to perform the filtering.

References

[1] Lim, Jae S., *Two-Dimensional Signal and Image Processing*, Englewood Cliffs, NJ, Prentice Hall, 1990, pp. 469-476.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `medfilt2` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `medfilt2` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, the `padopt` argument must be a compile-time constant.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, the `padopt` argument must be a compile-time constant.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- Padding options, specified through the `padopt` argument, are not supported on the GPU.
- If you perform median filtering using a GPU, then the neighborhood `[m n]` must be square with odd-length sides between 3 and 15.

For more information, see “Image Processing on a GPU”.

See Also
`filter2` | `ordfilt2` | `wiener2` | `medfilt3`

Introduced before R2006a

medfilt3

3-D median filtering

Syntax

```
B = medfilt3(A)
B = medfilt3(A,[m n p])
B = medfilt3( ___ ,padopt)
```

Description

`B = medfilt3(A)` filters the 3-D image `A` with a 3-by-3-by-3 filter. By default, `medfilt3` pads the image by replicating the values in a mirrored way at the borders.

`B = medfilt3(A,[m n p])` performs median filtering of the 3-D image `A` in three dimensions. Each output voxel in `B` contains the median value in the *m*-by-*n*-by-*p* neighborhood around the corresponding voxel in `A`.

`B = medfilt3( ___ ,padopt)` controls how `medfilt3` pads the array boundaries.

Examples

Use Median Filtering to Remove Outliers in 3-D Data

Create a noisy 3-D surface.

```
[x,y,z,V] = flow(50);
noisyV = V + 0.1*double(rand(size(V))>0.95) - 0.1*double(rand(size(V))<0.05);
```

Apply median filtering.

```
filteredV = medfilt3(noisyV);
```

Display the noisy and filtered surfaces together.

```
subplot(1,2,1)
hpatch1 = patch(isosurface(x,y,z,noisyV,0));
isonormals(x,y,z,noisyV,hpatch1)
set(hpatch1,'FaceColor','red','EdgeColor','none')
daspect([1,4,4])
view([-65,20])
axis tight off
camlight left
lighting phong

subplot(1,2,2)
hpatch2 = patch(isosurface(x,y,z,filteredV,0));
isonormals(x,y,z,filteredV,hpatch2)
set(hpatch2,'FaceColor','red','EdgeColor','none')
daspect([1,4,4])
view([-65,20])
```

```
axis tight off
camlight left
lighting phong
```

Input Argument

A — Input image

3-D numeric array | 3-D logical array

Input image, specified as a 3-D numeric or logical array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

[m n p] — Neighborhood size

`[3 3 3]` (default) | 3-element vector

Neighborhood size, specified as a 3-element vector of positive odd integers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

padopt — Padding option

`'symmetric'` (default) | `'zeros'` | `'replicate'`

Padding option, specified as one of the following values:

Value	Description
`'symmetric'`	Pad array with mirror reflections of itself
`'replicate'`	Pad array by repeating border elements
`'zeros'`	Pad array with `0`s

Data Types: `char` | `string`

Output Arguments

B — Output image

3-D numeric array

Output image, returned as a 3-D numeric array of the same class and size as the input image `A`.

See Also

`medfilt2`

Introduced in R2016b

modefilt

2-D and 3-D mode filtering

Syntax

```
B = modefilt(A)
B = modefilt(A,filtsize)
B = modefilt( ___ ,padopt)
```

Description

`B = modefilt(A)` performs mode filtering on the 2-D image or 3-D volume `A`. Each output pixel in `B` contains the mode (most frequently occurring value) in the neighborhood around the corresponding pixel in `A`. If `A` is 2-D, `modefilt` uses a 3-by-3 mode filter. If `A` is 3-D, `modefilt` uses a 3-by-3-by-3 mode filter. `modefilt` pads `A` by mirroring border elements.

Mode filtering can be useful for processing categorical data, where other types of filtering, such as median filtering, are not available.

`B = modefilt(A,filtsize)` also specifies the size of the filter neighborhood. `filtsize` is a vector of positive, odd integers. When `A` is 2-D, specify `filtsize` as a 1-by-2 vector. When `A` is 3-D, specify `filtsize` as a 1-by-3 vector.

`B = modefilt( ___ ,padopt)` also specifies how `modefilt` pads array boundaries.

Examples

Apply Mode Filter to Categorical Labeled Image

Load an image (`img`) and the corresponding categorical labeled version of the image (`label`) into the workspace.

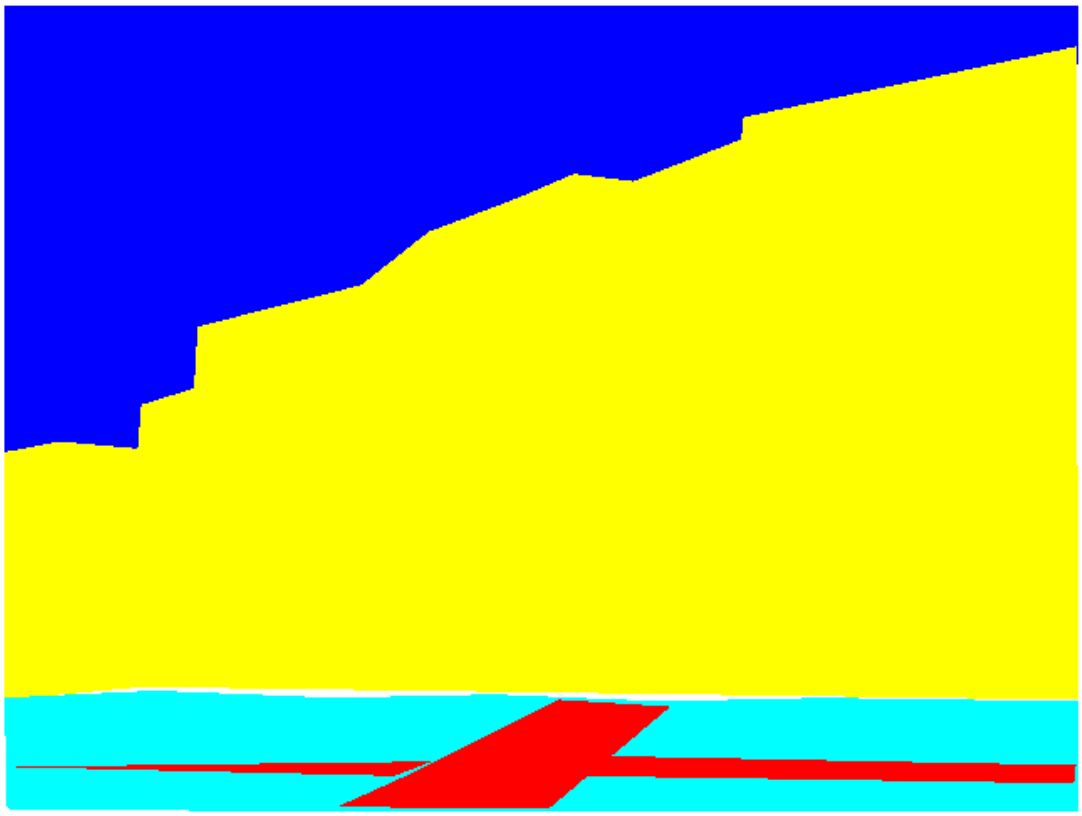
```
load buildingPixelLabeled;
```

View the original image, `img`.

```
imshow(img)
```

View the categorical labeled image, `label`. The categorical image labels four separate categories: sky, grass, building, and sidewalk. For viewing, convert these categories to colors using the `label2rgb` function.

```
imshow(label2rgb(label))
```



Perform mode filtering on the categorical labeled image, `label`, using the default filter size and padding method.

```
b = modefilt(label);
```

View the filtered categorical labeled image, `b`. In the filtered image, the edges between labeled regions are more distinct.

```
figure
imshow(label2rgb(b));
```

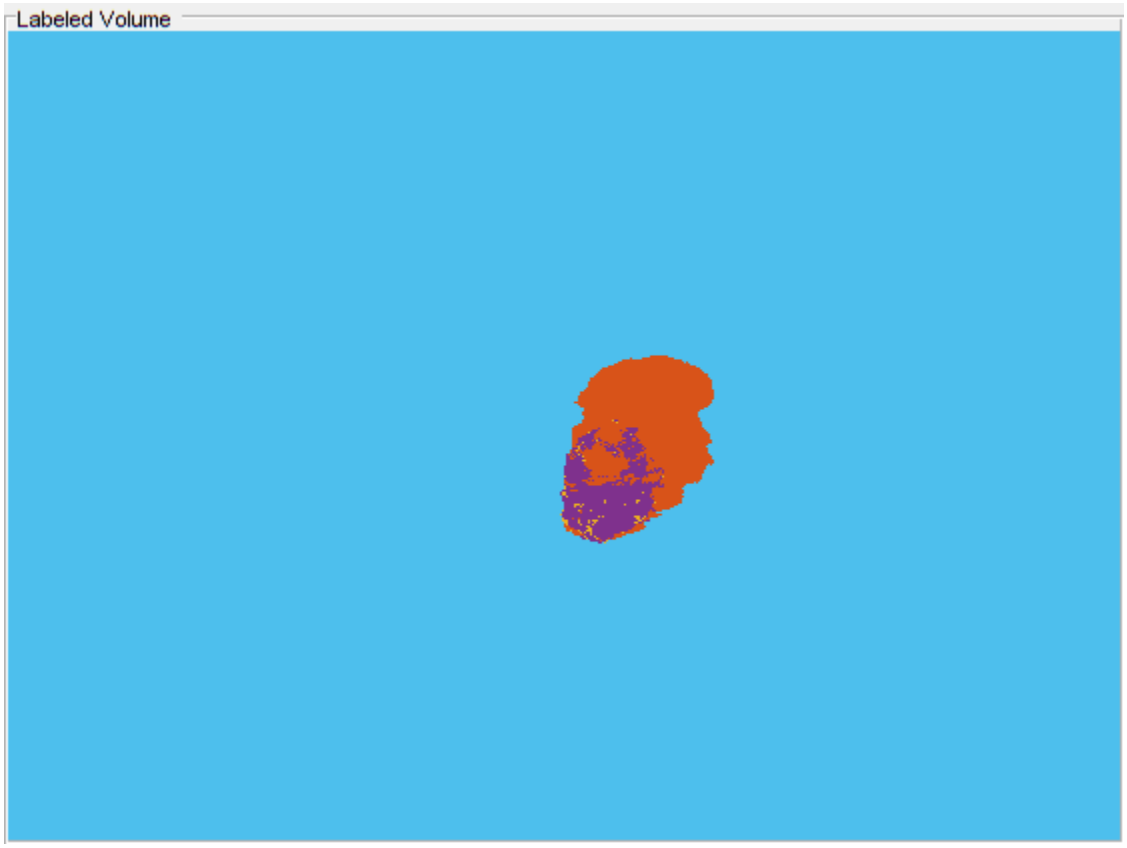
Use Mode Filter on Labeled Volume

Read a labeled volume of an MRI. The volume is stored in the workspace variable `label`.

```
load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled', ...
    'labels','label_001.mat'));
```

Display the labeled volume. For clarity, add a title to the display.

```
ViewPnl = uipanel(figure,'Title','Labeled Volume');
labelvolshow(label,'Parent',ViewPnl);
```

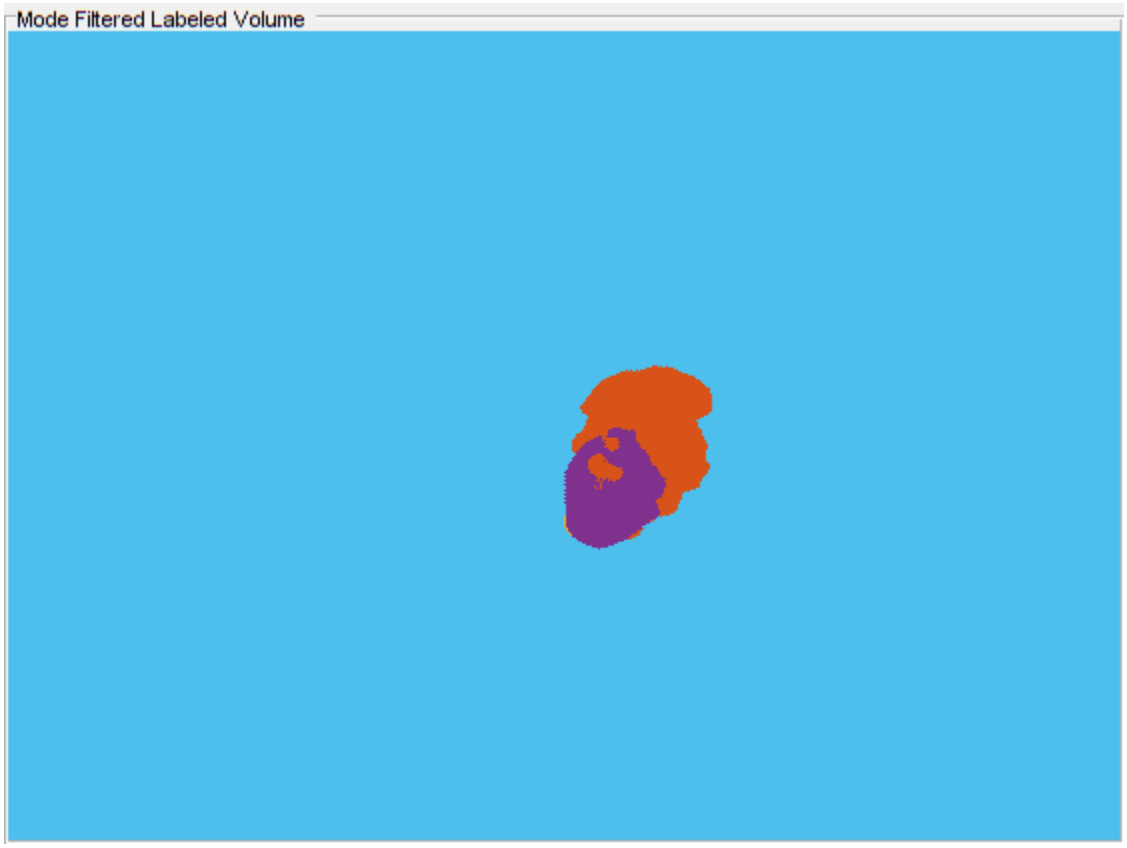


Perform mode filtering on the labeled volume, specifying the size of the filter.

```
labelOut = modefilt(label,[5 5 5]);
```

Display the filtered labeled volume. For clarity, add a title to the display.

```
ViewPnlFiltered = uipanel(figure,'Title','Mode Filtered Labeled Volume');
labelvolshow(labelOut,'Parent',ViewPnlFiltered);
```



Input Arguments

A — 2-D image or 3-D volume

2-D or 3-D categorical, logical, or numeric array

2-D image or 3-D volume, specified as a categorical, logical, or numeric array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical` | `categorical`

`filtsize` — Filter size

vector of positive odd integers.

Filter size, specified as a vector of positive odd integers. For 2-D images, specify a vector of the form `[height width]`. The default for 2-D images is `[3 3]`. For 3-D volumes, specify a vector of the form `[height width depth]`. The default for 3-D volumes is `[3 3 3]`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`padopt` — Padding method

`'symmetric'` (default) | `'replicate'` | `'zeros'`

Padding method, specified as one of the following values.

Value	Description
`'symmetric'`	Pad array with a mirror reflection of itself.
`'replicate'`	Pad array by repeating border elements.
`'zeros'`	Pad array with `0`s for numeric data or with `<undefined>`s for categorical data.

Example: `labelOut = modefilt(label,'replicate');`

Data Types: `char` | `string`

Output Arguments

B — Filtered image or volume

numeric array

Filtered image or volume, returned as a numeric array of the same size and class as the input image `A`.

Tips

- When the neighborhood has more than one pixel in a tie for the mode value, the function uses the following tie-breaking algorithm:
  - If the center pixel is one of the mode values in the tie, the function uses this value.
  - If the center pixel is not one of the mode values in the tie, the function uses the mode with the smallest numeric value.
  - For categorical input, the function chooses the first category (among the categories tied for mode) that appears in the list returned by `categories(A)`.
- `modefilt` treats RGB images as 3-D volumes. To do channel-wise filtering of an RGB image, specify `filtsize` as `[3 3 1]`, as in this code: `b = modefilt(a,[3 3 1]);` .

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `modefilt` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- Only 1-D and 2-D inputs are supported.
- Input images of data type `categorical` are not supported.
- The `padopt` argument must be a compile-time constant.

See Also

`mode` | `ordfilt2` | `medfilt2` | `medfilt3`

Introduced in R2020a

montage

Display multiple image frames as rectangular montage

Syntax

```
montage(I)
montage(imagelist)
montage(filenames)
montage(imds)
montage( ___ ,map)
montage( ___ ,Name,Value)
img = montage( ___ )
```

Description

`montage(I)` displays all frames of a multiframe image array `I`. By default, the `montage` function arranges the images so that they roughly form a square.

`montage(imagelist)` displays a montage of images specified in the cell array `imagelist`. The images can be of different types and sizes.

`montage(filenames)` displays a montage of the images with file names specified in `filenames`.

`montage(imds)` displays a montage of the images specified in the image datastore `imds`.

`montage( ___ ,map)` treats all grayscale and binary images (specified using any of the preceding syntaxes) as indexed images and displays them with the specified colormap `map`. If you specify images using file names or an image datastore, then `map` overrides any internal colormap present in the image files. `montage` does not modify the colormap of RGB images.

`montage( ___ ,Name,Value)` uses name-value pair arguments to customize the display of the image montage.

`img = montage( ___ )` returns a handle to the single image object that contains all the frames displayed.

Examples

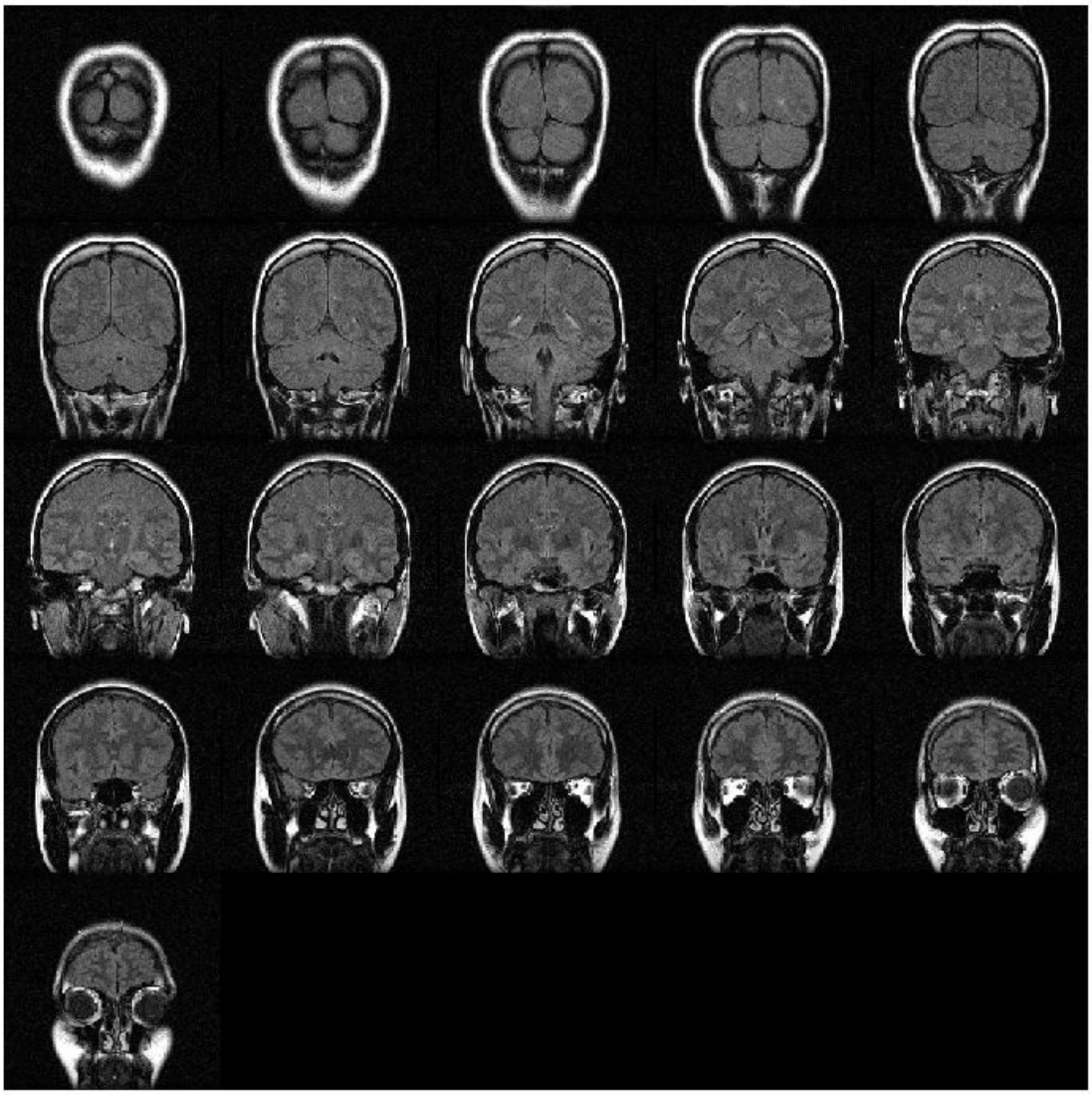
Create Montage from Multiframe Image

Load an MRI dataset.

```
load mristack
```

Display the dataset. `montage` treats the data as a multiframe image and displays each slice.

```
montage(mristack)
```



Create Montage Containing Images of Different Types and Sizes

Read several images of different types and sizes into the workspace.

```
imRGB = imread('peppers.png');
imGray = imread('coins.png');
```

Display a montage containing all of the images.

```
figure
montage({imRGB, imGray, 'cameraman.tif'})
```

Create Montage from Images in Files

Create a montage from a series of images in files. Make the montage a 2-by-5 rectangle. Then, create a second montage, this time using the `'DisplayRange'` name-value argument to highlight structures in the image.

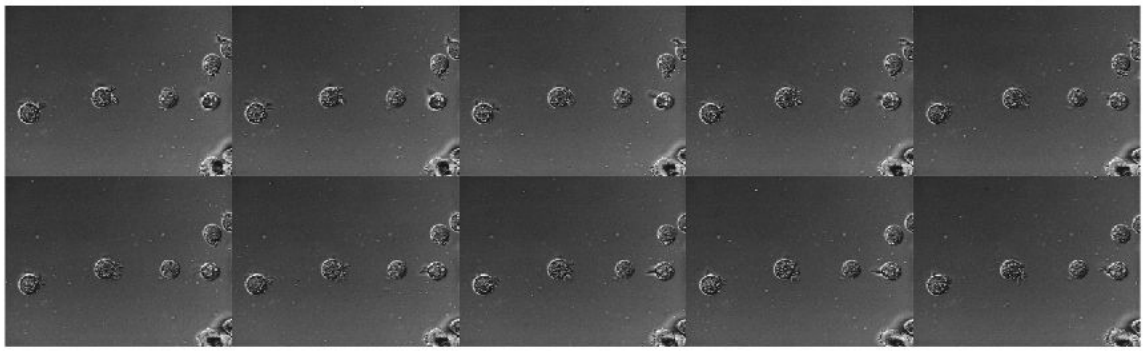
Display the Images as a Rectangular Montage

Create a string array containing a series of file names.

```
fileFolder = fullfile(matlabroot,'toolbox','images','imdata');
dirOutput = dir(fullfile(fileFolder,'AT3_1m4_*.tif'));
fileNames = string({dirOutput.name});
```

Display the images as a montage. Specify the shape of the montage as a 2-by-5 rectangle.

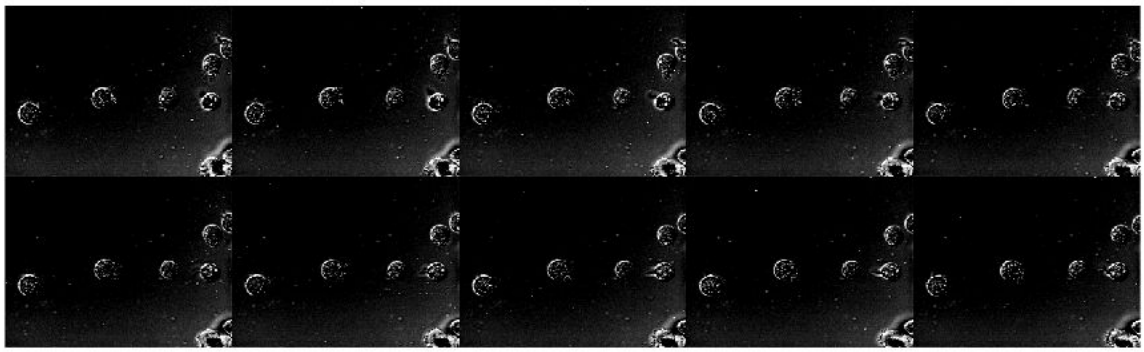
```
montage(fileNames,'Size',[2 5]);
```



Adjust the Contrast of the Images in the Montage

In another figure, create the same 2-by-5 montage. In addition, specify the display range to adjust the contrast of the images in the montage.

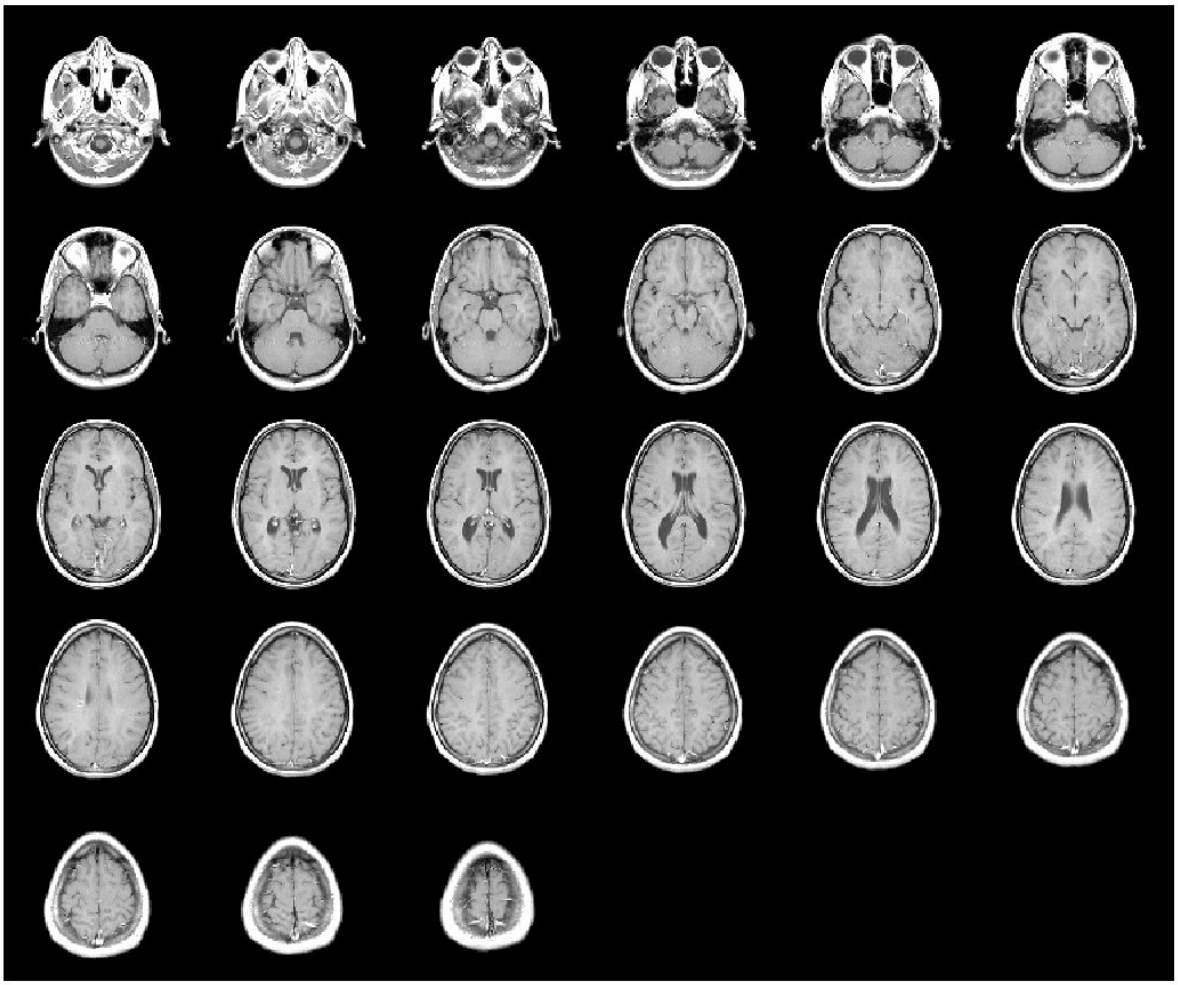
```
figure
montage(fileNames,'Size',[2 5],'DisplayRange',[75 200]);
```



Customize Number of Images in Montage

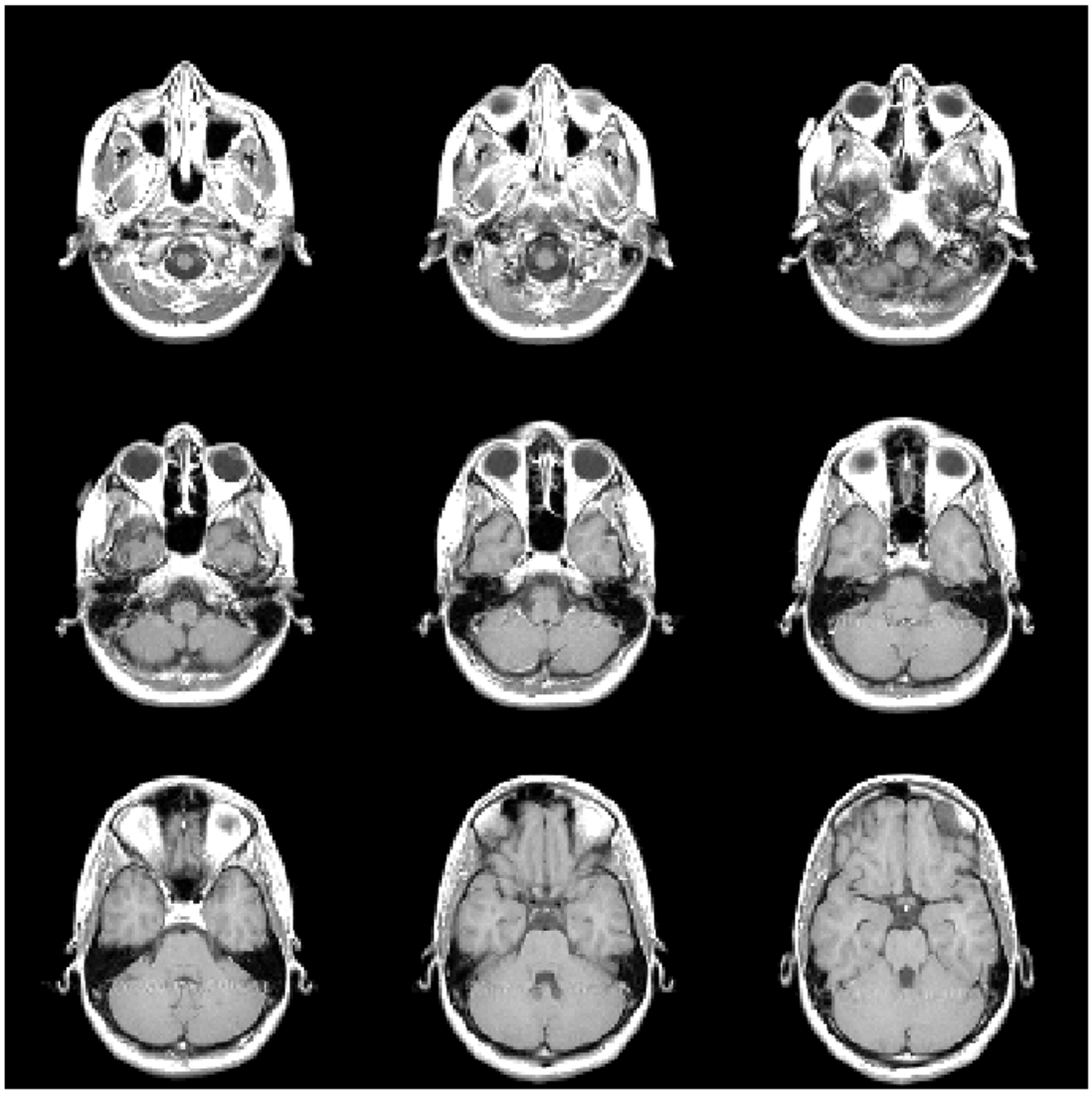
View all the images in a MRI data set using montage with default settings. There are 27 images in the set.

```
load mri
montage(D,map)
```



Create a new montage containing only the first 9 images.

```
figure
montage(D,map,Indices=1:9);
```



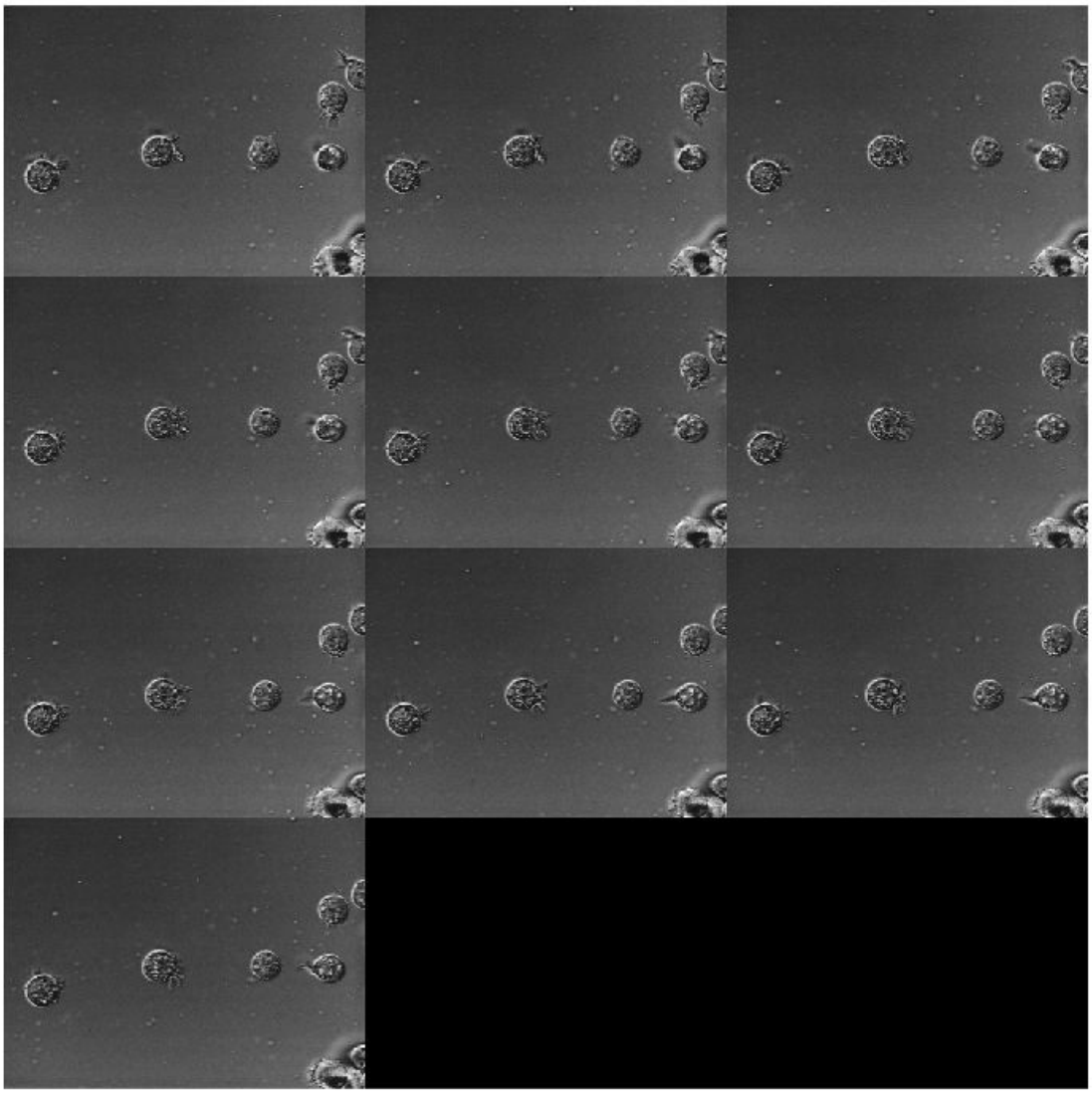
Create Montage from Image Datastore

Create an `ImageDatastore` object containing a series of ten images from the Image Processing Toolbox™ sample image folder.

```
fileFolder = fullfile(matlabroot,'toolbox','images','imdata');
imds = imageDatastore(fullfile(fileFolder,'AT3*'));
```

Display the contents of the datastore as a montage.

```
montage(imds)
```



Input Arguments

`I` — Multiframe image array

numeric array

Multiframe image array, specified as one of the following:

- *m*-by-*n*-by-*k* numeric array representing a sequence of *k* binary or grayscale images
- *m*-by-*n*-by-1-by-*k* numeric array representing a sequence of *k* binary or grayscale images
- *m*-by-*n*-by-3-by-*k* numeric array representing a sequence of *k* truecolor images

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`imagelist` — Set of images
cell array of numeric matrices

Set of images, specified as a cell array of numeric matrices of size *m*-by-*n* or *m*-by-*n*-by-3.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical` | `cell`

`filenames` — Names of files containing images
cell array of character vectors | vector of strings

Names of files containing image, specified as a cell array of character vectors or a vector of strings. If the files are not in the current folder or in a folder on the MATLAB path, then specify the full path name. For more information, see `imread`.

Data Types: `char` | `string` | `cell`

`imds` — Image datastore
`ImageDatastore` object

Image datastore, specified as an `ImageDatastore` object.

`map` — Colormap
c-by-3 numeric matrix

Colormap, specified as a *c*-by-3 numeric matrix with values in the range [0, 1]. Each row is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'Indices',1:9` creates a montage of the first nine frames

`BackgroundColor` — Background color
`'black'` (default) | RGB triplet | color name | short color name

Background color, specified as specified as an RGB triplet, a color name, or a short color name. The `montage` function fills all blank spaces with the background color, including the space specified by `BorderSize`. If you specify a background color, then the `montage` function renders the output as an RGB image.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'BackgroundColor','r'`

Example: `'BackgroundColor','green'`

Example: `'BackgroundColor',[0 0.4470 0.7410]`

`BorderSize` — Padding around each thumbnail image
`[0 0]` (default) | nonnegative integer | 1-by-2 vector of nonnegative integers

Padding around each thumbnail image, in pixels, specified as a nonnegative integer or a 1-by-2 vector of nonnegative integers. The `montage` function pads the image borders with the background color, `BackgroundColor`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`DisplayRange` — Display range
1-by-2 vector

Display range of grayscale images in array `I`, specified as 1-by-2 vector of the form `[`*`low high`*`]`. All pixel values less than or equal to *low* display as black. All pixel values greater than or equal to *high* display as white. If you specify an empty matrix (`[]`), then `montage` uses the minimum and maximum pixel values of the images.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Indices` — Frames to display
array of positive integers

Frames to display in the montage, specified as an array of positive integers. The `montage` function interprets the values as indices into array `I` or into cell array `filenames` or `imagelist`.

By default, the `montage` function displays all frames or image files.

Example: `'Indices',1:4` create a montage of the first four frames in `I`

Example: `'Indices',1:2:20` displays every other frame.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Interpolation` — Interpolation technique
`'nearest'` (default) | `'bilinear'`

Interpolation technique used when scaling an image, specified as the comma-separated pair consisting of `'Interpolation'` and one of these values.

Value	Description
`'nearest'`	Nearest neighbor interpolation (default)
`'bilinear'`	Bilinear interpolation

`Parent` — Parent of image object
axes object

Parent of the image object created by `montage`, specified as an axes object. The `montage` function resizes the image to fit the extents available in the parent axes.

`Size` — Number of rows and columns of images
2-element vector

Number of rows and columns of images, specified as a 2-element vector of the form `[nrows ncols]`.

If you specify `NaN` or `Inf` for a particular dimension, then the `montage` function calculates the value of the dimension to display all images in the montage. For example, if `'Size'` is `[2 NaN]`, then the montage will have two rows and the minimum number of columns to display all images. When there is a mismatch between `'Size'` and number of images (frames) specified, then the `montage` function creates the tiled image based on `'Size'`.

Data Types: `single` | `double`

`ThumbnailSize` — Size of each thumbnail
2-element vector of positive integers | `[]`

Size of each thumbnail, in pixels, specified as a 2-element vector of positive integers. The aspect ratio of each image is preserved, and any blank space is filled with the background color, `BackgroundColor`.

If you specify an empty array (`[]`), then the thumbnail size is the full size of the first image. If you specify either element as `NaN` or `Inf`, then the `montage` function calculates the corresponding value automatically to preserve the aspect ratio of the first image.

Data Types: `single` | `double`

Output Arguments

`img` — Montage image
`Image` object

Montage image, returned as an Image object.

Tips

- If you specify an indexed image, then `montage` converts it to RGB using the colormap present in the file.
- If there is a data type mismatch between images, then the `montage` function converts all images to data type `double` using the `im2double` function.
- When calculating the number of images to display horizontally and vertically, `montage` considers the aspect ratio of the images, so that the displayed montage is nearly square.

See Also

Apps
Video Viewer | Volume Viewer

Functions
`imshow` | `ImageDatastore` | `imtile`

Introduced before R2006a

multissim

Multiscale structural similarity (MS-SSIM) index for image quality

Syntax

```
score = multissim(I,Iref)
score = multissim(I,Iref,Name,Value)
[score,qualityMaps] = multissim( ___ )
```

Description

`score = multissim(I,Iref)` calculates the multi-scale structural similarity (MS-SSIM) index, `score`, for image `I`, using `Iref` as the reference image. A value closer to 1 indicates better image quality and a value closer to 0 indicates poorer quality.

MS-SSIM is only defined for grayscale images. For inputs with more than two dimensions, `multissim` treats each element of higher dimensions as a separate 2-D grayscale image.

`score = multissim(I,Iref,Name,Value)` controls aspects of the computation using one or more name-value arguments. For example, specify the number of scales using the `'NumScales'` argument.

`[score,qualityMaps] = multissim( ___ )` also returns the local MS-SSIM index value for each pixel in each scaled version of `I`. The `qualitymap` output is a cell array containing maps for each of the scaled versions of `I`. Each quality map is the same size as the corresponding scaled version of `I`.

Examples

Calculate MS-SSIM

Load an image into the workspace.

```
Iref = imread('pout.tif');
```

Create a noisy version of the image for comparison purposes.

```
I = imnoise(Iref,'salt & pepper',0.05);
```

Display the original image and noisy image.

```
figure;
montage({Iref,I});
```

Calculate the MS-SSIM index that measures the quality of the input image compared to the reference image.

```
score = multissim(I,Iref)
```

```
score = single
    0.6732
```

Calculate MS-SSIM and Get Local MS-SSIM Maps

Load an image into the workspace.

```
Iref = imread('pout.tif');
I = Iref;
```

Add noise to a localized part of the image.

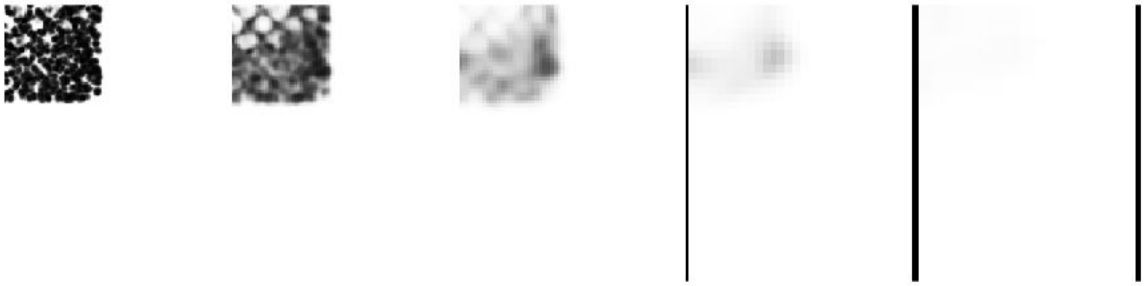
```
I(1:100,1:100) = imnoise(Iref(1:100,1:100),'salt & pepper',0.05);
```

Display the original image and the noisy image.

```
figure;
montage({Iref,I});
```

Calculate the local MS-SSIM index maps for the noisy image, `qualitymaps`, using the original image as the reference. The return value, `qualitymaps`, is a cell array containing a quality map for each of the scaled versions of the image. Each map is the same size as the corresponding scaled version of the image.

```
[~, qualitymaps] = multissim(I,Iref);
figure
montage(qualitymaps,'Size',[1 5])
```



Calculate MS-SSIM Specifying Scale Weights

Load an image into the workspace.

```
Iref = imread('pout.tif');
```

Create a noisy version of the image for comparison purposes.

```
I = imnoise(Iref,'salt & pepper',0.05);
```

Display the original image and the noisy version of the image.

```
figure;
montage({Iref,I});
```

Calculate the MS-SSIM index for the noisy image, using the original image as the reference. Specify how much to weigh the local MS-SSIM index calculations for each scaled image, using the `'ScaleWeights'` argument. The example uses the weight values defined in the article by Wang, Simoncelli, and Bovik.

```
score = multissim(I,Iref,'ScaleWeights',[0.0448,0.2856,0.3001,0.2363,0.1333])
```

```
score = single
    0.6773
```

Calculate MS-SSIM of Color Image

Read a color image into the workspace.

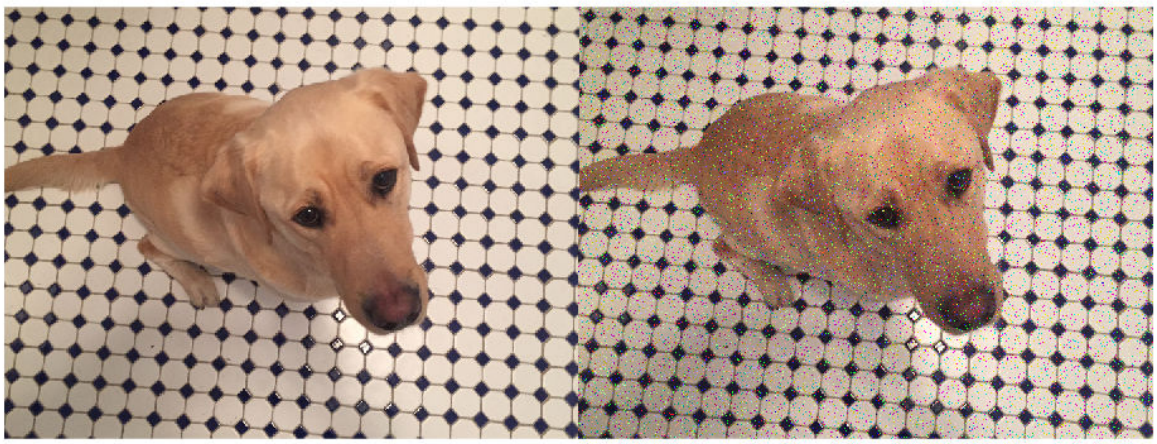
```
RGB = imread("kobi.png");
```

Create a version of the image with added salt and pepper noise.

```
RGBNoisy = imnoise(RGB,"salt & pepper");
```

Display the two images in a montage.

```
montage({RGB,RGBNoisy})
```



Calculate the MS-SSIM of each color channel of the noisy image.

```
score = multissim(RGBNoisy,RGB);
score = squeeze(score)
```

```
score = 3x1 single column vector

    0.7084
    0.7135
    0.7066
```

Calculate MS-SSIM for `dlarray` Input

Read a color image into the workspace.

```
ref = imread("strawberries.jpg");
ref = im2single(ref);
```

Simulate a batch of six images by replicating the image along the fourth dimension.

```
refBatch = repmat(ref,[1 1 1 6]);
```

Create a copy of the batch of images, adding salt and pepper noise.

```
noisyBatch = imnoise(refBatch,"salt & pepper");
```

Create a formatted `dlarray` object for the original and noisy batch of images. The format is "SSCB" for spatial-spatial-channel-batch.

```
dlrefBatch = dlarray(refBatch,"SSCB");
dlnoisyBatch = dlarray(noisyBatch,"SSCB");
```

Calculate the MS-SSIM score of the noisy data with respect to the original data.

```
scores = multissim(dlnoisyBatch,dlrefBatch);
```

Remove the singleton dimensions corresponding to the spatial dimensions and display the scores. Each element is the MS-SSIM score for one color channel of one image of the batch.

```
squeeze(scores)
```

```
ans =
  3(C) x 6(B) single dlarray

    0.8334    0.8335    0.8348    0.8335    0.8340    0.8349
    0.8325    0.8316    0.8309    0.8310    0.8317    0.8326
    0.8140    0.8123    0.8166    0.8129    0.8136    0.8123
```

Input Arguments

`I` — Input image

numeric array | `dlarray` object

Input image, specified as a numeric array of any dimension or a `dlarray` object. Formatted `dlarray` objects cannot include more than one channel label, more than one batch label, and more than two spatial labels.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`Iref` — Reference image

numeric array | `dlarray` object

Reference image, specified as a numeric array of any dimension or a `dlarray` object. Formatted `dlarray` objects cannot include more than one channel label, more than one batch label, and more than two spatial labels.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `score = multissim(I,Iref,'NumScales',3);`

`NumScales` — Number of scales
5 (default) | positive integer

Number of scales used to calculate the MS-SSIM index, specified as the comma-separated pair consisting of `'NumScales'` and a positive integer. Setting `'NumScales'` to `1` is equivalent to using the `ssim` function with its `'Exponents'` name-value pair argument set to `[1 1 1]`. The size of the input image limits the number of scales. The `multissim` function scales the image (`NumScales` - 1) times, downsampling the image by a factor of 2 with each scaling.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ScaleWeights` — Relative values across scales
vector of positive numbers

Relative values across the scales, specified as the comma-separated pair consisting of `'ScaleWeights'` and a vector of positive elements. The length of the vector is equal to the number of scales, because each element corresponds to one of the scaled versions of the original image. The `multissim` function normalizes the values to 1. By default, the scale weights equal `fspecial('gaussian',[1,numScales],1)`. The `multissim` function uses a Gaussian distribution as the default because the human visual sensitivity peaks at middle frequencies and decreases in both directions. For an example of setting `'ScaleWeights'`, see “Calculate MS-SSIM Specifying Scale Weights” on page 1-2409.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Sigma` — Standard deviation
`1.5` (default) | positive number

Standard deviation of the isotropic Gaussian function, specified as the comma-separated pair consisting of `'Sigma'` and a positive number. This value specifies the weighting of the neighborhood pixels around a pixel for estimating local statistics. The `multissim` function uses weighting to avoid blocking artifacts when estimating local statistics.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`DynamicRange` — Dynamic range of input image
positive number

Dynamic range of the input image, specified as a positive number. The default value of `DynamicRange` depends on the data type of image `I`, and is calculated as `diff(getrangefromclass(I))`. For example, the default dynamic range is `255` for images of data type `uint8`, and the default is `1` for images of data type `double` or `single` with pixel values in the range [0, 1].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output Arguments

`score` — MS-SSIM index
numeric scalar | numeric array | `dlarray` object

MS-SSIM index for image quality, returned as a numeric scalar, numeric array, or `dlarray` object as indicated in the table. The value of `score` is typically in the range [0, 1]. The value 1 indicates the highest quality and occurs when `I` and `Iref` are equivalent. Smaller values correspond to poorer quality. For some combinations of inputs and name-value pair arguments, `score` can be negative.

Input Image Type	MS-SSIM Value
2-D numeric matrices	Numeric scalar with a single MS-SSIM measurement.
2-D `dlarray` objects	1-by-1 `dlarray` object with a single MS-SSIM measurement.
N-D numeric arrays with N>2	Numeric array of the same dimensionality as the input images. The first two dimensions of `score` are singleton dimensions. There is one MS-SSIM measurement for each element along the higher dimensions.
Unformatted N-D `dlarray` objects with N>2	`dlarray` object of the same dimensionality as the input images. The first two dimensions of `score` are singleton dimensions. There is one MS-SSIM measurement for each element along the higher dimensions.
Formatted N-D `dlarray` objects with N>2	`dlarray` object of the same dimensionality as the input images. The spatial dimensions of `score` are singleton dimensions. There is one MS-SSIM measurement for each element along any channel or batch dimension.

`qualityMaps` — Local MS-SSIM index values
cell array of numeric arrays | cell array of `dlarray` objects

Local MS-SSIM index values for each pixel in each scaled version, returned as a cell array of numeric arrays or a cell array of `dlarray` objects. The size of the cell array is 1-by-`NumScales`. Each element in `qualityMaps` indicates the quality of the corresponding pixel at the corresponding scale factor. The format of each element uses the formatting of the `scores` argument, based on the format of the input images.

Algorithms

The structural similarity (SSIM) index measures perceived quality by quantifying the SSIM between an image and a reference image (see `ssim`). The `multissim` function calculates the MS-SSIM index by combining the SSIM index of several versions of the image at various scales. The MS-SSIM index can be more robust when compared to the SSIM index with regard to variations in viewing conditions.

The `multissim` function uses double-precision arithmetic for input images of class `double`. All other types of input images use single-precision arithmetic.

References

[1] Wang, Z., Simoncelli, E.P., Bovik, A.C. *Multiscale Structural Similarity for Image Quality Assessment*. In: The Thirty-Seventh Asilomar Conference on Signals, Systems & Computers, 2003, 1398–1402. Pacific Grove, CA, USA: IEEE, 2003. https://doi.org/10.1109/ACSSC.2003.1292216.

Extended Capabilities

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`ssim` | `multissim3` | `psnr`

Topics

“Image Quality Metrics”

Introduced in R2020a

multissim3

Multiscale structural similarity (MS-SSIM) index for volume quality

Syntax

```
score = multissim3(V,Vref)
score = multissim3(V,Vref,Name,Value)
[score,qualityMaps] = multissim3( ___ )
```

Description

`score = multissim3(V,Vref)` calculates the multiscale structural similarity (MS-SSIM) index, `score`, for volume `V`, using `Vref` as the reference volume. A value closer to 1 indicates better quality and a value closer to 0 indicates poorer quality.

The 3-D MS-SSIM operation is defined for grayscale volumes. For inputs with more than three dimensions, `multissim3` treats each element of higher dimensions as separate 3-D grayscale volumes. `multissim3` treats 2-D RGB images as 3-D grayscale volumes. To calculate the MS-SSIM of color channels in an RGB image, use the `multissim` function.

`score = multissim3(V,Vref,Name,Value)` controls aspects of the computation using one or more name-value arguments. For example, specify the number of scales using the `'NumScales'` argument.

`[score,qualityMaps] = multissim3( ___ )` also returns the local MS-SSIM index value for each voxel in `V`, and each of the scaled versions of `V`. The `qualityMaps` output is a cell array containing maps for each of the scaled versions of `V`, with each quality map the same size as the corresponding scaled version.

Examples

Calculate the MS-SSIM Index for Volume

Load a 3-D volume into the workspace.

```
load mri D
Vref = squeeze(D);
```

Create a noisy version of the original volume for quality measurement comparison purposes.

```
V = imnoise(Vref,'salt & pepper',0.05);
```

Calculate the MS-SSIM index that measures the quality of the input volume compared to the reference volume.

```
score = multissim3(V,Vref)
```

```
score = single
    0.7261
```

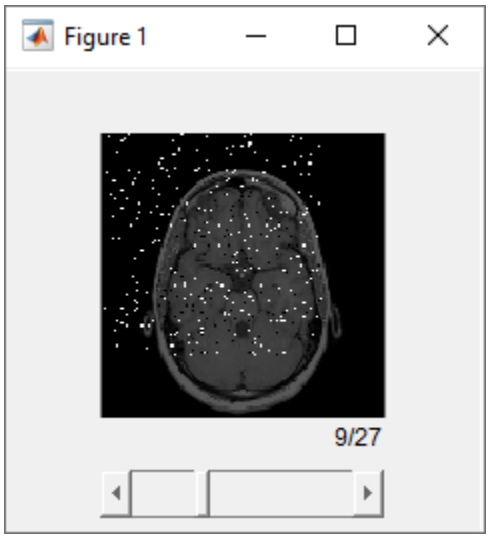
Calculate MS-SSIM and Retrieve Local Structural Similarity Maps

Load a volume into the workspace. This volume will be the reference volume. Create a copy of the reference volume.

```
load mri D
Vref = squeeze(D);
V = Vref;
```

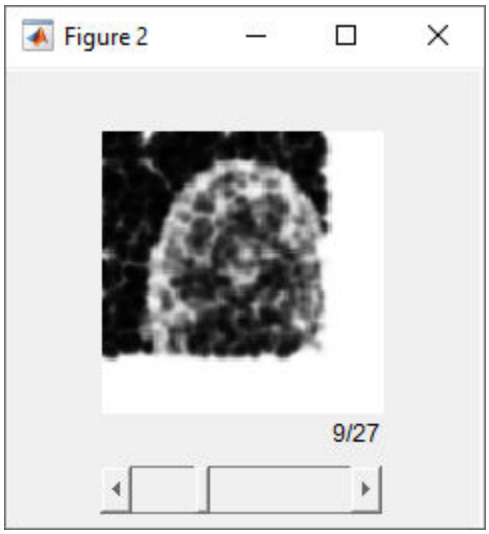
Add noise to a localized part of the volume for quality comparison purposes.

```
V(1:100,1:100,1:10) = imnoise(Vref(1:100,1:100,1:10),'salt & pepper',0.05);
figure
sliceViewer(V);
```



Calculate the MS-SSIM index for the volumes and retrieve the local structural similarity maps. The `multissim3` function returns `qualitymaps`, a cell array containing a local structural similarity map for each of the scaled versions of the volume. In the quality map, the value 1 indicates the highest quality.

```
[score, qualitymaps] = multissim3(V,Vref);
figure
sliceViewer(V);
```



Calculate MS-SSIM Specifying Weights for Each Scaled Volume

Load a volume into the workspace.

```
load mri D
Vref = squeeze(D);
```

Create a noisy version of the volume for quality measurement comparison purposes.

```
V = imnoise(Vref,'salt & pepper',0.05);
```

Calculate the MS-SSIM index for the noisy volume, using the original volume as the reference. Specify how much to weigh the local MS-SSIM index calculations for each scaled volume using the `'ScaleWeights'` argument. The example uses the weights defined in the article by Wang, Simoncelli, and Bovik.

```
score = multissim3(V,Vref,'ScaleWeights',[0.0448,0.2856,0.3001,0.2363,0.1333]);
```

Calculate MS-SSIM of 3-D Volume Stack

Read a volumetric image into the workspace.

```
VRef = load("mristack.mat");
VRef = im2single(VRef.mristack);
```

Simulate a batch of six images by replicating the image along the fourth dimension.

```
VRefBatch = repmat(VRef,[1 1 1 6]);
```

Create a version of the image stack with added salt and pepper noise.

```
VNoisyBatch = imnoise(VRefBatch,"salt & pepper");
```

Calculate the MS-SSIM of each volume in the stack.

```
score = multissim3(VNoisyBatch,VRefBatch)

score = 1x1x1x6 single array
score(:,:,1,1) =

    0.8341


score(:,:,1,2) =

    0.8347


score(:,:,1,3) =

    0.8337


score(:,:,1,4) =

    0.8333


score(:,:,1,5) =

    0.8348


score(:,:,1,6) =

    0.8343
```

Calculate MS-SSIM for 3-D `dlarray` Input

Read a volumetric image into the workspace.

```
VRef = load("mristack.mat");
VRef = im2single(VRef.mristack);
```

Create a copy of the batch of images, adding salt and pepper noise.

```
Vnoisy = imnoise(VRef,"salt & pepper");
```

Create unformatted `dlarray` objects for the original and noisy batch of images.

```
dlref = dlarray(VRef);
dlnoisy = dlarray(Vnoisy);
```

Calculate the MS-SSIM score of the noisy data with respect to the original data.

```
score = multissim3(dlnoisy,dlref)

score =
  1x1 single dlarray
```

```
    0.8341
```

Input Arguments

V — Input volume

numeric array | `dlarray` object

Input volume, specified as a numeric array of three or more dimensions or a `dlarray` object. Formatted `dlarray` objects cannot include more than one channel label, more than one batch label, and more than three spatial labels.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

Vref — Reference volume

numeric array

Reference volume, specified as a numeric array of three or more dimensions or a `dlarray` object. Formatted `dlarray` objects cannot include more than one channel label, more than one batch label, and more than three spatial labels. The reference volume must be of the same size and data type as the input volume, `V`.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `score = multissim3(V,Vref,'NumScales',3);`

NumScales — Number of scales

5 (default) | positive integer

Number of scales used to calculate MS-SSIM, specified as the comma-separated pair consisting of `'NumScales'` and a positive integer. Setting `'NumScales'` to `1` is equivalent to the use of the `ssim` function with the `'Exponents'` name-value pair argument set to `[1 1 1]`. The size of the input volume limits the number of scales. The `multissim3` function scales the volume (`NumScales` - 1) times, downsampling the volume by a factor of 2 with each scaling.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

ScaleWeights — Relative values across scales

vector of positive numbers

Relative values across the scales, specified as the comma-separated pair consisting of `'ScaleWeights'` and a vector of positive numbers. The length of the vector is equal to the number of scales, because each element corresponds to one of the scaled versions of the original volume. The `multissim3` function normalizes the values to 1. By default, the scale weights equal `fspecial('gaussian',[1,numScales],1)`. The `multissim3` function uses a Gaussian distribution as the default because the human visual sensitivity peaks at middle frequencies and

decreases in both directions. For an example of setting `'ScaleWeights'`, see “Calculate MS-SSIM Specifying Weights for Each Scaled Volume” on page 1-2418.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Sigma` — Standard deviation
`1.5` (default) | positive number

Standard deviation of the isotropic Gaussian function, specified as the comma-separated pair consisting of `'Sigma'` and a positive number. This value specifies the weighting of the neighborhood voxels around a voxel for estimating local statistics. The `multissim3` function uses this weighting to avoid blocking artifacts in estimating local statistics.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`DynamicRange` — Dynamic range of input volume
positive number

Dynamic range of the input volume, specified as a positive number. The default value of `DynamicRange` depends on the data type of volume `V`, and is calculated as `diff(getrangefromclass(V))`. For example, the default dynamic range is `255` for volumes of data type `uint8`, and the default is `1` for volumes of data type `double` or `single` with voxel values in the range [0, 1].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output Arguments

`score` — MS-SSIM index
numeric scalar | numeric array | `dlarray` object

MS-SSIM index for image quality, returned as a numeric scalar, numeric array, or `dlarray` object as indicated in the table. The value of `score` is typically in the range [0, 1]. The value 1 indicates the highest quality and occurs when `V` and `Vref` are equivalent. Smaller values correspond to poorer quality. For some combinations of inputs and name-value pair arguments, `score` can be negative.

Input Volume Type	MS-SSIM Value
3-D numeric matrices	Numeric scalar with a single MS-SSIM measurement.
3-D `dlarray` objects	1-by-1 `dlarray` object with a single MS-SSIM measurement.
N-D numeric arrays with N>3	Numeric array of the same dimensionality as the input volumes. The first three dimensions of `score` are singleton dimensions. There is one MS-SSIM measurement for each element along the higher dimensions.
Unformatted N-D `dlarray` objects with N>3	`dlarray` object of the same dimensionality as the input volumes. The first three dimensions of `score` are singleton dimensions. There is one MS-SSIM measurement for each element along the higher dimensions.

Input Volume Type	MS-SSIM Value
Formatted N-D `dlarray` objects with N>3	`dlarray` object of the same dimensionality as the input volumes. The spatial dimensions of `score` are singleton dimensions. There is one MS-SSIM measurement for each element along any channel or batch dimension.

`qualityMaps` — Local MS-SSIM index values
cell array of numeric arrays | cell array of `dlarray` objects

Local MS-SSIM index values for each pixel in each scaled version, returned as a cell array of numeric arrays or a cell array of `dlarray` objects. The size of the cell array is 1-by-`NumScales`. Each element in `qualityMaps` indicates the quality of the corresponding pixel at the corresponding scale factor. The format of each element uses the formatting of the `score` argument, based on the format of the input volumes.

Algorithms

The structural similarity (SSIM) index measures perceived quality by quantifying the structural similarity between a volume and a reference volume (see `ssim`). The `multissim3` function calculates the MS-SSIM by combining the SSIM index of several versions of the volume at various scales. The MS-SSIM index can be more robust when compared to the SSIM index with regard to variations in viewing conditions.

The `multissim3` function uses double-precision arithmetic for input volumes of class `double`. All other types of input volumes use single-precision arithmetic.

References

[1] Wang, Z., Simoncelli, E.P., Bovik, A.C. *Multiscale Structural Similarity for Image Quality Assessment*. In: The Thirty-Seventh Asilomar Conference on Signals, Systems & Computers, 2003, 1398–1402. Pacific Grove, CA, USA: IEEE, 2003. https://doi.org/10.1109/ACSSC.2003.1292216.

Extended Capabilities

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also
`ssim` | `multissim` | `psnr`

Topics
“Image Quality Metrics”

Introduced in R2020a

multithresh

Multilevel image thresholds using Otsu’s method

Syntax

```
thresh = multithresh(A)
thresh = multithresh(A,N)
[thresh,metric] = multithresh( ___ )
```

Description

`thresh = multithresh(A)` returns the single threshold value `thresh` computed for image `A` using Otsu’s method. You can use `thresh` as an input argument to `imquantize` to convert an image into a two-level image.

`thresh = multithresh(A,N)` returns `thresh` a 1-by-N vector containing `N` threshold values using Otsu’s method. You can use `thresh` as an input argument to `imquantize` to convert image `A` into an image with `N+1` discrete levels.

`[thresh,metric] = multithresh( ___ )` returns `metric`, a measure of the effectiveness of the computed thresholds.

Examples

Segment Image Into Two Regions

Read image and display it.

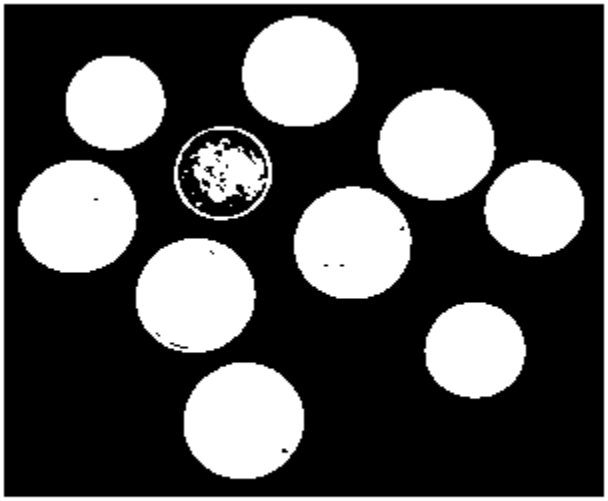
```
I = imread('coins.png');
imshow(I)
```

Calculate a single threshold value for the image.

```
level = multithresh(I);
```

Segment the image into two regions using `imquantize` , specifying the threshold level returned by `multithresh` .

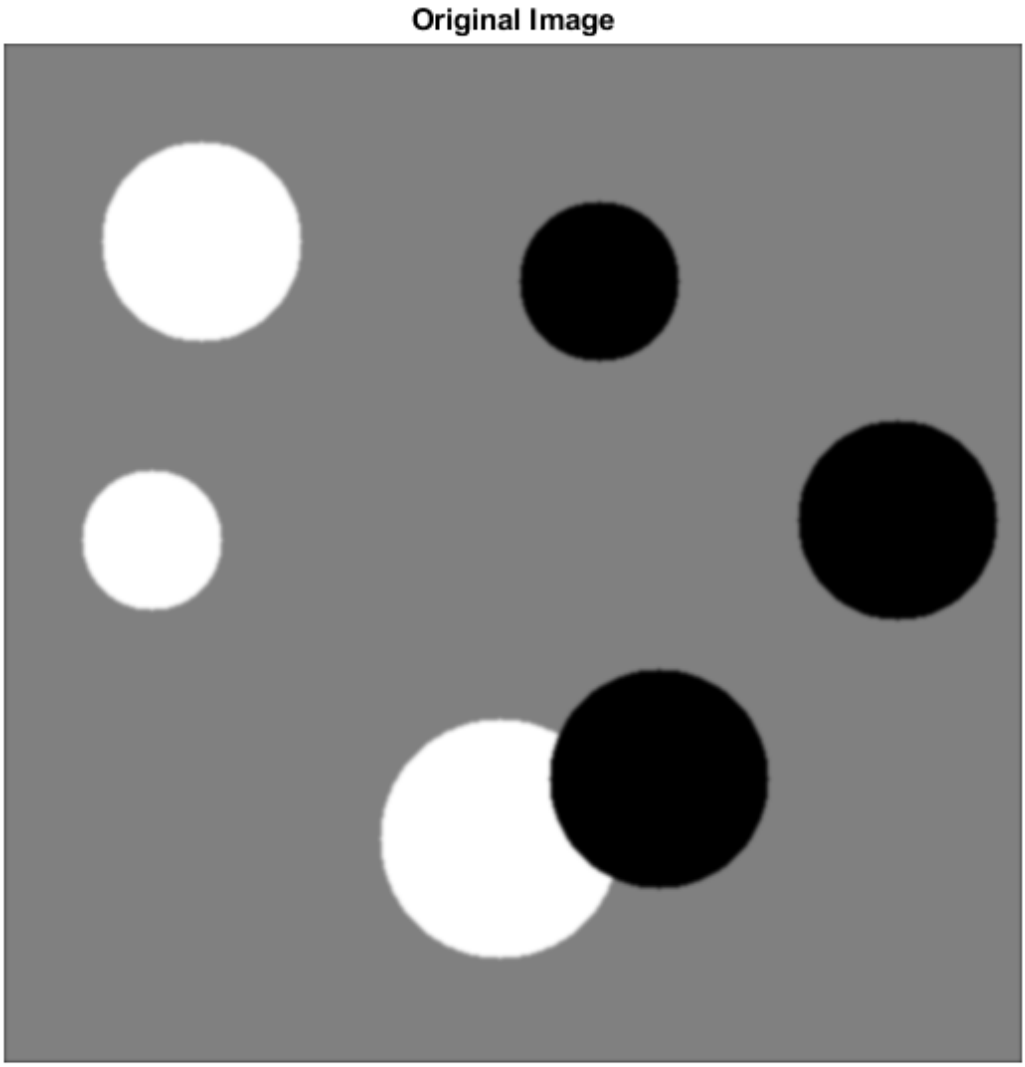
```
seg_I = imquantize(I,level);
figure
imshow(seg_I,[])
```



Segment Image into Three Levels Using Two Thresholds

Read image and display it.

```
I = imread('circlesBrightDark.png');
imshow(I)
axis off
title('Original Image')
```



Calculate two threshold levels.

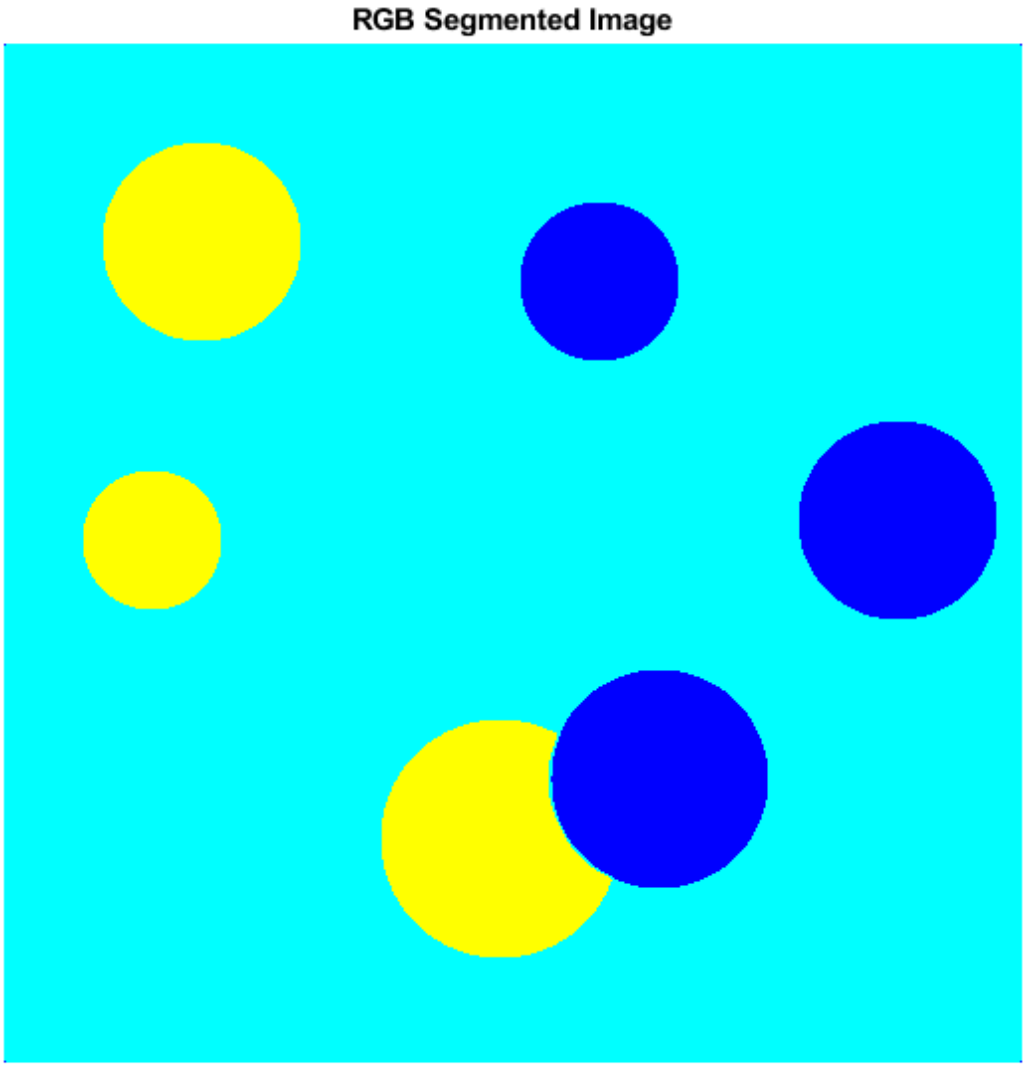
```
thresh = multithresh(I,2);
```

Segment the image into three levels using `imquantize` .

```
seg_I = imquantize(I,thresh);
```

Convert segmented image into color image using `label2rgb` and display it.

```
RGB = label2rgb(seg_I);
figure;
imshow(RGB)
axis off
title('RGB Segmented Image')
```



Compare Thresholding Entire Image Versus Plane-by-Plane Thresholding

Read truecolor (RGB) image and display it.

```
I = imread('peppers.png');
imshow(I)
axis off
title('RGB Image');
```

Generate thresholds for seven levels from the entire RGB image.

```
threshRGB = multithresh(I,7);
```

Generate thresholds for each plane of the RGB image.

```
threshForPlanes = zeros(3,7);

for i = 1:3
    threshForPlanes(i,:) = multithresh(I(:,:,i),7);
end
```

Process the entire image with the set of threshold values computed from entire image.

```
value = [0 threshRGB(2:end) 255];
quantRGB = imquantize(I, threshRGB, value);
```

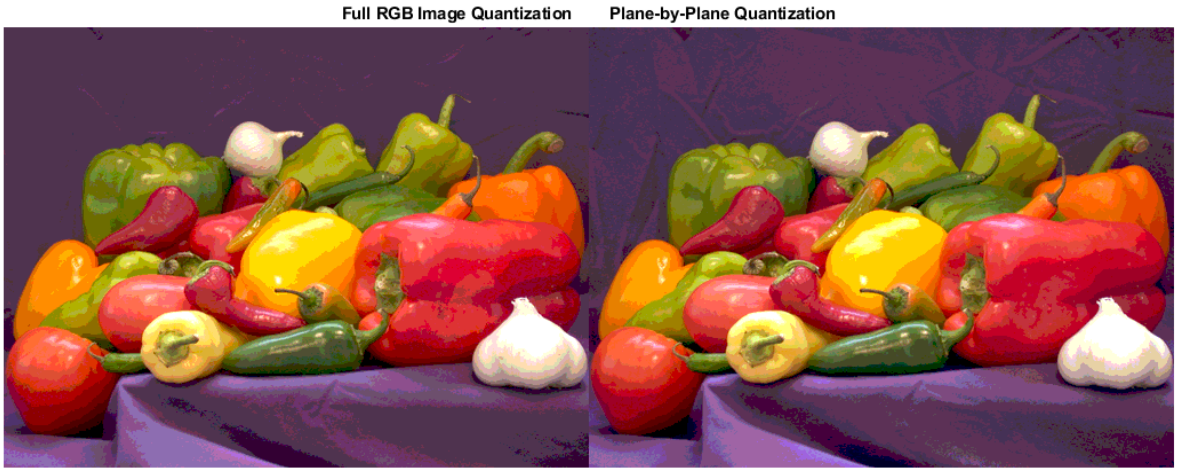
Process each RGB plane separately using the threshold vector computed from the given plane. Quantize each RGB plane using threshold vector generated for that plane.

```
quantPlane = zeros( size(I) );

for i = 1:3
    value = [0 threshForPlanes(i,2:end) 255];
```

```
    quantPlane(:,:,i) = imquantize(I(:,:,i),threshForPlanes(i,:),value);
end

quantPlane = uint8(quantPlane);
```

Display both posterized images and note the visual differences in the two thresholding schemes.

```
imshowpair(quantRGB,quantPlane,'montage')
axis off
title('Full RGB Image Quantization        Plane-by-Plane Quantization')
```



To compare the results, calculate the number of unique RGB pixel vectors in each output image. Note that the plane-by-plane thresholding scheme yields about 23% more colors than the full RGB image scheme.

```
dim = size( quantRGB );
quantRGBmx3   = reshape(quantRGB,   prod(dim(1:2)), 3);
quantPlanemx3 = reshape(quantPlane, prod(dim(1:2)), 3);

colorsRGB   = unique(quantRGBmx3,   'rows' );
colorsPlane = unique(quantPlanemx3, 'rows' );

disp(['Unique colors in RGB image            : ' int2str(length(colorsRGB))]);
```

```
Unique colors in RGB image            : 188
```

```
disp(['Unique colors in Plane-by-Plane image : ' int2str(length(colorsPlane))]);
```

```
Unique colors in Plane-by-Plane image : 231
```

Check Results Using the Metric Output Argument

Read image.

```
I = imread('circlesBrightDark.png');
```

Find all unique grayscale values in image.

```
uniqLevels = unique(I(:));

disp(['Number of unique levels = ' int2str( length(uniqLevels) )]);
```

```
Number of unique levels = 148
```

Compute a series of thresholds at monotonically increasing values of N.

```
Nvals = [1 2 4 8];
for i = 1:length(Nvals)
    [thresh, metric] = multithresh(I, Nvals(i) );
    disp(['N = ' int2str(Nvals(i)) '  |  metric = ' num2str(metric)]);
end
```

```
N = 1  |  metric = 0.54767
N = 2  |  metric = 0.98715
N = 4  |  metric = 0.99648
N = 8  |  metric = 0.99902
```

Apply the set of 8 threshold values to obtain a 9-level segmentation using `imquantize` .

```
seg_Neq8 = imquantize(I,thresh);
uniqLevels = unique( seg_Neq8(:) )
```

```
uniqLevels = 9×1

     1
     2
     3
     4
     5
     6
     7
     8
     9
```

Threshold the image using `seg_Neq8` as an input to `multithresh`. Set N equal to 8, which is 1 less than the number of levels in this segmented image. `multithresh` returns a `metric` value of 1.

```
[thresh, metric] = multithresh(seg_Neq8,8)
```

```
thresh = 1×8

    1.8784    2.7882    3.6667    4.5451    5.4549    6.3333    7.2118    8.1216


metric = 1
```

Threshold the image again, this time increasing the value of N by 1. This value now equals the number of levels in the image. Note how the input is degenerate because the number of levels in the image is too few for the number of requested thresholds. Hence, `multithresh` returns a `metric` value of 0.

```
[thresh, metric] = multithresh(seg_Neq8,9)
```

```
Warning: No solution exists because the number of unique levels in the image are too few to find
```

```
thresh = 1×9

     1     2     3     4     5     6     7     8     9

metric = 0
```

Input Arguments

`A` — Image to be thresholded

numeric array

Image to be thresholded, specified as a numeric array of any dimension. `multithresh` finds the thresholds based on the aggregate histogram of the entire array. `multithresh` considers an RGB image as a 3-D numeric array and computes the thresholds for the combined data from all three color planes.

`multithresh` uses the range of the input image `A`, `[min(A(:)) max(A(:))]`, as the limits for computing the histogram used in subsequent computations. `multithresh` ignores any `NaNs` in computation. Any `Infs` and `-Infs` are counted in the first and last bin of the histogram, respectively.

For degenerate inputs where the number of unique values in `A` is less than or equal to `N`, there is no viable solution using Otsu's method. For such inputs, the return value `thresh` contains all the unique values from `A` and possibly some extra values that are chosen arbitrarily.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`N` — Number of threshold values

1 (default) | positive integer

Number of threshold values, specified as a positive integer. For `N > 2`, `multithresh` uses search-based optimization of Otsu's criterion to find the thresholds. The search-based optimization guarantees only locally optimal results. Since the chance of converging to local optimum increases with `N`, it is preferable to use smaller values of `N`, typically `N < 10`. The maximum allowed value for `N` is 20.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`thresh` — Set of threshold values

1-by-N numeric vector

Set of threshold values used to quantize an image, returned as a 1-by-N numeric vector, whose data type is the same as image `A`.

These thresholds are in the same range as the input image `A`, unlike the `graythresh` function, which returns a normalized threshold in the range [0, 1].

`metric` — Measure of effectiveness

number in the range [0, 1]

Measure of the effectiveness of the thresholds, returned as a number in the range [0, 1]. Higher values indicates greater effectiveness of the thresholds in separating the input image into `N+1` classes

based on Otsu's objective criterion. For degenerate inputs where the number of unique values in `A` is less than or equal to `N`, `metric` equals 0.

Data Types: `double`

References

[1] Otsu, N., "A Threshold Selection Method from Gray-Level Histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 9, No. 1, 1979, pp. 62-66.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `multithresh` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `multithresh` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- The input argument `N` must be a compile-time constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- The input argument `N` must be a compile-time constant.

See Also

`graythresh` | `imquantize` | `im2bw` | `rgb2ind`

Introduced in R2012b

niftiinfo

Read metadata from NIfTI file

Syntax

```
info = niftiinfo(filename)
```

Description

`info = niftiinfo(filename)` returns metadata, `info`, from the Neuroimaging Informatics Technology Initiative (NIfTI) file specified by `filename`. The `niftiinfo` function supports both the `NIfTI1` and `NIfTI2` file formats.

Examples

View Metadata Fields from NIfTI Header File

Load metadata from the NIfTI file `brain.nii`.

```
info = niftiinfo('brain.nii');
```

Display the version of the file.

```
info.Version
```

```
ans =
'NIfTI1'
```

Display the pixel dimensions of the file.

```
info.PixelDimensions
```

```
ans = 1×3

     1     1     1
```

Display the raw header content.

```
info.raw
```

```
ans = struct with fields:
        sizeof_hdr: 348
          dim_info: ' '
               dim: [3 256 256 21 1 1 1 1]
         intent_p1: 0
         intent_p2: 0
         intent_p3: 0
       intent_code: 0
          datatype: 2
            bitpix: 8
       slice_start: 0
```

```
           pixdim: [1 1 1 1 0 0 0 0]
       vox_offset: 352
        scl_slope: 0
        scl_inter: 0
        slice_end: 0
       slice_code: 0
       xyzt_units: 0
          cal_max: 0
          cal_min: 0
    slice_duration: 0
          toffset: 0
          descrip: ''
         aux_file: ''
       qform_code: 0
       sform_code: 0
        quatern_b: 0
        quatern_c: 0
        quatern_d: 0
        qoffset_x: 0
        qoffset_y: 0
        qoffset_z: 0
           srow_x: [0 0 0 0]
           srow_y: [0 0 0 0]
           srow_z: [0 0 0 0]
      intent_name: ''
            magic: 'n+1 '
```

Display the intent code from the raw structure.

```
info.raw.intent_code
```

```
ans = 0
```

Input Arguments

`filename` — Name of NIfTI file

character vector | string scalar

Name of NIfTI file, specified as a string scalar or a character vector.

- If you do not specify a file extension, then `niftiinfo` looks for a file with the extension `.nii` (or `.nii.gz` if the file is compressed).
- If `niftiinfo` cannot find a file with the `.nii` or `.nii.gz` extension, then it looks for a file with the file extension `.hdr` (or `.hdr.gz` if the file is compressed). In the dual-file NIfTI format, the `.hdr` file holds the metadata associated with the volume.

Data Types: `char` | `string`

Output Arguments

`info` — Metadata associated with a NIfTI volume

structure

Metadata associated with a NIfTI volume, returned as a structure.

`niftiinfo` returns the metadata from the header in simplified form. The function renames, reorders, and packages fields into easier to read MATLAB structures. For example, `niftiinfo` creates the `DisplayIntensityRange` field from the `cal_max` and `cal_min` fields of the file metadata. To view the metadata as it appears in the file, see the `raw` field of the structure returned.

References

[1] Cox, R. W., J. Ashburner, H. Breman, K. Fissell, C. Haselgrove, C. J. Holmes, J. L. Lancaster, D. E. Rex, S. M. Smith, J. B. Woodward, and S. C. Strother. "A (sort of) new image data format standard: NiFTI-1." 10th Annual Meeting of Organisation of Human Brain Mapping, Budapest, Hungary, June 2004.

See Also

`niftiread` | `niftiwrite`

Introduced in R2017b

niftiread

Read NIfTI image

Syntax

```
V = niftiread(filename)
V = niftiread(headerfile,imgfile)
V = niftiread(info)
```

Description

`V = niftiread(filename)` reads the NIfTI image file specified by `filename` in the current folder or on the path, and returns volumetric data in `V`. The `niftiread` function supports both the `NIfTI1` and `NIfTI2` file formats.

`V = niftiread(headerfile,imgfile)` reads a NIfTI header file (`.hdr`) and image file (`.img`) pair.

`V = niftiread(info)` reads a NIfTI file described by the metadata structure `info`. To create an `info` structure, use the `niftiinfo` function

Examples

Load Volume from NIfTI File Using File Name

Load volumetric data from a NIfTI file. The file uses the NIfTI combined format—the image and metadata are in the same file. This type of NIfTI file has the `.nii` file extension.

```
V = niftiread('brain.nii');
```

View the variable in the workspace.

```
whos V
```

```
  Name        Size                 Bytes  Class    Attributes

  V         256x256x21            1376256  uint8
```

Load Volume from NIfTI File Using Its Header Structure

Read the metadata from a NIfTI file.

```
info = niftiinfo('brain.nii');
```

Read the volumetric image using the metadata structure returned by `niftiinfo`.

```
V = niftiread(info);
```

View the variable in the workspace.

```
whos V
```

```
  Name        Size                 Bytes  Class    Attributes

  V         256x256x21            1376256  uint8
```

Input Arguments

`filename` — Name of NIfTI file

string scalar | character vector

Name of the NIfTI file, specified as a string scalar or character vector. The file can be in the `NIfTI1` or `NIfTI2` file format.

- If you do not specify a file extension, then `niftiread` looks for a file with the `.nii` extension.
- If `niftiread` cannot find a file with the `.nii` extension, then it looks for a gzipped version of the file, with extension `.nii.gz`.
- If `niftiread` cannot find a file with the `.nii.gz` extension, then it looks for a file with the `.hdr`, `.hdr.gz`, `.img`, or `.img.gz` file extension.
- If `niftiread` cannot find a file that matches any of these options, then it returns an error.

Data Types: `char` | `string`

`headerfile` — Name of file containing metadata

string scalar | character vector

Name of the file containing metadata, specified as a string scalar or a character vector. The NIfTI header file (`.hdr`) holds the metadata associated with a NIfTI volume. If you do not specify a corresponding `imgfile`, then `niftiread` looks in the same folder for a file with the same name and extension `.img`.

Data Types: `char` | `string`

`imgfile` — Name of file containing volume

string scalar | character vector

Name of the file containing volume, specified as a string scalar or a character vector. The NIfTI image file (`.img`) holds the volume data. If you do not specify a corresponding header file, then `niftiread` looks in the same folder for a file with the same name and extension `.hdr`.

Data Types: `char` | `string`

`info` — NIfTI file metadata

structure

NIfTI file metadata, specified as a structure returned by `niftiinfo`.

Data Types: `struct`

Output Arguments

`V` — Volumetric data

numeric array

Volumetric data, returned as a numeric array.

More About

NIfTI File Format

NIfTI (Neuroimaging Informatics Technology Initiative) is an NIH-sponsored working group to promote the interoperability of functional neuroimaging software tools. NIfTI uses a single or dual file storage format.

- The dual file format stores data in a pair of files: a header file (`.hdr`) containing the metadata and a data file (`.img`) containing image data.
- The single file format stores the data in a single file (`.nii`), which contains header information followed by image data.

References

[1] Cox, R. W., J. Ashburner, H. Breman, K. Fissell, C. Haselgrove, C. J. Holmes, J. L. Lancaster, D. E. Rex, S. M. Smith, J. B. Woodward, and S. C. Strother. "A (sort of) new image data format standard: NiFTI-1." 10th Annual Meeting of Organisation of Human Brain Mapping, Budapest, Hungary, June 2004.

See Also

`niftiwrite` | `niftiinfo`

Introduced in R2017b

niftiwrite

Write volume to file using NIfTI format

Syntax

```
niftiwrite(V,filename)
niftiwrite(V,filename,info)
niftiwrite(V,filename,info,Name,Value)
```

Description

`niftiwrite(V,filename)` writes the volumetric image data `V` to a file by using the Neuroimaging Informatics Technology Initiative (NIfTI) format. By default, `niftiwrite` creates a combined NIfTI file that contains both metadata and volumetric data. `niftiwrite` names the file `filename`, adding the `.nii` file extension. `niftiwrite` populates the metadata using appropriate default values and volume properties, such as size and data type.

`niftiwrite` supports both the `NIfTI1` and `NIfTI2` file formats. `NIfTI1` is the default file format. To write NifTI data in the `NIfTI2` format, use the syntax with `Name,Value` pair arguments. Specify the `Version` argument as `'NIfTI2'`.

`niftiwrite(V,filename,info)` writes the volumetric data `V` to a file, including the file metadata from `info`. If the metadata does not match the image contents and size, then `niftiwrite` returns an error.

`niftiwrite(V,filename,info,Name,Value)` writes the volumetric data to a file, using options specified in `Name,Value` pairs.

Examples

Write Median-Filtered Volume to NIfTI File

Load a NIfTI image by using its `.nii` file name.

```
V = niftiread('brain.nii');
```

Filter the image in 3-D by using a 3-by-3 median filter.

```
V = medfilt3(V);
```

Write the filtered image to a `.nii` file, using default header values.

```
niftiwrite(V,'outbrain.nii');
```

Write Data to NIfTI File and Modify Header Structure

Read the metadata from a NIfTI file by using its `.nii` file name.

```
info = niftiinfo('brain.nii');
```

Read volumetric data from the file by using the file metadata.

```
V = niftiread(info);
```

Edit the `Description` metadata field of the file.

```
info.Description = 'Modified using MATLAB R2017b';
```

Write the volumetric data with the modified metadata to a new `.nii` file.

```
niftiwrite(V,'outbrain.nii',info);
```

Input Arguments

`filename` — Name of NIfTI file

character vector | string scalar

Name of NIfTI file, specified as a string scalar or character vector. By default, `niftiwrite` creates a combined format file that contains both metadata and image data and has the file extension `.nii`. If you specify the `'Compressed'` name-value pair, `niftiwrite` adds the file extension `.nii.gz`. If you set the `'Combined'` name-value pair to `false`, then `niftiwrite` creates two files with the same name and different file extensions. One file contains the metadata associated with the volume and has the file extension `.hdr`. The other file contains image data and has the file extension `.img`.

Data Types: `char` | `string`

`V` — Volumetric data

numeric array

Volumetric data, specified as a numeric array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`info` — File metadata

structure

File metadata, specified as a structure returned by the `niftiinfo` function.

Data Types: `struct`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `niftiwrite(V,'outbrain.nii','Compressed',true)`

`Combined` — Type of NIfTI file to create

`true` (default) | `false`

Type of NIfTI file to create, specified as `true` or `false`. If `true` (the default), `niftiwrite` creates a single file with the file extension `.nii`. If `false`, `niftiwrite` creates a pair of files with the same name but with different file extensions: `.hdr` for the file containing metadata, and `.img` for the file containing the volumetric data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`Compressed` — Compress image data
`false` (default) | `true`

Compress image data, specified as `true` or `false`. If `'Compressed'` is `true`, then `niftiwrite` generates compressed files, using gzip, with the file name extension `.gz`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`Endian` — Endianness of the data
`'little'` (default) | `'big'`

Endianness of the data, specified as `'little'`, to indicate little-endian format (default) or `'big'`, to indicate big-endian format.

Data Types: `char` | `string`

`Version` — `NIfTI` data format
`'NIfTI1'` | `'NIfTI2'`

`NIfTI` data format, specified as `'NIfTI1'` or `'NIfTI2'`.

- If specified as `'NIfTI1'`, then `niftiwrite` writes the input according to `NIfTI1` data format.
- If specified as `'NIfTI2'`, then `niftiwrite` writes the input according to `NIfTI2` data format.
- If not specified, then the default value for `'Version'` is chosen based on the maximum dimension of the input volumetric data.
  - If the maximum dimension of the input is less than or equal to 32767, then the default value is `NIfTI1`.
  - If the maximum dimension of the input is greater than 32767, then the default value is `NIfTI2`.

Data Types: `char` | `string`

References

[1] Cox, R. W., J. Ashburner, H. Breman, K. Fissell, C. Haselgrove, C. J. Holmes, J. L. Lancaster, D. E. Rex, S. M. Smith, J. B. Woodward, and S. C. Strother. "A (sort of) new image data format standard: NiFTI-1." 10th Annual Meeting of Organisation of Human Brain Mapping, Budapest, Hungary, June 2004.

See Also
`niftiread` | `niftiinfo`

Introduced in R2017b

niqe

Naturalness Image Quality Evaluator (NIQE) no-reference image quality score

Syntax

```
score = niqe(A)
score = niqe(A,model)
```

Description

`score = niqe(A)` calculates the no-reference image quality score for image `A` using the Naturalness Image Quality Evaluator (NIQE). `niqe` compares `A` to a default model computed from images of natural scenes. A smaller score indicates better perceptual quality.

`score = niqe(A,model)` calculates the image quality score using a custom model.

Examples

Calculate NIQE Score Using Default Feature Model

Compute the NIQE score for a natural image and its distorted versions using the default model.

Read an image into the workspace. Create copies of the image with noise and blurring distortions.

```
I = imread('lighthouse.png');
Inoise = imnoise(I,'salt & pepper',0.02);
Iblur = imgaussfilt(I,2);
```

Display the images.

```
figure
montage({I,Inoise,Iblur},'Size',[1 3])
title('Original Image | Noisy Image | Blurry Image')
```

Calculate the NIQE score for each image using the default model. Display the score.

```
niqeI = niqe(I);
fprintf('NIQE score for original image is %0.4f.\n',niqeI)
```

```
NIQE score for original image is 2.5455.
```

```
niqeInoise = niqe(Inoise);
fprintf('NIQE score for noisy image is %0.4f.\n',niqeInoise)
```

```
NIQE score for noisy image is 10.8770.
```

```
niqeIblur = niqe(Iblur);
fprintf('NIQE score for blurry image is %0.4f.\n',niqeIblur)
```

```
NIQE score for blurry image is 5.2661.
```

The original undistorted image has the best perceptual quality and therefore the lowest NIQE score.

Calculate NIQE Score Using Custom Feature Model

Load a set of natural images into an image datastore. These images are shipped in Image Processing Toolbox™ in a directory named 'imdata'.

```
setDir = fullfile(toolboxdir('images'),'imdata');
imds = imageDatastore(setDir,'FileExtensions',{'.jpg'});
```

Train a custom NIQE model using the image datastore.

```
model = fitniqe(imds);
```

```
Extracting features from 38 images.
....
```

```
Completed 12 of 38 images.  Time: Calculating...
....
Completed 24 of 38 images.  Time: 00:21 of 00:32
...
Done.
```

Read an image of a natural scene. Display the image.

```
I = imread('car1.jpg');
imshow(I)
```

Calculate the NIQE score for the image using the custom model. Display the score.

```
niqeI = niqe(I,model);
fprintf('NIQE score for the image is %0.4f.\n',niqeI)
```

```
NIQE score for the image is 1.8728.
```

Input Arguments

A — Input image

2-D grayscale image | 2-D RGB image

Input image, specified as a 2-D grayscale or RGB image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`model` — Custom model
`niqeModel` object

Custom model of image features, specified as a `niqeModel` object. `model` is derived from natural scene statistics.

Output Arguments

`score` — No-reference image quality score
nonnegative scalar

No-reference image quality score, returned as a nonnegative scalar. Lower values of `score` reflect better perceptual quality of image `A` with respect to the input `model`.

Data Types: `double`

Algorithms

NIQE measures the distance between the NSS-based features calculated from image `A` to the features obtained from an image database used to train the model. The features are modeled as multidimensional Gaussian distributions.

References

[1] Mittal, A., R. Soundararajan, and A. C. Bovik. "Making a Completely Blind Image Quality Analyzer." *IEEE Signal Processing Letters*. Vol. 22, Number 3, March 2013, pp. 209–212.

See Also

Functions
`brisque` | `fitbrisque` | `fitniqe` | `piqe`

Objects
`niqeModel`

Topics
“Image Quality Metrics”

Introduced in R2017b

niqeModel

Naturalness Image Quality Evaluator (NIQE) model

Description

A `niqeModel` object encapsulates a model used to calculate the Naturalness Image Quality Evaluator (NIQE) perceptual quality score of an image.

Creation

You can create a `niqeModel` object using the following methods:

- `fitniqe` — Train a NIQE model with parameters derived from your image datastore. Use this function if you do not have a pretrained model.
- The `niqeModel` function described here. Use this function if you have a pretrained NIQE model, or if the default model is sufficient for your application.

Syntax

```
m = niqeModel
m = niqeModel(mean,covariance,blockSize,sharpnessThreshold)
```

Description

`m = niqeModel` creates a NIQE model object with default property values that are derived from the pristine image database noted in [1].

`m = niqeModel(mean,covariance,blockSize,sharpnessThreshold)` creates a custom NIQE model and sets the `Mean`, `Covariance`, `BlockSize`, and `SharpnessThreshold` properties. You must provide all four arguments to create a custom model.

Properties

`Mean` — Mean of natural scene statistics (NSS) based image feature vectors

36-element numeric row vector

Mean of natural scene statistics (NSS) based image feature vectors, specified as a 36-element numeric row vector.

Example: `rand(1,36)`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`Covariance` — Covariance matrix of NSS-based image feature vectors

36-by-36 numeric matrix

Covariance matrix of NSS-based image feature vectors, specified as a 36-by-36 numeric matrix.

Example: `rand(36,36)`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`BlockSize` — Block size used to partition an image
`[96 96]` (default) | 2-element row vector of positive even integers

Block size used to partition an image into nonoverlapping blocks, specified as a 2-element row vector of positive even integers. The two elements specify the number of rows and columns in each partition, respectively.

Example: `[10 10]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`SharpnessThreshold` — Sharpness threshold used to calculate feature vectors
`0` (default) | real scalar in the range [0, 1]

Sharpness threshold used to calculate feature vectors, specified as a real scalar in the range [0, 1]. The threshold determines which blocks are selected to calculate the feature vectors.

Example: `0.25`

Data Types: `single` | `double`

Examples

Create NIQE Model Object with Default Properties

```
model = niqeModel
```

```
model =
  niqeModel with properties:

                 Mean: [2.3167 0.7556 0.7429 0.0746 0.0951 0.1466 ... ]
           Covariance: [36x36 double]
            BlockSize: [96 96]
    SharpnessThreshold: 0
```

Create NIQE Model Object with Custom Properties

Create a `niqeModel` object using precomputed `Mean`, `Covariance`, `BlockSize`, and `SharpnessThreshold` properties. Random initializations are shown for illustrative purposes only.

```
model = niqeModel(rand(1,36),rand(36,36),[10 10],0.25);
```

You can use the custom model to calculate the NIQE score for an image.

```
I = imread('lighthouse.png');
score = niqe(I,model)
```

```
score = 3.6866
```

References

[1] Mittal, A., R. Soundararajan, and A. C. Bovik. "Making a Completely Blind Image Quality Analyzer." *IEEE Signal Processing Letters*. Vol. 22, Number 3, March 2013, pp. 209–212.

See Also

Functions
`niqe` | `fitniqe`

Objects
`brisqueModel`

Topics
“Image Quality Metrics”
“Train and Use No-Reference Quality Assessment Model”

Introduced in R2017b

nitfinfo

Read metadata from National Imagery Transmission Format (NITF) file

Syntax

```
info = nitfinfo(filename)
```

Description

`info = nitfinfo(filename)` returns the file-level metadata about the images, annotations, and graphics in a National Imagery Transmission Format (NITF) file specified by `filename`. NITF is an image format used by the U.S. government and military for transmitting documents. A NITF file can contain multiple images and include text and graphic layers.

Input Arguments

`filename` — Name of NITF file

character vector | string scalar

Name of NITF file, specified as a character vector or string scalar.

Data Types: `char` | `string`

Output Arguments

`info` — Metadata of NITF file

struct

Metadata of NITF file, returned as a structure.

Tips

- `nitfinfo` supports version 2.0 and 2.1 NITF files, at all Joint Interoperability Test Command (JITC) compliance levels, as well as the NATO Secondary Image Format (NSIF) 1.0. `nitfinfo` does not support NITF 1.1 files.

See Also

`isnitf` | `nitfread`

Introduced in R2007b

nitfread

Read image from NITF file

Syntax

```
X = nitfread(filename)
X = nitfread(filename,idx)
X = nitfread( ___ ,'PixelRegion',regions)
```

Description

`X = nitfread(filename)` reads the first image from the National Imagery Transmission Format (NITF) file specified by `filename`.

`X = nitfread(filename,idx)` reads the image with index number `idx` from an NITF file that contains multiple images.

`X = nitfread( ___ ,'PixelRegion',regions)` additionally specifies regions of the image to be read from an NITF file, .

Examples

Read Image Data from NITF File

To run this example, replace the name of the file with the name of an NITF file on your system. You can find sample NITF files on the web.

Read the second image from an NITF file containing multiple images. The example reads a subset of the image data starting at (row,column) location (100, 200), reading every other value until location (105, 205).

```
subsec = {[100 2 105],[200 2 205]}

ntfdata = nitfread('your_file.ntf',2,'PixelRegion',subsec);
```

Input Arguments

`filename` — Name of NITF file

character vector

Name of the NITF file, specified as a character vector. The file must be in the current folder or in a folder on the MATLAB path, or `filename` must contain the full path to the file.

Data Types: `char`

`idx` — Index number of image in NITF file

positive integer

Index number of the image in the NITF file, specified as a positive integer.

Data Types: `double`

`regions` — Regions of image to be read

2-column cell array

Regions of image to be read from the NITF file, specified as a 2-column cell array. The first column specifies row indices and the second column specifies column indices of the regions. Each element in the cell array is a 2-element vector of positive integers of the form `[start stop]` or a 3-element vector of positive integers of the form `[start increment stop]`.

Example: `{[100 150],[200 250]}` — read pixels starting at row/column location (100, 200) and ending at location (150, 250)

Example: `{[100 2 150],[200 2 250]}` — read every other pixel starting at row/column location (100, 200) and ending at location (150, 250)

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `cell`

Output Arguments

X — Image data from NITF file

numeric array

Image data from the NITF file, returned as a numeric array.

Tips

- `nitfread` supports version 2.0 and 2.1 NITF files, and NSIF 1.0 files. Image submasks and NITF 1.1 files are not supported.

See Also

`isnitf` | `nitfinfo` | `tiffreadVolume`

Introduced in R2007b

nlfilter

General sliding-neighborhood operations

Syntax

```
B = nlfilter(A,[m n],fun)
B = nlfilter(A,'indexed',___)
```

Description

`B = nlfilter(A,[m n],fun)` applies the function `fun` to each `m`-by-`n` sliding block of the grayscale image `A`.

`B = nlfilter(A,'indexed',___)` processes `A` as an indexed image, padding with `0`s if the class of `A` is `uint8`, `uint16`, or `logical`, and padding with `1`s otherwise.

Note `nlfilter` can take a long time to process large images. In some cases, the `colfilt` function can perform the same operation much faster.

Examples

Apply Median Filter to Image

This example shows how to apply a median filter to an image using `nlfilter`. This example produces the same result as calling `medfilt2` with a 3-by-3 neighborhood.

Read an image into the workspace.

```
A = imread('cameraman.tif');
```

Convert the image to `double`.

```
A = im2double(A);
```

Create the function you want to apply to the image—a median filter.

```
fun = @(x) median(x(:));
```

Apply the filter to the image.

```
B = nlfilter(A,[3 3],fun);
```

Display the original image and the filtered image, side-by-side.

```
montage({A,B})
title('Original Image (Left) and Median Filtered Image (Right)')
```

Input Arguments

`A` — Image to be filtered

numeric array

Image to be filtered, specified as a numeric array of any class supported by `fun`. When `A` is grayscale, it can be any numeric type or `logical`. When `A` is indexed, it can be `logical`, `uint8`, `uint16`, `single`, or `double`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`[m n]` — Block size

2-element vector of positive integers

Block size, specified as a 2-element vector of positive integers. `m` is the number of rows and `n` is the number of columns in the block.

Example: `B = nlfilter(A,[3 3],fun);`

Data Types: `single` | `double` | `logical`

`fun` — Function handle

handle

Function handle specified as a handle. The function must accept an `m`-by-`n` matrix as input and return a scalar result.

```
c = fun(x)
```

`c` is the output value for the center pixel in the `m`-by-`n` block `x`. `nlfilter` calls `fun` for each pixel in `A`. `nlfilter` zero-pads the `m`-by-`n` block at the edges, if necessary.

Data Types: `function_handle`

Output Arguments

`B` — Filtered image

numeric array

Filtered image, returned as numeric array. The class of `B` depends on the class of the output from `fun`.

See Also

`blockproc` | `colfilt`

Topics

“Anonymous Functions”
“Parameterizing Functions”
“Create Function Handle”

Introduced before R2006a

normxcorr2

Normalized 2-D cross-correlation

Syntax

```
C = normxcorr2(template,A)
```

Description

`C = normxcorr2(template,A)` computes the normalized cross-correlation of the matrices `template` and `A`. The resulting matrix `C` contains the correlation coefficients.

Examples

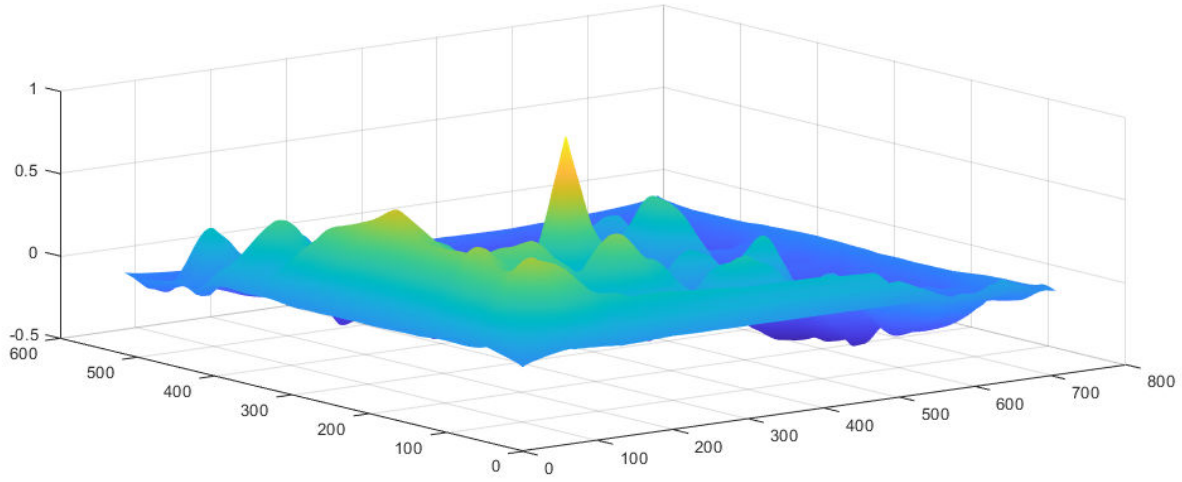
Use Cross-Correlation to Find Template in Image

Read two images into the workspace, and convert them to grayscale for use with `normxcorr2`. Display the images side-by-side.

```
onion = im2gray(imread('onion.png'));
peppers = im2gray(imread('peppers.png'));
montage({peppers,onion})
```

Perform cross-correlation, and display the result as a surface.

```
c = normxcorr2(onion,peppers);
surf(c)
shading flat
```



Find the peak in cross-correlation.

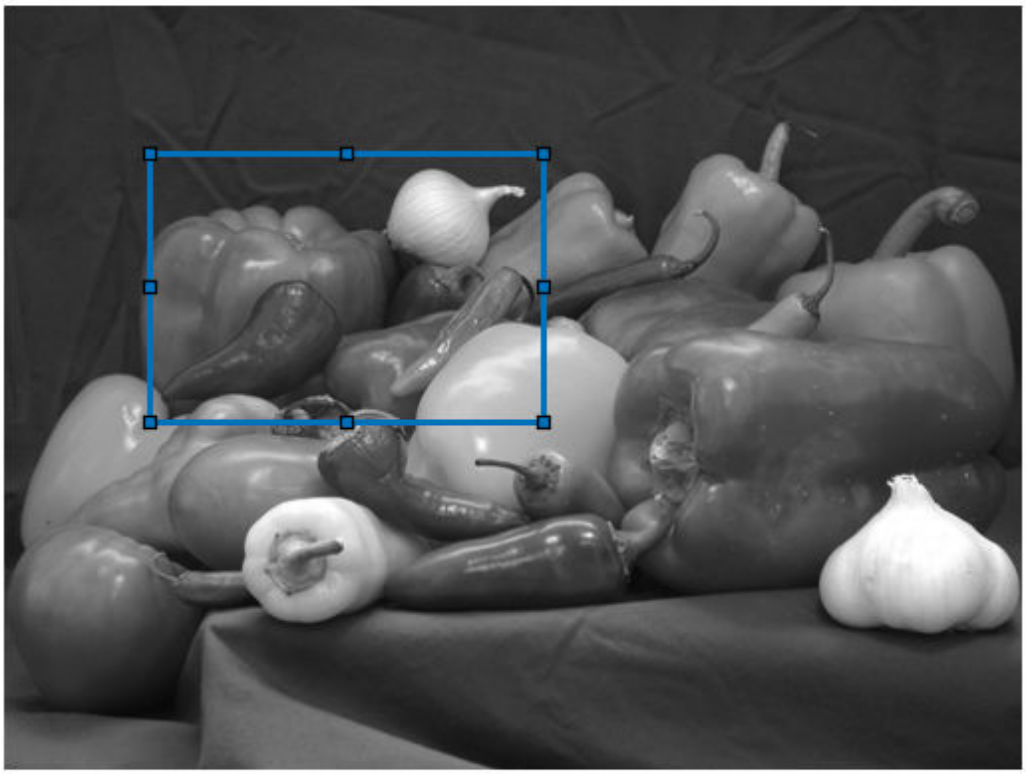
```
[ypeak,xpeak] = find(c==max(c(:)));
```

Account for the padding that `normxcorr2` adds.

```
yoffSet = ypeak-size(onion,1);
xoffSet = xpeak-size(onion,2);
```

Display the matched area by using the `drawrectangle` function. The 'Position' name-value pair argument specifies the upper left coordinate, width, and height of the ROI as the 4-element vector `[xmin,ymin,width,height]`. Specify the face of the ROI as fully transparent.

```
imshow(peppers)
drawrectangle(gca,'Position',[xoffSet,yoffSet,size(onion,2),size(onion,1)], ...
    'FaceAlpha',0);
```



Input Arguments

`template` — Input template

numeric matrix

Input template, specified as a numeric matrix. The values of `template` cannot all be the same.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`A` — Input image

numeric matrix

Input image, specified as a numeric image. `A` must be larger than the matrix `template` for the normalization to be meaningful.

Normalized cross-correlation is an undefined operation in regions where `A` has zero variance over the full extent of the template. In these regions, `normxcorr2` assigns correlation coefficients of zero to the output `C`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

C — Correlation coefficients

numeric matrix

Correlation coefficients, returned as a numeric matrix with values in the range [-1, 1].

Data Types: `double`

Algorithms

`normxcorr2` uses the following general procedure [1], [2]:

1. Calculate cross-correlation in the spatial or the frequency domain, depending on size of images.
2. Calculate local sums by precomputing running sums.
3. Use local sums to normalize the cross-correlation to get correlation coefficients.

The implementation closely follows the formula from [1]:

$$\gamma(u,v) = \frac{\sum_{x,y}\left[f(x,y) - \bar{f}_{u,v}\right]\left[t(x-u,y-v) - \bar{t}\right]}{\left\{\sum_{x,y}\left[f(x,y) - \bar{f}_{u,v}\right]^2 \sum_{x,y}\left[t(x-u,y-v) - \bar{t}\right]^2\right\}^{0.5}}$$

where

- f is the image.
- $\bar{t}$ is the mean of the template
- $\bar{f}_{u,v}$ is the mean of $f(x,y)$ in the region under the template.

References

[1] Lewis, J. P. "Fast Normalized Cross-Correlation." Industrial Light & Magic, 1995. http://scribblethink.org/Work/nvisionInterface/nip.pdf.

[2] Haralick, Robert M., and Linda G. Shapiro, *Computer and Robot Vision*, Volume II, Addison-Wesley, 1992, pp. 316-317.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`normxcorr2` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`corrcoef` | `corr2`

Introduced before R2006a

ntsc2rgb

Convert NTSC values to RGB color space

Syntax

```
RGB = ntsc2rgb(YIQ)
```

Description

`RGB = ntsc2rgb(YIQ)` converts the luma (*Y*) and chrominance (*I* and *Q*) values of an NTSC image to red, green, and blue values of an RGB image.

Examples

Convert Image from YIQ to RGB

This example shows how to convert an image from RGB to NTSC color space and back.

Read an RGB image into the workspace.

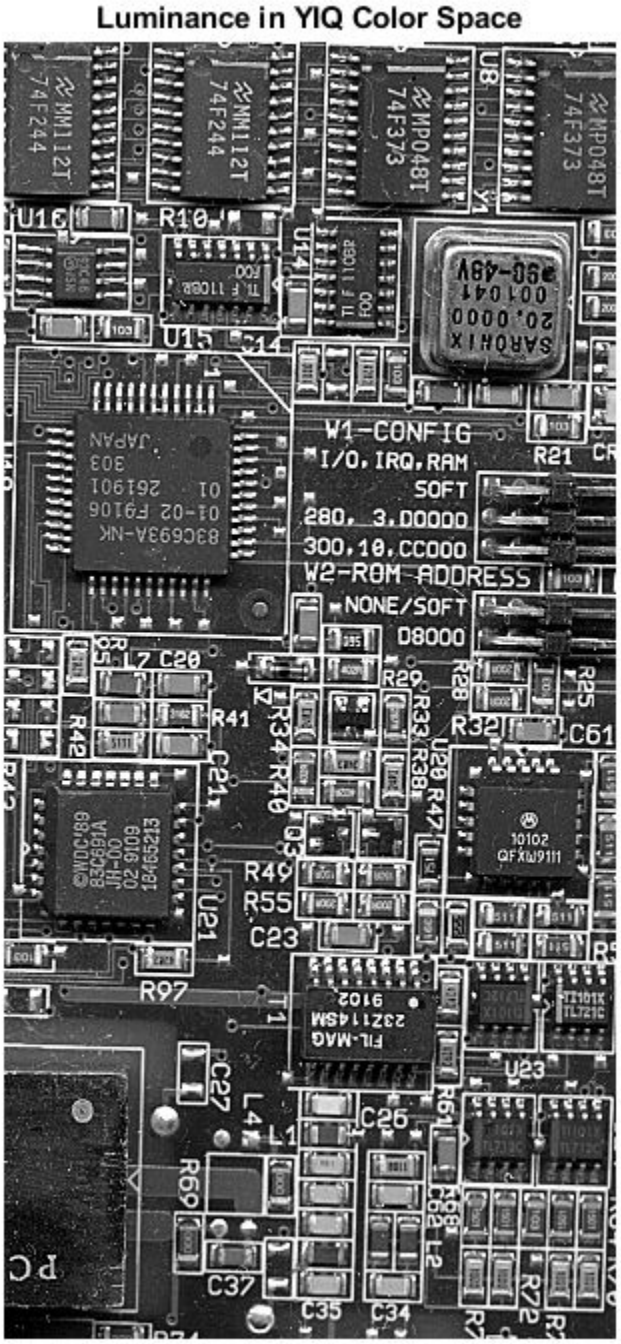
```
RGB = imread('board.tif');
```

Convert the image to YIQ color space.

```
YIQ = rgb2ntsc(RGB);
```

Display the NTSC luminance, represented by the first color channel in the YIQ image.

```
imshow(YIQ(:,:,1))
title('Luminance in YIQ Color Space')
```

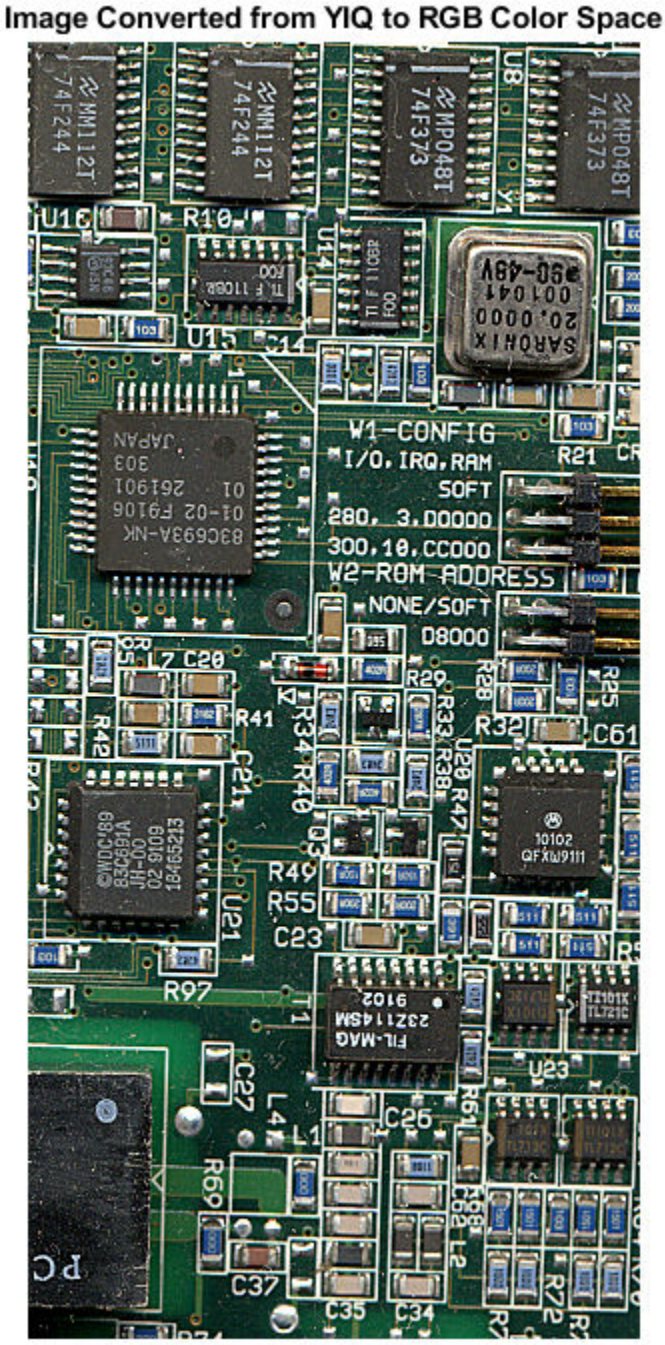


Convert the YIQ image back to RGB color space.

```
RGB2 = ntsc2rgb(YIQ);
```

Display the image that was converted from YIQ to RGB color space.

```
figure
imshow(RGB2)
title('Image Converted from YIQ to RGB Color Space')
```



Input Arguments

`YIQ` — YIQ color values

numeric array

YIQ color values to convert, specified as a numeric array in one of these formats.

- *c*-by-3 colormap. Each row specifies one YIQ color value. Values should be in the range [0, 1] with data type `double`.
- *m*-by-*n*-by-3 image. Values can be data type `single`, `double`, `uint8`, `uint16`, or `int16`.

Attribute	Description
Y	Luma, or brightness of the image. Values are in the range [0, 1], where 0 specifies black and 1 specifies white. Colors increase in brightness as *Y* increases.
I	*In-phase*, which is approximately the amount of blue or orange tones in the image. *I* in the range [-0.5959, 0.5959], where negative numbers indicate blue tones and positive numbers indicate orange tones. As the magnitude of *I* increases, the saturation of the color increases.
Q	*Quadrature*, which is approximately the amount of green or purple tones in the image. *Q* in the range [-0.5229, 0.5229], where negative numbers indicate green tones and positive numbers indicate purple tones. As the magnitude of *Q* increases, the saturation of the color increases.

Data Types: `single` | `double` | `uint8` | `uint16` | `int16`

Output Arguments

`RGB` — Converted RGB color values

numeric array

Converted RGB color values, returned as a numeric array of the same size as the input. Values are in the range [0, 1]. The output data type is `double` unless the input data type is `single`, in which case the output data type is also `single`.

Data Types: `double` | `single`

Algorithms

`ntsc2rgb` computes the RGB values from the NTSC components using

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.956 & 0.621 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}.$$

See Also

`rgb2ntsc` | `hsv2rgb` | `lab2rgb` | `xyz2rgb` | `ycbcr2rgb`

Topics

“Understanding Color Spaces and Color Space Conversion”

Introduced before R2006a

obliqueslice

Extract oblique slice from 3-D volumetric data

Syntax

```
B = obliqueslice(V,point,normal)
B = obliqueslice( ___ ,Name,Value)
[B,x,y,z] = obliqueslice( ___ )
```

Description

`B = obliqueslice(V,point,normal)` extracts a 2-D oblique slice from a 3-D volumetric data `V`. The slice is extracted with reference to a given point on the volume and a normal vector. The slicing plane is perpendicular to the normal vector and passes through the specified point.

For information about how the slice is extracted with respect to the given point and the normal, see “Oblique Slicing” on page 1-2473.

`B = obliqueslice( ___ ,Name,Value)` specifies options using one or more name-value arguments in addition to the input arguments in the previous syntax.

`[B,x,y,z] = obliqueslice( ___ )` also returns the 3-D Cartesian coordinates of the extracted slice in the input volume. For information about how the intensity values at these 3-D coordinates are mapped to 2-D plane, see “Mapping Values from 3-D Coordinate Space to Image Plane” on page 1-2473.

Examples

Extract Oblique Slice from 3-D Volumetric Data

Load a 3-D volumetric data set into the workspace.

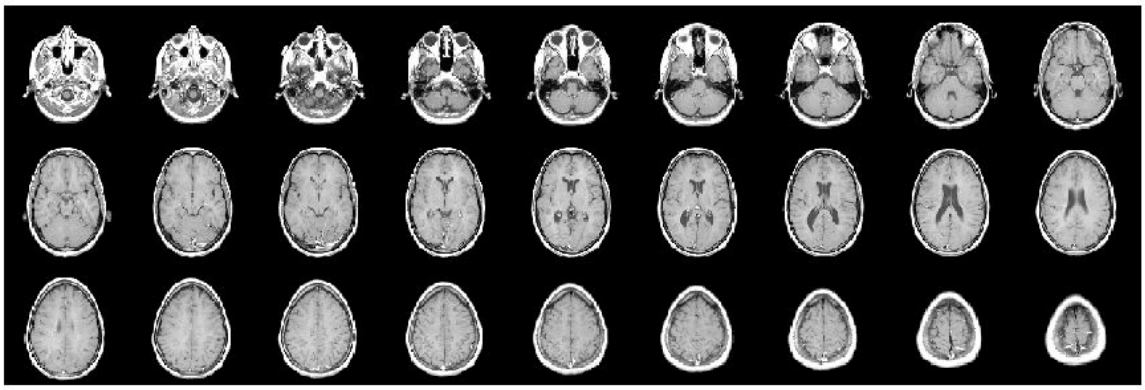
```
load mri
```

Remove singleton dimensions by using the `squeeze` function.

```
V = squeeze(D);
```

Display horizontal slices of the data by using the `montage` function.

```
montage(V,map,'Size',[3 9]);
```



Specify a point in the volume for the slice to pass through.

```
point = [73 50 15.5];
```

Specify a normal vector in 3-D coordinate space.

```
normal = [0 15 20];
```

Extract a slice from the volumetric data. The slice is perpendicular to the normal vector and passes through the specified point.

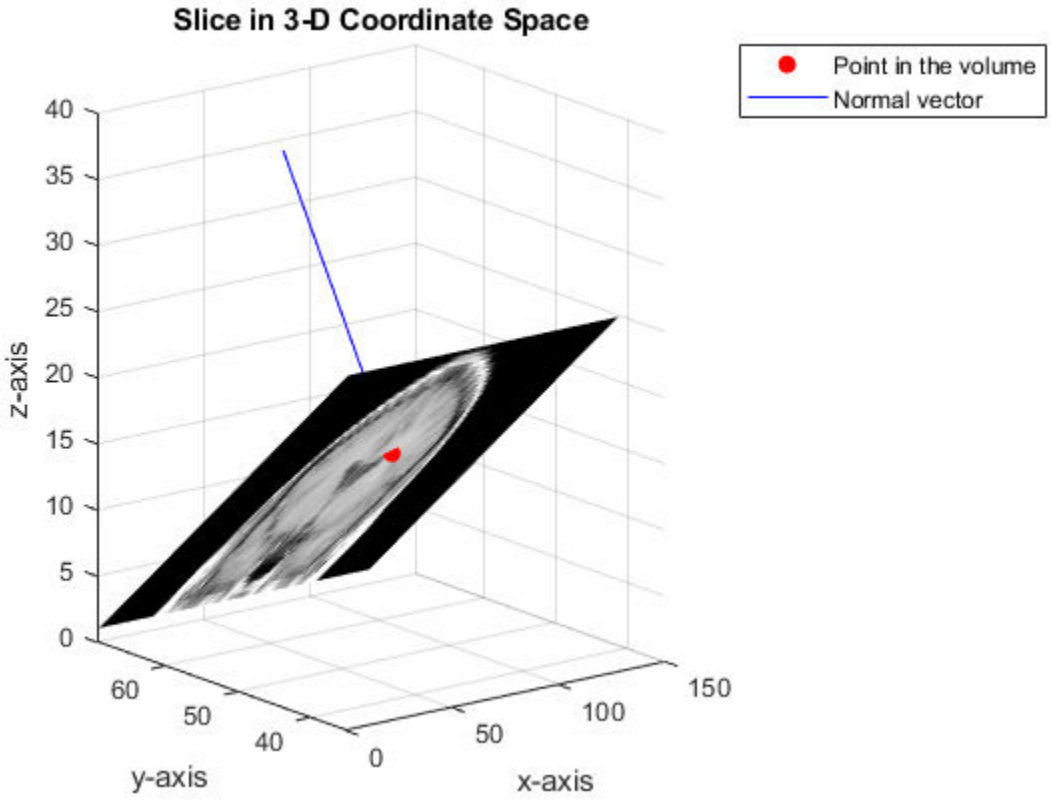
```
[B,x,y,z] = obliqueslice(V,point,normal);
```

Display the extracted slice in the 3-D coordinate space.

```
figure
surf(x,y,z,B,'EdgeColor','None','HandleVisibility','off');
grid on
view([-38 12])
colormap(gray)
xlabel('x-axis')
ylabel('y-axis');
zlabel('z-axis');
title('Slice in 3-D Coordinate Space')
```

Plot the point and the normal vector.

```
hold on
plot3(point(1),point(2),point(3),'or','MarkerFaceColor','r');
plot3(point(1)+[0 normal(1)],point(2)+[0 normal(2)],point(3)+[0 normal(3)], ...
    '-b','MarkerFaceColor','b');
hold off
legend('Point in the volume','Normal vector')
```



Display the extracted slice in the image plane.

```
figure
imshow(B,[])
title('Slice in Image Plane')
```

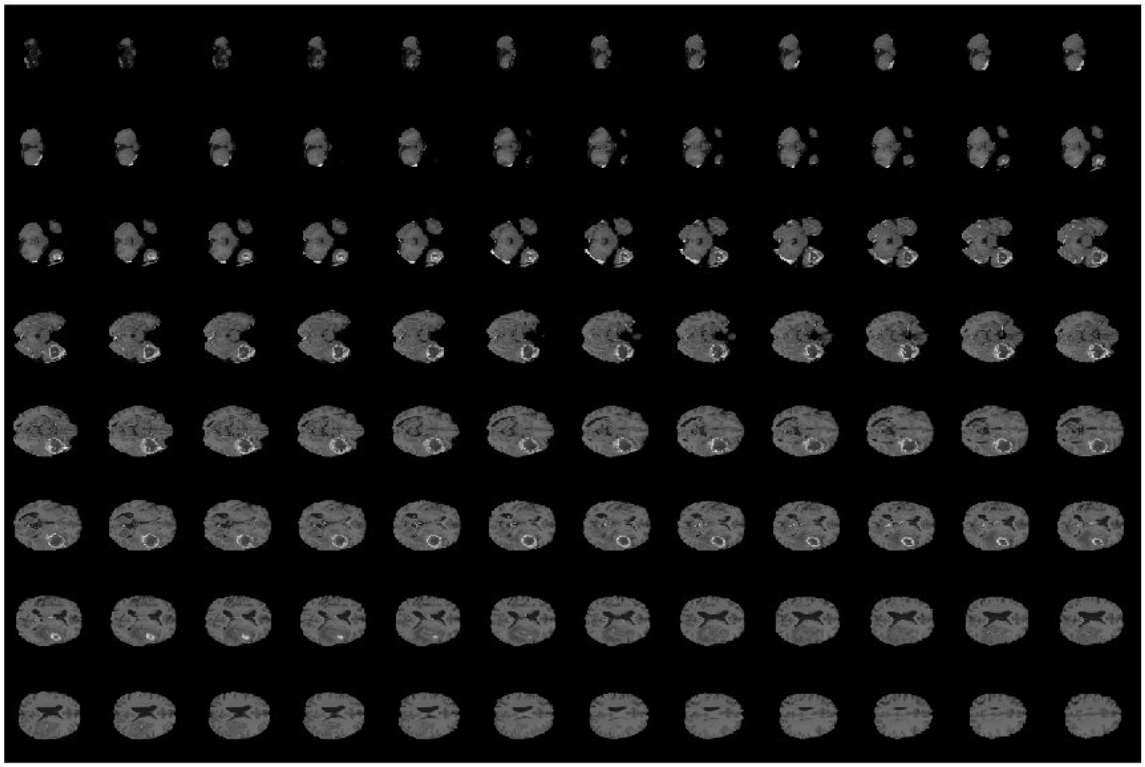
Extract Multiple Slices Along Normal Vector

Load a 3-D volumetric data set into the workspace.

```
s = load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled','images','vol_001.mat'));
V = s.vol;
```

Display horizontal slices of the data by using the `montage` function.

```
montage(V,'Indices',12:118,'Size',[8 12],'DisplayRange',[]);
```



Specify the normal vector to a plane in 3-D coordinate space.

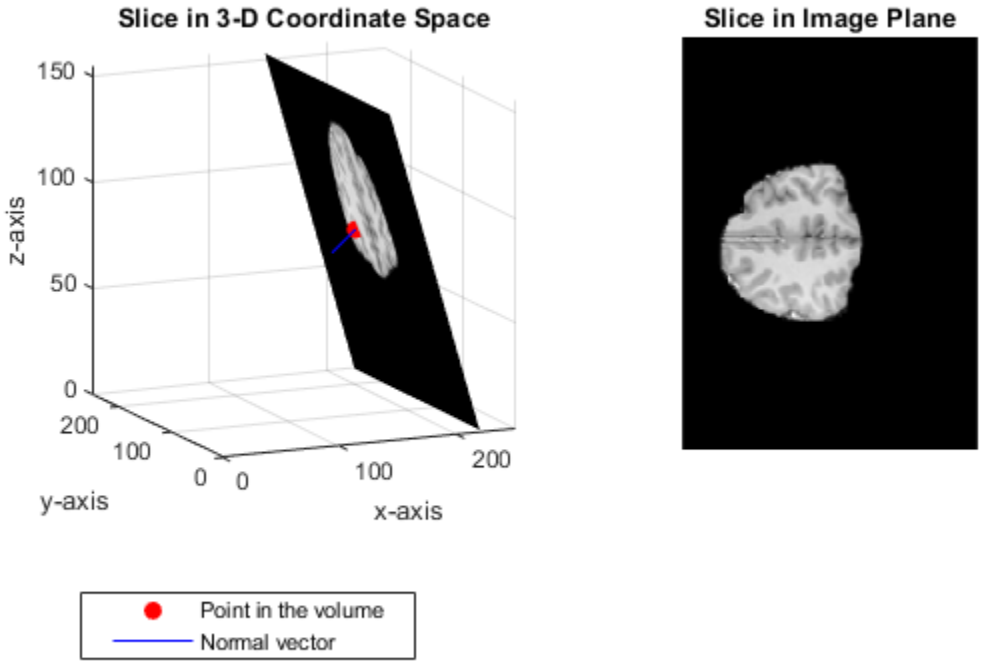
```
normal = [20 0 10];
```

Extract multiple slices along the direction of the normal vector using a `for` loop. In each iteration:

- Specify a point that the slice has to pass through.
- Extract the slice, specifying the output size to `'Full'` and the fill value for padding pixels as 255. The extracted slices are perpendicular to the normal vector and pass through the specified point.
- Display the extracted slices.

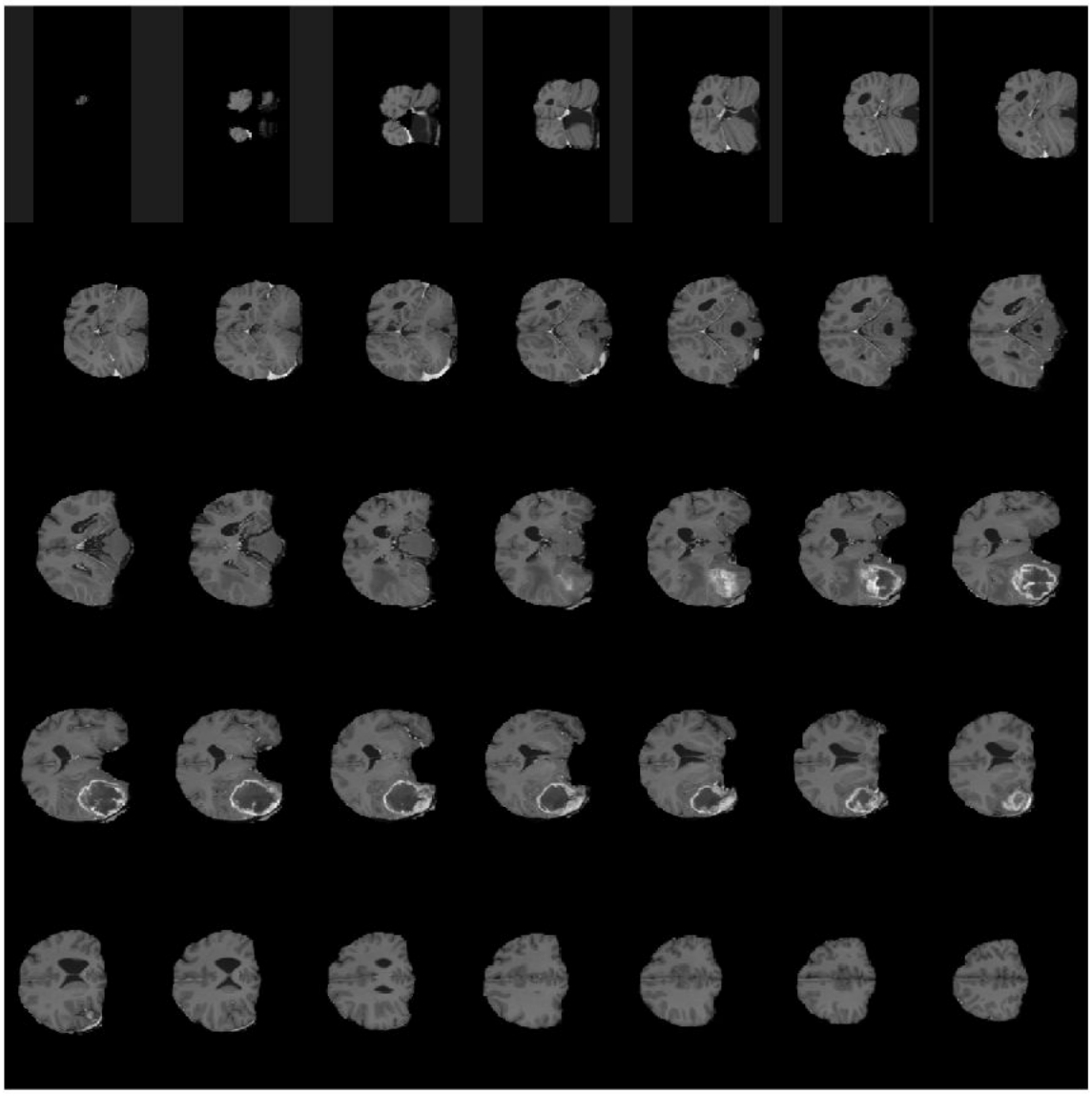
```
sliceIdx = 10:5:180;

figure
for s = 1:length(sliceIdx)

    pt = [sliceIdx(s) 150 80];
    [B,x,y,z] = obliqueslice(V,pt,normal,'OutputSize','Full','FillValues',255);

    Bslices(:,:,s) = B;

    % Display the slice in 3-D coordinate space
    subplot('Position',[0.11 0.36 0.38 0.5])
```

```
    surf(x,y,z,B,'EdgeColor','None','HandleVisibility','off');
    grid on
    view([-24 12])
    colormap(gray)
    xlabel('x-axis')
    ylabel('y-axis');
    zlabel('z-axis');
    zlim([0 155]);
    ylim([0 250]);
    xlim([0 250]);
    title('Slice in 3-D Coordinate Space')
    % Plot the point and the normal vector.
    hold on
    plot3(pt(1),pt(2),pt(3),'or','MarkerFaceColor','r')
    plot3( ...
        pt(1)+[-normal(1) normal(1)], ...
        pt(2)+[-normal(2) normal(2)], ...
        pt(3)+[-normal(3) normal(3)], ...
        '-b','MarkerFaceColor','b')
    legend('Point in the volume','Normal vector','Position',[0.1 0.12 0.3 0.08])
    hold off
    % Display the extracted slice.
    subplot('Position',[0.6 0.37 0.34 0.49])
    imshow(B,[])
    title('Slice in Image Plane')
    pause(0.5);
end
```



Display the extracted image slices by using the `montage` function.

```
figure
montage(Bslices,'Size',[5 7],'DisplayRange',[]);
```



Input Arguments

V — Input volume

3-D numeric array | 3-D categorical array

Input volume, specified as a 3-D numeric or 3-D categorical array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical` | `categorical`

`point` — Point in volume
3-element row vector

Point in the volume, specified as a 3-element row vector of the form $[p_x\ p_y\ p_z]$.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`normal` — Normal vector
3-element row vector

Normal vector, specified as a 3-element row vector of form `[a b c]`. The normal vector is a vector that is perpendicular to a surface or plane.

To extract an orthogonal slice, you can set the normal vector to one of these values:

- [1 0 0] — Extract slice in the *yz*-plane.
- [0 1 0] — Extract slice in the *xz*-plane.
- [0 0 1] — Extract slice in the *xy*-plane.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `obliqueslice(V,point,normal,'OutputSize','Full')`

`Method` — Interpolation method
`'linear'` | `'nearest'`

Interpolation method, specified as the comma-separated pair consisting of `'Method'` and one of these values:

- `'linear'` — linear interpolation
- `'nearest'` — nearest neighbor interpolation

If `V` is numeric, the interpolation method defaults to `'linear'` but can also be specified as `'nearest'`. If `V` is categorical, then the interpolation method must be `'nearest'`.

Data Types: `char` | `string`

`OuputSize` — Size of output image
`'limit'` (default) | `'full'`

Size of output image, specified as the comma-separated pair consisting of `'OutputSize'` and one of these values:

- `'limit'` — The size of the output image is the actual size of the 2-D slice with respect to the dimensions of input volume. If the extracted slice region is not square or rectangular, the function

automatically pads the extracted slice region with extra pixels to yield a square or rectangular image.

- `'full'` — The size of the output image may not be equal to the actual size of the 2-D slice. The size of the output image is set to the maximal slice size that can be obtained from the input volume with respect to the normal vector `normal`. To resize the image, the border of the extracted 2-D slice is padded with extra rows and columns.

  The fill value for the padded pixels is 0 by default. You can use the 'FillValues' name-value pair argument to change the value.

Data Types: `char` | `string`

`FillValues` — Fill value for padded pixels
`0` (default) | numeric scalar | character vector | `missing`

Fill value for padded pixels, specified as the comma-separated pair consisting of `'FillValues'` and a numeric scalar, character vector, or `missing`.

When `V` is a numeric array, specify

- `0` for zero padding.
- numeric scalar for constant padding.

When `V` is a categorical array, specify

- character vector that denotes a category in the input data. To know the categories, use the `categories` function.
- `missing`, if the category in input data is equal to `<undefined>`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char`

Output Arguments

`B` — Output 2-D slice
numeric matrix | categorical matrix

Output 2-D slice, returned as a numeric or categorical matrix. The data type of the output slice is same as the data type of the input volume.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical` | `categorical`

`x` — *x*-coordinates of output slice
numeric matrix

x-coordinates of the output slice in the 3-D volume, returned as a numeric matrix of the size same as the output slice, `B`.

Data Types: `single`

`y` — *y*-coordinates of output slice
numeric matrix

y-coordinates of the output slice in the 3-D volume, returned as a numeric matrix of the size same as the output slice, `B`.

Data Types: `single`

`z` — *z*-coordinates of output slice
numeric matrix

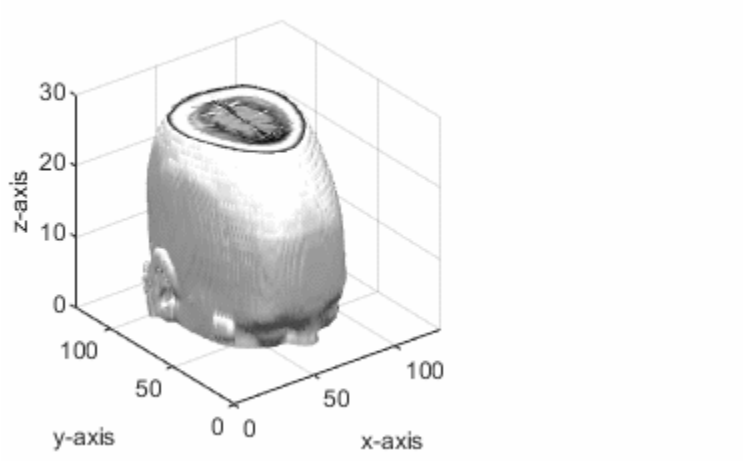
z-coordinates of the output slice in the 3-D volume, returned as a numeric matrix of the size same as the output slice, `B`.

Data Types: `single`

More About

Oblique Slicing

Given a point (p_x, p_y, p_z) and the normal vector (a, b, c), the function solves the plane equation $a(x-p_x)+b(y-p_y)+c(z-p_z) = 0$



The point (p_x, p_y, p_z) lies in the volumetric data. The slicing plane is perpendicular to the normal vector and passes through the given point.

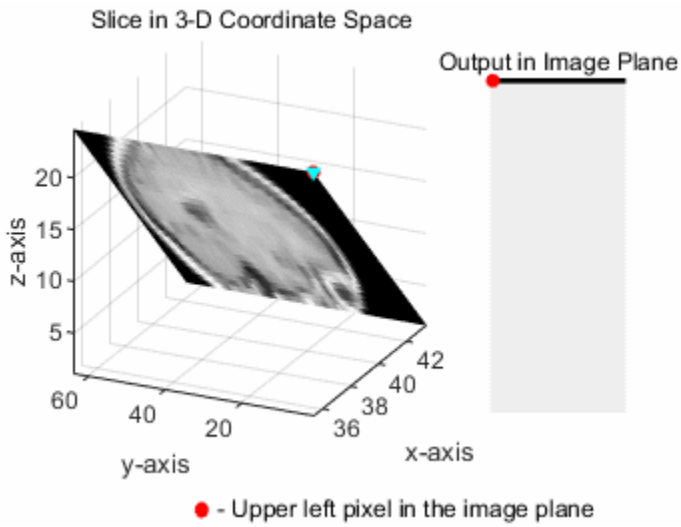
Mapping Values from 3-D Coordinate Space to Image Plane

The order in which the coordinates of the extracted slice in 3-D space is mapped to a 2-D plane depends on its inclination angle with respect to the horizontal and vertical planes.

The `obliqueslice` function returns the output matrices `x`, `y`, and `z` that contain the *x*, *y*, *z* coordinates of the points in 3-D coordinate space that form the image slice. The `obliqueslice` function interpolates the intensity values at these points and maps it to the 2-D plane. The first value in the output matrices *x*(1,1), *y*(1,1), *z*(1,1) specify 3-D coordinate of a point that maps as the upper-left pixel, (1, 1) in the image plane. Starting from this point, the 3-D coordinates that constitute the image slice along with the associated intensity values are read in left-to-right, top-to-bottom scan order. These intensity values fill the 2-D image plane in the same left-to-right, top-to-bottom scan order.

Suppose *V* is the input volumetric data and *B* is the output 2-D image, then $B(i,j) = V(a,b,c)$.

Where, $a = x(i,j)$, $b = y(i,j)$, and $c = z(i,j)$.



See Also

Functions
`slice`

Objects
`sliceViewer` | `orthosliceViewer` | `volshow`

Introduced in R2020a

offsetstrel

Morphological offset structuring element

Description

An `offsetstrel` object represents a nonflat morphological structuring element, which is an essential part of morphological dilation and erosion operations.

A nonflat structuring element is a matrix that identifies the pixel in the image being processed and defines the neighborhood used in the processing of that pixel. A nonflat structuring element contains finite values used as additive offsets in the morphological computation. The center pixel of the matrix, called the origin, identifies the pixel in the image that is being processed. Pixels in the neighborhood with the value `-Inf` are not used in the computation.

You can only use `offsetstrel` objects for morphological operations on grayscale images.

To create a flat structuring element, use `strel`.

Creation

Syntax

```
SE = offsetstrel(offset)

SE = offsetstrel('ball',r,h)
SE = offsetstrel('ball',r,h,n)
```

Description

`SE = offsetstrel(offset)` creates a nonflat structuring element with the additive offset specified in the matrix `offset`.

`SE = offsetstrel('ball',r,h)` creates a nonflat, ball-shaped structuring element whose radius in the *x*-*y* plane is `r` and whose maximum offset height is `h`. For improved performance, `offsetstrel` approximates this shape by a sequence of eight nonflat line-shaped structuring elements.

`SE = offsetstrel('ball',r,h,n)` creates a nonflat ball-shaped structuring element, where `n` specifies the number of nonflat, line-shaped structuring elements that `offsetstrel` uses to approximate the shape. Morphological operations using ball approximations run much faster when you specify a value for `n` greater than `0`.

Input Arguments

`offset` — Values to be added to each pixel location in the neighborhood
numeric matrix

Values to be added to each pixel location in the neighborhood when performing the morphological operation, specified as a numeric matrix. Values that are `-Inf` are not considered in the computation.

Data Types: `double`

`r` — Radius of the ball-shaped structuring element
positive integer

Radius of the ball-shaped structuring element in the *x*-*y* plane, specified as a positive integer.

Data Types: `double`

`h` — Maximum offset height
real scalar

Maximum offset height, specified as a real scalar.

Data Types: `double`

`n` — Number of nonflat line-shaped structuring elements used to approximate the shape
`8` (default) | positive even number

Number of nonflat line-shaped structuring elements used to approximate the shape, specified as a positive even number or `0`.

Value of `n`	Behavior
`n > 0`	`offsetstrel` uses a sequence of `n` nonflat, line-shaped structuring elements to approximate the shape. `n` must be an even number.
`n = 0`	`offsetstrel` does not use any approximation. The structuring element members comprise all pixels whose centers are no greater than `r` away from the origin. The corresponding height values are determined from the formula of the ellipsoid specified by `r` and `h`.

Data Types: `double`

Properties

`Offset` — Structuring element neighborhood with offsets
numeric matrix

Structuring element neighborhood with offsets, specified as a numeric matrix.

Data Types: `double`

`Dimensionality` — Dimensions of structuring element
nonnegative scalar

Dimensions of structuring element, specified as a nonnegative scalar.

Data Types: `double`

Object Functions

imdilate	Dilate image
imerode	Erode image
imclose	Morphologically close image
imopen	Morphologically open image
imbothat	Bottom-hat filtering

imtophat	Top-hat filtering
decompose	Return sequence of decomposed structuring elements
reflect	Reflect structuring element
translate	Translate structuring element

Examples

Create Ball-shaped Structuring Element

Create a ball-shaped structuring element.

```
SE = offsetstrel('ball',5, 6)
```

```
SE =
offsetstrel is a ball shaped offset structuring element with properties:

            Offset: [11x11 double]
    Dimensionality: 2
```

View the structuring element.

```
SE.Offset
```

```
ans = 11×11

      -Inf      -Inf         0    0.7498    1.4996    2.2494    1.4996    0.7498         0      
      -Inf    0.7498    1.4996    2.2494    2.9992    2.9992    2.9992    2.2494    1.4996    0.7
         0    1.4996    2.2494    2.9992    3.7491    3.7491    3.7491    2.9992    2.2494    1.4
    0.7498    2.2494    2.9992    3.7491    4.4989    4.4989    4.4989    3.7491    2.9992    2.2
    1.4996    2.9992    3.7491    4.4989    5.2487    5.2487    5.2487    4.4989    3.7491    2.9
    2.2494    2.9992    3.7491    4.4989    5.2487    5.9985    5.2487    4.4989    3.7491    2.9
    1.4996    2.9992    3.7491    4.4989    5.2487    5.2487    5.2487    4.4989    3.7491    2.9
    0.7498    2.2494    2.9992    3.7491    4.4989    4.4989    4.4989    3.7491    2.9992    2.2
         0    1.4996    2.2494    2.9992    3.7491    3.7491    3.7491    2.9992    2.2494    1.4
      -Inf    0.7498    1.4996    2.2494    2.9992    2.9992    2.9992    2.2494    1.4996    0.7
      ⋮
```

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `offsetstrel` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- The `'ball'` input argument must be compile-time constants.
- The methods associated with `offsetstrel` objects are not supported in code generation.

See Also

`strel`

Topics
“Structuring Elements”

Introduced before R2006a

openrset

Open R-Set file and display R-Set

Syntax

```
openrset(filename)
```

Description

`openrset(filename)` opens the reduced resolution dataset (R-Set) file and displays the R-Set in the **Image Viewer** app.

Examples

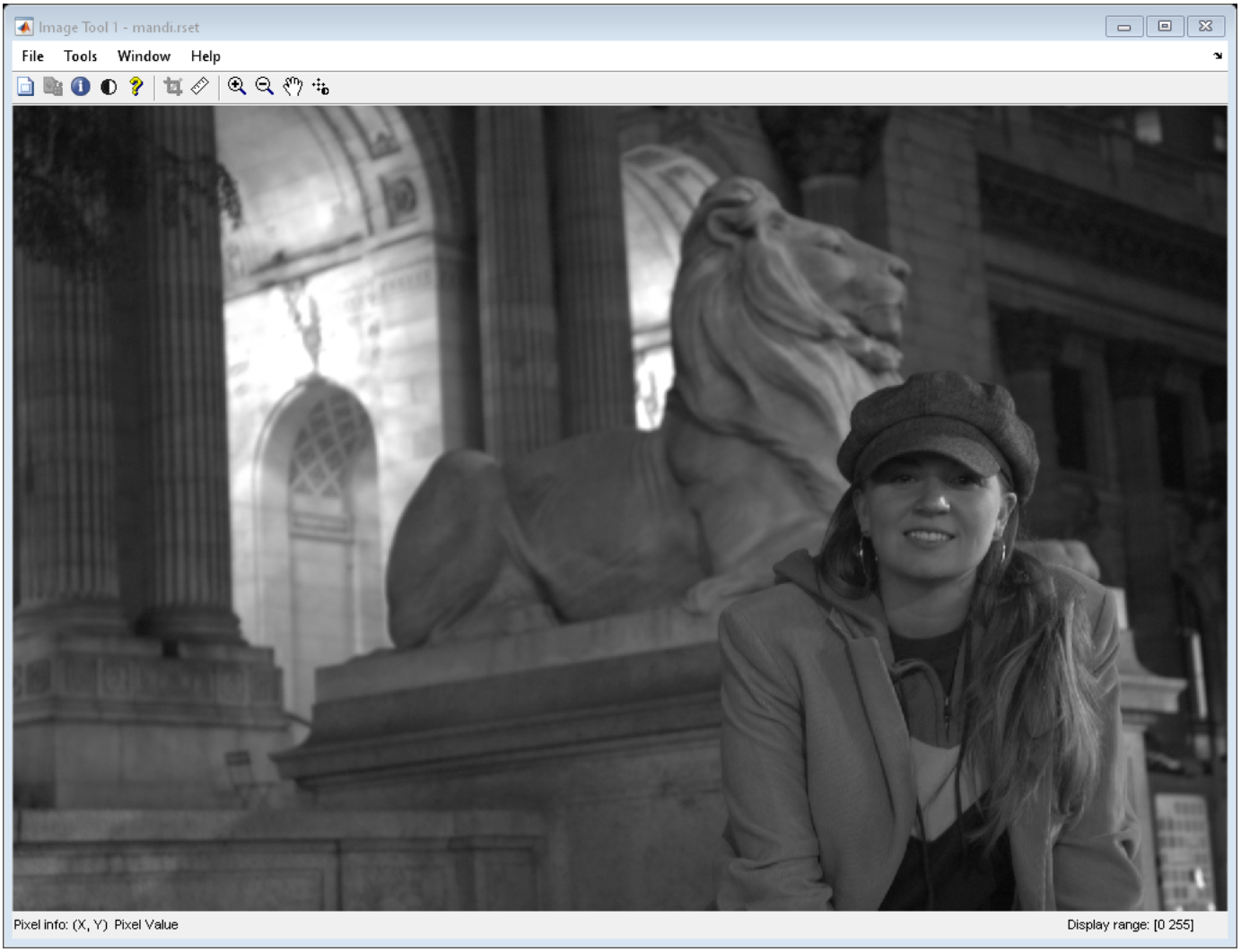
Open Image Stored as R-Set File

Load an R-Set file into the workspace.

```
filename = 'mandi.rset';
```

Open the R-Set file and display the R-Set data.

```
openrset(filename)
```



Input Arguments

`filename` — Name of R-Set file

character vector | string scalar

Name of the R-Set file, specified as a character vector or string scalar. Create an R-Set file by using the `rsetwrite` function.

Data Types: `char` | `string`

See Also

Image Viewer | `rsetwrite` | `isrset`

Introduced in R2010a

ordfilt2

2-D order-statistic filtering

Syntax

```
B = ordfilt2(A,order,domain)
B = ordfilt2(A,order,domain,S)
B = ordfilt2( ___ ,padopt)
```

Description

`B = ordfilt2(A,order,domain)` replaces each element in `A` by the `order`th element in the sorted set of neighbors specified by the nonzero elements in `domain`.

`B = ordfilt2(A,order,domain,S)` filters `A`, where `ordfilt2` uses the values of `S` corresponding to the nonzero values of `domain` as additive offsets. You can use this syntax to implement grayscale morphological operations, including grayscale dilation and erosion.

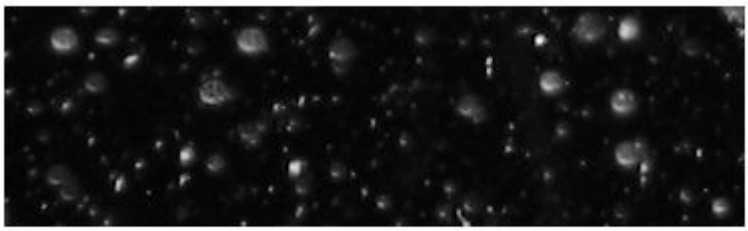
`B = ordfilt2( ___ ,padopt)` filters `A`, where `padopt` specifies how `ordfilt2` pads the matrix boundaries.

Examples

Filter Image with Maximum Filter

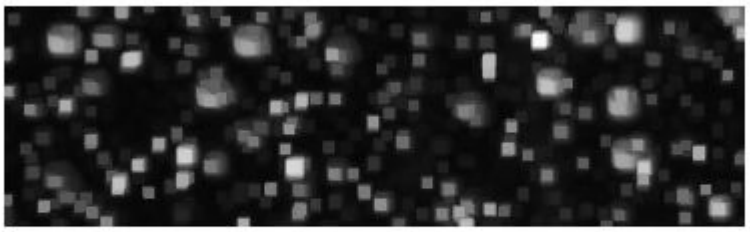
Read image into workspace and display it.

```
A = imread('snowflakes.png');
figure
imshow(A)
```



Filter the image and display the result.

```
B = ordfilt2(A,25,true(5));
figure
imshow(B)
```



Input Arguments

`A` — Data to filter

2-D numeric matrix | 2-D logical matrix

Data to filter, specified as a 2-D numeric matrix or 2-D logical matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`order` — Element to replace target pixel

positive integer

Element to replace the target pixel, specified as a real scalar integer.

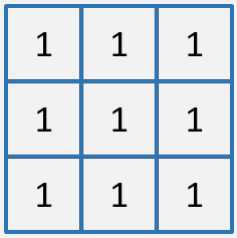
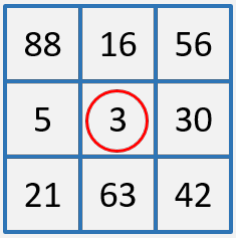
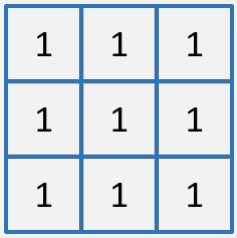
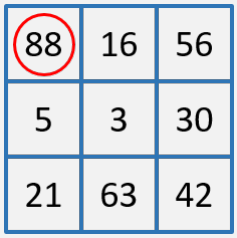
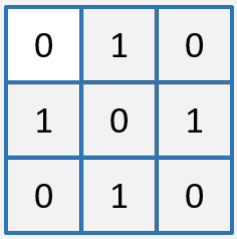
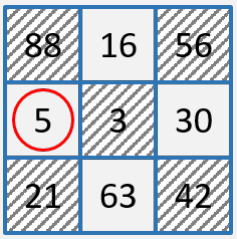
Data Types: `double`

`domain` — Neighborhood

2-D numeric matrix | 2-D logical matrix

Neighborhood, specified as a numeric or logical matrix containing `1`s and `0`s. `domain` is equivalent to the structuring element used for binary image operations. The `1`-valued elements define the neighborhood for the filtering operation. The table gives examples of some common filters.

Type of Filtering Operation	MATLAB code	Neighborhood	Sample Image Data, Indicating Selected Element
Median filter	`B = ordfilt2(A,5,ones(3,3))`	1 1 1 1 1 1 1 1 1	88 16 56 5 3 30 21 63 42

Type of Filtering Operation	MATLAB code	Neighborhood	Sample Image Data, Indicating Selected Element
Minimum filter	`B = ordfilt2(A,1,ones(3,3))`		
Maximum filter	`B = ordfilt2(A,9,ones(3,3))`		
Minimum of north, east, south, and west neighbors	`B = ordfilt2(A,1, [0 1 0; 1 0 1; 0 1 0])`		

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

S — Additive offsets

numeric matrix

Additive offsets, specified as a numeric matrix of the same size as `domain`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

padopt — Padding option

`'zeros'` (default) | `'symmetric'`

Padding option, specified as one of the following values.

Option	Description
`'zeros'`	Pad array boundaries with 0's.
`'symmetric'`	Pad array with mirror reflections of itself.

Data Types: `char` | `string`

Output Arguments

B — Filtered data

2-D numeric matrix | 2-D logical matrix

Filtered data, returned as a 2-D numeric matrix or 2-D logical matrix of the same class as the input data `A`.

Tips

- When working with large domain matrices that do not contain any zero-valued elements, `ordfilt2` can achieve higher performance if `A` is in an integer data format (`uint8`, `int8`, `uint16`, `int16`). The gain in speed is larger for `uint8` and `int8` than for the 16-bit data types. For 8-bit data formats, the domain matrix must contain seven or more rows. For 16-bit data formats, the domain matrix must contain three or more rows and 520 or more elements.

References

[1] Haralick, Robert M., and Linda G. Shapiro, *Computer and Robot Vision,* Volume I, Addison-Wesley, 1992.

[2] Huang, T.S., G.J.Yang, and G.Y.Tang. "A fast two-dimensional median filtering algorithm.", IEEE transactions on Acoustics, Speech and Signal Processing, Vol ASSP 27, No. 1, February 1979

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `ordfilt2` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `ordfilt2` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- When generating code, the `padopt` argument must be a compile-time constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- GPU code generation requires the inputs to be bounded. If the input is of variable dimension, the software generates C code.
- When generating code, the `padopt` argument must be a compile-time constant.
- The generated GPU code is not optimized if the `domain` value that defines the neighborhood for the filtering operation is of size greater than `11x11`.

  For better performance, consider setting the `StackLimitPerThread` option in the `coder.gpuConfig` object to `Inf`.

See Also

`medfilt2`

Introduced before R2006a

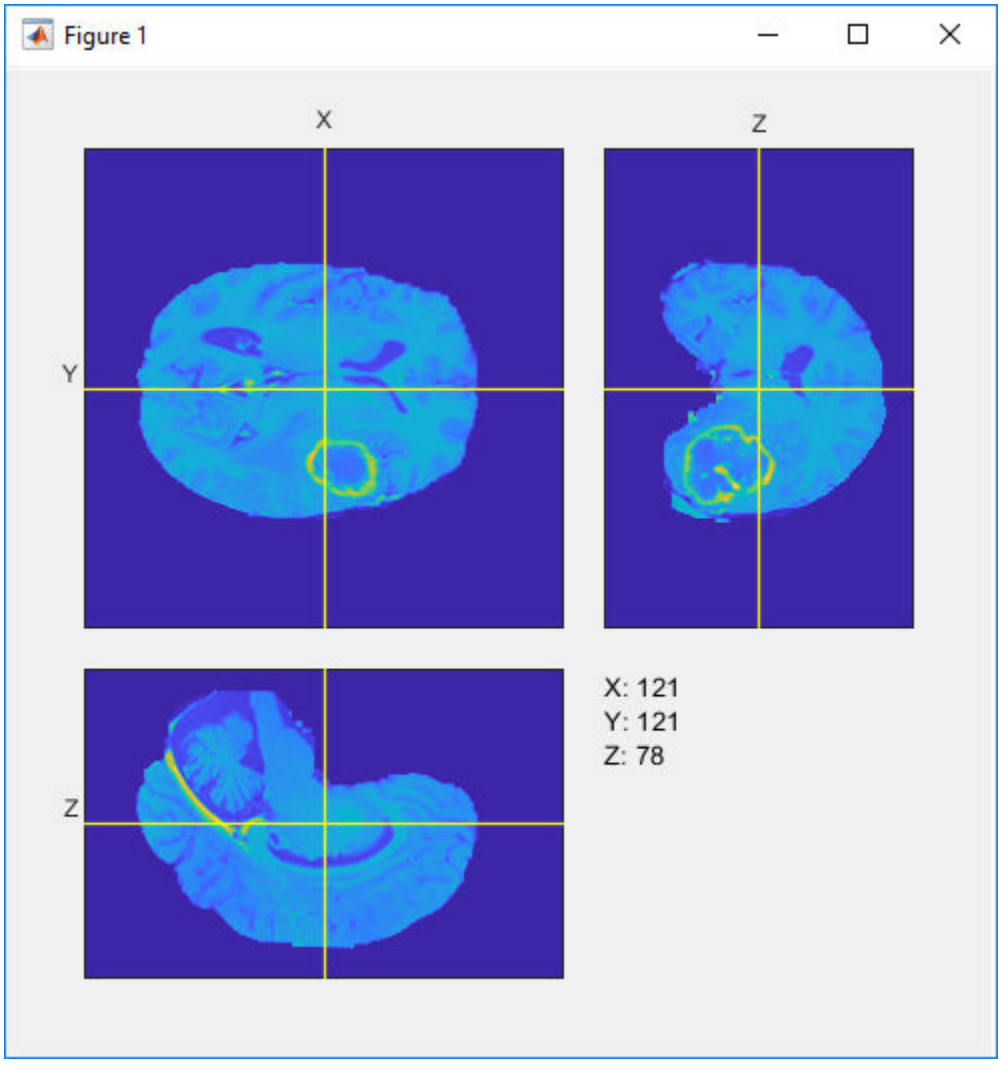
orthosliceViewer

Browse orthogonal slices in grayscale or RGB volume

Description

An `orthosliceViewer` object displays volumetric image data by presenting three orthogonal views of the volume along the *x*, *y*, and *z* dimensions.

Use `orthosliceViewer` to look at individual slices in a volume. The `orthosliceViewer` opens, displaying the center slice in each dimension. Each view of the image stack includes a crosshair that you can use to view the different slices of the image stack. The crosshairs are linked so that if you move one, the crosshairs in the related views also move.



The `orthosliceViewer` object supports properties, object functions, and events that you can use to customize its appearance and functioning. The `orthosliceViewer` object can send notifications

when certain events occur, such as the crosshair moving. For more information, see “Events” on page 1-2495.

Note By default, clicking and dragging the mouse in the slices displayed interactively changes their brightness and contrast, a technique called window/level. Dragging the mouse horizontally from left to right changes the contrast. Dragging the mouse vertically up and down changes the brightness. Holding down the **Ctrl** key when clicking and dragging the mouse accelerates changes. Holding down the **Shift** key while clicking and dragging the mouse slows the rate of change. Press these keys before clicking and dragging. To control this behavior, use the DisplayRangeInteraction property.

Creation

Description

`orthosliceViewer(V)` displays the volume `V` in a figure.

`orthosliceViewer(___,Name,Value)` sets properties on page 1-2487 using name-value pair arguments. You can specify multiple name-value pairs. Enclose each property name in single quotes.

Example: `orthosliceViewer(V,'Colormap',cmap)` creates an `orthosliceViewer` object and specifies the colormap used to display the volume.

`s = orthosliceViewer(___)` returns an `orthosliceViewer` object, `s`, with properties that can be used to control the visualization of the images. Use input arguments from any of the previous syntaxes.

Input Arguments

`V` — Input volume
numeric array

Input volume, specified as an *m*-by-*n*-by-*p*-by-*c* numeric array. For grayscale volumes, `c` is 1. For RGB volumes, *c* is 3. RGB volumes can only be of class `uint8`, `uint16`, `single`, and `double`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Properties

General Properties

`Colormap` — Colormap of image stack
`gray(256)` (default) | *m*-by-3 numeric array

Colormap of the image stack, specified as an *m*-by-3 numeric array with values in the range [0, 1]. The `Colormap` property has no effect when `V` is an RGB image stack.

`DisplayRange` — Display range of grayscale volume
`[min(V(:)) max(V(:))]` (default) | 2-element vector

Display range of grayscale volume, specified as a 2-element vector of the form `[low high]`. The value `low` (and any value less than `low`) displays as black. The value `high` (and any value greater than `high`) displays as white. Values in between are displayed as intermediate shades of gray, using

the default number of gray levels. If you specify an empty matrix (`[]`), `orthosliceViewer` uses the default value. `DisplayRange` has no effect when you specify an RGB volume.

`DisplayRangeInteraction` — Interactive control of display range

`'on'` | `'off'`

Interactive control of the display range, specified as one of the following values. This property has no affect when you specify an RGB image stack. For more information about using this capability, see Events on page 1-2495.

Value	Description
`'on'` (default for grayscale intensity volumes)	You can control the display range of a grayscale image stack by left-clicking the mouse and dragging it on the axes.
`'off'` (default for logical and RGB volumes)	No display range interactivity.

`Parent` — Parent of `orthosliceViewer` object

`gcf` (default) | uipanel | figure

Parent of the `orthosliceViewer` object, specified as a handle to a `uipanel` or as a figure created with either the `figure` or `uifigure` function. If you do not specify a parent, the parent of the `orthosliceViewer` object is `gcf`.

`ScaleFactors` — Scale factors used to rescale the volume

`[1 1 1]` (default) | 1-by-3 positive numeric vector

Scale factors used to rescale the volume, specified as a 1-by-3 positive numeric vector. The values in the array correspond to the scale factor applied in the *x*, *y*, and *z* directions.

`SliceNumbers` — Indices of image slices to be displayed

center slices in each orthogonal direction | 1-by-3 nonnegative numeric array

Indices of image slices to be displayed, specified as a 1-by-3 nonnegative numeric array. `orthosliceViewer` displays the corresponding slices at the `[x,y,z]` indices in the YZ, XZ, and XY views.

Crosshair Properties

`CrosshairColor` — Crosshair color

`[1 1 0]` (default) | RGB triplet | color name | short color name

Crosshair color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	

Color Name	Short Name	RGB Triplet	Appearance
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'CrosshairColor','r'

Example: 'CrosshairColor','green'

Example: 'CrosshairColor',[0 0.4470 0.7410]

CrosshairEnable — State of linked crosshair objects

'on' (default) | 'inactive' | 'off'

State of the linked crosshair objects, specified as one of the values in this table.

Value	Description
'on'	Crosshair is visible and can be interacted with.
'inactive'	Crosshair is visible but cannot be interacted with
'off'	Crosshair is not visible.

CrosshairLineWidth — Width of crosshair line

number of points per screen pixel (default) | positive numeric scalar

Width of the crosshair line, specified as a positive numeric scalar, measured in points. The default value is the number of points per screen pixel.

CrosshairStripeColor — Color of crosshair stripe

'none' (default) | RGB triplet | color name | short color name

Color of the crosshair stripe, specified as an RGB triplet, a color name, a short color name, or 'none'. If you specify 'none', then the crosshair is a solid color specified by the CrosshairColor

property. Otherwise, the crosshair is striped, with colors alternating between the color specified by this property and the color specified by the `CrosshairColor` property.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'CrosshairStripeColor','r'`

Example: `'CrosshairStripeColor','green'`

Example: `'CrosshairStripeColor',[0 0.4470 0.7410]`

Object Functions

addlistener Create event listener bound to event source
getAxesHandles Get handles to axes in Orthoslice Viewer

Examples

View MRI Data in Orthoslice Viewer

Load an image stack into the workspace.

```
load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled','images','vol_001.mat'));
```

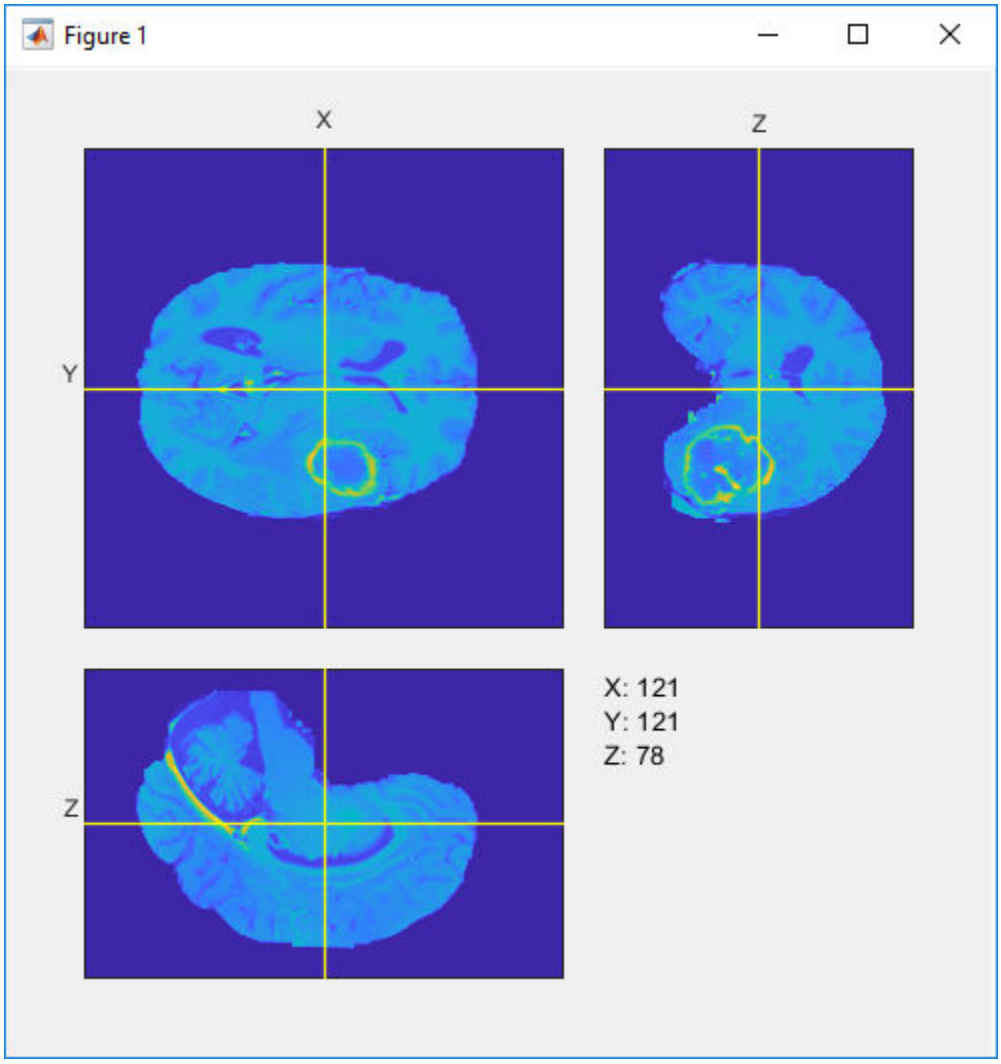
Create a custom Colormap.

```
cmap = parula(256);
```

View the MRI data in the Orthoslice Viewer.

```
s = orthosliceViewer(vol,'Colormap',cmap)
```

```
s =
  orthosliceViewer with properties:

              SliceNumbers: [121 121 78]
             CrosshairColor: [1 1 0]
        CrosshairLineWidth: 1
       CrosshairStripeColor: 'none'
            CrosshairEnable: 'on'
                     Parent: [1×1 Panel]
                   Colormap: [256×3 double]
              DisplayRange: [0 2239]
              ScaleFactors: [1 1 1]
    DisplayRangeInteraction: 'on'
```



Create GIF of MRI Data Slices using Orthoslice Viewer

Load MRI data and view it in the Orthoslice Viewer.

```
load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled','images','vol_001.mat'));
s = orthosliceViewer(vol);
```

Get the handle of the axes that contains the slice.

```
[hXYAxes, hYZAxes, hXZAxes] = getAxesHandles(s);
```

Turn off crosshair for better visibility.

```
set(s,'CrosshairEnable','off');
```

Specify the name of the GIF file.

```
filename = 'animatedYZSlice.gif';
```

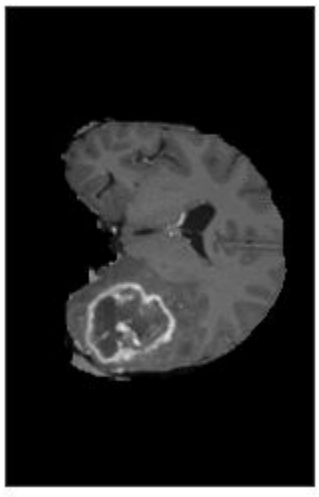
Create an array of slice numbers in the required direction. Consider the YZ direction.

```
sliceNums = 1:240;
```

Loop through and create an image at the specified slice position.

```
for idx = sliceNums
    % Update X slice number to get YZ Slice.
    s.SliceNumbers(1) = idx;

    % Use getframe to capture image.
    I = getframe(hYZAxes);
    [indI,cm] = rgb2ind(I.cdata,256);

    % Write frame to the GIF File.
    if idx == 1
        imwrite(indI,cm,filename,'gif','Loopcount',inf,'DelayTime',0.05);
    else
        imwrite(indI,cm,filename,'gif','WriteMode','append','DelayTime',0.05);
    end
end
```

View the animated GIF.



Set Up Listener for Orthoslice Viewer Crosshair Events

Load a stack of images.

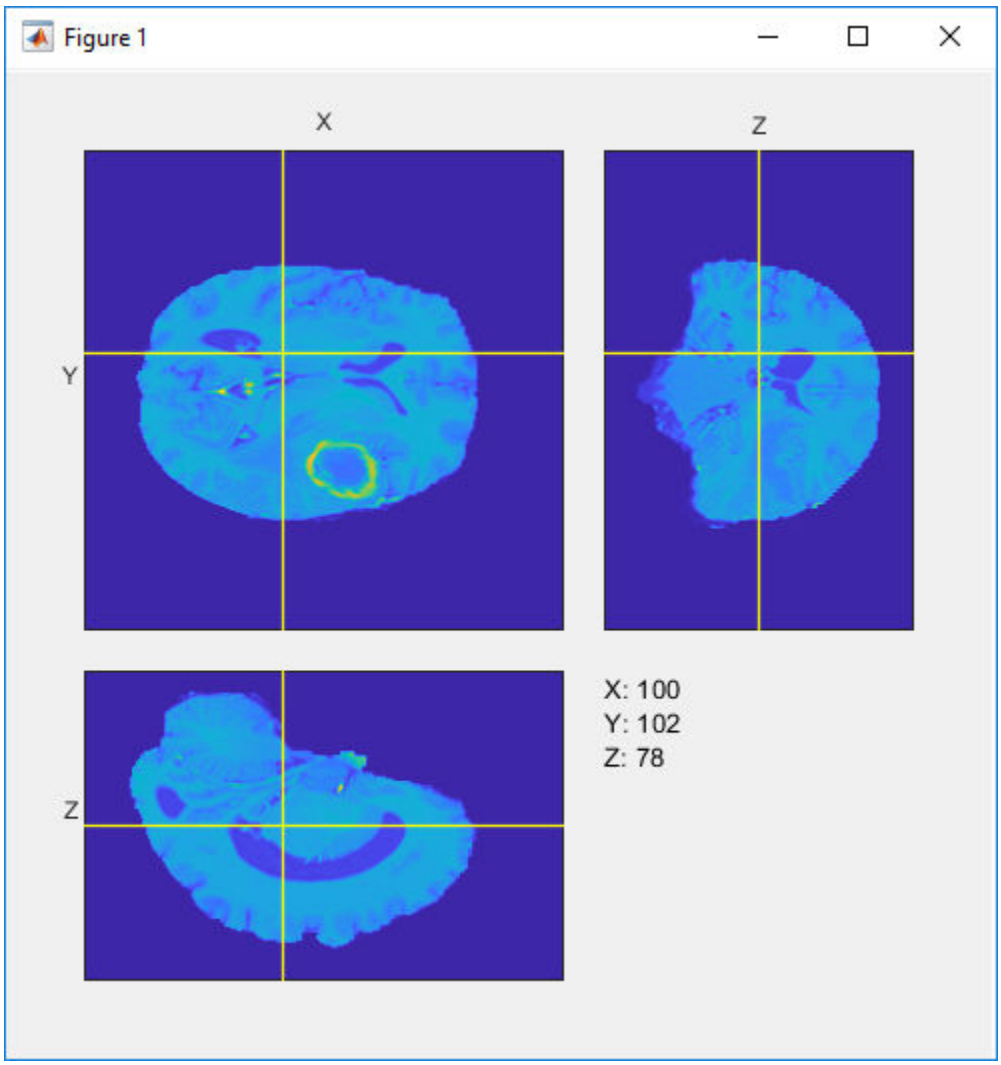
```
load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled','images','vol_001.mat'));
```

Create a custom colormap for viewing slices.

```
cmap = parula(256);
```

View the image stack in the Orthoslice Viewer.

```
os = orthosliceViewer(vol,'Colormap',cmap);
```



Set up listeners for the two Orthoslice Viewer crosshair moving events. When you move the crosshair, the Orthoslice Viewer sends notifications of these events and executes the callback function you specify.

```
addlistener(os,'CrosshairMoving',@allevents);
addlistener(os,'CrosshairMoved',@allevents);
```

The `allevents` callback function displays the name of each event with the previous position and the current position of the crosshair.

```
function allevents(src,evt)
evname = evt.EventName;
    switch(evname)
        case{'CrosshairMoved'}
            disp(['Crosshair moved previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['Crosshair moved current position: ' mat2str(evt.CurrentPosition)]);
        case{'CrosshairMoving'}
            disp(['Crosshair moving previous position: ' mat2str(evt.PreviousPosition)]);
```

```
            disp(['Crosshair moving current position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

More About

Events

The `orthosliceViewer` object can send notifications when the crosshair moves. To receive these notifications, use the `addListener` function to set up a listener. To set up a listener, specify the name of the event, for example, `'CrosshairMoving'`, and the function you want executed when the event occurs. The following table lists events supported by the `orthosliceViewer` object. For an example, see “Set Up Listener for Orthoslice Viewer Crosshair Events” on page 1-2493.

Event Name	Trigger	Event Data	Event Attributes

`CrosshairMoving`	The crosshair in the `orthosliceViewer` is moving.	`images.stack.browser.CrosshairMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`CrosshairMoved`	The crosshair in the `orthosliceViewer` has stopped moving.	`images.stack.browser.CrosshairMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

See Also

`sliceViewer` | **Volume Viewer** | `volshow` | `slice` | `Crosshair` | `obliqueslice`

Introduced in R2019b

getAxesHandles

Get handles to axes in Orthoslice Viewer

Syntax

```
[hXY hYZ hXZ] = getAxesHandles(s)
```

Description

`[hXY hYZ hXZ] = getAxesHandles(s)` returns the axes containing each of the views of the image volume in the `orthosliceViewer` object `s`.

Input Arguments

s — Orthoslice Viewer

`orthosliceViewer` object

Orthoslice Viewer, specified as an `orthosliceViewer` object.

Output Arguments

[hXY hYZ hXZ] — Axes in Orthoslice Viewer

1-by-3 vector of `Axes` objects

Axes in Orthoslice Viewer, returned as a 1-by-3 vector of `Axes` objects.

Examples

Create GIF of MRI Data Slices using Orthoslice Viewer

Load MRI data and view it in the Orthoslice Viewer.

```
load(fullfile(toolboxdir('images'),'imdata','BrainMRILabeled','images','vol_001.mat'));
s = orthosliceViewer(vol);
```

Get the handle of the axes that contains the slice.

```
[hXYAxes, hYZAxes, hXZAxes] = getAxesHandles(s);
```

Turn off crosshair for better visibility.

```
set(s,'CrosshairEnable','off');
```

Specify the name of the GIF file.

```
filename = 'animatedYZSlice.gif';
```

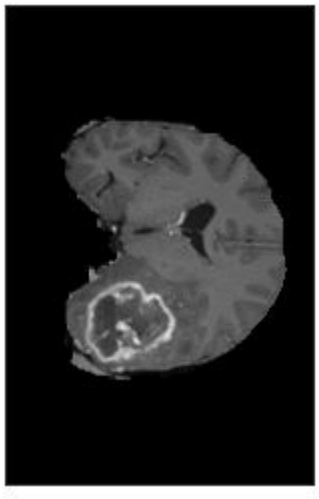
Create an array of slice numbers in the required direction. Consider the YZ direction.

```
sliceNums = 1:240;
```

Loop through and create an image at the specified slice position.

```
for idx = sliceNums
    % Update X slice number to get YZ Slice.
    s.SliceNumbers(1) = idx;

    % Use getframe to capture image.
    I = getframe(hYZAxes);
    [indI,cm] = rgb2ind(I.cdata,256);

    % Write frame to the GIF File.
    if idx == 1
        imwrite(indI,cm,filename,'gif','Loopcount',inf,'DelayTime',0.05);
    else
        imwrite(indI,cm,filename,'gif','WriteMode','append','DelayTime',0.05);
    end
end
```

View the animated GIF.



See Also

`orthosliceViewer`

Introduced in R2019b

otf2psf

Convert optical transfer function to point-spread function

Syntax

```
PSF = otf2psf(OTF)
PSF = otf2psf(OTF,sz)
```

Description

`PSF = otf2psf(OTF)` computes the inverse Fast Fourier Transform of the optical transfer function (`OTF`) and creates a point-spread function (`PSF`), centered at the origin.

`PSF = otf2psf(OTF,sz)` specifies the size, `sz`, of the output point-spread function.

Examples

Convert OTF to PSF

Create a point-spread function (PSF).

```
PSF  = fspecial('gaussian',13,1);
```

Convert the PSF to an Optical Transfer Function (OTF).

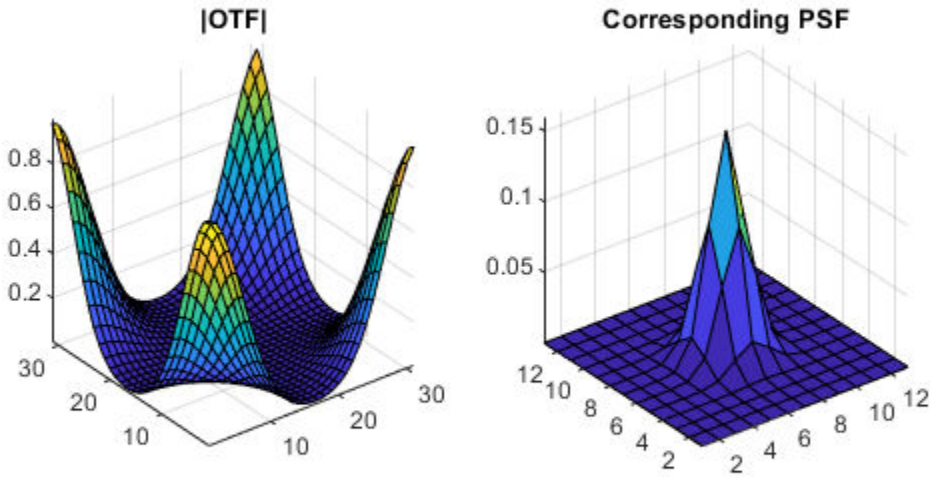
```
OTF  = psf2otf(PSF,[31 31]);
```

Convert the OTF back to a PSF.

```
PSF2 = otf2psf(OTF,size(PSF));
```

Plot the PSF and the OTF.

```
subplot(1,2,1)
surf(abs(OTF))
title('|OTF|');
axis square
axis tight
subplot(1,2,2)
surf(PSF2)
title('Corresponding PSF');
axis square
axis tight
```



Input Arguments

OTF — Optical transfer function

numeric array

Optical transfer function, specified as a numeric array of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`
Complex Number Support: Yes

sz — Size of point-spread function

vector of positive integers

Size of the output point-spread function PSF, specified as a vector of positive integers. The size of PSF must not exceed the size of OTF in any dimension. By default, PSF is the same size as OTF.

Data Types: `double`

Output Arguments

PSF — Point-spread function

numeric array

Point-spread function, centered at the origin, returned as a numeric array of size `sz`.

Data Types: `double`
Complex Number Support: Yes

Tips

- To center the PSF at the origin, `otf2psf` circularly shifts the values of the output array down (or to the right) until the (1,1) element reaches the central position, then it crops the result to match dimensions specified by `sz`.
- This function is used in image convolution and deconvolution when the operations involve the FFT.

See Also

`psf2otf` | `circshift` | `padarray` | `ifftn` | `fftn`

Topics
"Create Your Own Deblurring Functions"

Introduced before R2006a

otsuthresh

Global histogram threshold using Otsu's method

Syntax

```
T = otsuthresh(counts)
[T,EM] = otsuthresh(counts)
```

Description

`T = otsuthresh(counts)` computes a global threshold `T` from histogram counts, `counts`, using Otsu's method [1]. Otsu's method chooses a threshold that minimizes the intraclass variance of the thresholded black and white pixels. The global threshold `T` can be used with `imbinarize` to convert a grayscale image to a binary image.

`[T,EM] = otsuthresh(counts)` returns the effectiveness metric, `EM`, which indicates the effectiveness of the thresholding.

Examples

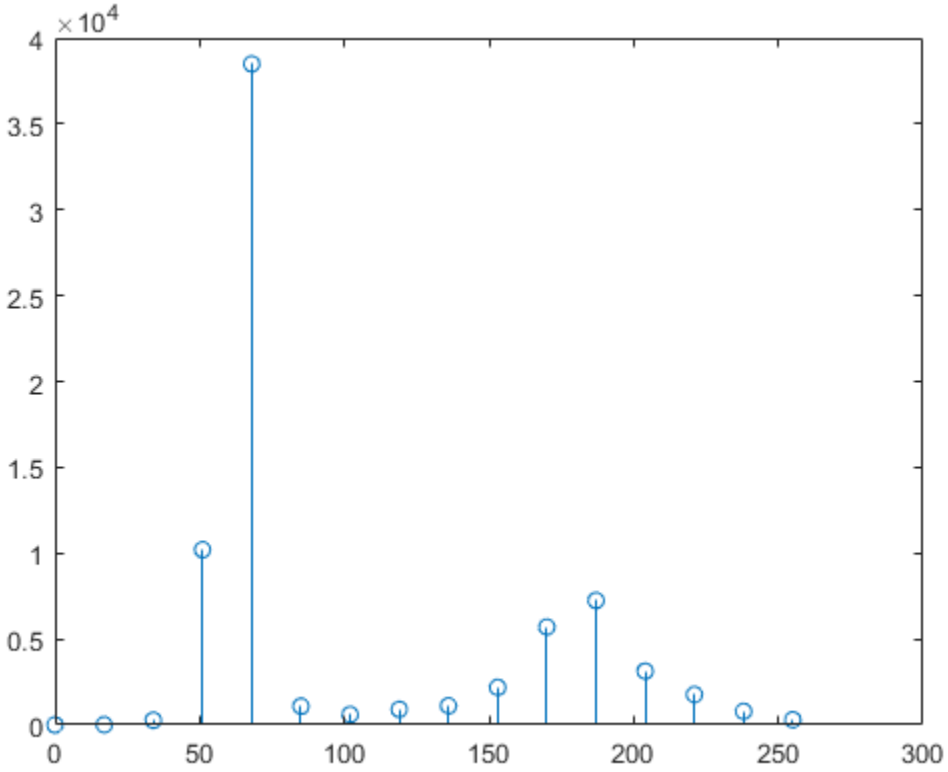
Compute Threshold from Image Histogram and Binarize Image

Read image into the workspace.

```
I = imread('coins.png');
```

Calculate a 16-bin histogram for the image.

```
[counts,x] = imhist(I,16);
stem(x,counts)
```

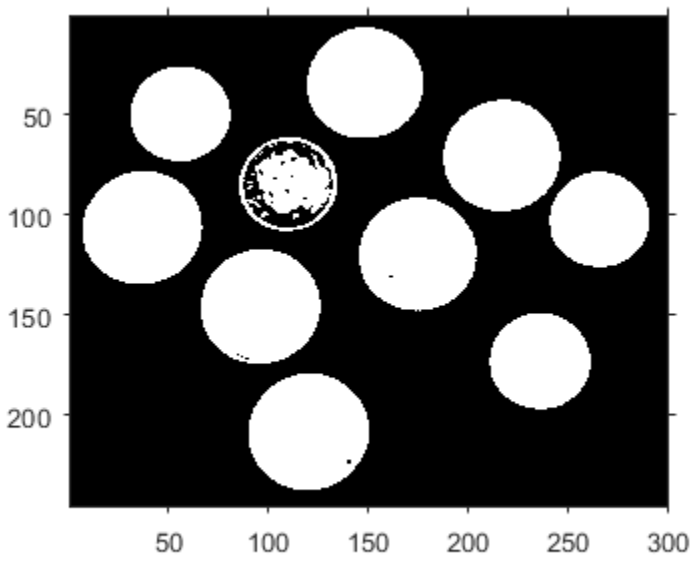


Compute a global threshold using the histogram counts.

```
T = otsuthresh(counts);
```

Create a binary image using the computed threshold and display the image.

```
BW = imbinarize(I,T);
figure
imshow(BW)
```



Input Arguments

`counts` — Histogram counts

vector of nonnegative numbers

Histogram counts, specified as a vector of nonnegative numbers.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`T` — Global threshold

numeric scalar

Global threshold, returned as a numeric scalar in the range [0, 1].

Data Types: `double`

`EM` — Effectiveness metric

numeric scalar

Effectiveness metric of the threshold, returned as a numeric scalar in the range [0, 1]. The lower bound is attainable only by histogram counts with all data in a single non-zero bin. The upper bound is attainable only by histogram counts with two non-zero bins.

Data Types: `double`

References

[1] Otsu, N., "A Threshold Selection Method from Gray-Level Histograms." *IEEE Transactions on Systems, Man, and Cybernetics*. Vol. 9, No. 1, 1979, pp. 62–66.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`otsuthresh` supports the generation of C code (requires MATLAB Coder). For more information, see "Code Generation for Image Processing".

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see "Run MATLAB Functions in Thread-Based Environment".

See Also

`imbinarize` | `adaptthresh` | `graythresh`

Introduced in R2016a

outputLimits

Package:

Find output spatial limits given input spatial limits

Syntax

```
[xLimitsOut,yLimitsOut] = outputLimits(tform,xLimitsIn,yLimitsIn)
[xLimitsOut,yLimitsOut,zLimitsOut] = outputLimits(tform,xLimitsIn,yLimitsIn,
zLimitsIn)
```

Description

`[xLimitsOut,yLimitsOut] = outputLimits(tform,xLimitsIn,yLimitsIn)` estimates the output spatial limits corresponding to a set of input spatial limits, `xLimitsIn` and `yLimitsIn`, given 2-D geometric transformation `tform`.

`[xLimitsOut,yLimitsOut,zLimitsOut] = outputLimits(tform,xLimitsIn,yLimitsIn, zLimitsIn)` estimates the output spatial limits, given 3-D geometric transformation `tform`.

Examples

Estimate the Output Limits for a 2-D Affine Transformation

Create an `affine2d` object that defines a rotation of 10 degrees counter-clockwise.

```
theta = 10;
tform = affine2d([cosd(theta) -sind(theta) 0; sind(theta) cosd(theta) 0; 0 0 1]);
```

```
tform =

  affine2d with properties:

                T: [3x3 double]
    Dimensionality: 2
```

Estimate the output spatial limits, given the geometric transformation.

```
[xlim, ylim] = outputLimits(tform,[1 240],[1 291])
```

```
xlim =

    1.1585  286.8855


ylim =

  -40.6908  286.4054
```

Estimate the Output Limits for a 3-D Affine Transformation

Create an `affine3d` object that defines a different scale factor in each dimension.

```
Sx = 1.2;
Sy = 1.6;
Sz = 2.4;
tform = affine3d([Sx 0 0 0; 0 Sy 0 0; 0 0 Sz 0; 0 0 0 1]);
```

```
tform =

  affine3d with properties:

                T: [4x4 double]
    Dimensionality: 3
```

Estimate the output spatial limits, given the geometric transformation.

```
[xlim, ylim, zlim] = outputLimits(tform,[1 128],[1 128],[1 27])
```

```
xlim =

    1.2000  153.6000


ylim =

    1.6000  204.8000


zlim =

    2.4000   64.8000
```

Input Arguments

`tform` — Geometric transformation

geometric transformation object

Geometric transformation, specified as a geometric transformation object.

For 2-D geometric transformations, `tform` can be a `rigid2d`, `affine2d`, `projective2d`, `LocalWeightedMeanTransformation2D`, `PiecewiseLinearTransformation2D`, or `PolynomialTransformation2D` geometric transformation object.

For 3-D geometric transformations, `tform` can be an `affine3d` or `rigid3d` object.

`xLimitsIn` — Input spatial limits in the *x*-dimension

1-by-2 numeric vector

Input spatial limits in the *x*-dimension, specified as a 1-by-2 numeric vector.

Data Types: `double`

`yLimitsIn` — Input spatial limits in the *y*-dimension

1-by-2 numeric vector

Input spatial limits in the *y*-dimension, specified as a 1-by-2 numeric vector.

Data Types: `double`

`zLimitsIn` — Input spatial limits in the *z*-dimension
1-by-2 numeric vector

Input spatial limits in the *z*-dimension, specified as a 1-by-2 numeric vector. Provide `zLimitsIn` only when `tform` is an `affine3d` object or a `rigid3d` object.

Data Types: `double`

Output Arguments

`xLimitsOut` — Output spatial limits in the *x*-dimension
1-by-2 numeric vector

Output spatial limits in the *x*-dimension, returned as a 1-by-2 numeric vector.

Data Types: `double`

`yLimitsOut` — Output spatial limits in the *y*-dimension
1-by-2 numeric vector

Output spatial limits in the *y*-dimension, returned as a 1-by-2 numeric vector.

Data Types: `double`

`zLimitsOut` — Output spatial limits in the *z*-dimension
1-by-2 numeric vector

Output spatial limits in the *z*-dimension, returned as a 1-by-2 numeric vector. `outputLimits` returns `zLimitsIn` only when `tform` is an `affine3d` object or a `rigid3d` object.

Data Types: `double`

See Also

`rigid3d` | `affine2d` | `affine3d` | `projective2d` | `LocalWeightedMeanTransformation2D` | `PiecewiseLinearTransformation2D` | `PolynomialTransformation2D`

Introduced in R2013a

padarray

Pad array

Syntax

```
B = padarray(A,padsize)
B = padarray(A,padsize,padval)
B = padarray( ___ ,direction)
```

Description

`B = padarray(A,padsize)` pads array `A` with an amount of padding in each dimension specified by `padsize`. The `padarray` function pads numeric or logical images with the value `0` and categorical images with the category `<undefined>`. By default, `paddarray` adds padding before the first element and after the last element of each dimension.

`B = padarray(A,padsize,padval)` pads array `A` where `padval` specifies a constant value to use for padded elements or a method to replicate array elements.

`B = padarray( ___ ,direction)` pads `A` in the direction specified by `direction`.

Examples

Add Padding to 2-D and 3-D Arrays

Pad the Beginning of a Vector

Add three elements of padding to the beginning of a vector with padding value 9.

```
A = [ 1 2 3 4 ]
```

```
A = 1×4

     1     2     3     4
```

```
B = padarray(A,3,9,'pre')
```

```
B = 4×4

     9     9     9     9
     9     9     9     9
     9     9     9     9
     1     2     3     4
```

Pad Each Dimension of a 2-D Array

Add three elements of padding to the end of the first dimension of the array and two elements of padding to the end of the second dimension. Use the value of the last array element on each dimension as the padding value.

```
A = [ 1 2; 3 4 ]
```

```
A = 2×2

     1     2
     3     4
```

```
B = padarray(A,[3 2],'replicate','post')
```

```
B = 5×4

     1     2     2     2
     3     4     4     4
     3     4     4     4
     3     4     4     4
     3     4     4     4
```

Pad Each Dimension of a 3-D Array

Add three elements of padding to each dimension of a three-dimensional array. Each pad element contains the value 0.

First create the 3-D array.

```
A = [1 2; 3 4];
B = [5 6; 7 8];
C = cat(3,A,B)
```

```
C =
C(:,:,1) =

     1     2
     3     4


C(:,:,2) =

     5     6
     7     8
```

Pad the 3-D Array

```
D = padarray(C,[3 3],0,'both')
```

```
D =
D(:,:,1) =

     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

```
     0     0     0     0     0     0     0     0
     0     0     0     1     2     0     0     0
     0     0     0     3     4     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0

D(:,:,2) =

     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     5     6     0     0     0
     0     0     0     7     8     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

Input Arguments

A — Array to be padded

numeric array | logical array | categorical array

Array to be padded, specified as a numeric, logical, or categorical array of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `categorical`

padsize — Amount of padding

vector of nonnegative integers

Amount of padding to add to each dimension, specified as a vector of nonnegative integers. For example, a `padsize` value of `[2 3]` adds two elements of padding along the first dimension and three elements of padding along the second dimension.

Data Types: `double`

padval — Pad value

`0` | numeric scalar | `'circular'` | `'replicate'` | `'symmetric'` | string scalar | character vector | `missing`

Pad value, specified as one of the following.

Image Type	Format of Fill Values
Numeric image or logical image	• Numeric scalar — Pad array with elements of constant value. The default pad value of numeric and logical images is `0`. • `'circular'` — Pad with circular repetition of elements within the dimension. • `'replicate'` — Pad by repeating border elements of array. • `'symmetric'` — Pad with mirror reflections of the array along the border.

Image Type	Format of Fill Values
Categorical image	• Valid category in the image, specified as a string scalar or character vector. • `missing`, which corresponds to the `<undefined>` category. This is the default pad value for categorical images. For more information, see `missing`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `string`

`direction` — Direction to pad array
`'both'` (default) | `'post'` | `'pre'`

Direction to pad array along each dimension, specified as one of the following values:

Value	Meaning
`'both'`	Pads before the first element and after the last array element along each dimension.
`'post'`	Pad after the last array element along each dimension.
`'pre'`	Pad before the first array element along each dimension.

Data Types: `char` | `string`

Output Arguments

`B` — Padded array
numeric array | logical array | categorical array

Padded array, returned as an array of the same data type as `A`.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `padarray` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- Input arrays of data type categorical are not supported.
- When generating code, `padarray` supports only up to 3-D inputs
- The input arguments `padval` and `direction` must be compile-time constants.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- Input arrays of data type categorical are not supported.

- When generating code, `padarray` supports only up to 3-D inputs.
- The input arguments `padval` and `direction` must be compile-time constants.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also
`circshift` | `imfilter`

Introduced before R2006a

para2fan

Convert parallel-beam projections to fan-beam

Syntax

```
F = para2fan(P,D)
F = para2fan(P,D,Name,Value)
[F,fanSensorPos,fanRotAngles] = para2fan( ___ )
```

Description

`F = para2fan(P,D)` converts the parallel-beam data `P` to the fan-beam data `F`. Each column of `P` contains the parallel-beam sensor samples at one rotation angle. `D` is the distance from the fan-beam vertex to the center of rotation.

The parallel-beam sensors are assumed to have a one-pixel spacing. The parallel-beam rotation angles are spaced equally to cover [0, 180] degrees. The calculated fan-beam rotation angles have the same spacing as the parallel-beam rotation angles, and cover [0,360) degrees. The calculated fan-beam angles are equally spaced with the spacing set to the smallest angle implied by the sensor spacing.

`F = para2fan(P,D,Name,Value)` uses name-value arguments to control aspects of the data conversion.

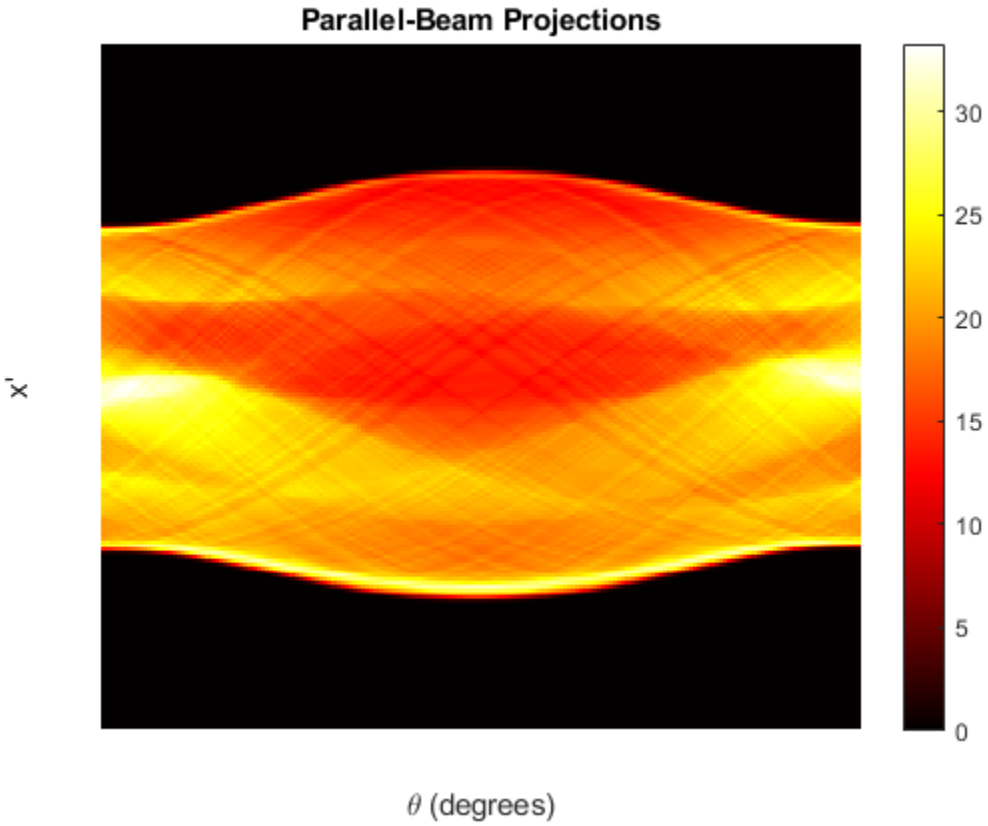
`[F,fanSensorPos,fanRotAngles] = para2fan( ___ )` returns the fan-beam sensor locations in `fanSensorPos` and rotation angles in `fanRotAngles`.

Examples

Convert Parallel-beam Projections to Fan-beam Projections

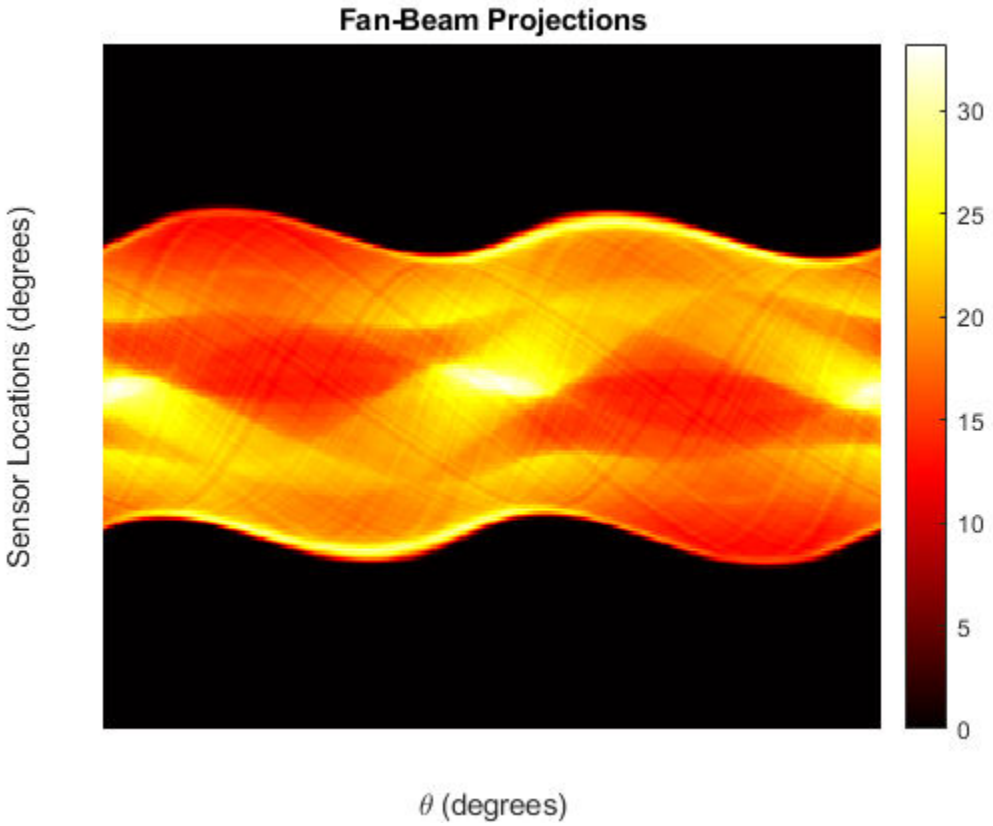
Generate parallel-beam projections

```
ph = phantom(128);
theta = 0:180;
[P,xp] = radon(ph,theta);
imshow(P,[],'XData',theta,'YData',xp,'InitialMagnification','fit')
axis normal
title('Parallel-Beam Projections')
xlabel('\theta (degrees)')
ylabel('x''')
colormap(gca,hot), colorbar
```



Convert to fan-beam projections

```
[F,Fpos,Fangles] = para2fan(P,100);
figure
imshow(F,[],'XData',Fangles,'YData',Fpos,'InitialMagnification','fit')
axis normal
title('Fan-Beam Projections')
xlabel('\theta (degrees)')
ylabel('Sensor Locations (degrees)')
colormap(gca,hot), colorbar
```



Input Arguments

`P` — Parallel-beam projection data

numeric matrix

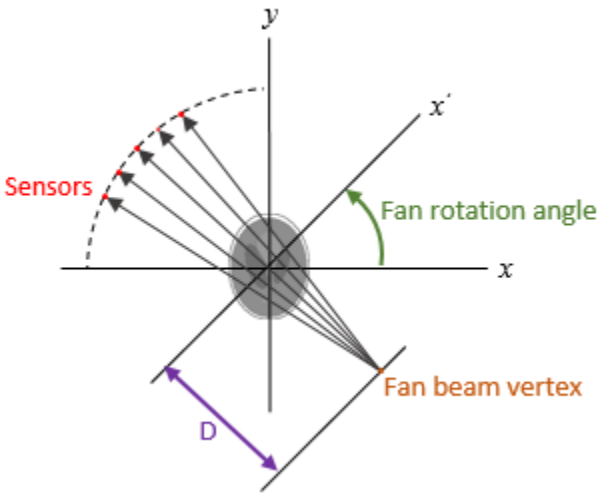
Parallel-beam projection data, specified as a numeric matrix. Each column of `P` contains the parallel-beam data at one rotation angle. The number of columns indicates the number of parallel-beam rotation angles and the number of rows indicates the number of parallel-beam sensors.

Data Types: `double` | `single`

`D` — Distance from fan beam vertex to center of rotation

positive number

Distance in pixels from the fan beam vertex to the center of rotation, specified as a positive number. `para2fan` assumes that the center of rotation is the center point of the projections, which is defined as `ceil(size(F,1)/2)`. The value of `D` must be greater than or equal to `ParallelSensorSpacing*(size(P,1)-1)/2`. The figure illustrates `D` in relation to the fan-beam vertex for one fan-beam projection.



Data Types: `double` | `single`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `F = para2fan(P,D,FanRotationIncrement=5)` specifies a fan rotation increment of 5 degrees.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `F = para2fan(P,D,"FanRotationIncrement",5)` specifies a fan rotation increment of 5 degrees.

`FanCoverage` — Range of fan-beam rotation
`"cycle"` (default) | `"minimal"`

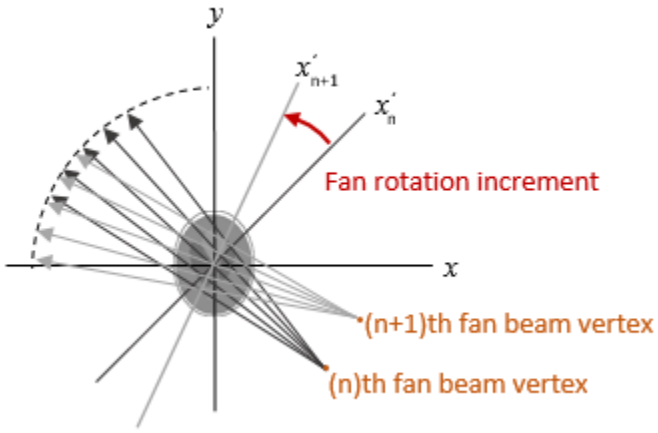
Range of fan-beam rotation, specified as `"cycle"` or `"minimal"`.

- `"cycle"` — Rotate through the full range [0, 360) degrees.
- `"minimal"` — Rotate through the minimum range necessary to represent the object.

`FanRotationIncrement` — Fan-beam rotation angle increment
positive scalar

Fan-beam rotation angle increment in degrees, specified as a positive scalar.

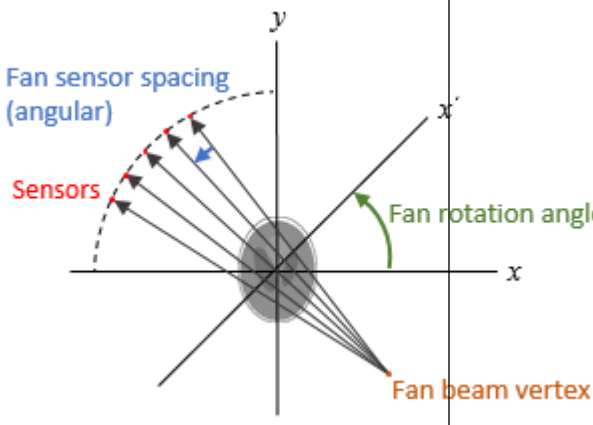
- If `FanCoverage` is `"cycle"`, then `360/FanRotationIncrement` must be an integer.
- If you do not specify `FanRotationIncrement`, then the default value is equal to the parallel-beam rotation angle.

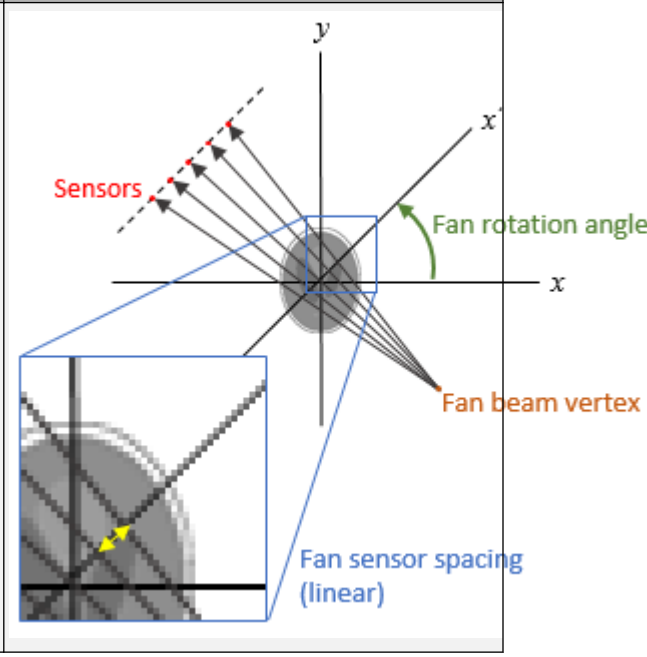


Data Types: `double`

`FanSensorGeometry` — Fan-beam sensor positioning
`"arc"` (default) | `"line"`

Fan-beam sensor positioning, specified as `"arc"` or `"line"`.

Value	Meaning	Diagram
`"arc"`	Sensors are spaced at equal angles along a circular arc. The center of the arc is the fan-beam vertex. `FanSensorSpacing` defines the angular spacing in degrees.	

Value	Meaning	Diagram
`"line"`	Sensors are spaced at equal distances along a line that is parallel to the x' axis. The closest sensor is distance `D` from the center of rotation. `FanSensorSpacing` defines the distance between fan-beams on the x' axis, in pixels.	

`FanSensorSpacing` — Fan-beam sensor spacing
positive scalar

Fan-beam sensor spacing, specified as a positive scalar.

- If `FanSensorGeometry` is `"arc"`, then `FanSensorSpacing` defines the angular spacing in degrees.
- If `FanSensorGeometry` is `"line"`, then `FanSensorSpacing` defines the linear distance between fan-beams, in pixels. Linear spacing is measured on the x' axis.

If you do not specify `FanSensorGeometry`, then the default value of `FanSensorSpacing` is the smallest value implied by `ParallelSensorSpacing` such that:

- If `FanSensorGeometry` is `"arc"`, then `FanSensorSpacing` is `180/pi*asin(ParallelSensorSpacing/D)`
- If `FanSensorGeometry` is `"line"`, then `FanSensorSpacing` is `D*asin(ParallelSensorSpacing/D)`

Data Types: `double`

`Interpolation` — Type of interpolation
`"Linear"` (default) | `"nearest"` | `"spline"` | `"pchip"`

Type of interpolation used between the parallel-beam and fan-beam data, specified as one of these values.

`"nearest"` — Nearest-neighbor

`"linear"` — Linear (the default)

`"spline"` — Piecewise cubic spline

`"pchip"` — Piecewise cubic Hermite (PCHIP)

`ParallelCoverage` — Range of parallel-beam rotation
`"halfcycle"` (default) | `"cycle"`

Range of parallel-beam rotation, specified as `"halfcycle"` or `"cycle"`.

- `"cycle"` — Parallel data covers the full range of [0, 360) degrees.
- `"halfcycle"` — Parallel data covers [0, 180) degrees.

`ParallelSensorSpacing` — Parallel-beam sensor spacing
1 | positive scalar

Parallel-beam sensor spacing in pixels, specified as a positive scalar.

Data Types: `double`

Output Arguments

`F` — Fan-beam projection data
numeric matrix

Fan-beam projection data, returned as a numeric matrix. Each column of `F` contains the fan-beam sensor samples at one rotation angle.

Parallel-beam projection data, returned as a numeric matrix. Each column of `F` contains the fan-beam data at one rotation angle. The number of columns indicates the total number of fan-beam rotation angles and is equal to the length of `fanRotAngles`. The number of rows indicates the total number of parallel-beam sensors and is equal to the length of `fanSensorPos`.

Data Types: `double`

`fanSensorPos` — Fan-beam sensor locations
numeric column vector

Fan-beam sensor locations, returned as a numeric column vector.

- If `FanSensorGeometry` is `"arc"` (the default), then `fanSensorPos` contains the fan-beam sensor measurement angles.
- If `FanSensorGeometry` is `"line"`, then `fanSensorPos` contains the fan-beam sensor positions along the line of sensors.

Data Types: `double`

`fanRotAngles` — Fan-beam rotation angles
numeric row vector

Fan-beam rotation angles, returned as a numeric row vector.

Data Types: `double`

See Also
`fan2para` | `fanbeam` | `iradon` | `ifanbeam` | `phantom` | `radon`

Introduced before R2006a

patchGANDiscriminator

Create PatchGAN discriminator network

Syntax

```
net = patchGANDiscriminator(inputSize)
net = patchGANDiscriminator(inputSize,Name,Value)
```

Description

`net = patchGANDiscriminator(inputSize)` creates a PatchGAN discriminator network for input of size `inputSize`. For more information about the PatchGAN network architecture, see “PatchGAN Discriminator Network” on page 1-2525.

This function requires Deep Learning Toolbox.

`net = patchGANDiscriminator(inputSize,Name,Value)` controls properties of the PatchGAN network using name-value arguments.

You can create a 1-by-1 PatchGAN discriminator network, called a pixel discriminator network, by specifying the '`NetworkType`' argument as "`pixel`". For more information about the pixel discriminator network architecture, see “Pixel Discriminator Network” on page 1-2526.

Examples

Create PatchGAN Discriminator for Color Images

Specify the input size of the network for a color image of size 256-by-256 pixels.

```
inputSize = [256 256 3];
```

Create the PatchGAN discriminator network with the specified input size.

```
net = patchGANDiscriminator(inputSize)
```

```
net =
  dlnetwork with properties:

         Layers: [13x1 nnet.cnn.layer.Layer]
    Connections: [12x2 table]
     Learnables: [16x3 table]
          State: [6x3 table]
     InputNames: {'input_top'}
    OutputNames: {'conv2d_final'}
    Initialized: 1
```

Display the network.

```
analyzeNetwork(net)
```

Create Pixel Discriminator

Specify the input size of the network for a color image of size 256-by-256 pixels.

```
inputSize = [256 256 3];
```

Create the pixel discriminator network with the specified input size.

```
net = patchGANDiscriminator(inputSize,"NetworkType","pixel")
```

```
net =
  dlnetwork with properties:

         Layers: [7x1 nnet.cnn.layer.Layer]
    Connections: [6x2 table]
     Learnables: [8x3 table]
          State: [2x3 table]
     InputNames: {'input_top'}
    OutputNames: {'conv2d_final'}
    Initialized: 1
```

Display the network.

```
analyzeNetwork(net)
```

Input Arguments

`inputSize` — Network input size

3-element vector of positive integers

Network input size, specified as a 3-element vector of positive integers. `inputSize` has the form [*H W C*], where *H* is the height, *W* is the width, and *C* is the number of channels. If the input to the discriminator is a channel-wise concatenated `dlarray` object, then *C* must be the concatenated size.

Example: `[28 28 3]` specifies an input size of 28-by-28 pixels for a 3-channel image.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'FilterSize',5` creates a discriminator whose convolution layers have a filter of size 5-by-5 pixels

`NetworkType` — Type of discriminator network

`"patch"` (default) | `"pixel"`

Type of discriminator network, specified as one of these values.

- `"patch"` – Create a PatchGAN discriminator

- `"pixel"` – Create a pixel discriminator, which is a 1-by-1 PatchGAN discriminator

Data Types: `char` | `string`

`NumDownsamplingBlocks` — Number of downsampling blocks
3 (default) | positive integer

Number of downsampling operations of the network, specified as a positive integer. The discriminator network downsamples the input by a factor of `2^NumDownsamplingBlocks`. This argument is ignored when you specify '`NetworkType`' as `"pixel"`.

`NumFiltersInFirstBlock` — Number of output channels in first discriminator block
64 (default) | positive integer

Number of filters in the first discriminator block, specified as a positive integer.

`FilterSize` — Filter size of convolution layers
4 (default) | positive integer | 2-element vector of positive integers

Filter size of convolution layers, specified as a positive integer or 2-element vector of positive integers of the form [*height width*]. When you specify the filter size as a scalar, the filter has equal height and width. Typical filters have height and width between 1 and 4. This argument has an effect only when you specify '`NetworkType`' as `"patch"`.

`ConvolutionPaddingValue` — Style of padding
`0` (default) | numeric scalar | `"replicate"` | `"symmetric-include-edge"` | `"symmetric-exclude-edge"`

Style of padding used in the network, specified as one of these values.

PaddingValue	**Description**	**Example**
Numeric scalar	Pad with the specified numeric value	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}2&2&2&2&2&2&2\\2&2&2&2&2&2&2\\2&2&3&1&4&2&2\\2&2&1&5&9&2&2\\2&2&2&6&5&2&2\\2&2&2&2&2&2&2\\2&2&2&2&2&2&2\end{bmatrix}$
`'symmetric-include-edge'`	Pad using mirrored values of the input, including the edge values	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}5&1&1&5&9&9&5\\1&3&3&1&4&4&1\\1&3&3&1&4&4&1\\5&1&1&5&9&9&5\\6&2&2&6&5&5&6\\6&2&2&6&5&5&6\\5&1&1&5&9&9&5\end{bmatrix}$

PaddingValue	Description	Example
`'symmetric-exclude-edge'`	Pad using mirrored values of the input, excluding the edge values	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}5&6&2&6&5&6&2\\9&5&1&5&9&5&1\\4&1&3&1&4&1&3\\9&5&1&5&9&5&1\\5&6&2&6&5&6&2\\9&5&1&5&9&5&1\\4&1&3&1&4&1&3\end{bmatrix}$
`'replicate'`	Pad using repeated border elements of the input	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}3&3&3&1&4&4&4\\3&3&3&1&4&4&4\\3&3&3&1&4&4&4\\1&1&1&5&9&9&9\\2&2&2&6&5&5&5\\2&2&2&6&5&5&5\\2&2&2&6&5&5&5\end{bmatrix}$

`ConvolutionWeightsInitializer` — Weight initialization used in convolution layers
`"glorot"` (default) | `"he"` | `"narrow-normal"` | function

Weight initialization used in convolution layers, specified as `"glorot"`, `"he"`, `"narrow-normal"`, or a function handle. For more information, see “Specify Custom Weight Initialization Function” (Deep Learning Toolbox).

`ActivationLayer` — Activation function
`"leakyRelu"` (default) | `"relu"` | `"elu"` | layer object

Activation function to use in the network, specified as one of these values. For more information and a list of available layers, see “Activation Layers” (Deep Learning Toolbox).

- `"relu"` — Use a `reluLayer`
- `"leakyRelu"` — Use a `leakyReluLayer` with a scale factor of 0.2
- `"elu"` — Use an `eluLayer`
- A layer object

`FinalActivationLayer` — Activation function after final convolution
`"none"` (default) | `"sigmoid"` | `"softmax"` | `"tanh"` | layer object

Activation function after the final convolution layer, specified as one of these values. For more information and a list of available layers, see “Output Layers” (Deep Learning Toolbox).

- `"tanh"` — Use a `tanhLayer`
- `"sigmoid"` — Use a `sigmoidLayer`
- `"softmax"` — Use a `softmaxLayer`
- `"none"` — Do not use a final activation layer
- A layer object

`NormalizationLayer` — Normalization operation
`"batch"` (default) | `"none"` | `"instance"` | layer object

Normalization operation to use after each convolution, specified as one of these values. For more information and a list of available layers, see “Normalization, Dropout, and Cropping Layers” (Deep Learning Toolbox).

- `"instance"` — Use an `instanceNormalizationLayer`
- `"batch"` — Use a `batchNormalizationLayer`
- `"none"` — Do not use a normalization layer
- A layer object

`NamePrefix` — Prefix to all layer names
`""` (default) | string | character vector

Prefix to all layer names in the network, specified as a string or character vector.

Data Types: `char` | `string`

Output Arguments

`net` — PatchGAN discriminator network
`dlnetwork` object

PatchGAN discriminator network, returned as a `dlnetwork` object.

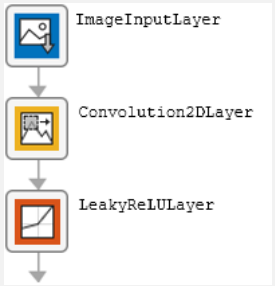
More About

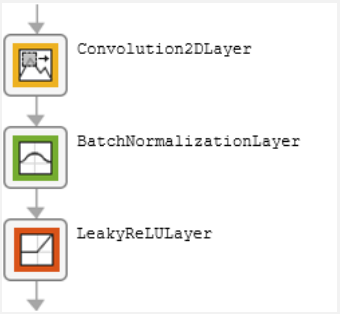
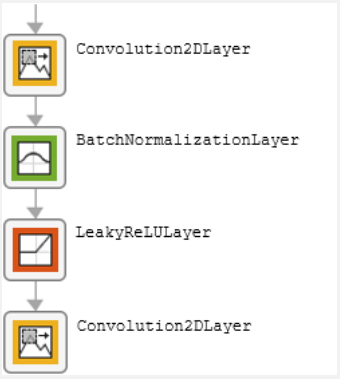
PatchGAN Discriminator Network

A PatchGAN discriminator network consists of an encoder module that downsamples the input by a factor of `2^NumDownsamplingBlocks`. The default network follows the architecture proposed by Zhu et. al. [2].

The encoder module consists of an initial block of layers that performs one downsampling operation, `NumDownsamplingBlocks`–1 downsampling blocks, and a final block.

The table describes the blocks of layers that comprise the encoder module.

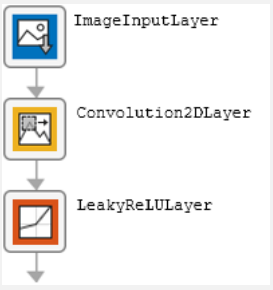
Block Type	Layers	Diagram of Default Block
Initial block	• An `imageInputLayer` • A `convolution2dLayer` with a stride of [2 2] that performs downsampling • An activation layer specified by the `ActivationLayer` name-value argument	

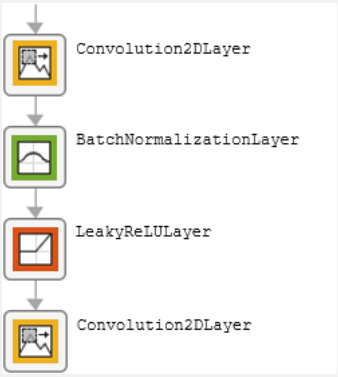
Block Type	Layers	Diagram of Default Block
Downsampling block	• A `convolution2dLayer` with a stride of [2 2] to perform downsampling • An optional normalization layer, specified by the `NormalizationLayer` name-value argument • An activation layer specified by the `ActivationLayer` name-value argument	
Final block	• A `convolution2dLayer` with a stride of [1 1] • An optional normalization layer, specified by the `NormalizationLayer` name-value argument • An activation layer specified by the `ActivationLayer` name-value argument • A second `convolution2dLayer` with a stride of [1 1] and 1 output channel • An optional activation layer specified by the `FinalActivationLayer` name-value argument	

Pixel Discriminator Network

A pixel discriminator network consists of an initial block and final block that return an output of size [*H W C*]. This network does not perform downsampling. The default network follows the architecture proposed by Zhu et. al. [2].

The table describes the blocks of layers that comprise the network.

Block Type	Layers	Diagram of Default Block
Initial block	• An `imageInputLayer` • A `convolution2dLayer` with a stride of [1 1] • An activation layer specified by the `ActivationLayer` name-value argument	

Block Type	Layers	Diagram of Default Block
Final block	• A `convolution2dLayer` with a stride of [1 1] • An optional normalization layer, specified by the `NormalizationLayer` name-value argument • An activation layer specified by the `ActivationLayer` name-value argument • A second `convolution2dLayer` with a stride of [1 1] and 1 output channel • An optional activation layer specified by the `FinalActivationLayer` name-value argument	

References

[1] Isola, Phillip, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. "Image-to-Image Translation with Conditional Adversarial Networks." In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5967–76. Honolulu, HI: IEEE, 2017. https://arxiv.org/abs/1611.07004.

[2] Zhu, Jun-Yan, Taesung Park, and Tongzhou Wang. "CycleGAN and pix2pix in PyTorch." https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix.

See Also

`cycleGANGenerator`

Topics

“Unsupervised Day-to-Dusk Image Translation Using UNIT”
“Get Started with GANs for Image-to-Image Translation”
“Create Modular Neural Networks”
“List of Deep Learning Layers” (Deep Learning Toolbox)

Introduced in R2021a

phantom

Create head phantom image

Syntax

```
P = phantom(def,n)
P = phantom(E,n)
[P,E] = phantom( ___ )
```

Description

`P = phantom(def,n)` generates an image of a head phantom that can be used to test the numerical accuracy of `radon` and `iradon` or other two-dimensional reconstruction algorithms. `P` is a grayscale image that consists of one large ellipse (representing the brain) containing several smaller ellipses (representing features in the brain). `def` specifies the type of head phantom to generate, and `n` specifies the number of rows and columns in the phantom image.

`P = phantom(E,n)` generates a user-defined phantom, where each row of the matrix `E` specifies an ellipse in the image. `E` has six columns, with each column containing a different parameter for the ellipses.

`[P,E] = phantom( ___ )` returns the matrix `E` used to generate the phantom.

Examples

Create Modified Shepp-Logan Head Phantom Image

Create the modified Shepp-Logan head phantom image and display it.

```
P = phantom('Modified Shepp-Logan',200);
imshow(P)
```

Input Arguments

def — Type of head phantom

`'Modified Shepp-Logan'` (default) | `'Shepp-Logan'`

Type of head phantom to generate, specified as one of the following.

- `'Shepp-Logan'` — Test image used widely by researchers in tomography
- `'Modified Shepp-Logan'` — Variant of the Shepp-Logan phantom in which the contrast is improved for better visual perception

Data Types: `char` | `string`

n — Number of rows and columns

`256` (default) | positive integer

Number of rows and columns in the phantom image, specified as a positive integer.

Data Types: `double`

E — Ellipses

e-by-6 numeric matrix

Ellipses that define the phantom, specified as an *e*-by-6 numeric matrix defining *e* ellipses. The six columns of `E` are the ellipse parameters.

Column	Parameter	Meaning
Column 1	`A`	Additive intensity value of the ellipse
Column 2	`a`	Length of the horizontal semiaxis of the ellipse
Column 3	`b`	Length of the vertical semiaxis of the ellipse
Column 4	`x0`	*x*-coordinate of the center of the ellipse

Column	Parameter	Meaning
Column 5	`y0`	*y*-coordinate of the center of the ellipse
Column 6	`phi`	Angle (in degrees) between the horizontal semiaxis of the ellipse and the *x*-axis of the image

The domains for the *x*- and *y*-axes are [-1,1]. Columns 2 through 5 must be specified in terms of this range.

Data Types: `double`

Output Arguments

`P` — Phantom image

`n`-by-`n` numeric matrix

Phantom image, returned as an `n`-by-`n` numeric matrix.

Data Types: `double`

Tips

For any given pixel in the output image, the pixel's value is equal to the sum of the additive intensity values of all ellipses that the pixel is a part of. If a pixel is not part of any ellipse, its value is 0.

The additive intensity value `A` for an ellipse can be positive or negative; if it is negative, the ellipse will be darker than the surrounding pixels. Note that, depending on the values of `A`, some pixels can have values outside the range [0, 1].

References

[1] Jain, Anil K., *Fundamentals of Digital Image Processing*, Englewood Cliffs, NJ, Prentice Hall, 1989, p. 439.

See Also

`radon` | `iradon`

Introduced before R2006a

piqe

Perception based Image Quality Evaluator (PIQE) no-reference image quality score

Syntax

```
score = piqe(A)
[score,activityMask,noticeableArtifactsMask,noiseMask] = piqe(A)
```

Description

`score = piqe(A)` calculates the no-reference image quality score for image `A` using a perception based image quality evaluator. A smaller score indicates better perceptual quality.

`[score,activityMask,noticeableArtifactsMask,noiseMask] = piqe(A)` also returns the spatial quality masks computed from the input image.

Examples

Calculate PIQE Score for Images and Display Results

Calculate PIQE score for an image and the corresponding distorted images. Display the results with their corresponding image.

Read an image into the workspace. Generate distorted images by adding noise and blur. Use `imnoise` function to generate the noisy image and `imgaussfilt` function to generate the blurred image.

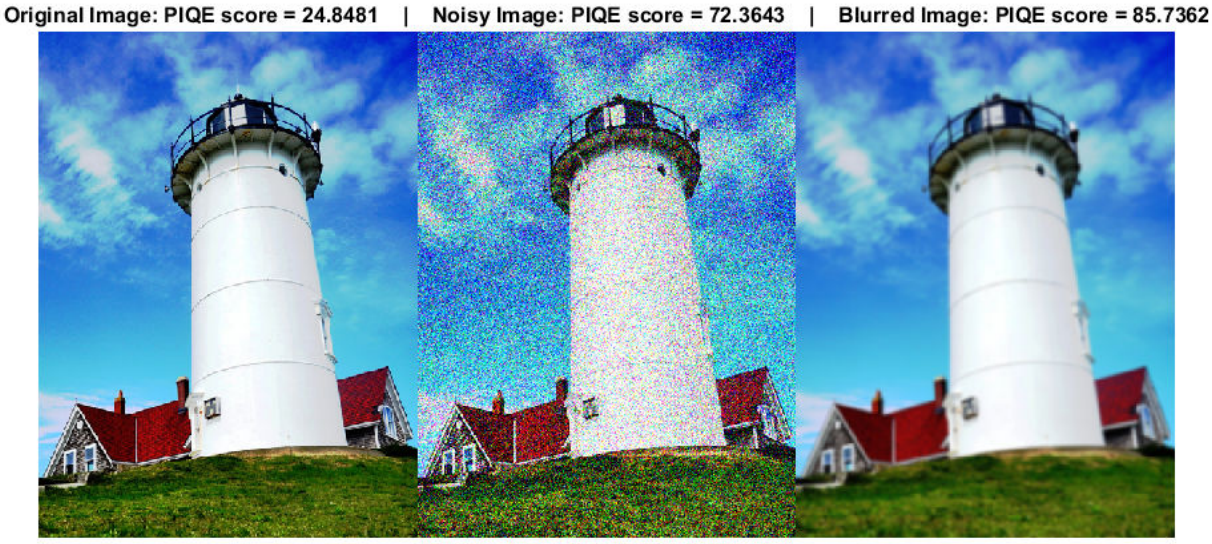
```
A = imread('lighthouse.png');
Anoise = imnoise(A,'Gaussian',0,0.05);
Ablur = imgaussfilt(A,2);
```

Calculate PIQE score for the original image and the distorted images.

```
score = piqe(A);
score_noise = piqe(Anoise);
score_blur = piqe(Ablur);
```

Display the images as a montage with their corresponding scores as a part of the figure title.

```
figure
montage({A,Anoise,Ablur},'Size',[1 3])
title({['Original Image: PIQE score = ', num2str(score),'    |    Noisy Image: PIQE score = ', nu
    '|    Blurred Image: PIQE score = ', num2str(score_blur)]}, 'FontSize',12)
```



Calculate PIQE Score, Spatial Quality Masks for Image, and Display Results

Calculate PIQE score of an image distorted due to blocking artifacts and Gaussian noise. Generate spatial quality masks that indicate the high spatially active blocks, noticeable artifacts blocks, and noise blocks in the image. Visualize the spatial quality masks by overlaying them on the distorted image. Display the image with and without the masks and the PIQE score for the image.

Read a distorted image (distortion due to JPEG2K) into the workspace.

```
Adistorted = imread('DistortedImage.png');
```

Calculate PIQE score and the spatial quality masks.

```
[score,activityMask,noticeableArtifactsMask,noiseMask] = piqe(Adistorted);
```

Overlay the spatial quality masks on the input image.

```
mask_1 = labeloverlay(Adistorted,activityMask,'Colormap','winter','Transparency',0.25);
mask_2 = labeloverlay(Adistorted,noticeableArtifactsMask,'Colormap','autumn','Transparency',0.25
mask_3 = labeloverlay(Adistorted,noiseMask,'Colormap','hot','Transparency',0.25);
```

Display the original distorted image and the distorted images with overlaid spatial quality masks as a montage.

```
figure
montage({Adistorted,mask_1,mask_2,mask_3},'Size',[1 4])
title('Distorted Image   |    Overlay activityMask    |    Overlay noticeableArtifactsMask    |
```

Display PIQE score for the distorted image.

```
fprintf('PIQE score for the distorted image is %0.4f.\n',score)
```

```
PIQE score for the distorted image is 65.1855.
```

Input Arguments

A — Input image

2-D grayscale image | 2-D RGB image

Input image, specified as a 2-D grayscale image of size *m*-by-*n* or 2-D RGB image of size *m*-by-*n*-by-3.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

Output Arguments

score — PIQE score

nonnegative scalar

PIQE score for the input image `A`, returned as a nonnegative scalar in the range [0, 100]. The PIQE score is the no-reference image quality score and it is inversely correlated to the perceptual quality of an image. A low `score` value indicates high perceptual quality and high `score` value indicates low perceptual quality.

Data Types: `double`

activityMask — Spatial quality mask of active blocks

2-D Binary image

Spatial quality mask of active blocks, returned as a 2-D binary image of size *m*-by-*n*, where *m* and *n* are the dimensions of the input image `A`. The `activityMask` is composed of high spatially active blocks in the input image. The high spatially active blocks in the input image are the regions with more spatial variability due to factors that include compression artifacts and noise. The high spatially active blocks are assigned a value `'1'` in the `activityMask`.

Data Types: `logical`

`noticeableArtifactsMask` — Spatial quality mask of noticeable artifacts
2-D Binary image

Spatial quality mask of noticeable artifacts, returned as a 2-D binary image of size *m*-by-*n*, where *m* and *n* are the dimensions of the input image `A`. The `noticeableArtifactsMask` is composed of blocks in `activityMask` that contain blocking artifacts (due to compression) or sudden distortions.

Data Types: `logical`

`noiseMask` — Spatial quality mask of Gaussian noise
2-D Binary image

Spatial quality mask of Gaussian noise, returned as a 2-D binary image of size *m*-by-*n*, where *m* and *n* are the dimensions of the input image `A`. The `noiseMask` is composed of blocks in `activityMask` that contain Gaussian noise.

Data Types: `logical`

Algorithms

`PIQE` calculates the no-reference quality score for an image through block-wise distortion estimation, using these steps:

1. Compute the Mean Subtracted Contrast Normalized (MSCN) coefficient for each pixel in the image using the algorithm proposed by N. Venkatanath and others [1].
2. Divide the input image into nonoverlapping blocks of size 16-by-16.
3. Identify high spatially active blocks based on the variance of the MSCN coefficients.
4. Generate `activityMask` using the identified high spatially active blocks.
5. In each block, evaluate distortion due to blocking artifacts and noise using the MSCN coefficients.
6. Use threshold criteria to classify the blocks as distorted blocks with blocking artifacts, distorted blocks with Gaussian noise, and undistorted blocks.
7. Generate `noticeableArtifactsMask` from the distorted blocks with blocking artifacts and `noiseMask` from the distorted blocks with Gaussian noise.
8. Compute the PIQE score for the input image as the mean of scores in the distorted blocks.
9. The quality scale of the image based on its PIQE score is given in this table. The quality scale and respective score range are assigned through experimental analysis on the dataset in LIVE Image Quality Assessment Database Release 2 [2].

Qualitative Scale	Score Range
Excellent	[0, 20]
Good	[21, 35]
Fair	[36, 50]
Poor	[51, 80]

References

[1] N. Venkatanath, D. Praneeth, Bh. M. Chandrasekhar, S. S. Channappayya, and S. S. Medasani. "Blind Image Quality Evaluation Using Perception Based Features", In *Proceedings of the 21st National Conference on Communications (NCC).* Piscataway, NJ: IEEE, 2015.

[2] Sheikh, H. R., Z. Wang, L. Cormack and A.C. Bovik, "LIVE Image Quality Assessment Database Release 2 ", https://live.ece.utexas.edu/research/quality/.

See Also

Functions
`immse` | `ssim` | `psnr` | `brisque` | `niqe`

Topics
“Image Quality Metrics”

Introduced in R2018b

pix2pixHDGlobalGenerator

Create pix2pixHD global generator network

Syntax

```
net = pix2pixHDGlobalGenerator(inputSize)
net = pix2pixHDGlobalGenerator(inputSize,Name,Value)
```

Description

`net = pix2pixHDGlobalGenerator(inputSize)` creates a pix2pixHD generator network for input of size `inputSize`. For more information about the network architecture, see “pix2pixHD Generator Network” on page 1-2541.

This function requires Deep Learning Toolbox.

`net = pix2pixHDGlobalGenerator(inputSize,Name,Value)` modifies properties of the pix2pixHD network using name-value arguments.

Examples

Create Pix2PixHD Generator

Specify the network input size for 32-channel data of size 512-by-1024 pixels.

```
inputSize = [512 1024 32];
```

Create a pix2pixHD global generator network.

```
net = pix2pixHDGlobalGenerator(inputSize)
```

```
net =
  dlnetwork with properties:

         Layers: [84x1 nnet.cnn.layer.Layer]
    Connections: [92x2 table]
     Learnables: [110x3 table]
          State: [0x3 table]
     InputNames: {'GlobalGenerator_inputLayer'}
    OutputNames: {'GlobalGenerator_fActivation'}
    Initialized: 1
```

Display the network.

```
analyzeNetwork(net)
```

Create Pix2PixHD Generator with Batch Normalization

Specify the network input size for 32-channel data of size 512-by-1024 pixels.

```
inputSize = [512 1024 32];
```

Create a pix2pixHD generator network that performs batch normalization after each convolution.

```
net = pix2pixHDGlobalGenerator(inputSize,"Normalization","batch")
```

```
net =
  dlnetwork with properties:

         Layers: [84x1 nnet.cnn.layer.Layer]
    Connections: [92x2 table]
     Learnables: [110x3 table]
          State: [54x3 table]
     InputNames: {'GlobalGenerator_inputLayer'}
    OutputNames: {'GlobalGenerator_fActivation'}
    Initialized: 1
```

Display the network.

```
analyzeNetwork(net)
```

Input Arguments

`inputSize` — Network input size

3-element vector of positive integers

Network input size, specified as a 3-element vector of positive integers. `inputSize` has the form [*H W C*], where *H* is the height, *W* is the width, and *C* is the number of channels.

Example: `[28 28 3]` specifies an input size of 28-by-28 pixels for a 3-channel image.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'NumFiltersInFirstBlock',32` creates a network with 32 filters in the first convolution layer

`NumDownsamplingBlocks` — Number of downsampling blocks

4 (default) | positive integer

Number of downsampling blocks in the network encoder module, specified as a positive integer. In total, the network downsamples the input by a factor of `2^NumDownsamplingBlocks`. The decoder module consists of the same number of upsampling blocks.

`NumFiltersInFirstBlock` — Number of filters in first convolution layer

64 (default) | positive even integer

Number of filters in the first convolution layer, specified as a positive even integer.

`NumOutputChannels` — Number of output channels
3 (default) | positive integer

Number of output channels, specified as a positive integer.

`FilterSizeInFirstAndLastBlocks` — Filter size in first and last convolution layers
7 (default) | positive odd integer | 2-element vector of positive odd integers

Filter size in the first and last convolution layers of the network, specified as a positive odd integer or 2-element vector of positive odd integers of the form [*height width*]. When you specify the filter size as a scalar, the filter has equal height and width.

`FilterSizeInIntermediateBlocks` — Filter size in intermediate convolution layers
3 (default) | 2-element vector of positive odd integers | positive odd integer

Filter size in intermediate convolution layers, specified as a positive odd integer or 2-element vector of positive odd integers of the form [*height width*]. The intermediate convolution layers are the convolution layers excluding the first and last convolution layer. When you specify the filter size as a scalar, the filter has identical height and width. Typical values are between 3 and 7.

`NumResidualBlocks` — Number of residual blocks
9 (default) | positive integer

Number of residual blocks, specified as a positive integer.

`ConvolutionPaddingValue` — Style of padding
`"symmetric-exclude-edge"` (default) | `"symmetric-include-edge"` | `"replicate"` | numeric scalar

Style of padding used in the network, specified as one of these values.

`PaddingValue`	Description	Example
Numeric scalar	Pad with the specified numeric value	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}2&2&2&2&2&2&2\\2&2&2&2&2&2&2\\2&2&3&1&4&2&2\\2&2&1&5&9&2&2\\2&2&2&6&5&2&2\\2&2&2&2&2&2&2\\2&2&2&2&2&2&2\end{bmatrix}$
`'symmetric-include-edge'`	Pad using mirrored values of the input, including the edge values	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}5&1&1&5&9&9&5\\1&3&3&1&4&4&1\\1&3&3&1&4&4&1\\5&1&1&5&9&9&5\\6&2&2&6&5&5&6\\6&2&2&6&5&5&6\\5&1&1&5&9&9&5\end{bmatrix}$

PaddingValue	Description	Example
`'symmetric-exclude-edge'`	Pad using mirrored values of the input, excluding the edge values	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}5&6&2&6&5&6&2\\9&5&1&5&9&5&1\\4&1&3&1&4&1&3\\9&5&1&5&9&5&1\\5&6&2&6&5&6&2\\9&5&1&5&9&5&1\\4&1&3&1&4&1&3\end{bmatrix}$
`'replicate'`	Pad using repeated border elements of the input	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}3&3&3&1&4&4&4\\3&3&3&1&4&4&4\\3&3&3&1&4&4&4\\1&1&1&5&9&9&9\\2&2&2&6&5&5&5\\2&2&2&6&5&5&5\\2&2&2&6&5&5&5\end{bmatrix}$

`UpsampleMethod` — Method used to upsample activations
`"transposedConv"` (default) | `"bilinearResize"` | `"pixelShuffle"`

Method used to upsample activations, specified as one of these values:

- `"transposedConv"` — Use a `transposedConv2dLayer` with a stride of [2 2]
- `"bilinearResize"` — Use a `convolution2dLayer` with a stride of [1 1] followed by a `resize2dLayer` with a scale of [2 2]
- `"pixelShuffle"` — Use a `convolution2dLayer` with a stride of [1 1] followed by a `depthToSpace2dLayer` with a block size of [2 2]

Data Types: `char` | `string`

`ConvolutionWeightsInitializer` — Weight initialization used in convolution layers
`"narrow-normal"` (default) | `"glorot"` | `"he"` | function

Weight initialization used in convolution layers, specified as `"glorot"`, `"he"`, `"narrow-normal"`, or a function handle. For more information, see “Specify Custom Weight Initialization Function” (Deep Learning Toolbox).

`ActivationLayer` — Activation function
`"relu"` (default) | `"leakyRelu"` | `"elu"` | layer object

Activation function to use in the network, specified as one of these values. For more information and a list of available layers, see “Activation Layers” (Deep Learning Toolbox).

- `"relu"` — Use a `reluLayer`
- `"leakyRelu"` — Use a `leakyReluLayer` with a scale factor of 0.2
- `"elu"` — Use an `eluLayer`
- A layer object

`FinalActivationLayer` — Activation function after final convolution
`"tanh"` (default) | `"sigmoid"` | `"softmax"` | `"none"` | layer object

Activation function after the final convolution layer, specified as one of these values. For more information and a list of available layers, see “Output Layers” (Deep Learning Toolbox).

- `"tanh"` — Use a `tanhLayer`
- `"sigmoid"` — Use a `sigmoidLayer`
- `"softmax"` — Use a `softmaxLayer`
- `"none"` — Do not use a final activation layer
- A layer object

`NormalizationLayer` — Normalization operation
`"instance"` (default) | `"none"` | `"batch"` | layer object

Normalization operation to use after each convolution, specified as one of these values. For more information and a list of available layers, see “Normalization, Dropout, and Cropping Layers” (Deep Learning Toolbox).

- `"instance"` — Use an `instanceNormalizationLayer`
- `"batch"` — Use a `batchNormalizationLayer`
- `"none"` — Do not use a normalization layer
- A layer object

`Dropout` — Probability of dropout
`0` (default) | number in the range [0, 1]

Probability of dropout, specified as a number in the range [0, 1]. If you specify a value of `0`, then the network does not include dropout layers. If you specify a value greater than `0`, then the network includes a `dropoutLayer` in each residual block.

`NamePrefix` — Prefix to all layer names
`"GlobalGenerator_"` (default) | string | character vector

Prefix to all layer names in the network, specified as a string or character vector.

Data Types: `char` | `string`

Output Arguments

`net` — pix2pixHD generator network
`dlnetwork` object

Pix2pixHD generator network, returned as a `dlnetwork` object.

More About

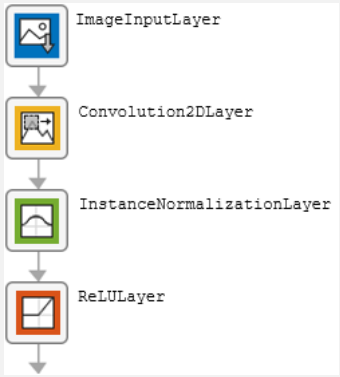
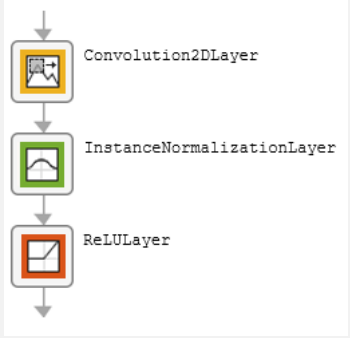
pix2pixHD Generator Network

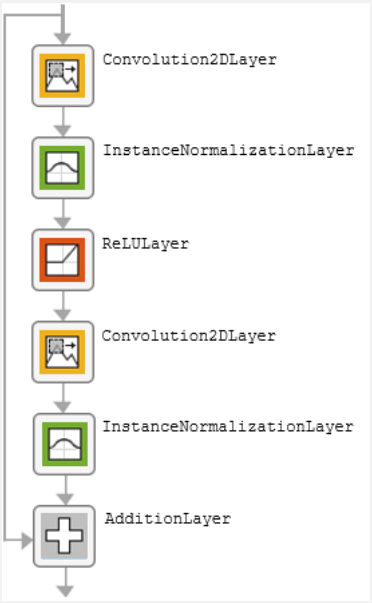
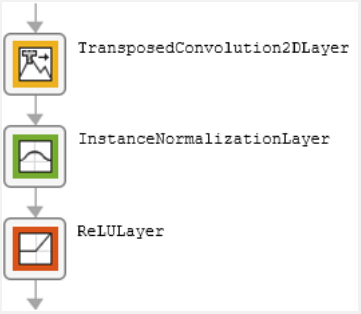
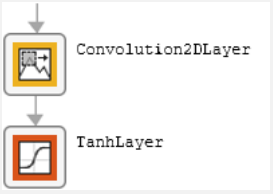
A pix2pixHD generator network consists of an encoder module followed by a decoder module. The default network follows the architecture proposed by Wang et. al. [1].

The encoder module downsamples the input by a factor of `2^NumDownsamplingBlocks`. The encoder module consists of an initial block of layers, `NumDownsamplingBlocks` downsampling blocks, and `NumResidualBlocks` residual blocks. The decoder module upsamples the input by a

factor of 2^`NumDownsamplingBlocks`. The decoder module consists of `NumDownsamplingBlocks` upsampling blocks and a final block.

The table describes the blocks of layers that comprise the encoder and decoder modules.

Block Type	Layers	Diagram of Default Block
Initial block	• An `imageInputLayer` • A `convolution2dLayer` with a stride of [1 1] and a filter size of `FilterSizeInFirstAndLastBlocks` • An optional normalization layer, specified by the `NormalizationLayer` name-value argument. • An activation layer specified by the `ActivationLayer` name-value argument.	
Downsampling block	• A `convolution2dLayer` with a stride of [2 2] to perform downsampling. The convolution layer has a filter size of `FilterSizeInIntermediateBlocks`. • An optional normalization layer, specified by the `NormalizationLayer` name-value argument. • An activation layer specified by the `ActivationLayer` name-value argument.	

Block Type	Layers	Diagram of Default Block
Residual block	• A `convolution2dLayer` with a stride of [1 1] and a filter size of `FilterSizeInIntermediateBlocks`. • An optional normalization layer, specified by the `NormalizationLayer` name-value argument. • An activation layer specified by the `ActivationLayer` name-value argument. • An optional `dropoutLayer`. By default, residual blocks omit a dropout layer. Include a dropout layer by specifying the `Dropout` name-value argument as a value in the range (0, 1]. • A second `convolution2dLayer`. • An optional second normalization layer. • An `additionLayer` that provides a skip connection between every block.	
Upsampling block	• An upsampling layer that upsamples by a factor of 2 according to the `UpsampleMethod` name-value argument. The convolution layer has a filter size of `FilterSizeInIntermediateBlocks`. • An optional normalization layer, specified by the `NormalizationLayer` name-value argument. • An activation layer specified by the `ActivationLayer` name-value argument.	
Final block	• A `convolution2dLayer` with a stride of [1 1] and a filter size of `FilterSizeInFirstAndLastBlocks`. • An optional activation layer specified by the `FinalActivationLayer` name-value argument.	

Tips

- You can create the discriminator network for pix2pixHD by using the `patchGANDiscriminator` function.
- Train the pix2pixHD GAN network using a custom training loop.

References

[1] Wang, Ting-Chun, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. "High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs." In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 8798–8807. Salt Lake City, UT, USA: IEEE, 2018. https://doi.org/10.1109/CVPR.2018.00917.

See Also

`addPix2PixHDLocalEnhancer` | `encoderDecoderNetwork` | `blockedNetwork` | `cycleGANGenerator` | `unitGenerator`

Topics

“Generate Image from Segmentation Map Using Deep Learning” (Computer Vision Toolbox)
“Get Started with GANs for Image-to-Image Translation”
“Create Modular Neural Networks”
“List of Deep Learning Layers” (Deep Learning Toolbox)

Introduced in R2021a

planar2raw

Combine planar sensor images into full Bayer pattern CFA

Syntax

```
cfa = planar2raw(I)
```

Description

`cfa = planar2raw(I)` combines the individual sensor element images, stored as channels of the input image `I`, into a complete Bayer pattern Color Filter Array (CFA) image, `cfa`.

Examples

Combine Sensor Data into Complete CFA Image

Read RAW image data into the workspace.

```
cfa = rawread("colorCheckerTestImage.NEF");
```

Create an image with individual channels for each sensor in the CFA image.

```
rggb = raw2planar(cfa);
```

Convert the image with separate channels for each sensor into a complete CFA image.

```
cfaFull = planar2raw(rggb);
```

Input Arguments

`I` — Image with channel for each sensor element

M-by-*N*-by-4 numeric array

Image with a channel for each sensor element, specified as an *M*-by-*N*-by-4 numeric array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`cfa` — CFA image

(2**M*)-by-(2**N*) numeric matrix

CFA image, returned as a (2**M*)-by-(2**N*) numeric matrix of the same class as `I`.

`planar2raw` constructs the CFA image by placing `I(:,:,1)` starting at `cfa(1,1)`, placing `I(:,:,2)` starting at `cfa(1,2)`, placing `I(:,:,3)` starting at `cfa(2,1)`, and placing `I(:,:,4)` starting at `cfa(2,2)`.

See Also

`raw2planar` | `rawread` | `rawinfo` | `raw2rgb`

Topics

“Implement Digital Camera Processing Pipeline”

Introduced in R2021a

plotChromaticity

Plot color reproduction on chromaticity diagram

Syntax

```
plotChromaticity(colorTable)
plotChromaticity
plotChromaticity( ___ ,Name,Value)
```

Description

`plotChromaticity(colorTable)` plots on a chromaticity diagram the measured and reference colors, `colorTable`, for color patch regions of interest (ROIs) in a test chart.

`plotChromaticity` plots an empty chromaticity diagram.

`plotChromaticity( ___ ,Name,Value)` adjusts aspects of the display using name-value arguments.

Examples

Display Chromaticity Diagram from Color Accuracy Measurements

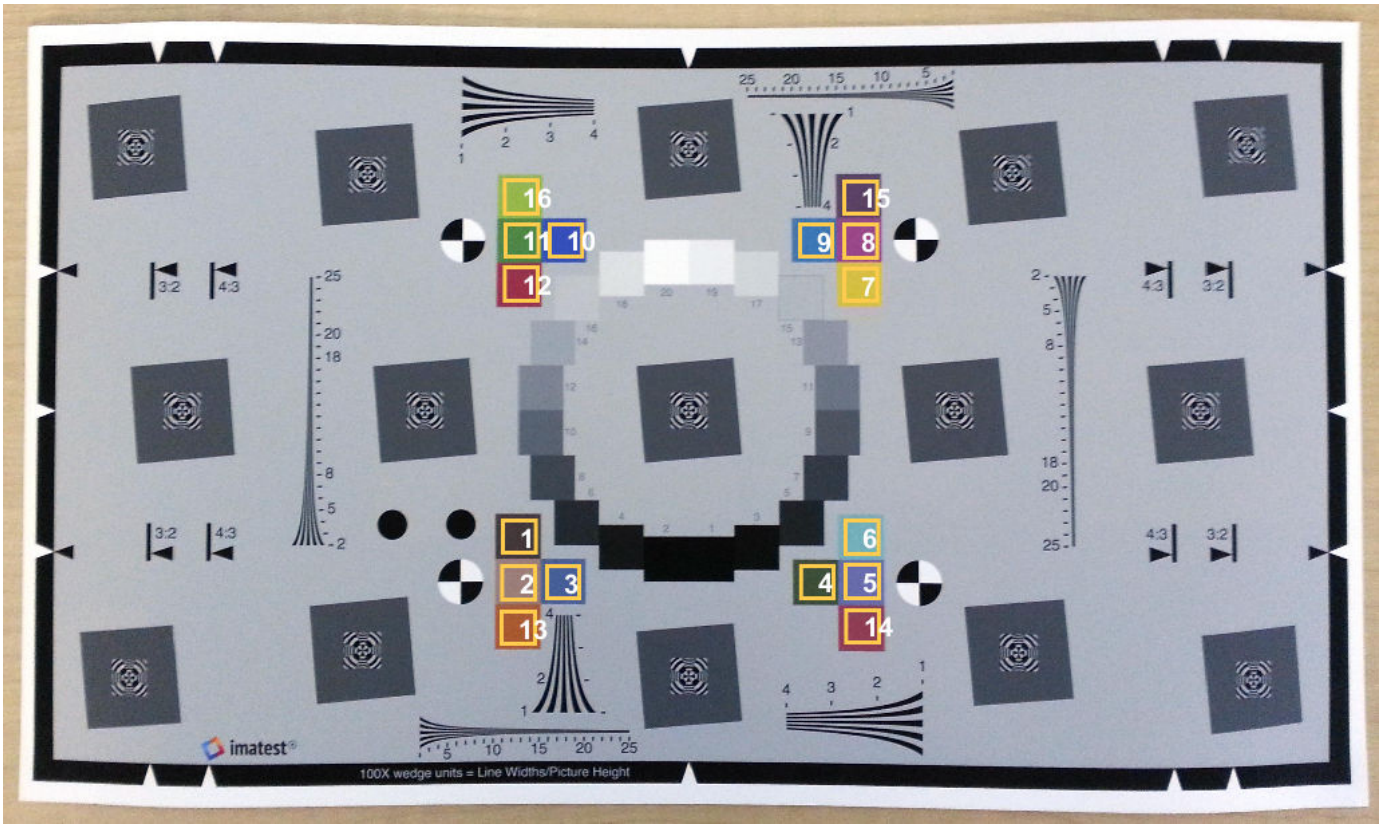
This example shows how to display the chromaticity diagram from measurements of color accuracy on an Imatest® eSFR chart.

Read an image of an eSFR chart into the workspace.

```
I = imread('eSFRTestImage.jpg');
```

Create an `esfrChart` object. Display the chart, highlighting the 16 color patches.

```
chart = esfrChart(I);
displayChart(chart,'displayEdgeROIs',false, ...
    'displayGrayROIs',false,'displayRegistrationPoints',false)
```

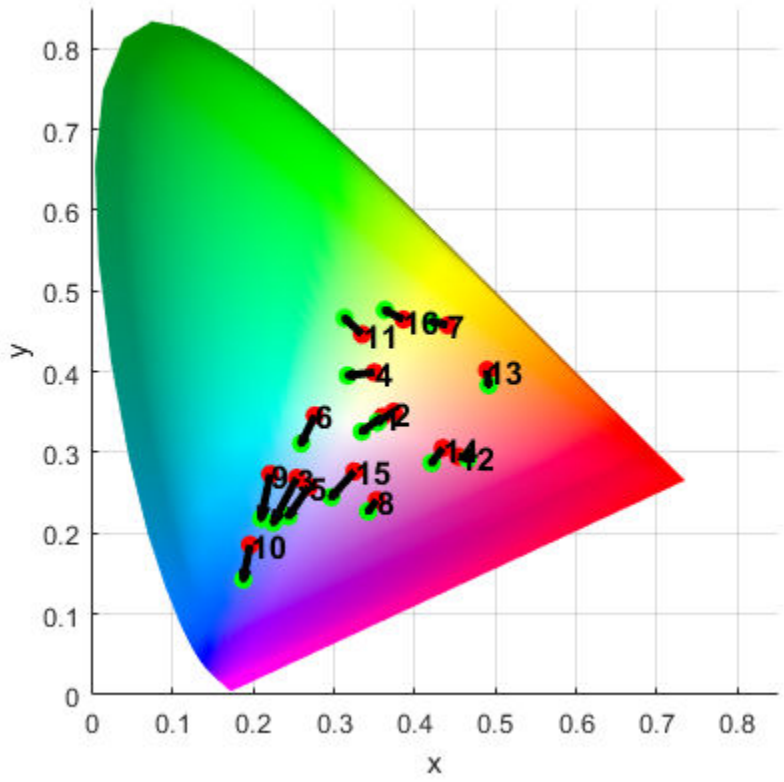


Measure the color in all color patch ROIs.

```
colorTable = measureColor(chart);
```

Plot the measured and reference colors in the CIE 1976 L*a*b* color space on a chromaticity diagram. Red circles indicate the reference color and green circles indicate the measured color of each color patch. The chromaticity diagram does not portray the brightness of color.

```
figure
plotChromaticity(colorTable)
```



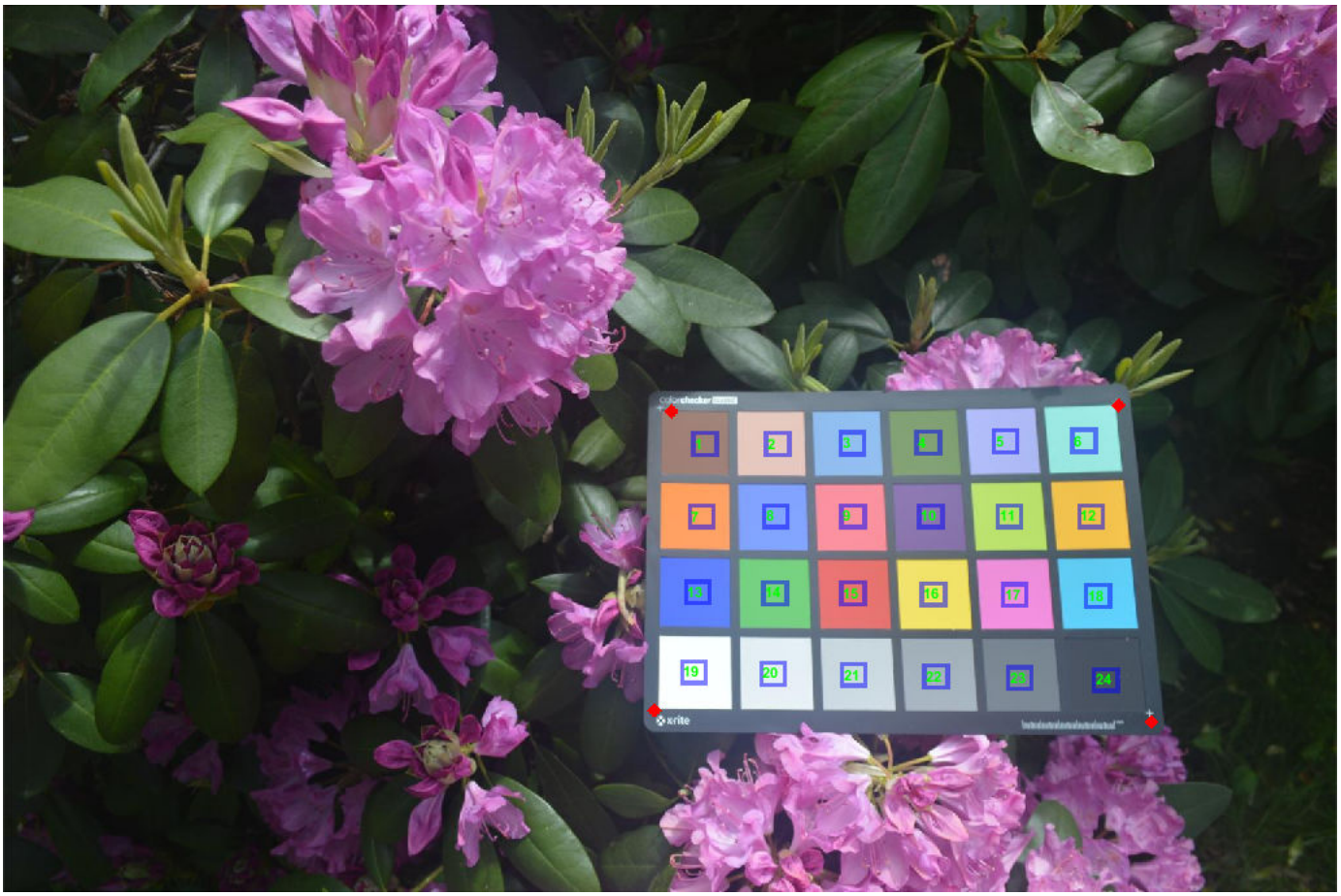
Display Chromaticity Diagram from ColorChecker Chart

Read an image of a ColorChecker® chart into the workspace.

```
I = imread("colorCheckerTestImage.jpg");
```

Create a `colorChecker` object, then display the chart with ROI annotations.

```
chart = colorChecker(I);
displayChart(chart)
```

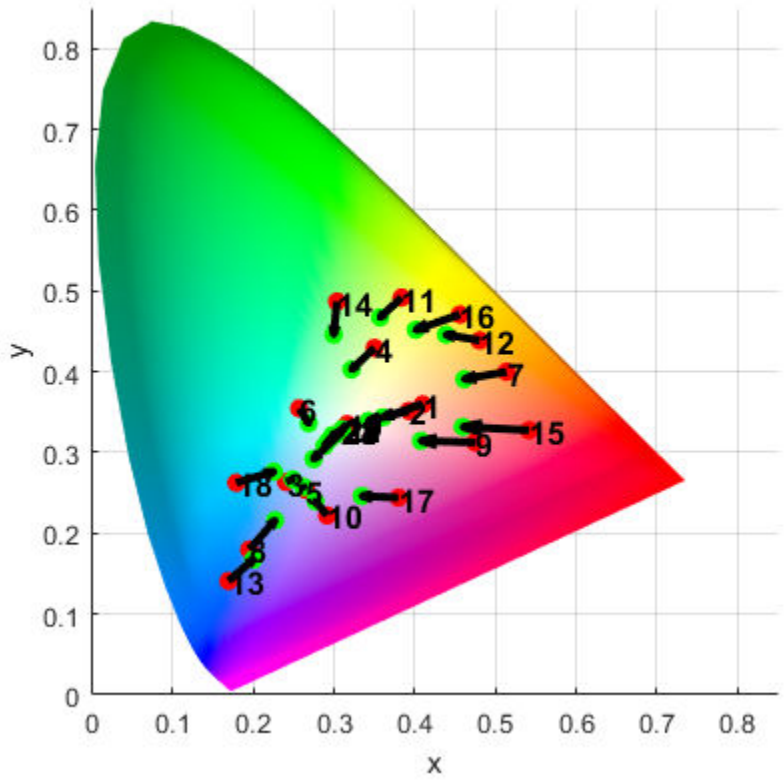


Measure the color in each color patch ROI.

```
colorTable = measureColor(chart);
```

Plot the measured and reference colors on a chromaticity diagram.

```
figure
plotChromaticity(colorTable)
```



Plot sRGB Primaries and White Point on Chromaticity Diagram

Convert sRGB primary colors to the XYZ color space.

```
xyz_primaries = rgb2xyz([1 0 0; 0 1 0; 0 0 1]);
```

Normalize the *x* and *y* values of the primary colors.

```
xyzMag = sum(xyz_primaries,2);
x_primary = xyz_primaries(:,1)./xyzMag;
y_primary = xyz_primaries(:,2)./xyzMag;
```

Calculate and normalize the D65 white point.

```
wp = whitepoint('D65');
```

Normalize the *x* and *y* values of the white point.

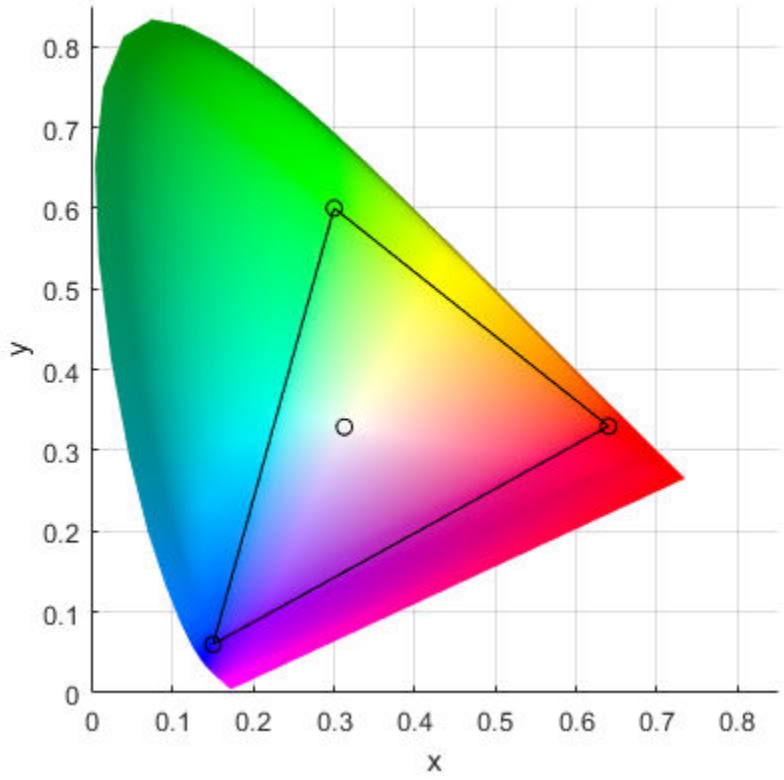
```
wpMag = sum(wp,2);
x_whitepoint = wp(:,1)./wpMag;
y_whitepoint = wp(:,2)./wpMag;
```

Create an empty 2-D chromaticity diagram.

```
plotChromaticity
```

Add the (*x*,*y*) coordinates of the primaries and white point to the chromaticity diagram.

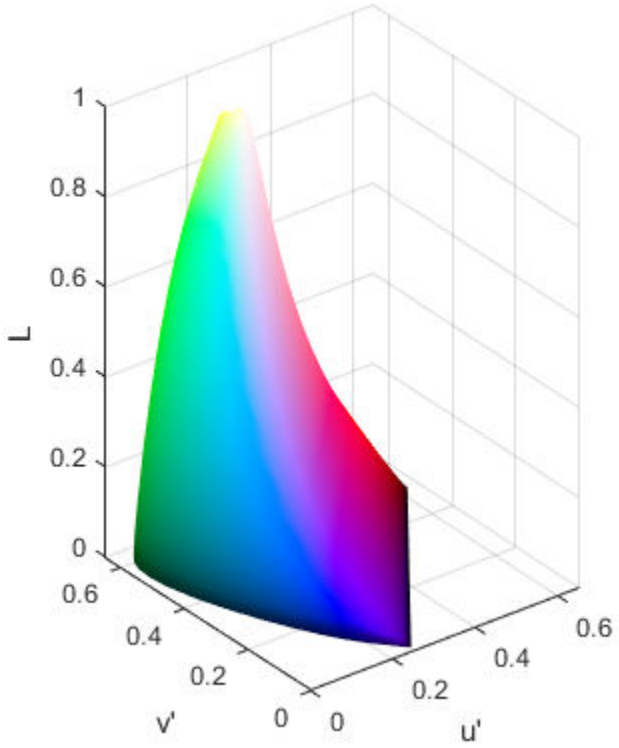
```
hold on
scatter(x_whitepoint,y_whitepoint,36,'black')
scatter(x_primary,y_primary,36,'black')
plot([x_primary; x_primary],[y_primary; y_primary],'k')
hold off
```



Display 3-D Color Solid in u'v'L Color Space

Display a 3-D color solid of the u'v'L color space on an empty chromaticity diagram. Include all u'v'L colors by specifying the brightness threshold as `0`.

```
plotChromaticity("ColorSpace","uv","View",3,"BrightnessThreshold",0)
```



Input Arguments

`colorTable` — Color values

color table

Color values in each color patch, specified as an *m*-by-8 color table, where *m* is the number of patches. The eight columns represent these variables:

Variable	Description
`ROI`	Index of the sampled ROI. The value of `ROI` is an integer in the range [1, 16]. The indices match the ROI numbers displayed by `displayChart`.
`Measured_R`	Mean value of red channel pixels in the ROI. `Measured_R` is a scalar of the same data type as `chart.Image`, which can be of type `single`, `double`, `uint8`, or `uint16`.
`Measured_G`	Mean value of green channel pixels in the ROI. `Measured_G` is a scalar of the same data type as `chart.Image`.
`Measured_B`	Mean value of blue channel pixels in the ROI. `Measured_B` is a scalar of the same data type as `chart.Image`.
`Reference_L`	Reference L* value of the ROI. `Reference_L` is a scalar of type `double`.
`Reference_a`	Reference a* value of the ROI. `Reference_a` is a scalar of type `double`.

Variable	Description
`Reference_b`	Reference b* value of the ROI. `Reference_b` is a scalar of type `double`.
`Delta_E`	Euclidean color distance between the measured and reference color values in the L*a*b* color space, as outlined in CIE 1976. `Delta_E` is a scalar of type `double`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'displayROIIndex',false` turns off the display of the ROI indices on the chromaticity diagram.

`BrightnessThreshold` — Brightness threshold

`0.15` (default) | number in the range [0, 1]

Brightness threshold, specified as the comma-separated pair consisting of `'BrightnessThreshold'` and a number in the range [0, 1]. The `plotChromaticity` function does not display color values with a *Y* or *L* value (depending on the color space) less than the brightness threshold.

`ColorSpace` — Color space

`'xy'` (default) | `'uv'`

Color space, specified as the comma-separated pair consisting of `'ColorSpace'` and `'xy'` to plot in the xyY color space or `'uv'` to plot in the u'v'L color space.

Data Types: `char` | `string`

`displayROIIndex` — Display ROI index labels

`true` or `1` (default) | `false` or `0`

Display ROI index labels, specified as the comma-separated pair consisting of `'displayROIIndex'` and a numeric or logical `1` (`true`) or `0` (`false`). When `displayROIIndex` is `true`, then the `plotChromaticity` function overlays color patch ROI index labels on the chromaticity diagram. The indices match the ROI numbers displayed by the `displayChart` function.

`Parent` — Parent axes

`Axes` object

Parent axes of the chromaticity diagram, specified as the comma-separated pair consisting of `'Parent'` and an `Axes` object.

`View` — Dimensionality of chromaticity diagram

`2` (default) | `3`

Dimensionality of chromaticity diagram, specified as the comma-separated pair consisting of `'View'` and `2` for a 2-D projection or `3` for a 3-D color solid.

Tips

- To obtain a color table of the correct format from an `esfrChart` or `colorChecker` object, use the `measureColor` function. You can also create your own color table containing measured and reference colors for an arbitrary number of color ROIs.
- The reference L*a*b* values of a `colorTable` measured from a `colorChecker` object are for the "After November 2014" version of the ColorChecker chart. The white point of the reference values is the CIE standard illuminant D50.

See Also

Functions
`measureColor` | `displayColorPatch` | `displayChart`

Objects
`esfrChart` | `colorChecker`

Introduced in R2017b

plotSFR

Plot spatial frequency response of edge

Syntax

```
plotSFR(sharpnessMeasurementTable)
plotSFR(sharpnessMeasurementTable,Name,Value)
```

Description

`plotSFR(sharpnessMeasurementTable)` plots the spatial frequency response (SFR) in a sharpness measurement table or aggregate sharpness measurement table.

`plotSFR(sharpnessMeasurementTable,Name,Value)` plots the SFR, specifying additional parameters to control aspects of the display.

Examples

Plot Spatial Frequency Response of Specific ROIs from an eSFR Chart

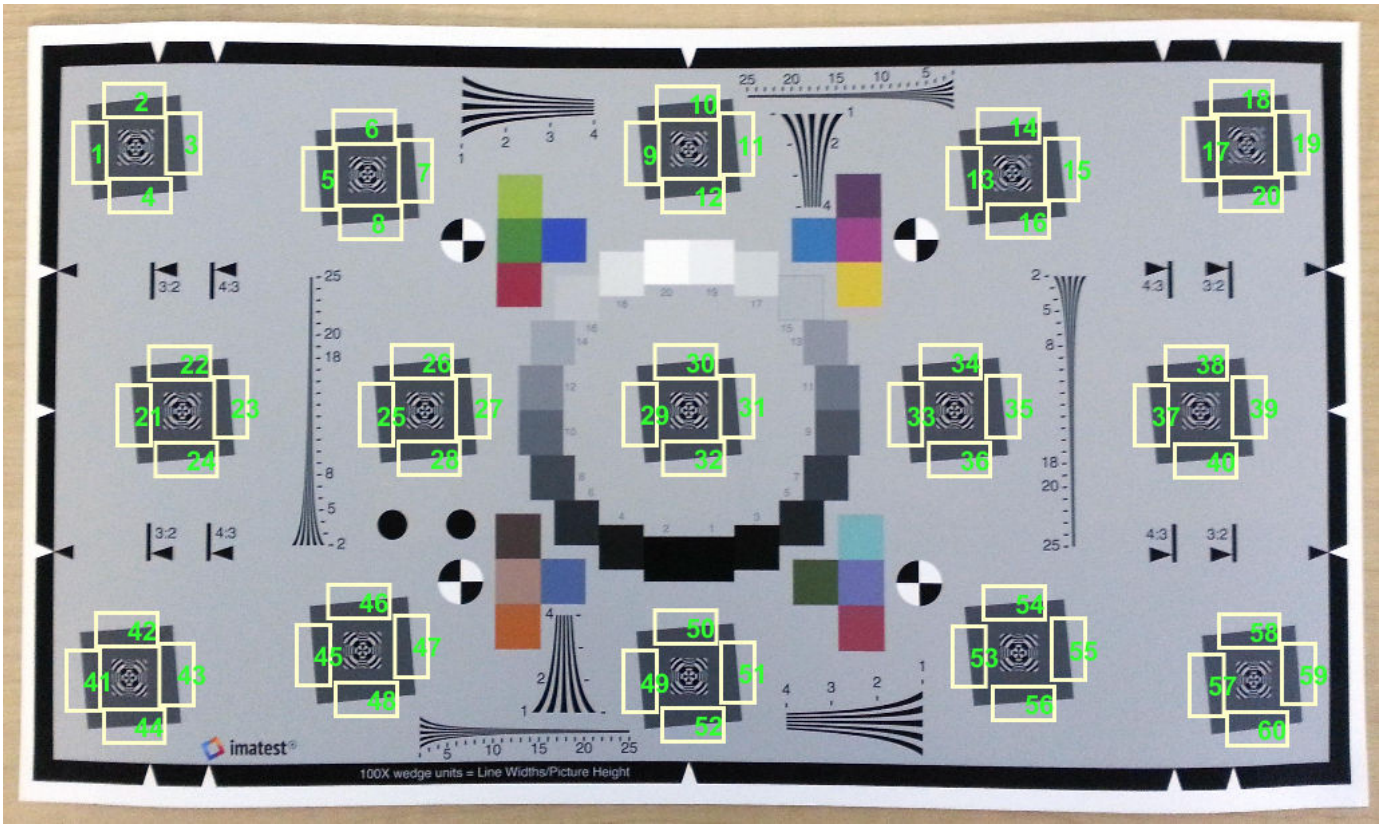
This example shows how to display the spatial frequency response (SFR) plot of a specified subset of the 60 slanted edge ROIs on an Imatest® eSFR chart.

Read an image of an eSFR chart into the workspace.

```
I = imread('eSFRTestImage.jpg');
```

Create an `esfrChart` object, then display the chart with ROI annotations. The 60 slanted edge ROIs are labeled with green numbers.

```
chart = esfrChart(I);
displayChart(chart,'displayGrayROIs',false,...
    'displayColorROIs',false,'displayRegistrationPoints',false)
```

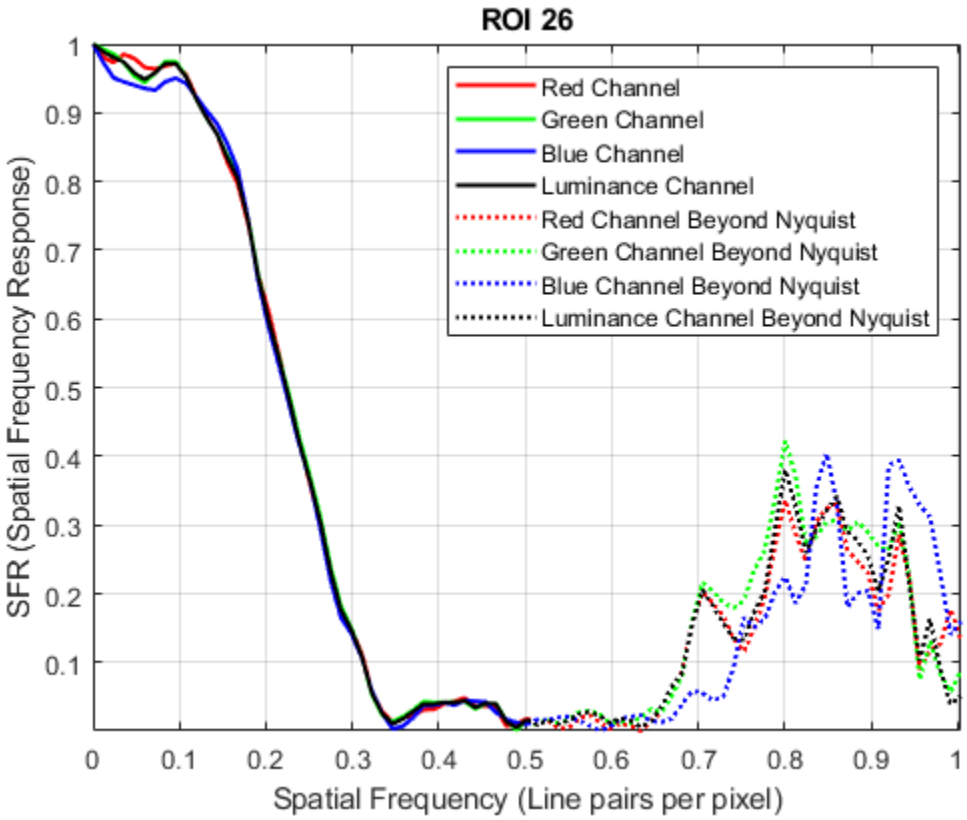


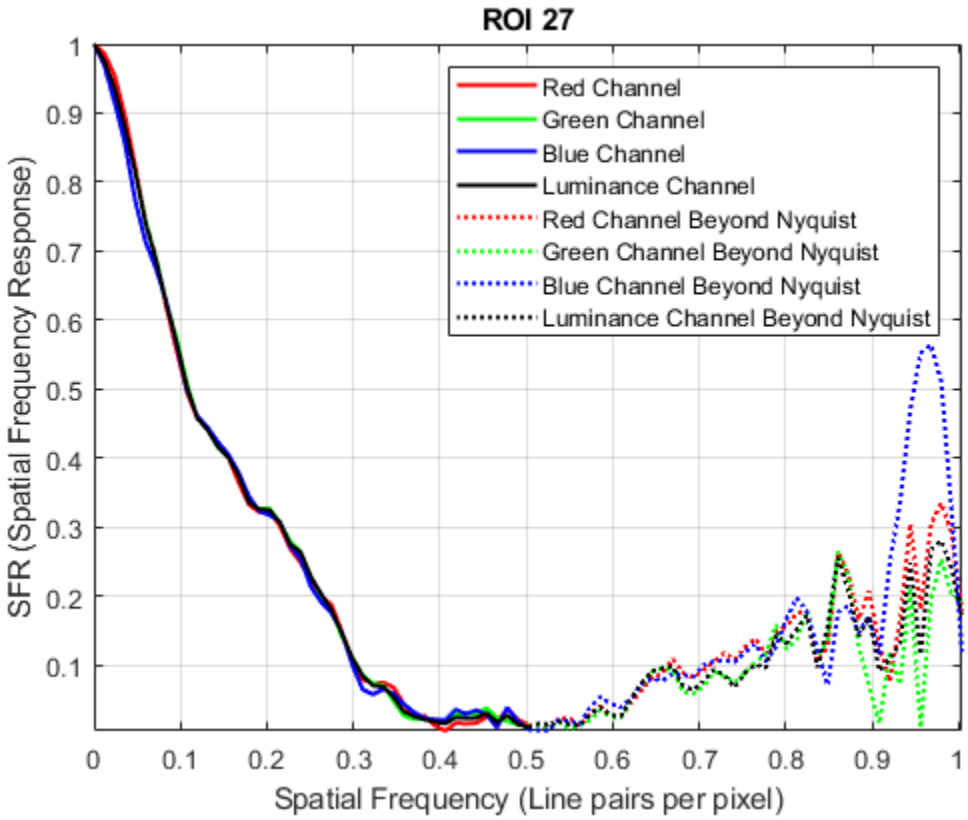
Measure the edge sharpness in all ROIs and return the measurements in `sharpnessTable`.

```
sharpnessTable = measureSharpness(chart);
```

Display the SFR plot of ROIs 26 and 27 only.

```
plotSFR(sharpnessTable,'ROIIndex',[26 27]);
```


ROI 26
Red Channel
Green Channel
Blue Channel
Luminance Channel
Red Channel Beyond Nyquist
Green Channel Beyond Nyquist
Blue Channel Beyond Nyquist
Luminance Channel Beyond Nyquist
SFR (Spatial Frequency Response)
Spatial Frequency (Line pairs per pixel)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



Input Arguments

`sharpnessMeasurementTable` — SFR measurements

sharpness table | aggregate sharpness table

SFR measurements of edges, specified as a sharpness table or aggregate sharpness table with *m* rows:

- When `sharpnessMeasurementTable` is a sharpness table, *m* is the number of sampled ROIs.
- When `sharpnessMeasurementTable` is an aggregate sharpness table, *m* is either 1 or 2, corresponding to the number of sampled orientations.

To obtain a sharpness table or aggregate sharpness table, use the `measureSharpness` function.

Data Types: `table`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `plotSFR(myTable,'ROIIndex',2)` displays the measured sharpness only of ROI 2.

ROIIndex — ROI indices
scalar | vector

ROI indices to display, specified as the comma-separated pair consisting of `'ROIIndex'` and a scalar or vector of integers in the range [1, 60]. The indices match the ROI numbers displayed by `displayChart`.

- When `sharpnessMeasurementTable` is a sharpness table, by default `plotSFR` creates only one figure, showing the SFR plot from the first row of the table.
- When `sharpnessMeasurementTable` is an aggregate sharpness table, `plotSFR` ignores the specified `ROIIndex`, and creates one figure for each row in the table.

Example: `29:32`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

displayLegend — Display plot legend
`true` (default) | `false`

Display plot legend, specified as the comma-separated pair consisting of `'displayLegend'` and `true` or `false`. When `displayLegend` is `true`, the SFR plot shows a legend that identifies the different curves on the plot.

Data Types: `logical`

displayTitle — Display plot title
`true` (default) | `false`

Display plot title, specified as the comma-separated pair consisting of `'displayTitle'` and `true` or `false`. When `displayTitle` is `true`, the SFR plot shows a title that indicates the individual ROI index or aggregate ROI orientation.

Data Types: `logical`

Parent — Axes handle of displayed image object
axes handle

Axes handle of the displayed image object, specified as the comma-separated pair consisting of `'Parent'` and an axes handle. `Parent` specifies the parent of the image object created by `plotSFR`.

See Also

Functions
`measureSharpness` | `displayChart`

Objects
`esfrChart`

Introduced in R2017b

poly2label

Create label matrix from set of ROIs

Syntax

```
L = poly2label(roiPositions,roiLabelIDs,imageSize)
L = poly2label(roiPositions,roiLabelIDs,R)
```

Description

`L = poly2label(roiPositions,roiLabelIDs,imageSize)` creates a numeric label matrix `L` from the regions of interest (ROIs) defined in `roiPositions`. `roiLabelIDs` specifies the numeric ID for each ROI in `roiPositions`. `imageSize` specifies the size of the output label matrix.

`L = poly2label(roiPositions,roiLabelIDs,R)` creates a numeric label matrix where the spatial referencing object `R` specifies the coordinate system used by the ROI positions in `roiPositions`. The function assumes that the ROI positions are in world limits defined by `R`. The `ImageSize` property of `R` specifies the size of the label matrix `L`.

Examples

Create Label Matrix from Multiple ROIs

Read an image into the workspace and display it.

```
figure
I = imread('baby.jpg');
imshow(I)
```

Initialize the ROI position cell array and image size variables. If you pass `poly2label` a size value containing three dimensions, it only uses the first two, *m*-by-*n*.

```
numPolygon = 3;
roiPositions = cell(numPolygon,1);
imSize = size(I);
```

Specify the coordinates of three ROIs in the `roiPositions` cell array. In this example, the first ROI is a triangle, requiring coordinates for three corners. The other two ROIs are quadrilaterals, requiring coordinates for four corners.

```
roiPositions{1} = [500 500; 250 1300; 1000 500];
roiPositions{2} = [1500 1100; 1500 1400; 2000 1400; 2000 700];
roiPositions{3} = [80 2600; 480 2700; 470 3000; 100 3000];
```

Create an array for label IDs the same size as the `roiPositions` cell array.

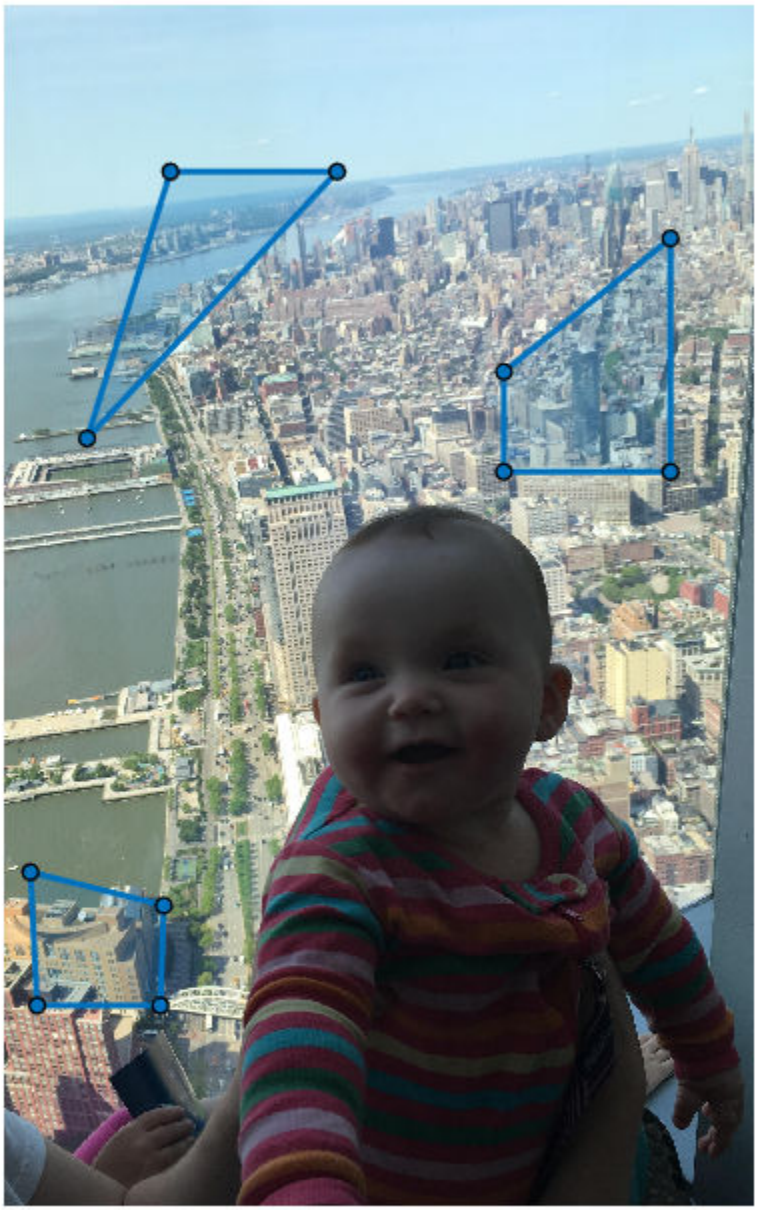
```
roilabelID = zeros(numPolygon,1,'uint8');
```

Specify label ID values that correspond to the order in which you listed the ROIs in `roiPositions`. The first ROI is a triangle so give it the label `1`. The next two ROIs are both quadrilaterals so give them the label `2`.

```
roilabelID(1) = 1;
roilabelID(2) = 2;
roilabelID(3) = 2;
```

Draw the three ROIs on the figure.

```
for id = 1:numPolygon
    drawpolygon('Position',roiPositions{id});
end
```

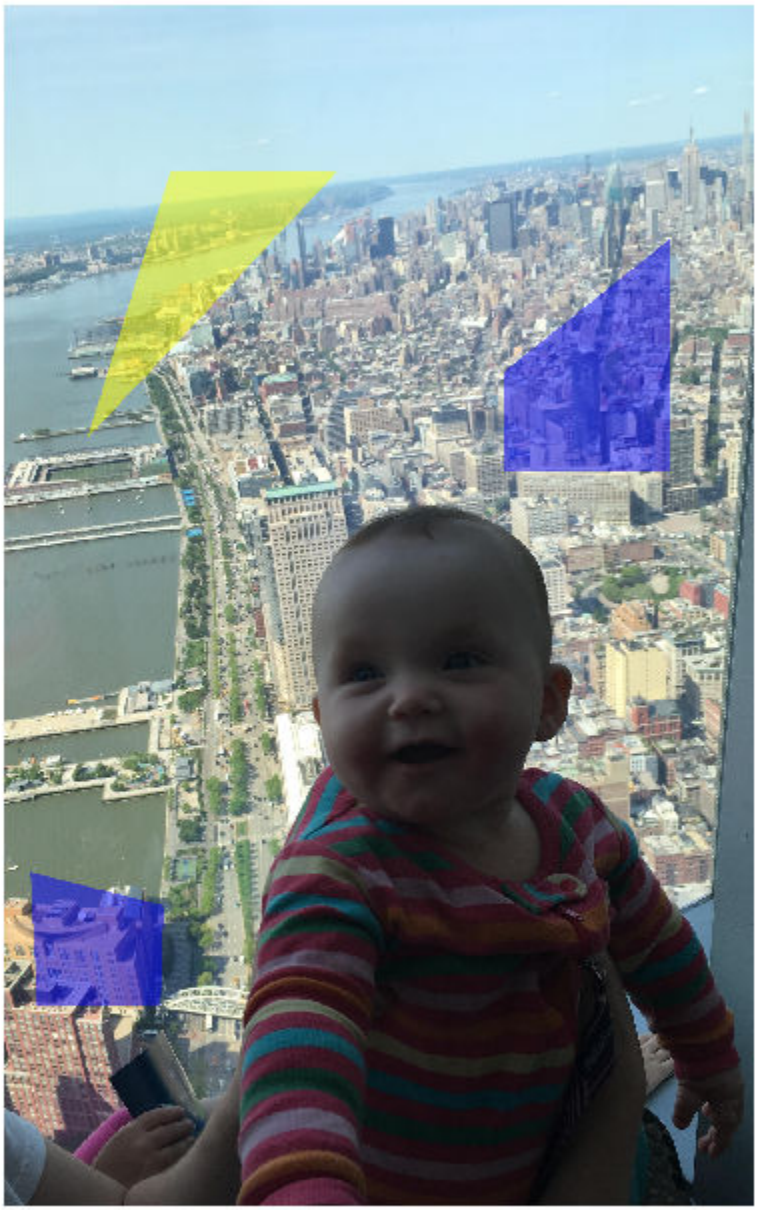


Create a label matrix from the ROIs. The label matrix is the same size, *m*-by-*n*, as the original image.

```
L = poly2label(roiPositions,roilabelID,imSize);
```

Display the label matrix overlaid on the original image.

```
figure;
B = labeloverlay(I,L);
imshow(B);
```



Input Arguments

`roiPositions` — Coordinate vectors

1-by-*P* cell array of numeric vectors

Coordinate vectors, specified as a 1-by-*P* cell array of numeric vectors, where *P* is the total number of ROIs. Each cell array element is an *s*-by-2 coordinate vector of the form `[x1 y1; …; xs ys]`, where *s* is the total number of vertices for that ROI. Each *x*,*y* pair defines a vertex of the ROI. If the ROI shape is not already closed, the `poly2label` function closes the shape automatically. You can specify any number of ROIs.

Data Types: `double` | `cell`

`roiLabelIDs` — Labels for each ROI

numeric vector

Labels for each ROI, specified as a numeric vector of the same length as the `roiPositions` argument. Each label in the vector corresponds to the ROI in the associated position in the `roiPositions` cell array.

`poly2label` assigns the value `0` to all background pixels in the output image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`imageSize` — Size of the output label matrix

2-element numeric vector | 3-element numeric vector

Size of the output label matrix, specified as a 2- or 3-element numeric vector. If you specify a 3-element vector, the `poly2label` function uses only the first two dimensions, *m*-by-*n*.

Data Types: `double`

`R` — Spatial referencing information

`imref2d` object

Spatial referencing information, specified as an `imref2d` object.

Output Arguments

`L` — Label matrix

m-by-*n* matrix of nonnegative values

Label matrix, returned as an *m*-by-*n* matrix of nonnegative values of the same data type as `roiLabelIDs`. Pixels labeled `0` are the background.

Tips

- The `poly2label` function sets pixels that are inside an ROI to a label value. For more information about classifying pixels on the ROI boundary, see “Classify Pixels That Are Partially Enclosed by ROI”.
- When the positions of several ROIs overlap each other, the ROI label with the lowest index number in the `roiPositions` cell array overwrites the other ROIs.

See Also

poly2mask | drawpolygon | labeloverlay | roipoly | polyToBlockedImage

Introduced in R2020b

poly2mask

Convert region of interest (ROI) polygon to region mask

Syntax

```
BW = poly2mask(xi,yi,m,n)
```

Description

`BW = poly2mask(xi,yi,m,n)` computes a binary region of interest (ROI) mask, `BW`, of size `m`-by-`n`, from an ROI polygon with vertices at coordinates `xi` and `yi`. If the polygon is not already closed, then `poly2mask` closes the polygon automatically.

The `poly2mask` function sets pixels that are inside the polygon to `1` and sets pixels outside the polygon to `0`. For more information about classifying pixels on the ROI boundary, see “Classify Pixels That Are Partially Enclosed by ROI”.

Examples

Define Polygon and Create Mask

Specify the x- and y-coordinates of the polygon.

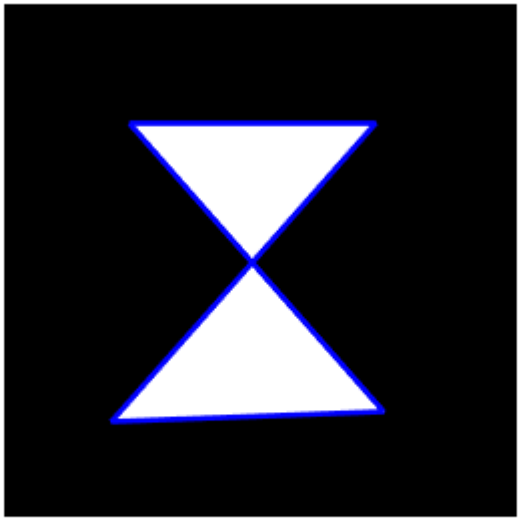
```
x = [63 186 54 190 63];
y = [60 60 209 204 60];
```

Create the mask specifying the size of the image.

```
bw = poly2mask(x,y,256,256);
```

Display the mask, drawing a line around the polygon.

```
imshow(bw)
hold on
plot(x,y,'b','LineWidth',2)
hold off
```



Create Mask Using Random Points to Define Polygon

Define two sets of random points for the x- and y-coordinates.

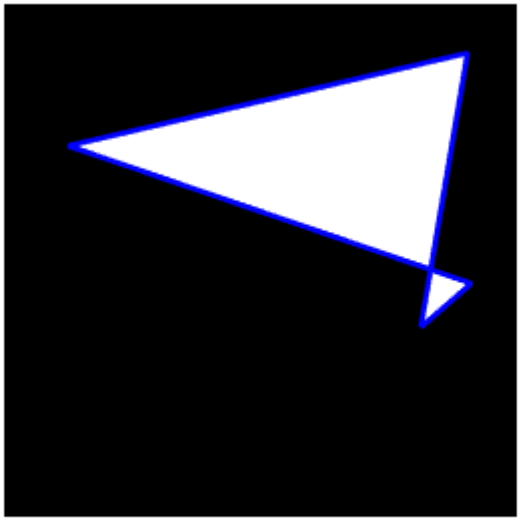
```
x = 256*rand(1,4);
y = 256*rand(1,4);
x(end+1) = x(1);
y(end+1) = y(1);
```

Create the mask.

```
bw = poly2mask(x,y,256,256);
```

Display the mask and draw a line around the polygon.

```
imshow(bw)
hold on
plot(x,y,'b','LineWidth',2)
hold off
```



Input Arguments

`xi` — *x*-coordinate of polygon vertices

numeric vector

x-coordinate of polygon vertices, specified as a numeric vector. The length of `xi` and `yi` must match.

Data Types: `double`

`yi` — *y*-coordinate of polygon vertices

numeric vector

y-coordinate of polygon vertices, specified as a numeric vector. The length of `xi` and `yi` must match.

Data Types: `double`

`m` — Number of rows in mask

nonnegative integer

Number of rows in the mask, specified as a nonnegative integer.

Data Types: `double`

`n` — Number of columns in mask

nonnegative integer

Number of columns in the mask, specified as a nonnegative integer.

Data Types: `double`

Output Arguments

BW — Binary image

m-by-n logical matrix

Binary image, returned as an m-by-n logical matrix.

Data Types: logical

Tips

- To specify a polygon that includes a given rectangular set of pixels, make the edges of the polygon lie along the outside edges of the bounding pixels, instead of the center of the pixels.

  For example, to include pixels in columns 4 through 10 and rows 4 through 10, you might specify the polygon vertices like this:

```
x = [4 10 10 4 4];
y = [4 4 10 10 4];
mask = poly2mask(x,y,12,12)

mask =

     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     1     1     1     1     1     1     0     0
     0     0     0     0     1     1     1     1     1     1     0     0
     0     0     0     0     1     1     1     1     1     1     0     0
     0     0     0     0     1     1     1     1     1     1     0     0
     0     0     0     0     1     1     1     1     1     1     0     0
     0     0     0     0     1     1     1     1     1     1     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
```

  In this example, the polygon goes through the center of the bounding pixels, with the result that only some of the desired bounding pixels are determined to be inside the polygon (the pixels in row 4 and column 4 and not in the polygon). To include these elements in the polygon, use fractional values to specify the outside edge of the 4th row (3.5) and the 10th row (10.5), and the outside edge of the 4th column (3.5) and the outside edge of the 10th column (10.5) as vertices, as in the following example:

```
x = [3.5 10.5 10.5 3.5 3.5];
y = [3.5 3.5 10.5 10.5 3.5];
mask = poly2mask(x,y,12,12)

mask =

     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     1     1     1     1     1     1     1     0     0
     0     0     0     1     1     1     1     1     1     1     0     0
     0     0     0     1     1     1     1     1     1     1     0     0
     0     0     0     1     1     1     1     1     1     1     0     0
     0     0     0     1     1     1     1     1     1     1     0     0
     0     0     0     1     1     1     1     1     1     1     0     0
     0     0     0     1     1     1     1     1     1     1     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
```

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`poly2mask` supports the generation of C code (requires MATLAB Coder). For more information, see "Code Generation for Image Processing".

See Also

`roipoly` | `roifilt2`

Topics

"Classify Pixels That Are Partially Enclosed by ROI"

Introduced before R2006a

polyToBlockedImage

Create labeled `blockedImage` object from set of ROIs

Syntax

```
Bout = polyToBlockedImage(ROIpositions,ROILabelIDs,imageSize)
Bout = polyToBlockedImage( ___ ,Name=Value)
```

Description

`Bout = polyToBlockedImage(ROIpositions,ROILabelIDs,imageSize)` creates a numeric, labeled 2-D blocked image `Bout` of the specified size `imageSize` from the regions of interest (ROIs) defined in `ROIpositions` and label IDs defined in `ROILabelIDs`.

`Bout = polyToBlockedImage( ___ ,Name=Value)` specifies object properties of `Bout` using name-value arguments in addition to the input arguments from the previous syntax.

Example: `polyToBlockedImage(ROIpositions,ROILabelIDs,BlockSize=[512,512])` creates a blocked image with a block size of 512-by-512 pixels.

Examples

Create Labeled Blocked Image from ROIs

Create a blocked image.

```
bim = blockedImage("baby.jpg");
```

Initialize the ROI position cell array.

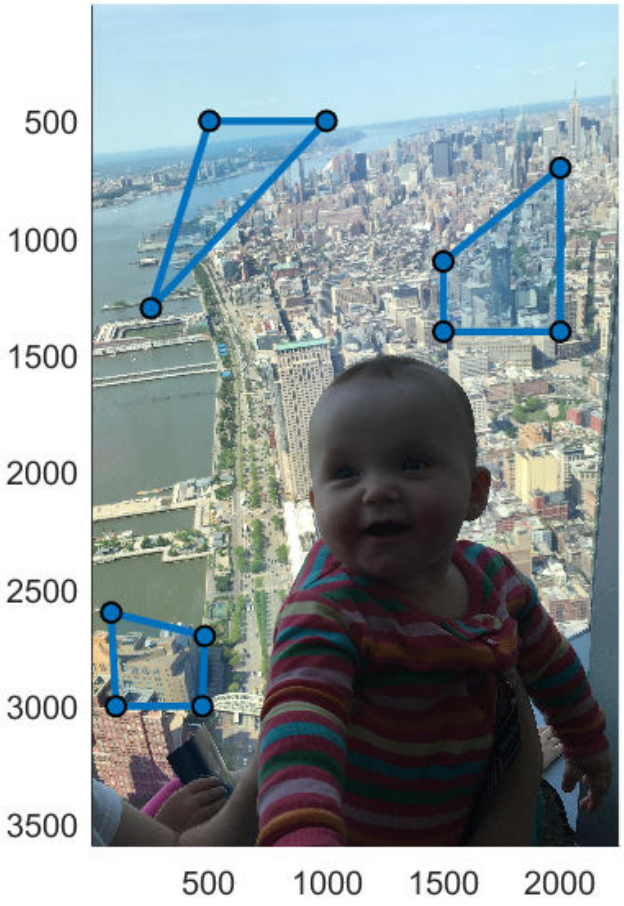
```
numPolygon = 3;
roiPositions = cell(numPolygon,1);
```

Specify the coordinates of three ROIs in the `roiPositions` cell array. The first ROI is a triangle, requiring `x`,`y` coordinates for three vertices. The other two ROIs are quadrilaterals, requiring `x`,`y` coordinates for four vertices.

```
roiPositions{1} = [500 500; 250 1300; 1000 500];
roiPositions{2} = [1500 1100; 1500 1400; 2000 1400; 2000 700];
roiPositions{3} = [80 2600; 480 2700; 470 3000; 100 3000];
```

Display the blocked image, and draw the three ROIs on the figure.

```
bigimageshow(bim);
for id = 1:numPolygon
    drawpolygon(Position=roiPositions{id});
end
```



Create an array for label IDs the same size as the `roiPositions` cell array.

```
roilabelID = zeros(numPolygon,1,"uint8");
```

Specify label ID values that correspond to the order in which you listed the ROIs in `roiPositions`. The first ROI is a triangle, so label it `1`. The other two ROIs are quadrilaterals, so label them `2`.

```
roilabelID(1) = 1;
roilabelID(2) = 2;
roilabelID(3) = 2;
```

Specify the size of the new labeled blocked image to match the size of the initial blocked image.

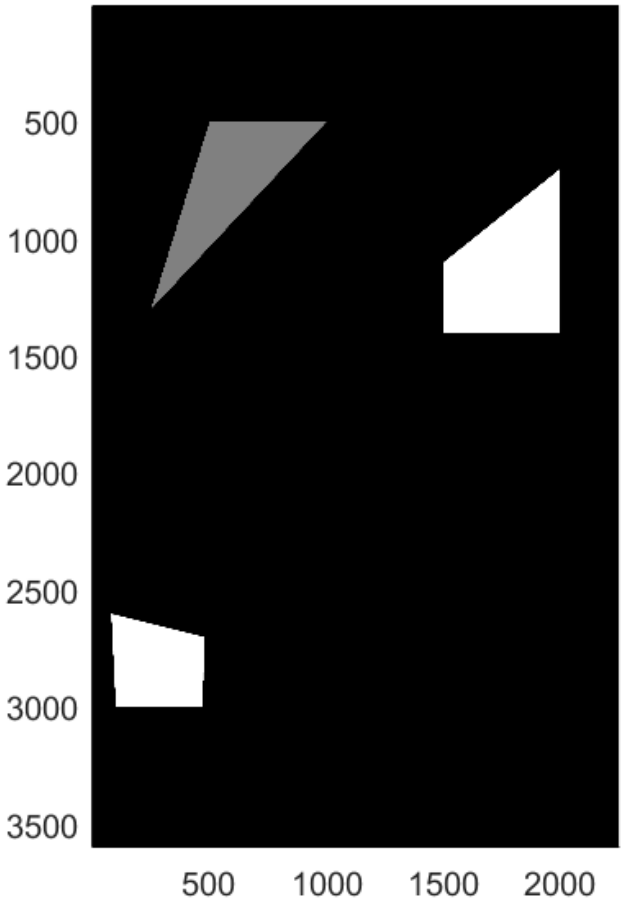
```
imageSize = bim.Size(1:2);
```

Create a labeled `blockedImage` object from the ROIs.

```
L = polyToBlockedImage(roiPositions,roilabelID,imageSize);
```

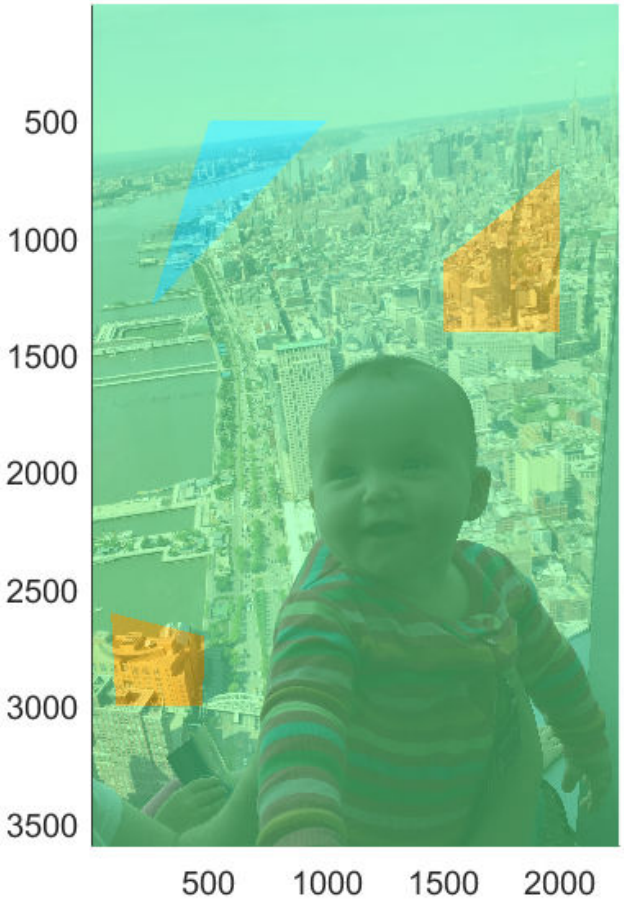
Display the labeled blocked image. Use color scaling and color axis limits to visualize the differences between the labels for triangular and quadrilateral ROIs.

```
bigimageshow(L,CDataMapping="scaled")
caxis([0 2]);
```



Display the labeled blocked image overlaid on the original blocked image.

```
hbim = bigimageshow(bim);
showlabels(hbim,L)
```



Create Labeled Blocked Image Specifying World Coordinates

Create a blocked image.

```
bim = blockedImage("tumor_091R.tif");
```

Initialize the ROI position cell array.

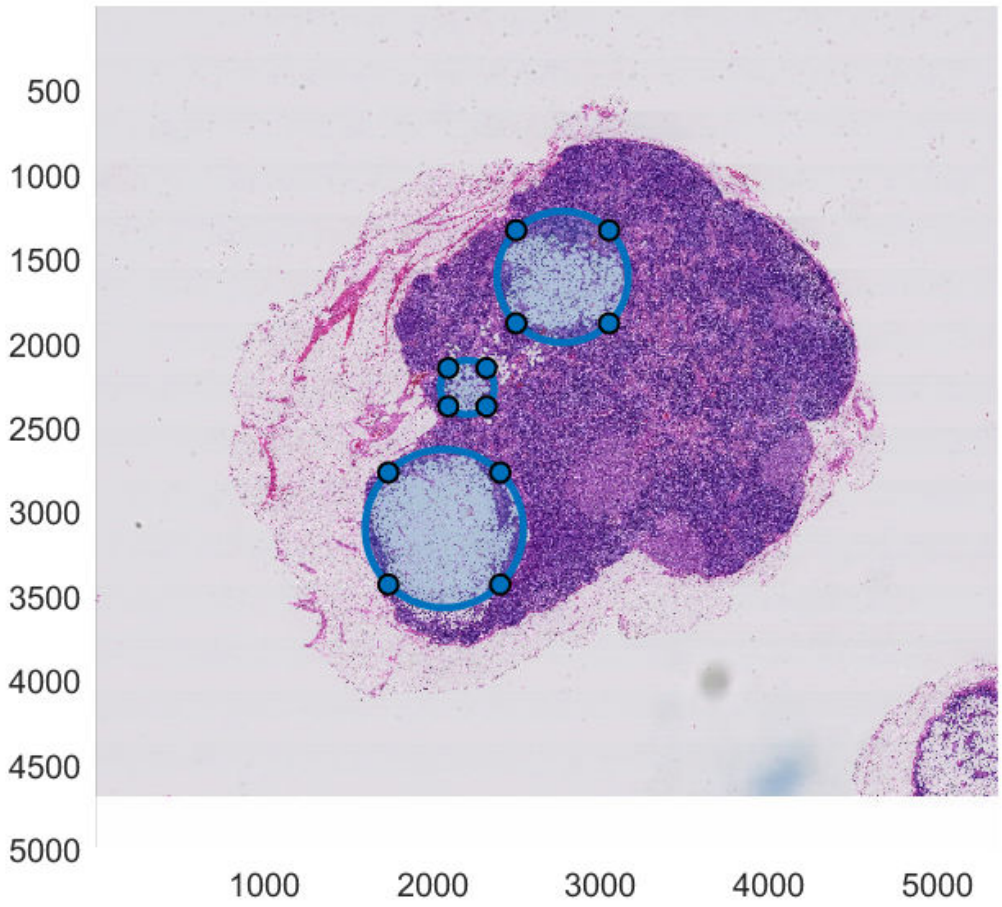
```
numPolygon = 3;
roiPositions = cell(numPolygon,1);
```

Specify the center and radius parameters for three circular ROIs.

```
center = [2774 1607; 2071 3100; 2208 2262];
radius = [390; 470; 161];
```

Display the blocked image, and use `drawcircle` to draw the three circular ROIs on the figure. Add the `x`,`y` coordinates of the vertices for each ROI to the `roiPositions` cell array.

```
hbim = bigimageshow(bim);
for id = 1:numPolygon
    hROI = drawcircle(Radius=radius(id),Center=center(id,:));
    roiPositions{id} = hROI.Vertices;
end
```



Create an array for label IDs the same size as the `roiPositions` cell array.

```
roilabelID = zeros(numPolygon,1,"uint8");
```

Specify label ID values that correspond to the order in which you listed the ROIs in `roiPositions`. You can assign each ROI a different label, or group multiple ROIs under the same label.

```
roilabelID(1) = 1;
roilabelID(2) = 2;
roilabelID(3) = 2;
```

Specify an image size for the new labeled blocked image equal to that of the second resolution level of the initial blocked image, `bim`.

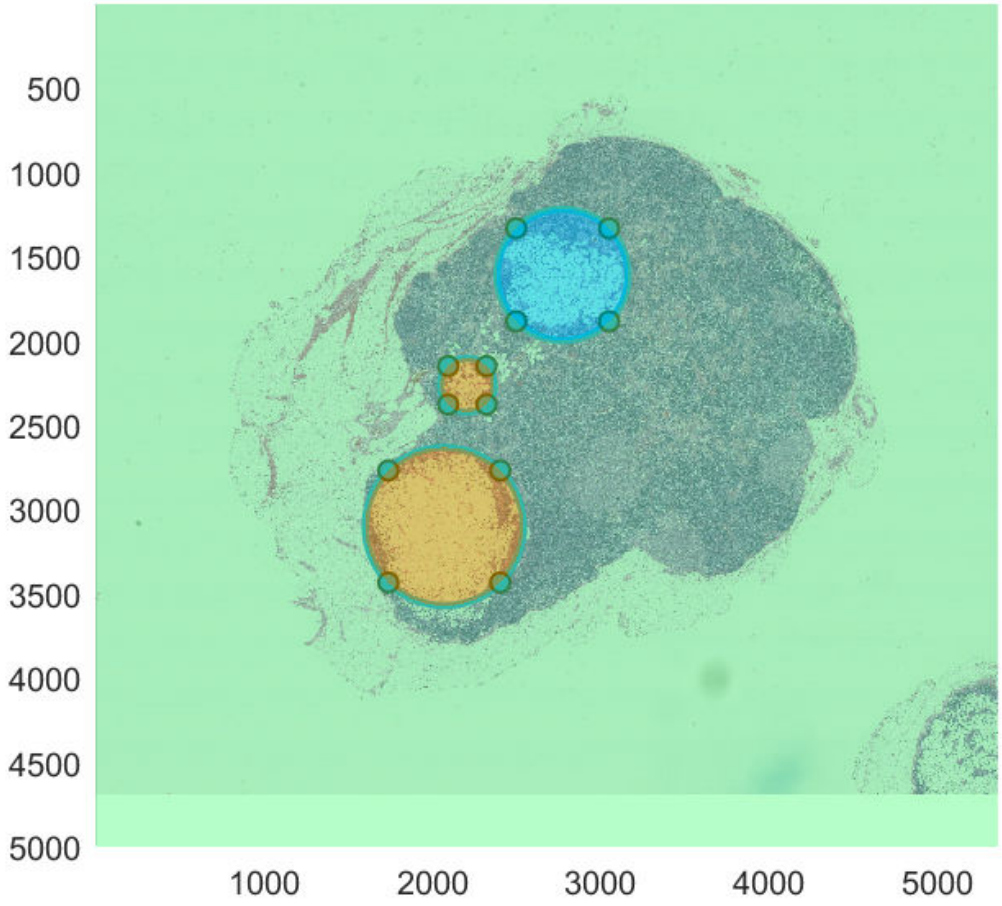
```
maskLevel = 2;
imageSize = bim.Size(maskLevel,1:2);
```

Create a labeled `blockedImage` object from the ROIs. Use the `WorldStart` and `WorldEnd` name-value arguments to maintain the same world coordinates as the initial blocked image at the specified resolution level.

```
L = polyToBlockedImage(roiPositions,roilabelID, ...
imageSize,WorldStart=bim.WorldStart(maskLevel, 1:2), ...
WorldEnd=bim.WorldEnd(maskLevel, 1:2));
```

Display the labeled blocked image overlaid on the original blocked image.

```
showlabels(hbim,L)
```



Input Arguments

`ROIpositions` — Coordinate vectors

P-element cell array of numeric vectors

Coordinate vectors, specified as a *P*-element cell array of numeric vectors, where *P* is the total number of ROIs. Each cell array element is an *s*-by-2 coordinate vector of the form `[x1 y1; … ; xs`

`ys]`, where *s* is the total number of vertices for that ROI. Each *x*,*y* pair defines a vertex of the ROI. If the ROI shape is not already closed, the `polyToBlockedImage` function closes the shape automatically. You can specify any number of ROIs.

Data Types: `cell`

`ROILabelIDs` — Labels for each ROI

P-element numeric vector | *P*-element logical vector

Labels for each ROI, specified as a *P*-element numeric vector or *P*-element logical vector, where *P* is the total number of ROIs.

`polyToBlockedImage` assigns the value `0` to all background pixels in the output image.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`imageSize` — Size of output labeled blocked image

numeric vector of positive integers

Size of the output labeled blocked image, specified as a numeric vector of positive integers. If you specify more than two dimensions, the `polyToBlockedImage` function uses only the first two dimensions.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example:
`polyToBlockedImage(ROIpositions,ROILabelIDs,imageSize,BlockSize=[512,512])`
creates a labeled blocked image with a block size of 512-by-512 pixels.

`Adapter` — Read and write interface for blocked image object

`InMemory` object | `MATBlocks` object | `PNGBlocks` object | `TIFF` object | ...

Read and write interface for the blocked image object, specified as one of these adapter objects.

Adapter	Description
`BINBlocks`	Store each block as a binary file in a folder
`GenericImage`	Store blocks in a single image
`GenericImageBlocks`	Store each block as an image file in a folder
`H5`	Store blocks in a single HDF5 image
`H5Blocks`	Store each block as an HDF5 file in a folder
`InMemory`	Store blocks in a variable in main memory
`JPEGBlocks`	Store each block as a JPEG file in a folder
`MATBlocks`	Store each block as a MAT file in a folder
`PNGBlocks`	Store each block as a PNG file in a folder
`TIFF`	Store blocks in a single TIFF file

You can also create your own adapter using the `images.blocked.Adapter` class.

If `OutputLocation` is specified, then the function automatically selects an adapter based on the output location. If `OutputLocation` is not specified, then the default adapter is an `InMemory` object.

`BlockSize` — Block size
`[512,512]` (default) | numeric vector of positive integers

Block size, specified as a numeric vector of positive integers. If you specify more than two dimensions, the `polyToBlockedImage` function uses only the first two dimensions.

Data Types: `double`

`DisplayWaitbar` — Wait bar display toggle
`true` or `1` (default) | `false` or `0`

Wait bar display toggle, specified as a numeric or logical `1` (`true`) or `0` (`false`). When set to `true`, the `polyToBlockedImage` function displays a wait bar for long-running operations. If you cancel the wait bar, the `polyToBlockedImage` function returns a partial output, if available.

Data Types: `logical`

`OutputLocation` — Location to store output labeled blocked image
`[]` (default) | character vector | string scalar

Location to store the output labeled blocked image, specified as a character vector or string scalar. By default, the output blocked image is stored in memory.

Destination Type	Image Format
Folder name (without a file extension)	The `polyToBlockedImage` function creates the folder and stores blocks of data as files within the folder. • For numeric image data, `polyToBlockedImage` stores each block as a binary file using the `BINBlocks` adapter. • For categorical and structure image data, `polyToBlockedImage` stores each block as a MAT file in the folder using the `MATBlocks` adapter.
File name with TIF or TIFF file extension	The `polyToBlockedImage` function stores data as a single TIFF image using the `TIFF` adapter. The `initialValue` must be numeric or logical with data type `uint8`, `int8`, `uint16`, `int16`, `uint32`, `int32`, `single`, `double` or `logical`.
File name with H5 file extension	The `polyToBlockedImage` function stores data as a single HDF5 image using the `H5` adapter. The `initialValue` must be numeric with data type `uint8`, `int8`, `uint16`, `int16`, `uint32`, `int32`, `single`, or `double`.

Destination Type	Image Format
`[]`	The `blockedImage` object stores data as a variable in memory using the `InMemory` adapter.

To specify a custom adapter for other output formats, use the `Adapter` property.

`WorldStart` — World coordinates of starting edge of image
numeric vector

World coordinates of the starting edge of the image, specified as a numeric vector. By default, the value is `[0.5,0.5]`.

Data Types: `double`

`WorldEnd` — World coordinates of ending edge of image
numeric vector

World coordinates of the ending edge of the image, specified as a numeric vector. By default, the value is `imageSize + 0.5`, resulting in pixels that are one unit wide.

Data Types: `double`

Output Arguments

`Bout` — Numeric labeled 2-D blocked image
`blockedImage` object

Numeric labeled 2-D blocked image, returned as a `blockedImage` object.

Tips

- To create a labeled blocked image, `Bout`, to overlay on an existing blocked image, match `imageSize` to the size of the existing blocked image at the desired resolution level. If the resolution level of `Bout` matches the finest resolution level of the existing blocked image, you can use the default values for `WorldStart` and `WorldEnd`. To display the overlay at a coarse resolution level, specify `WorldStart` and `WorldEnd` to match the world extents of the existing blocked image at the desired resolution level.
- Creating the labeled blocked image at a coarser resolution level decreases the memory required to store the new blocked image, but decreases the smoothness of the ROI edges.
- If a pixel is inside multiple overlapping ROIs, the function assigns the pixel the label corresponding to the overlapping ROI with the lowest index in `ROIpositions`.

See Also
`blockedImage` | `bigimageshow` | `showlabels` | `poly2label` | `drawcircle`

Topics
“Convert Image Labeler Polygons to Labeled Blocked Image for Semantic Segmentation”

Introduced in R2021b

pretrainedEncoderNetwork

Create encoder network from pretrained network

Syntax

```
net = pretrainedEncoderNetwork(networkName,depth)
[net,outputNames] = pretrainedEncoderNetwork(networkName,depth)
```

Description

`net = pretrainedEncoderNetwork(networkName,depth)` creates an encoder network, `net`, from a pretrained network, `networkName`. The encoder network performs `depth` downsampling operations.

This function requires Deep Learning Toolbox.

`[net,outputNames] = pretrainedEncoderNetwork(networkName,depth)` also returns the names, `outputNames`, of activation layers that occur directly before downsampling operations. These activations correspond to features of interest at particular spatial resolutions or scales.

Examples

Create Encoder Network from Pretrained SqueezeNet Network

Create an encoder with three downsampling operations based on the SqueezeNet pretrained network.

```
encoderNet = pretrainedEncoderNetwork('squeezenet',3)
```

```
encoderNet =
  dlnetwork with properties:

         Layers: [33x1 nnet.cnn.layer.Layer]
    Connections: [36x2 table]
     Learnables: [26x3 table]
          State: [0x3 table]
     InputNames: {'data'}
    OutputNames: {'fire5-concat'}
    Initialized: 1
```

Display the encoder network.

```
analyzeNetwork(encoderNet)
```

Create U-Net from Pretrained GoogLeNet

Create a GAN encoder network with four downsampling operations from a pretrained GoogLeNet network.

```
depth = 4;
[encoder,outputNames] = pretrainedEncoderNetwork('googlenet',depth);
```

Determine the input size of the encoder network.

```
inputSize = encoder.Layers(1).InputSize;
```

Determine the output size of the activation layers in the encoder network by creating a sample data input and then calling `forward`, which returns the activations.

```
exampleInput = dlarray(zeros(inputSize),'SSC');
exampleOutput = cell(1,length(outputNames));
[exampleOutput{:}] = forward(encoder,exampleInput,'Outputs',outputNames);
```

Determine the number of channels in the decoder blocks as the length of the third channel in each activation.

```
numChannels = cellfun(@(x) size(extractdata(x),3),exampleOutput);
numChannels = fliplr(numChannels(1:end-1));
```

Define a function that creates an array of layers for one decoder block.

```
decoderBlock = @(block) [
    transposedConv2dLayer(2,numChannels(block),'Stride',2)
    convolution2dLayer(3,numChannels(block),'Padding','same')
    reluLayer
    convolution2dLayer(3,numChannels(block),'Padding','same')
    reluLayer];
```

Create the decoder module with the same number of upsampling blocks as there are downsampling blocks in the encoder module.

```
decoder = blockedNetwork(decoderBlock,depth);
```

Create the U-Net network by connecting the encoder module and decoder module and adding skip connections.

```
net = encoderDecoderNetwork([224 224 3],encoder,decoder, ...
   'OutputChannels',3,'SkipConnections','concatenate')
```

```
net =
  dlnetwork with properties:

         Layers: [139x1 nnet.cnn.layer.Layer]
    Connections: [167x2 table]
     Learnables: [116x3 table]
          State: [0x3 table]
     InputNames: {'data'}
    OutputNames: {'encoderDecoderFinalConvLayer'}
    Initialized: 1
```

Display the network.

```
analyzeNetwork(net)
```

Input Arguments

networkName — Pretrained network name

`"googlenet"` | `"inceptionv3"` | `"resnet101"` | `"vgg19"` | ...

Pretrained network name, specified as one of these string values. You must install the associated Add-On for the selected pretrained network.

- `"alexnet"` — See `alexnet` for more information.
- `"googlenet"` — See `googlenet` for more information.
- `"inceptionresnetv2"` — See `inceptionresnetv2` for more information.
- `"inceptionv3"` — See `inceptionv3` for more information.
- `"mobilenetv2"` — See `mobilenetv2` for more information.
- `"resnet18"` — See `resnet18` for more information.
- `"resnet50"` — See `resnet50` for more information.
- `"resnet101"` — See `resnet101` for more information.
- `"squeezenet"` — See `squeezenet` for more information.
- `"vgg16"` — See `vgg16` for more information.
- `"vgg19"` — See `vgg19` for more information.

Data Types: `char` | `string`

depth — Number of downsampling operations

2 (default) | positive integer

Number of downsampling operations in the encoder, specified as a positive integer. The encoder downsamples the input by a factor of `2^depth`. You cannot specify a depth greater than the depth of the pretrained network.

Output Arguments

net — Encoder network

`dlnetwork` object

Encoder network, returned as a `dlnetwork` object. The network has `depth` distinct spatial resolutions. The final layer of the encoder network is the layer that comes directly before the next downsampling operation of the pretrained network.

outputNames — Layer names

string vector

Layer names in the network `net` that come directly before downsampling operations, returned as a string vector.

See Also

`encoderDecoderNetwork`

Topics
“Create Modular Neural Networks”
“Get Started with GANs for Image-to-Image Translation”

Introduced in R2021a

projective2d

2-D projective geometric transformation

Description

A `projective2d` object encapsulates a 2-D projective geometric transformation.

Creation

You can create a `projective2d` object using the following methods:

- `fitgeotrans` — Estimates a geometric transformation that maps pairs of control points between two images
- The `projective2d` function described here

Syntax

```
tform = projective2d
tform = projective2d(A)
```

Description

`tform = projective2d` creates a `projective2d` object with default property settings that correspond to the identity transformation.

`tform = projective2d(A)` sets the property `T` with a valid projective transformation defined by nonsingular matrix `A`.

Properties

T — Forward 2-D projective transformation

nonsingular 3-by-3 numeric matrix

Forward 2-D projective transformation, specified as a nonsingular 3-by-3 numeric matrix.

The matrix `T` uses the convention:

```
[x y 1] = [u v 1] * T
```

where `T` has the form:

```
[a b c;...
 d e f;...
 g h i];
```

The default of `T` is the identity transformation.

Data Types: `double` | `single`

`Dimensionality` — Dimensionality of the geometric transformation

2

Dimensionality of the geometric transformation for both input and output points, specified as the value 2.

Object Functions

invert	Invert geometric transformation
outputLimits	Find output spatial limits given input spatial limits
transformPointsForward	Apply forward geometric transformation
transformPointsInverse	Apply inverse geometric transformation

Examples

Apply Projective Transformation to Image

This example shows how to apply rotation and tilt to an image, using a `projective2d` geometric transformation object created directly from a transformation matrix.

Read a grayscale image into the workspace.

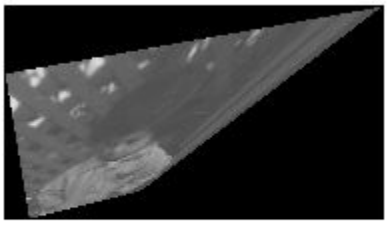
```
I = imread('pout.tif');
```

Combine rotation and tilt into a transformation matrix, `tm`. Use this transformation matrix to create a `projective2d` geometric transformation object, `tform`.

```
theta = 10;
tm = [cosd(theta) -sind(theta) 0.001; ...
    sind(theta) cosd(theta) 0.01; ...
    0 0 1];
tform = projective2d(tm);
```

Apply the transformation using `imwarp`. View the transformed image.

```
outputImage = imwarp(I,tform);
imshow(outputImage)
```



Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `projective2d` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- When generating code, you can only specify singular objects—arrays of objects are not supported.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, you can only specify singular objects—arrays of objects are not supported.

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Functions

`imwarp` | `fitgeotrans`

Objects

`affine2d` | `rigid2d` | `geometricTransform2d` | `LocalWeightedMeanTransformation2D` | `PiecewiseLinearTransformation2D` | `PolynomialTransformation2D`

Topics

“Register Images with Projection Distortion Using Control Points”
“2-D and 3-D Geometric Transformation Process Overview”
“Matrix Representation of Geometric Transformations”

Introduced in R2013a

psf2otf

Convert point-spread function to optical transfer function

Syntax

```
OTF = psf2otf(PSF)
OTF = psf2otf(PSF,sz)
```

Description

`OTF = psf2otf(PSF)` computes the fast Fourier transform (FFT) of the point-spread function (`PSF`) array and creates the optical transfer function array, `OTF`, that is not influenced by the `PSF` off-centering.

`OTF = psf2otf(PSF,sz)` specifies the size, `sz`, of the optical transfer function.

Examples

Convert PSF to OTF

Create a point-spread function (PSF).

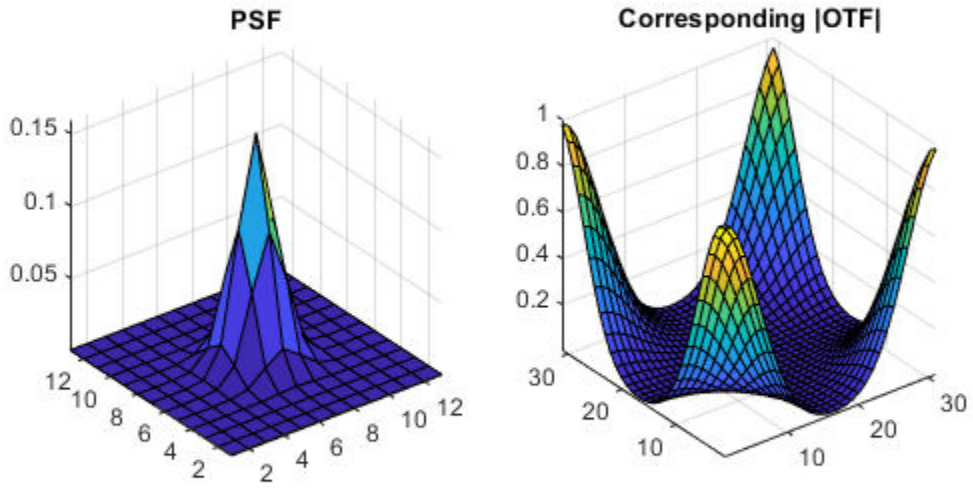
```
PSF  = fspecial('gaussian',13,1);
```

Convert the PSF to an Optical Transfer Function (OTF).

```
OTF  = psf2otf(PSF,[31 31]);
```

Plot the PSF and the OTF.

```
subplot(1,2,1);
surf(PSF);
title('PSF');
axis square;
axis tight
subplot(1,2,2);
surf(abs(OTF));
title('Corresponding |OTF|');
axis square;
axis tight
```



Input Arguments

PSF — Point-spread function

numeric array

Point-spread function, specified as a numeric array of any dimension.

Example: `PSF = fspecial('gaussian',13,1);`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`
Complex Number Support: Yes

sz — Size of optical transfer function

vector of positive integers

Size of the output optical transfer function `OTF`, specified as a vector of positive integers. The size of `OTF` must not exceed the size of `PSF` in any dimension. By default, `OTF` is the same size as `PSF`.

Data Types: `double`

Output Arguments

OTF — Optical transfer function

numeric array

Optical transfer function, returned as a numeric array of size `sz`.

Data Types: `double`
Complex Number Support: Yes

Tips

- To ensure that `OTF` is not altered because of `PSF` off-centering, `psf2otf` postpads `PSF` (down or to the right) with `0`s to match dimensions specified in `sz`. Then `psf2otf` circularly shifts the values of `PSF` up (or to the left) until the central pixel reaches (1,1) position.
- This function is used in image convolution and deconvolution when the operations involve the FFT.

See Also

`otf2psf` | `circshift` | `padarray` | `fftn` | `ifftn`

Topics
“Create Your Own Deblurring Functions”

Introduced before R2006a

psnr

Peak signal-to-noise ratio (PSNR)

Syntax

```
peaksnr = psnr(A,ref)
peaksnr = psnr(A,ref,peakval)
peaksnr = psnr(___,'DataFormat',dataFormat)
[peaksnr,snr] = psnr(___)
```

Description

`peaksnr = psnr(A,ref)` calculates the peak signal-to-noise ratio (PSNR) for the image `A`, with the image `ref` as the reference. A greater PSNR value indicates better image quality.

`peaksnr = psnr(A,ref,peakval)` calculates the PSNR of image `A` using the peak signal value `peakval`.

`peaksnr = psnr(___,'DataFormat',dataFormat)` also specifies the dimension labels, `dataFormat`, of unformatted image data. Use this syntax to return a separate PSNR for each element along a batch dimension.

`[peaksnr,snr] = psnr(___)` also returns the simple signal-to-noise ratio, `snr`.

Examples

Calculate PSNR for Noisy Image Given Original Image as Reference

Read image and create a copy with added noise. The original image is the reference image.

```
ref = imread('pout.tif');
A = imnoise(ref,'salt & pepper', 0.02);
```

Calculate the PSNR.

```
[peaksnr, snr] = psnr(A, ref);

fprintf('\n The Peak-SNR value is %0.4f', peaksnr);
```

```
 The Peak-SNR value is 22.6437
```

```
fprintf('\n The SNR value is %0.4f \n', snr);
```

```
 The SNR value is 15.5524
```

Calculate PSNR for `dlarray` Input

Read an image into the workspace, then create an unformatted `dlarray` object with the image data.

```
ref = imread("strawberries.jpg");
ref = im2single(ref);
dlref = dlarray(ref);
```

Add salt and pepper noise to the image, then create an unformatted `dlarray` object with the noisy image data.

```
noisy = imnoise(ref,'salt & pepper');
dlnoisy = dlarray(noisy);
```

Calculate the peak SNR and SNR of the noisy data with respect to the original data.

```
[peaksnr,snr] = psnr(dlnoisy,dlref)
```

```
peaksnr =
  1x1 single dlarray

   17.5941


snr =
  1x1 single dlarray

   11.1265
```

Calculate PSNR of Images in Image Sequence

Read a reference image into the workspace.

```
ref = imread("office_1.jpg");
```

Preallocate two arrays that store a sequence of six images of the size of the reference image.

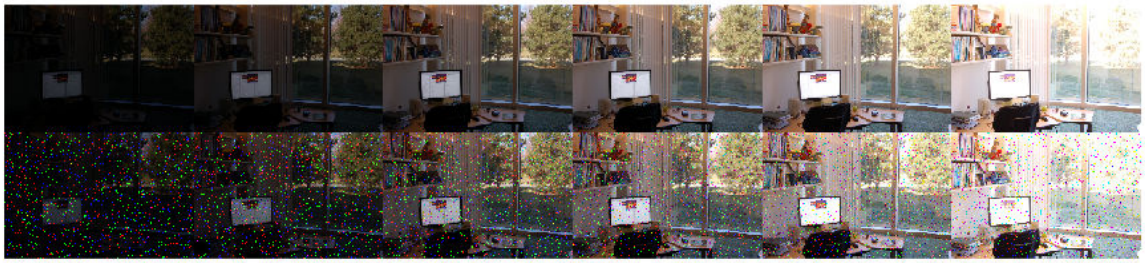
```
numFrames = 6;
imsOriginal = zeros([size(ref) numFrames],class(ref));
imsNoisy = zeros([size(ref) numFrames],class(ref));
```

Read and add images to the preallocated arrays. One array stores the original image data. The second array stores the image data with added salt and pepper noise.

```
for p = 1:numFrames
    filename = strcat("office_",num2str(p),".jpg");
    im = imread(filename);
    imsOriginal(:,:,:,p) = im;
    imsNoisy(:,:,:,p) = imnoise(im,"salt & pepper");
end
```

Display the image sequences in a montage. The first row shows the sequence with original image data. The second row shows the sequence with noisy image data.

```
montage(cat(4,imsOriginal,imsNoisy),"Size",[2 numFrames])
```



Calculate the PSNR of each noisy image with respect to the corresponding pristine image by specifying the data format of the input arrays as "SSCB" (spatial, spatial, channel, batch).

```
peak_psnrs = psnr(imsNoisy,imsOriginal,"DataFormat","SSCB");
peak_psnrs = squeeze(peak_psnrs)
```

```
peak_psnrs = 6×1

   16.3560
   16.9698
   17.8079
   18.1843
   18.0656
   17.1682
```

Input Arguments

`A` — Image to be analyzed

numeric array | `dlarray` object

Image to be analyzed, specified as a numeric array of any dimension or a `dlarray` object.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`ref` — Reference image

numeric matrix | `dlarray` object

Reference image, specified as a numeric array or a `dlarray` object. The reference image has the same size and data type as image `A`.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`peakval` — Peak signal level

nonnegative number

Peak signal level, specified as a nonnegative number. If not specified, the default value for `peakval` depends on the class of `A` and `ref`.

- If the images are of data type `double` or `single`, then `psnr` assumes that image data is in the range [0, 1]. The default value of `peakval` is `1`.

- If the images are of integer data types, then the default value of `peakval` is the largest value allowed by the range of the class. For `uint8` data, the default value of `peakval` is `255`. For `uint16` or `int16`, the default is `65535`.

`dataFormat` — Dimension labels
string scalar | character vector

Dimension labels of the input images `A` and `ref`, specified as a string scalar or character vector. Each character in `dataFormat` must be one of these labels:

- `S` — Spatial
- `C` — Channel
- `B` — Batch observations

The format cannot include more than one channel label or batch label. Do not specify the '`dataFormat`' argument when the input images are formatted `dlarray` objects.

Example: `'SSC'` indicates that the array has two spatial dimensions and one channel dimension, appropriate for 2-D RGB image data.

Example: `'SSCB'` indicates that the array has two spatial dimensions, one channel dimension, and one batch dimension, appropriate for a sequence of 2-D RGB image data.

Output Arguments

`peaksnr` — PSNR
numeric scalar | numeric array | `dlarray` object

PSNR in decibels, returned as one of these values.

Input Image Type	PSNR Value
• Unformatted numeric arrays • Formatted numeric arrays without a batch (`'B'`) dimension	Numeric scalar with a single PSNR measurement.
• Unformatted `dlarray` objects	1-by-1 `dlarray` object with a single PSNR measurement.
• Numeric arrays with a batch dimension specified using the `dataFormat` argument	Numeric array of the same dimensionality as the input images. The spatial and channel dimensions of `peaksnr` are singleton dimensions. There is one PSNR measurement for each element along the batch dimension.
• Formatted `dlarray` objects with a batch dimension • Unformatted `dlarray` objects with a batch dimension specified using the `dataFormat` argument	`dlarray` object of the same dimensionality as the input images. The spatial and channel dimensions of `peaksnr` are singleton dimensions. There is one PSNR measurement for each element along the batch dimension.

If `A` and `ref` have data type `single`, then `peaksnr` has data type `single`. Otherwise, `peaksnr` has data type `double`.

`snr` — Signal-to-noise ratio
numeric scalar | numeric array | `dlarray` object

Signal-to-noise ratio in decibels, returned as one of these values.

Input Image Type	PSNR Value
• Unformatted numeric arrays • Formatted numeric arrays without a batch (`'B'`) dimension	Numeric scalar with a single SNR measurement.
• Unformatted `dlarray` objects	1-by-1 `dlarray` object with a single SNR measurement.
• Numeric arrays with a batch dimension specified using the `dataFormat` argument	Numeric array of the same dimensionality as the input images. The spatial and channel dimensions of `snr` are singleton dimensions. There is one SNR measurement for each element along the batch dimension.
• Formatted `dlarray` objects with a batch dimension • Unformatted `dlarray` objects with a batch dimension specified using the `dataFormat` argument	`dlarray` object of the same dimensionality as the input images. The spatial and channel dimensions of `peaksnr` are singleton dimensions. There is one SNR measurement for each element along the batch dimension.

If `A` and `ref` have data type `single`, then `snr` has data type `single`. Otherwise, `snr` has data type `double`.

Algorithms

The `psnr` function implements this equation to calculate PSNR:

$$PSNR = 10\log_{10}\left(peakval^2/MSE\right)$$

`peakval` is either specified by the user or taken from the range of the image data type. For example, for an image of data type `uint8`, the `peakval` is `255`. *MSE* is the mean square error between `A` and `ref`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`psnr` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`multissim` | `multissim3` | `ssim` | `immse` | `mean` | `median` | `var`

Topics
“Image Quality Metrics”

Introduced in R2014a

qtdecomp

Quadtree decomposition

Syntax

```
S = qtdecomp(I)
S = qtdecomp(I,threshold)
S = qtdecomp(I,threshold,mindim)
S = qtdecomp(I,threshold,[mindim maxdim])
S = qtdecomp(I,fun)
```

Description

`S = qtdecomp(I)` performs a quadtree decomposition on the grayscale image `I` and returns the quadtree structure in the sparse matrix `S`. By default, `qtdecomp` splits a block unless all elements in the block are equal.

`S = qtdecomp(I,threshold)` splits a block if the maximum value of the block elements minus the minimum value of the block elements is greater than `threshold`.

`S = qtdecomp(I,threshold,mindim)` will not produce blocks smaller than `mindim`, even if the resulting blocks do not meet the threshold condition.

`S = qtdecomp(I,threshold,[mindim maxdim])` will not produce blocks smaller than `mindim` or larger than `maxdim`. Blocks larger than `maxdim` are split even if they meet the threshold condition.

`S = qtdecomp(I,fun)` uses the function `fun` to determine whether to split a block.

Examples

Perform Quadtree Decomposition of Sample Matrix

Create a small sample matrix.

```
I = uint8([1 1 1 1 2 3 6 6;...
           1 1 2 1 4 5 6 8;...
           1 1 1 1 7 7 7 7;...
           1 1 1 1 6 6 5 5;...
           20 22 20 22 1 2 3 4;...
           20 22 22 20 5 4 7 8;...
           20 22 20 20 9 12 40 12;...
           20 22 20 20 13 14 15 16]);
```

Perform the quadtree decomposition and display the results.

```
S = qtdecomp(I,.05);
disp(full(S));
```

```
     4     0     0     0     4     0     0     0
     0     0     0     0     0     0     0     0
```

```
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     4     0     0     0     2     0     2     0
     0     0     0     0     0     0     0     0
     0     0     0     0     2     0     1     1
     0     0     0     0     0     0     1     1
```

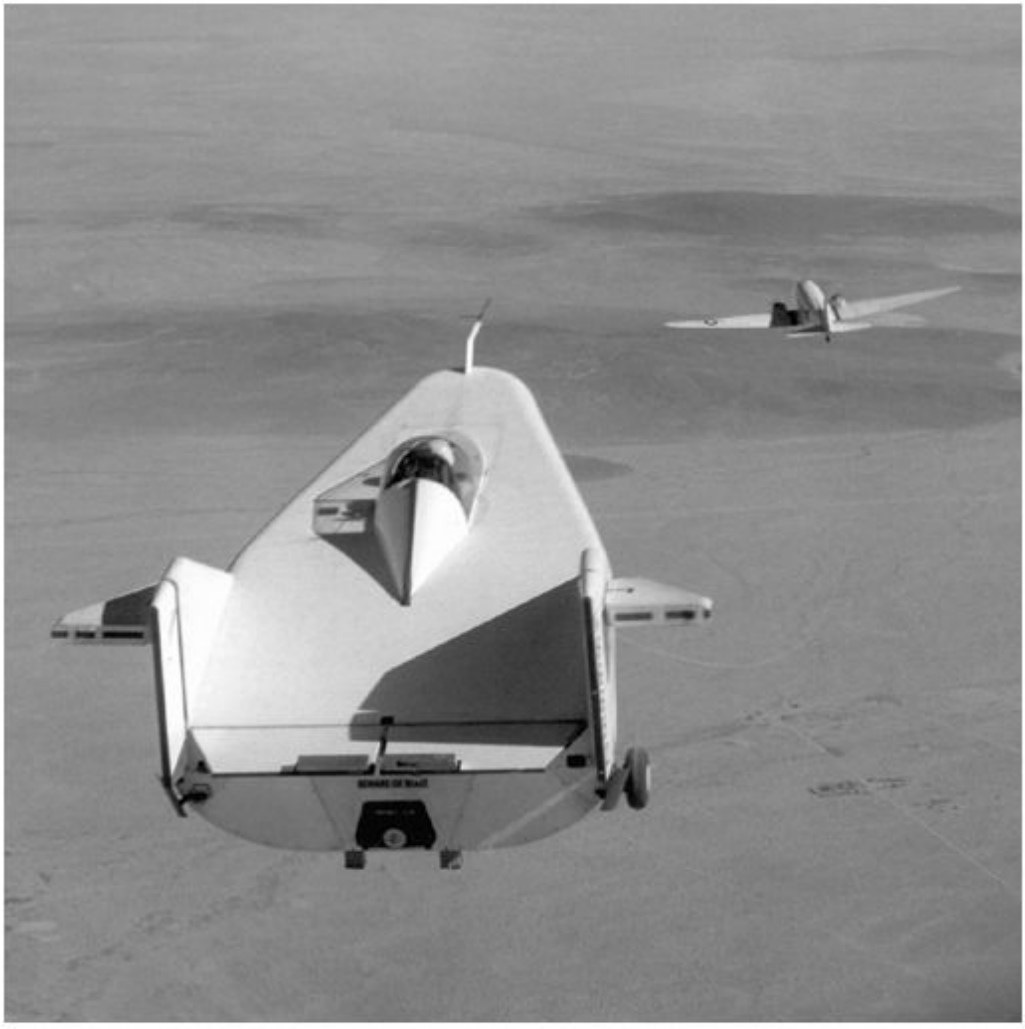
View Block Representation of Quadtree Decomposition

Read image into the workspace.

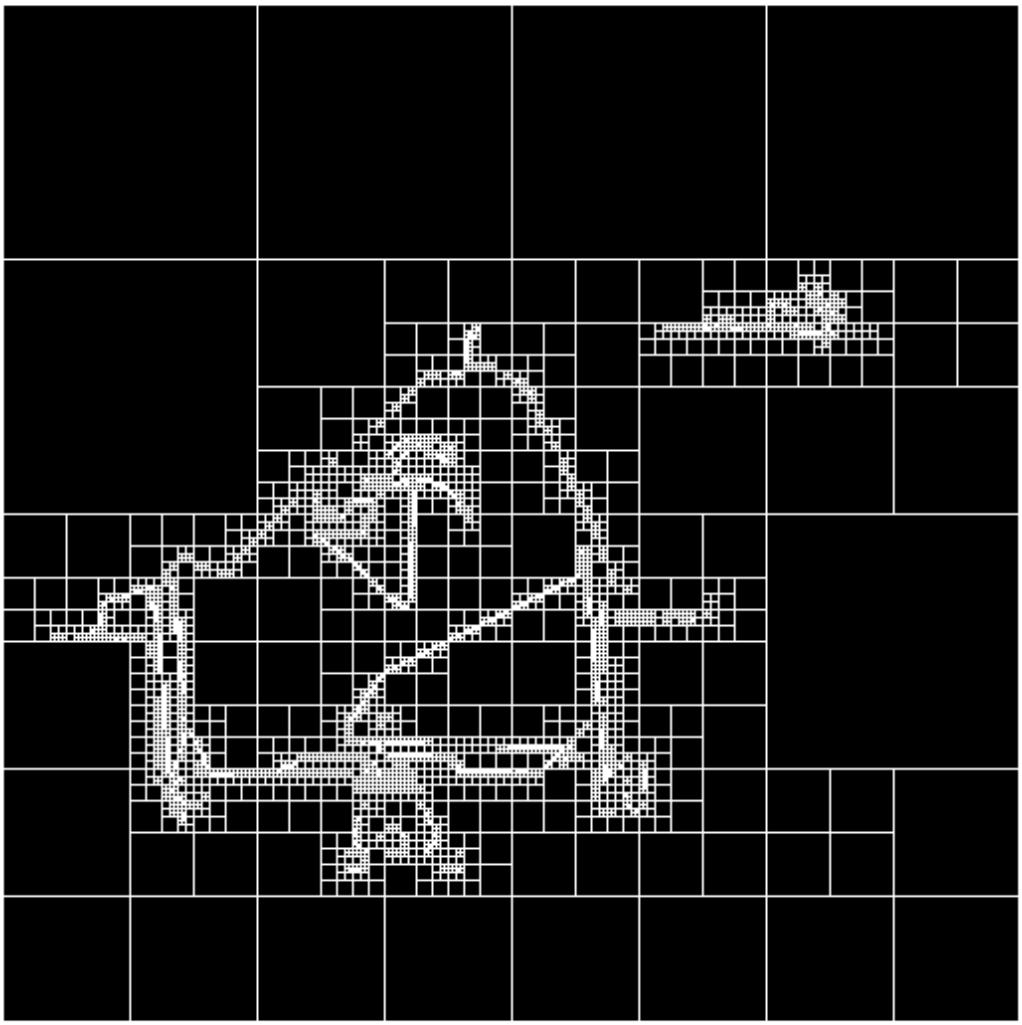
```
I = imread('liftingbody.png');
```

Perform the quadtree decomposition and display the block representation in a figure.

```
S = qtdecomp(I,.27);
blocks = repmat(uint8(0),size(S));

for dim = [512 256 128 64 32 16 8 4 2 1];
  numblocks = length(find(S==dim));
  if (numblocks > 0)
    values = repmat(uint8(1),[dim dim numblocks]);
    values(2:dim,2:dim,:) = 0;
    blocks = qtsetblk(blocks,S,dim,values);
  end
end

blocks(end,1:end) = 1;
blocks(1:end,end) = 1;

imshow(I)
```



```
figure
imshow(blocks,[])
```



Input Arguments

`I` — Grayscale image

m-by-*n* numeric matrix

Grayscale image, specified as an *m*-by-*n* numeric matrix. If the syntax includes a function handle, `fun`, then the image can be of any class supported by the function.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`threshold` — Threshold of block homogeneity

scalar in the range [0, 1]

Threshold of block homogeneity, specified as a scalar in the range [0, 1].

- If `I` is of class `uint8`, then `qtdecomp` multiplies the value of `threshold` by 255 to determine the actual threshold to use.
- If `I` is of class `uint8`, then `qtdecomp` multiplies the value of `threshold` by 65535 to determine the actual threshold to use.

`mindim` — Minimum block dimension
positive integer

Minimum block size, specified as a positive integer. `mindim` must be a factor of the image size.

`maxdim` — Maximum block dimension
positive integer

Maximum block size, specified as a positive integer. `maxdim/mindim` must be a power of 2.

`fun` — Function handle
handle

Function handle, specified as a handle. The function must accept as input all *m*-by-*m* blocks stacked into an *m*-by-*m*-by-*k* array, where *k* is the number of blocks. The function must return a logical *k*-element vector, whose values are `1` if the corresponding block should be split, and `0` otherwise. For example, if *k*(3) is `0`, then the third *m*-by-*m* block should not be split.

For more information about function handles, see “Create Function Handle”.

Output Arguments

`S` — Quadtree structure
sparse matrix

Quadtree structure, returned as a sparse matrix. If `S(k,m)` is nonzero, then `(k,m)` is the upper left corner of a block in the decomposition, and the size of the block is given by `S(k,m)`.

Data Types: `double`

Tips

- `qtdecomp` is appropriate primarily for square images whose dimensions are a power of 2, such as 128-by-128 or 512-by-512. These images can be divided until the blocks are as small as 1-by-1. If you use `qtdecomp` with an image whose dimensions are not a power of 2, at some point the blocks cannot be divided further. For example, if an image is 96-by-96, it can be divided into blocks of size 48-by-48, then 24-by-24, 12-by-12, 6-by-6, and finally 3-by-3. No further division beyond 3-by-3 is possible. To process this image, you must set `mindim` to 3 (or to 3 times a power of 2); if you are using the syntax that includes a function, `fun`, the function must return `0` at the point when the block cannot be divided further.

Algorithms

The `qtdecomp` function divides a square image into four equal-sized square blocks, and then tests each block to see if it meets some criterion of homogeneity. If a block meets the criterion, it is not divided any further. If it does not meet the criterion, it is subdivided again into four blocks, and the test criterion is applied to those blocks. This process is repeated iteratively until each block meets the criterion. The result can have blocks of several different sizes.

See Also

`qtgetblk` | `qtsetblk`

Introduced before R2006a

qtgetblk

Block values in quadtree decomposition

Syntax

```
[vals,r,c] = qtgetblk(I,S,dim)
[vals,idx] = qtgetblk(I,S,dim)
```

Description

`[vals,r,c] = qtgetblk(I,S,dim)` returns blocks of size `dim`-by-`dim` from image `I` with quadtree decomposition `S`. The function returns the block values in `vals` and the row and column coordinates of the upper left corner of the blocks in `r` and `c`.

`[vals,idx] = qtgetblk(I,S,dim)` returns the block values in `vals` and the linear indices of the upper left corners of the blocks in `idx`.

Examples

Get Blocks from Quadtree Decomposition

Create a sample matrix representing a small image.

```
I = [1    1    1    1    2    3    6    6
     1    1    2    1    4    5    6    8
     1    1    1    1   10   15    7    7
     1    1    1    1   20   25    7    7
    20   22   20   22    1    2    3    4
    20   22   22   20    5    6    7    8
    20   22   20   20    9   10   11   12
    22   22   20   20   13   14   15   16];
```

Perform a quadtree decomposition of the image, specifying a threshold of 5. `qtdecomp` splits a block if the maximum value of the block elements minus the minimum value of the block elements is greater than the threshold.

```
S = qtdecomp(I,5)
```

```
S =
   (1,1)        4
   (5,1)        4
   (1,5)        2
   (3,5)        1
   (4,5)        1
   (5,5)        2
   (7,5)        2
   (3,6)        1
   (4,6)        1
   (1,7)        2
   (3,7)        2
   (5,7)        2
```

```
   (7,7)        2
```

Get the blocks of size 4-by-4 from the quadtree decomposition. `qtgetblk` finds two blocks of this size.

```
[vals,r,c] = qtgetblk(I,S,4);
```

Select the second returned block. Display the values and the (row,column) coordinates of the upper left corner of the block.

```
blknum = 2;
blockValues = vals(:,:,blknum)
```

```
blockValues = 4×4

    20    22    20    22
    20    22    22    20
    20    22    20    20
    22    22    20    20
```

```
blockCoordinates = ['(',num2str(r(blknum)),',',num2str(c(blknum)),')']
```

```
blockCoordinates =
'(5,1)'
```

Input Arguments

`I` — Grayscale image

numeric matrix

Grayscale image, specified as a numeric matrix.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`S` — Quadtree structure

sparse matrix

Quadtree structure, specified as a sparse matrix. If `S`(*m*,*n*) is nonzero, then the coordinate (*m*,*n*) is the upper left corner of a block in the decomposition, and the size of the block is given by `S`(*m*,*n*). You can get a quadtree structure by using the `qtdecomp` function.

Data Types: `double`

`dim` — Block size

positive integer

Block size, specified as a positive integer.

Output Arguments

`vals` — Block values

`dim`-by-`dim`-by-*k* array | `[]`

Block values, returned as a `dim`-by-`dim`-by-*k* array, where *k* is the number of `dim`-by-`dim` blocks in the quadtree decomposition. If the quadtree decomposition contains no blocks of the specified size, then `vals` is returned as an empty matrix.

The ordering of the blocks in `vals` matches the column-wise order of the blocks in `I`. For example, if `vals` is 4-by-4-by-2, then `vals(:,:,1)` contains the values from the first 4-by-4 block in `I`, and `vals(:,:,2)` contains the values from the second 4-by-4 block.

`r` — Row coordinates of upper left corners of blocks
k-element column vector | `[ ]`

Row coordinates of the upper left corners of blocks, returned as a *k*-element column vector of positive integers, where *k* is the number of `dim`-by-`dim` blocks in the quadtree decomposition. If the quadtree decomposition contains no blocks of the specified size, then `r` is returned as an empty matrix.

`c` — Column coordinates of upper left corners of blocks
k-element column vector | `[ ]`

Column coordinates of the upper left corners of blocks, returned as a *k*-element column vector of positive integers, where *k* is the number of `dim`-by-`dim` blocks in the quadtree decomposition. If the quadtree decomposition contains no blocks of the specified size, then `c` is returned as an empty matrix.

`idx` — Linear indices of upper left corners of blocks
k-element column vector | `[ ]`

Linear indices of upper left corners of blocks, returned as a *k*-element column vector of positive integers, where *k* is the number of `dim`-by-`dim` blocks in the quadtree decomposition. If the quadtree decomposition contains no blocks of the specified size, then `idx` is returned as an empty matrix.

See Also

`qtdecomp` | `qtsetblk`

Topics
“Quadtree Decomposition”

Introduced before R2006a

qtsetblk

Set block values in quadtree decomposition

Syntax

```
J = qtsetblk(I,S,dim,vals)
```

Description

`J = qtsetblk(I,S,dim,vals)` replaces each `dim`-by-`dim` block in the quadtree decomposition of image `I` with the corresponding block in `vals`. `S` contains the quadtree structure.

Examples

Set Blocks in Quadtree Decomposition

Create a sample matrix representing a small image.

```
I = [1    1    1    1    2    3    6    6
     1    1    2    1    4    5    6    8
     1    1    1    1   10   15    7    7
     1    1    1    1   20   25    7    7
    20   22   20   22    1    2    3    4
    20   22   22   20    5    6    7    8
    20   22   20   20    9   10   11   12
    22   22   20   20   13   14   15   16];
```

Perform a quadtree decomposition of the image, specifying a threshold of 5. `qtdecomp` splits a block if the maximum value of the block elements minus the minimum value of the block elements is greater than the threshold.

```
S = qtdecomp(I,5);
```

Get the blocks of size 4-by-4 from the quadtree decomposition.

```
vals = qtgetblk(I,S,4);
```

Calculate the mode of each 4-by-4 block, and set all values to equal the mode.

```
valmodes = zeros(size(vals));
for blknum = 1:size(vals,3)
    valmodes(:,:,blknum) = mode(vals(:,:,blknum),'all');
end
```

Set the blocks in the image to the new values. The 4-by-4 blocks in the image are now homogenous.

```
J = qtsetblk(I,S,4,valmodes)
```

```
J = 8×8

     1     1     1     1     2     3     6     6
```

```
     1     1     1     1     4     5     6     8
     1     1     1     1    10    15     7     7
     1     1     1     1    20    25     7     7
    20    20    20    20     1     2     3     4
    20    20    20    20     5     6     7     8
    20    20    20    20     9    10    11    12
    20    20    20    20    13    14    15    16
```

Input Arguments

`I` — Grayscale image

numeric matrix

Grayscale image, specified as a numeric matrix.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16` | `logical`

`S` — Quadtree structure

sparse matrix

Quadtree structure, specified as a sparse matrix. If `S`(*m*,*n*) is nonzero, then the coordinate (*m*,*n*) is the upper left corner of a block in the decomposition, and the size of the block is given by `S`(*m*,*n*). You can get a quadtree structure by using the `qtdecomp` function.

Data Types: `double`

`dim` — Block size

positive integer

Block size, specified as a positive integer.

`vals` — Block values

`dim`-by-`dim`-by-*k* array

Block values, specified as a `dim`-by-`dim`-by-*k* array, where *k* is the number of `dim`-by-`dim` blocks in the quadtree decomposition.

The ordering of the blocks in `vals` must match the column-wise order of the blocks in `I`. For example, if `vals` is 4-by-4-by-2, then `vals(:,:,1)` contains the values used to replace the first 4-by-4 block in `I`, and `vals(:,:,2)` contains the values used to replace the second 4-by-4 block.

See Also

`qtdecomp` | `qtgetblk`

Introduced before R2006a

radon

Radon transform

Syntax

```
R = radon(I)
R = radon(I,theta)
[R,xp] = radon( ___ )
```

Description

`R = radon(I)` returns the Radon transform `R` of 2-D grayscale image `I` for angles in the range [0, 179] degrees. The Radon transform is the projection of the image intensity along a radial line oriented at a specific angle.

`R = radon(I,theta)` returns the Radon transform for the angles specified by `theta`.

`[R,xp] = radon( ___ )` returns a vector `xp` containing the radial coordinates corresponding to each row of the image.

Examples

Calculate Radon Transform and Display Plot

Make the axes scale visible for this image.

```
iptsetpref('ImshowAxesVisible','on')
```

Create a sample image.

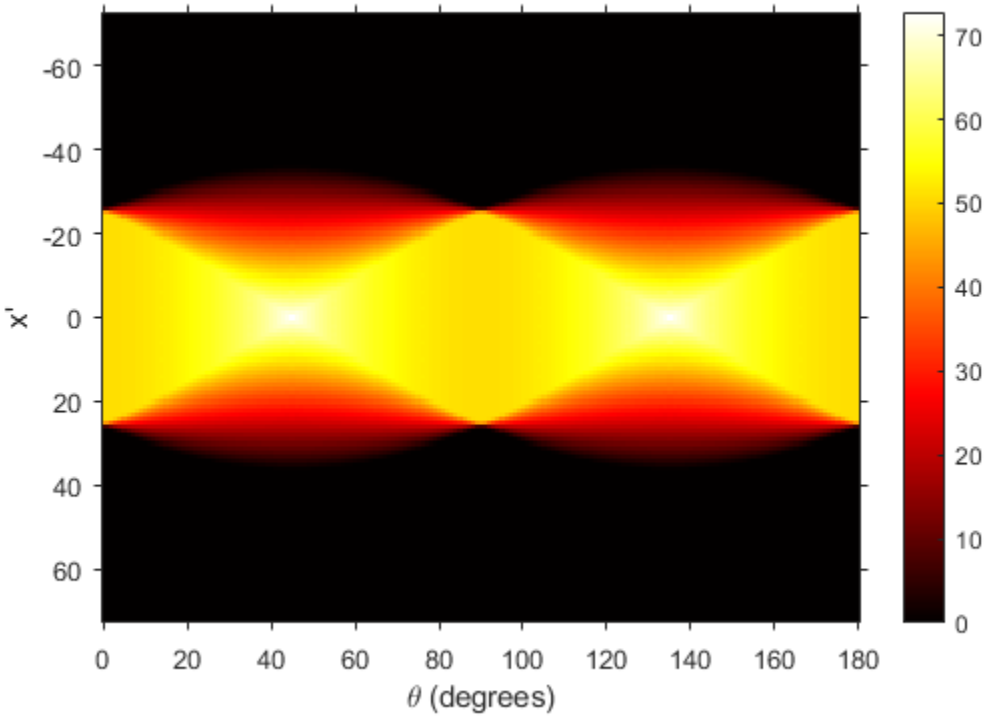
```
I = zeros(100,100);
I(25:75, 25:75) = 1;
```

Calculate the Radon transform.

```
theta = 0:180;
[R,xp] = radon(I,theta);
```

Display the transform.

```
imshow(R,[],'Xdata',theta,'Ydata',xp,'InitialMagnification','fit')
xlabel('\theta (degrees)')
ylabel('x''')
colormap(gca,hot), colorbar
```



Make the axes scale invisible.

```
iptsetpref('ImshowAxesVisible','off')
```

Input Arguments

`I` — Grayscale image

2-D numeric matrix

Grayscale image, specified as a 2-D numeric matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`theta` — Projection angles

`0:179` (default) | numeric scalar | numeric vector

Projection angles in degrees, specified as a numeric scalar or numeric vector.

Data Types: `double`

Output Arguments

`R` — Radon transform

numeric column vector | numeric matrix

Radon transform of image `I`, returned as one of the following.

- If `theta` is a scalar, then `R` is a numeric column vector containing the Radon transform for `theta` degrees.
- If `theta` is a vector, then `R` is a matrix in which each column is the Radon transform for one of the angles in `theta`.

xp — Radial coordinates
numeric vector

Radial coordinates corresponding to each row of `R`, returned as a numeric vector. The radial coordinates are the values along the *x*'-axis, which is oriented at `theta` degrees counterclockwise from the *x*-axis. The origin of both axes is the center pixel of the image, which is defined as

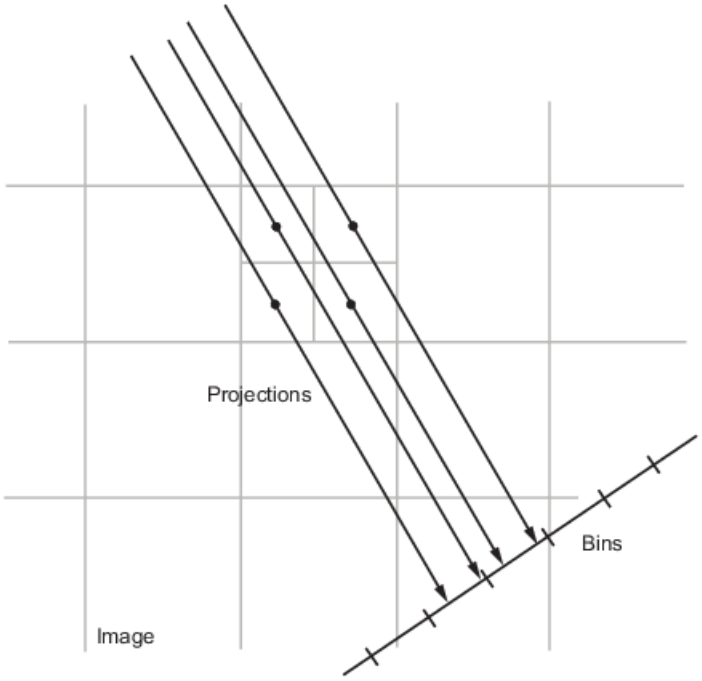
```
floor((size(I)+1)/2)
```

For example, in a 20-by-30 image, the center pixel is (10,15).

Algorithms

The Radon transform of an image is the sum of the Radon transforms of each individual pixel.

The algorithm first divides pixels in the image into four subpixels and projects each subpixel separately, as shown in the following figure.



Each subpixel's contribution is proportionally split into the two nearest bins, according to the distance between the projected location and the bin centers. If the subpixel projection hits the center point of a bin, the bin on the axes gets the full value of the subpixel, or one-fourth the value of the pixel. If the subpixel projection hits the border between two bins, the subpixel value is split evenly between the bins.

References

[1] Bracewell, Ronald N., *Two-Dimensional Imaging*, Englewood Cliffs, NJ, Prentice Hall, 1995, pp. 505-537.

[2] Lim, Jae S., *Two-Dimensional Signal and Image Processing*, Englewood Cliffs, NJ, Prentice Hall, 1990, pp. 42-45.

Extended Capabilities

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`fan2para` | `fanbeam` | `ifanbeam` | `iradon` | `para2fan` | `phantom`

Introduced before R2006a

randomAffine2d

Create randomized 2-D affine transformation

Syntax

```
tform = randomAffine2d
tform = randomAffine2d(Name,Value)
```

Description

`tform = randomAffine2d` creates an `affine2d` object with default property values consistent with the identity transformation.

`tform = randomAffine2d(Name,Value)` specifies the type of affine transformations using name-value pair arguments.

Examples

Randomly Rotate Image

Read and display an image.

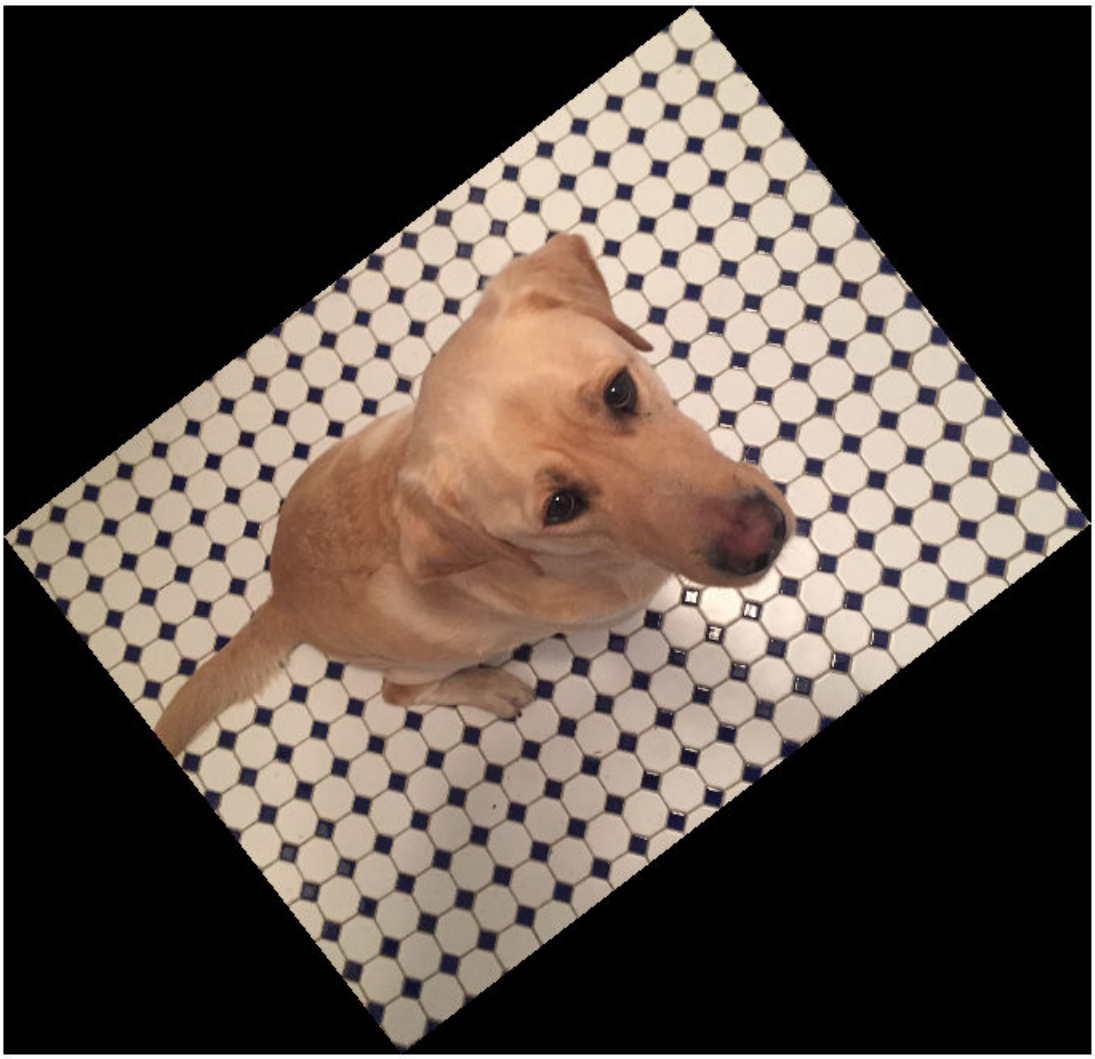
```
I = imread('kobi.png');
imshow(I)
```

Create an `affine2d` transformation object that rotates images. The `randomAffine2d` function picks a rotation angle randomly from a continuous uniform distribution within the interval [35, 55] degrees.

```
tform1 = randomAffine2d('Rotation',[35 55]);
```

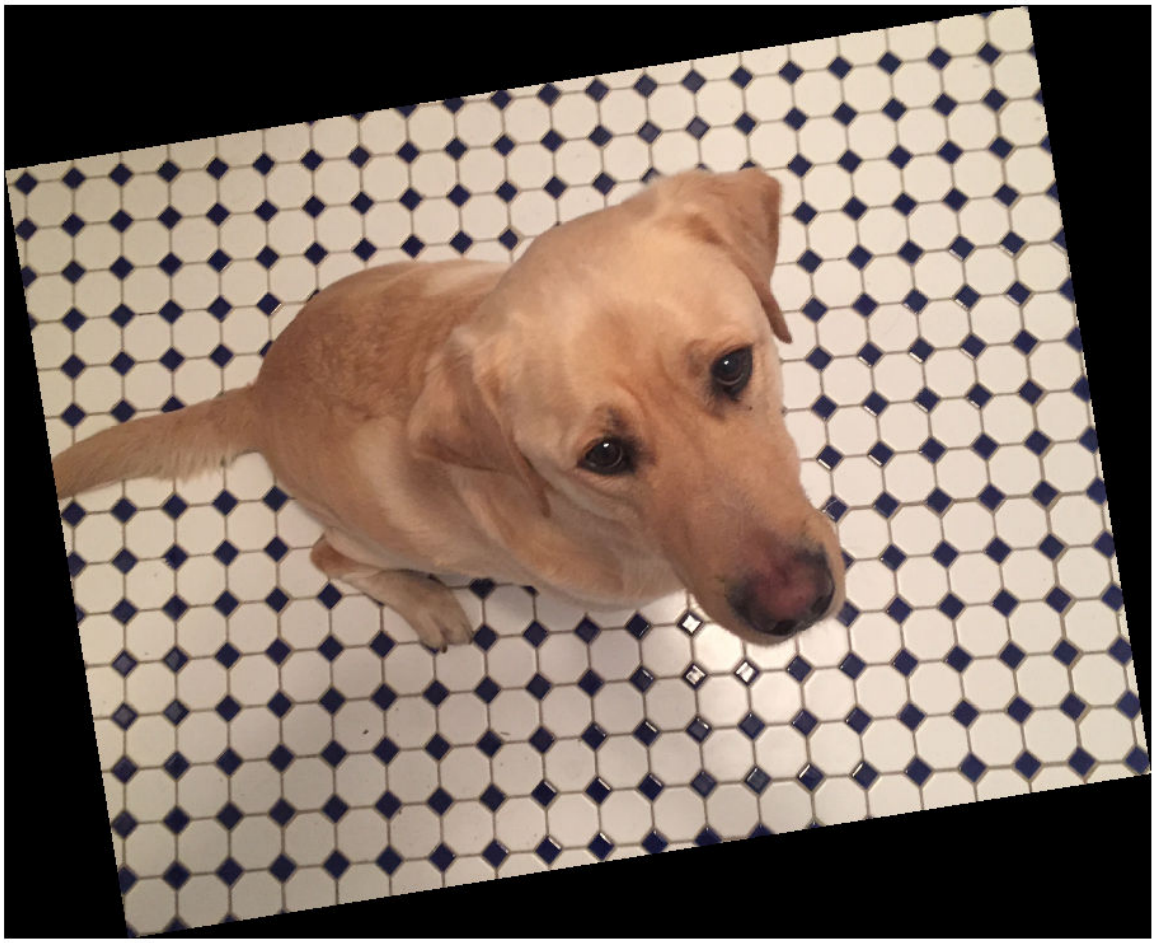
Rotate the image and display the result.

```
J = imwarp(I,tform1);
imshow(J)
```



The transformation object, `tform1`, rotates all images by the same amount. To rotate an image by a different randomly selected amount, create a new `affine2d` transformation object.

```
tform2 = randomAffine2d('Rotation',[-10 10]);
J2 = imwarp(I,tform2);
imshow(J2)
```



Randomly Rotate Image with Custom Rotation Range

Read and display an image.

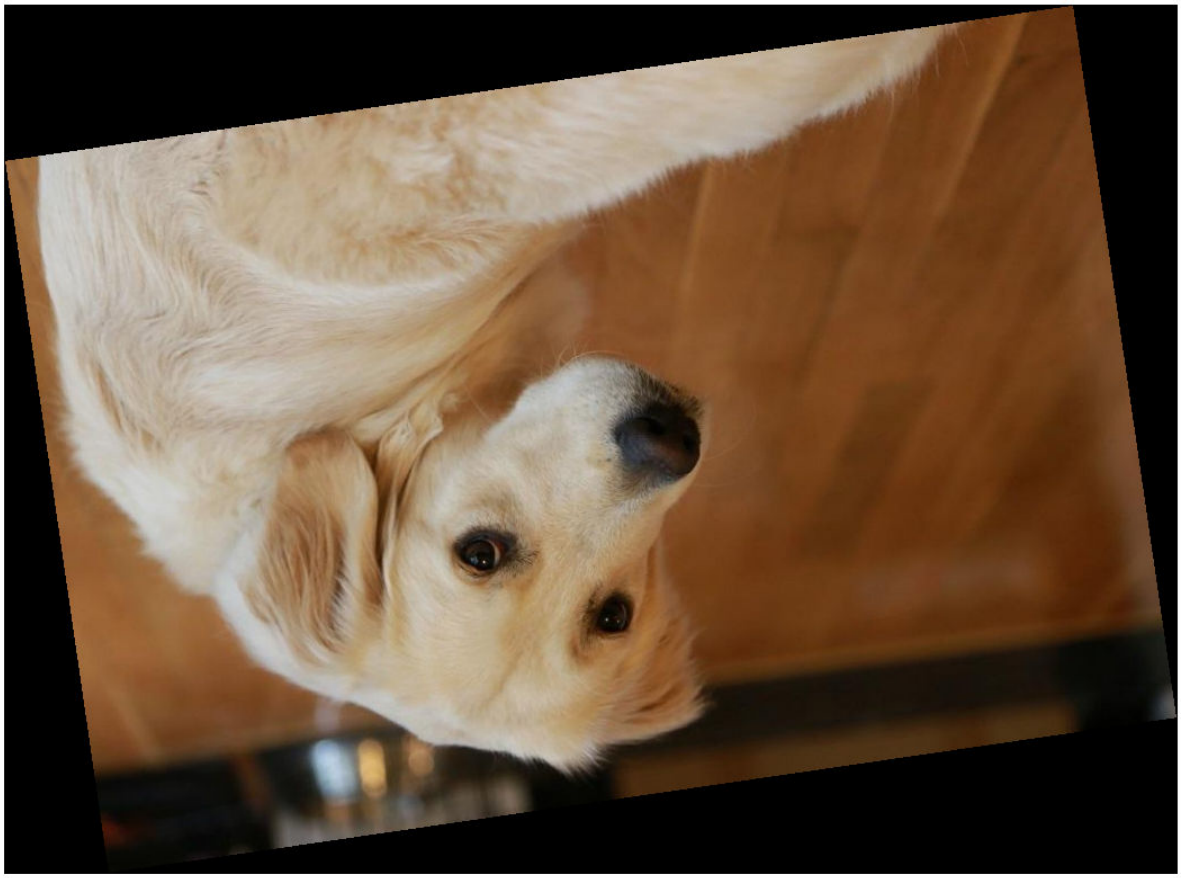
```
I = imread('sherlock.jpg');
imshow(I)
```

Create an `affine2d` transformation object that rotates images. To select a rotation angle from a custom range, specify the `'Rotation'` name-value pair argument as a function handle. This example specifies a function called `myrange` (defined at the end of the example) that selects an angle from within two disjoint intervals.

```
tform = randomAffine2d('Rotation',@myrange);
```

Rotate the image and display the result.

```
J = imwarp(I,tform);
imshow(J)
```



Supporting Function

This example defines the `myrange` function that first randomly selects one of two intervals (-10, 10) and (170, 190) with equal probability. Within the selected interval, the function returns a single random number from a uniform distribution.

```
function angle = myrange()
    if randi([0 1],1)
        a = -10;
        b = 10;
    else
        a = 170;
        b = 190;
    end
```

```
    angle = a + (b-a).*rand(1);
end
```

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `tform = randomAffine2d('XReflection',true)`

`XReflection` — Random horizontal reflection
`false` (default) | `true`

Random horizontal reflection, specified as the comma-separated pair consisting of `'XReflection'` and `false` or `true`. When `XReflection` is `true` (`1`), the transformation `tform` reflects images horizontally with 50% probability. By default, the transformation does not reflect images in the horizontal direction.

`YReflection` — Random vertical reflection
`false` (default) | `true`

Random vertical reflection, specified as the comma-separated pair consisting of `'YReflection'` and `false` or `true`. When `YReflection` is `true` (`1`), the transformation `tform` reflects images vertically with 50% probability. By default, the transformation does not reflect images in the vertical direction.

`Rotation` — Range of rotation
`[0 0]` (default) | 2-element numeric vector | function handle

Range of rotation, in degrees, applied to the input image, specified as the comma-separated pair consisting of `'Rotation'` and one of the following.

- 2-element numeric vector. The second element must be larger than or equal to the first element. The rotation angle is picked randomly from a continuous uniform distribution within the specified interval.
- function handle. The function must accept no input arguments and return the rotation angle as a numeric scalar. Use a function handle to pick rotation angles from a disjoint interval or using a nonuniform probability distribution. For more information about function handles, see “Create Function Handle”.

By default, the transformation `tform` does not rotate images.

Example: `[-45 45]`

`Scale` — Range of uniform scaling
`[1 1]` (default) | 2-element numeric vector | function handle

Range of uniform (isotropic) scaling applied to the input image, specified as the comma-separated pair consisting of `'Scale'` and one of the following.

- 2-element numeric vector. The second element must be larger than or equal to the first element. The scale factor is picked randomly from a continuous uniform distribution within the specified interval.
- function handle. The function must accept no input arguments and return the scale factor as a numeric scalar. Use a function handle to pick scale factors from a disjoint interval or using a nonuniform probability distribution. For more information about function handles, see “Create Function Handle”.

By default, the transformation `tform` does not scale images.

Example: `[0.5 4]`

`XShear` — Range of horizontal shear
`[0 0]` (default) | 2-element numeric vector | function handle

Range of horizontal shear applied to the input image, specified as the comma-separated pair consisting of `'XShear'` and one of the following. Shear is measured as an angle in degrees, and is in the range (–90, 90).

- 2-element numeric vector. The second element must be larger than or equal to the first element. The horizontal shear angle is picked randomly from a continuous uniform distribution within the specified interval.
- function handle. The function must accept no input arguments and return the horizontal shear angle as a numeric scalar. Use a function handle to pick horizontal shear angles from a disjoint interval or using a nonuniform probability distribution. For more information about function handles, see “Create Function Handle”.

By default, the transformation `tform` does not shear images in the horizontal direction.

Example: `[0 45]`

`YShear` — Range of vertical shear
`[0 0]` (default) | 2-element numeric vector | function handle

Range of vertical shear applied to the input image, specified as the comma-separated pair consisting of `'YShear'` and one of the following. Shear is measured as an angle in degrees, and is in the range (–90, 90).

- 2-element numeric vector. The second element must be larger than or equal to the first element. The vertical shear angle is picked randomly from a continuous uniform distribution within the specified interval.
- function handle. The function must accept no input arguments and return the vertical shear angle as a numeric scalar. Use a function handle to pick vertical shear angles from a disjoint interval or using a nonuniform probability distribution. For more information about function handles, see “Create Function Handle”.

By default, the transformation `tform` does not shear images in the vertical direction.

Example: `[0 45]`

`XTranslation` — Range of horizontal translation
`[0 0]` (default) | 2-element numeric vector | function handle

Range of horizontal translation applied to the input image, specified as the comma-separated pair consisting of `'XTranslation'` and one of the following. Translation distance is measured in pixels.

- 2-element numeric vector. The second element must be larger than or equal to the first element. The horizontal translation distance is picked randomly from a continuous uniform distribution within the specified interval.
- function handle. The function must accept no input arguments and return the horizontal translation distance as a numeric scalar. Use a function handle to pick horizontal translation distances from a disjoint interval or using a nonuniform probability distribution. For more information about function handles, see “Create Function Handle”.

By default, the transformation `tform` does not translate images in the horizontal direction.

Example: `[-5 5]`

`YTranslation` — Range of vertical translation

`[0 0]` (default) | 2-element numeric vector | function handle

Range of vertical translation applied to the input image, specified as the comma-separated pair consisting of `'YTranslation'` and one of the following. Translation distance is measured in pixels.

- 2-element numeric vector. The second element must be larger than or equal to the first element. The vertical translation distance is picked randomly from a continuous uniform distribution within the specified interval.
- function handle. The function must accept no input arguments and return the vertical translation distance as a numeric scalar. Use a function handle to pick vertical translation distances from a disjoint interval or using a nonuniform probability distribution. For more information about function handles, see “Create Function Handle”.

By default, the transformation `tform` does not translate images in the vertical direction.

Example: `[-5 5]`

Output Arguments

`tform` — Affine transformation

`affine2d` object

Affine transformation, specified as an `affine2d` object.

See Also

`imwarp` | `randomAffine3d` | `randomWindow2d` | `centerCropWindow2d`

Topics

“Augment Images for Deep Learning Workflows Using Image Processing Toolbox” (Deep Learning Toolbox)

Introduced in R2019b

randomAffine3d

Create randomized 3-D affine transformation

Syntax

```
tform = randomAffine3d
tform = randomAffine3d(Name,Value)
```

Description

`tform = randomAffine3d` creates an `affine3d` object with default property values consistent with the identity transformation.

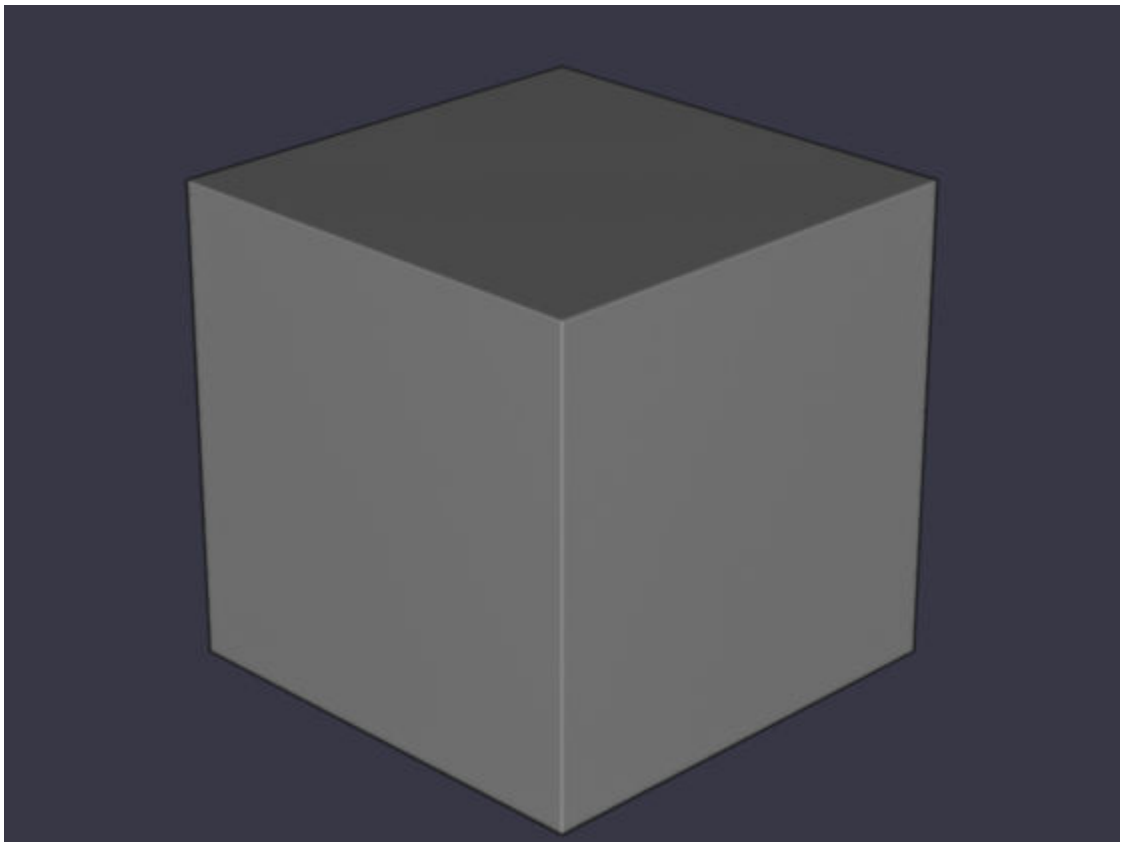
`tform = randomAffine3d(Name,Value)` specifies the type of affine transformations using name-value pair arguments.

Examples

Randomly Shear 3-D Volume

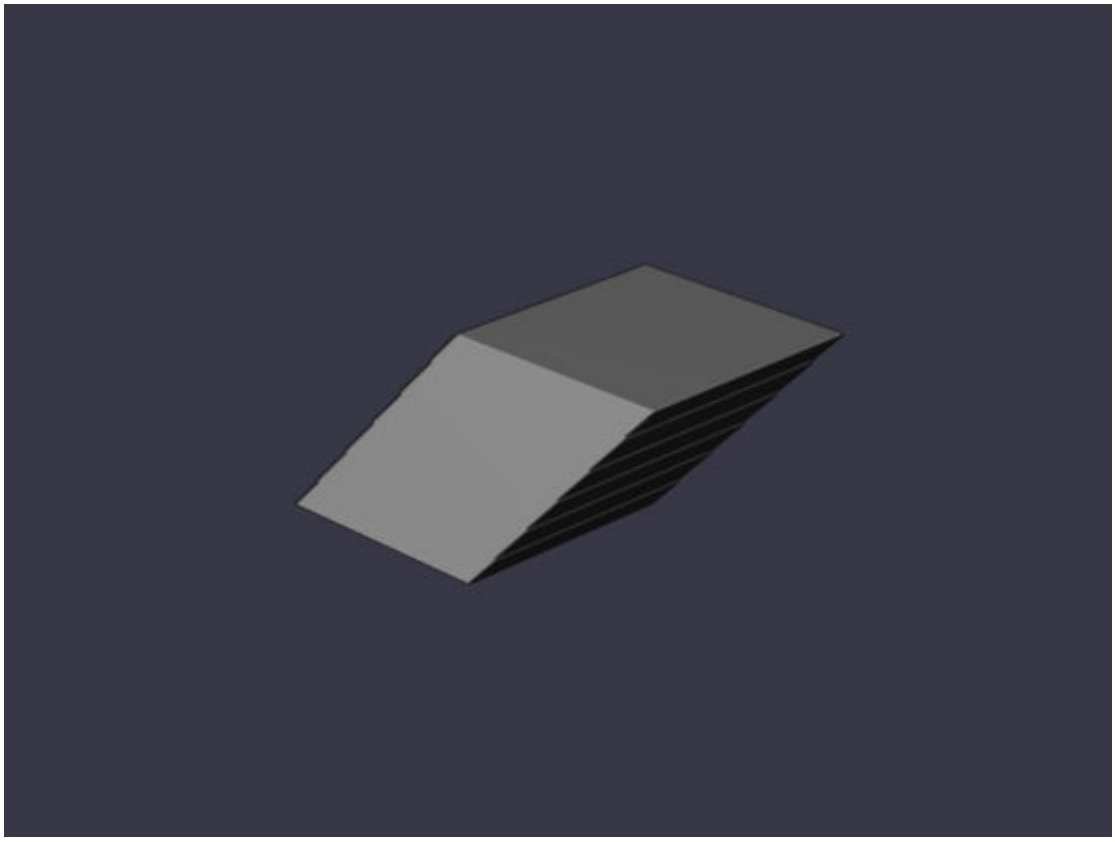
Create a sample volume.

```
volumeCube = ones(100,100,100);
figure
volshow(volumeCube);
```



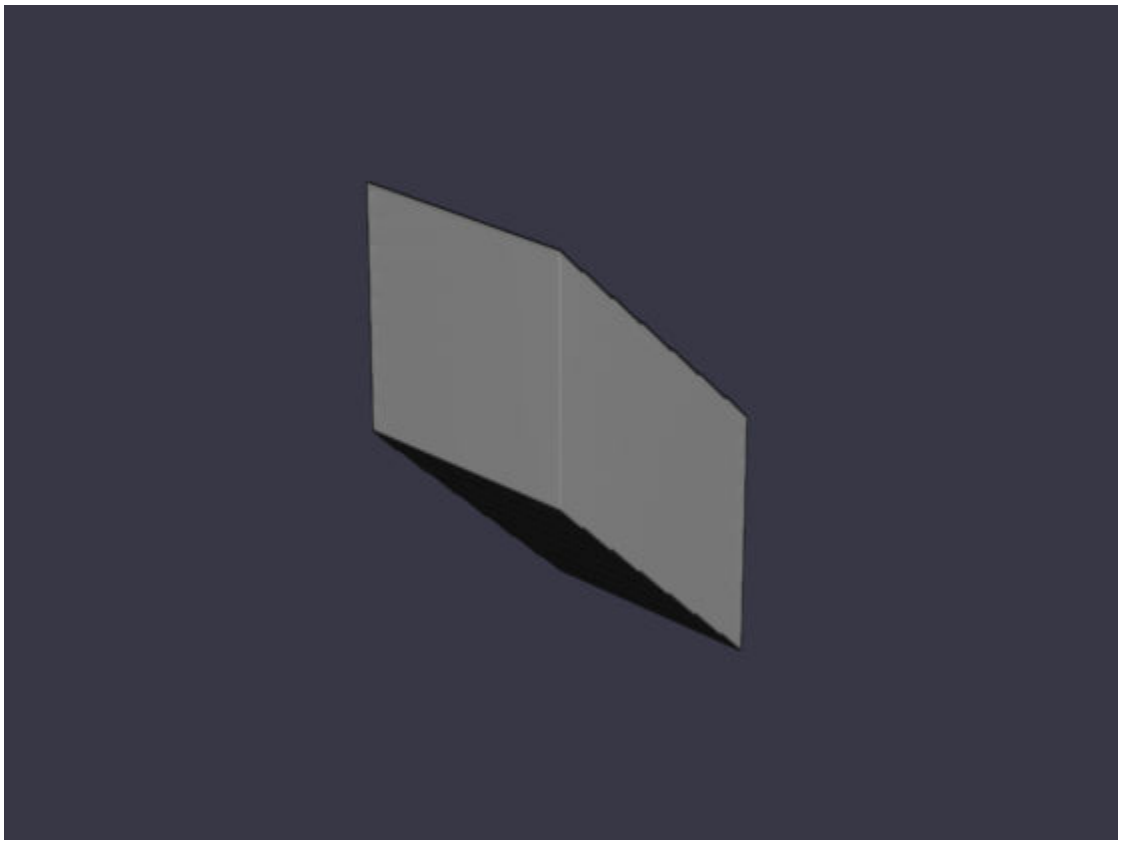
Create an `affine3d` transformation object that shears 3-D volumes. The `randomAffine3d` function picks a shear amount randomly from a continuous uniform distribution within the interval [40, 60] degrees. `randomAffine3d` picks a random shear direction aligned with the *x*-, *y*-, or *z*-axis.

```
tform1 = randomAffine3d('Shear',[40 60]);
J1 = imwarp(volumeCube,tform1);
figure
volshow(J1);
```



To shear a volume by a different randomly selected amount, create a new `affine3d` transformation object. Note the difference in the shear direction.

```
tform2 = randomAffine3d('Shear',[40 60]);
J2 = imwarp(volumeCube,tform2);
figure
volshow(J2);
```



Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `tform = randomAffine3d('XReflection',true)`

`XReflection` — Random horizontal reflection
`false` (default) | `true`

Random horizontal reflection, specified as the comma-separated pair consisting of `'XReflection'` and `false` or `true`. When `XReflection` is `true` (1), the transformation `tform` reflects images horizontally with 50% probability. By default, the transformation does not reflect images in the horizontal direction.

`YReflection` — Random vertical reflection
`false` (default) | `true`

Random vertical reflection, specified as the comma-separated pair consisting of `'YReflection'` and `false` or `true`. When `YReflection` is `true` (1), the transformation `tform` reflects images vertically with 50% probability. By default, the transformation does not reflect images in the vertical direction.

`ZReflection` — Random reflection along depth
`false` (default) | `true`

Random reflection along the depth direction, specified as the comma-separated pair consisting of `'ZReflection'` and `false` or `true`. When `ZReflection` is `true` (`1`), the transformation `tform` reflects images along the depth direction with 50% probability. By default, the transformation does not reflect images in the depth direction.

`Rotation` — Range of rotation
`[0 0]` (default) | 2-element numeric vector | function handle

Range of rotation applied to the input image, specified as the comma-separated pair consisting of `'Rotation'` and one of the following. Rotation is measured in degrees.

- 2-element numeric vector. The second element must be larger than or equal to the first element. `randomAffine3d` picks a rotation angle randomly from a continuous uniform distribution within the specified interval. `randomAffine3d` selects a random axis of rotation from the unit sphere.
- function handle of the form

  ```
  [rotationAxis,theta] = selectRotation
  ```

  The function `selectRotation` must accept no input arguments. The function must return two output arguments: `rotationAxis`, a 3-element vector defining the axis of rotation, and `theta`, a rotation angle in degrees.

  Use a function handle to pick rotation angles from a disjoint interval or using a nonuniform probability distribution. You can also use a function handle to specify an axis of rotation. For more information about function handles, see “Create Function Handle”.

By default, the transformation `tform` does not rotate images.

Example: `[-45 45]`

`Scale` — Range of uniform scaling
`[1 1]` (default) | 2-element numeric vector | function handle

Range of uniform (isotropic) scaling applied to the input image, specified as the comma-separated pair consisting of `'Scale'` and one of the following.

- 2-element numeric vector. The second element must be larger than or equal to the first element. The scale factor is picked randomly from a continuous uniform distribution within the specified interval.
- function handle. The function must accept no input arguments and return the scale factor as a numeric scalar. Use a function handle to pick scale factors from a disjoint interval or using a nonuniform probability distribution. For more information about function handles, see “Create Function Handle”.

By default, the transformation `tform` does not scale images.

Example: `[0.5 4]`

`Shear` — Range of shear
`[0 0]` (default) | 2-element numeric vector | function handle

Range of shear applied to the input image, specified as the comma-separated pair consisting of `'Shear'` and one of the following. Shear is measured as an angle in degrees, and is in the range (–90, 90).

- 2-element numeric vector. The second element must be larger than or equal to the first element. The shear angle is picked randomly from a continuous uniform distribution within the specified interval. `randomAffine3d` applies shear with uniform randomness to one of the principle *x*-, *y*-, and *z*-directions with respect to one of the two possible orthogonal directions.
- function handle. The function must accept no input arguments and return the shear angle as a numeric scalar. Use a function handle to pick a shear angle from a disjoint interval or using a nonuniform probability distribution. For more information about function handles, see “Create Function Handle”.

By default, the transformation `tform` does not shear images in the horizontal direction.

Example: `[0 45]`

`XTranslation` — Range of horizontal translation
`[0 0]` (default) | 2-element numeric vector | function handle

Range of horizontal translation applied to the input image, specified as the comma-separated pair consisting of `'XTranslation'` and one of the following. Translation distance is measured in pixels.

- 2-element numeric vector. The second element must be larger than or equal to the first element. The translation distance is picked randomly from a continuous uniform distribution within the specified interval.
- function handle. The function must accept no input arguments and return the translation distance as a numeric scalar. Use a function handle to pick a translation distance from a disjoint interval or using a nonuniform probability distribution. For more information about function handles, see “Create Function Handle”.

By default, the transformation `tform` does not translate images in the horizontal direction.

Example: `[-5 5]`

`YTranslation` — Range of vertical translation
`[0 0]` (default) | 2-element numeric vector | function handle

Range of vertical translation applied to the input image, specified as the comma-separated pair consisting of `'YTranslation'` and one of the following. Translation distance is measured in pixels.

- 2-element numeric vector. The second element must be larger than or equal to the first element. The translation distance is picked randomly from a continuous uniform distribution within the specified interval.
- function handle. The function must accept no input arguments and return the translation distance as a numeric scalar. Use a function handle to pick a translation distance from a disjoint interval or using a nonuniform probability distribution. For more information about function handles, see “Create Function Handle”.

By default, the transformation `tform` does not translate images in the vertical direction.

Example: `[-5 5]`

`ZTranslation` — Range of translation along depth
`[0 0]` (default) | 2-element numeric vector | function handle

Range of translation along the depth direction applied to the input image, specified as the comma-separated pair consisting of `'ZTranslation'` and one of the following. Translation distance is measured in pixels.

- 2-element numeric vector. The second element must be larger than or equal to the first element. The translation distance is picked randomly from a continuous uniform distribution within the specified interval.
- function handle. The function must accept no input arguments and return the translation distance as a numeric scalar. Use a function handle to pick a translation distance from a disjoint interval or using a nonuniform probability distribution. For more information about function handles, see “Create Function Handle”.

By default, the transformation `tform` does not translate images in the depth direction.

Example: `[-5 5]`

Output Arguments

`tform` — Affine transformation

`affine3d` object

Affine transformation, specified as an `affine3d` object.

See Also

`imwarp` | `randomAffine2d` | `randomCropWindow3d` | `centerCropWindow3d`

Topics

“Augment Images for Deep Learning Workflows Using Image Processing Toolbox” (Deep Learning Toolbox)

Introduced in R2019b

randomCropWindow2d

(Not recommended) Create randomized rectangular cropping window

Note `randomCropWindow2d` is not recommended. Use `randomWindow2d` instead. For more information, see “Compatibility Considerations”.

Syntax

```
win = randomCropWindow2d(inputSize,targetSize)
```

Description

`win = randomCropWindow2d(inputSize,targetSize)` determines the window to crop from a 2-D input image of size `inputSize` such that the size of the cropped image is `targetSize`. The coordinates of the window are selected from a random position in the input image.

Examples

Randomly Crop Image To Target Size

Read and display an image.

```
A = imread('kobi.png');
imshow(A)
```

Specify the target size of the cropping window.

```
targetSize = [1000 1000];
```

Create three random crop windows. Each window has a different position from the input image.

```
win1 = randomCropWindow2d(size(A),targetSize);
win2 = randomCropWindow2d(size(A),targetSize);
win3 = randomCropWindow2d(size(A),targetSize);
```

Crop the original image using each of the random crop windows.

```
B1 = imcrop(A,win1);
B2 = imcrop(A,win2);
B3 = imcrop(A,win3);
```

Display the three cropped images as a montage.

```
montage({B1,B2,B3},'Size',[1 3]);
```

Input Arguments

`inputSize` — Input image size

2-element vector of positive integers | 3-element vector of positive integers

Input image size, specified as one of the following.

Type of Input Image	Format of `inputSize`
2-D grayscale or binary image	2-element vector of positive integers of the form `[height width]`
2-D RGB or multispectral image of size	3-element vector of positive integers of the form `[height width channels]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`targetSize` — Target image size

2-element vector of positive integers | 3-element vector of positive integers

Target image size, specified as one of the following.

Type of Target Image	Format of `targetSize`
2-D grayscale or binary image	2-element vector of positive integers of the form `[height width]`
2-D RGB or multispectral image of size	3-element vector of positive integers of the form `[height width channels]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`win` — Cropping window

`Rectangle` object

Cropping window, returned as a `Rectangle` object.

Compatibility Considerations

`randomCropWindow2d` is not recommended
Not recommended starting in R2021a

`randomCropWindow2d` is limited to selecting regions of fixed size. In R2021a, the `randomWindow2d` function was introduced. This function enables randomizing the size and shape of the cropped region.

To update your code, change instances of the function name `randomCropWindow2d` to `randomWindow2d`. You do not need to change the input arguments.

There are no plans to remove `randomCropWindow2d` at this time.

See Also

`centerCropWindow2d` | `randomCropWindow3d` | `imcrop`

Topics
"Augment Images for Deep Learning Workflows Using Image Processing Toolbox" (Deep Learning Toolbox)

Introduced in R2019b

randomCropWindow3d

Create randomized cuboidal cropping window

Syntax

```
win = randomCropWindow3d(inputSize,targetSize)
```

Description

`win = randomCropWindow3d(inputSize,targetSize)` determines the window to crop from a 3-D input image of size `inputSize` such that the size of the cropped image is `targetSize`. The coordinates of the window are selected from a random position in the input image.

Examples

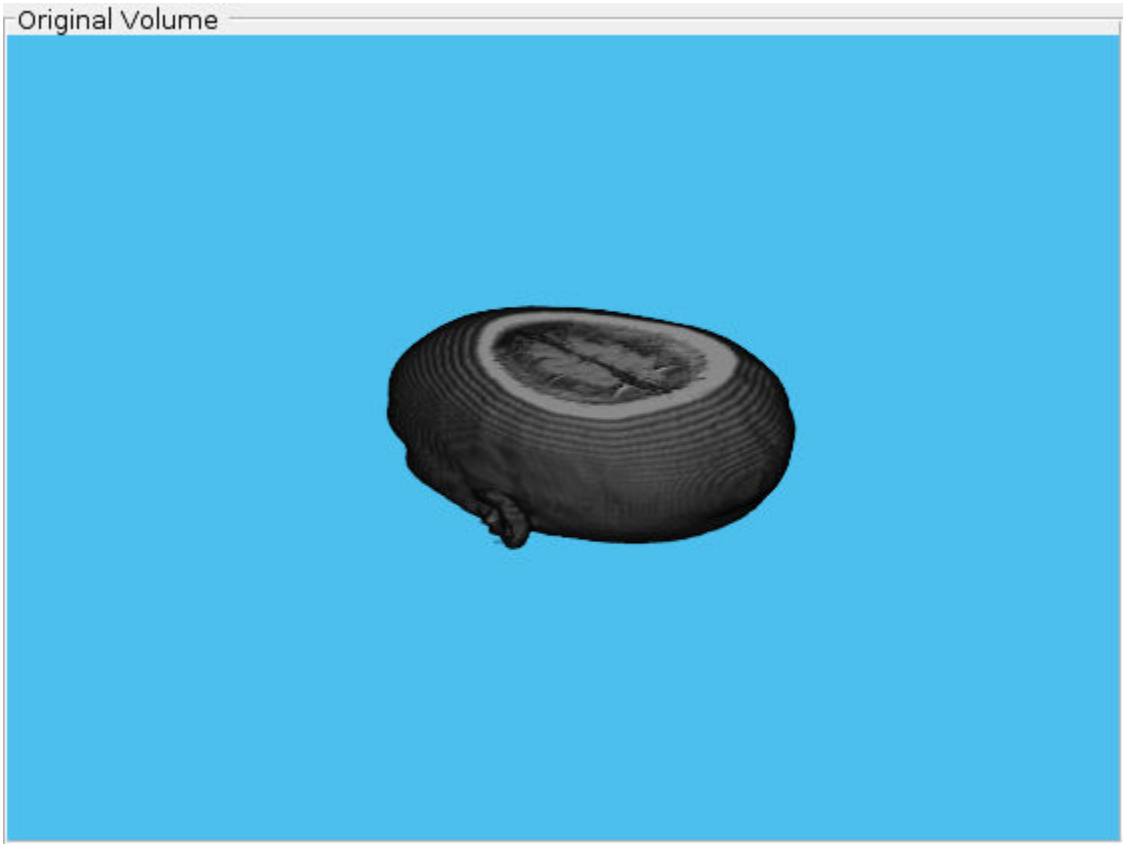
Randomly Crop 3-D Image Volume To Target Size

Read a 3-D MRI volume. Use the `squeeze` function to remove any singleton dimensions.

```
load mri;
D = squeeze(D);
```

Display the volume in a display panel.

```
fullViewPnl = uipanel(figure,'Title','Original Volume');
volshow(D,'Parent',fullViewPnl);
```



Specify the target size of the cropping window.

```
targetSize = [64 64 10];
```

Create a random crop window that crops the input volume from a randomly-selected position.

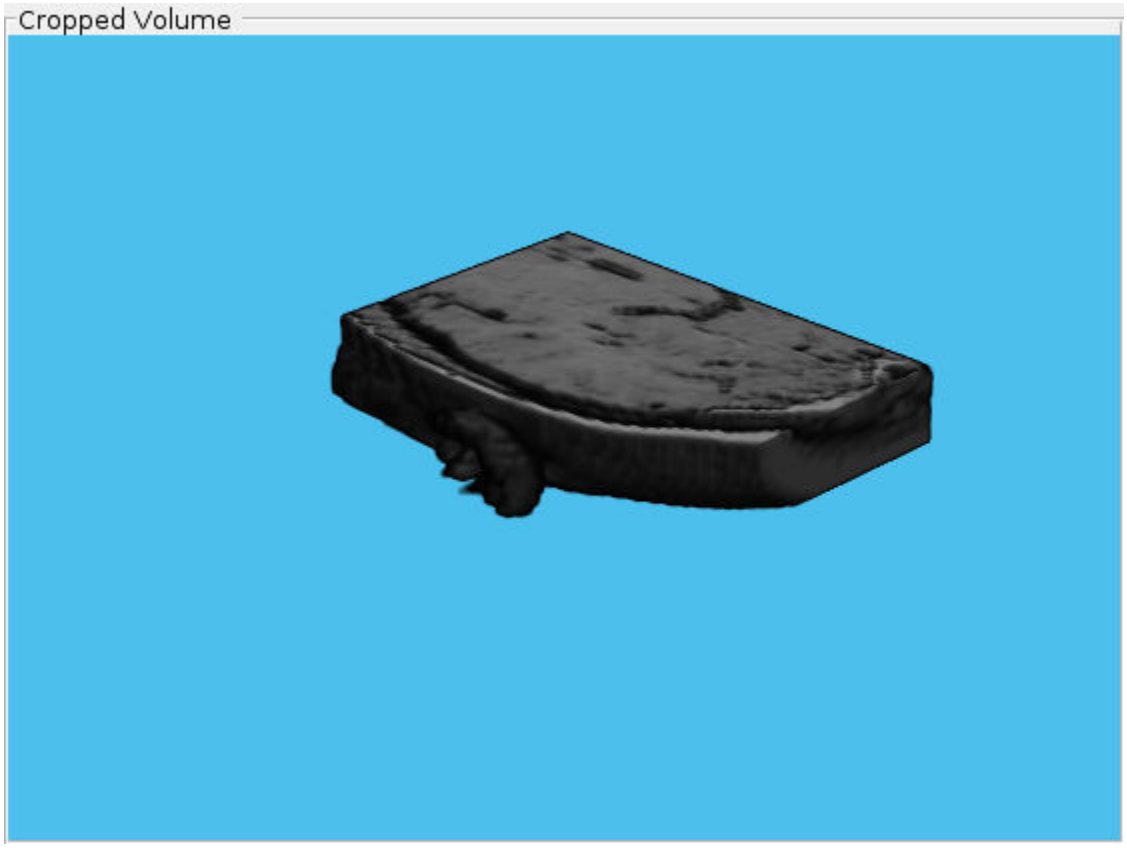
```
win = randomCropWindow3d(size(D),targetSize);
```

Crop the volume using the random crop window.

```
Dcrop = imcrop3(D,win);
```

Display the cropped volume in a display panel.

```
fullViewPnl = uipanel(figure,'Title','Cropped Volume');
volshow(Dcrop,'Parent',fullViewPnl);
```



Input Arguments

`inputSize` — Input image size

3-element vector of positive integers | 4-element vector of positive integers

Input image size, specified as one of the following.

Type of Input Image	Format of `inputSize`
3-D grayscale or binary image	3-element vector of positive integers of the form `[height width depth]`
3-D RGB or multispectral image	4-element vector of positive integers of the form `[height width depth channels]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`targetSize` — Target image size

3-element vector of positive integers | 4-element vector of positive integers

Target image size, specified as one of the following.

Type of Target Image	Format of `targetSize`
3-D grayscale or binary image	3-element vector of positive integers of the form `[height width depth]`
3-D RGB or multispectral image	4-element vector of positive integers of the form `[height width depth channels]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`win` — Cropping window

`Cuboid` object

Cropping window, returned as a `Cuboid` object.

See Also

`centerCropWindow3d` | `randomWindow2d` | `imcrop3`

Topics

“Augment Images for Deep Learning Workflows Using Image Processing Toolbox” (Deep Learning Toolbox)

Introduced in R2019b

randomPatchExtractionDatastore

Datastore for extracting random 2-D or 3-D random patches from images or pixel label images

Description

A `randomPatchExtractionDatastore` extracts corresponding randomly-positioned patches from two image-based datastores. For example, the input datastores can be two image datastores that contain the network inputs and desired network responses for training image-to-image regression networks, or ground truth images and pixel label data for training semantic segmentation networks.

This object requires that you have Deep Learning Toolbox.

Note When you use a `randomPatchExtractionDatastore` as a source of training data, the datastore extracts multiple random patches from each image for each epoch, so that each epoch uses a slightly different data set. The actual number of training patches at each epoch is the number of training images multiplied by `PatchesPerImage`. The image patches are not stored in memory.

Creation

Syntax

```
patchds = randomPatchExtractionDatastore(ds1,ds2,PatchSize)
patchds = randomPatchExtractionDatastore(ds1,ds2,PatchSize,Name,Value)
```

Description

`patchds = randomPatchExtractionDatastore(ds1,ds2,PatchSize)` creates a datastore that extracts randomly-positioned patches of size `PatchSize` from input data in datastore `ds1` and response data in datastore `ds2`.

`patchds = randomPatchExtractionDatastore(ds1,ds2,PatchSize,Name,Value)` uses name-value pairs to set the `PatchesPerImage`, `DataAugmentation`, and `DispatchInBackground` properties. You can specify multiple name-value pairs. Enclose each property name in quotes.

For example, `randomPatchExtractionDatastore(imds1,imds2,50,'PatchesPerImage',40)` creates a datastore that randomly generates 40 patches of size 50-by-50 pixels from each image in image datastores `imds1` and `imds2`.

Input Arguments

ds1 — Input data
`ImageDatastore` | `PixelLabelDatastore` | `TransformedDatastore`

Input data containing training input to the network, specified as an `ImageDatastore`, `PixelLabelDatastore`, or `TransformedDatastore`.

Specifying a `PixelLabelDatastore` requires Computer Vision Toolbox.

Note `ImageDatastore` allows batch-reading of JPG or PNG image files using prefetching. If you use a custom function for reading the images, then prefetching does not happen.

`ds2` — Response data
`ImageDatastore` | `PixelLabelDatastore` | `TransformedDatastore`

Response data representing the desired network responses, specified as an `ImageDatastore`, `PixelLabelDatastore`, or `TransformedDatastore`. If you specify a `TransformedDatastore`, then the underlying datastore must be an `ImageDatastore` or a `PixelLabelDatastore`.

Specifying a `PixelLabelDatastore` requires Computer Vision Toolbox.

Note `ImageDatastore` allows batch-reading of JPG or PNG image files using prefetching. If you use a custom function for reading the images, then prefetching does not happen.

Properties

`PatchSize` — Patch size
2-element vector of positive integers | 3-element vector of positive integers

This property is read-only.

Patch size, specified as one of the following.

- A 2-element vector of positive integers for 2-D patches. `PatchSize` has the form [*r c*] where *r* specifies the number of rows and *c* specifies the number of columns in the patch.
- A 3-element vector of positive integers for 3-D patches. `PatchSize` has the form [*r c p*] where *r* specifies the number of rows, *c* specifies the number of columns, and *p* specifies the number of planes in the patch.

`PatchesPerImage` — Number of random patches per image
`128` (default) | positive integer

Number of random patches per image, specified as a positive integer.

`DataAugmentation` — Preprocessing applied to input images
`'none'` (default) | `imageDataAugmenter` object

Preprocessing applied to input images, specified as an `imageDataAugmenter` object or `'none'`. When `DataAugmentation` is `'none'`, no preprocessing is applied to input images.

Augment data with random transformations, such as resizing, rotation, and reflection, to help prevent the network from overfitting and memorizing the exact details of the training data. The `randomPatchExtractionDatastore` applies the same random transformation to both patches in each pair. The datastore augments data in real-time while training.

The `DataAugmentation` property is not supported for 3-D data. To preprocess 3-D data, use the `transform` function.

`DispatchInBackground` — Dispatch observations in background
`false` (default) | `true`

Dispatch observations in the background during training, prediction, or classification, specified as `false` or `true`. To use background dispatching, you must have Parallel Computing Toolbox.

`MiniBatchSize` — Number of observations in each batch
128 | positive integer

Number of observations that are returned in each batch. You can change the value of `MiniBatchSize` only after you create the datastore. For training, prediction, and classification, the `MiniBatchSize` property is set to the mini-batch size defined in `trainingOptions`.

`NumObservations` — Total number of observations in the datastore
positive integer

This property is read-only.

Total number of observations in the `randomPatchExtractionDatastore`. The number of observations is the length of one training epoch.

Object Functions

combine	Combine data from multiple datastores
hasdata	Determine if data is available to read
numpartitions	Number of datastore partitions
partition	Partition a datastore
partitionByIndex	Partition randomPatchExtractionDatastore according to indices
preview	Preview subset of data in datastore
read	Read data from randomPatchExtractionDatastore
readall	Read all data in datastore
readByIndex	Read data specified by index from randomPatchExtractionDatastore
reset	Reset datastore to initial state
shuffle	Shuffle data in datastore
transform	Transform datastore
isPartitionable	Determine whether datastore is partitionable
isShuffleable	Determine whether datastore is shuffleable

Examples

Create Random Patch Extraction Datastore

Create an image datastore containing training images. The datastore in this example contains JPEG color images.

```
imageDir = fullfile(toolboxdir('images'),'imdata');
imds1 = imageDatastore(imageDir,'FileExtensions','.jpg');
```

Create a second datastore that transforms the images in `imds1` by applying a Gaussian blur.

```
imds2 = transform(imds1,@(x)imgaussfilt(x,2));
```

Create an `imageDataAugmenter` that rotates images by random angles in the range [0, 90] degrees and randomly reflects image data horizontally.

```
augmenter = imageDataAugmenter('RandRotation',[0 90],'RandXReflection',true)
```

```
augmenter =
  imageDataAugmenter with properties:

          FillValue: 0
     RandXReflection: 1
     RandYReflection: 0
        RandRotation: [0 90]
           RandScale: [1 1]
          RandXScale: [1 1]
          RandYScale: [1 1]
          RandXShear: [0 0]
          RandYShear: [0 0]
    RandXTranslation: [0 0]
    RandYTranslation: [0 0]
```

Create a `randomPatchExtractionDatastore` object that extracts random patches of size 100-by-100 from the unprocessed training images and corresponding smoothed response images. Specify the augmentation options by setting the `DataAugmentation` property.

```
patchds = randomPatchExtractionDatastore(imds1,imds2,[100 100], ...
    'DataAugmentation',augmenter)
```

```
patchds =
  randomPatchExtractionDatastore with properties:

         PatchesPerImage: 128
               PatchSize: [100 100]
        DataAugmentation: [1×1 imageDataAugmenter]
           MiniBatchSize: 128
         NumObservations: []
    DispatchInBackground: 0
```

Preview a set of augmented image patches and the corresponding smoothed image patches.

```
minibatch = preview(patchds);
inputs = minibatch.InputImage;
responses = minibatch.ResponseImage;
test = cat(2,inputs,responses);
montage(test','Size',[8 2])
title('Inputs (Left) and Responses (Right)')
```


Inputs (Left) and Responses (Right)

Train Semantic Segmentation Network Using Random Patch Extraction Datastore

Create an image datastore containing training images.

```
dataDir = fullfile(toolboxdir('vision'),'visiondata','triangleImages');
imageDir = fullfile(dataDir,'trainingImages');
imds = imageDatastore(imageDir);
```

Define class names and their associated label IDs. Then, create a pixel label datastore containing the ground truth pixel labels for the training images.

```
classNames = ["triangle","background"];
labelIDs = [255 0];
labelDir = fullfile(dataDir,'trainingLabels');
pxds = pixelLabelDatastore(labelDir,classNames,labelIDs);
```

Create a random patch extraction datastore to extract random patches of size 32-by-32 pixels from the images and corresponding pixel labels. Set the optional `PatchesPerImage` property to extract 512 random patches from each image and pixel label pair.

```
patchds = randomPatchExtractionDatastore(imds,pxds,32, ...
     'PatchesPerImage',512);
```

Create a network for semantic segmentation.

```
layers = [
    imageInputLayer([32 32 1])
    convolution2dLayer(3,64,'Padding',1)
    reluLayer()
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,64,'Padding',1)
    reluLayer()
    transposedConv2dLayer(4,64,'Stride',2,'Cropping',1)
    convolution2dLayer(1,2)
    softmaxLayer()
    pixelClassificationLayer()
    ]
```

```
layers =
  10x1 Layer array with layers:

     1   ''   Image Input                 32x32x1 images with 'zerocenter' normalization
     2   ''   Convolution                 64 3x3 convolutions with stride [1  1] and padding [1
     3   ''   ReLU                        ReLU
     4   ''   Max Pooling                 2x2 max pooling with stride [2  2] and padding [0  0
     5   ''   Convolution                 64 3x3 convolutions with stride [1  1] and padding [1
     6   ''   ReLU                        ReLU
     7   ''   Transposed Convolution      64 4x4 transposed convolutions with stride [2  2] and
     8   ''   Convolution                 2 1x1 convolutions with stride [1  1] and padding [0
     9   ''   Softmax                     softmax
    10   ''   Pixel Classification Layer  Cross-entropy loss
```

Set up training options. To reduce training time, set `MaxEpochs` to 5.

```
options = trainingOptions('sgdm', ...
    'InitialLearnRate',1e-3, ...
```

```
    'MaxEpochs',5, ...
    'Verbose',false);
```

Train the network.

```
net = trainNetwork(patchds,layers,options);
```

Tips

- The `randomPatchExtractionDatastore` expects that the output from the `read` operation on the input datastores return arrays of the same size.
- If the input datastore is an `ImageDatastore`, then the values in its `Labels` property are ignored by the `randomPatchExtractionDatastore`.
- To visualize 2-D data in a `randomPatchExtractionDatastore`, you can use the `preview` function, which returns a subset of data in a table. Visualize all of the patches in the same figure by using the `montage` function. For example, this code displays a preview of image patches from a `randomPatchExtractionDatastore` called `patchds`.

  ```
  minibatch = preview(patchds);
  montage(minibatch.InputImage)
  ```

See Also

`augmentedImageDatastore` | `pixelLabelDatastore` | `imageDatastore` | `pixelLabelImageDatastore` | `trainNetwork` | `imageDataAugmenter` | `TransformedDatastore`

Topics

“Increase Image Resolution Using Deep Learning”
“JPEG Image Deblocking Using Deep Learning”
“Image Processing Operator Approximation Using Deep Learning”
“Semantic Segmentation of Multispectral Images Using Deep Learning”
“Datastores for Deep Learning” (Deep Learning Toolbox)
“Preprocess Images for Deep Learning” (Deep Learning Toolbox)
“Deep Learning in MATLAB” (Deep Learning Toolbox)

Introduced in R2018b

partitionByIndex

Partition `randomPatchExtractionDatastore` according to indices

Syntax

```
patchds2 = partitionByIndex(patchds,ind)
```

Description

`patchds2 = partitionByIndex(patchds,ind)` partitions a subset of observations in a random patch extraction datastore, `patchds`, into a new datastore, `patchds2`. The desired observations are specified by indices, `ind`.

Input Arguments

`patchds` — Random patch extraction datastore

`randomPatchExtractionDatastore`

Random patch extraction datastore, specified as a `randomPatchExtractionDatastore` object.

`ind` — Indices

vector of positive integers

Indices of observations, specified as a vector of positive integers.

Output Arguments

`patchds2` — Output datastore

`randomPatchExtractionDatastore` object

Output datastore, returned as a `randomPatchExtractionDatastore` object containing a subset of files from `patchds`.

See Also

`randomPatchExtractionDatastore` | `read` | `readall` | `readByIndex`

Introduced in R2018b

read

Read data from `randomPatchExtractionDatastore`

Syntax

```
data = read(patchds)
[data,info] = read(patchds)
```

Description

`data = read(patchds)` returns a batch of data from a random patch extraction datastore, `patchds`. Subsequent calls to the `read` function continue reading from the endpoint of the previous call.

`[data,info] = read(patchds)` also returns information about the extracted data, including metadata, in `info`.

Input Arguments

`patchds` — Random patch extraction datastore

`randomPatchExtractionDatastore`

Random patch extraction datastore, specified as a `randomPatchExtractionDatastore` object. The datastore specifies a `MiniBatchSize` number of observations in each batch, and a `numObservations` total number of observations.

Output Arguments

`data` — Output data

table

Output data, returned as a table with `MiniBatchSize` number of rows.

- The first variable is `InputImage` and contains input image patches.
- If the network responses are images in an image datastore, then the second variable is `ResponseImage`.
- If the network responses are pixel label images in a pixel label datastore, then the second variable is `ResponsePixelLabelImage`.

Each column contains a cell array of patches of size determined by `PatchSize` and the type of image data.

- For 2-D single-channel images, the patches are size *m*-by*n*, where *m* specifies the number of rows and *n* specifies the number of columns in the patch.
- For 2-D multi-channel images, the patches are size *m*-by*n*-by-*c*, where *c* specifies the number of color channels of the patch. *c* is 3 for RGB images.
- For 3-D single-channel volumetric images, the patches are size *m*-by*n*-by-*p*, where *p* specifies the number of planes of the volume.

For the last batch of data in the datastore `patchds`, if `numObservations` is not cleanly divisible by `MiniBatchSize`, then `read` returns a partial batch containing all the remaining observations in the datastore.

`info` — Information about read data
structure array

Information about read data, returned as a structure array. The structure array can contain the following fields.

Field Name	Description
`RandomPatchRectangles`	`MiniBatchSize`-by-4 numeric matrix. Each row specifies the size and position of the patch in the format [*xywidthheight*]. The elements define the x- and y-coordinate of the top left corner, and the width and height of the patch.
`ImageIndices`	`MiniBatchSize`-by-1 numeric vector that specifies the indices of the read images in the input datastores.
`InputImageFilename`	`MiniBatchSize`-by-1 cell array that specifies the fully resolved path containing the path string, name of the file, and file extension of each input image.
`ResponseImageFilename`	`MiniBatchSize`-by-1 cell array that specifies the fully resolved path containing the path string, name of the file, and file extension of each response image or pixel label image.

See Also

`randomPatchExtractionDatastore` | `read (Datastore)` | `readByIndex` | `readall`

Introduced in R2018b

readByIndex

Read data specified by index from `randomPatchExtractionDatastore`

Syntax

```
data = readByIndex(patchds,ind)
[data,info] = readByIndex(patchds,ind)
```

Description

`data = readByIndex(patchds,ind)` returns a subset of observations from a random patch extraction datastore, `patchds`. The desired observations are specified by indices, `ind`.

`[data,info] = readByIndex(patchds,ind)` also returns information about the observations, including metadata, in `info`.

Input Arguments

`patchds` — Random patch extraction datastore

`randomPatchExtractionDatastore`

Random patch extraction datastore, specified as a `randomPatchExtractionDatastore` object.

`ind` — Indices

vector of positive integers

Indices of observations, specified as a vector of positive integers.

Output Arguments

`data` — Observations from datastore

table

Observations from the datastore, returned as a table with `length(ind)` number of rows.

`info` — Information about read data

structure array

Information about read data, returned as a structure array. The structure array can contain the following fields.

Field Name	Description
`RandomPatchRectangles`	`length(ind)`-by-4 numeric matrix. Each row specifies the size and position of the patch in the format [*xywidthheight*]. The elements define the x- and y-coordinate of the top left corner, and the width and height of the patch.

Field Name	Description
`ImageIndices`	`length(ind)`-by-1 numeric vector that specifies the indices of the read images in the input datastores.
`InputImageFilename`	`length(ind)`-by-1 cell array that specifies the fully resolved path containing the path string, name of the file, and file extension of each input image.
`ResponseImageFilename`	`length(ind)`-by-1 cell array that specifies the fully resolved path containing the path string, name of the file, and file extension of each response image or pixel label image.

See Also

`randomPatchExtractionDatastore` | `read` | `readall` | `partitionByIndex`

Introduced in R2018b

shuffle

Shuffle data in datastore

Syntax

```
dsrand = shuffle(ds)
```

Description

`dsrand = shuffle(ds)` returns a datastore that contains a random ordering of the data from datastore `ds`.

Input Arguments

ds — Datastore

`randomPatchExtractionDatastore` | `blockedImageDatastore` | `denoisingImageDatastore`

Datastore, specified as a `randomPatchExtractionDatastore`, `blockedImageDatastore`, or `denoisingImageDatastore`.

Output Arguments

dsrand — Output datastore

datastore

Output datastore, returned as a datastore of the same type as `ds` that contains randomly ordered data from `ds`.

See Also

`randomPatchExtractionDatastore` | `blockedImageDatastore` | `denoisingImageDatastore`

Introduced in R2018b

randomWindow2d

Randomly select rectangular region in image

Syntax

```
win = randomWindow2d(inputSize,targetSize)
win = randomWindow2d(inputSize,'Scale',scale,'DimensionRatio',dimensionRatio)
```

Description

`win = randomWindow2d(inputSize,targetSize)` selects a rectangular region of size `targetSize` from a random position in an image of size `inputSize`.

`win = randomWindow2d(inputSize,'Scale',scale,'DimensionRatio',dimensionRatio)` selects a rectangular region, specifying the size of the region relative to the input image, `scale`, and the aspect ratio of the region, `dimensionRatio`.

Examples

Select Random Rectangular Region of Target Size

Read and display an image.

```
I = imread("flamingos.jpg");
imshow(I)
```

Specify the size of the input image and the target size of the rectangular region.

```
inputSize = size(I);
targetSize = [40 60];
```

Select a region of the target size from a random location in the image.

```
rect = randomWindow2d(inputSize,targetSize);
```

Convert the region from a `Rectangle` object to a 4-element vector of the form [*xmin ymin width height*].

```
rectXYWH = [rect.XLimits(1) rect.YLimits(1) ...
    diff(rect.XLimits)+1 diff(rect.YLimits)+1];
```

Display the boundary of the rectangular region overlaid on the original image.

```
annotatedI = insertShape(I,"Rectangle",rectXYWH,"LineWidth",3);
imshow(annotatedI)
```

Select Rectangular Region Specifying Scale and Dimension Ratio

Read and display an image.

```
I = imread("strawberries.jpg");
imshow(I)
```

Specify the size of the input image.

```
inputSize = size(I);
```

Specify a fractional area of the region between 2% and 13% of the area of the input image. Specify a range of aspect ratios between 1:5 and 4:3.

```
scale = [0.02 0.13];
dimensionRatio = [1 5;4 3];
```

Specify a region with a randomly selected fractional area and aspect ratio from a random location in the image.

```
rect = randomWindow2d(inputSize,"Scale",scale,"DimensionRatio",dimensionRatio);
```

Crop the original image to the randomly selected region and display the result.

```
Icrop = imcrop(I,rect);
imshow(Icrop)
```

Input Arguments

`inputSize` — Input image size

2-element vector of positive integers | 3-element vector of positive integers

Input image size, specified as one of the following.

Type of Input Image	Format of `inputSize`
2-D grayscale or binary image	2-element vector of positive integers of the form `[height width]`
2-D RGB or multispectral image	3-element vector of positive integers of the form `[height width channels]`

`targetSize` — Target image size

2-element vector of positive integers | 3-element vector of positive integers

Target image size, specified as one of the following.

Type of Target Image	Format of `targetSize`
2-D grayscale or binary image	2-element vector of positive integers of the form `[height width]`
2-D RGB or multispectral image	3-element vector of positive integers of the form `[height width channels]`

`scale` — Region area as fraction of input image area

2-element numeric vector | function handle

Region area as a fraction of the input image area, specified as one of these values.

- 2-element nondecreasing numeric vector with values in the range [0, 1]. The elements define a minimum and maximum fractional area of the region, respectively. `randomWindow2d` selects a random value within the range to use as the fractional region area. To use a fixed region area, specify the same value for both elements.
- Function handle. The function must take no input arguments and return one number in the range [0, 1] specifying a valid fractional region area. For more information about function handles, see "Create Function Handle".

`dimensionRatio` — Range of aspect ratios of rectangular region
2-by-2 matrix of positive numbers | function handle

Range of aspect ratios of the rectangular region, specified as one of these values.

- 2-by-2 matrix of positive numbers. The first row defines the defines the minimum aspect ratio and the second row defines the maximum aspect ratio. `randomWindow2d` selects a random value within the range to use as the aspect ratio. To use a fixed aspect ratio, specify identical values for the first and second rows.
- Function handle. The function must take no input arguments and return one positive number specifying a valid dimension ratio. For example, a value of `1.2` specifies a 5:4 aspect ratio. For more information about function handles, see "Create Function Handle".

Example: `[1 8;1 4]` selects an aspect ratio in the range 1:8 to 1:4

Output Arguments

`win` — Rectangular window
`Rectangle` object

Rectangular window, returned as a `Rectangle` object.

See Also
`centerCropWindow2d` | `randomCropWindow3d` | `imcrop`

Topics
"Augment Images for Deep Learning Workflows Using Image Processing Toolbox" (Deep Learning Toolbox)

Introduced in R2021a

rangefilt

Local range of image

Syntax

```
J = rangefilt(I)
J = rangefilt(I,nhood)
```

Description

`J = rangefilt(I)` returns the array `J`, where each output pixel contains the range value (maximum value − minimum value) of the 3-by-3 neighborhood around the corresponding pixel in the input image `I`.

`J = rangefilt(I,nhood)` returns the local range of image `I` using the specified neighborhood, `nhood`.

Examples

Identify Objects in 2-D Image

Read an image into the workspace.

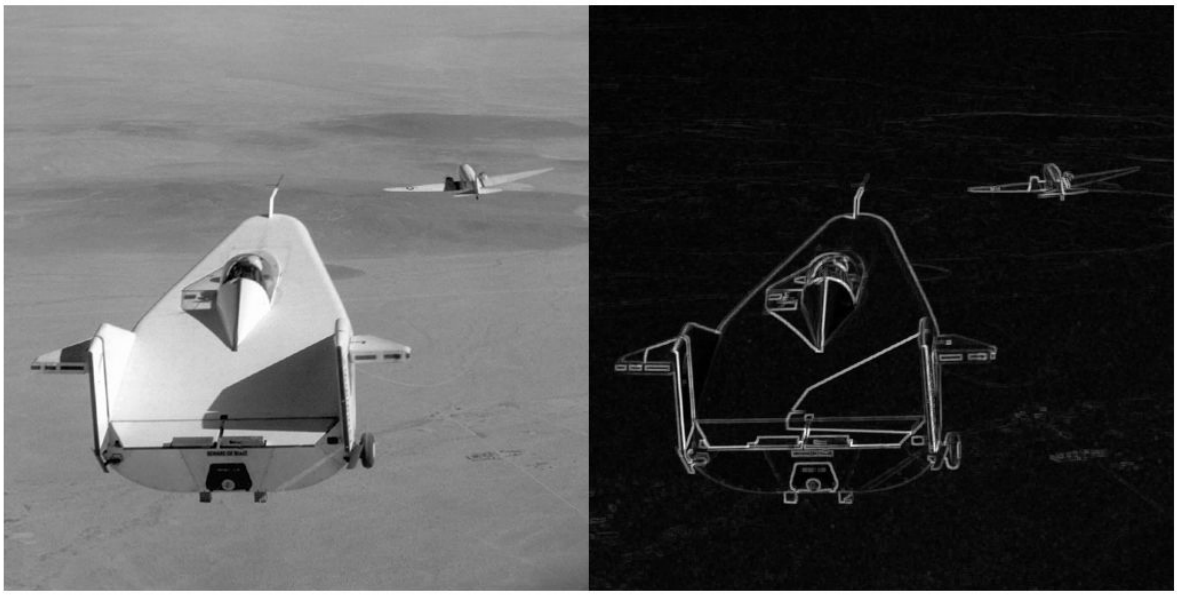
```
I = imread('liftingbody.png');
```

Filter the image. The `rangefilt` function returns an array where each output pixel contains the range value (maximum value - minimum value) of the 3-by-3 neighborhood around the corresponding pixel in the input image.

```
J = rangefilt(I);
```

Display the original image and the filtered image side-by-side.

```
imshowpair(I,J,'montage')
```



Detect Regions of Texture in Images

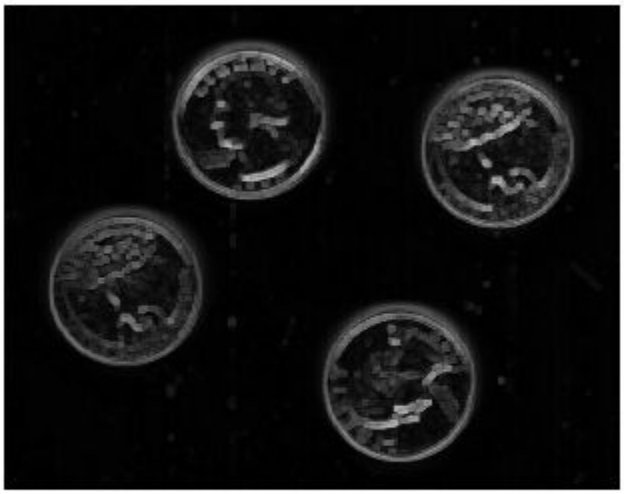
This example shows how to detect regions of texture in an image using the texture filter functions

Read an image into the workspace and display it. In the figure, the background is smooth—there is very little variation in the gray-level values. In the foreground, the surface contours of the coins exhibit more texture. In this image, foreground pixels have more variability and thus higher range values.

```
I = imread('eight.tif');
imshow(I)
```

Filter the image with the `rangefilt` function and display the results. Range filtering makes the edges and contours of the coins visible.

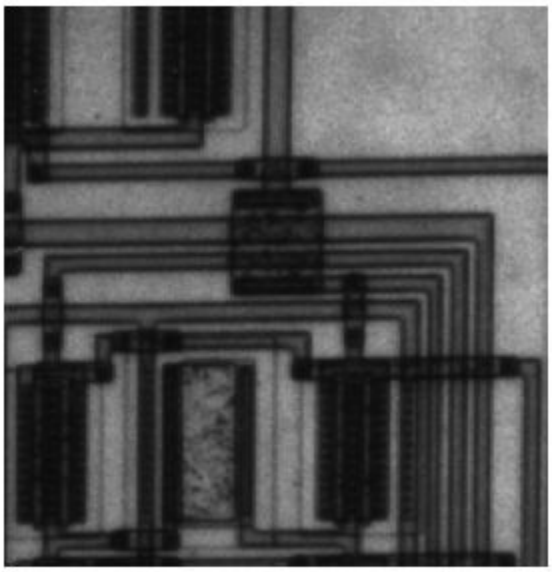
```
K = rangefilt(I);
figure
imshow(K)
```



Identify Vertical Edges Using Range Filtering

Read an image into the workspace, and display it.

```
I = imread('circuit.tif');
imshow(I);
```



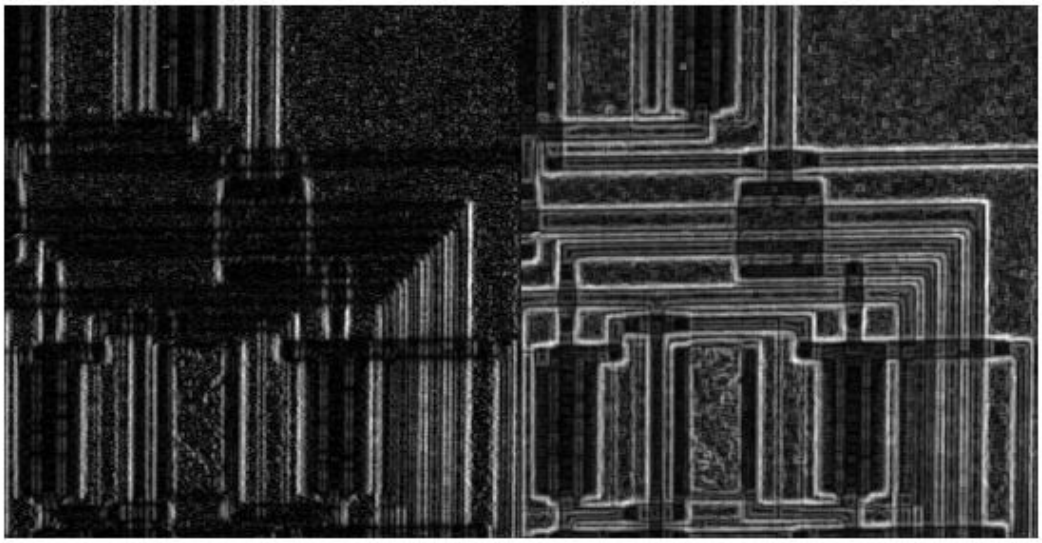
Define a neighborhood. In this example, the neighborhood returns a large value when there is a large difference between pixel values to the left and right of an input pixel. The filtering does not consider pixels above and blow the input pixel. Thus, this neighborhood emphasizes vertical edges.

```
nhood = [1 1 1];
```

Perform the range filtering operation using this neighborhood. For comparison, also perform range filtering using the default 3-by-3 neighborhood. Compare the results.

```
J = rangefilt(I,nhood);
K = rangefilt(I);
figure
imshowpair(J,K,'montage');
title('Range filtering using specified neighborhood (left) and default neighborhood (right)');
```

Range filtering using specified neighborhood (left) and default neighborhood (right)



The result using the specified neighborhood emphasizes vertical edges, as expected. In comparison, the default filter is not sensitive to edge directionality.

Input Arguments

`I` — Image to be filtered

numeric array

Image to be filtered, specified as a numeric array of any dimension.

Data Types: `double` | `uint8` | `uint16` | `uint32` | `logical`

`nhood` — Neighborhood

`true(3)` (default) | logical or numeric array containing zeros and ones

Neighborhood, specified as a logical or numeric array containing zeros and ones. The size of `nhood` must be odd in each dimension. `rangefilt` determines the center element of the neighborhood by `floor((size(NHOOD) + 1)/2)`.

To specify neighborhoods of other shapes, such as a disk, use the `strel` function to create a structuring element object of the desired shape. Then, extract the neighborhood from the structuring element object's `neighborhood` property.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

J — Filtered image

numeric array

Filtered image, returned as a numeric array, the same size and class as the input image `I`, except for signed integer data types. The output class for signed data types is the corresponding unsigned integer data type. For example, if the class of `I` is `int8`, then the class of `J` is `uint8`.

Algorithms

`rangefilt` uses the morphological functions `imdilate` and `imerode` to determine the maximum and minimum values in the specified neighborhood. Consequently, `rangefilt` uses the padding behavior of these morphological functions.

See Also

Functions

`stdfilt` | `entropyfilt` | `getnhood`

Objects

`strel` | `offsetstrel`

Topics

“Calculate Statistical Measures of Texture”
“What Is Image Filtering in the Spatial Domain?”

Introduced before R2006a

rawinfo

Read metadata from RAW file

Syntax

```
info = rawinfo(filename)
```

Description

`info = rawinfo(filename)` reads metadata from a RAW file specified by `filename`.

Examples

Determine Sensor Layout in RAW File

Retrieve metadata from a RAW image file.

```
info = rawinfo("colorCheckerTestImage.NEF");
```

Examine the `CFALayout` field to determine the sensor layout.

```
sensor_layout = info.CFALayout
```

```
sensor_layout =
"RGGB"
```

Input Arguments

`filename` — Name of RAW file

character vector | string scalar

Name of RAW file, specified as a string scalar or character vector. Specify `filename` as a full path, containing the file name and extension, or as a relative path from the current folder or from any folder on the MATLAB path.

Data Types: `char` | `string`

Output Arguments

`info` — Metadata from RAW file

`struct`

Metadata from the RAW file, returned as a `struct` with these fields.

Field	Description
`Filename`	String scalar specifying the full name, including the path, to the RAW image file.

Field	Description
`CFASensorType`	String scalar specifying the type of the sensor that captured the image. `rawinfo` supports these sensors: `"Bayer"`, `"Fuji X-Trans"`, `"Foveon"`, `"Fuji Super-CCD"`, and `"Non-Bayer"`.
`CFALayout`	String scalar specifying the sensor layout for Bayer sensors. Sensor layouts supported by `rawinfo` include but are not limited to `"BGGR"`, `"RGGB"`, `"GBRG"`, and `"GRGB"`. For non-Bayer sensors, this value is empty.
`CFAImageSize`	Two-element row vector of type `double` specifying the total number of rows and columns present in the CFA image.
`SamplesPerPixel`	Scalar of type `double` specifying the number of samples in every pixel of the image.
`ImageSizeInfo`	Structure containing all the size information that describes a CFA image.
`ColorInfo`	Structure containing all of the color information required to render an RGB image from the CFA image.
`ExifTags`	Structure containing the EXIF Tags, if any, present in the file. The `Makernotes`, if any, are included in this field.
`LensInfo`	Structure containing information about the lens used to capture the image.
`MiscInfo`	Structure containing information, if any, about the camera and image creator present in the file.
`XMPData`	String scalar containing the Adobe Extensible Metadata Platform (XMP) data, if any, present in the file.
`FormatSpecificInfo`	Structure containing format specific information, such as DNG Tags and X-Trans sensor layout descriptions. If no format specific information is available, this value is an empty structure.
`LibrawVersion`	String scalar specifying the version of the LibRaw library currently being used. LibRaw is a library for reading RAW files obtained from digital photo cameras. For more information, see libraw.org.

Limitations

- The `rawinfo` function does not support RAW file formats that employ JPEG compression.

More About

RAW File Format

The RAW file format preserves image data in its most unedited state, recorded directly from the camera sensor. Most camera manufacturers define their own proprietary RAW file format, such as the Nikon NEF file format and the Canon CRW format. Adobe has also defined a RAW file format, DNG (digital negative), which is supported by several cameras. The name of the format is typically capitalized, like other file formats such as JPG and TIF. However, unlike other file formats, RAW is not an acronym.

Tips

- The function uses LibRaw version 0.20.2 for reading the CFA image data.

See Also

`raw2planar` | `rawread` | `planar2raw` | `raw2rgb`

Topics

“Implement Digital Camera Processing Pipeline”

Introduced in R2021a

rawread

Read color filter array (CFA) image from RAW file

Syntax

```
cfaimage = rawread(filename)
cfaimage = rawread(filename,'VisibleImageOnly',visibleImageOnly)
```

Description

`cfaimage = rawread(filename)` reads a CFA image from the RAW image file specified by `filename`.

`cfaimage = rawread(filename,'VisibleImageOnly',visibleImageOnly)` specifies whether to read only the visible portion of the CFA or to read the entire CFA including the frame.

Examples

Read CFA Image Data from File

Read only the visible Color Filter Array (CFA) image data from a file. By default, `rawread` returns only the visible portion of the CFA image.

```
cfa = rawread("colorCheckerTestImage.NEF");
```

Read the entire CFA image from a file, including the image frame. In a RAW image, the frame is typically used to calculate the black-level surrounding the visible image.

```
cfa = rawread("colorCheckerTestImage.NEF", "VisibleImageOnly", false);
```

Input Arguments

`filename` — Name of RAW file

character vector | `string` scalar

Name of RAW file, specified as a `string` scalar or `char` vector. Specify `filename` as a full path, containing the file name and extension, or as a relative path from the current folder or from any folder on the MATLAB path.

Data Types: `char` | `string`

`visibleImageOnly` — Return only visible CFA image data

`true` (default) | `false`

Return only visible CFA image data, specified as a logical scalar `true` or `false`. In a RAW image, the frame is typically used to calculate the black-level surrounding the visible image. To read only the visible portion of the CFA, specify `true`. To read the entire CFA, including the frame, specify `false`. This table provides more detail for each option.

Value	Description
`true`	The dimensions of the `cfaimage` are `VisibleImageSize(1)`-by-`VisibleImageSize(2)`-by-*P*, where *P* is the number of planes. `VisibleImageSize` is a field in the `ImageSizeInfo` structure returned by `rawinfo`.
`false`	The dimensions of the `cfaimage` are `CFAImageSize(1)`-by-`CFAImageSize(2)`-by-*P*, where *P* is the number of planes. `CFAImageSize` is a field in the structure returned by `rawinfo`.

Data Types: `logical`

Output Arguments

`cfaimage` — CFA image

m-by-*n*-by-*p* numeric array

CFA image, returned as an *m*-by-*n*-by-*p* numeric array.

By default, `rawread` returns only the visible portion of the CFA image. In this case, the values of *m* and *n* correspond to the first and second elements of the `VisibleImageSize` field reported by `rawinfo`, respectively. If you choose to include the frame in the returned image, the values of *m* and *n* correspond to the first and second elements of the `CFAImageSize` field reported by `rawinfo`, respectively. For both types of returned images, the value of *p* depends on the type of CFA sensor. For Bayer type sensors, the value is 1. For a Foveon sensor, the value is 3.

Data Types: `uint16` | `single`

Limitations

- The `rawread` function does not support RAW file formats that employ JPEG compression.

More About

RAW File Format

The RAW file format preserves image data in its most unedited state, recorded directly from the camera sensor. Most camera manufacturers define their own proprietary RAW file format, such as the Nikon NEF file format and the Canon CRW format. Adobe has also defined a RAW file format, DNG (digital negative), which is supported by several cameras. The name of the format is typically capitalized, like other file formats such as JPG and TIF. However, unlike other file formats, RAW is not an acronym.

Tips

- The function uses LibRaw version 0.20.2 for reading the CFA image data.

See Also

`raw2planar` | `rawinfo` | `planar2raw` | `raw2rgb`

Topics

“Implement Digital Camera Processing Pipeline”

Introduced in R2021a

raw2planar

Separate Bayer pattern Color Filter Array (CFA) image into sensor element images

Syntax

```
I = raw2planar(cfa)
```

Description

`I = raw2planar(cfa)` separates the channels of the Bayer pattern CFA image `cfa` into a multidimensional image, `I`, with a channel for each individual sensor element.

Examples

Split CFA Image into Individual Sensor Component Images

Read a Color Filter Array (CFA) image into the workspace. The `rawread` function returns `cfa`, a 4012-by-6034 image.

```
cfa = rawread("colorCheckerTestImage.NEF");
```

Split the returned CFA image into several individual images, each representing a CFA sensor component. The CFA image has a Bayer pattern of RGGB. The `raw2planar` function returns 2206-by-3017-by-4 array representing each component of the RGGB pattern.

```
rggb = raw2planar(cfa);
```

Input Arguments

`cfa` — Bayer pattern CFA image

M-by-*N* numeric matrix

Bayer pattern CFA image, specified as an *M*-by-*N* numeric matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`I` — Image with channel for each sensor element

(*M*/2)-by-(*N*/2)-by-4 numeric array

Image with a channel for each sensor element, returned as an (*M*/2)-by-(*N*/2)-by-4 numeric array of the same class as `cfa`.

The order of the channels in the output image depends on the Bayer pattern of the CFA: the order of the red, green, and blue sensors. The 2-by-2 grid of pixels in the upper-left corner of the CFA image describes the channel order, from left-to-right, top-to-bottom. `I(:,:,1)` corresponds to the sensor at

`cfa(1,1)`, `I(:,:,2)` to the sensor at `cfa(1,2)`, `I(:,:,3)` to the sensor at `cfa(2,1)`, and `I(:,:,4)` to the sensor at `cfa(2,2)`.

See Also

`rawread` | `rawinfo` | `planar2raw` | `raw2rgb`

Topics

"Implement Digital Camera Processing Pipeline"

Introduced in R2021a

raw2rgb

Transform Color Filter Array (CFA) image in RAW file into RGB image

Syntax

```
rgbimage = raw2rgb(filename)
rgbimage = raw2rgb(filename,Name,Value)
```

Description

`rgbimage = raw2rgb(filename)` transforms the CFA image in the RAW file specified by `filename` into an RGB image.

`rgbimage = raw2rgb(filename,Name,Value)` specifies additional options with name-value arguments.

Examples

Convert CFA Image to RGB

Convert the Color Filter Array (CFA) image in the file into a 16-bit RGB image in the sRGB colorspace.

```
rgb = raw2rgb("colorCheckerTestImage.NEF");
imshow(rgb)
```

Convert the CFA image into a 8-bit RGB image in the Adobe RGB 1998 colorspace.

```
rgb = raw2rgb("colorCheckerTestImage.NEF", "BitsPerSample", 8, "ColorSpace", "adobe-rgb-1998");
imshow(rgb)
```

Convert the CFA image into a 16-bit image, in the camera's native colorspace, white-balanced against a D65 illuminant.

```
rgb = raw2rgb("colorCheckerTestImage.NEF", "ColorSpace", "camera", "WhiteBalanceMultipliers", "D
imshow(rgb)
```

Input Arguments

`filename` — Name of RAW file

character vector | `string` scalar

Name of RAW file, specified as a `string` scalar or character vector. Specify `filename` as a full path, containing the file name and extension, or as a relative path from the current folder or from any folder on the MATLAB path.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `rgb = raw2rgb("colorCheckerTestImage.NEF","BitsPerSample",8,"ColorSpace","adobe-rgb-1998");`

`ApplyContrastStretch` — Apply contrast stretching when rendering the RGB image
`false` or `0` (default) | `true` or `1`

Apply contrast stretching when rendering the RGB image, specified as a logical 0 (`false`) or 1 (`true`).

Data Types: `logical`

`BitsPerSample` — Bits per sample of output RGB image
16 (default) | 8

Bits per sample of the output RGB image, specified as the numeric scalar 8 or 16.

`ColorSpace` — Color space of output RGB values
`'srgb'` (default) | `'camera'` | `'adobe-rgb-1998'`

Color space of the output RGB values, specified as `'srgb'`,`'camera'`, or `'adobe-rgb-1998'`. For more information about the `'srgb'` and `'adobe-rgb-1998'` color spaces, see “Understanding Color Spaces and Color Space Conversion”. The `'camera'` color space is native to the device.

Data Types: `char` | `string`

`WhiteBalanceMultipliers` — White balance adjustment for rendering
`'AsTaken'` (default) | `'D65'` | `'ComputeFromImage'` | 1-by-*N* numeric vector

White balance adjustment for rendering the RGB image, specified as one of the strings in this list, or as a 1-by-*N* vector of class `double`.

Value	Description
`'AsTaken'`	White balance multipliers used by the camera to capture the image
`'D65'`	White balance multipliers required to balance image using the D65 illuminant
`'ComputeFromImage'`	White balance multipliers determined by analyzing the CFA image
1-by-*N* vector	Custom white balance multipliers specified as a 1-by-*N* vector of class `single` or `double`. For Bayer sensor images, *N* must be 4 and the order of the coefficients should match the `CFALayout` field reported by `rawinfo`. For non-Bayer sensors, *N* should match the `SamplesPerPixel` field reported by `rawinfo`.

Data Types: `double` | `char` | `string`

Output Arguments

`rgbimage` — RGB image
numeric array

RGB image, returned as an *M*-by-*N*-by-3 numeric array. The values of *M* and *N* correspond to the first and second elements of the `RenderedImageSize` field reported by `rawinfo`, respectively.

`rgbimage` can be either `uint8` or `uint16`, depending on the value of the `BitsPerSample` name-value pair.

Limitations

- The `raw2rgb` function does not support RAW file formats that employ JPEG compression.

Tips

- The function uses LibRaw version 0.20.2 for reading the CFA image data.

See Also

`raw2planar` | `rawread` | `rawinfo` | `planar2raw`

Topics

“Implement Digital Camera Processing Pipeline”

Introduced in R2021a

reducepoly

Reduce density of points in ROI using Ramer–Douglas–Peucker algorithm

Syntax

```
P_reduced = reducepoly(P)
P_reduced = reducepoly(P,tolerance)
```

Description

`P_reduced = reducepoly(P)` reduces the density of points in array `P`. The `reducepoly` function uses the Ramer-Douglas-Peucker line simplification algorithm, removing points along straight lines and leaving only knickpoints (points where the line curves).

`P_reduced = reducepoly(P,tolerance)` reduces the density of points in array `P`, where `tolerance` specifies how much a point can deviate from a straight line.

Examples

Compare Polygon with Full and Reduced Vertices

Read an image into the workspace.

```
I = imread('coins.png');
imshow(I)
```

Convert the image from grayscale to binary.

```
bw = imbinarize(I);
```

Obtain the boundaries of all the coins in the binary image.

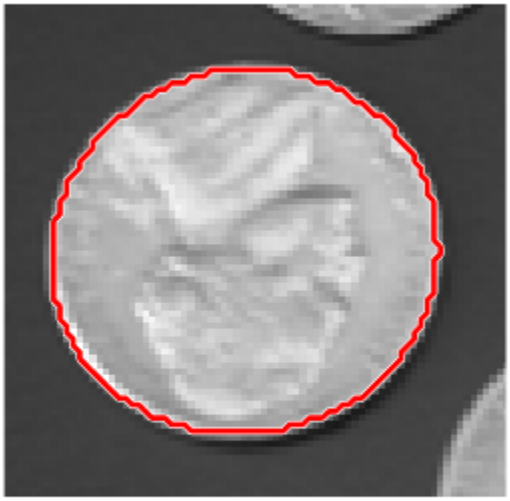
```
[B,L] = bwboundaries(bw,'noholes');
```

Select the boundary of the first detected coin.

```
coinNumber = 1;
boundary = B{coinNumber};
```

Plot the boundary for the first detected coin over the original image.

```
hold on
visboundaries({boundary})
xlim([min(boundary(:,2))-10 max(boundary(:,2))+10])
ylim([min(boundary(:,1))-10 max(boundary(:,1))+10])
hold off
```

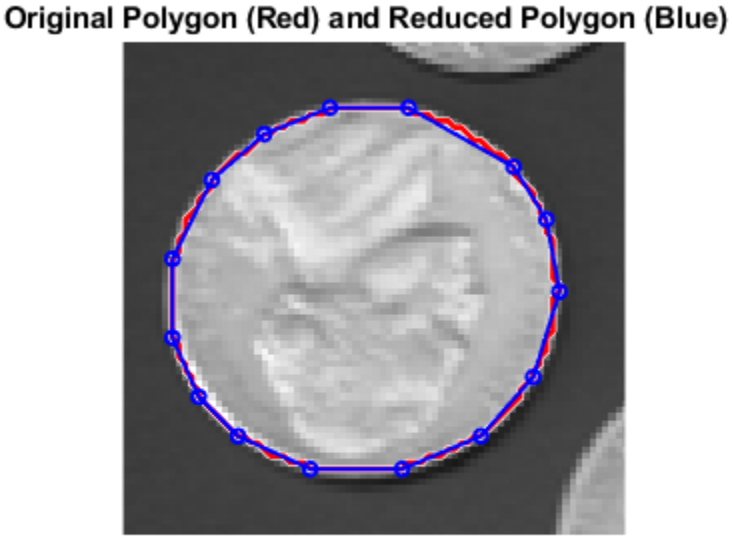


Use `reducepoly` to reduce the number of points defining the coin boundary. Return a smaller number of points by increasing the tolerance from the default value of `0.001`.

```
tolerance = 0.02;
p_reduced = reducepoly(boundary,tolerance);
```

To see how well the reduced polygon matches the original polygon, plot the reduced polygon vertices over the image.

```
line(p_reduced(:,2),p_reduced(:,1), ...
       'color','b','linestyle','-','linewidth',1.5,...
       'marker','o','markersize',5);
title('Original Polygon (Red) and Reduced Polygon (Blue)');
```



Input Arguments

P — Points to be reduced

n-by-2 numeric matrix

Points to be reduced, specified as an *n*-by-2 numeric matrix of the form `[x1 y1; ...; xn yn]`. Each row in the array defines a vertex in an ROI shape, such as a polyline, polygon, or freehand.

For example, you can draw a freehand ROI by using the `drawfreehand` function. Then, get the ROI vertices from the `Position` property of the freehand ROI object.

```
roi = drawfreehand;
P = roi.Position;
```

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

tolerance — Sensitivity of reduction algorithm

`0.001` (default) | numeric scalar

Sensitivity of the reduction algorithm, specified as a numeric scalar in the range [0, 1]. Increasing the tolerance increases the number of points removed. A tolerance value of `0` has a minimum reduction in points. A tolerance value of `1` results in maximum reduction in points, leaving only the end points of the line.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

P_reduced — Reduced data set

m-by-2 numeric matrix

Reduced data set, returned as an *m*-by-2 numeric matrix. The number of reduced points is usually smaller than the number of original points in `P`.

Data Types: `double`

Algorithms

The Ramer-Douglas-Peucker line simplification algorithm recursively subdivides a shape looking to replace a run of points with a straight line. The algorithm checks that no point in the run deviates from the straight line by more than the value specified by `tolerance`.

See Also

`drawfreehand` | `drawpolygon` | `drawpolyline` | `drawassisted` | `bwboundaries`

Introduced in R2019b

reflect

Reflect structuring element

Syntax

```
SE2 = reflect(SE)
```

Description

`SE2 = reflect(SE)` reflects the structuring element (or structuring elements) specified by `SE`. This method reflects the structuring element through its center. The effect is the same as if you rotated the structuring element's domain 180 degrees around its center (for a 2-D structuring element).

Examples

Reflect a Structuring Element

Create a structuring element.

```
se = strel([0 0 1; 0 0 0; 0 0 0])
```

```
se =
strel is a arbitrary shaped structuring element with properties:

      Neighborhood: [3x3 logical]
    Dimensionality: 2
```

Look at the neighborhood.

```
se.Neighborhood
```

```
ans = 3x3 logical array

   0   0   1
   0   0   0
   0   0   0
```

Reflect it.

```
se2 = reflect(se)
```

```
se2 =
strel is a arbitrary shaped structuring element with properties:

      Neighborhood: [3x3 logical]
    Dimensionality: 2
```

Look at the reflected neighborhood.

```
se2.Neighborhood

ans = 3x3 logical array

   0   0   0
   0   0   0
   1   0   0
```

Reflect Offset Structuring Element

Create an `offsetstrel` structuring element.

```
se = offsetstrel("ball",5,6.5)

se =
offsetstrel is a ball shaped offset structuring element with properties:

            Offset: [11x11 double]
    Dimensionality: 2
```

Reflect the structuring element.

```
se2 = reflect(se)

se2 =
offsetstrel is a ball shaped offset structuring element with properties:

            Offset: [11x11 double]
    Dimensionality: 2
```

Input Arguments

SE — Structuring element

`strel` or `offsetstrel` object or array of objects

Structuring element, specified as a `strel` or `offsetstrel` object or array of objects. If SE is an array of structuring element objects, then `reflect` reflects each element of SE.

Output Arguments

SE2 — Reflected structuring element

`strel` or `offsetstrel` object or array of objects

Reflected structuring element, returned as a `strel` or `offsetstrel` object or array of objects. SE2 has the same size as SE.

See Also

`translate`

Topics
“Structuring Elements”

Introduced before R2006a

regionfill

Fill in specified regions in image using inward interpolation

Syntax

```
J = regionfill(I,mask)
J = regionfill(I,x,y)
```

Description

`J = regionfill(I,mask)` fills the regions in image `I` specified by `mask`. Nonzero pixels in `mask` designate the pixels of image `I` to fill. You can use `regionfill` to remove objects in an image or to replace invalid pixel values using their neighbors.

`J = regionfill(I,x,y)` fills the region in image `I` corresponding to the polygon with vertices specified by `x` and `y`.

Examples

Fill Region in Grayscale Image

Read and display a grayscale image.

```
I = imread('eight.tif');
imshow(I)
```

Specify the x- and y-coordinates of a polygon that completely surrounds one of the coins in the image.

```
x = [222 272 300 270 221 194];
y = [21 21 75 121 121 75];
```

Fill the polygon by using the `regionfill` function.

```
J = regionfill(I,x,y);
```

Display the filled image.

```
imshow(J)
title('Filled Image with One Fewer Coin')
```

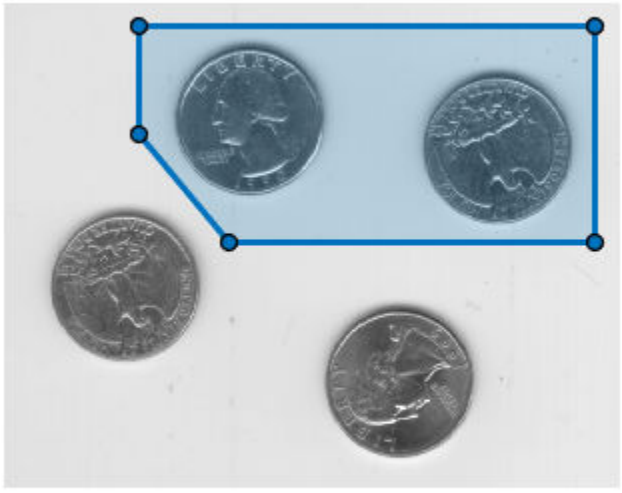
Fill Regions Using Mask Image

Read and display a grayscale image.

```
I = imread('eight.tif');
imshow(I)
```

Specify the vertices of a polygon ROI that completely surrounds two of the coins by using the `drawpolygon` function. Specify the 'Position' name-value pair argument as the x-coordinates and y-coordinates of the polygon vertices. If you want to draw the polygon interactively, then omit the 'Position' name-value pair argument.

```
x = [68 296 296 113 68];
y = [12 12 120 120 66];
roi = drawpolygon(gca,'Position',[x;y]');
```



Create a mask image in which the ROI is `true` and the background is `false`. Display the mask.

```
mask = createMask(roi);
imshow(mask)
```

Fill the regions in the input image using the mask image. Display the filled image.

```
J = regionfill(I,mask);
imshow(J)
```

Input Arguments

`I` — Grayscale image

2-D numeric matrix

Grayscale image, specified as a 2-D numeric matrix of size greater than or equal to 3-by-3.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`mask` — Mask image

logical matrix | numeric matrix

Mask image, specified as a logical matrix or numeric matrix of the same size as `I`. For numeric input, any nonzero pixels are considered to be `1` (`true`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`x` — *x*-coordinates of polygon vertices

numeric vector

x-coordinates of polygon vertices, specified as a numeric vector. `x` must be the same length as `y`.

Example: `[222 272 300 270 221 194];`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`y` — *y*-coordinates of polygon vertices

numeric vector

y-coordinates of polygon vertices, specified as a numeric vector. `y` must be the same length as `x`.

Example: `[21 21 75 121 121 75];`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output Arguments

`J` — Filled grayscale image

2-D numeric array

Filled grayscale image, returned as a 2-D numeric array. `J` has the same size and class as `I`.

Tips

- To interactively create the ROI mask `mask`, you can use the `roipoly` function or the `drawpolygon` function followed by `createMask`.
- For more information about how `regionfill` determines which pixels are on the ROI boundaries when you specify the ROI using (`x`, `y`) polygon coordinates, see “Classify Pixels That Are Partially Enclosed by ROI”.

Algorithms

`regionfill` smoothly interpolates inward from the pixel values on the outer boundary of the regions. `regionfill` calculates the discrete Laplacian over the regions and solves the Dirichlet boundary value problem.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`regionfill` supports the generation of C code (requires MATLAB Coder). For more information, see "Code Generation for Image Processing".

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- GPU Coder generates optimized CUDA code for only the `regionfill(I,mask)` syntax. The `regionfill(I,x,y)` syntax generates code that is not optimized for GPUs.

See Also

`imfill` | `drawpolygon` | `Polygon` | `poly2mask` | `roifilt2` | `roipoly` | `inpaintCoherent` | `inpaintExemplar`

Introduced in R2015a

regionprops

Measure properties of image regions

Syntax

```
stats = regionprops(BW,properties)
stats = regionprops(CC,properties)
stats = regionprops(L,properties)
stats = regionprops( ___ ,I,properties)
stats = regionprops(output, ___ )
```

Description

`stats = regionprops(BW,properties)` returns measurements for the set of properties for each 8-connected component (object) in the binary image, `BW`. You can use `regionprops` on contiguous regions and discontiguous regions (see “More About” on page 1-2698).

Note To return measurements of a 3-D volumetric image, consider using `regionprops3`. While `regionprops` can accept 3-D images, `regionprops3` calculates more statistics for 3-D images than `regionprops`.

For all syntaxes, you can omit the `properties` argument. In this case, `regionprops` returns the `"Area"`, `"Centroid"`, and `"BoundingBox"` measurements.

`stats = regionprops(CC,properties)` measures a set of properties for each connected component (object) in `CC`, which is a structure returned by `bwconncomp`.

`stats = regionprops(L,properties)` measures a set of properties for each labeled region in label image `L`.

`stats = regionprops( ___ ,I,properties)` returns measurements for the set of properties specified by `properties` for each labeled region in the image `I`. The first input to `regionprops` (`BW`, `CC`, or `L`) identifies the regions in `I`.

`stats = regionprops(output, ___ )` returns measurements for a set of properties, where `output` specifies the format of the returned measurements as an array of structures or a table.

Examples

Calculate Centroids and Superimpose Locations on Image

Read a binary image into workspace.

```
BW = imread('text.png');
```

Calculate centroids for connected components in the image using `regionprops`. The `regionprops` function returns the centroids in a structure array.

```
s = regionprops(BW,'centroid');
```

Store the *x*- and *y*-coordinates of the centroids into a two-column matrix.

```
centroids = cat(1,s.Centroid);
```

Display the binary image with the centroid locations superimposed.

```
imshow(BW)
hold on
plot(centroids(:,1),centroids(:,2),'b*')
hold off
```

Estimate Center and Radii of Circular Objects and Plot Circles

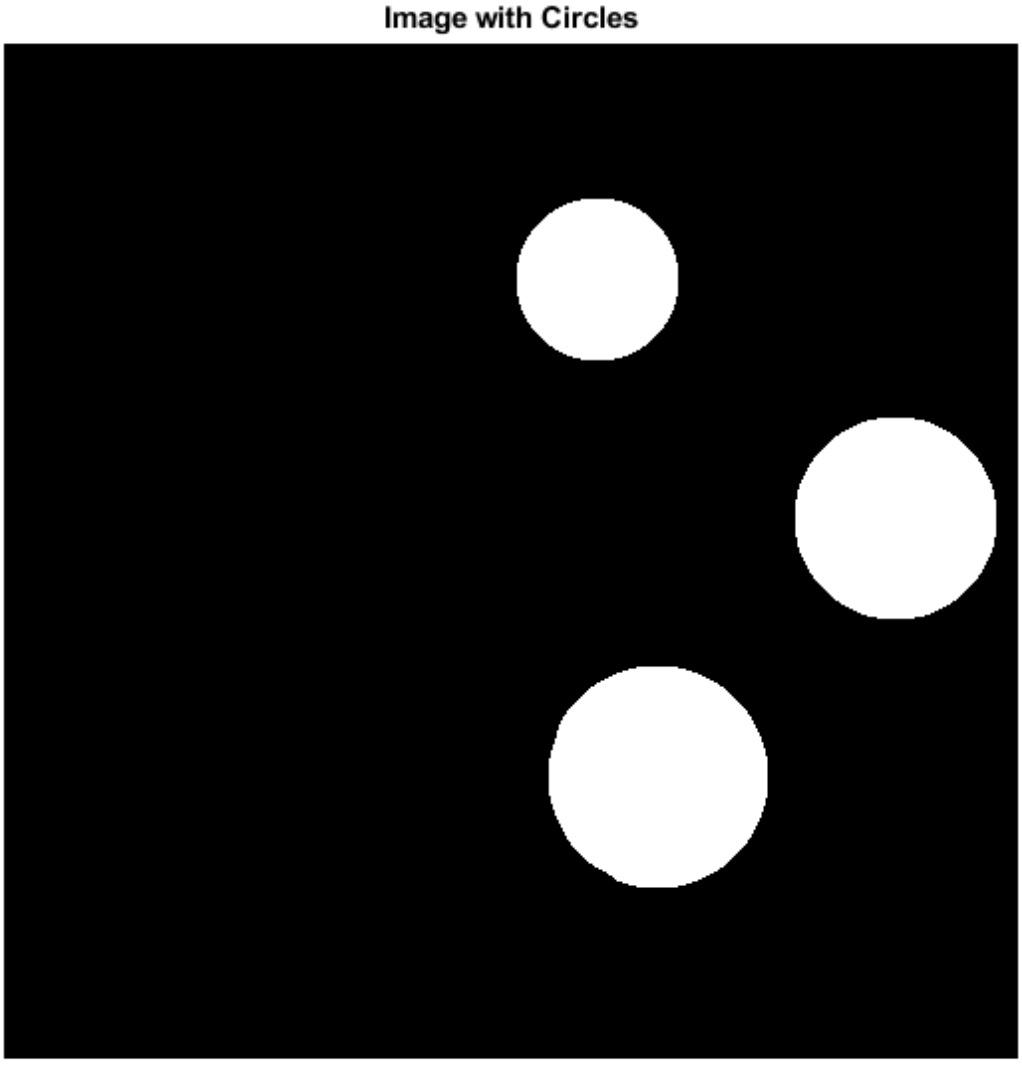
Estimate the center and radii of circular objects in an image and use this information to plot circles on the image. In this example, `regionprops` returns the measured region properties in a table.

Read an image into workspace.

```
a = imread('circlesBrightDark.png');
```

Turn the input image into a binary image.

```
bw = a < 100;
imshow(bw)
title('Image with Circles')
```



Calculate properties of regions in the image and return the data in a table.

```
stats = regionprops('table',bw,'Centroid',...
    'MajorAxisLength','MinorAxisLength')
```

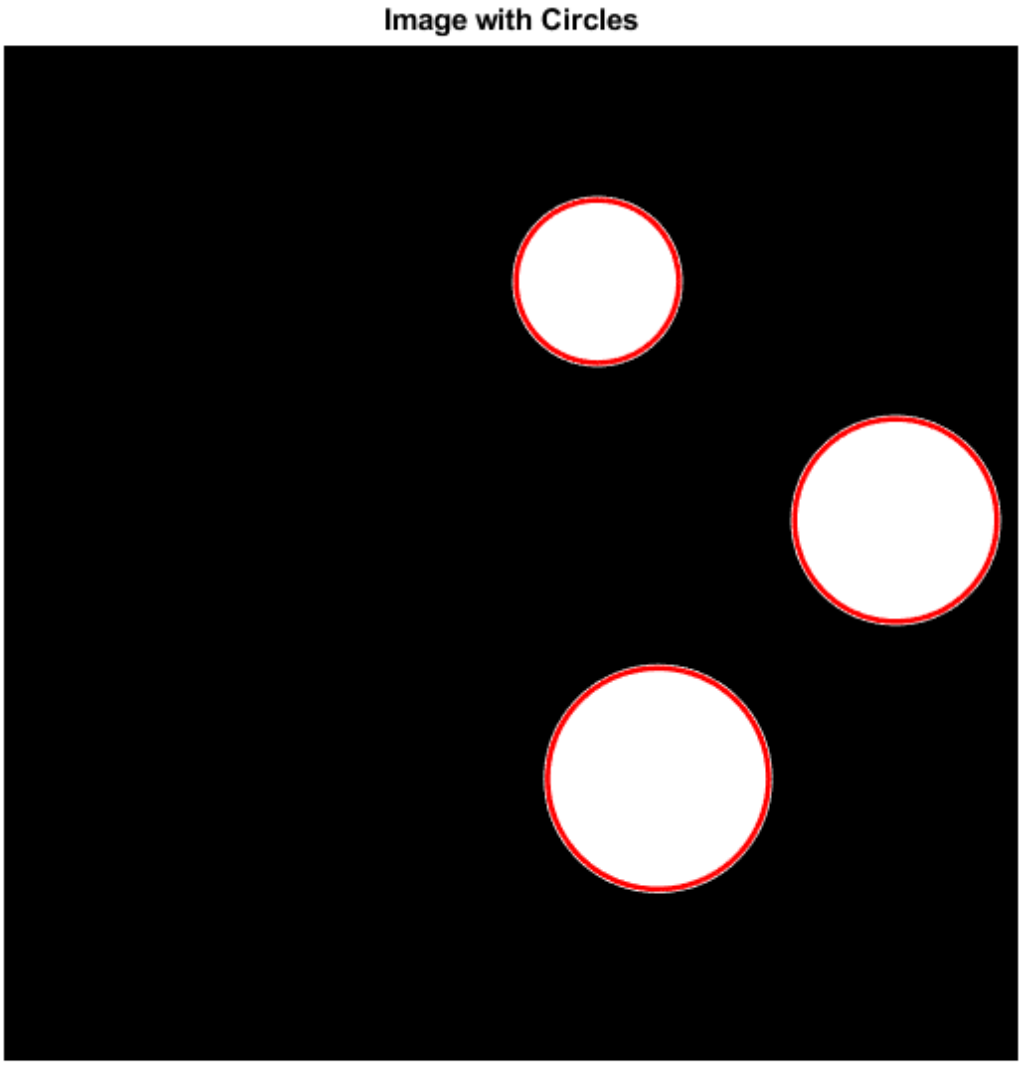
```
stats=4×3 table
        Centroid        MajorAxisLength    MinorAxisLength
    ________________    _______________    _______________

     256.5     256.5        834.46             834.46
       300       120        81.759             81.759
    330.47    369.83        111.78             110.36
       450       240        101.72             101.72
```

Get centers and radii of the circles.

```
centers = stats.Centroid;
diameters = mean([stats.MajorAxisLength stats.MinorAxisLength],2);
radii = diameters/2;
```

Plot the circles.

```
hold on
viscircles(centers,radii);
hold off
```



Input Arguments

`BW` — Binary image

logical array

Binary image, specified as a logical array of any dimension.

Data Types: `logical`

`CC` — Connected components

structure

Connected components, specified as a structure returned by `bwconncomp`.

Data Types: `struct`

`L` — Label image

numeric array | categorical array

Label image, specified as one of the following.

- A numeric array of any dimension. Pixels labeled `0` are the background. Pixels labeled `1` make up one object; pixels labeled `2` make up a second object; and so on. `regionprops` treats negative-valued pixels as background and rounds down input pixels that are not integers. You can get a numeric label image from labeling functions such as `watershed` or `labelmatrix`.
- A categorical array. Each category corresponds to a different region.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `categorical`

`properties` — Type of measurement

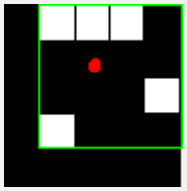
`"basic"` (default) | comma-separated list of string scalars or character vectors | array of string scalars | cell array of character vectors | `"all"`

Type of measurement, specified as a comma-separated list of string scalars or character vectors, an array of string scalars, a cell array of character vectors, or as `"all"` or `"basic"`.

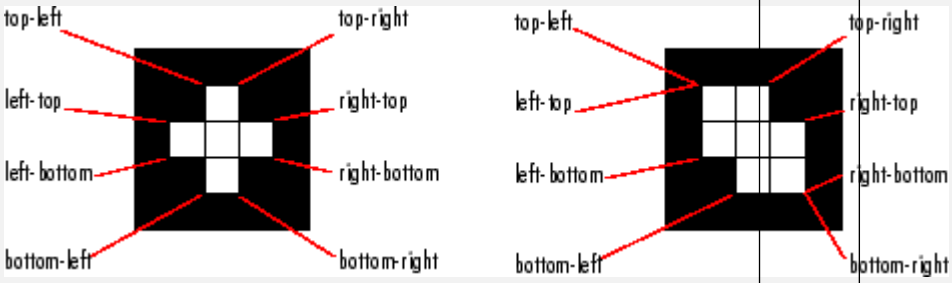
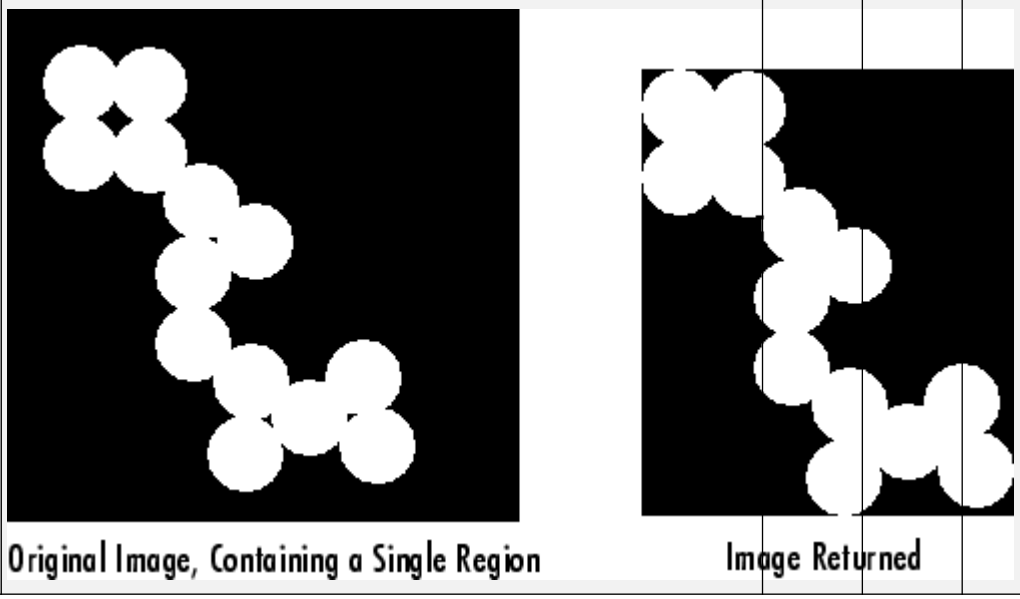
- If you specify `"all"`, then `regionprops` computes all the shape measurements and, for grayscale images, the pixel value measurements as well.
- If you specify `"basic"`, then `regionprops` computes only the `"Area"`, `"Centroid"`, and `"BoundingBox"` measurements.

The following tables list all the properties that provide shape measurements. The properties listed in the Pixel Value Measurements table are valid only when you specify a grayscale image.

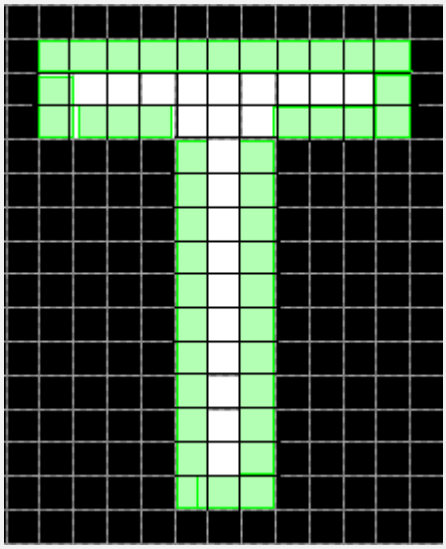
Shape Measurements

Property Name	Description	N-D Support	GPU Support	Code Generation
`"Area"`	Actual number of pixels in the region, returned as a scalar. (This value might differ slightly from the value returned by `bwarea`, which weights different patterns of pixels differently.) To find the equivalent to the area of a 3-D volume, use the `"Volume"` property of `regionprops3`.	Yes	Yes	Yes
`"BoundingBox"`	Position and size of the smallest box containing the region, returned as a 1-by-(2*Q) vector, where Q is the image dimensionality. The first Q elements are the coordinates of the minimum corner of the box. The second Q elements are the size of the box along each dimension. For example, a 2-D bounding box with value `[5.5 8.5 11 14]` indicates that the (x,y) coordinate of the top-left corner of the box is (5.5, 8.5), the horizontal width of the box is 11 pixels, and the vertical height of the box is 14 pixels.	Yes	Yes	Yes
`"Centroid"`	Center of mass of the region, returned as a 1-by-Q vector, where Q is the image dimensionality. The first element of `Centroid` is the horizontal coordinate (or x-coordinate) of the center of mass. The second element is the vertical coordinate (or y-coordinate). All other elements of `Centroid` are in order of dimension. This figure illustrates the centroid and bounding box for a discontiguous region. The region consists of the white pixels. The green box is the bounding box, and the red dot is the centroid. 	Yes	Yes	Yes

Property Name	Description	N-D Support	GPU Support	Code Generation
`"Circularity"`	Roundness of objects, returned as a structure with field `Circularity`. The structure contains the circularity value for each object in the input image. The circularity value is computed as `(4*Area*pi)/(Perimeter^2)`. For a perfect circle, the circularity value is 1. The input must be a label matrix or binary image with contiguous regions. If the image contains discontiguous regions, `regionprops` returns unexpected results. **Note** `Circularity` is not recommended for very small objects such as a 3-by-3 square. For such cases, the results might exceed the circularity value for a perfect circle.	2-D only	No	Yes
`"ConvexArea"`	Number of pixels in `ConvexImage`, returned as a scalar.	2-D only	No	No
`"ConvexHull"`	Smallest convex polygon that can contain the region, returned as a *p*-by-2 matrix. Each row of the matrix contains the *x*- and *y*-coordinates of one vertex of the polygon.	2-D only	No	No
`"ConvexImage"`	Image that specifies the convex hull, with all pixels within the hull filled in (set to `on`), returned as a binary image. The image is the size of the bounding box of the region. For pixels that the boundary of the hull passes through, `regionprops` uses the algorithm described by “Classify Pixels That Are Partially Enclosed by ROI”.	2-D only	No	No
`"Eccentricity"`	Eccentricity of the ellipse that has the same second-moments as the region, returned as a scalar. The eccentricity is the ratio of the distance between the foci of the ellipse and its major axis length. The value is between 0 and 1. (0 and 1 are degenerate cases. An ellipse whose eccentricity is 0 is actually a circle, while an ellipse whose eccentricity is 1 is a line segment.)	2-D only	Yes	Yes
`"EquivDiameter"`	Diameter of a circle with the same area as the region, returned as a scalar. Computed as `sqrt(4*Area/pi)`.	2-D only	Yes	Yes
`"EulerNumber"`	Number of objects in the region minus the number of holes in those objects, returned as a scalar. This property is supported only for 2-D label matrices. `regionprops` uses 8-connectivity to compute the Euler number (also known as the Euler characteristic). To learn more about connectivity, see “Pixel Connectivity”.	2-D only	No	Yes
`"Extent"`	Ratio of pixels in the region to pixels in the total bounding box, returned as a scalar. Computed as the `Area` divided by the area of the bounding box.	2-D only	Yes	Yes

Property Name	Description	N-D Support	GPU Support	Code Generation
`"Extrema"`	Extrema points in the region, returned as an 8-by-2 matrix. Each row of the matrix contains the *x*- and *y*-coordinates of one of the points. The format of the vector is `[top-left top-right right-top right-bottom bottom-right bottom-left left-bottom left-top]`. For some shapes, multiple extrema points can have identical coordinates. This figure illustrates the extrema of two different regions. In the region on the left, each extrema point is distinct. For the region on the right, certain extrema points (such as `top-left` and `left-top`) are identical. 	2-D only	Yes	Yes
`"FilledArea"`	Number of `on` pixels in `FilledImage`, returned as a scalar.	Yes	No	Yes
`"FilledImage"`	Image the same size as the bounding box of the region, returned as a binary array. The `on` pixels correspond to the region, with all holes filled in, as shown in this figure. 	Yes	No	Yes

<table>
<tr><th>Property Name</th><th>Description</th><th>N-D Support</th><th>GPU Support</th><th>Code Generation</th></tr>
<tr><td>"Image"</td><td>Image the same size as the bounding box of the region, returned as a binary array. The on pixels correspond to the region, and all other pixels are off.</td><td>Yes</td><td>Yes</td><td>Yes</td></tr>
<tr><td>"MajorAxisLength"</td><td>Length (in pixels) of the major axis of the ellipse that has the same normalized second central moments as the region, returned as a scalar.</td><td>2-D only</td><td>Yes</td><td>Yes</td></tr>
<tr><td>"MaxFeretProperties"</td><td>Feret properties that include maximum Feret diameter, its relative angle, and coordinate values, returned as a structure with fields:
<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>MaxFeretDiameter</td><td>Maximum Feret diameter measured as the maximum distance between any two boundary points on the antipodal vertices of convex hull that enclose the object.</td></tr>
<tr><td>MaxFeretAngle</td><td>Angle of the maximum Feret diameter with respect to horizontal axis of the image.</td></tr>
<tr><td>MaxFeretCoordinates</td><td>Endpoint coordinates of the maximum Feret diameter.</td></tr>
</table>
The input can be a binary image, connected component, or a label matrix.</td><td>2-D only</td><td>No</td><td>No</td></tr>
<tr><td>"MinFeretProperties"</td><td>Feret properties that include minimum Feret diameter, its relative angle, and coordinate values, returned as a structure with fields:
<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>MinFeretDiameter</td><td>Minimum Feret diameter measured as the minimum distance between any two boundary points on the antipodal vertices of convex hull that enclose the object.</td></tr>
<tr><td>MinFeretAngle</td><td>Angle of the minimum Feret diameter with respect to horizontal axis of the image.</td></tr>
<tr><td>MinFeretCoordinates</td><td>Endpoint coordinates of the minimum Feret diameter.</td></tr>
</table>
The input can be a binary image, a connected component, or a label matrix.</td><td>2-D only</td><td>No</td><td>No</td></tr>
</table>

Property Name	Description	N-D Support	GPU Support	Code Generation
`"MinorAxisLength"`	Length (in pixels) of the minor axis of the ellipse that has the same normalized second central moments as the region, returned as a scalar.	2-D only	Yes	Yes
`"Orientation"`	Angle between the *x*-axis and the major axis of the ellipse that has the same second-moments as the region, returned as a scalar. The value is in degrees, ranging from -90 degrees to 90 degrees. This figure illustrates the axes and orientation of the ellipse. The left side of the figure shows an image region and its corresponding ellipse. The right side shows the same ellipse with the solid blue lines representing the axes. The red dots are the foci. The orientation is the angle between the horizontal dotted line and the major axis. 	2-D only	Yes	Yes
`"Perimeter"`	Distance around the boundary of the region returned as a scalar. `regionprops` computes the perimeter by calculating the distance between each adjoining pair of pixels around the border of the region. If the image contains discontiguous regions, `regionprops` returns unexpected results. This figure illustrates the pixels included in the perimeter calculation for this object. 	2-D only	No	Yes
`"PixelIdxList"`	Linear indices of the pixels in the region, returned as a *p*-element vector.	Yes	Yes	Yes

Property Name	Description	N-D Support	GPU Support	Code Generation
`"PixelList"`	Locations of pixels in the region, returned as a *p*-by-`Q` matrix. Each row of the matrix has the form `[x y z ...]` and specifies the coordinates of one pixel in the region.	Yes	Yes	Yes
`"Solidity"`	Proportion of the pixels in the convex hull that are also in the region, returned as a scalar. Computed as `Area/ConvexArea`.	2-D only	No	No
`"SubarrayIdx"`	Elements of `L` inside the object bounding box, returned as a cell array that contains indices such that `L(idx{:})` extracts the elements.	Yes	Yes	No

The pixel value measurement properties in the following table are valid only when you specify a grayscale image, `I`.

Pixel Value Measurements

Property Name	Description	N-D Support	GPU Support	Code Generation
`"MaxIntensity"`	Value of the pixel with the greatest intensity in the region, returned as a scalar.	Yes	Yes	Yes
`"MeanIntensity"`	Mean of all the intensity values in the region, returned as a scalar.	Yes	Yes	Yes
`"MinIntensity"`	Value of the pixel with the lowest intensity in the region, returned as a scalar.	Yes	Yes	Yes
`"PixelValues"`	Number of pixels in the region, returned as a *p*-by-1 vector, where *p* is the number of pixels in the region. Each element in the vector contains the value of a pixel in the region.	Yes	Yes	Yes
`"WeightedCentroid"`	Center of the region based on location and intensity value, returned as a *p*-by-`Q` vector of coordinates. The first element of `WeightedCentroid` is the horizontal coordinate (or *x*-coordinate) of the weighted centroid. The second element is the vertical coordinate (or *y*-coordinate). All other elements of `WeightedCentroid` are in order of dimension.	Yes	Yes	Yes

Data Types: `char` | `string` | `cell`

`I` — Image to be measured

grayscale image

Image to be measured, specified as a grayscale image. The size of the image must match the size of the binary image `BW`, connected component structure `CC`, or label image `L`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32`

`output` — Return type

`"struct"` (default) | `"table"`

Return type, specified as either of the following values.

Value	Description
`"struct"`	Returns an array of structures with length equal to the number of objects in `BW`, `CC.NumObjects`, or `max(L(:))`. The fields of the structure array denote different properties for each region, as specified by `properties`.
`"table"`	Returns a `table` with height (number of rows) equal to the number of objects in `BW`, `CC.NumObjects`, or `max(L(:))`. The variables (columns) denote different properties for each region, as specified by `properties`.

Data Types: `char` | `string`

Output Arguments

`stats` — Measurement values

`struct` array (default) | table

Measurement values, returned as an array of structures or a table. The number of structures in the array or the number of rows in the table is equal to the number of objects in `BW`, `CC.NumObjects`, or `max(L(:))`. The fields of each structure or the variables in each row denote the properties calculated for each region, as specified by `properties`. If the input image is a categorical label image `L`, then `stats` includes an additional field or variable with the property `"LabelName"`.

More About

Contiguous Regions and Discontiguous Regions

Contiguous regions are also called objects, connected components, or blobs. A label image `L` containing contiguous regions might look like this:

```
1 1 0 2 2 0 3 3
1 1 0 2 2 0 3 3
```

Elements of `L` equal to 1 belong to the first contiguous region or connected component; elements of `L` equal to 2 belong to the second connected component; and so on.

Discontiguous regions are regions that can contain multiple connected components. A label image containing discontiguous regions might look like this:

```
1 1 0 1 1 0 2 2
1 1 0 1 1 0 2 2
```

Elements of `L` equal to 1 belong to the first region, which is discontiguous and contains two connected components. Elements of `L` equal to 2 belong to the second region, which is a single connected component.

Tips

- The `ismember` function is useful for creating a binary image containing only objects or regions that meet certain criteria. For example, these commands create a binary image containing only the regions whose area is greater than 80 and whose eccentricity is less than 0.8.

  ```
  cc = bwconncomp(BW);
  stats = regionprops(cc,"Area","Eccentricity");
  ```

```
idx = find([stats.Area] > 80 & [stats.Eccentricity] < 0.8);
BW2 = ismember(labelmatrix(cc),idx);
```

- The default connectivity is 8-connected for 2-D images, and maximal connectivity for higher dimensions. To specify nondefault connectivity, use `bwconncomp` to create the connected components and then pass the result to `regionprops`.
- `regionprops` takes advantage of intermediate results when computing related measurements. Therefore, it is fastest to compute all the desired measurements in a single call to `regionprops`.
- Most of the measurements take little time to compute. However, these measurements can take longer, depending on the number of regions in `L`:
  - `"ConvexHull"`
  - `"ConvexImage"`
  - `"ConvexArea"`
  - `"FilledImage"`

Compatibility Considerations

The `regionprops` function stores the `Image`, `ConvexImage`, and `FilledImage` properties as cell arrays in the output table for all inputs

Behavior changed in R2022a

Starting in R2022a, when a table output format is specified, the `regionprops` function stores the `Image`, `ConvexImage`, and `FilledImage` property values as cell arrays, regardless of the size of the image objects. In previous releases, if the size of the bounding box of an object was 1-by-1 or 1-by-*n*, these properties were stored in the output table as a numeric scalar or row vector.

To update your code, access the value of the `Image`, `ConvexImage`, and `FilledImage` properties by using dot notation with curly braces, `{}`. For example, use the below code to access the `Image` property for the first object in the input image `BW`. In previous releases, curly braces were not required to access values stored as a numeric scalar or row vector.

```
stats = regionprops("table",BW,"Image");
imdata = stats.Image{1};
```

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `regionprops` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `regionprops` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.
- Supports only binary images or numeric label images. Input label images of data type categorical are not supported.
- Specifying the output type `"table"` is not supported.

- Passing a cell array of properties is not supported. Use a comma-separated list instead.
- All properties are supported except `"ConvexArea"`, `"ConvexHull"`, `"ConvexImage"`, `"MaxFeretProperties"`, `"MinFeretProperties"`, `"Solidity"`, and `"SubarrayIdx"`.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- GPU Coder generates optimized CUDA code for only binary images. Code generated for input label images is not optimized. Input label images of data type categorical are not supported.
- Specifying the output type `"table"` is not supported.
- Passing a cell array of properties is not supported. Use a comma-separated list instead.
- Only `"Area"`, `"BoundingBox"`, `"Centroid"`, `"Eccentricity"`, `"EquivDiameter"`, `"Extent"`, `"MajorAxisLength"`, `"MinorAxisLength"`, `"Orientation"`, `"PixelIdxList""PixelList"`, `"MaxIntensity"`, `"MeanIntensity"`, `"MinIntensity"`, `"PixelValues"`, and `"WeightedCentroid"` properties are supported.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- `gpuArray` input must be a 2-D logical matrix or a 2-D label matrix.
- The connected component structure (`CC`) input is not supported.
- The following properties are not supported: `"ConvexArea"`, `"ConvexHull"`, `"ConvexImage"`, `"Circularity"`, `"EulerNumber"`, `"FilledArea"`, `"FilledImage"`, `"MaxFeretProperties"`, `"MinFeretProperties"`, and `"Solidity"`.
- `"struct"` is the only return type supported.

For more information, see “Image Processing on a GPU”.

See Also

Image Region Analyzer | `regionprops3` | `bwpropfilt` | `bwconncomp` | `bwferet` | `watershed` | `labelmatrix`

Topics
“Measuring Regions in Grayscale Images”
“Identifying Round Objects”
“Classify Pixels That Are Partially Enclosed by ROI”

Introduced before R2006a

regionprops3

Measure properties of 3-D volumetric image regions

Syntax

```
stats = regionprops3(BW,properties)
stats = regionprops3(CC,properties)
stats = regionprops3(L,properties)
stats = regionprops3(___,V,properties)
```

Description

`stats = regionprops3(BW,properties)` measures a set of properties for each connected component (object) in the 3-D volumetric binary image `BW`. The output `stats` denote different properties for each object.

For all syntaxes, you can omit the `properties` argument. In this case, `regionprops3` returns the `"Volume"`, `"Centroid"`, and `"BoundingBox"` measurements.

`stats = regionprops3(CC,properties)` measures a set of properties for each connected component (object) in `CC`, which is a structure returned by `bwconncomp`.

`stats = regionprops3(L,properties)` measures a set of properties for each labeled region in the 3-D label image `L`.

`stats = regionprops3(___,V,properties)` measures a set of properties for each labeled region in the 3-D volumetric grayscale image `V`. The first input (`BW`, `CC`, or `L`) identifies the regions in `V`.

Examples

Estimate Centers and Radii of Objects in 3-D Volumetric Image

Create a binary image with two spheres.

```
[x,y,z] = meshgrid(1:50,1:50,1:50);
bw1 = sqrt((x-10).^2 + (y-15).^2 + (z-35).^2) < 5;
bw2 = sqrt((x-20).^2 + (y-30).^2 + (z-15).^2) < 10;
bw = bw1 | bw2;
```

Get the centers and radii of the two spheres.

```
s = regionprops3(bw,"Centroid","PrincipalAxisLength");
centers = s.Centroid
```

```
centers = 2×3

    20    30    15
    10    15    35
```

```
diameters = mean(s.PrincipalAxisLength,2)
```

```
diameters = 2×1

   19.9641
    9.8241
```

```
radii = diameters/2
```

```
radii = 2×1

    9.9820
    4.9120
```

Get All Statistics for Cube Within a Cube

Make a 9-by-9 cube of 0s that contains a 3-by-3 cube of 1s at its center.

```
innercube = ones(3,3,3);
cube_in_cube = padarray(innercube,[3 3],0,'both');
```

Get all statistics on the cube within the cube.

```
stats = regionprops3(cube_in_cube,'all')
```

```
stats=1×18 table
    Volume     Centroid      BoundingBox                SubarrayIdx                      Image
    ______    ___________    ___________    ____________________________________    _____________

      27      5    5    2    1x6 double     {[4 5 6]}    {[4 5 6]}    {[1 2 3]}    {3x3x3 logica
```

Input Arguments

BW — Volumetric binary image

3-D logical array

Volumetric binary image, specified as a 3-D logical array.

Data Types: `logical`

CC — Connected components

structure

Connected components of a 3-D volumetric image, specified as a structure returned by `bwconncomp` using a 3-D connectivity value, such as 6, 18, or 26. `CC.ImageSize` must be a 1-by-3 vector.

Data Types: `struct`

L — Label image

3-D numeric array | 3-D categorical array

Label image, specified as one of the following.

- A 3-D numeric array. Voxels labeled `0` are the background. Voxels labeled `1` make up one object; voxels labeled `2` make up a second object; and so on. `regionprops3` treats negative-valued voxels as background and rounds down input voxels that are not integers. You can get a numeric label image from labeling functions such as `watershed` or `labelmatrix`.
- A 3-D categorical array. Each category corresponds to a different region.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `categorical`

`properties` — Type of measurement
`"basic"` (default) | comma-separated list of strings or character vectors | cell array of strings or character vectors | `"all"`

Type of measurement, specified as a comma-separated list of strings or character vectors, a cell array of strings or character vectors, `"all"` or `"basic"`.

- If you specify `"all"`, then `regionprops3` computes all the shape measurements. If you also specify a grayscale image, then `regionprops3` returns all of the voxel value measurements.
- If you specify `"basic"` or do not specify the `properties` argument, then `regionprops3` computes only the `"Volume"`, `"Centroid"`, and `"BoundingBox"` measurements.

The following table lists all the properties that provide shape measurements. The Voxel Value Measurements table lists additional properties that are valid only when you specify a grayscale image.

Shape Measurements

Property Name	Description
`"BoundingBox"`	Smallest cuboid containing the region, returned as a 1-by-6 vector of the form `[ulf_x ulf_y ulf_z width_x width_y width_z]`. `ulf_x`, `ulf_y`, and `ulf_z` specify the upper-left front corner of the cuboid. `width_x`, `width_y`, and `width_z` specify the width of the cuboid along each dimension.
`"Centroid"`	Center of mass of the region, returned as a 1-by-3 vector. The three elements specify the (*x*, *y*, *z*) coordinates of the center of mass.
`"ConvexHull"`	Smallest convex polygon that can contain the region, returned as a *p*-by-3 matrix. Each row of the matrix contains the *x*-, *y*-, and *z*-coordinates of one vertex of the polygon.
`"ConvexImage"`	Image of the convex hull, returned as a volumetric binary image with all voxels within the hull filled in (set to `on`). The image is the size of the bounding box of the region.
`"ConvexVolume"`	Number of voxels in `ConvexImage`, returned as a scalar.
`"EigenValues"`	Eigenvalues of the voxels representing a region, returned as a 3-by-1 vector. `regionprops3` uses the eigenvalues to calculate the principal axes lengths.
`"EigenVectors"`	Eigenvectors of the voxels representing a region, returned as a 3-by-3 vector. `regionprops3` uses the eigenvectors to calculate the orientation of the ellipsoid that has the same normalized second central moments as the region.
`"EquivDiameter"`	Diameter of a sphere with the same volume as the region, returned as a scalar. Computed as `(6*Volume/pi)^(1/3)`.
`"Extent"`	Ratio of voxels in the region to voxels in the total bounding box, returned as a scalar. Computed as the value of `Volume` divided by the volume of the bounding box. `[Volume/(bounding box width * bounding box height * bounding box depth)]`
`"Image"`	Bounding box of the region, returned as a volumetric binary image that is the same size as the bounding box of the region. The `on` voxels correspond to the region, and all other voxels are `off`.
`"Orientation"`	Euler angles [2], returned as a 1-by-3 vector. The angles are based on the right-hand rule. `regionprops3` interprets the angles by looking at the origin along the *x*-, *y*-, and *z*-axis representing roll, pitch, and yaw respectively. A positive angle represents a rotation in the counterclockwise direction. Rotation operations are not commutative so they must be applied in the correct order to have the intended effect.
`"PrincipalAxisLength"`	Length (in voxels) of the major axes of the ellipsoid that have the same normalized second central moments as the region, returned as 1-by-3 vector. `regionprops3` sorts the values from highest to lowest.
`"Solidity"`	Proportion of the voxels in the convex hull that are also in the region, returned as a scalar. Computed as `Volume/ConvexVolume`.
`"SubarrayIdx"`	Indices used to extract elements inside the object bounding box, returned as a cell array such that `L(idx{:})` extracts the elements of `L` inside the object bounding box.
`"SurfaceArea"`	Distance around the boundary of the region [1], returned as a scalar.

Property Name	Description
"Volume"	Count of the actual number of on voxels in the region, returned as a scalar. Volume represents the metric or measure of the number of voxels in the regions within the volumetric binary image, BW.
"VoxelIdxList"	Linear indices of the voxels in the region, returned as a *p*-element vector.
"VoxelList"	Locations of voxels in the region, returned as a *p*-by-3 matrix. Each row of the matrix has the form [x y z] and specifies the coordinates of one voxel in the region.

The voxel value measurement properties in the following table are valid only when you specify a grayscale volumetric image, V.

Voxel Value Measurements

Property Name	Description
"MaxIntensity"	Value of the voxel with the greatest intensity in the region, returned as a scalar.
"MeanIntensity"	Mean of all the intensity values in the region, returned as a scalar.
"MinIntensity"	Value of the voxel with the lowest intensity in the region, returned as a scalar.
"VoxelValues"	Value of the voxels in the region, returned as a *p*-by-1 vector, where *p* is the number of voxels in the region. Each element in the vector contains the value of a voxel in the region.
"WeightedCentroid"	Center of the region based on location and intensity value, returned as a *p*-by-3 vector of coordinates. The first element of WeightedCentroid is the horizontal coordinate (or *x*-coordinate) of the weighted centroid. The second element is the vertical coordinate (or *y*-coordinate). The third element is the planar coordinate (or *z*-coordinate).

Data Types: char | string | cell

V — Volumetric grayscale image

3-D numeric array

Volumetric grayscale image, specified as a 3-D numeric array. The size of the image must match the size of the binary image BW, connected component structure CC, or label matrix L.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32

Output Arguments

stats — Measurement values

table

Measurement values, returned as a table. The number of rows in the table corresponds to the number of objects in BW, CC.NumObjects, or max(L(:)). The variables (columns) in each table row denote the properties calculated for each region, as specified by properties. If the input image is a categorical label image L, then stats includes an additional variable with the property "LabelName".

References

[1] Lehmann, Gaetan and David Legland. *Efficient N-Dimensional surface estimation using Crofton formula and run-length encoding*, The Insight Journal, 2012. https://insight-journal.org/browse/publication/852.

[2] Shoemake, Ken, *Graphics Gems IV*. Edited by Paul S. Heckbert, Morgan Kaufmann, 1994, pp. 222–229.

See Also

`bwconncomp` | `bwlabeln` | `ismember` | `regionprops`

Introduced in R2017b

MattesMutualInformation

Mattes mutual information metric configuration

Description

A `MattesMutualInformation` object describes a mutual information metric configuration that you pass to the function `imregister` to solve image registration problems.

Creation

You can create a `MattesMutualInformation` object using the following methods:

- `imregconfig` — Returns a `MattesMutualInformation` object paired with an appropriate optimizer for registering multimodal images
- Entering

  ```
  metric = registration.metric.MattesMutualInformation;
  ```

  on the command line creates a `MattesMutualInformation` object with default settings

Properties

`NumberOfSpatialSamples` — Number of spatial samples used to compute the mutual information metric
`500` (default) | positive integer scalar

Number of spatial samples used to compute the mutual information metric, specified as a positive integer scalar. `NumberOfSpatialSamples` defines the number of random pixels `imregister` uses to compute the metric. Your registration results are more reproducible (at the cost of performance) as you increase this value. `imregister` only uses `NumberOfSpatialSamples` when `UseAllPixels = 0` (`false`).

Data Types: `double` | `single` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

`NumberOfHistogramBins` — Number of histogram bins used to compute the mutual information metric
`50` (default) | positive integer scalar

Number of histogram bins used to compute the mutual information metric, specified as a positive integer scalar. `NumberOfHistogramBins` defines the number of bins `imregister` uses to compute the joint distribution histogram. The minimum value is `5`.

Data Types: `double` | `single` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

`UseAllPixels` — Option to include all pixels in the overlap region when computing the mutual information metric
`1` (`true`) (default) | logical scalar

Option to compute the metric using all pixels in the overlap region of the images when computing the mutual information metric, specified as a logical scalar.

You can achieve significantly better performance if you set this property to `0` (`false`). When `UseAllPixels = 0`, the `NumberOfSpatialSamples` property controls the number of random pixel locations that `imregister` uses to compute the metric. The results of your registration might not be reproducible when `UseAllPixels = 0`. This is because `imregister` selects a random subset of pixels from the images to compute the metric.

Examples

Register Images with Mattes Mutual Information Metric

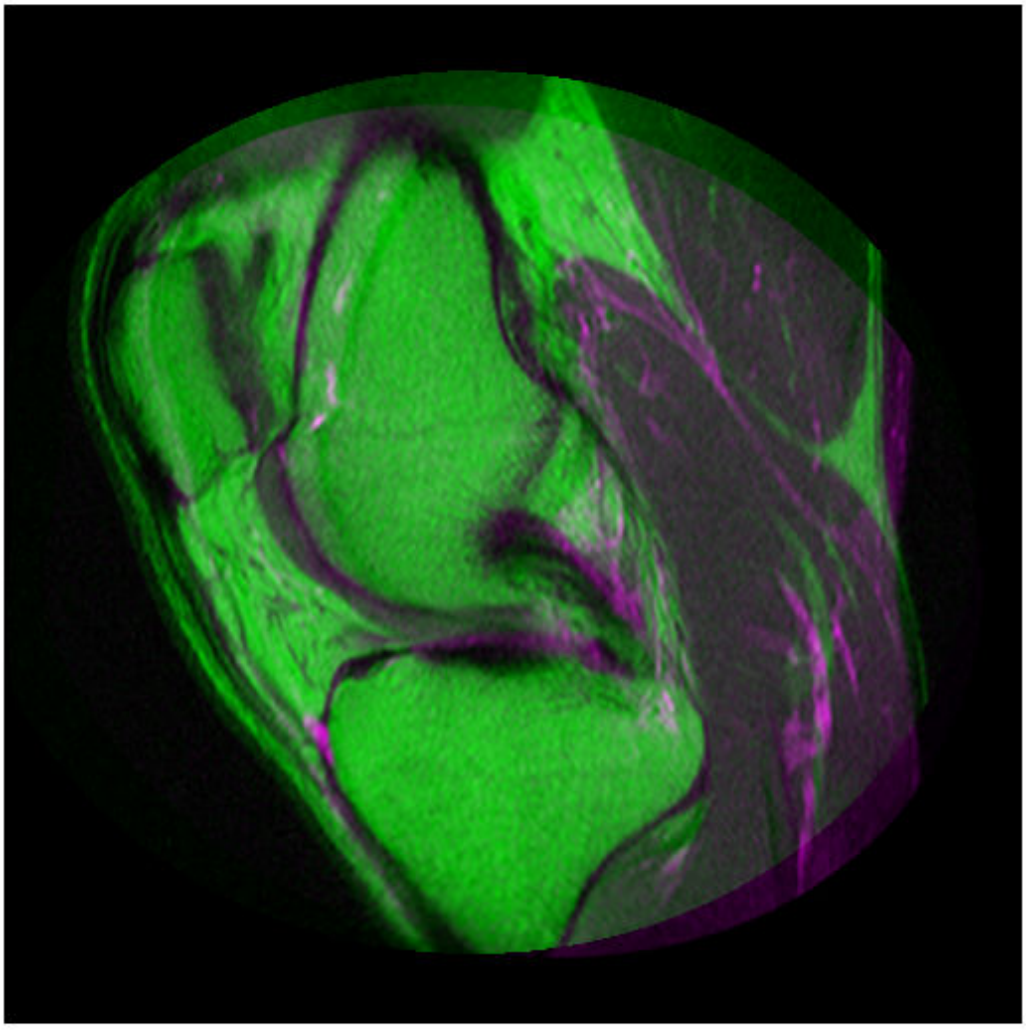
Create a `MattesMutualInformation` object and use it to register two MRI images of a knee that were obtained using different protocols.

Read the images into the workspace. The images are multimodal because they have different brightness and contrast.

```
fixed  = dicomread('knee1.dcm');
moving = dicomread('knee2.dcm');
```

View the misaligned images.

```
figure
imshowpair(fixed, moving,'Scaling','joint');
```



Create the optimizer configuration object suitable for registering multimodal images.

```
optimizer = registration.optimizer.OnePlusOneEvolutionary;
```

Create the metric configuration object suitable for registering multimodal images.

```
metric = registration.metric.MattesMutualInformation
```

```
metric =
  registration.metric.MattesMutualInformation

  Properties:
    NumberOfSpatialSamples: 500
     NumberOfHistogramBins: 50
               UseAllPixels: 1
```

Tune the properties of the optimizer so that the problem will converge on a global maxima. Increase the number of iterations the optimizer will use to solve the problem.

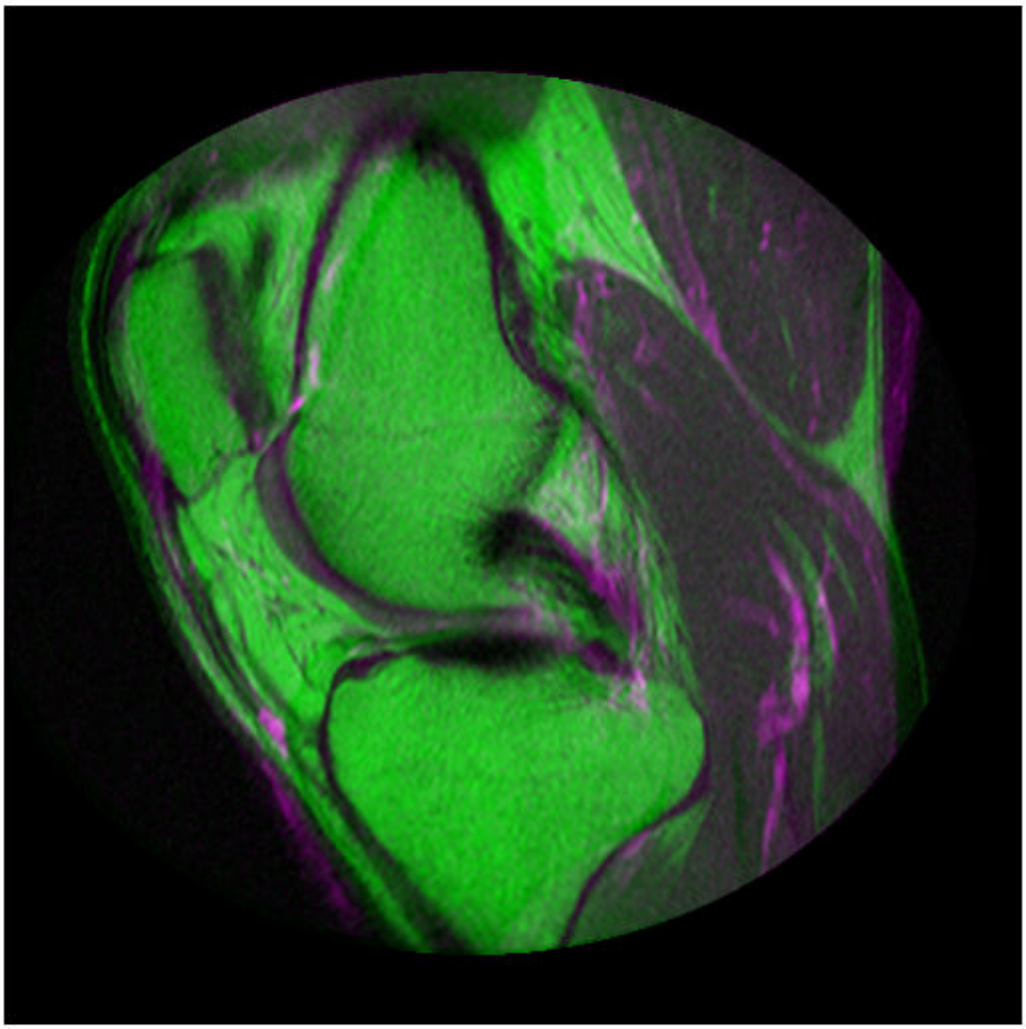
```
optimizer.InitialRadius = 0.009;
optimizer.Epsilon = 1.5e-4;
optimizer.GrowthFactor = 1.01;
optimizer.MaximumIterations = 300;
```

Perform the registration.

```
movingRegistered = imregister(moving,fixed,'affine',optimizer,metric);
```

View the registered images.

```
figure
imshowpair(fixed, movingRegistered,'Scaling','joint');
```



Tips

- Larger values of mutual information correspond to better registration results. You can examine the computed values of Mattes mutual information if you enable `'DisplayOptimization'` when you call `imregister`, for example:

```
movingRegistered = imregister(moving,fixed,'rigid',optimizer,metric,'DisplayOptimization',true);
```

Algorithms

Mutual information metrics are information theoretic techniques for measuring how related two variables are. These algorithms use the joint probability distribution of a sampling of pixels from two images to measure the certainty that the values of one set of pixels map to similar values in the other image. This information is a quantitative measure of how similar the images are. High mutual information implies a large reduction in the uncertainty (entropy) between the two distributions, signaling that the images are likely better aligned.

The Mattes mutual information algorithm uses a single set of pixel locations for the duration of the optimization, instead of drawing a new set at each iteration. The number of samples used to compute the probability density estimates and the number of bins used to compute the entropy are both user selectable. The marginal and joint probability density function is evaluated at the uniformly spaced bins using the samples. Entropy values are computed by summing over the bins. Zero-order and third-order B-spline kernels are used to compute the probability density functions of the fixed and moving images, respectively [1].

References

[1] Rahunathan, Smriti, D. Stredney, P. Schmalbrock, and B.D. Clymer. Image Registration Using Rigid Registration and Maximization of Mutual Information. Poster presented at: MMVR13. The 13th Annual Medicine Meets Virtual Reality Conference; 2005 January 26–29; Long Beach, CA.

[2] D. Mattes, D.R. Haynor, H. Vesselle, T. Lewellen, and W. Eubank. "Non-rigid multimodality image registration." (Proceedings paper).*Medical Imaging 2001: Image Processing*. SPIE Publications, 3 July 2001. pp. 1609–1620.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Functions

`imregister` | `imregconfig`

Objects
`MeanSquares` | `OnePlusOneEvolutionary` | `RegularStepGradientDescent`

Topics
“Create an Optimizer and Metric for Intensity-Based Image Registration”

Introduced in R2012a

MeanSquares

Mean square error metric configuration

Description

A `MeanSquares` object describes a mean square error metric configuration that you pass to the function `imregister` to solve image registration problems.

Creation

You can create a `MeanSquares` object using the following methods:

- `imregconfig` — Returns a `MeanSquares` object paired with an appropriate optimizer for registering monomodal images
- Entering

  ```
  metric = registration.metric.MeanSquares;
  ```

  on the command line creates a `MeanSquares` object

Examples

Register Images with Mean Squares Metric

Create a `MeanSquares` object and use it to register two images with similar brightness and contrast.

Read the reference image and create an unregistered copy.

```
fixed = imread('pout.tif');
moving = imrotate(fixed,5,'bilinear','crop');
```

View the misaligned images.

```
imshowpair(fixed,moving,'Scaling','joint');
```

Create the metric configuration object suitable for registering monomodal images.

```
metric = registration.metric.MeanSquares
```

```
metric =
  registration.metric.MeanSquares

  This class has no properties.
```

Create the optimizer configuration object.

```
optimizer = registration.optimizer.RegularStepGradientDescent;
```

Modify the optimizer configuration to get more precision.

```
optimizer.MaximumIterations = 300;
optimizer.MinimumStepLength = 5e-4;
```

Perform the registration.

```
movingRegistered = imregister(moving,fixed,'rigid',optimizer,metric);
```

View the registered images.

```
imshowpair(fixed, movingRegistered,'Scaling','joint');
```

Tips

- The mean squares metric is an element-wise difference between two input images. The ideal value is zero. You can examine the computed values of mean square error if you enable `'DisplayOptimization'` when you call `imregister`. For example, `movingRegistered = imregister(moving,fixed,'rigid',optimizer,metric,'DisplayOptimization',true);`

Algorithms

The mean squares image similarity metric is computed by squaring the difference of corresponding pixels in each image and taking the mean of the squared differences.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Functions
`imregister` | `imregconfig`

Objects
`MattesMutualInformation` | `OnePlusOneEvolutionary` | `RegularStepGradientDescent`

Topics
“Create an Optimizer and Metric for Intensity-Based Image Registration”

Introduced in R2012a

OnePlusOneEvolutionary

One-plus-one evolutionary optimizer configuration

Description

A `OnePlusOneEvolutionary` object describes a one-plus-one evolutionary optimization configuration that you pass to the function `imregister` to solve image registration problems.

Creation

You can create a `OnePlusOneEvolutionary` object using the following methods:

- `imregconfig` — Returns a `OnePlusOneEvolutionary` object paired with an appropriate metric for registering multimodal images
- Entering

  ```
  metric = registration.optimizer.OnePlusOneEvolutionary;
  ```

  on the command line creates a `OnePlusOneEvolutionary` object with default settings

Properties

`GrowthFactor` — Growth factor of the search radius
1.05 (default) | positive scalar

Growth factor of the search radius, specified as a positive scalar. The optimizer uses `GrowthFactor` to control the rate at which the search radius grows in parameter space. If you set `GrowthFactor` to a large value, the optimization is fast, but it might result in finding only the metric's local extrema. If you set `GrowthFactor` to a small value, the optimization is slower, but it is likely to converge on a better solution.

Data Types: `double` | `single` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

`Epsilon` — Minimum size of the search radius
`1.5e-6` (default) | positive scalar

Minimum size of the search radius, specified as a positive scalar. `Epsilon` controls the accuracy of convergence by adjusting the minimum size of the search radius. If you set `Epsilon` to a small value, the optimization of the metric is more accurate, but the computation takes longer. If you set `Epsilon` to a large value, the computation time decreases at the expense of accuracy.

Data Types: `double` | `single` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

`InitialRadius` — Initial size of search radius
`6.25e-3` | positive scalar

Initial size of search radius, specified as a positive scalar. If you set `InitialRadius` to a large value, the computation time decreases. However, overly large values of `InitialRadius` might result in an optimization that fails to converge.

Data Types: `double` | `single` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

`MaximumIterations` — Maximum number of optimizer iterations
100 (default) | positive integer scalar

Maximum number of optimizer iterations, specified as a positive integer scalar. `MaximumIterations` determines the maximum number of iterations the optimizer performs at any given pyramid level. The registration could converge before the optimizer reaches the maximum number of iterations.

Data Types: `double` | `single` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

Examples

Register Images with One Plus One Evolutionary Optimizer

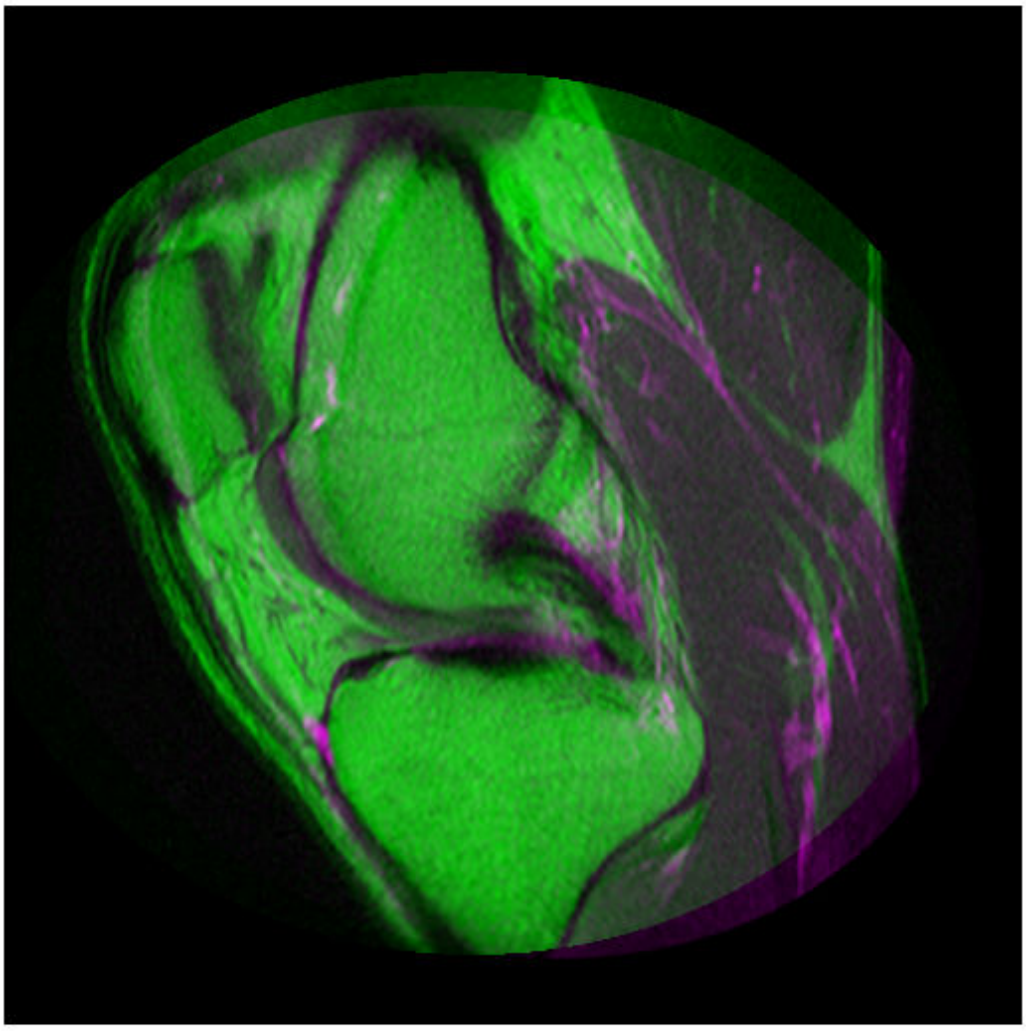
Create a `OnePlusOneEvolutionary` object and use it to register two MRI images of a knee that were obtained using different protocols.

Read the images into the workspace. The images are multimodal because they have different brightness and contrast.

```
fixed  = dicomread('knee1.dcm');
moving = dicomread('knee2.dcm');
```

View the misaligned images.

```
figure
imshowpair(fixed, moving,'Scaling','joint');
```



Create the optimizer configuration object suitable for registering multimodal images.

```
optimizer = registration.optimizer.OnePlusOneEvolutionary
```

```
optimizer =
  registration.optimizer.OnePlusOneEvolutionary

  Properties:
         GrowthFactor: 1.050000e+00
              Epsilon: 1.500000e-06
        InitialRadius: 6.250000e-03
    MaximumIterations: 100
```

Create the metric configuration object suitable for registering multimodal images.

```
metric = registration.metric.MattesMutualInformation;
```

Tune the properties of the optimizer so that the problem will converge on a global maxima. Increase the number of iterations the optimizer will use to solve the problem.

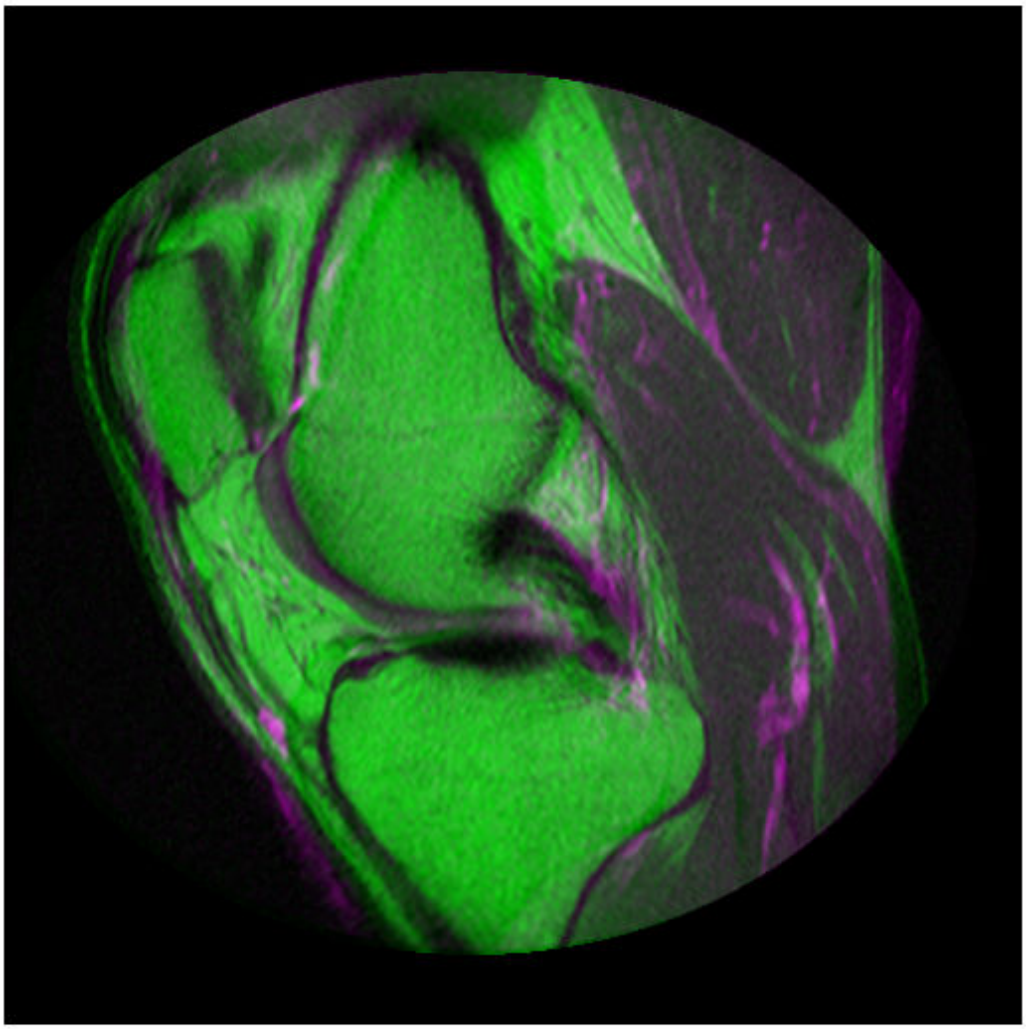
```
optimizer.InitialRadius = 0.009;
optimizer.Epsilon = 1.5e-4;
optimizer.GrowthFactor = 1.01;
optimizer.MaximumIterations = 300;
```

Perform the registration.

```
movingRegistered = imregister(moving,fixed,'affine',optimizer,metric);
```

View the registered images.

```
figure
imshowpair(fixed, movingRegistered,'Scaling','joint');
```



Algorithms

An evolutionary algorithm iterates to find a set of parameters that produce the best possible registration result. It does this by perturbing, or mutating, the parameters from the last iteration (the parent). If the new (child) parameters yield a better result, then the child becomes the new parent whose parameters are perturbed, perhaps more aggressively. If the parent yields a better result, it remains the parent and the next perturbation is less aggressive.

References

[1] Styner, M., C. Brechbuehler, G. Székely, and G. Gerig. "Parametric estimate of intensity inhomogeneities applied to MRI." *IEEE Transactions on Medical Imaging*. Vol. 19, Number 3, 2000, pp. 153-165.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Functions

`imregister` | `imregconfig`

Objects

`MattesMutualInformation` | `MeanSquares` | `RegularStepGradientDescent`

Topics

“Create an Optimizer and Metric for Intensity-Based Image Registration”

Introduced in R2012a

RegularStepGradientDescent

Regular step gradient descent optimizer configuration

Description

A `RegularStepGradientDescent` object describes a regular step gradient descent optimization configuration that you pass to the function `imregister` to solve image registration problems.

Creation

You can create a `RegularStepGradientDescent` object using the following methods:

- `imregconfig` — Returns a `RegularStepGradientDescent` object paired with an appropriate metric for registering monomodal images
- Entering

  ```
  metric = registration.optimizer.RegularStepGradientDescent;
  ```

  on the command line creates a `RegularStepGradientDescent` object with default settings

Properties

`GradientMagnitudeTolerance` — Gradient magnitude tolerance
`1e-4` (default) | positive scalar

Gradient magnitude tolerance, specified as a positive scalar. `GradientMagnitudeTolerance` controls the optimization process. When the value of the gradient is smaller than `GradientMagnitudeTolerance`, it is an indication that the optimizer might have reached a plateau.

Data Types: `double` | `single` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

`MinimumStepLength` — Tolerance for convergence
`1e-5` (default) | positive scalar

Tolerance for convergence, specified as a positive scalar. `MinimumStepLength` controls the accuracy of convergence. If you set `MinimumStepLength` to a small value, the optimization takes longer to compute, but it is likely to converge on a more accurate metric value.

Data Types: `double` | `single` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

`MaximumStepLength` — Initial step length
`0.0625` (default) | positive scalar

Initial step length, specified as a positive scalar. The initial step length is the maximum step length because the optimizer reduces the step size during convergence. If you set `MaximumStepLength` to a large value, the computation time decreases. However, the optimizer might fail to converge if you set `MaximumStepLength` to an overly large value.

Data Types: `double` | `single` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

`MaximumIterations` — Maximum number of iterations

100 (default) | positive integer scalar

Maximum number of iterations, specified as a positive integer scalar. `MaximumIterations` is a positive scalar integer value that determines the maximum number of iterations the optimizer performs at any given pyramid level. The registration could converge before the optimizer reaches the maximum number of iterations.

Data Types: `double` | `single` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

`RelaxationFactor` — Step length reduction factor

0.5 (default) | positive scalar between 0 and 1

Step length reduction factor, specified as a positive scalar between 0 and 1. `RelaxationFactor` defines the rate at which the optimizer reduces step size during convergence. Whenever the optimizer determines that the direction of the gradient changed, it reduces the size of the step length. If your metric is noisy, you can set `RelaxationFactor` to a larger value. This leads to a more stable convergence at the expense of computation time.

Data Types: `double` | `single` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64`

Examples

Register Images with Regular Step Gradient Descent Optimizer

Create a `RegularStepGradientDescent` object and use it to register two images with similar brightness and contrast.

Read the reference image and create an unregistered copy.

```
fixed  = imread('pout.tif');
moving = imrotate(fixed, 5, 'bilinear', 'crop');
```

View the misaligned images.

```
figure
imshowpair(fixed, moving,'Scaling','joint');
```

Create the optimizer configuration object suitable for registering monomodal images.

```
optimizer = registration.optimizer.RegularStepGradientDescent
```

```
optimizer =
  registration.optimizer.RegularStepGradientDescent

  Properties:
    GradientMagnitudeTolerance: 1.000000e-04
               MinimumStepLength: 1.000000e-05
               MaximumStepLength: 6.250000e-02
               MaximumIterations: 100
                RelaxationFactor: 5.000000e-01
```

Create the metric configuration object.

```
metric = registration.metric.MeanSquares;
```

Modify the optimizer configuration to get more precision.

```
optimizer.MaximumIterations = 300;
optimizer.MinimumStepLength = 5e-4;
```

Perform the registration.

```
movingRegistered = imregister(moving,fixed,'rigid',optimizer,metric);
```

View the registered images.

```
figure
imshowpair(fixed, movingRegistered,'Scaling','joint');
```

Algorithms

The regular step gradient descent optimization adjusts the transformation parameters so that the optimization follows the gradient of the image similarity metric in the direction of the extrema. It uses constant length steps along the gradient between computations until the gradient changes direction. At this point, the step length is reduced based on the `RelaxationFactor`, which halves the step length by default.

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

Functions

`imregister` | `imregconfig`

Objects

`MattesMutualInformation` | `MeanSquares` | `OnePlusOneEvolutionary`

Topics
“Create an Optimizer and Metric for Intensity-Based Image Registration”

Introduced in R2012a

rgb2lab

Convert RGB to CIE 1976 L*a*b*

Syntax

```
lab = rgb2lab(RGB)
lab = rgb2lab(RGB,Name,Value)
```

Description

`lab = rgb2lab(RGB)` converts sRGB values to CIE 1976 L*a*b* values.

`lab = rgb2lab(RGB,Name,Value)` specifies additional conversion options, such as the color space of the RGB image, using one or more name-value pair arguments.

Examples

Convert RGB White to L*a*b*

Use `rgb2lab` to convert the RGB white value to L*a*b.

```
rgb2lab([1 1 1])
```

```
ans = 1×3

  100.0000         0    0.0000
```

Convert Color Value to L*a*b* Specifying Color Space

Convert an Adobe RGB (1998) color value to L*a*b* using the `ColorSpace` parameter.

```
rgb2lab([.2 .3 .4],'ColorSpace','adobe-rgb-1998')
```

```
ans = 1×3

   30.1783   -5.6902  -20.8223
```

Convert RGB color to L*a*b* Specifying Reference White

Use `rgb2lab` to convert an RGB color to L*a*b using the D50 reference white.

```
rgb2lab([.2 .3 .4],'WhitePoint','d50')
```

```
ans = 1×3
```

```
   31.3294   -4.0732  -18.1750
```

Convert RGB Image to L*a*b* and Display L* Component

Read RGB image into the workspace.

```
rgb = imread('peppers.png');
```

Convert the RGB image to the L*a*b* color space.

```
lab = rgb2lab(rgb);
```

Display the L* component of the L*a*b* image.

```
imshow(lab(:,:,1),[0 100])
```

Input Arguments

`RGB` — RGB color values

numeric array

RGB color values to convert, specified as a numeric array in one of these formats.

- *c*-by-3 colormap. Each row specifies one RGB color value.
- *m*-by-*n*-by-3 image
- *m*-by-*n*-by-3-by-*p* stack of images

Data Types: `single` | `double` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `rgb2lab([0.25 0.40 0.10],'WhitePoint','d50')`

`ColorSpace` — Color space of the input RGB values

`'srgb'` (default) | `'adobe-rgb-1998'` | `'linear-rgb'`

Color space of the input RGB values, specified as the comma-separated pair consisting of `'ColorSpace'` and one of `'srgb'`, `'adobe-rgb-1998'`, or `'linear-rgb'`. If you specify `'linear-rgb'`, then `rgb2lab` assumes the input RGB values are linearized sRGB values.

Data Types: `char`

`WhitePoint` — Reference white point

`'d65'` (default) | `'a'` | `'c'` | `'e'` | `'d50'` | `'d55'` | `'icc'` | 1-by-3 vector

Reference white point, specified as the comma-separated pair consisting of `'WhitePoint'` and a 1-by-3 vector or one of the CIE standard illuminants listed in the table.

Value	White Point
`'a'`	CIE standard illuminant A, `[1.0985, 1.0000, 0.3558]`. Simulates typical, domestic, tungsten-filament lighting with correlated color temperature of 2856 K.
`'c'`	CIE standard illuminant C, `[0.9807, 1.0000, 1.1822]`. Simulates average or north sky daylight with correlated color temperature of 6774 K. Deprecated by CIE.
`'e'`	Equal-energy radiator, `[1.000, 1.000, 1.000]`. Useful as a theoretical reference.
`'d50'`	CIE standard illuminant D50, `[0.9642, 1.0000, 0.8251]`. Simulates warm daylight at sunrise or sunset with correlated color temperature of 5003 K. Also known as horizon light.
`'d55'`	CIE standard illuminant D55, `[0.9568, 1.0000, 0.9214]`. Simulates mid-morning or mid-afternoon daylight with correlated color temperature of 5500 K.

Value	White Point
`'d65'`	CIE standard illuminant D65, `[0.9504, 1.0000, 1.0888]`. Simulates noon daylight with correlated color temperature of 6504 K.
`'icc'`	Profile Connection Space (PCS) illuminant used in ICC profiles. Approximation of `[0.9642, 1.000, 0.8249]` using fixed-point, signed, 32-bit numbers with 16 fractional bits. Actual value: `[31595,32768, 27030]/32768`.

Data Types: `single` | `double` | `char`

Output Arguments

`lab` — Converted L*a*b* color values

numeric array

Converted L*a*b* color values, returned as a numeric array of the same size as the input. The output type is `double` unless the input type is `single`, in which case the output type is also `single`.

Attribute	Description
*L**	Luminance or brightness of the image. Values are in the range [0, 100], where 0 specifies black and 100 specifies white. As *L** increases, colors become brighter.
*a**	Amount of red or green tones in the image. A large positive *a** value corresponds to red/magenta. A large negative *a** value corresponds to green. Although there is no single range for *a**, values commonly fall in the range [-100, 100] or [-128, 127).
*b**	Amount of yellow or blue tones in the image. A large positive *b** value corresponds to yellow. A large negative *b** value corresponds to blue. Although there is no single range for *b**, values commonly fall in the range [-100, 100] or [-128, 127).

Data Types: `double` | `single`

Tips

- If you specify the input RGB color space as `'linear-rgb'`, then `rgb2lab` assumes the input values are linearized sRGB values. If instead you want the input color space to be linearized Adobe RGB (1998), then you can use the `lin2rgb` function.

  For example, to convert linearized Adobe RGB (1998) image `RGBlinadobe` to the CIE 1976 L*a*b* color space, perform the conversion in two steps:

  ```
  RGBadobe = lin2rgb(RGBlinadobe,'ColorSpace','adobe-rgb-1998');
  LAB = rgb2lab(RGBadobe,'ColorSpace','adobe-rgb-1998');
  ```

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `rgb2lab` supports the generation of C code (requires MATLAB Coder). For more information, see "Code Generation for Image Processing".
- When generating code, all character vector input arguments must be compile-time constants.

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- When generating code, all character vector input arguments must be compile-time constants.

See Also

`rgb2xyz` | `lab2rgb` | `xyz2lab`

Topics
"Understanding Color Spaces and Color Space Conversion"
"Device-Independent Color Spaces"

Introduced in R2014b

rgb2lightness

Convert RGB color values to lightness values

Syntax

```
lightness = rgb2lightness(rgb)
```

Description

`lightness = rgb2lightness(rgb)` converts RGB color values to lightness values, excluding the color components. `lightness` is same as the L* component in the CIE 1976 L*a*b* color space.

Examples

Convert RGB Color to Lightness Component

Read RGB image into the workspace.

```
rgb = imread('peppers.png');
```

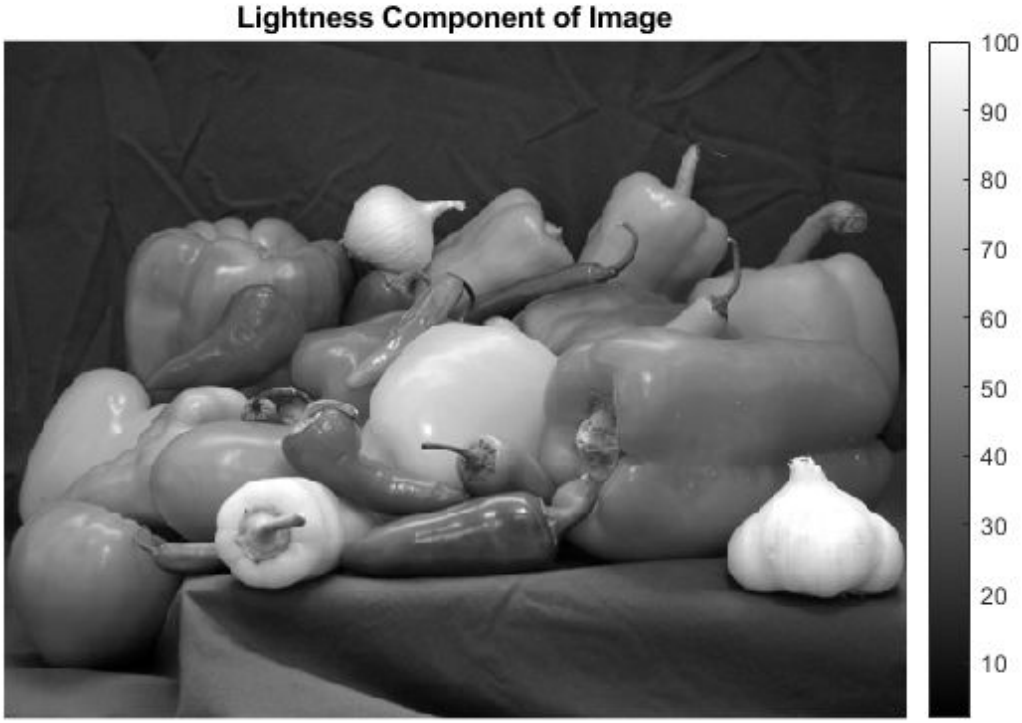
Convert the RGB color values to lightness component, excluding the color information.

```
lightness = rgb2lightness(rgb);
```

Display the RGB image and the derived lightness component of image.

```
figure
imshow(rgb)
title('Input RGB Image')
```


```
figure
imshow(lightness,[])
title('Lightness Component of Image')
colorbar
```



Input Arguments

`rgb` — RGB color values

m-by-*n*-by-3 image array

RGB color values, specified as an *m*-by-*n*-by-3 image array. The input `rgb` must be in sRGB color space with a reference white point of D65.

Data Types: `single` | `double` | `uint8` | `uint16`

Output Arguments

`lightness` — Converted lightness values

m-by-*n* image array

Converted lightness values, returned as an *m*-by-*n* image array. If the input data type is `double`, the output data type is `double`. Otherwise, the output data type is `single`.

Data Types: `single` | `double`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`rgb2lightness` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

See Also

`rgb2xyz` | `lab2rgb` | `xyz2lab` | `rgb2lab` | `lab2xyz` | `xyz2rgb`

Introduced in R2019a

rgb2lin

Linearize gamma-corrected RGB values

Syntax

```
B = rgb2lin(A)
B = rgb2lin(A,Name,Value)
```

Description

`B = rgb2lin(A)` undoes the gamma correction of the sRGB values in image `A` so that `B` contains linear RGB values.

`B = rgb2lin(A,Name,Value)` undoes gamma correction using name-value pairs to control additional options.

Examples

Linearize an sRGB Image

Open an image. The JPEG file format saves images in the gamma-corrected sRGB color space.

```
A = imread('foosball.jpg');
```

Display the image.

```
imshow(A)
title('Scene With sRGB Gamma Correction')
```

Undo the gamma correction and linearize the image by using the `rgb2lin` function. Optionally, specify the data type of the linearized values.

```
B = rgb2lin(A,'OutputType','double');
```

Display the linearized image. Shadows in the linearized image are darker than in the original image, as expected.

```
imshow(B)
title('Scene Without sRGB Gamma Correction')
```

Input Arguments

A — Gamma-corrected RGB color values

numeric array

Gamma-corrected RGB color values, specified as a numeric array in one of the following formats.

- *c*-by-3 colormap. Each row specifies one RGB color value.
- *m*-by-*n*-by-3 image
- *m*-by-*n*-by-3-by-*p* stack of images

Data Types: `single` | `double` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `B = lin2rgb(I,'ColorSpace','adobe-rgb-1998')` linearizes the gamma-corrected image, `I`, according to the Adobe RGB (1998) standard.

`ColorSpace` — Color space of the input image
`'srgb'` (default) | `'adobe-rgb-1998'`

Color space of the input image, specified as the comma-separated pair consisting of `'ColorSpace'` and `'srgb'` or `'adobe-rgb-1998'`.

Data Types: `char` | `string`

`OutputType` — Data type of output RGB values
`'double'` | `'single'` | `'uint8'` | `'uint16'`

Data type of the output RGB values, specified as the comma-separated pair consisting of `'OutputType'` and `'double'`, `'single'`, `'uint8'`, or `'uint16'`. By default, the output data type is the same as the data type of `A`.

Data Types: `char` | `string`

Output Arguments

`B` — Linearized RGB color values
numeric array

Linearized RGB color values, returned as a numeric array of the same size as the input `A`.

Algorithms

Linearization Using the sRGB Standard

sRGB tristimulus values are linearized using the following parametric curve:

$f(u) = -f(-u), \qquad u < 0$

$f(u) = c \cdot u, \qquad 0 \le u < d$

$f(u) = (a \cdot u + b)^{\gamma}, \qquad u \ge d,$

where u represents a color value with these parameters:

$a = 1/1.055$

$b = 0.055/1.055$

$c = 1/12.92$

$d = 0.04045$

$\gamma = 2.4$

Linearization Using the Adobe RGB (1998) Standard

Adobe RGB (1998) tristimulus values are linearized using a simple power function:

$v = u^{\gamma},$

with

γ = 2.19921875

References

[1] Ebner, Marc. "Gamma Correction." *Color Constancy*. Chichester, West Sussex: John Wiley & Sons, 2007.

[2] Adobe Systems Incorporated. "Inverting the color component transfer function." *Adobe RGB (1998) Color Image Encoding*. Section 4.3.5.2, May 2005, p.12.

See Also

`lin2rgb`

Introduced in R2017b

rgb2ntsc

Convert RGB color values to NTSC color space

Syntax

```
YIQ = rgb2ntsc(RGB)
```

Description

`YIQ = rgb2ntsc(RGB)` converts the red, green, and blue values of an RGB image to luminance (*Y*) and chrominance (*I* and *Q*) values of an NTSC image.

Examples

Convert Image from RGB to YIQ

This example shows how to convert an image from RGB to NTSC color space.

Read an RGB image into the workspace.

```
RGB = imread('board.tif');
```

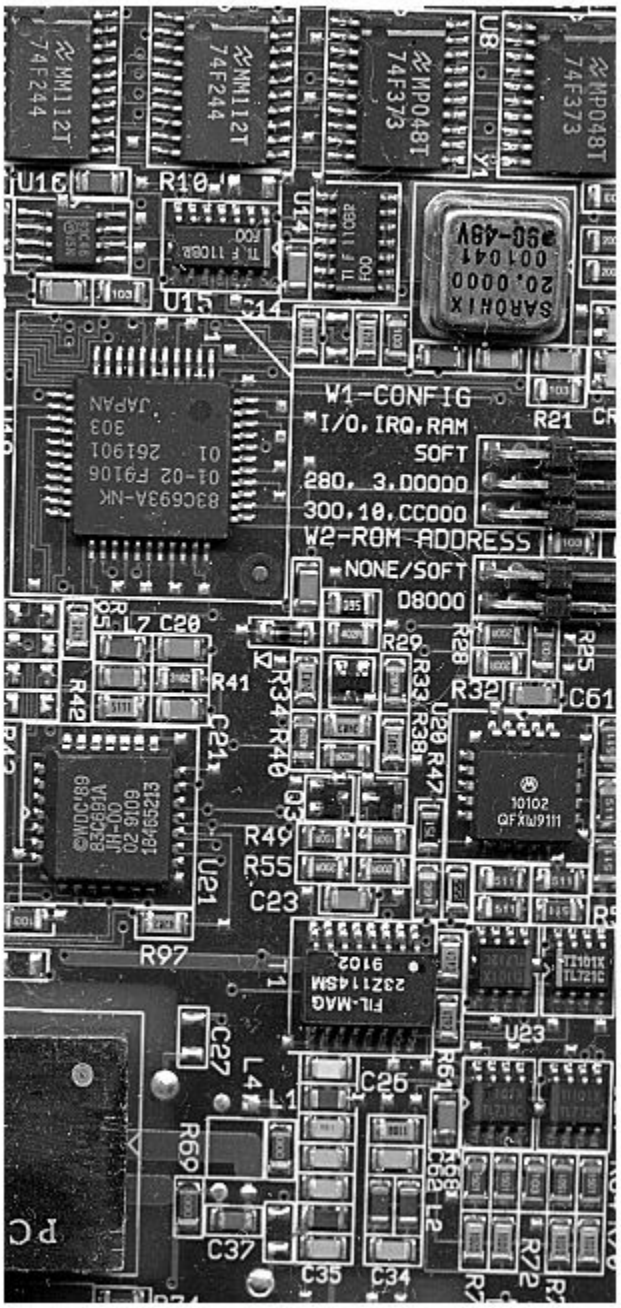
Convert the image to YIQ color space.

```
YIQ = rgb2ntsc(RGB);
```

Display the NTSC luminance value, represented by the first color channel in the YIQ image.

```
imshow(YIQ(:,:,1));
title('Luminance in YIQ Color Space');
```

Luminance in YIQ Color Space



Input Arguments

`RGB` — RGB color values

numeric array

RGB color values to convert, specified as a numeric array in one of these formats.

- *c*-by-3 colormap. Each row specifies one RGB color value.
- *m*-by-*n*-by-3 image

Data Types: `single` | `double` | `uint8` | `uint16` | `int16`

Output Arguments

`YIQ` — Converted YIQ color values

numeric array

Converted YIQ color values, returned as a numeric array of the same size as the input.

Attribute	Description
Y	Luma, or brightness of the image. Values are in the range [0, 1], where 0 specifies black and 1 specifies white. Colors increase in brightness as *Y* increases.
I	*In-phase*, which is approximately the amount of blue or orange tones in the image. *I* in the range [-0.5959, 0.5959], where negative numbers indicate blue tones and positive numbers indicate orange tones. As the magnitude of *I* increases, the saturation of the color increases.
Q	*Quadrature*, which is approximately the amount of green or purple tones in the image. *Q* in the range [-0.5229, 0.5229], where negative numbers indicate green tones and positive numbers indicate purple tones. As the magnitude of *Q* increases, the saturation of the color increases.

Data Types: `double`

Algorithms

In the NTSC color space, the luminance is the grayscale signal used to display pictures on monochrome (black and white) televisions. The other components carry the hue and saturation information. The value `0` corresponds to the absence of the component, while the value `1` corresponds to full saturation of the component.

`rgb2ntsc` defines the NTSC components using

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

See Also

`ntsc2rgb` | `rgb2ycbcr` | `rgb2lab` | `rgb2xyz` | `rgb2hsv`

Topics
“Understanding Color Spaces and Color Space Conversion”

Introduced before R2006a

rgb2xyz

Convert RGB to CIE 1931 XYZ

Syntax

```
XYZ = rgb2xyz(RGB)
XYZ = rgb2xyz(RGB,Name,Value)
```

Description

`XYZ = rgb2xyz(RGB)` converts the red, green, and blue values of an sRGB image to CIE 1931 XYZ values (2° observer).

`XYZ = rgb2xyz(RGB,Name,Value)` specifies additional conversion options, such as the color space of the RGB image, using one or more name-value pair arguments.

Examples

Convert RGB to XYZ

Convert images and color values from RGB to CIE 1931 XYZ color space.

Convert RGB Image to XYZ

Read an RGB image into the workspace.

```
RGB = imread('peppers.png');
```

Convert the image to XYZ color space.

```
XYZ = rgb2xyz(RGB);
```

Display the original image alongside the new image.

```
figure
imshowpair(RGB,XYZ,'montage');
title('Image in RGB Color Space (Left) and XYZ Color Space (Right)');
```

Convert RGB Color Value to XYZ

Convert the value of white from RGB to XYZ color space. In RGB, white is represented by the vector [1 1 1].

```
rgb2xyz([1 1 1])
```

```
ans = 1×3

    0.9505    1.0000    1.0888
```

Convert RGB Color to XYZ using D50 as Reference White

```
XYZ_D50 = rgb2xyz(RGB,'WhitePoint','d50');
```

Display the first output XYZ image alongside the XYZ image with D50 as reference white.

```
figure
imshowpair(XYZ,XYZ_D50,'montage');
title('XYZ Image, Without (Left) and With (Right) Reference White');
```

Convert Adobe RGB (1998) Color to XYZ

```
XYZ_Adobe = rgb2xyz(RGB,'ColorSpace','adobe-rgb-1998');
```

Display the XYZ images generated from the default RGB and the Adobe RGB (1998) color spaces.

```
figure
imshowpair(XYZ,XYZ_Adobe,'montage');
title(['XYZ Image, Starting From Default RGB (Left) and Adobe RGB ',...
  '(Right) Color Space']);
```

Input Arguments

RGB — RGB color values

numeric array

RGB color values to convert, specified as a numeric array in one of these formats.

- *c*-by-3 colormap. Each row specifies one RGB color value.
- *m*-by-*n*-by-3 image
- *m*-by-*n*-by-3-by-*p* stack of images

Data Types: `single` | `double` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `rgb2xyz([.2 .3 .4],'WhitePoint','d50')`

`ColorSpace` — Color space of the input RGB values

`'srgb'` (default) | `'adobe-rgb-1998'` | `'linear-rgb'`

Color space of the input RGB values, specified as the comma-separated pair consisting of `'ColorSpace'` and one of `'srgb'`, `'adobe-rgb-1998'`, or `'linear-rgb'`. If you specify `'linear-rgb'`, then `rgb2xyz` assumes the input RGB values are linearized sRGB values.

Data Types: `char`

`WhitePoint` — Reference white point

`'d65'` (default) | `'a'` | `'c'` | `'e'` | `'d50'` | `'d55'` | `'icc'` | 1-by-3 vector

Reference white point, specified as the comma-separated pair consisting of `'WhitePoint'` and a 1-by-3 vector or one of the CIE standard illuminants listed in the table.

Value	White Point
`'a'`	CIE standard illuminant A, `[1.0985, 1.0000, 0.3558]`. Simulates typical, domestic, tungsten-filament lighting with correlated color temperature of 2856 K.
`'c'`	CIE standard illuminant C, `[0.9807, 1.0000, 1.1822]`. Simulates average or north sky daylight with correlated color temperature of 6774 K. Deprecated by CIE.
`'e'`	Equal-energy radiator, `[1.000, 1.000, 1.000]`. Useful as a theoretical reference.
`'d50'`	CIE standard illuminant D50, `[0.9642, 1.0000, 0.8251]`. Simulates warm daylight at sunrise or sunset with correlated color temperature of 5003 K. Also known as horizon light.
`'d55'`	CIE standard illuminant D55, `[0.9568, 1.0000, 0.9214]`. Simulates mid-morning or mid-afternoon daylight with correlated color temperature of 5500 K.
`'d65'`	CIE standard illuminant D65, `[0.9504, 1.0000, 1.0888]`. Simulates noon daylight with correlated color temperature of 6504 K.
`'icc'`	Profile Connection Space (PCS) illuminant used in ICC profiles. Approximation of `[0.9642, 1.000, 0.8249]` using fixed-point, signed, 32-bit numbers with 16 fractional bits. Actual value: `[31595,32768, 27030]/32768`.

Data Types: `single` | `double` | `char`

Output Arguments

XYZ — Converted XYZ color values

numeric array

Converted XYZ color values, returned as a numeric array of the same size as the input. The output type is class `double` unless the input type is `single`, in which case the output type is also `single`.

Tips

- If you specify the input RGB color space as `'linear-rgb'`, then `rgb2xyz` assumes the input values are linearized sRGB values. If instead you want the input color space to be linearized Adobe RGB (1998), then you can use the `lin2rgb` function.

  For example, to convert linearized Adobe RGB (1998) image `RGBlinadobe` to CIE 1931 XYZ color space, perform the conversion in two steps:

  ```
  RGBadobe = lin2rgb(RGBlinadobe,'ColorSpace','adobe-rgb-1998');
  XYZ = rgb2xyz(RGBadobe,'ColorSpace','adobe-rgb-1998');
  ```

See Also

`xyz2rgb` | `rgb2lab` | `lab2xyz` | `lin2rgb` | `rgbwide2xyz` | `xyz2rgbwide`

Topics

“Understanding Color Spaces and Color Space Conversion”
“Device-Independent Color Spaces”

Introduced in R2014b

rgbwide2xyz

Convert wide-gamut RGB color values to CIE 1931 XYZ color values

Syntax

```
XYZ = rgbwide2xyz(RGB,BPS)
XYZ = rgbwide2xyz(RGB,BPS,Name,Value)
```

Description

`XYZ = rgbwide2xyz(RGB,BPS)` converts wide-gamut RGB values in the BT.2020 or BT.2100 color spaces to CIE 1931 XYZ color values. `BPS` specifies the number of bits required to represent each input channel.

`XYZ = rgbwide2xyz(RGB,BPS,Name,Value)` specifies options using one or more name-value pair arguments.

Examples

Convert Wide-Gamut RGB Values to CIE 1931 XYZ Values

Convert 10-bit or 12-bit wide-gamut RGB color values in the BT.2020 or BT.2100 color spaces to CIE 1931 XYZ color values.

Convert 10-bit BT.2020 RGB Green Value to XYZ

Create a wide-gamut RGB value for the color green.

```
rgbvalue = uint16([64 940 64]);
```

Convert the 10-bit BT.2020 RGB color value to an XYZ color value.

```
xyzvalue = rgbwide2xyz(rgbvalue,10);
```

Convert 12-bit BT.2100 RGB Blue Value to XYZ

Create a wide-gamut RGB color value for the color blue.

```
rgbvalue = uint16([64 64 940]);
```

Convert the 12-bit BT.2100 RGB value to an XYZ color value.

```
xyzvalue = rgbwide2xyz(rgbvalue, 12,'ColorSpace','BT.2100');
```

Convert 10-bit BT.2100 RGB White Value to XYZ Using HLG

Create a wide-gamut RGB value for the color white.

```
rgbvalue = uint16([940 940 940]);
```

Convert the 10-bit BT.2100 RGB color value to an XYZ color value, using the Hybrid Log Gamma (HLG) transfer function.

```
xyzvalue = rgbwide2xyz(rgbvalue,10,'ColorSpace','BT.2100','LinearizationFcn','HLG');
```

Input Arguments

RGB — Wide-gamut RGB color values

p-by-*3* | *m*-by-*n*-by-3 numeric array | *m*-by-*n*-by-3-by-*f* numeric array

Wide-gamut RGB color values, specified as one of the following:

- *p*-by-*3* numeric matrix of color values (one color per row)
- *m*-by-*n*-by-3 numeric array representing an image
- *m*-by-*n*-by-3-by-*f* numeric array representing a stack of images

The following table shows the data range for wide-gamut, integer color values for 10- and 12-bit data. The minimum value in the range maps to black, and the maximum value in the range maps to white. Only pixels with RGB values within the supported data range for wide-gamut values are guaranteed to be mapped to realizable colors.

Data Type	Full Data Range	Data Range for Wide-Gamut RGB
10-bit	[0, 1023]	[64, 940]
12-bit	[0, 4095]	[256, 3760]

Data Types: `uint16`

BPS — Bits per sample for each channel of input image

`10` | `12`

Bits per sample for each channel of the input wide-gamut RGB image, specified as `10` or `12`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `xyzvalue = rgbwide2xyz(rgbvalue,12,'ColorSpace','BT.2100');`

ColorSpace — Color space of wide-gamut RGB values

`'BT.2020'` (default) | `'BT.2100'`

Color space of the wide-gamut RGB values, specified as the comma-separated pair consisting of `'ColorSpace'` and the value `'BT.2020'` or `'BT.2100'`.

Data Types: `char` | `string`

WhitePoint — Reference white point

`'d65'` (default) | `'a'` | `'c'` | `'d50'` | `'d55'` | `'icc'` | `'e'` | 3-element row vector

Reference white point, specified as the comma-separated pair consisting of `'WhitePoint'` and a 3-element row vector or one of the CIE standard illuminants listed in the table.

Value	White Point
`'a'`	CIE standard illuminant A, `[1.0985, 1.0000, 0.3558]`. Simulates typical, domestic, tungsten-filament lighting with correlated color temperature of 2856 K.
`'c'`	CIE standard illuminant C, `[0.9807, 1.0000, 1.1822]`. Simulates average or north sky daylight with correlated color temperature of 6774 K. Deprecated by CIE.
`'e'`	Equal-energy radiator, `[1.000, 1.000, 1.000]`. Useful as a theoretical reference.
`'d50'`	CIE standard illuminant D50, `[0.9642, 1.0000, 0.8251]`. Simulates warm daylight at sunrise or sunset with correlated color temperature of 5003 K. Also known as horizon light.
`'d55'`	CIE standard illuminant D55, `[0.9568, 1.0000, 0.9214]`. Simulates mid-morning or mid-afternoon daylight with correlated color temperature of 5500 K.
`'d65'`	CIE standard illuminant D65, `[0.9504, 1.0000, 1.0888]`. Simulates noon daylight with correlated color temperature of 6504 K.
`'icc'`	Profile Connection Space (PCS) illuminant used in ICC profiles. Approximation of `[0.9642, 1.000, 0.8249]` using fixed-point, signed, 32-bit numbers with 16 fractional bits. Actual value: `[31595,32768, 27030]/32768`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `string`

`LinearizationFcn` — Transfer function for transformation

`'PQ'` (default) | `'HLG'`

Transfer function for transformation, specified as the comma-separated pair consisting of `'LinearizationFcn'` and either of these values:

Value	Description
`'PQ'`	Perceptual Quantization
`'HLG'`	Hybrid Log Gamma

Data Types: `char` | `string`

Output Arguments

XYZ — Values in CIE 1931 XYZ color space

numeric array

Values in the CIE 1931 XYZ color space, returned as a numeric array of the same size as the RGB input color values.

Data Types: `double`

References

[1] Rec. ITU-R BT.2020-2 (10/2015). "Parameter values for ultra-high definition television systems for production and international programme exchange." *International Telecommunication Union; Broadcasting service (television)*. https://www.itu.int/rec/R-REC-BT.2020.

[2] Rec. ITU-R BT.2100-2 (07/2018). "Image parameter values for dynamic range television for use in production and international programme exchange." *International Telecommunication Union; Broadcasting service (television)*. https://www.itu.int/rec/R-REC-BT.2100.

[3] Rec. ITU-R BT.2390-7 (07/2019). "High dynamic range television for production and international programme exchange." *International Telecommunication Union; Broadcasting service (television)*. https://www.itu.int/pub/R-REP-BT.2390.

See Also

`rgb2xyz` | `xyz2rgb` | `xyz2rgbwide` | `rgbwide2ycbcr`

Introduced in R2020b

rgb2ycbcr

Convert RGB color values to YCbCr color space

Syntax

```
YCBCR = rgb2ycbcr(RGB)
```

Description

`YCBCR = rgb2ycbcr(RGB)` converts the red, green, and blue values of an RGB image to luminance (*Y*) and chrominance (*Cb* and *Cr*) values of a YCbCr image.

Examples

Convert RGB to YCbCr

Convert Image from RGB to YCbCr

Read an RGB image into the workspace.

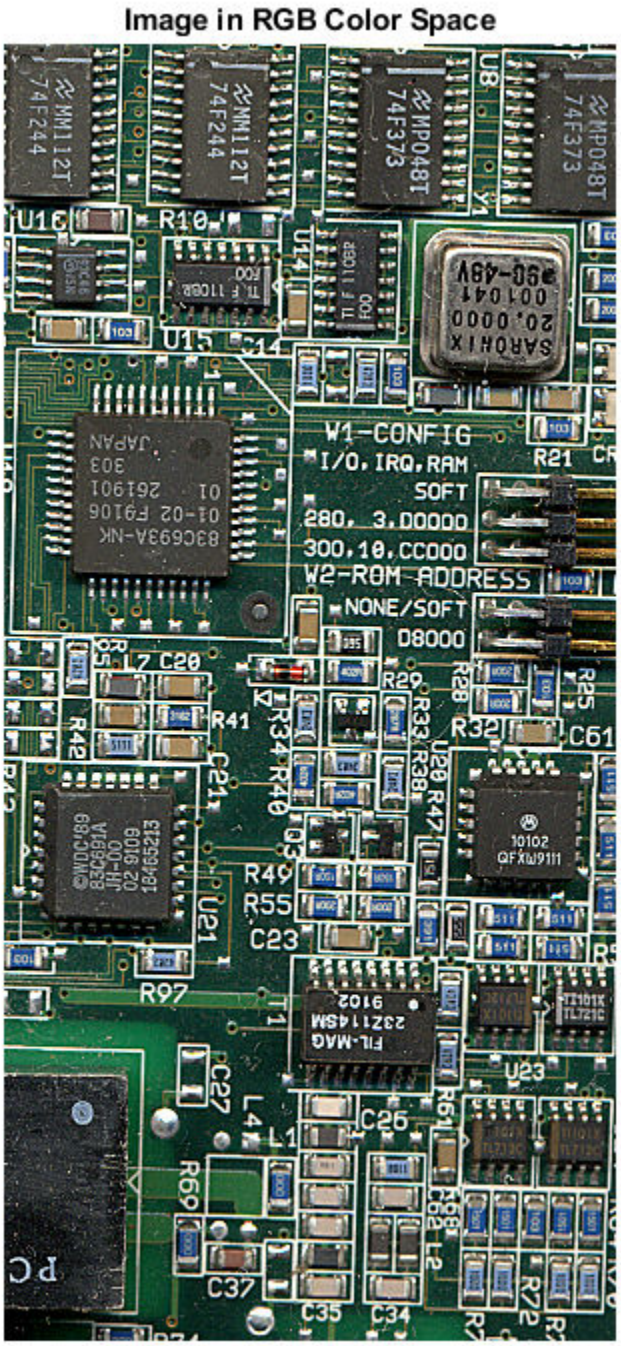
```
RGB = imread('board.tif');
```

Convert the image to YCbCr.

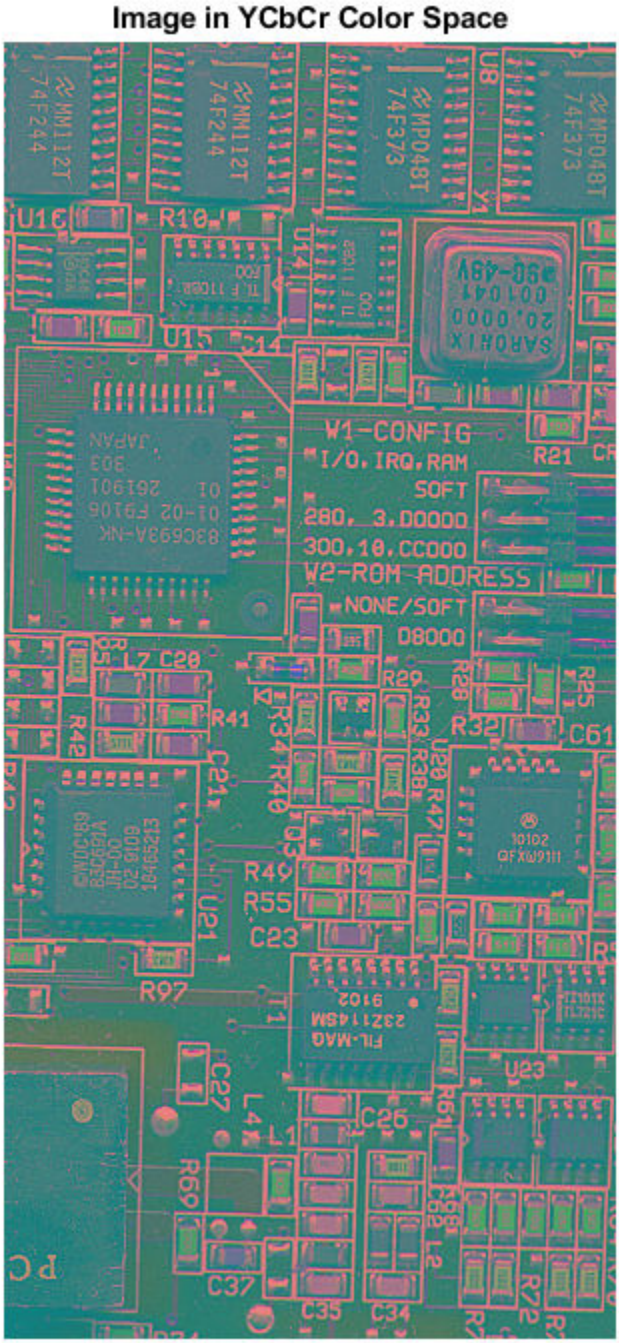
```
YCBCR = rgb2ycbcr(RGB);
```

Display the original image and the new image

```
figure
imshow(RGB);
title('Image in RGB Color Space');
```




```
figure
imshow(YCBCR);
title('Image in YCbCr Color Space');
```



Convert Colormap from RGB to YCbCr.

Load an indexed image into the workspace. The colormap is in RGB colorspace.

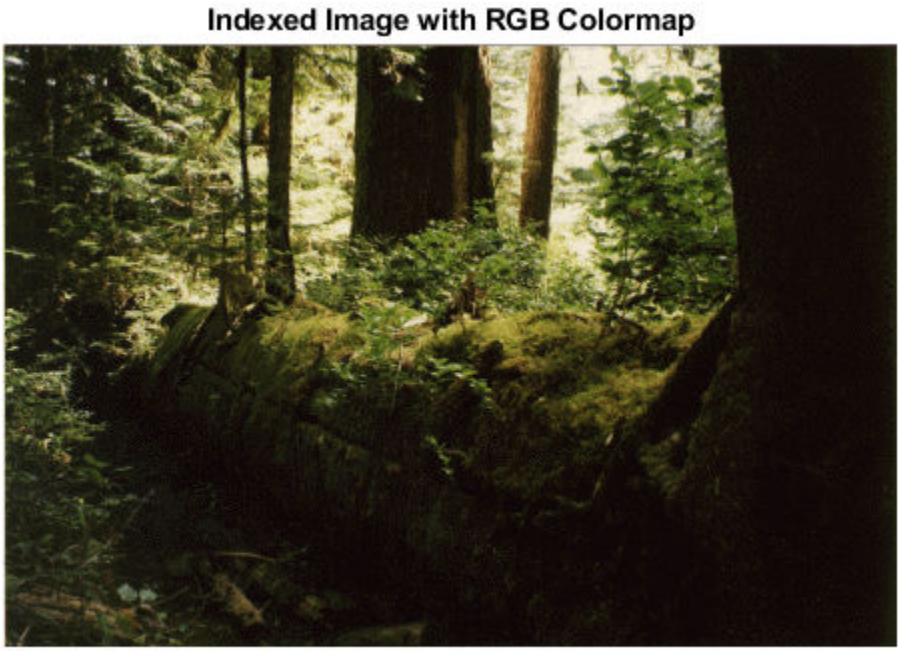
```
[I,map] = imread('forest.tif');
```

Convert the colormap to YCbCr.

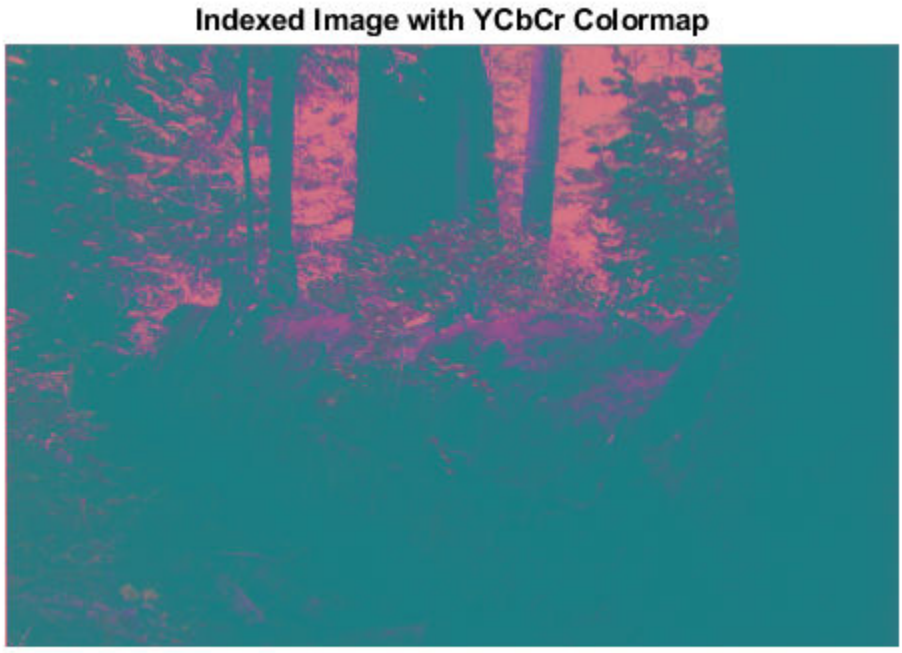
```
newmap = rgb2ycbcr(map);
```

Display the grayscale image with the original map and with the new map.

```
figure
imshow(I,map)
title('Indexed Image with RGB Colormap');
```




```
figure
imshow(I,newmap)
title('Indexed Image with YCbCr Colormap');
```



Input Arguments

RGB — RGB color values

numeric array

RGB color values to convert, specified as a numeric array in one of these formats.

- *c*-by-3 colormap. Each row specifies one RGB color value.
- *m*-by-*n*-by-3 image

Data Types: `single` | `double` | `uint8` | `uint16`

Output Arguments

YCBCR — Converted YCbCr color values

numeric array

Converted YCbCr color values, returned as a numeric array of the same size as the input.

- If the input is `double` or `single`, then `Y` is in the range [16/255, 235/255] and `Cb` and `Cr` are in the range [16/255, 240/255].
- If the input is `uint8`, then `Y` is in the range [16, 235] and `Cb` and `Cr` are in the range [16, 240].
- If the input is `uint16`, then `Y` is in the range [4112, 60395] and `Cb` and `Cr` are in the range [4112, 61680].

References

[1] Poynton, C. A.*A Technical Introduction to Digital Video*, John Wiley & Sons, Inc., 1996, p. 175.

[2] Rec. ITU-R BT.601-5, *Studio Encoding Parameters of Digital Television for Standard 4:3 and Widescreen 16:9 Aspect Ratios*, (1982-1986-1990-1992-1994-1995), Section 3.5.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `rgb2ycbcr` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `rgb2ycbcr` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`rgb2lab` | `rgb2xyz` | `rgb2ntsc` | `ycbcr2rgb` | `rgbwide2ycbcr` | `ycbcr2rgbwide`

Topics

“Understanding Color Spaces and Color Space Conversion”

Introduced before R2006a

rgbwide2ycbcr

Convert wide-gamut RGB color values to YCbCr color values

Syntax

```
ycbcr = rgbwide2ycbcr(RGB,BPS)
```

Description

`ycbcr = rgbwide2ycbcr(RGB,BPS)` converts wide-gamut RGB values in the BT.2020 or BT.2100 color spaces into the nonconstant luminance YCbCr color space. `BPS` specifies the number of bits required to represent each channel of the input values.

Examples

Convert Wide-Gamut RGB to YCbCr

Convert 10-bit or 12-bit wide-gamut RGB color values in the BT.2020 or BT.2100 color space to the YCbCr color space.

Convert 10-bit BT.2020 or BT.2100 Wide-Gamut RGB White Color Value to YCbCr

Create a 10-bit wide-gamut RGB value for the color white.

```
rgblist = uint16([940 940 940]);
```

Convert the wide-gamut white color value to a YCbCr color value.

```
ycbcrlist = rgbwide2ycbcr(rgblist,10)
```

```
ycbcrlist = 1x3 uint16 row vector

   940   512   512
```

Convert 12-bit BT.2020 or BT.2100 Wide-Gamut RGB Image to YCbCr

Simulate a wide-gamut RGB image. Read a normal RGB image into the workspace, convert the image to the XYZ color space, then convert the resulting image to the wide-gamut RGB color space.

```
RGBWide = imread('peppers.png');
XYZ = rgb2xyz(RGBWide);
RGBWide = xyz2rgbwide(XYZ,12);
```

Convert the wide-gamut RGB image to the YCbCr color space.

```
YCBCR = rgbwide2ycbcr(RGBWide,12);
```

Input Arguments

RGB — Wide-gamut RGB color values

p-by-*3* numeric array | *m*-by-*n*-by-3 numeric array

Wide-gamut RGB color values, specified as one of these options:

- *p*-by-*3* numeric matrix of color values (one color per row)
- *m*-by-*n*-by-3 numeric array representing an image

This table shows the data range for wide-gamut, integer color values for 10- and 12-bit data. The minimum value in each range maps to black, and the maximum value in each range maps to white. The `rgbwide2ycbcr` function maps only pixels with RGB values within the supported data range to valid YCbCr values.

Data Type	Full Data Range	Data Range for Wide-Gamut RGB
10-bit	[0, 1023]	[64, 940]
12-bit	[0, 4095]	[256, 3760]

Data Types: `uint16`

BPS — Bits per sample for each channel of input image

`10` | `12`

Bits per sample for each channel of the input wide-gamut RGB image, specified as `10` or `12`.

Output Arguments

ycbcr — YCbCr color values

numeric array

YCbCr color values, returned as a numeric array of the same size as the input `RGB` color values.

Data Types: `uint16`

Tips

- This function does not support the full data range of 10-bit and 12-bit RGB values, [0, 1023] and [0, 4095] respectively. The table shows the data ranges of the YCbCr values for the BT.2020 and BT.2100 color spaces.

Component	10-bit	12-bit
Y	`[64, 940]`	`[256, 3760]`
Cb, Cr	`[64, 960]`	`[256, 3840]`

References

[1] Rec. ITU-R BT.2020-2 (10/2015). "Parameter values for ultra-high definition television systems for production and international programme exchange." *International Telecommunication Union; Broadcasting service (television)*. https://www.itu.int/rec/R-REC-BT.2020.

[2] Rec. ITU-R BT.2100-2 (07/2018). "Image parameter values for dynamic range television for use in production and international programme exchange." *International Telecommunication Union; Broadcasting service (television)*. https://www.itu.int/rec/R-REC-BT.2100.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`rgbwide2ycbcr` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

`rgb2ycbcr` | `ycbcr2rgbwide` | `ycbcr2rgb` | `rgbwide2xyz`

Introduced in R2020b

rigid2d

2-D rigid geometric transformation

Description

A `rigid2d` object stores information about a 2-D rigid geometric transformation and enables forward and inverse transformations.

Creation

Syntax

```
tform = rigid2d
tform = rigid2d(t)
tform = rigid2d(rot,trans)
```

Description

`tform = rigid2d` creates a default `rigid2d` object that corresponds to an identity transformation.

`tform = rigid2d(t)` creates a `rigid2d` object based on a specified forward rigid transformation matrix, `t`. The `t` input sets the `T` property.

`tform = rigid2d(rot,trans)` creates a `rigid2d` object based on the rotation, `rot`, and translation, `trans`, components of the transformation. `rot` sets the `Rotation` property. `trans` sets the `Translation` property.

Properties

T — Forward rigid transformation

3-by-3 identity matrix (default) | 3-by-3 numeric matrix

Forward rigid transformation, specified as a 3-by-3 numeric matrix.

Data Types: `single` | `double`

Dimensionality — Dimensionality of geometric transformation

2 (default)

This property is read-only.

Dimensionality of the geometric transformation, specified as the number 2.

Rotation — Rotation component of transformation

2-by-2 identity matrix (default) | 2-by-2 numeric matrix

Rotation component of the transformation, specified as a 2-by-2 numeric matrix.

Data Types: `single` | `double`

`Translation` — Translation component of transformation
`[0 0]` (default) | 2-element numeric row vector

Translation component of the transformation, specified as a 2-element numeric row vector.

Data Types: `single` | `double`

Object Functions

invert	Invert geometric transformation
isTranslation	Determine if transformation is pure translation
outputLimits	Find output spatial limits given input spatial limits
transformPointsForward	Apply forward geometric transformation
transformPointsInverse	Apply inverse geometric transformation

Examples

Create Rigid 2-D Object with Defined Translation and Rotation

Specify an angle of rotation in degrees and create a 2-by-2 rotation matrix.

```
theta = 30;
rot = [ cosd(theta) sind(theta); ...
       -sind(theta) cosd(theta)];
```

Specify the amount of horizontal and vertical translation, respectively.

```
trans = [2 3];
```

Create a `rigid2d` object that performs the rotation and translation.

```
tform = rigid2d(rot,trans)
```

```
tform =
  rigid2d with properties:

       Rotation: [2x2 double]
    Translation: [2 3]
```

See Also

`affine2d` | `rigid3d` | `projective2d` | `geometricTransform2d` | `imwarp`

Topics
“2-D and 3-D Geometric Transformation Process Overview”
“Matrix Representation of Geometric Transformations”

Introduced in R2020b

rigid3d

3-D rigid geometric transformation

Description

A `rigid3d` object stores information about a 3-D rigid geometric transformation and enables forward and inverse transformations.

Creation

Syntax

```
tform = rigid3d
tform = rigid3d(t)
tform = rigid3d(rot,trans)
```

Description

`tform = rigid3d` creates a default `rigid3d` object that corresponds to an identity transformation.

`tform = rigid3d(t)` creates a `rigid3d` object based on a specified forward rigid transformation matrix, `t`. The `t` input sets the `T` property.

`tform = rigid3d(rot,trans)` creates a `rigid3d` object based on the rotation, `rot`, and translation, `trans`, components of the transformation. `rot` sets the `Rotation` property. `trans` sets the `Translation` property.

Properties

T — Forward rigid transformation

4-by-4 identity matrix (default) | 4-by-4 numeric matrix

Forward rigid transformation, specified as a 4-by-4 numeric matrix. This matrix must be a homogeneous transformation matrix that satisfies the post-multiply convention given by:

$$[x\ y\ z\ 1] = [u\ v\ w\ 1] * T$$

`T` has the form

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & 0; \ \ldots \\ r_{21} & r_{22} & r_{23} & 0; \ \ldots \\ r_{31} & r_{32} & r_{33} & 0; \ \ldots \\ t_x & t_y & t_z & 1]; \end{bmatrix}$$

Data Types: `single` | `double`

`Dimensionality` — Dimensionality of geometric transformation
3 (default) | positive integer

This property is read-only.

Dimensionality of the geometric transformation, specified as a positive integer.

`Rotation` — Rotation component of transformation
3-by-3 identity matrix (default) | 3-by-3 numeric matrix

Rotation component of the transformation, specified as a 3-by-3 numeric matrix. This rotation matrix satisfies the post-multiply convention given by

$$[x\ y\ z] = [u\ v\ w] * R$$

Data Types: `single` | `double`

`Translation` — Translation component of transformation
`[0 0 0]` (default) | 3-element numeric row vector

Translation component of the transformation, specified as a 3-element numeric row vector. This translation vector satisfies the convention given by

$$[x\ y\ z] = [u\ v\ w] + t$$

Data Types: `single` | `double`

Object Functions

invert	Invert geometric transformation
outputLimits	Find output spatial limits given input spatial limits
transformPointsForward	Apply forward geometric transformation
transformPointsInverse	Apply inverse geometric transformation

Examples

Create Rigid 3-D Object with Defined Translation and Rotation

Specify an angle of rotation in degrees and create a 3-by-3 rotation matrix.

```
theta = 30;
rot = [ cosd(theta) sind(theta) 0; ...
       -sind(theta) cosd(theta) 0; ...
       0 0 1];
```

Specify the amount of horizontal, vertical, and depthwise translation, respectively.

```
trans = [2 3 4];
```

Create a `rigid3d` object that performs the rotation and translation.

```
tform = rigid3d(rot,trans)
```

```
tform =
  rigid3d with properties:
```

```
    Rotation: [3x3 double]
Translation: [2 3 4]
```

See Also

`rigid2d` | `affine3d` | `geometricTransform3d` | `imwarp`

Topics

“2-D and 3-D Geometric Transformation Process Overview”
“Matrix Representation of Geometric Transformations”

Introduced in R2020a

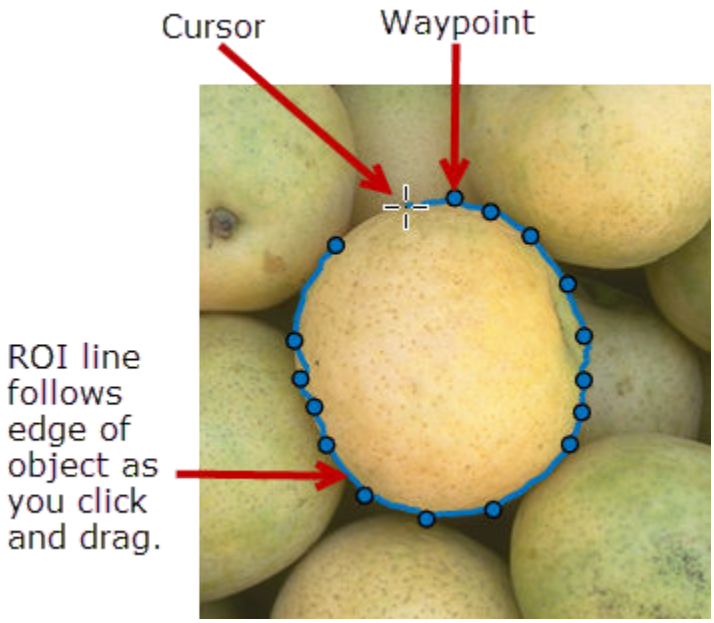
AssistedFreehand

Assisted freehand region of interest

Description

An `AssistedFreehand` object specifies the shape and position of a hand-drawn region-of-interest (ROI), where the line drawn automatically follows edges in the underlying image. You can customize the appearance and interactive behavior of the ROI.

For more information about using this ROI, including keyboard shortcuts, see “Tips” on page 1-2779.



Creation

There are two ways to create an `AssistedFreehand` object. For more information, see “Create ROI Shapes”.

- Use the `drawassisted` function. Use this function when you want to create the ROI and set the appearance in a single command. You can specify the shape and position of the ROI interactively by drawing the ROI over an image using the mouse.
- Use the `images.roi.AssistedFreehand` function described here. Use this function when you want to specify the appearance and behavior of the ROI before you specify the shape and position of the ROI. After creating the object, you can specify the shape and position interactively by using the `draw` function.

Syntax

```
roi = images.roi.AssistedFreehand
```

```
roi = images.roi.AssistedFreehand(ax)
roi = images.roi.AssistedFreehand( ___ ,Name,Value)
```

Description

`roi = images.roi.AssistedFreehand` creates an `AssistedFreehand` object with default properties.

`roi = images.roi.AssistedFreehand(ax)` creates the ROI on the axes specified by `ax`.

`roi = images.roi.AssistedFreehand( ___ ,Name,Value)` sets properties on page 1-2769 of the ROI using name-value arguments. You can specify multiple name-value arguments. Enclose each property name in single quotes.

Example: `images.roi.AssistedFreehand('Color','y')` creates a yellow colored `AssistedFreehand` object.

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Properties

Closed — Close ROI

`true` or `1` (default) | `false` or `0`

Close the ROI, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `true`, the `AssistedFreehand` object closes the ROI by connecting the last point drawn to the first point drawn.

Color — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	

Color Name	Short Name	RGB Triplet	Appearance
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0.4470 0.7410]

ContextMenu — Context menu
ContextMenu object

Context menu that displays when you right-click the ROI, specified as a ContextMenu object. You can create a custom context menu by using the uicontextmenu function and then configuring context menu properties.

Deletable — Context menu provides option to delete the ROI
true or 1 (default) | false or 0

Context menu provides an option to delete the ROI, specified as a numeric or logical 1 (true) or 0 (false). When the value is true, you can delete the ROI interactively using the context menu. When the value is false, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the delete function.

DrawingArea — Area of axes in which you can interactively place ROI
'auto' (default) | 'unlimited' | [x,y,w,h]

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
'auto'	The drawing area is the current axes limits (default).

Value	Description
`'unlimited'`	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
`[x,y,w,h]`	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

`FaceAlpha` — Transparency of ROI face
`0.2` (default) | number in the range [0, 1]

Transparency of the ROI face, specified as a number in the range [0, 1]. When the value is `1`, the ROI face is completely opaque. When the value is `0`, the ROI face is completely transparent.

`FaceSelectable` — ROI face can capture clicks
`true` or `1` (default) | `false` or `0`

ROI face can capture clicks, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `true`, the ROI face captures mouse clicks. When `false`, the ROI face does not capture mouse clicks.

`HandleVisibility` — Visibility of ROI handle in `Children` property of parent
`'on'` (default) | `'off'` | `'callback'`

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`Image` — Image on which to draw ROI
handle to `Image` object

Image on which to draw ROI, specified as a handle to an `Image` object.

`InteractionsAllowed` — Interactivity of ROI
`'all'` (default) | `'none'` | `'reshape'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable (default).
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area but not reshaped.

Value	Description
`'reshape'`	The ROI can be reshaped but not translated.

`Label` — ROI label
`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background
1 (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color
`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'LabelTextColor','r'`

Example: `'LabelTextColor','green'`

Example: `'LabelTextColor',[0 0.4470 0.7410]`

`LabelVisible` — Label visibility

`'on'` (default) | `'off'` | `'hover'`

Label visibility, specified as one of these values.

Value	Description
`'on'`	Label is visible when the ROI is visible.
`'hover'`	Label is visible only when the mouse is hovering over the ROI.
`'off'`	Label is not visible.

`LineWidth` — Width of ROI border

positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

`MarkerSize` — Marker size

positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

`Parent` — ROI parent

`Axes` object | `UIAxes` object

ROI parent, specified as an `Axes` or `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

`Position` — Position of ROI

`[]` (default) | *n*-by-2 numeric matrix

Position of the ROI, specified as an *n*-by-2 numeric matrix where *n* is the number of vertices or points defining the ROI. Each row represents the [*x* *y*] coordinates of a vertex or point. The `AssistedFreehand` object generates these points as you draw the ROI shape interactively. To work with fewer points, use the `reduce` function.

`Selected` — Selection state of ROI

`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`). You can also set this property interactively. For example, clicking on the ROI selects the ROI and sets this property to `true`. Similarly, pressing the **Ctrl** key and clicking the ROI deselects the ROI and sets the value of this property to `false`.

`SelectedColor` — Color of ROI when selected

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'SelectedColor','r'`

Example: `'SelectedColor','green'`

Example: `'SelectedColor',[0 0.4470 0.7410]`

`Smoothing` — Smoothing applied to edge of ROI

1 (default) | nonnegative number

Smoothing applied to the edge of ROI after interactive placement, specified as a nonnegative scalar. The `AssistedFreehand` object filters the *x* and *y* coordinates of the ROI using a Gaussian smoothing kernel with a default standard deviation of 1. The size of the Gaussian filter is `2*ceil(2*Smoothing)+1`.

You can see the smoothing effect only after completing the drawing. Changing the value of `Smoothing` after completing the drawing has no effect on the ROI.

`StripeColor` — Color of ROI stripe

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

`Tag` — Tag to associate with ROI

`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI

any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `AssistedFreehand` object does not use this data.

`Visible` — ROI visibility
`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

`Waypoints` — Position point is waypoint
`[ ]` (default) | logical vector

Position point is waypoint, specified as a logical vector of the same length as the `Position` property. Elements in `Waypoints` with the value `true` identify points in the `Position` matrix that are waypoints. By default, the `AssistedFreehand` object generates all the points that define the ROI and only makes points at locations of increased curvature into waypoints. You can turn all the points, or some subset of points, into waypoints by using code similar to `roi.Waypoints(1:4:end) = true;`.

Waypoints appear as circular shapes on the ROI edge. You can use waypoints to reshape the ROI by clicking and dragging the waypoint with the mouse. Moving waypoints modifies the freehand-drawn region between the waypoint that you clicked and the adjacent waypoints.

Object Functions

addlistener	Create event listener bound to event source
beginDrawingFromPoint	Begin drawing ROI from specified point
bringToFront	Bring ROI to front of Axes stacking order
createMask	Create binary mask image from ROI
draw	Begin drawing ROI interactively
inROI	Query if points are located in ROI
reduce	Reduce density of points in ROI
wait	Block MATLAB command line until ROI operation is finished

Examples

Create Freehand ROI Using Assisted Freehand

Read an image into the workspace and display it.

```
figure
imshow(imread('baby.jpg'))
```

Create an `AssistedFreehand` object. By default, the class creates the ROI on the current axes. Note that the axes must contain an image.

```
roi = images.roi.AssistedFreehand;
```

Call the `draw` function, specifying the `AssistedFreehand` object as an argument. The pointer changes to a cross-hair shape when you move it over the image. You can being drawing the ROI. Note how, as you move the pointer, the line you draw follows the edges in the underlying image. Click to add vertices along the edge as you draw.

```
draw(roi);
```

Set Up Listeners for AssistedFreehand ROI Events

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Display the image. Use the `imshow` syntax that returns the image object displayed.

```
img = imshow(I);
```

Create an `AssistedFreehand` ROI on the image. Call the `draw` object function to enable interactive drawing of the ROI shape. Note how the ROI line automatically follow edges in the underlying image.

```
roi = images.roi.AssistedFreehand(img);
draw(roi)
```

Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays at the command line the previous position and the current position of the ROI.

```
function allevents(src,evt)
    evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving current position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved current position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

More About

Events

To receive notification from the ROI when certain events happen, set up listeners for these events. You can specify a callback function that executes when one of these events occurs. When the ROI notifies your application through the listener, it returns data specific to the event. Look at the event class for the specific event to see what is returned.

For an example, see “Set Up Listeners for AssistedFreehand ROI Events” on page 1-2777.

Event Name	Trigger	Event Data	Event Attributes

`DeletingROI`	ROI is about to be interactively deleted.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingStarted`	ROI is about to be interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingFinished`	ROI has been interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`MovingROI`	ROI shape or location is being interactively changed.	`images.roi.ROIMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

Event Name	Trigger	Event Data	Event Attributes

`ROIMoved`	ROI shape or location has been interactively changed.	`images.roi.ROIMovingEventData`	`NotifyAccess`: `private`
			
			`ListenAccess`: `public`

`ROIClicked`	ROI has been clicked.	`images.roi.ROIClickedEventData`	`NotifyAccess`: `private`
			
			`ListenAccess`: `public`

`AddingWaypoint`	A waypoint is about to be interactively added to the ROI.	`event.EventData`	`NotifyAccess`: `private`
			
			`ListenAccess`: `public`

`WaypointAdded`	A waypoint has been interactively added to the ROI.	`event.EventData`	`NotifyAccess`: `private`
			
			`ListenAccess`: `public`

`RemovingWaypoint`	A waypoint is about to be interactively removed from the ROI.	`event.EventData`	`NotifyAccess`: `private`
			
			`ListenAccess`: `public`

`WaypointRemoved`	A waypoint has been interactively removed from the ROI.	`event.EventData`	`NotifyAccess`: `private`
			
			`ListenAccess`: `public`

Tips

- To draw the ROI interactively using the `draw` or `drawassisted` function, position the pointer on the image, click and release to place the first vertex (waypoint). Then move the pointer to draw a line. As you draw, the line automatically follows the edges of objects in the image. Double-click to finish the ROI.
- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.

Behavior	Keyboard shortcut
Finish drawing (close) the ROI.	Double-click, which adds a point at the pointer position and draws a line connecting this point to the first point drawn, closing the ROI. Right-click, which draws a line connecting the last point to the first point drawn. Position the pointer over the first point and click. Press **Enter**, which draws a line connecting the last point to the first point drawn.
Resize (reshape) the ROI.	Position pointer over a waypoint and then click and drag. No assistance (snapping to edges) is available in this mode.
Add a waypoint.	Position the pointer on an edge of the ROI, right-click, and select **Add Waypoint**. You can also position the pointer on an edge of the ROI and double-click.
Remove a waypoint.	Position the pointer on a waypoint, right-click, and select **Remove Waypoint**.
Move the ROI.	Position the pointer over the ROI. The pointer changes to the fleur shape. Click and drag to move the ROI.
Delete an ROI.	Position the pointer on the ROI (not on a vertex), right-click, and select **Delete Freehand** from the context menu. You can also delete the ROI programmatically using the `delete` function.

- For information about using an ROI in an app created with App Designer, see “Using ROIs in Apps Created with App Designer”.

Compatibility Considerations

`UIContextMenu` property is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` property to assign a context menu to a graphics object or UI component is not recommended. Use the `ContextMenu` property instead. The property values are the same.

There are no plans to remove support for the `UIContextMenu` property at this time. However, the `UIContextMenu` property no longer appears in the list returned by calling the `get` function on a graphics object or UI component.

See Also

`drawassisted` | `Freehand` | `Polygon` | `Polyline`

Topics
“Create ROI Shapes”

Introduced in R2018b

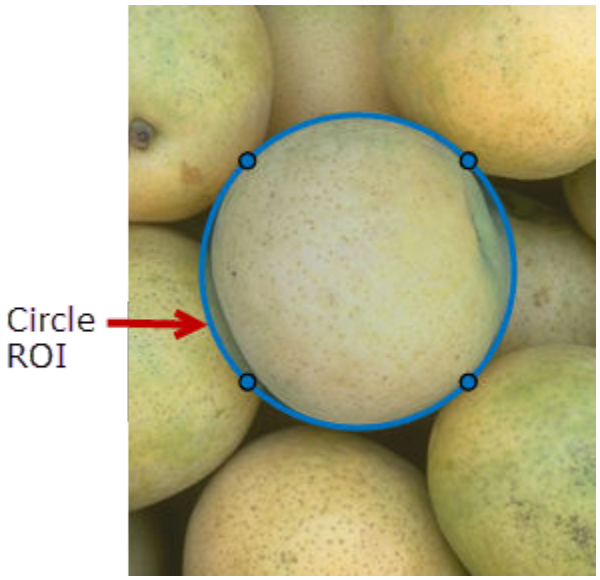
Circle

Circular region of interest

Description

A `Circle` object specifies the size and position of a circular region-of-interest (ROI). You can customize the appearance and interactive behavior of the ROI.

For more information about using this ROI, including keyboard shortcuts, see “Tips” on page 1-2794.



Creation

There are two ways to create a `Circle` object. For more information, see “Create ROI Shapes”.

- Use the `drawcircle` function. Use this function when you want to create the ROI and set the appearance in a single command. You can specify the size and position of the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments.
- Use the `images.roi.Circle` function described here. Use this function when you want to specify the appearance and behavior of the ROI before you specify the size and position of the ROI. After creating the object, you can specify the size and position interactively by using the `draw` function or programmatically by modifying properties of the object.

Syntax

```
roi = images.roi.Circle
roi = images.roi.Circle(ax)
roi = images.roi.Circle( ___ ,Name,Value)
```

Description

`roi = images.roi.Circle` creates a `Circle` object with default properties.

`roi = images.roi.Circle(ax)` creates the ROI on the axes specified by `ax`.

`roi = images.roi.Circle( ___ ,Name,Value)` sets properties on page 1-2783 using name-value arguments. You can specify multiple name-value arguments. Enclose each property name in single quotes.

Example: `images.roi.Circle('Color','y')` creates a yellow colored `Circle` object.

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Properties

`Center` — Center of ROI

`[]` (default) | 1-by-2 numeric vector

Center of the ROI, specified as a 1-by-2 numeric vector of the form `[x y]`. The values *x* and *y* are the coordinates of the center point of the ROI. The value of this property changes automatically when you draw or move the ROI.

`Color` — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0.4470 0.7410]

ContextMenu — Context menu
ContextMenu object

Context menu that displays when you right-click the ROI, specified as a ContextMenu object. You can create a custom context menu by using the uicontextmenu function and then configuring context menu properties.

Deletable — Context menu provides option to delete the ROI
true or 1 (default) | false or 0

Context menu provides an option to delete the ROI, specified as a numeric or logical 1 (true) or 0 (false). When the value is true, you can delete the ROI interactively using the context menu. When the value is false, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the delete function.

DrawingArea — Area of axes in which you can interactively place ROI
'auto' (default) | 'unlimited' | [x,y,w,h]

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
'auto'	The drawing area is the current axes limits (default).
'unlimited'	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
[x,y,w,h]	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

FaceAlpha — Transparency of ROI face
0.2 (default) | number in the range [0, 1]

Transparency of the ROI face, specified as a number in the range [0, 1]. When the value is `1`, the ROI face is completely opaque. When the value is `0`, the ROI face is completely transparent.

`FaceSelectable` — ROI face can capture clicks
`true` or `1` (default) | `false` or `0`

ROI face can capture clicks, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `true`, the ROI face captures mouse clicks. When `false`, the ROI face does not capture mouse clicks.

`HandleVisibility` — Visibility of ROI handle in `Children` property of parent
`'on'` (default) | `'off'` | `'callback'`

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI
`'all'` (default) | `'none'` | `'reshape'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable (default).
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area but not reshaped.
`'reshape'`	The ROI can be reshaped but not translated.

`Label` — ROI label
`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background
`1` (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color
`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'LabelTextColor','r'`

Example: `'LabelTextColor','green'`

Example: `'LabelTextColor',[0 0.4470 0.7410]`

`LabelVisible` — Label visibility

`'on'` (default) | `'off'` | `'hover'`

Label visibility, specified as one of these values.

Value	Description
`'on'`	Label is visible when the ROI is visible.
`'hover'`	Label is visible only when the mouse is hovering over the ROI.
`'off'`	Label is not visible.

`LineWidth` — Width of ROI border

positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

`MarkerSize` — Marker size
positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

`Parent` — ROI parent
`Axes` object | `UIAxes` object

ROI parent, specified as an `Axes` or `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

`Radius` — Radius of circle
nonnegative number

Radius of the circle, specified as a nonnegative number. You can also set this property by drawing or resizing the circle.

`Selected` — Selection state of ROI
`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`). You can also set this property interactively. For example, clicking on the ROI selects the ROI and sets this property to `true`. Similarly, pressing the **Ctrl** key and clicking the ROI deselects the ROI and sets the value of this property to `false`.

`SelectedColor` — Color of ROI when selected
`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	

Color Name	Short Name	RGB Triplet	Appearance
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'SelectedColor','r'

Example: 'SelectedColor','green'

Example: 'SelectedColor',[0 0.4470 0.7410]

StripeColor — Color of ROI stripe

'none' (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or 'none'. If you specify 'none', then the ROI edge is a solid color specified by Color. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by Color and StripeColor.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

Tag — Tag to associate with ROI
`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI
any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `Circle` object does not use this data.

`Vertices` — Locations of points on perimeter
n-by-2 numeric matrix

This property is read-only.

Locations of points on the perimeter of the circle, returned as an *n*-by-2 numeric matrix, where *n* is the total number of vertices.

`Visible` — ROI visibility
`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Object Functions

addlistener Create event listener bound to event source

beginDrawingFromPoint	Begin drawing ROI from specified point
bringToFront	Bring ROI to front of Axes stacking order
createMask	Create binary mask image from ROI
draw	Begin drawing ROI interactively
inROI	Query if points are located in ROI
wait	Block MATLAB command line until ROI operation is finished

Examples

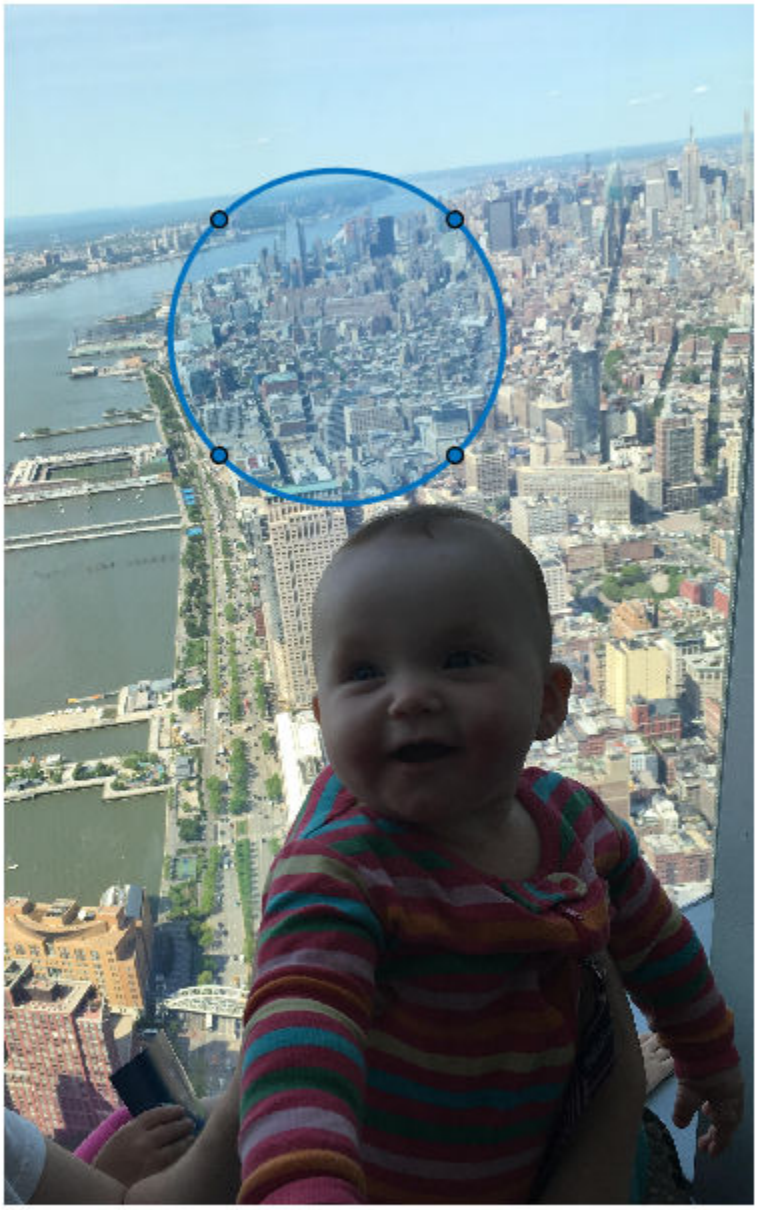
Create Circular ROI Non-interactively

Read an image into the workspace and display it.

```
I = imread('baby.jpg');
figure
imshow(I)
```

Create a circular ROI on the image. Use the `'Center'` property to specify the location and the `'Radius'` property to specify the size. For programmatically created ROIs, if you want the ROI drawn in a specific axes, you must specify that axes as an input argument. Otherwise, an instance of the `images.roi.Circle` class is created but not displayed. In this example, specify the current axes (`gca`) to draw the ROI on the image in that axes.

```
h = images.roi.Circle(gca,'Center',[1000 1000],'Radius',500);
```



Set Up Listeners for Circle ROI Events

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a circular ROI on the image. Because this example specifies the size and location of the ROI, you do not have to call the `draw` method to enable interactive drawing.

```
roi = images.roi.Circle(gca,'Center',[100 100],'Radius',50);
```

Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays at the command line the current center and the current radius of the `Circle` ROI.

```
function allevents(src,evt)
evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving Current Center: ' mat2str(evt.CurrentCenter)]);
            disp(['ROI moving Current Radius: ' mat2str(evt.CurrentRadius)]);
        case{'ROIMoved'}
            disp(['ROI moved Current Center: ' mat2str(evt.CurrentCenter)]);
            disp(['ROI moved Current Radius: ' mat2str(evt.CurrentRadius)]);
    end
end
```

More About

Events

To receive notification from the ROI when certain events happen, set up listeners for these events. You can specify a callback function that executes when one of these events occurs. When the ROI notifies your application through the listener, it returns data specific to the event. Look at the event class for the specific event to see what is returned.

For an example, see “Set Up Listeners for Circle ROI Events” on page 1-2792.

Event Name	Trigger	Event Data	Event Attributes

`DeletingROI`	ROI is about to be interactively deleted.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingStarted`	ROI is about to be interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingFinished`	ROI has been interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`MovingROI`	ROI shape or location is being interactively changed.	`images.roi.CircleMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIMoved`	ROI shape or location has been interactively changed.	`images.roi.CircleMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIClicked`	ROI has been clicked.	`images.roi.ROIClickedEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

Tips

- To draw the ROI interactively using the `draw` or `drawcircle` function, position the cursor on the axes and click and drag to create the shape. To finish drawing, release the pointer.
- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Finish drawing the ROI.	Release the mouse cursor.
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Resize (reshape) the ROI.	Position pointer over a vertex and then click and drag.
Move the ROI.	Position the cursor anywhere inside the ROI, press and hold the mouse, and move the ROI over the image.
Delete the ROI.	Position the cursor on the circle, right-click, and select **Delete Circle** from the context menu. You can also delete the ROI programmatically using the `delete` function.

- For information about using an ROI in an app created with App Designer, see “Using ROIs in Apps Created with App Designer”.

Compatibility Considerations

`UIContextMenu` property is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` property to assign a context menu to a graphics object or UI component is not recommended. Use the `ContextMenu` property instead. The property values are the same.

There are no plans to remove support for the `UIContextMenu` property at this time. However, the `UIContextMenu` property no longer appears in the list returned by calling the `get` function on a graphics object or UI component.

See Also

`drawcircle` | `Ellipse`

Topics

“Create Image Comparison Tool Using ROIs”
“Create ROI Shapes”

Introduced in R2018b

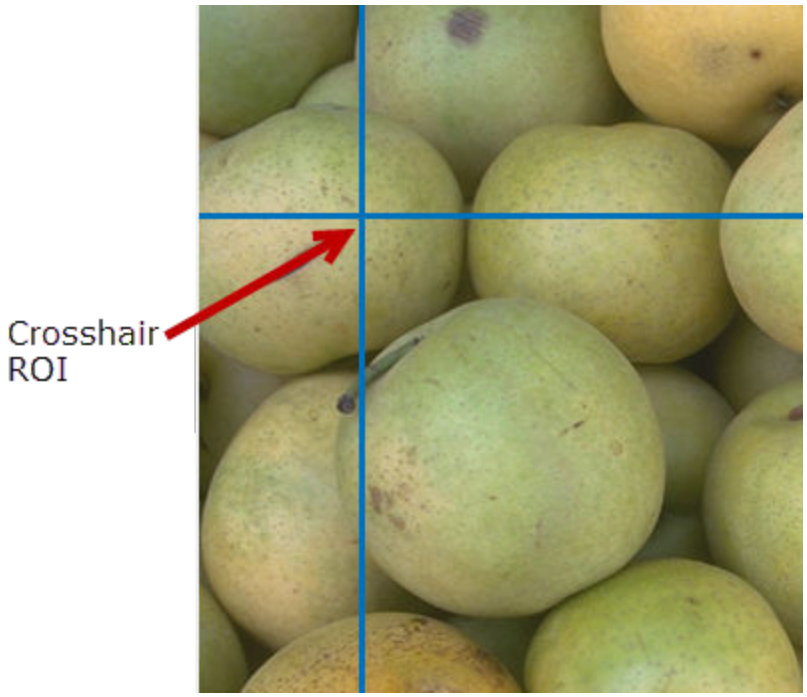
Crosshair

Crosshair region of interest

Description

A `Crosshair` object specifies the position of a crosshair region-of-interest (ROI). You can customize the appearance and interactive behavior of the ROI.

For more information about using this ROI, including keyboard shortcuts, see “Tips” on page 1-2808.



Creation

There are two ways to create a `Crosshair` object. For more information, see “Create ROI Shapes”.

- Use the `drawcrosshair` function. Use this function when you want to create the ROI and set the appearance in a single command. You can specify the position of the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments.
- Use the `images.roi.Crosshair` function described here. Use this function when you want to specify the appearance and behavior of the ROI before you specify the position of the ROI. After creating the object, you can specify the position interactively by using the `draw` function or programmatically by modifying properties of the object.

Syntax

```
roi = images.roi.Crosshair
```

```
roi = images.roi.Crosshair(ax)
roi = images.roi.Crosshair( ___ ,Name,Value)
```

Description

`roi = images.roi.Crosshair` creates a `Crosshair` object with default properties.

`roi = images.roi.Crosshair(ax)` creates the ROI in the axes specified by `ax`.

`roi = images.roi.Crosshair( ___ ,Name,Value)` sets properties on page 1-2797 of the ROI using name-value arguments. You can specify multiple name-value arguments. Enclose each property name in single quotes.

Example: `images.roi.Crosshair('Color','y')` creates a yellow colored `Crosshair` object.

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Properties

Color — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0.4470 0.7410]

ContextMenu — Context menu
ContextMenu object

Context menu that displays when you right-click the ROI, specified as a ContextMenu object. You can create a custom context menu by using the uicontextmenu function and then configuring context menu properties.

Deletable — Context menu provides option to delete the ROI
true or 1 (default) | false or 0

Context menu provides an option to delete the ROI, specified as a numeric or logical 1 (true) or 0 (false). When the value is true, you can delete the ROI interactively using the context menu. When the value is false, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the delete function.

DrawingArea — Area of axes in which you can interactively place ROI
'auto' (default) | 'unlimited' | [x,y,w,h]

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
'auto'	The drawing area is the current axes limits (default).
'unlimited'	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
[x,y,w,h]	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

HandleVisibility — Visibility of ROI handle in Children property of parent
'on' (default) | 'off' | 'callback'

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI

`'all'` (default) | `'none'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable.
`'none'`	The ROI is not interactable, and no drag points are visible.

`Label` — ROI label

`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background

1 (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color

`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	

Color Name	Short Name	RGB Triplet	Appearance
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'LabelTextColor','r'

Example: 'LabelTextColor','green'

Example: 'LabelTextColor',[0 0.4470 0.7410]

LabelVisible — Label visibility
'on' (default) | 'off' | 'hover'

Label visibility, specified as one of these values.

Value	Description
'on'	Label is visible when the ROI is visible.
'hover'	Label is visible only when the mouse is hovering over the ROI.
'off'	Label is not visible.

LineWidth — Width of ROI border
positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

Parent — ROI parent
Axes object | UIAxes object

ROI parent, specified as an Axes or UIAxes object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Position — Position of ROI
1-by-2 numeric vector

Position of the ROI, specified as a 1-by-2 numeric vector of the form `[x y]`. The values `x` and `y` specify the *x*- and *y*-coordinates of the location where the horizontal line crosses the vertical line in the crosshair ROI. This value changes automatically when you draw or move the ROI.

`Selected` — Selection state of ROI

`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`). You can also set this property interactively. For example, clicking on the ROI selects the ROI and sets this property to `true`. Similarly, pressing the **Ctrl** key and clicking the ROI deselects the ROI and sets the value of this property to `false`.

`SelectedColor` — Color of ROI when selected

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'SelectedColor','r'`

Example: `'SelectedColor','green'`

Example: `'SelectedColor',[0 0.4470 0.7410]`

`StripeColor` — Color of ROI stripe

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

Tag — Tag to associate with ROI

`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI

any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `Crosshair` object does not use this data.

`Visible` — ROI visibility

`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Object Functions

addlistener	Create event listener bound to event source
beginDrawingFromPoint	Begin drawing ROI from specified point
bringToFront	Bring ROI to front of Axes stacking order
draw	Begin drawing ROI interactively
wait	Block MATLAB command line until ROI operation is finished

Examples

Create Crosshair ROI

Read an image into the workspace and display it.

```
I = imread('baby.jpg');
figure;
imshow(I)
```

Place a crosshair ROI on the image programmatically. When you specify the position of the ROI, you must specify the axes.

```
h = images.roi.Crosshair(gca,'Position',[100,100]);
```

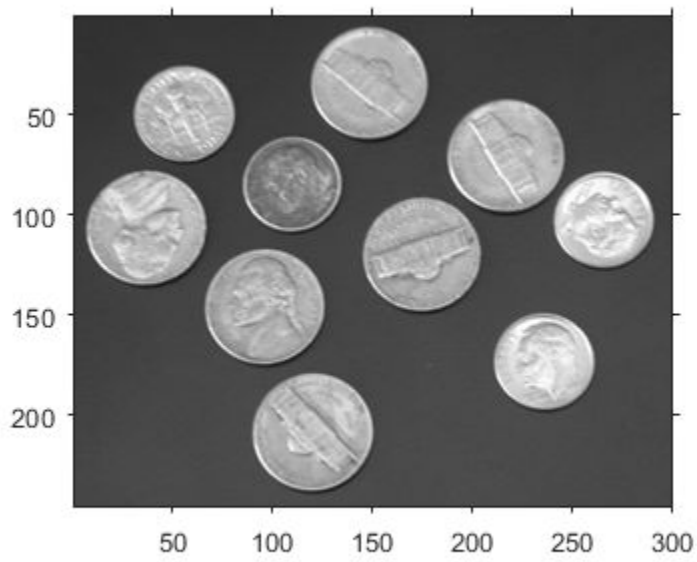
Create Crosshair Tool to Check Pixel Values

Read an image into the workspace.

```
img = imread('coins.png');
```

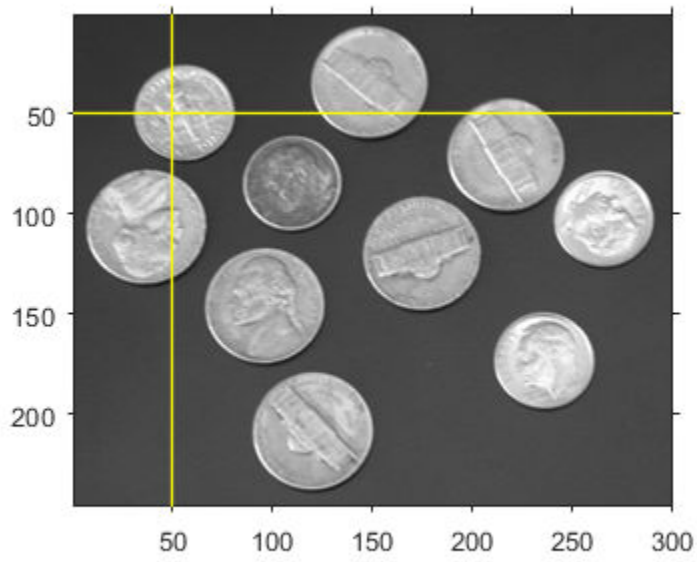
Display the image in a figure.

```
hAx = gca;
imObj = imshow(img,'Parent',hAx);
imObj.Parent.Visible = 'on';
```



Create a crosshair ROI on the image.

```
h = images.roi.Crosshair('Parent',hAx,'Position',[50 50],'LineWidth',1,'Color','y');
```



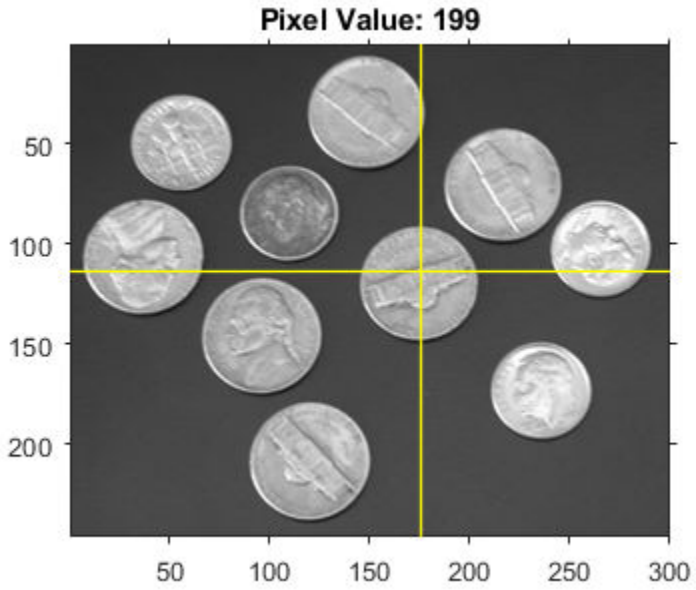
Set up a listener to get notification when the ROI moves over the image. Because the code displays the value of the pixel under the crosshair, you must pass the image as an argument to the listener.

```
addlistener(h,'MovingROI',@(src,data)displayInfo(src,data,hAx,img));
```

Define the `displayInfo` function called by the listener when a `'MovingROI'` event occurs.

```
function displayInfo(~,data,hAx,img)
pos = ceil(data.CurrentPosition);
title(hAx,['Pixel Value: ',num2str(img(pos(2),pos(1)))])
end
```

Appearance of the image with title during interactive movement of the Crosshair ROI.



More About

Events

To receive notification from the ROI when certain events happen, set up listeners for these events. You can specify a callback function that executes when one of these events occurs. When the ROI notifies your application through the listener, it returns data specific to the event. Look at the event class for the specific event to see what is returned.

For an example of using event listeners, see “Create Crosshair Tool to Check Pixel Values” on page 1-2805.

Event Name	Trigger	Event Data	Event Attributes

`DeletingROI`	ROI is about to be interactively deleted.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

Event Name	Trigger	Event Data	Event Attributes

`DrawingStarted`	ROI is about to be interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingFinished`	ROI has been interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`MovingROI`	ROI shape or location is being interactively changed.	`images.roi.ROIMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIMoved`	ROI shape or location has been interactively changed.	`images.roi.ROIMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIClicked`	ROI has been clicked.	`images.roi.ROIClickedEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

Tips

- To draw the ROI interactively using the `draw` or `drawcrosshair` function, position the cursor over the image and click the mouse to draw the ROI.
- The ROI supports the following interactivity, including keyboard shortcuts.

Task	Description
Cancel drawing operation.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Move the ROI.	Position the cursor over the center of the crosshair ROI (where the two lines cross) and click and drag the crosshair. Another way to move the crosshair ROI is to position the cursor anywhere on one of the two lines and click. The other line in the crosshair jumps to the new crosshair center position.
Delete the ROI.	Position the cursor over the ROI, right-click, and then choose **Delete Crosshair** from the context menu. You can also delete the ROI programmatically by using the `delete` function.

- For information about using an ROI in an app created with App Designer, see “Using ROIs in Apps Created with App Designer”.

Compatibility Considerations

`UIContextMenu` property is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` property to assign a context menu to a graphics object or UI component is not recommended. Use the `ContextMenu` property instead. The property values are the same.

There are no plans to remove support for the `UIContextMenu` property at this time. However, the `UIContextMenu` property no longer appears in the list returned by calling the `get` function on a graphics object or UI component.

See Also

`drawcrosshair` | `Line` | `Point` | `Polyline`

Topics

“Create ROI Shapes”

Introduced in R2019b

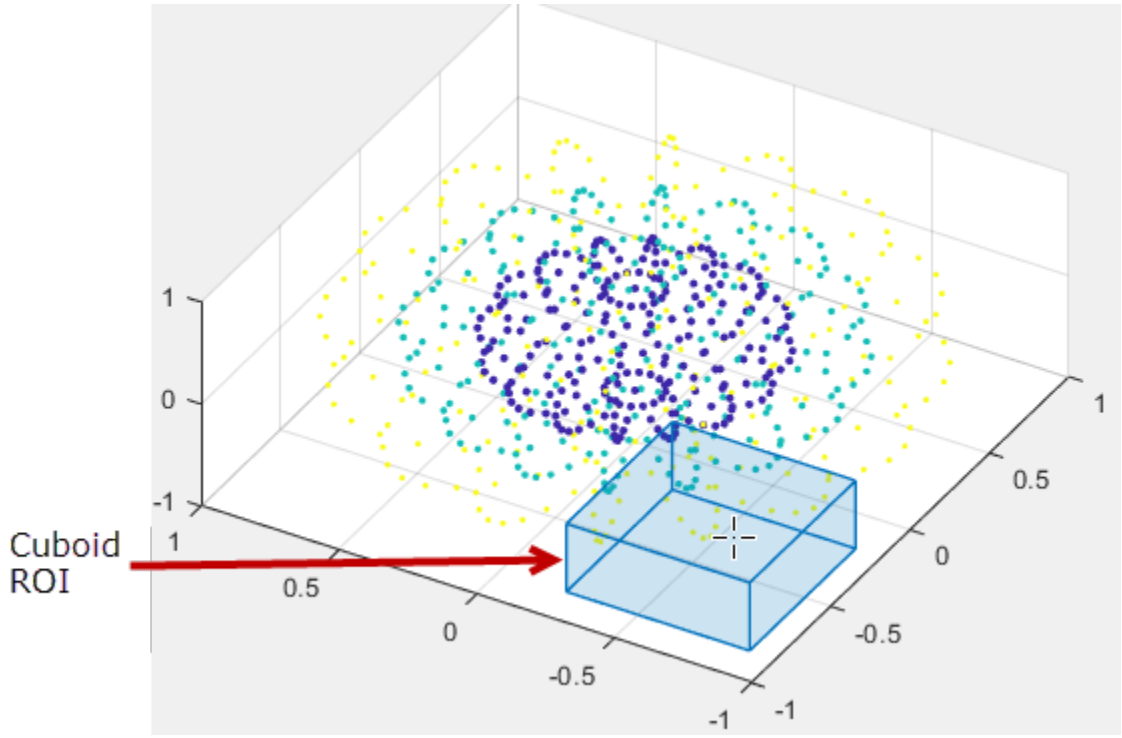
Cuboid

Cuboidal region of interest

Description

A `Cuboid` object specifies the shape and position of a 3-D cuboidal region of interest (ROI). You can customize the appearance and interactive behavior of the ROI.

For more information about using this ROI, including keyboard shortcuts and a right-click context menu, see “Tips” on page 1-2822.



Creation

There are two ways to create a `Cuboid` object. For more information, see “Create ROI Shapes”.

- Use the `drawcuboid` function. Use this function when you want to create the ROI and set the appearance in a single command. You can specify the shape and position of the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments.
- Use the `images.roi.Cuboid` function described here. Use this function when you want to specify the appearance and behavior of the ROI before you specify the shape and position of the ROI. After creating the object, you can specify the shape and position interactively by using the `draw` function or programmatically by modifying properties of the object.

Syntax

```
roi = images.roi.Cuboid
roi = images.roi.Cuboid(ax)
roi = images.roi.Cuboid( ___ ,Name,Value)
```

Description

`roi = images.roi.Cuboid` creates a `Cuboid` object with default properties.

`roi = images.roi.Cuboid(ax)` creates an ROI in the axes specified by `ax`.

`roi = images.roi.Cuboid( ___ ,Name,Value)` sets properties on page 1-2811 using name-value arguments. You can specify multiple name-value arguments. Enclose each property name in single quotes.

Example: `images.roi.Cuboid('Color','y')` creates a yellow colored `Cuboid` object.

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Properties

Color — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0.4470 0.7410]

ContextMenu — Context menu
ContextMenu object

Context menu that displays when you right-click the ROI, specified as a ContextMenu object. You can create a custom context menu by using the uicontextmenu function and then configuring context menu properties.

Deletable — Context menu provides option to delete the ROI
true or 1 (default) | false or 0

Context menu provides an option to delete the ROI, specified as a numeric or logical 1 (true) or 0 (false). When the value is true, you can delete the ROI interactively using the context menu. When the value is false, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the delete function.

DrawingArea — Area of axes in which you can interactively place ROI
'auto' (default) | 'unlimited' | 1-by-6 numeric array

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
'auto'	The drawing area is a superset of the current axes limits and a bounding box that surrounds the ROI (default).
'unlimited'	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
[x,y,z,w,h,d]	The drawing area is restricted to a region beginning at (*x*,*y*,*z*), with width *w*, height *h*, and depth *d*.

EdgeAlpha — Transparency of ROI edge
1 (default) | number in the range [0, 1]

Transparency of ROI edge, specified as a number in the range [0, 1]. When set to `1`, the ROI edge is completely opaque. When set to `0`, the ROI edge is completely transparent.

`FaceAlpha` — Transparency of ROI faces

`0.2` (default) | number in the range [0, 1]

Transparency of the ROI faces, specified as a number in the range [0, 1]. When the value is `1`, the ROI faces are completely opaque. When the value is `0`, the ROI faces are completely transparent.

`FaceAlphaOnHover` — Transparency of ROI face directly underneath mouse pointer

`0.4` (default) | number in the range [0, 1] | `'none'`

Transparency of ROI face directly underneath the mouse pointer, specified as a number in the range [0, 1] or `'none'`, to indicate no change to face transparency. When set to `1`, the face under the mouse pointer is completely opaque. When set to `0`, the face is completely transparent.

`FaceColorOnHover` — Color of ROI face directly underneath mouse pointer

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI face directly underneath the mouse pointer, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the face color does not when the mouse hovers over the face. When you are not hovering over a face of the ROI, the value of the ROI `Color` property determines the face color.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	

RGB Triplet	Appearance
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'FaceColorOnHover','r'

Example: 'FaceColorOnHover','green'

Example: 'FaceColorOnHover',[0 0.4470 0.7410]

HandleVisibility — Visibility of ROI handle in Children property of parent
'on' (default) | 'off' | 'callback'

Visibility of the ROI handle in the Children property of the parent, specified as one of the values in this table.

Value	Description
'on'	The object handle is always visible (default).
'off'	The object handle is hidden at all times.
'callback'	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

InteractionsAllowed — Interactivity of ROI
'all' (default) | 'none' | 'translate'

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
'all'	The ROI is fully interactable.
'none'	The ROI is not interactable, and no drag points are visible.
'translate'	The ROI can be translated (moved) within the drawing area.

Label — ROI label
'' (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label ('').

LabelAlpha — Transparency of text background
1 (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to 1, the text background is completely opaque. When set to 0, the text background is completely transparent.

LabelTextColor — Label text color
'black' (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'LabelTextColor','r'`

Example: `'LabelTextColor','green'`

Example: `'LabelTextColor',[0 0.4470 0.7410]`

`LabelVisible` — Label visibility

`'on'` (default) | `'off'` | `'hover'`

Label visibility, specified as one of these values.

Value	Description
`'on'`	Label is visible when the ROI is visible.
`'hover'`	Label is visible only when the mouse is hovering over the ROI.
`'off'`	Label is not visible.

`LineWidth` — Width of ROI edge

1 (default) | positive number

Width of the ROI edge, specified as a positive number in points.

`Parent` — ROI parent
`Axes` object | `UIAxes` object

ROI parent, specified as an `Axes` or `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

`Position` — Position of cuboid
1-by-6 numeric vector

Position of the cuboid, specified as a 1-by-6 numeric vector of the form `[xmin, ymin, zmin, width, height, depth]`. This property updates automatically when you draw or move the cuboid.

`Rotatable` — Ability of cuboid to be rotated
`'none'` (default) | `'x'` | `'y'` | `'z'` | `'all'`

Ability of the cuboid to be rotated, specified as one of these values.

Value	Description
`'all'`	ROI is fully rotatable.
`'x'`	ROI can only be rotated about the *x* axis.
`'y'`	ROI can only be rotated about the *y* axis.
`'z'`	ROI can only be rotated about the *z* axis.
`'none'`	ROI is not rotatable.

`RotationAngle` — Angle of ROI rotation
`[0 0 0]` (default) | 1-by-3 numeric vector

Angle of ROI rotation, specified as a 1-by-3 numeric vector of rotation angles, measured in degrees. The rotation angles array is of the form `[x_angle y_angle z_angle]`, measured about the *x*-, *y*-, and *z*-axis, respectively. Rotation is applied about the ROI centroid in order *z*, then *y*, then *x*.

The value of `RotationAngle` does not impact the values in the `Position` property. `Position` represents the cuboid prior to any rotation. When you rotate the cuboid, use the `Vertices` property to determine the location of the rotated cuboid.

`ScrollWheelDuringDraw` — Ability of scroll wheel to adjust size
`'all'` (default) | `xresize` | `yresize` | `zresize` | `'none'`

Ability of the scroll wheel to adjust the size of the cuboid during interactive placement, specified as one of these values.

Value	Description
`'allresize'`	Scroll wheel impacts all ROI dimensions.
`'xresize'`	Scroll wheel impacts only the *x* dimension.
`'yresize'`	Scroll wheel impacts only the *y* dimension.
`'zresize'`	Scroll wheel impacts only the *z* dimension.
`'none'`	Scroll wheel has no effect.

`Selected` — Selection state of ROI

`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`). You can also set this property interactively. For example, clicking on the ROI selects the ROI and sets this property to `true`. Similarly, pressing the **Ctrl** key and clicking the ROI deselects the ROI and sets the value of this property to `false`.

`SelectedColor` — Color of ROI when selected

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'SelectedColor','r'`

Example: `'SelectedColor','green'`

Example: `'SelectedColor',[0 0.4470 0.7410]`

`StripeColor` — Color of ROI stripe

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

`Tag` — Tag to associate with ROI

`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI

any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `Cuboid` object does not use this data.

`Vertices` — Locations of corners of cuboid
8-by-3 numeric matrix

This property is read-only.

Locations of the corners of the cuboid, returned as an 8-by-3 numeric matrix.

`Visible` — ROI visibility
`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Object Functions

addlistener	Create event listener bound to event source
beginDrawingFromPoint	Begin drawing ROI from specified point
bringToFront	Bring ROI to front of Axes stacking order
draw	Begin drawing ROI interactively
inROI	Query if points are located in ROI
wait	Block MATLAB command line until ROI operation is finished

Examples

Create Cuboid ROI on Scatter Plot

Define vectors for 3-D scatter data.

```
[x,y,z] = sphere(16);
X = [x(:)*.5 x(:)*.75 x(:)];
Y = [y(:)*.5 y(:)*.75 y(:)];
Z = [z(:)*.5 z(:)*.75 z(:)];
```

Specify the size and color of each marker.

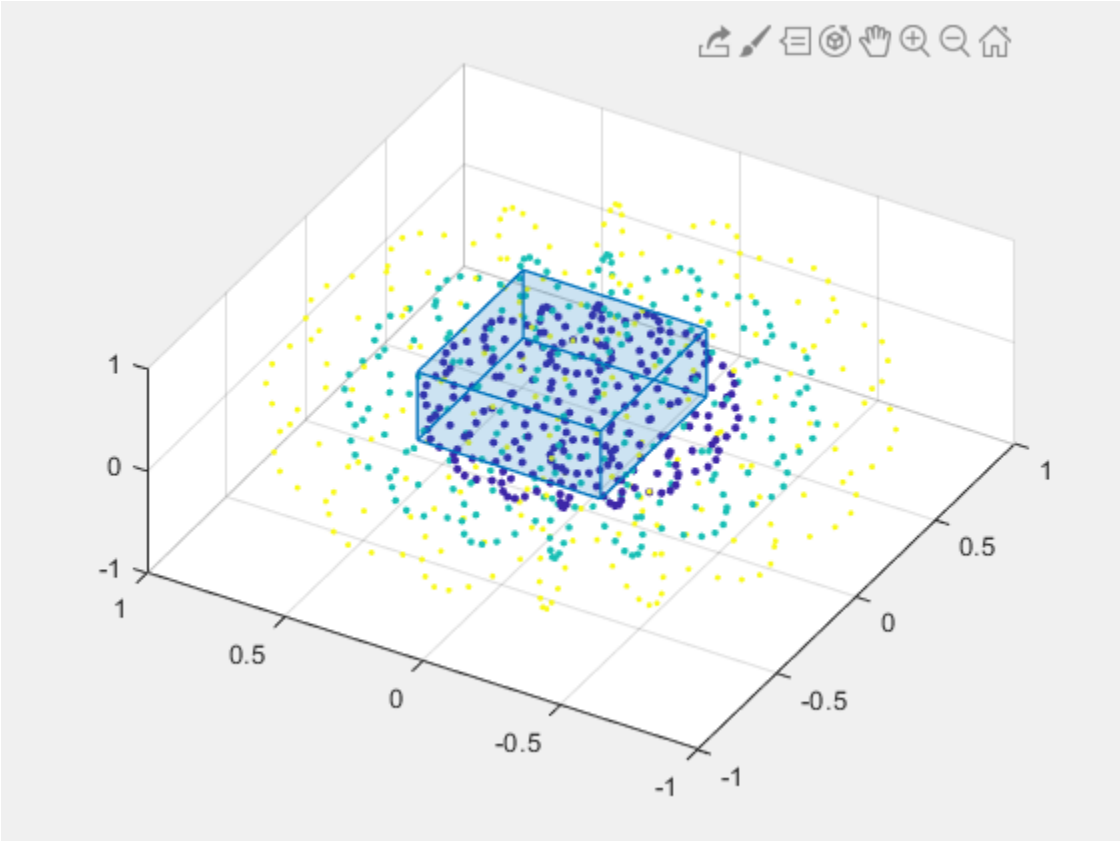
```
S = repmat([1 .75 .5]*10,numel(x),1);
C = repmat([1 2 3],numel(x),1);
```

Create a 3-D scatter plot and use view to the change the angle of the axes in the figure.

```
figure
hScatter = scatter3(X(:),Y(:),Z(:),S(:),C(:),'filled');
view(-60,60);
```

Begin placing a cuboid in the axes that snaps to the nearest point from the scatter plot. Adjust the size of the cuboid during interactive placement by using the scroll wheel.

```
ax = gca;
h = images.roi.Cuboid(ax);
draw(h)
```



Set Up Listeners for Cuboid ROI Events

Define vectors for 3-D scattered data.

```
[x,y,z] = sphere(16);
X = [x(:)*.5 x(:)*.75 x(:)];
Y = [y(:)*.5 y(:)*.75 y(:)];
Z = [z(:)*.5 z(:)*.75 z(:)];
```

Specify the size and color of each marker.

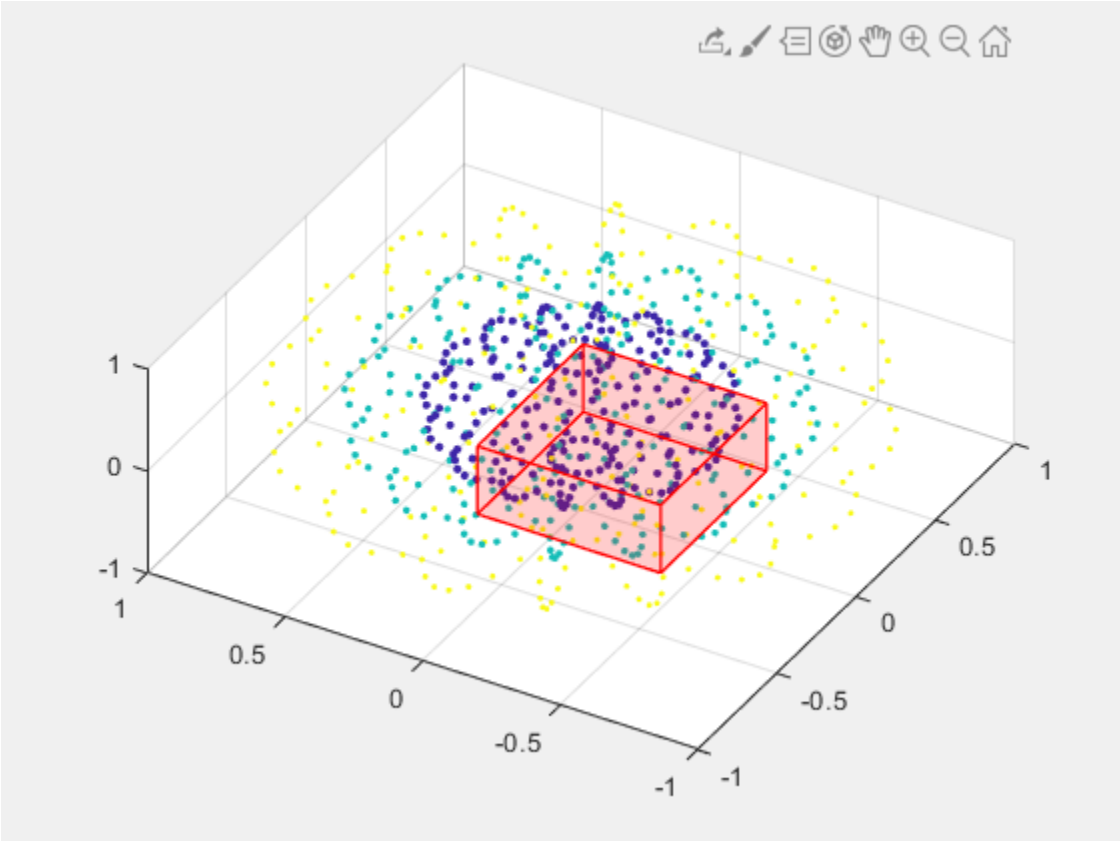
```
S = repmat([1 .75 .5]*10,numel(x),1);
C = repmat([1 2 3],numel(x),1);
```

Create a 3-D scatter plot and use view to change the angle of the axes in the figure.

```
figure
hScatter = scatter3(X(:),Y(:),Z(:),S(:),C(:),'filled');
view(-60,60);
```

Create a Cuboid ROI object, specifying the color. Call the `draw` object function to enable interactive drawing of the cuboid shape.

```
roi = images.roi.Cuboid(gca,'Color','r');
draw(roi)
```



Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays the previous position and the current position of the ROI.

```
function allevents(src,evt)
    evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving current position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved previous position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved current position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

More About

Events

To receive notification from the ROI when certain events happen, set up listeners for these events. You can specify a callback function that executes when one of these events occurs. When the ROI notifies your application through the listener, it returns data specific to the event. Look at the event class for the specific event to see what is returned.

For an example, see “Set Up Listeners for Cuboid ROI Events” on page 1-2820.

Event Name	Trigger	Event Data	Event Attributes

`DeletingROI`	ROI is about to be interactively deleted.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingStarted`	ROI is about to be interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingFinished`	ROI has been interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`MovingROI`	ROI shape or location is being interactively changed.	`images.roi.CuboidMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIMoved`	ROI shape or location has been interactively changed.	`images.roi.CuboidMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIClicked`	ROI has been clicked.	`images.roi.ROIClickedEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

Tips

- To draw the ROI interactively using the `draw` or `drawcuboid` function, position the cursor on the ROI and click and drag to move or change the size of the shape. To finish the ROI, release the mouse button.
- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Fine-tune size of ROI as you are drawing.	As you draw the ROI, use the scroll wheel to make small changes to its size.
Resize (reshape) the ROI.	Position the pointer over a surface of the ROI that is visible from your point of view and then click and drag.
Move the ROI.	Position the pointer over a surface of the ROI that is visible from your point of view. Press **Shift** as you click and drag to move the ROI.
Delete the ROI.	Position the pointer over the ROI, right-click, and select **Delete Cuboid** from the context menu. You can also delete the ROI programmatically using the `delete` function.

- For information about using an ROI in an app created with App Designer, see “Using ROIs in Apps Created with App Designer”.

Compatibility Considerations

`UIContextMenu` property is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` property to assign a context menu to a graphics object or UI component is not recommended. Use the `ContextMenu` property instead. The property values are the same.

There are no plans to remove support for the `UIContextMenu` property at this time. However, the `UIContextMenu` property no longer appears in the list returned by calling the `get` function on a graphics object or UI component.

See Also

`drawcuboid` | `Rectangle`

Topics

“Create ROI Shapes”

Introduced in R2019a

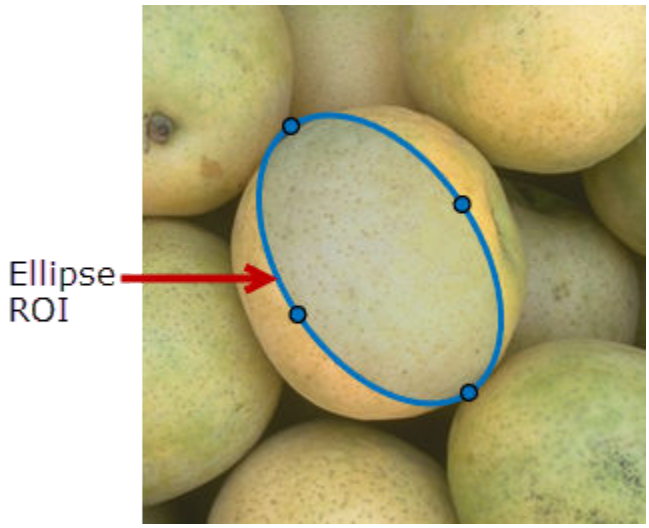
Ellipse

Elliptical region of interest

Description

An `Ellipse` object specifies the shape and position of an elliptical region-of-interest (ROI). You can customize the appearance and interactive behavior of the ROI.

For more information about using this ROI, including keyboard shortcuts, see “Tips” on page 1-2836.



Creation

There are two ways to create an `Ellipse` object. For more information, see “Create ROI Shapes”.

- Use the `drawellipse` function. Use this function when you want to create the ROI and set the appearance in a single command. You can specify the shape and position of the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments.
- Use the `images.roi.Ellipse` function described here. Use this function when you want to specify the appearance and behavior of the ROI before you specify the shape and position of the ROI. After creating the object, you can specify the shape and position interactively by using the `draw` function or programmatically by modifying properties of the object.

Syntax

```
roi = images.roi.Ellipse
roi = images.roi.Ellipse(ax)
roi = images.roi.Ellipse( ___ ,Name,Value)
```

Description

`roi = images.roi.Ellipse` creates an `Ellipse` object with default property values.

`roi = images.roi.Ellipse(ax)` creates the ROI on the axes specified by `ax`.

`roi = images.roi.Ellipse( ___ ,Name,Value)` sets properties on page 1-2825 using name-value arguments. You can specify multiple name-value arguments. Enclose each property name in single quotes.

Example: `images.roi.Ellipse('Color','y')` creates a yellow colored `Ellipse` object.

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Properties

`AspectRatio` — Aspect ratio of ellipse

positive number

Aspect ratio of the ellipse, specified as a positive number. The value of this property changes automatically when you draw or resize the ellipse, or by setting the `SemiAxes` property. The `Ellipse` object calculates this value as `height/width`. The default value is `(1+sqrt(5))/2`.

`Center` — Center of ROI

`[]` (default) | 1-by-2 numeric vector

Center of the ROI, specified as a 1-by-2 numeric vector of the form `[x y]`. The values *x* and *y* are the coordinates of the center point of the ROI. The value of this property changes automatically when you draw or move the ROI.

`Color` — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	

Color Name	Short Name	RGB Triplet	Appearance
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0.4470 0.7410]

ContextMenu — Context menu
ContextMenu object

Context menu that displays when you right-click the ROI, specified as a ContextMenu object. You can create a custom context menu by using the uicontextmenu function and then configuring context menu properties.

Deletable — Context menu provides option to delete the ROI
true or 1 (default) | false or 0

Context menu provides an option to delete the ROI, specified as a numeric or logical 1 (true) or 0 (false). When the value is true, you can delete the ROI interactively using the context menu. When the value is false, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the delete function.

DrawingArea — Area of axes in which you can interactively place ROI
'auto' (default) | 'unlimited' | [x,y,w,h]

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
'auto'	The drawing area is the current axes limits (default).

Value	Description
`'unlimited'`	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
`[x,y,w,h]`	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

`FaceAlpha` — Transparency of ROI face
`0.2` (default) | number in the range [0, 1]

Transparency of the ROI face, specified as a number in the range [0, 1]. When the value is `1`, the ROI face is completely opaque. When the value is `0`, the ROI face is completely transparent.

`FaceSelectable` — ROI face can capture clicks
`true` or `1` (default) | `false` or `0`

ROI face can capture clicks, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `true`, the ROI face captures mouse clicks. When `false`, the ROI face does not capture mouse clicks.

`FixedAspectRatio` — Aspect ratio remains constant
`false` or `0` (default) | `true` or `1`

Aspect ratio remains constant during interaction, specified as a numeric or logical `0` (`false`) or `1` (`true`). When the value is `true`, the aspect ratio remains constant when you draw or resize the ROI. When the value is `false`, you can change the aspect ratio when drawing or resizing the ROI. You can change the state of this property using the default context menu.

`HandleVisibility` — Visibility of ROI handle in `Children` property of parent
`'on'` (default) | `'off'` | `'callback'`

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI
`'all'` (default) | `'none'` | `'reshape'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable (default).
`'none'`	The ROI is not interactable, and no drag points are visible.

Value	Description
`'translate'`	The ROI can be translated (moved) within the drawing area but not reshaped.
`'reshape'`	The ROI can be reshaped but not translated.

`Label` — ROI label
`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background
1 (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color
`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	

RGB Triplet	Appearance
[0.6350 0.0780 0.1840]	

Example: 'LabelTextColor','r'

Example: 'LabelTextColor','green'

Example: 'LabelTextColor',[0 0.4470 0.7410]

LabelVisible — Label visibility
'on' (default) | 'off' | 'hover'

Label visibility, specified as one of these values.

Value	Description
'on'	Label is visible when the ROI is visible.
'hover'	Label is visible only when the mouse is hovering over the ROI.
'off'	Label is not visible.

LineWidth — Width of ROI border
positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

MarkerSize — Marker size
positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

Parent — ROI parent
Axes object | UIAxes object

ROI parent, specified as an Axes or UIAxes object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

RotationAngle — Angle around center of ROI
0 (default) | nonnegative numeric scalar

Angle around the center of the ROI, specified as a nonnegative numeric scalar. The angle is measured in degrees in a clockwise direction. The value of this property changes automatically when you draw or move the ROI.

The value of RotationAngle does not impact the value of Position. The Position property represents the initial position of the ROI, before rotation. To determine the location of a rotated ROI, use the Vertices property.

Selected — Selection state of ROI
false or 0 (default) | true or 1

Selection state of the ROI, specified as a numeric or logical 0 (false) or 1 (true). You can also set this property interactively. For example, clicking on the ROI selects the ROI and sets this property to

`true`. Similarly, pressing the **Ctrl** key and clicking the ROI deselects the ROI and sets the value of this property to `false`.

`SelectedColor` — Color of ROI when selected

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'SelectedColor','r'`

Example: `'SelectedColor','green'`

Example: `'SelectedColor',[0 0.4470 0.7410]`

`SemiAxes` — Lengths of semiaxes of ellipse

1-by-2 numeric vector

Lengths of the semiaxis of the ellipse, specified as a 1-by-2 numeric vector of the form `[semiaxis1 semiaxis2]`. The `Ellipse` object assigns the length of the semiaxis that is closest to the *x* direction

to `semiaxis1`. Note however that the shape and orientation of the ellipse can change through interaction. The value of this property changes automatically when you draw or reshape the ROI.

`StripeColor` — Color of ROI stripe

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

`Tag` — Tag to associate with ROI

`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI

any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `Ellipse` object does not use this data.

`Vertices` — Locations of points on perimeter

n-by-2 numeric matrix

This property is read-only.

Locations of points on the perimeter of the ellipse, returned as an *n*-by-2 numeric matrix.

`Visible` — ROI visibility

`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Object Functions

addlistener	Create event listener bound to event source
beginDrawingFromPoint	Begin drawing ROI from specified point
bringToFront	Bring ROI to front of Axes stacking order
createMask	Create binary mask image from ROI
draw	Begin drawing ROI interactively
inROI	Query if points are located in ROI
wait	Block MATLAB command line until ROI operation is finished

Examples

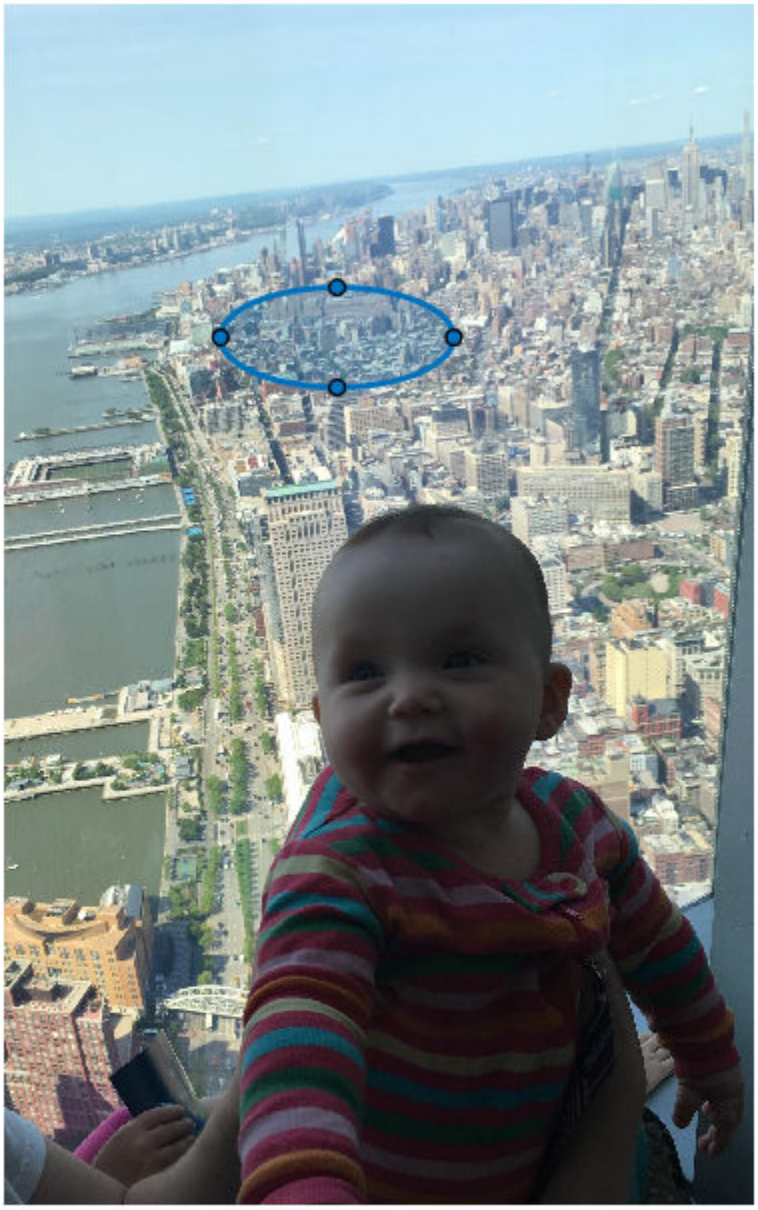
Create Elliptical ROI Non-interactively

Read an image into the workspace and display it.

```
I = imread('baby.jpg');
figure
imshow(I)
```

Create an elliptical ROI on the image, using the `Center` property to specify the location and the `SemiAxes` property to specify its shape. For programmatically created ROIs, if you want the ROI drawn in a specific axes, you must specify that axes as an input argument. Otherwise, an instance of the images.roi.Ellipse class is created but not displayed. In this example, specify the current axes (`gca`) to draw the ROI on the image in that axes.

```
h = images.roi.Ellipse(gca,'Center',[1000 1000],'Semiaxes',[350 150]);
```



Set Up Listeners for Ellipse ROI Events

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw an elliptical ROI on the image. Because this example specifies the size and location of the ROI, you do not have to call the `draw` method to enable interactive drawing.

```
roi = images.roi.Ellipse(gca,'Center',[100 100],'Semiaxes',[50 80]);
```

Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays at the command line the current center and the current semiaxes of the Ellipse ROI.

```
function allevents(src,evt)
evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving Current Center: ' mat2str(evt.CurrentCenter)]);
            disp(['ROI moving Current SemiAxes: ' mat2str(evt.CurrentSemiAxes)]);
        case{'ROIMoved'}
            disp(['ROI moved Current Center: ' mat2str(evt.CurrentCenter)]);
            disp(['ROI moved Current SemiAxes: ' mat2str(evt.CurrentSemiAxes)]);
    end
end
```

More About

Events

To receive notification from the ROI when certain events happen, set up listeners for these events. You can specify a callback function that executes when one of these events occurs. When the ROI notifies your application through the listener, it returns data specific to the event. Look at the event class for the specific event to see what is returned.

For an example, see “Set Up Listeners for Ellipse ROI Events” on page 1-2834.

Event Name	Trigger	Event Data	Event Attributes

`DeletingROI`	ROI is about to be interactively deleted.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingStarted`	ROI is about to be interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingFinished`	ROI has been interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`MovingROI`	ROI shape or location is being interactively changed.	`images.roi.EllipseMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIMoved`	ROI shape or location has been interactively changed.	`images.roi.EllipseMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIClicked`	ROI has been clicked.	`images.roi.ROIClickedEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

Tips

- To draw the ROI interactively using the `draw` or `drawellipse` function, position the cursor on the axes and click and drag to create the shape. To finish the ROI, release the mouse button.
- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Fine-tune width of ellipse as you are drawing.	As you draw the ellipse, use the scroll wheel to make small changes to the width of the ellipse.
Rotate the ROI.	Position the pointer near a vertex. The pointer changes to the rotate pointer. Click and rotate the ROI on its center. To make the rotation snap at 15 degree angles, press **Shift** as you rotate.
Maintain aspect ratio while drawing.	Hold the **Shift** key as you draw. Creates a circular ROI. To lock the aspect ratio, position the pointer on the ROI, right-click, and select **Fix Aspect Ratio** from the context menu
Resize (reshape) the ROI.	Position pointer over a vertex and then click and drag. To main the aspect ratio as you resize, hold the **Shift** key.
Move the ROI.	Position the pointer over the ROI. The pointer changes to the fleur shape. Click and drag to move the ROI.
Delete the ROI.	Position the pointer anywhere in the ROI and right-click. Select Delete Ellipse from the context menu. You can also delete the ROI programmatically using the `delete` function.

- For information about using an ROI in an app created with App Designer, see “Using ROIs in Apps Created with App Designer”.
- To draw a circular ROI, use the `Circle` object.

Compatibility Considerations

`UIContextMenu` property is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` property to assign a context menu to a graphics object or UI component is not recommended. Use the `ContextMenu` property instead. The property values are the same.

There are no plans to remove support for the `UIContextMenu` property at this time. However, the `UIContextMenu` property no longer appears in the list returned by calling the `get` function on a graphics object or UI component.

See Also

`drawellipse` | `Circle`

Topics
“Create ROI Shapes”

Introduced in R2018b

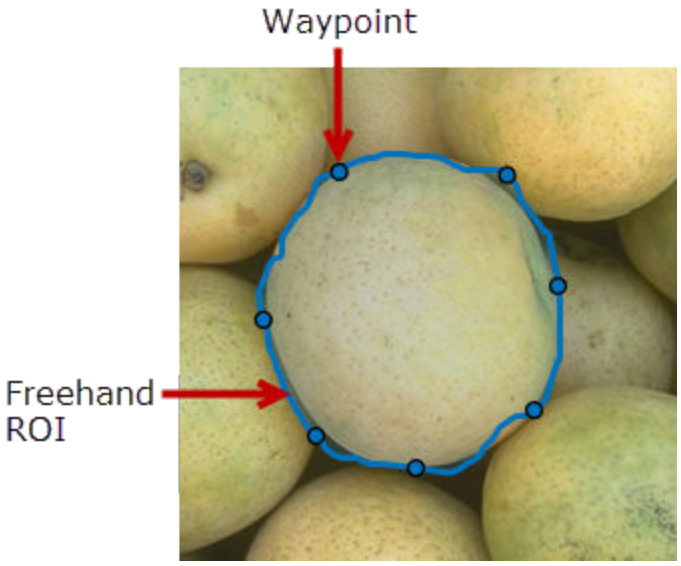
Freehand

Freehand region of interest

Description

A `Freehand` object specifies the shape and position of a hand-drawn region of interest (ROI). You can customize the appearance and interactive behavior of the ROI.

For more information about using a freehand ROI, including keyboard shortcuts, see “Tips” on page 1-2852.



Creation

There are two ways to create a `Freehand` object. For more information, see “Create ROI Shapes”.

- Use the `drawfreehand` function. Use this function when you want to create the ROI and set the appearance in a single command. You can specify the shape and position of the ROI interactively by drawing the ROI over an image using the mouse.
- Use the `images.roi.Freehand` function described here. Use this function when you want to specify the appearance and behavior of the ROI before you specify the shape and position of the ROI. After creating the object, you can specify the shape and position interactively by using the `draw` function.

Syntax

```
roi = images.roi.Freehand
roi = images.roi.Freehand(ax)
roi = images.roi.Freehand( ___ ,Name,Value)
```

Description

`roi = images.roi.Freehand` creates a `Freehand` object with default properties.

`roi = images.roi.Freehand(ax)` creates the ROI in the axes specified by `ax`.

`roi = images.roi.Freehand( ___ ,Name,Value)` sets properties on page 1-2840 of the ROI using name-value arguments. You can specify multiple name-value arguments. Enclose each property name in single quotes.

Example: `images.roi.Freehand('Color','y')` creates a yellow colored `Freehand` object.

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Properties

Closed — Close ROI

`true` or `1` (default) | `false` or `0`

Close the ROI, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `true`, the `Freehand` object closes the ROI by connecting the last point drawn to the first point drawn.

Color — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'Color','r'`

Example: `'Color','green'`

Example: `'Color',[0 0.4470 0.7410]`

`ContextMenu` — Context menu
`ContextMenu` object

Context menu that displays when you right-click the ROI, specified as a `ContextMenu` object. You can create a custom context menu by using the `uicontextmenu` function and then configuring context menu properties.

`Deletable` — Context menu provides option to delete the ROI
`true` or `1` (default) | `false` or `0`

Context menu provides an option to delete the ROI, specified as a numeric or logical `1` (`true`) or `0` (`false`). When the value is `true`, you can delete the ROI interactively using the context menu. When the value is `false`, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the `delete` function.

`DrawingArea` — Area of axes in which you can interactively place ROI
`'auto'` (default) | `'unlimited'` | `[x,y,w,h]`

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
`'auto'`	The drawing area is the current axes limits (default).
`'unlimited'`	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
`[x,y,w,h]`	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

`FaceAlpha` — Transparency of ROI face
`0.2` (default) | number in the range [0, 1]

Transparency of the ROI face, specified as a number in the range [0, 1]. When the value is `1`, the ROI face is completely opaque. When the value is `0`, the ROI face is completely transparent.

`FaceSelectable` — ROI face can capture clicks
`true` or `1` (default) | `false` or `0`

ROI face can capture clicks, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `true`, the ROI face captures mouse clicks. When `false`, the ROI face does not capture mouse clicks.

`HandleVisibility` — Visibility of ROI handle in `Children` property of parent
`'on'` (default) | `'off'` | `'callback'`

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI
`'all'` (default) | `'none'` | `'reshape'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable (default).
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area but not reshaped.
`'reshape'`	The ROI can be reshaped but not translated.

`Label` — ROI label
`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background
`1` (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color
`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'LabelTextColor','r'`

Example: `'LabelTextColor','green'`

Example: `'LabelTextColor',[0 0.4470 0.7410]`

`LabelVisible` — Label visibility

`'on'` (default) | `'off'` | `'hover'`

Label visibility, specified as one of these values.

Value	Description
`'on'`	Label is visible when the ROI is visible.
`'hover'`	Label is visible only when the mouse is hovering over the ROI.
`'off'`	Label is not visible.

`LineWidth` — Width of ROI border

positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

`MarkerSize` — Marker size
positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

`Multiclick` — Control freehand drawing style during interactive placement
`false` or `0` (default) | `true` or `1`

Control the freehand drawing style during interactive placement, specified as a numeric or logical `0` (`false`) or `1` (`true`). When the value is `false`, a single click and drag gesture completes the freehand ROI. When the value is `true`, multiple click and drag gestures can be combined with straight edges to make a more complex freehand ROI shape.

`Parent` — ROI parent
`Axes` object | `UIAxes` object

ROI parent, specified as an `Axes` or `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

`Position` — Position of ROI
`[]` (default) | *n*-by-2 numeric matrix

Position of the ROI, specified as an *n*-by-2 numeric matrix where *n* is the number of vertices or points defining the ROI. Each row represents the [*x* *y*] coordinates of a vertex or point. The `Freehand` object generates these points as you draw the ROI shape interactively. To work with fewer points, use the `reduce` function.

`Selected` — Selection state of ROI
`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`). You can also set this property interactively. For example, clicking on the ROI selects the ROI and sets this property to `true`. Similarly, pressing the **Ctrl** key and clicking the ROI deselects the ROI and sets the value of this property to `false`.

`SelectedColor` — Color of ROI when selected
`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'SelectedColor','r'`

Example: `'SelectedColor','green'`

Example: `'SelectedColor',[0 0.4470 0.7410]`

`Smoothing` — Smoothing applied to edge of ROI
1 (default) | nonnegative number

Smoothing applied to the edge of ROI after interactive placement, specified as a nonnegative scalar. The `Freehand` object filters the x and y coordinates of the ROI using a Gaussian smoothing kernel with a default standard deviation of 1. The size of the Gaussian filter is `2*ceil(2*Smoothing)+1`.

You can see the smoothing effect only after completing the drawing. Changing the value of `Smoothing` after completing the drawing has no effect on the ROI.

`StripeColor` — Color of ROI stripe
`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

Tag — Tag to associate with ROI
`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI
any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `Freehand` object does not use this data.

`Visible` — ROI visibility
`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

`Waypoints` — Position point is waypoint
`[]` (default) | logical vector

Position point is waypoint, specified as a logical vector of the same length as the `Position` property. Elements in `Waypoints` with the value `true` identify points in the `Position` matrix that are waypoints. By default, the `Freehand` object generates all the points that define the ROI and only makes points at locations of increased curvature into waypoints. You can turn all the points, or some subset of points, into waypoints by using code similar to `roi.Waypoints(1:4:end) = true;`.

Waypoints appear as circular shapes on the ROI edge. You can use waypoints to reshape the ROI by clicking and dragging the waypoint with the mouse. Moving waypoints modifies the freehand-drawn region between the waypoint that you clicked and the adjacent waypoints.

Object Functions

addlistener	Create event listener bound to event source
beginDrawingFromPoint	Begin drawing ROI from specified point
bringToFront	Bring ROI to front of Axes stacking order
createMask	Create binary mask image from ROI
draw	Begin drawing ROI interactively
inROI	Query if points are located in ROI
reduce	Reduce density of points in ROI
wait	Block MATLAB command line until ROI operation is finished

Examples

Create Freehand ROI Non-interactively

Read an image into the workspace and display it.

```
I = imread('baby.jpg');
figure
imshow(I)
```

Create a freehand ROI on the image, using the `Position` property to specify the vertices of the ROI. For programmatically created ROIs, if you want the ROI drawn in a specific axes, you must specify that axes as an input argument. Otherwise, an instance of the `images.roi.Freehand` class is created but not displayed. In this example, specify the current axes (`gca`) to draw the ROI on the image in that axes.

```
h = images.roi.Freehand(gca,'Position',[100 150;200 250;300 350;150 450]);
```

Set Up Listeners for Freehand ROI Events

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a freehand ROI on the image. Because this example specifies the size and location of the ROI, you do not have to call the `draw` method to enable interactive drawing.

```
roi = images.roi.Freehand(gca,'Position',[10 15;100 50;100 150;15 145]);
```

Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays at the command line the previous position and the current position of the `Freehand` ROI.

```
function allevents(src,evt)
evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving Previous Position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving Current Position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved Previous Position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved Current Position: ' mat2str(evt.CurrentPosition)]);
    end

end
```

More About

Events

To receive notification from the ROI when certain events happen, set up listeners for these events. You can specify a callback function that executes when one of these events occurs. When the ROI notifies your application through the listener, it returns data specific to the event. Look at the event class for the specific event to see what is returned.

For an example, see “Set Up Listeners for Freehand ROI Events” on page 1-2849.

Event Name	Trigger	Event Data	Event Attributes

`DeletingROI`	ROI is about to be interactively deleted.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingStarted`	ROI is about to be interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingFinished`	ROI has been interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`MovingROI`	ROI shape or location is being interactively changed.	`images.roi.ROIMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIMoved`	ROI shape or location has been interactively changed.	`images.roi.ROIMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIClicked`	ROI has been clicked.	`images.roi.ROIClickedEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`AddingWaypoint`	A waypoint is about to be interactively added to the ROI.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

Event Name	Trigger	Event Data	Event Attributes

`WaypointAdded`	A waypoint has been interactively added to the ROI.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`RemovingWaypoint`	A waypoint is about to be interactively removed from the ROI.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`WaypointRemoved`	A waypoint has been interactively removed from the ROI.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

Tips

- To draw the ROI interactively using the `draw` or `drawfreehand` function, position the cursor on the image and click and drag to draw the ROI shape. To finish drawing the ROI, release the mouse button.
- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Finish drawing (close) the ROI.	Double-click, which adds a point at the pointer position and draws a line connecting this point to the first point drawn, closing the ROI. Right-click, which draws a line connecting the last point to the first point drawn. Position the pointer over the first point and click. Press **Enter**, which draws a line connecting the last point to the first point drawn.
Resize (reshape) the ROI.	Position pointer over a waypoint and then click and drag. No assistance (snapping to edges) is available in this mode.
Add a waypoint.	Position the pointer on an edge of the ROI, right-click, and select **Add Waypoint**. You can also position the pointer on an edge of the ROI and double-click.
Remove a waypoint.	Position the pointer on a waypoint, right-click, and select **Remove Waypoint**.

Behavior	Keyboard shortcut
Move the ROI.	Position the pointer over the ROI. The pointer changes to the fleur shape. Click and drag to move the ROI.
Delete an ROI.	Position the pointer on the ROI (not on a vertex), right-click, and select **Delete Freehand** from the context menu. You can also delete the ROI programmatically using the `delete` function.

- For information about using an ROI in an app created with App Designer, see “Using ROIs in Apps Created with App Designer”.

Compatibility Considerations

`UIContextMenu` property is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` property to assign a context menu to a graphics object or UI component is not recommended. Use the `ContextMenu` property instead. The property values are the same.

There are no plans to remove support for the `UIContextMenu` property at this time. However, the `UIContextMenu` property no longer appears in the list returned by calling the `get` function on a graphics object or UI component.

See Also

`drawfreehand` | `AssistedFreehand` | `Polygon` | `Polyline`

Topics

“Create ROI Shapes”

Introduced in R2018b

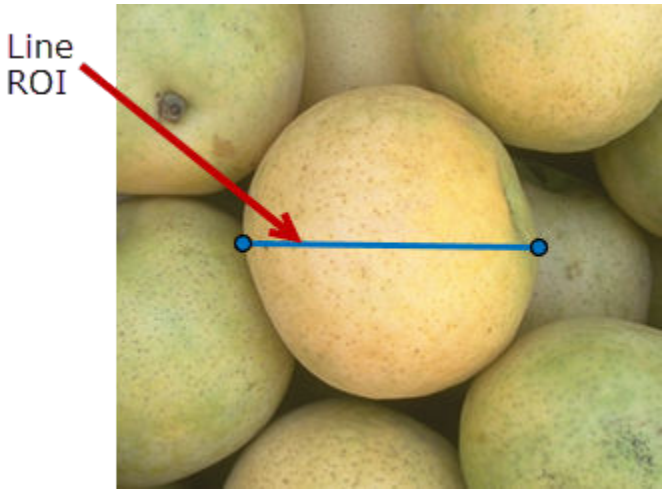
Line

Line region of interest

Description

A `Line` object specifies the length and position of a linear region-of-interest (ROI). You can customize the appearance and interactive behavior of the ROI.

For more information about using this ROI, including keyboard shortcuts, see “Tips” on page 1-2865.



Creation

There are two ways to create a `Line` object. For more information, see “Create ROI Shapes”.

- Use the `drawline` function. Use this function when you want to create the ROI and set the appearance in a single command. You can specify the length and position of the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments.
- Use the `images.roi.Line` function described here. Use this function when you want to specify the appearance and behavior of the ROI before you specify the length and position of the ROI. After creating the object, you can specify the length and position interactively by using the `draw` function or programmatically by modifying properties of the object.

Syntax

```
roi = images.roi.Line
roi = images.roi.Line(ax)
roi = images.roi.Line( ___ ,Name,Value)
```

Description

`roi = images.roi.Line` creates a `Line` object with default properties.

`roi = images.roi.Line(ax)` creates the ROI on the axes specified by `ax`.

`roi = images.roi.Line( ___ ,Name,Value)` sets properties on page 1-2855 of the ROI using name-value arguments. You can specify multiple name-value arguments. Enclose each property name in single quotes.

Example: `images.roi.Line('Color','y')` creates a yellow colored `Line` object.

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Properties

`Color` — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	

RGB Triplet	Appearance
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0.4470 0.7410]

ContextMenu — Context menu
ContextMenu object

Context menu that displays when you right-click the ROI, specified as a ContextMenu object. You can create a custom context menu by using the uicontextmenu function and then configuring context menu properties.

Deletable — Context menu provides option to delete the ROI
true or 1 (default) | false or 0

Context menu provides an option to delete the ROI, specified as a numeric or logical 1 (true) or 0 (false). When the value is true, you can delete the ROI interactively using the context menu. When the value is false, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the delete function.

DrawingArea — Area of axes in which you can interactively place ROI
'auto' (default) | 'unlimited' | [x,y,w,h]

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
'auto'	The drawing area is the current axes limits (default).
'unlimited'	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
[x,y,w,h]	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

HandleVisibility — Visibility of ROI handle in Children property of parent
'on' (default) | 'off' | 'callback'

Visibility of the ROI handle in the Children property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI

`'all'` (default) | `'none'` | `'reshape'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable (default).
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area but not reshaped.
`'reshape'`	The ROI can be reshaped but not translated.

`Label` — ROI label

`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background

`1` (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color

`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	

Color Name	Short Name	RGB Triplet	Appearance
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'LabelTextColor','r'`

Example: `'LabelTextColor','green'`

Example: `'LabelTextColor',[0 0.4470 0.7410]`

`LabelVisible` — Label visibility
`'on'` (default) | `'off'` | `'hover'`

Label visibility, specified as one of these values.

Value	Description
'on'	Label is visible when the ROI is visible.
'hover'	Label is visible only when the mouse is hovering over the ROI.
'off'	Label is not visible.

`LineWidth` — Width of ROI border
positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

`MarkerSize` — Marker size
positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

`Parent` — ROI parent
`Axes` object | `UIAxes` object

ROI parent, specified as an `Axes` or `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

`Position` — Position of ROI

`[]` (default) | 2-by-2 numeric matrix

Position of the ROI, specified as a 2-by-2 numeric matrix of the form `[x1 y1; x2 y2]`. Each row specifies the respective end-point of the line segment. You can also set this property by drawing or moving the line.

`Selected` — Selection state of ROI

`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`). You can also set this property interactively. For example, clicking on the ROI selects the ROI and sets this property to `true`. Similarly, pressing the **Ctrl** key and clicking the ROI deselects the ROI and sets the value of this property to `false`.

`SelectedColor` — Color of ROI when selected

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	

RGB Triplet	Appearance
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'SelectedColor','r'

Example: 'SelectedColor','green'

Example: 'SelectedColor',[0 0.4470 0.7410]

StripeColor — Color of ROI stripe
'none' (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or 'none'. If you specify 'none', then the ROI edge is a solid color specified by Color. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by Color and StripeColor.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

Tag — Tag to associate with ROI

`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

UserData — Data to associate with ROI

any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `Line` object does not use this data.

Visible — ROI visibility

`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Object Functions

addlistener	Create event listener bound to event source
beginDrawingFromPoint	Begin drawing ROI from specified point
bringToFront	Bring ROI to front of Axes stacking order
createMask	Create binary mask image from ROI
draw	Begin drawing ROI interactively
wait	Block MATLAB command line until ROI operation is finished

Examples

Create Linear ROI Non-interactively

Read an image into the workspace and display it.

```
I = imread('baby.jpg');
figure
imshow(I)
```

Create a linear ROI on the image, using the `Position` property to specify the ROI location. For programmatically created ROIs, if you want the ROI drawn in a specific axes, you must specify that axes as an input argument. Otherwise, an instance of the `images.roi.Line` class is created but not displayed. In this example, specify the current axes (`gca`) to draw the ROI on the image in that axes.

```
h = images.roi.Line(gca,'Position',[100 150;400 650]);
```

Set Up Listeners for Line ROI Events

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a Line ROI on the image. Because this example specifies the length and location of the ROI, you do not have to call the `draw` method to enable interactive drawing.

```
roi = images.roi.Line(gca,'Position',[10 15; 200 15]);
```

Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays at the command line the previous position and the current position of the ROI.

```
function allevents(src,evt)
evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving Previous Position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving Current Position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved Previous Position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved Current Position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

More About

Events

To receive notification from the ROI when certain events happen, set up listeners for these events. You can specify a callback function that executes when one of these events occurs. When the ROI notifies your application through the listener, it returns data specific to the event. Look at the event class for the specific event to see what is returned.

For an example, see “Set Up Listeners for Line ROI Events” on page 1-2863.

Event Name	Trigger	Event Data	Event Attributes

`DeletingROI`	ROI is about to be interactively deleted.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingStarted`	ROI is about to be interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingFinished`	ROI has been interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`MovingROI`	ROI shape or location is being interactively changed.	`images.roi.ROIMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIMoved`	ROI shape or location has been interactively changed.	`images.roi.ROIMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIClicked`	ROI has been clicked.	`images.roi.ROIClickedEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

Tips

- To draw the ROI interactively using the `draw` or `drawline` function, position the cursor on the axes and click and drag to draw the ROI shape. To finish the ROI, release the mouse button.
- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Make drawn line snap to 15 degree angles.	Hold the **Shift** key while drawing.
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Resize the ROI.	Position pointer over either endpoint and then click and drag to resize the ROI. Hold the **Shift** key while resizing to snap the line drawn at 15 degree angles.
Move the ROI.	Position the pointer over the ROI. The pointer changes to the fleur shape. Then click and drag the ROI.

- For information about using an ROI in an app created with App Designer, see “Using ROIs in Apps Created with App Designer”.

Compatibility Considerations

`UIContextMenu` property is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` property to assign a context menu to a graphics object or UI component is not recommended. Use the `ContextMenu` property instead. The property values are the same.

There are no plans to remove support for the `UIContextMenu` property at this time. However, the `UIContextMenu` property no longer appears in the list returned by calling the `get` function on a graphics object or UI component.

See Also

`drawline` | `Crosshair` | `Point` | `Polyline`

Topics

“Create ROI Shapes”

Introduced in R2018b

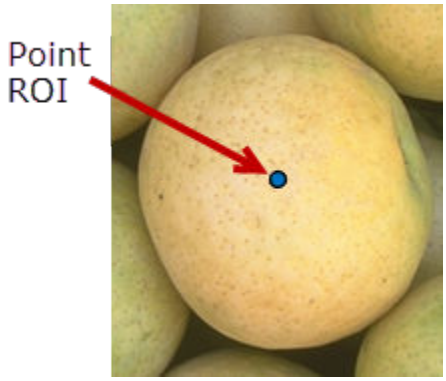
Point

Point region of interest

Description

A `Point` object specifies the position of a point region-of-interest (ROI). You can customize the appearance and interactive behavior of the ROI.

For more information about using the ROI, including keyboard shortcuts, see “Tips” on page 1-2878.



Creation

There are two ways to create a `Point` object. For more information, see “Create ROI Shapes”.

- Use the `drawpoint` function. Use this function when you want to create the ROI and set the appearance in a single command. You can specify the position of the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments.
- Use the `images.roi.Point` function described here. Use this function when you want to specify the appearance and behavior of the ROI before you specify the position of the ROI. After creating the object, you can specify the position interactively by using the `draw` function or programmatically by modifying properties of the object.

Syntax

```
roi = images.roi.Point
roi = images.roi.Point(ax)
roi = images.roi.Point( ___ ,Name,Value)
```

Description

`roi = images.roi.Point` creates a `Point` object with default properties.

`roi = images.roi.Point(ax)` creates an ROI object in the axes specified by `ax`.

`roi = images.roi.Point( ___ ,Name,Value)` sets properties on page 1-2868 of the ROI using name-value arguments. You can specify multiple name-value arguments. Enclose each property name in single quotes.

Example: `images.roi.Point('Color','y')` creates a yellow colored `Point` object.

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Properties

`Color` — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	

RGB Triplet	Appearance
[0.6350 0.0780 0.1840]	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0.4470 0.7410]

ContextMenu — Context menu
ContextMenu object

Context menu that displays when you right-click the ROI, specified as a ContextMenu object. You can create a custom context menu by using the uicontextmenu function and then configuring context menu properties.

Deletable — Context menu provides option to delete the ROI
true or 1 (default) | false or 0

Context menu provides an option to delete the ROI, specified as a numeric or logical 1 (true) or 0 (false). When the value is true, you can delete the ROI interactively using the context menu. When the value is false, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the delete function.

DrawingArea — Area of axes in which you can interactively place ROI
'auto' (default) | 'unlimited' | [x,y,w,h]

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
'auto'	The drawing area is the current axes limits (default).
'unlimited'	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
[x,y,w,h]	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

HandleVisibility — Visibility of ROI handle in Children property of parent
'on' (default) | 'off' | 'callback'

Visibility of the ROI handle in the Children property of the parent, specified as one of the values in this table.

Value	Description
'on'	The object handle is always visible (default).
'off'	The object handle is hidden at all times.

Value	Description
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI

`'all'` (default) | `'none'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable.
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area.

`Label` — ROI label

`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background

1 (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color

`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	

Color Name	Short Name	RGB Triplet	Appearance
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'LabelTextColor','r'

Example: 'LabelTextColor','green'

Example: 'LabelTextColor',[0 0.4470 0.7410]

LabelVisible — Label visibility
'on' (default) | 'off' | 'hover'

Label visibility, specified as one of these values.

Value	Description
'on'	Label is visible when the ROI is visible.
'hover'	Label is visible only when the mouse is hovering over the ROI.
'off'	Label is not visible.

LineWidth — Width of ROI border
positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

MarkerSize — Marker size
positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

Parent — ROI parent
Axes object | UIAxes object

ROI parent, specified as an Axes or UIAxes object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Position — Position of ROI
1-by-2 numeric vector

Position of the ROI, specified as a 1-by-2 numeric vector that represents the [*x* *y*] coordinates of the point. You can modify this property by drawing or moving the point.

`Selected` — Selection state of ROI

`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`). You can also set this property interactively. For example, clicking on the ROI selects the ROI and sets this property to `true`. Similarly, pressing the **Ctrl** key and clicking the ROI deselects the ROI and sets the value of this property to `false`.

`SelectedColor` — Color of ROI when selected

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'SelectedColor','r'`

Example: `'SelectedColor','green'`

Example: `'SelectedColor',[0 0.4470 0.7410]`

`StripeColor` — Color of ROI stripe

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

`Tag` — Tag to associate with ROI

`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI

any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `Point` object does not use this data.

`Visible` — ROI visibility

`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Object Functions

addlistener	Create event listener bound to event source
beginDrawingFromPoint	Begin drawing ROI from specified point
bringToFront	Bring ROI to front of Axes stacking order
draw	Begin drawing ROI interactively
wait	Block MATLAB command line until ROI operation is finished

Examples

Create Point ROI Non-interactively

Read an image into the workspace and display it.

```
I = imread('baby.jpg');
figure;
imshow(I)
```

Create a point ROI on the image, using the `'Position'` property to specify the location. For programmatically created ROIs, if you want the ROI drawn in a specific axes, you must specify that axes as an input argument. Otherwise, an instance of the `images.roi.Point` class is created but not displayed. In this example, specify the current axes (`gca`) to draw the ROI on the image in that axes.

```
h = images.roi.Point(gca,'Position',[400 650]);
```

Set Up Listeners for Point ROI Events

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a Point ROI on the image. Because this example specifies the length and location of the ROI, you do not have to call the `draw` method to enable interactive drawing.

```
roi = images.roi.Point(gca,'Position',[40 65]);
```

Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays at the command line the previous position and the current position of the ROI.

```
function allevents(src,evt)
evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving Previous Position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving Current Position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved Previous Position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved Current Position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

More About

Events

To receive notification from the ROI when certain events happen, set up listeners for these events. You can specify a callback function that executes when one of these events occurs. When the ROI notifies your application through the listener, it returns data specific to the event. Look at the event class for the specific event to see what is returned.

For an example, see “Set Up Listeners for Point ROI Events” on page 1-2876.

Event Name	Trigger	Event Data	Event Attributes

`DeletingROI`	ROI is about to be interactively deleted.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingStarted`	ROI is about to be interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingFinished`	ROI has been interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`MovingROI`	ROI shape or location is being interactively changed.	`images.roi.ROIMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIMoved`	ROI shape or location has been interactively changed.	`images.roi.ROIMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIClicked`	ROI has been clicked.	`images.roi.ROIClickedEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

Tips

- To draw the ROI interactively using the `draw` or `drawpoint` function, position the pointer on the image and then click and release.
- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Cancel drawing ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Move the ROI.	Position the pointer over the ROI. The pointer changes to a circle. Click and drag to move the ROI.
Delete the ROI.	Position the pointer over the point, right-click, and then choose **Delete Point** from the context menu. You can also delete the ROI programmatically using the `delete` function.

- For information about using an ROI in an app created with App Designer, see “Using ROIs in Apps Created with App Designer”.

Compatibility Considerations

`UIContextMenu` property is not recommended
Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` property to assign a context menu to a graphics object or UI component is not recommended. Use the `ContextMenu` property instead. The property values are the same.

There are no plans to remove support for the `UIContextMenu` property at this time. However, the `UIContextMenu` property no longer appears in the list returned by calling the `get` function on a graphics object or UI component.

See Also

`drawpoint` | `Circle` | `Crosshair`

Topics
“Create ROI Shapes”

Introduced in R2018b

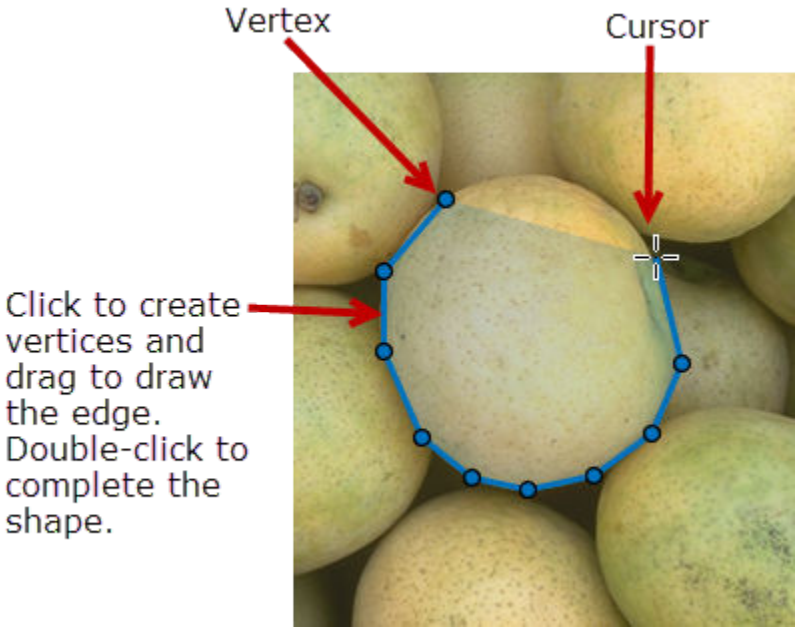
Polygon

Polygonal region of interest

Description

A `Polygon` object specifies the shape and position of a closed polygonal region-of-interest (ROI). You can customize the appearance and interactive behavior of the ROI.

For more information about using this ROI, including keyboard shortcuts, see `Tips`.



Creation

There are two ways to create a `Polygon` object. For more information, see “Create ROI Shapes”.

- Use the `drawpolygon` function. Use this function when you want to create the ROI and set the appearance in a single command. You can specify the shape and position of the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments.
- Use the `images.roi.Polygon` function described here. Use this function when you want to specify the appearance and behavior of the ROI before you specify the shape and position of the ROI. After creating the object, you can specify the shape and position interactively by using the `draw` function or programmatically by modifying properties of the object.

Syntax

```
roi = images.roi.Polygon
```

```
roi = images.roi.Polygon(ax)
roi = images.roi.Polygon( ___ ,Name,Value)
```

Description

`roi = images.roi.Polygon` creates a `Polygon` object with default properties.

`roi = images.roi.Polygon(ax)` creates the ROI in the axes specified by `ax`.

`roi = images.roi.Polygon( ___ ,Name,Value)` sets properties on page 1-2881 of the ROI using name-value arguments. You can specify multiple name-value arguments. Enclose each property name in single quotes.

Example: `images.roi.Polygon('Color','y')` creates a yellow colored `Polygon` object.

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Properties

Color — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0.4470 0.7410]

ContextMenu — Context menu
ContextMenu object

Context menu that displays when you right-click the ROI, specified as a ContextMenu object. You can create a custom context menu by using the uicontextmenu function and then configuring context menu properties.

Deletable — Context menu provides option to delete the ROI
true or 1 (default) | false or 0

Context menu provides an option to delete the ROI, specified as a numeric or logical 1 (true) or 0 (false). When the value is true, you can delete the ROI interactively using the context menu. When the value is false, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the delete function.

DrawingArea — Area of axes in which you can interactively place ROI
'auto' (default) | 'unlimited' | [x,y,w,h]

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
'auto'	The drawing area is the current axes limits (default).
'unlimited'	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
[x,y,w,h]	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

FaceAlpha — Transparency of ROI face
0.2 (default) | number in the range [0, 1]

Transparency of the ROI face, specified as a number in the range [0, 1]. When the value is `1`, the ROI face is completely opaque. When the value is `0`, the ROI face is completely transparent.

`FaceSelectable` — ROI face can capture clicks
`true` or `1` (default) | `false` or `0`

ROI face can capture clicks, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `true`, the ROI face captures mouse clicks. When `false`, the ROI face does not capture mouse clicks.

`HandleVisibility` — Visibility of ROI handle in `Children` property of parent
`'on'` (default) | `'off'` | `'callback'`

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI
`'all'` (default) | `'none'` | `'reshape'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable (default).
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area but not reshaped.
`'reshape'`	The ROI can be reshaped but not translated.

`Label` — ROI label
`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background
`1` (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color
`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'LabelTextColor','r'`

Example: `'LabelTextColor','green'`

Example: `'LabelTextColor',[0 0.4470 0.7410]`

`LabelVisible` — Label visibility

`'on'` (default) | `'off'` | `'hover'`

Label visibility, specified as one of these values.

Value	Description
`'on'`	Label is visible when the ROI is visible.
`'hover'`	Label is visible only when the mouse is hovering over the ROI.
`'off'`	Label is not visible.

`LineWidth` — Width of ROI border

positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

`MarkerSize` — Marker size

positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

`Parent` — ROI parent

`Axes` object | `UIAxes` object

ROI parent, specified as an `Axes` or `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

`Position` — Position of ROI

`[]` (default) | *n*-by-2 numeric matrix

Position of the ROI, specified as an *n*-by-2 numeric matrix where *n* is the number of vertices or points defining the ROI. Each row represents the [*x* *y*] coordinates of a vertex or point. The `Polygon` object generates these points as you draw the ROI shape interactively. To work with fewer points, use the `reduce` function.

`Selected` — Selection state of ROI

`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`). You can also set this property interactively. For example, clicking on the ROI selects the ROI and sets this property to `true`. Similarly, pressing the **Ctrl** key and clicking the ROI deselects the ROI and sets the value of this property to `false`.

`SelectedColor` — Color of ROI when selected

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	

Color Name	Short Name	RGB Triplet	Appearance
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'SelectedColor','r'

Example: 'SelectedColor','green'

Example: 'SelectedColor',[0 0.4470 0.7410]

StripeColor — Color of ROI stripe

'none' (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or 'none'. If you specify 'none', then the ROI edge is a solid color specified by Color. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by Color and StripeColor.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

Tag — Tag to associate with ROI
`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI
any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `Polygon` object does not use this data.

`Visible` — ROI visibility
`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Object Functions

addlistener	Create event listener bound to event source
beginDrawingFromPoint	Begin drawing ROI from specified point
bringToFront	Bring ROI to front of Axes stacking order
createMask	Create binary mask image from ROI
draw	Begin drawing ROI interactively
inROI	Query if points are located in ROI
reduce	Reduce density of points in ROI
wait	Block MATLAB command line until ROI operation is finished

Examples

Create Polygonal ROI Non-interactively

Read an image into the workspace and display it.

```
I = imread('baby.jpg');
figure;
imshow(I)
```

Create a polygonal ROI on the image, using the `Position` property to specify the vertices of the ROI. Note that you must specify the axes where you want to draw the ROI as the first argument.

```
h = images.roi.Polygon(gca,'Position',[100 150; 200 250; 300 350; 150 450]);
```

Set Up Listeners for Polygon ROI Events

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a Point ROI on the image. Because this example specifies the length and location of the ROI, you do not have to call the `draw` method to enable interactive drawing.

```
roi = images.roi.Polygon(gca,'Position',[115 30; 80 45; 80 80; 115 90; 145 65]);
```

Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays at the command line the previous position and the current position of the ROI.

```
function allevents(src,evt)
evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving Previous Position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving Current Position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved Previous Position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved Current Position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

More About

Events

To receive notification from the ROI when certain events happen, set up listeners for these events. You can specify a callback function that executes when one of these events occurs. When the ROI notifies your application through the listener, it returns data specific to the event. Look at the event class for the specific event to see what is returned.

For an example, see “Set Up Listeners for Polygon ROI Events” on page 1-2890.

Event Name	Trigger	Event Data	Event Attributes

`DeletingROI`	ROI is about to be interactively deleted.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingStarted`	ROI is about to be interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingFinished`	ROI has been interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`MovingROI`	ROI shape or location is being interactively changed.	`images.roi.ROIMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIMoved`	ROI shape or location has been interactively changed.	`images.roi.ROIMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIClicked`	ROI has been clicked.	`images.roi.ROIClickedEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`AddingVertex`	A vertex is about to be interactively added to the ROI.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

Event Name	Trigger	Event Data	Event Attributes

`VertexAdded`	A vertex has been interactively added to the ROI.	`event.EventData`	`NotifyAccess`: `private`
			
			`ListenAccess`: `public`

`DeletingVertex`	A vertex is about to be interactively removed from the ROI.	`event.EventData`	`NotifyAccess`: `private`
			
			`ListenAccess`: `public`

`VertexDeleted`	A vertex has been interactively removed from the ROI.	`event.EventData`	`NotifyAccess`: `private`
			
			`ListenAccess`: `public`

Tips

- To draw the ROI interactively using the `draw` or `drawpolygon` function, position the cursor on the axes, click and drag the pointer to create the shape. As you draw the line, click to create a vertex. Double-click to finish drawing and close the polygon.
- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Make drawn line snap at 15 degree angles.	Hold the **Shift** key while drawing.
Finish drawing (close) the ROI.	Double-click, which adds a new vertex at the pointer position and draws a line to the first vertex to close the polygon. Press **Enter**, which adds a new vertex at the pointer position and draws a line to the first vertex to close the polygon. Right-click, which does not add a new vertex but closes the polygon from the previous vertex. Position pointer over the first vertex and click.
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` field.
Add a new vertex to the ROI.	Position the pointer over the edge of the ROI and double-click. Position the pointer over the edge of the ROI, right-click, and select **Add Vertex** from the context menu.

Behavior	Keyboard shortcut
Remove the most recently added vertex but keep drawing.	Press **Backspace**. The function redraws the line from the previous vertex to the current position of the pointer. You can only back up to the first vertex you drew.
Resize (reshape) the ROI	Position pointer over a vertex and then click and drag. Add a new vertex to the polygon and then click and drag. Remove a vertex. The ROI redraws the line connecting the two neighboring vertices.
Move the ROI.	Position the pointer over the ROI. Hover over the edge of the polygon (not on a vertex). The pointer changes to the fleur shape. Click and drag to move the ROI.
Delete the ROI.	Position the pointer on the ROI, right-click, and choose **Delete Polygon** from the context menu. You can also delete the ROI programmatically using the `delete` object method.

- For information about using an ROI in an app created with App Designer, see “Using ROIs in Apps Created with App Designer”.
- To draw an open polygon ROI, use the `Polyline` object.

Compatibility Considerations

`UIContextMenu` property is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` property to assign a context menu to a graphics object or UI component is not recommended. Use the `ContextMenu` property instead. The property values are the same.

There are no plans to remove support for the `UIContextMenu` property at this time. However, the `UIContextMenu` property no longer appears in the list returned by calling the `get` function on a graphics object or UI component.

See Also

`drawpolygon` | `AssistedFreehand` | `Freehand` | `Polyline` | `Rectangle`

Topics

“Create ROI Shapes”

Introduced in R2018b

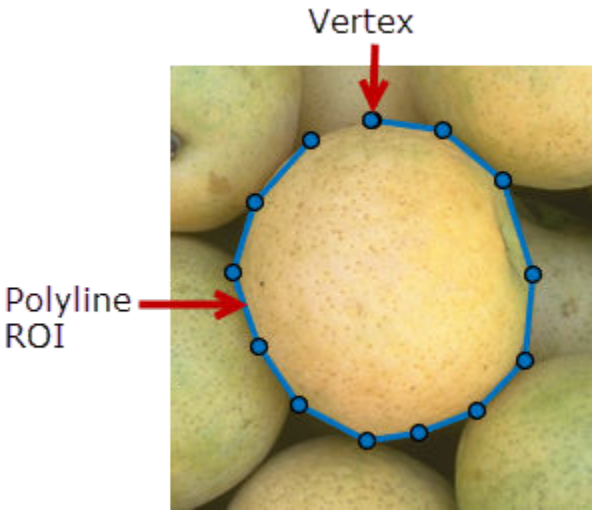
Polyline

Polyline region of interest

Description

A `Polyline` object specifies the shape and position of a polyline region-of-interest (ROI). You can customize the appearance and interactive behavior of the ROI.

For more information about using this ROI, including keyboard shortcuts, see “Tips” on page 1-2907.



Creation

There are two ways to create a `Polyline` object. For more information, see “Create ROI Shapes”.

- Use the `drawpolyline` function. Use this function when you want to create the ROI and set the appearance in a single command. You can specify the shape and position of the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments.
- Use the `images.roi.Polyline` function described here. Use this function when you want to specify the appearance and behavior of the ROI before you specify the shape and position of the ROI. After creating the object, you can specify the shape and position interactively by using the `draw` function or programmatically by modifying properties of the object.

Syntax

```
roi = images.roi.Polyline
roi = images.roi.Polyline(ax, ___ )
roi = images.roi.Polyline( ___ ,Name,Value)
```

Description

`roi = images.roi.Polyline` creates a `Polyline` object with default properties.

`roi = images.roi.Polyline(ax, ___ )` creates the ROI in the axes specified by `ax`.

`roi = images.roi.Polyline( ___ ,Name,Value)` sets properties on page 1-2896 of the ROI using name-value arguments. You can specify multiple name-value arguments. Enclose each property name in single quotes.

Example: `images.roi.Polyline('Color','y')` creates a yellow colored `Polyline` object.

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Properties

`Color` — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	

RGB Triplet	Appearance
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0.4470 0.7410]

ContextMenu — Context menu
ContextMenu object

Context menu that displays when you right-click the ROI, specified as a ContextMenu object. You can create a custom context menu by using the uicontextmenu function and then configuring context menu properties.

Deletable — Context menu provides option to delete the ROI
true or 1 (default) | false or 0

Context menu provides an option to delete the ROI, specified as a numeric or logical 1 (true) or 0 (false). When the value is true, you can delete the ROI interactively using the context menu. When the value is false, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the delete function.

DrawingArea — Area of axes in which you can interactively place ROI
'auto' (default) | 'unlimited' | [x,y,w,h]

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
'auto'	The drawing area is the current axes limits (default).
'unlimited'	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
[x,y,w,h]	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

HandleVisibility — Visibility of ROI handle in Children property of parent
'on' (default) | 'off' | 'callback'

Visibility of the ROI handle in the Children property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI

`'all'` (default) | `'none'` | `'reshape'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable (default).
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area but not reshaped.
`'reshape'`	The ROI can be reshaped but not translated.

`Label` — ROI label

`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background

`1` (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color

`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	

Color Name	Short Name	RGB Triplet	Appearance
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'LabelTextColor','r'`

Example: `'LabelTextColor','green'`

Example: `'LabelTextColor',[0 0.4470 0.7410]`

`LabelVisible` — Label visibility

`'on'` (default) | `'off'` | `'hover'`

Label visibility, specified as one of these values.

Value	Description
'on'	Label is visible when the ROI is visible.
'hover'	Label is visible only when the mouse is hovering over the ROI.
'off'	Label is not visible.

`LineWidth` — Width of ROI border

positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

`MarkerSize` — Marker size

positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

`Parent` — ROI parent

`Axes` object | `UIAxes` object

ROI parent, specified as an `Axes` or `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

`Position` — Position of ROI

`[ ]` (default) | *n*-by-2 numeric matrix

Position of the ROI, specified as an *n*-by-2 numeric matrix where *n* is the number of vertices or points defining the ROI. Each row represents the [*x* *y*] coordinates of a vertex or point. The `Polyline` object generates these points as you draw the ROI shape interactively. To work with fewer points, use the `reduce` function.

`Selected` — Selection state of ROI

`false` or `0` (default) | `true` or `1`

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`). You can also set this property interactively. For example, clicking on the ROI selects the ROI and sets this property to `true`. Similarly, pressing the **Ctrl** key and clicking the ROI deselects the ROI and sets the value of this property to `false`.

`SelectedColor` — Color of ROI when selected

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	

RGB Triplet	Appearance
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: 'SelectedColor','r'

Example: 'SelectedColor','green'

Example: 'SelectedColor',[0 0.4470 0.7410]

StripeColor — Color of ROI stripe
'none' (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or 'none'. If you specify 'none', then the ROI edge is a solid color specified by Color. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by Color and StripeColor.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

Tag — Tag to associate with ROI

`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

UserData — Data to associate with ROI

any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `Polyline` object does not use this data.

Visible — ROI visibility

`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Object Functions

addlistener	Create event listener bound to event source
beginDrawingFromPoint	Begin drawing ROI from specified point
bringToFront	Bring ROI to front of Axes stacking order
createMask	Create binary mask image from ROI
draw	Begin drawing ROI interactively
reduce	Reduce density of points in ROI
wait	Block MATLAB command line until ROI operation is finished

Examples

Create Polyline ROI Non-interactively

Read an image into the workspace and display it.

```
I = imread('baby.jpg');
figure;
imshow(I)
```

Create a polyline ROI on the image, using properties to specify the location of vertices. For programmatically created ROIs, if you want the ROI drawn in a specific axes, you must specify that axes as an input argument. Otherwise, an instance of the `images.roi.Polyline` class is created but not displayed. In this example, specify the current axes (`gca`) to draw the ROI on the image in that axes.

```
h = images.roi.Polyline(gca,'Position',[100 150; 200 250; 300 350; 150 450]);
```

Set Up Listeners for Polyline ROI Events

Read an image into the workspace.

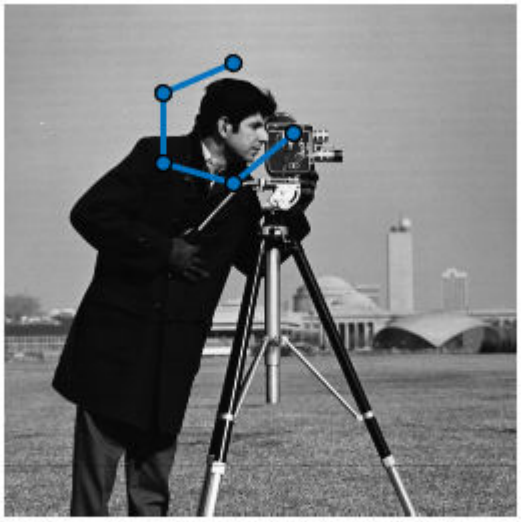
```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a Polyline ROI on the image. Because this example specifies the length and location of the ROI, you do not have to call the `draw` method to enable interactive drawing.

```
roi = images.roi.Polyline(gca,'Position',[115 30; 80 45; 80 80; 115 90; 145 65]);
```



Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays at the command line the previous position and the current position of the ROI.

```
function allevents(src,evt)
evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving Previous Position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving Current Position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved Previous Position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved Current Position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

More About

Events

To receive notification from the ROI when certain events happen, set up listeners for these events. You can specify a callback function that executes when one of these events occurs. When the ROI notifies your application through the listener, it returns data specific to the event. Look at the event class for the specific event to see what is returned.

For an example, see “Set Up Listeners for Polyline ROI Events” on page 1-2904.

Event Name	Trigger	Event Data	Event Attributes

`DeletingROI`	ROI is about to be interactively deleted.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingStarted`	ROI is about to be interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingFinished`	ROI has been interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`MovingROI`	ROI shape or location is being interactively changed.	`images.roi.ROIMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIMoved`	ROI shape or location has been interactively changed.	`images.roi.ROIMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIClicked`	ROI has been clicked.	`images.roi.ROIClickedEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`AddingVertex`	A vertex is about to be interactively added to the ROI.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

Event Name	Trigger	Event Data	Event Attributes

`VertexAdded`	A vertex has been interactively added to the ROI.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DeletingVertex`	A vertex is about to be interactively deleted from the ROI.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`VertexDeleted`	A vertex has been interactively deleted from the ROI.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

Tips

- To draw the ROI interactively using the `draw` or `drawpolyline` function, position the cursor on the axes and click and drag to create the shape. As you draw, click to place vertices along the line. Double-click to finish drawing the polyline.
- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Make drawn line snap at 15 degree angles.	Hold the **Shift** key while drawing.
Finish drawing the ROI.	Double-click, which adds a final new vertex at the pointer position. Right-click, which adds a final new vertex at the pointer position. Press **Enter**, which adds a final new vertex at the pointer position..
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Add a new vertex to the ROI.	Position the pointer over the polygon and double-click. You can also position the pointer over the ROI, right-click, and choose **Add Vertex**.
Remove a vertex from the ROI.	Position the pointer over the ROI, right-click, and choose **Delete Vertex**.
Remove the most recently added vertex but keep drawing.	Press **Backspace**. The function redraws the line from the previous vertex to the current position of the pointer. You can only back up to the first vertex you drew.

Behavior	Keyboard shortcut
Resize (reshape) the ROI.	Position pointer over a vertex and then click and drag. Add a new vertex and then click and drag. Remove a vertex and the shape of the ROI adjusts.
Move the ROI.	Position the pointer over the line, not on a vertex. The pointer changes to the fleur shape. Click and drag the polygon.
Delete the ROI.	Position the pointer over the line, right-click, and select **Delete Polyline** from the context menu. You can also delete the ROI programmatically using the `delete` function.

- For information about using an ROI in an app created with App Designer, see “Using ROIs in Apps Created with App Designer”.

Compatibility Considerations

`UIContextMenu` property is not recommended

Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` property to assign a context menu to a graphics object or UI component is not recommended. Use the `ContextMenu` property instead. The property values are the same.

There are no plans to remove support for the `UIContextMenu` property at this time. However, the `UIContextMenu` property no longer appears in the list returned by calling the `get` function on a graphics object or UI component.

See Also

`drawpolyline` | `Crosshair` | `Line` | `Polygon`

Topics

“Use Polyline to Create Angle Measurement Tool”
“Create ROI Shapes”

Introduced in R2018b

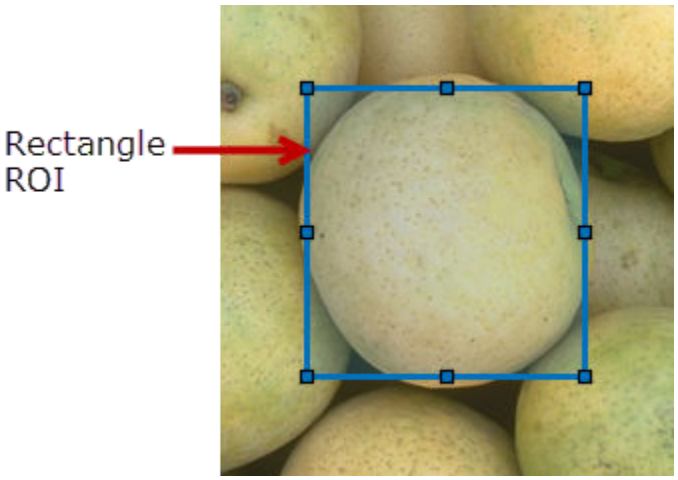
Rectangle

Rectangular region of interest

Description

A `Rectangle` object specifies the shape and position of a rectangular region-of-interest (ROI). You can customize the appearance and interactive behavior of the ROI.

For more information about using this ROI, including keyboard shortcuts, see “Tips” on page 1-2921.



Creation

There are two ways to create a `Rectangle` object. For more information, see “Create ROI Shapes”.

- Use the `drawrectangle` function. Use this function when you want to create the ROI and set the appearance in a single command. You can specify the shape and position of the ROI interactively by drawing the ROI over an image using the mouse, or programmatically by using name-value arguments.
- Use the `images.roi.Rectangle` function described here. Use this function when you want to specify the appearance and behavior of the ROI before you specify the shape and position of the ROI. After creating the object, you can specify the shape and position interactively by using the `draw` function or programmatically by modifying properties of the object.

Syntax

```
roi = images.roi.Rectangle
roi = images.roi.Rectangle(ax)
roi = images.roi.Rectangle( ___ ,Name,Value)
```

Description

`roi = images.roi.Rectangle` creates a `Rectangle` object with default properties.

`roi = images.roi.Rectangle(ax)` creates an ROI in the axes specified by `ax`.

`roi = images.roi.Rectangle( ___ ,Name,Value)` sets properties on page 1-2910 using name-value arguments. You can specify multiple name-value arguments. Enclose each property name in single quotes.

Example: `images.roi.Rectangle('Color','y')` creates a yellow colored `Rectangle` object.

Input Arguments

ax — Parent of ROI

`gca` (default) | `Axes` object | `UIAxes` object

Parent of ROI, specified as an `Axes` object or a `UIAxes` object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Properties

`AspectRatio` — Aspect ratio of rectangle

`1` (default) | positive number

Aspect ratio of the rectangle, specified as a positive number. The value of this property changes automatically when you draw or resize the rectangle. The `Rectangle` object calculates this value as `height/width`.

`Color` — ROI color

`[0 0.4470 0.7410]` (default) | RGB triplet | color name | short color name

ROI color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'Color','r'`

Example: `'Color','green'`

Example: `'Color',[0 0.4470 0.7410]`

`ContextMenu` — Context menu
`ContextMenu` object

Context menu that displays when you right-click the ROI, specified as a `ContextMenu` object. You can create a custom context menu by using the `uicontextmenu` function and then configuring context menu properties.

`Deletable` — Context menu provides option to delete the ROI
`true` or `1` (default) | `false` or `0`

Context menu provides an option to delete the ROI, specified as a numeric or logical `1` (`true`) or `0` (`false`). When the value is `true`, you can delete the ROI interactively using the context menu. When the value is `false`, the context menu option to delete the ROI is disabled.

In both cases, you can delete the ROI outside of the context menu by using the `delete` function.

`DrawingArea` — Area of axes in which you can interactively place ROI
`'auto'` (default) | `'unlimited'` | `[x,y,w,h]`

Area of the axes in which you can interactively place the ROI, specified as one of the values in this table.

Value	Description
`'auto'`	The drawing area is the current axes limits (default).
`'unlimited'`	The drawing area has no boundary and ROIs can be drawn or dragged to extend beyond the axes limits.
`[x,y,w,h]`	The drawing area is restricted to a rectangular region beginning at (*x*,*y*), and extending to width *w* and height *h*.

`FaceAlpha` — Transparency of ROI face
`0.2` (default) | number in the range [0, 1]

Transparency of the ROI face, specified as a number in the range [0, 1]. When the value is `1`, the ROI face is completely opaque. When the value is `0`, the ROI face is completely transparent.

`FaceSelectable` — ROI face can capture clicks
`true` or `1` (default) | `false` or `0`

ROI face can capture clicks, specified as a numeric or logical `1` (`true`) or `0` (`false`). When `true`, the ROI face captures mouse clicks. When `false`, the ROI face does not capture mouse clicks.

`FixedAspectRatio` — Aspect ratio remains constant
`false` or `0` (default) | `true` or `1`

Aspect ratio remains constant during interaction, specified as a numeric or logical `0` (`false`) or `1` (`true`). When the value is `true`, the aspect ratio remains constant when you draw or resize the ROI. When the value is `false`, you can change the aspect ratio when drawing or resizing the ROI. You can change the state of this property using the default context menu.

`HandleVisibility` — Visibility of ROI handle in `Children` property of parent
`'on'` (default) | `'off'` | `'callback'`

Visibility of the ROI handle in the `Children` property of the parent, specified as one of the values in this table.

Value	Description
`'on'`	The object handle is always visible (default).
`'off'`	The object handle is hidden at all times.
`'callback'`	The object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line.

`InteractionsAllowed` — Interactivity of ROI
`'all'` (default) | `'none'` | `'translate'`

Interactivity of the ROI, specified as one of the values in this table.

Value	Description
`'all'`	The ROI is fully interactable.
`'none'`	The ROI is not interactable, and no drag points are visible.
`'translate'`	The ROI can be translated (moved) within the drawing area.

`Label` — ROI label
`''` (default) | character vector | string scalar

ROI label, specified as a character vector or string scalar. By default, the ROI has no label (`''`).

`LabelAlpha` — Transparency of text background
`1` (default) | number in the range [0, 1]

Transparency of the text background, specified as a number in the range [0, 1]. When set to `1`, the text background is completely opaque. When set to `0`, the text background is completely transparent.

`LabelTextColor` — Label text color

`'black'` (default) | RGB triplet | color name | short color name

Label text color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'LabelTextColor','r'`

Example: `'LabelTextColor','green'`

Example: `'LabelTextColor',[0 0.4470 0.7410]`

`LabelVisible` — Visibility of ROI label

`'on'` (default) | `'off'` | `'hover'` | `'inside'`

Visibility of the ROI label, specified as one of these values:

Value	Description
`'on'`	Label is visible when the ROI is visible and the `Label` property is nonempty (default).

Value	Description
'hover'	Label is visible only when the mouse hovers over the ROI.
'inside'	Label is visible only when there is adequate space inside the ROI to display it.
'off'	Label is not visible.

LineWidth — Width of ROI border
positive number

Width of the ROI border, specified as a positive number in points. The default value is three times the number of points per screen pixel, such that the border is three pixels wide.

MarkerSize — Marker size
positive number

Marker size, specified as a positive number in points. The default value is eight times the number of points per screen pixel, such that markers are eight pixels in size.

Parent — ROI parent
Axes object | UIAxes object

ROI parent, specified as an Axes or UIAxes object. For information about using an ROI in a UIAxes, including important limitations, see “Using ROIs in Apps Created with App Designer”.

Position — Position of ROI
1-by-4 numeric vector

Position of the ROI, specified as a 1-by-4 numeric vector of the form [xmin, ymin, width, height]. xmin and ymin specify the location of the upper left corner of the rectangle. width and height specify the extent to the rectangle in two dimensions.

Rotatable — Rectangle can be rotated
false or 0 (default) | true or 1

Rectangle can be rotated, specified as a numeric or logical 0 (false) or 1 (true). When the value is false (default), the rectangle cannot be rotated. When the value is true, you can rotate the rectangle by clicking near the markers at the corners.

RotationAngle — Angle around center of rectangle
0 (default) | numeric scalar

Angle around the center of the rectangle, specified as a numeric scalar. The angle is measured in degrees in a clockwise direction. The value of this property changes automatically when you draw or move the ROI.

The value of RotationAngle does not impact the values in the Position property. The Position property represents the initial position of the ROI, before rotation. To determine the location of a rotated ROI, use the Vertices property.

Selected — Selection state of ROI
false or 0 (default) | true or 1

Selection state of the ROI, specified as a numeric or logical `0` (`false`) or `1` (`true`). You can also set this property interactively. For example, clicking on the ROI selects the ROI and sets this property to `true`. Similarly, pressing the **Ctrl** key and clicking the ROI deselects the ROI and sets the value of this property to `false`.

`SelectedColor` — Color of ROI when selected

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI when selected, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the value of `Color` defines the color of the ROI for all states, selected or not.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'SelectedColor','r'`

Example: `'SelectedColor','green'`

Example: `'SelectedColor',[0 0.4470 0.7410]`

`StripeColor` — Color of ROI stripe

`'none'` (default) | RGB triplet | color name | short color name

Color of the ROI stripe, specified as an RGB triplet, a color name, a short color name, or `'none'`. If you specify `'none'`, then the ROI edge is a solid color specified by `Color`. Otherwise, the edge of the ROI is striped, with colors alternating between the colors specified by `Color` and `StripeColor`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'StripeColor','r'`

Example: `'StripeColor','green'`

Example: `'StripeColor',[0 0.4470 0.7410]`

Tag — Tag to associate with ROI
`''` (default) | character vector | string scalar

Tag to associate with the ROI, specified as a character vector or string scalar. Use the tag value to find the ROI object in a hierarchy of objects using the `findobj` function.

`UserData` — Data to associate with ROI
any MATLAB data

Data to associate with the ROI, specified as any MATLAB data. For example, you can specify a scalar, vector, matrix, cell array, string, character array, table, or structure. The `Rectangle` object does not use this data.

`Vertices` — Locations of points on edge of rectangle

4-by-2 numeric matrix

This property is read-only.

Locations of points on the corners of the ROI, returned as a 4-by-2 numeric matrix.

`Visible` — ROI visibility

`'on'` (default) | `'off'` | on/off logical value

ROI visibility, specified as `'on'` or `'off'`, or as a numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to true, and `'off'` is equivalent to false. The value is stored as an on/off logical value of type `OnOffSwitchState`.

Value	Description
`'on'`	Display the ROI.
`'off'`	Hide the ROI without deleting it. You still can access the properties of an invisible ROI.

Object Functions

addlistener	Create event listener bound to event source
beginDrawingFromPoint	Begin drawing ROI from specified point
bringToFront	Bring ROI to front of Axes stacking order
createMask	Create binary mask image from ROI
draw	Begin drawing ROI interactively
inROI	Query if points are located in ROI
wait	Block MATLAB command line until ROI operation is finished

Examples

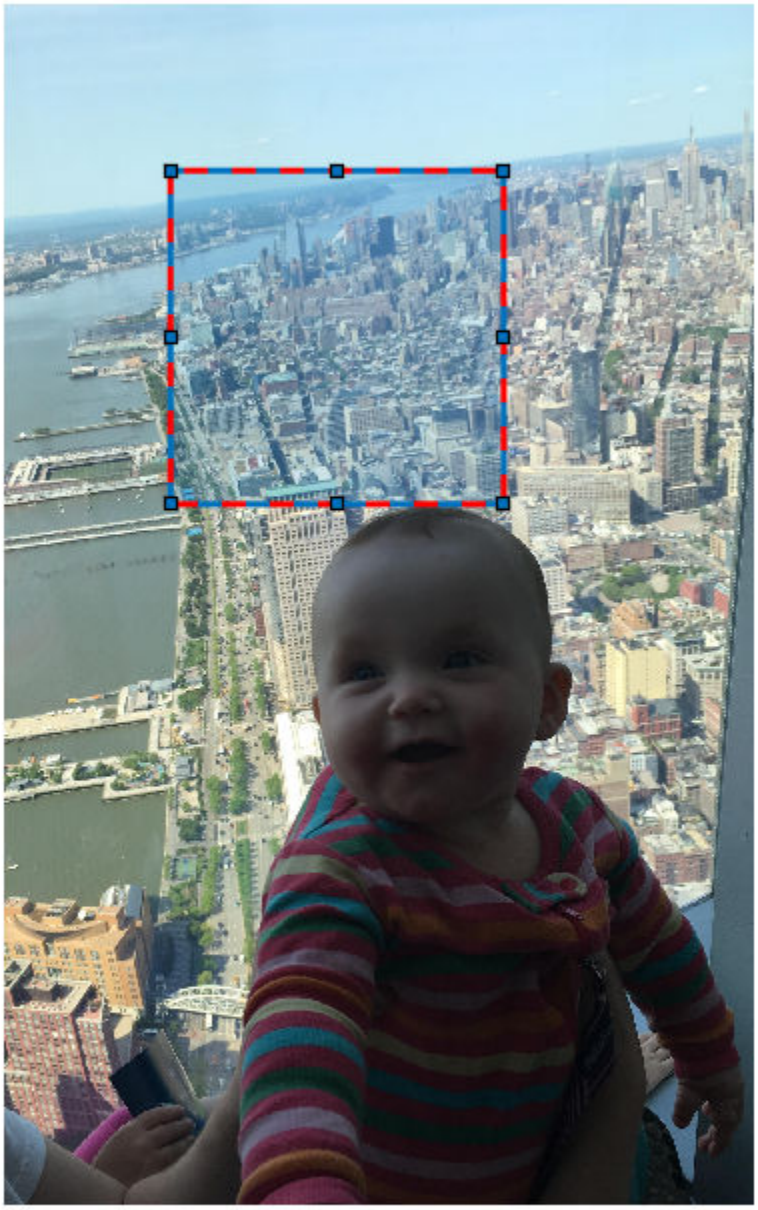
Create Rectangular ROI Non-interactively

Read image into the workspace and display it.

```
I = imread('baby.jpg');
figure
imshow(I)
```

Create a rectangular ROI on the image, using the `Position` parameter to specify its location and size. The example also specifies that the edge of the rectangle is a striped line. For programmatically created ROIs, if you want the ROI drawn in a specific axes, you must specify that axes as an input argument. Otherwise, an instance of the `images.roi.Rectangle` class is created but not displayed. In this example, specify the current axes (`gca`) to draw the ROI on the image in that axes.

```
h = images.roi.Rectangle(gca,'Position',[500,500,1000,1000],'StripeColor','r');
```



Set Up Listeners for Rectangle ROI Events

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a Rectangle ROI on the image. Because this example specifies the length and location of the ROI, you do not have to call the `draw` method to enable interactive drawing.

```
roi = images.roi.Rectangle(gca,'Position',[80,45,80,100]);
```

Set up listeners for ROI moving events. When you move it, the ROI sends notifications of these events and executes the callback function you specify.

```
addlistener(roi,'MovingROI',@allevents);
addlistener(roi,'ROIMoved',@allevents);
```

The `allevents` callback function displays at the command line the previous position and the current position of the ROI.

```
function allevents(src,evt)
evname = evt.EventName;
    switch(evname)
        case{'MovingROI'}
            disp(['ROI moving Previous Position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moving Current Position: ' mat2str(evt.CurrentPosition)]);
        case{'ROIMoved'}
            disp(['ROI moved Previous Position: ' mat2str(evt.PreviousPosition)]);
            disp(['ROI moved Current Position: ' mat2str(evt.CurrentPosition)]);
    end
end
```

More About

Events

To receive notification from the ROI when certain events happen, set up listeners for these events. You can specify a callback function that executes when one of these events occurs. When the ROI notifies your application through the listener, it returns data specific to the event. Look at the event class for the specific event to see what is returned.

For an example, see “Set Up Listeners for Rectangle ROI Events” on page 1-2919.

Event Name	Trigger	Event Data	Event Attributes

`DeletingROI`	ROI is about to be interactively deleted.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingStarted`	ROI is about to be interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`DrawingFinished`	ROI has been interactively drawn.	`event.EventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`MovingROI`	ROI shape or location is being interactively changed.	`images.roi.RectangleMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIMoved`	ROI shape or location has been interactively changed.	`images.roi.RectangleMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`ROIClicked`	ROI has been clicked.	`images.roi.ROIClickedEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

Tips

- To draw the ROI interactively using the `draw` or `drawrectangle` function, position the cursor on the axes and click and drag to create the shape. To finish the ROI, release the mouse button.
- The ROI supports the following interactivity, including keyboard shortcuts.

Behavior	Keyboard shortcut
Cancel drawing the ROI.	Press **Esc**. The function returns a valid ROI object with an empty `Position` property.
Resize (reshape) the ROI.	Position the cursor over a vertex and then click and drag. The rectangle has vertices at each corner and at the midpoint of each side. To preserve the aspect ratio while resizing, press the **Shift** key. To lock the aspect ratio, use the **Fix Aspect Ratio** in the right-click context menu.
Move the ROI.	Position the cursor over the ROI. The cursor changes to the fleur shape. Click and drag the ROI.
Delete the ROI.	Position the pointer on the rectangle, right-click, and choose **Delete Rectangle** from the context menu. You can also delete the ROI programmatically using the `delete` function.

- For information about using an ROI in an app created with App Designer, see “Using ROIs in Apps Created with App Designer”.

Compatibility Considerations

`UIContextMenu` property is not recommended
Not recommended starting in R2020a

Starting in R2020a, using the `UIContextMenu` property to assign a context menu to a graphics object or UI component is not recommended. Use the `ContextMenu` property instead. The property values are the same.

There are no plans to remove support for the `UIContextMenu` property at this time. However, the `UIContextMenu` property no longer appears in the list returned by calling the `get` function on a graphics object or UI component.

See Also

`drawrectangle` | `Cuboid` | `Polygon`

Topics
“Rotate Image Interactively Using Rectangle ROI”
“Create ROI Shapes”

Introduced in R2018b

beginDrawingFromPoint

Package: `images.roi`

Begin drawing ROI from specified point

Syntax

```
beginDrawingFromPoint(ROI,[x y])
beginDrawingFromPoint(ROI,[x y z])
beginDrawingFromPoint(ROI,[x y z],s)
beginDrawingFromPoint(ROI,[x y z],pos)
```

Description

`beginDrawingFromPoint(ROI,[x y])` enters interactive mode to draw the shape for object `ROI`. The drawing starts at location (*x*,*y*) in the axes. This method is intended to be used within the `ButtonDownFcn` callback of an `Image` or `Axes` object.

`beginDrawingFromPoint(ROI,[x y z])` enters interactive mode to draw a cuboidal `ROI` (`images.roi.Cuboid`). The drawing starts at location (*x*,*y*,*z*) in the axes.

`beginDrawingFromPoint(ROI,[x y z],s)` enters interactive mode to draw a cuboidal `ROI` (`images.roi.Cuboid`). The drawing starts at location (*x*,*y*,*z*) in the axes, snapping to the nearest location to the mouse from the `Scatter` object `s`.

`beginDrawingFromPoint(ROI,[x y z],pos)` enters interactive mode to draw a cuboidal `ROI` (`images.roi.Cuboid`). The drawing starts at location (*x*,*y*,*z*) in the axes, snapping to the nearest location to the mouse from the position specified by `pos`.

Examples

Draw Line ROI When Button Pressed

Create a new script called `sampleDrawLine.m`. Inside the script, copy and paste this code, then save the file.

```
hIm = imshow(imread('coins.png'));
hIm.ButtonDownFcn = @(~,~) buttonPressedCallback(hIm.Parent);

function buttonPressedCallback(hAx)
    cp = hAx.CurrentPoint;
    cp = [cp(1,1) cp(1,2)];
    obj = images.roi.Line('Parent',hAx,'Color',rand([1,3]));
    beginDrawingFromPoint(obj,cp);
end
```

Return to the MATLAB command window. Run the script by entering the command:

```
sampleDrawLine
```

The code opens a figure window containing an image of coins. Each time you click the mouse over the figure, the script executes the callback function, `buttonPressedCallback`. The callback function begins drawing a new ROI starting from the pixel you clicked.

Draw Cuboid ROI When Button Pressed

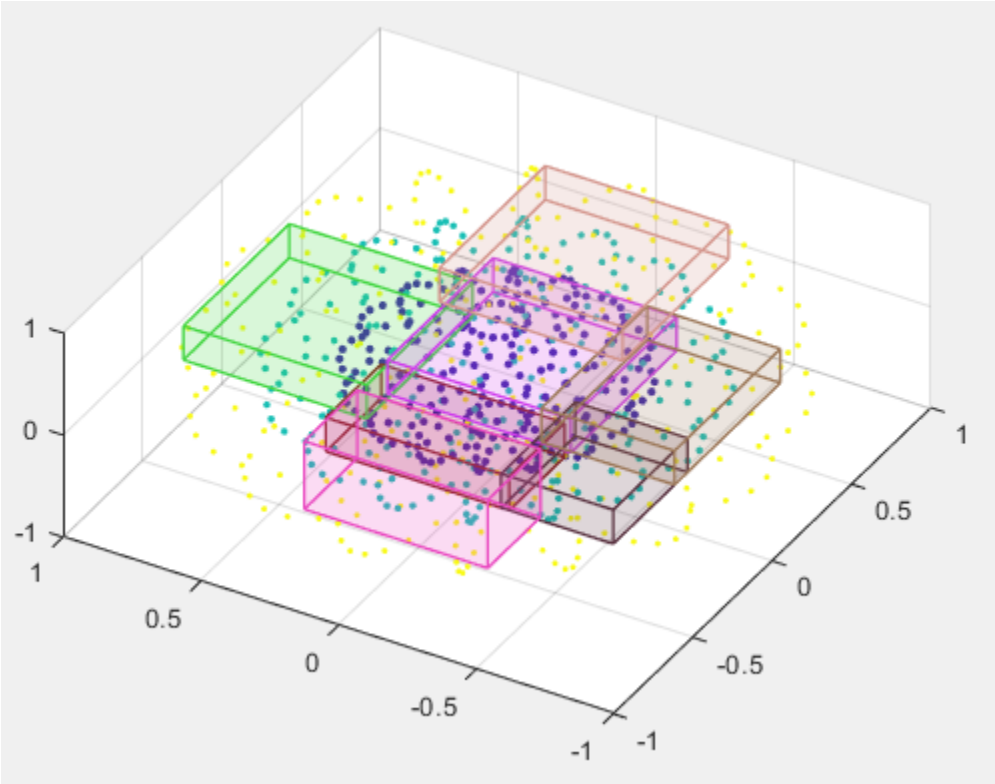
In the editor, open a file called `cuboidExample.m`. Copy and paste this code into the file and then save it.

```
function cuboidExample
    [x,y,z] = sphere(16);
    X = [x(:)*.5 x(:)*.75 x(:)];
    Y = [y(:)*.5 y(:)*.75 y(:)];
    Z = [z(:)*.5 z(:)*.75 z(:)];

    % Specify the size and color of each marker.
    S = repmat([1 .75 .5]*10,numel(x),1);
    C = repmat([1 2 3],numel(x),1);

    % Create a 3-D scatter plot
    figure
    hScatter = scatter3(X(:),Y(:),Z(:),S(:),C(:),'filled');
    view(-60,60);

    % Begin drawing cuboids when a scatter
    % point is clicked
    hScatter.ButtonDownFcn = @(~,~) buttonPressedCallback(hScatter.Parent);

    function buttonPressedCallback(hAx)
       cp = hAx.CurrentPoint;
       cp = cp(1,1:3);
       obj = images.roi.Cuboid('Parent',hAx,'Color',rand([1,3]));
       obj.beginDrawingFromPoint(cp);
    end
 end
```

Return to the MATLAB command window and run the function by entering the command:

```
cuboidExample
```

The code opens a figure window containing a scatter plot. Each time you click the mouse over the scatter plot, the function executes the callback function, `buttonPressedCallback`, and draws a new cuboidal ROI at the pixel you clicked.



Input Arguments

`ROI` — Region of interest

ROI object

Region of interest, specified as an ROI object of one of the following types:

`AssistedFreehand`	`Line`
`Circle`	`Point`
`Crosshair`	`Polygon`
`Cuboid`	`Polyline`
`Ellipse`	`Rectangle`
`Freehand`	

`[x y]` — Starting point in axes

numeric array

Starting point in the axes, specified as a numeric array.

`[x y z]` — Starting point in 3-D axes

numeric array

Starting point in 3-D axes, specified as a numeric array.

s — Scatter plot
`Scatter` object

Scatter plot, specified as a `matlab.graphics.chart.primitive.Scatter` object.

pos — Starting point in 3-D axes
N-by-3 numeric array

Starting point in 3-D axes, specified as an *N*-by-3 numeric array. Each row in `pos` represents a 3-D spatial location of a potential placement position.

See Also

`draw` | `drawassisted` | `drawcircle` | `drawcuboid` | `drawellipse` | `drawfreehand` | `drawline` | `drawpoint` | `drawpolygon` | `drawpolyline` | `drawrectangle`

Introduced in R2018b

bringToFront

Package: `images.roi`

Bring ROI to front of Axes stacking order

Syntax

```
bringToFront(ROI)
```

Description

`bringToFront(ROI)` moves the specified ROI to the front of the front-to-back visual stacking order of Axes children.

Use the `bringToFront` function when you want to bring a single ROI to the front of the visual stacking order. For other restacking behaviors, use the `uistack` function.

Examples

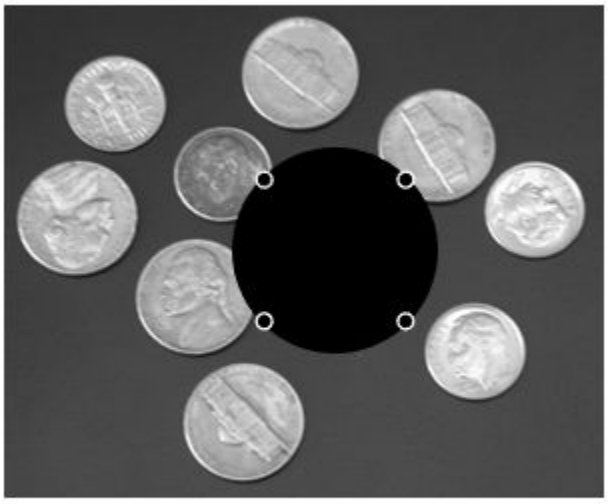
Change Stacking Order of ROIs

Read an image into the workspace and display it.

```
I = imread('coins.png');
imshow(I)
```

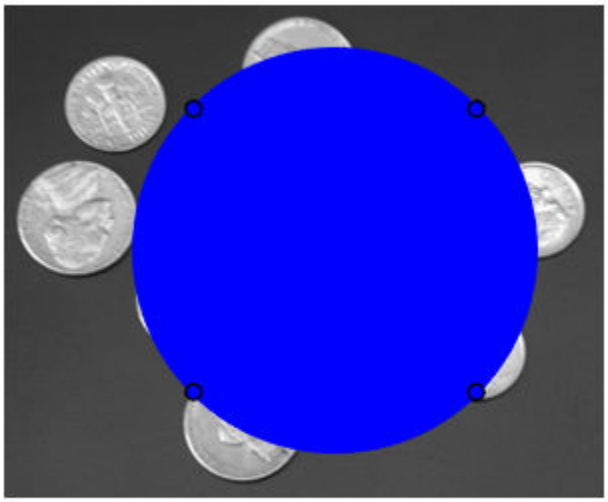
Create a circular ROI on the image, specifying where you want to place the circle and how wide to make the circle. To make it easy to see the changes to the stacking order, make the ROI opaque and specify the color black.

```
roi = images.roi.Circle(gca,'Center',[166 123],'Radius',50);
roi.FaceAlpha = 1.0;
roi.Color = 'black';
```



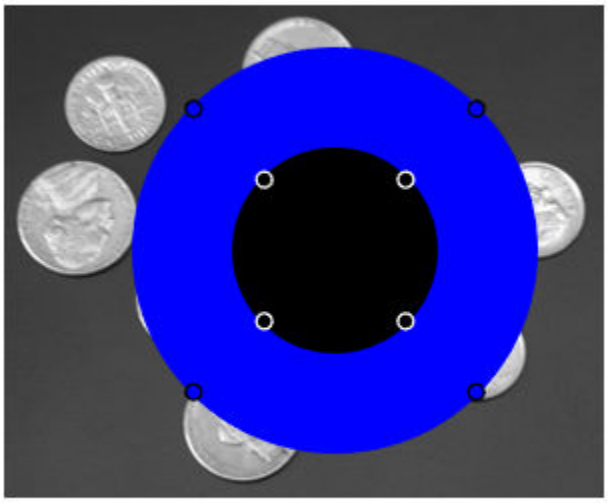
Create another circular ROI, specifying the same center point but make this ROI bigger. Again, to make the stacking order easy to see, make the ROI opaque and specify a different color, in this case, blue. This new ROI completely covers the first ROI.

```
roi2 = images.roi.Circle(gca,'Center',[166 123],'Radius',100);
roi2.FaceAlpha = 1.0;
roi2.Color = 'blue';
```



Bring the original ROI to the front by using the `bringToFront` function.

```
bringToFront(roi)
```



Input Arguments

ROI — Region of interest

ROI object

Region of interest, specified as an ROI object of one of these types:

`AssistedFreehand`	`Line`
`Circle`	`Point`
`Cuboid`	`Polygon`
`Ellipse`	`Polyline`
`Freehand`	`Rectangle`

See Also

`draw` | `drawassisted` | `drawcircle` | `drawellipse` | `drawfreehand` | `drawline` | `drawpoint` | `drawpolygon` | `drawpolyline` | `drawrectangle` | `uistack`

Topics

“Create ROI Shapes”

Introduced in R2019a

createMask

Package: `images.roi`

Create binary mask image from ROI

Syntax

```
bw = createMask(ROI)
bw = createMask(ROI,m,n)
bw = createMask(ROI,I)
bw = createMask(ROI,hImage)
```

Description

`bw = createMask(ROI)` returns a binary mask image with pixels inside the ROI set to `true` and pixels outside the ROI set to `false`.

`bw = createMask(ROI,m,n)` returns a binary mask image that is size `[m,n]`.

`bw = createMask(ROI,I)` returns a binary mask image that is the size of the image `I`.

`bw = createMask(ROI,hImage)` returns a binary mask image that is the size of the Image object `hImage`.

Examples

Create Mask From Ellipse ROI

Read image into the workspace and display it.

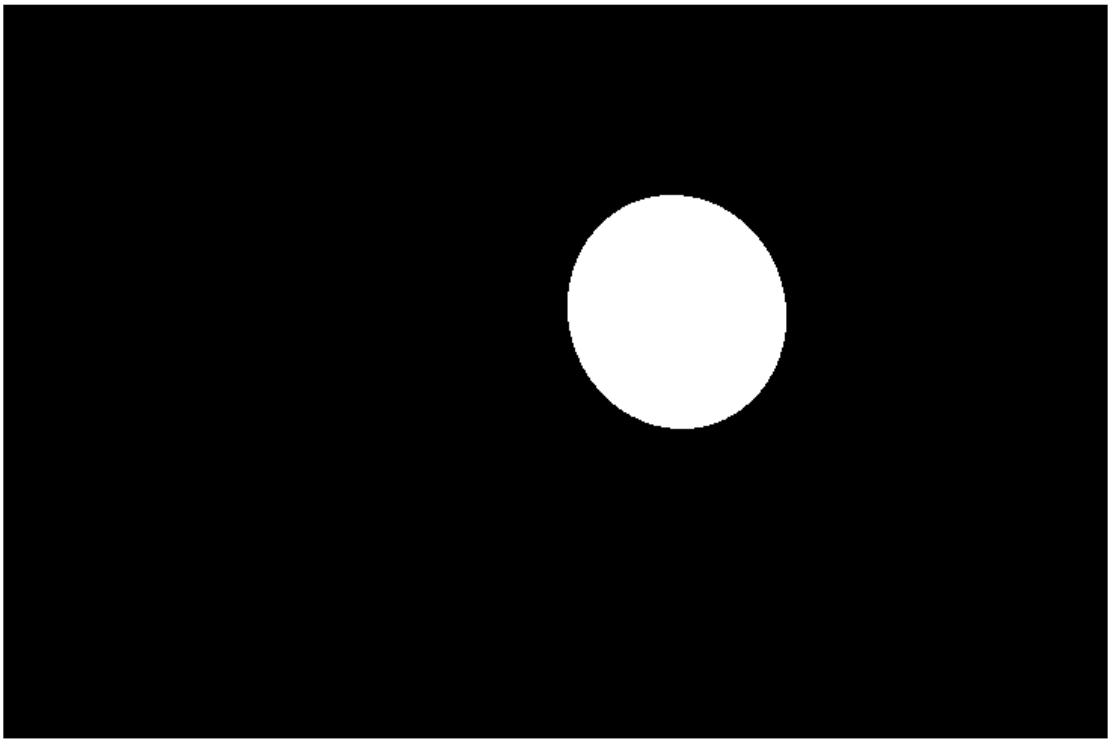
```
I = imread('pears.png');
imshow(I)
```

Draw an elliptical ROI on the image, using the `Center` parameter to specify the location of the ellipse and the`SemiAxes` parameter to specify the shape of the ellipse. The example also specifies that the edge of the ellipse is a striped line.

```
h = drawellipse('Center',[447 204],'SemiAxes',[78 72], ...
    'RotationAngle',287,'StripeColor','m');
```

Get a binary mask from the ROI. Pixels inside the ROI are `true` and pixels outside the ROI are `false`. Display the mask.

```
mask = createMask(h);
imshow(mask)
```



Input Arguments

`ROI` — Region of interest

ROI object

Region of interest, specified as an ROI object of one of the following types:

`AssistedFreehand`	`Freehand`	`Polyline`
`Circle`	`Line`	`Rectangle`
`Ellipse`	`Polygon`	

`m` — Row dimension of mask image

positive integer

Row dimension of the mask image, specified as a positive integer.

`n` — Column dimension of mask image

positive integer

Column dimension of the mask image, specified as a positive integer.

`I` — Input image array

numeric array

Input image array, specified as a numeric array.

`hImage` — Input image object
Image object

Input image object, specified as an Image object. For more details, see Image Properties.

Output Arguments

`bw` — Binary mask image
logical array

Binary mask image, returned as a logical array.

Data Types: `logical`

Tips

- For more information about classifying pixels on the ROI boundary, see “Classify Pixels That Are Partially Enclosed by ROI”.
- If you specify a mask size that does not match the size of the image associated with the `ROI` object, then `createMask` crops or zero-pads the mask to the specified size. The image associated with the ROI object is `ROI.Parent.Children(2,1).CData`.

See Also

`drawassisted` | `drawcircle` | `drawellipse` | `drawfreehand` | `drawline` | `drawpolygon` | `drawpolyline` | `drawrectangle` | `inROI`

Topics
“Create Binary Mask Using an ROI Function”
“Create ROI Shapes”

Introduced in R2018b

draw

Package: `images.roi`

Begin drawing ROI interactively

Syntax

```
draw(ROI)
draw(ROI,s)
draw(ROI,pos)
```

Description

`draw(ROI)` enters interactive mode to draw the shape for object `ROI` in the current axes (`gca`).

`draw(ROI,s)` enters interactive mode to draw the shape for an `Cuboid` object, snapping the ROI to the nearest location to the mouse from the `Scatter` object `s`.

`draw(ROI,pos)` enters interactive mode to draw the shape for the `Cuboid` object, snapping to the nearest location to the mouse from the position specified by `pos`. Specify `pos` as an *N*-by-3 numeric array where each row represents the (*x*,*y*,*z*) location of a potential placement position.

Examples

Draw ROI Interactively

Read an image into the workspace and display it.

```
I = imread('wagon.jpg');
figure
imshow(I);
```

Draw a triangular ROI on the image interactively. To improve the visibility of the ROI edge, specify a thick line width and bright cyan color of the ROI edge.

```
p = drawpolygon('LineWidth',7,'Color','cyan');
```

Get the coordinates of the vertices.

```
p.Position
```

```
ans =

  284.7500  725.5000
  331.2500  871.0000
  359.7500  707.5000
```

The spokes of the wheels define many other triangles. Suppose you want to get the vertices of a second triangle. You can use the `draw` function to start over and begin drawing a new polygonal ROI interactively. The line width and color parameters of the ROI are preserved.

```
draw(p)
```

```
p.Position
```

```
ans =
```

```
  398.7500  710.5000
  377.7500  865.0000
  461.7500  734.5000
```

Input Arguments

ROI — Region of interest

ROI object

Region of interest, specified as an ROI object of one of the following types:

`AssistedFreehand`	`Line`
`Circle`	`Point`
`Crosshair`	`Polygon`
`Cuboid`	`Polyline`
`Ellipse`	`Rectangle`
`Freehand`	

s — Scatter plot

`Scatter` object

Scatter plot, specified as a `matlab.graphics.chart.primitive.Scatter` object.

pos — Position of ROI

N-by-3 numeric array

Position of ROI, specified as an *N*-by-3 numeric array where each row represents the (*x*,*y*,*z*) location of a potential placement position.

See Also

`beginDrawingFromPoint` | `drawassisted` | `drawcircle` | `drawcuboid` | `drawellipse` | `drawfreehand` | `drawline` | `drawpoint` | `drawpolygon` | `drawpolyline` | `drawrectangle`

Introduced in R2018b

inROI

Package: `images.roi`

Query if points are located in ROI

Syntax

```
tf = inROI(ROI,x,y)
tf = inROI(ROI,x,y,z)
```

Description

`tf = inROI(ROI,x,y)` returns a logical array, `tf`, that indicates whether points with coordinates (`x`,`y`) are inside or outside the ROI.

`tf = inROI(ROI,x,y,z)` returns a logical array, `tf`, that indicates whether points with coordinates (`x`,`y`,`z`) are inside or outside the `Cuboid` ROI.

Examples

Query Points Inside Rectangular ROI

Read an image into the workspace and display it.

```
I = imread('trailer.jpg');
figure
imshow(I)
```

Draw a rectangular ROI on the image, using the `Position` argument to specify the position of the rectangle as `[xmin,ymin,width,height]`.

```
h = drawrectangle('Position',[190 308 682 276],'StripeColor','r');
```

Specify the *x*- and *y*-coordinates of three points. The last point is the upper left corner of the rectangular ROI.

```
xcoords = [335 335 190];
ycoords = [200 400 308];
```

Query if the three points are inside the ROI.

```
tf = inROI(h,xcoords,ycoords)
```

```
tf = 3x1 logical array

   0
   1
   1
```

Input Arguments

ROI — Region of interest

ROI object

Region of interest, specified as an ROI object of one of the following types:

`AssistedFreehand`	`Freehand`
`Circle`	`Polygon`
`Cuboid`	`Rectangle`
`Ellipse`	

x — *X*-coordinates of query points
numeric scalar or vector

X-coordinates of the query points, specified as a numeric scalar or vector.

y — *Y*-coordinates of query points
numeric scalar or vector

Y-coordinates of the query points, specified as a numeric scalar or vector.

z — *Z*-coordinates of query points
numeric scalar or vector

Y-coordinates of the query points, specified as a numeric scalar or vector.

Output Arguments

`tf` — Status of query points
logical array

Status of query points, returned as a logical array. The array is the same length as the input arrays `x`, `y`, and `z`. Elements of the logical array set to `true` indicate that the corresponding query point is inside the ROI. Elements that are `false` indicate the point is not inside the ROI.

See Also

`createMask` | `drawassisted` | `drawcircle` | `drawcuboid` | `drawellipse` | `drawfreehand` | `drawpolygon` | `drawrectangle`

Introduced in R2018b

reduce

Package: `images.roi`

Reduce density of points in ROI

Syntax

```
reduce(ROI)
reduce(ROI,tolerance)
```

Description

`reduce(ROI)` reduces the number of points that define the region-of-interest `ROI`. The ROI object stores the point array in the `Position` property. `reduce` replaces the original value of the `Position` property with the reduced value.

The `reduce` method calls the `reducepoly` function which uses the Ramer–Douglas–Peucker line simplification algorithm. This algorithm removes points along a straight line and leaves only knickpoints (points where the line curves).

`reduce(ROI,tolerance)` reduces the number of points that define the `ROI`, where `tolerance` specifies the sensitivity of the reduction. Specify the `tolerance` value in the range `[0,1]`.

Examples

Reduce Number of Points Used to Define Freehand ROI

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a `Freehand` ROI on the image.

```
roi = drawfreehand;
```

View the number of points in the `Position` property after completing the shape.

```
disp(['Original Size of Position property: ' mat2str(size(roi.Position))]);
```

```
Original Size of Position property: [272 2]
```

Use the `reduce` object function to reduce the number of points required to define the shape.

```
reduce(roi)
```

View the reduced number of points in the `Position` property.

```
disp(['Reduced Size of Position property: ' mat2str(size(roi.Position))]);
```

```
Reduced Size of Position property: [100 2]
```

Use Tolerance Parameter to Improve Results

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Display the image.

```
imshow(I);
```

Draw a `Polyline` ROI on the image.

```
roi = drawpolyline;
```

View the number of points in the `Position` property after completing the shape.

```
disp(['Original Size of Position property: ' mat2str(size(roi.Position))]);
```

```
Original Size of Position property: [12 2]
```

Use the `reduce` object function to reduce the number of points required to define the shape.

```
reduce(roi)
```

View the reduced number of points in the `Position` property.

```
disp(['First try at reducing the number of points: ' mat2str(size(roi.Position))]);
```

```
First try at reducing the number of points: [12 2]
```

Note that the number of points is not changed. To improve the result, change the Tolerance parameter. By default, tolerance is set to .01. Increase the value and try it again.

```
reduce(roi,0.3)
```

View the size of the `Position` property again. Changing the tolerance resulted in a reduction.

```
disp(['Reduction after resetting tolerance parameter: ' mat2str(size(roi.Position))]);
```

```
Reduction after resetting tolerance parameter: [4 2]
```

Input Arguments

`ROI` — ROI

`AssistedFreehand` object | `Freehand` object | `Polygon` object | `Polyline` object

ROI object, specified as one of the following ROI objects: `AssistedFreehand`, `Freehand`, `Polygon`, and `Polyline`.

`tolerance` — Sensitivity of reduction

`0.001` (default) | number in the range [0, 1]

Sensitivity of reduction, specified as a number in the range [0, 1]. Increasing the tolerance increases the number of points removed. A tolerance value of `0` reduces a minimum number of points. A tolerance value of `1` results in the maximum reduction in points, leaving only the end points of the line.

Algorithms

The Ramer–Douglas–Peucker line simplification algorithm recursively subdivides a shape looking to replace a run of points with a straight line. The algorithm checks that no point in the run deviates from the straight line by more than the value specified by `tolerance`.

See Also

`reducepoly` | `drawassisted` | `drawfreehand` | `drawpolygon` | `drawpolyline`

Topics

“Create ROI Shapes”

Introduced in R2019b

wait

Package: `images.roi`

Block MATLAB command line until ROI operation is finished

Syntax

```
wait(ROI)
```

Description

`wait(ROI)` blocks execution of the MATLAB command line until the operation on the ROI object `ROI` completes. Indicate completion by double-clicking the ROI object.

Examples

Use wait to Pause the Command Line

Read an image into the workspace.

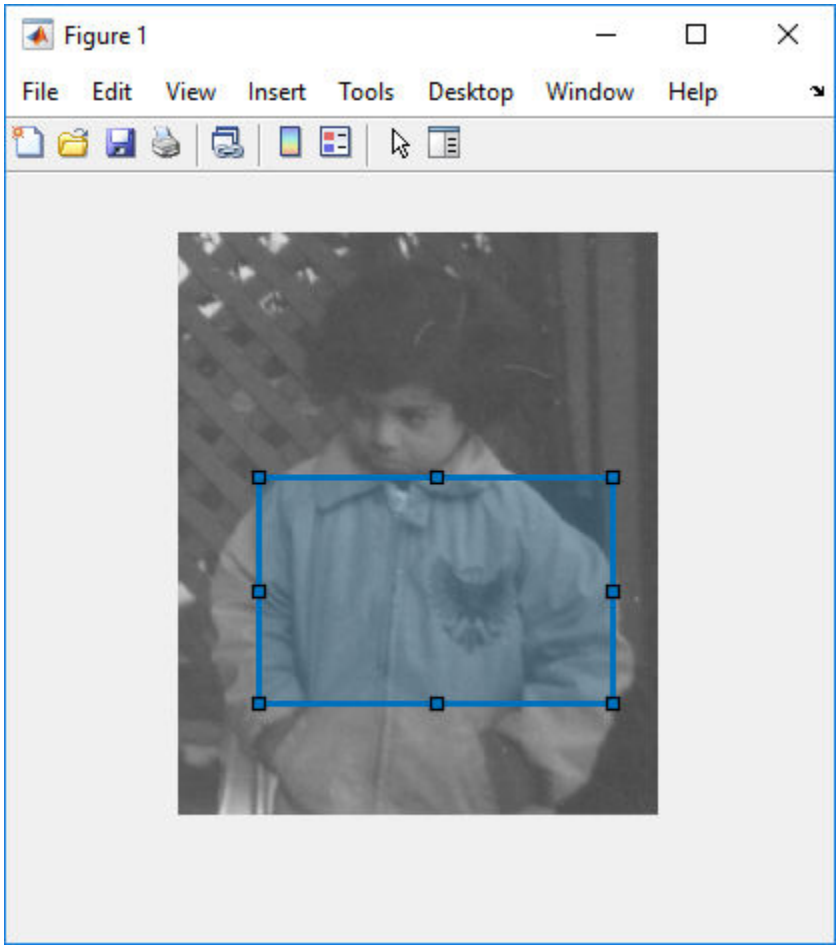
```
I = imread('pout.tif');
```

Display the image.

```
imshow(I)
```

Create an ROI on the axes. Click and drag the mouse to create the rectangular ROI.

```
roi = drawrectangle;
```



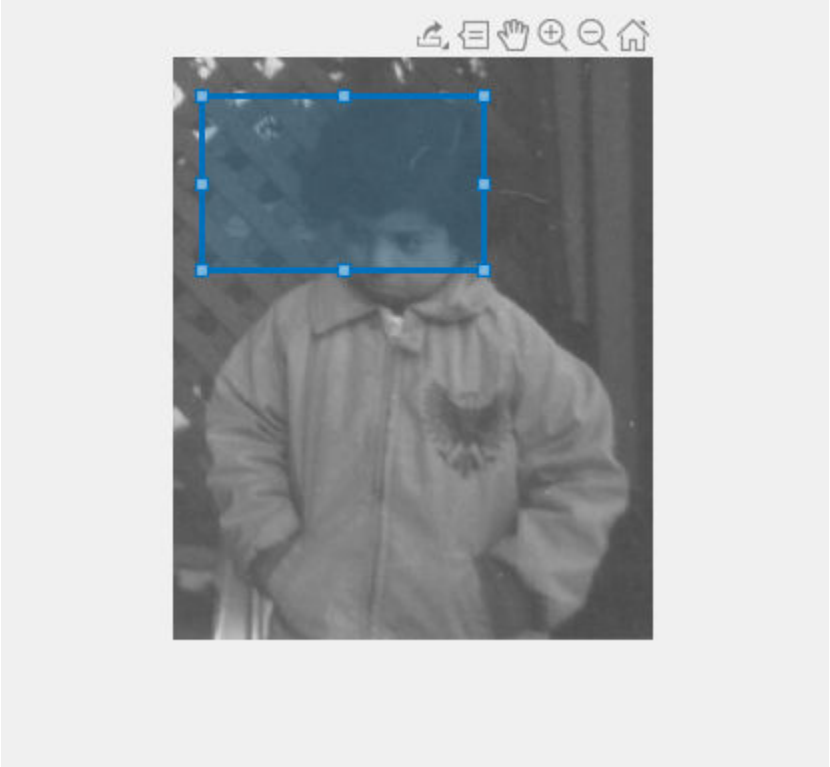
At the command line, view the value of the `Position` property of the ROI.

```
roi.Position
```

```
ans = 1×4

   57.0000  146.0000  141.0000   87.0000
```

Call the `wait` method of the ROI. This blocks the command line until an operation on the ROI completes. For example, you can move the ROI, reshape it, or rotate it (if you have enabled rotation). For this example, position the cursor inside the ROI, click and drag the ROI over the image to a new location. To indicate completeness, double-click the ROI.

```
wait(roi)
```



Back at the command line, check the value of the ROI's `Position` property. You can see that the values have changed to represent the new position.

```
roi.Position
```

```
ans = 1×4

   15.0000   20.0000  141.0000   87.0000
```

Input Arguments

`ROI` — Region of interest

ROI object

Region of interest, specified as an ROI object of one of the types listed in this table.

`AssistedFreehand`	`Line`
`Circle`	`Point`
`Crosshair`	`Polygon`
`Cuboid`	`Polyline`
`Ellipse`	`Rectangle`
`Freehand`	

See Also

`beginDrawingFromPoint` | `draw` | `inROI` | `createMask` | `bringToFront`

Topics

“Create ROI Shapes”
“Use Wait Function After Drawing ROI”

Introduced in R2019b

roicolor

Select region of interest (ROI) based on color

Syntax

```
BW = roicolor(I,low,high)
BW = roicolor(I,v)
```

Description

`BW = roicolor(I,low,high)` returns an ROI selected as those pixels in image `I` that lie within the range [`low high`]. The returned value, `BW`, is a binary image with `0`s outside the region of interest and `1`s inside.

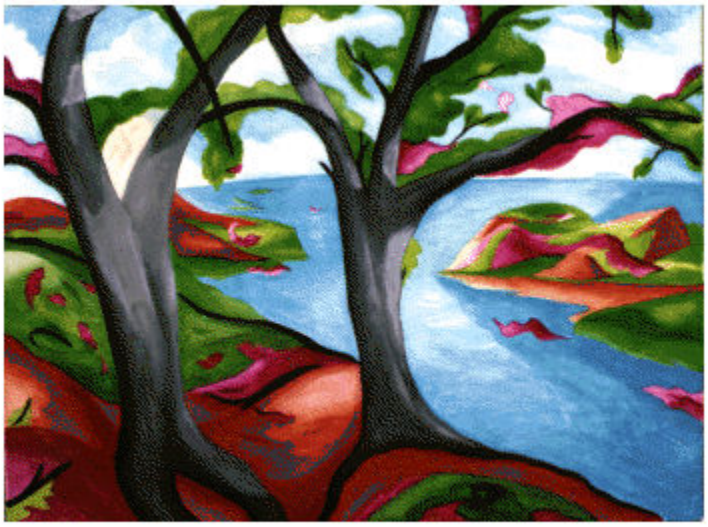
`BW = roicolor(I,v)` returns an ROI selected as those pixels in image `I` that match values in vector `v`.

Examples

Select Region of Interest Based on Color

Load an indexed image `X` with colormap `map`. The colormap has 128 colors. Display the indexed image.

```
load trees
imshow(X,map)
```



Create a binary mask image based on color. The mask is `true` for pixels with index in the range [10, 20]. The mask is `false` for pixels with index outside this range.

```
BW = roicolor(X,10,20);
```

Display the binary mask.

```
imshow(BW)
```

Input Arguments

`I` — Indexed or grayscale image

m-by-*n* numeric matrix

Indexed or grayscale image, specified as an *m*-by-*n* numeric matrix.

`low` — Minimum value

numeric scalar

Minimum value to include in the ROI, specified as a numeric scalar.

`high` — Maximum value

numeric scalar

Minimum value to include in the ROI, specified as a numeric scalar.

`v` — Set of values

numeric vector

Set of values to include in the ROI, specified as a numeric vector.

Output Arguments

BW — Binary image

m-by-*n* logical matrix

Binary image, returned as an *m*-by-*n* logical matrix.

Data Types: `logical`

Tips

- You can use the returned image as a mask for masked filtering using `roifilt2`.
- If you specify a colormap range, `[low high]`, then

  ```
  BW = (I >= low) & (I <= high)
  ```

- If you specify a set of colormap values, `v`, then the mask generated by `roicolor` is equivalent to:

  ```
  BW = ismember(I,v)
  ```

See Also

`roifilt2` | `roipoly` | `ismember`

Introduced before R2006a

roifill

(Not recommended) Fill in specified region of interest (ROI) polygon in grayscale image

Note `roifill` is not recommended. Use `regionfill` instead. If you want to define the polygon interactively with `regionfill`, then use `regionfill` either with `roipoly` or with `drawpolygon` followed by `createMask`.

Syntax

```
J = roifill
J = roifill(I)

J = roifill(I,mask)
J = roifill(I,xi,yi)
J = roifill(x,y,I,xi,yi)

[J,BW] = roifill( ___ )
[x2,y2,J,BW,xi2,yi2] = roifill( ___ )
roifill( ___ )
```

Description

Use the `roifill` function to fill in a specified region of interest (ROI) polygon in a grayscale image. `roifill` smoothly interpolates inward from the pixel values on the boundary of the polygon by solving Laplace's equation. The boundary pixels are not modified. `roifill` can be used, for example, to erase objects in an image.

`J = roifill` creates an interactive polygon selection tool associated with the image displayed in the current figure. With this syntax and the other interactive syntaxes, the polygon selection tool blocks the MATLAB command line until you complete the operation. `roifill` fills the selected polygon and returns the filled image, `J`.

For more information about using the polygon selection tool to define and fill ROIs, see “Interactive Behavior” on page 1-2957.

`J = roifill(I)` displays the image `I` in a figure window and creates an interactive polygon tool associated with the image.

`J = roifill(I,mask)` fills regions in `I` corresponding to the nonzero pixels in the mask. If there are multiple regions, then `roifill` performs the interpolation on each region independently.

`J = roifill(I,xi,yi)` fills in the polygon with vertices defined by X-Y coordinates `xi` and `yi` in the default spatial coordinate system.

`J = roifill(x,y,I,xi,yi)` defines a nondefault spatial coordinate system using the vectors `x` and `y`. The polygon vertices have coordinates `xi` and `yi` in this coordinate system.

`[J,BW] = roifill( ___ )` returns the binary image `BW` with `1`s for pixels corresponding to the interpolated region of `I` and `0`s elsewhere.

`[x2,y2,J,BW,xi2,yi2] = roifill( ___ )` also returns the image `XData` and `YData` in `x2` and `y2` and the polygon coordinates in `xi2` and `yi2`.

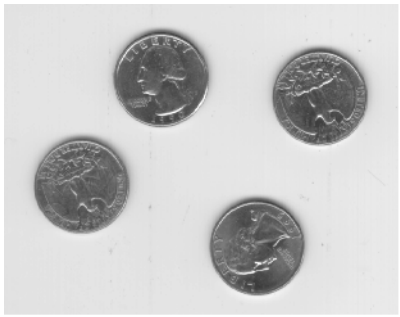
`roifill( ___ )` without an output argument displays the filled image in a new figure window.

Examples

Fill Region Using `roifill`

This example uses `roifill` to fill a region in the input image, `I`. For more examples, especially of the interactive syntaxes, see “Fill Region of Interest in an Image”.

```
I = imread('eight.tif');
c = [222 272 300 270 221 194];
r = [21 21 75 121 121 75];
J = roifill(I,c,r);
imshow(I)
figure
imshow(J)
```



Input Arguments

`I` — Grayscale image

numeric matrix

Grayscale image, specified as a numeric matrix.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`mask` — Mask

numeric matrix | logical matrix

Mask, specified as a numeric or logical matrix of the same size as the input image, `I`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`xi`, `yi` — *x*- or *y*-coordinate of vertices

numeric vector

x- or *y*-coordinate of polygon vertices, specified as numeric vectors of equal length. If you specify a nondefault coordinate system using the `x` and `y` arguments, then `xi` and `yi` specify coordinates in this coordinate system. Otherwise, `xi` and `yi` specify coordinates in the default coordinate system.

x, y — Image extent in world coordinates
2-element numeric vector

Image extent in world X-Y coordinates, specified as a 2-element numeric vector of the form `[min max]`. The two elements of `x` give the *x*-coordinates (horizontal) of the first and last columns of image `I`, respectively. The two elements of `y` give the *y*-coordinates (vertical) of the first and last rows of `I`.

Output Arguments

`J` — Filled image
numeric matrix

Filled image, returned as a numeric matrix.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`BW` — Binary image
logical matrix

Binary image, returned as a logical matrix of the same size as the input image, `I`.

Data Types: `logical`

`xi2, yi2` — *x*- or *y*-coordinate of vertices
numeric vector

x- or *y*-coordinate of polygon vertices, specified as numeric vectors. `xi` and `yi` are empty if you specify the polygon ROI using the `mask` argument.

x2, y2 — Image extent in world coordinates
2-element numeric vector

Image extent in world X-Y coordinates,, returned as 2-element numeric vectors of the form `[min max]`. If you specify image limits `x` and `y`, then `x2` and `y2` are equal to these values. Otherwise, `x2` and `y2` are equal to the original image `XData` and `YData`.

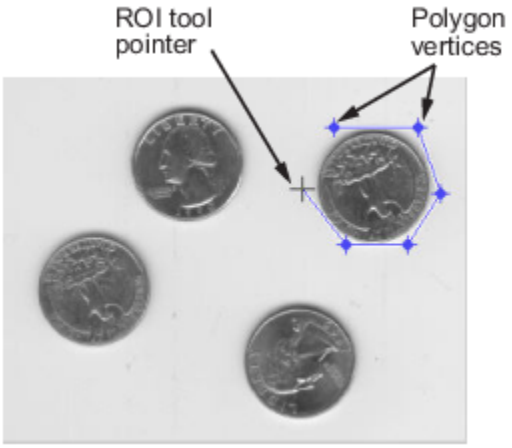
More About

Interactive Behavior

When you call `roifill` with an interactive syntax, a polygon selection tool opens that enables you to select and adjust polygon vertices interactively using the mouse.

When the polygon tool is active, the pointer changes to cross hairs  when you move the pointer over the image in the figure. Using the mouse, specify the region by selecting vertices of the polygon. You can move or resize the polygon using the mouse. When you are finished positioning and sizing the polygon, fill the polygon by double-clicking, or by right-clicking inside the region and selecting **Fill Area** from the context menu.

The figure illustrates a polygon defined by multiple vertices.



Interactive Behavior	Description
Closing the polygon. (Completing the region-of-interest.)	Use any of the following mechanisms: • Move the pointer over the initial vertex of the polygon that you selected. The shape changes to a circle ○. Click either mouse button. • Double-click the left mouse button. This action creates a vertex at the point under the mouse and draws a straight line connecting this vertex with the initial vertex. • Click the right mouse button. This action draws a line connecting the last vertex selected with the initial vertex; it does not create a new vertex.
Deleting the polygon	Press **Backspace**, **Escape** or **Delete**, or right-click inside the region and select **Cancel** from the context menu. Note: If you delete the ROI, the function returns empty values.
Moving the polygon	Move the pointer inside the region. The pointer changes to a fleur ✥. Click and drag the mouse to move the polygon.
Changing the color of the polygon	Move the pointer inside the region. Right-click and select **Set color** from the context menu.
Adding a new vertex.	Move the pointer over an edge of the polygon and press the **A** key. The shape of the pointer changes ✧. Click the left mouse button to create a new vertex at that position on the line.
Moving a vertex. (Reshaping the region-of-interest.)	Move the pointer over a vertex. The pointer changes to a circle ○. Click and drag the vertex to its new position.
Deleting a vertex.	Move the pointer over a vertex. The pointer changes to a circle ○. Right-click and select **Delete Vertex** from the context menu. This action deletes the vertex and adjusts the shape of the polygon, drawing a new straight line between the two vertices that were neighbors of the deleted vertex.

Interactive Behavior	Description
Retrieving the coordinates of the vertices	Move the pointer inside the region. Right-click and select **Copy position** from the context menu to copy the current position to the Clipboard. Position is an *n*-by-2 array containing the *x*- and *y*-coordinates of each vertex, where *n* is the number of vertices you selected.

See Also

`drawpolygon` | `Polygon` | `roifilt2` | `roipoly` | `regionfill` | `inpaintCoherent`

Introduced before R2006a

roifilt2

Filter region of interest (ROI) in image

Syntax

```
J = roifilt2(h,I,BW)
J = roifilt2(I,BW,fun)
```

Description

`J = roifilt2(h,I,BW)` filters regions of interest (ROIs) in the 2-D image `I` using the 2-D linear filter `h`. `BW` is a binary mask, the same size as `I`, that defines the ROIs in `I`. `roifilt2` returns an image that consists of filtered values for pixels in locations where `BW` contains `1`s, and unfiltered values for pixels in locations where `BW` contains `0`s.

`J = roifilt2(I,BW,fun)` processes the data in ROIs of `I` using the function `fun`. The value `fun` must be a function handle.

Examples

Filter Image Using Polygonal Mask

Read an image into the workspace.

```
I = imread('eight.tif');
```

Define the vertices of the mask polygon.

```
c = [222 272 300 270 221 194];
r = [21 21 75 121 121 75];
```

Create the binary mask image.

```
BW = roipoly(I,c,r);
```

Filter the region of the image `I` specified by the mask `BW`.

```
H = fspecial('unsharp');
J = roifilt2(H,I,BW);
```

Display the original image and the filtered image.

```
imshow(I)
```

```
figure
imshow(J)
```

Input Arguments

h — Linear filter

2-D numeric matrix

Linear filter, specified as a 2-D numeric matrix.

Data Types: `double`

`I` — Image
2-D numeric matrix

Image, specified as a 2-D numeric matrix.

- If you specify a filter, `h`, then `I` can be any of the listed data types.
- If you specify a function handle, `fun`, then `I` can be any class supported by `fun`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`BW` — Mask
2-D logical matrix | 2-D numeric matrix

Mask, specified as a 2-D logical matrix or a 2-D numeric matrix, the same size as `I`, containing `0`s and `1`s.

Data Types: `logical`

`fun` — Function handle
handle

Function handle, specified as a handle. For more information about function handles, see “Create Function Handle”.

Output Arguments

`J` — Filtered image
2-D matrix

Filtered image, returned as a 2-D matrix.

- If you specify a filter, `h`, then `J` has the same class as the input image, `I`.
- If you specify a function handle, `fun`, then the class of `J` is determined by `fun`.

Algorithms

If you specify a filter, `h`, then `roifilt2` calls `imfilter` to implement the filter.

See Also
`imfilter` | `filter2` | `roipoly`

Introduced before R2006a

roipoly

Specify polygonal region of interest (ROI)

Syntax

```
BW = roipoly
BW = roipoly(I)

BW = roipoly(I,xi,yi)
BW = roipoly(xref,yref,I,xi,yi)

[BW,xi2,yi2] = roipoly( ___ )
[xrefout,yrefout,BW,xi2,yi2] = roipoly( ___ )
roipoly( ___ )
```

Description

Create Polygon Interactively

`BW = roipoly` creates an interactive polygon tool associated with the image displayed in the current figure. `roipoly` returns the mask as a binary image, setting pixels inside the ROI to `1` and pixels outside the ROI to `0`.

With this syntax and the other interactive syntaxes, the polygon selection tool blocks the MATLAB command line until you complete the operation. For more information about using the polygon selection tool, see “Interactive Behavior” on page 1-2966.

`BW = roipoly(I)` displays the grayscale or RGB image `I` in a figure window and creates an interactive polygon selection tool associated with the image.

Create Polygon by Specifying Vertices

`BW = roipoly(I,xi,yi)` specifies the (*x*, *y*) coordinates of polygon vertices as (`xi`, `yi`).

`BW = roipoly(xref,yref,I,xi,yi)` specifies the coordinates of polygon vertices in the world coordinate system defined by `xref` and `yref`. The polygon vertices have (*x*, *y*) coordinates `xi` and `yi` in this coordinate system.

Specify Additional Output Options

`[BW,xi2,yi2] = roipoly( ___ )` also returns the coordinates of the vertices of the closed polygon, `xi2` and `yi2`. You can use the input arguments of any other syntax.

`[xrefout,yrefout,BW,xi2,yi2] = roipoly( ___ )` also returns the image limits in `xrefout` and `yrefout`.

`roipoly( ___ )` without output arguments displays the resulting mask image in a new figure window.

Examples

Create Polygonal Mask

Read an image into the workspace.

```
I = imread('eight.tif');
```

Define the vertices of the mask polygon.

```
c = [222 272 300 270 221 194];
r = [21 21 75 121 121 75];
```

Create the binary mask image.

```
BW = roipoly(I,c,r);
```

Display the original image and the polygonal mask.

```
imshow(I)
```

```
figure
imshow(BW)
```

Input Arguments

`I` — Grayscale or RGB image

m-by-*n* numeric matrix | *m*-by-*n*-by-3 numeric array

Grayscale or RGB image, specified as an *m*-by-*n* numeric matrix or an *m*-by-*n*-by-3 numeric array, respectively.

`xi` — *x*-coordinate of polygon vertices

numeric vector

x-coordinate of polygon vertices, specified as a numeric vector of the same length as `yi`. If you specify image limits in a world coordinate system using `xref`, then `xi` is in this coordinate system. Otherwise, `xi` is in the default coordinate system.

`yi` — *y*-coordinate of polygon vertices

numeric vector

y-coordinate of polygon vertices, specified as a numeric vector of the same length as `xi`. If you specify image limits in a world coordinate system using `yref`, then `yi` is in this coordinate system. Otherwise, `yi` is in the default coordinate system.

`xref` — Image limits in world coordinates along *x*-dimension

2-element numeric vector

Image limits in world coordinates along the *x*-dimension, specified as a 2-element numeric vector of the form [*xmin xmax*]. The value of `xref` sets the image `XData`.

`yref` — Image limits in world coordinates along *y*-dimension

2-element numeric vector

Image limits in world coordinates along the *y*-dimension, specified as a 2-element numeric vector of the form [*ymin ymax*]. The value of `yref` sets the image `YData`.

Output Arguments

`BW` — Binary image

m-by-*n* logical matrix

Binary image, returned as an *m*-by-*n* logical matrix.

Data Types: `logical`

`xi2` — *x*-coordinate of vertices

numeric vector

x-coordinate of vertices of the closed polygon, returned as a numeric vector of the same length as `yi2`. The first and last element in the vector are identical, so that the polygon is closed. If you specify image limits in a world coordinate system using `xref`, then `xi2` is in this coordinate system. Otherwise, `xi2` is in the default coordinate system.

`yi2` — *y*-coordinate of vertices

numeric vector

y-coordinate of vertices of the closed polygon, returned as a numeric vector of the same length as `xi2`. The first and last element in the vector are identical, so that the polygon is closed. If you specify image limits in a world coordinate system using `yref`, then `yi2` is in this coordinate system. Otherwise, `yi2` is in the default coordinate system.

`xrefout` — Image limits in world coordinates along *x*-dimension

2-element numeric vector

Image limits in world coordinates along the *x*-dimension, returned as a 2-element numeric vector of the form [*xmin xmax*]. If you specify image limits in a world coordinate system using `xref`, then `xrefout` is equal to `xref`. Otherwise, `xrefout` is equal to the original image `XData`.

`yrefout` — Image limits in world coordinates along *y*-dimension

2-element numeric vector

Image limits in world coordinates along the *y*-dimension, returned as a 2-element numeric vector of the form [*ymin ymax*]. If you specify image limits in a world coordinate system using `yref`, then `yrefout` is equal to `yref`. Otherwise, `yrefout` is equal to the original image `YData`.

More About

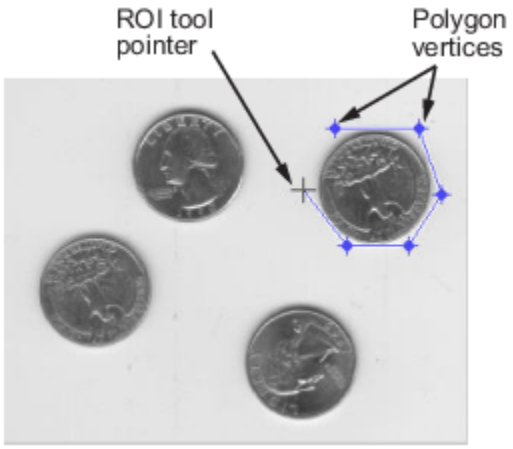
Interactive Behavior

The polygon selection tool enables you to select and adjust polygon vertices interactively using the mouse.

When the polygon tool is active, the pointer changes to cross hairs  when you move the pointer over the image in the figure. Using the mouse, you specify the region by selecting vertices of the polygon. You can move or resize the polygon using the mouse. When you are finished positioning and

sizing the polygon, create the mask by double-clicking, or by right-clicking inside the region and selecting **Create mask** from the context menu.

The figure illustrates a polygon defined by multiple vertices. The following table describes all the interactive behavior of the polygon tool.



Interactive Behavior	Description
Closing the polygon. (Completing the region-of-interest.)	Use any of the following mechanisms: • Move the pointer over the initial vertex of the polygon that you selected. The pointer changes to a circle ○. Click either mouse button. • Double-click the left mouse button. This action creates a vertex at the point under the mouse pointer and draws a straight line connecting this vertex with the initial vertex. • Right-click the mouse. This draws a line connecting the last vertex selected with the initial vertex; it does not create a new vertex at the point under the mouse.
Moving the entire polygon	Move the pointer inside the region. The pointer changes to a fleur shape ✥. Click and drag the polygon over the image.
Deleting the polygon	Press **Backspace**, **Escape** or **Delete**, or right-click inside the region and select **Cancel** from the context menu. Note: If you delete the ROI, the function returns empty values.
Moving a vertex. (Reshaping the region-of-interest.)	Move the pointer over a vertex. The pointer changes to a circle ○. Click and drag the vertex to its new position.
Adding a new vertex.	Move the pointer over an edge of the polygon and press the **A** key. The pointer changes shape to ✜. Click the left mouse button to create a new vertex at that point on the edge.

Interactive Behavior	Description
Deleting a vertex. (Reshaping the region-of-interest.)	Move the pointer over the vertex. The pointer changes to a circle ◯. Right-click and select **Delete vertex** from the context menu. `roipoly` draws a new straight line between the two vertices that were neighbors of the deleted vertex.
Changing the color of the polygon	Move the pointer anywhere inside the boundary of the region and click the right mouse button. Select **Set color** from the context menu.
Retrieving the coordinates of the vertices	Move the pointer inside the region. Right-click and select **Copy position** from the context menu to copy the current position to the Clipboard. The position is an *n*-by-2 array containing the *x*- and *y*-coordinates of each vertex, where *n* is the number of vertices.

Tips

- `roipoly` always produces a closed polygon. If you specify input vertex positions of a closed polygon (such that the last pair of coordinates is identical to the first pair), then the length of the output coordinate vectors is equal to the number of points specified. If the points specified do not describe a closed polygon, then `roipoly` adds a final point having the same coordinates as the first point. In this case the length of the output coordinate vectors is one greater than the number of points specified.
- For more information about classifying pixels on the ROI boundary, see “Classify Pixels That Are Partially Enclosed by ROI”.
- For any of the `roipoly` syntaxes, you can replace the input image `I` with two arguments, `m` and `n`, that specify the row and column dimensions of an arbitrary image. For example, these commands create a 100-by-200 binary mask.

  ```
  c = [112 112 79 79];
  r = [37 66 66 37];
  BW = roipoly(100,200,c,r);
  ```

  If you specify `m` and `n` with an interactive form of `roipoly`, an `m`-by-`n` black image is displayed. Use the mouse to specify a polygon within this image.

See Also

`drawpolygon` | `poly2mask` | `roifilt2` | `roicolor` | `regionfill`

Topics

“Image Types in the Toolbox”
“Define World Coordinate System of Image”
“Classify Pixels That Are Partially Enclosed by ROI”

Introduced before R2006a

rsetwrite

Create R-Set file from image file

Syntax

```
rsetfile = rsetwrite(filename)
rsetfile = rsetwrite(filename,rsetfilename)
rsetfile = rsetwrite(adapter,rsetfilename)
```

Description

`rsetfile = rsetwrite(filename)` creates a reduced resolution dataset (R-Set) file from the specified input. The input file must be a TIFF or NITF image file. The function writes the generated R-Set file to the current working folder and has same file name as the input but with an `rset` extension.

`rsetfile = rsetwrite(filename,rsetfilename)` specifies the name of R-Set file using `rsetfilename`.

`rsetfile = rsetwrite(adapter,rsetfilename)` creates an R-Set file named `rsetfilename` from an `ImageAdapter` object, `adapter`. Use this syntax when creating an R-Set file from a type of image file that is not TIFF or NITF.

Examples

Create R-Set From Large TIFF Image File

Load a TIFF image file into the workspace.

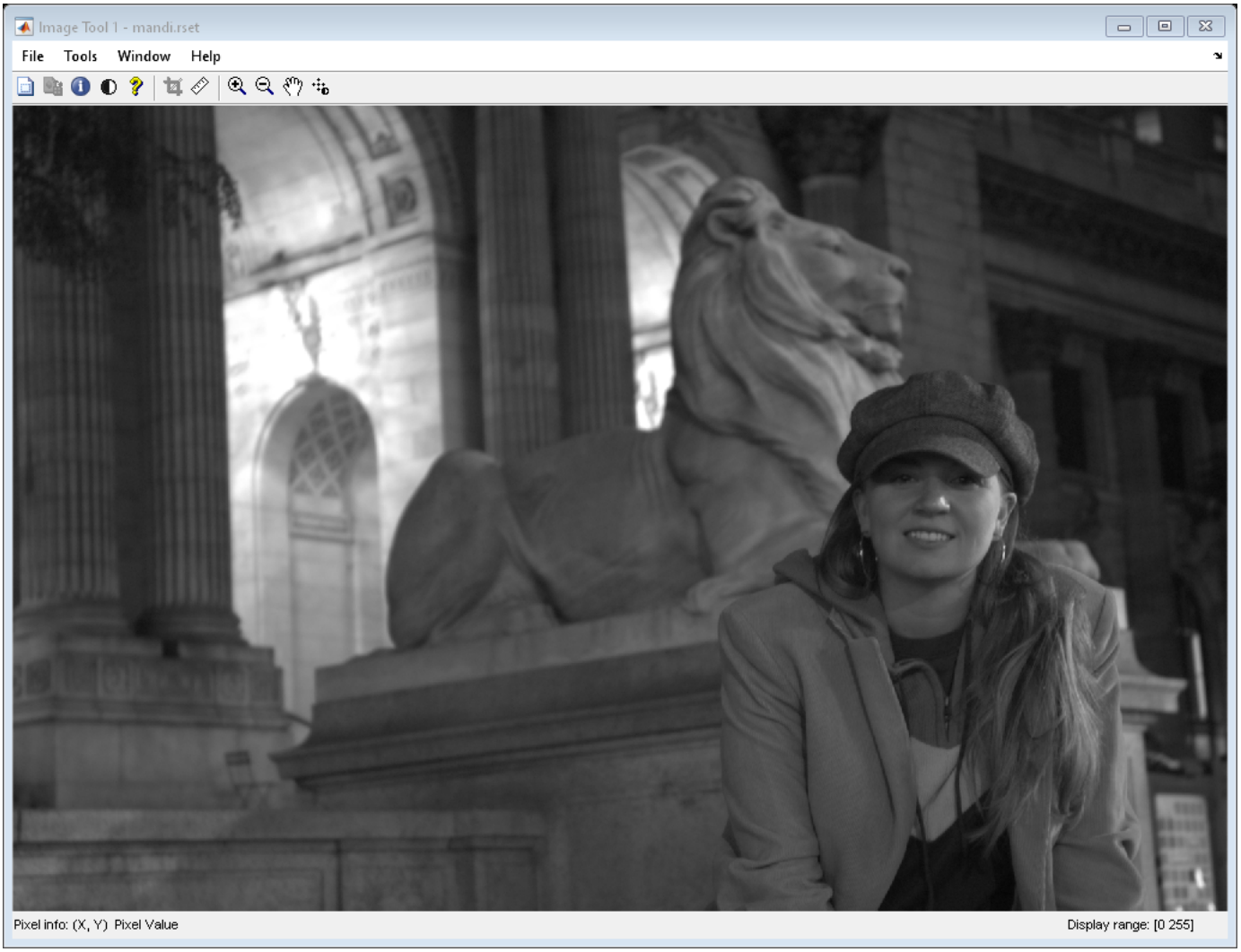
```
filename = 'mandi.tif';
```

Create an R-Set file from the image file. The function creates the R-Set in the current working folder.

```
rsetfile = rsetwrite(filename);
```

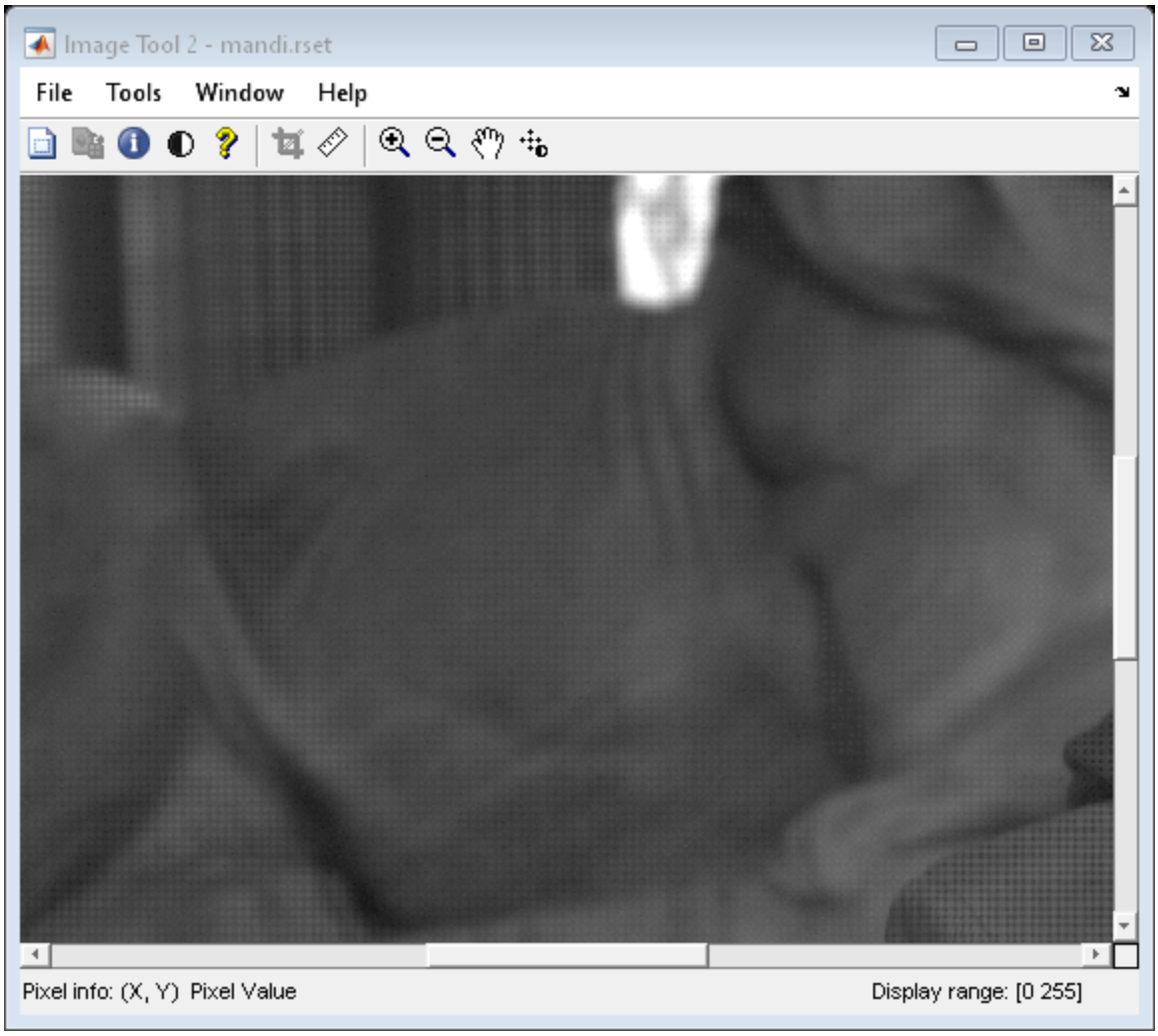
Display the R-Set file by using the Image Viewer function.

```
imtool(rsetfile)
```



Zoom in on the R-Set by 65% to view the spatial tiles.

```
imtool(rsetfile,'InitialMagnification',65);
```



Create R-Set from `ImageAdapter` Object

Load a file containing an `ImageAdapter` object into the workspace.

```
load('MandiImageAdapter.mat')
```

Specify a name for the R-Set file to be created.

```
rsetfilename = 'MandiRSet';
```

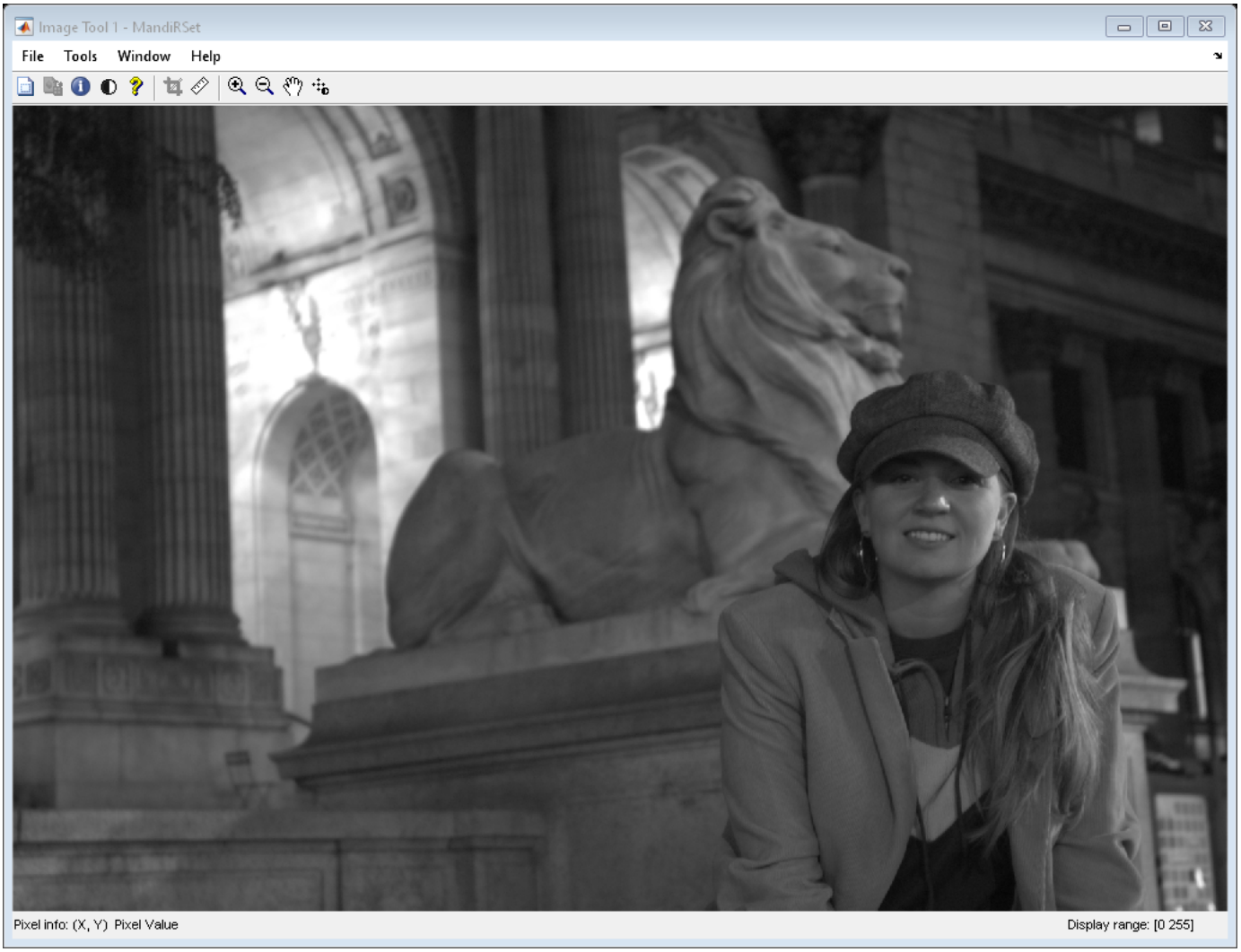
Create an R-Set file from the `ImageAdapter` object. The function creates the R-Set in the current working folder.

```
rsetfile = rsetwrite(adapter,rsetfilename)
```

```
rsetfile =
'MandiRSet'
```

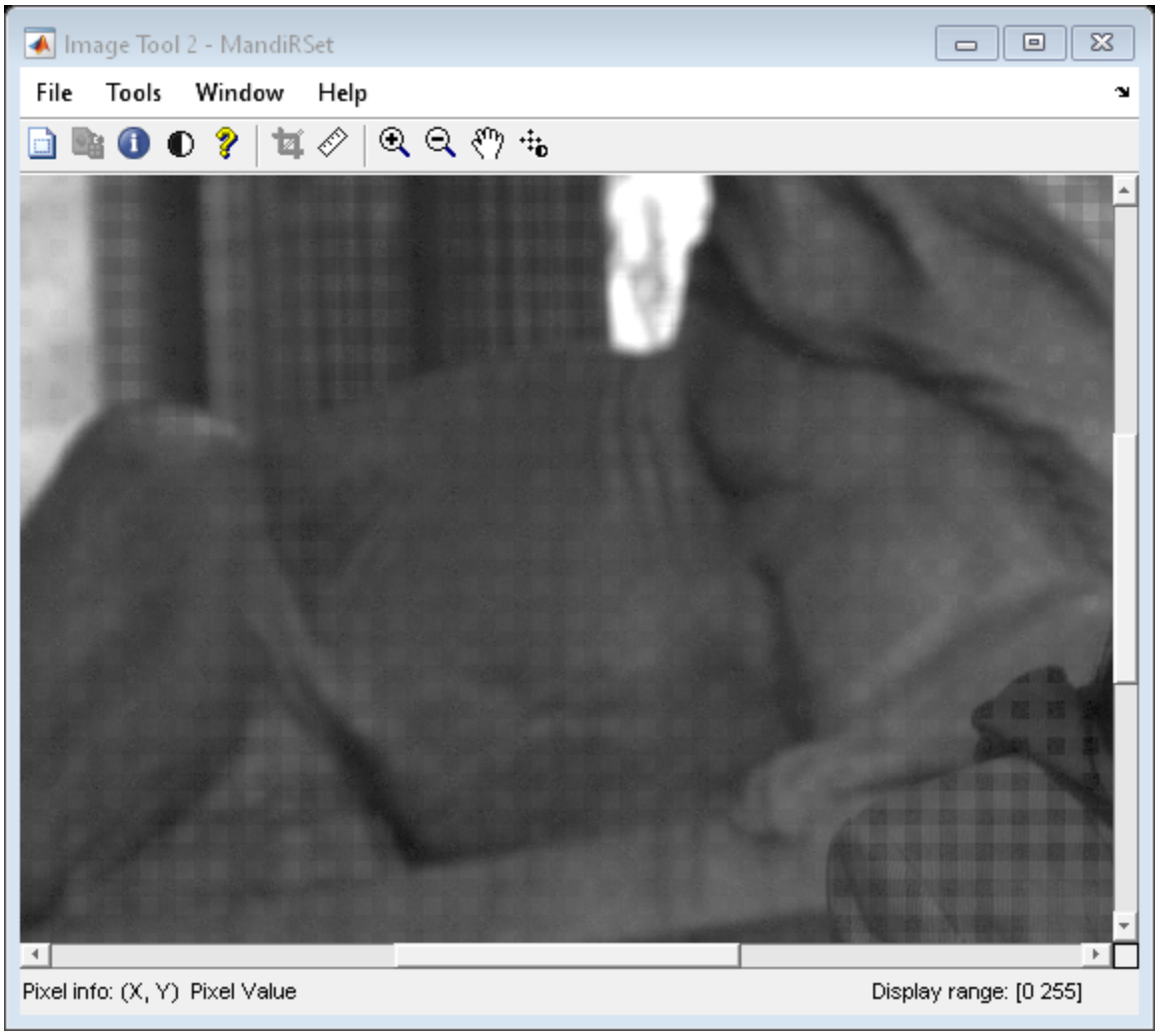
Display the R-Set file using the Image Viewer function.

```
imtool(rsetfile)
```



Zoom in on the R-Set by 53% to view the spatial tiles.

```
imtool(rsetfile,'InitialMagnification',53)
```



Input Arguments

`filename` — Name of TIFF or NITF image file

character vector | string scalar

Name of a TIFF or NITF image file, specified as a character vector or string scalar.

Data Types: `char` | `string`

`rsetfilename` — Name for output R-Set file

character vector | string scalar

Name for the output R-Set file, specified as a character vector or string scalar. If `rsetfilename` is not specified, `filename` sets the name of R-Set file, `rsetfile`.

Data Types: `char` | `string`

`adapter` — Image adapter object

`ImageAdapter` object

Image adapter object, specified as an `ImageAdapter` object. An `ImageAdapter` is a user-defined object that provides `rsetwrite` function with a common API to read a particular image file format.

Output Arguments

`rsetfile` — Name of R-Set file

string scalar

Name of the R-Set file, returned as a string scalar. This value specifies the name of the file to which the R-Set is stored.

Tips

- `rsetwrite` creates an R-Set by dividing an image into spatial tiles and resampling the image at different resolution levels. The R-Set file contains a compressed copy of the full-resolution image data. You can use the **Image Viewer** app to open the R-Set file and zoom in to view the tiles at a higher resolution. When you zoom out, the function displays tiles at a lower resolution. In this way, an R-Set file balances clarity of the image and memory usage for optimal performance.
- When creating an R-Set, a progress bar shows the status of the completion. If you cancel the creation process before it is complete, the function does not create an R-Set and returns an empty `rsetfile`.
- `rsetwrite` supports NITF image files that are uncompressed and Version 2.0 or higher. This function does not support NITF files with more than three data bands or with floating-point data. Images with more than one data band are accepted if they contain unsigned integer data.
- You can create an R-Set from an image whose dimensions are smaller than the size of a single R-Set tile. However, the resulting R-Set file might be larger and take longer to load than the original file. The current size of a tile in an R-Set is 512-by-512 pixels.

See Also

Image Viewer | `imread` | `isrset` | `openrset`

Introduced in R2009a

sizesMatch

Determine if object and image are size-compatible

Syntax

```
TF = sizesMatch(R,A)
```

Description

`TF = sizesMatch(R,A)` returns `True` if the size of image `A` is consistent with the `ImageSize` property of spatial referencing object `R`.

Examples

Check If 2-D Grayscale Image and 2-D Spatial Referencing Object Are Size-Compatible

Read a 2-D grayscale image into the workspace. View the size of the image.

```
I = imread('cameraman.tif');
size(I)
```

```
ans = 1×2

   256   256
```

Create an `imref2d` spatial referencing object with the same dimensions as the image.

```
R = imref2d(size(I))
```

```
R =
  imref2d with properties:

           XWorldLimits: [0.5000 256.5000]
           YWorldLimits: [0.5000 256.5000]
              ImageSize: [256 256]
    PixelExtentInWorldX: 1
    PixelExtentInWorldY: 1
    ImageExtentInWorldX: 256
    ImageExtentInWorldY: 256
       XIntrinsicLimits: [0.5000 256.5000]
       YIntrinsicLimits: [0.5000 256.5000]
```

Confirm that the size of the image matches the `ImageSize` property of the object.

```
res = sizesMatch(R,I)
```

```
res = logical
   1
```

Read another 2-D grayscale image that has a different size. View the size of this image.

```
I2 = imread('coins.png');
size(I2)
```

```
ans = 1×2

   246   300
```

Check if the size of this image matches the size of the original spatial referencing object.

```
res2 = sizesMatch(R,I2)
```

```
res2 = logical
   0
```

The result is false, as expected.

Check If 2-D RGB Image and 2-D Spatial Referencing Object Are Size-Compatible

Read an RGB image into the workspace. View the size of the image.

```
I = imread('peppers.png');
size(I)
```

```
ans = 1×3

   384   512     3
```

Create an `imref2d` spatial referencing object with the same dimensions as the image. The object does not retain information about the third dimension of the image array.

```
R = imref2d(size(I))
```

```
R =
  imref2d with properties:

           XWorldLimits: [0.5000 512.5000]
           YWorldLimits: [0.5000 384.5000]
              ImageSize: [384 512]
    PixelExtentInWorldX: 1
    PixelExtentInWorldY: 1
    ImageExtentInWorldX: 512
    ImageExtentInWorldY: 384
       XIntrinsicLimits: [0.5000 512.5000]
       YIntrinsicLimits: [0.5000 384.5000]
```

Check if the size of the image is compatible with the `ImageSize` property of the object.

```
res = sizesMatch(R,I)
```

```
res = logical
   1
```

Check If 3-D Image Array and 3-D Spatial Referencing Object Are Size-Compatible

Read a 3-D volume into the workspace. This image consists of 27 frames of 128-by-128 pixel grayscale images.

```
load mri;
D = squeeze(D);
D = ind2gray(D,map);
size(D)
```

```
ans = 1×3

   128   128    27
```

Create an `imref3d` spatial referencing object associated with the volume.

```
R = imref3d(size(D))
```

```
R =
  imref3d with properties:

          XWorldLimits: [0.5000 128.5000]
          YWorldLimits: [0.5000 128.5000]
          ZWorldLimits: [0.5000 27.5000]
             ImageSize: [128 128 27]
    PixelExtentInWorldX: 1
    PixelExtentInWorldY: 1
    PixelExtentInWorldZ: 1
    ImageExtentInWorldX: 128
    ImageExtentInWorldY: 128
    ImageExtentInWorldZ: 27
       XIntrinsicLimits: [0.5000 128.5000]
       YIntrinsicLimits: [0.5000 128.5000]
       ZIntrinsicLimits: [0.5000 27.5000]
```

Confirm that the size of the volume matches the `ImageSize` property of the object.

```
res = sizesMatch(R,D)
```

```
res = logical
   1
```

The sizes match, as expected.

Read another image that has a different size. This image a 3-D array representing an RGB image.

```
I = imread('peppers.png');
size(I)
```

```
ans = 1×3

   384   512     3
```

Check if the size of this image matches the size of the original spatial referencing object.

```
res2 = sizesMatch(R,I)
```

```
res2 = logical
   0
```

The result is false, as expected.

Input Arguments

R — Spatial referencing object

`imref2d` or `imref3d` object

Spatial referencing object, specified as an `imref2d` or `imref3d` object.

A — Input image

numeric *m*-by-*n* or *m*-by-*n*-by-*p* array

Input image, specified as a numeric *m*-by-*n* or *m*-by-*n*-by-*p* array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

TF — Flag indicating size compatibility

logical scalar

Flag indicating size compatibility, returned as a logical scalar. `TF` is `True` if the size of the image `A` is consistent with the referencing object `R`. When `R` is:

- An `imref2d` spatial referencing object, `TF` returns true when `R.ImageSize == [size(A,1) size(A,2)]`.

  Note The dimensionality of `A` does not need to match the dimensionality of an `imref2d` spatial referencing object. For example, an RGB image can be consistent with an `imref2d` object. In this case, `sizesMatch` ignores the third image dimension.

- An `imref3d` spatial referencing object, `TF` returns true when `R.ImageSize == size(A)`. `A` must be a 3-D array.

Data Types: `logical`

See Also

Introduced in R2013a

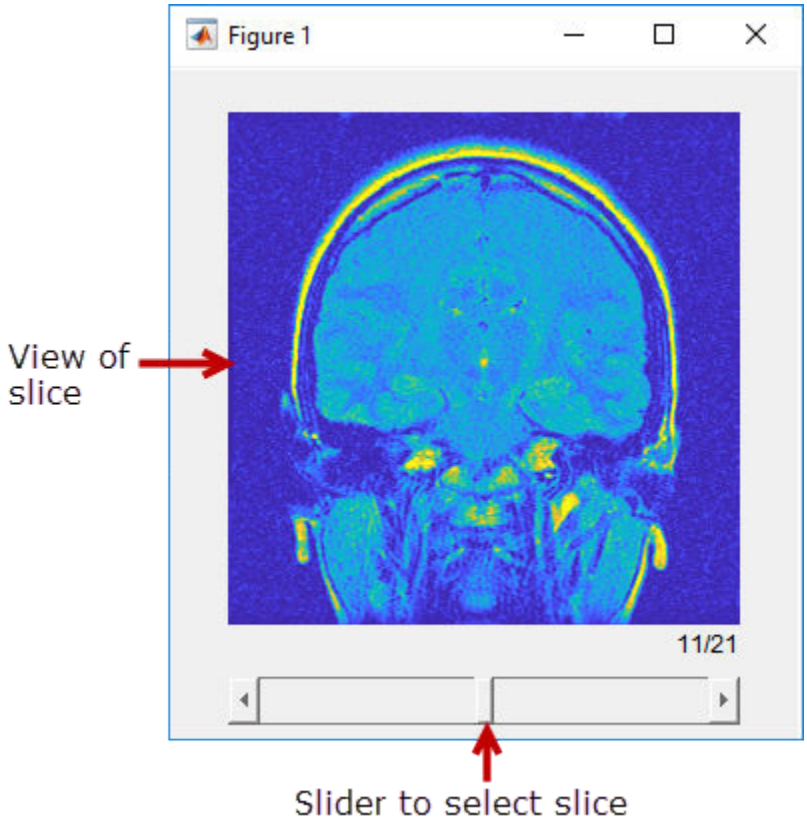
sliceViewer

Browse image slices

Description

A `sliceViewer` object displays individual slices of grayscale and RGB volumes. You can control which slice to display by using a slider.

When it opens, the `sliceViewer` object displays the middle image in the stack in the direction specified by `sliceDirection`. Use the slider to navigate through the volume and view individual slices.



The `sliceViewer` object supports properties, object functions, and events that you can use to customize its appearance and functioning. The `sliceViewer` object can send notifications when certain events occur, such as the slider moving. For more information, see “Events” on page 1-2984.

Note By default, clicking and dragging the mouse in the slice displayed interactively changes their brightness and contrast, a technique called window/level. Dragging the mouse horizontally from left to right changes the contrast. Dragging the mouse vertically up and down changes the brightness. Holding down the **Ctrl** key when clicking and dragging the mouse accelerates changes. Holding down the **Shift** key while clicking and dragging the mouse slows the rate of change. Press these keys

before clicking and dragging. To control this behavior, use the “DisplayRangeInteraction” on page 1-0 property.

Creation

Description

`sliceViewer(V)` displays the grayscale or RGB volume `V` in a figure. The figure includes a slider that you can use to view individual slices of the volume.

`sliceViewer(___,Name,Value)` sets properties on page 1-2980 using name-value pair arguments. You can specify multiple name-value pairs. Enclose each property name in single quotes.

Example: `sliceViewer(V,'Colormap',cmap)` creates a `sliceViewer` object and specifies the colormap used to display the volume.

`sv = sliceViewer(___)` returns a `sliceViewer` object, `sv`, with properties that can be used to control visualization of the volume. Use input arguments from any of the previous syntaxes.

Input Arguments

`V` — Input volume

m-by-*n*-by-*p*-by-*c* numeric array

Input volume, specified as an *m*-by-*n*-by-*p*-by-*c* numeric array. For grayscale volumes, `c` is 1. For RGB volumes, *c* is 3. RGB volumes can only be of class `uint8`, `uint16`, `single`, and `double`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Properties

`Colormap` — Colormap of image stack

`gray(256)` (default) | *m*-by-3 numeric array

Colormap of the image stack, specified as an *m*-by-3 numeric array with values in the range `[0 1]`. The `Colormap` property has no effect when `V` is an RGB image stack.

`DisplayRange` — Display range of image stack

`[min(V(:)) max(V(:))]` (default) | two-element vector

Display range of image stack, specified as a two-element vector of the form `[low high]`. The value `low` (and any value less than `low`) displays as black. The value `high` (and any value greater than `high`) displays as white. Values in between are displayed as intermediate shades of gray, using the default number of gray levels. If you specify an empty matrix (`[]`), `sliceViewer` uses the default value. `DisplayRange` has no effect when you specify a stack of RGB images.

`DisplayRangeInteraction` — Enable interactive control of display range

`'on'` | `'off'`

Enable interactive control of display range, specified as one of these values. To learn more about interactive behavior, see Events on page 1-2984.

Value	Description
`'on'` (default for grayscale intensity volumes)	Control the display range of a grayscale image stack by left-clicking the mouse and dragging it on the axes.
`'off'` (default for logical and RGB volumes)	No display range interactivity.

`Parent` — Parent of `sliceViewer` object
`gcf` (default) | uipanel | figure

Parent of the `sliceViewer` object, specified as a handle to a `uipanel` or a figure created with the `figure` or `uifigure` commands. If you do not specify a parent, the parent of the `sliceViewer` object is `gcf`.

`ScaleFactors` — Scale factors used to rescale the volume
`[1 1 1]` (default) | 1-by-3 positive numeric vector

Scale factors used to rescale the volume, specified as a 1-by-3 positive numeric vector. The values in the array correspond to the scale factor applied in the *x*, *y*, and *z* directions.

`SliceDirection` — Direction in which to browse image stack
`[0 0 1]` (default) | 1-by-3 logical vector | `'X'` | `'Y'` | `'Z'`

Direction in which to browse image stack, specified as 1-by-3 logical vector or one of the character vectors in this table.

Character Vector	Logical Vector	Description
`'X'`	`[1 0 0]`	Browse in *X* direction
`'Y'`	`[0 1 0]`	Browse in *Y* direction
`'Z'` (default)	`[0 0 1]`	Browse in *Z* direction

`SliceNumber` — Index of slice to be displayed
center slice | positive numeric scalar

Index of the slice to be displayed from the volume, specified as a positive numeric scalar.

Object Functions

addlistener	Create event listener bound to event source
getAxesHandle	Get handle to axes in Slice Viewer

Examples

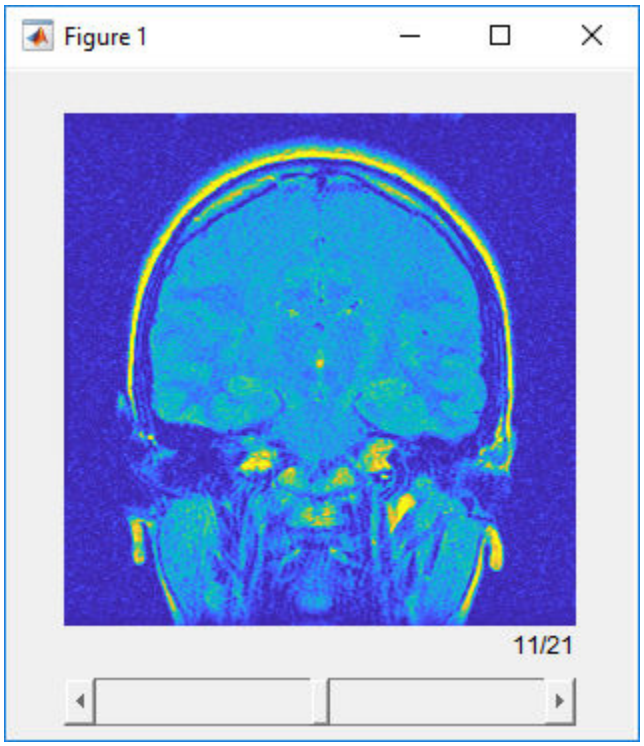
View MRI Data in Slice Viewer

Load MRI data into the workspace.

```
load mristack
```

View the data in the slice viewer, specifying a custom colormap for viewing the slices. The slice viewer opens the stack of images and displays the one in the middle. Use the slider to view a different slice.

```
cmap = parula(256);
s = sliceViewer(mristack,'Colormap',cmap);
```



Create GIF of Slices Using the Slice Viewer

Load MRI data into the workspace.

```
load mristack
```

View the data in the slice viewer.

```
s = sliceViewer(mristack);
```

Get the handle of the axes containing the displayed slice.

```
hAx = getAxesHandle(s);
```

Specify the name of the GIF file you want to create.

```
filename = 'animatedSlice.gif';
```

Create an array of slice numbers.

```
sliceNums = 1:21;
```

Loop through the slice numbers and create an image of each displayed slice. Write the images to a GIF file.

```
for idx = sliceNums
    % Update slice number
```

```
    s.SliceNumber = idx;
    % Use getframe to capture image
    I = getframe(hAx);
    [indI,cm] = rgb2ind(I.cdata,256);
    % Write frame to the GIF file
    if idx == 1
        imwrite(indI,cm,filename,'gif','Loopcount',inf,'DelayTime', 0.05);
    else
        imwrite(indI,cm,filename,'gif','WriteMode','append','DelayTime', 0.05);
    end
end
```

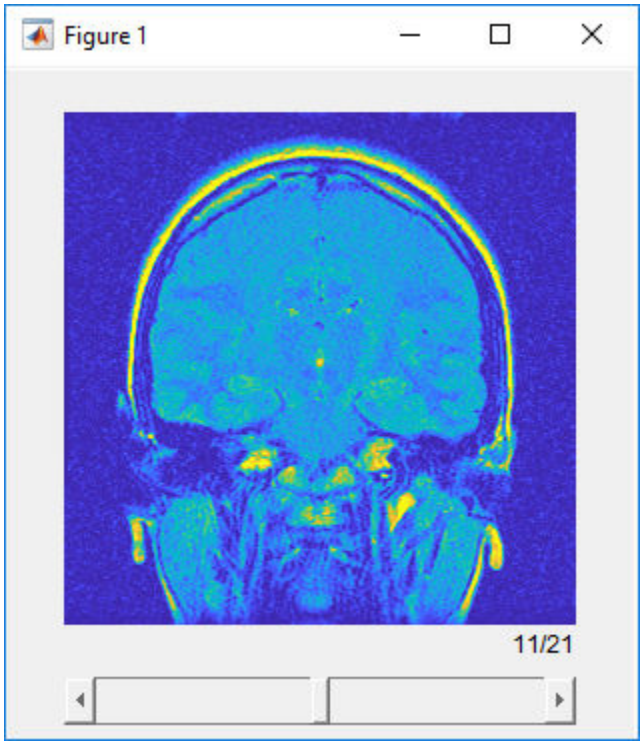
Set Up Listener for Slice Viewer Slider Events

Load a stack of images into the workspace.

```
load mristack
```

View the data in the slice viewer, specifying a custom colormap for viewing the slices. The slice viewer opens the stack of images and displays the one in the middle. Use the slider to view a different slice.

```
cmap = parula(256);
s = sliceViewer(mristack,'Colormap',cmap);
```



Set up listeners for the two `sliceViewer` object slider events: when the slider is moving and when the slider has been moved. When you move the slider, the slice viewer sends notifications of these events and executes the specified callback function.

```
addlistener(s,'SliderValueChanging',@allevents);
addlistener(s,'SliderValueChanged',@allevents);
```

Use this `allevents` callback function to display the name of each event and the current position of the slider.

```
function allevents(src,evt)
    evname = evt.EventName;
    switch(evname)
        case{'SliderValueChanging'}
            disp(['Slider value changing event: ' mat2str(evt.CurrentValue)]);
        case{'SliderValueChanged'}
            disp(['Slider value changed event: ' mat2str(evt.CurrentValue)]);
    end
end
```

More About

Events

The `sliceViewer` object can send notifications when the slider moves. To receive these notifications, use the `addListener` object function to set up a listener. To set up a listener, specify the name of the event, for example, `'SliderValueChanging'`, and the function you want executed when the event occurs. The following table lists events supported by the `sliceViewer` object. For an example, see “Set Up Listener for Slice Viewer Slider Events” on page 1-2983.

Event Name	Trigger	Event Data	Event Attributes

`SliderValueChanging`	The slider in the `sliceViewer` object is moving.	`images.stack.browser.SliderMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

`SliderValueChanged`	The slider in the `sliceViewer` object has been moved.	`images.stack.browser.SliderMovingEventData`	`NotifyAccess: private`
			
			`ListenAccess: public`

See Also

`orthosliceViewer` | **Volume Viewer** | `volshow` | `slice` | `obliqueslice`

Introduced in R2019b

getAxesHandle

Get handle to axes in Slice Viewer

Syntax

```
hAx = getAxesHandle(s)
```

Description

`hAx = getAxesHandle(s)` returns the axes in the Slice Viewer `s`.

Examples

Get Handle to Axes in Slice Viewer

Load MRI data into the workspace.

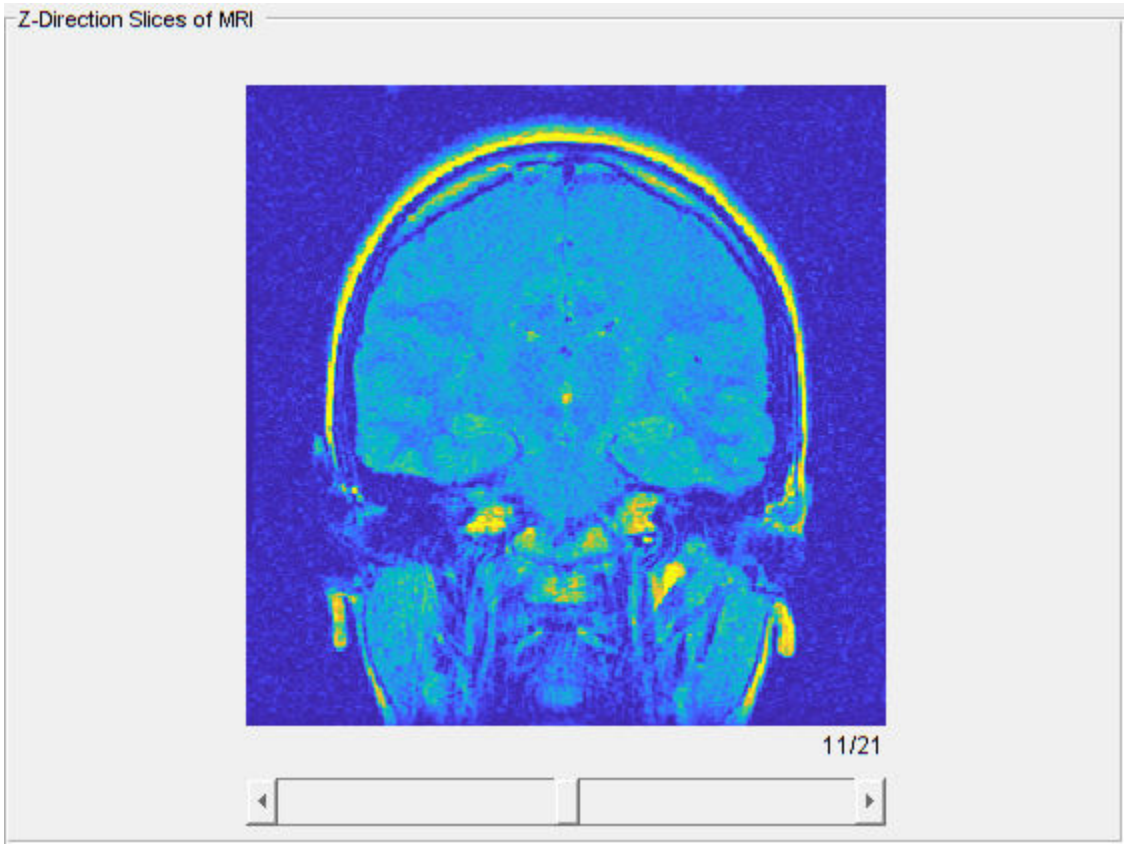
```
load mristack
```

Create a display panel used to display the slices and the slider.

```
ViewPnl = uipanel(figure,'Title','Z-Direction Slices of MRI');
```

View the data in the slice viewer, specifying a custom colormap for viewing the slices. The slice viewer opens the stack of images and displays the one in the middle. Use the slider to view a different slice.

```
cmap = parula(256);
s = sliceViewer(mristack,'Colormap',cmap,'Parent',ViewPnl);
```



Get the handle to the axes that contains the image slices in the slice viewer.

```
hAx = getAxesHandle(s)
```

```
hAx =
  Axes with properties:

            XLim: [0.5000 256.5000]
            YLim: [0.5000 256.5000]
          XScale: 'linear'
          YScale: 'linear'
    GridLineStyle: '-'
         Position: [120 57.5000 320 320]
            Units: 'pixels'

  Show all properties
```

Input Arguments

s — Slice Viewer

`sliceViewer` object

Slice Viewer, specified as a `sliceViewer` object.

Output Arguments

hAx — Axes in Slice Viewer

`Axes` object

Axes in Slice Viewer, returned as an `Axes` object.

See Also

`sliceViewer`

Introduced in R2019b

selectBlockLocations

Select blocks from blocked images

Syntax

```
blset = selectBlockLocations(bims)
blset = selectBlockLocations(bims,Name,Value)
```

Description

`blset = selectBlockLocations(bims)` selects a set of nonoverlapping unique blocks from one or more `blockedImage` objects `bims` at the finest resolution available in each image. Returns `blset`, a `blockLocationsSet` object.

`blset = selectBlockLocations(bims,Name,Value)` specifies additional options about the blocks to select, such as the overlap and spacing between blocks, using one or more name-value pair arguments.

Examples

Create blockedImageDatastore Using Non-overlapping Blocks

Create a blocked image.

```
bim = blockedImage('tumor_091R.tif');
```

Create a block location set excluding incomplete blocks.

```
 bls = selectBlockLocations(bim,'ExcludeIncompleteBlocks',true);
```

Create a `blockedImageDatastore` from this set of blocks.

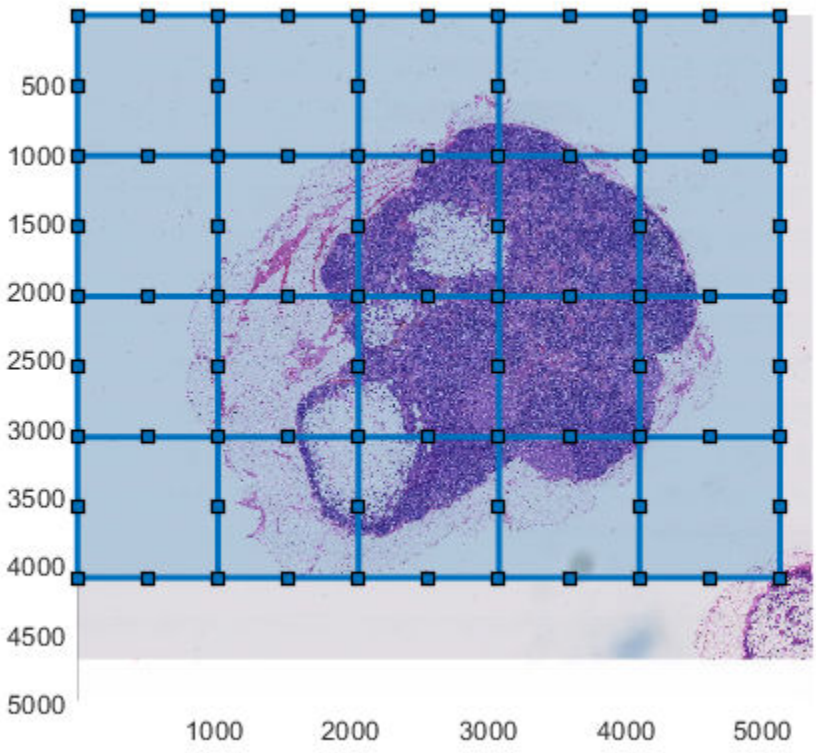
```
bimds = blockedImageDatastore(bim,'BlockLocationSet', bls);
```

Visualize the blocked locations.

```
bigimageshow(bim)
```

Block size is in row-col (height-width) order.

```
blockedWH = fliplr(bls.BlockSize(1,1:2));
for ind = 1:size(bls.BlockOrigin,1)
    % BlockOrigin is already in x,y order.
    drawrectangle('Position', [bls.BlockOrigin(ind,1:2),blockedWH]);
end
```



Create blockedImageDatastore with Overlapping Blocks

Create a blocked image.

```
bim = blockedImage('tumor_091R.tif');
```

Create a `blockLocationSet` object.

```
blockSize = [2048 3072];
overlapPct = 0.5;
blockOffsets = round(blockSize.*overlapPct);
bls = selectBlockLocations(bim,...
     'BlockSize', blockSize,...
     'BlockOffSets', blockOffsets,...
     'ExcludeIncompleteBlocks', true);
```

Create a `blockedImageDatastore` from this set of blocks.

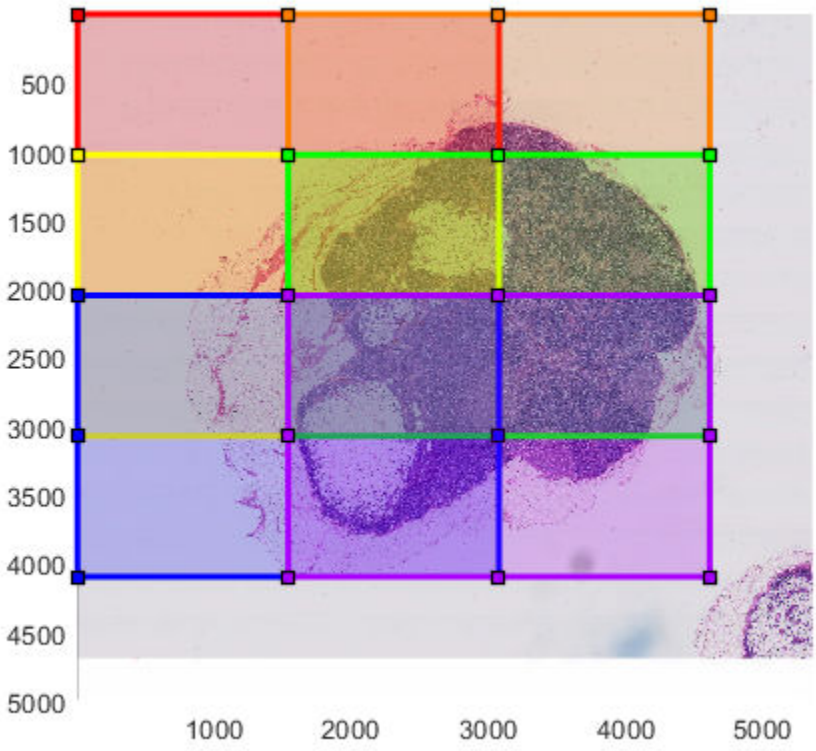
```
bimds = blockedImageDatastore(bim, 'BlockLocationSet', bls);
```

Visualize the blocked locations.

```
bigimageshow(bim)
```

Block size is in row-col (height-width) order.

```
blockedWH = fliplr(bls.BlockSize(1,1:2));
colors = prism(size(bls.BlockOrigin,1));
for ind = 1:size(bls.BlockOrigin,1)
     blockedColor = colors(ind,:);
     % BlockOrigin is already in x-y order
     drawrectangle('Position', [bls.BlockOrigin(ind,1:2), blockedWH],'Color', blockedColor);
end
```



Create blockedImageDatastore with Sparse Blocks

Create a blocked image.

```
bim = blockedImage('tumor_091R.tif');
```

Create a `blockLocationSet` object.

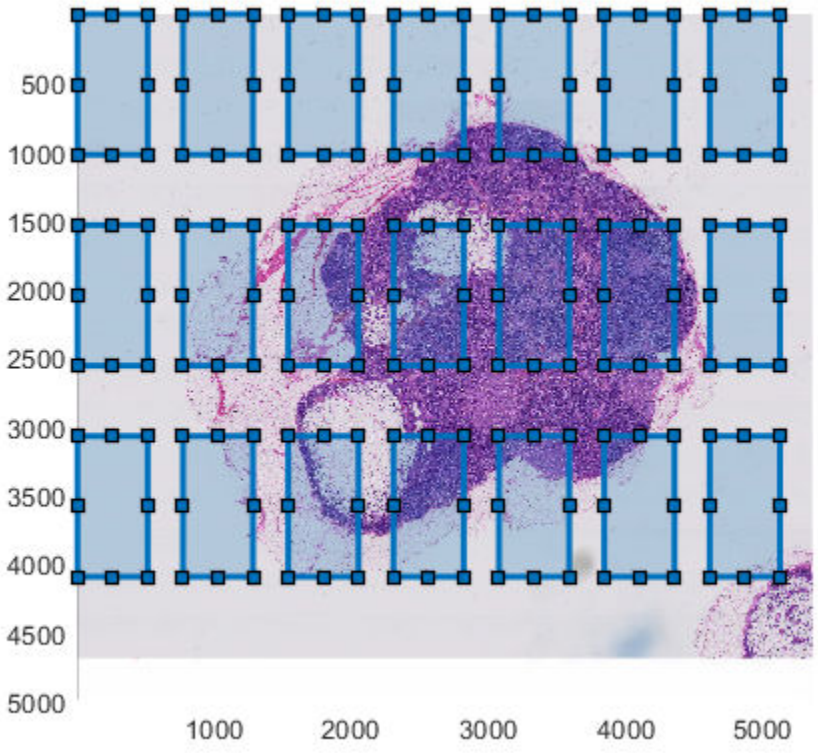
```
blockedSize = [1024 512];
spacePct = 0.5;
blockedOffsets = blockedSize + blockedSize.*spacePct;
bls = selectBlockLocations(bim,...
          'BlockSize', blockedSize,...
          'BlockOffSets', blockedOffsets,...
          'ExcludeIncompleteBlocks', true);
```

Create a `blockedImageDatastore` object from this set of blocks.

```
bimds = blockedImageDatastore(bim, 'BlockLocationSet', bls);
```

Visualize the block locations.

```
bigimageshow(bim)
% Block size is in row-col (height-width) order
blockedWH = fliplr(bls.BlockSize(1,1:2));
for ind = 1:size(bls.BlockOrigin,1)
     % BlockOrigin is already in x-y order
     drawrectangle('Position', [bls.BlockOrigin(ind,1:2), blockedWH]);
end
```



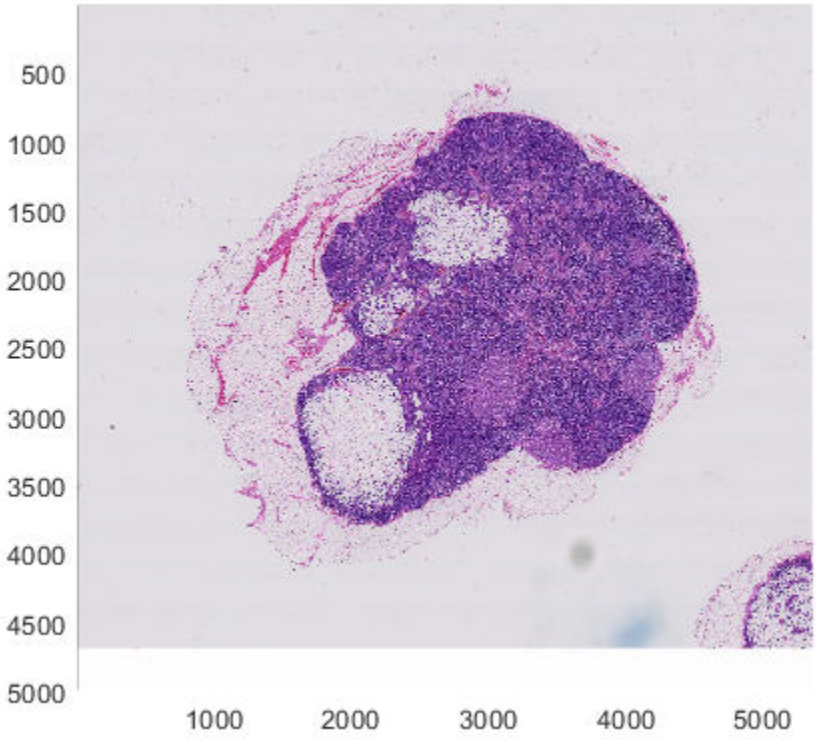
Create blockedImageDatastore Using Coarse Level Mask

Create a blocked image.

```
bim = blockedImage("tumor_091R.tif");
```

Display the blocked image.

```
h = bigimageshow(bim);
```

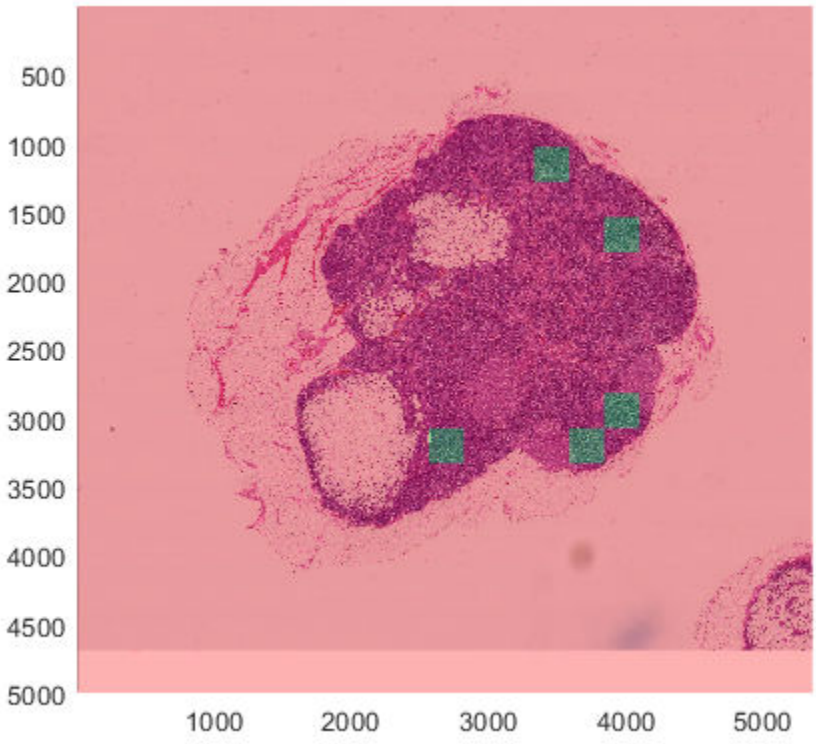


Create a mask at the coarsest level.

```
clevel = bim.NumLevels;
bmask = apply(bim,@(b)~imbinarize(im2gray(b.Data)),"Level",clevel);
```

Use `showMask` to estimate an `InclusionThreshold` value.

```
showmask(h,bmask,"BlockSize",[256 256],"InclusionThreshold",0.9)
```



Create a `blockedImagedatastore` for blocks in which at least 90% of pixels are `true` in the stained region as defined by the mask.

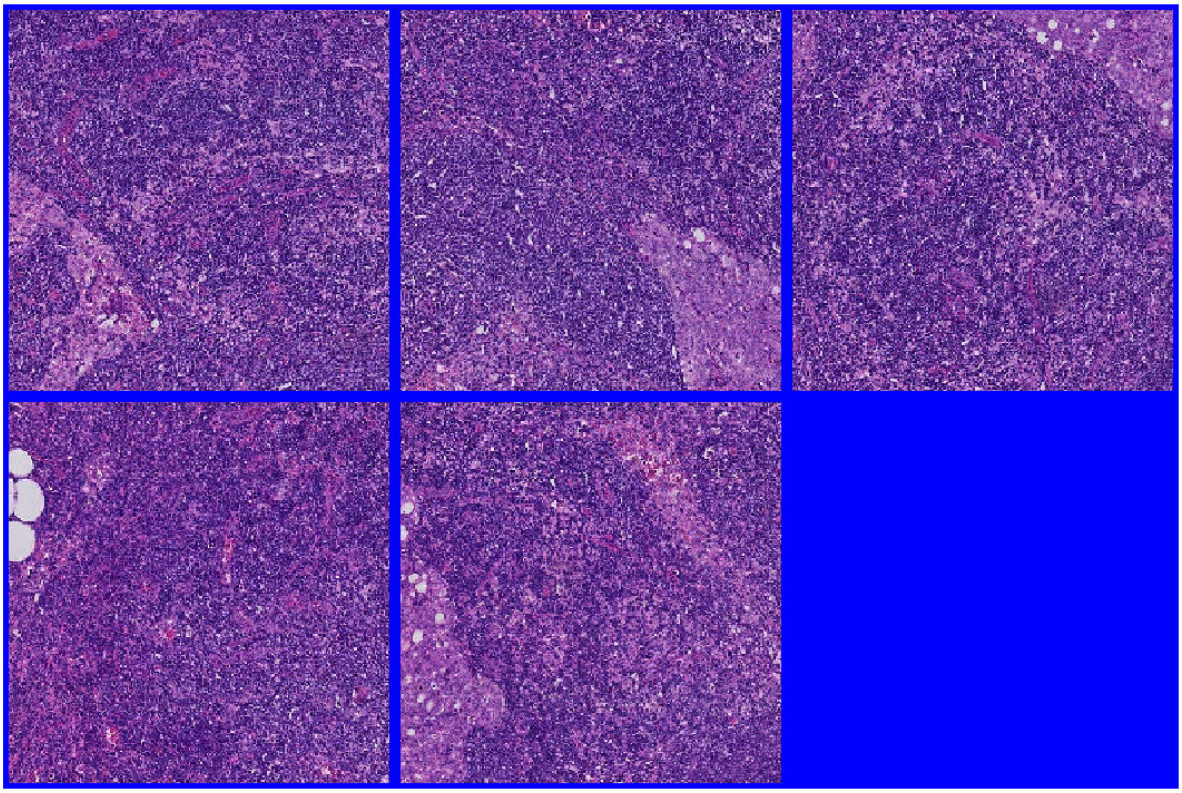
```
mbls = selectBlockLocations(bim, ...
                "Levels",1, ...
                "Masks",bmask,"InclusionThreshold",0.90, ...
                "BlockSize",[256 256]);
```

Create a `blockedImageDatastore` from this set of blocks.

```
bimds = blockedImageDatastore(bim,"BlockLocationSet",mbls);
```

Verify.

```
bimds.ReadSize = 10;
blocks = read(bimds);
figure
montage(blocks,"BorderSize",5,"BackgroundColor","b");
```



Input Arguments

`bims` — Blocked images

`blockedImage` object | *b*-element vector of `blockedImage` objects

Blocked images, specified as a `blockedImage` object or *b*-element vector of `blockedImage` objects.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'BlockSize',[224 224]` sets the block size to 224-by-224.

`BlockOffsets` — Offset of adjacent blocks

2-element row vector of positive integers

Offset of adjacent blocks, specified as a 2-element row vector of positive integers of the form [*rows columns*].

The default value is equal to `BlockSize`, resulting in non-overlapping blocks. To overlap blocks, specify a smaller value. To add a gap between blocks, specify a larger value.

`BlockSize` — Block size
2-element row vector of positive integers

Block size, specified as a 2-element row vector of positive integers of the form [*rows columns*]. The default value is equal to the `BlockSize` property at the finest resolution level of the first blocked image in `bims`.

`ExcludeIncompleteBlocks` — Exclude incomplete blocks
`false` or `0` (default) | `true` or `1`

Exclude incomplete blocks that are smaller than '`BlockSize`', specified as a numeric or logical `0` (`false`) or `1` (`true`).

`InclusionThreshold` — Inclusion threshold for mask blocks
`0.5` (default) | numeric scalar | *b*-element numeric vector

Inclusion threshold for mask blocks, specified as a numeric scalar or a *b*-element numeric vector with values in the range [0, 1]. The `InclusionThreshold` argument must have the same number of elements as the `Masks` argument. The `selectBlockLocations` function selects blocks that overlap the foreground of the corresponding mask block by a percentage greater than or equal to the value specified by '`InclusionThreshold`'.

- When the inclusion threshold is `0`, the `selectBlockLocations` function selects a block when at least one pixel in the corresponding mask block is nonzero.
- When the inclusion threshold is `1`, the `selectBlockLocations` function selects a block only when all pixels in the mask block are nonzero.

`Levels` — Resolution level
positive integer | *b*-element vector of positive integers

Resolution level of blocks from each blocked image in `bims`, specified as a scalar positive integer or an array of the same size as `bims`. If you specify a scalar value, the `selectBlockLocations` function selects blocks from all blocked images at the same resolution level. Default value is the finest level of each image in the array of blocked images, `bims`.

Data Types: `double`

`Masks` — Mask images
`blockedImage` object | array of `blockedImage` objects

Mask images, specified as an array the same size as `bims`. The underlying data type of the mask images is `logical`. The `selectBlockLocations` function selects blocks that overlap the foreground of the corresponding mask block by an amount specified by `InclusionThreshold`. Masks are expected to be in the same world coordinate system as the corresponding `blockedImage` in the `bims` array.

`UseParallel` — Use parallel processing
`false` or `0` (default) | `true` or `1`

Use parallel processing to evaluate mask blocks, specified as a numeric or logical `0` (`false`) or `1` (`true`). Parallel evaluation of masks is beneficial when the masks do not fit in memory.

Use of parallel processing requires Parallel Computing Toolbox. The `selectBlockLocations` function uses an existing parallel pool of workers, or opens a new pool when no parallel pool is active. The `Source` property of each blocked image in `bims` must be a valid path on all of the parallel workers.

Output Arguments

`blset` — Block locations

`blockLocationSet` object

Block locations, returned as a `blockLocationSet` object.

References

[1] Bejnordi, Babak Ehteshami, Mitko Veta, Paul Johannes van Diest, Bram van Ginneken, Nico Karssemeijer, Geert Litjens, Jeroen A. W. M. van der Laak, et al. “Diagnostic Assessment of Deep Learning Algorithms for Detection of Lymph Node Metastases in Women With Breast Cancer.” JAMA 318, no. 22 (December 12, 2017): 2199–2210. https://doi.org/10.1001/jama.2017.14585.

See Also

`blockLocationSet` | `blockedImage` | `blockedImageDatastore`

External Websites

https://camelyon17.grand-challenge.org/Data/

Introduced in R2020b

spaceToDepth

Rearrange spatial blocks of `dlarray` data along depth dimension

Syntax

```
Y = spaceToDepth(X,blockSize)
Y = spaceToDepth(X,blockSize,'DataFormat',dataFormat)
```

Description

`Y = spaceToDepth(X,blockSize)` rearranges spatial blocks of the formatted `dlarray` object, `X`, along the depth dimension. The blocks of data have size `blockSize`.

Given an input feature map of size [*H W C*] and blocks of size [*height width*], the output feature map size is [`floor`(*H*/*height*) `floor`(*W*/*width*) *C***height***width*].

This function requires Deep Learning Toolbox.

`Y = spaceToDepth(X,blockSize,'DataFormat',dataFormat)` rearranges spatial blocks of the unformatted `dlarray` object, `X`, along the depth dimension. `dataFormat` specifies the dimension labels.

Examples

Rearrange Formatted dlarray Data from Spatial to Depth Dimension

Create a numeric array with three channels that simulates a 4-by-4 RGB image.

```
X = reshape(1:48,4,4,3);
```

Create a `dlarray` object that contains the numeric data, specifying the format of the data as 'SSC' (spatial, spatial, channel).

```
X = dlarray(X,'SSC')
```

```
X =
  4(S) x 4(S) x 3(C) dlarray


(:,:,1) =

     1     5     9    13
     2     6    10    14
     3     7    11    15
     4     8    12    16


(:,:,2) =

    17    21    25    29
    18    22    26    30
```

```
    19    23    27    31
    20    24    28    32


(:,:,3) =

    33    37    41    45
    34    38    42    46
    35    39    43    47
    36    40    44    48
```

Specify a 2-by-2 block size for reordering input activations.

```
blockSize = 2;
```

Rearrange blocks of data from the spatial dimension to the depth dimension.

```
Z = spaceToDepth(X,blockSize)
```

```
Z =
  2(S) x 2(S) x 12(C) dlarray


(:,:,1) =

     1     9
     3    11


(:,:,2) =

    17    25
    19    27


(:,:,3) =

    33    41
    35    43


(:,:,4) =

     5    13
     7    15


(:,:,5) =

    21    29
    23    31


(:,:,6) =

    37    45
    39    47
```

```
(:,:,7) =

     2    10
     4    12


(:,:,8) =

    18    26
    20    28


(:,:,9) =

    34    42
    36    44


(:,:,10) =

     6    14
     8    16


(:,:,11) =

    22    30
    24    32


(:,:,12) =

    38    46
    40    48

  2(S) x 2(S) x 12(C) dlarray
```

Rearrange Unformatted Data from Spatial to Depth Dimension

Create a numeric array with three channels that simulates a 4-by-4 RGB image.

```
X = reshape(1:48,4,4,3);
```

Create an unformatted `dlarray` object that contains the numeric data.

```
dlX = dlarray(X);
```

Specify a 2-by-2 block size for reordering input activations.

```
blockSize = 2;
```

Rearrange blocks of data from the spatial dimension to the depth dimension. Specify the format of the input data as "SSC".

```
dlZ = spaceToDepth(dlX,blockSize,"DataFormat","SSC");
```

Compare the dimensions of the original and rearranged data.

```
whos dlX dlZ
```

```
  Name      Size              Bytes  Class      Attributes

  dlX       4x4x3               384  dlarray
  dlZ       2x2x12              384  dlarray
```

Input Arguments

X — Deep learning data to rearrange

`dlarray` object

Deep learning data to rearrange, specified as a `dlarray` object.

blockSize — Block size to reorder input activation

positive integer | vector of two positive integers

Block size to reorder the input activation, specified as a positive integer or vector of two positive integers of the form `[h w]`, where `h` is the height and `w` is the width. When you specify `blockSize` as a scalar, the function uses the same value for both dimensions.

Example: `[2 4]` specifies blocks of height 2 and width 4.

Example: `32` specifies blocks of height and width 32.

dataFormat — Dimension labels

string scalar | character vector

Dimension labels when the input deep learning data `X` is unlabeled, specified as a string scalar or character vector. The number of labels must match the number of dimensions of the input data, `X`. Each character in `dataFormat` must be one of these labels:

- `S` — Spatial
- `C` — Channel
- `B` — Batch observations

The "T" (time or sequence) and "U" (unspecified) labels are not supported. Do not specify the `dataFormat` argument when the input deep learning data is a formatted `dlarray` object.

Example: `'SSC'` indicates the array has two spatial dimensions and one channel dimension, appropriate for 2-D RGB image data.

Data Types: `char` | `string`

Output Arguments

Y — Rearranged deep learning data

`dlarray` object

Rearranged deep learning data, returned as a `dlarray` object.

Extended Capabilities

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`depthToSpace` | `dlresize`

Topics

“List of Functions with dlarray Support” (Deep Learning Toolbox)

Introduced in R2021a

Cuboid

Spatial extents of 3-D cuboidal region

Description

A `Cuboid` object stores the spatial extents of a 3-D volumetric image.

Creation

You can create a `Cuboid` object in several ways.

- `centerCropWindow3d`— Create a `Cuboid` of a specified size whose position is centered on an image.
- `randomCropWindow3d`— Create a `Cuboid` of a specified size whose position is selected randomly from inside an image.
- Running the command

  ```
  c = images.spatialref.Cuboid(XLimits,YLimits,ZLimits);
  ```

  creates a `Cuboid` object and sets the `XLimits`, `YLimits`, and `ZLimits` properties.

Properties

`XLimits` — Minimum and maximum limits of *x*-axis
2-element numeric vector

Minimum and maximum limits of the cropping window along the *x*-axis, specified as a 2-element numeric vector of the form `[min max]`, where `max` is greater than `min`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32`

`YLimits` — Minimum and maximum limits of *y*-axis
2-element numeric vector

Minimum and maximum limits of the cropping window along the *y*-axis, specified as a 2-element numeric vector of the form `[min max]`, where `max` is greater than `min`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32`

`ZLimits` — Minimum and maximum limits of *z*-axis
2-element numeric vector

Minimum and maximum limits of the cropping window along the *z*-axis, specified as a 2-element numeric vector of the form `[min max]`, where `max` is greater than `min`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32`

Examples

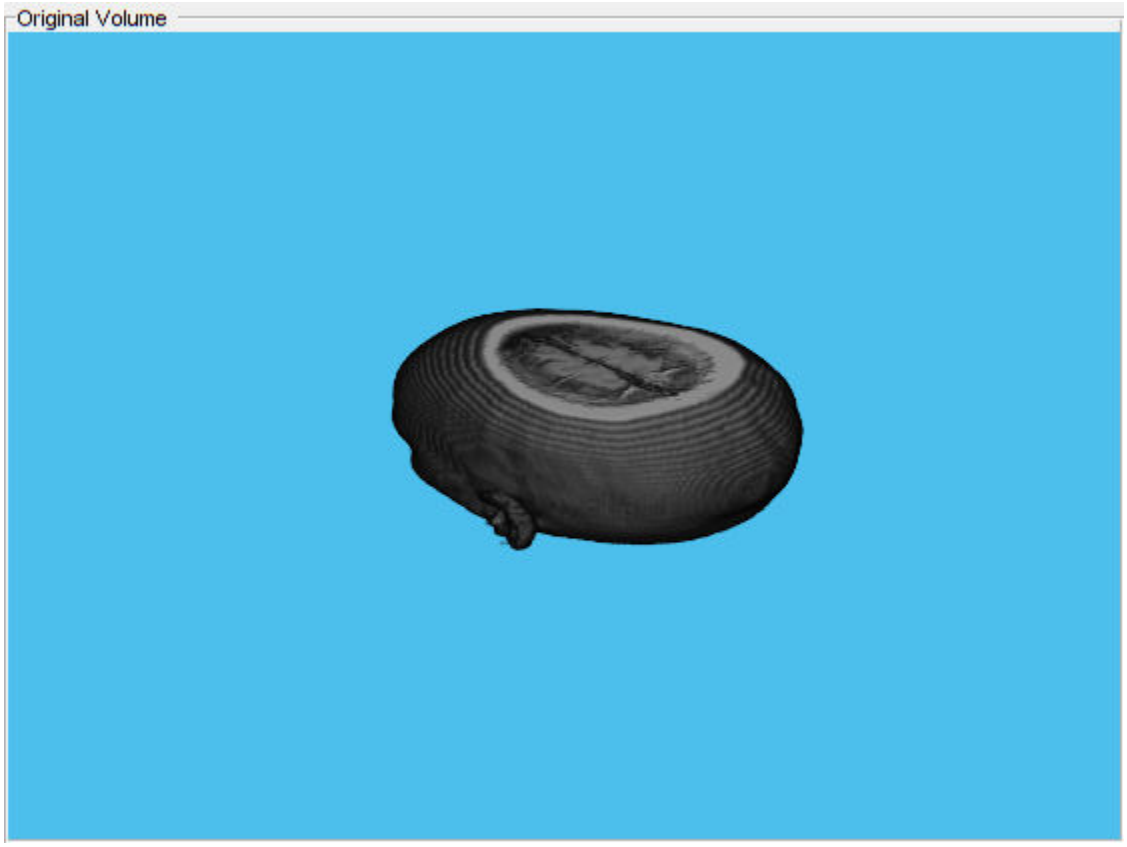
Center Crop 3-D Image to Target Size

Load a 3-D MRI image. Use the `squeeze` function to remove any singleton dimensions.

```
load mri;
D = squeeze(D);
```

Display the image.

```
fullViewPnl = uipanel(figure,'Title','Original Volume');
volshow(D,'Parent',fullViewPnl);
```



Specify the target size of the cropping window.

```
targetSize = [64 64 10];
```

Create a center cropping window that crops the specified image from its center.

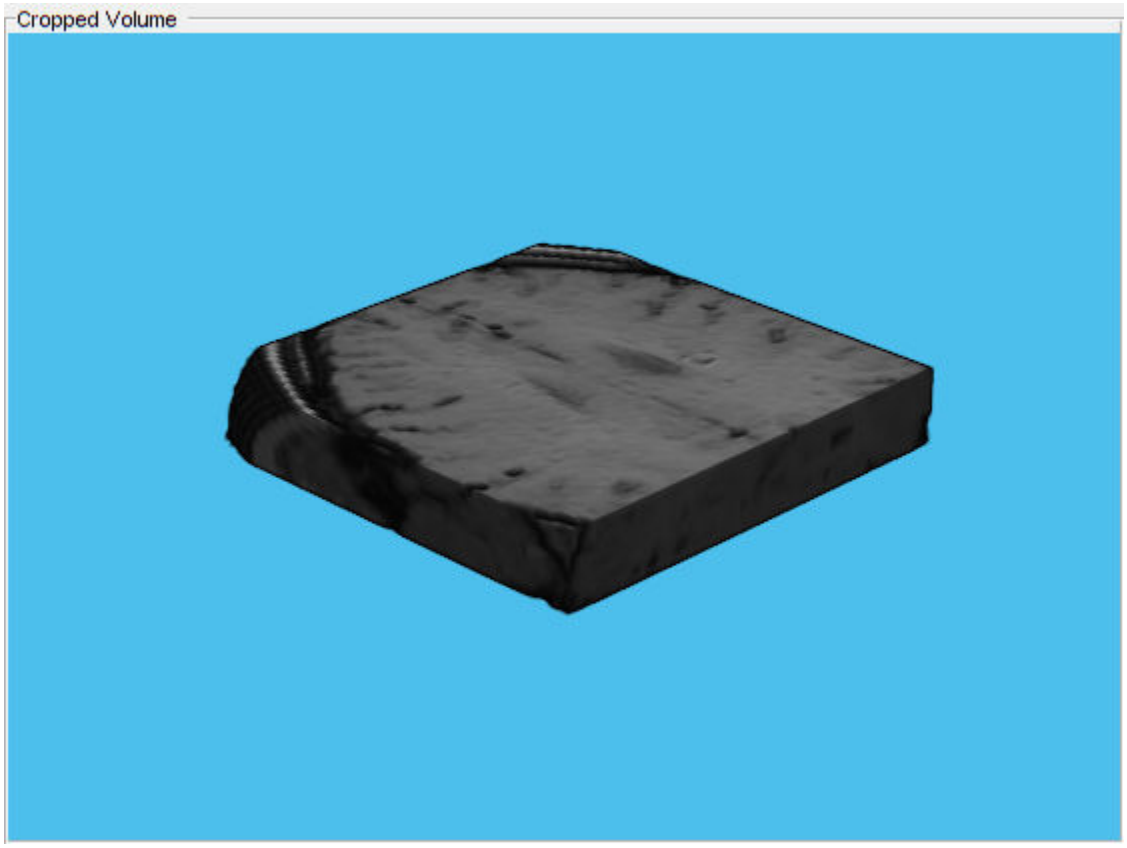
```
win = centerCropWindow3d(size(D),targetSize);
```

Crop the image using the center cropping window.

```
Dcrop = imcrop3(D,win);
```

Display the cropped image in a display panel.

```
fullViewPnl = uipanel(figure,'Title','Cropped Volume');
volshow(Dcrop,'Parent',fullViewPnl);
```



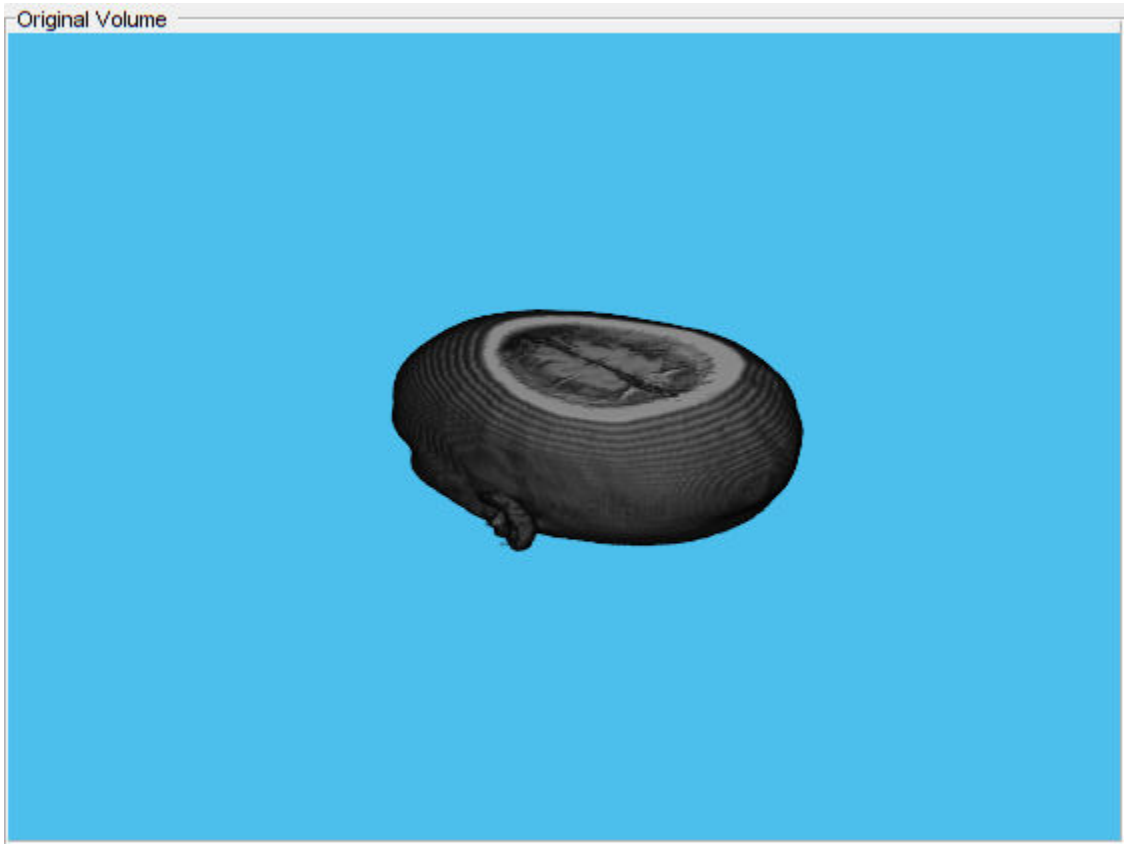
Crop 3-D Image Volume Using Fixed Off-Center Spatial Extent

Load a 3-D MRI image. Use the `squeeze` function to remove any singleton dimensions.

```
S = load('mri.mat','D');
volumeData = squeeze(S.D);
```

Display the image.

```
fullViewPnl = uipanel(figure,'Title','Original Volume');
volshow(volumeData,'Parent',fullViewPnl);
```



Create a `Cuboid` object and specify the cropping window size in all three dimensions.

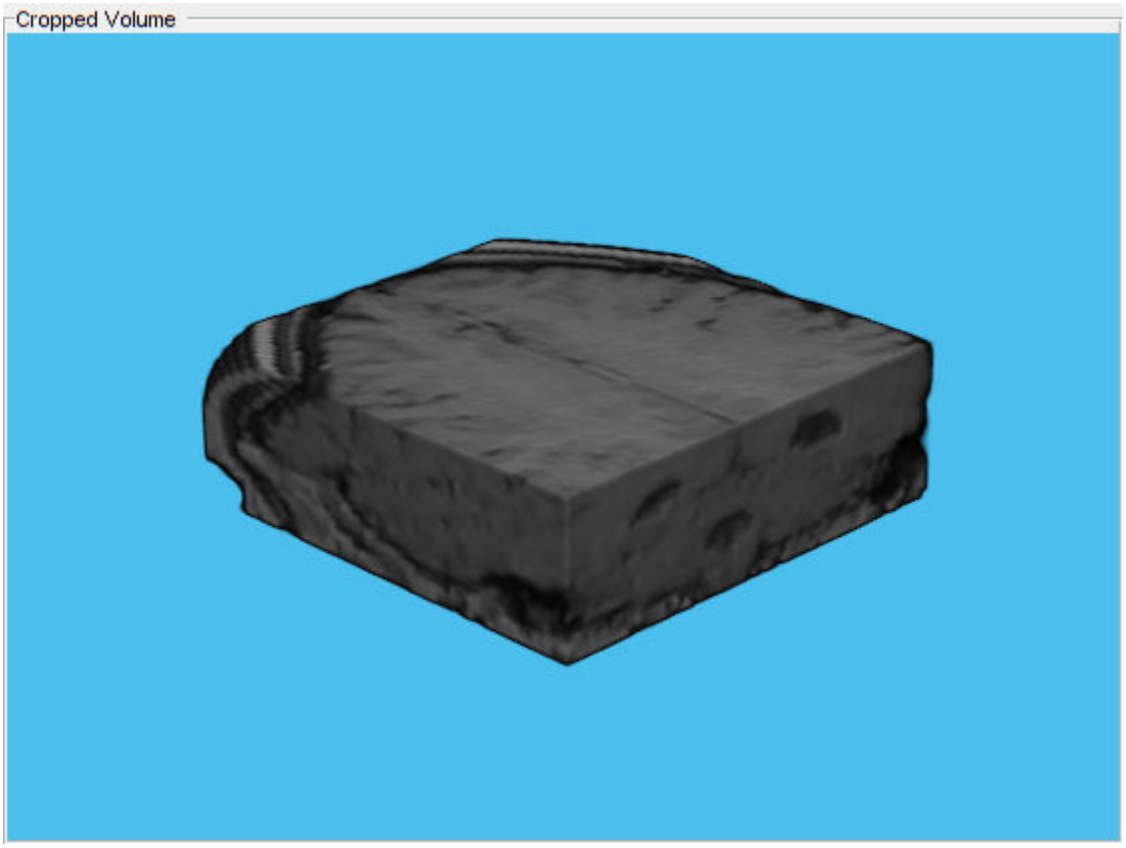
```
c = images.spatialref.Cuboid([30,90],[30,90],[1,20]);
```

Crop the image based on the `Cuboid` dimensions.

```
croppedVolume = imcrop3(volumeData,c);
```

Display the cropped image.

```
fullViewPnl = uipanel(figure,'Title','Cropped Volume');
volshow(croppedVolume,'Parent',fullViewPnl);
```



See Also

`imcrop3` | `centerCropWindow3d` | `randomCropWindow3d` | `Rectangle`

Topics

"Augment Images for Deep Learning Workflows Using Image Processing Toolbox" (Deep Learning Toolbox)

Introduced in R2019b

Rectangle

Spatial extents of 2-D rectangular region

Description

A `Rectangle` object stores the spatial extents of a 2-D rectangular region.

Creation

You can create a `Rectangle` object in these ways.

- `centerCropWindow2d` — Create a `Rectangle` of a specified size whose position is centered on an image of different size.
- `randomWindow2d` — Create a `Rectangle` whose position is selected randomly from within an image of different size. You can specify the size of the rectangle or a range of valid aspect ratios and relative areas of the rectangle.
- Running the command

  ```
  r = images.spatialref.Rectangle(XLimits,YLimits)
  ```

  creates a `Rectangle` object and sets the `XLimits` and `YLimits` properties.

Properties

`XLimits` — Minimum and maximum limits of *x*-axis

2-element numeric vector

Minimum and maximum limits of the *x*-axis, specified as a 2-element numeric vector of the form `[min max]`, where `max` is greater than `min`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`YLimits` — Minimum and maximum limits of *y*-axis

2-element numeric vector

Minimum and maximum limits of the *y*-axis, specified as a 2-element numeric vector of the form `[min max]`, where `max` is greater than `min`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Examples

Center Crop Image Using Spatial Referencing Rectangle

Read and display an image.

```
I = imread('parkavenue.jpg');
imshow(I)
```

Specify a target window size as a two-element vector of the form [*width*, *height*].

```
targetSize = [300 600];
```

Create a `Rectangle` object that specifies the spatial extent of the crop window.

```
r = centerCropWindow2d(size(I),targetSize);
```

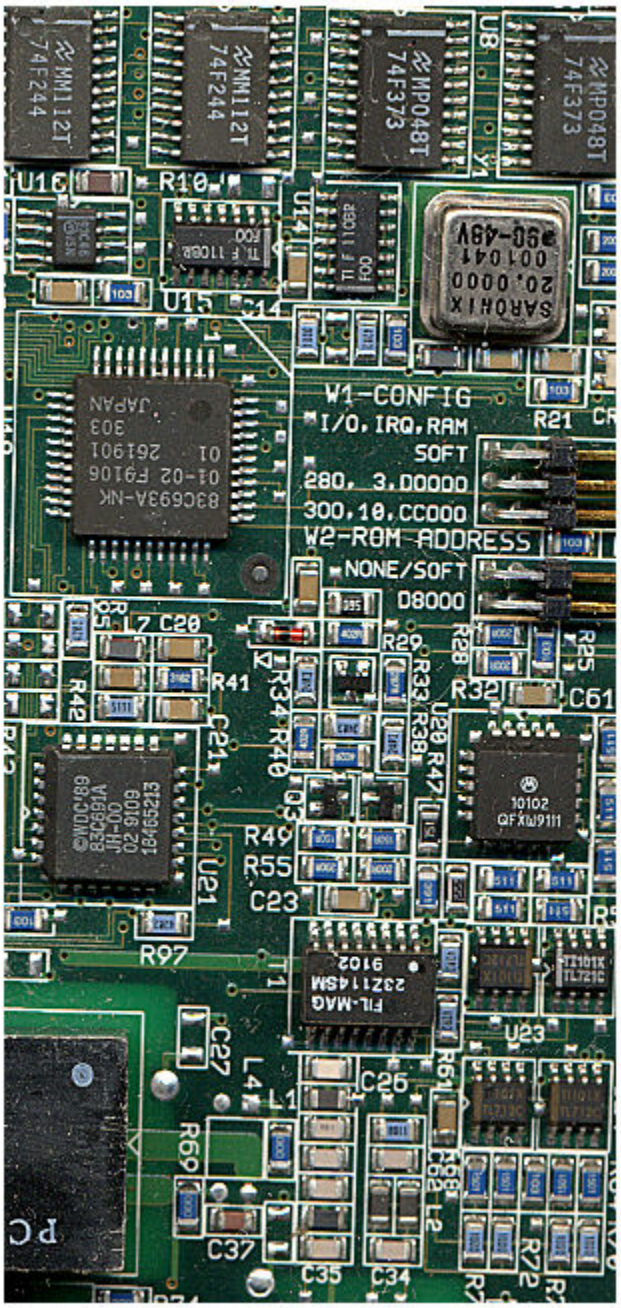
Crop the image to the spatial extents. Display the cropped region.

```
J = imcrop(I,r);
imshow(J)
```

Crop Image Using Fixed Off-Center Spatial Extent

Read and display an image.

```
I = imread('board.tif');
imshow(I)
```



Create a `Rectangle` object by specifying the horizontal and vertical spatial extents of the cropping window.

```
r = images.spatialref.Rectangle([200 300],[50 200])
```

```
r =
  Rectangle with properties:

    XLimits: [200 300]
    YLimits: [50 200]
```

Crop the image to the spatial extents. Display the cropped region.

```
J = imcrop(I,r);
imshow(J)
```

See Also

`Cuboid` | `centerCropWindow2d` | `randomWindow2d` | `imcrop`

Topics

“Augment Images for Deep Learning Workflows Using Image Processing Toolbox” (Deep Learning Toolbox)

Introduced in R2019b

ssim

Structural similarity (SSIM) index for measuring image quality

Syntax

```
ssimval = ssim(A,ref)
ssimval = ssim(A,ref,Name,Value)
[ssimval,ssimmap] = ssim( ___ )
```

Description

`ssimval = ssim(A,ref)` calculates the structural similarity (SSIM) index for grayscale image or volume `A` using `ref` as the reference image or volume. A value closer to 1 indicates better image quality.

`ssimval = ssim(A,ref,Name,Value)` calculates the SSIM, using name-value pairs to control aspects of the computation.

`[ssimval,ssimmap] = ssim( ___ )` also returns the local SSIM value for each pixel or voxel in `A`.

Examples

Calculate Structural Similarity Index (SSIM)

Read an image into the workspace. Create another version of the image, applying a blurring filter.

```
ref = imread("pout.tif");
H = fspecial("Gaussian",[11 11],1.5);
A = imfilter(ref,H,"replicate");
```

Display both images as a montage. The images differ most along sharp high-contrast regions, such as the edges of the trellis.

```
montage({ref,A})
title("Reference Image (Left) vs. Blurred Image (Right)")
```

Calculate the global SSIM value for the image and local SSIM values for each pixel.

```
[ssimval,ssimmap] = ssim(A,ref);
```

Display the local SSIM map. Include the global SSIM value in the figure title. Small values of local SSIM appear as dark pixels in the local SSIM map. Regions with small local SSIM value correspond to areas where the blurred image noticeably differs from the reference image. Large values of local SSIM value appear as bright pixels. Regions with large local SSIM correspond to uniform regions of the reference image, where blurring has less of an impact on the image.

```
imshow(ssimmap,[])
title("Local SSIM Map with Global SSIM Value: "+num2str(ssimval))
```

Calculate SSIM for `dlarray` Input

Read an image into the workspace. Create another version of the image, applying a blurring filter.

```
ref = imread("pout.tif");
A = imgaussfilt(ref,1.5,"FilterSize",11,"Padding","replicate");
```

Display both images as a montage.

```
montage({ref A})
title("Reference Image (Left) vs. Blurred Image (Right)")
```

Simulate batches of images by replicating the reference image and the blurred image 16 times along the 4th dimension.

```
A = repmat(A,[1 1 1 16]);
ref = repmat(ref,[1 1 1 16]);
```

Create formatted `dlarray` objects for the reference image batch and the blurred image batch. The format is "`SSCB`", for spatial-spatial-channel-batch.

```
A = dlarray(single(A),"SSCB");
ref = dlarray(single(ref),"SSCB");
```

Calculate the global SSIM value for the image and local SSIM values for each pixel. `ssimVal` returns a scalar SSIM value for each image in the batch. `ssimMap` returns a map of SSIM values, the same size as the image, for each image in the batch.

```
[ssimVal,ssimMap] = ssim(A,ref);
size(ssimVal)
```

```
ans = 1×4

     1     1     1    16
```

```
size(ssimMap)
```

```
ans = 1×4
```

```
   291   240     1    16
```

Input Arguments

`A` — Image for quality measurement

numeric array | `dlarray` object

Image for quality measurement, specified as a numeric array or a `dlarray` object. If `A` is not a 2-D grayscale image or 3-D grayscale volume, such as an RGB image or stack of grayscale images, specify the `DataFormat` name-value argument. Do not specify the `DataFormat` name-value argument if `A` is a formatted `dlarray` object.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`ref` — Reference image

numeric array | `dlarray` object

Reference image against which to measure quality, specified as a numeric array or a `dlarray` object of the same size and data type as `A`. If `ref` is not a 2-D grayscale image or 3-D grayscale volume, such as an RGB image or stack of grayscale images, specify the `DataFormat` name-value argument. Do not specify the `DataFormat` name-value argument if `ref` is a formatted `dlarray` object.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `ssim(A,ref,"DynamicRange",100)`

`DataFormat` — Dimension labels

string scalar | character vector

Dimension labels of the input images `A` and `ref`, specified as a string scalar or character vector. Each character in `DataFormat` must be one of these labels:

- `S` — Spatial
- `C` — Channel
- `B` — Batch observations

The format cannot include more than one channel label or batch label. Do not specify the `DataFormat` name-value argument when the input images are formatted `dlarray` objects.

Example: `"SSC"` indicates that the array has two spatial dimensions and one channel dimension, appropriate for 2-D RGB image data.

Example: `"SSCB"` indicates that the array has two spatial dimensions, one channel dimension, and one batch dimension, appropriate for a sequence of 2-D RGB image data.

Data Types: `char` | `string`

`DynamicRange` — Dynamic range of the input image
`diff(getrangefromclass(A))` (default) | positive scalar

Dynamic range of the input image, specified as a positive scalar. The default value of `"DynamicRange"` depends on the data type of image `A`, and is calculated as `diff(getrangefromclass(A))`. For example, the default dynamic range is `255` for images of data type `uint8`, and the default is `1` for images of data type `double` or `single` with pixel values in the range [0, 1].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`Exponents` — Exponents for luminance, contrast, and structural terms
`[1 1 1]` (default) | 3-element vector of nonnegative numbers

Exponents for the luminance, contrast, and structural terms, specified as a 3-element vector of nonnegative numbers of the form `[alpha beta gamma]`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`Radius` — Standard deviation of isotropic Gaussian function
`1.5` (default) | positive number

Standard deviation of isotropic Gaussian function, specified as a positive number. This value is used for weighting the neighborhood pixels around a pixel for estimating local statistics. This weighting is used to avoid blocking artifacts in estimating local statistics.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`RegularizationConstants` — Regularization constants for luminance, contrast, and structural terms
3-element vector of nonnegative numbers

Regularization constants for the luminance, contrast, and structural terms, specified as a 3-element vector of nonnegative numbers of the form `[c1 c2 c3]`. The `ssim` function uses these regularization constants to avoid instability for image regions where the local mean or standard deviation is close to zero. Therefore, small non-zero values should be used for these constants.

By default,

- `C1 = (0.01*L).^2`, where `L` is the specified `DynamicRange` value.
- `C2 = (0.03*L).^2`, where `L` is the specified `DynamicRange` value.
- `C3 = C2/2`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output Arguments

`ssimval` — SSIM index
numeric scalar | numeric array | `dlarray` object

SSIM index, returned as one of these values.

Input Image Type	SSIM Value
• Unformatted numeric arrays • Formatted numeric arrays with neither a channel (`"C"`) nor batch (`"B"`) dimension	Numeric scalar with a single SSIM measurement.
• Unformatted `dlarray` objects	Scalar `dlarray` object with a single SSIM measurement.
• Numeric arrays with a channel or batch dimension specified using the `DataFormat` name-value argument	Numeric array of the same dimensionality as the input images. The spatial dimensions of `ssimval` are singleton dimensions. There is one SSIM measurement for each element along any channel or batch dimension.
• Formatted `dlarray` objects with a channel or batch dimension • Unformatted `dlarray` objects with a channel or batch dimension specified using the `DataFormat` name-value argument	`dlarray` object of the same dimensionality as the input images. The spatial dimensions of `ssimval` are singleton dimensions. There is one SSIM measurement for each element along any channel or batch dimension.

`ssimval` is of data type `double` except when `A` is of data type `single`, in which case `ssimval` is of data type `single`.

The value of `ssimval` is typically in the range [0, 1]. The value 1 indicates the highest quality and occurs when `A` and `ref` are equivalent. Smaller values correspond to poorer quality. For some combinations of inputs and name-value pair arguments, `ssimval` can be negative.

`ssimmap` — Local values of SSIM index
numeric array | `dlarray` object

Local values of the SSIM index, returned as one of these values.

Input Image Type	SSIM Value
• Unformatted numeric arrays • Formatted numeric arrays with neither a channel (`"C"`) nor batch (`"B"`) dimension	Numeric array the same size as the input images. There is one SSIM measurement for each element in the input image.
• Unformatted `dlarray` objects	`dlarray` object the same size as the input images. There is one SSIM measurement for each element in the input image.
• Numeric arrays with a channel or batch dimension specified using the `DataFormat` name-value argument	Numeric array the same size as the input images. Each spatial element in the input image has an SSIM measurement along any channel or batch dimension.
• Formatted `dlarray` objects with a channel or batch dimension • Unformatted `dlarray` objects with a channel or batch dimension specified using the `DataFormat` name-value argument	`dlarray` object the same size as the input images. Each spatial element in the input image has an SSIM measurement along any channel or batch dimension.

`ssimmap` is of data type `double` except when `A` is of data type `single`, in which case `ssimmap` is of data type `single`.

More About

Structural Similarity Index

An image quality metric that assesses the visual impact of three characteristics of an image: luminance, contrast and structure.

Tips

- If `A` and `ref` specify RGB image data, use the `"DataFormat"` name-value argument to label the channel dimension, `"C"`. You can then apply the `mean` function along the channel dimension of `ssimval` and `ssimmap` to approximate the SSIM index for the overall image.

Algorithms

The SSIM Index quality assessment index is based on the computation of three terms, namely the luminance term, the contrast term and the structural term. The overall index is a multiplicative combination of the three terms.

$$SSIM(x,y) = [l(x,y)]^{\alpha} \cdot [c(x,y)]^{\beta} \cdot [s(x,y)]^{\gamma}$$

where

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1},$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2},$$

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}$$

where μ_x, μ_y, σ_x,σ_y, and σ_{xy} are the local means, standard deviations, and cross-covariance for images x, y. If $\alpha = \beta = \gamma = 1$ (the default for `Exponents`), and $C_3 = C_2/2$ (default selection of C_3) the index simplifies to:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

When you specify a noninteger value for `"Exponents"`, the `ssim` function prevents complex valued outputs by clamping the intermediate luminance, contrast, and structural terms to the range [0, `inf`].

References

[1] Zhou, W., A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. "Image Quality Assessment: From Error Visibility to Structural Similarity." *IEEE Transactions on Image Processing*. Vol. 13, Issue 4, April 2004, pp. 600–612.

Extended Capabilities

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`psnr` | `immse` | `multissim` | `multissim3`

Topics

“Compare Image Quality at Various Compression Levels”
“List of Functions with dlarray Support” (Deep Learning Toolbox)
“Define Custom Training Loops, Loss Functions, and Networks” (Deep Learning Toolbox)

Introduced in R2014a

std2

Standard deviation of matrix elements

Syntax

```
B = std2(A)
```

Description

`B = std2(A)` computes the standard deviation of all values in array `A`.

Examples

Compute 2-D Standard Deviation

Read a grayscale image into the workspace, then calculate the standard deviation of the pixel intensity values.

```
I = imread('liftingbody.png');
val = std2(I)
```

```
val = 31.6897
```

Input Arguments

`A` — Input data

numeric array | logical array

Input data, specified as a numeric or logical array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

`B` — Standard deviation

numeric scalar

Standard deviation of input data, returned as a numeric scalar. If the data type of `A` is `single`, then the data type of `B` is also `single`. Otherwise, the data type of `B` is `double`.

Data Types: `single` | `double`

Extended Capabilities

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`corr2` | `mean` | `mean2` | `std`

Introduced before R2006a

stdfilt

Local standard deviation of image

Syntax

```
J = stdfilt(I)
J = stdfilt(I,nhood)
```

Description

`J = stdfilt(I)` performs standard deviation filtering of image `I` and returns the filtered image `J`. The value of each output pixel is the standard deviation of the 3-by-3 neighborhood around the corresponding input pixel. For pixels on the borders of `I`, `stdfilt` uses symmetric padding. In symmetric padding, the values of padding pixels are a mirror reflection of the border pixels in `I`.

`J = stdfilt(I,nhood)` specifies the neighborhood, `nhood`, used to compute the standard deviation.

Examples

Perform Standard Deviation Filtering

This example shows how to perform standard deviation filtering using `stdfilt`. Brighter pixels in the filtered image correspond to neighborhoods in the original image with larger standard deviations.

Read an image into the workspace.

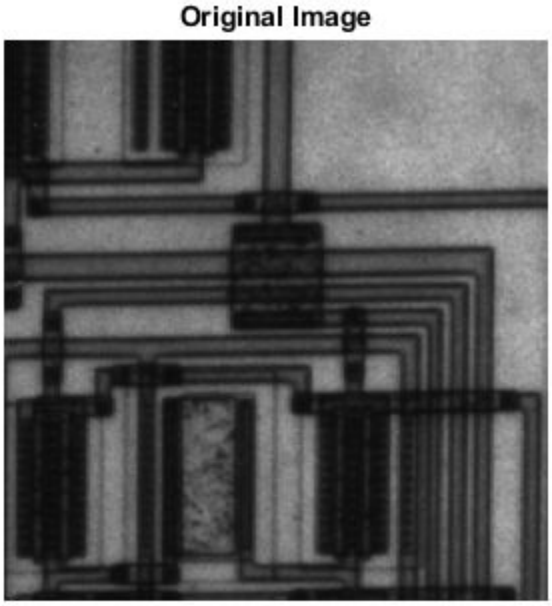
```
I = imread('circuit.tif');
```

Perform standard deviation filtering using `stdfilt`.

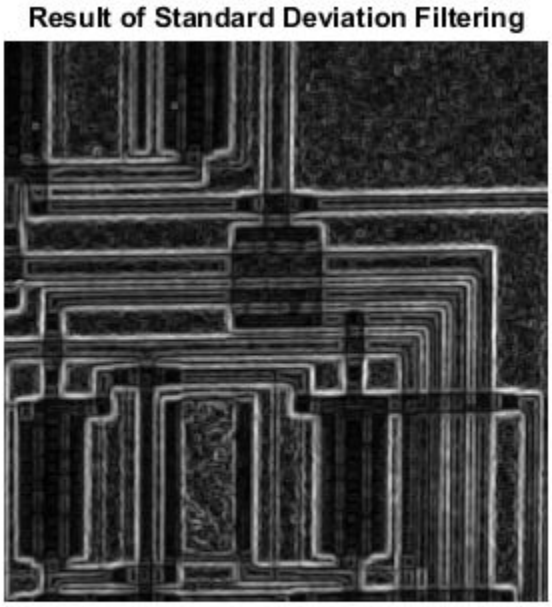
```
J = stdfilt(I);
```

Show the original image and the processed image.

```
imshow(I)
title('Original Image')
```




```
figure
imshow(J,[])
title('Result of Standard Deviation Filtering')
```



Input Arguments

`I` — Image to be filtered

numeric array | logical array

Image to be filtered, specified as a numeric array or logical array of any dimension.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`nhood` — Neighborhood

`true(3)` (default) | numeric array | logical array

Neighborhood, specified as a numeric or logical array containing `0`s and `1`s. The size of `nhood` must be odd in each dimension.

By default, `stdfilt` uses the neighborhood `true(3)`. `stdfilt` determines the center element of the neighborhood by `floor((size(nhood) + 1)/2)`.

To specify neighborhoods of various shapes, such as a disk, use the `strel` function to create a structuring element object of the desired shape. Then extract the neighborhood from the `neighborhood` property of the structuring element.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

`J` — Filtered image

numeric array

Filtered image, returned as a numeric array of the same size as the input image `I`. The data type of `J` is `double`.

Tips

- The `double` array `J` contains standard deviation values, which can exceed the range [0, 1]. Because some Image Processing Toolbox functions expect inputs of type `double` to be in the range [0, 1], to pass `J` as an input argument to these functions, use the `rescale` function to rescale the values of `J` to [0, 1].
- If the image contains `Inf`s or `NaN`s, then the behavior of `stdfilt` is undefined. Propagation of `Inf`s or `NaN`s might not be localized to the neighborhood around the `Inf` or `NaN` pixel.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`stdfilt` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Usage notes and limitations:

- The filtering neighborhood must be two-dimensional.

For more information, see “Image Processing on a GPU”.

See Also

Functions
`entropyfilt` | `getnhood` | `rangefilt` | `std2`

Objects
`strel` | `offsetstrel`

Topics
“What Is Image Filtering in the Spatial Domain?”
“Image Types in the Toolbox”

Introduced before R2006a

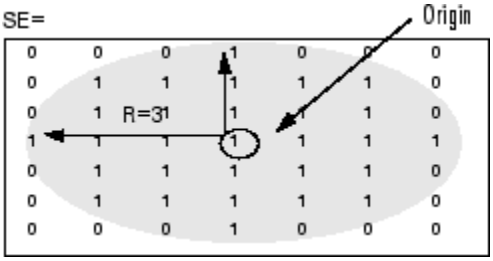
strel

Morphological structuring element

Description

A `strel` object represents a flat morphological structuring element, which is an essential part of morphological dilation and erosion operations.

A flat structuring element is a binary valued neighborhood, either 2-D or multidimensional, in which the `true` pixels are included in the morphological computation, and the `false` pixels are not. The center pixel of the structuring element, called the *origin*, identifies the pixel in the image being processed. Use the `strel` function (described below) to create a flat structuring element. You can use flat structuring elements with both binary and grayscale images. The following figure illustrates a flat structuring element.



To create a nonflat structuring element, use `offsetstrel`.

Creation

Syntax

```
SE = strel(nhood)

SE = strel("diamond",r)
SE = strel("disk",r)
SE = strel("disk",r,n)
SE = strel("octagon",r)
SE = strel("line",len,deg)
SE = strel("rectangle",[m n])
SE = strel("square",w)

SE = strel("cube",w)
SE = strel("cuboid",[m n p])
SE = strel("sphere",r)
```

Description

Arbitrary Neighborhood Shape

`SE = strel(nhood)` creates a flat structuring element with specified neighborhood `nhood`.

2-D Geometric Neighborhood Shapes

`SE = strel("diamond",r)` creates a diamond-shaped structuring element, where `r` specifies the distance from the structuring element origin to the points of the diamond.

`SE = strel("disk",r)` creates a disk-shaped structuring element, where `r` specifies the radius.

`SE = strel("disk",r,n)` creates a disk-shaped structuring element, where `r` specifies the radius and `n` specifies the number of line structuring elements used to approximate the disk shape. Morphological operations run much faster when the structuring element uses approximations.

`SE = strel("octagon",r)` creates a octagonal structuring element, where `r` specifies the distance from the structuring element origin to the sides of the octagon, as measured along the horizontal and vertical axes. `r` must be a nonnegative multiple of 3.

`SE = strel("line",len,deg)` creates a linear structuring element that is symmetric with respect to the neighborhood center, with approximate length `len` and angle `deg`.

`SE = strel("rectangle",[m n])` creates a rectangular structuring element of size `[m n]`.

`SE = strel("square",w)` creates a square structuring element whose width is `w` pixels.

3-D Geometric Neighborhood Shapes

`SE = strel("cube",w)` creates a 3-D cubic structuring element whose width is `w` pixels.

`SE = strel("cuboid",[m n p])` creates a 3-D cuboidal structuring element of size *m*-by-*n*-by-*p* pixels.

`SE = strel("sphere",r)` creates a 3-D spherical structuring element whose radius is `r` pixels.

Compatibility

The following syntaxes still work, but `offsetstrel` is the preferred way to create these nonflat structuring element shapes:

- `SE = strel("arbitrary",nhood,h)`, where `h` is a matrix of the same size as `nhood` containing the height values associated with each nonzero element of `nhood`.
- `SE = strel("ball",r,h,n)`

Input Arguments

nhood — Neighborhood

numeric array

Neighborhood, specified as numeric array of any dimension. All nonzero pixels of `nhood` belong to the neighborhood for the morphological operation. The center (or origin) of `nhood` is its center element, given by `floor((size(nhood) + 1)/2)`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`r` — Radius of structuring element

positive integer

Radius of the structuring element, specified as a positive integer.

- For the disk shape, `r` is the distance from the origin to the edge of the disk.
- For the diamond shape, `r` is the distance from the structuring element origin to the points of the diamond.
- For the octagon shape, `r` is the distance from the structuring element origin to the sides of the octagon, as measured along the horizontal and vertical axes. `r` must be a multiple of 3.
- For the sphere shape, `r` is the distance from the origin to the edge of the sphere.

Data Types: `double`

`n` — Number of periodic line structuring elements used to approximate shape

`4` (default) | `0` | `6` | `8`

Number of periodic line structuring elements used to approximate shape, specified as `0`, `4`, `6`, or `8`. Morphological operations using disk approximations run much faster when the structuring element uses approximations (`n` > 0).

Value of `n`	Behavior
`n` > 0	`strel` uses a sequence of `n` periodic line-shaped structuring elements to approximate the shape. Sometimes `strel` must use two extra line structuring elements in the approximation, in which case the actual number of decomposed structuring elements is `n+2`.
`n` = 0	`strel` does not use any approximation. The structuring element members comprise all pixels whose centers are no greater than `r` away from the origin.

Data Types: `double`

`len` — Length of linear structuring element

positive number

Length of linear structuring element, specified as a positive number. `len` is approximately the distance between the centers of the structuring element members at opposite ends of the line.

Data Types: `double`

`deg` — Angle of linear structuring element

numeric scalar

Angle of linear structuring element, in degrees, specified as numeric scalar. The angle is measured in a counterclockwise direction from the horizontal axis.

Data Types: `double`

`[m n]` — Size of rectangular structuring element

2-element vector of positive integers

Size of rectangular structuring element, specified as a 2-element vector of positive integers. The structuring element has *m* rows and *n* columns.

Data Types: `double`

`w` — Width of square or cubic structuring element
positive integer

Width of square or cubic structuring element, specified as a positive integer.

Data Types: `double`

`[m n p]` — Size of cuboidal structuring element
3-element vector of positive integers

Size of cuboidal structuring element, specified as a 3-element vector of positive integers. The structuring element has *m* rows, *n* columns, and *p* planes.

Data Types: `double`

Properties

`Neighborhood` — Structuring element neighborhood
logical array

Structuring element neighborhood, specified as a logical array.

Data Types: `logical`

`Dimensionality` — Dimensions of structuring element
nonnegative scalar

Dimensions of structuring element, specified as a nonnegative scalar.

Data Types: `double`

Object Functions

imdilate	Dilate image
imerode	Erode image
imclose	Morphologically close image
imopen	Morphologically open image
imbothat	Bottom-hat filtering
imtophat	Top-hat filtering
bwhitmiss	Binary hit-miss operation
decompose	Return sequence of decomposed structuring elements
reflect	Reflect structuring element
translate	Translate structuring element

Examples

Create Square Structuring Element

Create an 11-by-11 square structuring element.

```
SE = strel('square', 11)
```

```
SE =
strel is a square shaped structuring element with properties:
```

```
      Neighborhood: [11x11 logical]
    Dimensionality: 2
```

Create Line-Shaped Structuring Element

Create a line-shaped structuring element with a length of 10 at an angle of 45 degrees.

```
SE = strel('line', 10, 45)
```

```
SE =
strel is a line shaped structuring element with properties:

      Neighborhood: [7x7 logical]
    Dimensionality: 2
```

View the structuring element.

```
SE.Neighborhood
```

```
ans = 7x7 logical array

   0   0   0   0   0   0   1
   0   0   0   0   0   1   0
   0   0   0   0   1   0   0
   0   0   0   1   0   0   0
   0   0   1   0   0   0   0
   0   1   0   0   0   0   0
   1   0   0   0   0   0   0
```

Create Disk-Shaped Structuring Element

Create a disk-shaped structuring element with a radius of 15.

```
SE3 = strel('disk', 15)
```

```
SE3 =
strel is a disk shaped structuring element with properties:

      Neighborhood: [29x29 logical]
    Dimensionality: 2
```

Display the disk-shaped structuring element.

```
figure
imshow(SE3.Neighborhood)
```

Create 3-D Sphere-shaped Structuring Element

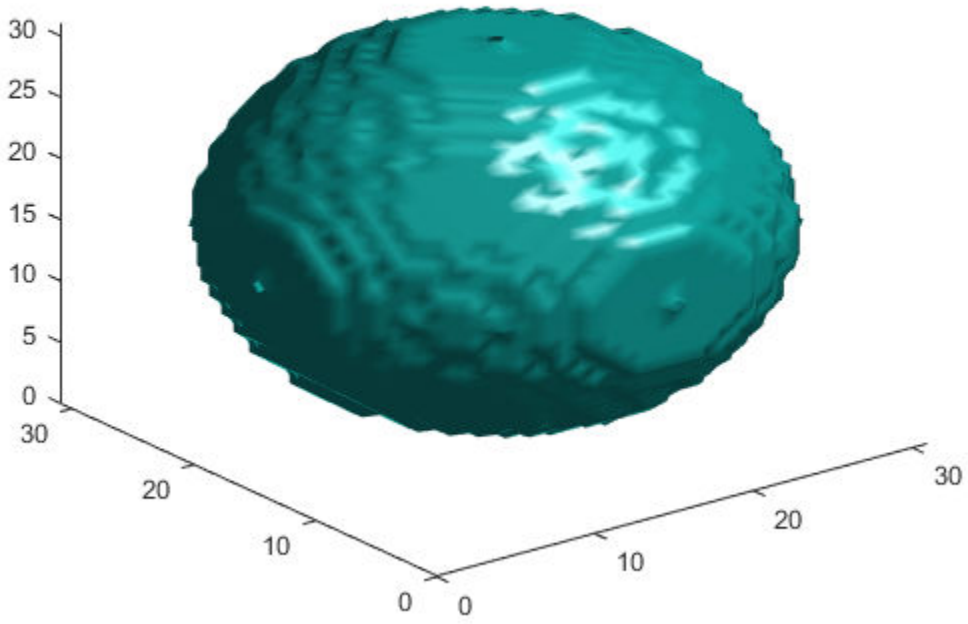
Create a 3-D sphere-shaped structuring element with a radius of 15.

```
SE = strel('sphere', 15)
```

```
SE =
strel is a sphere shaped structuring element with properties:

      Neighborhood: [31x31x31 logical]
    Dimensionality: 3
```

Display the structuring element.

```
figure
isosurface(SE.Neighborhood)
```



Tips

- Structuring elements that do not use approximations (`n = 0`) are not suitable for computing granulometries.

Algorithms

For all of the geometrical shapes, structuring elements are constructed using a family of techniques known collectively as *structuring element decomposition*. The principle is that dilation by some large structuring elements can be computed faster by dilation with a sequence of smaller structuring elements. For example, dilation by an 11-by-11 square structuring element can be accomplished by dilating first with a 1-by-11 structuring element and then with an 11-by-1 structuring element. This results in a theoretical performance improvement of a factor of 5.5, although in practice the actual performance improvement is somewhat less. Structuring element decompositions used for the `"disk"` shape is an approximations—all other decompositions are exact.

Compatibility Considerations

Linear Structuring Elements Use Angle in Range [0, 180]
Behavior changed in R2017b

Starting in R2017b, `strel` constrains linear structuring elements to have an angle in the range [0, 180]. If you specify a value of `deg` outside this range, then `strel` calculates the angle as `mod(deg,180)`.

Prior to R2017b, in some situations, `strel` would create different linear structuring elements for angles that differ by a factor of 180 degrees.

References

[1] van den Boomgard, R, and R. van Balen, "Methods for Fast Morphological Image Transforms Using Bitmapped Images," *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*, Vol. 54, Number 3, pp. 252–254, May 1992.

[2] Adams, R., "Radial Decomposition of Discs and Spheres," *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*, Vol. 55, Number 5, pp. 325–332, September 1993.

[3] Jones, R., and P. Soille, "Periodic lines: Definition, cascades, and application to granulometrie," *Pattern Recognition Letters*, Vol. 17, pp. 1057–1063, 1996.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `strel` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- All input arguments of type `char` must be compile-time constants.
- The methods associated with `strel` objects are not supported in code generation.
- Arrays of `strel` objects are not supported.

See Also

`offsetstrel`

Topics

“Structuring Elements”

Introduced before R2006a

stretchlim

Find limits to contrast stretch image

Syntax

```
lowhigh = stretchlim(I)
lowhigh = stretchlim(I,Tol)
```

Description

`lowhigh = stretchlim(I)` computes the lower and upper limits that can be used for contrast stretching grayscale or RGB image `I`. The limits are returned in `lowhigh`. By default, the limits specify the bottom 1% and the top 1% of all pixel values.

`lowhigh = stretchlim(I,Tol)` specifies the fraction, `Tol`, of the image to saturate at low and high pixel values.

Examples

Find Limits to Stretch Contrast in Grayscale Image

Read grayscale image into the workspace and display it.

```
I = imread('pout.tif');
figure
imshow(I)
```

Adjust the contrast in the image using `stretchlim` to set the limits, and display the result. The example uses the default limits `[0.01 0.99]`, saturating the upper 1% and the lower 1%.

```
J = imadjust(I,stretchlim(I),[]);
figure
imshow(J)
```

Input Arguments

`I` — Image to be contrast stretched

2-D grayscale image | 2-D RGB image

Image to be contrast stretched, specified as a 2-D grayscale image or 2-D RGB image.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`Tol` — Fraction of image to saturate

`[0.01 0.99]` (default) | numeric scalar | 2-element numeric vector

Fraction of the image to saturate, specified as a numeric scalar or 2-element vector `[Low_Fract High_Fract]` in the range [0, 1].

Value	Description
Scalar	If `Tol` is a scalar, then `Low_Fract = Tol`, and `High_Fract = 1 - Low_Fract`, which saturates equal fractions at low and high pixel values.
0	If `Tol = 0`, then `lowhigh = [min(I(:)); max(I(:))]`.
Default	If you omit the `Tol` argument, then `[Low_Fract High_Fract]` defaults to `[0.01 0.99]`, saturating 2%.
Too big	If `Tol` is too big, such that no pixels would be left after saturating low and high pixel values, then `stretchlim` returns `[0 1]`.

Example: `[.02 .80]`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`lowhigh` — Lower and upper limits for contrast stretching

2-element numeric vector | 2-by-3 numeric matrix

Lower and upper limits for contrast stretching, returned as one of the following.

- A 2-element numeric vector when `I` is a grayscale image.
- A 2-by-3 numeric matrix when `I` is an RGB image. The columns indicate the lower and upper limit for each of the three color channels.

Data Types: `double`

Tips

- Use the `imadjust` function to adjust the contrast of image `I` using the limits, `lowhigh`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `stretchlim` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `stretchlim` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`brighten` | `decorrstretch` | `histeq` | `imadjust`

Introduced before R2006a

subimage

Display multiple images in single figure

Note `subimage` is not recommended. Use `imshow` with `tiledlayout` to display multiple images in the same figure window. For more information, see “Compatibility Considerations”.

Syntax

```
subimage(I)
subimage(X,map)
subimage(x,y, ___ )
h = subimage( ___ )
```

Description

`subimage(I)` displays the RGB (truecolor), grayscale, or binary image `I` in the current axes.

You can use `subimage` in conjunction with `subplot` to create figures with multiple images, even if the images have different colormaps. `subimage` converts images to RGB for display purposes, thus avoiding colormap conflicts.

`subimage(X,map)` displays the indexed image `X` with colormap `map` in the current axes.

`subimage(x,y, ___ )` displays an image using a nondefault spatial coordinate system, where `x` and `y` specify the image limits in the world coordinate system.

`h = subimage( ___ )` returns a handle to an image object.

Examples

Display Two Indexed Images in Same Figure

```
load trees
[X2,map2] = imread('forest.tif');
subplot(1,2,1), subimage(X,map)
subplot(1,2,2), subimage(X2,map2)
```

Input Arguments

`I` — Image to display

RGB image | grayscale image | binary image

Image to display, specified as an RGB (truecolor), grayscale, or binary image.

Data Types: `double` | `uint8` | `uint16` | `logical`

`X` — Indexed image to display

m-by-*n* matrix of integers

Indexed image, specified as an *m*-by-*n* matrix of integers.

- If you specify `X` as an array of integer data type, then the value 0 corresponds to the first color in the colormap `map`. For a colormap containing *c* colors, values of image `X` are clipped to the range [0, *c*-1].
- If you specify `X` as an array of data type `double`, then the value 1 corresponds to the first color in the colormap. For a colormap containing *c* colors, values of image `X` are clipped to the range [1, *c*].

Data Types: `double` | `uint8` | `uint16` | `logical`

`map` — Colormap
c-by-3 matrix

Colormap associated with indexed image `X`, specified as a *c*-by-3 matrix with values in the range [0, 1]. Each row of `map` is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap.

Data Types: `double`

`x` — Image limits in *x* direction
2-element numeric vector

Image limits in the *x* direction in world coordinates, specified as a 2-element numeric vector of the form `[xmin xmax]`. The value of `x` sets the image `XData`.

`y` — Image limits in *y* direction
2-element numeric vector

Image limits in the *y* direction in world coordinates, specified as a 2-element numeric vector of the form `[ymin ymax]`. The value of `y` sets the image `YData`.

Output Arguments

`h` — Handle to image object
handle

Handle to an image graphics object, specified as a handle.

Compatibility Considerations

`subimage` is not recommended
Not recommended starting in R2016b

Before R2016b, `imshow` set the colormap of a figure window, and all axes within the figure would have an identical colormap. `subimage` was introduced in R2006a as a workaround to display multiple images with different colormaps in the same figure. However, `subimage` does not provide all of the syntaxes and options that `imshow` provides, such as the ability to specify the display range.

In R2016b, `imshow` was enhanced so that images displayed within a figure could have different colormaps. This enhancement renders the `subimage` function irrelevant. There are no plans to remove `subimage` at this time.

To update your code, replace instances of `subimage` with `imshow`. You do not need to change the input arguments.

See Also

`imshow` | `subplot` | `montage`

Introduced before R2006a

superpixels

2-D superpixel oversegmentation of images

Syntax

```
[L,NumLabels] = superpixels(A,N)
[L,NumLabels] = superpixels(A,N,Name,Value)
```

Description

`[L,NumLabels] = superpixels(A,N)` computes superpixels of the 2-D grayscale or RGB image `A`. `N` specifies the number of superpixels you want to create. The function returns `L`, a label matrix of type `double`, and `NumLabels`, the actual number of superpixels that were computed.

The `superpixels` function uses the simple linear iterative clustering (SLIC) algorithm [1]. This algorithm groups pixels into regions with similar values. Using these regions in image processing operations, such as segmentation, can reduce the complexity of these operations.

`[L,NumLabels] = superpixels(A,N,Name,Value)` computes superpixels of image `A` using name-value pair arguments used to control aspects of the segmentation.

Examples

Compute Superpixels of Input RGB Image

Read image into the workspace.

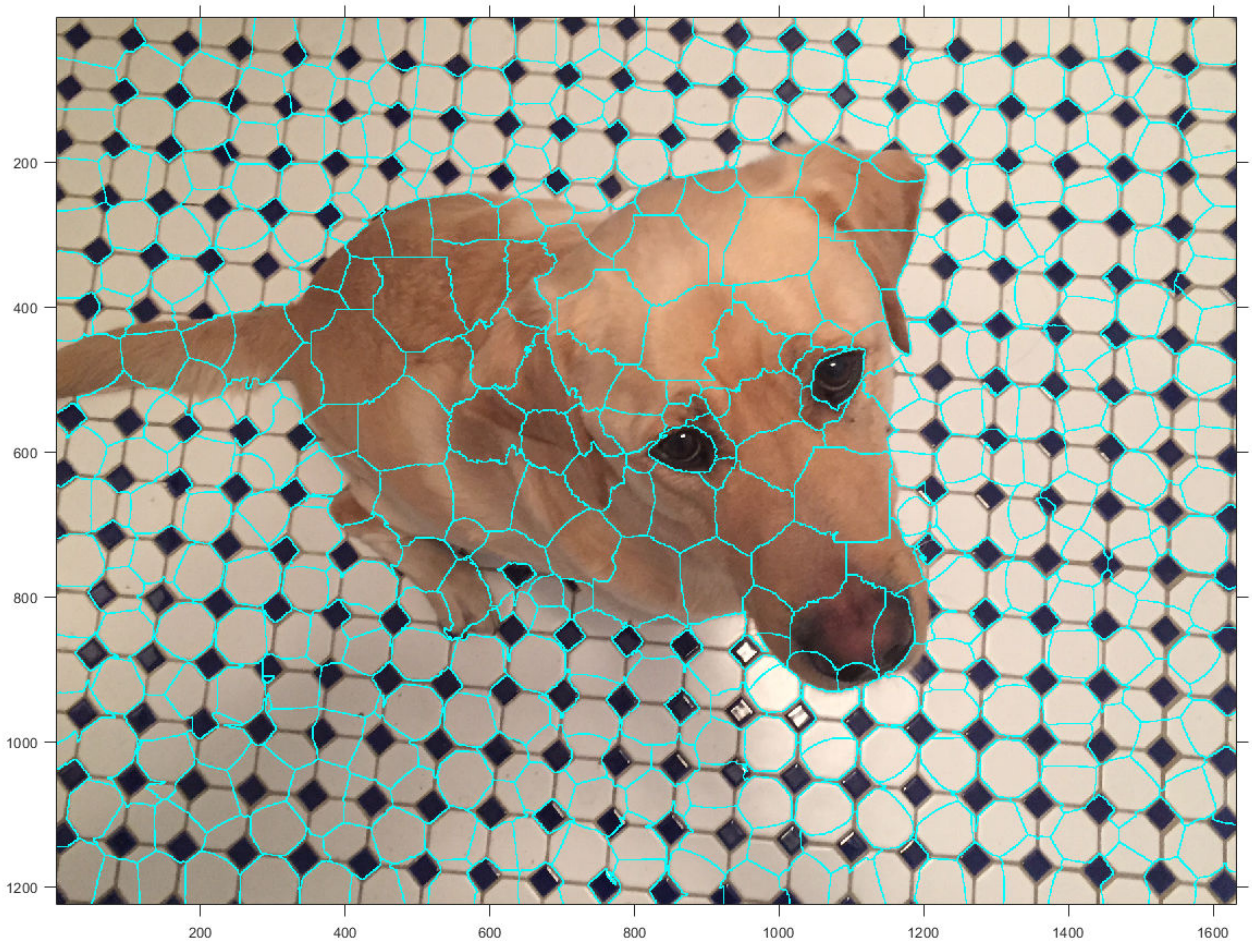
```
A = imread('kobi.png');
```

Calculate superpixels of the image.

```
[L,N] = superpixels(A,500);
```

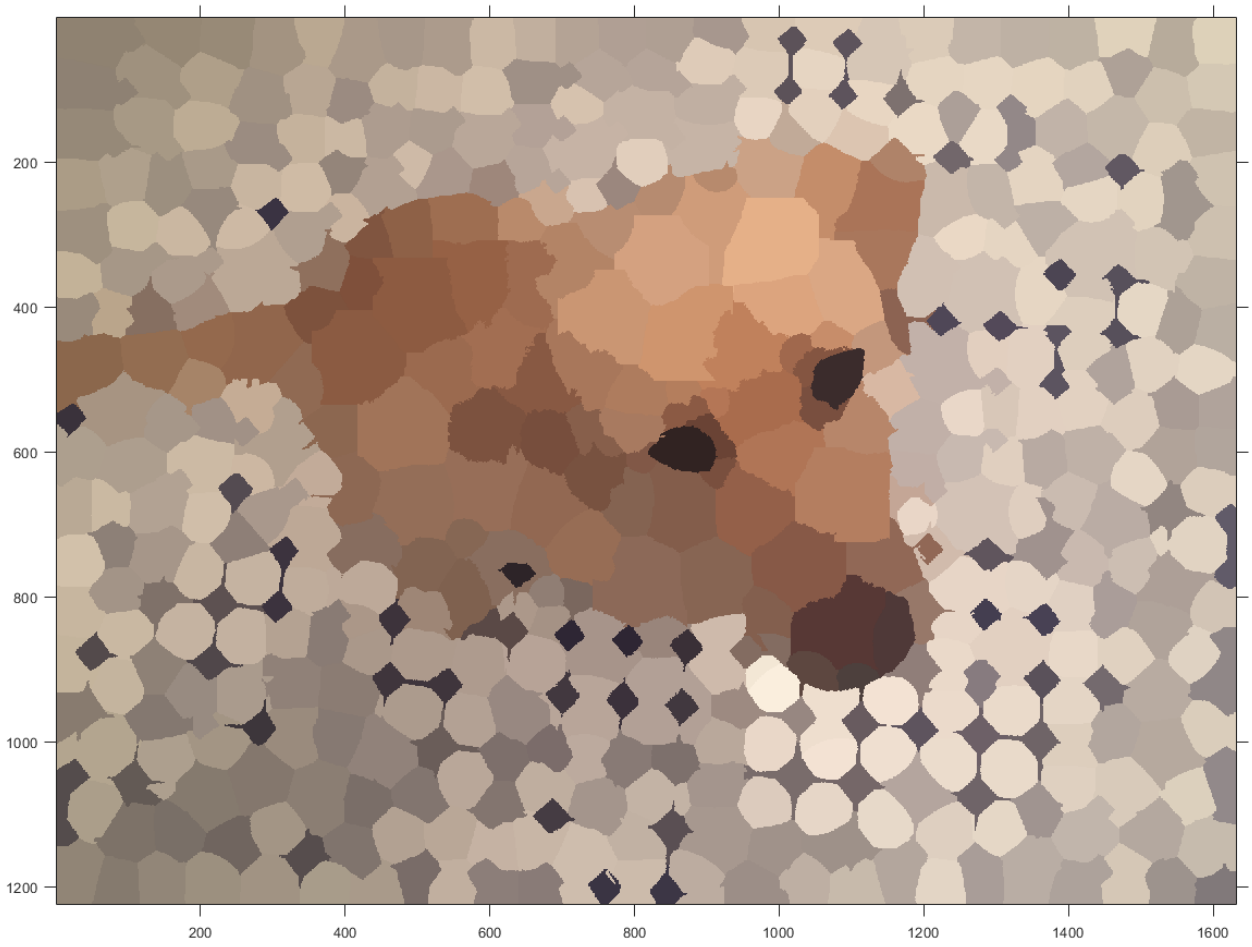
Display the superpixel boundaries overlaid on the original image.

```
figure
BW = boundarymask(L);
imshow(imoverlay(A,BW,'cyan'),'InitialMagnification',67)
```



Set the color of each pixel in the output image to the mean RGB color of the superpixel region.

```
outputImage = zeros(size(A),'like',A);
idx = label2idx(L);
numRows = size(A,1);
numCols = size(A,2);
for labelVal = 1:N
    redIdx = idx{labelVal};
    greenIdx = idx{labelVal}+numRows*numCols;
    blueIdx = idx{labelVal}+2*numRows*numCols;
    outputImage(redIdx) = mean(A(redIdx));
    outputImage(greenIdx) = mean(A(greenIdx));
    outputImage(blueIdx) = mean(A(blueIdx));
end

figure
imshow(outputImage,'InitialMagnification',67)
```



Input Arguments

`A` — Image to segment

2-D grayscale image | 2-D truecolor image

Image to segment, specified as a 2-D grayscale image or 2-D truecolor image. For `int16` data, `A` must be a grayscale image. When the parameter `isInputLab` is `true`, the input image must be data type `single` or `double`.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`N` — Desired number of superpixels

positive integer

Desired number of superpixels, specified as a positive integer.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `B = superpixels(A,100,'NumIterations', 20);`

`Compactness` — Shape of superpixels

`10` (default) | numeric scalar

Shape of superpixels, specified as a numeric scalar. The compactness parameter of the SLIC algorithm controls the shape of superpixels. A higher value makes superpixels more regularly shaped, that is, a square. A lower value makes superpixels adhere to boundaries better, making them irregularly shaped. The allowed range is `(0 Inf)`. Typical values for compactness are in the range `[1,20]`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`IsInputLab` — Input image data is in *L*a*b** color space

`false` (default) | `true`

Input image data is in the *L*a*b** color space, specified as `true` or `false`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`Method` — Algorithm used to compute superpixels

`'slic0'` (default) | `'slic'`

Algorithm used to compute superpixels, specified as one of the following values. The `superpixels` function uses two variations of the simple linear iterative clustering (SLIC) algorithm.

Value	Meaning
`'slic0'`	`superpixels` uses the SLIC0 algorithm to refine `'Compactness'` adaptively after the first iteration. This is the default.
`'slic'`	`'Compactness'` is constant during clustering.

Data Types: `char` | `string`

`NumIterations` — Number of iterations

`10` (default) | positive integer

Number of iterations used in the clustering phase of the algorithm, specified as a positive integer. For most problems, it is not necessary to adjust this parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`L` — Label matrix

array of positive integers

Label matrix, returned as an array of positive integers. The value 1 indicates the first region, 2 the second region, and so on for each superpixel region in the image.

Data Types: `double`

`NumLabels` — Number of superpixels computed
positive integer

Number of superpixels computed, returned as a positive integer.

Data Types: `double`

References

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Susstrunk, *SLIC Superpixels Compared to State-of-the-art Superpixel Methods*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 34, Issue 11, pp. 2274-2282, May 2012

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `superpixels` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.
- All character vector inputs must be compile-time constants.
- The value of `'IsInputLab'` (`true` or `false`) must be a compile-time constant.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- All character vector inputs must be compile-time constants.
- The value of `'IsInputLab'` (`true` or `false`) must be a compile-time constant.

See Also

`superpixels3` | `boundarymask` | `imoverlay` | `label2idx` | `label2rgb`

Topics

“Plot Land Classification with Color Features and Superpixels”

Introduced in R2016a

superpixels3

3-D superpixel oversegmentation of 3-D image

Syntax

```
[L,NumLabels] = superpixels3(A,N)
[L,NumLabels] = superpixels3(___,Name,Value)
```

Description

`[L,NumLabels] = superpixels3(A,N)` computes 3-D superpixels of the 3-D image `A`. `N` specifies the number of superpixels you want to create. The function returns `L`, a 3-D label matrix, and `NumLabels`, the actual number of superpixels returned.

`[L,NumLabels] = superpixels3(___,Name,Value)` computes superpixels of image `A` using name-value pairs to control aspects of the segmentation.

Examples

Compute 3-D Superpixels of Input Volumetric Intensity Image

Load 3-D MRI data, remove any singleton dimensions, and convert the data into a grayscale intensity image.

```
load mri;
D = squeeze(D);
A = ind2gray(D,map);
```

Calculate the 3-D superpixels. Form an output image where each pixel is set to the mean color of its corresponding superpixel region.

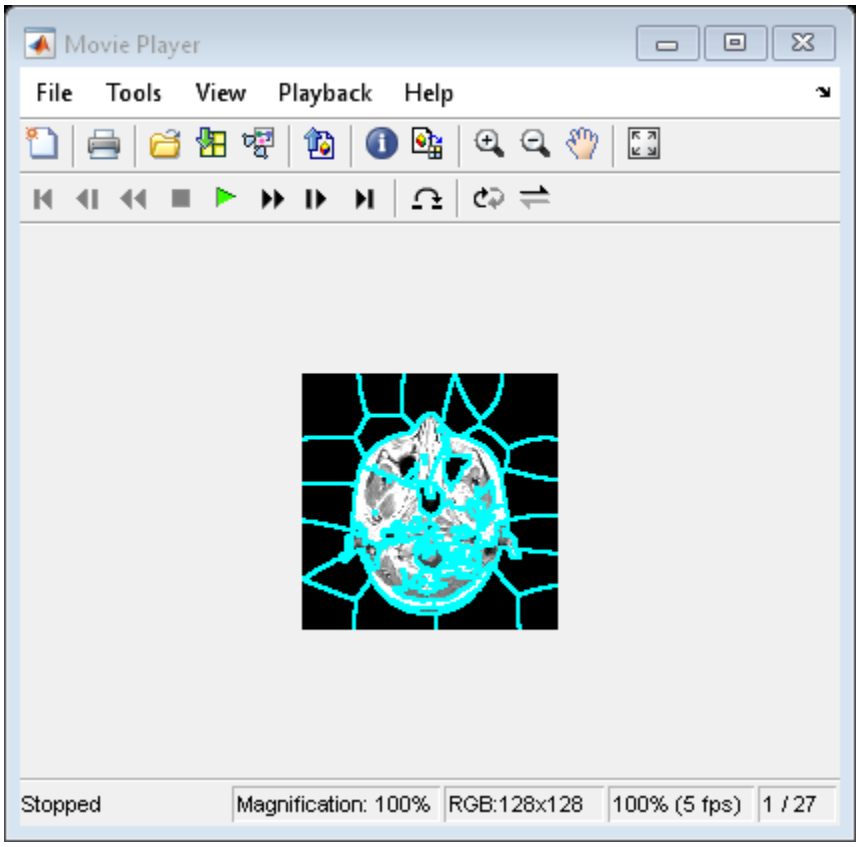
```
[L,N] = superpixels3(A,34);
```

Show all xy-planes progressively with superpixel boundaries.

```
imSize = size(A);
```

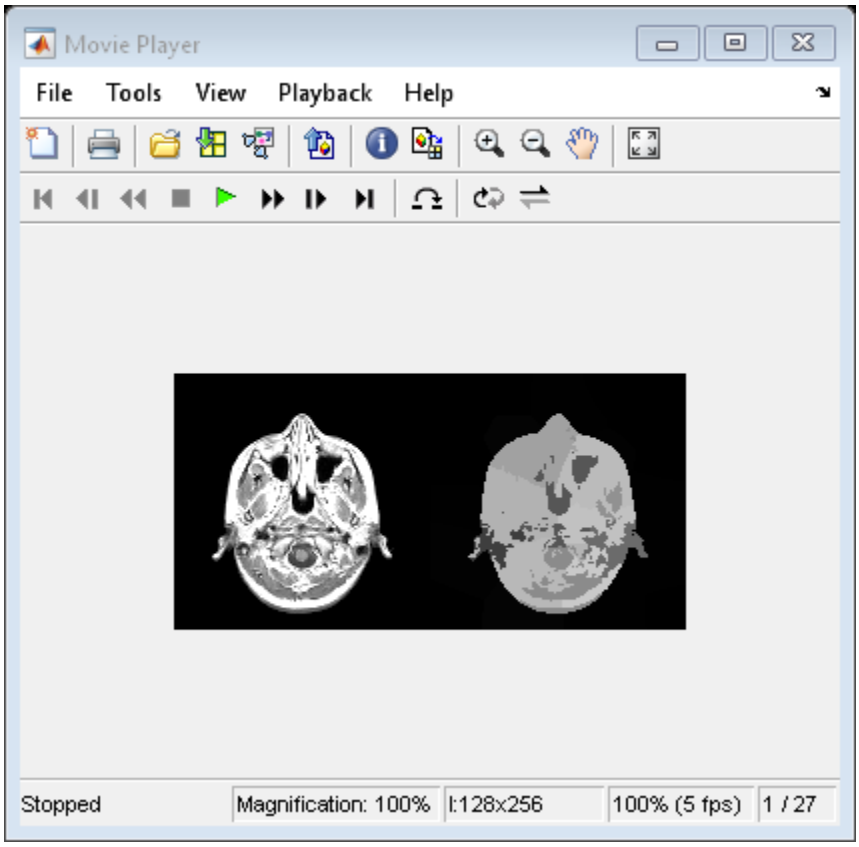
Create a stack of RGB images to display the boundaries in color.

```
imPlusBoundaries = zeros(imSize(1),imSize(2),3,imSize(3),'uint8');
for plane = 1:imSize(3)
  BW = boundarymask(L(:, :, plane));
  % Create an RGB representation of this plane with boundary shown
  % in cyan.
  imPlusBoundaries(:, :, :, plane) = imoverlay(A(:, :, plane), BW, 'cyan');
end

implay(imPlusBoundaries,5)
```



Set the color of each pixel in output image to the mean intensity of the superpixel region. Show the mean image next to the original. If you run this code, you can use `implay` to view each slice of the MRI data.

```
pixelIdxList = label2idx(L);
meanA = zeros(size(A),'like',D);
for superpixel = 1:N
     memberPixelIdx = pixelIdxList{superpixel};
     meanA(memberPixelIdx) = mean(A(memberPixelIdx));
end
implay([A meanA],5);
```



Input Arguments

`A` — Volume to segment

3-D numeric array

Volume to segment, specified as a 3-D numeric array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

`N` — Desired number of superpixels

positive integer

Desired number of superpixels, specified as a positive integer.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `B = superpixels3(A,100,'NumIterations', 20);`

`Compactness` — Shape of superpixels
`0.001` if method is `slic0` and `0.05` if method is `slic` (default) | numeric scalar

Shape of superpixels, specified as a numeric scalar. The compactness parameter of the SLIC algorithm controls the shape of the superpixels. A higher value makes the superpixels more regularly shaped, that is, a square. A lower value makes the superpixels adhere to boundaries better, making them irregularly shaped. You can specify any value in the range `[0 Inf)` but typical values are in the range `[0.01,0.1]`.

Note If you specify the `'slic0'` method, you typically do not need to adjust the `'Compactness'` parameter. With the `'slic0'` method, `superpixel3` adaptively refines the `'Compactness'` parameter automatically, thus eliminating the need to determine a good value.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Method` — Algorithm used to compute superpixels
`'slic0'` (default) | `'slic'`

Algorithm used to compute the superpixels, specified as one of the following values. For more information, see “Algorithms” on page 1-3051.

Value	Meaning
`'slic0'`	`superpixels3` uses the SLIC0 algorithm to refine `'Compactness'` adaptively after the first iteration. This is the default.
`'slic'`	`'Compactness'` is constant during clustering.

Data Types: `char` | `string`

`NumIterations` — Number of iterations
`10` (default) | positive integer

Number of iterations used in the clustering phase of the algorithm, specified as a positive integer. For most problems, it is not necessary to adjust this parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`L` — Label matrix
3-D array of positive integers

Label matrix, returned as a 3-D array of positive integers. The value `1` indicates the first region, `2` the second region, and so on for each superpixel region in the image.

Data Types: `double`

`NumLabels` — Number of superpixels computed
positive number

Number of superpixels computed, returned as a positive number.

Data Types: `double`

Algorithms

The algorithm used in `superpixels3` is a modified version of the Simple Linear Iterative Clustering (SLIC) algorithm used by `superpixels`. At a high level, it creates cluster centers and then iteratively alternates between assigning pixels to the closest cluster center and updating the locations of the cluster centers. `superpixels3` uses a distance metric to determine the closest cluster center for each pixel. This distance metric combines intensity distance and spatial distance.

The function's `Compactness` argument comes from the mathematical form of the distance metric. The compactness parameter of the algorithm is a scalar value that controls the shape of the superpixels. The distance between two pixels i and j, where m is the compactness value, is:

$$d_{intensity} = \sqrt{(l_i - l_j)^2}$$

$$d_{spatial} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

$$D = \sqrt{\left(\frac{d_{intensity}}{m}\right)^2 + \left(\frac{d_{spatial}}{S}\right)^2}$$

Compactness has the same meaning as in the 2-D `superpixels` function: It determines the relative importance of the intensity distance and the spatial distance in the overall distance metric. A lower value makes the superpixels adhere to boundaries better, making them irregularly shaped. A higher value makes the superpixels more regularly shaped. The allowable range for compactness is `(0 Inf)`, as in the 2-D function. The typical range has been found through experimentation to be `[0.01 0.1]`. The dynamic range of input images is normalized within the algorithm to be from 0 to 1. This enables a consistent meaning of compactness values across images.

See Also

`superpixels` | `boundarymask` | `imoverlay` | `label2idx` | `label2rgb`

Introduced in R2016b

tformarray

Apply spatial transformation to N-D array

Note The `tformarray` function is not recommended for 2-D and 3-D spatial transformations. Use the `imwarp` function instead. For more information, see “Compatibility Considerations”.

Syntax

```
B = tformarray(A,T,R,tdims_A,tdims_B,tsize_B,tmap_B,F)
```

Description

`B = tformarray(A,T,R,tdims_A,tdims_B,tsize_B,tmap_B,F)` applies a spatial transformation `T` to array `A` to produce array `B`.

Examples

Transform Checkerboard Image

Create a 2-by-2 square checkerboard image where each square is 20 pixels wide. Display the image.

```
I = checkerboard(20,1,1);
figure
imshow(I)
```

Transform the checkerboard with a projective transformation. First create a spatial transformation structure.

```
T = maketform('projective',[1 1; 41 1; 41 41;   1 41],...
                           [5 5; 40 5; 35 30; -10 30]);
```

Create a resampler. Use the pad method `'circular'` when creating the resampler, so that the output appears to be a perspective view of an infinite checkerboard.

```
R = makeresampler('cubic','circular');
```

Perform the transformation, specifying the transformation structure and the resampler. For this example, swap the output dimensions, and specify a 100-by-100 output image. Leave argument

`tmap_B` empty since you specify argument `tsize_B`. Leave argument `F` empty since the fill value is not needed.

```
J = tformarray(I,T,R,[1 2],[2 1],[100 100],[],[]);
figure
imshow(J)
```

Transform Checkerboard Image, with Nonuniform Mapping from Input to Output Space

Create a 2-by-2 square checkerboard image where each square is 20 pixels wide. Display the image.

```
I = checkerboard(20,1,1);
figure
imshow(I)
```

Transform the checkerboard with a projective transformation. First create a spatial transformation structure.

```
T = maketform('projective',[1 1; 41 1; 41 41;   1 41],...
                           [5 5; 40 5; 35 30; -10 30]);
```

Create a resampler. Use the pad method `'circular'` when creating the resampler, so that the output appears to be a perspective view of an infinite checkerboard.

```
R = makeresampler('cubic','circular');
```

Create arrays that specify the mapping of points from input space to output space. This example uses anisotropic sampling, where the distance between samples is larger in one direction than the other.

```
samp_x = 1:1.5:150;
samp_y = 1:100;
[x,y] = meshgrid(samp_x,samp_y);
```

```
tmap = cat(3,x,y);
size(tmap)
```

```
ans = 1×3

   100   100     2
```

Note the size of `tmap`. The output image will have dimensions 100-by-100.

Perform the transformation, specifying the transformation structure and the resampler. Specify the output map as `tmap`. Leave argument `tsize_B` empty, since you specify argument `tmap_B`. The fill value does not matter since the resampler is circular.

```
J = tformarray(I,T,R,[1 2],[1 2],[],tmap,[]);
figure
imshow(J)
```

The length of checkerboard squares is larger in the *y*-direction than in the *x*-direction, which agrees with the larger sampling distance between points in the vector `samp_x`. Compared to the result using isotopic point mapping (see example “Transform Checkerboard Image” on page 1-3052), three additional columns of the checkerboard appear at the right of the transformed image, and no new rows are added to the transformed image.

Input Arguments

A — Input image

numeric array

Input image, specified as a numeric array. `A` can be real or complex.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

T — Spatial transformation

`TFORM` spatial transformation structure

Spatial transformation, specified as a `TFORM` spatial transformation structure. You typically use the `maketform` function to create a `TFORM` structure.

`tformarray` uses `T` and the function `tforminv` to compute the corresponding location in the input transform subscript space for each location in the output transform subscript space. `tformarray`

defines the input transform space by `tdims_B` and `tsize_B` and the output transform subscript space by `tdims_A` and `size(A)`.

If `T` is empty, then `tformarray` operates as a direct resampling function. Further, if `tmap_B` is:

- Not empty, then `tformarray` applies the resampler defined in `R` to compute values at each transform space location defined in `tmap_B`
- Empty, then `tformarray` applies the resampler at each location in the output transform subscript grid

Data Types: `struct`

`R` — Resampler
structure

Resampler, specified as a structure. A resampler structure defines how to interpolate values of the input array at specified locations. `R` is created with `makeresampler`, which allows fine control over how to interpolate along each dimension. `makeresampler` also controls what input array values to use when interpolating close to the edge of the array.

Data Types: `struct`

`tdims_A` — Input transform dimensions
row vector of finite, positive integers

Input transform dimensions, specified as a row vector of finite, positive integers.

`tdims_A` and `tdims_B` indicate which dimensions of the input and output arrays are involved in the spatial transformation. Each element must be unique. The entries need not be listed in increasing order, but the order matters. The order specifies the precise correspondence between dimensions of arrays `A` and `B` and the input and output spaces of the transformation `T`.

`length(tdims_A)` must equal `T.ndims_in`, and `length(tdims_B)` must equal `T.ndims_out`.

For example, if `T` is a 2-D transformation, `tdims_A = [2 1]`, and `tdims_B = [1 2]`, then the row and column dimensions of `A` correspond to the second and first transformation input-space dimensions, respectively. The row and column dimensions of `B` correspond to the first and second output-space dimensions, respectively.

Data Types: `double`

`tdims_B` — Output transform dimensions
row vector of finite, positive integers

Output transform dimensions, specified as a row vector of finite, positive integers. For more information, see `tdims_A`.

Data Types: `double`

`tsize_B` — Size of output array in the transform dimensions
row vector of finite, positive integers

Size of the output array transform dimensions, specified as a row vector of finite, positive integers. The size of `B` along nontransform dimensions is taken directly from the size of `A` along those dimensions.

For example, if T is a 2-D transformation, `size(A) = [480 640 3 10]`, `tdims_B` is `[2 1]`, and `tsize_B` is `[300 200]`, then `size(B)` is `[200 300 3 10]`.

Data Types: `double`

`tmap_B` — Point locations in output space
finite, real-valued array | `[]`

Point locations in output space, specified as a finite real-valued array. `tmap_B` is an optional argument that provides an alternative way of specifying the correspondence between the position of elements of B and the location in output transform space. `tmap_B` can be used, for example, to compute the result of an image warp at a set of arbitrary locations in output space.

If `tmap_B` is not empty, then the size of `tmap_B` is

$$[D_1\ D_2\ D_3\ \ldots\ D_N\ L]$$

where N equals `length(tdims_B)`. `tsize_B` should be `[]`.

The value of L depends on whether T is empty. If T is:

- Not empty, then L is `T.ndims_out`, and each L-dimension point in `tmap_B` is transformed to an input-space location using T
- Empty, then L is `length(tdims_A)`, and each L-dimensional point in `tmap_B` is used directly as a location in input space.

Data Types: `double`

F — Fill values
numeric scalar | numeric array | `[]`

Fill values, specified as a numeric scalar, numeric array, or empty (`[]`). The fill values in F can be used in three situations:

- When a separable resampler is created with `makeresampler` and its `padmethod` is set to either `'fill'` or `'bound'`.
- When a custom resampler is used that supports the `'fill'` or `'bound'` pad methods (with behavior that is specific to the customization).
- When the map from the transform dimensions of B to the transform dimensions of A is deliberately undefined for some points. Such points are encoded in the input transform space by NaNs in either `tmap_B` or in the output of `tforminv`.

In the first two cases, fill values are used to compute values for output locations that map outside or near the edges of the input array. Fill values are copied into B when output locations map well outside the input array. See `makeresampler` for more information about `'fill'` and `'bound'`.

When F is:

- A scalar (including NaN), its value is replicated across all the nontransform dimensions.
- Nonscalar, its size depends on `size(A)` in the nontransform dimensions. Specifically, if K is the Jth nontransform dimension of A, then `size(F,J)` must be either `size(A,K)` or `1`. As a convenience, `tformarray` replicates F across any dimensions with unit size such that after the replication `size(F,J)` equals `size(A,K)`.

- Empty (`[]`), the `tformarray` function uses a fill value of `0`.

For example, suppose `A` represents 10 RGB images and has size 200-by-200-by-3-by-10, `T` is a 2-D transformation, and `tdims_A` and `tdims_B` are both [1 2]. In other words, `tformarray` applies the same 2-D transform to each color plane of each of the 10 RGB images. In this situation you have several options for `F`:

- `F` can be a scalar, in which case the same fill value is used for each color plane of all 10 images.
- `F` can be a 3-by-1 vector, `[R G B]'`. `tformarray` uses the RGB value as the fill value for the corresponding color planes of each of the 10 images.
- `F` can be a 1-by-10 vector. `tformarray` uses a different fill value for each of 10 images, with that fill value being used for all three color planes.
- `F` can be a 3-by-10 matrix. `tformarray` uses a different RGB fill color for each of the 10 images.

Data Types: `double`

Output Arguments

`B` — Transformed image

numeric array

Transformed image, returned as a numeric array.

Compatibility Considerations

`tformarray` is not recommended for 2-D and 3-D geometric transformations

Not recommended starting in R2018b

The `tformarray` function is intended for transformations involving higher-dimensioned arrays, mixed input/output dimensionality, or requiring greater user control or customization.

For many common tasks involving 2-D and 3-D images, the `imwarp` function is easier to use. For example, the `imwarp` function enables you to specify the type of interpolation using a name-value argument. The `imwarp` function also supports categorical images.

The `tformarray` function is not recommended for 2-D and 3-D geometric transformations. Instead, create a 2-D or 3-D geometric transformation object, then use the `imwarp` function. To perform a transformation that pads the output using the `'replicate'`, `'circular'`, or `'symmetric'` padding method, you can pad the input image using the `padarray` function before calling `imwarp`. For more information about geometric transformation objects, see “2-D and 3-D Geometric Transformation Process Overview”.

This table shows a syntax of `tformarray` with recommended replacement code.

Discouraged Usage	Recommended Replacement

This example performs a 2-D affine transformation of image `I` using cubic resampling and a fill value of `0`.
 	
`A = [0.5 0 0; 0.5 2 0; 0 0 1];`
`T = maketform("affine",A);`
`I = checkerboard(20,1,1);`
`R = makeresampler("cubic","bound");`
`J = tformarray(I,T,R,[1 2],[2 1],[40 80],[],[]);`
	Create an `affine2d` object then perform the transformation of image `I` using the `imwarp` function.
	
	`A = [0.5 0 0; 0.5 2 0; 0 0 1];`
	`T = affine2d(A);`
	`I = checkerboard(20,1,1);`
	`J = imwarp(I,T,"cubic")`

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see “Run MATLAB Functions in Thread-Based Environment”.

See Also

`findbounds` | `makeresampler` | `maketform` | `imwarp`

Topics

“N-Dimensional Spatial Transformations”

Introduced before R2006a

tformfwd

Apply forward N-D spatial transformation

Note The `tformfwd` function is not recommended for 2-D and 3-D geometric transformations. Use the `transformPointsForward` function instead. For more information, see "Compatibility Considerations".

Syntax

```
[X1,X2,...,X_ndims_out] = tformfwd(T,U1,U2,...,U_ndims_in)
X = tformfwd(T,U)
[X1,X2,...,X_ndims_out] = tformfwd(T,U)
X = tformfwd(T,U1,U2,...,U_ndims_in)
```

Description

`[X1,X2,...,X_ndims_out] = tformfwd(T,U1,U2,...,U_ndims_in)` applies the `ndims_in`-to-`ndims_out` spatial transformation defined in `T` to the coordinate arrays `U1,U2,...,U_ndims_in`. The transformation maps the point `[U1(k) U2(k) ...U_ndims_in(k)]` to the point `[X1(k) X2(k) ... X_ndims_out(k)]`.

The number of input coordinate arrays, `ndims_in`, must equal `T.ndims_in`. The number of output coordinate arrays, `ndims_out`, must equal `T.ndims_out`. The arrays `U1,U2,...,U_ndims_in` can have any dimensionality, but must be the same size. The output arrays `X1,X2,...,X_ndims_out` must be this size also.

`X = tformfwd(T,U)` applies the spatial transformation defined in `T` to coordinate array `U`.

- When `U` is a 2-D matrix with dimensions *m*-by-`ndims_in`, `X` is a 2-D matrix with dimensions *m*-by-`ndims_out`. `tformfwd` applies the `ndims_in`-to-`ndims_out` transformation to each row of `U`. `tformfwd` maps the point `U`(*k*, :) to the point `X`(*k*, :).
- When `U` is an (*N*+1)-dimensional array, `tformfwd` maps the point `U`(k_1, k_2, ... ,k_N, :) to the point `X`(k_1, k_2, ... ,k_N, :).

  `size(U,N+1)` must equal `ndims_in`. `X` is an (*N*+1)-dimensional array, with `size(X,I)` equal to `size(U,I)` for `I` = 1, ... ,*N*, and `size(X,N+1)` equal to `ndims_out`.

The syntax `X = tformfwd(U,T)` is an older form of this syntax that remains supported for backward compatibility.

`[X1,X2,...,X_ndims_out] = tformfwd(T,U)` maps one (*N*+1)-dimensional array to `ndims_out` equally sized *N*-dimensional arrays.

`X = tformfwd(T,U1,U2,...,U_ndims_in)` maps `ndims_in` *N*-dimensional arrays to one (*N*+1)-dimensional array.

Examples

Create Affine Transformation and Validate It with Forward Mapping

Create an affine transformation that maps the triangle with vertices (0,0), (6,3), (-2,5) to the triangle with vertices (-1,-1), (0,-10), (4,4).

```
u = [ 0   6  -2]';
v = [ 0   3   5]';
x = [-1   0   4]';
y = [-1 -10   4]';
tform = maketform('affine',[u v],[x y]);
```

Validate the mapping by applying `tformfwd`. The results should equal `x` and `y`.

```
[xm,ym] = tformfwd(tform,u,v)
```

```
xm = 3×1

    -1
     0
     4


ym = 3×1

    -1
   -10
     4
```

Input Arguments

T — Spatial transformation

`TFORM` structure

Spatial transformation, specified as a `TFORM` structure. Create `T` using the `maketform` function.

Data Types: `struct`

U — Input coordinate points

numeric array

Input coordinate points, specified as a numeric array. The size and dimensionality of `U` can have additional limitations depending on the syntax used.

Data Types: `double`

U1,U2,...,U_ndims_in — Input coordinate points

multiple numeric arrays

Input coordinate points, specified as multiple numeric arrays. The size and dimensionality of `U1,U2,...,U_ndims_in` can have additional limitations depending on the syntax used.

Data Types: `double`

Output Arguments

X — Coordinate array of output points

numeric array

Coordinate array of output points, returned as a numeric array. The size and dimensionality of X can have additional limitations depending on the syntax used.

X1,X2,...,X_ndims_out — Coordinates of output points

multiple numeric arrays

Coordinates of output points, returned as multiple numeric arrays. The size and dimensionality of `X1,X2,...,X_ndims_out` can have additional limitations depending on the syntax used.

Compatibility Considerations

tformfwd is not recommended for 2-D and 3-D geometric transformations

Not recommended starting in R2018b

The `tformfwd` function is not recommended for 2-D and 3-D geometric transformations. Instead, create a 2-D or 3-D geometric transformation object, then use the `transformPointsForward` function. For more information about geometric transformation objects, see "2-D and 3-D Geometric Transformation Process Overview".

This table shows a syntax of `tformfwd` with the recommended replacement code.

Discouraged Usage	Recommended Replacement
Apply a forward 2-D affine transformation defined in `TFORM` struct `T` to coordinate arrays `U` and `V`, mapping the point `[U(k) V(k)]` to the point `[X(k) Y(k)]`. `[X,Y] = tformfwd(T,U,V);`	Apply a forward 2-D affine transformation using the `affine2d` object `T` and the `transformPointsForward` function. `[X,Y] = transformPointsForward(T,U,V);`

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see "Run MATLAB Functions in Thread-Based Environment".

See Also

`maketform` | `fliptform` | `tforminv`

Topics

"N-Dimensional Spatial Transformations"

Introduced before R2006a

tforminv

Apply inverse N-D spatial transformation

Note The `tforminv` function is not recommended for 2-D and 3-D geometric transformations. Use the `transformPointsInverse` function instead. For more information, see "Compatibility Considerations".

Syntax

```
[U1,U2,...,U_ndims_in] = tforminv(T,X1,X2,...,X_ndims_out)
U = tforminv(T,X)
[U1,U2,...,U_ndims_in] = tforminv(T,X)
U = tforminv(T,X1,X2,...,X_ndims_out)
```

Description

`[U1,U2,...,U_ndims_in] = tforminv(T,X1,X2,...,X_ndims_out)` applies the `ndims_out`-to-`ndims_in` inverse transformation defined in `T` to the coordinate arrays `X1,X2,...,X_ndims_out`. The transformation maps the point `[X1(k) X2(k) ... X_ndims_out(k)]` to the point `[U1(k) U2(k) ... U_ndims_in(k)]`.

The number of input coordinate arrays, `ndims_out`, must equal `T.ndims_out`. The number of output coordinate arrays, `ndims_in`, must equal `T.ndims_in`. The arrays `X1,X2,...,X_ndims_out` can have any dimensionality, but must be the same size. The output arrays `U1,U2,...,U_ndims_in` must be this size also.

`U = tforminv(T,X)` applies the `ndims_out`-to-`ndims_in` inverse transformation defined in `T` to array `X`.

- When `X` is a 2-D matrix with dimensions *m*-by-`ndims_out` matrix, `U` is a 2-D matrix with dimensions *m*-by-`ndims_in`. `tforminv` applies the transformation to each row of `X`. `tforminv` maps the point `X`(*k*, :) to the point `U`(*k*, :).
- When `X` is an (*N*+1)-dimensional array, `tforminv` maps the point `X`(k_1, k_2, ... ,k_N, :) to the point `U`(k_1, k_2, ... ,k_N, :).

  `size(X,N+1)` must equal `ndims_out`. `U` is an (*N*+1)-dimensional array, with `size(U,I)` equal to `size(X,I)` for `I` = 1, ... ,*N*, and `size(U,N+1)` equal to `ndims_in`.

The syntax `U = tforminv(X,T)` is an older form of this syntax that remains supported for backward compatibility.

`[U1,U2,...,U_ndims_in] = tforminv(T,X)` maps one (*N*+1)-dimensional array to `ndims_in` equally sized *N*-dimensional arrays.

`U = tforminv(T,X1,X2,...,X_ndims_out)` maps `ndims_out` *N*-dimensional arrays to one (*N* +1)-dimensional array.

Examples

Create Affine Transformation and Validate It with Inverse Mapping

Create an affine transformation that maps the triangle with vertices (0,0), (6,3), (-2,5) to the triangle with vertices (-1,-1), (0,-10), (4,4).

```
u = [ 0   6  -2]';
v = [ 0   3   5]';
x = [-1   0   4]';
y = [-1 -10   4]';
tform = maketform('affine',[u v],[x y]);
```

Validate the mapping by applying `tforminv`. The results should equal `u` and `v`.

```
[um, vm] = tforminv(tform, x, y)
```

```
um = 3×1

          0
    6.0000
   -2.0000


vm = 3×1

          0
    3.0000
    5.0000
```

Input Arguments

T — Spatial transformation

`TFORM` spatial transformation structure

Spatial transformation, specified as a `TFORM` spatial transformation structure. Create `T` using the `maketform` function.

Data Types: `struct`

X — Input coordinate points

numeric array

Input coordinate points, specified as a numeric array. The size and dimensionality of `X` can have additional limitations depending on the syntax used.

Data Types: `double`

X1,X2,...,X_ndims_out — Input coordinate points

multiple numeric arrays

Input coordinate points, specified as multiple numeric arrays. The size and dimensionality of `X1,X2,...,X_ndims_out` can have additional limitations depending on the syntax used.

Data Types: `double`

Output Arguments

`U` — Coordinate array of output points

numeric array

Coordinate array of output points, returned as a numeric array. The size and dimensionality of `U` can have additional limitations depending on the syntax used.

`U1,U2,...,U_ndims_in` — Coordinates of output points

multiple numeric arrays

Coordinates of output points, returned as multiple arrays. The size and dimensionality of `U1,U2,...,U_ndims_in` can have additional limitations depending on the syntax used.

Compatibility Considerations

`tforminv` is not recommended for 2-D and 3-D geometric transformations

Not recommended starting in R2018b

The `tforminv` function is not recommended for 2-D and 3-D geometric transformations. Instead, create a 2-D or 3-D geometric transformation object, then use the `transformPointsInverse` function. For more information about geometric transformation objects, see "2-D and 3-D Geometric Transformation Process Overview".

This table shows a syntax of `tforminv` with the recommended replacement code.

Discouraged Usage	Recommended Replacement
Apply an inverse 2-D affine transformation defined in `TFORM` struct `T` to coordinate arrays `X` and `Y`, mapping the point `[X(k) Y(k)]` to the point `[U(k) V(k)]`. `[U,V] = tforminv(T,X,Y);`	Apply an inverse 2-D affine transformation using the `affine2d` object `T` and the `transformPointsInverse` function. `[U,V] = transformPointsInverse(T,X,Y);`

Extended Capabilities

Thread-Based Environment

Run code in the background using MATLAB® `backgroundPool` or accelerate code with Parallel Computing Toolbox™ `ThreadPool`.

This function fully supports thread-based environments. For more information, see "Run MATLAB Functions in Thread-Based Environment".

See Also

`maketform` | `fliptform` | `tformfwd`

Topics

"N-Dimensional Spatial Transformations"

Introduced before R2006a

tiffreadVolume

Read volume from TIFF file

Syntax

```
V = tiffreadVolume(filename)
V = tiffreadVolume(filename,'PixelRegion',{rows,columns,slices})
```

Description

`V = tiffreadVolume(filename)` loads all of the volumetric data in the TIFF file named `filename` into `V`. All of the spatial dimensions in `V` are first, and color (if present) is in the final dimension.

`V = tiffreadVolume(filename,'PixelRegion',{rows,columns,slices})` reads a subset of the volume `V`. `{rows,columns,slices}` is a cell array that specifies the subsampling along each dimension.

Examples

Read Volume from TIFF File

This example shows how to read volumetric data stored in a TIFF file.

Read Entire Volume from File

Read a volume from a TIFF file into the workspace. In this example, the volume is a stack of 27 MRI images. Each image is 128-by-128 pixels in size.

```
V1 = tiffreadVolume('mri.tif');
whos V1
```

```
  Name        Size                Bytes  Class    Attributes

  V1        128x128x27           442368  uint8
```

Read Subsection of Volume from File

Read a subsection of a volume from a TIFF file into the workspace. The example uses the `'PixelRegion'` parameter to specify which part of the volume to read. You specify the subsection in a cell array of the form: `{rows, columns, slices}`. The example specifies to start reading at the first pixel and reads every other pixel in the row and column dimensions. The example reads slices 10 through 15.

```
V2 = tiffreadVolume('mri.tif',...
    'PixelRegion', {[1 2 inf], [1 2 inf], [10 15]});
whos V2
```

```
  Name       Size              Bytes  Class    Attributes

  V2        64x64x6            24576  uint8
```

Input Arguments

`filename` — Name of TIFF File

string

Name of TIFF file, specified as a string.

Example: `'mri.tif'`

Data Types: `char` | `string`

`{rows,columns,slices}` — Subsampling instructions

cell array

Subsampling instructions, specified as a cell array containing three elements: `{row,column,slice}`. Specifying `slice` is optional. If you do not specify it, `tiffreadVolume` reads all the slices in the volume.

Each of the elements in the cell array is a numeric vector of the form `[start stop]` or `[start stride stop]`. `start` specifies where to start reading on a particular dimension. `stop` specifies where to stop reading on a particular dimension. To read to the end of the dimension, specify the value `inf` for `stop`. The `start` and `stop` values are inclusive. `stride` specifies whether to read every pixel along a particular dimension or subsample the dimension by skipping over pixels.

For example, to start reading at the first pixel, read every other pixel, and continue reading until the end of the dimension, specify `[1 2 inf]`.

Data Types: `cell` | `double` | `single`

Output Arguments

`V` — Volume

numeric array

Volume, returned as a numeric array.

Tips

This function supports the following kinds of TIFF volumes:

- Volumetric data stored in the file as individual Image File Directories (IFDs) of the same size and kind.
- Volumetric data stored in the file as one image using the TIFF `ImageDepth` tag .
- Volumetric data stored as large, non-BigTIFF volumes, greater than 4GB, created by ImageJ.

See Also

`dicomread` | `dicomreadVolume` | `imread` | `nitfread`

Introduced in R2020b

tonemap

Render high dynamic range image for viewing

Syntax

```
RGB = tonemap(HDR)
RGB = tonemap(HDR,Name,Value)
```

Description

`RGB = tonemap(HDR)` converts the high dynamic range image `HDR` to a lower dynamic range image, `RGB`, suitable for display, using a process called tone mapping. Tone mapping is a technique used to approximate the appearance of high dynamic range images on a display with a more limited dynamic range.

`RGB = tonemap(HDR,Name,Value)` uses name-value pairs to control various aspects of the tone mapping.

Examples

Display High Dynamic Range Image

This example shows how to display a high dynamic range (HDR) image. To view an HDR image, you must first convert the data to a dynamic range that can be displayed correctly on a computer.

Read a high dynamic range (HDR) image, using `hdrread`. If you try to display the HDR image, notice that it does not display correctly.

```
hdr_image = hdrread('office.hdr');
imshow(hdr_image)
```

Convert the HDR image to a dynamic range that can be viewed on a computer, using the `tonemap` function. This function converts the HDR image into an RGB image of class `uint8` .

```
rgb = tonemap(hdr_image);
whos
```

```
  Name             Size                  Bytes  Class     Attributes

  hdr_image      665x1000x3            7980000  single
  rgb            665x1000x3            1995000  uint8
```

Display the RGB image.

```
imshow(rgb)
```

Input Arguments

`HDR` — High dynamic range image

m-by-*n*-by-3 array

High dynamic range image, specified as an *m*-by-*n*-by-3 array.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `RGB = tonemap(HDR,'AdjustLightness',[0.05 0.95]);`

`AdjustLightness` — Overall lightness of the rendered image

2-element vector

Overall lightness of the rendered image, specified as a two-element vector. The vector takes the form `[low high]`, where `low` and `high` are luminance values of the low dynamic range image, in the range (0, 1]. These values are passed to `imadjust`.

Data Types: `double`

`AdjustSaturation` — Saturation of colors in the rendered image
1 (default) | positive scalar

Saturation of colors in the rendered image, specified as a positive scalar. When the value is greater than 1, the colors are more saturated. When the value is in the range (0, 1], colors are less saturated.

Data Types: `double`

`NumberOfTiles` — Number of tiles used during adaptive histogram equalization
`[4 4]` (default) | 2-element vector of positive integers

Number of tiles used during the adaptive histogram equalization part of the tone mapping operation, specified as a 2-element vector of positive integers. The vector takes the form `[rows cols]`, where `rows` and `cols` specify the number of rows and columns of tiles. Both `rows` and `cols` must be at least 2. The total number of image tiles is equal to `rows*cols`. A larger number of tiles results in an image with greater local contrast.

Data Types: `double`

Output Arguments

`RGB` — Low dynamic range image
m-by-*n*-by-3 array

Low dynamic range image, specified as an *m*-by-*n*-by-3 array.

Data Types: `uint8`

See Also
`adapthisteq` | `hdrread` | `stretchlim` | `tonemapfarbman` | `makehdr`

Introduced in R2007b

tonemapfarbman

Convert HDR image to LDR using edge-preserving multiscale decompositions

Syntax

```
LDR = tonemapfarbman(HDR)
LDR = tonemapfarbman(HDR,Name,Value)
```

Description

`LDR = tonemapfarbman(HDR)` converts the high dynamic range (HDR) image to a low dynamic range (LDR) image, suitable for display, using a process called edge-preserving decompositions for multiscale tone and detail manipulation.

`LDR = tonemapfarbman(HDR,Name,Value)` uses one or more name-value pairs to control various aspects of the tone mapping.

Examples

Compress Dynamic Range of HDR Image Using Edge-Preserving Multiscale Decompositions

Load a high dynamic range (HDR) image into the workspace.

```
HDR = hdrread('office.hdr');
```

Convert the HDR image to a low dynamic range (LDR) image using the basic tone mapping function `tonemap`. Display the result. The LDR image has an acceptable dynamic range but colors are muted.

```
LDR = tonemap(HDR);
imshow(LDR)
```

Repeat the conversion using the `tonemapfarbman` function with default argument values. Display the result. Colors appear more saturated than in the LDR image created using the `tonemap` function. However, the image is bright and has poor contrast, such as in the shadow of the tree. The brightness and poor contrast indicate that the default value of `'Exposure'` is too large.

```
RGB = tonemapfarbman(HDR);
imshow(RGB)
```

Repeat the conversion using the `tonemapfarbman` function with a lower value of `'Exposure'` to darken the image. Display the result. The image contrast is improved. The image also shows a decrease in the clipping of pixel values in bright regions, such as the sky, road, and monitor.

```
RGB2 = tonemapfarbman(HDR,'Exposure',1.5);
imshow(RGB2)
```

Input Arguments

`HDR` — High dynamic range image

m-by-*n* matrix | *m*-by-*n*-by-3 array

High dynamic range image, specified as an *m*-by-*n* matrix or an *m*-by-*n*-by-3 array.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `LDR = tonemapFarbman(HDR,'Saturation',2.1);`

`RangeCompression` — Range compression

`0.3` (default) | number in the range [0, 1]

Range compression, specified as the comma-separated pair consisting of `'RangeCompression'` and a number in the range [0, 1]. A value of `1` represents maximum compression and a value of `0` represents minimum compression.

`Saturation` — Saturation
`1.6` (default) | nonnegative number

Saturation, specified as the comma-separated pair consisting of `'Saturation'` and a nonnegative number. The recommended range for `'Saturation'` is [0, 5]. As the saturation value increases, colors become more rich and intense. As the saturation value decreases, colors fade away to gray. The `'Saturation'` argument does not affect grayscale `HDR` images.

`Exposure` — Exposure
3 (default) | positive number

Exposure, specified as the comma-separated pair consisting of `'Exposure'` and a positive number. The recommended range for `'Exposure'` is (0, 5]. As this value decreases, the exposure length decreases, so the image darkens. As this value increases, the exposure length increases, so the image brightens.

`NumberOfScales` — Number of scales
3 (default) | positive integer

Number of scales, specified as the comma-separated pair consisting of `'NumberOfScales'` and a positive integer. The recommended range for `'NumberOfScales'` is [1, 5]. The default number of scales is `length(Weights)` when you specify `'Weights'`. Otherwise, the default number of scales is 3.

`Weights` — Weights of detail layers
`[1.5 1.5 1.5]` (default) | *n*-element vector of positive numbers

Weights of detail layers, specified as the comma-separated pair consisting of `'Weights'` and an *n*-element vector of positive numbers, where *n* is the number of scales specified by `'NumberOfScales'`. The recommended range of each element in `Weights` is (0, 3]. The default value of `'Weights'` is an *n*-element numeric vector with all elements set to `1.5`. For `Weights` <1, the amount of detail in the output image decreases and `Weights` >1, the amount of detail in the output image increases.

Output Arguments

`LDR` — Low dynamic range image
numeric array

Low dynamic range image, specified as a numeric array of the same size as `HDR`.

Data Types: `uint8`

Tips

- This function uses an anisotropic diffusion filter, `imdiffusefilt`, for the approximation of the weighted least squares filter, as proposed by Farbman et al. [1]

References

[1] Farbman, Z., R. Fattal, D. Lischinski, and R. Szeliski. "Edge-Preserving Decompositions for Multi-Scale Tone and Detail Manipulation." *ACM Transactions on Graphics*. Vol. 27, Number 3, August 2008, pp. 1–10.

See Also

`tonemap` | `hdrread` | `makehdr` | `localtonemap` | `locallapfilt` | `imdiffusefilt`

Introduced in R2018b

transformPointsForward

Apply forward geometric transformation

Syntax

```
[x,y] = transformPointsForward(tform,u,v)
[x,y,z] = transformPointsForward(tform,u,v,w)
X = transformPointsForward(tform,U)
```

Description

`[x,y] = transformPointsForward(tform,u,v)` applies the forward transformation of 2-D geometric transformation `tform` to the points specified by coordinates `u` and `v`.

`[x,y,z] = transformPointsForward(tform,u,v,w)` applies the forward transformation of 3-D geometric transformation `tform` to the points specified by coordinates `u`, `v`, and `w`.

`X = transformPointsForward(tform,U)` applies the forward transformation of `tform` to the input coordinate matrix `U` and returns the coordinate matrix `X`. `transformPointsForward` maps the *k*th point `U`(*k*,`:`) to the point `X`(*k*,`:`).

Examples

Apply Forward Transformation of 2-D Geometric Transformation

Create an `affine2d` object that defines the transformation.

```
theta = 10;

tform = affine2d([cosd(theta) -sind(theta) 0; sind(theta) cosd(theta) 0; 0 0 1])
```

```
tform =

  affine2d with properties:

                T: [3x3 double]
    Dimensionality: 2
```

Apply forward geometric transformation to an input (`u`,`v`) point.

```
[X,Y] = transformPointsForward(tform,5,10)
```

```
X =

    6.6605


Y =
```

```
    8.9798
```

Transform Coordinate Arrays Using Custom 2-D Transformation

Specify the x- and y-coordinates vectors of five points to transform.

```
x = [10 11 15 2 2];
y = [15 32 34 7 10];
```

Define the inverse and forward mapping functions. Both functions accept and return points in packed (x,y) format.

```
inversefn = @(c) [c(:,1).^2,sqrt(c(:,2))];
forwardfn = @(c) [sqrt(c(:,1)),c(:,2).^2];
```

Create a 2-D geometric transform object, `tform`, that stores the inverse mapping function and the optional forward mapping function.

```
tform = geometricTransform2d(inversefn,forwardfn)
```

```
tform =
  geometricTransform2d with properties:

        InverseFcn: @(c)[c(:,1).^2,sqrt(c(:,2))]
        ForwardFcn: @(c)[sqrt(c(:,1)),c(:,2).^2]
    Dimensionality: 2
```

Apply the inverse geometric transform to the input points.

```
[u,v] = transformPointsInverse(tform,x,y)
```

```
u = 1×5

   100   121   225     4     4


v = 1×5

    3.8730    5.6569    5.8310    2.6458    3.1623
```

Apply the forward geometric transform to the transformed points `u` and `v`.

```
[x,y] = transformPointsForward(tform,u,v)
```

```
x = 1×5

    10    11    15     2     2


y = 1×5

   15.0000   32.0000   34.0000    7.0000   10.0000
```

Apply Forward Transformation of 3-D Geometric Transformation

Create an `affine3d` object that defines the transformation.

```
tform = affine3d([3 1 2 0;4 5 8 0;6 2 1 0;0 0 0 1])
```

```
tform =

  affine3d with properties:

                T: [4×4 double]
    Dimensionality: 3
```

Apply forward transformation of 3-D geometric transformation to an input (u,v,w) point.

```
[X,Y,Z] = transformPointsForward(tform,2,3,5)
```

```
X =

    48


Y =

    27


Z =

    33
```

Transform Coordinate Arrays Using Custom 3-D Transformation

Specify the *x*-, *y*- and the *z*-coordinate vectors of five points to transform.

```
x = [3 5 7 9 11];
y = [2 4 6 8 10];
z = [5 9 13 17 21];
```

Define the inverse and forward mapping functions that accept and return points in packed (*x*,*y*,*z*) format.

```
inverseFcn = @(c)[c(:,1).^2,c(:,2).^2,c(:,3).^2];
forwardFcn = @(c)[sqrt(c(:,1)),sqrt(c(:,2)),sqrt(c(:,3))];
```

Create a 3-D geometric transformation object, `tform`, that stores these inverse and forward mapping functions.

```
tform = geometricTransform3d(inverseFcn,forwardFcn)
```

```
tform =
  geometricTransform3d with properties:

        InverseFcn: @(c)[c(:,1).^2,c(:,2).^2,c(:,3).^2]
        ForwardFcn: @(c)[sqrt(c(:,1)),sqrt(c(:,2)),sqrt(c(:,3))]
```

```
    Dimensionality: 3
```

Apply the inverse transformation of this 3-D geometric transformation to the input points.

```
[u,v,w] = transformPointsInverse(tform,x,y,z)
```

```
u = 1×5

     9    25    49    81   121

v = 1×5

     4    16    36    64   100

w = 1×5

    25    81   169   289   441
```

Apply the forward geometric transform to the transformed points `u`, `v`, and `w`.

```
[x,y,z] = transformPointsForward(tform,u,v,w)
```

```
x = 1×5

     3     5     7     9    11

y = 1×5

     2     4     6     8    10

z = 1×5

     5     9    13    17    21
```

Input Arguments

`tform` — Geometric transformation

geometric transformation object

Geometric transformation, specified as a geometric transformation object.

For 2-D geometric transformations, `tform` can be a `rigid2d`, `affine2d`, `projective2d`, or `geometricTransform2d` geometric transformation object.

For 3-D geometric transformations, `tform` can be an `affine3d`, `rigid3d`, or `geometricTransform3d` geometric transformation object.

`u` — *x*-coordinates of points to be transformed

m-by-*n* or *m*-by-*n*-by-*p* numeric array

x-coordinates of points to be transformed, specified as an *m*-by-*n* or *m*-by-*n*-by-*p* numeric array. The number of dimensions of `u` matches the dimensionality of `tform`.

Data Types: `single` | `double`

`v` — *y*-coordinates of points to be transformed
m-by-*n* or *m*-by-*n*-by-*p* numeric array

y-coordinates of points to be transformed, specified as an *m*-by-*n* or *m*-by-*n*-by-*p* numeric array. The size of `v` must match the size of `u`.

Data Types: `single` | `double`

`w` — *z*-coordinates of points to be transformed
m-by-*n*-by-*p* numeric array

z-coordinates of points to be transformed, specified as an *m*-by-*n*-by-*p* numeric array. `w` is used only when `tform` is a 3-D geometric transformation. The size of `w` must match the size of `u`.

Data Types: `single` | `double`

`U` — Coordinates of points to be transformed
l-by-*2* or *l*-by-*3* numeric array

Coordinates of points to be transformed, specified as an *l*-by-*2* or *l*-by-*3* numeric array. The number of columns of `U` matches the dimensionality of `tform`.

The first column lists the *x*-coordinate of each point to transform, and the second column lists the *y*-coordinate. If `tform` represents a 3-D geometric transformation, `U` has size *l*-by-*3* and the third column lists the *z*-coordinate of the points to transform.

Data Types: `single` | `double`

Output Arguments

`x` — *x*-coordinates of points after transformation
m-by-*n* or *m*-by-*n*-by-*p* numeric array

x-coordinates of points after transformation, returned as an *m*-by-*n* or *m*-by-*n*-by-*p* numeric array. The number of dimensions of `x` matches the dimensionality of `tform`.

Data Types: `single` | `double`

`y` — *y*-coordinates of points after transformation
m-by-*n* or *m*-by-*n*-by-*p* numeric array

y-coordinates of points after transformation, returned as an *m*-by-*n* or *m*-by-*n*-by-*p* numeric array. The size of `y` matches the size of `x`.

Data Types: `single` | `double`

`z` — *z*-coordinates of points after transformation
m-by-*n*-by-*p* numeric array

z-coordinates of points after transformation, returned as an *m*-by-*n*-by-*p* numeric array. The size of `z` matches the size of `x`.

Data Types: `single` | `double`

X — Coordinates of points after transformation

numeric array

Coordinates of points after transformation, returned as a numeric array. The size of `X` matches the size of `U`.

The first column lists the *x*-coordinate of each point after transformation, and the second column lists the *y*-coordinate. If `tform` represents a 3-D geometric transformation, the third column lists the *z*-coordinate of the points after transformation.

Data Types: `single` | `double`

See Also

`transformPointsInverse` | `imwarp`

Introduced in R2013a

transformPointsInverse

Package:

Apply inverse geometric transformation

Syntax

```
[u,v] = transformPointsInverse(tform,x,y)
[u,v,w] = transformPointsInverse(tform,x,y,z)
U = transformPointsInverse(tform,X)
```

Description

`[u,v] = transformPointsInverse(tform,x,y)` applies the inverse transformation of 2-D geometric transformation `tform` to the points specified by coordinates `x` and `y`.

`[u,v,w] = transformPointsInverse(tform,x,y,z)` applies the inverse transformation of 3-D geometric transformation `tform` to the points specified by coordinates `x`, `y`, and `z`.

`U = transformPointsInverse(tform,X)` applies the inverse transformation of `tform` to the input coordinate matrix `X` and returns the coordinate matrix `U`. `transformPointsInverse` maps the *k*th point `X`(*k*,:) to the point `U`(*k*,:).

Examples

Apply Inverse Transformation of 2-D Geometric Transformation

Create an `affine2d` object that defines the transformation.

```
theta = 10;

tform = affine2d([cosd(theta) -sind(theta) 0; sind(theta) cosd(theta) 0; 0 0 1])
```

```
tform =

  affine2d with properties:

                T: [3x3 double]
    Dimensionality: 2
```

Apply forward transformation of 2-D geometric transformation to an input point.

```
[X,Y] = transformPointsForward(tform,5,10)
```

```
X =

    6.6605


Y =
```

```
    8.9798
```

Apply inverse transformation of 2-D geometric transformation to output point from the previous step to recover the original coordinates.

```
[U,V] = transformPointsInverse(tform,X,Y)
```

```
U =

    5.0000


V =

    10
```

Transform Packed Coordinates Using Custom 2-D Transformation

Specify the packed (x,y) coordinates of five input points. The packed coordinates are stored in a 5-by-2 matrix, where the *x*-coordinate of each point is in the first column, and the *y*-coordinate of each point is in the second column.

```
XY = [10 15;11 32;15 34;2 7;2 10];
```

Define the inverse mapping function. The function accepts and returns points in packed (x,y) format.

```
inversefn = @(c) [c(:,1)+c(:,2),c(:,1)-c(:,2)]
```

```
inversefn = function_handle with value:
    @(c)[c(:,1)+c(:,2),c(:,1)-c(:,2)]
```

Create a 2-D geometric transform object, `tform`, that stores the inverse mapping function.

```
tform = geometricTransform2d(inversefn)
```

```
tform =
  geometricTransform2d with properties:

        InverseFcn: @(c)[c(:,1)+c(:,2),c(:,1)-c(:,2)]
        ForwardFcn: []
    Dimensionality: 2
```

Apply the inverse geometric transform to the input points.

```
UV = transformPointsInverse(tform,XY)
```

```
UV = 5×2

    25    -5
    43   -21
    49   -19
     9    -5
    12    -8
```

Apply Inverse Transformation of 3-D Geometric Transformation

Create an `affine3d` object that defines the transformation.

```
tform = affine3d([3 1 2 0;4 5 8 0;6 2 1 0;0 0 0 1])
```

```
tform =

  affine3d with properties:

                T: [4×4 double]
    Dimensionality: 3
```

Apply forward transformation of 3-D geometric transformation to an input point.

```
[X,Y,Z] = transformPointsForward(tform,2,3,5)
```

```
X =

    48


Y =

    27


Z =

    33
```

Apply inverse transformation of 3-D geometric transformation to output point from the previous step to recover the original coordinates.

```
[U,V,W] = transformPointsInverse(tform,X,Y,Z)
```

```
U =

    2.0000


V =

     3


W =

    5.0000
```

Transform Packed Coordinates Using Custom 3-D Transformation

Specify the packed (*x*,*y*,*z*) coordinates of five input points. The packed coordinates are stored as a 5-by-3 matrix, where the first, second, and third columns contain the *x*-, *y*-, and *z*-coordinates,respectively.

```
XYZ = [5 25 20;10 5 25;15 10 5;20 15 10;25 20 15];
```

Define an inverse mapping function that accepts and returns points in packed (*x*,*y*,*z*) format.

```
inverseFcn = @(c) [c(:,1)+c(:,2),c(:,1)-c(:,2),c(:,3).^2];
```

Create a 3-D geometric transformation object, `tform`, that stores this inverse mapping function.

```
tform = geometricTransform3d(inverseFcn)
```

```
tform =
  geometricTransform3d with properties:

        InverseFcn: @(c)[c(:,1)+c(:,2),c(:,1)-c(:,2),c(:,3).^2]
        ForwardFcn: []
    Dimensionality: 3
```

Apply the inverse transformation of this 3-D geometric transformation to the input points.

```
UVW = transformPointsInverse(tform,XYZ)
```

```
UVW = 5×3

    30   -20   400
    15     5   625
    25     5    25
    35     5   100
    45     5   225
```

Input Arguments

`tform` — Geometric transformation

geometric transformation object

Geometric transformation, specified as a geometric transformation object.

For 2-D geometric transformations, `tform` can be a `rigid2d`, `affine2d`, `projective2d`, `geometricTransform2d`, `LocalWeightedMeanTransformation2D`, `PiecewiseLinearTransformation2D`, or `PolynomialTransformation2D` geometric transformation object.

For 3-D geometric transformations, `tform` can be an `affine3d`, `rigid3d`, or `geometricTransform3d` geometric transformation object.

x — *x*-coordinates of points to be transformed

m-by-*n* or *m*-by-*n*-by-*p* numeric array

x-coordinates of points to be transformed, specified as an *m*-by-*n* or *m*-by-*n*-by-*p* numeric array. The number of dimensions of `x` matches the dimensionality of `tform`.

Data Types: `single` | `double`

`y` — *y*-coordinates of points to be transformed
m-by-*n* or *m*-by-*n*-by-*p* numeric array

y-coordinates of points to be transformed, specified as an *m*-by-*n* or *m*-by-*n*-by-*p* numeric array. The size of `y` must match the size of `x`.

Data Types: `single` | `double`

`z` — *z*-coordinates of points to be transformed
m-by-*n*-by-*p* numeric array

z-coordinates of points to be transformed, specified as an *m*-by-*n*-by-*p* numeric array. `z` is used only when `tform` is a 3-D geometric transformation. The size of `z` must match the size of `x`.

Data Types: `single` | `double`

`X` — Coordinates of points to be transformed
l-by-*2* or *l*-by-*3* numeric array

Coordinates of points to be transformed, specified as an *l*-by-*2* or *l*-by-*3* numeric array. The number of columns of `X` matches the dimensionality of `tform`.

The first column lists the *x*-coordinate of each point to transform, and the second column lists the *y*-coordinate. If `tform` represents a 3-D geometric transformation, `X` has size *l*-by-*3* and the third column lists the *z*-coordinate of the points to transform.

Data Types: `single` | `double`

Output Arguments

`u` — *x*-coordinates of points after transformation
m-by-*n* or *m*-by-*n*-by-*p* numeric array

x-coordinates of points after transformation, returned as an *m*-by-*n* or *m*-by-*n*-by-*p* numeric array. The number of dimensions of `u` matches the dimensionality of `tform`.

Data Types: `single` | `double`

`v` — *y*-coordinates of points after transformation
m-by-*n* or *m*-by-*n*-by-*p* numeric array

y-coordinates of points after transformation, returned as an *m*-by-*n* or *m*-by-*n*-by-*p* numeric array. The size of `v` matches the size of `u`.

Data Types: `single` | `double`

`w` — *z*-coordinates of points after transformation
m-by-*n*-by-*p* numeric array

z-coordinates of points after transformation, returned as an *m*-by-*n*-by-*p* numeric array. The size of `w` matches the size of `u`.

Data Types: `single` | `double`

`U` — Coordinates of points after transformation

numeric array

Coordinates of points after transformation, returned as a numeric array. The size of `U` matches the size of `X`.

The first column lists the *x*-coordinate of each point after transformation, and the second column lists the *y*-coordinate. If `tform` represents a 3-D geometric transformation, the third column lists the *z*-coordinate of the points after transformation.

Data Types: `single` | `double`

See Also

`transformPointsForward` | `imwarp`

Introduced in R2013a

translate

Translate structuring element

Syntax

```
SE2 = translate(SE,v)
```

Description

`SE2 = translate(SE,v)` translates the structuring element `SE` in N-D space. `v` is an N-element vector containing the offsets of the desired translation in each dimension.

Examples

Translate Structuring Element

Read an image into the workspace.

```
I = imread('cameraman.tif');
```

Create a structuring element and translate it down and to the right by 25 pixels.

```
se = translate(strel(1), [25 25]);
```

Dilate the image using the translated structuring element.

```
J = imdilate(I,se);
```

Display the original image and the translated image.

```
figure
imshow(I), title('Original')
```


```
figure
imshow(J), title('Translated');
```

Translate Offset Structuring Element

Create an offset structuring element.

```
SE = offsetstrel('ball', 5, 6.5)
```

```
SE =
offsetstrel is a ball shaped offset structuring element with properties:

            Offset: [11x11 double]
    Dimensionality: 2
```

```
SE.Offset
```

```
ans = 11×11

      -Inf      -Inf         0    0.8123    1.6246    2.4369    1.6246    0.8123         0
      -Inf    0.8123    1.6246    2.4369    3.2492    3.2492    3.2492    2.4369    1.6246
         0    1.6246    2.4369    3.2492    4.0615    4.0615    4.0615    3.2492    2.4369
    0.8123    2.4369    3.2492    4.0615    4.8738    4.8738    4.8738    4.0615    3.2492
    1.6246    3.2492    4.0615    4.8738    5.6861    5.6861    5.6861    4.8738    4.0615
    2.4369    3.2492    4.0615    4.8738    5.6861    6.4984    5.6861    4.8738    4.0615
    1.6246    3.2492    4.0615    4.8738    5.6861    5.6861    5.6861    4.8738    4.0615
    0.8123    2.4369    3.2492    4.0615    4.8738    4.8738    4.8738    4.0615    3.2492
         0    1.6246    2.4369    3.2492    4.0615    4.0615    4.0615    3.2492    2.4369
      -Inf    0.8123    1.6246    2.4369    3.2492    3.2492    3.2492    2.4369    1.6246
      ⋮
```

Translate the structuring element.

```
V = [2 2];
SE2 = translate(SE,V)
```

```
SE2 =
offsetstrel is a ball shaped offset structuring element with properties:

            Offset: [15x15 double]
    Dimensionality: 2
```

```
SE2.Offset
```

```
ans = 15×15

      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf
      -Inf      -Inf      -Inf      -Inf      -Inf      -Inf         0    0.8123    1.6246
      -Inf      -Inf      -Inf      -Inf      -Inf    0.8123    1.6246    2.4369    3.2492
      -Inf      -Inf      -Inf      -Inf         0    1.6246    2.4369    3.2492    4.0615
      -Inf      -Inf      -Inf      -Inf    0.8123    2.4369    3.2492    4.0615    4.8738
      -Inf      -Inf      -Inf      -Inf    1.6246    3.2492    4.0615    4.8738    5.6861
      -Inf      -Inf      -Inf      -Inf    2.4369    3.2492    4.0615    4.8738    5.6861
      ⋮
```

Input Arguments

SE — Structuring element

`strel` or `offsetstrel` object

Structuring element, specified as a `strel` or `offsetstrel` object.

v — Translation offsets

numeric vector

Translation offsets, specified as a numeric vector. Each element specifies the amount of desired translation in the corresponding dimension.

Output Arguments

SE2 — Translated structuring element

`strel` or `offsetstrel` object

Translated structuring elements, returned as a `strel` or `offsetstrel` object.

See Also

`reflect`

Topics

“Structuring Elements”

Introduced before R2006a

truesize

Adjust display size of image

Syntax

```
truesize(fig,[mrows ncols])
truesize(fig)
```

Description

`truesize(fig,[mrows ncols])` adjusts the display size of an image in a figure, `fig`, to the dimensions `[mrows ncols]`, in pixels.

`truesize(fig)` adjusts the display size such that each image pixel covers one screen pixel.

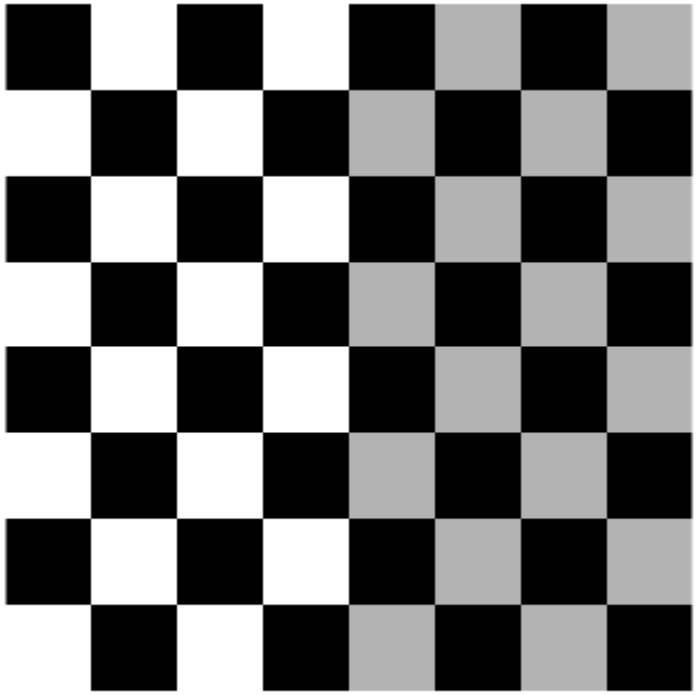
If you do not specify a figure, `truesize` adjusts the display size of the current figure.

Examples

Adjust Display Size of Image

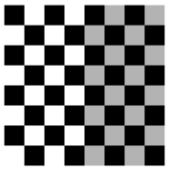
Create a default checkerboard image, which has size 80-by-80 pixels. Display the checkerboard image to fill the full size of the figure window. The image is magnified to fill the window.

```
c = checkerboard;
imshow(c,'InitialMagnification','fit')
```



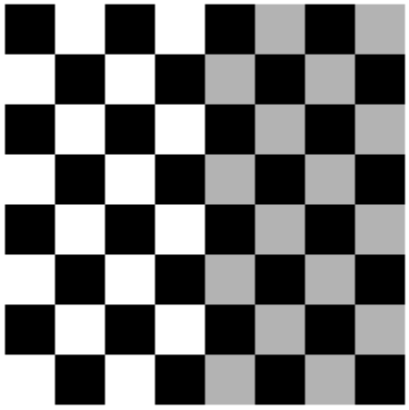
Display the checkerboard image so that each image pixel covers one screen pixel.

```
truesize
```



You can adjust the size of the figure window to arbitrary dimension. The image scales to fit within the figure window.

```
truesize([300 200]);
```



Input Arguments

`fig` — Figure

`gcf` (default) | figure handle

Figure containing a single image or a single image with a color bar, specified as a figure handle. By default, `truesize` uses the current figure, with handle `gcf`.

`[mrows ncols]` — Screen dimensions

2-element numeric row vector

Screen dimensions (in pixels) that the image should occupy, specified as a 2-element numeric row vector. By default, `[mrows ncols]` is equal to the image size, so each image pixel covers one screen pixel.

See Also

`imshow` | `iptsetpref` | `iptgetpref`

Introduced before R2006a

unitGenerator

Create unsupervised image-to-image translation (UNIT) generator network

Syntax

```
net = unitGenerator(inputSizeSource)
net = unitGenerator(inputSizeSource,Name,Value)
```

Description

`net = unitGenerator(inputSizeSource)` creates a UNIT generator network, `net`, for input of size `inputSizeSource`. For more information about the network architecture, see “UNIT Generator Network” on page 1-3102. The network has two inputs and four outputs:

- The two network inputs are images in the source and target domains. By default, the target image size is same as source image size. You can change the number of channels in the target image by specifying the '`NumTargetInputChannels`' name-value argument.
- Two of the network outputs are self-reconstructed outputs, in other words, source-to-source and target-to-target translated images. The other two network outputs are the source-to-target and target-to-source translated images.

This function requires Deep Learning Toolbox.

`net = unitGenerator(inputSizeSource,Name,Value)` modifies aspects of the UNIT generator network using name-value arguments.

Examples

Create UNIT Generator

Specify the network input size for RGB images of size 128-by-128.

```
inputSize = [128 128 3];
```

Create a UNIT generator that generates RGB images of the input size.

```
net = unitGenerator(inputSize)
```

```
net =
  dlnetwork with properties:

         Layers: [9x1 nnet.cnn.layer.Layer]
    Connections: [8x2 table]
     Learnables: [168x3 table]
          State: [0x3 table]
     InputNames: {'inputSource'  'inputTarget'}
    OutputNames: {1x4 cell}
    Initialized: 1
```

Display the network.

```
analyzeNetwork(net)
```

Create UNIT Generator with Five Residual Blocks

Specify the network input size for RGB images of size 128-by-128.

```
inputSize = [128 128 3];
```

Create a UNIT generator with five residual blocks, three of which are shared between the encoder and decoder modules.

```
net = unitGenerator(inputSize,"NumResidualBlocks",5, ...
    "NumSharedBlocks",3)
```

```
net =
  dlnetwork with properties:

         Layers: [9x1 nnet.cnn.layer.Layer]
    Connections: [8x2 table]
     Learnables: [152x3 table]
          State: [0x3 table]
     InputNames: {'inputSource'  'inputTarget'}
    OutputNames: {1x4 cell}
    Initialized: 1
```

Display the network.

```
analyzeNetwork(net)
```

Input Arguments

`inputSizeSource` — Input size of source image

3-element vector of positive integers

Input size of the source image, specified as a 3-element vector of positive integers. `inputSizeSource` has the form [*H W C*], where *H* is the height, *W* is the width, and *C* is the number of channels. The length of each dimension must be evenly divisible by `2^NumDownsamplingBlocks`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'NumDownsamplingBlocks',3` creates a network with 3 downsampling blocks

`NumDownsamplingBlocks` — Number of downsampling blocks

2 (default) | positive integer

Number of downsampling blocks in the source encoder and target encoder subnetworks, specified as a positive integer. In total, the encoder module downsamples the source and target input by a factor of 2^`NumDownsamplingBlocks`. The source decoder and target decoder subnetworks have the same number of upsampling blocks.

`NumResidualBlocks` — Number of residual blocks
5 (default) | positive integer

Number of residual blocks in the encoder module, specified as a positive integer. The decoder module has the same number of residual blocks.

`NumSharedBlocks` — Number of shared residual blocks
2 (default) | positive integer

Number of residual blocks in the shared encoder subnetwork, specified as a positive integer. The shared decoder subnetwork has the same number of residual blocks. The network should contain at least one shared residual block.

`NumTargetChannels` — Number of channels in target image
positive integer

Number of channels in the target image, specified as a positive integer. By default, '`NumTargetChannels`' is the same as the number of channels in the source image, `inputSizeSource`.

`NumFiltersInFirstBlock` — Number of filters in first convolution layer
64 (default) | positive even integer

Number of filters in the first convolution layer, specified as a positive even integer.

`FilterSizeInFirstAndLastBlocks` — Filter size in first and last convolution layers
7 (default) | positive odd integer | 2-element vector of positive odd integers

Filter size in the first and last convolution layers of the network, specified as a positive odd integer or 2-element vector of positive odd integers of the form [*height width*]. When you specify the filter size as a scalar, the filter has equal height and width.

`FilterSizeInIntermediateBlocks` — Filter size in intermediate convolution layers
3 (default) | 2-element vector of positive odd integers | positive odd integer

Filter size in intermediate convolution layers, specified as a positive odd integer or 2-element vector of positive odd integers of the form [*height width*]. The intermediate convolution layers are the convolution layers excluding the first and last convolution layer. When you specify the filter size as a scalar, the filter has identical height and width.

`ConvolutionPaddingValue` — Style of padding
`"symmetric-exclude-edge"` (default) | `"symmetric-include-edge"` | `"replicate"` | numeric scalar

Style of padding used in the network, specified as one of these values.

PaddingValue	Description	Example
Numeric scalar	Pad with the specified numeric value	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}2&2&2&2&2&2&2\\2&2&2&2&2&2&2\\2&2&3&1&4&2&2\\2&2&1&5&9&2&2\\2&2&2&6&5&2&2\\2&2&2&2&2&2&2\\2&2&2&2&2&2&2\end{bmatrix}$
`'symmetric-include-edge'`	Pad using mirrored values of the input, including the edge values	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}5&1&1&5&9&9&5\\1&3&3&1&4&4&1\\1&3&3&1&4&4&1\\5&1&1&5&9&9&5\\6&2&2&6&5&5&6\\6&2&2&6&5&5&6\\5&1&1&5&9&9&5\end{bmatrix}$
`'symmetric-exclude-edge'`	Pad using mirrored values of the input, excluding the edge values	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}5&6&2&6&5&6&2\\9&5&1&5&9&5&1\\4&1&3&1&4&1&3\\9&5&1&5&9&5&1\\5&6&2&6&5&6&2\\9&5&1&5&9&5&1\\4&1&3&1&4&1&3\end{bmatrix}$
`'replicate'`	Pad using repeated border elements of the input	$\begin{bmatrix}3&1&4\\1&5&9\\2&6&5\end{bmatrix} \rightarrow \begin{bmatrix}3&3&3&1&4&4&4\\3&3&3&1&4&4&4\\3&3&3&1&4&4&4\\1&1&1&5&9&9&9\\2&2&2&6&5&5&5\\2&2&2&6&5&5&5\\2&2&2&6&5&5&5\end{bmatrix}$

`UpsampleMethod` — Method used to upsample activations
`"transposedConv"` (default) | `"bilinearResize"` | `"pixelShuffle"`

Method used to upsample activations, specified as one of these values:

- `"transposedConv"` — Use a `transposedConv2dLayer` with a stride of [2 2].
- `"bilinearResize"` — Use a `convolution2dLayer` with a stride of [1 1] followed by a `resize2dLayer` with a scale of [2 2].
- `"pixelShuffle"` — Use a `convolution2dLayer` with a stride of [1 1] followed by a `depthToSpace2dLayer` with a block size of [2 2].

Data Types: `char` | `string`

`ConvolutionWeightsInitializer` — Weight initialization used in convolution layers
`"he"` (default) | `"glorot"` | `"narrow-normal"` | function

Weight initialization used in convolution layers, specified as `"glorot"`, `"he"`, `"narrow-normal"`, or a function handle. For more information, see “Specify Custom Weight Initialization Function” (Deep Learning Toolbox).

`ActivationLayer` — Activation function
`"relu"` (default) | `"leakyRelu"` | `"elu"` | layer object

Activation function to use in the network except after the first and final convolution layers, specified as one of these values. The `unitGenerator` function automatically adds a leaky ReLU layer after the first convolution layer. For more information and a list of available layers, see “Activation Layers” (Deep Learning Toolbox).

- `"relu"` — Use a `reluLayer`
- `"leakyRelu"` — Use a `leakyReluLayer` with a scale factor of 0.2
- `"elu"` — Use an `eluLayer`
- A layer object

`SourceFinalActivationLayer` — Activation function after final convolution in source decoder
`"tanh"` (default) | `"sigmoid"` | `"softmax"` | `"none"` | layer object

Activation function after the final convolution layer in the source decoder, specified as one of these values. For more information and a list of available layers, see “Output Layers” (Deep Learning Toolbox).

- `"tanh"` — Use a `tanhLayer`
- `"sigmoid"` — Use a `sigmoidLayer`
- `"softmax"` — Use a `softmaxLayer`
- `"none"` — Do not use a final activation layer
- A layer object

`TargetFinalActivationLayer` — Activation function after final convolution in target decoder
`"tanh"` (default) | `"sigmoid"` | `"softmax"` | `"none"` | layer object

Activation function after the final convolution layer in the target decoder, specified as one of these values. For more information and a list of available layers, see “Output Layers” (Deep Learning Toolbox).

- `"tanh"` — Use a `tanhLayer`
- `"sigmoid"` — Use a `sigmoidLayer`
- `"softmax"` — Use a `softmaxLayer`
- `"none"` — Do not use a final activation layer
- A layer object

Output Arguments

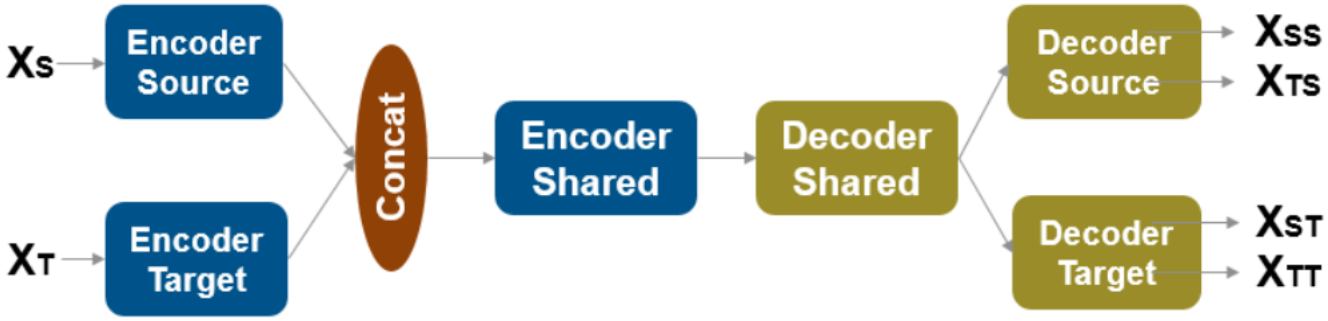
`net` — UNIT generator network
`dlnetwork` object

UNIT generator network, returned as a `dlnetwork` object.

More About

UNIT Generator Network

A UNIT generator network consists of three subnetworks in an encoder module followed by three subnetworks in a decoder module. The default network follows the architecture proposed by Liu, Breuel, and Kautz [1].



The encoder module downsamples the input by a factor of 2^`NumDownsamplingBlocks`. The encoder module consists of three subnetworks.

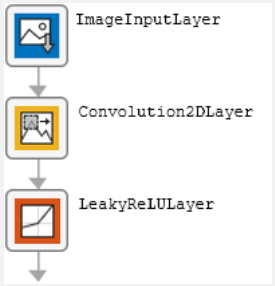
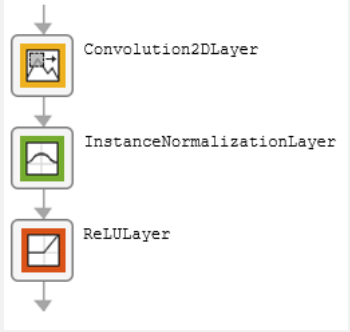
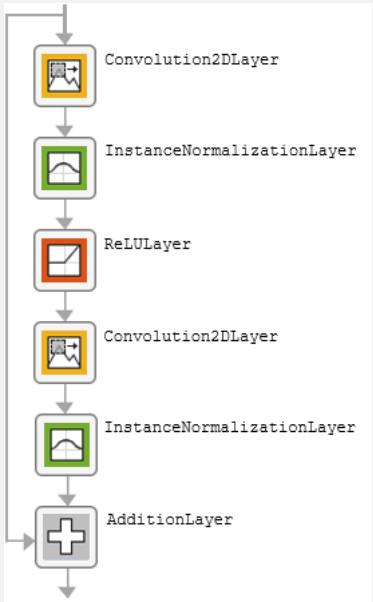
- The source encoder subnetwork, called 'encoderSourceBlock', has an initial block of layers that accepts data in the source domain, X_S. The subnetwork then has `NumDownsamplingBlocks` downsampling blocks that downsample the data and `NumResidualBlocks`–`NumSharedBlocks` residual blocks.
- The target encoder subnetwork, called 'encoderTargetBlock', has an initial block of layers that accepts data in the target domain, X_S. The subnetwork then has `NumDownsamplingBlocks` downsampling blocks that downsample the data, and `NumResidualBlocks`–`NumSharedBlocks` residual blocks.
- The output of the source encoder and target encoder are combined by a `concatenationLayer`
- The shared residual encoder subnetwork, called 'encoderSharedBlock', accepts the concatenated data and has `NumSharedBlocks` residual blocks.

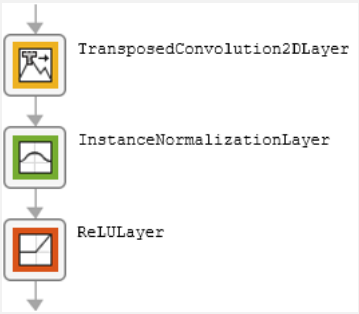
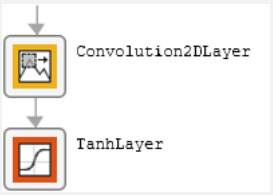
The decoder module consists of three subnetworks that perform a total of `NumDownsamplingBlocks` upsampling operations on the data.

- The shared residual decoder subnetwork, called 'decoderSharedBlock', accepts data from the encoder and has `NumSharedBlocks` residual blocks.
- The source decoder subnetwork, called 'decoderSourceBlock', has `NumResidualBlocks`–`NumSharedBlocks` residual blocks, `NumDownsamplingBlocks` downsampling blocks that downsample the data, and a final block of layers that returns the output. This subnetwork returns two outputs in the source domain: X_{TS} and X_{SS}. The output X_{TS} is an image translated from the target domain to the source domain. The output X_{SS} is a self-reconstructed image from the source domain to the source domain.
- The target decoder subnetwork, called 'decoderTargetBlock', has `NumResidualBlocks`–`NumSharedBlocks` residual blocks, `NumDownsamplingBlocks` downsampling blocks that downsample the data, and a final block of layers that returns the output. This subnetwork returns two outputs in the target domain: X_{ST} and X_{TT}. The output X_{TS} is an image translated from the

source domain to the target domain. The output X_{TT} is a self-reconstructed image from the target domain to the target domain.

The table describes the blocks of layers that comprise the subnetworks.

Block Type	Layers	Diagram of Default Block
Initial block	• An `imageInputLayer`. • A `convolution2dLayer` with a stride of [1 1] and a filter size of `FilterSizeInFirstAndLastBlocks`. • A `leakyReluLayer` with a scale factor of 0.2.	
Downsampling block	• A `convolution2dLayer` with a stride of [2 2] to perform downsampling. The convolution layer has a filter size of `FilterSizeInIntermediateBlocks`. • An `instanceNormalizationLayer`. • An activation layer specified by the `ActivationLayer` name-value argument.	
Residual block	• A `convolution2dLayer` with a stride of [1 1] and a filter size of `FilterSizeInIntermediateBlocks`. • An `instanceNormalizationLayer`. • An activation layer specified by the `ActivationLayer` name-value argument. • A second `convolution2dLayer`. • A second `instanceNormalizationLayer`. • An `additionLayer` that provides a skip connection between every block.	

Block Type	Layers	Diagram of Default Block
Upsampling block	• An upsampling layer that upsamples by a factor of 2 according to the `UpsampleMethod` name-value argument. The convolution layer has a filter size of `FilterSizeInIntermediateBlocks`. • An `instanceNormalizationLayer`. • An activation layer specified by the `ActivationLayer` name-value argument.	
Final block	• A `convolution2dLayer` with a stride of [1 1] and a filter size of `FilterSizeInFirstAndLastBlocks`. • An optional activation layer specified by the `SourceFinalActivationLayer` and `TargetFinalActivationLayer` name-value arguments.	

Tips

- You can create the discriminator network for UNIT by using the `patchGANDiscriminator` function.
- Train the UNIT GAN network using a custom training loop.
- To perform domain translation of source image to target image and vice versa, use the `unitPredict` function.
- For shared latent feature encoding, the arguments '`NumSharedBlocks`' and '`NumResidualBlocks`' must be greater than 0.

References

[1] Liu, Ming-Yu, Thomas Breuel, and Jan Kautz. "Unsupervised Image-to-Image Translation Networks." *Advances in Neural Information Processing Systems 30 (NIPS 2017)*. Long Beach, CA: 2017. https://arxiv.org/abs/1703.00848.

See Also

`unitPredict` | `patchGANDiscriminator` | `cycleGANGenerator` | `pix2pixHDGlobalGenerator`

Topics

“Unsupervised Day-to-Dusk Image Translation Using UNIT”
“Get Started with GANs for Image-to-Image Translation”
“Create Modular Neural Networks”
“List of Deep Learning Layers” (Deep Learning Toolbox)

Introduced in R2021a

unitPredict

Perform inference using unsupervised image-to-image translation (UNIT) network

Syntax

```
translatedImage = unitPredict(net,inputImage)
translatedImage = unitPredict(net,inputImage,"OutputType",outputType)
```

Description

`translatedImage = unitPredict(net,inputImage)` performs unsupervised image-to-image translation of image `inputImage` using the UNIT network `net`.

This function requires Deep Learning Toolbox.

`translatedImage = unitPredict(net,inputImage,"OutputType",outputType)` specifies the direction of image-to-image translation for inference using the `outputType` argument. The direction can be source-to-target or target-to-source.

Examples

Perform Source-to-Target Image Translation

Download a pretrained UNIT generator network that translates images between daytime and dusk lighting conditions using the helper function `downloadTrainedDayDuskGeneratorNet`. The source domain is daytime lighting and the target domain is dusk lighting.

```
trainedUNIT_url = 'https://ssd.mathworks.com/supportfiles/vision/data/trainedDayDuskUNITGenerato
trainedUNIT_filename = 'trainedDayDuskUNITGeneratorNet.mat';
downloadTrainedDayDuskGeneratorNet(trainedUNIT_url,pwd);
load(trainedUNIT_filename);
```

Read and display a test image acquired in daytime conditions.

```
input = imread("car1.jpg");
imshow(input)
```

Preprocess the image so that it is compatible with the network. Convert the data to data type `single` in the range [-1, 1]. Decrease the size of the image, and store the data in a `dlarray` object.

```
input = (im2single(input) - 0.5)/0.5;
input = imresize(input,0.25);
dlInput = dlarray(input,"SSCB");
```

Translate the source image to the target domain using the UNIT generator network.

```
dlOutput = unitPredict(gen,dlInput);
```

Extract the translated image data from the `dlarray` object and rescale the data to the range [0, 1]. Display the translated image. The translated image resembles images acquired in dusk conditions.

```
output = rescale(extractdata(dlOutput));
imshow(output)
```

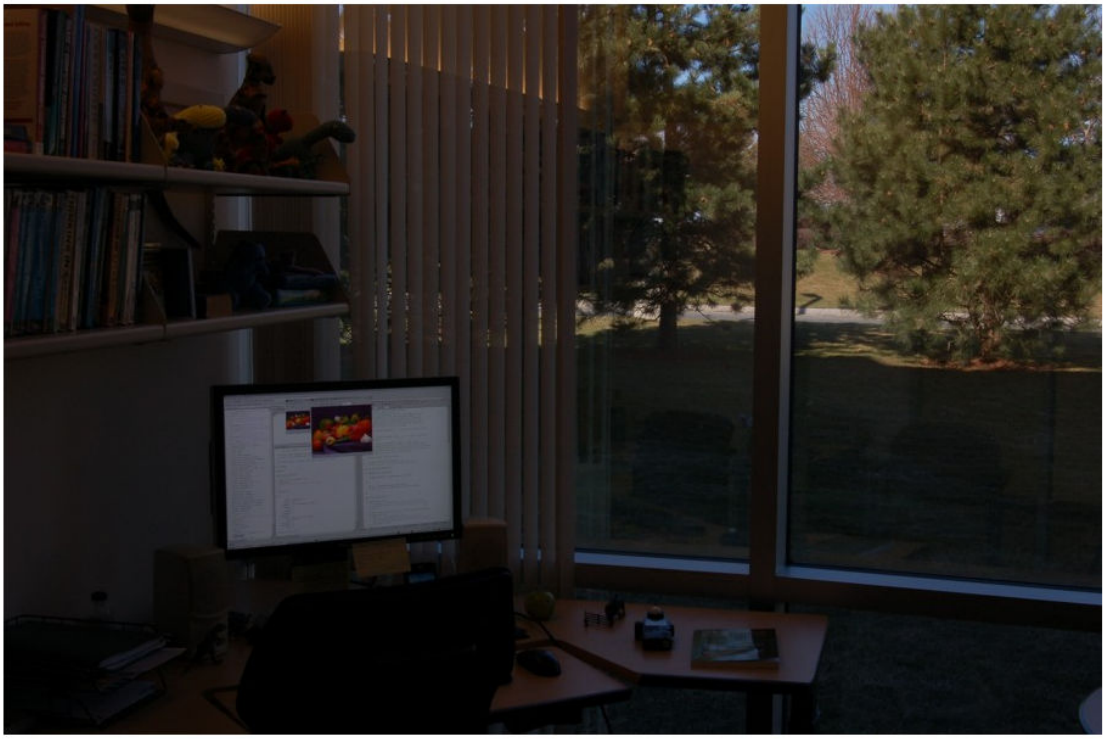
Perform Target-to-Source Image Translation

Download a pretrained UNIT generator network that translates images between daytime and dusk lighting conditions using the helper function `downloadTrainedDayDuskGeneratorNet`. The source domain is daytime lighting and the target domain is dusk lighting.

```
trainedUNIT_url = 'https://ssd.mathworks.com/supportfiles/vision/data/trainedDayDuskUNITGenerato
trainedUNIT_filename = 'trainedDayDuskUNITGeneratorNet.mat';
downloadTrainedDayDuskGeneratorNet(trainedUNIT_url,pwd);
load(trainedUNIT_filename);
```

Read and display a test image acquired in dusk conditions.

```
input = imread("office_2.jpg");
imshow(input)
```



Preprocess the image so that it is compatible with the network. Convert the data to data type `single` in the range [-1, 1]. Store the data in a `dlarray` object.

```
input = (im2single(input) - 0.5)/0.5;
dlInput = dlarray(input,"SSCB");
```

Translate the target image to the source domain using the pretrained UNIT generator network, `gen`.

```
dlOutput = unitPredict(gen,dlInput,"OutputType","TargetToSource");
```

Extract the translated image data from the `dlarray` object and rescale the data to the range [0, 1]. Display the translated image. The translated image resembles images acquired in daytime lighting conditions.

```
output = rescale(extractdata(dlOutput));
imshow(output)
```

Input Arguments

net — UNIT generator network
`dlnetwork` object

UNIT generator network, specified as a `dlnetwork` object. You can create a UNIT generator network using the `unitGenerator` function.

inputImage — Input image
formatted `dlarray` object

Input image for image-to-image translation, specified as a formatted `dlarray` object.

outputType — Direction of image-to-image translation
`"SourceToTarget"` (default) | `"TargetToSource"`

Direction of image-to-image translation for inference, specified as one of these values.

- `"SourceToTarget"` – translate from the source domain to the target domain
- `"TargetToSource"` – translate from the target domain to the source domain

Data Types: `char` | `string`

Output Arguments

`translatedImage` — Inferred image
`dlarray` object

Inferred image after image-to-image translation, returned as a `dlarray` object.

See Also

`unitGenerator`

Topics
“Unsupervised Day-to-Dusk Image Translation Using UNIT”

Introduced in R2021a

visboundaries

Plot region boundaries

Syntax

```
visboundaries(BW)
visboundaries(B)
visboundaries(ax, ___ )
visboundaries( ___ ,Name,Value)
h = visboundaries( ___ )
```

Description

`visboundaries(BW)` draws boundaries of regions in the binary image `BW` on the current axes. `BW` is a 2-D binary image where pixels that are logical `true` belong to the foreground region and pixels that are logical `false` constitute the background. `visboundaries` uses `bwboundaries` to find the boundary pixel locations in the image.

`visboundaries(B)` draws region boundaries specified by `B`, where `B` is a cell array containing the boundary pixel locations of the regions, similar in structure to the first output from `bwboundaries`.

`visboundaries(ax, ___ )` draws region boundaries on the axes specified by `ax`. Specify `ax` as the first input argument followed by any of the input argument combinations in the previous syntaxes.

`visboundaries( ___ ,Name,Value)` uses name-value arguments to specify additional properties of the boundaries.

`h = visboundaries( ___ )` returns a handle `h`, for the boundaries.

Examples

Compute Boundaries and Plot on Image

Read image.

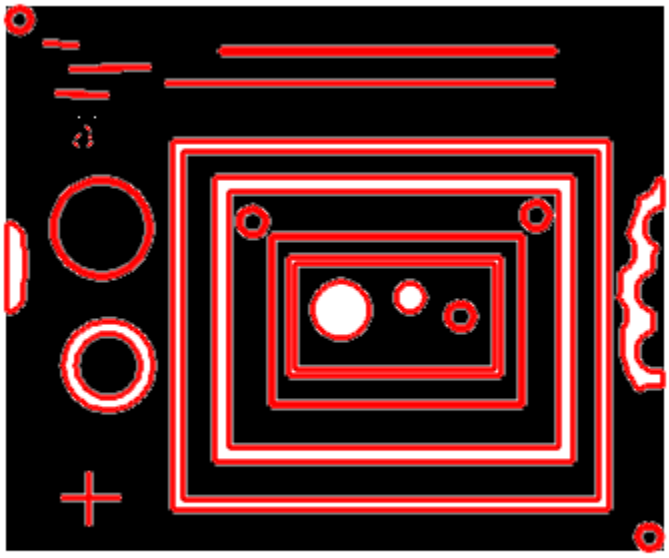
```
BW = imread('blobs.png');
```

Compute boundaries.

```
B = bwboundaries(BW);
```

Display image and plot boundaries on image.

```
imshow(BW)
hold on
visboundaries(B)
```



Visualize Segmentation Result

Read image and display it.

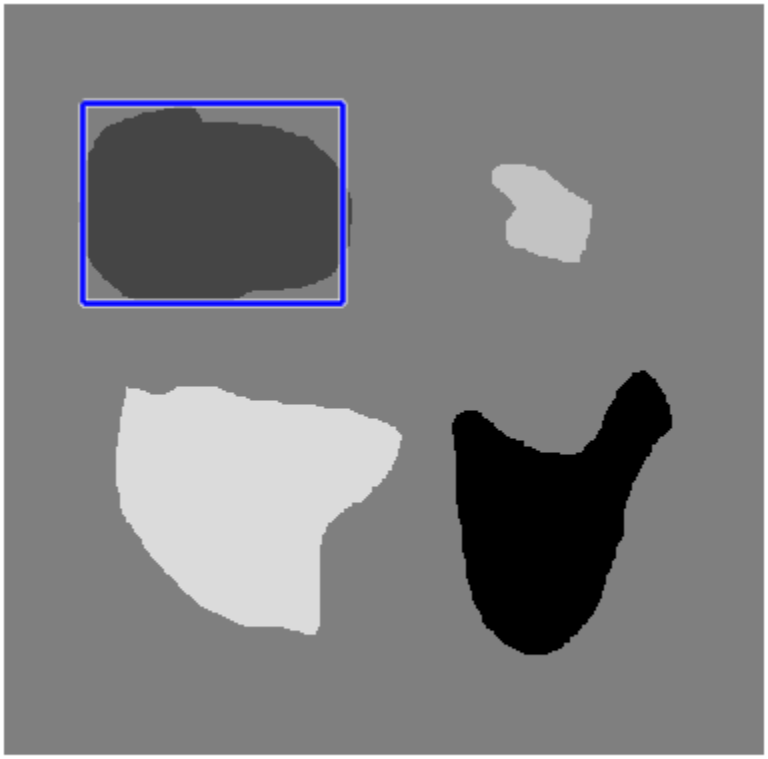
```
I = imread('toyobjects.png');
imshow(I)
hold on
```

Segment the image using the active contours (snakes) algorithm. First, specify the initial contour location close to the object that is to be segmented.

```
mask = false(size(I));
mask(50:150,40:170) = true;
```

Display the initial contour on the original image in blue.

```
visboundaries(mask,'Color','b');
```

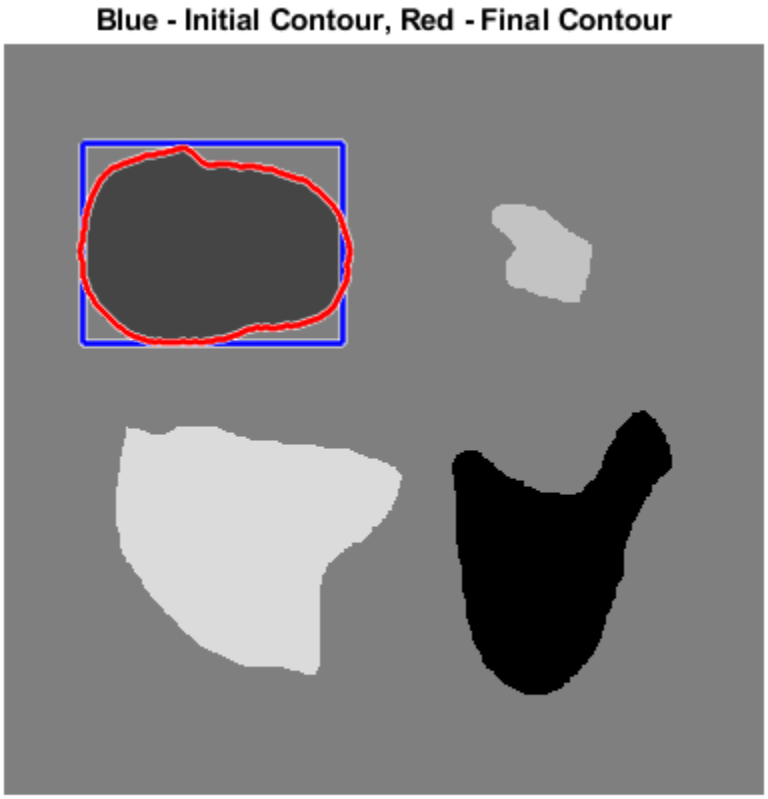


Segment the image using the `'edge'` method using 200 iterations.

```
bw = activecontour(I,mask,200,'edge');
```

Display the final contour on the original image in red.

```
visboundaries(bw,'Color','r');
title('Blue - Initial Contour, Red - Final Contour');
```



Input Arguments

`BW` — Binary image

logical array

Binary image, specified as a logical array.

Data Types: `logical`

`B` — Boundary pixel locations

cell array of *Q*-by-2 matrices

Boundary pixel locations, specified as a cell array. Each cell contains a *Q*-by-2 matrix, where *Q* is the number of boundary pixels for the corresponding region. Each row of these *Q*-by-2 matrices contains the row and column coordinates of a boundary pixel.

Data Types: `cell`

`ax` — Image on which to draw boundaries

current axes (default) | axes object

Image on which to draw boundaries, specified as an axes object.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `visboundaries(bw,'Color','b');`

`Color` — Color of boundary

`'red'` (default) | RGB triplet | hexadecimal color code | color name | short color name

Color of the boundary, specified as an RGB triplet, a hexadecimal color code, a color name, or a short color name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	
`'none'`	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	
`[0.8500 0.3250 0.0980]`	`'#D95319'`	

RGB Triplet	Hexadecimal Color Code	Appearance
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

Example: 'Color','r'

Example: 'Color','green'

Example: 'Color',[0 0 1]

Example: 'Color','#FF8800'

LineStyle — Style of boundary line
'-' (default) | '--' | ':' | '-.' | 'none'

Line style of boundary edge, specified as the comma-separated pair consisting of 'LineStyle' and any line specifier in the table below.

Line Style	Description	Resulting Line
'-'	Solid line	
'--'	Dashed line	
':'	Dotted line	
'-.'	Dash-dotted line	
'none'	No line	No line

Example: 'LineStyle','-.'

LineWidth — Width of the line used for the boundary
2 (default) | positive number

Width of the line used for the boundary, specified as a positive number. Specify this value in points, where one point = 1/72 inch.

Example: 'LineWidth',4

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

EnhanceVisibility — Augment drawn boundary with contrasting features
true or 1 (default) | false or 0

Augment the drawn boundary with contrasting features to improve visibility on a varying background, specified as a numeric or logical 1 (true) or 0 (false).

Example: 'EnhanceVisibility',true

Data Types: logical

Output Arguments

h — Boundary lines

`hggroup` object

Boundary line, returned as an `hggroup` object. `h` is the child of the axes `ax` if specified, otherwise `h` is the child of the current axes.

See Also

`bwboundaries` | `bwperim` | `bwtraceboundary` | `viscircles`

Introduced in R2015a

viscircles

Create circle

Syntax

```
viscircles(centers,radii)
viscircles(ax,centers,radii)
viscircles(___,Name,Value)
h = viscircles(___)
```

Description

`viscircles(centers,radii)` draws circles with specified `centers` and `radii` onto the current axes.

`viscircles(ax,centers,radii)` draws circles onto the axes specified by `ax`.

`viscircles(___,Name,Value)` uses name-value pair arguments to specify additional properties of the circles.

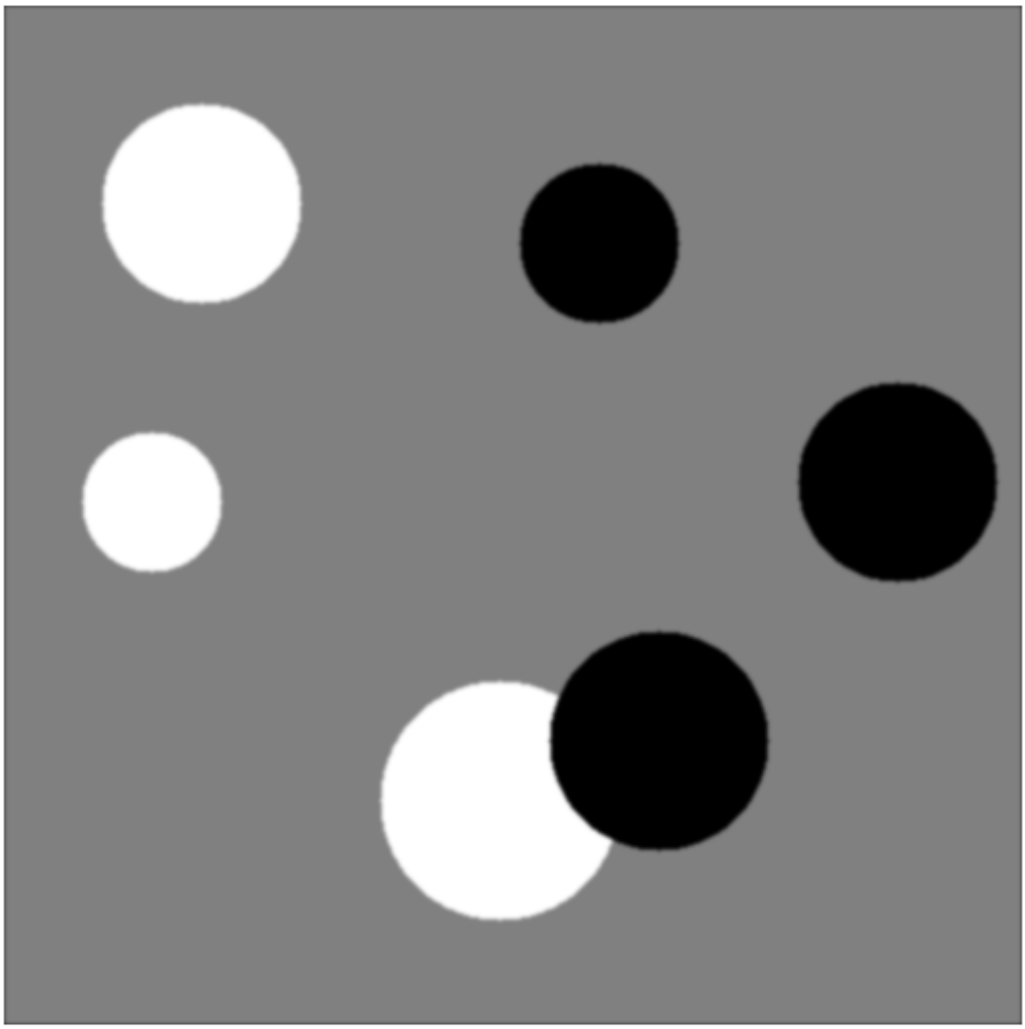
`h = viscircles(___)` returns a handle, `h`, to the drawn circles.

Examples

Draw Lines Around Bright and Dark Circles in Image

Read the image into the workspace and display it.

```
A = imread('circlesBrightDark.png');
imshow(A)
```



Define the radius range.

```
Rmin = 30;
Rmax = 65;
```

Find all the bright circles in the image within the radius range.

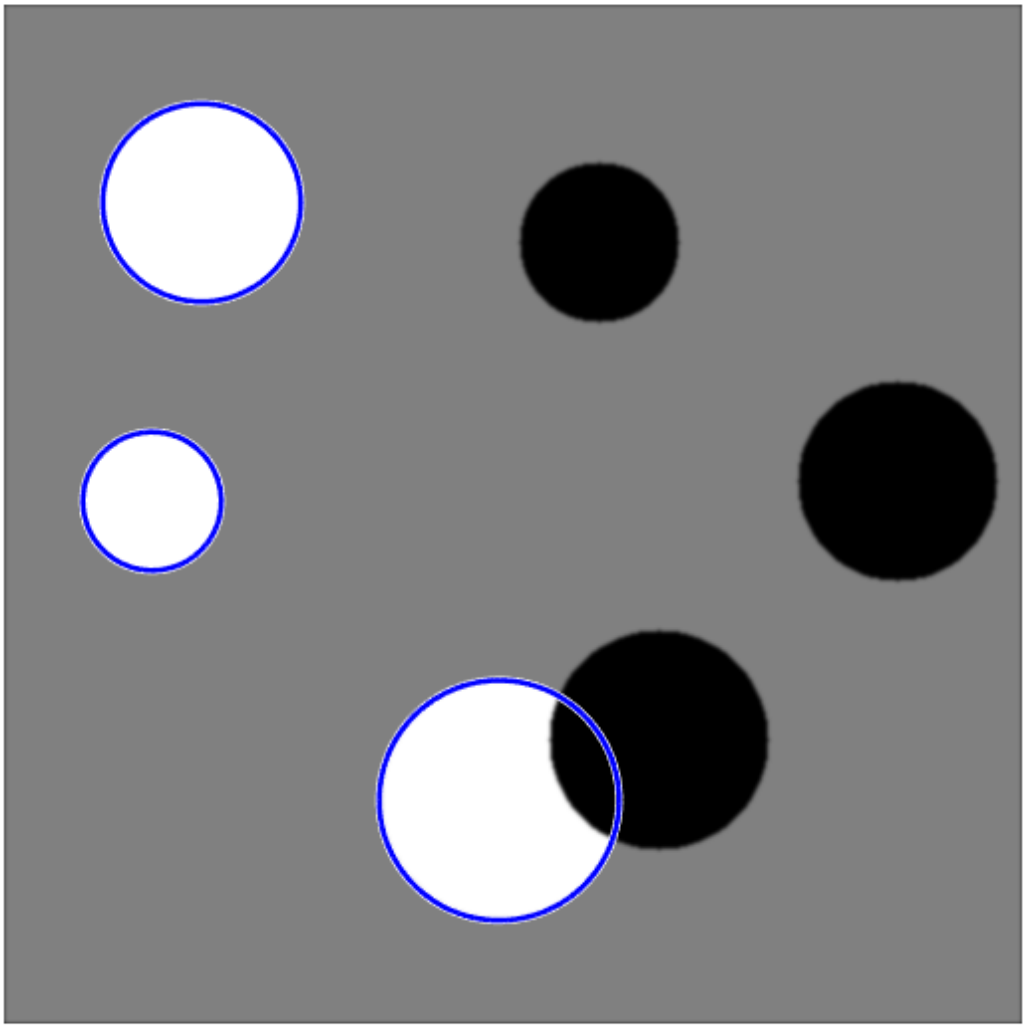
```
[centersBright, radiiBright] = imfindcircles(A,[Rmin Rmax],'ObjectPolarity','bright');
```

Find all the dark circles in the image within the radius range.

```
[centersDark, radiiDark] = imfindcircles(A,[Rmin Rmax],'ObjectPolarity','dark');
```

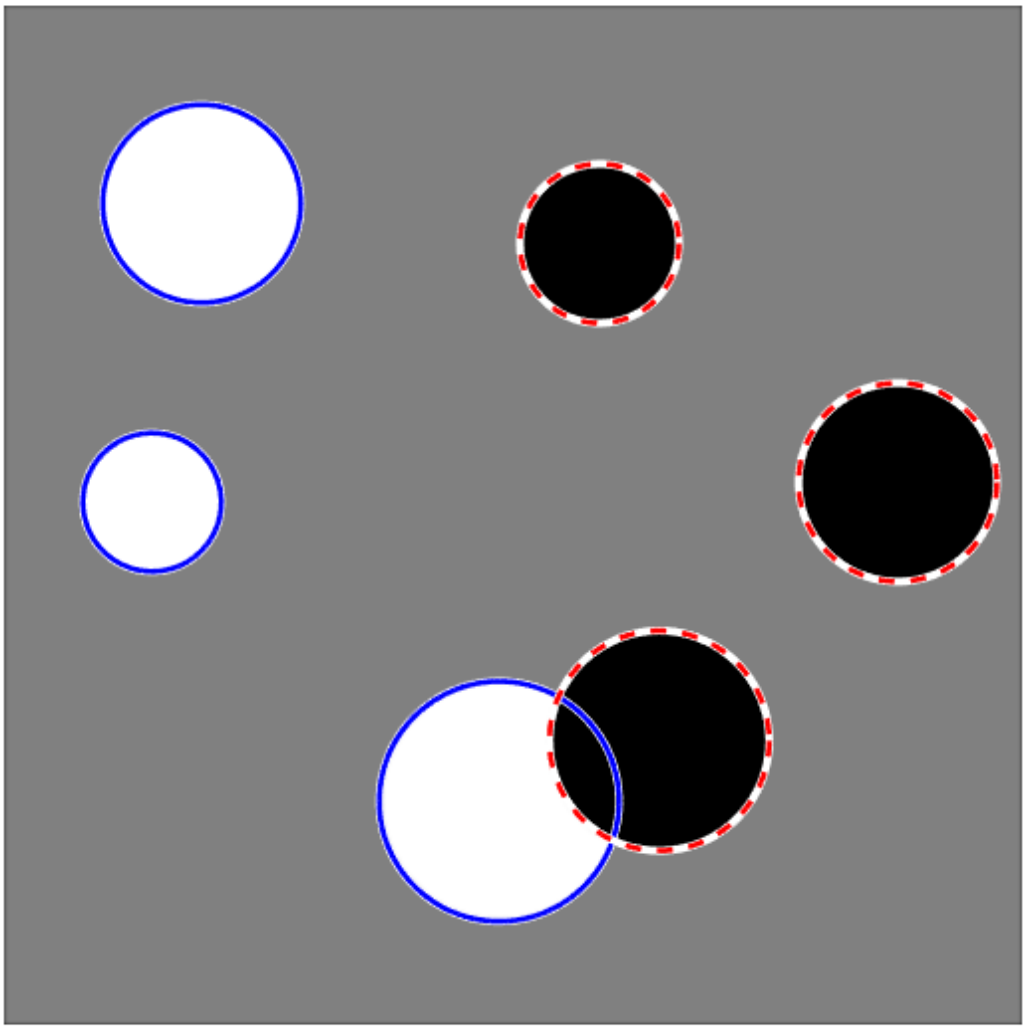
Draw blue lines around the edges of the bright circles.

```
viscircles(centersBright, radiiBright,'Color','b');
```



Draw red dashed lines around the edges of the dark circles.

```
viscircles(centersDark, radiiDark,'LineStyle','--');
```

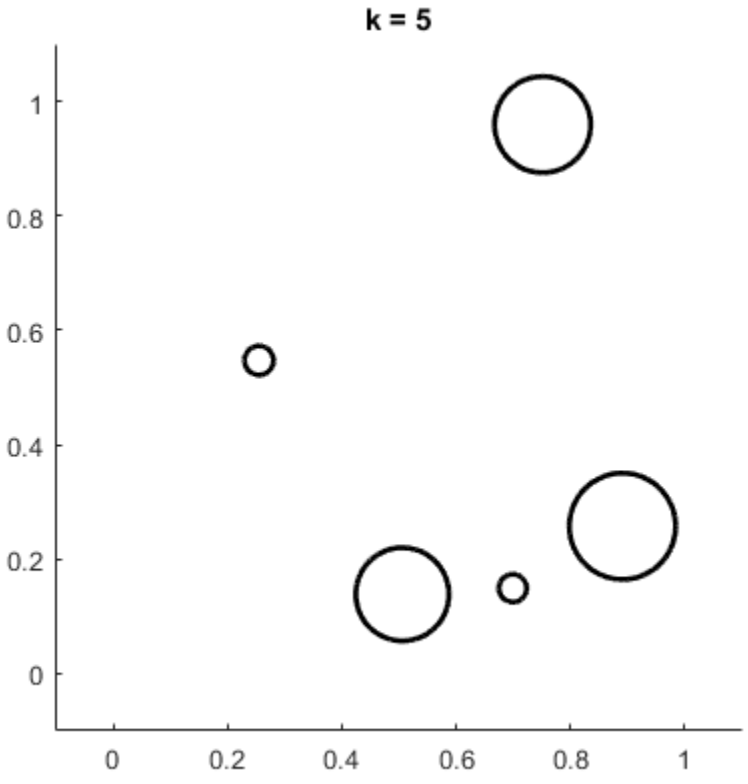


Clear Axes Before Plotting Circles

The `viscircles` function does not clear the target axes before plotting circles. To remove circles that have been previously plotted in an axes, use the `cla` function. To illustrate, this example creates a new figure and then loops, drawing a set of circles with each iteration, clearing the axes each time.

```
figure
colors = {'b','r','g','y','k'};

for k = 1:5
    % Create 5 random circles to display,
    X = rand(5,1);
```

```
    Y = rand(5,1);
    centers = [X Y];
    radii = 0.1*rand(5,1);

    % Clear the axes.
    cla

    % Fix the axis limits.
    xlim([-0.1 1.1])
    ylim([-0.1 1.1])

    % Set the axis aspect ratio to 1:1.
    axis square

    % Set a title.
    title(['k = ' num2str(k)])

    % Display the circles.
    viscircles(centers,radii,'Color',colors{k});

    % Pause for 1 second.
    pause(1)
end
```



Input Arguments

`centers` — Coordinates of circle centers

two-column matrix

Coordinates of circle centers, specified as a P-by-2 matrix, such as that obtained from `imfindcircles`. The *x*-coordinates of the circle centers are in the first column and the *y*-coordinates are in the second column. The coordinates can be integers (of any numeric type) or floating-point values (of type `double` or `single`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`radii` — Circle radii

column vector

Circle radii, specified as a column vector such as that returned by `imfindcircles`. The radius value at `radii(j)` corresponds to the circle with center coordinates `centers(j,:)`. The values of `radii` can be nonnegative integers (of any numeric type) or floating-point values (of type `double` or `single`).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ax` — Axes in which to draw circles

handle

Axes in which to draw circles, specified as a handle object returned by `gca` or `axes`.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `viscircles(centers,radii,'Color','b')` specifies blue circle edges, using the short color name for blue.

`EnhanceVisibility` — Augment drawn circles with contrasting features to improve visibility

`true` (default) | `false`

Augment drawn circles with contrasting features to improve visibility, specified as a logical value `true` or `false`. If you set the value to `true`, then `viscircles` draws a contrasting circle below the colored circle.

Data Types: `logical`

`Color` — Color of boundary

`'red'` (default) | RGB triplet | hexadecimal color code | color name | short color name

Color of the boundary, specified as an RGB triplet, a hexadecimal color code, a color name, or a short color name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	
`'none'`	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	
`[0.8500 0.3250 0.0980]`	`'#D95319'`	
`[0.9290 0.6940 0.1250]`	`'#EDB120'`	
`[0.4940 0.1840 0.5560]`	`'#7E2F8E'`	
`[0.4660 0.6740 0.1880]`	`'#77AC30'`	
`[0.3010 0.7450 0.9330]`	`'#4DBEEE'`	
`[0.6350 0.0780 0.1840]`	`'#A2142F'`	

Example: `viscircles(centers,radii,'Color','r');`

Example: `viscircles(centers,radii,'Color','green');`

Example: `viscircles(centers,radii,'Color',[0 0 1]);`

Example: `viscircles(centers,radii,'Color','#FF8800');`

`LineStyle` — Line style of circle edge

`'-'` (default) | `'--'` | `':'` | `'-.'` | `'none'`

Line style of circle edge, specified as the comma-separated pair consisting of `'LineStyle'` and any line specifier in the table below.

Line Style	Description	Resulting Line
`'-'`	Solid line	————
`'--'`	Dashed line	- — — — —
`':'`	Dotted line	.....
`'-.'`	Dash-dotted line	—·—·—·—·-
`'none'`	No line	No line

`LineWidth` — Width of circle edge
2 (default) | positive number

Width of circle edge, specified a positive number. Line width is expressed in points, where each point equals 1/72 of an inch.

Data Types: `double`

Output Arguments

`h` — Circles drawn
`hggroup` object

Circles drawn, returned as an `hggroup` object. `h` is the child of the axes `ax` if specified, otherwise `h` is the child of the current axes.

See Also
Image Viewer | `visboundaries` | `imfindcircles` | `imdistline`

Introduced in R2012a

volshow

Display volume

Description

A `volshow` object displays volume and enables you to modify the appearance of the display.

Creation

Syntax

```
volshow(V)
volshow(V,config)
volshow(V,Name,Value)
vs = volshow( ___ )
```

Description

`volshow(V)` displays 3-D grayscale volume `V` in a figure. You can rotate and zoom in and out on the display interactively using the mouse.

`volshow(V,config)` displays the 3-D grayscale volume `V`. `config` is a struct exported from the **Volume Viewer** app. The `config` struct controls visualization of the volume, containing values for `volshow` object properties.

`volshow(V,Name,Value)` displays the volume, using one or more name-value pairs to set properties that control the visualization of the volume. For a list of name-value pairs, see “Properties” on page 1-3128. Enclose each property name in quotes.

Example: `volshow(V,'BackgroundColor','w')` displays 3-D grayscale volume `V` in a figure with a white background color.

`vs = volshow( ___ )` returns a `volshow` object with properties that can be used to control visualization of the volume.

Input Arguments

`V` — 3-D grayscale volume

numeric array

3-D grayscale volume, specified as a numeric array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

`config` — Rendering information exported by Volume Viewer

`struct`

Rendering information exported by **Volume Viewer**, specified as a `struct`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `logical`

Properties

`Alphamap` — Transparency map for volume content

256-by-1 numeric vector

Transparency map for the volume content, specified as a 256-by-1 numeric array, with values in the range [0, 1]. The default transparency map is the vector `linspace(0,1,256)'`.

`BackgroundColor` — Background color

`[0.3 0.75 0.93]` (default) | RGB triplet | color name | short color name

Background color, specified as an RGB triplet, a color name, or a short color name.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
`[0 0.4470 0.7410]`	
`[0.8500 0.3250 0.0980]`	
`[0.9290 0.6940 0.1250]`	
`[0.4940 0.1840 0.5560]`	
`[0.4660 0.6740 0.1880]`	
`[0.3010 0.7450 0.9330]`	
`[0.6350 0.0780 0.1840]`	

Example: `'BackgroundColor','r'`

Example: `'BackgroundColor','green'`

Example: `'BackgroundColor',[0 0.4470 0.7410]`

`CameraPosition` — Location of camera

`[4 4 2.5]` (default) | 3-element vector

Location of camera, or the viewpoint, specified as a 3-element vector of the form `[x y z]`. This vector defines the axes coordinates of the camera location, which is the point from which you view the axes. The camera is oriented along the view axis, which is a straight line that connects the camera position and the camera target. Changing the `CameraPosition` property changes the point from which you view the volume. For an illustration, see “Camera Graphics Terminology”. Interactively rotating the volume modifies the value of this property.

`CameraTarget` — Point used as camera target
`[0 0 0]` (default) | 3-element vector

Point used as camera target, specified as a 3-element vector of the form `[x y z]`. The camera is oriented along the view axis, which is a straight line that connects the camera position and the camera target. For an illustration, see “Camera Graphics Terminology”.

`CameraUpVector` — Vector defining upwards direction
`[0 0 1]` (default) | 3-element vector

Vector defining upwards direction, specified as a 3-element direction vector of the form `[x y z]`. By default, `volshow` defines the *z*-axis as the up direction (`[0 0 1]`). For an illustration, see “Camera Graphics Terminology”. Interactively rotating the volume modifies the value of this property.

`CameraViewAngle` — Field of view
15 (default) | numeric scalar

Field of view, specified as a scalar angle in the range [0, 180). The larger the angle, the larger the field of view. Also, as the angle increases, objects appear smaller in the scene. For an illustration, see “Camera Graphics Terminology”.

`Colormap` — Colormap of volume content
`gray(256)` (default) | 256-by-3 numeric array

Colormap of the volume content, specified as a 256-by-3 numeric array with values in the range [0, 1].

`InteractionsEnabled` — Interactivity of volume
`true` (default) | `false`

Interactivity of the volume, specified as `true` or `false`. When `true`, you can zoom using the mouse scroll wheel, and rotate by clicking and dragging on the volume. Rotation and zoom are performed about the value specified by the `CameraTarget` property. When `false`, you cannot interact with the volume.

`IsosurfaceColor` — Isosurface color
RGB triplet | color name | short color name

Isosurface color, specified as an RGB triplet, a color name, or a short color name. This property specifies the volume color when the `Renderer` property is set to `'Isosurface'`.

You can specify any color using an RGB triplet. An RGB triplet is a 3-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1].

You can specify some common colors by name as a string scalar or character vector. This table lists the named color options and the equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Here are the RGB triplets for the default colors that MATLAB uses in many types of plots.

RGB Triplet	Appearance
[0 0.4470 0.7410]	
[0.8500 0.3250 0.0980]	
[0.9290 0.6940 0.1250]	
[0.4940 0.1840 0.5560]	
[0.4660 0.6740 0.1880]	
[0.3010 0.7450 0.9330]	
[0.6350 0.0780 0.1840]	

Example: `'IsosurfaceColor','r'`

Example: `'IsosurfaceColor','green'`

Example: `'IsosurfaceColor',[0 0.4470 0.7410]`

`Isovalue` — Value that defines volume surface
`0.49` (default) | nonnegative number

Value that defines the volume surface drawn when the `Renderer` property is set to `'Isosurface'`, specified as a nonnegative number in the range [0, 1].

`Lighting` — Include light source in rendering
`true` (default) | `false`

Include light source in rendering, specified as a logical scalar.

`Parent` — Parent of `volshow` object
`gcf` (default) | uipanel | figure

Parent of the `volshow` object, specified as a handle to a `uipanel` or `figure`. If you do not specify a parent, then the parent of the `volshow` object is `gcf`.

`Renderer` — Rendering style
`'VolumeRendering'` | `'MaximumIntensityProjection'` | `'Isosurface'`

Rendering style, specified as one of the values in this table. When the volume is `logical`, the default rendering style is `'Isosurface'`, otherwise the default rendering style is `'VolumeRendering'`.

Value	Description
`'VolumeRendering'`	View the volume based on the specified color and transparency for each voxel.
`'MaximumIntensityProjection'`	View the voxel with the highest intensity value for each ray projected through the data.
`'Isosurface'`	View an isosurface of the volume specified by the value in `Isovalue`.

`ScaleFactors` — Scale factors used to rescale volume
`[1 1 1]` (default) | 1-by-3 positive numeric array

Scale factors used to rescale volume, specified as a 1-by-3 positive numeric array. The values in the array correspond to the scale factor applied in the *x*, *y*, and *z* direction.

Object Functions

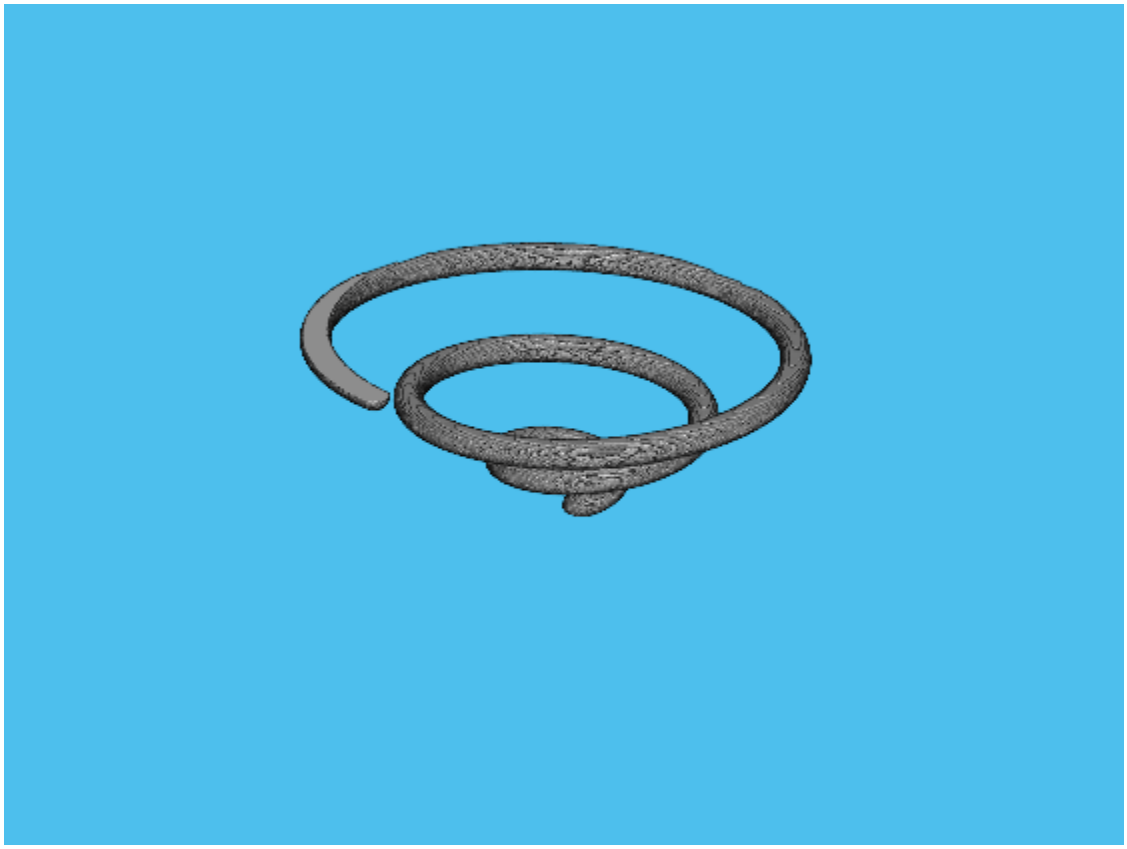
setVolume Set new volume

Examples

Create Animated GIF of Spiral Volume

Load and view the volume.

```
load('spiralVol.mat');
h = volshow(spiralVol);
```



Specify the name of the GIF file.

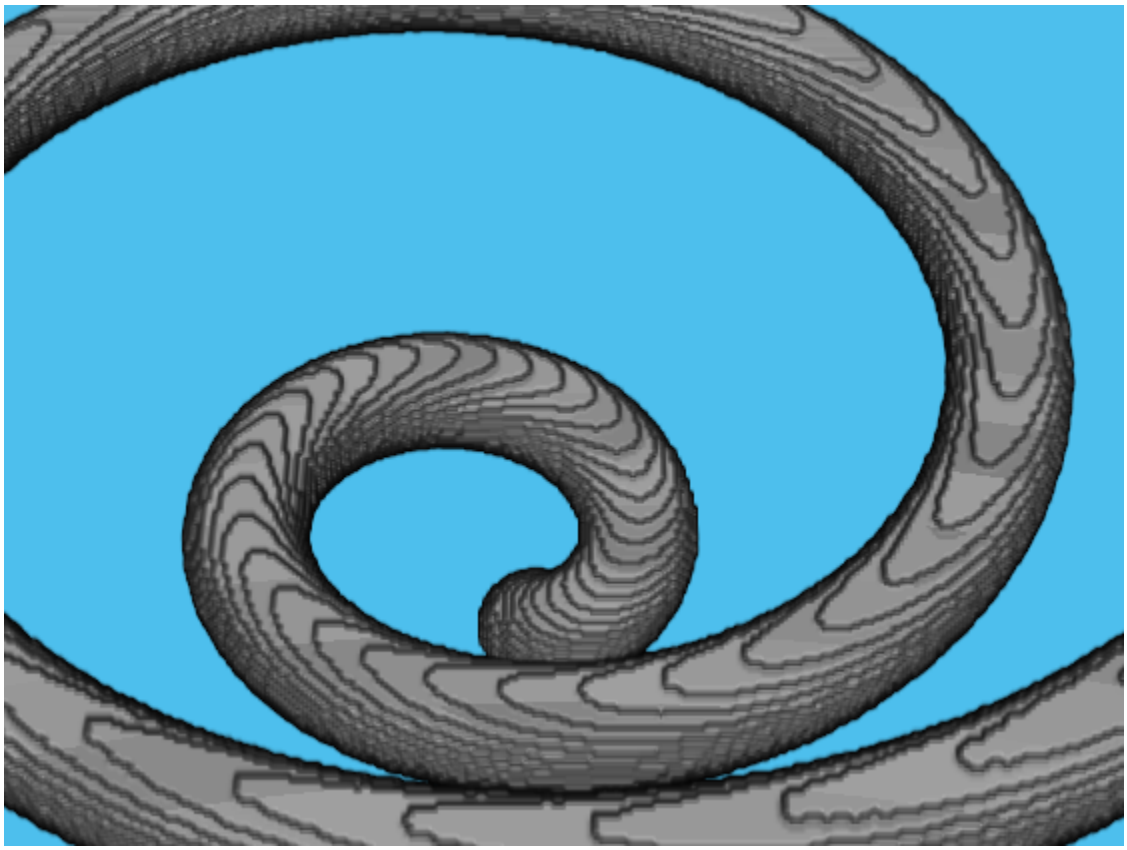
```
filename = 'animatedSpiral.gif';
```

Create an array of camera positions around the unit circle.

```
vec = linspace(0,2*pi(),120)';
myPosition = [cos(vec) sin(vec) ones(size(vec))];
```

Loop through and create an image at each camera position.

```
for idx = 1:120
    % Update current view.
    h.CameraPosition = myPosition(idx,:);
    % Use getframe to capture image.
    I = getframe(gcf);
    [indI,cm] = rgb2ind(I.cdata,256);
    % Write frame to the GIF File.
    if idx == 1
        imwrite(indI, cm, filename, 'gif', 'Loopcount', inf, 'DelayTime', 0.05);
    else
        imwrite(indI, cm, filename, 'gif', 'WriteMode', 'append', 'DelayTime', 0.05);
    end
end
```



Visualize Volume of MRI Data

Load MRI data and remove the singleton dimension.

```
load mri
V = squeeze(D);
```

Generate a colormap and transparency (alpha) map suited for MRI images.

```
intensity = [0 20 40 120 220 1024];
alpha = [0 0 0.15 0.3 0.38 0.5];
color = ([0 0 0; 43 0 0; 103 37 20; 199 155 97; 216 213 201; 255 255 255]) ./ 255;
queryPoints = linspace(min(intensity),max(intensity),256);
alphamap = interp1(intensity,alpha,queryPoints)';
colormap = interp1(intensity,color,queryPoints);
```

View the volume with the custom colormap and transparency map. Click and drag the mouse to rotate the volume. Use the scroll wheel to zoom in and out of the volume.

```
vol = volshow(V,'Colormap',colormap,'Alphamap',alphamap);
```

Visualize Volume of CT Data

This example uses 3-D volumetric human chest CT scan data. To run this example, you must download the sample data from MathWorks™ using the Add-On Explorer. See “Install Sample Data Using Add-On Explorer”.

Load the data into the workspace.

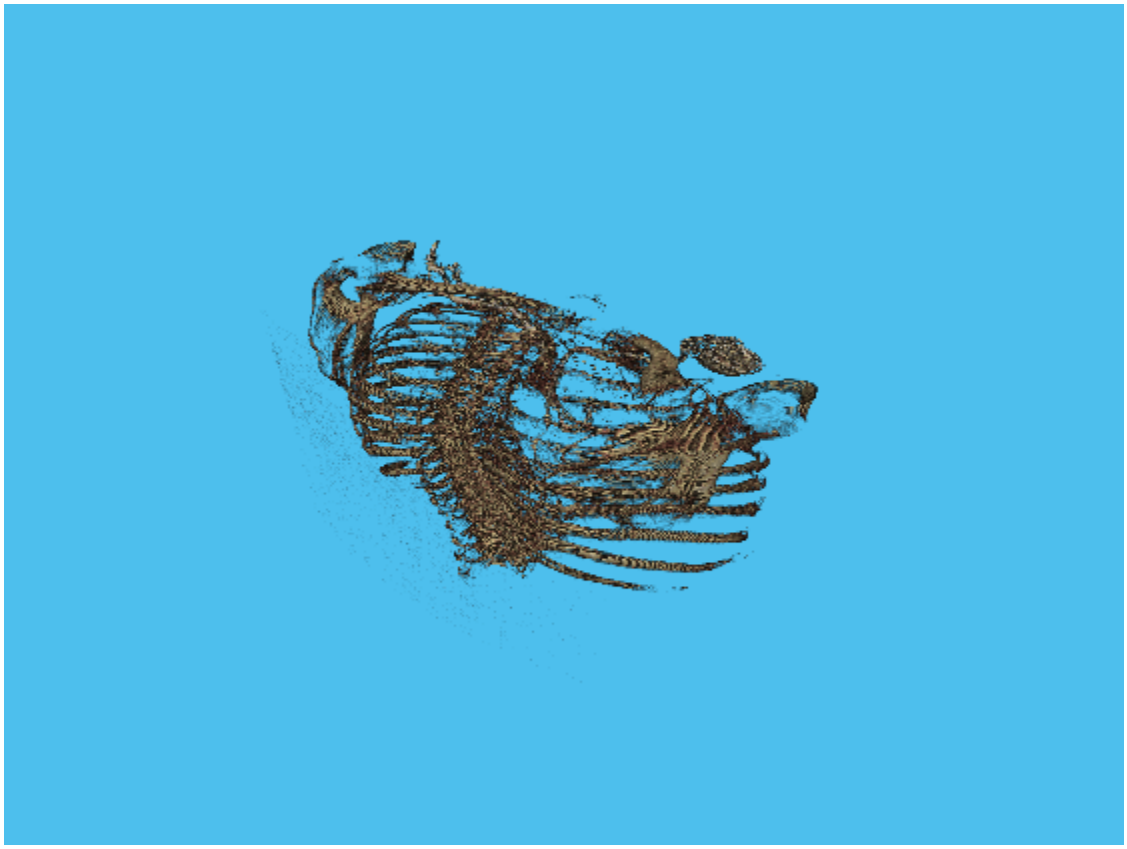
```
load chestVolume
```

Create a colormap and transparency map suited for CT images.

```
intensity = [-3024,-16.45,641.38,3071];
alpha = [0, 0, 0.72, 0.72];
color = ([0 0 0; 186 65 77; 231 208 141; 255 255 255]) ./ 255;
queryPoints = linspace(min(intensity),max(intensity),256);
alphamap = interp1(intensity,alpha,queryPoints)';
colormap = interp1(intensity,color,queryPoints);
```

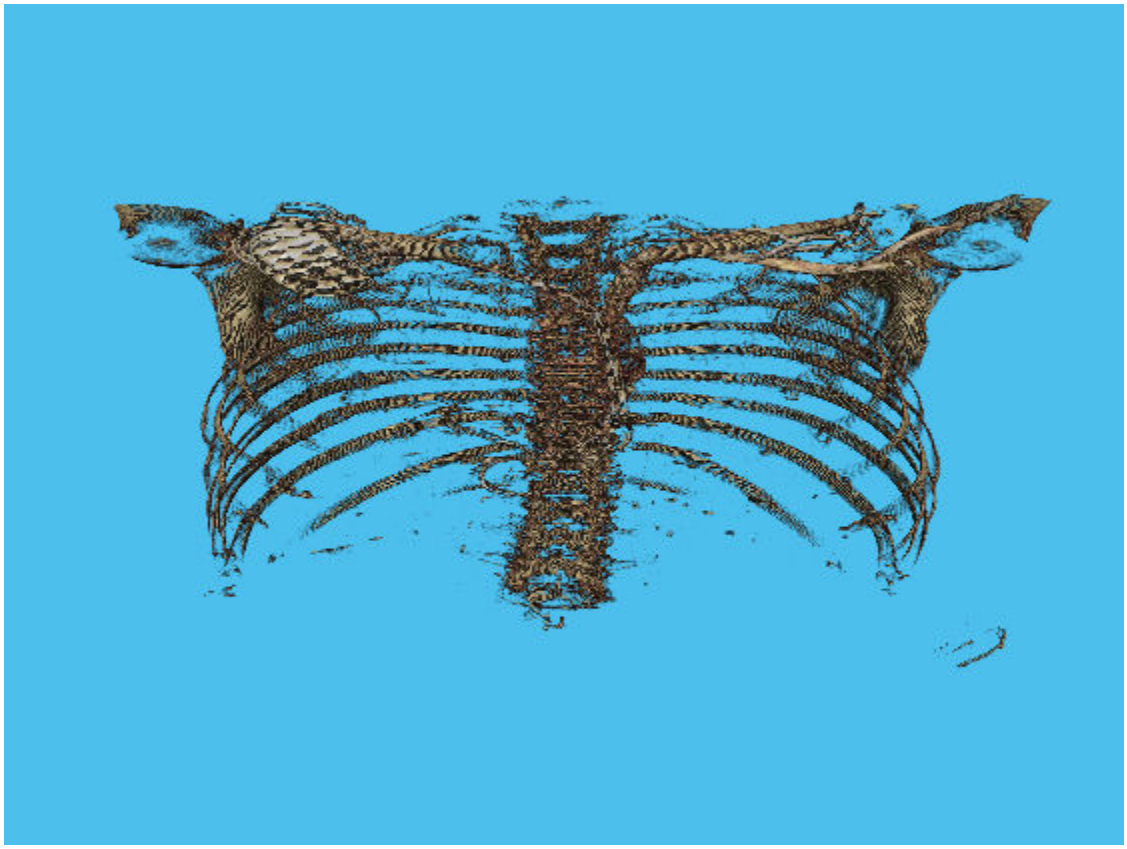
View the volume with the custom colormap and transparency map. Click and drag the mouse to rotate the volume. Use the scroll wheel to zoom in and out of the volume.

```
vol = volshow(V,'Colormap',colormap,'Alphamap',alphamap);
```



Programatically change the camera position to see a different view of the volume.

```
vol.CameraPosition = [-1.0533 -0.0093, 0.1593];
```



Tips

- The `volshow` function creates a `uipanel` object in the specified parent figure. Panels are containers that group UI components together. `volshow` displays volumetric data in the `uipanel`. In contrast, `imshow` displays images in an `Axes`. If you call `imshow` to display an image in a figure in which `volshow` has displayed a volume, then `imshow` does not overwrite the volume displayed by `volshow`. The `Axes` created by `imshow` displays behind the `uipanel`.

See Also

Volume Viewer | `labelvolshow` | `isosurface` | `slice` | `obliqueslice`

Introduced in R2018b

setVolume

Set new volume

Syntax

```
setVolume(hVol,V)
```

Description

`setVolume(hVol,V)` updates the `volshow` object `hVol` with a new volume `V`. `setVolume` preserves the current viewpoint and other visualization settings remain unchanged.

Examples

Change the Volume in volshow Object

Load two volumes.

```
load mri
V = squeeze(D);

load spiralVol
```

Display one of the volumes, using `volshow`.

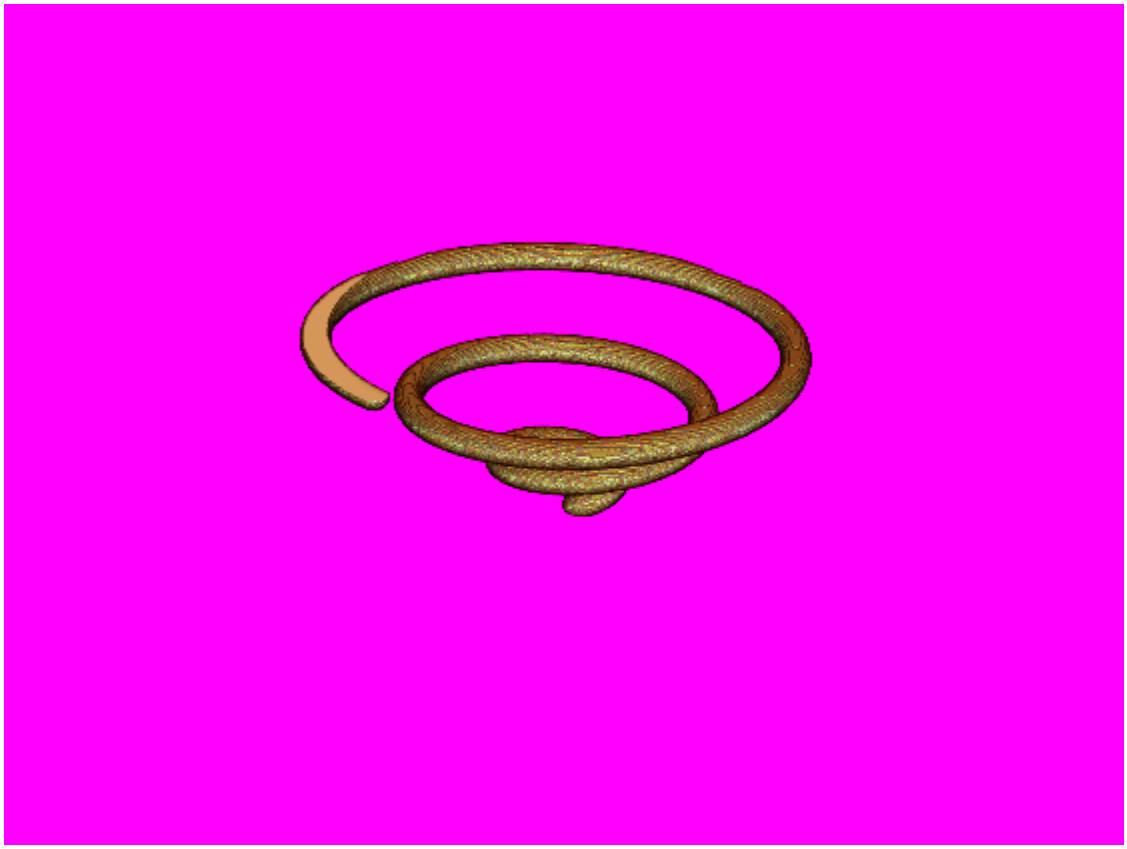
```
intensity = [0 20 40 120 220 1024];
alpha = [0 0 0.15 0.3 0.38 0.5];
color = ([0 0 0; 43 0 0; 103 37 20; 199 155 97; 216 213 201; 255 255 255]) ./ 255;
queryPoints = linspace(min(intensity),max(intensity),256);
alphamap = interp1(intensity,alpha,queryPoints)';
colormap = interp1(intensity,color,queryPoints);
vol = volshow(V,'Colormap',colormap,'Alphamap',alphamap);
```

Change rendering settings.

```
newColormap = hot(256);
vol.Colormap = newColormap;
vol.BackgroundColor = 'magenta';
```

Change the volume in the `volshow` object. Note that, when changing the volume, the `volshow` object preserves your rendering settings.

```
setVolume(vol,spiralVol)
```



Input Arguments

`hVol` — Volume visualization

`volshow` object

Volume visualization, specified as a `volshow` object.

`V` — Volumetric data

3-D grayscale volume

Volumetric data, specified as a 3-D grayscale volume.

See Also

`volshow`

Introduced in R2019a

warp

Display image as texture-mapped surface

Syntax

```
warp(X,map)
warp(I,n)
warp(BW)
warp(RGB)
warp(Z,___)
warp(X,Y,Z,___)
h = warp(___)
```

Description

`warp(X,map)` displays the indexed image `X` with colormap `map` as a texture map on a simple rectangular surface.

`warp(I,n)` displays the intensity image `I` with `n` levels as a texture map on a simple rectangular surface.

`warp(BW)` displays the binary image `BW` as a texture map on a simple rectangular surface.

`warp(RGB)` displays the truecolor image `RGB` as a texture map on a simple rectangular surface.

`warp(Z,___)` displays the image on the surface `Z`.

`warp(X,Y,Z,___)` displays the image on the surface `(X,Y,Z)`.

`h = warp(___)` returns the texture-mapped surface.

Examples

Warp Indexed Image over Curved Surface

This example shows how to warp an indexed image over a nonuniform surface. This example uses a curved surface centered at the origin.

Read an indexed image into the workspace.

```
[I,map] = imread('forest.tif');
```

Create the surface. First, define the *x*- and *y*-coordinates of the surface. This example uses arbitrary coordinates that are unrelated to the indexed image. Note that the size of the coordinate matrices `X` and `Y` do not need to match the size of the image.

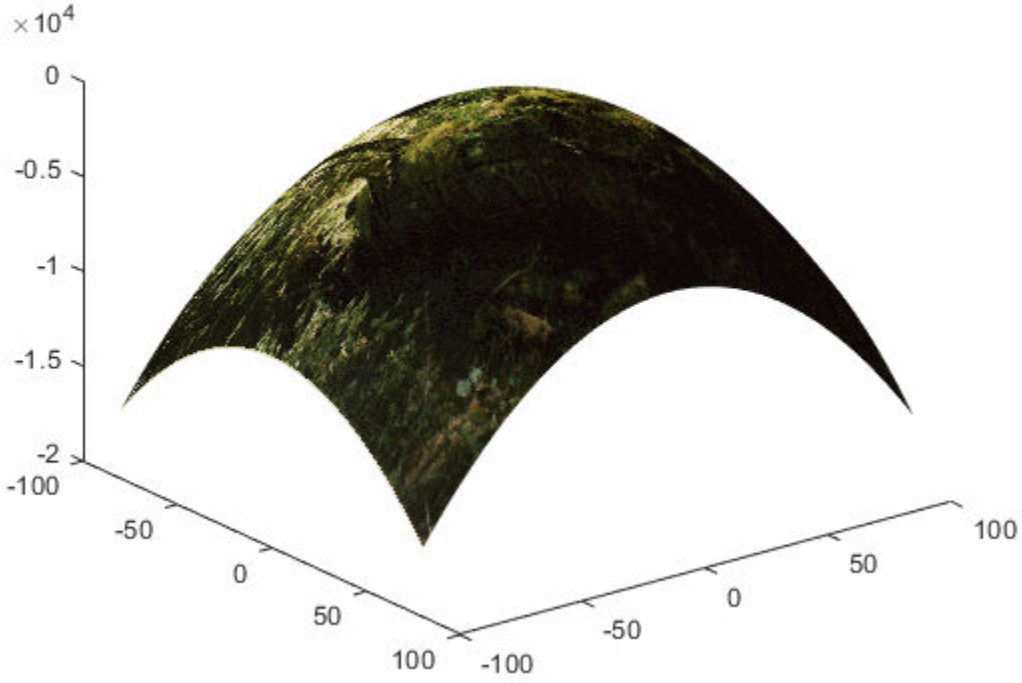
```
[X,Y] = meshgrid(-100:100,-80:80);
```

Define the height `Z` of the surface at the coordinates given by `(X,Y)`.

```
Z = -(X.^2 + Y.^2);
```

Warp the image over the surface defined by the coordinates (`X,Y,Z`).

```
figure
warp(X,Y,Z,I,map);
```



Explore the warped image interactively using the rotate and data cursor tools.

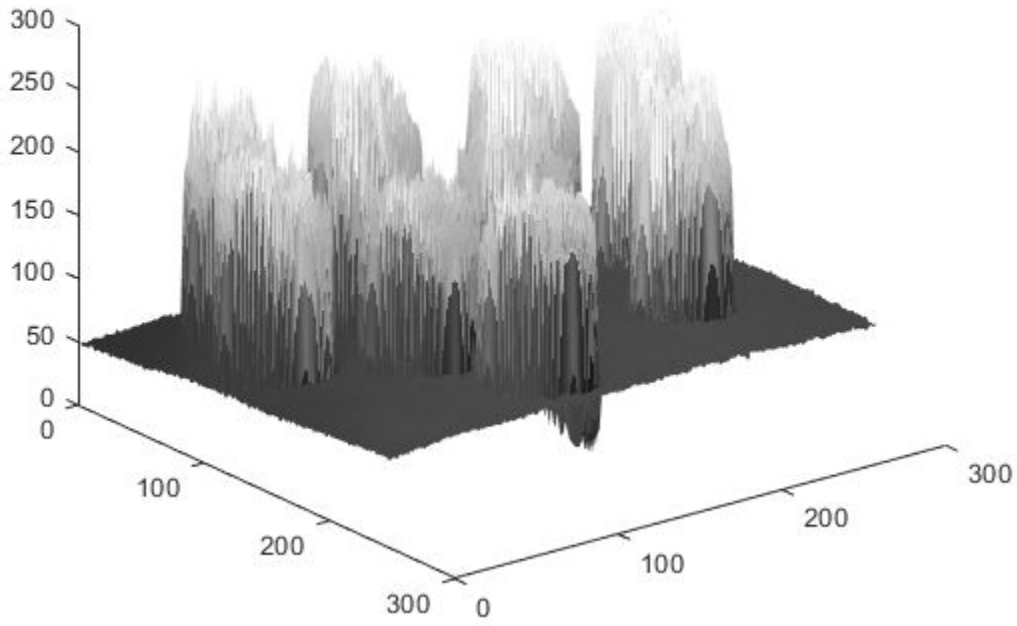
Warp Grayscale Image Based on Intensity

Read a grayscale image into the workspace.

```
I = imread('coins.png');
```

Warp the image over the surface whose height is equal to the intensity of the image `I`. Specify the number of graylevels.

```
figure
warp(I,I,128);
```



Note that the *x*- and *y*-coordinates of the surface were not specified in the call to `warp` and thus default to the image pixel indices. Explore the warped image interactively using the rotate and data cursor tools.

Input Arguments

`X` — Indexed image

2-D numeric matrix

Indexed image, specified as a 2-D numeric matrix. The values in `X` are an index into `map`, an *n*-by-3 array of RGB values.

Data Types: `single` | `double` | `uint8` | `uint16` | `int16` | `logical`

`map` — Colormap

n-by-3 numeric matrix

Colormap, specified as an *n*-by-3 numeric matrix. Each row specifies an RGB color value. When `map` is type `single` or `double`, values must be in the range [0, 1].

Data Types: `single` | `double` | `uint8`

`I` — Grayscale image

2-D numeric matrix

Grayscale image, specified as a 2-D numeric matrix.

Data Types: `single` | `double` | `uint8` | `uint16` | `int16` | `logical`

`n` — Number of grayscale levels
positive integer

Number of grayscale levels, specified as a positive integer.

Data Types: `double` | `uint8` | `uint16` | `logical`

`BW` — Binary image
2-D logical matrix

Binary image, specified as a 2-D logical matrix.

Data Types: `logical`

`RGB` — Truecolor image
m-by-*n*-by-3 numeric array

Truecolor image, specified as an *m*-by-*n*-by-3 numeric array.

Data Types: `single` | `double` | `uint8` | `uint16` | `int16` | `logical`

`Z` — Height of surface
2-D numeric matrix

Height of surface, specified as a 2-D numeric matrix. When `Z` is not specified, the surface is flat with a uniform height of 0.

Data Types: `single` | `double` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64` | `logical`

`X` — *x*-coordinates of surface
2-D numeric matrix

x-coordinates of surface, specified as a 2-D numeric matrix.

Data Types: `single` | `double` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64` | `logical`

`Y` — *y*-coordinates of surface
2-D numeric matrix

y-coordinates of surface, specified as a 2-D numeric matrix.

Data Types: `single` | `double` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64` | `logical`

Output Arguments

`h` — Texture-mapped surface
`Surface` object

Texture-mapped surface, returned as a `Surface` object. For more information, see Surface Properties.

Tips

- Texture-mapped surfaces are generally rendered more slowly than images.
- The `warp` function sets the `YDir` axes property to `"reverse"`. Values along the *y*-axis increase from top to bottom. To decrease the values from top to bottom, set `YDir` to `"normal"`. This setting reverses both the *y*-axis and the image.

See Also

`imshow` | `image` | `imagesc` | `surf`

Introduced before R2006a

Warper

Apply same geometric transformation to many images efficiently

Description

A `Warper` object applies an `affine2d` or `projective2d` geometric transformation to images with a specific size.

Creation

Syntax

```
w = images.geotrans.Warper(tform,inputSize)
w = images.geotrans.Warper(tform,inputRef)
w = images.geotrans.Warper(tform,inputRef,outputRef)
w = images.geotrans.Warper(sourceX,sourceY)
w = images.geotrans.Warper( ___ ,Name,Value)
```

Description

`w = images.geotrans.Warper(tform,inputSize)` creates an image warper from the geometric transformation object `tform` and sets the InputSize property.

`w = images.geotrans.Warper(tform,inputRef)` specifies the coordinate system of the input images, `inputRef`.

`w = images.geotrans.Warper(tform,inputRef,outputRef)` specifies the coordinate system of the output image, `outputRef`. This syntax can be used to improve performance by limiting the application of the geometric transformation to a specific output region of interest.

`w = images.geotrans.Warper(sourceX,sourceY)` specifies the input image coordinates, `sourceX` and `sourceY`, required to perform the geometric transformation.

`w = images.geotrans.Warper( ___ ,Name,Value)` sets the Interpolation and FillValue properties using one or more name-value pair arguments. Enclose each property name in single quotes.

For example, `warper = images.geotrans.Warper(tform,size(im),'FillValue',1)` specifies a fill value of `1` for pixels outside the original image.

Input Arguments

`tform` — Geometric transformation

`affine2d` object | `projective2d` object

Geometric transformation, specified as an `affine2d` or `projective2d` geometric transformation object.

`inputRef` — Referencing object associated with input image
`imref2d` object

Referencing object associated with the input image, specified as an `imref2d` spatial referencing object.

`outputRef` — Referencing object associated with output image
`imref2d` object

Referencing object associated with the output image, specified as an `imref2d` spatial referencing object.

`sourceX`, `sourceY` — Input image coordinates
2-D matrix

Input image coordinates, specified as a 2-D matrix the same size as the required output image. Each (*x*, *y*) index in `sourceX` and `sourceY` specifies the location in the input image for the corresponding output pixel.

Data Types: `single`

Properties

`InputSize` — Size of the input images
2-element vector of positive integers | 3-element vector of positive integers

Size of the input images, specified as a 2- or 3-element vector of positive integers.

`OutputSize` — Size of the first two dimensions of the output image
2-element vector of positive integers

Size of the first two dimensions of the output image, specified as a 2-element vector of positive integers.

`Interpolation` — Interpolation method
`'linear'` (default) | `'nearest'` | `'cubic'`

Interpolation method, specified as `'linear'`, `'nearest'`, or `'cubic'`.

Data Types: `char` | `string`

`FillValue` — Value used for output pixels outside the input image boundaries
`0` (default) | numeric scalar

Value used for output pixels outside the input image boundaries, specified as a numeric scalar. `Warper` casts the fill value to the data type of the input image.

Object Functions

warp Apply geometric transformation

Examples

Apply Shear to Multiple Images

Pick a set of images of the same size. The example uses a set of images that show cells.

```
imds = imageDatastore(fullfile(matlabroot,'toolbox','images','imdata','AT*'));
```

Create a geometric transform to rotate each image by 45 degrees and to shrink each image.

```
tform = affine2d([ 0.5*cos(pi/4) sin(pi/4)     0;
                  -sin(pi/4)     0.5*cos(pi/4) 0;
                   0             0             1]);
```

Create a `Warper` object, specifying the geometric transformation object, `tform`, and the size of the input images.

```
im = readimage(imds,1);
warper = images.geotrans.Warper(tform,size(im));
```

Determine the number of images to be processed and preallocate the output array.

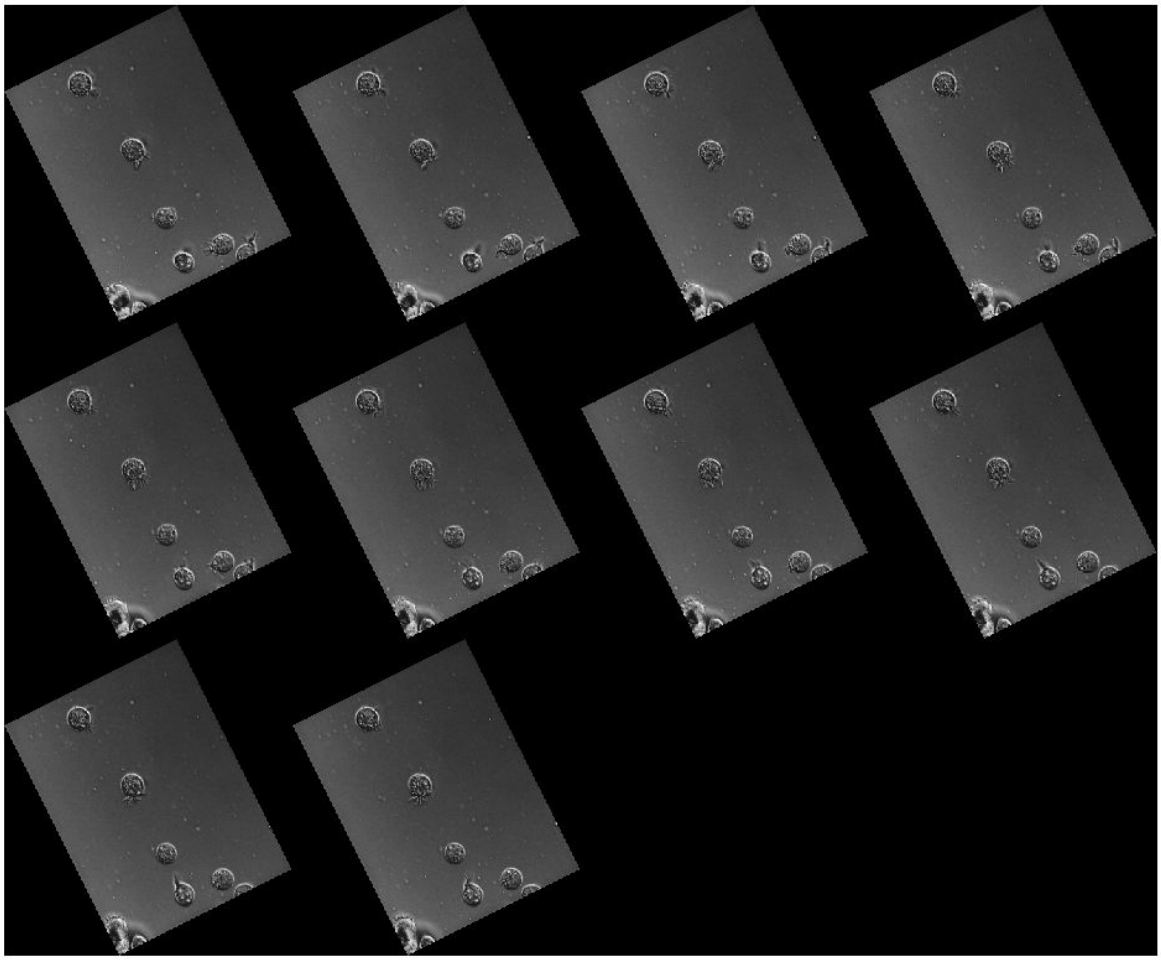
```
numFiles = numel(imds.Files);
imr = zeros([warper.OutputSize 1 numFiles],'like',im);
```

Apply the geometric transformation to each of the input images by calling the `warp` function of the `Warper` object.

```
for ind = 1:numFiles
    im = read(imds);
    imr(:,:,1,ind) = warp(warper,im);
end
```

Visualize the output images. (Turn off the warning message about the images being scaled for display.)

```
warning('off','images:initSize:adjustingMag')
montage(imr);
```



Tips

- If the input images are RGB images or 3-D grayscale images of size *m*-by-*n*-by-*p*, then `warp` applies the transformation to each color channel or plane *p* independently.

Algorithms

`Warper` is optimized to apply the same geometric transformation across a batch of same size images. `Warper` achieves this optimization by splitting the warping process into two steps: computation of the transformed coordinates (done once) and interpolation on the image (done for each image). Compared to `imwarp`, this approach speeds up the whole process significantly for small to medium-sized images, with diminishing returns for larger images.

See Also

Functions
`warp` | `imtranslate` | `imwarp` | `imrotate`

Objects
`affine2d` | `projective2d` | `imref2d`

Introduced in R2017b

warp

Package: `images.geotrans`

Apply geometric transformation

Syntax

```
B = warp(w,A)
```

Description

`B = warp(w,A)` performs the geometrical transformation defined in `w` on input image `A` and returns the warped image in `B`.

Input Arguments

`w` — Image warper

`Warper` object

Image warper, specified as a `Warper` object.

`A` — Input image

numeric matrix

Input image, specified as a numeric matrix, with size *m*-by-*n* or *m*-by-*n*-by-*p*. The size of `A` must match `w.InputSize`.

Data Types: `single` | `int16` | `uint8`

Output Arguments

`B` — Transformed image

numeric matrix

Transformed image, returned as a numeric matrix. `B` has the same type as `A` and its first two dimensions are `w.OutputSize`. If `A` has *p* planes, `B` will also have *p* planes.

See Also

`Warper`

Introduced in R2017b

watershed

Watershed transform

Syntax

```
L = watershed(A)
L = watershed(A,conn)
```

Description

The watershed transform finds "catchment basins" or "watershed ridge lines" in an image by treating it as a surface where light pixels represent high elevations and dark pixels represent low elevations. The watershed transform can be used to segment contiguous regions of interest into distinct objects.

`L = watershed(A)` returns a label matrix `L` that identifies the watershed regions of the input matrix `A`.

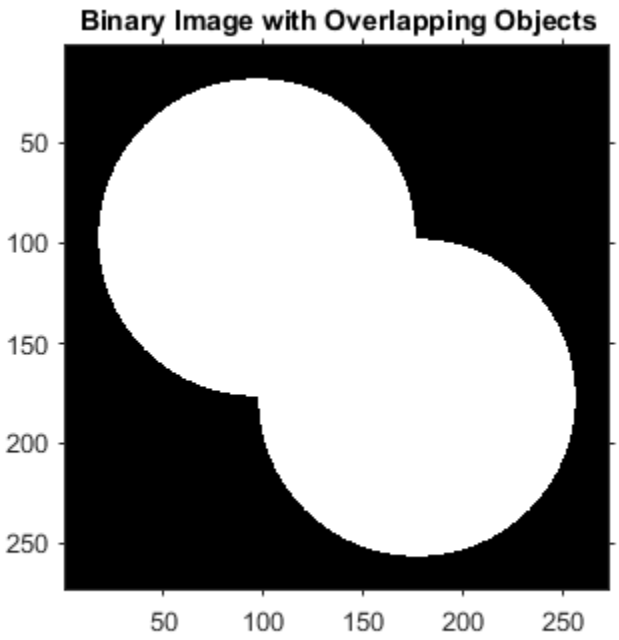
`L = watershed(A,conn)` specifies the connectivity to be used in the watershed computation.

Examples

Compute Watershed Transform and Display Resulting Label Matrix

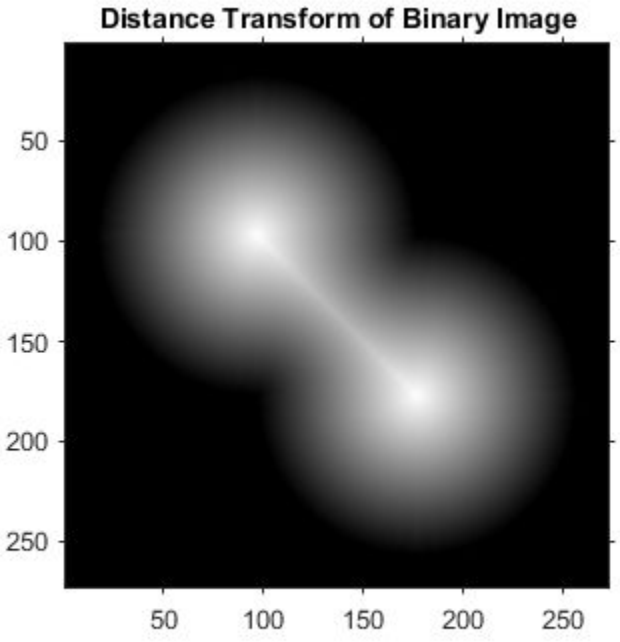
Create a binary image containing two overlapping circular objects. Display the image.

```
center1 = -40;
center2 = -center1;
dist = sqrt(2*(2*center1)^2);
radius = dist/2 * 1.4;
lims = [floor(center1-1.2*radius) ceil(center2+1.2*radius)];
[x,y] = meshgrid(lims(1):lims(2));
bw1 = sqrt((x-center1).^2 + (y-center1).^2) <= radius;
bw2 = sqrt((x-center2).^2 + (y-center2).^2) <= radius;
bw = bw1 | bw2;
imshow(bw)
title('Binary Image with Overlapping Objects')
```



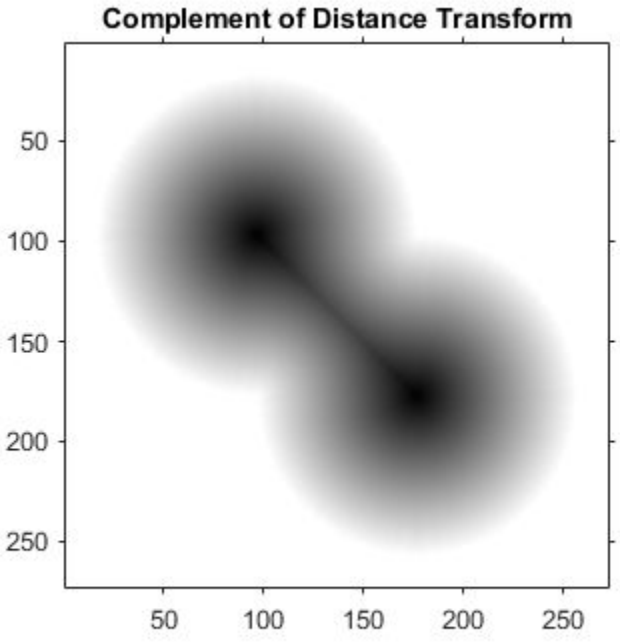
Calculate the distance transform of the complement of the binary image. The value of each pixel in the output image is the distance between that pixel and the nearest nonzero pixel of bw.

```
D = bwdist(~bw);
imshow(D,[])
title('Distance Transform of Binary Image')
```



Take the complement of the distance transformed image so that light pixels represent high elevations and dark pixels represent low elevations for the watershed transform.

```
D = -D;
imshow(D,[])
title('Complement of Distance Transform')
```

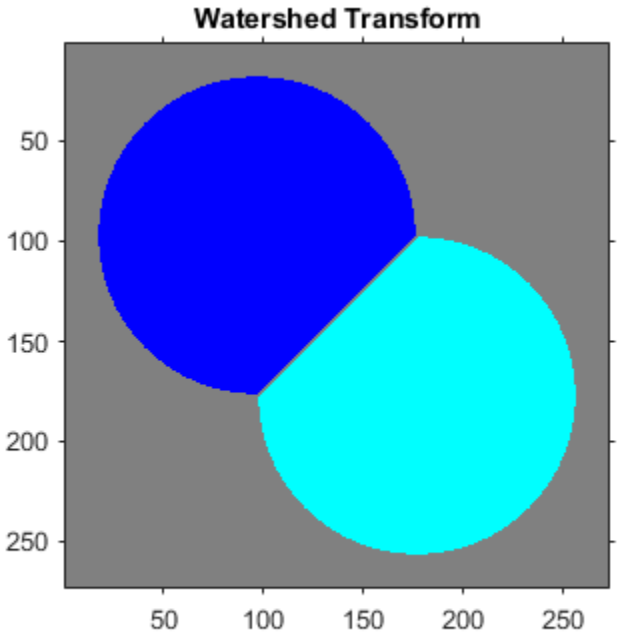


Calculate the watershed transform. Set pixels that are outside the ROI to `0`.

```
L = watershed(D);
L(~bw) = 0;
```

Display the resulting label matrix as an RGB image.

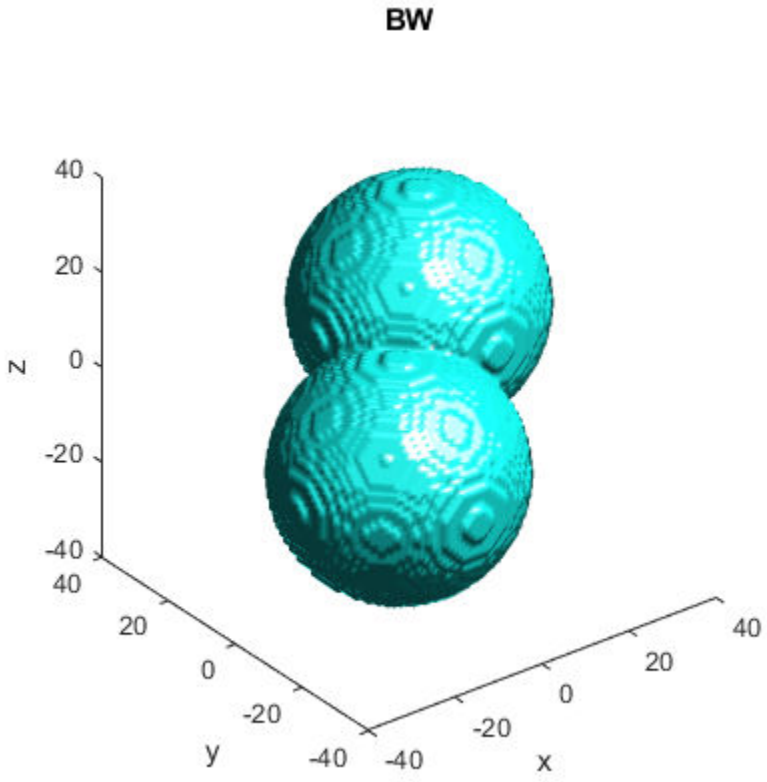
```
rgb = label2rgb(L,'jet',[.5 .5 .5]);
imshow(rgb)
title('Watershed Transform')
```



Compute Watershed Transform of 3-D Binary Image

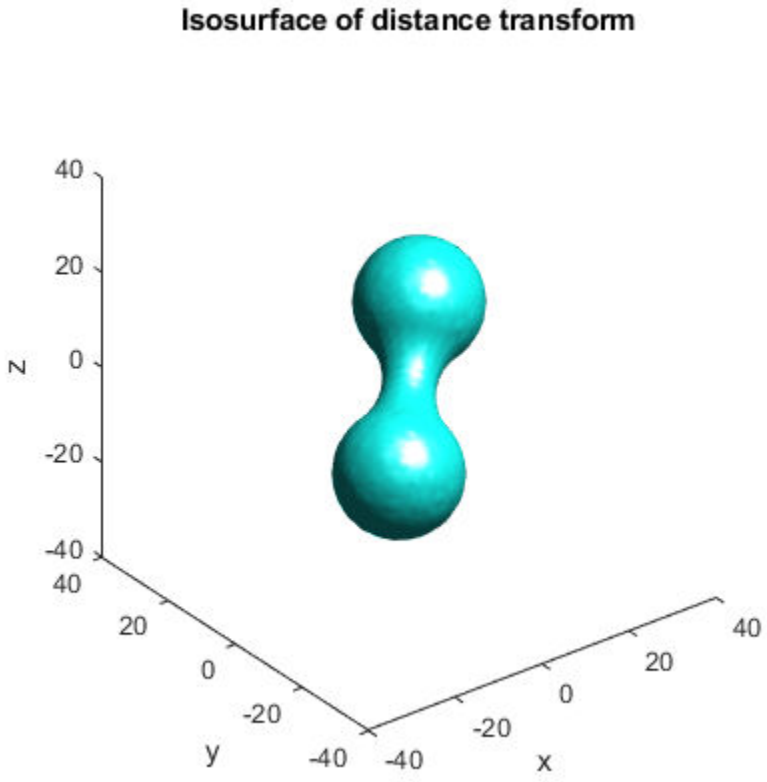
Make a 3-D binary image containing two overlapping spheres.

```
center1 = -10;
center2 = -center1;
dist = sqrt(3*(2*center1)^2);
radius = dist/2 * 1.4;
lims = [floor(center1-1.2*radius) ceil(center2+1.2*radius)];
[x,y,z] = meshgrid(lims(1):lims(2));
bw1 = sqrt((x-center1).^2 + (y-center1).^2 + ...
           (z-center1).^2) <= radius;
bw2 = sqrt((x-center2).^2 + (y-center2).^2 + ...
           (z-center2).^2) <= radius;
bw = bw1 | bw2;
figure, isosurface(x,y,z,bw,0.5), axis equal, title('BW')
xlabel x, ylabel y, zlabel z
xlim(lims), ylim(lims), zlim(lims)
view(3), camlight, lighting gouraud
```



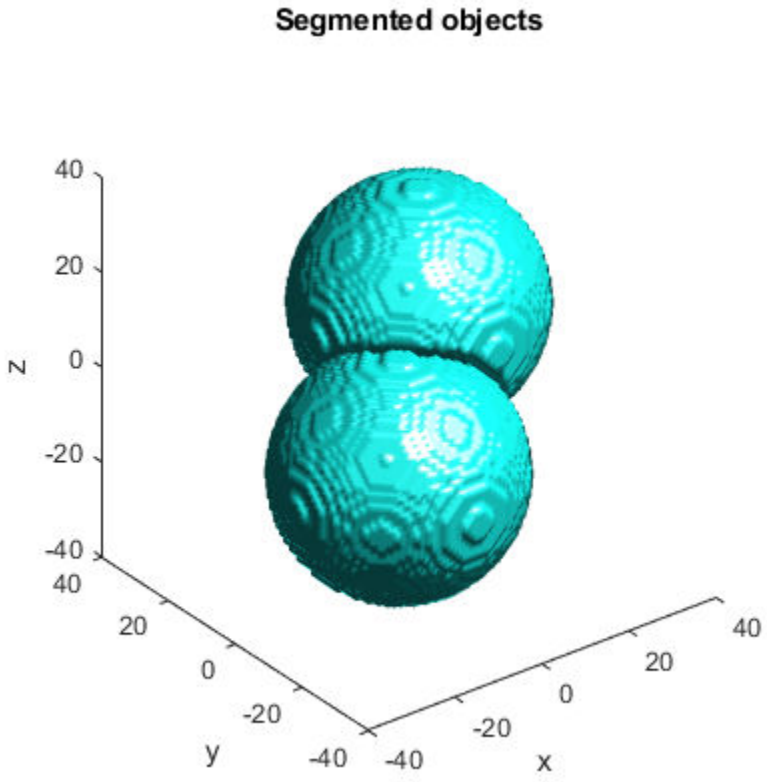
Compute the distance transform.

```
D = bwdist(~bw);
figure, isosurface(x,y,z,D,radius/2), axis equal
title('Isosurface of distance transform')
xlabel x, ylabel y, zlabel z
xlim(lims), ylim(lims), zlim(lims)
view(3), camlight, lighting gouraud
```



Complement the distance transform, force nonobject pixels to be `Inf`, and then compute the watershed transform.

```
D = -D;
D(~bw) = Inf;
L = watershed(D);
L(~bw) = 0;
figure
isosurface(x,y,z,L==1,0.5)
isosurface(x,y,z,L==2,0.5)
axis equal
title('Segmented objects')
xlabel x, ylabel y, zlabel z
xlim(lims), ylim(lims), zlim(lims)
view(3), camlight, lighting gouraud
```



Input Arguments

A — Input image

numeric array | logical array

Input image, specified as a numeric or logical array of any dimension.

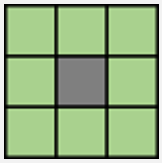
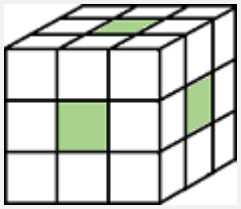
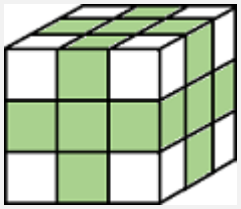
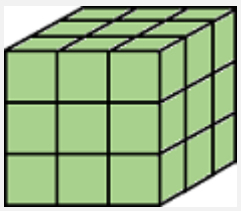
Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

conn — Pixel connectivity

4 | 8 | 6 | 18 | 26 | 3-by-3-by- ... -by-3 matrix of `0`s and `1`s

Pixel connectivity, specified as one of the values in this table. The default connectivity is `8` for 2-D images, and `26` for 3-D images.

Value	Meaning
Two-Dimensional Connectivities	

Value	Meaning	
4	Pixels are connected if their edges touch. The neighborhood of a pixel are the adjacent pixels in the horizontal or vertical direction.	 Current pixel is shown in gray.
8	Pixels are connected if their edges or corners touch. The neighborhood of a pixel are the adjacent pixels in the horizontal, vertical, or diagonal direction.	 Current pixel is shown in gray.
Three-Dimensional Connectivities		
6	Pixels are connected if their faces touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down	 Current pixel is shown in gray.
18	Pixels are connected if their faces or edges touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up	 Current pixel is center of cube.
26	Pixels are connected if their faces, edges, or corners touch. The neighborhood of a pixel are the adjacent pixels in: • One of these directions: in, out, left, right, up, and down • A combination of two directions, such as right-down or in-up • A combination of three directions, such as in-right-up or in-left-down	 Current pixel is center of cube.

For higher dimensions, `watershed` uses the default value `conndef(ndims(A),'maximal')`.

Connectivity can also be defined in a more general way for any dimension by specifying a 3-by-3-by- ... -by-3 matrix of `0`s and `1`s. The `1`-valued elements define neighborhood locations relative to the

center element of `conn`. Note that `conn` must be symmetric about its center element. See “Specifying Custom Connectivities” for more information.

Note If you specify a nondefault connectivity, pixels on the edge of the image might not be considered to be border pixels. For example, if `conn = [0 0 0; 1 1 1; 0 0 0]`, elements on the first and last row are not considered to be border pixels because, according to that connectivity definition, they are not connected to the region outside the image.

Data Types: `double` | `logical`

Output Arguments

`L` — Label matrix

numeric array of nonnegative integers

Label matrix, specified as a numeric array of nonnegative integers. The elements labeled `0` do not belong to a unique watershed region. The elements labeled `1` belong to the first watershed region, the elements labeled 2 belong to the second watershed region, and so on.

Tips

- The watershed transform algorithm used by this function changed in version 5.4 (R2007a) of the Image Processing Toolbox software. The previous algorithm occasionally produced labeled watershed basins that were not contiguous. If you need to obtain the same results as the previous algorithm, use the function `watershed_old`.
- To prevent oversegmentation, remove shallow minima from the image by using the `imhmin` function before you use the `watershed` function.

Algorithms

`watershed` uses the Fernand Meyer algorithm [1].

References

[1] Meyer, Fernand, "Topographic distance and watershed lines,” *Signal Processing* , Vol. 38, July 1994, pp. 113-125.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `watershed` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `watershed` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

- Supports only 2-D images
- Supports only 4 or 8 connectivity
- Supports images containing up to 65,535 regions
- Output type is always `uint16`

GPU Code Generation
Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

Usage notes and limitations:

- Supports only 2-D images
- Supports only 4 or 8 connectivity
- Supports images containing up to 65,535 regions
- Output type is always `uint16`

See Also

`bwlabel` | `bwlabeln` | `bwdist` | `regionprops` | `imhmin`

Introduced before R2006a

whitepoint

XYZ color values of standard illuminants

Syntax

```
xyz = whitepoint
xyz = whitepoint(illuminant)
```

Description

`xyz = whitepoint` returns the XYZ value corresponding to the default ICC white reference illuminant, scaled so that Y = 1.

`xyz = whitepoint(illuminant)` returns the XYZ value corresponding to the white reference illuminant, `illuminant`, scaled so that Y = 1.

Examples

Get XYZ Value of ICC Illuminant

Return the XYZ color space representation of the default white reference illuminant, `'icc'`.

```
wp_icc = whitepoint
```

```
wp_icc =

    0.9642    1.0000    0.8249
```

Note that the second element, corresponding to the Y value, is 1.

Get XYZ Value of d65 Illuminant

Return the XYZ color space representation of the `'d65'` white reference illuminant.

```
wp_d65 = whitepoint('d65')
```

```
wp_d65 =

    0.9504    1.0000    1.0888
```

Input Arguments

`illuminant` — White reference illuminant

`'icc'` (default) | `'a'` | `'c'` | `'e'` | `'d50'` | `'d55'` | `'d65'`

White reference illuminant, specified as one of these values.

Value	White Point
`'a'`	CIE standard illuminant A, `[1.0985, 1.0000, 0.3558]`. Simulates typical, domestic, tungsten-filament lighting with correlated color temperature of 2856 K.

Value	White Point
`'c'`	CIE standard illuminant C, `[0.9807, 1.0000, 1.1822]`. Simulates average or north sky daylight with correlated color temperature of 6774 K. Deprecated by CIE.
`'e'`	Equal-energy radiator, `[1.000, 1.000, 1.000]`. Useful as a theoretical reference.
`'d50'`	CIE standard illuminant D50, `[0.9642, 1.0000, 0.8251]`. Simulates warm daylight at sunrise or sunset with correlated color temperature of 5003 K. Also known as horizon light.
`'d55'`	CIE standard illuminant D55, `[0.9568, 1.0000, 0.9214]`. Simulates mid-morning or mid-afternoon daylight with correlated color temperature of 5500 K.
`'d65'`	CIE standard illuminant D65, `[0.9504, 1.0000, 1.0888]`. Simulates noon daylight with correlated color temperature of 6504 K.
`'icc'`	Profile Connection Space (PCS) illuminant used in ICC profiles. Approximation of `[0.9642, 1.000, 0.8249]` using fixed-point, signed, 32-bit numbers with 16 fractional bits. Actual value: `[31595,32768, 27030]/32768`.

Data Types: `char` | `string`

Output Arguments

xyz — XYZ values

3-element numeric row vector

XYZ values corresponding to the illuminant, returned as a 3-element numeric row vector. The values are scaled so that Y = 1.

Data Types: `double`

See Also

`applycform` | `makecform` | `xyz2double` | `xyz2uint16` | `xyz2lab` | `xyz2rgb`

Introduced before R2006a

wiener2

2-D adaptive noise-removal filtering

Note The syntax `wiener2(I,[m n],[mblock nblock],noise)` has been removed. Use the `wiener2(I,[m n],noise)` syntax instead.

Syntax

```
J = wiener2(I,[m n],noise)
[J,noise_out] = wiener2(I,[m n])
```

Description

`J = wiener2(I,[m n],noise)` filters the grayscale image `I` using a pixel-wise adaptive low-pass Wiener filter. `[m n]` specifies the size (`m`-by-`n`) of the neighborhood used to estimate the local image mean and standard deviation. The additive noise (Gaussian white noise) power is assumed to be `noise`.

The input image has been degraded by constant power additive noise. `wiener2` uses a pixelwise adaptive Wiener method based on statistics estimated from a local neighborhood of each pixel.

`[J,noise_out] = wiener2(I,[m n])` returns the estimates of the additive noise power `wiener2` calculates before doing the filtering.

Examples

Remove Noise By Adaptive Filtering

This example shows how to use the `wiener2` function to apply a Wiener filter (a type of linear filter) to an image adaptively. The Wiener filter tailors itself to the local image variance. Where the variance is large, `wiener2` performs little smoothing. Where the variance is small, `wiener2` performs more smoothing.

This approach often produces better results than linear filtering. The adaptive filter is more selective than a comparable linear filter, preserving edges and other high-frequency parts of an image. In addition, there are no design tasks; the `wiener2` function handles all preliminary computations and implements the filter for an input image. `wiener2`, however, does require more computation time than linear filtering.

`wiener2` works best when the noise is constant-power ("white") additive noise, such as Gaussian noise. The example below applies `wiener2` to an image of Saturn with added Gaussian noise.

Read the image into the workspace.

```
RGB = imread('saturn.png');
```

Convert the image from truecolor to grayscale.

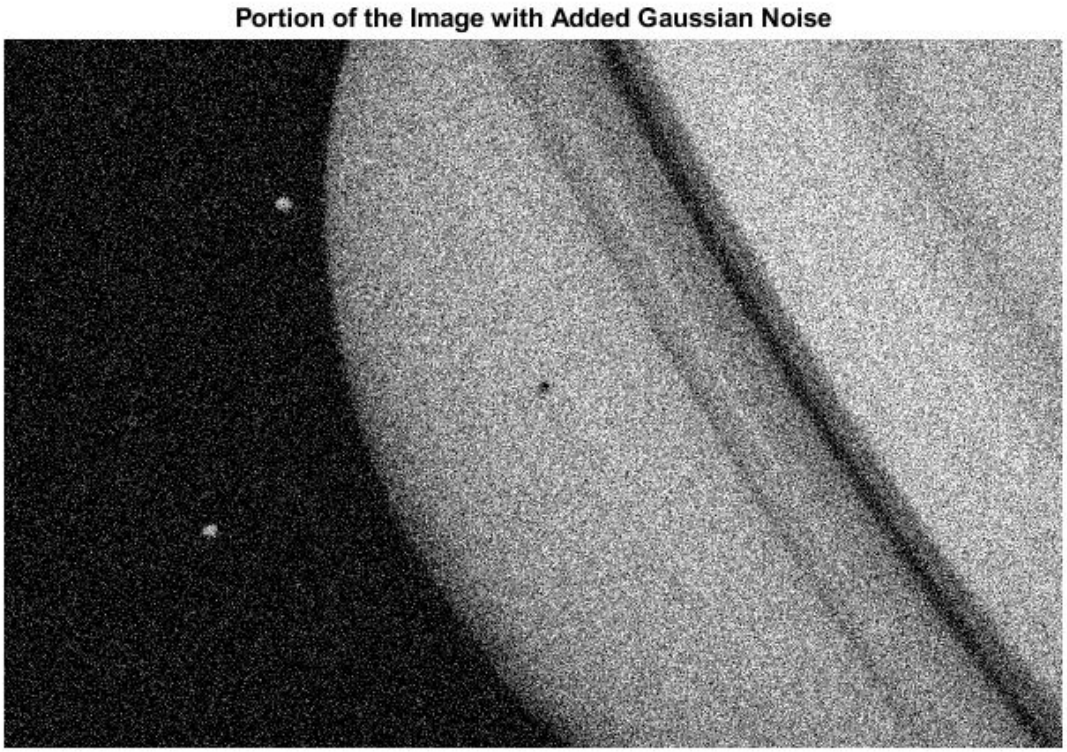
```
I = im2gray(RGB);
```

Add Gaussian noise to the image

```
J = imnoise(I,'gaussian',0,0.025);
```

Display the noisy image. Because the image is quite large, display only a portion of the image.

```
imshow(J(600:1000,1:600));
title('Portion of the Image with Added Gaussian Noise');
```

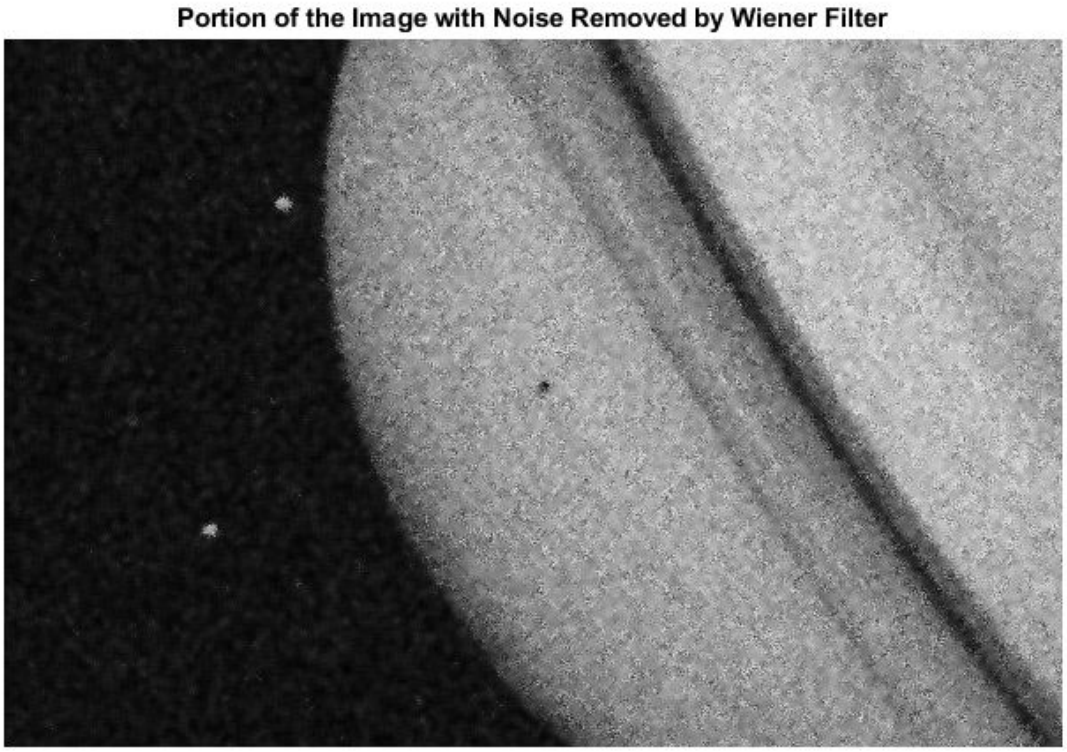


Remove the noise using the `wiener2` function.

```
K = wiener2(J,[5 5]);
```

Display the processed image. Because the image is quite large, display only a portion of the image.

```
figure
imshow(K(600:1000,1:600));
title('Portion of the Image with Noise Removed by Wiener Filter');
```



Input Arguments

`I` — Input image

2-D numeric array

Input image, specified as a 2-D numeric array.

Data Types: `single` | `double` | `int16` | `uint8` | `uint16`

`[m n]` — Neighborhood size

`[3 3]` (default) | 2-element numeric vector

Neighborhood size, specified as a 2-element vector of the form `[m n]` where `m` is the number of rows and `n` is the number of columns.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`noise` — Additive noise

numeric array

Additive noise, specified as a numeric array. If you do not specify noise, then `wiener2` uses the mean of the local variance, `mean2(localVar)`.

Data Types: `single` | `double`

Output Arguments

J — Filtered image

numeric array

Filtered image, returned as a numeric array of the same size and data type as the input image `I`.

noise_out — Estimate of additive noise power

numeric array

Estimate of additive noise power, returned as a numeric array.

Algorithms

`wiener2` estimates the local mean and variance around each pixel.

$$\mu = \frac{1}{NM} \sum_{n_1, n_2 \in \eta} a(n_1, n_2)$$

and

$$\sigma^2 = \frac{1}{NM} \sum_{n_1, n_2 \in \eta} a^2(n_1, n_2) - \mu^2,$$

where η is the N-by-M local neighborhood of each pixel in the image `A`. `wiener2` then creates a pixelwise Wiener filter using these estimates,

$$b(n_1, n_2) = \mu + \frac{\sigma^2 - \nu^2}{\sigma^2}(a(n_1, n_2) - \mu),$$

where ν^2 is the noise variance. If the noise variance is not given, `wiener2` uses the average of all the local estimated variances.

References

[1] Lim, Jae S. *Two-Dimensional Signal and Image Processing*, Englewood Cliffs, NJ, Prentice Hall, 1990, p. 548, equations 9.44, 9.45, and 9.46.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

`wiener2` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`filter2` | `medfilt2` | `deconvwnr`

Topics

“Remove Noise By Adaptive Filtering”
“Noise Removal”

Introduced before R2006a

worldToIntrinsic

Convert from world to intrinsic coordinates

Syntax

```
[xIntrinsic, yIntrinsic] = worldToIntrinsic(R,xWorld,yWorld)
[xIntrinsic,yIntrinsic,zIntrinsic] = worldToIntrinsic(R,xWorld,yWorld,zWorld)
```

Description

`[xIntrinsic, yIntrinsic] = worldToIntrinsic(R,xWorld,yWorld)` maps points from the 2-D world system (`xWorld`,`yWorld`) to the 2-D intrinsic system (`xIntrinsic`,`yIntrinsic`) based on the relationship defined by 2-D spatial referencing object `R`.

If the *k*th input coordinates (`xWorld`(*k*),`yWorld`(*k*)) fall outside the image bounds in the world coordinate system, `worldToIntrinsic` extrapolates `xIntrinsic`(*k*) and `yIntrinsic`(*k*) outside the image bounds in the intrinsic coordinate system.

`[xIntrinsic,yIntrinsic,zIntrinsic] = worldToIntrinsic(R,xWorld,yWorld,zWorld)` maps points from the world coordinate system to the intrinsic coordinate system using 3-D spatial referencing object `R`.

Examples

Convert 2-D World Coordinates to Intrinsic Coordinates

Read a 2-D grayscale image of a knee into the workspace.

```
m = dicominfo('knee1.dcm');
A = dicomread(m);
```

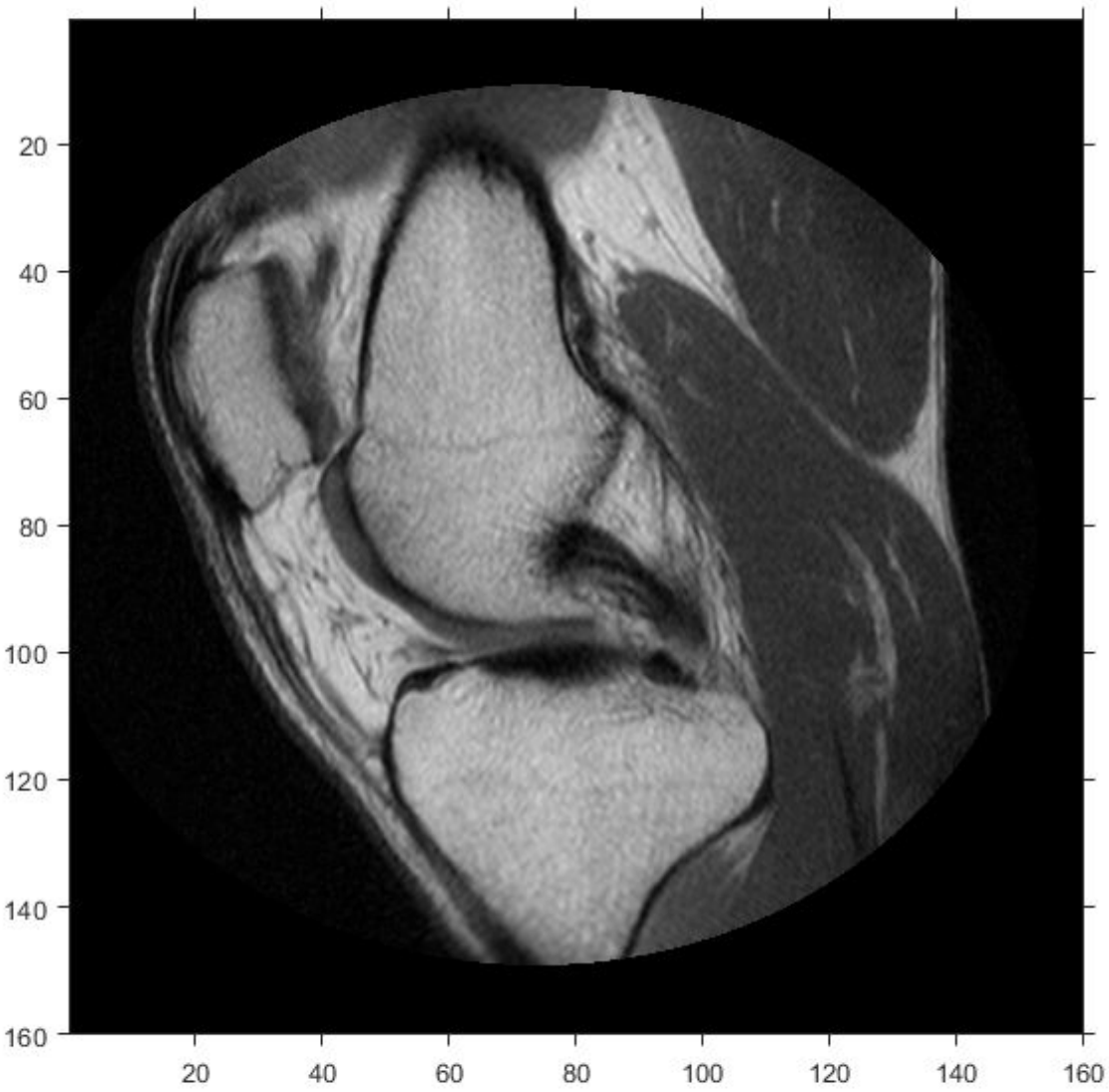
Create an `imref2d` object, specifying the size and the resolution of the pixels. The DICOM file contains a metadata field `PixelSpacing` that specifies the image resolution in each dimension in millimeters per pixel.

```
RA = imref2d(size(A),m.PixelSpacing(2),m.PixelSpacing(1))
```

```
RA =
  imref2d with properties:

           XWorldLimits: [0.1562 160.1562]
           YWorldLimits: [0.1562 160.1562]
              ImageSize: [512 512]
    PixelExtentInWorldX: 0.3125
    PixelExtentInWorldY: 0.3125
    ImageExtentInWorldX: 160
    ImageExtentInWorldY: 160
       XIntrinsicLimits: [0.5000 512.5000]
       YIntrinsicLimits: [0.5000 512.5000]
```

Display the image, including the spatial referencing object. The axes coordinates reflect the world coordinates. Notice that the coordinate (0,0) is in the upper left corner.

```
figure
imshow(A,RA,'DisplayRange',[0 512])
```



Select sample points, and store their world *x*- and *y*- coordinates in vectors. For example, the first point has world coordinates (38.44,68.75), the second point is 1 mm to the right of it, and the third point is 7 mm below it. The last point is outside the image boundary.

```
xW = [38.44 39.44 38.44 -0.2];
yW = [68.75 68.75 75.75 -1];
```

Convert the world coordinates to intrinsic coordinates using `worldToIntrinsic`.

```
[xI, yI] = worldToIntrinsic(RA,xW,yW)
```

```
xI = 1×4

  123.0080  126.2080  123.0080   -0.6400

yI = 1×4

  220.0000  220.0000  242.4000   -3.2000
```

The resulting vectors are the intrinsic *x*- and *y*- coordinates in units of pixels. Note that the intrinsic coordinate system is continuous, and some returned intrinsic coordinates have noninteger values. Also, `worldToIntrinsic` extrapolates the intrinsic coordinates of the point outside the image boundary.

Convert 3-D World Coordinates to Intrinsic Coordinates

Read a 3-D volume into the workspace. This image consists of 27 frames of 128-by-128 pixel images.

```
load mri;
D = squeeze(D);
D = ind2gray(D,map);
```

Create an `imref3d` spatial referencing object associated with the volume. For illustrative purposes, provide a pixel resolution in each dimension. The resolution is in millimeters per pixel.

```
R = imref3d(size(D),2,2,4)
```

```
R =
  imref3d with properties:

           XWorldLimits: [1 257]
           YWorldLimits: [1 257]
           ZWorldLimits: [2 110]
              ImageSize: [128 128 27]
    PixelExtentInWorldX: 2
    PixelExtentInWorldY: 2
    PixelExtentInWorldZ: 4
    ImageExtentInWorldX: 256
    ImageExtentInWorldY: 256
    ImageExtentInWorldZ: 108
       XIntrinsicLimits: [0.5000 128.5000]
       YIntrinsicLimits: [0.5000 128.5000]
       ZIntrinsicLimits: [0.5000 27.5000]
```

Select sample points, and store their world *x*-, *y*-, and *z*-coordinates in vectors. For example, the first point has world coordinates (108,92,52), the second point is 3 mm above it in the +*z*-direction, and the third point is 0.2 mm to the right of it in the +*x*-direction. The last point is outside the image boundary.

```
xW = [108 108 108.2 2];
yW = [92 92 92 -1];
zW = [52 55 52 0.33];
```

Convert the world coordinates to intrinsic coordinates using `worldToIntrinsic`.

```
[xI, yI, zI] = worldToIntrinsic(R,xW,yW,zW)
```

```
xI = 1×4

   54.0000   54.0000   54.1000    1.0000


yI = 1×4

   46.0000   46.0000   46.0000   -0.5000


zI = 1×4

   13.0000   13.7500   13.0000    0.0825
```

The resulting vectors are the intrinsic *x*-, *y*-, and *z*-coordinates in units of pixels. Note that the intrinsic coordinate system is continuous, and some returned intrinsic coordinates have noninteger values. Also, `worldToIntrinsic` extrapolates the intrinsic coordinates of the point outside the image boundary.

Input Arguments

R — Spatial referencing object

`imref2d` or `imref3d` object

Spatial referencing object, specified as an `imref2d` or `imref3d` object.

xWorld — Coordinates along the *x*-dimension in the world coordinate system

numeric scalar or vector

Coordinates along the *x*-dimension in the world coordinate system, returned as a numeric scalar or vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

yWorld — Coordinates along the *y*-dimension in the world coordinate system

numeric scalar or vector

Coordinates along the *y*-dimension in the world coordinate system, returned as a numeric scalar or vector. `yWorld` is the same length as `xWorld`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

zWorld — Coordinates along the *z*-dimension in the world coordinate system

numeric scalar or vector

Coordinates along the *z*-dimension in the world coordinate system, returned as a numeric scalar or vector. `zWorld` is the same length as `xWorld`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`xIntrinsic` — Coordinates along the *x*-dimension in the intrinsic coordinate system
numeric scalar or vector

Coordinates along the *x*-dimension in the intrinsic coordinate system, specified as a numeric scalar or vector. `xIntrinsic` is the same length as `xWorld`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`yIntrinsic` — Coordinates along the *y*-dimension in the intrinsic coordinate system
numeric scalar or vector

Coordinates along the *y*-dimension in the intrinsic coordinate system, specified as a numeric scalar or vector. `yIntrinsic` is the same length as `xWorld`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`zIntrinsic` — Coordinates along the *z*-dimension in the intrinsic coordinate system
numeric scalar or vector

Coordinates along the *z*-dimension in the intrinsic coordinate system, specified as a numeric scalar or vector. `zIntrinsic` is the same length as `xWorld` and `yWorld`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

See Also

`imref2d` | `imref3d` | `intrinsicToWorld` | `worldToSubscript`

Introduced in R2013a

worldToSubscript

Convert world coordinates to row and column subscripts

Syntax

```
[I, J] = worldToSubscript(R,xWorld,yWorld)
[I, J, K] = worldToSubscript(R,xWorld,yWorld,zWorld)
```

Description

`[I, J] = worldToSubscript(R,xWorld,yWorld)` maps points from the 2-D world system (`xWorld`,`yWorld`) to subscript arrays `I` and `J` based on the relationship defined by 2-D spatial referencing object `R`.

If the *k*th input coordinates (`xWorld`(*k*),`yWorld`(*k*)) fall outside the image bounds in the world coordinate system, `worldToSubscript` sets the corresponding subscripts `I`(*k*) and `J`(*k*) to `NaN`.

`[I, J, K] = worldToSubscript(R,xWorld,yWorld,zWorld)` maps points from the 3-D world system to subscript arrays `I`, `J`, and `K`, using 3-D spatial referencing object `R`.

Examples

Convert 2-D World Coordinates to Row and Column Subscripts

Read a 2-D grayscale image of a knee into the workspace.

```
m = dicominfo('knee1.dcm');
A = dicomread(m);
```

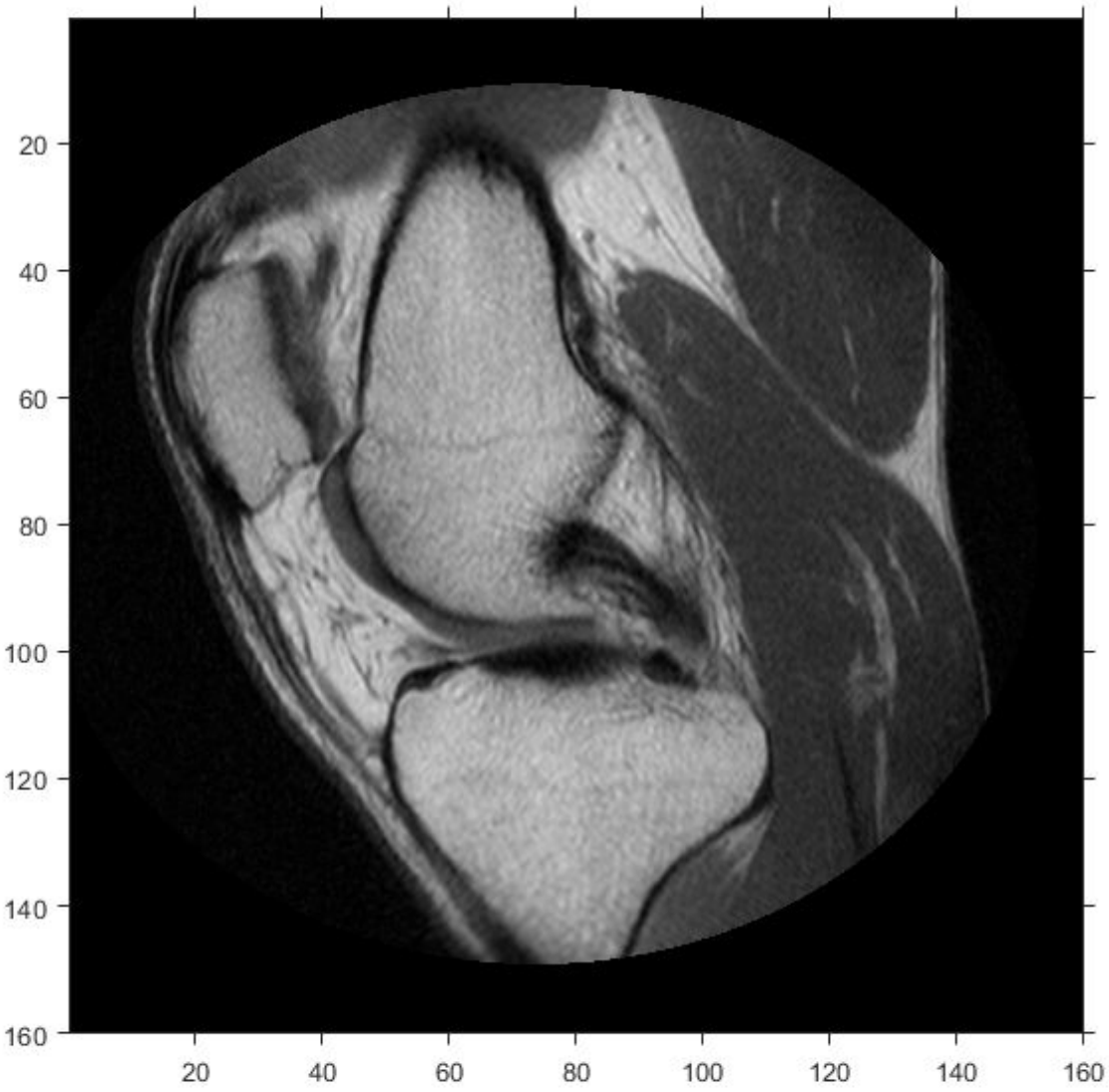
Create an `imref2d` object, specifying the size and the resolution of the pixels. The DICOM file contains a metadata field `PixelSpacing` that specifies the image resolution in each dimension in millimeters per pixel.

```
RA = imref2d(size(A),m.PixelSpacing(2),m.PixelSpacing(1))
```

```
RA =
  imref2d with properties:

          XWorldLimits: [0.1562 160.1562]
          YWorldLimits: [0.1562 160.1562]
             ImageSize: [512 512]
    PixelExtentInWorldX: 0.3125
    PixelExtentInWorldY: 0.3125
    ImageExtentInWorldX: 160
    ImageExtentInWorldY: 160
       XIntrinsicLimits: [0.5000 512.5000]
       YIntrinsicLimits: [0.5000 512.5000]
```

Display the image, including the spatial referencing object. The axes coordinates reflect the world coordinates. Notice that the coordinate (0,0) is in the upper left corner.

```
figure
imshow(A,RA,'DisplayRange',[0 512])
```



Select sample points, and store their world *x*- and *y*- coordinates in vectors. For example, the first point has world coordinates (38.44,68.75), the second point is 1 mm to the right of it, and the third point is 7 mm below it. The last point is outside the image boundary.

```
xW = [38.44 39.44 38.44 -0.2];
yW = [68.75 68.75 75.75 1];
```

Convert the world coordinates to row and column subscripts using `worldToSubscript`.

```
[rS, cS] = worldToSubscript(RA,xW,yW)
```

```
rS = 1×4
```

```
   220   220   242   NaN

cS = 1×4

   123   126   123   NaN
```

The resulting vectors contain the row and column indices that are closest to the point. Note that the indices are discrete, and that points outside the image boundary have `NaN` for both row and column indices.

Also, the order of the input and output coordinates is reversed. The world *x*-coordinate vector, `xW`, corresponds to the second output vector, `cS`. The world *y*-coordinate vector, `yW`, corresponds to the first output vector, `rS`.

Convert 3-D World Coordinates to Row, Column, and Plane Subscripts

Read a 3-D volume into the workspace. This image consists of 27 frames of 128-by-128 pixel images.

```
load mri;
D = squeeze(D);
D = ind2gray(D,map);
```

Create an `imref3d` spatial referencing object associated with the volume. For illustrative purposes, provide a pixel resolution in each dimension. The resolution is in millimeters per pixel.

```
R = imref3d(size(D),2,2,4)
```

```
R =
  imref3d with properties:

           XWorldLimits: [1 257]
           YWorldLimits: [1 257]
           ZWorldLimits: [2 110]
              ImageSize: [128 128 27]
    PixelExtentInWorldX: 2
    PixelExtentInWorldY: 2
    PixelExtentInWorldZ: 4
    ImageExtentInWorldX: 256
    ImageExtentInWorldY: 256
    ImageExtentInWorldZ: 108
       XIntrinsicLimits: [0.5000 128.5000]
       YIntrinsicLimits: [0.5000 128.5000]
       ZIntrinsicLimits: [0.5000 27.5000]
```

Select sample points, and store their world *x*-, *y*-, and *z*-coordinates in vectors. For example, the first point has world coordinates (108,92,52), the second point is 3 mm above it in the +*z*-direction, and the third point is 5.2 mm to the right of it in the +*x*-direction. The last point is outside the image boundary.

```
xW = [108 108 113.2 2];
yW = [92 92 92 -1];
zW = [52 55 52 0.33];
```

Convert the world coordinates to row, column, and plane subscripts using `worldToSubscript`.

```
[rS, cS, pS] = worldToSubscript(R,xW,yW,zW)
```

```
rS = 1×4

    46    46    46   NaN


cS = 1×4

    54    54    57   NaN


pS = 1×4

    13    14    13   NaN
```

The resulting vectors contain the column, row, and plane indices that are closest to the point. Note that the indices are discrete, and that points outside the image boundary have index values of `NaN`.

Also, the order of the input and output coordinates is reversed. The world *x*-coordinate vector, `xW`, corresponds to the second output vector, `cS`. The world *y*-coordinate vector, `yW`, corresponds to the first output vector, `rS`.

Input Arguments

`R` — Spatial referencing object

`imref2d` or `imref3d` object

Spatial referencing object, specified as an `imref2d` or `imref3d` object.

`xWorld` — Coordinates along the *x*-dimension in the world coordinate system

numeric scalar or vector

Coordinates along the *x*-dimension in the world coordinate system, specified as a numeric scalar or vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`yWorld` — Coordinates along the *y*-dimension in the world coordinate system

numeric scalar or vector

Coordinates along the *y*-dimension in the world coordinate system, specified as a numeric scalar or vector. `yWorld` is the same length as `xWorld`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`zWorld` — Coordinates along the *z*-dimension in the world coordinate system

numeric scalar or vector

Coordinates along the *z*-dimension in the world coordinate system, specified as a numeric scalar or vector. `zWorld` is the same length as `xWorld`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`I` — Row indices

positive integer scalar or vector

Row indices, returned as a positive integer scalar or vector. `I` is the same length as `yWorld`. For an *m*-by-*n* or *m*-by-*n*-by-*p* image, $1 \le I \le m$.

Data Types: `double`

`J` — Column indices

positive integer scalar or vector

Column indices, returned as a positive integer scalar or vector. `J` is the same length as `xWorld`. For an *m*-by-*n* or *m*-by-*n*-by-*p* image, $1 \le J \le n$.

Data Types: `double`

`K` — Plane indices

positive integer scalar or vector

Plane indices, returned as a positive integer scalar or vector. `K` is the same length as `zWorld`. For an *m*-by-*n*-by-*p* image, $1 \le K \le p$.

Data Types: `double`

See Also

`imref2d` | `imref3d` | `worldToIntrinsic`

Introduced in R2013a

xyz2double

Convert XYZ color values to `double`

Syntax

```
xyzD = xyz2double(xyz)
```

Description

`xyzD = xyz2double(xyz)` converts XYZ color values to type `double`.

Examples

Convert XYZ Color Values to double

This example shows how to convert `uint16`-encoded XYZ values to `double`.

Create a `uint16` vector specifying a color in XYZ colorspace.

```
c = uint16([100 32768 65535]);
```

Convert the XYZ color value to `double`.

```
xyz2double(c)
```

```
ans = 1×3

    0.0031    1.0000    2.0000
```

Input Arguments

xyz — Color values to convert

m-by-3 numeric matrix | *m*-by-*n*-by-3 numeric array

Color values to convert, specified as a *m*-by-3 numeric matrix of color values (one color per row), or an *m*-by-*n*-by-3 numeric array.

Data Types: `uint16`

Output Arguments

xyzD — Converted color values

numeric array

Converted color values, returned as a numeric array of same size as the input.

Data Types: `double`

Algorithms

The Image Processing Toolbox software follows the convention that double-precision XYZ arrays contain 1931 CIE XYZ values (2° observer). The XYZ arrays that are `uint16` follow the convention in the ICC profile specification (ICC.1:2001-4, `www.color.org`) for representing XYZ values as unsigned 16-bit integers. There is no standard representation of XYZ values as unsigned 8-bit integers. The ICC encoding convention is illustrated by this table.

Value (X, Y, or Z)	`uint16` Value
`0.0`	`0`
`1.0`	32768
`1.0 + (32767/32768)`	65535

See Also

`applycform` | `lab2double` | `lab2uint16` | `lab2uint8` | `makecform` | `whitepoint` | `xyz2uint16`

Introduced before R2006a

xyz2rgb

Convert CIE 1931 XYZ to RGB

Syntax

```
RGB = xyz2rgb(XYZ)
RGB = xyz2rgb(XYZ,Name,Value)
```

Description

`RGB = xyz2rgb(XYZ)` converts CIE 1931 XYZ values (2° observer) to sRGB values.

`RGB = xyz2rgb(XYZ,Name,Value)` specifies additional conversion options, such as the color space of the RGB image, using one or more name-value pair arguments.

Examples

Convert XYZ color to sRGB

Convert a color value in the XYZ color space to the sRGB color space.

```
xyz2rgb([0.25 0.40 0.10])
```

```
ans = 1×3

    0.4174    0.7434    0.2152
```

Convert XYZ Color to Adobe RGB

Convert the color value in XYZ color space to the Adobe RGB (1998) color space.

```
xyz2rgb([0.25 0.40 0.10],'ColorSpace','adobe-rgb-1998')
```

```
ans = 1×3

    0.5323    0.7377    0.2730
```

Convert XYZ color to sRGB Specifying Whitepoint

Convert an XYZ color value to sRGB specifying the D50 whitepoint.

```
xyz2rgb([0.25 0.40 0.10],'WhitePoint','d50')
```

```
ans = 1×3
```

```
    0.3276    0.7517    0.2869
```

Convert XYZ color to 8-bit-encoded RGB Color

Convert an XYZ color value to an 8-bit encoded RGB color value.

```
xyz2rgb([0.25 0.40 0.10],'OutputType','uint8')
```

```
ans = 1x3 uint8 row vector

   106   190    55
```

Input Arguments

XYZ — XYZ color values

numeric array

XYZ color values to convert, specified as a numeric array in one of the following formats.

- *c*-by-3 colormap. Each row specifies one XYZ color value.
- *m*-by-*n*-by-3 image.
- *m*-by-*n*-by-3-by-*p* stack of images.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `xyz2rgb([0.25 0.40 0.10],'ColorSpace','adobe-rgb-1998')`

`ColorSpace` — Color space of the output RGB values

`'srgb'` (default) | `'adobe-rgb-1998'` | `'linear-rgb'`

Color space of the output RGB values, specified as the comma-separated pair consisting of `'ColorSpace'` and `'srgb'`, `'adobe-rgb-1998'`, or `'linear-rgb'`. If you specify `'linear-rgb'`, then `xyz2rgb` returns linearized sRGB values.

Data Types: `char`

`WhitePoint` — Reference white point

`'d65'` (default) | `'a'` | `'c'` | `'e'` | `'d50'` | `'d55'` | `'icc'` | 1-by-3 vector

Reference white point, specified as the comma-separated pair consisting of `'WhitePoint'` and a 1-by-3 vector or one of the CIE standard illuminants listed in the table.

Value	White Point
`'a'`	CIE standard illuminant A, `[1.0985, 1.0000, 0.3558]`. Simulates typical, domestic, tungsten-filament lighting with correlated color temperature of 2856 K.
`'c'`	CIE standard illuminant C, `[0.9807, 1.0000, 1.1822]`. Simulates average or north sky daylight with correlated color temperature of 6774 K. Deprecated by CIE.
`'e'`	Equal-energy radiator, `[1.000, 1.000, 1.000]`. Useful as a theoretical reference.
`'d50'`	CIE standard illuminant D50, `[0.9642, 1.0000, 0.8251]`. Simulates warm daylight at sunrise or sunset with correlated color temperature of 5003 K. Also known as horizon light.
`'d55'`	CIE standard illuminant D55, `[0.9568, 1.0000, 0.9214]`. Simulates mid-morning or mid-afternoon daylight with correlated color temperature of 5500 K.
`'d65'`	CIE standard illuminant D65, `[0.9504, 1.0000, 1.0888]`. Simulates noon daylight with correlated color temperature of 6504 K.
`'icc'`	Profile Connection Space (PCS) illuminant used in ICC profiles. Approximation of `[0.9642, 1.000, 0.8249]` using fixed-point, signed, 32-bit numbers with 16 fractional bits. Actual value: `[31595,32768, 27030]/32768`.

Data Types: `single` | `double` | `char`

`OutputType` — Data type of returned RGB values
`'double'` | `'single'` | `'uint8'` | `'uint16'`

Data type of returned RGB values, specified as one of the following values: `'double'`, `'single'`, `'uint8'`, or `'uint16'`. If you do not specify `OutputType`, the output type is the same type as the input.

Data Types: `char`

Output Arguments

`RGB` — Converted RGB color values
numeric array

Converted RGB color values, returned as a numeric array of the same size as the input. The output type is the same as the input type unless you specify the `OutputType` parameter.

Tips

- If you specify the output RGB color space as `'linear-rgb'`, then the output values are linearized sRGB values. If instead you want the output color space to be linearized Adobe RGB (1998), then you can use the `rgb2lin` function.

  For example, to convert CIE 1931 XYZ image `XYZ` to linearized Adobe RGB (1998) color space, perform the conversion in two steps:

  ```
  RGBadobe = xyz2rgb(XYZ,'ColorSpace','adobe-rgb-1998');
  RGBlinadobe = rgb2lin(RGBadobe,'ColorSpace','adobe-rgb-1998');
  ```

See Also

`rgb2xyz` | `xyz2lab` | `lab2rgb` | `rgb2lin` | `xyz2rgbwide` | `rgbwide2xyz`

Topics

“Understanding Color Spaces and Color Space Conversion”
“Device-Independent Color Spaces”

Introduced in R2014b

xyz2rgbwide

Convert CIE 1931 XYZ color values to wide-gamut RGB color values

Syntax

```
RGB = xyz2rgbwide(XYZ,BPS)
RGB = xyz2rgbwide(XYZ,BPS,Name,Value)
```

Description

`RGB = xyz2rgbwide(XYZ,BPS)` converts the specified CIE 1931 XYZ color values to wide-gamut RGB values in the BT.2020 or BT.2100 color space. `BPS` specifies the number of bits required to represent each channel of the output RGB values.

`RGB = xyz2rgbwide(XYZ,BPS,Name,Value)` specifies options using one or more name-value pair arguments.

Examples

Convert CIE 1931 XYZ Values into Wide-Gamut RGB

Convert XYZ color values into 10-bit or 12-bit wide-gamut RGB values in the BT.2020/BT.2100 color space.

Convert XYZ Color into 10-bit BT.2020 RGB Value

Create an XYZ value.

```
xyzvalue = [0.25 0.40 0.10];
```

Convert the XYZ value to a 10-bit BT.2020 RGB value.

```
rgbvalue = xyz2rgbwide(xyzvalue,10)
```

```
rgbvalue = 1x3 uint16 row vector

   504   670   289
```

Convert XYZ Color into 12-bit BT.2100 RGB Value

Create an XYZ value.

```
xyzvalue = [0.25 0.40 0.10];
```

Convert the XYZ value to a 12-bit BT.2100 RGB value.

```
rgbvalue = xyz2rgbwide(xyzvalue,12,'Colorspace','BT.2100')
```

```
rgbvalue = 1x3 uint16 row vector
```

```
   2015   2681   1155
```

Convert XYZ Color into 10-bit BT.2100 RGB Value Using HLG

Create an XYZ value.

```
xyzvalue = [0.25 0.40 0.10];
```

Convert the XYZ value to a 10-bit BT.2100 RGB value using the Hybrid Log Gamma (HLG) transfer function.

```
rgbvalue = xyz2rgbwide(xyzvalue,12,'Colorspace','BT.2100','LinearizationFcn','HLG')
```

```
rgbvalue = 1x3 uint16 row vector

   2875   3285   1989
```

Input Arguments

XYZ — Color values in CIE 1931 XYZ color space

p-by-*3* numeric matrix | *m*-by-*n*-by-3 numeric array | *m*-by-*n*-by-3-by-*f* numeric array

Color values in the CIE 1931 XYZ color space, specified as one of the following:

- *p*-by-*3* numeric matrix of color values (one color per row)
- *m*-by-*n*-by-3 numeric array representing an image
- *m*-by-*n*-by-3-by-*f* numeric array representing a stack of images

Data Types: `single` | `double`

BPS — Bits per sample for each channel of output RGB image

`10` | `12`

Bits per sample for each channel of the output wide-gamut RGB image, specified as `10` or `12`.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `rgb = xyz2rgbwide([0.25 0.40 0.10],12,'ColorSpace','BT.2100')`

`ColorSpace` — Color space of output RGB values

`'BT.2020'` (default) | `'BT.2100'`

Color space of the output RGB values, specified as the comma-separated pair consisting of `'ColorSpace'` and the value `'BT.2020'` or `'BT.2100'`.

Data Types: `char` | `string`

`WhitePoint` — Reference white point

`'d65'` (default) | `'a'` | `'c'` | `'d50'` | `'d55'` | `'icc'` | `'e'` | 1-by-3 vector

Reference white point, specified as the comma-separated pair consisting of `'WhitePoint'` and a 1-by-3 vector or one of the CIE standard illuminants, listed in the table.

Value	White Point
`'a'`	CIE standard illuminant A, `[1.0985, 1.0000, 0.3558]`. Simulates typical, domestic, tungsten-filament lighting with correlated color temperature of 2856 K.
`'c'`	CIE standard illuminant C, `[0.9807, 1.0000, 1.1822]`. Simulates average or north sky daylight with correlated color temperature of 6774 K. Deprecated by CIE.
`'e'`	Equal-energy radiator, `[1.000, 1.000, 1.000]`. Useful as a theoretical reference.
`'d50'`	CIE standard illuminant D50, `[0.9642, 1.0000, 0.8251]`. Simulates warm daylight at sunrise or sunset with correlated color temperature of 5003 K. Also known as horizon light.
`'d55'`	CIE standard illuminant D55, `[0.9568, 1.0000, 0.9214]`. Simulates mid-morning or mid-afternoon daylight with correlated color temperature of 5500 K.
`'d65'`	CIE standard illuminant D65, `[0.9504, 1.0000, 1.0888]`. Simulates noon daylight with correlated color temperature of 6504 K.
`'icc'`	Profile Connection Space (PCS) illuminant used in ICC profiles. Approximation of `[0.9642, 1.000, 0.8249]` using fixed-point, signed, 32-bit numbers with 16 fractional bits. Actual value: `[31595,32768, 27030]/32768`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

`LinearizationFcn` — Transfer function for transformation
`'PQ'` (default) | `'HLG'`

Transfer function for transformation, specified as the text string `'LinearizationFcn'` and either of the following values:

Value	Description
`'PQ'`	Perceptual Quantization
`'HLG'`	Hybrid Log Gamma

Data Types: `char` | `string`

Output Arguments

`RGB` — Output RGB color values
numeric array

Output RGB color values, returned as a numeric array of the same size as the `XYZ` input value. The table shows the data range for the wide-gamut color values for 10- and 12-bit data. The minimum value in each range maps to black, and the maximum value in each range maps to white.

Data Type	Full Data Range	Data Range for Wide-Gamut RGB
10-bit	[0, 1023]	[64, 940]
12-bit	[0, 4095]	[256, 3760]

Data Types: `uint16`

References

[1] Rec. ITU-R BT.2020-2 (10/2015). "Parameter values for ultra-high definition television systems for production and international programme exchange." *International Telecommunication Union; Broadcasting service (television)*. https://www.itu.int/rec/R-REC-BT.2020.

[2] Rec. ITU-R BT.2100-2 (07/2018). "Image parameter values for dynamic range television for use in production and international programme exchange." *International Telecommunication Union; Broadcasting service (television)*. https://www.itu.int/rec/R-REC-BT.2100.

[3] Rec. ITU-R BT.2390-7 (07/2019). "High dynamic range television for production and international programme exchange." *International Telecommunication Union; Broadcasting service (television)*. https://www.itu.int/pub/R-REP-BT.2390.

See Also

`rgb2xyz` | `xyz2rgb` | `rgbwide2xyz`

Introduced in R2020b

xyz2lab

Convert CIE 1931 XYZ to CIE 1976 L*a*b*

Syntax

```
lab = xyz2lab(xyz)
lab = xyz2lab(xyz,'WhitePoint',whitePoint)
```

Description

`lab = xyz2lab(xyz)` converts CIE 1931 XYZ values (2° observer) to CIE 1976 L*a*b* values.

`lab = xyz2lab(xyz,'WhitePoint',whitePoint)` specifies the reference white point of the illuminant.

Examples

Convert XYZ Color to L*a*b*

Convert an XYZ color value to L*a*b* using the default reference white point, D65.

```
xyz2lab([0.25 0.40 0.10])
```

```
ans = 1×3

   69.4695  -48.0439   57.1259
```

Convert XYZ Color to L*a*b* Specifying Whitepoint

Convert an XYZ color value to L*a*b* specifying the D50 whitepoint.

```
xyz2lab([0.25 0.40 0.10],'WhitePoint','d50')
```

```
ans = 1×3

   69.4695  -49.5717   48.3864
```

Input Arguments

xyz — XYZ color values

numeric array

XYZ color values to convert, specified as a numeric array in one of these formats.

- *c*-by-3 colormap. Each row specifies one XYZ color value.
- *m*-by-*n*-by-3 image
- *m*-by-*n*-by-3-by-*p* stack of images

Data Types: `single` | `double`

`whitePoint` — Reference white point

`'d65'` (default) | `'a'` | `'c'` | `'e'` | `'d50'` | `'d55'` | `'icc'` | 1-by-3 vector

Reference white point, specified as a 1-by-3 vector or one of the CIE standard illuminants, listed in the table.

Value	White Point
`'a'`	CIE standard illuminant A, `[1.0985, 1.0000, 0.3558]`. Simulates typical, domestic, tungsten-filament lighting with correlated color temperature of 2856 K.
`'c'`	CIE standard illuminant C, `[0.9807, 1.0000, 1.1822]`. Simulates average or north sky daylight with correlated color temperature of 6774 K. Deprecated by CIE.
`'e'`	Equal-energy radiator, `[1.000, 1.000, 1.000]`. Useful as a theoretical reference.
`'d50'`	CIE standard illuminant D50, `[0.9642, 1.0000, 0.8251]`. Simulates warm daylight at sunrise or sunset with correlated color temperature of 5003 K. Also known as horizon light.
`'d55'`	CIE standard illuminant D55, `[0.9568, 1.0000, 0.9214]`. Simulates mid-morning or mid-afternoon daylight with correlated color temperature of 5500 K.
`'d65'`	CIE standard illuminant D65, `[0.9504, 1.0000, 1.0888]`. Simulates noon daylight with correlated color temperature of 6504 K.
`'icc'`	Profile Connection Space (PCS) illuminant used in ICC profiles. Approximation of `[0.9642, 1.000, 0.8249]` using fixed-point, signed, 32-bit numbers with 16 fractional bits. Actual value: `[31595,32768, 27030]/32768`.

Data Types: `single` | `double` | `char`

Output Arguments

`lab` — Converted L*a*b* color values

numeric array

Converted L*a*b* color values, returned as a numeric array of the same size and data type as the input.

Attribute	Description
*L**	Luminance or brightness of the image. Values are in the range [0, 100], where 0 specifies black and 100 specifies white. As *L** increases, colors become brighter.

Attribute	Description
*a**	Amount of red or green tones in the image. A large positive *a** value corresponds to red/magenta. A large negative *a** value corresponds to green. Although there is no single range for *a**, values commonly fall in the range [-100, 100] or [-128, 127).
*b**	Amount of yellow or blue tones in the image. A large positive *b** value corresponds to yellow. A large negative *b** value corresponds to blue. Although there is no single range for *b**, values commonly fall in the range [-100, 100] or [-128, 127).

Data Types: `single` | `double`

See Also

`rgb2lab` | `xyz2rgb` | `lab2xyz`

Introduced in R2014b

xyz2uint16

Convert XYZ color values to `uint16`

Syntax

```
xyz16 = xyz2uint16(xyz)
```

Description

`xyz16 = xyz2uint16(xyz)` converts XYZ color values to type `uint16`.

Examples

Convert XYZ Color Values to uint16

This example shows how to convert XYZ color values from `double` to `uint16`.

Create a `double` vector specifying a color in XYZ colorspace.

```
c = [0.1 0.5 1.0];
```

Convert the XYZ color value to `uint16`.

```
xyz2uint16(c)
```

```
ans = 1x3 uint16 row vector

    3277   16384   32768
```

Input Arguments

xyz — Color values to convert

m-by-3 numeric matrix | *m*-by-*n*-by-3 numeric array

Color values to convert, specified as a *m*-by-3 numeric matrix of color values (one color per row), or an *m*-by-*n*-by-3 numeric array.

Data Types: `double`

Output Arguments

xyz16 — Converted color values

numeric array

Converted color values, returned as a numeric array of the same size as the input.

Data Types: `uint16`

Algorithms

The Image Processing Toolbox software follows the convention that double-precision XYZ arrays contain 1931 CIE XYZ values (2° observer). The XYZ arrays that are `uint16` follow the convention in the ICC profile specification (ICC.1:2001-4, `www.color.org`) for representing XYZ values as unsigned 16-bit integers. There is no standard representation of XYZ values as unsigned 8-bit integers. The ICC encoding convention is illustrated by this table.

Value (X, Y, or Z)	uint16 Value
0.0	0
1.0	32768
1.0 + (32767/32768)	65535

See Also

`applycform` | `lab2double` | `lab2uint16` | `lab2uint8` | `makecform` | `whitepoint` | `xyz2double`

Introduced before R2006a

ycbcr2rgb

Convert YCbCr color values to RGB color space

Syntax

```
RGB = ycbcr2rgb(YCBCR)
```

Description

`RGB = ycbcr2rgb(YCBCR)` converts the luminance (*Y*) and chrominance (*Cb* and *Cr*) values of a YCbCr image to red, green, and blue values of an RGB image.

Examples

Convert Image from YCbCr to RGB

This example shows how to convert an image from RGB to YCbCr color space and back.

Read an RGB image into the workspace.

```
RGB = imread('board.tif');
```

Convert the image to YCbCr color space.

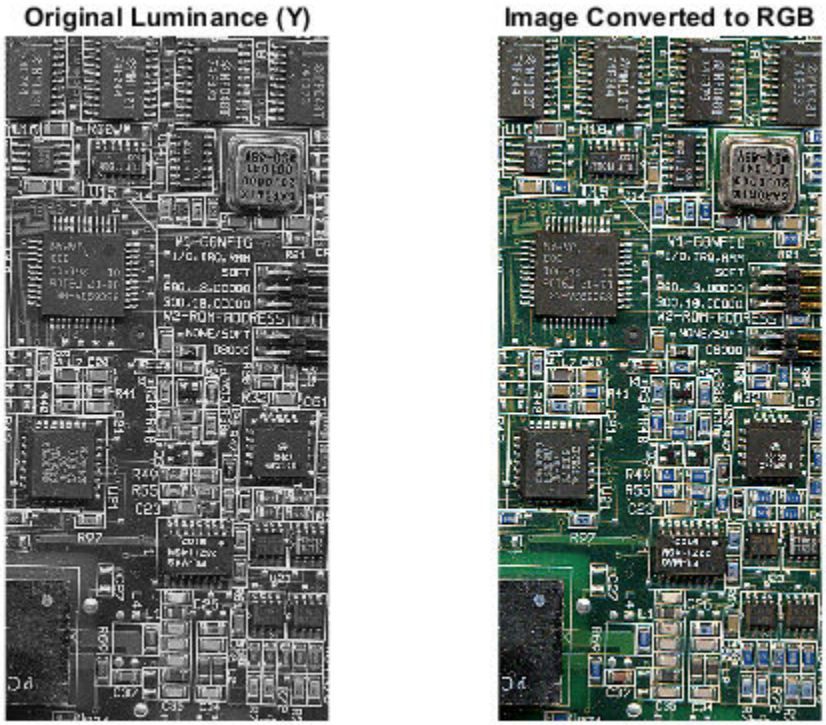
```
YCBCR = rgb2ycbcr(RGB);
```

Convert the YCbCr image back to RGB color space.

```
RGB2 = ycbcr2rgb(YCBCR);
```

Display the luminance channel of the image in YCbCr color space alongside the image that was converted from YCbCr to RGB color space.

```
figure
subplot(1,2,1)
imshow(YCBCR(:,:,1))
title('Original Luminance (Y)');
subplot(1,2,2)
imshow(RGB2);
title('Image Converted to RGB');
```



Input Arguments

YCBCR — YCbCr color values

numeric array

YCbCr color values to convert, specified as a numeric array in one of these formats.

- *c*-by-3 colormap. Each row specifies one YCbCr color value.
- *m*-by-*n*-by-3 image.

Data Types: `single` | `double` | `uint8` | `uint16`

Output Arguments

RGB — Converted RGB color values

numeric array

Converted RGB color values, returned as a numeric array of the same size as the input. The output data type is the same as the input data type.

References

[1] Poynton, C. A. *A Technical Introduction to Digital Video*, John Wiley & Sons, Inc., 1996, p. 175.

[2] Rec. ITU-R BT.601-5, *Studio Encoding Parameters of Digital Television for Standard 4:3 and Widescreen 16:9 Aspect Ratios*, (1982-1986-1990-1992-1994-1995), Section 3.5.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `ycbcr2rgb` supports the generation of C code (requires MATLAB Coder). Note that if you choose the generic `MATLAB Host Computer` target platform, `ycbcr2rgb` generates code that uses a precompiled, platform-specific shared library. Use of a shared library preserves performance optimizations but limits the target platforms for which code can be generated. For more information, see “Types of Code Generation Support in Image Processing Toolbox”.

GPU Code Generation

Generate CUDA® code for NVIDIA® GPUs using GPU Coder™.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

This function fully supports GPU arrays. For more information, see “Image Processing on a GPU”.

See Also

`ntsc2rgb` | `rgb2ntsc` | `rgb2ycbcr` | `ycbcr2rgbwide` | `rgbwide2ycbcr`

Topics

“Understanding Color Spaces and Color Space Conversion”

Introduced before R2006a

ycbcr2rgbwide

Convert YCbCr color values to wide-gamut RGB color values

Syntax

```
RGB = ycbcr2rgbwide(YCbCr,BPS)
```

Description

`RGB = ycbcr2rgbwide(YCbCr,BPS)` converts non-constant luminance `YCbCr` values into wide-gamut RGB values in the BT.2020 or BT.2100 color spaces. `BPS` specifies the number of bits required to represent each channel in the output image.

Examples

Convert YCbCr Color Values to Wide-Gamut RGB Color Values

Convert 10-bit and 12-bit YCbCr color values to the wide-gamut RGB color values in the BT.2020 or BT.2100 color spaces.

Convert 12-bit YCbCr Color Value to Wide-Gamut RGB Color Value

Create a 12-bit YCbCr color value in the workspace.

```
ycbcrlist = uint16([3760 2048 2048]);
```

Convert the YCbCr color value to a wide-gamut RGB color value.

```
rgblist = ycbcr2rgbwide(ycbcrlist, 12);
```

Convert 10-bit YCbCr Image to Wide-Gamut RGB Image

Create a synthetic YCbCr image in the workspace.

```
YCBCR = reshape(uint16([64 512 512; 940 512 512]),[2 1 3]);
```

Convert the YCbCr image to a wide-gamut RGB image.

```
RGB = ycbcr2rgbwide(YCBCR,10);
```

Input Arguments

`YCbCr` — YCbCr color values

p-by-*3* numeric matrix | *m*-by-*n*-by-3 numeric array

YCbCr color values, specified as one of these options:

- *p*-by-*3* numeric matrix of color values (one color per row)
- *m*-by-*n*-by-3 numeric array representing an image

Data Types: `uint16`

BPS — Bits per sample for each channel of output image
`10` | `12`

Bits per sample for each channel of the output wide-gamut RGB image, specified as `10` or `12`.

Output Arguments

RGB — Wide-gamut RGB values
numeric array

Wide-gamut RGB values, returned as a numeric array of the same size as the input YCbCr values.

The following table shows the data range for the wide-gamut, integer color values for 10- and 12-bit data. The minimum value in the range maps to black, and the maximum value in the range maps to white. The `ycbcr2rgbwide` function maps only pixels with RGB values within the supported data range to valid YCbCr values.

Data Type	Full Data Range	Data Range for Wide-Gamut RGB
10-bit	[0, 1023]	[64, 940]
12-bit	[0, 4095]	[256, 3760]

Data Types: `uint16`

Tips

- This table shows the data ranges of the YCbCr values for BT.2020 and BT.2100 color spaces.

Component	10-bit	12-bit
Y	`[64, 940]`	`[256, 3760]`
Cb, Cr	`[64, 960]`	`[256, 3840]`

References

[1] Rec. ITU-R BT.2020-2 (10/2015). "Parameter values for ultra-high definition television systems for production and international programme exchange." *International Telecommunication Union; Broadcasting service (television)*. https://www.itu.int/rec/R-REC-BT.2020.

[2] Rec. ITU-R BT.2100-2 (07/2018). "Image parameter values for dynamic range television for use in production and international programme exchange." *International Telecommunication Union; Broadcasting service (television)*. https://www.itu.int/rec/R-REC-BT.2100.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

`ycbcr2rgbwide` supports the generation of C code (requires MATLAB Coder). For more information, see “Code Generation for Image Processing”.

GPU Arrays
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

See Also

`rgb2ycbcr` | `rgbwide2ycbcr` | `ycbcr2rgb`

Introduced in R2020b

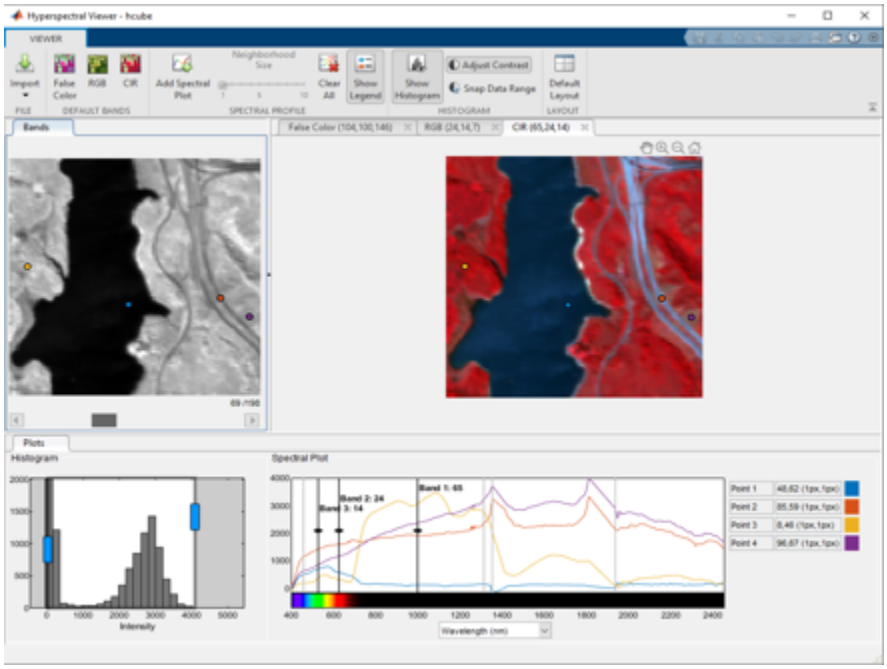
Hyperspectral Viewer

Visualize hyperspectral data

Description

The **Hyperspectral Viewer** app visualizes hyperspectral data and enables you to create spectral profiles of points and regions in the data.

Note This app requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.



Open the Hyperspectral Viewer App

- MATLAB Toolstrip: On the **Apps** tab, under **Image Processing and Computer Vision**, click the **Hyperspectral Viewer** app icon.
- MATLAB command prompt: Enter `hyperspectralViewer`. For more information, see “Programmatic Use” on page 1-3202.

Examples

Open Hyperspectral Viewer

Construct a `hypercube` object with the Indian Pines hyperspectral data set.

```
hcube = hypercube('indian_pines.dat');
```

Open the **Hyperspectral Viewer** app with the Indian Pines data.

```
hyperspectralViewer(hcube);
```

- “Explore Hyperspectral Data in the Hyperspectral Viewer”

Programmatic Use

`hyperspectralViewer` opens the **Hyperspectral Viewer** app.

`hyperspectralViewer(hcube)` opens the **Hyperspectral Viewer** app, loading the `hypercube` object `hcube` into the app.

`hyperspectralViewer(cube)` opens the **Hyperspectral Viewer** app, loading the 3-D array `cube`. When not loading a `hypercube` object, the capabilities of the app are limited.

`hyperspectralViewer close` closes all open **Hyperspectral Viewer** apps.

See Also

Objects
`hypercube`

Topics
“Explore Hyperspectral Data in the Hyperspectral Viewer”

Introduced in R2020a

anomalyRX

Detect anomalies using Reed-Xiaoli detector

Syntax

```
rxScore = anomalyRX(inputData)
```

Description

`rxScore = anomalyRX(inputData)` detects anomalous pixels in the hyperspectral data using the Reed-Xialoi (RX) detector. The RX detector calculates a score for each pixel as the Mahalanobis distance between the pixel and the background. The higher score indicates a likely anomaly. The background is characterized by the spectral mean and covariance of the data cube. For more information about computing the score and detecting anomalies, see “Algorithms” on page 1-3207.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Detect Anomalous Pixels in Hyperspectral Data Using RX Detector

Detect anomalous pixels in hyperspectral data by computing the RX score for each pixel in a hyperspectral data cube. Then compute the threshold for detecting true anomalous pixels by using cumulative probability distribution of RX score values.

Read hyperspectral data containing anomalous pixels into the workspace.

```
hcube = hypercube('indian_pines.dat');
```

Find anomalous pixels in the input hyperspectral data by using the RX detector. The detector searches for pixels with a high intensity difference within a homogeneous region.

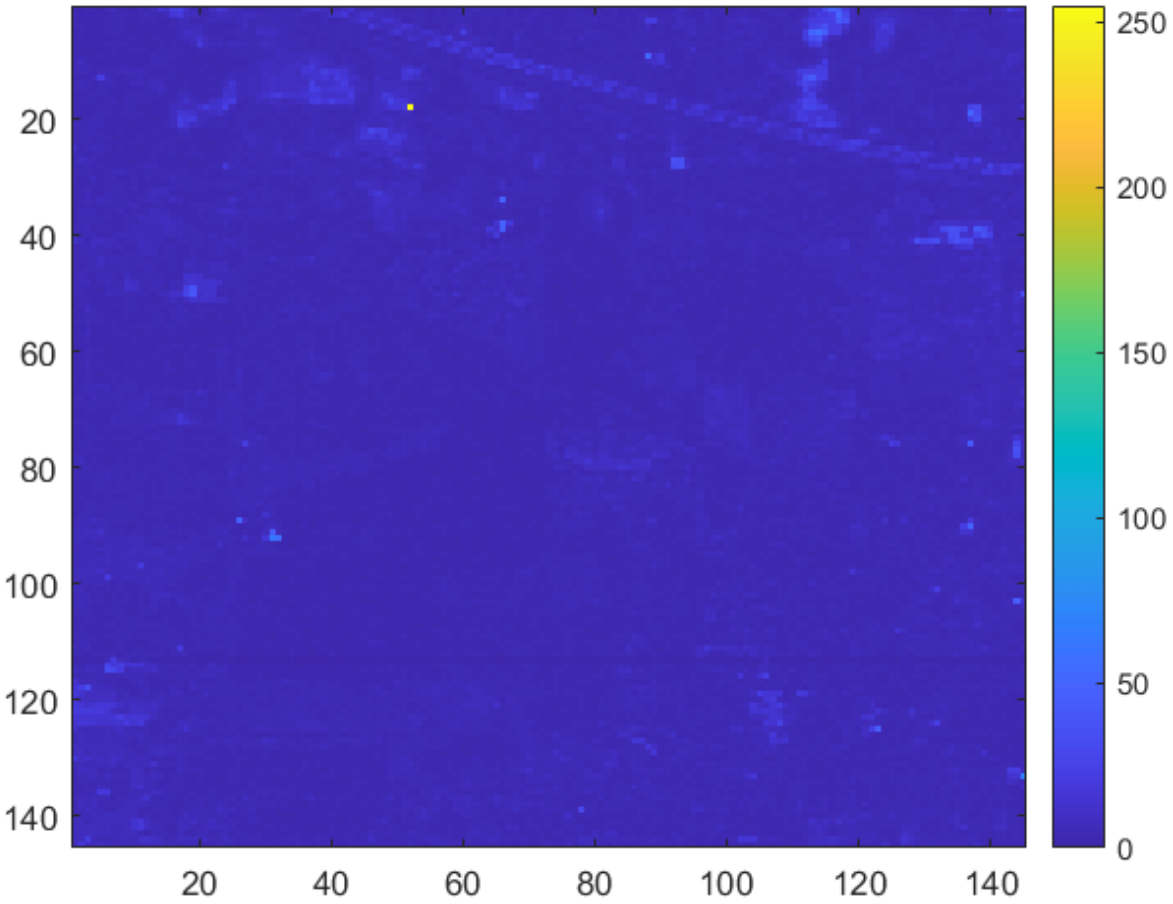
```
rxScore = anomalyRX(hcube);
```

Reduce the dynamic range of the RX score values by rescaling them to the range [0, 255].

```
rxScore = im2uint8(rescale(rxScore));
```

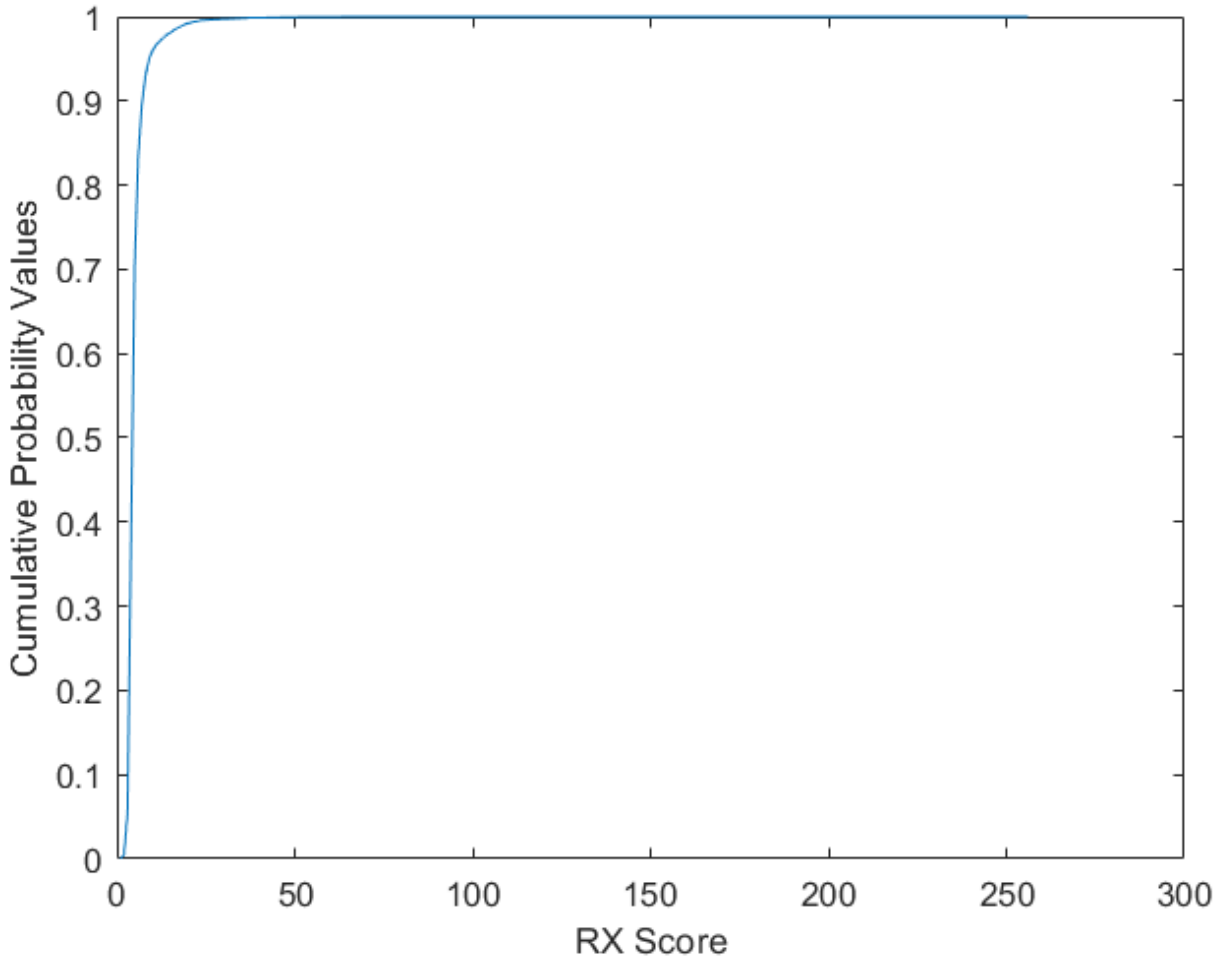
Display the RX score map. Pixels with a high RX score are likely anomalous pixels.

```
figure
imagesc(rxScore)
colorbar
```



Compute and plot the cumulative probability distribution of RX score values.

```
count = imhist(rxScore);
pdf = count/prod(size(rxScore,[1 2]));
cdf = cumsum(pdf(:));
figure
plot(cdf)
xlabel('RX Score')
ylabel('Cumulative Probability Values')
```



Set the confidence coefficient value to 0.998. Select the first RX score with cumulative probability distribution value greater than the confidence coefficient as the threshold. This threshold represents the RX score above which a pixel is an anomaly with 99.8 percent confidence.

```
confCoefficient = 0.998;
rxThreshold = find(cdf > confCoefficient,1);
```

Apply thresholding to detect anomalous pixels with RX score greater than the computed threshold. The result is a binary image in which the anomalous pixels are assigned the intensity value 1 and other pixels are assigned 0.

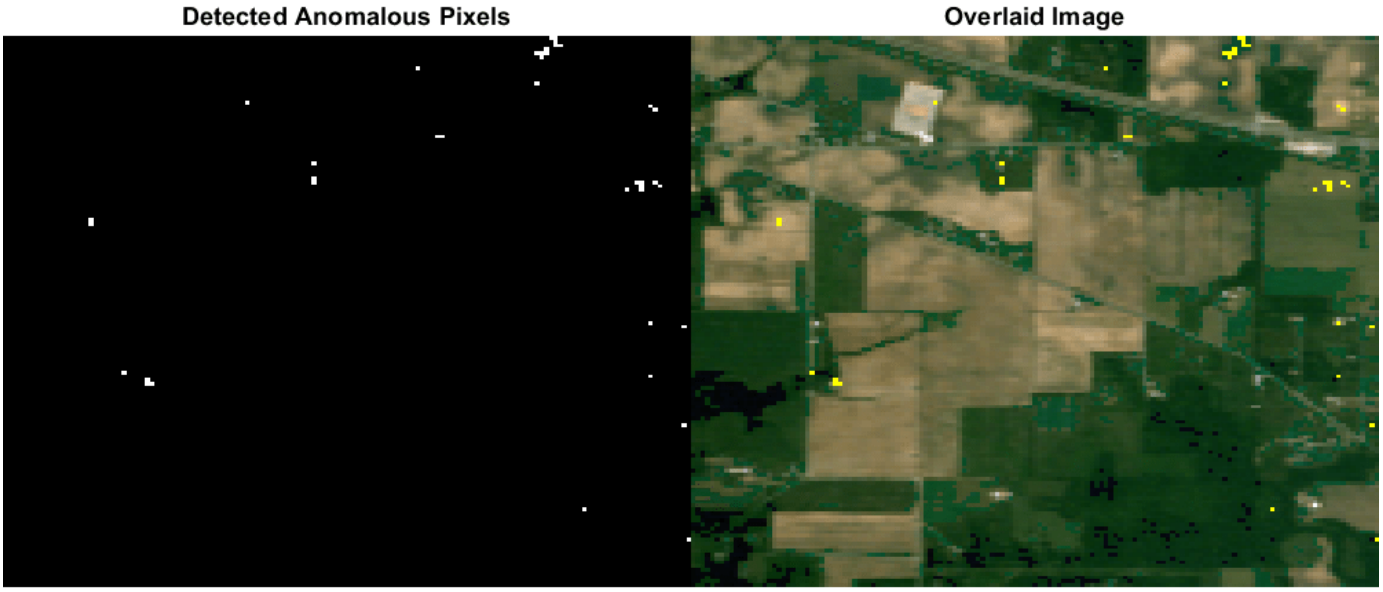
```
bw = rxScore > rxThreshold;
```

Derive the RGB version of the data cube by using the `colorize` function. Overlay the binary image of anomalous pixels on the RGB image.

```
rgbImg  = colorize(hcube,'Method','rgb');
B = imoverlay(rgbImg,bw);
```

Display both the binary image and the overlaid image.

```
fig = figure('Position',[0 0 800 400]);
axes1 = axes('Parent',fig,'Position',[0 0.1 0.5 0.8]);
imagesc(bw,'Parent',axes1);
title('Detected Anomalous Pixels')
axis off
colormap gray
axes2 = axes('Parent',fig,'Position',[0.5 0.1 0.5 0.8]);
imagesc(B,'Parent',axes2)
title('Overlaid Image');
axis off
```



Input Arguments

`inputData` — Input hyperspectral data

3-D numeric array | `hypercube` object

Input hyperspectral data, specified as a 3-D numeric array or `hypercube` object. If the input is an 3-D numeric array of size *M*-by-*N*-by-*C*, the function reads it as a hyperspectral data cube of *M*-by-*N* pixels with *C* spectral bands and computes the RX score. If the input is a `hypercube` object, the function reads the data cube stored in the `DataCube` property and then computes the RX score. The hyperspectral data cube must be real and non-sparse.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`rxScore` — Output RX score

matrix

Output RX score for each pixel in the hyperspectral data cube, returned as a matrix of size *M*-by-*N*, same as the spatial dimensions of the input data.

Data Types: `double`

Algorithms

The RX score for each pixel is computed as

r is the pixel under test and μ_C and Σ_C are the spectral mean and covariance respectively. Anomalous pixels typically have the high RX scores.

You can estimate a threshold from the cumulative probability distribution of the RX scores to further tune the anomalous pixel detection. See the “Detect Anomalous Pixels in Hyperspectral Data Using RX Detector” on page 1-3203 example.

References

[1] Reed, I.S., and X. Yu. “Adaptive Multiple-Band CFAR Detection of an Optical Pattern with Unknown Spectral Distribution.” *IEEE Transactions on Acoustics, Speech, and Signal Processing* 38, no. 10 (October 1990): 1760–70. https://doi.org/10.1109/29.60107.

[2] Chein-I Chang and Shao-Shan Chiang. “Anomaly Detection and Classification for Hyperspectral Imagery.” *IEEE Transactions on Geoscience and Remote Sensing* 40, no. 6 (June 2002): 1314–25. https://doi.org/10.1109/TGRS.2002.800280.

See Also

`hypercube` | `spectralMatch` | `ndvi`

Introduced in R2020a

assignData

Assign new data to hyperspectral data cube

Syntax

```
newhcube = assignData(hcube,row,column,band,data)
```

Description

`newhcube = assignData(hcube,row,column,band,data)` assigns the specified `data` to a hyperspectral data cube. The function reads the data cube stored in the `hypercube` object `hcube`, assigns the new data to the spectral bands `band` at the locations specified by `row` and `column`, and returns a new `hypercube` object.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Assign New Reflectance Values to Hyperspectral Data

Read hyperspectral data from an ENVI format file.

```
hcube = hypercube('paviaU.dat');
```

Normalize the reflectance values to the range [0, 1].

```
data = rescale(hcube.DataCube);
```

Assign the normalized reflectance values to the data cube.

```
newhcube = assignData(hcube,':',':',':',data);
```

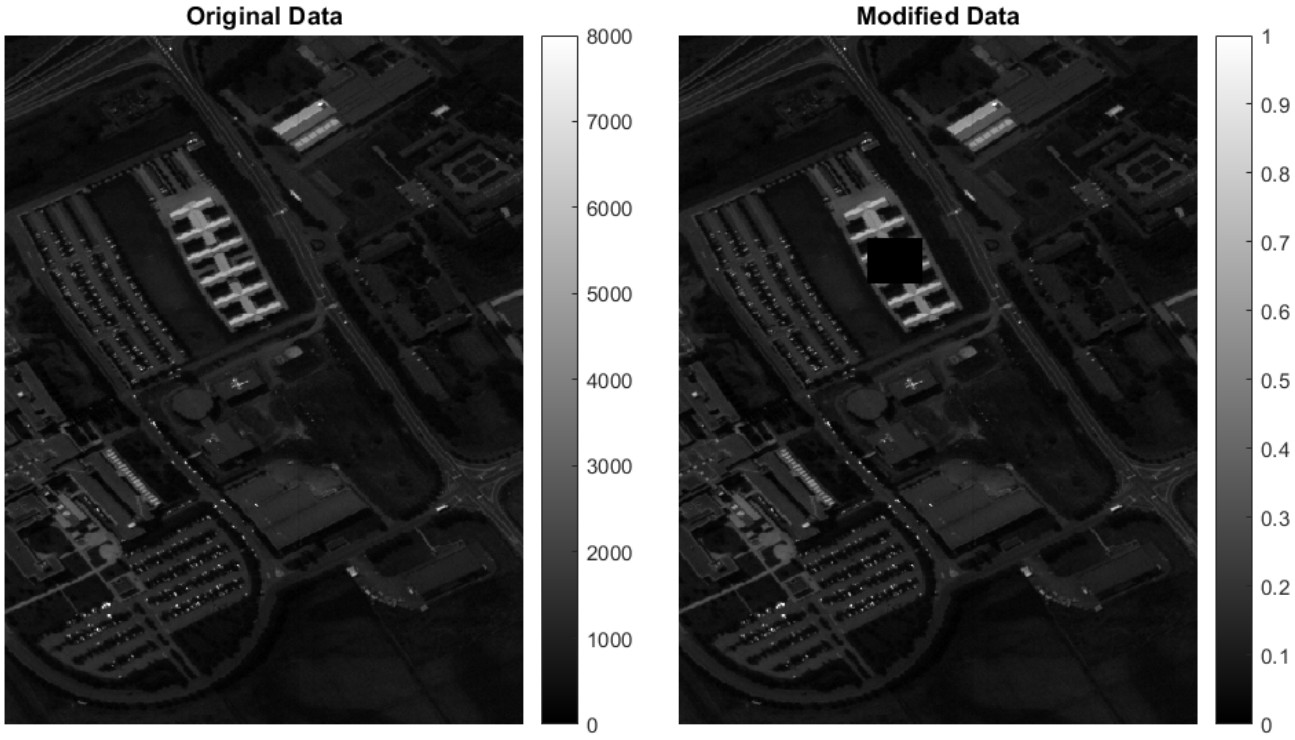
Specify the row and column indices of a region of interest (ROI). Assign all indices within the ROI a value of zero.

```
row = 180:220;
column = 125:160;
newhcube = assignData(newhcube,row,column,':',0);
```

Display the original and the modified versions of a spectral band.

```
fig = figure('Position',[0 0 800 500]);
axes1 = axes('Parent',fig,'Position',[0.06 0.05 0.45 0.8]);
imagesc(hcube.DataCube(:,:,10),'Parent',axes1);
title('Original Data')
colorbar
axis off
```

```
axes2 = axes('Parent',fig,'Position',[0.55 0.05 0.45 0.8]);
imagesc(newhcube.DataCube(:,:,10),'Parent',axes2);
title('Modified Data')
colorbar
axis off
colormap gray
```



Input Arguments

`hcube` — Input hyperspectral data

`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `DataCube` property of the `hypercube` object contains the hyperspectral data cube.

`row` — Row indices of data cube

`':'` | positive integer | vector of positive integers

Row indices of the data cube, specified as `':'`, a positive integer, or a vector of positive integers.

- To select all the rows in the data cube, use `':'`.
- To select a particular row or rows, specify the row index as a positive integer or vector of positive integers respectively. If the data cube is of size *M*-by-*N*-by-*C*, the specified row index values must all be less than or equal to *M*. To specify a range of row indices, or indices at a regular interval, use the `colon` operator. For example, `row = 1:10`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

`column` — Column indices of data cube

`':'` | positive integer | vector of positive integers

Column indices of the data cube, specified as `':'`, a positive integer, or a vector of positive integers.

- To select all the columns in the data cube, use `':'`.
- To select a particular column or columns, specify the column index as a positive integer or vector of positive integers respectively. If the data cube is of size *M*-by-*N*-by-*C*, the specified column index values must all be less than or equal to *N*. To specify a range of column indices, or indices at a regular interval, use the `colon` operator. For example, `column = 1:10`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

`band` — Spectral band numbers

`':'` | positive integer | vector of positive integers

Spectral band numbers, specified as `':'`, a positive integer or a vector of positive integers.

- To select all the bands in the data cube, use `':'`.
- To select a particular band or bands, specify the band number as a positive integer or vector of positive integers respectively. If the data cube is of size *M*-by-*N*-by-*C*, the specified band number values must all be less than or equal to *C*. To specify a range of band numbers or numbers at a regular interval, use the `colon` operator. For example, `band = 1:10`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

`data` — Values to assign

scalar | vector | matrix | 3-D array

Values to assign, specified as a scalar, vector, matrix, or 3-D array depending on the values of the `row`, `column`, and `band` inputs.

If `row` is	If `column` is	If `band` is	`data` must be
scalar	scalar	scalar	scalar
M- element vector	scalar	scalar	*M*- element row vector or *M*-by-1 matrix or *M*-by-1-by-1 array
scalar	*N*-element vector	scalar	*N*- element column vector or 1-by-*N* matrix or 1-by-*N*-by-1 array
scalar	scalar	*C*-element vector	1-by-1-by-*C* array
M- element vector	*N*-element vector	scalar	*M*-by-*N* matrix or *M*-by-*N*-by-1 array
M- element vector	*N*-element vector	*C*-element vector	*M*-by-*N*-by-*C* array

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

newhcube — Output hyperspectral data
`hypercube` object

Output hyperspectral data, returned as a `hypercube` object.

See Also

`hypercube` | `removeBands` | `selectBands` | `cropData`

Introduced in R2020a

countEndmembersHFC

Find number of endmembers

Syntax

```
numEndmembers = countEndmembersHFC(inputData)
numEndmembers = countEndmembersHFC(inputData,Name,Value)
```

Description

`numEndmembers = countEndmembersHFC(inputData)` finds the number of endmembers present in a hyperspectral data cube by using the noise-whitened Harsanyi–Farrand–Chang (NWHFC) method.

`numEndmembers = countEndmembersHFC(inputData,Name,Value)` specifies additional options using one or more name-value pair arguments. For example, `'NoiseWhiten',false` does not perform noise-whitening of the data before extracting the endmembers.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Find Number of Endmembers Using NWHFC Method

Read hyperspectral data into the workspace.

```
hcube = hypercube('jasperRidge2_R198.hdr');
```

Find the number of endmembers in the hyperspectral data by using the NWHFC method.

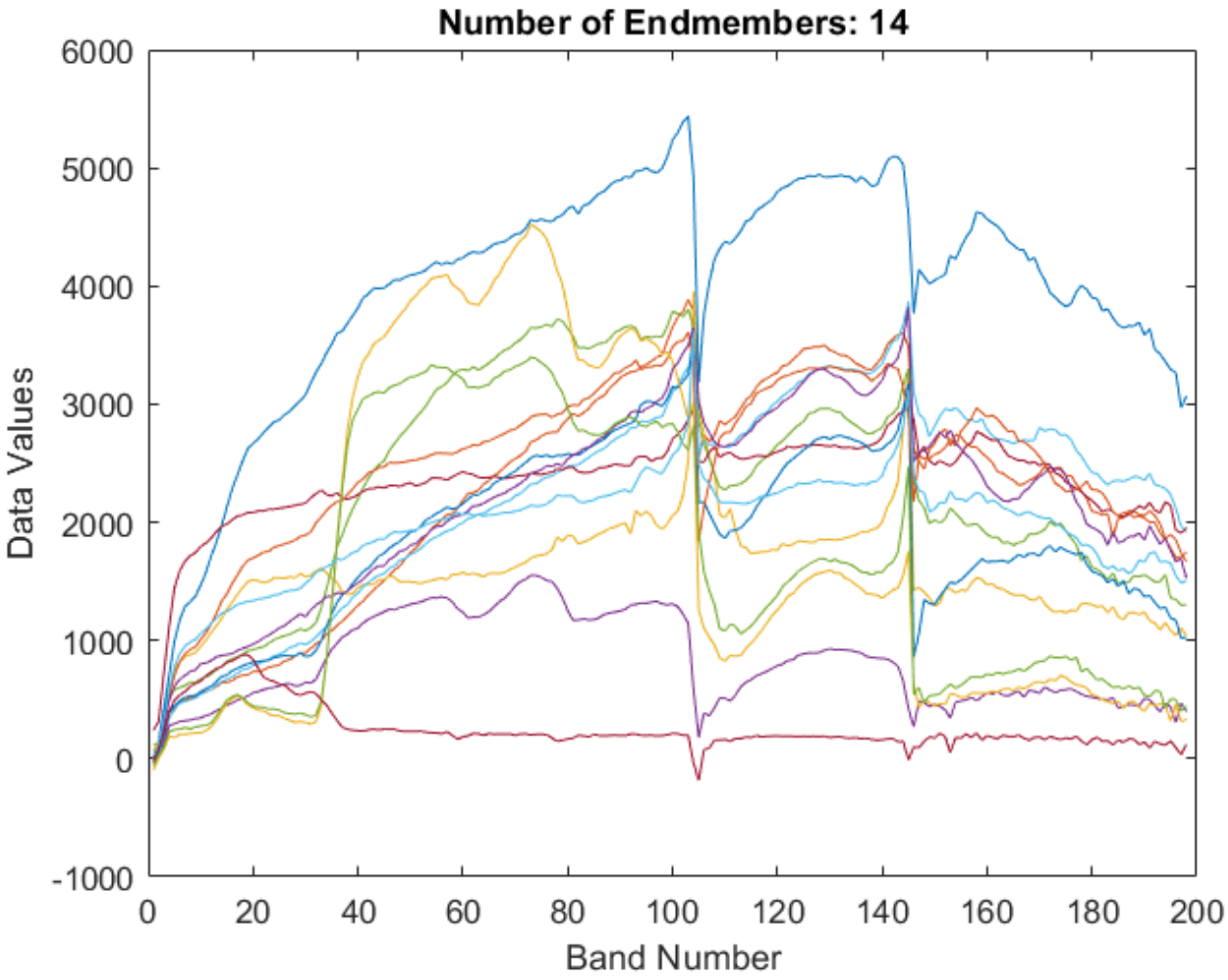
```
numEndmembers = countEndmembersHFC(hcube);
```

Estimate the endmember spectra using the N-FINDR method.

```
endmembers = nfindr(hcube,numEndmembers);
```

Plot the endmember spectra.

```
figure
plot(endmembers)
title(['Number of Endmembers: ' num2str(numEndmembers)])
xlabel('Band Number')
ylabel('Data Values')
```



Find Number of Endmembers Using HFC Method

Read hyperspectral data into the workspace.

```
hcube = hypercube('jasperRidge2_R198.hdr');
```

Find the number of endmembers in the hyperspectral data by using the HFC method. To use the HFC method, set the `'NoiseWhiten'` parameter value to `false`.

```
numEndmembers = countEndmembersHFC(hcube,'NoiseWhiten',false);
```

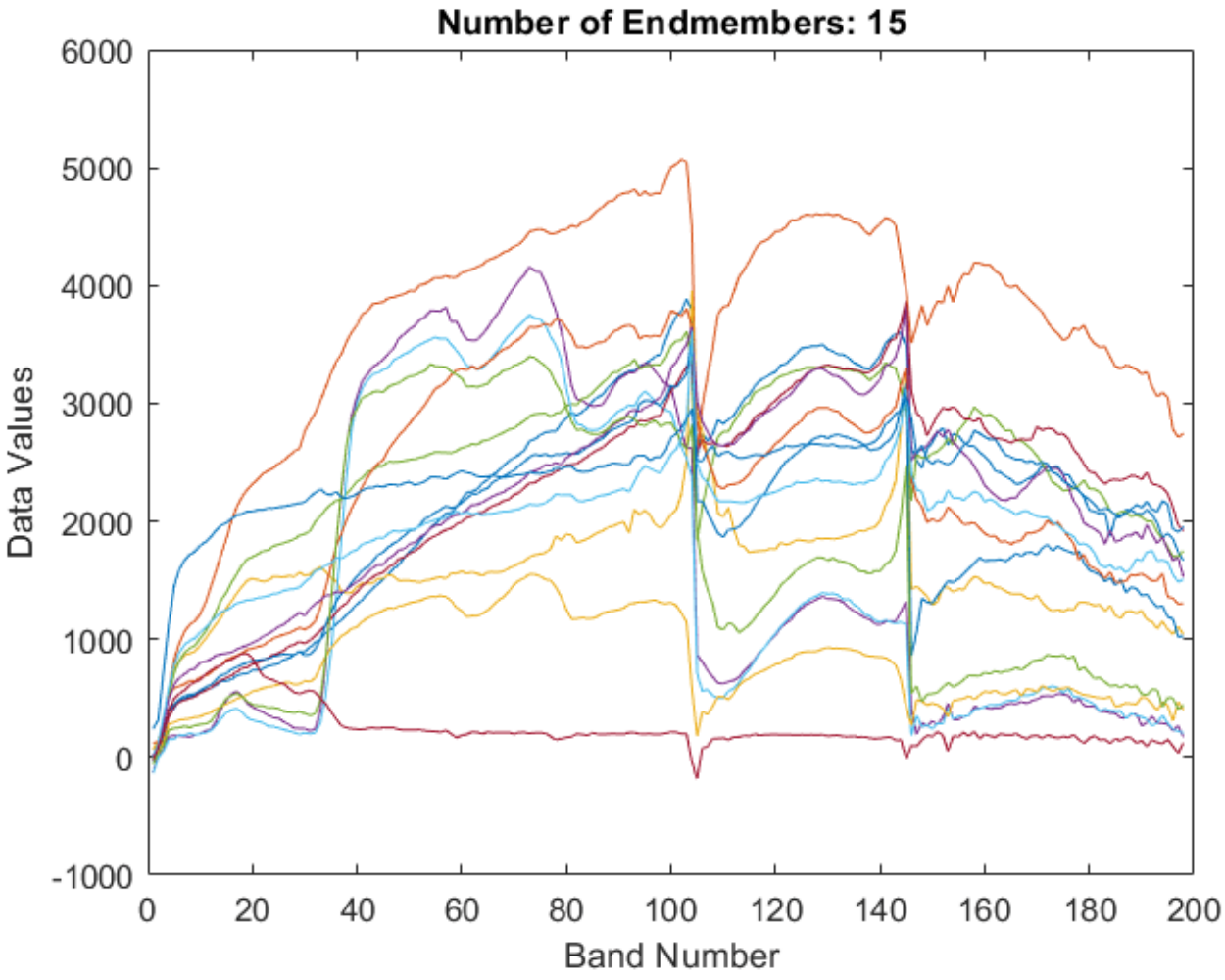
Estimate the endmember spectra using the N-FINDR method.

```
endmembers = nfindr(hcube,numEndmembers);
```

Plot the endmember spectra.

```
figure
plot(endmembers)
```

```
title(['Number of Endmembers: ' num2str(numEndmembers)])
xlabel('Band Number')
ylabel('Data Values')
```



Input Arguments

`inputData` — Input hyperspectral data

3-D numeric array | `hypercube` object

Input hyperspectral data, specified as a 3-D numeric array that represent the hyperspectral data cube of size *M*-by-*N*-by-*C* or `hypercube` object. If the input is a `hypercube` object, the function reads the data cube stored in the `DataCube` property of the object. The hyperspectral data cube must be real and non-sparse.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `countEndmembersHFC(inputData,'NoiseWhiten',false)`

`PFA` — Probability of false alarm

`10^-3` (default) | positive scalar

Probability of false alarm, specified as the comma-separated pair consisting of `'PFA'` and a positive scalar in the range (0, 1].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`NoiseWhiten` — Perform noise whitening

`true` or `1` (default) | `false` or `0`

Perform noise-whitening, specified as the comma-separated pair consisting of `'NoiseWhiten'` and a numeric or logical `1` (`true`) or `0` (`false`).

- `true` or `1` — Perform noise-whitening of input data before computing the number of endmembers. This approach is the NWHFC method.
- `false` or `0` — Do not perform noise-whitening of input data before computing the number of endmembers. This approach is the Harsanyi–Farrand–Chang (HFC) method.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

`numEndmembers` — Number of endmembers in hyperspectral data

positive numeric scalar

Number of endmembers in the hyperspectral data, returned as a positive numeric scalar.

Data Types: `double`

References

[1] Chang, C.-I., and Q. Du. “Estimation of Number of Spectrally Distinct Signal Sources in Hyperspectral Imagery.” *IEEE Transactions on Geoscience and Remote Sensing* 42, no. 3 (March 2004): 608–19. https://doi.org/10.1109/TGRS.2003.819189.

See Also

`hypercube` | `fippi` | `ppi` | `nfindr` | `estimateAbundanceLS`

Introduced in R2020a

colorize

Estimate color image of hyperspectral data

Syntax

```
coloredImage = colorize(hcube)
coloredImage = colorize(hcube,band)
[coloredImage,indices] = colorize( ___ )
 ___  = colorize( ___ ,Name,Value)
```

Description

`coloredImage = colorize(hcube)` estimates a false-color image of hyperspectral data based on the three most informative bands of the `hypercube` object `hcube`.

`coloredImage = colorize(hcube,band)` returns a false-color image using the specified spectral bands `band`.

`[coloredImage,indices] = colorize( ___ )` returns the indices of the bands used in the color image.

` ___  = colorize( ___ ,Name,Value)` specifies options using one or more name-value pair arguments in addition to any combination of arguments from previous syntaxes. Use this syntax to specify the options to estimate false-colored and color-infrared (CIR) images of the input data.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Estimate False-Color Image of Hyperspectral Data

Read a hyperspectral data into the workspace.

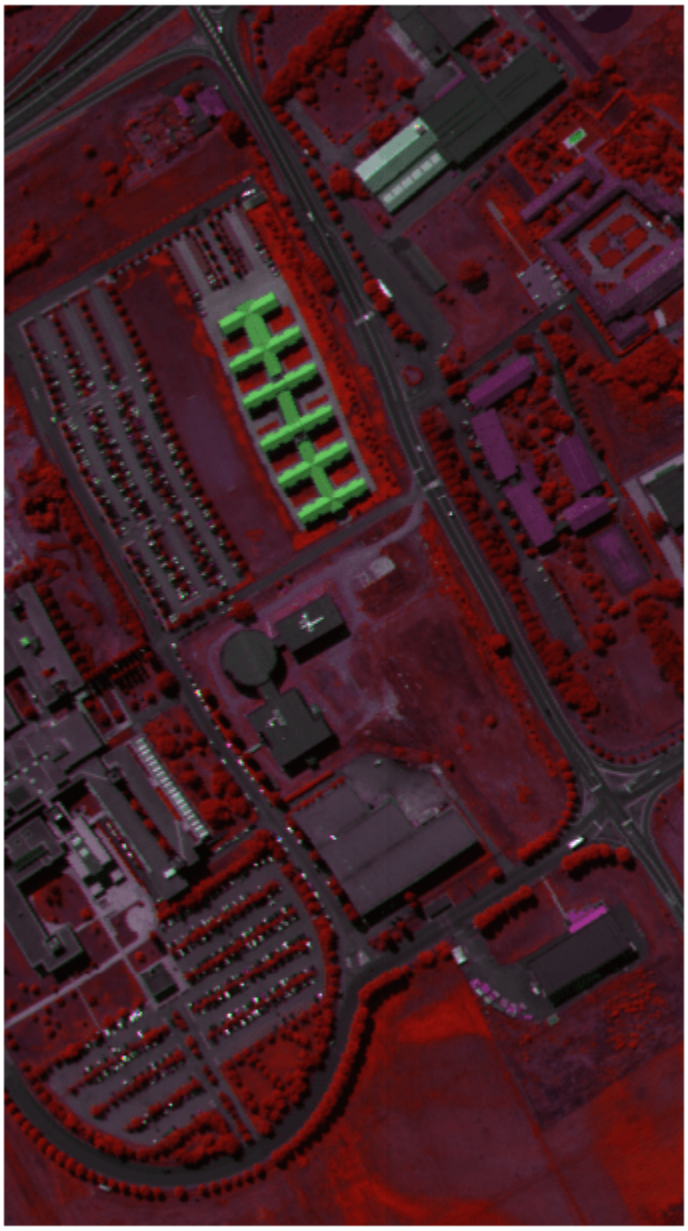
```
hcube = hypercube('paviaU.dat');
```

Estimate a false-color image of the hyperspectral data.

```
coloredImg = colorize(hcube);
```

Display the false-color image.

```
imshow(coloredImg)
```



Create Contrast-Stretched RGB Image of Hyperspectral Data

Read a hyperspectral data into the workspace.

```
hcube = hypercube('paviaU.dat');
```

Estimate an RGB image of the hyperspectral data. Increase the image contrast by applying contrast stretching.

```
coloredImg = colorize(hcube,"Method","rgb","ContrastStretching",true);
```

Display the contrast-stretched RGB image.

```
imshow(coloredImg)
```

Input Arguments

hcube — Input hyperspectral data
`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `DataCube` property of the `hypercube` object stores the hyperspectral data cube as an *M*-by-*N*-by-*C* numeric array, where *C* is the number of bands.

band — Spectral band numbers
3-element vector of positive integers

Spectral band numbers, specified as a 3-element vector of positive integers. All elements of the vector must be less than or equal to the total number of bands *C* in the input data.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `colorize(hcube,'Method','rgb')`

`Method` — Method used to visualize bands
`'falsecolored'` (default) | `'rgb'` | `'cir'`

Method used to visualize the bands, specified as the comma-separated pair consisting of `'Method'` and one of these options.

- `'falsecolored'` — Create a false-color image consisting of the three most informative bands selected using `selectBands` function.
- `'rgb'` — Create an RGB image by dividing the spectral range into red (R), green (G), and blue (B) bands. The red band ranges from 600 nm to 700 nm, the green band ranges from 500 nm to 600 nm, and the blue band ranges from 400 nm to 500 nm. The displayed R, G, and B channels consist of the most representative bands within the corresponding spectral range based on the correlation coefficient metric.
- `'cir'` — Create a color-infrared (CIR) image by dividing the spectral range into near infrared (NIR), R, and G bands. The NIR band ranges from 760 nm to 960 nm, the red band ranges from 600 nm to 700 nm, and the green band ranges from 500 nm to 600 nm. The displayed channels consist of the most representative bands within the corresponding spectral range based on the correlation coefficient metric.

To create RGB or CIR images, the `Wavelength` property of the `hypercube` object `hcube` must have wavelengths in each of the corresponding ranges.

Data Types: `char` | `string`

`ContrastStretching` — Perform contrast stretching of image
`0` (`false`) (default) | `1` (`true`)

Perform contrast stretching of the image, specified as the comma-separated pair consisting of `'ContrastStretching'` and the logical `0` (`false`) or `1` (`true`). When `true`, the `colorize` function applies contrast-limited adaptive histogram equalization, using the `adapthisteq` function.

Data Types: `logical`

Output Arguments

`coloredImage` — Color image

M-by-*N*-by-3 numeric array

Color image, returned as an *M*-by-*N*-by-3 numeric array. Each of the three color planes contains one band of the hyperspectral image.

Data Types: `single` | `double`

`indices` — Indices of selected bands

3-element column vector of positive integers

Indices of the selected bands, returned as a 3-element column vector of positive integers.

Data Types: `double`

See Also

`hypercube` | `selectBands`

Introduced in R2020a

correctOOB

Correct out-of-band effect using sensor spectral response

Syntax

```
newhcube = correctOOB(hcube,spectralResponse)
[newhcube,oobEffect] = correctOOB(hcube,spectralResponse)
[___] = correctOOB(hcube,spectralResponse,'RegionMask',mask)
newhcube = correctOOB(___,'BlockSize',blocksize)
```

Description

`newhcube = correctOOB(hcube,spectralResponse)` corrects the out-of-band (OOB) effect in the input satellite data by using the sensor's spectral response characteristics. This method is suitable for OOB correction in multispectral satellite data.

Use this function to correct OOB effects over different regions such as clear waters, turbid waters, green vegetation, sand, and soil. This method gives best results if the input data is compensated for Rayleigh and aerosol scattering. To measure the OOB effect on scenes with large water bodies, you must first compute the water leaving radiance spectra from the input satellite data.

`[newhcube,oobEffect] = correctOOB(hcube,spectralResponse)` also returns the relative OOB effect for each spectral band.

`[___] = correctOOB(hcube,spectralResponse,'RegionMask',mask)` specifies region mask by using the name-value pair argument `'RegionMask'`. The region mask indicates the homogeneous regions in the input satellite data.

`newhcube = correctOOB(___,'BlockSize',blocksize)` specifies the block size for block processing of the hyperspectral data cube by using the name-value pair argument `'BlockSize'`. You can specify the `'BlockSize'` name-value pair argument in addition to the input arguments in the previous syntaxes.

The function divides the input image into distinct blocks, processes each block, and then concatenates the processed output of each block to form the output matrix. Hyperspectral images are multi-dimensional data sets that can be too large to fit in system memory in their entirety. This can cause the system to run out of memory while running the `correctOOB` function. If you encounter such an issue, perform block processing by using this syntax.

For example, `correctOOB(hcube,spectralResponse,'BlockSize',[50 50])` divides the input image into non-overlapping blocks of size 50-by-50 and then performs out-of-band correction on each block.

Note To perform block processing by specifying the `'BlockSize'` name-value pair argument, you must have MATLAB R2021a or a later release.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Perform OOB Correction Using Clear-Water Pixels

Read multispectral data into the workspace.

```
hcube = hypercube('LC08_L1TP_097070_20201101_20201101_01_cropped.dat');
```

Convert the pixel values from digital numbers to top of atmosphere (TOA) radiance values.

```
hcube = dn2radiance(hcube);
```

Estimate remotely sensed water reflectance.

```
[rrshcube,mask] = rrs(hcube);
```

Read the sensor spectral response.

```
spectralResponse = readtable('spectralResponse.txt');
spectralResponse = table2array(spectralResponse);
```

Perform OOB correction using the clear-water pixels specified by the region mask.

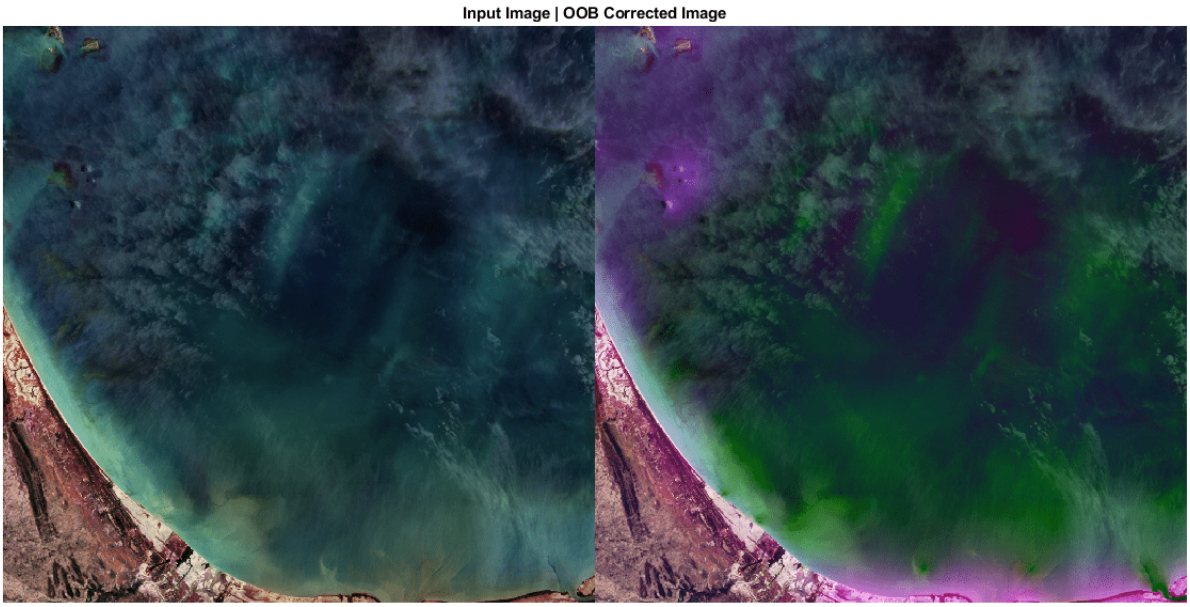
```
[newhcube,oobEffect] = correctOOB(rrshcube,spectralResponse,'RegionMask',mask);
```

Estimate RGB images of the input and the OOB corrected output data. Increase the image contrast by applying contrast stretching.

```
imgIn = colorize(hcube,'Method','rgb','ContrastStretching',true);
imgOut = colorize(newhcube,'Method','rgb','ContrastStretching',true);
```

Display the input and the OOB corrected output images.

```
figure
montage({imgIn,imgOut})
title('Input Image | OOB Corrected Image')
```



Input Arguments

`hcube` — Input satellite data

`hypercube` object

Input satellite data, specified as a `hypercube` object. The functions reads the data cube from the `DataCube` property of the object. The data cube is of size *M*-by-*N*-by-*C*. *C* is the number of spectral bands in the input satellite data.

`spectralResponse` — Sensor spectral response

matrix | table

Sensor spectral response, specified as a matrix or a table. The size of the matrix or the table must be *K*-by-*C*+1. The first column of the matrix or table contains the wavelength values and the spectral resolution is 1 nm.

You can download the spectral response for different sensors from https://oceancolor.gsfc.nasa.gov/docs/rsr/rsr_tables/.

`mask` — Region mask indicating homogeneous regions

matrix

Region mask indicating homogeneous regions, specified as a matrix of size *M*-by-*N*. The region mask is a binary image with intensity values 0 and 1. The regions with intensity value 1 corresponds to the homogeneous regions in the input satellite data.

Example: `correctOOB(hcube,spectralResponse,'RegionMask',mask)`

Data Types: `logical`

`blocksize` — Size of data blocks
2-element vector of positive integers

Size of the data blocks, specified as a 2-element vector of positive integers. The elements of the vector correspond to the number of rows and columns in each block, respectively. The size of the data blocks must be less than the size of the input image. Dividing the hyperspectral images into smaller blocks enables you process large data sets without running out of memory.

- If the `blocksize` value is too small, the memory usage of the function reduces at the cost of increased execution time.
- If the `blocksize` value is large or equal to the input image size, the execution time reduces at the cost of increased memory usage.

Example: `'BlockSize',[20 20]` specifies the size of each data block as 20-by-20.

Output Arguments

`newhcube` — Out-of-band corrected data
`hypercube`

Out-of-band corrected data, returned as a `hypercube`.

`oobEffect` — Out-of-band effect for each band
C-element vector

Out-of-band effect for each band, returned as a *C*-element vector. The out-of-band effect for each spectral band is measured as the relative difference between the values of homogeneous region pixels in the input data `hcube` and the corrected data `newhcube`.

See Also
`dn2radiance` | `fastInScene` | `hypercube` | `rrs`

Introduced in R2020b

cropData

Crop regions-of-interest

Syntax

```
newhcube = cropData(hcube,row,column)
newhcube = cropData(hcube,row,column,band)
```

Description

`newhcube = cropData(hcube,row,column)` crops the regions of interest (ROIs), specified by `row` and `column`, across all the spectral bands in the hyperspectral data cube `hcube`. The function returns the cropped data as a new `hypercube` object `newhcube`.

`newhcube = cropData(hcube,row,column,band)` crops the ROIs across the specified spectral bands `band`.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Crop ROI from Hyperspectral Data Cube

Read hyperspectral data from an ENVI format file.

```
hcube = hypercube('paviaU.dat');
```

Crop the first 10 spectral bands of the input data cube.

```
newhcube = cropData(hcube,':',':',1:10);
```

Specify the row and column indices of the ROI to crop from the extracted bands.

```
row = 130:250;
column = 60:200;
```

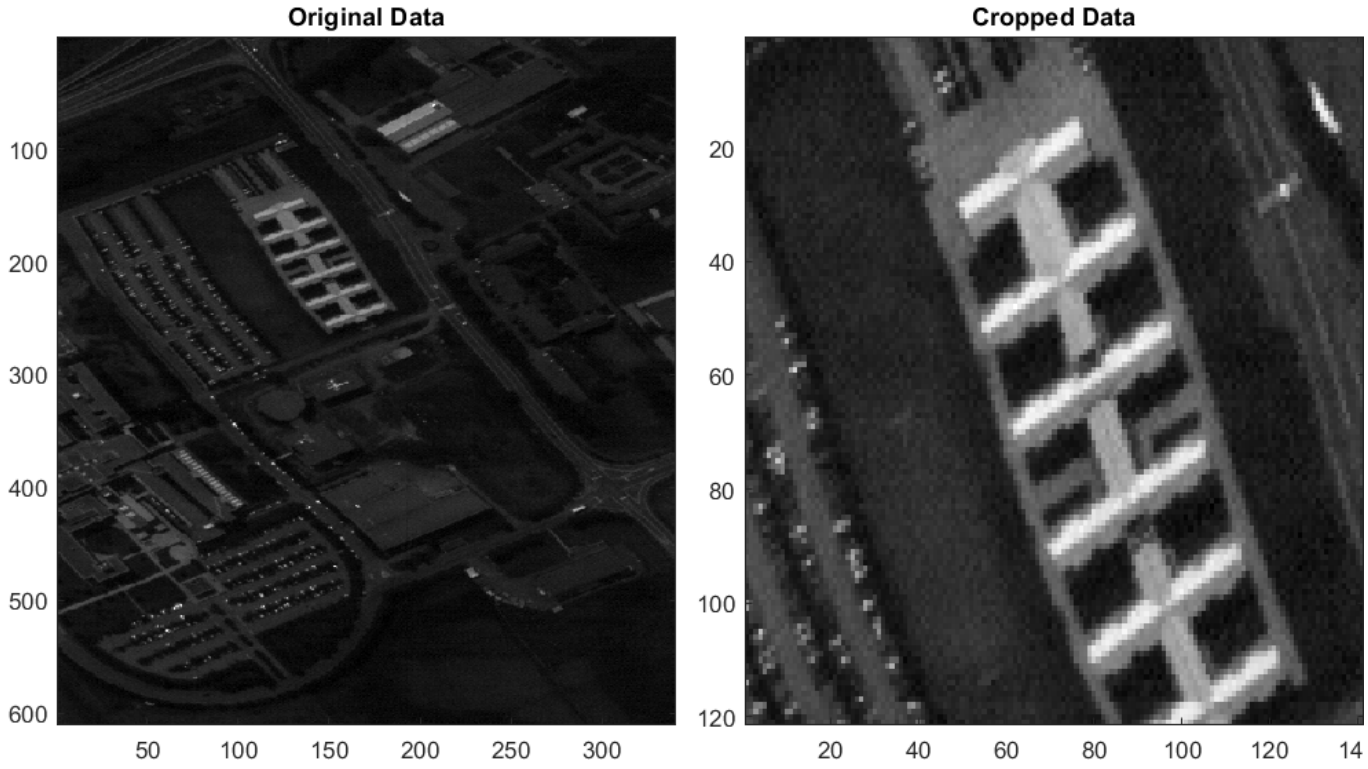
Crop the ROI.

```
newhcube = cropData(newhcube,row,column,':');
```

Display both bands in the original and the cropped versions of a spectral band.

```
fig = figure('Position',[0 0 800 500]);
axes1 = axes('Parent',fig,'Position',[0.05 0.05 0.45 0.8]);
imagesc(hcube.DataCube(:,:,5),'Parent',axes1)
title('Original Data')
axes2 = axes('Parent',fig,'Position',[0.55 0.05 0.45 0.8]);
```

```
imagesc(newhcube.DataCube(:,:,5),'Parent',axes2)
title('Cropped Data')
colormap gray
```



Input Arguments

`hcube` — Input hyperspectral data

`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `DataCube` property of the `hypercube` object contains the hyperspectral data cube.

`row` — Row indices of data cube

`':'` | positive integer | vector of positive integers

Row indices of the data cube, specified as `':'`, a positive integer, or a vector of positive integers.

- To select all the rows in the data cube, use `':'`.
- To select a particular row or rows, specify the row index as a positive integer or vector of positive integers respectively. If the data cube is of size *M*-by-*N*-by-*C*, the specified row index values must all be less than or equal to *M*. To specify a range of row indices, or indices at a regular interval, use the `colon` operator. For example, `row = 1:10`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

`column` — Column indices of data cube
`':'` | positive integer | vector of positive integers

Column indices of the data cube, specified as `':'`, a positive integer, or a vector of positive integers.

- To select all the columns in the data cube, use `':'`.
- To select a particular column or columns, specify the column index as a positive integer or vector of positive integers respectively. If the data cube is of size *M*-by-*N*-by-*C*, the specified column index values must all be less than or equal to *N*. To specify a range of column indices, or indices at a regular interval, use the `colon` operator. For example, `column = 1:10`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

`band` — Spectral band numbers
`':'` | positive integer | vector of positive integers

Spectral band numbers, specified as `':'`, a positive integer or a vector of positive integers.

- To select all the bands in the data cube, use `':'`.
- To select a particular band or bands, specify the band number as a positive integer or vector of positive integers respectively. If the data cube is of size *M*-by-*N*-by-*C*, the specified band number values must all be less than or equal to *C*. To specify a range of band numbers or numbers at a regular interval, use the `colon` operator. For example, `band = 1:10`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `char` | `string`

Output Arguments

`newhcube` — Output hyperspectral data
`hypercube` object

Output hyperspectral data, returned as a `hypercube` object.

See Also
`hypercube` | `removeBands` | `selectBands` | `assignData`

Introduced in R2020a

denoiseNGMeet

Denoise hyperspectral images using non-local meets global approach

Syntax

```
outputData = denoiseNGMeet(inputData)
outputData = denoiseNGMeet(inputData,Name,Value)
```

Description

`outputData = denoiseNGMeet(inputData)` reduces noise in hyperspectral data by using the non-local meets global (NGMeet) approach. This is an iterative approach that integrates both the spatial non-local similarity and spectral low-rank approximation for estimating the original pixel values. For more information, see “Algorithms” on page 1-3232.

`outputData = denoiseNGMeet(inputData,Name,Value)` also specifies options using one or more name-value pair arguments. Use this syntax to set the parameter values for NGMeet approach.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Denoise Hyperspectral Data Using NGmeet Method

Read hyperspectral data into the workspace.

```
hcube = hypercube('paviaU.hdr');
```

Normalize the input data and add Gaussian noise to the normalized input data.

```
hcube = hypercube(rescale(hcube.DataCube),hcube.Wavelength);
inputData = imnoise(hcube.DataCube,'Gaussian',0,0.005);
inputData = assignData(hcube,':',':',':',inputData);
```

Denoise the noisy hyperspectral data using NGmeet method.

```
outputData = denoiseNGMeet(inputData);
```

Estimate RGB images for the input, noisy, and the denoised output datacube. Increase the image contrast by applying contrast stretching.

```
originalImg = colorize(hcube,'Method','rgb','ContrastStretching',true);
noisyImg = colorize(inputData,'Method','rgb','ContrastStretching',true);
denoisedImg = colorize(outputData,'Method','rgb','ContrastStretching',true);
```

Display RGB images of the original, noisy, and denoised data.

```
figure
montage({originalImg,noisyImg,denoisedImg})
title('Input Image | Noisy Image | Denoised Image');
```

Set NGMeet Parameters for Hyperspectral Denoising

Read hyperspectral data into the workspace.

```
hcube = hypercube('paviaU.hdr');
```

Normalize the input data and add Gaussian noise to the normalized input data.

```
hcube = hypercube(rescale(hcube.DataCube),hcube.Wavelength);
inputData = imnoise(hcube.DataCube,'Gaussian',0,0.005);
inputData = assignData(hcube,':',':',':',inputData);
```

Denoise the noisy hyperspectral data by using the NGmeet method. Set the smoothing parameter value to 0.01 and the number of iterations to 4.

```
outputData = denoiseNGMeet(inputData,'Sigma',0.01,'NumIterations',4);
```

Estimate RGB images for the input, noisy, and the denoised output datacube. Increase the image contrast by applying contrast stretching.

```
originalImg = colorize(hcube,'Method','rgb','ContrastStretching',true);
noisyImg = colorize(inputData,'Method','rgb','ContrastStretching',true);
denoisedImg = colorize(outputData,'Method','rgb','ContrastStretching',true);
```

Display RGB images of the original, noisy, and denoised data.

```
figure
montage({originalImg,noisyImg,denoisedImg})
title('Input Image | Noisy Image | Denoised Image');
```

Input Arguments

`inputData` — Input hyperspectral data

3-D numeric array | `hypercube` object

Input hyperspectral data, specified as a 3-D numeric array that represent the hyperspectral data cube of size *M*-by-*N*-by-*C* or `hypercube` object. If the input is a `hypercube` object, the function reads the data cube stored in the `DataCube` property of the object. The hyperspectral data cube must be real and non-sparse.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `denoiseNGMeet(hcube,'Sigma',0.3)`

`Sigma` — Smoothing parameter
$0.1\sigma_n$ (default) | positive scalar

Smoothing parameter, specified as the comma-separated pair consisting of `'Sigma'` and a positive scalar. The default value is 0.1 times the noise variance (σ_n). Increasing this value, increases the level of smoothing in the denoised output.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`SpectralSubspace` — Number of spectral bands in low-rank approximation
6 (default) | positive integer scalar

Number of spectral bands in low-rank approximation, specified as the comma-separated pair consisting of `'SpectralSubspace'` and a positive integer scalar in the range (0, C]. C is the number of bands in the input data. The number of endmembers in the hyperspectral data can be a good estimate for the number of spectral bands to use for low-rank approximation. You can find the number of endmembers in the input hyperspectral data by using the `countEndmembersHFC` function.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`NumIterations` — Number of iterations
2 (default) | positive integer scalar

Number of iterations, specified as the comma-separated pair consisting of `'NumIterations'` and a positive integer scalar. Increase this value for better denoising results.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`outputData` — Denoised hyperspectral data
3-D numeric array | `hypercube` object

Denoised hyperspectral data, returned as a 3-D numeric array or `hypercube` object.

Algorithms

The NGMeet method estimates the denoised data cube by using these steps. For each iteration, i

1. Compute spectral low-rank approximation of the noisy input data (Y_i) by using singular value decomposition. The approximation results in a reduced data cube (M_i) and the related orthogonal basis A_i.
2. Perform spatial denoising of the reduced data cube M_i by using non-local similarity filtering. You can control the degree of smoothing by specifying the smoothing parameter `'Sigma'`.
3. Perform inverse projection. Map the denoised reduced data cube M_i to original space by using the orthogonal basis A_i. The result is the denoised output (X_i) obtained at iteration i.
4. Perform iterative regularization. Update the noisy input data, $Y_{i+1} = \lambda X_i + (1-\lambda)Y_i$.
5. Repeat steps 1 to 4, for the specified number of iterations. The final value X_i is the denoised hyperspectral data.

References

[1] He, Wei, Quanming Yao, Chao Li, Naoto Yokoya, and Qibin Zhao. “Non-Local Meets Global: An Integrated Paradigm for Hyperspectral Denoising.” In *2019 IEEE/CVF Conference on*

Computer Vision and Pattern Recognition (CVPR), 6861–70. Long Beach, CA, USA: IEEE, 2019. https://doi.org/10.1109/CVPR.2019.00703.

See Also

`hypercube` | `countEndmembersHFC`

Introduced in R2020b

dn2radiance

Convert digital number to radiance

Syntax

```
newhcube = dn2radiance(hcube)
newhcube = dn2radiance(hcube,'BlockSize',blocksize)
```

Description

`newhcube = dn2radiance(hcube)` converts the pixel values of the hyperspectral data cube from digital number to radiance values. The function returns a new `hypercube` object and the pixel values of the data cube are the top of atmosphere (TOA) radiances.

`newhcube = dn2radiance(hcube,'BlockSize',blocksize)` specifies the block size for block processing of the hyperspectral data cube by using the name-value pair argument `'BlockSize'`.

The function divides the input image into distinct blocks, processes each block, and then concatenates the processed output of each block to form the output matrix. Hyperspectral images are multi-dimensional data sets that can be too large to fit in system memory in their entirety. This can cause the system to run out of memory while running the `dn2radiance` function. If you encounter such an issue, perform block processing by using this syntax.

For example, `dn2radiance(hcube,'BlockSize',[50 50])` divides the input image into non-overlapping blocks of size 50-by-50 and then computes the radiance values for pixels in each block.

Note To perform block processing by specifying the `'BlockSize'` name-value pair argument, you must have MATLAB R2021a or a later release.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Convert Digital Number to Radiance

Read hyperspectral data into the workspace.

```
hcube = hypercube('EO1H0440342002212110PY_cropped.hdr');
```

Determine the bad spectral band numbers using the `BadBands` parameter in the metadata.

```
bandNumber = find(~hcube.Metadata.BadBands);
```

Remove the bad spectral bands from the data cube.

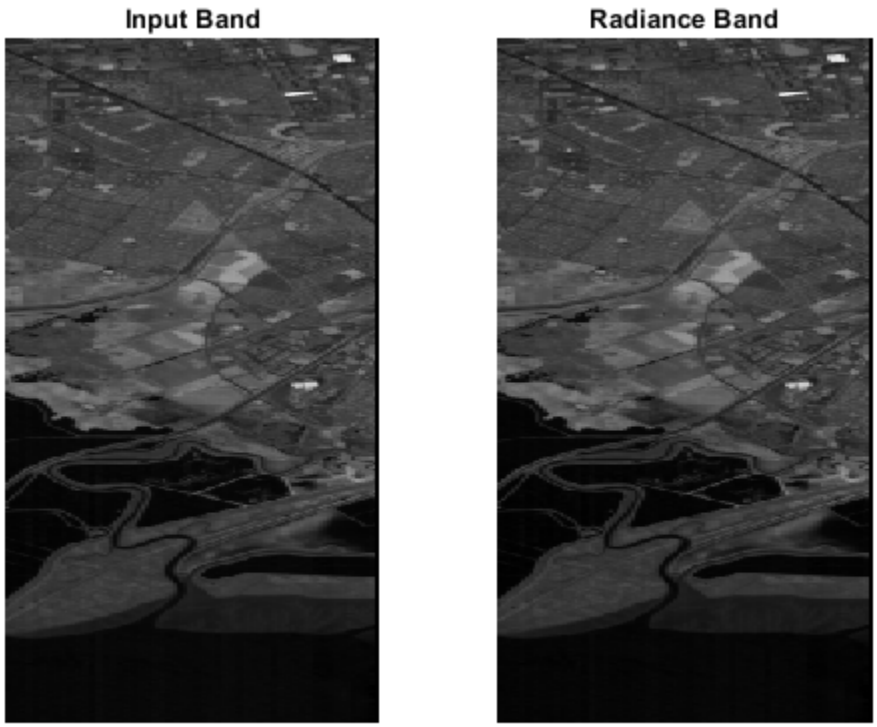
```
hcube = removeBands(hcube,'BandNumber',bandNumber);
```

Compute the radiance values using the `dn2radiance` function.

```
newhcube = dn2radiance(hcube);
```

Read and display a spectral band image in the input and the output radiance data.

```
inputBand = hcube.DataCube;
radianceBand = newhcube.DataCube;
band = 80;
figure
subplot(1,2,1)
imagesc(inputBand(:,:,band))
title('Input Band')
axis off
subplot(1,2,2)
imagesc(radianceBand(:,:,band))
title('Radiance Band')
axis off
colormap gray
```



Input Arguments

hcube — Input hyperspectral data

`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `DataCube` property of the `hypercube` object stores the hyperspectral data cube. To convert the pixel values in digital numbers

to radiance values, the `Metadata` property of the `hypercube` object must contain the `Gain` and `Offset` fields.

`blocksize` — Size of data blocks
2-element vector of positive integers

Size of the data blocks, specified as a 2-element vector of positive integers. The elements of the vector correspond to the number of rows and columns in each block, respectively. The size of the data blocks must be less than the size of the input image. Dividing the hyperspectral images into smaller blocks enables you process large data sets without running out of memory.

- If the `blocksize` value is too small, the memory usage of the function reduces at the cost of increased execution time.
- If the `blocksize` value is large or equal to the input image size, the execution time reduces at the cost of increased memory usage.

Example: `'BlockSize',[20 20]` specifies the size of each data block as 20-by-20.

Output Arguments

`newhcube` — Output hyperspectral data
`hypercube` object

Output hyperspectral data, returned as a `hypercube` object. The pixels values of the output data cube are radiances specifying the amount of radiation from the surface being imaged. The radiance values are computed from digital numbers by using the equation:

Gain and *Bias* are the gain and offset values for each spectral bands respectively. The `Metadata` property of `hypercube` object contains the gain and the offset values.

See Also
`dn2reflectance` | `radiance2Reflectance` | `empiricalLine` | `iarr` | `sharc` | `hypercube`

Introduced in R2020b

dn2reflectance

Convert digital number to reflectance

Syntax

```
newhcube = dn2reflectance(hcube)
newhcube = dn2reflectance(hcube,'BlockSize',blocksize)
```

Description

`newhcube = dn2reflectance(hcube)` converts the pixel values of the hyperspectral data cube from digital number (DN) to reflectance values. The function returns a new `hypercube` object and the pixel values of the data cube are the top of atmosphere (TOA) reflectance values. For details on TOA reflectance values, see “Compute TOA Reflectance values from DNs” on page 1-3239.

`newhcube = dn2reflectance(hcube,'BlockSize',blocksize)` specifies the block size for block processing of the hyperspectral data cube by using the name-value pair argument `'BlockSize'`.

The function divides the input image into distinct blocks, processes each block, and then concatenates the processed output of each block to form the output matrix. Hyperspectral images are multi-dimensional data sets that can be too large to fit in system memory in their entirety. This can cause the system to run out of memory while running the `dn2reflectance` function. If you encounter such an issue, perform block processing by using this syntax.

For example, `dn2reflectance(hcube,'BlockSize',[50 50])` divides the input image into non-overlapping blocks of size 50-by-50 and then computes the reflectance values for pixels in each block.

Note To perform block processing by specifying the `'BlockSize'` name-value pair argument, you must have MATLAB R2021a or a later release.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Convert Digital Number to Reflectance

Read hyperspectral data into the workspace.

```
hcube = hypercube('EO1H0440342002212110PY_cropped.hdr');
```

Determine the bad spectral band numbers using the `BadBands` parameter in the metadata.

```
bandNumber = find(~hcube.Metadata.BadBands);
```

Remove the bad spectral bands from the data cube.

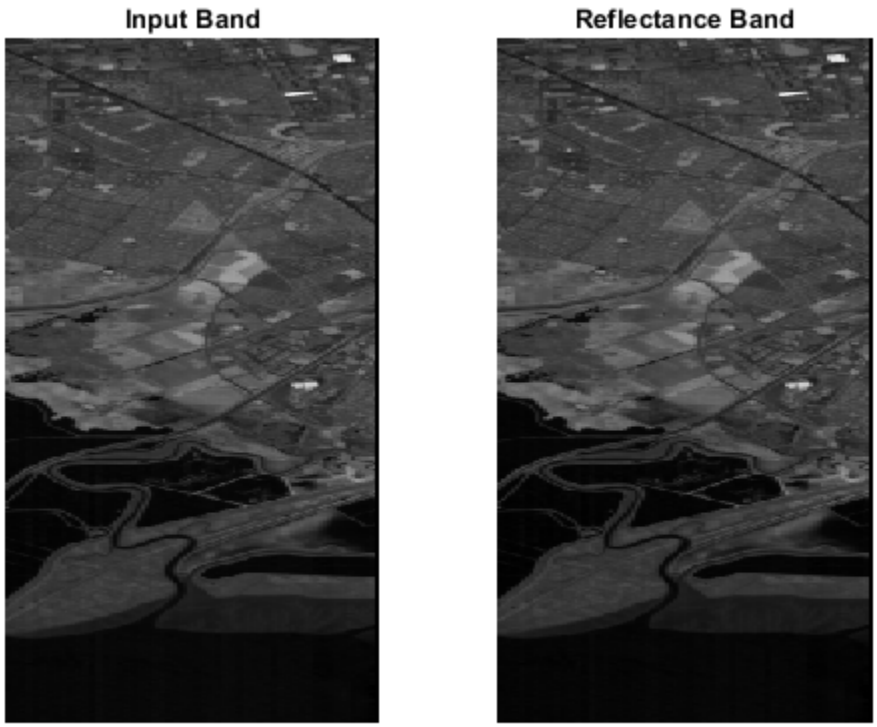
```
hcube = removeBands(hcube,'BandNumber',bandNumber);
```

Convert digital numbers to top of atmosphere (TOA) reflectances. The pixel values in the output data cube are the TOA reflectances.

```
newhcube = dn2reflectance(hcube);
```

Read and display the 80th spectral band image in the input and the output reflectance data cubes.

```
inputBand = hcube.DataCube;
reflectanceBand = newhcube.DataCube;
band = 80;
figure
subplot(1,2,1)
imagesc(inputBand(:,:,band))
title('Input Band')
axis off
subplot(1,2,2)
imagesc(reflectanceBand(:,:,band))
title('Reflectance Band')
axis off
colormap gray
```



Input Arguments

`hcube` — Input hyperspectral data

`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `DataCube` property of the `hypercube` object stores the hyperspectral data cube. The `MetaData` property of the `hypercube` object must contain reflectance gain values.

`blocksize` — Size of data blocks

2-element vector of positive integers

Size of the data blocks, specified as a 2-element vector of positive integers. The elements of the vector correspond to the number of rows and columns in each block, respectively. The size of the data blocks must be less than the size of the input image. Dividing the hyperspectral images into smaller blocks enables you process large data sets without running out of memory.

- If the `blocksize` value is too small, the memory usage of the function reduces at the cost of increased execution time.
- If the `blocksize` value is large or equal to the input image size, the execution time reduces at the cost of increased memory usage.

Example: `'BlockSize',[20 20]` specifies the size of each data block as 20-by-20.

Output Arguments

`newhcube` — Output hyperspectral data

`hypercube` object

Output hyperspectral data, returned as a `hypercube` object. The pixel values of the data cube returned at the output specifies the top of atmosphere (TOA) reflectance values.

More About

Compute TOA Reflectance values from DNs

Given a digital number (DN), the TOA reflectance is computed by using the reflectance gain (R_{Gain}) and reflectance offset (R_{Offset}) of each spectral band in the data cube.

The reflectance gain and reflectance offset values of each spectral band are stored in the header file.

Alternatively, the TOA reflectance values can be estimated from digital numbers (DN) by using these two steps:

1. Compute the radiance values from the digital number (DN).

   $Gain_\lambda$ and $Bias_\lambda$ are the gain and offset values for each spectral band (λ) respectively. The `Metadata` property of `hypercube` object contains the gain and offset values.

2. Compute the TOA reflectance values from the radiance values.

d is the earth-sun distance in astronomical units, $ESUN_\lambda$ is the mean solar irradiance for each spectral band, and θ_E is the sun elevation angle.

See Also

`dn2radiance` | `radiance2Reflectance` | `empiricalLine` | `iarr` | `sharc` | `hypercube`

Introduced in R2020b

empiricalLine

Empirical line calibration of hyperspectral data

Syntax

```
newhcube = empiricalLine(hcube,imgSpectra,fieldSpectra,fieldWL)
```

Description

`newhcube = empiricalLine(hcube,imgSpectra,fieldSpectra,fieldWL)` performs empirical line calibration of the hyperspectral data, `hcube`. The function calculates empirical line factors to force the image spectral data, `imgSpectra`, to match the field reflectance spectra, `fieldSpectra`, with wavelengths `fieldWL`. For more information, see “Algorithms” on page 1-3244.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Perform Empirical Line Calibration of Hyperspectral Data

Read hyperspectral data into the workspace. This data is from the EO-1 Hyperion sensor, with pixel values in digital numbers.

```
hcube = hypercube('EO1H0440342002212110PY_cropped.hdr');
```

Remove bad bands from the input data.

```
hcube = removeBands(hcube,'BandNumber',find(~hcube.Metadata.BadBands));
```

Convert the pixel values from digital numbers to top of atmosphere (TOA) radiance values.

```
hcube_toa = dn2radiance(hcube);
```

Select a pixel with a low brightness value as the target pixel, and store the pixel value as a cell array. The pixel belongs to the tar region of the input data.

```
targetPixel = hcube_toa.DataCube(100,86,:);
imgSpec = {permute(targetPixel,[3 1 2])};
```

Read the reflectance spectral signature of the tar material from the ECOSTRESS library.

```
filename = 'manmade.road.tar.solid.all.0099uuutar.jhu.becknic.spectrum.txt';
info = readEcostressSig(filename);
```

Get the field reflectance spectra and the corresponding wavelength information from the metadata of the ECOSTRESS spectral signature. Store these values as cell arrays.

```
refSpec = {info.Reflectance};
refWL = {info.Wavelength};
```

Perform empirical line calibration of the radiance value hyperspectral data.

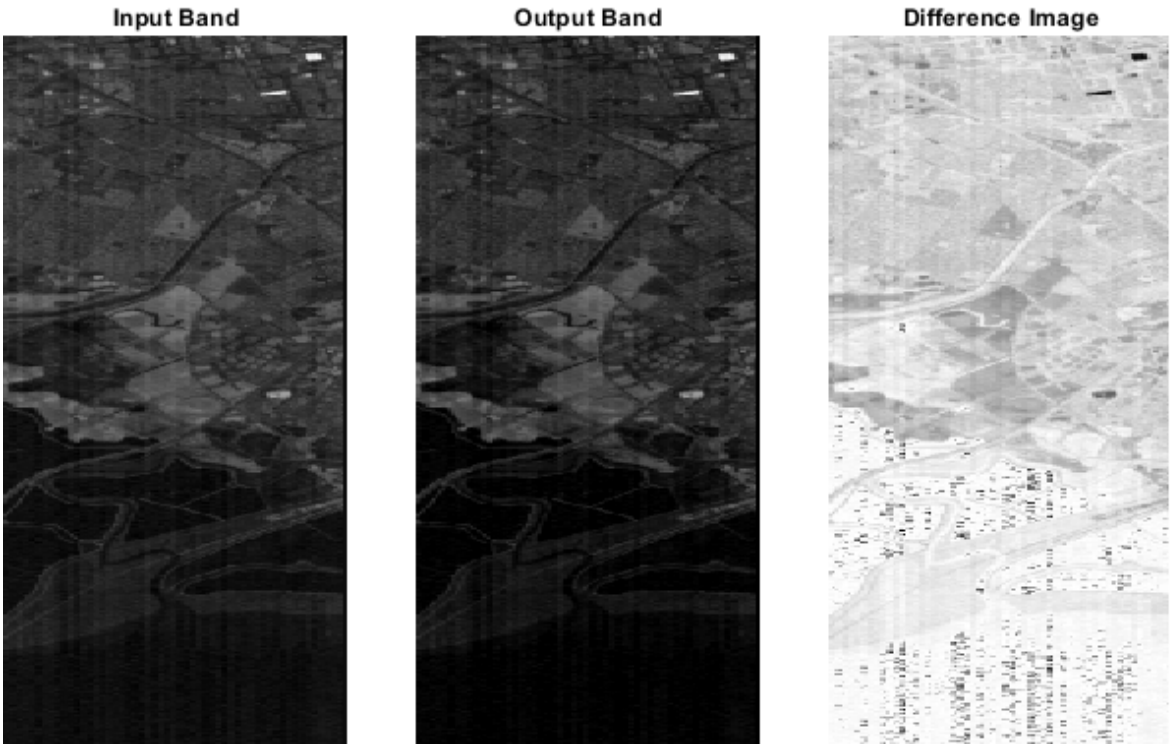
```
hcube_empirical = empiricalLine(hcube_toa,imgSpec,refSpec,refWL);
```

Inspect the results by selecting the same band image from both the uncalibrated digital number hypercube and the calibrated hypercube. For visualization purposes, rescale the values of the two band images to the range [0, 1]. Calculate the absolute value of the difference between the rescaled band images.

```
inputBand = rescale(hcube.DataCube(:,:,159));
outputBand = rescale(hcube_empirical.DataCube(:,:,159));
diffBand = abs(inputBand-outputBand);
```

Display the original spectral band image, the empirically corrected spectral band image, and the absolute value difference between the band images, to visualize the changes.

```
figure('Position',[0 0 700 400])
subplot('Position',[0 0 0.25 0.9])
imagesc(inputBand)
title('Input Band')
axis off
subplot('Position',[0.3 0 0.25 0.9])
imagesc(outputBand)
axis off
title('Output Band')
subplot('Position',[0.6 0 0.25 0.9])
imagesc(diffBand)
axis off
title('Difference Image')
colormap gray
```



Input Arguments

`hcube` — Input hyperspectral data

`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `DataCube` property of the `hypercube` object stores the hyperspectral data cube.

The input pixel values can be digital numbers, TOA radiance values, or TOA reflectance values. To convert a hypercube containing digital numbers to a hypercube containing TOA radiance or TOA reflectance data, use the `dn2radiance` or `dn2reflectance` function, respectively.

`imgSpectra` — Image spectral data

N-by-1 cell array

Image spectral data, specified as an *N*-by-1 cell array. *N* is the number of pixels or fields used in the empirical line calibration. Each cell contains a *C*-by-1 numeric vector, where *C* is the number of hyperspectral bands present in `hcube`.

`fieldSpectra` — Field reflectance spectra

N-by-1 cell array

Field reflectance spectra, specified as an *N*-by-1 cell array. *N* is the number of pixels or fields used in the empirical line calibration. Each cell contains a vector of field reflectances. The vectors can vary in size between cells, but the length of the vector in each cell of `fieldSpectra` and `fieldWL` must match.

`fieldWL` — Wavelength of field reflectance spectra
N-by-1 cell array

Wavelength of field reflectance spectra in nanometers, specified as an *N*-by-1 cell array. *N* is the number of pixels or fields used in the empirical line calibration. Each cell contains a *P*-by-1 vector of field wavelengths, of varying lengths. The length of the vector in each cell of `fieldSpectra` and `fieldWL` must match.

Output Arguments

`newhcube` — Calibrated hyperspectral data
`hypercube` object | 3-D numeric array

Calibrated hyperspectral data, returned as a `hypercube` object or 3-D numeric array consistent with the input data, `inputData`. The data type of numeric output is `single`. When the input data in `inputData` is of data type `double`, then the corrected data is also of data type `double`. Otherwise, the corrected data is of data type `single`.

Algorithms

The `empiricalLine` function performs linear regression for each band to equate the digital number (DN), or TOA radiance or TOA reflectance, with the surface reflectance. Solving the linear regression equation provides gain and offset values for each band. This equation shows how the empirical line factors gain and offset values are calculated:

$$\rho_{surface_\lambda} = m r_\lambda + offset$$

The gain (*m*) and the offset values (*offset*) are the unknown parameters in the empirical line equation. ρ_λ is the known surface reflectance value of a reference material in the input hyperspectral data (known as the field reference spectra). r_λ is the measured value for the reference material in the input hyperspectral data (known as the image spectral data). The measured value can be a digital number, TOA radiance, or TOA reflectance.

The field reference spectra is an *a priori* measurement which can also be read from the spectral libraries. The empirical line approach solves the linear equation to find the gain and the offset values. The surface reflectance values for all the other pixels in the input hyperspectral data is calculated using the estimated gain and the offset values.

The `empiricalLine` function automatically resamples the input field spectra to match the selected data wavelengths in `hcube`.

To solve the linear regression equation, at least two field spectrum values must be known for each band. If the `empiricalLine` function is provided with only one field spectrum value for each band, the offset value is set as zero. If there is no field spectrum value available for any of the bands, then this function throws an error.

References

[1] Roberts, D. A., Y. Yamaguchi, and R. J. P. Lyon. "Comparison of Various Techniques for Calibration of AIS Data." In *Proceedings of the Second Airborne Imaging Spectrometer Data Analysis Workshop*, ed. Gregg Vane and Alexander F. H. Goetz, 21–30. Pasadena: Jet Propulsion Laboratory, 1986.

[2] Kruse, F. A., K. S. Kierein-Young, and J.W. Boardman. "Mineral Mapping at Cuprite, Nevada with a 63-Channel Imaging Spectrometer," *Photogrammetric Engineering and Remote Sensing* 56, no. 1 (January 1990): 83–92.

See Also

`hypercube` | `iarr` | `flatField` | `logResiduals` | `subtractDarkPixel` | `reduceSmile` | `sharc`

Introduced in R2020b

enviinfo

Read metadata from ENVI header file

Syntax

```
info = enviinfo(file)
```

Description

`info = enviinfo(file)` reads the metadata from ENVI (Environment for Visualizing Images) header file.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Read ENVI Header File

Read an ENVI header file into the workspace.

```
info = enviinfo('paviaU.hdr');
```

Create a `hypercube` object using the `Filename` of the ENVI header file.

```
hcube = hypercube(info.Filename)
```

```
hcube =
  hypercube with properties:

      DataCube: [610×340×103 double]
    Wavelength: [103×1 double]
      Metadata: [1×1 struct]
```

Read ENVI Binary Data File Using Metadata from ENVI Header File

Read an ENVI header file into the workspace.

```
info = enviinfo('indian_pines');
```

Read from the ENVI binary data file by using the metadata from the ENVI header file.

```
data = multibandread('indian_pines.dat',...
    [info.Height info.Width info.Bands],...
    info.DataType,info.HeaderOffset,info.Interleave,info.ByteOrder);
```

Input Arguments

`file` — Name of ENVI header file

string scalar | character vector

Name of ENVI header file, specified as a string scalar or character vector. An ENVI header file must have the extension `.hdr`. If you do not specify a file extension, then the function looks for a file with the specified name and the `.hdr` file extension.

Data Types: `char` | `string`

Output Arguments

`info` — Information about ENVI data and metadata

struct

Information about ENVI data and metadata, returned as a structure array containing at least these fields. If the ENVI header file contains additional fields, then the structure array contains those additional fields as well.

Field	Description
`Height`	Height of the image or number of rows in the image, returned as a positive integer.
`Width`	Width of the image or number of columns in the image, returned as a positive integer.
`Bands`	Number of spectral bands, returned as a positive integer.
`DataType`	Data type of data in the ENVI file, returned as any of these values: • `"single"` • `"double"` • `"uint8"` • `"uint16"` • `"int16"` • `"uint32"` • `"int32"` • `"uint64"` • `"int64"`
`Interleave`	Data interleave, returned as any one of these values: • `"bsq"` — Band-sequential • `"bil"` — Band-interleaved-by-line • `"bip"` — Band-interleaved-by-pixel

Field	Description
`HeaderOffset`	Zero-based location of the first element in the image file, returned as a positive integer. The header offset represents the number of bytes from the beginning of the image file to the start of the image data.
`ByteOrder`	Endianness of the data, returned as the string `"ieee-le"` for little endian or `"ieee-be"` for big endian.

See Also

`hypercube` | `multibandread`

Introduced in R2020a

enviwrite

Write hyperspectral data to ENVI file format

Syntax

```
enviwrite(hcube,filename)
enviwrite( ___ ,Name,Value)
```

Description

`enviwrite(hcube,filename)` writes the hyperspectral data stored in the `hypercube` object to an ENVI (Environment for Visualizing Images) file format. The function creates an ENVI header file and ENVI binary data file with file extensions `.hdr` and `.dat`, respectively. The function writes the wavelength and metadata information to the ENVI header file and the data cube containing the hyperspectral images to the ENVI binary data file.

`enviwrite( ___ ,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input arguments in the previous syntax.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Write Hyperspectral Bands to ENVI File

Read a hyperspectral data into the workspace.

```
hcube = hypercube('paviaU.hdr');
```

Extract the twenty most informative bands from the hyperspectral data.

```
sig = fippi(hcube,5);
newhcube = selectBands(hcube,sig,'NumberOfBands',20);
```

Write the selected hyperspectral data to the ENVI file format. The binary data file is named `newData.dat` and the header file is named `newData.hdr`.

```
enviwrite(newhcube,'newData');
```

Input Arguments

hcube — Input hyperspectral data

`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `hypercube` object contains the data cube, wavelength, and related metadata information.

`filename` — Name of ENVI files
character vector | string scalar

Name of the ENVI files, specified as a character vector or string scalar. The function uses the specified value as the name of both the binary data file (`.dat`) and header file (`.hdr`).

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

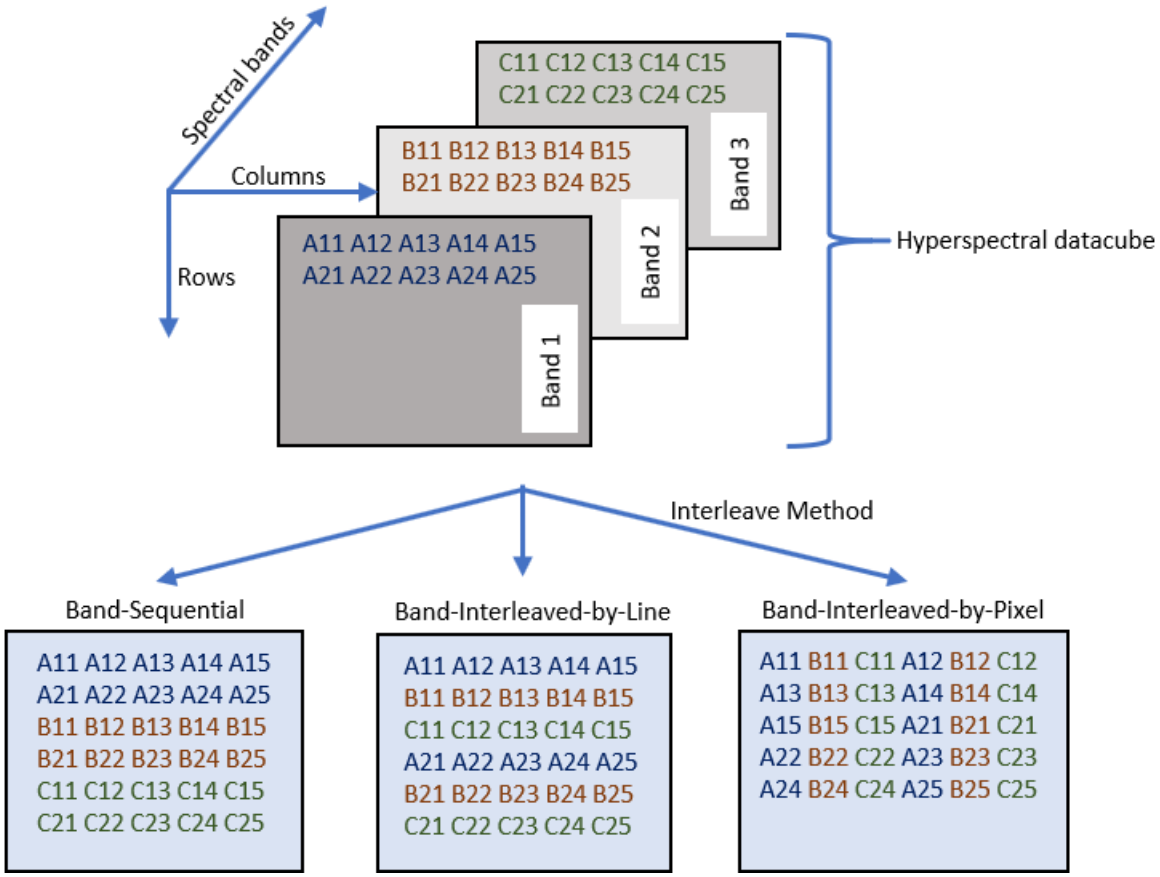
Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `enviwrite(hcube,filename,'Interleave','bip')`

`Interleave` — Band interleaving method
`'bsq'` (default) | `'bil'` | `'bip'`

Band interleaving method, specified as the comma-separated pair consisting of '`Interleave`' and one of these values:

- `'bsq'` — The function uses the band-sequential interleaving method. It writes the entirety of each band before writing the next band. This is the default method.
- `'bil'` — The function uses the band-interleaved-by-line interleaving method. It writes an entire row from each band before writing the next row.
- `'bip'` — The function uses the band-interleaved-by-pixel interleaving method. It writes a pixel from each band before writing the next pixel.



Data Types: `char` | `string`

`DataType` — Data type to write to ENVI binary data file
`'single'` | `'double'` | `'uint8'` | `'uint16'` | `'uint32'` | `'uint64'` | `'int8'` | `'int16'` | `'int32'` | `'int64'`

Data type to write to the ENVI binary data file, specified as the comma-separated pair consisting of `'DataType'` and a valid data type.

Data Types: `char` | `string`

`ByteOrder` — Endianness of binary data file
`'ieee-le'` | `'ieee-be'`

Endianness of binary data file, specified as the comma-separated pair consisting of `'ByteOrder'` and `'ieee-le'` or `'ieee-be'`. Specify the value as `'ieee-le'` for little-endian format and `'ieee-be'` for big-endian format. By default, the function uses endianness format of your machine.

Data Types: `char` | `string`

`HeaderOffset` — Number of bytes before data starts
`0` (default) | positive integer

Number of bytes before the data starts, specified as the comma-separated pair consisting of `'HeaderOffset'` and a positive integer. If the header file does not exist, the function writes ASCII null values will be written to fill the space by default.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

See Also

`hypercube` | `removeBands` | `selectBands` | `multibandwrite`

Introduced in R2020a

estimateAbundanceLS

Estimate abundance maps

Syntax

```
abundanceMap = estimateAbundanceLS(inputData,endmembers)
abundanceMap = estimateAbundanceLS( ___ ,'Method',estMethod)
```

Description

`abundanceMap = estimateAbundanceLS(inputData,endmembers)` estimates the abundance maps of the endmembers in a hyperspectral data cube by using the least-squares method.

A hyperspectral data cube can contain both pure and mixed pixels. Pure pixels exhibit the spectral characteristics of a single class, while the mixed pixels exhibit the spectral characteristics of multiple classes. The spectral signatures of the pure pixels comprise the endmembers that identify the unique classes present in a hyperspectral data cube. The spectral signature of mixed pixels can be a linear combination of two or more endmember spectra. The abundance map identifies the proportion of each endmember present in the spectra of each pixel. For a hyperspectral data cube of spatial dimensions *M*-by-*N* containing *P* endmembers, there exist *P* abundance maps, each of size *M*-by-*N*.

The abundance map estimation process is known as *spectral unmixing*, which is the decomposition of the spectra of each pixel into a given set of endmember spectra.

`abundanceMap = estimateAbundanceLS( ___ ,'Method',estMethod)` specifies the least-squares method to use for estimating the abundance maps.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

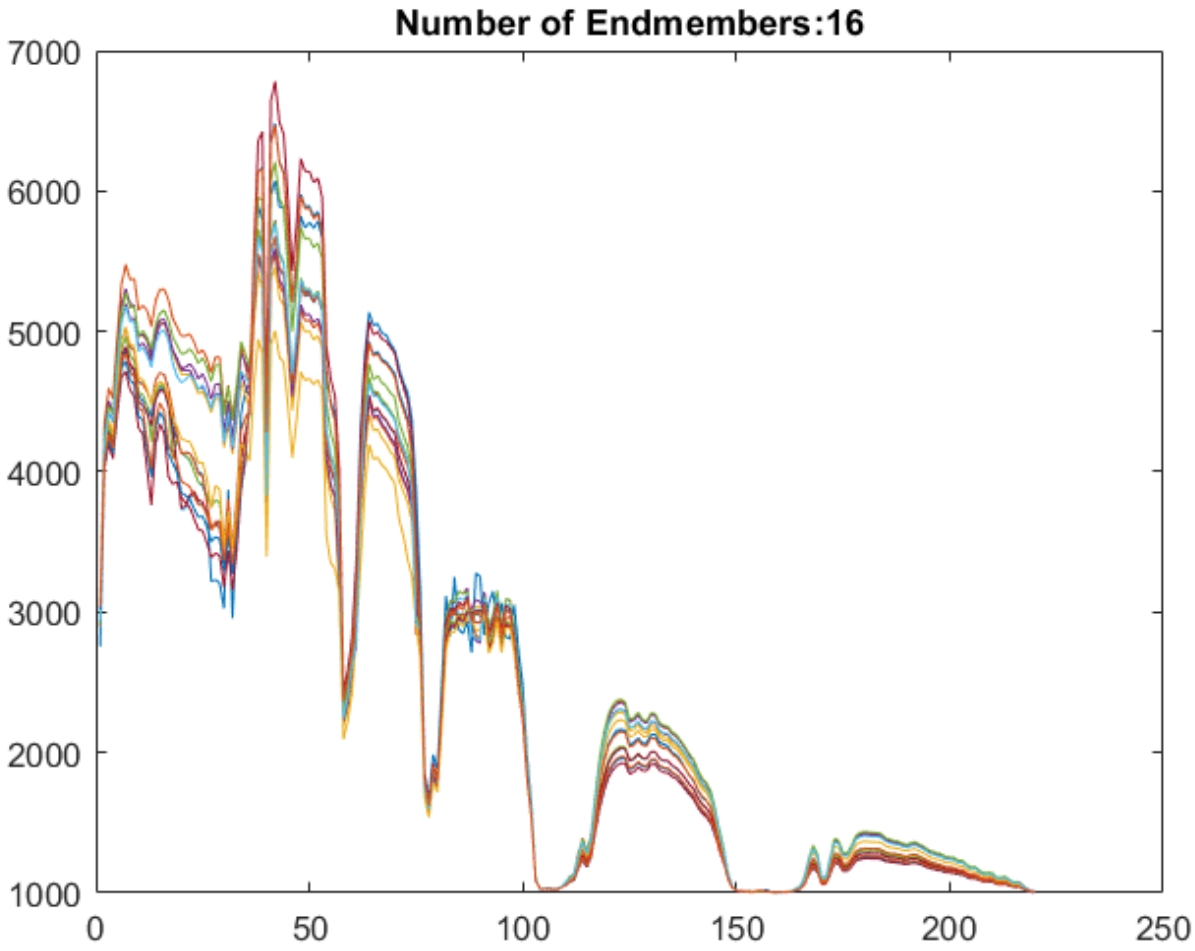
Estimate Abundance Maps For Hyperspectral Data

Load a MAT-file containing a hyperspectral data cube and endmember signatures into the workspace. Extract the data cube and endmember signatures.

```
data = load('indian_pines.mat');
dataCube = data.indian_pines;
endmembers = data.signatures;
```

Plot the endmember spectra.

```
plot(endmembers)
numEndMem = num2str(size(endmembers,2));
title(['Number of Endmembers:' numEndMem])
```

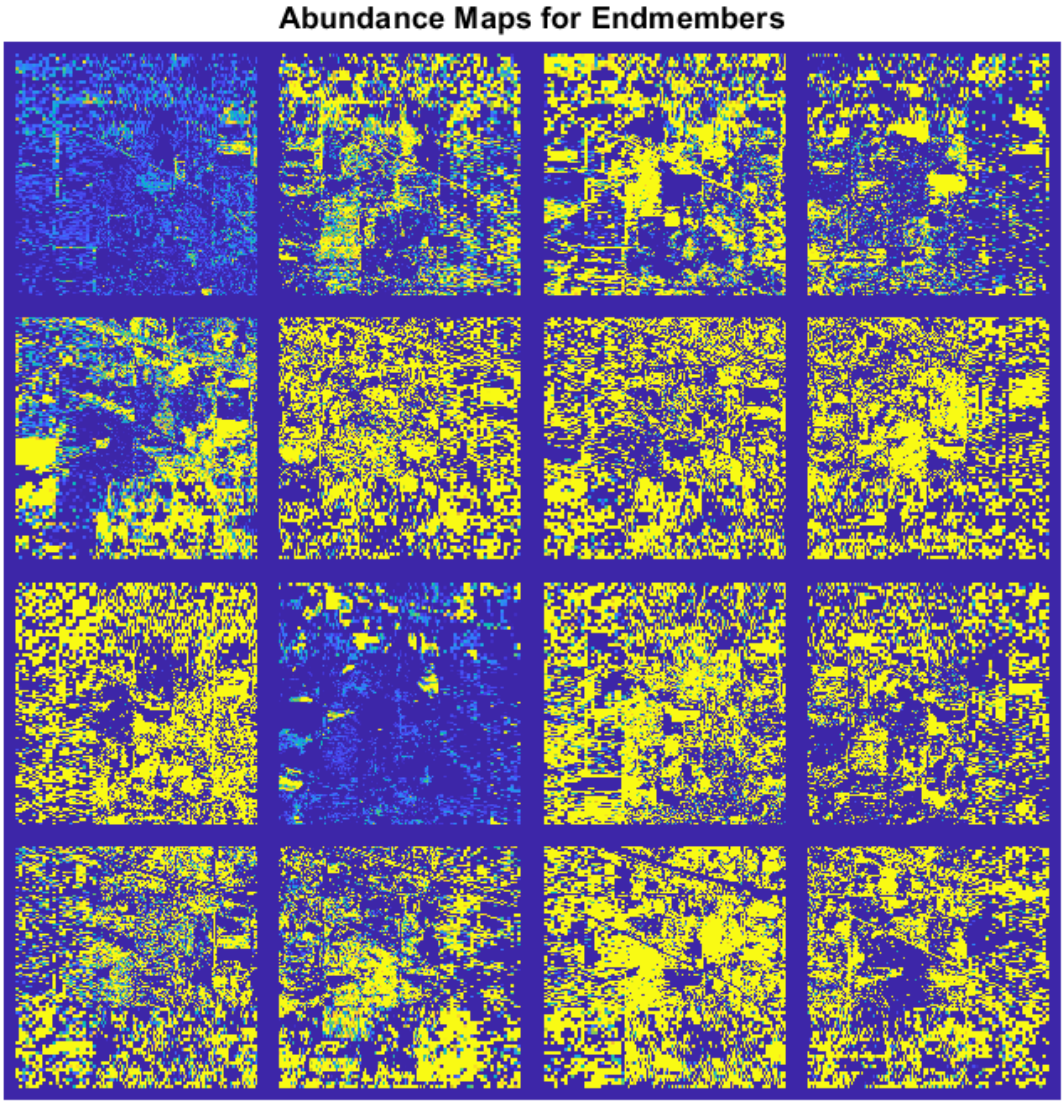


Estimate the abundance maps for the endmember spectra.

```
abundanceMap = estimateAbundanceLS(dataCube,endmembers);
```

Display the abundance map for each endmember spectra. The order of the abundance maps is the same as the order of the endmembers in the `endmembers` input.

```
montage(abundanceMap,'Size',[4 4],'BorderSize',[10 10]);
colormap default
title('Abundance Maps for Endmembers');
```



Extract Endmembers and Abundance Maps for Spectral Unmixing

Read hyperspectral data into the workspace.

```
hcube = hypercube('jasperRidge2_R198.hdr');
```

Extract six endmembers from the hyperspectral data.

```
endmembers = nfindr(hcube,6);
```

Estimate the abundance map for each endmember using the fully constrained least squares method.

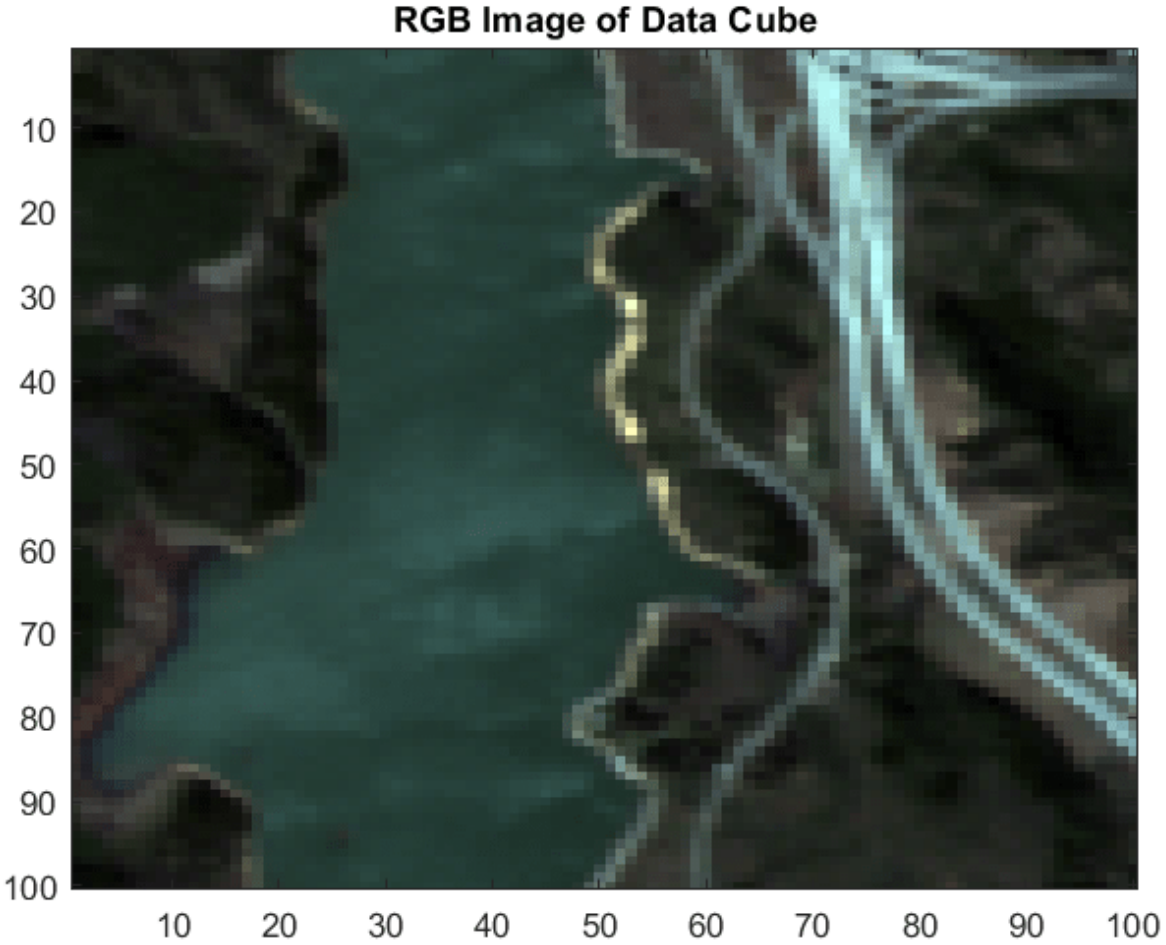
```
abundanceMap = estimateAbundanceLS(hcube.DataCube,endmembers,'Method','fcls');
```

Estimate an RGB image of the data cube using the `colorize` function.

```
rgbImg = colorize(hcube,'Method','RGB');
```

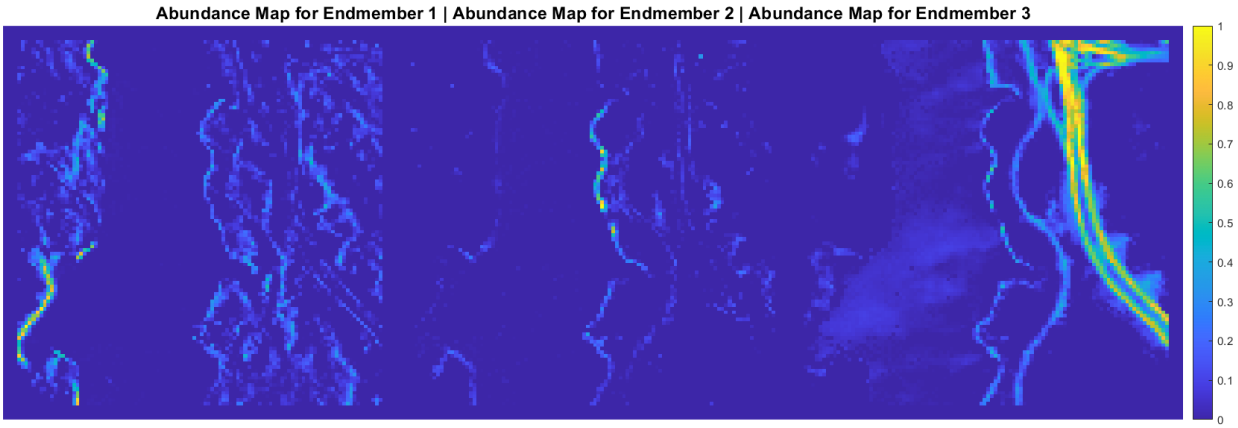
Display the RGB image.

```
figure
imagesc(rgbImg)
title('RGB Image of Data Cube')
```

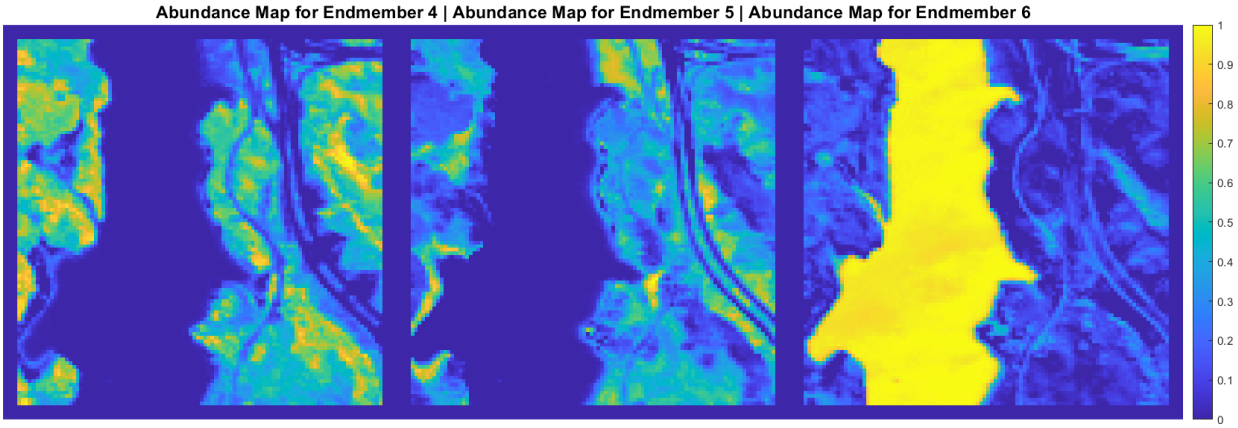


Display the abundance maps for the endmember spectra. The order of the abundance maps is the same as the order of the endmembers in the `endmembers` input argument.

```
figure
montage(abundanceMap(:,:,1:3),'Size',[1 3],'BorderSize',[20 20])
colormap default
colorbar
title('Abundance Map for Endmember 1 | Abundance Map for Endmember 2 | Abundance Map for Endmembe
```




```
figure
montage(abundanceMap(:,:,4:6),'Size',[1 3],'BorderSize',[20 20])
colormap default
colorbar
title('Abundance Map for Endmember 4 | Abundance Map for Endmember 5 | Abundance Map for Endmembe
```



Input Arguments

`inputData` — Input hyperspectral data

3-D numeric array | `hypercube` object

Input hyperspectral data, specified as a 3-D numeric array that represent the hyperspectral data cube of size *M*-by-*N*-by-*C* or `hypercube` object. If the input is a `hypercube` object, the function reads the data cube stored in the `DataCube` property of the object. The hyperspectral data cube must be real and non-sparse.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`endmembers` — Endmember signatures

C-by-*P* matrix

Endmember signatures, specified as a matrix of size *C*-by-*P*. where *C* is the number of spectral bands in the input hyperspectral data and *P* is the number of endmember spectral signatures.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`estMethod` — Method for estimating abundance maps
`'ucls'` (default) | `'fcls'` | `'ncls'`

Method for estimating abundance maps, specified as one of these values.

- `'ucls'` — Unconstrained least-squares method.
- `'fcls'` — Fully constrained least-squares method.
- `'ncls'` — Nonnegative constrained least-squares method.

Example: `estimateAbundanceLS(inputData,endmembers,'Method','ncls')`

Data Types: `char` | `string`

Output Arguments

`abundanceMap` — Abundance maps
3-D numeric array

Abundance maps, returned as a 3-D numeric array of size *M*-by-*N*-by-*P*.

Data Types: `double`

References

[1] Keshava, N., and J.F. Mustard. “Spectral Unmixing.” *IEEE Signal Processing Magazine* 19, no. 1 (January 2002): 44–57. https://doi.org/10.1109/79.974727.

[2] Kay, Steven M. *Fundamentals of Statistical Signal Processing*. Prentice Hall Signal Processing Series. Englewood Cliffs, N.J: Prentice-Hall PTR, 1993.

See Also

`hypercube` | `fippi` | `ppi` | `nfindr`

Introduced in R2020a

fastInScene

Perform fast in-scene atmospheric correction

Syntax

```
newhcube = fastInScene(hcube)
```

Description

`newhcube = fastInScene(hcube)` returns atmospherically corrected data cube by computing the surface reflectance values from the top of atmosphere (TOA) reflectance values in input hyperspectral data. Use this function to perform fast atmospheric correction that uses the in-scene characteristics for estimating the correction parameters.

The fast in-scene method gives best correction results, if the hyperspectral data

- is radiometrically calibrated
- is uniformly illuminated
- does not contain large water bodies, cloud, or cloud shadows
- contain adequate dark pixels for approximately computing the baseline spectrum
- contain heterogeneous regions that include soil, water, vegetation, and man-made structures. The method assumes that the mean reflectance spectrum of different endmember spectra are scene-independent.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Perform Fast In-Scene Atmospheric Correction of Hyperspectral Data

Read hyperspectral data into the workspace.

```
inputCube = hypercube('EO1H0440342002212110PY_cropped.dat');
```

Remove low signal-to-noise ratio (SNR) bands from the hyperspectral data cube.

```
inputCube = removeBands(inputCube,'BandNumber',find(~inputCube.Metadata.BadBands));
```

Convert digital number (DN) to top of atmosphere (TOA) reflectance values.

```
inputCube = dn2reflectance(inputCube);
```

Remove atmospheric effects from the input hyperspectral data based on the in-scene characteristics.

```
correctedCube = fastInScene(inputCube);
```

Input Arguments

hcube — Input hyperspectral data

`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `DataCube` property of the `hypercube` object stores the hyperspectral data cube. For better results, the input values must be TOA reflectance values. If the input values are digital numbers, use the `dn2reflectance` function to estimate the TOA reflectance values.

Output Arguments

newhcube — Output hyperspectral data

`hypercube` object

Output hyperspectral data, returned as a `hypercube` object. The pixel values of the data cube returned at the output specifies the surface reflectance values.

References

[1] Bernstein, L.S., S.M. Adler-Golden, R.L. Sundberg, R.Y. Levine, T.C. Perkins, A. Berk, A.J. Ratkowski, G. Felde, and M.L. Hoke. "A New Method for Atmospheric Correction and Aerosol Optical Property Retrieval for VIS-SWIR Multi- and Hyperspectral Imaging Sensors: QUAC (QUick Atmospheric Correction)." In *Proceedings. 2005 IEEE International Geoscience and Remote Sensing Symposium, 2005. IGARSS '05.*, 5:3549–52. Seoul, Korea: IEEE, 2005. https://doi.org/10.1109/IGARSS.2005.1526613.

See Also

`dn2reflectance` | `dn2radiance` | `radiance2Reflectance` | `sharc`

Introduced in R2020b

fippi

Extract endmember signatures using fast iterative pixel purity index

Syntax

```
endmembers = fippi(inputData,numEndmembers)
endmembers = fippi(inputData,numEndmembers,'ReductionMethod',method)
```

Description

`endmembers = fippi(inputData,numEndmembers)` extracts endmember signatures from hyperspectral data `inputData` by using the fast iterative pixel purity index (FIPPI) algorithm. `numEndmembers` is the number of endmember signatures to be extracted using the FIPPI algorithm. For more information about the FIPPI method, see “Algorithms” on page 1-3264.

`endmembers = fippi(inputData,numEndmembers,'ReductionMethod',method)` additionally specifies the option for selecting the dimensionality reduction method to be used before computing the endmembers.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Extract Endmembers Using Fast Iterative Pixel Purity Index

Read a hyperspectral data into the workspace.

```
hcube = hypercube('paviaU.hdr');
```

Find the number of spectrally distinct endmembers present in the hyperspectral data cube by using `countEndmembersHFC` function.

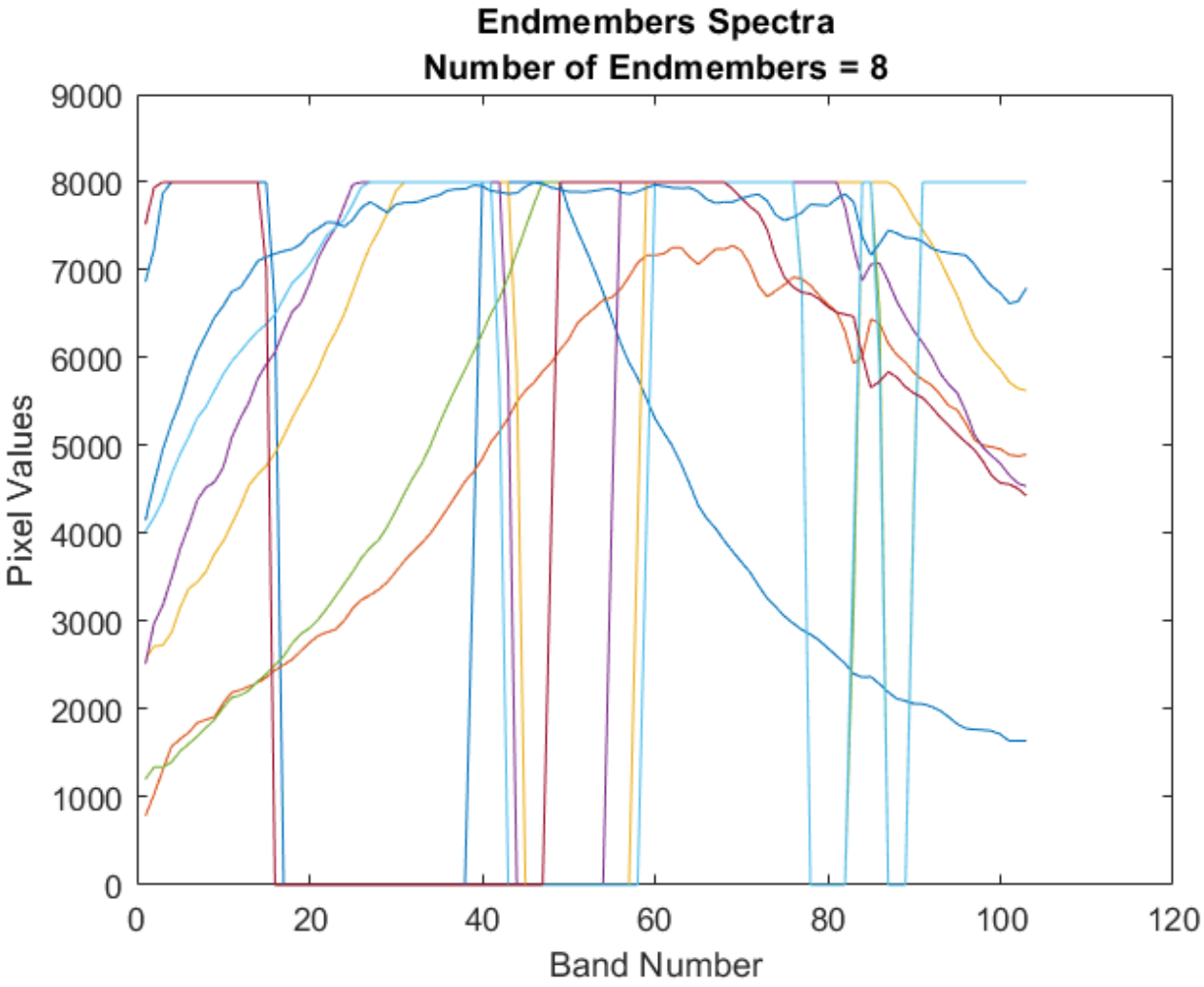
```
numEndmembers = countEndmembersHFC(hcube,'PFA',10^-7);
```

Compute the endmembers using the fast iterative pixel purity index (FIPPI) method. By default, the `fippi` function uses maximum noise fraction (MNF) transform for preprocessing.

```
endmembers = fippi(hcube.DataCube,numEndmembers);
```

Plot the endmembers of the hyperspectral data.

```
figure
plot(endmembers)
xlabel('Band Number')
ylabel('Pixel Values')
ylim([0 9000])
title({'Endmembers Spectra',['Number of Endmembers = ' num2str(numEndmembers)]});
```



Use PCA Reduction Method For Computing Endmembers

Read a hyperspectral data into the workspace.

```
hcube = hypercube('paviaU.hdr');
```

Find the number of spectrally distinct endmembers present in the hyperspectral data cube by using `countEndmembersHFC` function.

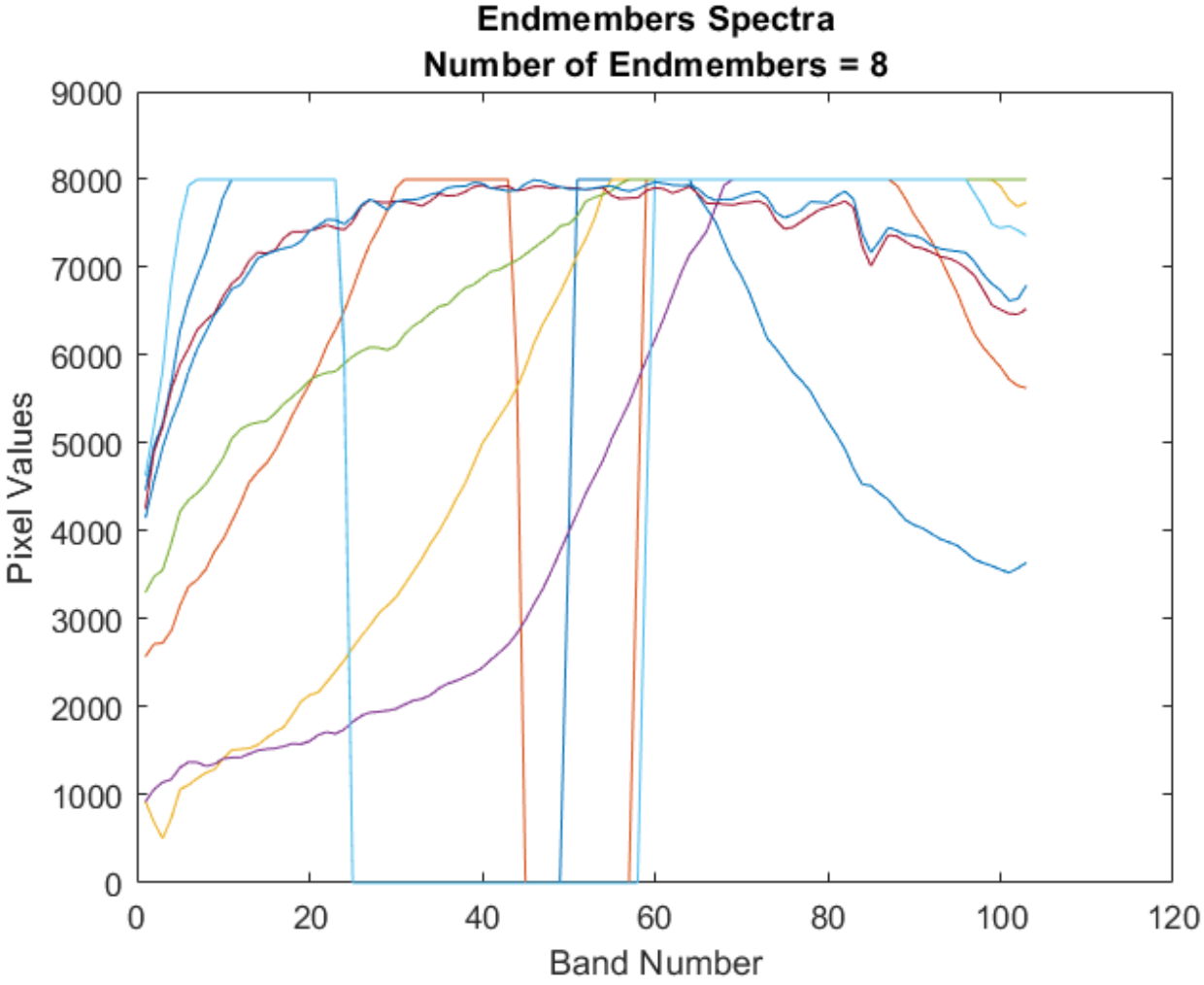
```
numEndmembers = countEndmembersHFC(hcube,'PFA',10^-7);
```

Compute the endmembers using the fast iterative pixel purity index (FIPPI) method. Select principal component analysis (PCA) as the dimensionality reduction method for preprocessing.

```
endmembers = fippi(hcube.DataCube,numEndmembers,'ReductionMethod','PCA');
```

Plot the endmembers of the hyperspectral data.

```
figure
plot(endmembers)
xlabel('Band Number')
ylabel('Pixel Values')
ylim([0 9000])
title({'Endmembers Spectra',['Number of Endmembers = ' num2str(numEndmembers)]});
```



Input Arguments

`inputData` — Input hyperspectral data

3-D numeric array | `hypercube` object

Input hyperspectral data, specified as an 3-D numeric array or a `hypercube` object. If the input is a `hypercube` object, then the function reads the hyperspectral data from its `DataCube` property.

The hyperspectral data is an numeric array of size *M*-by-*N*-by-*C*. *M* and*N* are the number of rows and columns in the hyperspectral data respectively. *C* is the number of spectral bands in the hyperspectral data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`numEndmembers` — Number of endmembers
positive scalar integer

Number of endmembers to be extracted, specified as a positive scalar integer. The value must be in the range [1 *C*]. *C* is the number of spectral bands in the input hyperspectral data. You can find the number of spectrally distinct endmembers in the input data by using the `countEndmembersHFC` function.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`method` — Dimensionality reduction method
`'MNF'` (default) | `'PCA'`

Dimensionality reduction method, specified as a comma-separated pair of `'ReductionMethod'` and one of `'MNF'` or `'PCA'`.

Specify the value as

- `'MNF'` — To perform dimensionality reduction using the maximum noise fraction (MNF) method.
- `'PCA'` — To perform dimensionality reduction using the principal component analysis (PCA) method.

The function computes the endmembers from the reduced data.

Data Types: `char` | `string`

Output Arguments

`endmembers` — Endmember signatures
C-by-*K* matrix

Endmember signatures, returned as a matrix of size *C*-by-*K* and datatype same as the input hyperspectral data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Algorithms

FIPPI is an iterative approach that iteratively selects the better candidates for endmembers after each iteration. Unlike pixel purity index (PPI) technique, the FIPPI method selects the initial set of skewers by using the automatic target generation process (ATGP) [1]. As a result the algorithm converges faster and generates unique pixel for each endmember. The steps involved in FIPPI approach are summarized as follows:

1. Compute principal component bands and reduce the spectral dimensionality of the input data by using MNF or PCA. The number of principal component bands to be extracted is set equal to the number of endmembers to be extracted.
2. Find the initial set of endmembers by using the ATGP method. The initial set of endmembers form the set of skewers $\left\{\text{skewer}_j^{(1)}\right\}_{j=1}^{p}$ onto which you project the input data.
3. For iteration 1, Let r_1 be a sample vector that denote a pixel spectra. Then, orthogonally project the sample vector onto each skewers and find the extrema.

4. Store the location of each extreme value and count their occurrences. The number of occurrences is known as the PPI count.
5. Find the PPI count for each pixel spectra and identify the set of sample vectors $\{r_k\}$ with maximum PPI count as endmembers.
6. Generate a new set of skewers by combining the set of new endmembers with the initial set of skewers.

   $$\left\{\text{skewer}_j^{(2)}\right\} = \left\{\text{r}_k^{(1)}\right\} \cup \left\{\text{skewer}_j^{(1)}\right\}$$

7. For iteration 2, project all the sample vectors onto the new set of skewers and identify the new set of endmembers. Then, generate the new set of skewers for the next iteration, $\left\{\text{skewer}_j^{(3)}\right\}$.
8. The iteration stops, if the set of skewers generated in two consecutive iterations remain same. This final set of skewers are the endmembers of the input data.

   $$\left\{\text{skewer}_j^{(n+1)}\right\} = \left\{\text{skewer}_j^{(n)}\right\}$$

References

[1] Chang, C.-I., and A. Plaza. "A Fast Iterative Algorithm for Implementation of Pixel Purity Index." *IEEE Geoscience and Remote Sensing Letters* 3, no. 1 (January 2006): 63–67. https://doi.org/10.1109/LGRS.2005.856701.

See Also

`hypercube` | `ppi` | `countEndmembersHFC`

Introduced in R2020a

flatField

Apply flat field correction to hyperspectral data cube

Syntax

```
correctedData = flatField(inputData,roi)
```

Description

`correctedData = flatField(inputData,roi)` applies flat field correction to the hyperspectral data, `inputData`, using the flat field mean spectrum calculated in the specified region of interest (ROI) of the hyperspectral data. A valid ROI has these characteristics:

- Topographically flat
- Spectrally flat (uniform spectral response)
- Strong signal source to reduce the impact of random noise

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Apply Flat Field Correction to Hyperspectral Data

Read hyperspectral data into the workspace.

```
hcube = hypercube('paviaU');
```

Specify the ROI from which to calculate the flat field mean spectrum.

```
roi = [1 1 10 10];
```

Apply the flat field correction to the hyperspectral data.

```
hcube_flatfield = flatField(hcube,roi);
```

Input Arguments

`inputData` — Input hyperspectral data

`hypercube` object | *M*-by-*N*-by-*C* numeric array

Input hyperspectral data, specified as one of these options:

- `hypercube` object — The `DataCube` property of the `hypercube` object stores the hyperspectral data cube.

- *M*-by-*N*-by-*C* numeric array — *M* and *N* are the number of rows and columns of pixels in the hyperspectral data, respectively. *C* is the number of spectral bands in the hyperspectral data.

The input pixel values can be digital numbers, TOA radiance values, or TOA reflectance values. To convert a hypercube containing digital numbers to a hypercube containing TOA radiance or TOA reflectance data, use the `dn2radiance` or `dn2reflectance` function, respectively.

`roi` — ROI for calculation of flat field mean spectrum
4-element vector

ROI for the calculation of the flat field mean spectrum, specified as a 4-element vector of positive integers of the form [*xmin ymin width height*]. The vector defines a rectangular ROI within the hyperspectral data. *xmin* and *ymin* are the *xy*-coordinates of the upper-left corner of the ROI. *width* and *height* are the width and height, respectively, of the ROI, in pixels

Output Arguments

`correctedData` — Corrected hyperspectral data
`hypercube` object | *M*-by-*N*-by-*C* numeric array

Corrected hyperspectral data, returned as a `hypercube` object or *M*-by-*N*-by-*C* numeric array consistent with the input data, `inputData`. If the input data in `inputData` is of data type `double`, then the corrected data is also of data type `double`. Otherwise, the corrected data is of data type `single`.

References

[1] Roberts, D. A., Y. Yamaguchi, and R. J. P. Lyon. "Comparison of Various Techniques for Calibration of AIS Data." In *Proceedings of the Second Airborne Imaging Spectrometer Data Analysis Workshop*, ed. Gregg Vane and Alexander F. H. Goetz, 21 -30. Pasadena: Jet Propulsion Laboratory, 1986.

See Also

`hypercube` | `iarr` | `logResiduals` | `subtractDarkPixel` | `empiricalLine` | `reduceSmile` | `sharc`

Introduced in R2020b

hypercube

Read hyperspectral data

Description

The `hypercube` function reads hyperspectral data and returns an `hypercube` object. The object contains the hyperspectral data cube and its related properties. Use the object functions to remove or select a desired hyperspectral band, assign new pixels values, generate colored image, and write hyperspectral data to the ENVI (environment for visualizing images) file format.

Creation

Syntax

```
hcube = hypercube(filename)
hcube = hypercube(img,hdr)
hcube = hypercube( ___ ,wavelength)
hcube = hypercube(tifFile,wavelength)
hcube = hypercube(image,wavelength)
hcube = hypercube(image,wavelength,metadata)
```

Description

`hcube = hypercube(filename)` reads hyperspectral data from the specified input file `filename`. The input file can be a national imagery transmission format (NITF) file, Hyperion level 1R (L1R) file stored in hierarchical data format (HDF), ENVI header or image file, or metadata text extension (MTL) file that contains satellite data from earth observing (EO) satellites.

- EO-1 Hyperion
- EO-1 Advanced Land Imager (EO-1 ALI)
- Landsat Multispectral Scanner (Landsat MSS)
- Landsat Thematic Mapper (Landsat TM)
- Landsat Enhanced Thematic Mapper Plus (Landsat ETM+)
- Landsat Operational Land Imager / Thermal Infrared Scanner (Landsat OLI / TIRS)

Note The `hypercube` function reads satellite data that are stored in georeferenced tagged image file format (GeoTIFF).

`hcube = hypercube(img,hdr)` reads hyperspectral data from the data file `img`. The data file can be an ENVI image file or Hyperion L1R file. The function uses the metadata in the header file `hdr` to interpret the data from `img`.

`hcube = hypercube( ___ ,wavelength)` specifies the wavelength for each spectral band in the input data and sets the `Wavelength` property of the output `hypercube` object.

`hcube = hypercube(tifFile,wavelength)` reads hyperspectral data from a tagged image file format (TIFF) file `tifFile`.

`hcube = hypercube(image,wavelength)` creates a `hypercube` object from the hyperspectral data cube `image` and the specified center wavelength values `wavelength`.

`hcube = hypercube(image,wavelength,metadata)` creates a `hypercube` object from the hyperspectral data cube `image`, specified center wavelength values `wavelength`, and the metadata `metadata`. You can use this syntax to modify the `Metadata` property of a hypercube object.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Input Arguments

`filename` — Input file name

character vector | string scalar

Input file name, specified as a character vector or string scalar. The input file name must be one of these file types.

File Format	Extensions	Additional Requirements
NITF files	`.ntf` `.nsf`	None
GeoTIFF metadata files from EO satellites	`.txt`	File name must end with suffix "MTL".
ENVI image files ENVI header files	`.dat`, `.img`, and `.raw` `.hdr`	Image and header file must be in the same folder and have the same file name.
Hyperion level 1R image files Hyperion header files	`.L1R` `.hdr`	Image and header file must be in the same folder and have the same file name.

Data Types: `char` | `string`

`img` — Image file name

character vector | string scalar

Image file name, specified as a character vector or string scalar. The input file must be a flat binary raster file with the `.dat`, `.raw`, or `.L1R` file extensions. The binary data must be in band sequential (BSQ), band-interleaved-by-pixel (BIP), or band-interleaved-by-line (BIL) format.

Data Types: `char` | `string`

`hdr` — Header file name

character vector | string scalar

Header file name, specified as a character vector or string scalar. The header file contains the metadata for the image file `img` and has the extension `.hdr`.

Data Types: `char` | `string`

`wavelength` — Center wavelength values
C-element vector

Center wavelength values of each spectral band, specified as *C*-element vector. *C* is the spectral dimension, defined as the number of spectral bands, of the input hyperspectral data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`tifFile` — TIFF file name
character vector | string scalar

TIFF file name, specified as a character vector or string scalar. The file name must include the extension `.tiff` or `.tif`.

Data Types: `char` | `string`

`image` — Input hyperspectral data
3-D numeric array

Input hyperspectral data, specified as a 3-D numeric array of size *M*-by-*N*-by-*C*. *M* and *N* are the number of rows and columns in the hyperspectral data, respectively. *C* is the number of spectral bands in the hyperspectral data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`metadata` — Metadata of hyperspectral data
structure array

Metadata of hyperspectral data, specified as a structure array.

Data Types: `struct`

Properties

`DataCube` — Hyperspectral data cube
3-D numeric array

This property is read-only.

Hyperspectral data cube, stored as a 3-D numeric array of size *M*-by-*N*-by-*C*. The data cube stores the hyperspectral data read from a file or numeric array as an array of 2-D monochromatic images. *C* is the number of images or spectral bands, *M* and *N* are the spatial resolution of the images in pixels. The data cube is of the same size and data type as the input data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Wavelength` — Center wavelength values
C-element vector

This property is read-only.

Center wavelength values of each spectral band, specified as a *C*-element vector. *C* is the spectral dimension, defined as the number of spectral bands, of the input hyperspectral data. You can set this property by using `wavelength` input argument.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Metadata` — Metadata of hyperspectral data
structure array

This property is read-only.

Metadata of hyperspectral data, stored as a structure array with these fields as defaults.

Field	Description
`Height`	Height of the image or number of rows in the data cube, specified as a positive integer
`Width`	Width of the image or number of columns in the data cube, specified as a positive integer
`Bands`	Number of spectral bands comprising the data cube, specified as a positive integer
`DataType`	Data type of data, specified as any of these values: • `"single"` • `"double"` • `"uint8"` • `"uint16"` • `"int16"` • `"uint32"` • `"int32"` • `"uint64"` • `"int64"`
`Interleave`	Data interleave, specified as any one of these values: • `"bsq"` — Band-sequential • `"bil"` — Band-interleaved-by-line • `"bip"` — Band-interleaved-by-pixel
`HeaderOffset`	Zero-based location of the first element in the image file, specified as a positive integer The header offset represents the number of bytes from the beginning of the image file to the start of the image data. The default value is 0.
`ByteOrder`	Endianness of the data, specified as the string `"ieee-le"` for little endian or `"ieee-be"` for big endian.
`WavelengthUnits`	Units for the wavelengths of spectral bands, specified as a string. The default value is `"Nanometers"`.

Note The `Metadata` property of `hypercube` object can have one or more additional fields depending on the parameter values stored in the header file of the input hyperspectral data. You can modify the parameters values of the `Metadata` property or add new `Metadata` to the `hypercube` object by specifying the input argument `metadata`.

Data Types: `struct`

Object Functions

assignData	Assign new data to hyperspectral data cube
cropData	Crop regions-of-interest
enviwrite	Write hyperspectral data to ENVI file format
selectBands	Select most informative bands
removeBands	Remove spectral bands from data cube
colorize	Estimate color image of hyperspectral data

Examples

Read and Visualize Hyperspectral Data from ENVI File

Read hyperspectral data stored in the ENVI format into the workspace. Create a `hypercube` object by specifying an ENVI data file and the associated ENVI header file.

```
hcube = hypercube('paviaU.dat','paviaU.hdr');
```

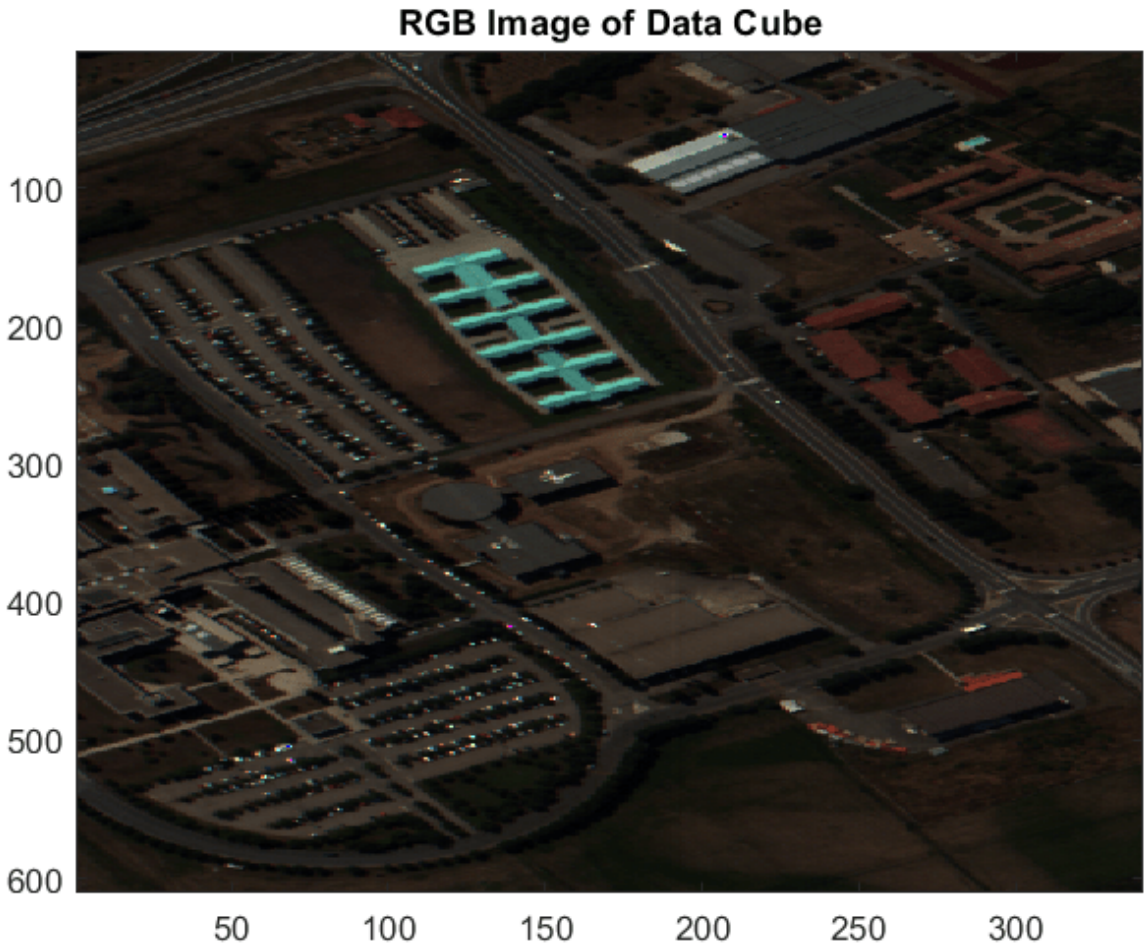
Display the properties of the `hypercube` object.

```
hcube
```

```
hcube =
  hypercube with properties:

      DataCube: [610×340×103 double]
    Wavelength: [103×1 double]
      Metadata: [1×1 struct]
```

Estimate an RGB image from the hyperspectral data by using the `colorize` function. Visualize the RGB image.

```
rgbImg = colorize(hcube,'Method','RGB');
figure
imagesc(rgbImg)
title('RGB Image of Data Cube')
```



Inspect the metadata of the `hypercube` object.

```
hcube.Metadata
```

```
ans = struct with fields:
             Filename: "Y:\jobarchive\Bspkg20b\2020_06_16_h06m34s27_job1406120_pass\matlab\toolbo
          FileModDate: "25-Feb-2020 14:29:34"
             FileSize: 654
               Format: "HDR"
        FormatVersion: ''
           SensorType: [0×0 string]
          Description: [0×0 string]
      AcquisitionTime: [0×0 string]
         RasterFormat: "ENVI"
               Height: 610
                Width: 340
                Bands: 103
             DataType: "double"
           Interleave: "bsq"
         HeaderOffset: 0
            ByteOrder: "ieee-le"
```

```
            BandNames: [0×0 string]
                 FWHM: []
                 Gain: []
               Offset: []
      ReflectanceGain: []
    ReflectanceOffset: []
             BadBands: []
           CloudCover: []
           SunAzimuth: []
        SunElevation: []
```

Read Data from ENVI File and Replace Wavelength Values

Read ENVI format data into the workspace by specifying a header file that contains information about hyperspectral data. The associated ENVI binary data file must be stored in the same folder as the ENVI header file.

```
hcube = hypercube('paviaU.hdr');
```

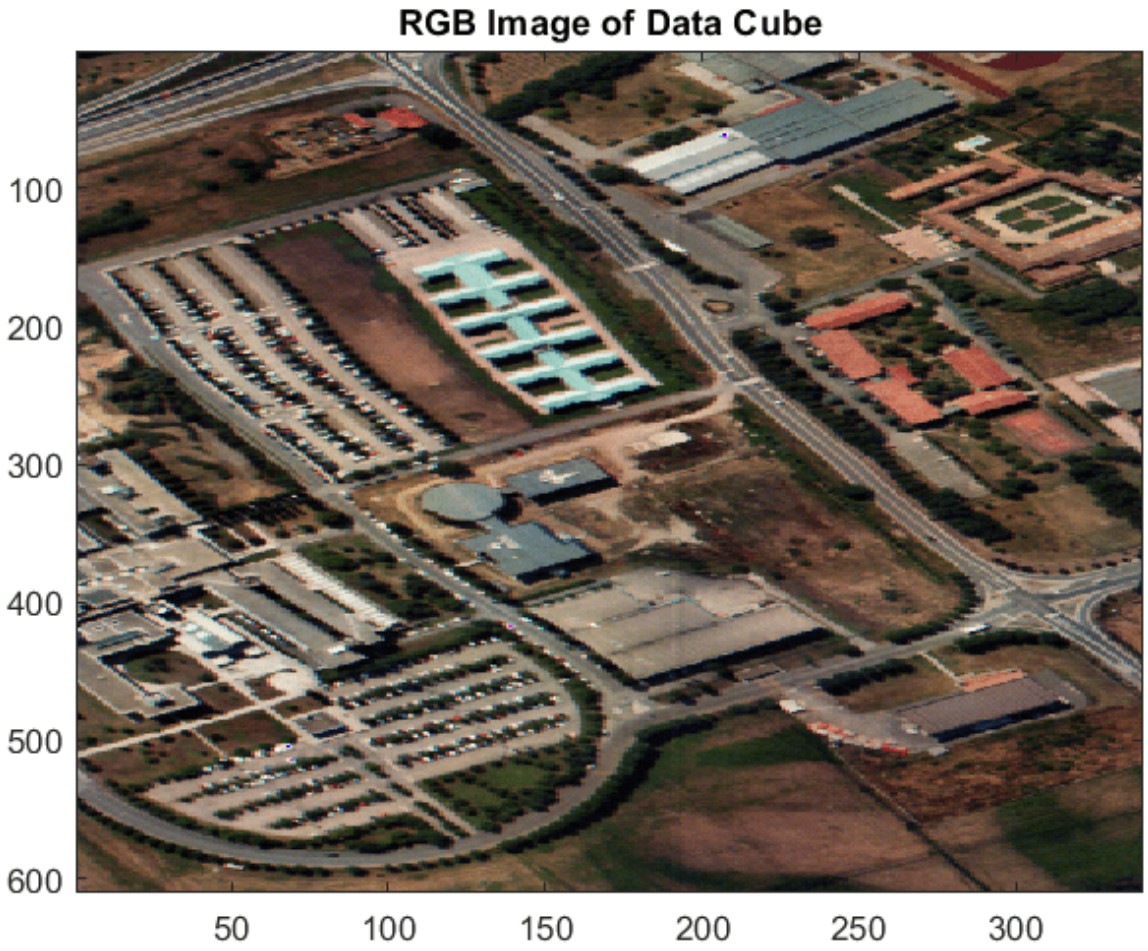
Display the properties of the `hypercube` object.

```
hcube
```

```
hcube =
  hypercube with properties:

      DataCube: [610×340×103 double]
    Wavelength: [103×1 double]
      Metadata: [1×1 struct]
```

Estimate an RGB image from the data cube by using the `colorize` function. Increase the contrast of the RGB image using contrast stretching. Visualize the RGB image.

```
rgbImg = colorize(hcube,'Method','RGB','ContrastStretching',true);
figure
imagesc(rgbImg)
title('RGB Image of Data Cube')
```



Assign new center wavelength values for the hyperspectral data. The number of wavelength values must be equal to the number of bands in the hyperspectral data cube. Each wavelength value must be unique values.

```
minWavelength = 500;
maxWavelength = 1010;
newWavelength = minWavelength:5:maxWavelength;
```

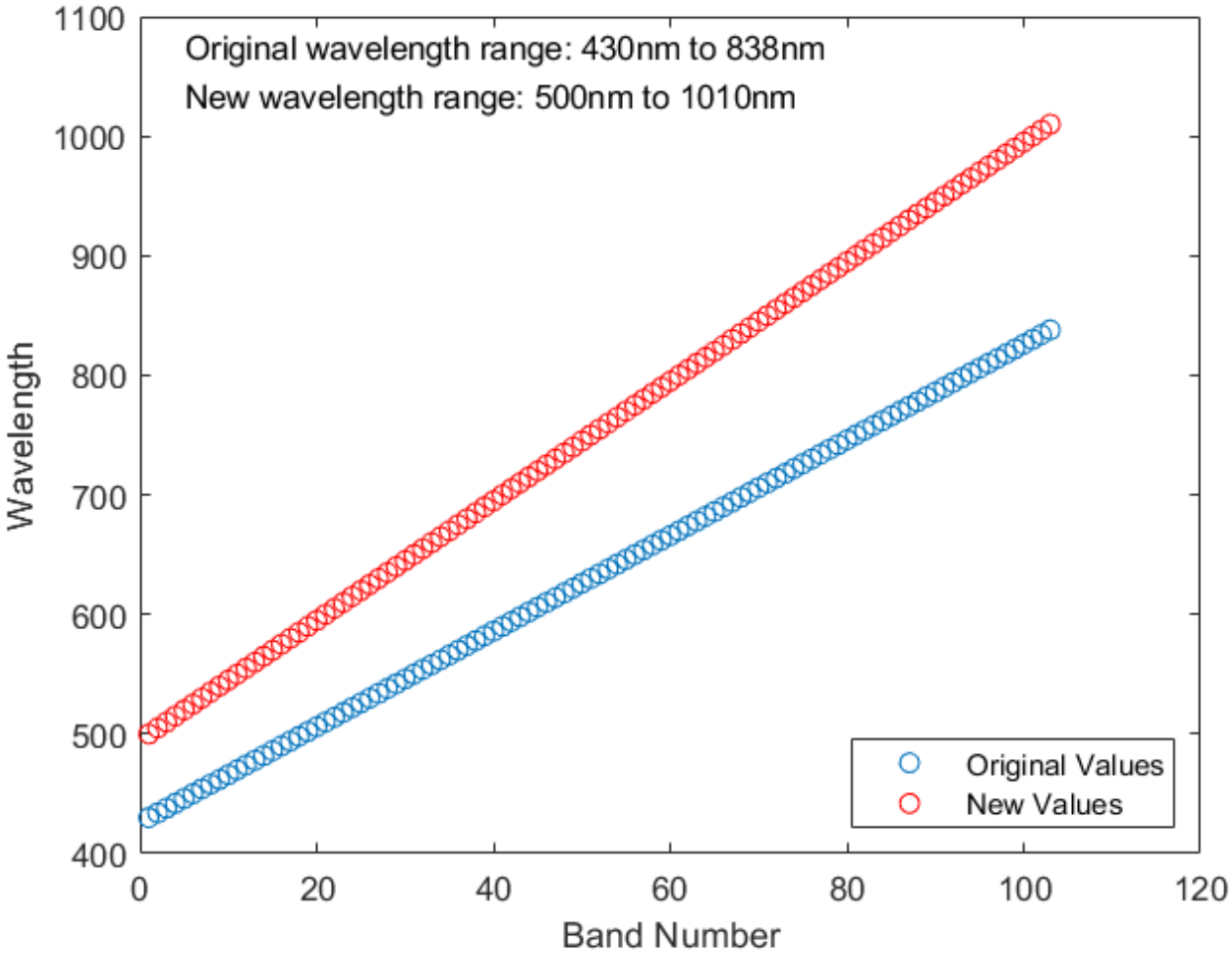
Create a new `hypercube` object with the new wavelength values.

```
newhcube = hypercube('paviaU.hdr',newWavelength);
```

Plot the old and the new wavelength values. Display the wavelength range.

```
figure
plot(hcube.Wavelength,'o')
hold on
plot(newhcube.Wavelength,'or')
xlabel('Band Number')
ylabel('Wavelength')
str1 = ['Original wavelength range: ' num2str(min(hcube.Wavelength))  'nm to ' num2str(max(hcube
```

```
text(5,1075,str1)
str2 = ['New wavelength range: ' num2str(min(newhcube.Wavelength))  'nm to ' num2str(max(newhcube
text(5,1035,str2)
legend('Original Values','New Values','Location','SouthEast')
```



Create Data Cube from Numeric Inputs

Read an RGB image into the workspace. An RGB image contains three spectral channels: red, green, and blue channels.

```
image = imread('peppers.png');
```

Specify the center wavelength values for the red, green, and blue channels as 700, 530, and 470 nanometers (nm) respectively.

```
wavelength = [700 530 470];
```

Create a `hypercube` object using the image and the wavelength values.

```
hcube = hypercube(image,wavelength)
```

```
hcube =
  hypercube with properties:

      DataCube: [384×512×3 uint8]
    Wavelength: [3×1 double]
      Metadata: [1×1 struct]
```

Modify Metadata Property

Read a hyperspectral data into the workspace and inspect its properties.

```
hcube = hypercube('paviaU.dat');
```

Inspect the `Metadata` property of the `hypercube` object.

```
hcube.Metadata
```

```
ans = struct with fields:
             Filename: "B:\matlab\toolbox\images\supportpackages\hyperspectral\hyperdata\paviaU.h
          FileModDate: "25-Feb-2020 03:59:34"
             FileSize: 654
               Format: "HDR"
        FormatVersion: ''
           SensorType: [0x0 string]
          Description: [0x0 string]
      AcquisitionTime: [0x0 string]
         RasterFormat: "ENVI"
               Height: 610
                Width: 340
                Bands: 103
             DataType: "double"
           Interleave: "bsq"
         HeaderOffset: 0
            ByteOrder: "ieee-le"
            BandNames: [0x0 string]
                 FWHM: []
                 Gain: []
               Offset: []
      ReflectanceGain: []
    ReflectanceOffset: []
              BadBands: []
           CloudCover: []
           SunAzimuth: []
         SunElevation: []
      SolarIrradiance: []
     EarthSunDistance: []
      WavelengthUnits: "Nanometers"
```

Find and remove the empty fields from the metadata.

```
metadata = hcube.Metadata;
fields = fieldnames(metadata);
```

```
indx = find(structfun(@isempty,metadata)==1);
newMetadata = rmfield(metadata,fields(indx));
```

Set the value for the `AcquistionTime` field to current date.

```
currentDate = string(datetime("now",Format="yyyy-MM-dd"));
newMetadata.AcquistionTime = currentDate;
```

Create a `hypercube` object with the new metadata. The `DataCube` and `Wavelength` properties of the new `hypercube` object is same as that of the input data.

```
nhcube = hypercube(hcube.DataCube,hcube.Wavelength,newMetadata);
```

Inspect the `Metadata` property of the new `hypercube` object.

```
nhcube.Metadata
```

```
ans = struct with fields:
              Height: 610
               Width: 340
               Bands: 103
            DataType: "double"
          Interleave: "bsq"
        HeaderOffset: 0
           ByteOrder: "ieee-le"
      AcquistionTime: "2022-02-26"
    WavelengthUnits: "Nanometers"
```

See Also

`enviinfo` | `fippi` | `nfindr` | `estimateAbundanceLS` | `ndvi` | `countEndmembersHFC` | `hyperpca` | `hypermnf` | `inverseProjection` | `multibandread` | `multibandwrite` | `nitfread` | `h5read`

Introduced in R2020a

hypermnf

Maximum noise fraction transform of hyperspectral data

Syntax

```
outputDataCube = hypermnf(inputData,numComponents)
[outputDataCube,coeff] = hypermnf(inputData,numComponents)
[___] = hypermnf(inputData,numComponents,'MeanCentered',flag)
```

Description

`outputDataCube = hypermnf(inputData,numComponents)` computes specified number of principal component bands `numComponents` by using the maximum noise fraction (MNF) transform. To achieve spectral dimensionality reduction, the specified number of principal components must be less than the number of spectral bands in the input data cube.

The components derived using MNF transform are also called non-adjusted principal components and the MNF transform arranges principal components (PC) in the decreasing order of PC image quality.

`[outputDataCube,coeff] = hypermnf(inputData,numComponents)` also returns the MNF coefficients estimated across the spectral bands of the input data cube.

`[___] = hypermnf(inputData,numComponents,'MeanCentered',flag)` computes MNF transform from mean centered spectral bands. The option for mean centering each spectral band in the input data cube is specified by `flag`.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Reduce Spectral Dimensionality of Data Cube Using MNF

Read a hyperspectral data into the workspace.

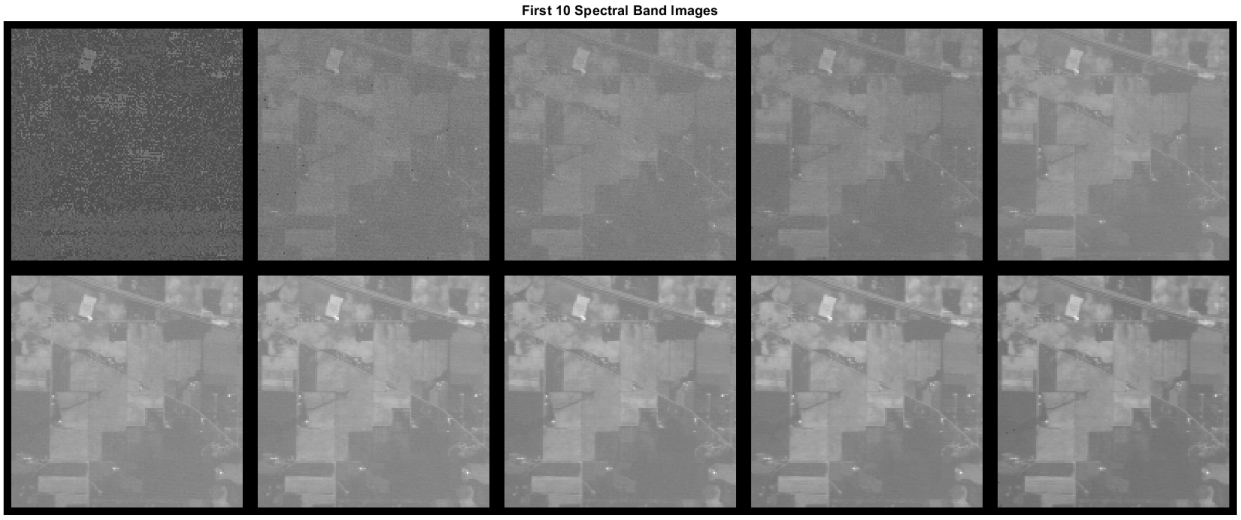
```
hcube = hypercube('indian_pines.dat');
```

Compute 10 principal component bands of hyperspectral data and the associated transformation coefficients.

```
[outputDataCube,coeff] = hypermnf(hcube,10);
```

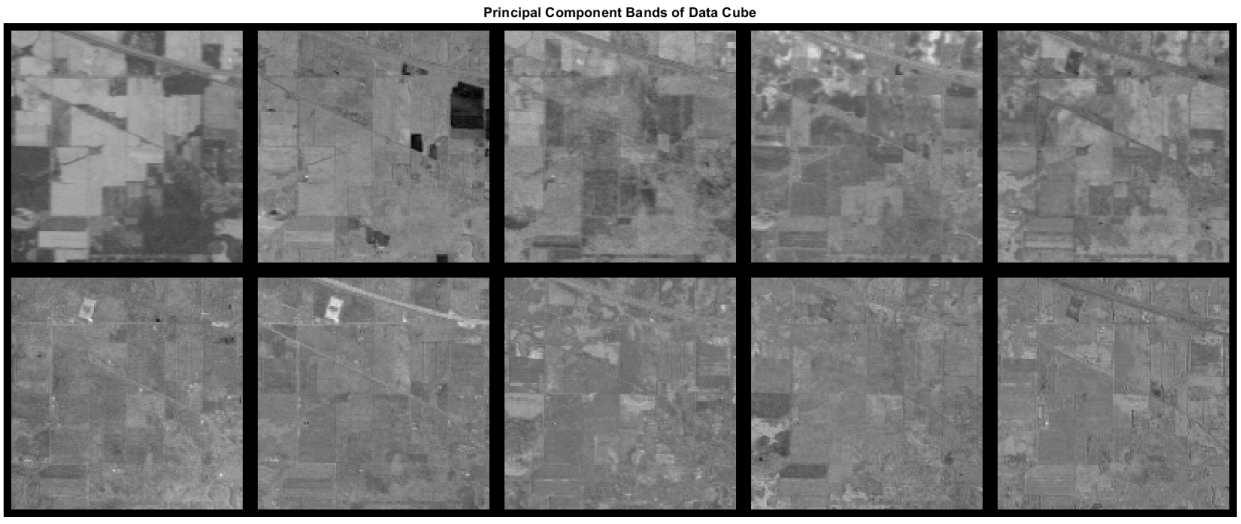
Display the first 10 spectral bands in input data cube.

```
figure
montage(hcube.DataCube(:,:,1:10),'BorderSize',[10 10],'Size',[2 5],'DisplayRange',[]);
title('First 10 Spectral Band Images')
```



For the purpose of visualization, rescale the principal component values to lie in the range [0, 1]. Display all the principal component bands extracted from the data cube. The principal component bands are arranged in the order of decreasing image quality (or increasing noise level).

```
figure
rescalePC = rescale(outputDataCube,0,1);
montage(rescalePC,'BorderSize',[10 10],'Size',[2 5]);
title('Principal Component Bands of Data Cube')
```



Input Arguments

`inputData` — Input hyperspectral data

3-D numeric array | `hypercube` object

Input hyperspectral data, specified as a 3-D numeric array that represent the hyperspectral data cube of size *M*-by-*N*-by-*C* or `hypercube` object. If the input is a `hypercube` object, the function reads the data cube stored in the `DataCube` property of the object. The hyperspectral data cube must be real and non-sparse.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`numComponents` — Number of principal components to extract
positive integer scalar

Number of principal component bands to extract from the data cube, specified as a positive integer scalar. The value must be less than or equal to the number of spectral bands in the input data cube.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`flag` — Indicator for mean centering spectral bands
`true`or `1` (default) | `false` or `0`

Indicator for mean centering spectral bands, specified as one of these values:

- `true` or `1` — Mean center each spectral bands in the input data cube by subtracting the mean of spectral bands before computing the MNF transform.
- `false` or `0` — Compute principal component bands without mean centering the spectral bands in the input data cube.

Data Types: `logical`

Output Arguments

`outputDataCube` — MNF transformed data cube
3-D numeric array

MNF transformed data cube, returned as a 3-D numeric array of size *M*-by-*N*-by-*numComponents*. The spatial dimension of the output data cube is same as that of the input data cube. The spectral dimension of the output data cube is equal to the number of principal components specified at the input.

If the input data type is double, the output data type is also double. Otherwise, the output data type is single.

Data Types: `single` | `double`

`coeff` — MNF coefficients
matrix

MNF coefficients, returned as a matrix of size *C*-by-*numComponents*. *C* is the number of spectral bands in the input data cube. Each column of `coeff` contains the coefficients for one principal component. The columns are in the order of principal component image quality.

If the input data type is double, the data type of `coeff` is also double. Otherwise, the data type is single.

Data Types: `single` | `double`

References

[1] Green, A.A., M. Berman, P. Switzer, and M.D. Craig. “A Transformation for Ordering Multispectral Data in Terms of Image Quality with Implications for Noise Removal.” *IEEE Transactions on Geoscience and Remote Sensing* 26, no. 1 (January 1988): 65–74. https://doi.org/10.1109/36.3001.

See Also

`hyperpca` | `inverseProjection` | `hypercube`

Introduced in R2020a

hyperpca

Principal component analysis of hyperspectral data

Syntax

```
outputDataCube = hyperpca(inputData,numComponents)
[outputDataCube,coeff] = hyperpca( ___ )
[outputDataCube,coeff,var] = hyperpca( ___ )
[ ___ ] = hyperpca( ___ ,Name,Value)
```

Description

`outputDataCube = hyperpca(inputData,numComponents)` computes the specified number of principal components from the spectral bands of the hyperspectral data cube. The function returns a new data cube that contains the principal component bands. The number of spectral bands in the output data cube is equal to the number of specified principal components `numComponents`. To achieve spectral dimensionality reduction, the specified number of principal components must be less than the number of spectral bands in the hyperspectral data cube `inputData`.

`[outputDataCube,coeff] = hyperpca( ___ )` also returns the principal component coefficients estimated across the spectral dimension of the hyperspectral data cube.

`[outputDataCube,coeff,var] = hyperpca( ___ )` returns the percentage of variance retained by the principal component bands in addition to the output arguments mention in the previous syntaxes.

`[ ___ ] = hyperpca( ___ ,Name,Value)` specifies the principal component analysis (PCA) method and additional options by using the name-value pair arguments.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Reduce Spectral Dimensionality of Data Cube Using PCA

Read a hyperspectral data into the workspace.

```
hcube = hypercube('paviaU.dat');
```

Compute the principal component bands of the hyperspectral data cube. Specify the number of principal components to extract as 10. By default, the function uses the singular value decomposition (SVD) method for extracting principal components.

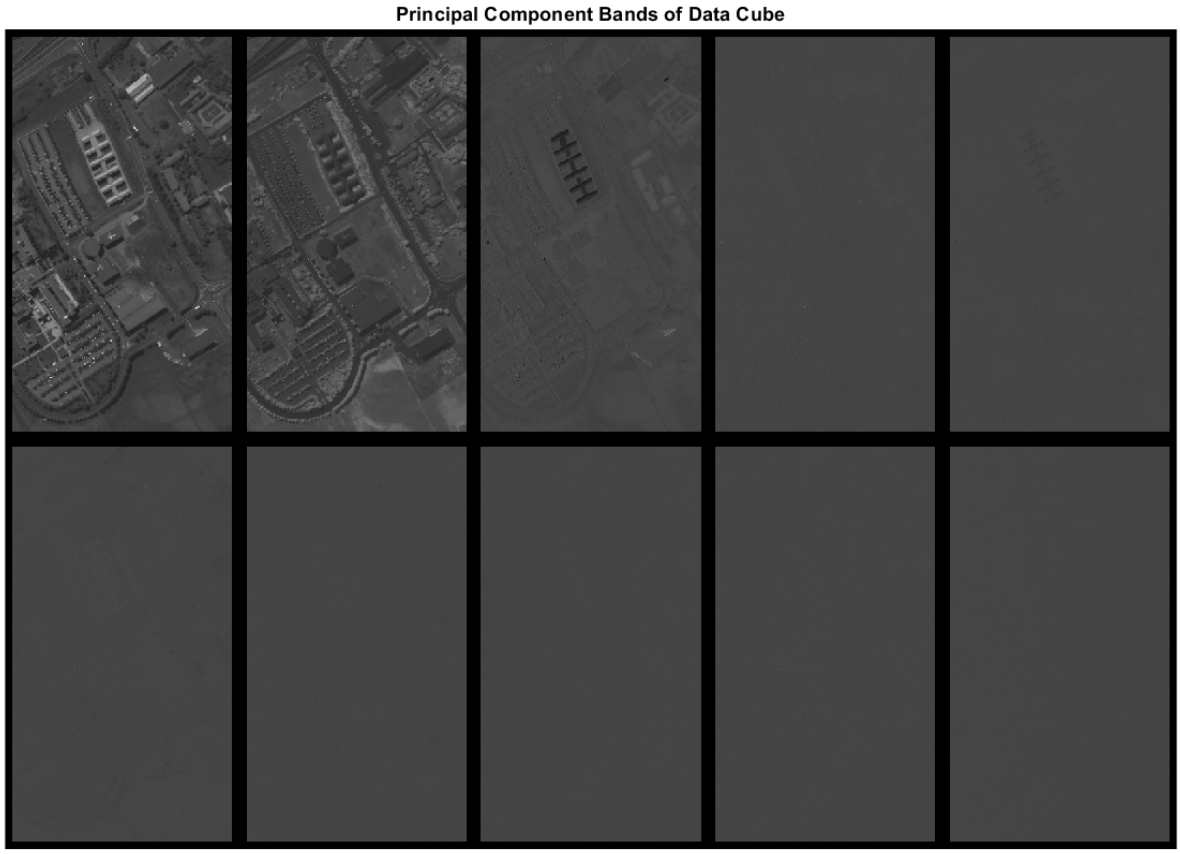
```
reducedDataCube = hyperpca(hcube,10);
```

Display the first 10 spectral bands in input data cube.

```
figure
montage(hcube.DataCube(:,:,1:10),'BorderSize',[10 10],'Size',[2 5],'DisplayRange',[]);
```

For the purpose of visualization, rescale the principal component values to lie in the range [0, 1]. Display all the principal component bands extracted from the data cube.

```
figure
rescalePC = rescale(reducedDataCube,0,1);
montage(rescalePC,'BorderSize',[10 10],'Size',[2 5]);
title('Principal Component Bands of Data Cube')
```



Derive Principal Component Coefficients of Data Cube

Read a hyperspectral data into the workspace.

```
hcube = hypercube('paviaU.dat');
```

Perform PCA of input data cube using Eigen value decomposition. Specify the number of principal components to extract as 3. Derive the principal component (PC) bands, coefficients, and retained variance.

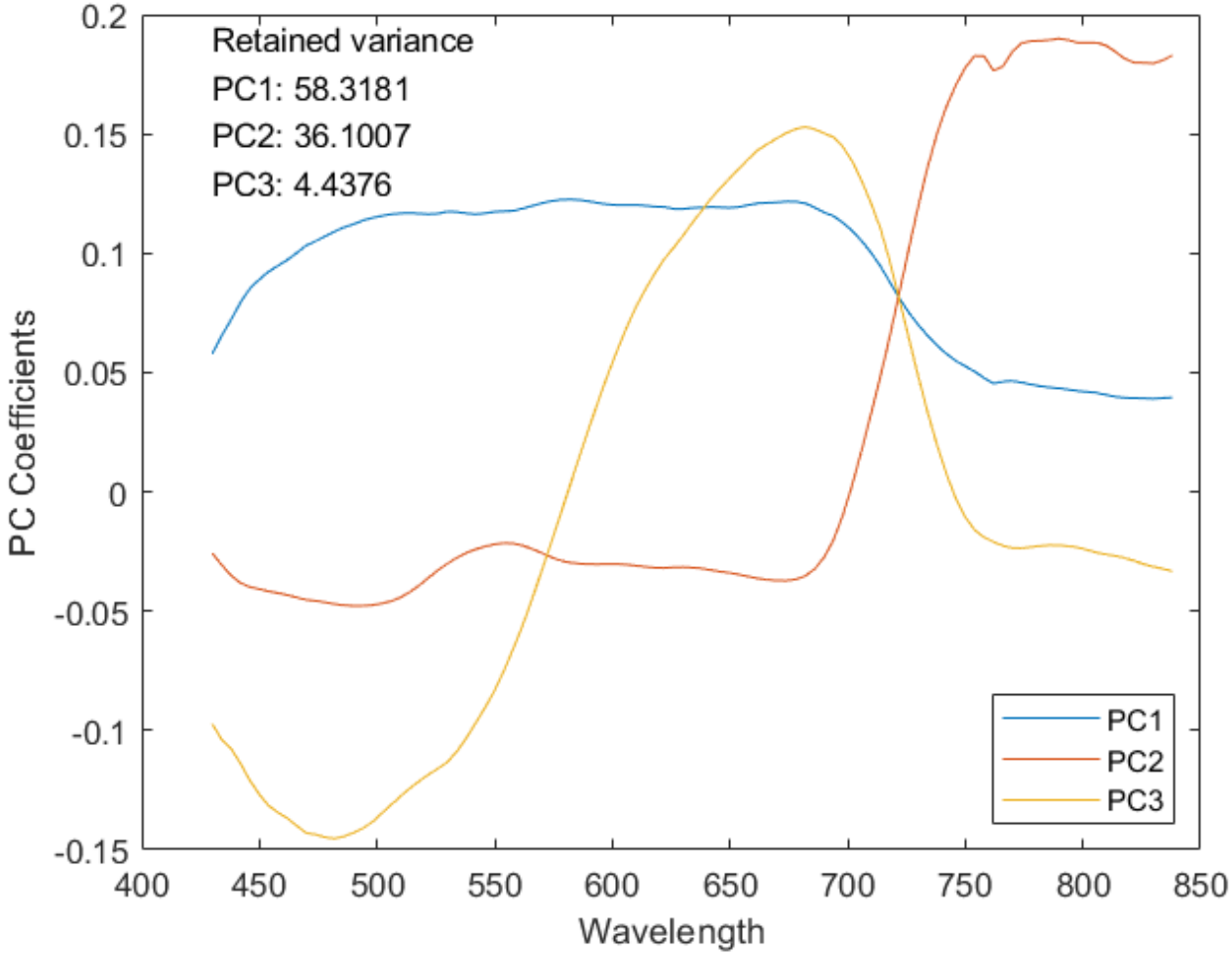
```
[outputDataCube,coeff,var] = hyperpca(hcube,3,'Method','Eig');
```

For the purpose of visualization, rescale the principal component values to lie in the range [0, 1]. Display all the principal component bands extracted from the data cube.

```
figure
rescalePC = rescale(outputDataCube,0,1);
montage(rescalePC,'BorderSize',[10 10],'Size',[1 3]);
title('Principal Component Bands of Data Cube')
```

Plot the principal component coefficients and display the percentage of variance retained by each of the principal components. The summation of retained variance values imply that almost 99% of the information in input hyperspectral data is captured by the 3 principal components.

```
figure
plot(hcube.Wavelength,coeff);
legend(['PC1';'PC2';'PC3'],'Location','SouthEast')
text(430,0.19,'Retained variance');
text(430,0.17,['PC1: ' num2str(var(1))])
text(430,0.15,['PC2: ' num2str(var(2))])
text(430,0.13,['PC3: ' num2str(var(3))])
xlabel('Wavelength')
ylabel('PC Coefficients')
```



Input Arguments

`inputData` — Input hyperspectral data

3-D numeric array | `hypercube` object

Input hyperspectral data, specified as a 3-D numeric array that represent the hyperspectral data cube of size *M*-by-*N*-by-*C* or `hypercube` object. If the input is a `hypercube` object, the function reads the data from the `DataCube` property of the object. The hyperspectral data cube must be real and non-sparse.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`numComponents` — Number of principal component bands to extract

positive integer scalar

Number of principal component bands to extract from the data cube, specified as a positive integer scalar. The value must be less than or equal to the number of spectral bands in the input data cube.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `hyperpca(hcube,10,'Method','eig')`

`Method` — PCA method

`'svd'` (default) | `'eig'`

Method for PCA, specified as one of these values:

- `'svd'` — To derive principal components by using the singular value decomposition method.
- `'eig'` — To derive principal components by using the eigen value decomposition method.

Data Types: `char` | `string`

`MeanCentered` — Indicator for mean centering spectral bands

`true` or `1` (default) | `false` or `0`

Indicator for mean centering spectral bands, specified as one of these values:

- `true` or 1 — To center each spectral bands in the input data cube by subtracting the mean of spectral bands before computing the principal component bands.
- `false` or 0 — To compute principal component bands without mean centering the spectral bands in the input data cube.

Data Types: `logical`

Output Arguments

`outputDataCube` — PCA transformed data cube

3-D numeric array

PCA transformed data cube, returned as a 3-D numeric array of size *M*-by-*N*-by-*numComponents*. The spatial dimension of the output data cube is same as that of the input data cube. The spectral dimension of the output data cube is equal to the specified number of principal components `numComponents`.

If the input data type is double, the output data type is also double. Otherwise, the output data type is single.

Data Types: `single` | `double`

`coeff` — Principal component coefficients

matrix

Principal component coefficients, returned as a matrix of size *C*-by-*numComponents*. *C* is the number of spectral bands in the input data cube. Each column of `coeff` contains the coefficients for one principal component. The columns are in the order of descending component variance.

If the input data type is double, the data type of `coeff` is also double. Otherwise, the data type is single.

Data Types: `single` | `double`

`var` — Variance retained by each principal component
vector

Variance retained by each principal component, returned as a vector of length equal to `numComponents`. The retained variance specifies the total percentage of variance explained by each principal component.

If the input data type is double, the data type of `var` is also double. Otherwise, the data type is single.

Data Types: `single` | `double`

See Also

`hypermnf` | `inverseProjection` | `hypercube`

Introduced in R2020a

iarr

Apply internal average relative reflectance (IARR) correction to hyperspectral data cube

Syntax

```
correctedData = iarr(inputData)
```

Description

`correctedData = iarr(inputData)` applies IARR based correction to the hyperspectral data `inputData`. The IARR method computes the mean spectrum from the entire hyperspectral dataset, then divides the spectrum of each pixel by the mean spectrum.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Apply IARR Correction to Hyperspectral Data

Read hyperspectral data into the workspace. This data is from the EO-1 Hyperion sensor, with pixel values in digital numbers.

```
hcube = hypercube('EO1H0440342002212110PY_cropped.hdr');
```

Convert the digital numbers to top of atmosphere (TOA) reflectance values.

```
hcube_toa = dn2reflectance(hcube);
```

Apply IARR correction to the reflectance data.

```
hcube_iarr = iarr(hcube_toa);
```

Input Argument

`inputData` — Input hyperspectral data

`hypercube` object | *M*-by-*N*-by-*C* numeric array

Input hyperspectral data, specified as one of these options:

- `hypercube` object — The `DataCube` property of the `hypercube` object stores the hyperspectral data cube.
- *M*-by-*N*-by-*C* numeric array — *M* and *N* are the number of rows and columns of pixels in the hyperspectral data, respectively. *C* is the number of spectral bands in the hyperspectral data.

The input pixel values can be digital numbers, TOA radiance values, or TOA reflectance values. To convert a hypercube containing digital numbers to a hypercube containing TOA radiance or TOA reflectance data, use the `dn2radiance` or `dn2reflectance` function, respectively.

Output Arguments

`correctedData` — Corrected hyperspectral data

`hypercube` object | *M*-by-*N*-by-*C* numeric array

Corrected hyperspectral data, returned as a `hypercube` object or *M*-by-*N*-by-*C* numeric array consistent with the input data, `inputData`. If the input data in `inputData` is of data type `double`, then the corrected data is also of data type `double`. Otherwise, the corrected data is of data type `single`.

References

[1] Kruse, Fred A. “Use of Airborne Imaging Spectrometer Data to Map Minerals Associated with Hydrothermally Altered Rocks in the Northern Grapevine Mountains, Nevada, and California.” *Remote Sensing of Environment* 24, no. 1 (February 1988): 31–51. https://doi.org/10.1016/0034-4257(88)90004-1.

See Also

`hypercube` | `logResiduals` | `flatField` | `subtractDarkPixel` | `empiricalLine` | `reduceSmile` | `sharc`

Introduced in R2020b

inverseProjection

Reconstruct data cube from principal component bands

Syntax

```
reconstructedData = inverseProjection(pcDataCube,coeff)
```

Description

`reconstructedData = inverseProjection(pcDataCube,coeff)` reconstructs the original spectral bands in a hyperspectral data cube from the PCA (principal component analysis) or MNF (maximum noise fraction) transformed data cube and their related coefficients.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Reconstruct Data From Principal Component Coefficients

Read a hyperspectral data into the workspace.

```
hcube = hypercube('indian_pines.dat');
```

Extract 10 principal component bands and the transformation coefficients using the principal component analysis method.

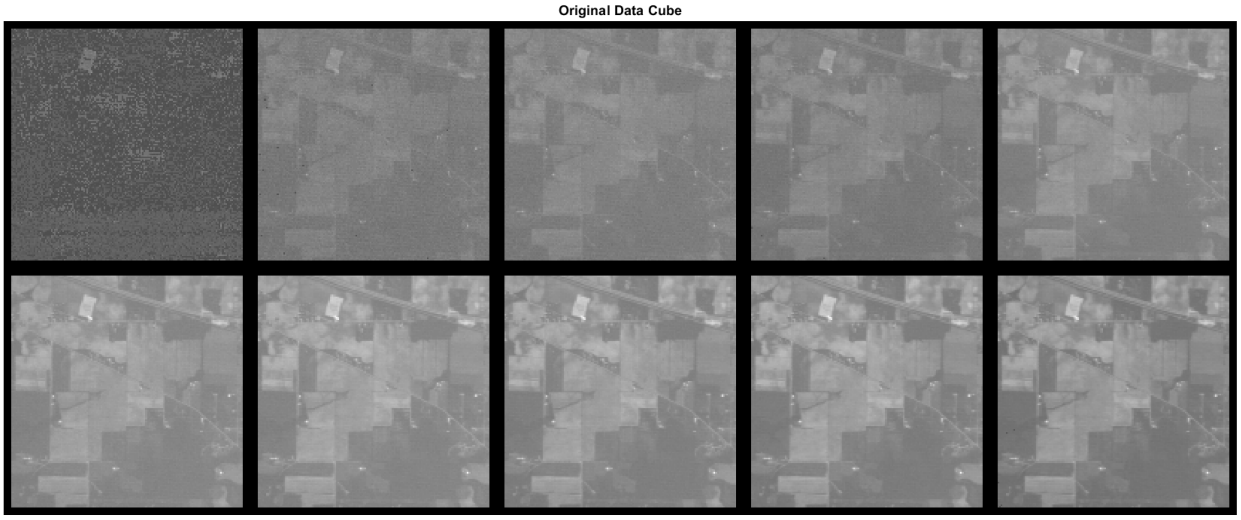
```
[pcDataCube,coeff] = hyperpca(hcube,10);
```

Reconstruct the original data from 10 principal component bands.

```
reconstructedData = inverseProjection(pcDataCube,coeff);
```

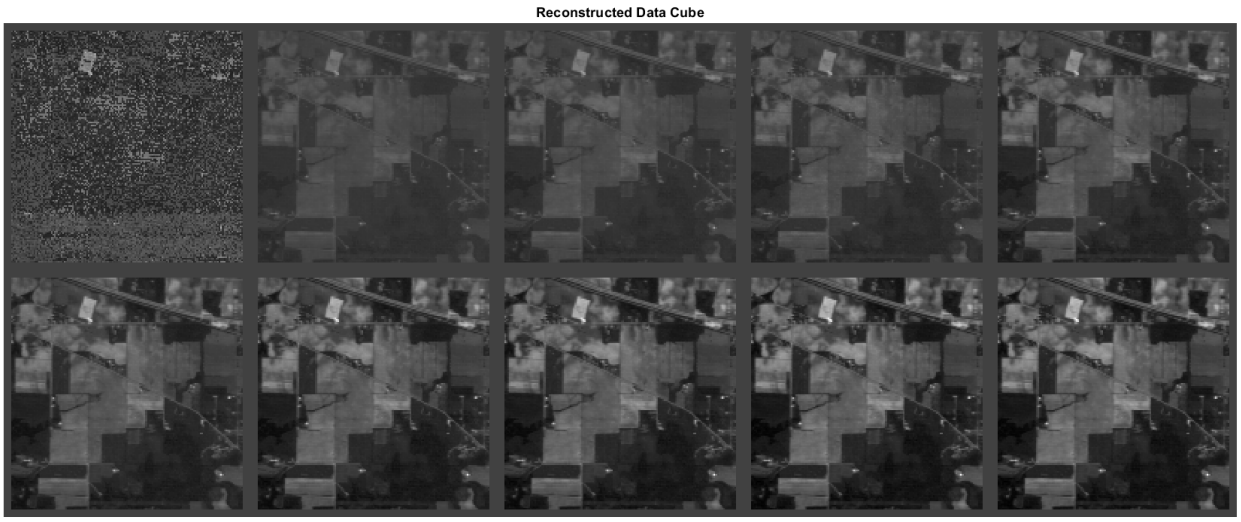
Display the first 10 spectral bands in input data cube.

```
figure
montage(hcube.DataCube(:,:,1:10),'BorderSize',[10 10],'Size',[2 5],'DisplayRange',[]);
title('Original Data Cube')
```



Display the first 10 spectral bands in the reconstructed data cube.

```
figure
montage(reconstructedData(:,:,1:10),'BorderSize',[10 10],'Size',[2 5],'DisplayRange',[]);
title('Reconstructed Data Cube')
```



Input Arguments

pcDataCube — PCA or MNF transformed data cube

3-D numeric array

PCA or MNF transformed data cube, specified as a 3-D numeric array of size *M*-by-*N*-by-*P*. The PCA or MNF transformed data cube of a hyperspectral data cube is computed using the `hyperpca` or

`hypermnf` functions respectively. *P* specifies the number of principal component bands in the transformed data cube.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`coeff` — PCA or MNF coefficients
matrix

PCA or MNF coefficients, specified as a matrix of size *C*-by-*P*.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`reconstructedData` — Reconstructed data cube
3-D numeric array

Reconstructed data cube, returned as a 3-D numeric array of size *M*-*N*-by-*C*. The data type of the reconstructed data cube is same as that of the transformed data cube at the input.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

See Also
`hypermnf` | `hyperpca`

Introduced in R2020a

jmsam

Measure spectral similarity using Jeffries Matusita-Spectral Angle Mapper method

Syntax

```
score = jmsam(inputData,refSpectrum)
score = jmsam(testSpectrum,refSpectrum)
```

Description

`score = jmsam(inputData,refSpectrum)` measures the spectral similarity between the spectrum of each pixel in the hyperspectral data `inputData` and the specified reference spectrum `refSpectrum` by using Jeffries Matusita-Spectral Angle Mapper (JMSAM) method. Use this syntax to identify different regions or materials in a hyperspectral data cube. For information about the JMSAM method, see “More About” on page 1-3300.

`score = jmsam(testSpectrum,refSpectrum)` measures the spectral similarity between the specified test spectrum `testSpectrum` and reference spectrum `refSpectrum` by using the JMSAM method. Use this syntax to compare the spectral signature of an unknown material against the reference spectrum or to compute spectral variability between two spectral signatures.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Distinguish Hyperspectral Regions Using JM-SAM Hybrid Measure

Read hyperspectral data into the workspace.

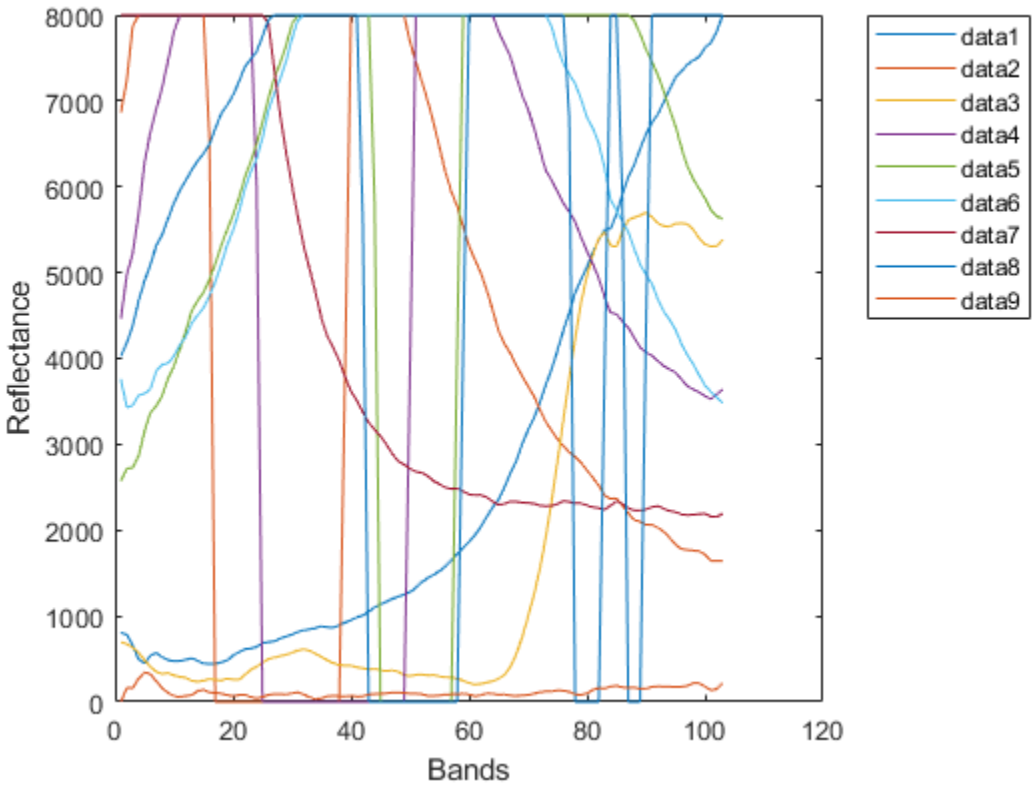
```
hcube = hypercube('paviaU.dat');
```

Extract the first 9 endmember spectral signatures from the data cube by using the NFINDR algorithm.

```
numEndmembers = 9;
endmembers = nfindr(hcube,numEndmembers);
```

Plot the spectral signatures of the extracted endmembers.

```
figure
plot(endmembers)
xlabel('Bands')
ylabel('Reflectance')
legend('Location','Bestoutside')
```



Compute the JM-SAM distance between each endmember and the spectrum of each pixel in the data cube.

```
score = zeros(size(hcube.DataCube,1),size(hcube.DataCube,2),numEndmembers);
for i = 1:numEndmembers
    score(:,:,i) = jmsam(hcube,endmembers(:,i));
end
```

Compute the minimum score value from the distance scores obtained for each pixel spectrum with respect to all the endmembers. The index of each minimum score identifies the endmember spectrum to which a pixel spectrum exhibits maximum similarity. An index value, *n*, at the spatial location (*x*, *y*) in the score matrix indicates that the spectral signature of the pixel at spatial location (*x*, *y*) in the data cube best matches the spectral signature of the *n*th endmember.

```
[~,matchingIdx] = min(score,[],3);
```

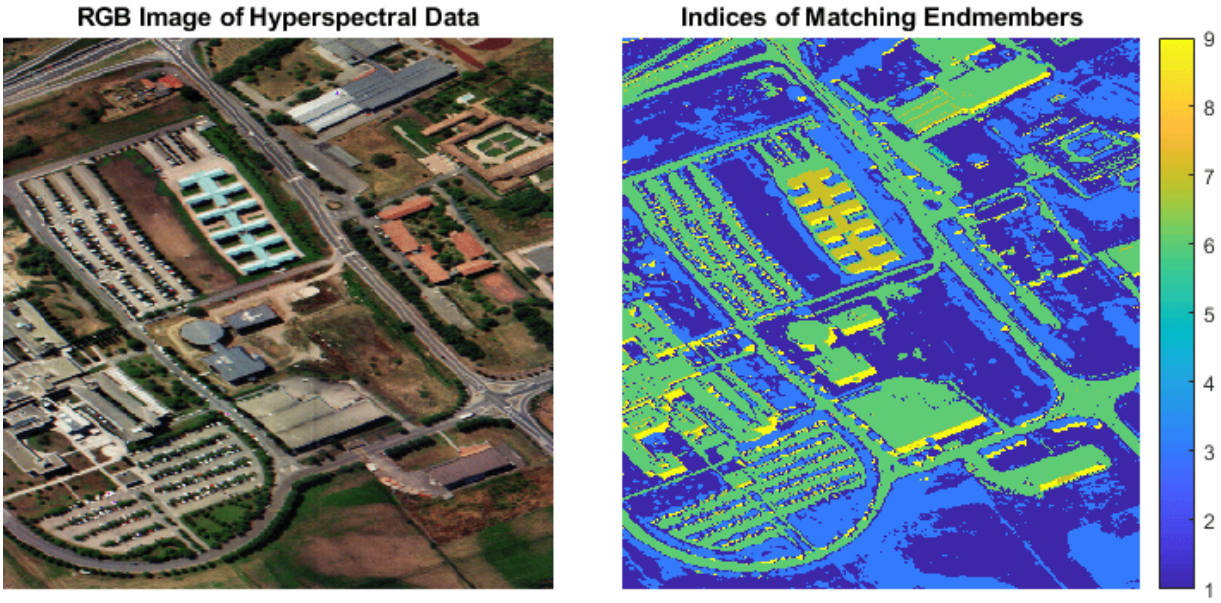
Estimate an RGB image of the input data by using the `colorize` function.

```
rgbImg = colorize(hcube,'Method','rgb','ContrastStretching',true);
```

Display both the RGB image and the matrix of matched index values.

```
figure('Position',[0 0 800 400])
subplot('Position',[0 0.1 0.4 0.8])
imagesc(rgbImg)
axis off
title('RGB Image of Hyperspectral Data')
```

```
subplot('Position',[0.45 0.1 0.45 0.8])
imagesc(matchingIdx)
axis off
title('Indices of Matching Endmembers')
colorbar
```



Determine Similarity of Endmember Spectra Using JM-SAM

Read hyperspectral data into the workspace.

```
hcube = hypercube('indian_pines.dat');
```

Find the first 10 endmembers of the hyperspectral data.

```
numEndmembers = 10;
endmembers = nfindr(hcube,numEndmembers);
```

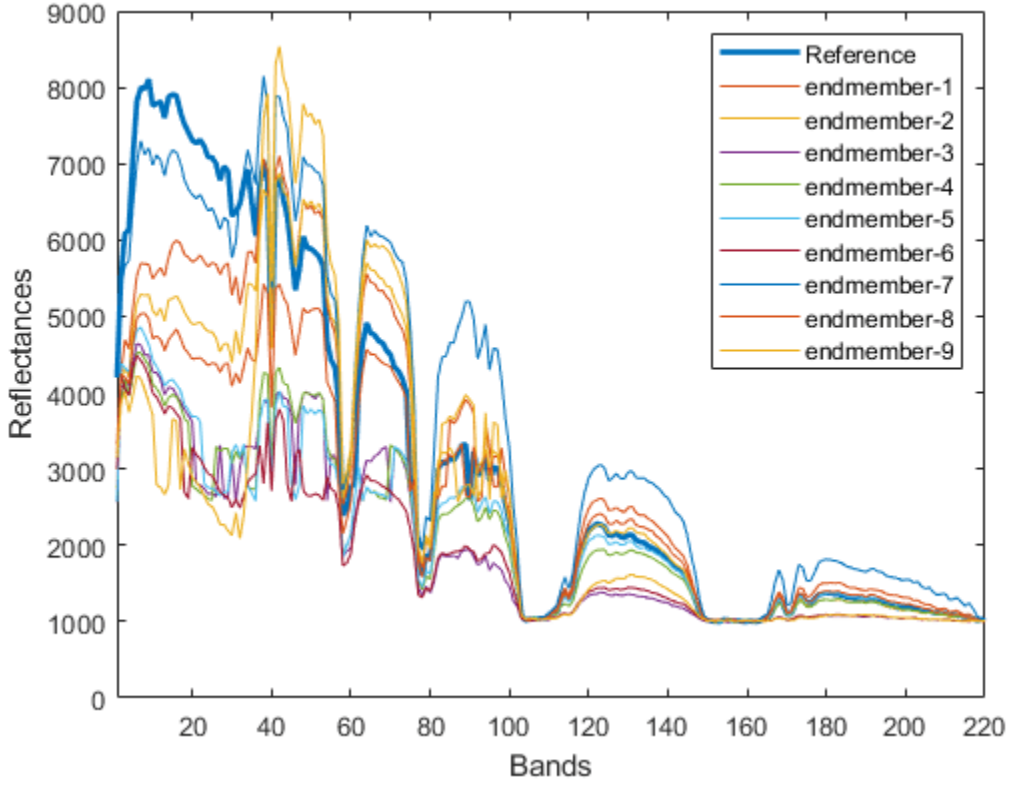
Consider the first endmember as the reference spectrum and the rest of the endmembers as the test spectra.

```
refSpectrum = endmembers(:,1);
testSpectra = endmembers(:,2:end);
```

Plot the reference spectrum and other endmember spectra.

```
figure
plot(refSpectrum,'LineWidth',2)
hold on
plot(testSpectra)
hold off
```

```
label = cell(1,numEndmembers-1);
label{1} = 'Reference';
for itr = 1:numEndmembers-1
    label{itr+1} = ['endmember-' num2str(itr)];
end
xlabel('Bands')
ylabel('Reflectances')
legend(label)
xlim([1 size(hcube.DataCube,3)]);
```



Compute the JM-SAM score between the reference and test spectra.

```
score = zeros(1,numEndmembers-1);
for itr = 1:numEndmembers-1
    score(itr) = jmsam(testSpectra(:,itr),refSpectrum);
end
```

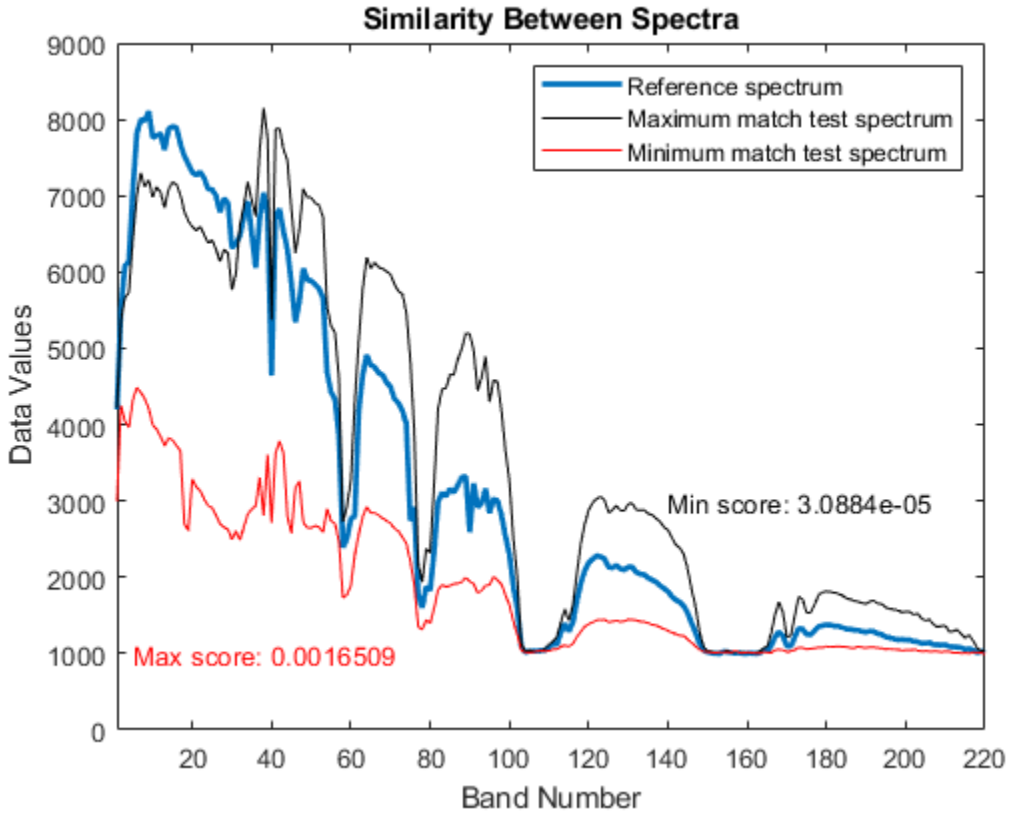
Find the test spectrum that exhibit maximum similarity (minimum distance) to the reference spectrum. Then find the test spectrum that exhibit minimum similarity (maximum distance) to the reference spectrum.

```
[minval,minidx] = min(score);
maxMatch = testSpectra(:,minidx);
[maxval,maxidx] = max(score);
minMatch = testSpectra(:,maxidx);
```

Plot the reference spectrum, the maximum similarity, and the minimum similarity test spectrum. The test spectrum with the minimum score value indicates highest similarity to the reference endmember.

On the other hand, the test spectrum with the maximum score value has the highest spectral variability and characterises the spectral behaviour of two different materials.

```
figure
plot(refSpectrum,'LineWidth',2)
hold on
plot(maxMatch,'k')
plot(minMatch,'r')
xlabel('Band Number')
ylabel('Data Values')
xlim([1 size(hcube.DataCube,3)]);
legend('Reference spectrum','Maximum match test spectrum','Minimum match test spectrum')
title('Similarity Between Spectra')
text(5,1000,['Max score: ' num2str(maxval)],'Color','r')
text(140,3000,['Min score: ' num2str(minval)],'Color','k')
```



Input Arguments

`inputData` — Input hyperspectral data

`hypercube` object | 3-D numeric array

Input hyperspectral data, specified as a `hypercube` object or a 3-D numeric array containing the data cube. If the input is a `hypercube` object, the data is read from the `DataCube` property of the object.

`testSpectrum` — Test spectrum

C-element vector

Test spectrum, specified as a *C*-element vector. The test spectrum is the spectral signature of an unknown region or material.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`refSpectrum` — Reference spectrum
C-element vector

Reference spectrum, specified as a *C*-element vector. The reference spectrum is the spectral signature of a known region or material. The function matches the test spectrum against these values.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`score` — JMSAM score
scalar | matrix

JMSAM score, returned as a scalar or matrix. The output is a

- scalar — If you specify the `testSpectrum` input argument. The function matches the test spectral signature against the reference spectral signature and returns a scalar value. Both the test and the reference spectra must be vectors of same length.
- matrix — If you specify the `inputData` input argument. The function matches the spectral signature of each pixel in the data cube against the reference spectral signature and returns a matrix. If the data cube is of size *M*-by-*N*-by-*C* and the reference spectra is a vector of length *C*, the output matrix is of size *M*-by-*N*.

A smaller JMSAM score indicates a strong match between the test signature and the reference signature.

Data Types: `single` | `double`

More About

Jeffries Matusita-Spectral Angle Mapper (JMSAM)

The JMSAM method computes spectral similarity based on the Jeffries Matusita (JM) and SAM distances between two spectra. Let *r* and *t* be the reference and test spectra respectively.

First, compute the JM distance,

where *B* is the Bhattacharyya distance,

$$B = \frac{1}{8}(\mu_t - \mu_r)^T\left[\frac{\sigma_t + \sigma_r}{2}\right]^{-1}(\mu_t - \mu_r) + \frac{1}{2}\ln\left[\frac{\left|\frac{\sigma_t + \sigma_r}{2}\right|}{\sqrt{|\sigma_t||\sigma_r|}}\right]$$

μ_r and μ_t are the mean values of the reference and test spectra respectively. σ_r and σ_t are the covariance values of the reference and test spectra respectively.

Then, compute the SAM value α by using the test spectra *t* and a reference spectra *r* of length *C*,

Finally, compute the JMSAM score as:

$$JMSAM = JM_{distance} \times \tan(\alpha)$$

References

[1] Padma, S., and S. Sanjeevi. “Jeffries Matusita Based Mixed-Measure for Improved Spectral Matching in Hyperspectral Image Analysis.” *International Journal of Applied Earth Observation and Geoinformation* 32 (October 2014): 138–51. https://doi.org/10.1016/j.jag.2014.04.001.

See Also

`spectralMatch` | `readEcostressSig` | `sid` | `hypercube` | `sidsam` | `ns3` | `sam`

Introduced in R2020b

logResiduals

Apply log residual correction to hyperspectral data cube

Syntax

```
correctedData = logResiduals(inputData)
```

Description

`correctedData = logResiduals(inputData)` applies a log residual correction to the hyperspectral data `inputData`. The log residual method divides the spectrum of each pixel by the spectral geometric mean and the spatial geometric mean, which produces a pseudoreflectance data set.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Apply Log Residual Correction to Hyperspectral Data

Read hyperspectral data into the workspace. This data is from the EO-1 Hyperion sensor, with pixel values in digital numbers.

```
hcube = hypercube('EO1H0440342002212110PY_cropped.hdr');
```

Convert the digital numbers to top of atmosphere (TOA) reflectance values.

```
hcube_toa = dn2reflectance(hcube);
```

Apply a log residual correction to the reflectance data.

```
hcube_logR = logResiduals(hcube_toa);
```

Input Arguments

`inputData` — Input hyperspectral data

`hypercube` object | *M*-by-*N*-by-*C* numeric array

Input hyperspectral data, specified as one of these options:

- `hypercube` object — The `DataCube` property of the `hypercube` object stores the hyperspectral data cube.
- *M*-by-*N*-by-*C* numeric array — *M* and *N* are the number of rows and columns of pixels in the hyperspectral data, respectively. *C* is the number of spectral bands in the hyperspectral data.

The input pixel values can be digital numbers, TOA radiance values, or TOA reflectance values. To convert a hypercube containing digital numbers to a hypercube containing TOA radiance or TOA reflectance data, use the `dn2radiance` or `dn2reflectance` function, respectively.

Output Arguments

`correctedData` — Corrected hyperspectral data

`hypercube` object | *M*-by-*N*-by-*C* numeric array

Corrected hyperspectral data, returned as a `hypercube` object or *M*-by-*N*-by-*C* numeric array consistent with the input data, `inputData`. If the input data in `inputData` is of data type `double`, then the corrected data is also of data type `double`. Otherwise, the corrected data is of data type `single`.

References

[1] Green, A. A. and M. D. Craig. "Analysis of Aircraft Spectrometer Data with Logarithmic Residuals." In *Proceedings of the Airborne Imaging Spectrometer Data Analysis Workshop*, ed. Gregg Vane and Alexander F. H. Goetz, 111–119. Pasadena: Jet Propulsion Laboratory, 1985.

See Also

`hypercube` | `iarr` | `flatField` | `subtractDarkPixel` | `empiricalLine` | `reduceSmile` | `sharc`

Introduced in R2020b

nfindr

Extract endmember signatures using N-FINDR

Syntax

```
endmembers = nfindr(inputData,numEndmembers)
endmembers = nfindr(inputData,numEndmembers,Name,Value)
```

Description

`endmembers = nfindr(inputData,numEndmembers)` extracts endmember signatures from hyperspectral data `cube` by using the N-finder (N-FINDR) algorithm. `numEndmembers` is the number of endmember signatures to be extracted using N-FINDR algorithm. For more information about the N-FINDR method, see “Algorithms” on page 1-3308.

`endmembers = nfindr(inputData,numEndmembers,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input arguments in the previous syntax. Use this syntax to set the options for number of iterations and dimensionality reduction.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Extract Endmembers Using N-FINDR Method

Read a hyperspectral data into the workspace.

```
hcube = hypercube('paviaU.hdr');
```

Find the number of spectrally distinct endmembers present in the hyperspectral data cube by using `countEndmembersHFC` function.

```
numEndmembers = countEndmembersHFC(hcube,'PFA',10^-7);
```

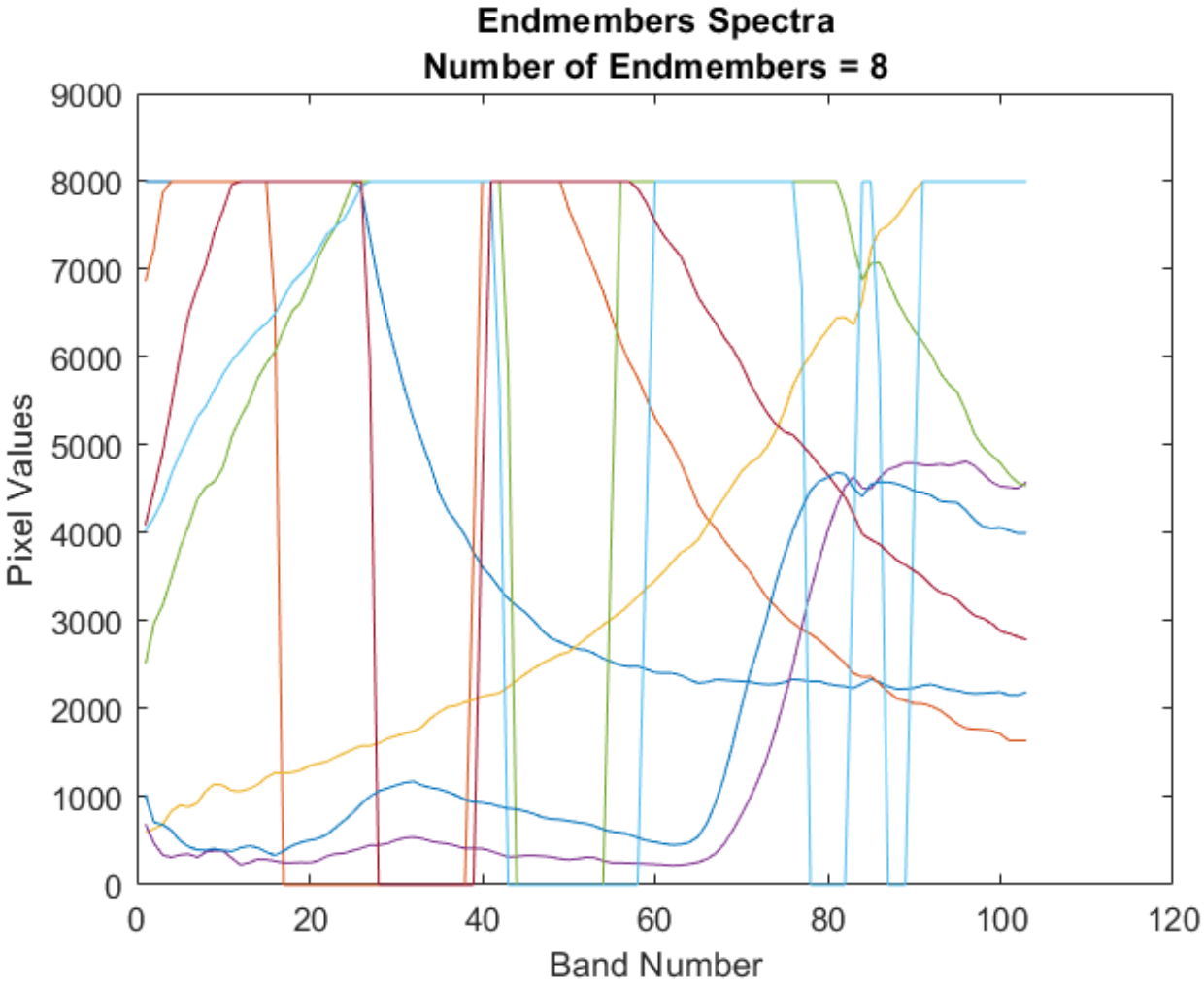
Compute the endmembers using the N-FINDR method. By default, the `nfindr` function uses maximum noise fraction (MNF) transform for preprocessing. The default value for number of iterations is 3 times the number of estimated endmembers.

```
endmembers = nfindr(hcube.DataCube,numEndmembers);
```

Plot the endmembers of the hyperspectral data.

```
figure
plot(endmembers)
xlabel('Band Number')
ylabel('Pixel Values')
```

```
ylim([0 9000])
title({'Endmembers Spectra',['Number of Endmembers = ' num2str(numEndmembers)]});
```



Set N-FINDR Parameter Values for Finding Endmembers

Read a hyperspectral data into the workspace.

```
hcube = hypercube('paviaU.hdr');
```

Find the number of spectrally distinct endmembers present in the hyperspectral data cube by using `countEndmembersHFC` function.

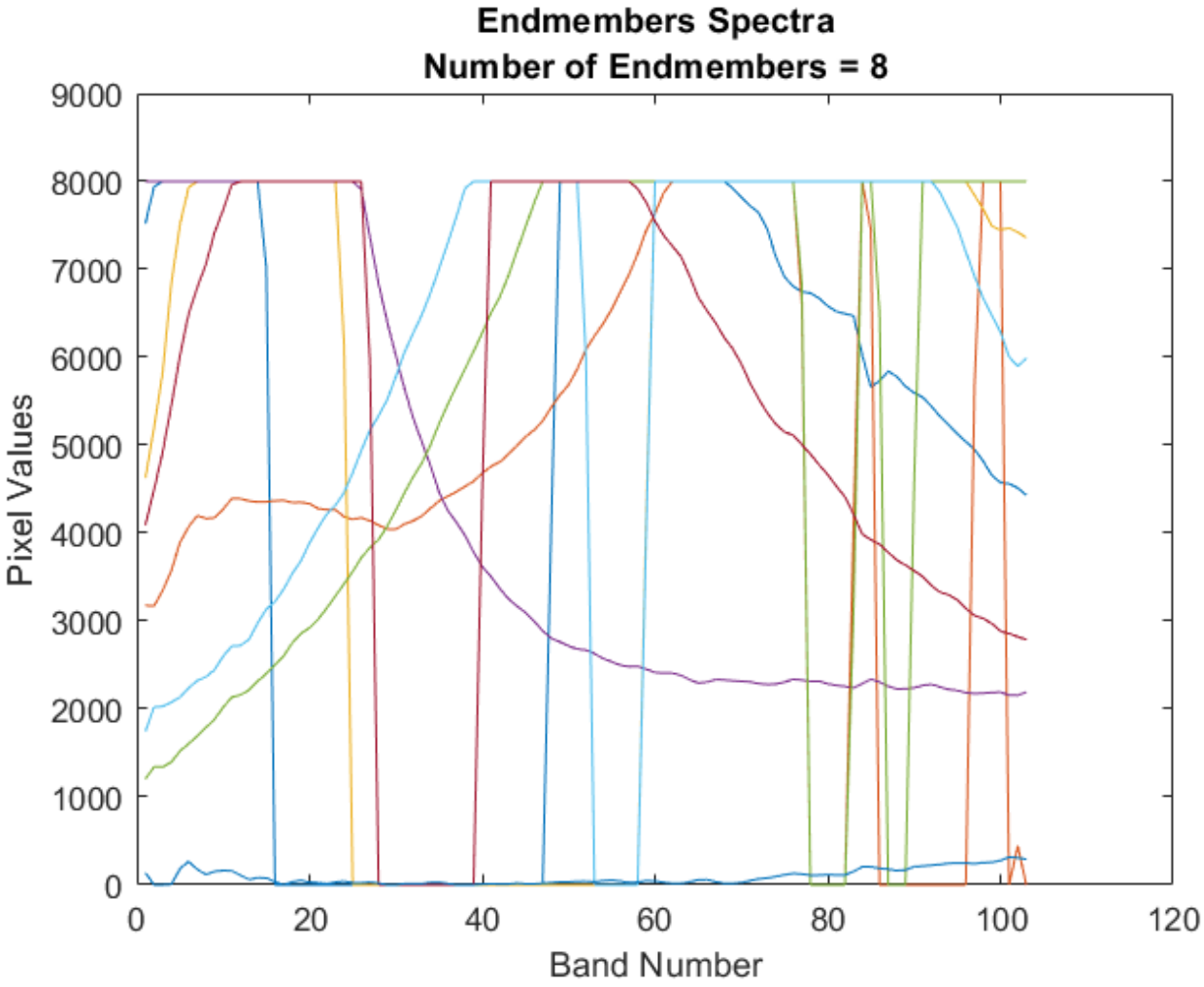
```
numEndmembers = countEndmembersHFC(hcube,'PFA',10^-7);
```

Compute the endmembers using the N-FINDR method. Specify the value for number of iterations as 1000. Select principal component analysis (PCA) as the dimensionality reduction method for preprocessing.

```
endmembers = nfindr(hcube.DataCube,numEndmembers,'NumIterations',1000,'ReductionMethod','PCA');
```

Plot the endmembers of the hyperspectral data.

```
figure
plot(endmembers)
xlabel('Band Number')
ylabel('Pixel Values')
ylim([0 9000])
title({'Endmembers Spectra',['Number of Endmembers = ' num2str(numEndmembers)]});
```



Input Arguments

`inputData` — Input hyperspectral data

3-D numeric array | `hypercube` object

Input hyperspectral data, specified as an 3-D numeric array or a `hypercube` object. If the input is a `hypercube` object, then the function reads the hyperspectral data from its `DataCube` property.

The hyperspectral data is an numeric array of size *M*-by-*N*-by-*C*. *M* and *N* are the number of rows and columns in the hyperspectral data respectively. *C* is the number of spectral bands in the hyperspectral data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`numEndmembers` — Number of endmembers
positive scalar integer

Number of endmembers to extract, specified as a positive scalar integer. The value must be in the range [1 *C*]. *C* is the number of spectral bands in the input hyperspectral data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `nfindr(cube,7,'NumIterations',100,'Method','None')`

`NumIterations` — Number of iterations
3*P* (default) | positive scalar integer

Number of iterations, specified as a positive scalar integer. The default value is 3*P*. *P* is the number of endmember signatures to be extracted. The computation time of the algorithm increases with the increase in the number of iterations.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ReductionMethod` — Dimensionality reduction method
`'MNF'` (default) | `'PCA'`

Dimensionality reduction method, specified as one of these values:

- `'MNF'` — To perform dimensionality reduction by using the maximum noise fraction (MNF) method. This is the default.
- `'PCA'` — To perform dimensionality reduction by using the principal component analysis (PCA) method.

If you specify this argument, the function first reduces the spectral dimension of the input data by using the specified method. Then, it computes the endmember signatures from the reduced data.

Data Types: `char` | `string`

Output Arguments

`endmembers` — Endmember signatures
C-by-*P* matrix

Endmember signatures, returned as a matrix of size *C*-by-*P* and datatype same as the datatype of the input hyperspectral data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Algorithms

N-FINDR is an iterative approach for finding the endmembers of a hyperspectral data. The method assumes that the volume of a simplex formed by the endmembers (purest pixels) is larger than any other volume defined by any other combination of pixels [1]. The steps involved are as follows:

1. Compute principal component bands and reduce the spectral dimensionality of the input data by using MNF or PCA. The number of principal component bands to be extracted is set equal to the number of endmembers to be extracted. The endmembers are extracted from the principal component bands.
2. Randomly select n number of pixel spectra from the reduced data as initial set of endmembers.
3. For iteration 1, denote the initial set of endmembers as $\left\{e_1^{(1)},\ e_2^{(1)},\ \cdots,\ e_p^{(1)}\right\}$.

   Consider the endmembers as vertices of a simplex and compute the volume by using

   $$V\left(E^{(1)}\right) = \left|\det\left(E^{(1)}\right)\right|$$

   where $E^{(1)} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ e_1^{(1)} & e_2^{(1)} & \cdots & e_p^{(1)} \end{bmatrix}$.
4. For iteration 2, Select a new pixel spectra **r**, such that $r \notin \left\{e_1^{(1)},\ e_2^{(1)},\ \cdots,\ e_p^{(1)}\right\}$.
5. Replace each endmember in the set with **r** and compute the volume of the simplex $V(\mathbf{E}^{(2)})$.
6. Replace the i^{th} endmember in the set with **r**, if the computed volume $V(\mathbf{E}^{(2)})$ is greater than $V(\mathbf{E}^{(1)})$. This results in an updated set of endmembers. For example, if $i = 2$, the new set of endmembers derived at the end of the second iteration is $\left\{e_1^{(2)},\ e_2^{(2)} = r,\ \cdots,\ e_p^{(2)}\right\}$.
7. For each iteration, select a new pixel spectra **r** and repeat steps 5 and 6. Each iteration results in an update set of endmembers. The iteration ends when the total number of iterations reaches the specified value `NumIterations`.

References

[1] Winter, Michael E. “N-FINDR: An Algorithm for Fast Autonomous Spectral End-Member Determination in Hyperspectral Data.” *Proc. SPIE Imaging Spectrometry V* 3753, (October 1999): 266–75. https://doi.org/10.1117/12.366289.

See Also

`hypercube` | `ppi` | `countEndmembersHFC` | `fippi`

Introduced in R2020a

ndvi

Normalized difference vegetation index

Syntax

```
output = ndvi(hcube)
output = ndvi(hcube,'BlockSize',blocksize)
```

Description

`output = ndvi(hcube)` computes the normalized difference vegetation index (NDVI) value for each pixel in the data cube and returns an NDVI image. The NDVI image displays the vegetation cover regions of the input hyperspectral data. The function computes the NDVI value using the red (R) band and the near-infra red (NIR) band images in the data cube. The `ndvi` function uses the 670 nm and 800 nm band reflectance values for the red and NIR band images respectively.

`output = ndvi(hcube,'BlockSize',blocksize)` specifies the block size for block processing of the hyperspectral data cube by using the name-value pair argument `'BlockSize'`.

The function divides the input image into distinct blocks, processes each block, and then concatenates the processed output of each block to form the output matrix. Hyperspectral images are multi-dimensional data sets that can be too large to fit in system memory in their entirety. This can cause the system to run out of memory while running the `ndvi` function. If you encounter such an issue, perform block processing by using this syntax.

For example, `ndvi(hcube,'BlockSize',[50 50])` divides the input image into non-overlapping blocks of size 50-by-50 and then computes the NDVI values for pixels in each block.

Note To perform block processing by specifying the `'BlockSize'` name-value pair argument, you must have MATLAB R2021a or a later release.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Measure Vegetation Cover in Hyperspectral Data Using NDVI Image

Read hyperspectral data into the workspace.

```
hcube = hypercube('indian_pines.dat');
```

Compute the NDVI value for each pixel in the data cube.

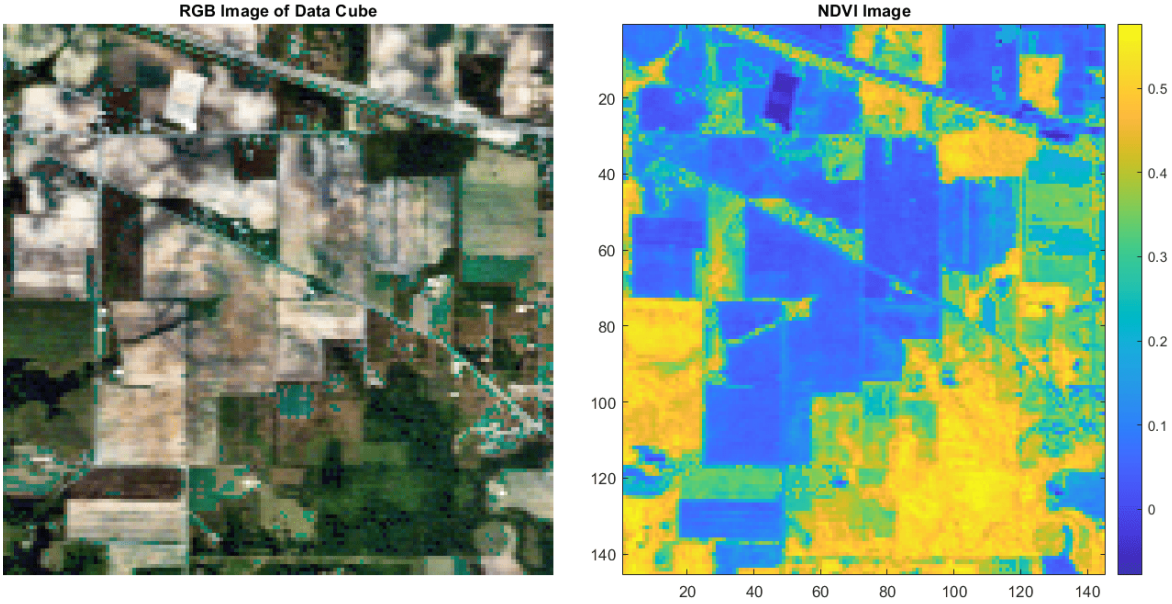
```
ndviImg = ndvi(hcube);
```

Estimate a contrast-stretched RGB image from the original data cube by using the `colorize` function.

```
rgbImg = colorize(hcube,'Method','RGB','ContrastStretching',true);
```

Display the original and the NDVI image.

```
fig = figure('Position',[0 0 1200 600]);
axes1 = axes('Parent',fig,'Position',[0 0.1 0.4 0.8]);
imshow(rgbImg,'Parent',axes1)
title('RGB Image of Data Cube')
axes2 = axes('Parent',fig,'Position',[0.45 0.1 0.4 0.8]);
imagesc(ndviImg,'Parent',axes2)
colorbar
title('NDVI Image')
```



Vegetation regions typically have NDVI values from 0.2 and 0.8. NDVI values less than or equal to 0.2 indicate the absence of vegetation. Perform thresholding of NDVI image to segment the vegetation regions. Specify the threshold value.

```
threshold = 0.2;
```

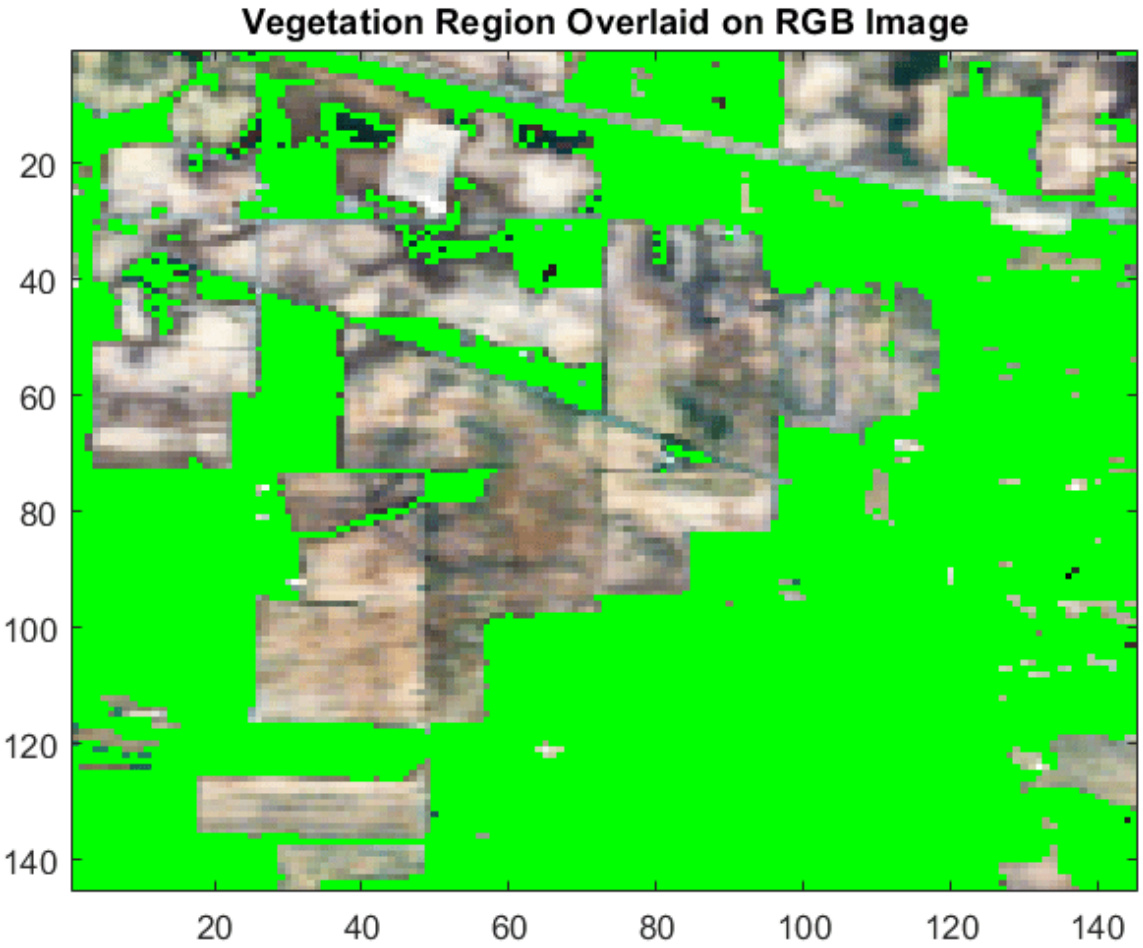
Generate a binary image with a intensity value 1 for pixels with a score greater than or equal to the specified threshold. All other pixels have a value 0. The regions in the binary image with a value of 1 correspond to the vegetation regions in the data cube with NDVI values greater than the threshold.

```
bw = ndviImg > threshold;
```

Overlay the binary image on to the RGB image and display the overlaid image.

```
overlayImg = imoverlay(rgbImg,bw,[0 1 0]);
figure
```

```
imagesc(overlayImg)
title('Vegetation Region Overlaid on RGB Image')
```



Compute the vegetation cover based on the total number of pixels in a spectral band and the number of pixels with an NDVI value greater than 0.2.

```
numVeg = find(bw == 1);
imgSize = size(hcube.DataCube,1)*size(hcube.DataCube,2);
vegetationCover = length(numVeg)/imgSize
```

```
vegetationCover = 0.5696
```

Identify Vegetation and Non-Vegetation Spectra

Load 2-D spectral data containing 20 endmembers of the Indian Pines data set into the workspace.

```
load("indian_pines_endmembers_20.mat")
```

Load the wavelength values for each band of the Indian Pines data set into the workspace.

```
load("indian_pines_wavelength.mat")
```

Reshape the 2-D spectral data into a 3-D volume data using the `reshape` function.

```
[numSpectra,spectralDim] = size(endmembers);
dataCube = reshape(endmembers,[numSpectra 1 spectralDim]);
```

Create a 3-D `hypercube` object, with a singleton dimension, by specifying the 3-D volume data `dataCube` and wavelength information `wavelength` to the `hypercube` function.

```
hCube = hypercube(dataCube,wavelength);
```

Compute the NDVI value for each spectrum in the hypercube object.

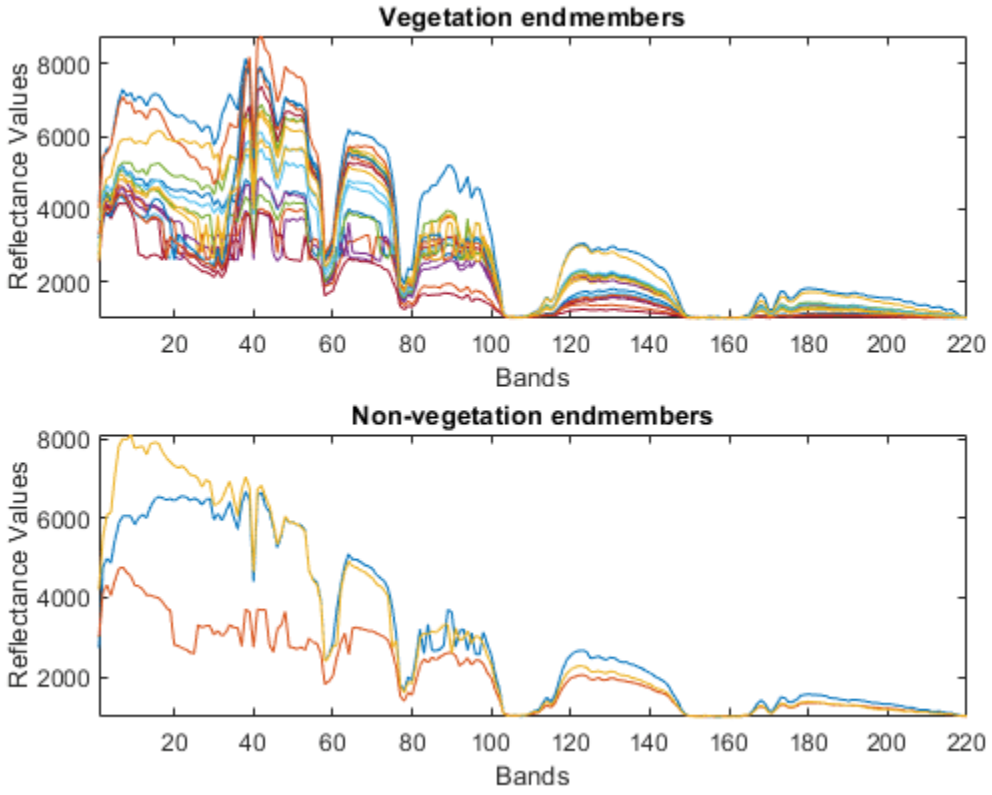
```
ndviVal = ndvi(hCube);
```

Vegetation spectra typically have NDVI values greater than zero and non-vegetation spectra typically have NDVI values less than zero. Perform thresholding to separate the vegetation and non-vegetation spectra.

```
index = ndviVal > 0;
```

Plot the vegetation and non-vegetation endmembers.

```
subplot(2,1,1)
plot(endmembers(index,:)')
title("Vegetation endmembers")
xlabel("Bands")
ylabel("Reflectance Values")
axis tight
subplot(2,1,2)
plot(endmembers(~index,:)')
title("Non-vegetation endmembers")
xlabel("Bands")
ylabel("Reflectance Values")
axis tight
```



Input Arguments

`hcube` — Input hyperspectral data

`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The functions reads the hyperspectral data cube from the `DataCube` property of the object and then computes the NDVI value of each pixel.

`blocksize` — Size of data blocks

2-element vector of positive integers

Size of the data blocks, specified as a 2-element vector of positive integers. The elements of the vector correspond to the number of rows and columns in each block, respectively. The size of the data blocks must be less than the size of the input image. Dividing the hyperspectral images into smaller blocks enables you process large data sets without running out of memory.

- If the `blocksize` value is too small, the memory usage of the function reduces at the cost of increased execution time.
- If the `blocksize` value is large or equal to the input image size, the execution time reduces at the cost of increased memory usage.

Example: `'BlockSize',[20 20]` specifies the size of each data block as 20-by-20.

Output Arguments

`output` — Output NDVI image

matrix

Output NDVI image, returned as a matrix of size *M*-by-*N*. *M* and *N* are spatial dimensions of the input data cube. If the data type of the input data cube is `double`, the output data type is also `double`. Otherwise, the output data type is `single`.

The function computes the NDVI value for each pixel as

The values are in the range [-1, 1]. A value close to 1 indicates healthy vegetation, 0 indicates unhealthy vegetation, and -1 indicates no vegetation.

Data Types: `single` | `double`

References

[1] Haboudane, D. “Hyperspectral Vegetation Indices and Novel Algorithms for Predicting Green LAI of Crop Canopies: Modeling and Validation in the Context of Precision Agriculture.” *Remote Sensing of Environment* 90, no. 3 (April 15, 2004): 337–52. https://doi.org/10.1016/j.rse.2003.12.013.

See Also

`hypercube` | `spectralMatch` | `anomalyRX`

Topics

“Identify Vegetation Regions Using Interactive NDVI Thresholding”

Introduced in R2020a

ns3

Measure normalized spectral similarity score

Syntax

```
score = ns3(inputData,refSpectrum)
score = ns3(testSpectrum,refSpectrum)
```

Description

`score = ns3(inputData,refSpectrum)` measures the spectral similarity between the spectrum of each pixel in the hyperspectral data `inputData` and the specified reference spectrum `refSpectrum` by using the normalized spectral similarity score (NS3) method. Use this syntax to identify different regions or materials in a hyperspectral data cube. For information about the NS3 method, see “More About” on page 1-3320.

`score = ns3(testSpectrum,refSpectrum)` measures the spectral similarity between the specified test spectrum `testSpectrum` and reference spectrum `refSpectrum` by using the NS3 method. Use this syntax to compare the spectral signature of an unknown material against the reference spectrum or to compute spectral variability between two spectral signatures.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Distinguish Hyperspectral Regions Using NS3 Measure

Read hyperspectral data into the workspace.

```
hcube = hypercube('jasperRidge2_R198.hdr');
```

Estimate the number of spectrally distinct endmembers in the data cube by using `countEndmembersHFC` function.

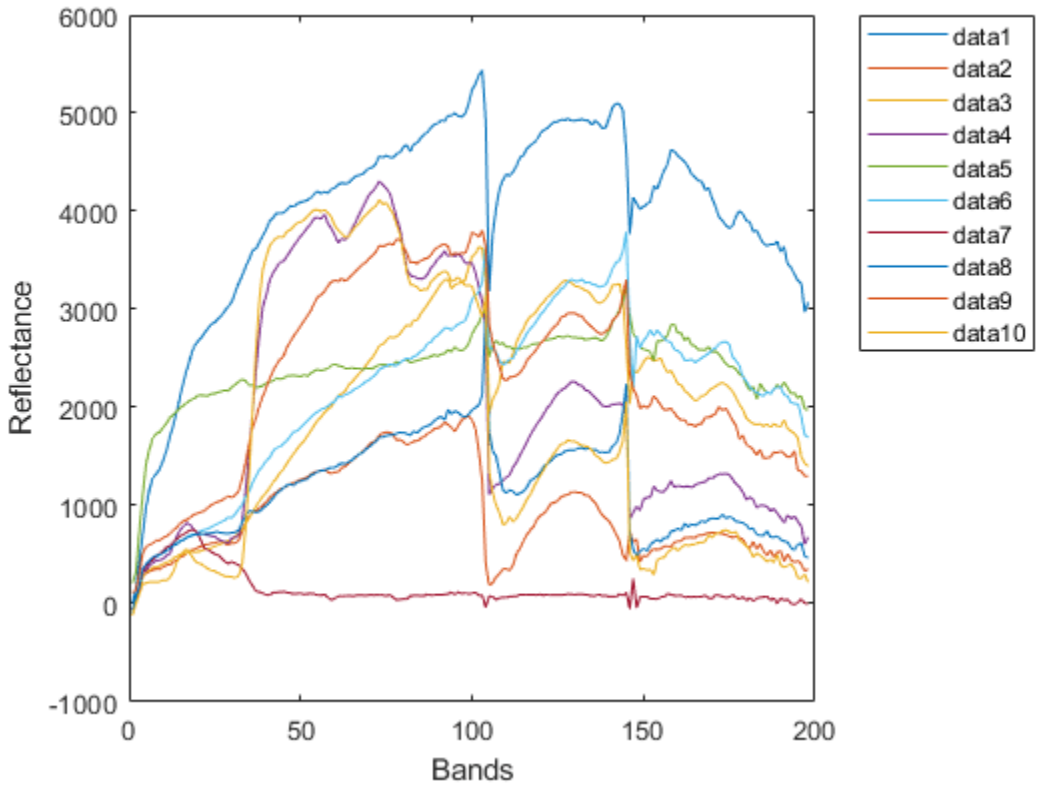
```
numEndmembers = countEndmembersHFC(hcube,'PFA',10^-7);
```

Extract the endmember spectral signatures from the data cube by using the NFINDR algorithm.

```
endmembers = nfindr(hcube,numEndmembers);
```

Plot the spectral signatures of the extracted endmembers.

```
figure
plot(endmembers)
xlabel('Bands')
ylabel('Reflectance')
legend('Location','Bestoutside')
```



Compute the NS3 distance between each endmember and the spectrum of each pixel in the data cube.

```
score = zeros(size(hcube.DataCube,1),size(hcube.DataCube,2),numEndmembers);
for i = 1:numEndmembers
    score(:,:,i) = ns3(hcube,endmembers(:,i));
end
```

Compute the minimum score value from the distance scores obtained for each pixel spectrum with respect to all the endmembers. The index of each minimum score identifies the endmember spectrum to which a pixel spectrum exhibits maximum similarity. An index value, *n*, at the spatial location (*x*, *y*) in the score matrix indicates that the spectral signature of the pixel at spatial location (*x*, *y*) in the data cube best matches the spectral signature of the *n*th endmember.

```
[~,matchingIdx] = min(score,[],3);
```

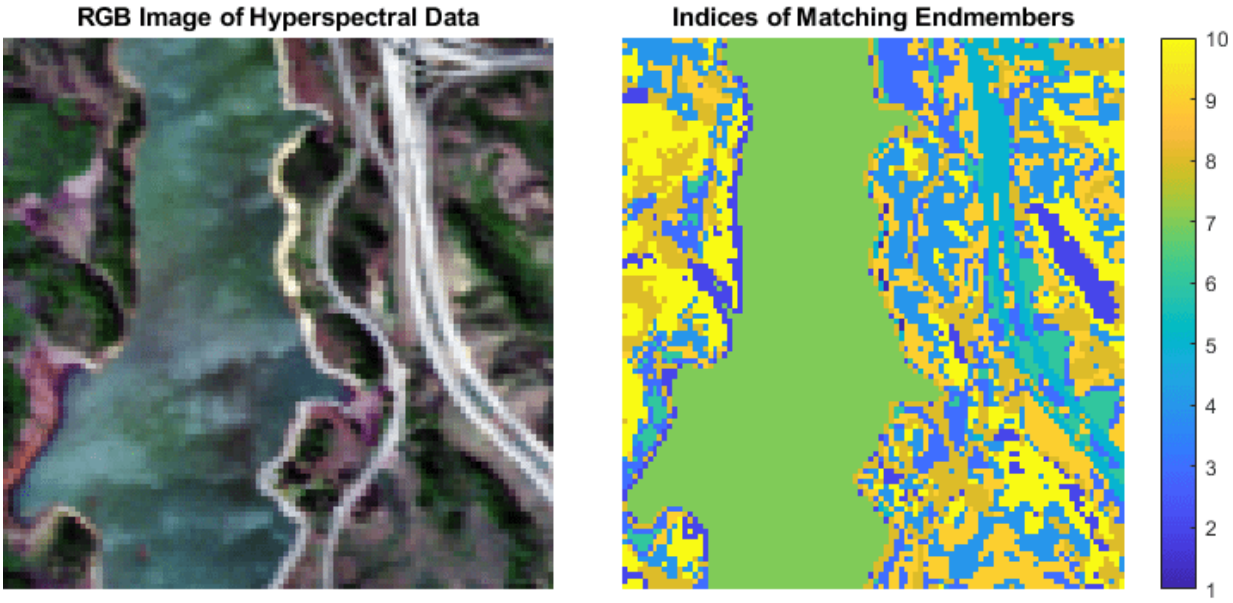
Estimate an RGB image of the input data by using the `colorize` function.

```
rgbImg = colorize(hcube,'Method','rgb','ContrastStretching',true);
```

Display both the RGB image and the matrix of matched index values.

```
figure('Position',[0 0 800 400])
subplot('Position',[0 0.1 0.4 0.8])
imagesc(rgbImg)
axis off
title('RGB Image of Hyperspectral Data')
```

```
subplot('Position',[0.45 0.1 0.45 0.8])
imagesc(matchingIdx)
axis off
title('Indices of Matching Endmembers')
colorbar
```



Determine Similarity of Endmember Spectra Using NS3

Read hyperspectral data into the workspace.

```
hcube = hypercube('jasperRidge2_R198.hdr');
```

Find the first 10 endmembers of the hyperspectral data.

```
numEndmembers = 10;
endmembers = nfindr(hcube,numEndmembers);
```

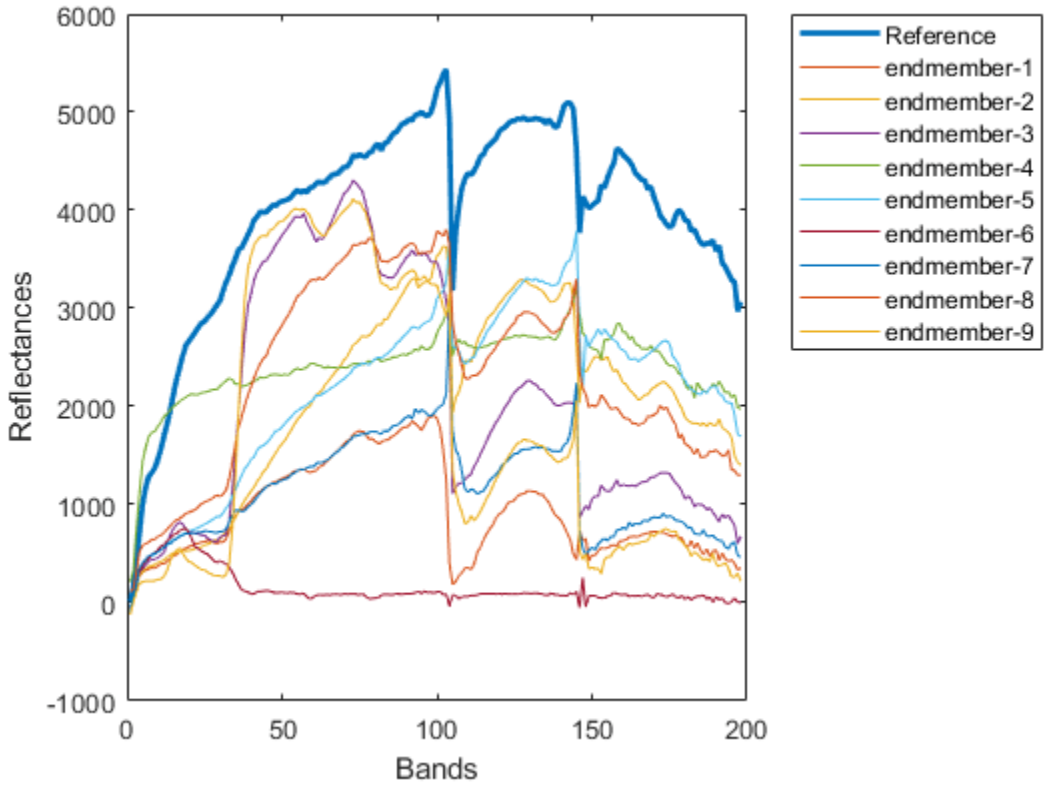
Consider the first endmember as the reference spectrum and the rest of the endmembers as the test spectrum.

```
refSpectrum = endmembers(:,1);
testSpectra = endmembers(:,2:end);
```

Plot the reference spectrum and other endmember spectra.

```
figure
plot(refSpectrum,'LineWidth',2)
hold on
plot(testSpectra)
hold off
```

```
label = cell(1,numEndmembers-1);
label{1} = 'Reference';
for itr = 1:numEndmembers-1
    label{itr+1} = ['endmember-' num2str(itr)];
end
xlabel('Bands')
ylabel('Reflectances')
legend(label,'Location','Bestoutside')
```



Compute the NS3 score between the reference and test spectra.

```
score = zeros(1,numEndmembers-1);
for itr = 1:numEndmembers-1
    score(itr) = ns3(testSpectra(:,itr),refSpectrum);
end
```

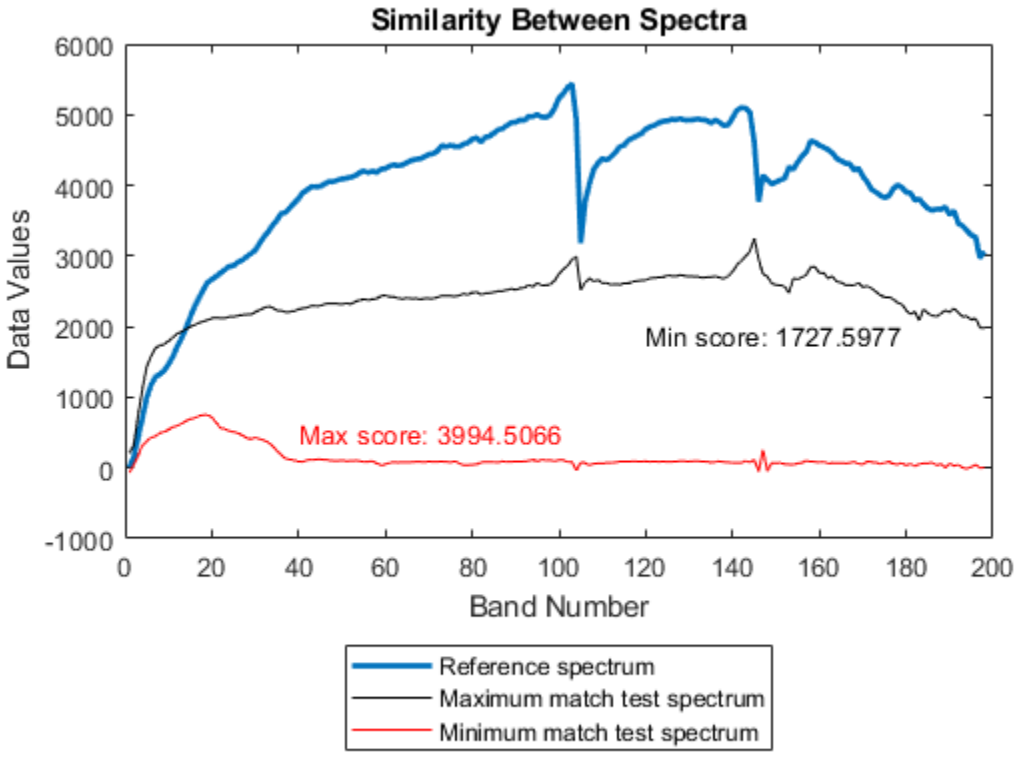
Find the test spectrum that exhibit maximum similarity (minimum distance) to the reference spectrum. Then, find the test spectrum that exhibit minimum similarity (maximum distance) to the reference spectrum.

```
[minval,minidx] = min(score);
maxMatch = testSpectra(:,minidx);
[maxval,maxidx] = max(score);
minMatch = testSpectra(:,maxidx);
```

Plot the reference spectrum, the maximum similarity, and the minimum similarity test spectrum. The test spectrum with the minimum score value indicates highest similarity to the reference endmember.

On the other hand, the test spectrum with the maximum score value has the highest spectral variability.

```
figure
plot(refSpectrum,'LineWidth',2)
hold on
plot(maxMatch,'k')
plot(minMatch,'r')
xlabel('Band Number')
ylabel('Data Values')
legend('Reference spectrum','Maximum match test spectrum','Minimum match test spectrum',...
       'Location','Southoutside')
title('Similarity Between Spectra')
text(40,500,['Max score: ' num2str(maxval)],'Color','r')
text(120,1900,['Min score: ' num2str(minval)],'Color','k')
```



Input Arguments

`inputData` — Input hyperspectral data

`hypercube` object | 3-D numeric array

Input hyperspectral data, specified as a `hypercube` object or a 3-D numeric array containing the data cube. If the input is a `hypercube` object, the data is read from the `DataCube` property of the object.

`testSpectrum` — Test spectrum

C-element vector

Test spectrum, specified as a *C*-element vector. The test spectrum is the spectral signature of an unknown region or material.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`refSpectrum` — Reference spectrum
C-element vector

Reference spectrum, specified as a *C*-element vector. The reference spectrum is the spectral signature of a known region or material. The function matches the test spectrum against these values.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`score` — NS3 score
scalar | matrix

NS3 score, returned as a scalar or matrix. The output is a

- scalar — If you specify the `testSpectrum` input argument. The function matches the test spectral signature against the reference spectral signature and returns a scalar value. Both the test and the reference spectra must be vectors of same length.
- matrix — If you specify the `inputData` input argument. The function matches the spectral signature of each pixel in the data cube against the reference spectral signature and returns a matrix. If the data cube is of size *M*-by-*N*-by-*C* and the reference spectrum is a vector of length *C*, the output matrix is of size *M*-by-*N*.

A smaller NS3 score indicates a strong match between the test signature and the reference signature.

Data Types: `single` | `double`

More About

Normalized Spectral Similarity Score (NS3)

The NS3 method computes spectral similarity based on the Euclidean and SAM distances between two spectra. Let r and t be the reference and test spectra respectively. Compute the Euclidean distance between two spectra as:

Then, compute the SAM value α

Finally, compute the NS3 score as:

References

[1] Nidamanuri, Rama Rao, and Bernd Zbell. “Normalized Spectral Similarity Score (NS^3) as an Efficient Spectral Library Searching Method for Hyperspectral Image Classification.” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 4, no. 1 (March 2011): 226–40. https://doi.org/10.1109/JSTARS.2010.2086435.

See Also

`spectralMatch` | `readEcostressSig` | `sid` | `hypercube` | `jmsam` | `sidsam` | `sam`

Introduced in R2020b

ppi

Extract endmember signatures using pixel purity index

Syntax

```
endmembers = ppi(inputData,numEndmembers)
endmembers = ppi(inputData,numEndmembers,Name,Value)
```

Description

`endmembers = ppi(inputData,numEndmembers)` extracts endmember signatures from hyperspectral data `cube` by using the pixel purity index (PPI) algorithm. `numEndmembers` is the number of endmember signatures to be extracted using PPI algorithm.

The function projects the hyperspectral data onto a set of randomly generated unit vectors. The pixels with extreme values in the direction of an unit vector are considered pure pixels and they constitute the endmembers. The value of an endmember across all the spectral bands in the input data comprises the endmember signature. For more information, see “Algorithms” on page 1-3326.

`endmembers = ppi(inputData,numEndmembers,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input arguments in the previous syntax. Use this syntax to set the options for

- number of randomly generated unit vectors to be used for projection.
- extracting endmember signatures from a reduced hyperspectral data.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Find Endmembers Using Pixel Purity Index

Read a hyperspectral data into the workspace.

```
hcube = hypercube('paviaU.hdr');
```

Find the number of spectrally distinct endmembers present in the hyperspectral data cube by using `countEndmembersHFC` function.

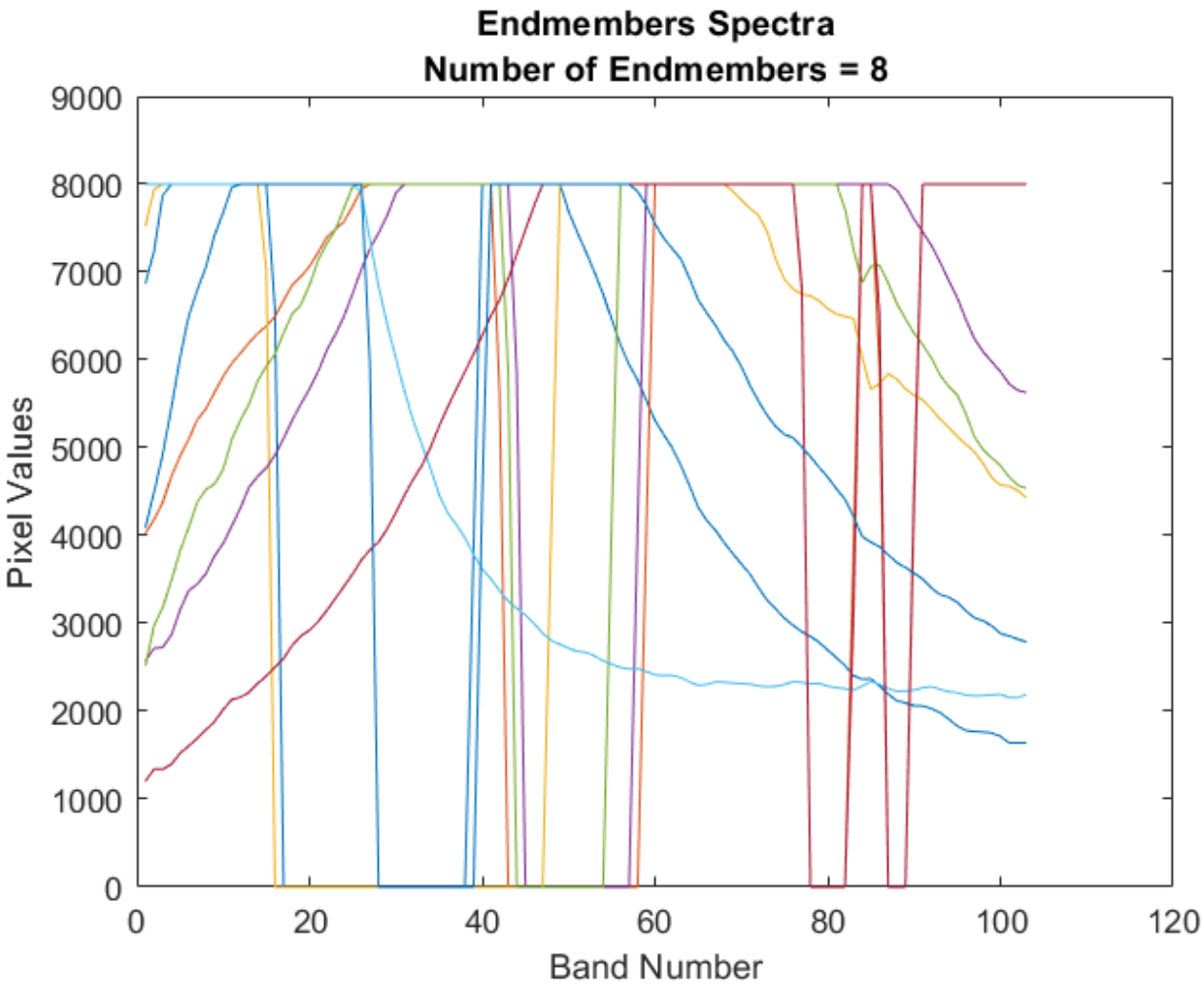
```
numEndmembers = countEndmembersHFC(hcube,'PFA',10^-7);
```

Compute the endmembers using the pixel purity index (PPI) method. By default, the `ppi` function uses maximum noise fraction (MNF) transform for preprocessing. The default number of skewers used for projection is 10^4.

```
endmembers = ppi(hcube.DataCube,numEndmembers);
```

Plot the endmembers of the hyperspectral data.

```
figure
plot(endmembers)
xlabel('Band Number')
ylabel('Pixel Values')
title({'Endmembers Spectra',['Number of Endmembers = ' num2str(numEndmembers)]});
ylim([0 9000])
```



Specify Parameters to Compute Endmembers

Read a hyperspectral data into the workspace.

```
hcube = hypercube('paviaU.hdr');
```

Find the number of spectrally distinct endmembers present in the hyperspectral data cube by using `countEndmembersHFC` function.

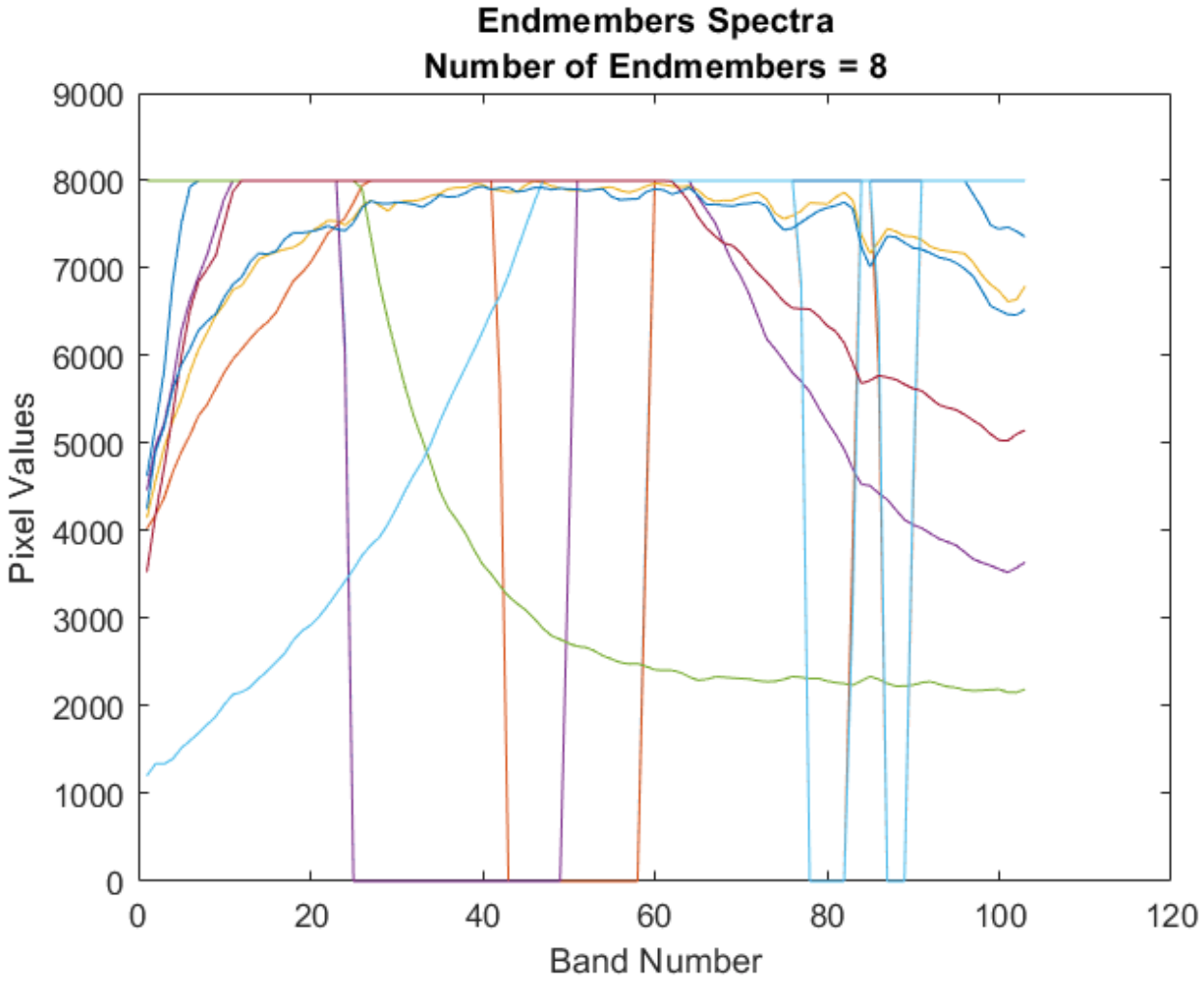
```
numEndmembers = countEndmembersHFC(hcube,'PFA',10^-7);
```

Compute the endmembers using the pixel purity index (PPI) method. Specify the number of unit vectors to be used for projection as 100. Also, select principal component analysis (PCA) method for dimensionality reduction.

```
endmembers = ppi(hcube.DataCube,numEndmembers,'NumVectors',100,'ReductionMethod','PCA');
```

Plot the endmembers of the hyperspectral data.

```
figure
plot(endmembers)
xlabel('Band Number')
ylabel('Pixel Values')
ylim([0 9000])
title({'Endmembers Spectra',['Number of Endmembers = ' num2str(numEndmembers)]});
```



Input Arguments

`inputData` — Input hyperspectral data

3-D numeric array | `hypercube` object

Input hyperspectral data, specified as an 3-D numeric array or a `hypercube` object. If the input is a `hypercube` object, then the function reads the hyperspectral data cube from its `DataCube` property.

The hyperspectral data is an numeric array of size *M*-by-*N*-by-*C*. *M* and*N* are the number of rows and columns in the hyperspectral data respectively. *C* is the number of spectral bands in the hyperspectral data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`numEndmembers` — Number of endmembers
positive scalar integer

Number of endmembers to be extracted, specified as a positive scalar integer. The value must be in the range [1 *C*]. *C* is the number of spectral bands in the input hyperspectral data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `ppi(inputData,7,'NumVectors',100,'Method','None')`

`NumVectors` — Number of random unit vectors
10^4 (default) | positive scalar integer

Number of random unit vectors, specified as the comma-separated pair of '`NumVectors`' and a positive scalar integer. The accuracy of the extracted endmembers increases with the number of vectors used for projection. However, increasing the number of vectors also increases the computational complexity.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ReductionMethod` — Dimensionality reduction method
`'MNF'` (default) | `'PCA'` | `'None'`

Dimensionality reduction method, specified as the comma-separated pair of '`ReductionMethod`' and `'MNF'`, `'PCA'`, or `'None'`.

If you specify the value as `'MNF'` or `'PCA'`, the function first reduces the spectral dimension of the input data by using the specified method. Then, it computes the endmember signatures from the reduced data.

- `'MNF'` — perform dimensionality reduction using the maximum noise fraction (MNF) method.
- `'PCA'` — perform dimensionality reduction using the principal component analysis (PCA) method.

If you specify the value as `'None'`, the function does not perform dimensionality reduction. The endmember signatures are extracted directly from the input data.

Data Types: `char` | `string`

Output Arguments

`endmembers` — Endmember signatures
C-by-*P* matrix

Endmember signatures, returned as a matrix of size *C*-by-*P* and datatype same as the input hyperspectral data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Algorithms

Pixel purity index (PPI) method computes the orthogonal projections of hyperspectral data values on a set of randomly generated unit vectors known as the skewers. Then, the method computes the PPI count for each data value. PPI count is the number of times a data value results as an extrema point when projected on to these skewers. Those data values with more than expected number of PPI count comprise the endmembers of the hyperspectral data. PPI is a non-iterative method and the steps involved are summarised as follows:

1. Compute principal component bands and reduce the dimensionality of the input data by using MNF or PCA. The number of principal component bands to be extracted is set equal to the number of endmembers to be extracted.
2. Generate *k* number of skewers of length same as the input data.
3. Let **r** be the sample vector that denote a pixel spectra. Then, orthogonally project the sample vector onto each skewers and find the extrema.
4. Store the location of each extreme value and count their occurrences. The number of occurrences is known as the PPI count.
5. Find the PPI count for each pixel spectra in the input data cube.
6. Arrange the pixel spectra in descending order of their PPI counts and identify the first *n* number of pixel spectra in the ordered set as endmembers. The number of endmembers to be selected is specified by the input argument `numEndmembers`.

References

[1] J.W Boardman, F.A. Kruse and R.O. Green, "Mapping target signatures via partial unmixing of AVIRIS data.", Technical Report, California, USA, 1995.

See Also

`fippi` | `nfindr` | `countEndmembersHFC` | `hypercube`

Introduced in R2020a

radiance2Reflectance

Convert radiance to reflectance

Syntax

```
newhcube = radiance2Reflectance(hcube)
newhcube = radiance2Reflectance(hcube,'BlockSize',blocksize)
```

Description

`newhcube = radiance2Reflectance(hcube)` converts the pixel values of the hyperspectral data cube from radiance to reflectance values. The function returns a new `hypercube` object and the pixel values of the data cube represent the top of atmosphere (TOA) reflectance. For more information, see “TOA Reflectance” on page 1-3329.

`newhcube = radiance2Reflectance(hcube,'BlockSize',blocksize)` specifies the block size for block processing of the hyperspectral data cube by using the name-value pair argument `'BlockSize'`.

The function divides the input image into distinct blocks, processes each block, and then concatenates the processed output of each block to form the output matrix. Hyperspectral images are multi-dimensional data sets that can be too large to fit in system memory in their entirety. This can cause the system to run out of memory while running the `radiance2Reflectance` function. If you encounter such an issue, perform block processing by using this syntax.

For example, `radiance2Reflectance(hcube,'BlockSize',[50 50])` divides the input image into non-overlapping blocks of size 50-by-50 and then computes the reflectance values for pixels in each block.

Note To perform block processing by specifying the `'BlockSize'` name-value pair argument, you must have MATLAB R2021a or a later release.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Convert Radiance to Reflectance Values

Read hyperspectral data into the workspace.

```
input = hypercube('EO1H0440342002212110PY_cropped.hdr');
```

Determine the bad spectral band numbers using the `BadBands` parameter in the metadata.

```
bandNumber = find(~input.Metadata.BadBands);
```

Remove the bad spectral bands from the data cube.

```
input = removeBands(input,'BandNumber',bandNumber);
```

Convert the digital numbers to radiance values by using the `dn2radiance` function.

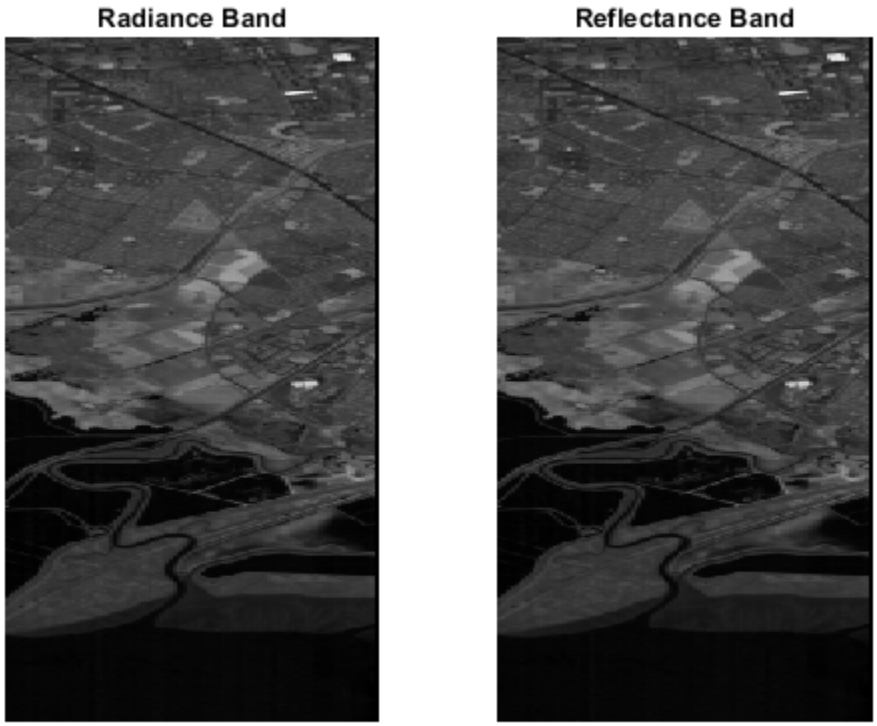
```
hcube = dn2radiance(input);
```

Convert the radiance values to reflectance values by using the `radiance2Reflectance` function.

```
newhcube = radiance2Reflectance(hcube);
```

Read and display the 80th spectral band in the input radiance and the output reflectance data.

```
radianceBand = hcube.DataCube;
reflectanceBand = newhcube.DataCube;
band = 80;

figure
subplot(1,2,1)
imagesc(radianceBand(:,:,band))
axis off
title('Radiance Band')
subplot(1,2,2)
imagesc(reflectanceBand(:,:,band))
title('Reflectance Band')

colormap(gray)
axis off
```



Input Arguments

`hcube` — Input hyperspectral data

`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `DataCube` property of the `hypercube` object stores the hyperspectral data cube. The pixels values of the data cube must be radiance values specifying the amount of radiation from the surface being imaged. You can convert the pixel values in digital numbers to radiance values by using `dn2radiance` function.

`blocksize` — Size of data blocks

2-element vector of positive integers

Size of the data blocks, specified as a 2-element vector of positive integers. The elements of the vector correspond to the number of rows and columns in each block, respectively. The size of the data blocks must be less than the size of the input image. Dividing the hyperspectral images into smaller blocks enables you process large data sets without running out of memory.

- If the `blocksize` value is too small, the memory usage of the function reduces at the cost of increased execution time.
- If the `blocksize` value is large or equal to the input image size, the execution time reduces at the cost of increased memory usage.

Example: `'BlockSize',[20 20]` specifies the size of each data block as 20-by-20.

Output Arguments

`newhcube` — Output hyperspectral data

`hypercube` object

Output hyperspectral data, returned as a `hypercube` object. The pixels values in the output data cube are top of atmosphere (TOA) reflectance values.

More About

TOA Reflectance

The TOA reflectance values specifies the ratio of radiation reflected by the surface to the radiation incident on the surface.

d is the earth-sun distance in astronomical units, $ESUN_\lambda$ is the mean solar irradiance for each spectral band, and θ_E is the sun elevation angle.

L_λ is the spectral radiance computed as:

Gain and *Bias* are the gain and offset values for each spectral bands respectively. The `Metadata` property of `hypercube` object contains the gain and offset values.

See Also

`dn2radiance` | `dn2reflectance` | `empiricalLine` | `iarr` | `sharc` | `hypercube`

Introduced in R2020b

readEcostressSig

Read data from ECOSTRESS spectral library

Syntax

```
libData = readEcostressSig(filenames)
libData = readEcostressSig(dirname)
libData = readEcostressSig(dirname,keyword)
```

Description

`libData = readEcostressSig(filenames)` reads spectral data from the specified ECOSTRESS spectrum files.

The function supports only ECOSTRESS spectrum files. All inputs must be text files with suffix `spectrum.txt`.

`libData = readEcostressSig(dirname)` reads spectral data from the ECOSTRESS spectrum files stored in the specified directory.

`libData = readEcostressSig(dirname,keyword)` reads spectral data from only those ECOSTRESS spectrum files stored in the specified directory with the specified keyword in their file names.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Read ECOSTRESS Spectrum Files

Specify the names of the spectrum files to read from the ECOSTRESS spectral library as a cell array of character vectors.

```
ecostressfiles = [{'soil.utisol.hapludult.none.all.87p707.jhu.becknic.spectrum.txt'},...
             {'water.seawater.none.liquid.tir.seafoam.jhu.becknic.spectrum.txt'},...
             {'vegetation.tree.eucalyptus.maculata.vswir.jpl087.jpl.asd.spectrum.txt'},...
             {'manmade.road.tar.solid.all.0099uuutar.jhu.becknic.spectrum.txt'}];
```

Read and display the data from the specified ECOSTRESS spectrum files. The function returns a structure array with a row for each specified ECOSTRESS spectrum file. Each row stores the spectral data read from the associated file.

```
libData = readEcostressSig(ecostressfiles)
```

```
libData=1×4 struct array with fields:
    Name
```

```
    Type
    Class
    SubClass
    ParticleSize
    Genus
    Species
    SampleNo
    Owner
    WavelengthRange
    Origin
    CollectionDate
    Description
    Measurement
    FirstColumn
    SecondColumn
    WavelengthUnit
    DataUnit
    FirstXValue
    LastXValue
    NumberOfXValues
    AdditionalInformation
    Wavelength
    Reflectance
      ⋮
```

Extract the details of the spectral data of the second file from the structure array.

```
libData(2)
```

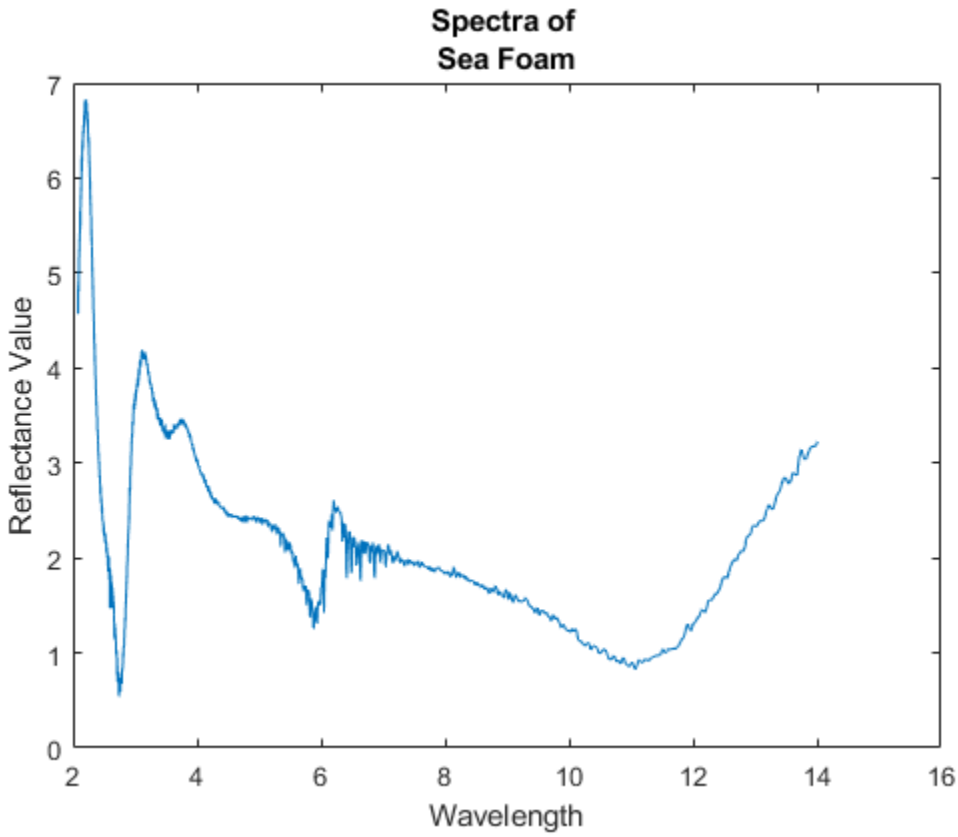
```
ans = struct with fields:
                     Name: "Sea Foam"
                     Type: "Water"
                    Class: "Sea Water"
                 SubClass: "none"
              ParticleSize: "Liquid"
                    Genus: [0x0 string]
                  Species: [0x0 string]
                 SampleNo: "seafoam"
                    Owner: "Dept. of Earth and Planetary Science, John Hopkins University"
          WavelengthRange: "TIR"
                   Origin: "JHU IR Spectroscopy Lab."
           CollectionDate: "N/A"
              Description: "Sea foam water. Original filename FOAM Original ASTER Spectral Libra
              Measurement: "Directional (10 Degree) Hemispherical Reflectance"
              FirstColumn: "X"
             SecondColumn: "Y"
           WavelengthUnit: "micrometer"
                 DataUnit: "Reflectance (percent)"
              FirstXValue: "14.0112"
               LastXValue: "2.0795"
          NumberOfXValues: "2110"
    AdditionalInformation: "none"
               Wavelength: [2110x1 double]
              Reflectance: [2110x1 double]
```

Extract the reflectance and the wavelength values from the spectral data of the second file.

```
reflectance = libData(2).Reflectance;
wavelength = libData(2).Wavelength;
```

Plot the spectral signature using the wavelength and reflectance values.

```
figure
plot(wavelength,reflectance)
title(['Spectra of ' libData(2).Name])
xlabel('Wavelength')
ylabel('Reflectance Value')
```



Read All ECOSTRESS Spectrum Files in Directory

Specify the full path of the directory that contains the ECOSTRESS spectrum files.

```
fileroot = matlabshared.supportpkg.getSupportPackageRoot();
dirname = fullfile(fileroot,'toolbox','images','supportpackages','hyperspectral','hyperdata','EC
```

Read and display the spectral data from all the files in the directory. The function returns a structure array with a row for each ECOSTRESS spectrum file in the specified directory.

```
libData = readEcostressSig(dirname)
```

```
libData=1×15 struct array with fields:
    Name
```

```
    Type
    Class
    SubClass
    ParticleSize
    Genus
    Species
    SampleNo
    Owner
    WavelengthRange
    Origin
    CollectionDate
    Description
    Measurement
    FirstColumn
    SecondColumn
    WavelengthUnit
    DataUnit
    FirstXValue
    LastXValue
    NumberOfXValues
    AdditionalInformation
    Wavelength
    Reflectance
      ⋮
```

Extract the details of the spectral data of the 15th file.

```
libData(15)
```

```
ans = struct with fields:
                      Name: "Tap water"
                      Type: "Water"
                     Class: "Tap Water"
                  SubClass: "none"
               ParticleSize: "Liquid"
                     Genus: [0×0 string]
                   Species: [0×0 string]
                  SampleNo: "tapwater"
                     Owner: "Dept. of Earth and Planetary Science, John Hopkins University"
           WavelengthRange: "All"
                    Origin: "JHU IR Spectroscopy Lab. Original filename TAPWATER."
            CollectionDate: "N/A"
               Description: "Tap water. Original ASTER Spectral Library name was jhu.becknic.wate
               Measurement: "Directional (10 Degree) Hemispherical Reflectance"
               FirstColumn: "X"
              SecondColumn: "Y"
            WavelengthUnit: "micrometer"
                  DataUnit: "Reflectance (percent)"
               FirstXValue: "14.0110"
                LastXValue: "0.4000"
           NumberOfXValues: "2844"
    AdditionalInformation: "none"
                Wavelength: [2844×1 double]
               Reflectance: [2844×1 double]
```

Search ECOSTRESS Spectrum Files Using Keyword

Specify full path of the directory that contains the ECOSTRESS spectrum files.

```
fileroot = matlabshared.supportpkg.getSupportPackageRoot();
dirname = fullfile(fileroot,'toolbox','images','supportpackages','hyperspectral','hyperdata','EC
```

Read and display the spectral data of the ECOSTRESS spectrum files with a specific keyword in their file names. The function returns a structure array with a row for each spectrum file in the specified directory with the keyword in their file names.

```
keyword = 'water';
libData = readEcostressSig(dirname,keyword)
```

```
libData=1×3 struct array with fields:
    Name
    Type
    Class
    SubClass
    ParticleSize
    Genus
    Species
    SampleNo
    Owner
    WavelengthRange
    Origin
    CollectionDate
    Description
    Measurement
    FirstColumn
    SecondColumn
    WavelengthUnit
    DataUnit
    FirstXValue
    LastXValue
    NumberOfXValues
    AdditionalInformation
    Wavelength
    Reflectance
      ⋮
```

Input Arguments

`filenames` — Names of ECOSTRESS files

character vector | string scalar | cell array of character vectors | vector of strings

Names of the ECOSTRESS files, specified as a character vector, string scalar, cell array of character vectors, or vector of strings. To read the data from multiple ECOSTRESS files simultaneously, use a cell array of character vectors or vector of strings. The function reads data from the files in the order in which you specify them. If the ECOSTRESS files are not in the current folder, you must specify the full path of each file.

Data Types: `char` | `string`

`dirname` — Name of directory

character vector | string scalar

Name of the directory containing the ECOSTRESS files, specified as a character vector or string scalar. If the directory is not in the current folder, you must specify the full path of the directory.

Data Types: `char` | `string`

`keyword` — File search keyword
character vector | string scalar

File search keyword, specified as a character vector or string scalar. The function returns data from only the ECOSTRESS spectrum files with the specified keyword in their file names. You cannot specify multiple keywords simultaneously.

Data Types: `char` | `string`

Output Arguments

`libData` — Spectral data from ECOSTRESS files
structure array

Spectral data from ECOSTRESS files, returned as a 1-by-*K* structure array. *K* is the number of spectrum files read by the function. Each element of the structure array has 24 fields that contain the header information of the spectrum files.

Field Names	Description
`Name`	Name of the measured sample or material
`Type`	Type of sample, such as `"mineral"`, `"rock"`, `"tree"`, or `"manmade"`
`Class`	Class of the sample type For example, if the sample type is `"mineral"` then the class can be: `"native elements"`,`"silicates"`, `"oxides"`, `"sulfides"`, `"sulfates"`, `"halides"`, `"carbonates"`, `"phosphates"`, or `"mineraloids"`.
`SubClass`	Subclass of the sample type This field contains an empty array or `"none"`, unless the `Type` value is `"mineral"`, `"rock"`, `"manmade"`, `"soil"`, `"lunar"`, or `"meteorite"`.
`ParticleSize`	Particle size of the sample type This field contains an empty array unless the `Type` value is `"mineral"`, `"rock"`, `"manmade"`, `"soil"`, `"lunar"`, or `"meteorite"`.
`Genus`	Genus of the sample This field contains an empty array unless the `Type` value is `"vegetation"` or `"nonphotosynthetic"`.

`Species`	Species of the sample This field contains an empty array unless the `Type` value is `"vegetation"` or `"nonphotosynthetic"`.
`SampleNo`	Sample number This value is an identifier for the associated sample.
`Owner`	Owner of the sample
`WavelengthRange`	Wavelength range of the measured sample The value must be `"All"`, `"TIR"`, or `"VSWIR"`.
`Origin`	Location from which the data was obtained
`CollectionDate`	Date on which the sample was collected This value is in `mm/dd/yy` format.
`Description`	Description of the measured sample This field provides additional information about the characteristics of the sample.
`Measurement`	Spectral measurement mode used to measure the sample
`FirstColumn`	First column of data values in the spectrum file
`SecondColumn`	Second column of data values in the spectrum file
`WavelengthUnit`	Measuring unit for the spectral wavelengths of the samples The value for every sample type is `"micrometer"`. This field corresponds to the `X Units` field of the header data in the ECOSTRESS spectrum file.
`DataUnit`	Unit of the spectral measurement mode Spectral measurement mode includes reflectance, transmittance, transittance, and transmission. The unit is percentage. This field corresponds to the `Y Units` field of the header data in the ECOSTRESS spectrum file.
`FirstXValue`	First value in the first column of data values in the spectrum file
`LastXValue`	Last value in the first column of data values in the spectrum file
`NumberofXValues`	Total number of data values in the first column of the spectrum file

`AdditionalInformation`	Additional information about the sample This field includes information that is not part of the spectral data.
`Wavelength`	Wavelength values at which the reflectances were measured
`Reflectance`	Reflectance values measured at each wavelengths

References

[1] Meerdink, Susan K., Simon J. Hook, Dar A. Roberts, and Elsa A. Abbott. “The ECOSTRESS Spectral Library Version 1.0.” *Remote Sensing of Environment* 230 (September 2019): 111196. https://doi.org/10.1016/j.rse.2019.05.015.

[2] Download the ECOSTRESS Spectral Library: https://speclib.jpl.nasa.gov/download

See Also

`spectralMatch` | `sam` | `sid` | `hypercube`

Introduced in R2020a

reduceSmile

Reduce spectral smile effect in hyperspectral data cube

Syntax

```
correctedData = reduceSmile(hcube)
correctedData = reduceSmile(hcube,'BlockSize',blocksize)
correctedData = reduceSmile( ___ ,Name,Value)
```

Description

`correctedData = reduceSmile(hcube)` reduces the spectral smile effect in the hyperspectral data `hcube` by averaging the pixel values of each band along the spectral dimension with a window of size `3`. The function averages the pixel values of each band with the corresponding pixel values of the previous band and the next band. The spectral smile effect occurs only in the data captured using push-broom hyperspectral sensors, such as the Hyperion EO-1 and the SEBASS.

`correctedData = reduceSmile(hcube,'BlockSize',blocksize)` specifies the block size for block processing of the hyperspectral data cube by using the name-value pair argument `'BlockSize'`. You can specify the `'BlockSize'` name-value pair argument in addition to the input arguments in the previous syntaxes.

The function divides the input image into distinct blocks, processes each block, and then concatenates the processed output of each block to form the output matrix. Hyperspectral images are multi-dimensional data sets that can be too large to fit in system memory in their entirety. This can cause the system to run out of memory while running the `reduceSmile` function. If you encounter such an issue, perform block processing by using this syntax.

For example, `reduceSmile(hcube,'BlockSize',[50 50])` divides the input image into non-overlapping blocks of size 50-by-50 and then performs smile correction on each block.

Note To perform block processing by specifying the `'BlockSize'` name-value pair argument, you must have MATLAB R2021a or a later release.

`correctedData = reduceSmile( ___ ,Name,Value)` specifies options using one or more name-value arguments in addition to the input arguments in the previous syntaxes. For example, `'Method','MNF'` specifies to perform smile correction using the maximum noise fraction (MNF) transform-based method.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Perform Spectral Smile Correction

Read hyperspectral data from the Hyperion EO-1 sensor into the workspace.

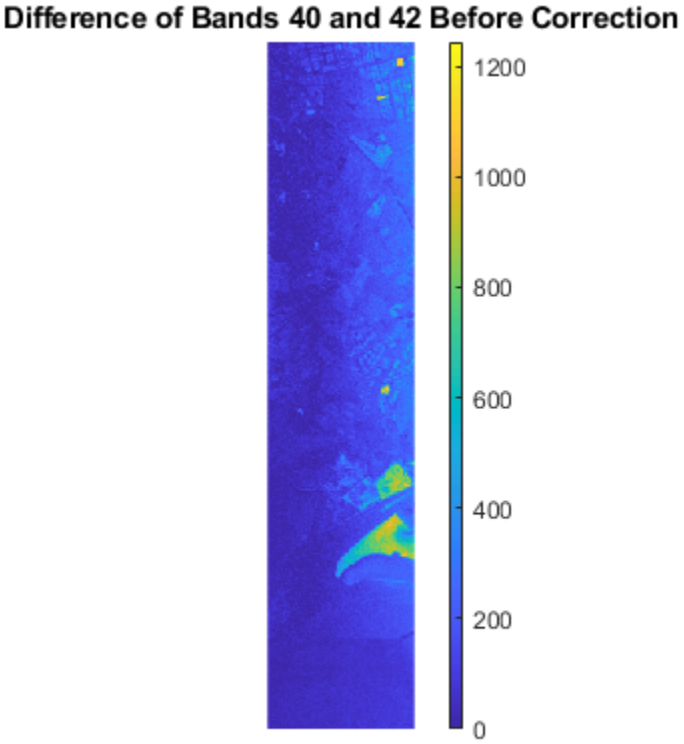
```
hCube =  hypercube('EO1H0440342002212110PY_cropped.dat');
```

Typically, Oxygen molecules have strong absorption features at 762 nm wavelength and causes spectral smile. In the Hyperion EO-1 sensor, 762 nm wavelength corresponds to band 41. The spectral smile effect in band 41 also affects the bands 40 and 42. Hence, calculate the absolute difference between the pixel values of bands 40 and 42.

```
originalData = hCube.DataCube;
band40 = originalData(:,:,40);
band42 = originalData(:,:,42);
bandDiffBeforeCorr = imabsdiff(band40,band42);
```

Display the difference image of bands 40 and 42. The spectral smile effect appears as a brightness gradient from left to right in the difference image.

```
imagesc(bandDiffBeforeCorr)
axis image off
title('Difference of Bands 40 and 42 Before Correction')
colorbar
```



Specify the size of the averaging window to use along the spectral dimension.

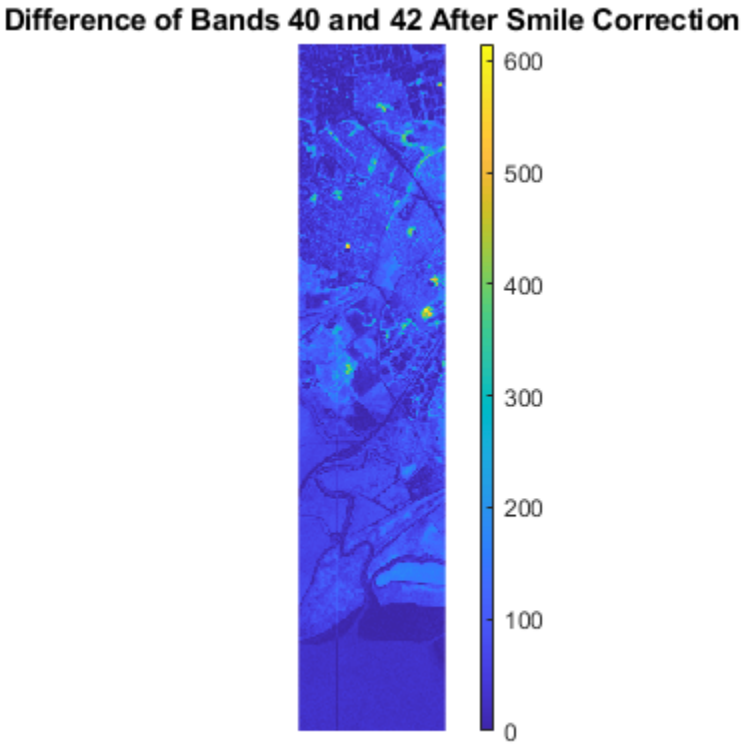
```
window = 5;
```

Perform spectral smile reduction.

```
correctedHcube = reduceSmile(hCube,'SpectralWindow',window);
```

Calculate and display the absolute difference of the corrected bands 40 and 42. There is no gradient in the difference image after smile correction.

```
correctedData = correctedHcube.DataCube;
corrBand40 = correctedData(:,:,40);
corrBand42 = correctedData(:,:,42);
bandDiffAfterCorr = imabsdiff(corrBand40,corrBand42);
imagesc(bandDiffAfterCorr)
axis image off
title('Difference of Bands 40 and 42 After Smile Correction')
colorbar
```



Input Arguments

`hcube` — Input hyperspectral data

`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `DataCube` property of the `hypercube` object stores the hyperspectral data cube.

`blocksize` — Size of data blocks

2-element vector of positive integers

Size of the data blocks, specified as a 2-element vector of positive integers. The elements of the vector correspond to the number of rows and columns in each block, respectively. The size of the data blocks must be less than the size of the input image. Dividing the hyperspectral images into smaller blocks enables you process large data sets without running out of memory.

- If the `blocksize` value is too small, the memory usage of the function reduces at the cost of increased execution time.
- If the `blocksize` value is large or equal to the input image size, the execution time reduces at the cost of increased memory usage.

Example: `'BlockSize',[20 20]` specifies the size of each data block as 20-by-20.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `reduceSmile(hcube,'Method','MNF')`

`Method` — Spectral smile correction method
`'SpectralSmoothing'` (default) | `'MNF'`

Spectral smile correction method, specified as `'SpectralSmoothing'` or `'MNF'`.

- `'SpectralSmoothing'` — Perform smile reduction using the spectral smoothing method.
- `'MNF'` — Perform smile reduction using the maximum noise fraction (MNF) transform-based method.

`SpectralWindow` — Size of averaging window
3 (default) | positive integer

Size of the averaging window along the spectral dimension, specified as a positive integer that is less than or equal to the number of hyperspectral bands in the hyperspectral data `inputData`. Increasing the window size can further reduce the smile effect, but using an averaging window greater than 9 can result in the loss of fine atmospheric absorption spectral features. To specify this argument, you must specify the `'Method'` argument as `'SpectralSmoothing'`.

Output Arguments

`correctedData` — Corrected hyperspectral data
`hypercube` object

Corrected hyperspectral data, returned as a `hypercube` object of the size same as the input data `hcube`. If the input data is of type `double`, then the corrected hyperspectral data is of type `double`. Otherwise, the data type of the corrected hyperspectral data is `single`.

References

[1] Perkins, Timothy, Steven M. Adler-Golden, Michael W. Matthew, Alexander Berk, Lawrence S. Bernstein, Jasmine Lee, and Marsha E. Fox. "Speed and Accuracy Improvements in FLAASH

Atmospheric Correction of Hyperspectral Imagery." *Optical Engineering* 51, no. 11 (June 13, 2012): 111707, https://doi.org/10.1117/1.OE.51.11.111707.

[2] Yokoya, Naoto, Norihide Miyamura, and Akira Iwasaki. “Detection and Correction of Spectral and Spatial Misregistrations for Hyperspectral Data Using Phase Correlation Method.” *Applied Optics* 49, no. 24 (August 20, 2010): 4568. https://doi.org/10.1364/AO.49.004568.

See Also

`hypercube` | `sharc` | `smileMetric` | `blockedImage`

Introduced in R2020b

removeBands

Remove spectral bands from data cube

Syntax

```
newhcube = removeBands(hcube,'Wavelength',wlrange)
newhcube = removeBands(hcube,'BandNumber',band)
```

Description

`newhcube = removeBands(hcube,'Wavelength',wlrange)` removes spectral bands from the data cube within the specified wavelength range. The function returns a new `hypercube` object with the remaining wavelengths, their metadata information, and the corresponding spectral bands from the original data cube.

`newhcube = removeBands(hcube,'BandNumber',band)` removes the spectral bands with the specified spectral band numbers from the hyperspectral data cube.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Remove Bands in Specified Wavelength Range

Read hyperspectral data into the workspace.

```
hcube = hypercube('paviaU.dat');
```

Inspect the properties of the `hypercube` object.

```
hcube
```

```
hcube =
  hypercube with properties:

      DataCube: [610×340×103 double]
    Wavelength: [103×1 double]
      Metadata: [1×1 struct]
```

Find the spectral wavelength range of the hyperspectral data cube.

```
range = [min(hcube.Wavelength) max(hcube.Wavelength)]
```

```
range = 1×2

   430   838
```

Specify the ranges of wavelengths to be remove from the hyperspectral data cube.

```
wlrange = [410 450; 620 850];
```

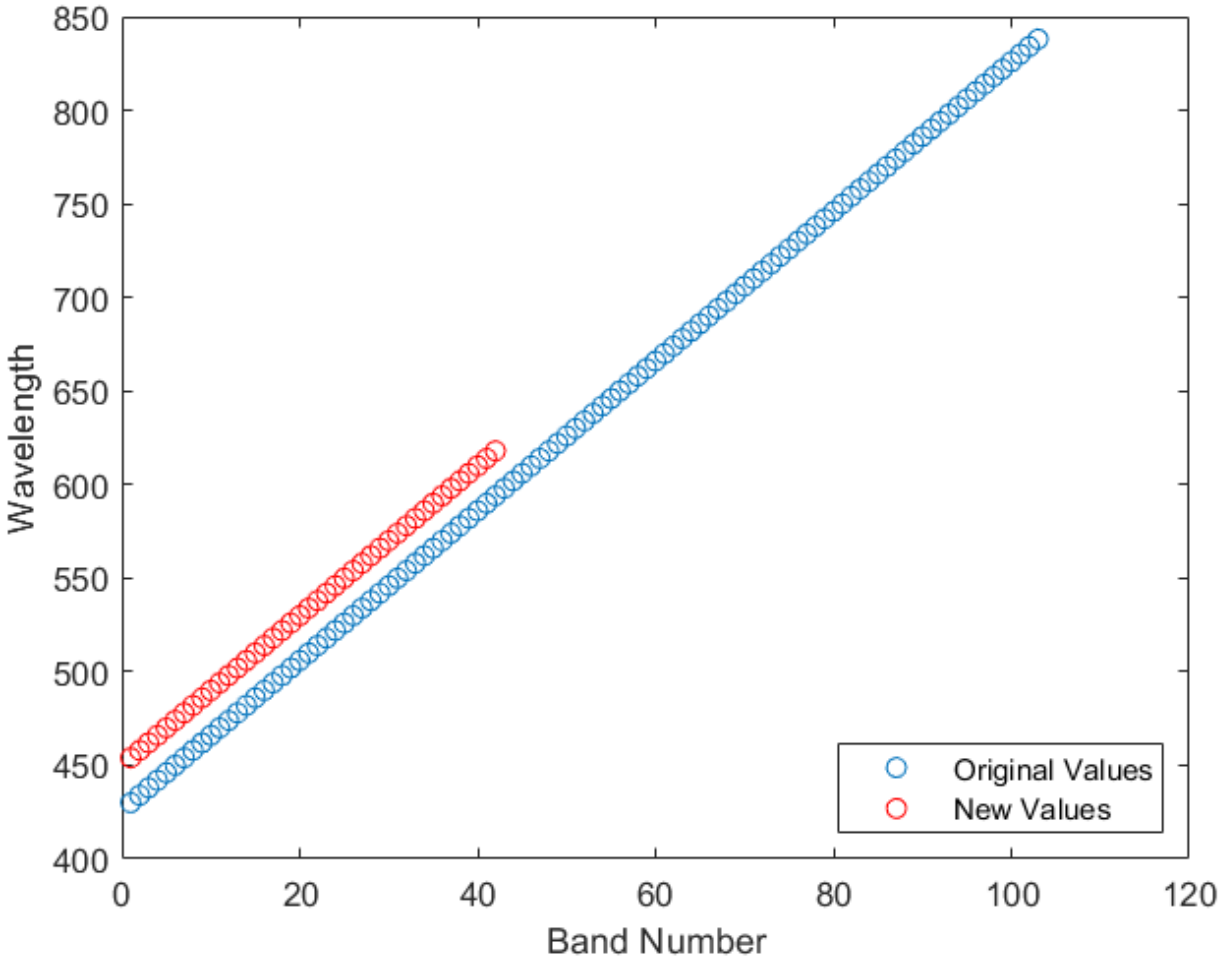
Remove the spectral bands that lie in the specified wavelength ranges. The function returns a new `hypercube` object without the removed bands.

```
newhcube = removeBands(hcube,'Wavelength',wlrange)
```

```
newhcube =
  hypercube with properties:

      DataCube: [610×340×42 double]
    Wavelength: [42×1 double]
      Metadata: [1×1 struct]
```

Plot the original and the new wavelength values.

```
figure
plot(hcube.Wavelength,'o')
hold on
plot(newhcube.Wavelength,'or')
xlabel('Band Number')
ylabel('Wavelength')
legend('Original Values','New Values','Location','SouthEast')
```



Remove NonInformative Bands from Data Cube

Read a hyperspectral data into the workspace.

```
hcube = hypercube('paviaU.dat')
```

```
hcube =
  hypercube with properties:

      DataCube: [610×340×103 double]
    Wavelength: [103×1 double]
      Metadata: [1×1 struct]
```

Compute five spectrally distinct endmembers of the hyperspectral data cube by using the `ppi` function.

```
endmembers = fippi(hcube,5);
```

Determine the 10 most informative bands of the input data cube based on the endmembers spectra.

```
[~,informativeband] = selectBands(hcube,endmembers,'NumberOfBands',10);
```

Find the band numbers of the noninformative bands of the data cube by using the band numbers of the informative bands.

```
band = setdiff(1:size(hcube.DataCube,3),informativeband);
```

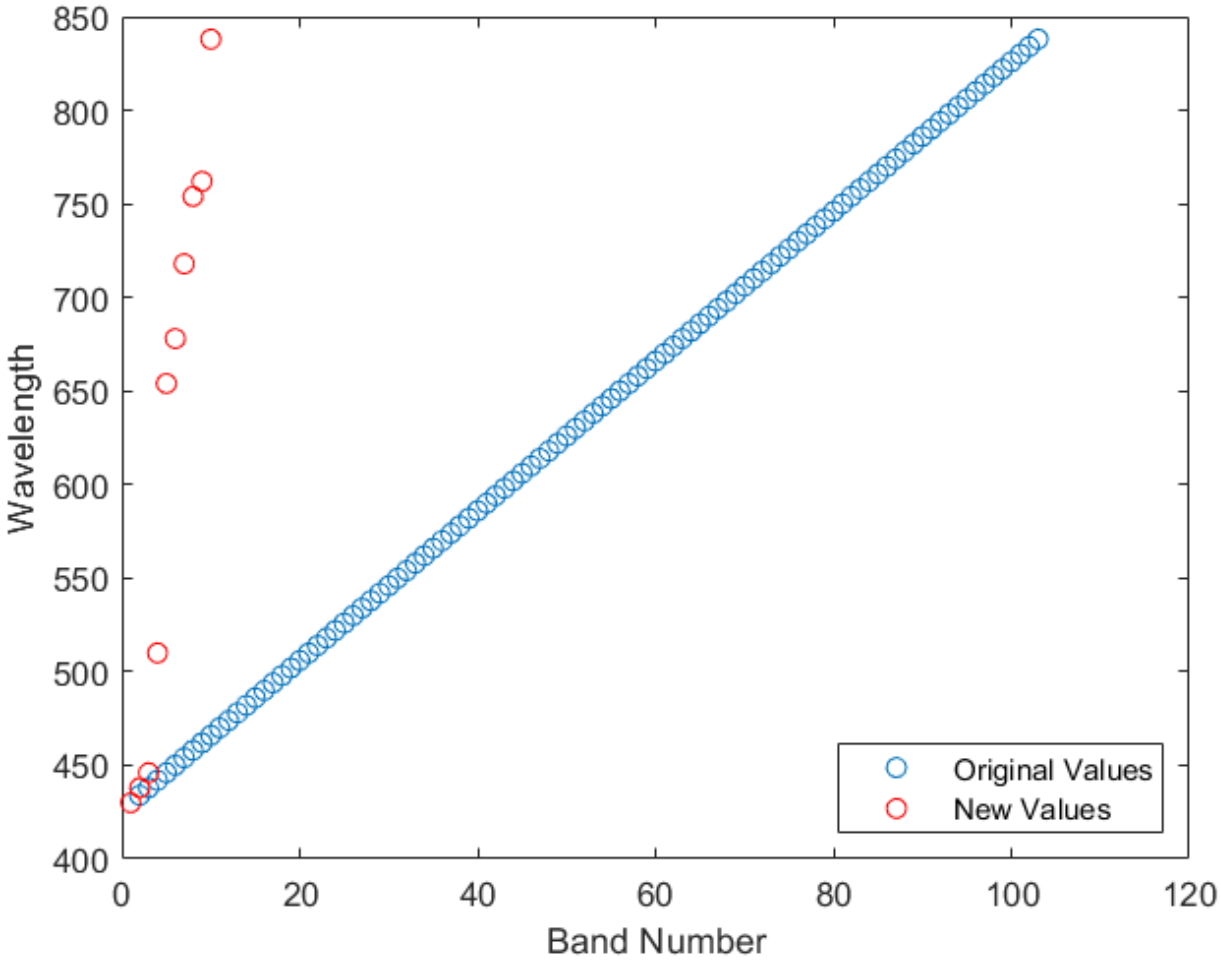
Remove the noninformative bands from the hyperspectral data cube. The function returns a new `hypercube` object with only the most informative bands.

```
newhcube = removeBands(hcube,'BandNumber',band)
```

```
newhcube =
  hypercube with properties:

      DataCube: [610×340×10 double]
    Wavelength: [10×1 double]
      Metadata: [1×1 struct]
```

Plot the original and the new wavelength values.

```
figure
plot(hcube.Wavelength,'o')
hold on
plot(newhcube.Wavelength,'or')
xlabel('Band Number')
ylabel('Wavelength')
legend('Original Values','New Values','Location','SouthEast')
```



Input Arguments

`hcube` — Input hyperspectral data

`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `DataCube` property of the `hypercube` object stores the hyperspectral data cube.

`wlrange` — Wavelength range to remove

K-by-2 matrix

Wavelength range to remove, specified as a *K*-by-2 matrix. *K* is the number of wavelength ranges to remove from the input data. Each row is of form $[W_{min}\ W_{max}]$. W_{min} and W_{max} are the minimum and the maximum wavelengths of the ranges to remove. At least one specified wavelength range must overlap the wavelength value of at least one spectral band within the input `hypercube` object.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

band — Spectral band number to remove
positive integer | vector of positive integers

Spectral band number to remove, specified as a positive integer or vector of positive integers. All of the specified band numbers must be less than or equal to the number of spectral bands in the input hyperspectral data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

newhcube — Output hyperspectral data
`hypercube` object

Output hyperspectral data, returned as a `hypercube` object.

See Also

`hypercube` | `selectBands` | `ppi` | `fippi` | `nfindr`

Introduced in R2020a

rrs

Compute remote sensing reflectance

Syntax

```
newhcube = rrs(hcube)
[newhcube,mask] = rrs(hcube)
```

Description

`newhcube = rrs(hcube)` computes remote sensing reflectance (RRS) values. The remote sensing reflectance values are the atmospherically corrected values. The function first computes the water leaving radiance and then, estimates RRS as the ratio of water leaving radiance to solar irradiance at the top of the atmosphere (TOA).

The pixels values of the data cube must be TOA radiance values. If the values are digital numbers, use the `dn2radiance` function to compute the TOA radiance values. This method gives best results for multispectral data.

`[newhcube,mask] = rrs(hcube)` also returns a region mask indicating the clear water regions in the input satellite data.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Estimate Remotely Sensed Water Reflectance

Read multispectral data into the workspace.

```
hcube = hypercube('LC08_L1TP_097070_20201101_20201101_01_cropped.dat');
```

Convert the pixel values from digital numbers to top of atmosphere (TOA) radiances.

```
hcube = dn2radiance(hcube);
```

Estimate remotely sensed water reflectance using the clear-water pixels approach.

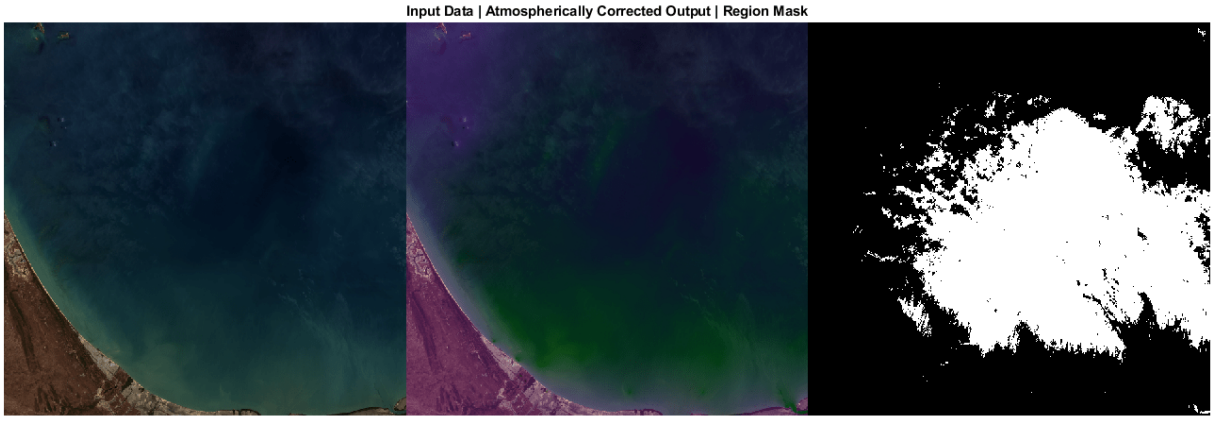
```
[newhcube,mask] = rrs(hcube);
```

Estimate RGB images of the input and the atmospherically corrected output data.

```
imgIn = colorize(hcube,'Method','rgb');
imgOut = colorize(newhcube,'Method','rgb');
```

Display the RGB images of the input and the atmospherically corrected output data. Also, display the clear-water pixel region mask used for computing the correction parameters.

```
figure
montage({imgIn;imgOut;mask},'Size',[1 3])
title('Input Data | Atmospherically Corrected Output | Region Mask')
```



Input Arguments

`hcube` — Input satellite data

`hypercube` object

Input satellite data, specified as a `hypercube` object. The functions reads the data cube from the `DataCube` property of the object.

Output Arguments

`newhcube` — Atmospherically corrected data

`hypercube` object

Atmospherically corrected data, returned as a `hypercube` object.

`mask` — Region mask indicating clear water regions

matrix

Region mask indicating clear water regions, returned as a matrix. The `mask` is a binary image of spatial dimension same as that of the input data cube.

References

[1] Wang, Deyu, Xuezhi Feng, Ronghua Ma, and Guoding Kang. “A Method for Retrieving Water-Leaving Radiance from Landsat TM Image in Taihu Lake, East China.” *Chinese Geographical Science* 17, no. 4 (December 2007): 364–69. https://doi.org/10.1007/s11769-007-0364-7.

See Also

`correctOOB` | `dn2radiance` | `fastInscene` | `hypercube`

Introduced in R2020b

sam

Measure spectral similarity using spectral angle mapper

Syntax

```
score = sam(inputData,refSpectra)
score = sam(testSpectra,refSpectra)
```

Description

`score = sam(inputData,refSpectra)` measures the spectral similarity between the spectra of each pixel in the hyperspectral data `inputData` and the specified reference spectra `refSpectra` by using the spectral angle mapper (SAM) classification algorithm. Use this syntax to identify different regions or materials in a hyperspectral data cube.

`score = sam(testSpectra,refSpectra)` measures the spectral similarity between the specified test spectra `testSpectra` and reference spectra `refSpectra` by using the SAM classification algorithm. Use this syntax to compare the spectral signature of an unknown material against the reference spectra or to compute spectral variability between two spectral signatures.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Distinguish Hyperspectral Regions Using Spectral Angle Mapper

Distinguish different regions in a hyperspectral data cube by computing the spectral angle distance between each pixel and the endmember spectra of the data cube.

Read hyperspectral data into the workspace.

```
hcube = hypercube('jasperRidge2_R198.hdr');
```

Identify the number of spectrally distinct bands in the data cube by using the `countEndmembersHFC` function.

```
numEndmembers = countEndmembersHFC(hcube)
```

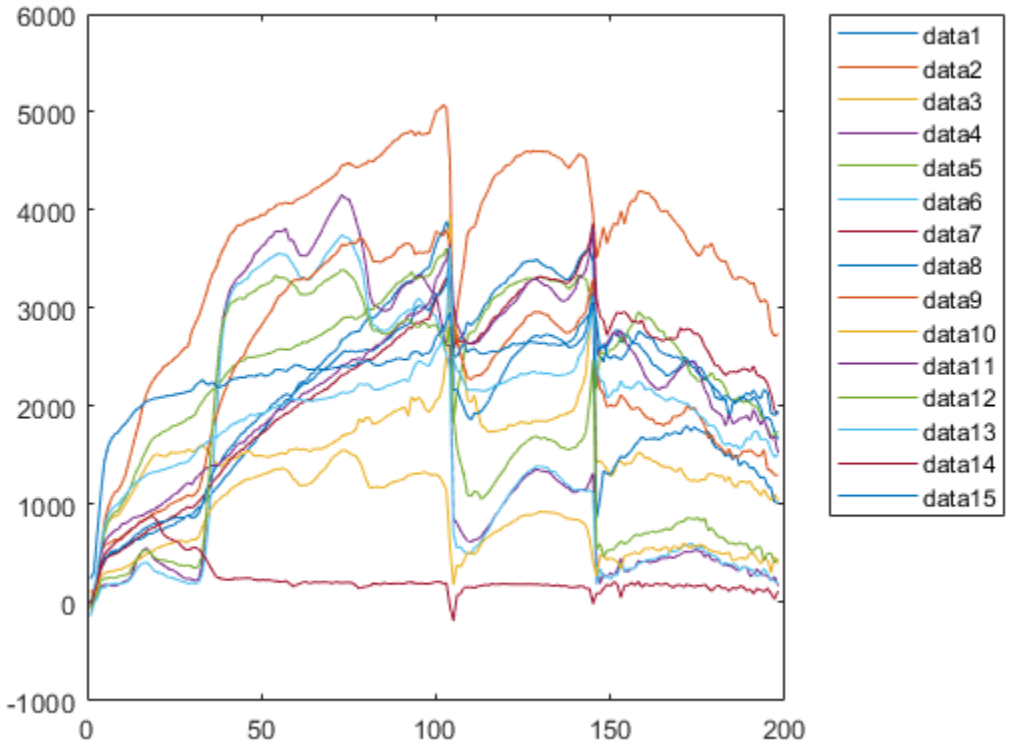
```
numEndmembers = 15
```

Extract the endmember spectral signatures from the data cube by using the NFINDR algorithm.

```
endmembers = nfindr(hcube,numEndmembers);
```

Plot the spectral signatures of the endmembers. The result shows the 14 spectrally distinct regions in the data cube.

```
figure
plot(endmembers)
legend('Location','Bestoutside')
```



Compute the spectral angular distance between each endmember and the spectrum of each pixel in the data cube.

```
score = zeros(size(hcube.DataCube,1),size(hcube.DataCube,2),numEndmembers);
for i = 1:numEndmembers
    score(:,:,i) = sam(hcube,endmembers(:,i));
end
```

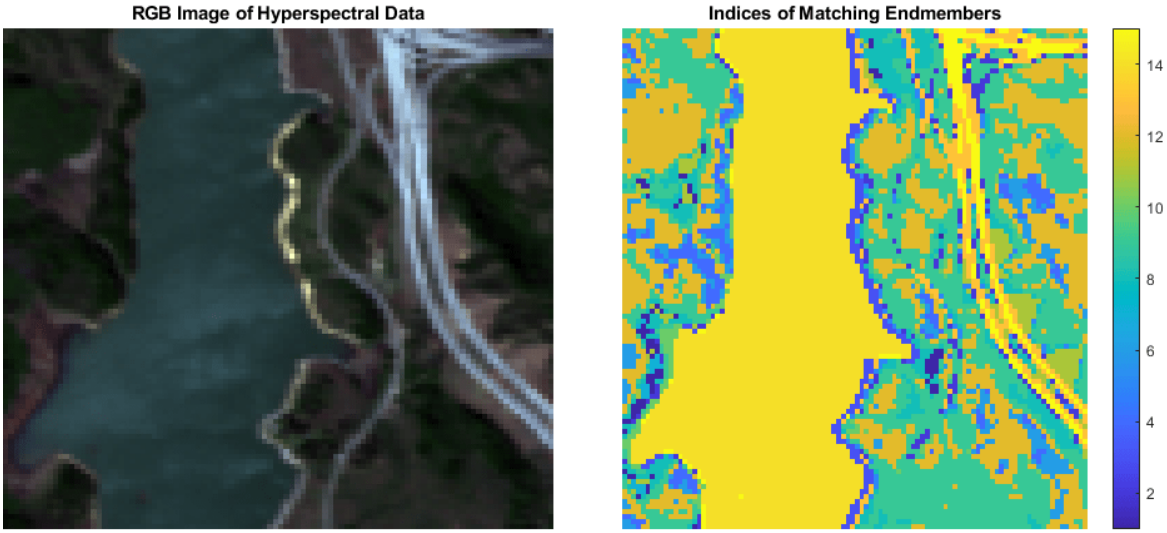
Compute the minimum score value from the distance scores obtained for each pixel spectrum with respect to all the endmembers. The index of each minimum score identifies the endmember spectrum to which a pixel spectrum exhibits maximum similarity. An index value, *n*, at the spatial location (*x*, *y*) in the score matrix indicates that the spectral signature of the pixel at spatial location (*x*, *y*) in the data cube best matches the spectral signature of the *n*th endmember.

```
[~,matchingIndx] = min(score,[],3);
```

Estimate an RGB image of the hyperspectral data cube by using the `colorize` function. Display both the RGB image and the matrix of matched index values.

```
rgbImg = colorize(hcube,'Method','RGB');
figure('Position',[0 0 1100 500])
subplot('Position',[0 0.15 0.4 0.8])
imagesc(rgbImg)
```

```
axis off
title('RGB Image of Hyperspectral Data')
subplot('Position',[0.45 0.15 0.4 0.8])
imagesc(matchingIndx)
axis off
title('Indices of Matching Endmembers')
colorbar
```



Determine Similarity of Endmember Spectra Using SAM

Read hyperspectral data into the workspace.

```
hcube = hypercube('indian_pines.dat');
```

Find ten endmembers of the hyperspectral data.

```
numEndmembers = 10;
endmembers = nfindr(hcube,numEndmembers);
```

Consider the first endmember as the reference spectrum and the rest of the endmembers as the test spectrum. Compute the SAM score between the reference and test spectrum.

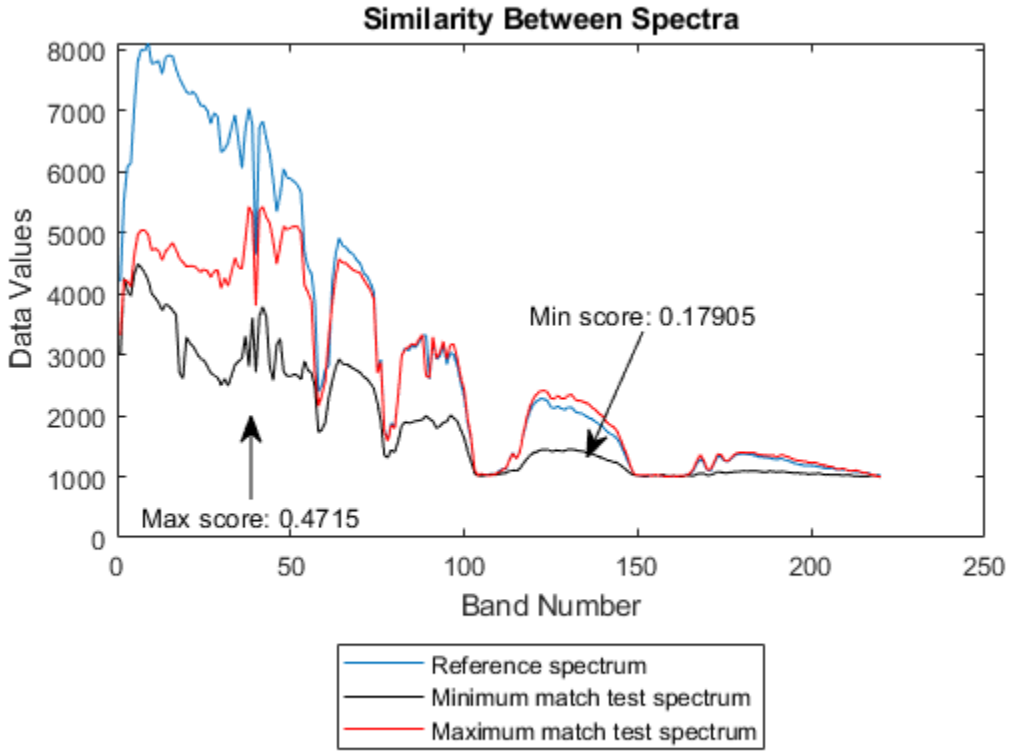
```
score = zeros(1,numEndmembers-1);
refSpectrum = endmembers(:,1);
for i = 2:numEndmembers
    testSpectrum = endmembers(:,i);
    score(i-1) = sam(testSpectrum,refSpectrum);
end
```

Find the test spectrum that exhibit maximum similarity (minimum distance) to the reference spectrum. Then find the test spectrum that exhibit minimum similarity (maximum distance) to the reference spectrum.

```
[minval,minidx] = min(score);
maxMatch = endmembers(:,minidx);
[maxval,maxidx] = max(score);
minMatch = endmembers(:,maxidx);
```

Plot the reference spectrum, the maximum similarity and the minimum similarity test spectrum. The test spectrum with the minimum score value indicates highest similarity to the reference endmember. On the other hand, the test spectrum with the maximum score value has the highest spectral variability and characterises the spectral behaviour of two different materials.

```
figure
plot(refSpectrum)
hold on
plot(maxMatch,'k')
plot(minMatch,'r')
xlabel('Band Number')
ylabel('Data Values')
legend('Reference spectrum','Minimum match test spectrum','Maximum match test spectrum',...
       'Location','Southoutside')
title('Similarity Between Spectra')
annotation('textarrow',[0.25 0.25],[0.4 0.5],'String',['Max score: ' num2str(maxval)])
annotation('textarrow',[0.6 0.55],[0.6 0.45],'String',['Min score: ' num2str(minval)])
```



Input Arguments

`inputData` — Input hyperspectral data

`hypercube` object | 3-D numeric array

Input hyperspectral data, specified as a `hypercube` object or a 3-D numeric array containing the data cube. If the input is a `hypercube` object, the data is read from the `DataCube` property of the object.

`testSpectra` — Test spectra

C-element vector

Test spectra, specified as a *C*-element vector. The test spectra is the spectral signature of an unknown region or material.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`refSpectra` — Reference spectra

C-element vector

Reference spectra, specified as a *C*-element vector. The reference spectra is the spectral signature of a known region or material. The function matches the test spectra against these values.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`score` — SAM score

scalar | matrix

SAM score, returned as a scalar or matrix. The output is a

- scalar — If you specify the `testSpectra` input argument. The function matches the test spectral signature against the reference spectral signature and returns a scalar value. Both the test and the reference spectra must be vectors of same length.
- matrix — If you specify the `inputData` input argument. The function matches the spectral signature of each pixel in the data cube against the reference spectral signature and returns a matrix. If the data cube is of size *M*-by-*N*-by-*C* and the reference spectra is a vector of length *C*, the output matrix is of size *M*-by-*N*.

Each element of the SAM score is a spectral angle in radians in the range [0, 3.142]. A smaller SAM score indicates a strong match between the test signature and the reference signature.

Data Types: `single` | `double`

More About

Spectral angle mapper

Given the test spectra t and a reference spectra r of length C, the SAM score α is calculated as

References

[1] Kruse, F.A., A.B. Lefkoff, J.W. Boardman, K.B. Heidebrecht, A.T. Shapiro, P.J. Barloon, and A.F.H. Goetz. “The Spectral Image Processing System (SIPS)—Interactive Visualization and Analysis

of Imaging Spectrometer Data.” *Remote Sensing of Environment* 44, no. 2–3 (May 1993): 145–63. https://doi.org/10.1016/0034-4257(93)90013-N.

See Also

`spectralMatch` | `readEcostressSig` | `sid` | `hypercube` | `sidsam` | `jmsam`

Introduced in R2020a

selectBands

Select most informative bands

Syntax

```
newhcube = selectBands(hcube,endmembers)
[newhcube,band] = selectBands(hcube,endmembers)
[___] = selectBands(hcube,endmembers,'NumberOfBands',numBands)
```

Description

`newhcube = selectBands(hcube,endmembers)` selects the most informative bands of the hyperspectral data cube by using orthogonal space projection method [1]. The function returns a new `hypercube` object that contains the data from only the most informative bands.

Note

- For preprocessing, the function removes the water absorption and low signal-to-noise ratio (SNR) bands prior to computing the most informative bands.
- To reduce the computational complexity, the function computes the most informative bands by considering only 10% of the pixel values in the pre-processed data cube. These values are selected randomly. The function also ensures that the random selection does not result in the removal of endmembers.

`[newhcube,band] = selectBands(hcube,endmembers)` also returns the band numbers of the most informative bands in the hyperspectral data cube.

`[___] = selectBands(hcube,endmembers,'NumberOfBands',numBands)` additionally specifies the number of most informative bands to select from the input data cube, in addition to any combination of arguments from previous syntaxes.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Select Most Informative Hyperspectral Bands

Read hyperspectral data into the workspace.

```
hcube = hypercube('paviaU.dat');
```

Estimate the endmembers of the data cube by using the FIPPI algorithm.

```
endmembers = fippi(hcube,9);
```

Create a new `hypercube` consisting of the ten most informative bands.

```
newhcube = selectBands(hcube,endmembers,'NumberOfBands',10);
```

Input Arguments

`hcube` — Input hyperspectral data

`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `DataCube` property of the `hypercube` object contains the hyperspectral data cube.

`endmembers` — Spectral signatures of endmembers

matrix

Spectral signatures of the endmembers, specified as a matrix of size *C*-by-*K*. *C* is the number of spectral bands in the hyperspectral data cube and *K* is the number of endmembers of the hyperspectral data cube. Use the `fippi`, `ppi`, or `nfindr` function to find the endmembers of a hyperspectral data cube.

Data Types: `single` | `double`

`numBands` — Number of bands to select

positive scalar

Number of most informative bands to select from the data cube, specified as a scalar.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`newhcube` — Output hyperspectral data

`hypercube` object

Output hyperspectral data, returned as a `hypercube` object.

`band` — Spectral band number of most informative bands

positive integer | vector of positive integers

Spectral band number of the most informative bands in the input data cube, returned as a positive integer or a vector of positive integers.

Data Types: `double`

References

[1] Du, Qian, and He Yang. “Similarity-Based Unsupervised Band Selection for Hyperspectral Image Analysis.” *IEEE Geoscience and Remote Sensing Letters*, Vol. 5, no. 4 (October 2008): 564–68. https://doi.org/10.1109/LGRS.2008.2000619.

See Also

`hypercube` | `removeBands` | `ppi` | `fippi` | `nfindr`

Introduced in R2020a

sharc

Perform atmospheric correction using satellite hypercube atmospheric rapid correction (SHARC)

Syntax

```
newhcube = sharc(hcube)
newhcube = sharc(hcube,Name,Value)
```

Description

`newhcube = sharc(hcube)` returns an atmospherically corrected data cube by computing the surface radiance or surface reflectance values from the input hyperspectral data. The function implements the SHARC algorithm to compute the atmospherically corrected hyperspectral data.

The input must be radiometrically corrected hyperspectral data. The pixel values of the input data cube must be either top of atmosphere (TOA) radiance or TOA reflectance values. For better results, use TOA reflectance values.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

`newhcube = sharc(hcube,Name,Value)` also specifies options using one or more name-value pair arguments. Use this syntax to set the parameter values for SHARC.

Examples

Perform Atmospheric Correction of Hyperspectral Data

Read a hyperspectral data cube into the workspace.

```
input = hypercube('EO1H0440342002212110PY_cropped.dat');
```

Determine the bad spectral band numbers using the `BadBands` parameter in the metadata.

```
bandNumber = find(~input.Metadata.BadBands);
```

Remove the bad spectral bands from the data cube.

```
input = removeBands(input,'BandNumber',bandNumber);
```

Convert the digital numbers to radiance values by using the `dn2radiance` function.

```
hcube = dn2radiance(input);
```

Convert the radiance values to reflectance values by using the `radiance2Reflectance` function.

```
hcube = radiance2Reflectance(hcube);
```

Compute atmospherically corrected data by using the `sharc` function.

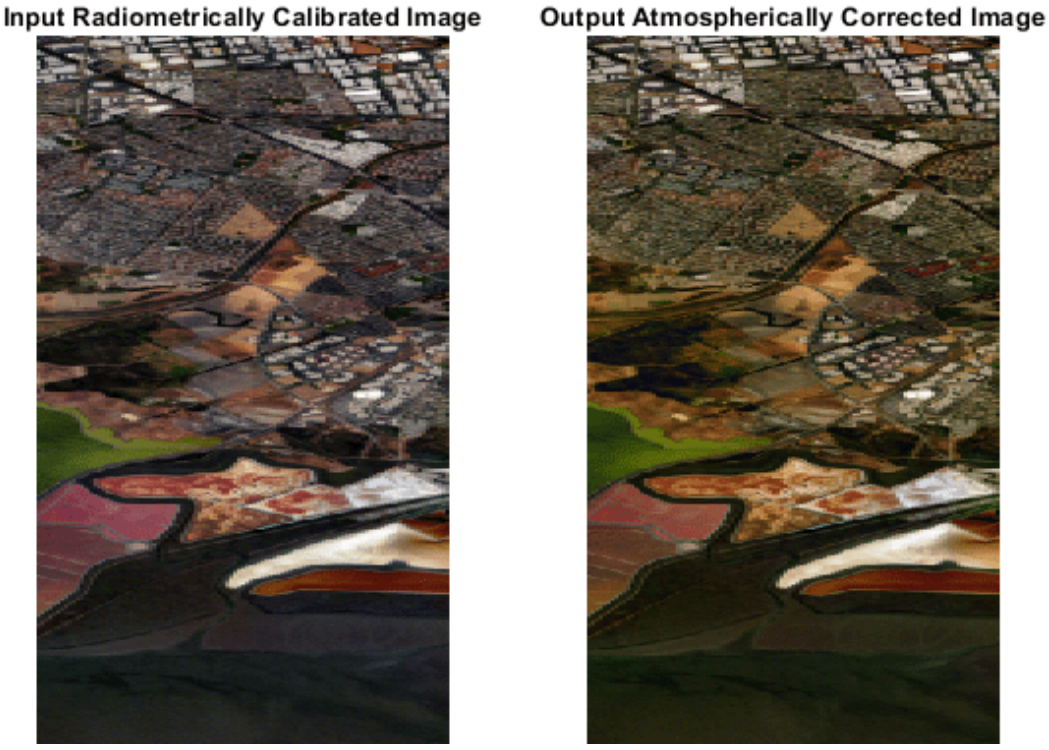
```
newhcube = sharc(hcube);
```

Estimate RGB images of the input and the atmospherically corrected output data. Increase the image contrast by applying contrast stretching.

```
inputImg = colorize(hcube,'Method','rgb','ContrastStretching',true);
outputImg = colorize(newhcube,'Method','rgb','ContrastStretching',true);
```

Display the contrast-stretched RGB images of the input and the atmospherically corrected output data.

```
figure('Position',[0 0 700 400])
subplot('Position',[0.1 0 0.3 0.9])
imagesc(inputImg)
title('Input Radiometrically Calibrated Image')
axis off
subplot('Position',[0.5 0 0.3 0.9])
imagesc(outputImg)
axis off
title('Output Atmospherically Corrected Image')
```



Specify Dark Pixel Location for Atmospheric Correction

Read a hyperspectral data cube into the workspace.

```
input = hypercube('EO1H0440342002212110PY_cropped.dat');
```

Determine the bad spectral band numbers using the `BadBands` parameter in the metadata.

```
bandNumber = find(~input.Metadata.BadBands);
```

Remove the bad spectral bands from the data cube.

```
input = removeBands(input,'BandNumber',bandNumber);
```

Convert the digital numbers to radiance values by using the `dn2radiance` function.

```
hcube = dn2radiance(input);
```

Convert the radiance values to reflectance values by using the `radiance2Reflectance` function.

```
hcube = radiance2Reflectance(hcube);
```

Compute atmospherically corrected data by using the `sharc` function. Specify the dark pixel location for computing the adjacency effect and the initial atmospheric parameters. The choice of the dark pixel affects the atmospheric correction results.

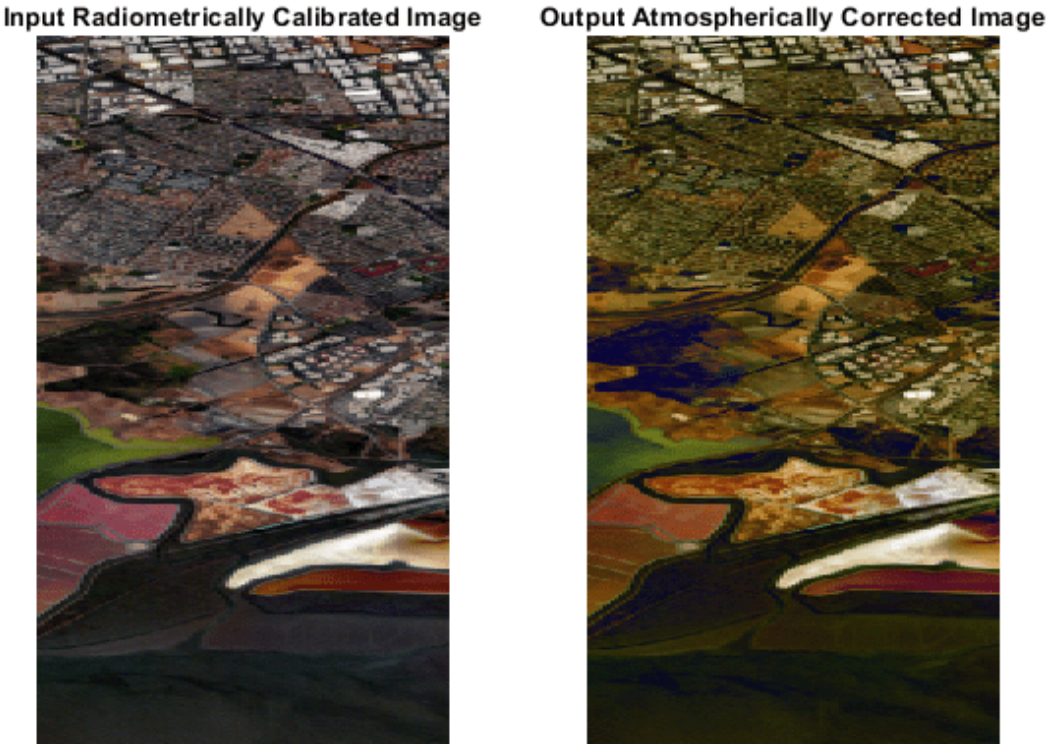
```
newhcube = sharc(hcube,'DarkPixelLocation',[217 7]);
```

Estimate RGB images of the input and the atmospherically corrected output data. Increase the image contrast by applying contrast stretching.

```
inputImg = colorize(hcube,'Method','rgb','ContrastStretching',true);
outputImg = colorize(newhcube,'Method','rgb','ContrastStretching',true);
```

Display the contrast-stretched RGB images of the input and the atmospherically corrected output data.

```
figure('Position',[0 0 700 400])
subplot('Position',[0.1 0 0.3 0.9])
imagesc(inputImg)
title('Input Radiometrically Calibrated Image')
axis off
subplot('Position',[0.5 0 0.3 0.9])
imagesc(outputImg)
axis off
title('Output Atmospherically Corrected Image')
```



Input Arguments

`hcube` — Input hyperspectral data

`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `DataCube` property of the `hypercube` object contains the hyperspectral data cube. The pixel values of the data cube must signify either the TOA radiance or TOA reflectance values.

If the pixel values are digital numbers, use the `dn2radiance` and `dn2reflectance` functions for computing the TOA radiance and reflectance values, respectively. You can also use the `radiance2Reflectance` function for directly computing TOA reflectance values from the radiance values.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `('AtmosphericModel','Tropical')`

`AtmosphericModel` — Atmospheric model for calculating optical thickness

`'1962 US Standard'` (default) | `'Tropical'` | `'Midlatitude Summer'` | `'Midlatitude Winter'` | `'Subarctic Summer'` | `'Subarctic Winter'`

Atmospheric model for calculating the optical thickness of the total atmosphere, specified as the comma-separated pair consisting of `'AtmosphericModel'` and one of these values:

- `'1962 US Standard'` — Standard atmospheric parameters that are defined for Earth's atmosphere over a wide range of temperature and pressure. This is a generic model and can be used for atmospheric correction of hyperspectral data acquired at different latitudes and seasons.
- `'Tropical'` — Atmospheric parameters of regions in the tropical zone. The latitude range of the tropical zone is [-23° 45', 23° 45']. Use this model if the hyperspectral data is collected from the tropical zone.
- `'Midlatitude Summer'` — Atmospheric parameters of regions in a midlatitude zone during the summer season. The latitude ranges for midlatitude zones are [23° 45', 66° 55'] and [-66° 55', -23° 45']. Use this model if the hyperspectral data is collected from mid-latitude zones during summer. You can determine the season from the acquisition date stored as `AcquistionTime` in the metadata.
- `'Midlatitude Winter'` — Atmospheric parameters of regions in a midlatitude zone during the winter season. The latitude ranges for midlatitude zones are [23° 45', 66° 55'] and [-66° 55', 23° 45']. Use this model if the hyperspectral data is collected from mid-latitude zones during winter. You can determine the season from the acquisition date stored as `AcquistionTime` in the metadata.
- `'Subarctic Summer'` — Atmospheric parameters of regions in a subarctic zone during the summer season. The latitude ranges for subarctic zones are [66° 55' ,90°] and [-90°, -66° 55'] degrees. Use this model if the hyperspectral data is collected from subarctic zones during summer. You can determine the season from the acquisition date stored as `AcquistionTime` in the metadata.
- `'Subarctic Winter'` — Atmospheric parameters of regions in a subarctic zone during the winter season. The latitude ranges for subarctic zones are [66° 55' ,90°] and [-90°, -66° 55'] degrees. Use this model if the hyperspectral data is collected from subarctic zones during winter. You can determine the season from the acquisition date stored as `AcquistionTime` in the metadata.

`DarkPixelLocation` — Location of dark pixel
1-by-2 vector of form [x y]

Location of the dark pixel, specified as the comma-separated pair consisting of `'DarkPixelLocation'` and a 1-by-2 vector of form [*x y*]. *x* and *y* are the spatial coordinates of the dark pixel to be selected from the blue-green band. The dark pixel has the lowest TOA reflectance value, and the `sharc` function uses it for computing the spectral illuminance of the surface.

By default, the `sharc` function chooses the pixel with the minimum blue-green band value as the dark pixel. Typically the wavelength range (in micrometers) for the blue-green band is [0.45, 0.57]. The TOA reflectance values in the blue-green band are most affected by atmospheric haze. Hence, the dark pixel from blue-green band is generally selected as the candidate pixel for estimating the atmospheric effects on the hyperspectral data.

`AdjacencyWindow` — Window size for computing adjacency effect
5 (default) | positive integer scalar

Window size for computing the adjacency effect, specified as the comma-separated pair consisting of `'AdjacencyWindow'` and a positive integer scalar. The value signifies the size of the window centered around the dark pixel. The `sharc` function uses all the pixels that lie inside this window for estimating the adjacency effect.

The window size must be less than the spatial dimension of the input hyperspectral data cube.

Output Arguments

`newhcube` — Atmospherically corrected data

`hypercube` object

Atmospherically corrected data, returned as a `hypercube` object. The `DataCube` property of the `hypercube` object contains the hyperspectral data cube. The pixel values of the output data cube are surface radiance or surface reflectance values depending on the input.

- If the pixel values of the input data cube are TOA radiances, the pixel values of the atmospherically corrected data at the output are surface radiance values.
- If the pixel values of the input data cube are TOA reflectances, the pixel values of the atmospherically corrected data at the output contains surface reflectance values.

References

[1] Katkovsky, Leonid, Anton Martinov, Volha Siliuk, Dimitry Ivanov, and Alexander Kokhanovsky. "Fast Atmospheric Correction Method for Hyperspectral Data." *Remote Sensing* 10, no. 11 (October 28, 2018): 1698. https://doi.org/10.3390/rs10111698.

See Also

`hypercube` | `dn2reflectance` | `dn2radiance`

Introduced in R2020b

sharpencnmf

Sharpen hyperspectral data using coupled nonnegative matrix factorization (CNMF) method

Syntax

```
outputData = sharpencnmf(lrData,hrData)
outputData = sharpencnmf(lrData,hrData,Name,Value)
```

Description

`outputData = sharpencnmf(lrData,hrData)` sharpens the low resolution hyperspectral data, `lrData` of a scene by using coupled nonnegative matrix factorization (CNMF) method. The CNMF method is an iterative approach that uses the high resolution multispectral or panchromatic data, `hrData` of the same scene for sharpening the hyperspectral data.

Hyperspectral image sharpening increases the spatial resolution of a hyperspectral data by fusing information from a high resolution multispectral data or a panchromatic data. The sharpening process is also called as image fusion (fusing multispectral and hyperspectral data) or pan-sharpening (fusing panchromatic and hyperspectral data).

Note

- The spatial dimension of the hyperspectral data must be less than the spatial dimension of the multispectral or panchromatic data.
- The number of spectral bands in the hyperspectral data must be greater than the number of bands in the multispectral data. For panchromatic data, the number of spectral bands is always 1.

`outputData = sharpencnmf(lrData,hrData,Name,Value)` also specifies options using one or more name-value pair arguments. Use this syntax to set the parameter values for CNMF method.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Sharpen Hyperspectral Image Using CNMF Method

Read a low spatial resolution hyperspectral image of a scene into the workspace.

```
hcube = hypercube('EO1H0440342002212110PY_hsi.hdr');
```

Read the high spatial resolution multispectral image of the same scene into the workspace.

```
pcube = hypercube('EO1H0440342002212110PY_msi.hdr');
```

Sharpen the low spatial resolution hyperspectral data by fusing information from the high spatial resolution multispectral data by using the CNMF method. The output is a high resolution hyperspectral data and the data cube has a spatial resolution same as that of the input multispectral data.

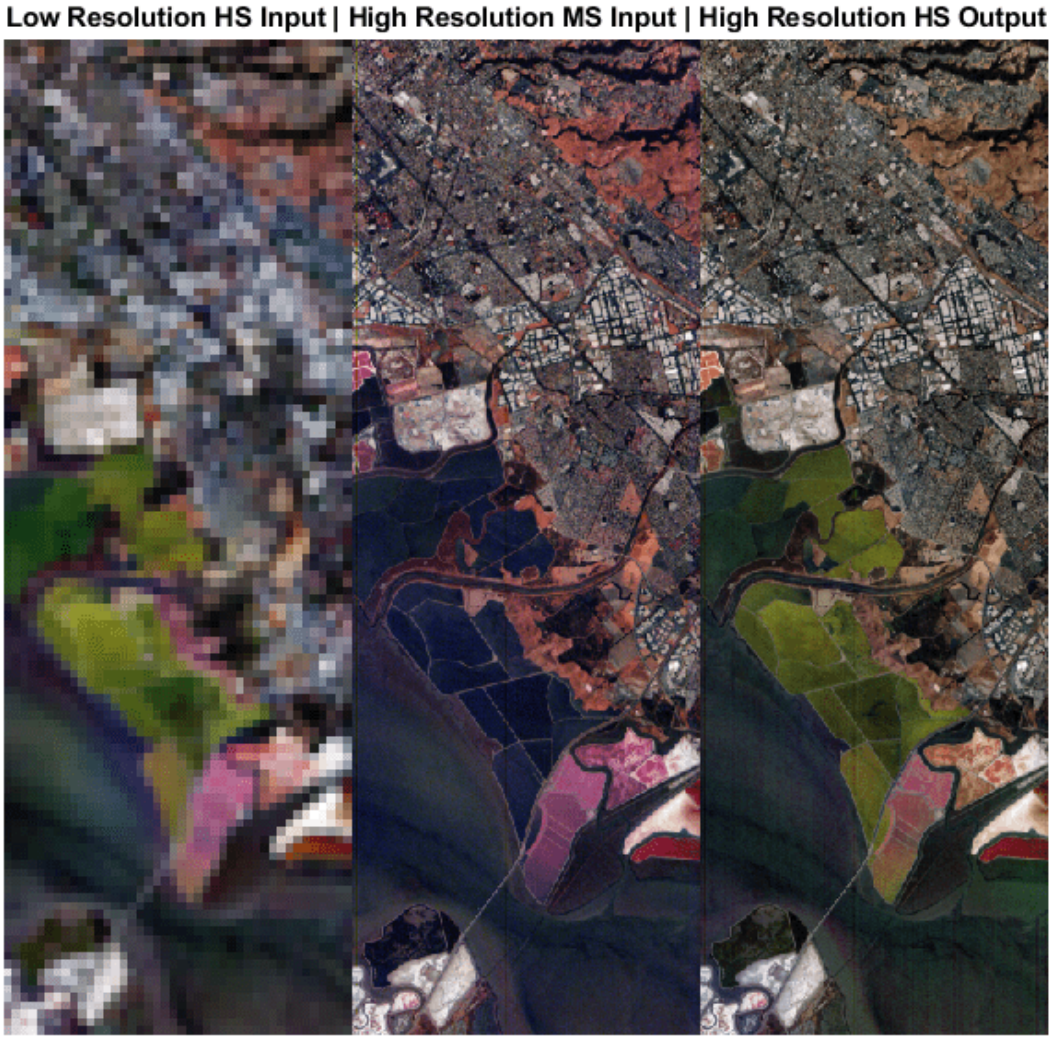
```
newhcube = sharpencnmf(hcube,pcube);
```

Estimate an RGB image of the input hyperspectral, input multispectral, and the sharpened hyperspectral output.

```
lrData = colorize(hcube,'method','rgb','ContrastStretching',true);
hrData = colorize(pcube,'method','rgb','ContrastStretching',true);
outputData = colorize(newhcube,'method','rgb','ContrastStretching',true);
```

Display the low spatial resolution hyperspectral (HS) image, high spatial resolution multispectral (MS) image, and high resolution HS output image.

```
figure
montage({lrData;hrData;outputData})
title('Low Resolution HS Input | High Resolution MS Input | High Resolution HS Output')
```



Set Convergence Threshold for Sharpening Hyperspectral Data

Read low spatial resolution hyperspectral image of a scene into the workspace.

```
hcube = hypercube('EO1H0440342002212110PY_hsi.hdr');
```

Read the high spatial resolution multispectral image of the same scene into the workspace.

```
pcube = hypercube('EO1H0440342002212110PY_msi.hdr');
```

Sharpen the low spatial resolution hyperspectral data by using the CNMF method. Set the convergence threshold value to 0.1.

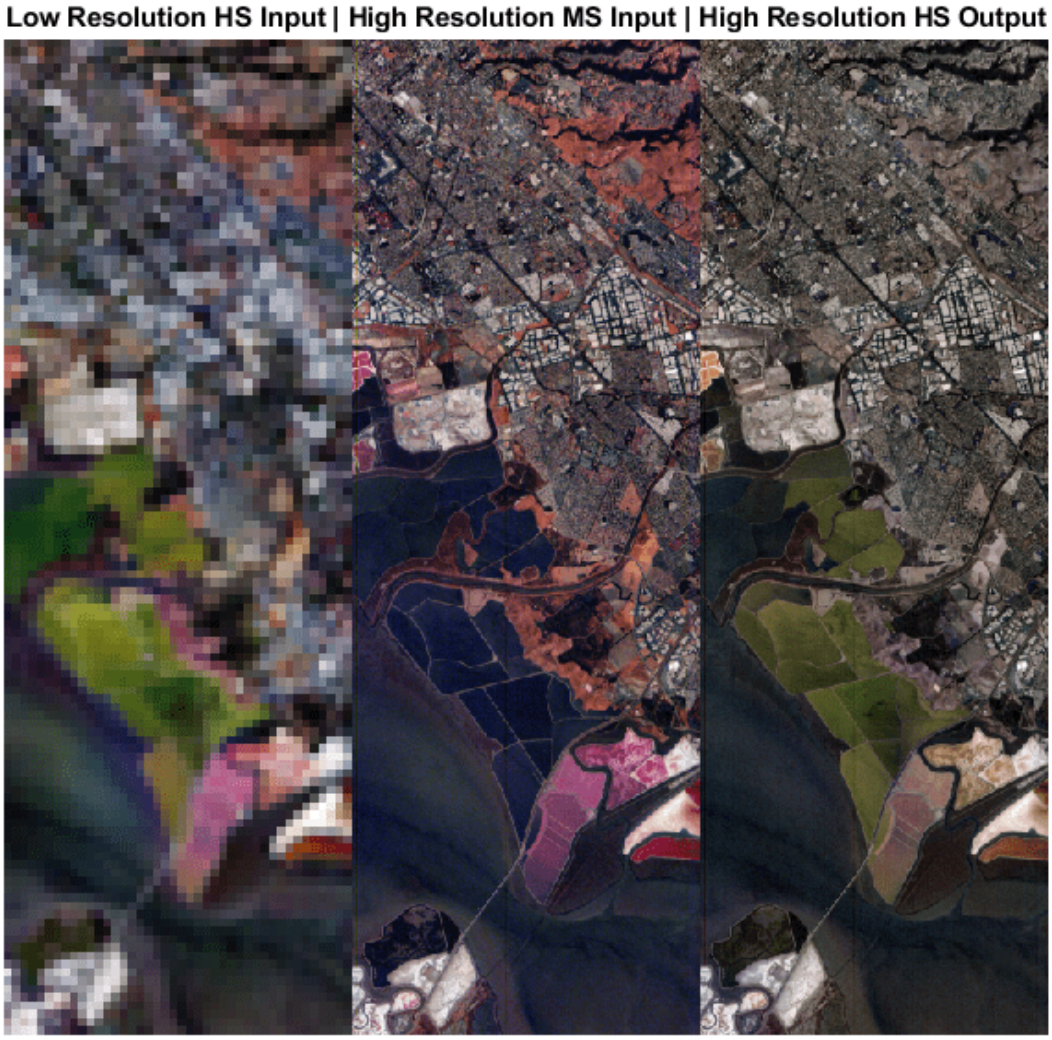
```
newhcube = sharpencnmf(hcube,pcube,'ConvergenceThreshold',0.1);
```

Estimate an RGB image of the input hyperspectral, input multispectral, and the sharpened hyperspectral output.

```
lrData = colorize(hcube,'method','rgb','ContrastStretching',true);
hrData = colorize(pcube,'method','rgb','ContrastStretching',true);
outputData = colorize(newhcube,'method','rgb','ContrastStretching',true);
```

Display the low spatial resolution hyperspectral (HS) image, high spatial resolution multispectral (MS) image, and high resolution HS output image.

```
figure
montage({lrData;hrData;outputData})
title('Low Resolution HS Input | High Resolution MS Input | High Resolution HS Output')
```



Input Arguments

`lrData` — Low resolution hyperspectral data

`hypercube` object | 3-D numeric array

Low resolution hyperspectral data, specified as a `hypercube` object or a 3-D numeric array containing the data cube. If the input is a `hypercube` object, the data is read from the `DataCube` property of the object.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`hrData` — High resolution input

`hypercube` object | 3-D numeric array | matrix

High resolution input, specified as a `hypercube` object, 3-D numeric array containing the data cube, or matrix. If the input is a `hypercube` object, the data is read from the `DataCube` property of the object.

The high resolution input is either a multispectral or panchromatic data.

- For multispectral data, the input value must be `hypercube` object or 3-D numeric array containing the data cube.
- For panchromatic data, the input value can be any of these:
  - A `hypercube` object or 3-D numeric array containing the data cube. The number of spectral bands in the data cube must be 1.
  - A matrix.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `cnmf(lrData,hrData,'MaxConvergenceIterations',40)`

`MaxConvergenceIterations` — Maximum number of iterations required for convergence

25 (default) | positive integer scalar

Maximum number of iterations required for convergence, specified as the comma-separated pair consisting of `'MaxConvergenceIterations'` and a positive integer scalar.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`MaxOptimizationIterations` — Maximum number of iterations to optimize unmixing

2 (default) | positive integer scalar

Maximum number of iterations required to optimize spectral unmixing of hyperspectral and multispectral data, specified as the comma-separated pair consisting of `'MaxOptimizationIterations'` and a positive integer scalar. If the high resolution input `hrData` is a panchromatic image, the spectral unmixing is performed only on the hyperspectral data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`ConvergenceThreshold` — Threshold for convergence
`0.0001` (default) | positive scalar

Threshold for convergence, specified as the comma-separated pair consisting of `'ConvergenceThreshold'` and a positive scalar. If you increase the convergence threshold, the accuracy of spectral unmixing decreases.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`NumEndmembers` — Number of endmembers for spectral unmixing
positive integer scalar

Number of endmembers for spectral unmixing, specified as the comma-separated pair consisting of `'NumEndmembers'` and a positive integer scalar.

For the default value, the `sharpencnmf` first counts the total number of endmembers ($Total_{EM}$) in the hyperspectral data by using the `countEndmembersHFC` function. Then, computes the number of endmembers for unmixing pixel values as, min(40,$Total_{EM}$).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`outputData` — Sharpened hyperspectral data
`hypercube` object | 3-D numeric array

Sharpened hyperspectral data, returned as a `hypercube` object or 3-D numeric array.

If the low resolution hyperspectral data cube specified by `lrData` is of size *P*-by-*Q*-by-*C* and the high resolution data `hrData` is of size *M*-by-*N*-by-*K* then the sharpened output

is of size *M*-by-*N*-by-*C*.

References

[1] Yokoya, Naoto, Takehisa Yairi, and Akira Iwasaki. “Coupled Nonnegative Matrix Factorization Unmixing for Hyperspectral and Multispectral Data Fusion.” *IEEE Transactions on Geoscience and Remote Sensing* 50, no. 2 (February 2012): 528–37. https://doi.org/10.1109/TGRS.2011.2161320.

See Also

`countEndmembersHFC` | `hypercube` | `nfindr` | `ppi`

Introduced in R2020b

sid

Measure spectral similarity using spectral information divergence

Syntax

```
score = sid(inputData,refSpectra)
score = sid(testSpectra,refSpectra)
```

Description

`score = sid(inputData,refSpectra)` measures the spectral similarity between the spectra of each pixel in the hyperspectral data `inputData` and the specified reference spectra `refSpectra` by using the spectral information divergence (SID) technique. Use this syntax to identify different regions or materials in a hyperspectral data cube.

`score = sid(testSpectra,refSpectra)` measures the spectral similarity between the specified test spectra `testSpectra` and reference spectra `refSpectra` by using the SID method. Use this syntax to compare the spectral signature of an unknown material against the reference spectra or to compute spectral variability between two spectral signatures.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Distinguish Hyperspectral Regions Using Spectral Information Divergence

Distinguish different regions in a hyperspectral data cube by computing the spectral information divergence (SID) between each pixel spectrum and the endmember spectrum of the data cube.

Read hyperspectral data into the workspace.

```
hcube = hypercube('jasperRidge2_R198.hdr');
```

Specify the number of spectrally distinct bands to identify in the data cube.

```
numEndmembers = 7;
```

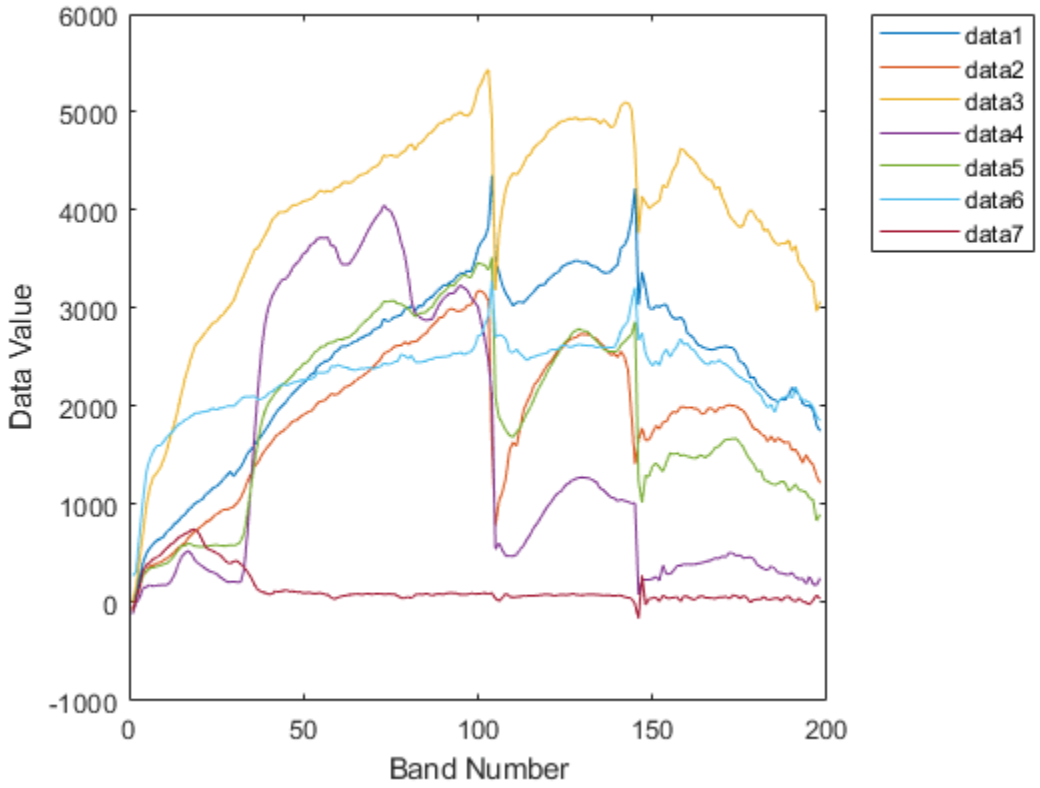
Extract endmember spectral signatures from the data cube by using the NFINDR algorithm.

```
endmembers = nfindr(hcube,numEndmembers);
```

Plot the spectral signatures of the endmembers.

```
figure
plot(endmembers)
xlabel('Band Number')
```

```
ylabel('Data Value')
legend('Location','Bestoutside')
```



Compute the spectral information divergence between each endmember and the spectrum of each pixel in the data cube.

```
score = zeros(size(hcube.DataCube,1),size(hcube.DataCube,2),numEndmembers);
for i= 1:numEndmembers
    score(:,:,i) = sid(hcube,endmembers(:,i));
end
```

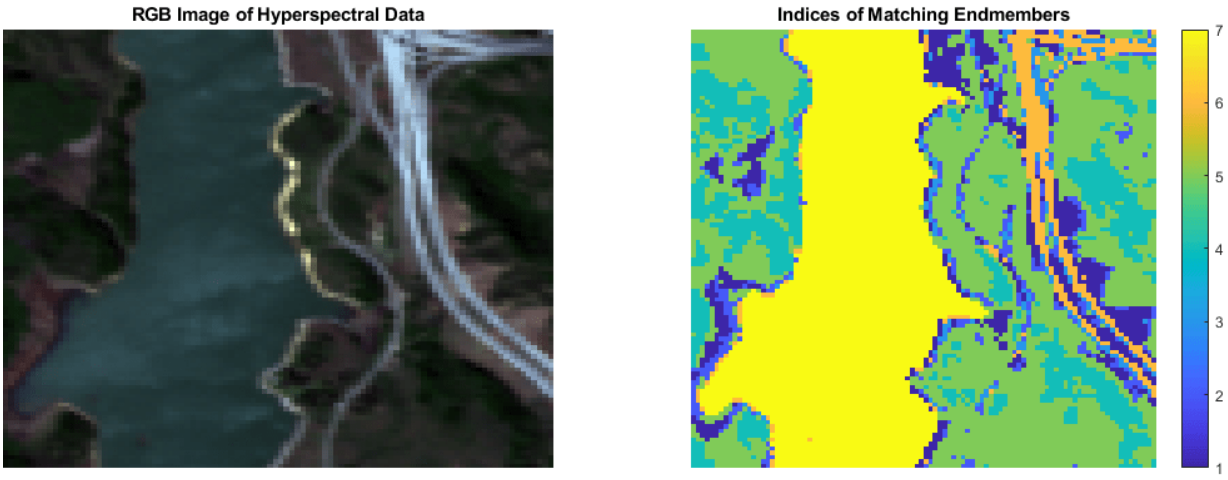
Compute the minimum score value from the distance scores obtained for each pixel spectrum with respect to all the endmembers. The index of each minimum score identifies the endmember spectrum to which a pixel spectrum exhibits maximum similarity. An index value, *n*, at the spatial location (*x*, *y*) in the score matrix indicates that the spectral signature of the pixel at spatial location (*x*, *y*) in the data cube best matches the spectral signature of the *n*th endmember.

```
[~,matchingIndx] = min(score,[],3);
```

Estimate an RGB image of the hyperspectral data cube by using the `colorize` function. Display both the RGB image and the matrix of matched index values.

```
rgbImg = colorize(hcube,'Method','RGB');
figure('Position',[0 0 1100 500])
subplot('Position',[0 0.2 0.4 0.7])
imagesc(rgbImg)
axis off
```

```
colormap default
title('RGB Image of Hyperspectral Data')
subplot('Position',[0.5 0.2 0.4 0.7])
imagesc(matchingIndx);
axis off
title('Indices of Matching Endmembers')
colorbar
```



Determine Similarity of Endmember Spectra Using SID

Read hyperspectral data into the workspace.

```
hcube = hypercube('jasperRidge2_R198.hdr');
```

Find 10 endmembers of the hyperspectral data cube by using the N-FINDR method.

```
numEndmembers = 10;
endmembers = nfindr(hcube,numEndmembers);
```

Consider the first endmember as the reference spectrum and the rest of the endmembers as the test spectrum. Compute the SID score between the reference and test spectra.

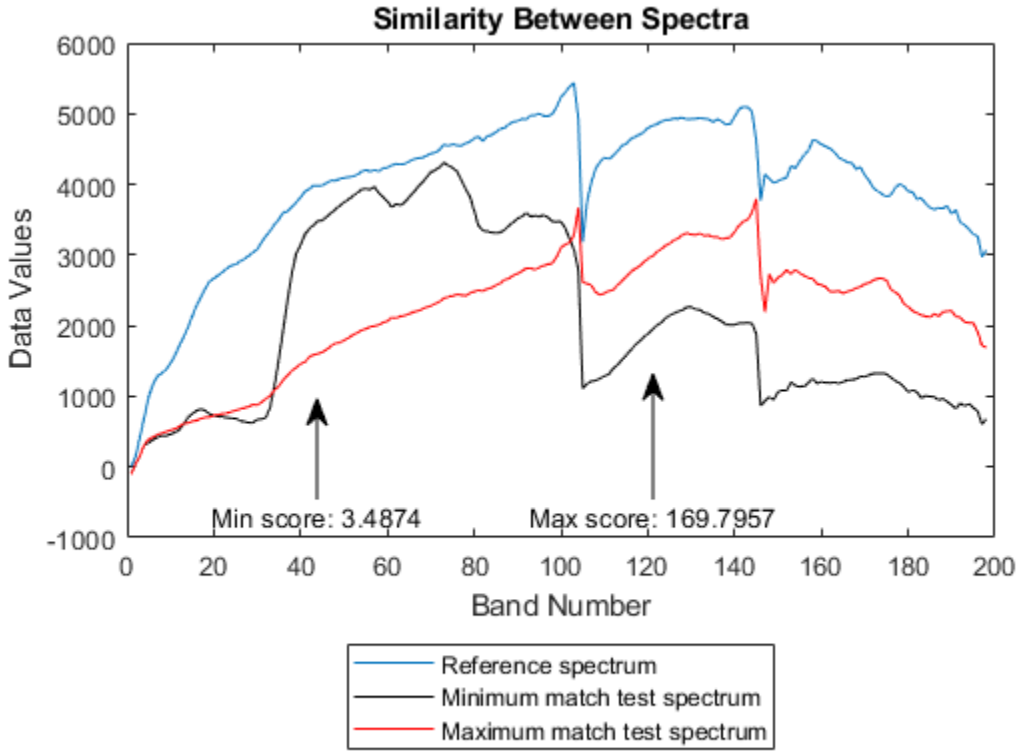
```
score = zeros(1,numEndmembers-1);
refSpectrum = endmembers(:,1);
for i = 2:numEndmembers
    testSpectrum = endmembers(:,i);
    score(i-1) = sid(testSpectrum,refSpectrum);
end
```

Find the test spectrum that exhibit maximum similarity (minimum distance) to the reference spectrum. Then find the test spectrum that exhibit minimum similarity (maximum distance) to the reference spectrum.

```
[minval,minidx] = min(score);
maxMatch = endmembers(:,minidx);
[maxval,maxidx] = max(score);
minMatch = endmembers(:,maxidx);
```

Plot the reference spectrum, maximum similarity test spectrum, and the minimum similarity test spectrum. The test spectrum with the minimum score has the highest similarity to the reference endmember. On the other hand, the test spectrum with maximum score has the highest spectral variability and characterises the spectral behaviour of two different materials.

```
figure
plot(refSpectrum)
hold on
plot(maxMatch,'k')
plot(minMatch,'r')
legend('Reference spectrum','Minimum match test spectrum','Maximum match test spectrum',...
       'Location','Southoutside');
title('Similarity Between Spectra')
annotation('textarrow',[0.3 0.3],[0.4 0.52],'String',['Min score: ' num2str(minval)])
annotation('textarrow',[0.6 0.6],[0.4 0.55],'String',['Max score: ' num2str(maxval)])
xlabel('Band Number')
ylabel('Data Values')
```



Input Arguments

`inputData` — Input hyperspectral data

`hypercube` object | 3-D numeric array

Input hyperspectral data, specified as a `hypercube` object or a 3-D numeric array containing the data cube. If the input is a `hypercube` object, the data is read from the `DataCube` property of the object.

`testSpectra` — Test spectra

C-element vector

Test spectra, specified as a *C*-element vector. The test spectra is the spectral signature of an unknown region or material.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`refSpectra` — Reference spectra

C-element vector

Reference spectra, specified as a *C*-element vector. The reference spectra is the spectral signature of a known region or material. The function matches the test spectra against these values.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`score` — SID score

scalar | matrix

SID score, returned as a scalar or matrix. The output is a

- scalar — If you specify the `testSpectra` input argument. The function matches the test spectral signature against the reference spectral signature and returns a scalar value. Both the test and the reference spectra must be vectors of same length.
- matrix — If you specify the `inputData` input argument. The function matches the spectral signature of each pixel in the data cube against the reference spectral signature and returns a matrix. If the data cube is of size *M*-by-*N*-by-*C* and the reference spectra is a vector of length *C*, the output matrix is of size *M*-by-*N*.

A smaller SAM score indicates a strong match between the test signature and the reference signature.

Data Types: `single` | `double`

More About

Spectral information divergence

The spectral information divergence (SID) method computes spectral similarity based on the divergence between the probability distributions of the two spectra. Let *r* and *t* be the reference and test spectra respectively. Calculate the distribution values for the reference spectra as:

.

Calculate the distribution values for the test spectra as:

.

Then, compute the SID value by using the probability distributions of the reference and the test spectra:

References

[1] Chein-I Chang. “An Information-Theoretic Approach to Spectral Variability, Similarity, and Discrimination for Hyperspectral Image Analysis.” *IEEE Transactions on Information Theory* 46, no. 5 (August 2000): 1927–32. https://doi.org/10.1109/18.857802.

See Also

`spectralMatch` | `readEcostressSig` | `sam` | `hypercube` | `sidsam`

Introduced in R2020a

sidsam

Measure spectral similarity using spectral information divergence-spectral angle mapper hybrid method

Syntax

```
score = sidsam(inputData,refSpectrum)
score = sidsam(testSpectrum,refSpectrum)
```

Description

`score = sidsam(inputData,refSpectrum)` measures the spectral similarity between the spectrum of each pixel in the hyperspectral data `inputData` and the specified reference spectrum `refSpectrum` by using the spectral information divergence-spectral angle mapper (SID-SAM) hybrid method. Use this syntax to identify different regions or materials in a hyperspectral data cube. For information about the SID-SAM method, see “More About” on page 1-3385.

`score = sidsam(testSpectrum,refSpectrum)` measures the spectral similarity between the specified test spectrum `testSpectrum` and reference spectrum `refSpectrum` by using the SID-SAM hybrid method. Use this syntax to compare the spectral signature of an unknown material against the reference spectrum or to compute spectral variability between two spectral signatures.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Distinguish Hyperspectral Regions Using SID-SAM Hybrid Measure

Read hyperspectral data into the workspace.

```
hcube = hypercube('jasperRidge2_R198.hdr');
```

Estimate the number of spectrally distinct endmembers in the data cube by using `countEndmembersHFC` function.

```
numEndmembers = countEndmembersHFC(hcube,'PFA',10^-7);
```

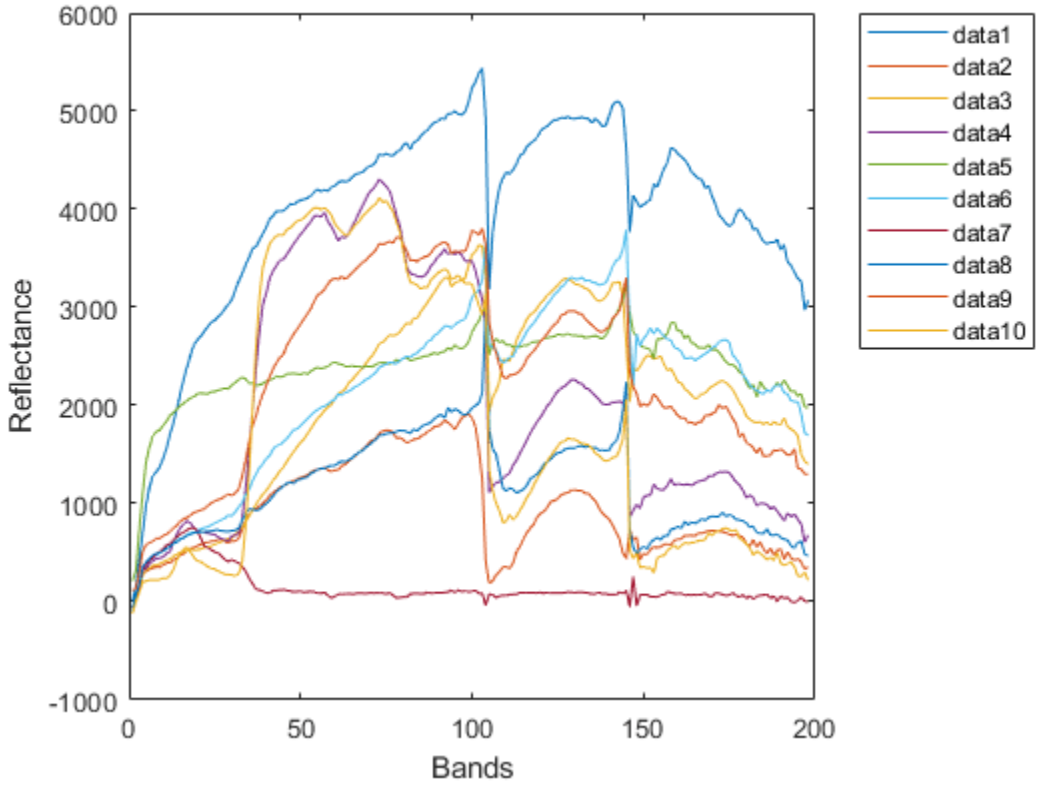
Extract the endmember spectral signatures from the data cube by using the NFINDR algorithm.

```
endmembers = nfindr(hcube,numEndmembers);
```

Plot the spectral signatures of the extracted endmembers.

```
figure
plot(endmembers)
xlabel('Bands')
```

```
ylabel('Reflectance')
legend('Location','Bestoutside')
```



Compute the SID-SAM distance between each endmember and the spectrum of each pixel in the data cube.

```
score = zeros(size(hcube.DataCube,1),size(hcube.DataCube,2),numEndmembers);
for i = 1:numEndmembers
    score(:,:,i) = sidsam(hcube,endmembers(:,i));
end
```

Compute the minimum score value from the distance scores obtained for each pixel spectrum with respect to all the endmembers. The index of each minimum score identifies the endmember spectrum to which a pixel spectrum exhibits maximum similarity. An index value, *n*, at the spatial location (*x*, *y*) in the score matrix indicates that the spectral signature of the pixel at spatial location (*x*, *y*) in the data cube best matches the spectral signature of the *n*th endmember.

```
[~,matchingIdx] = min(score,[],3);
```

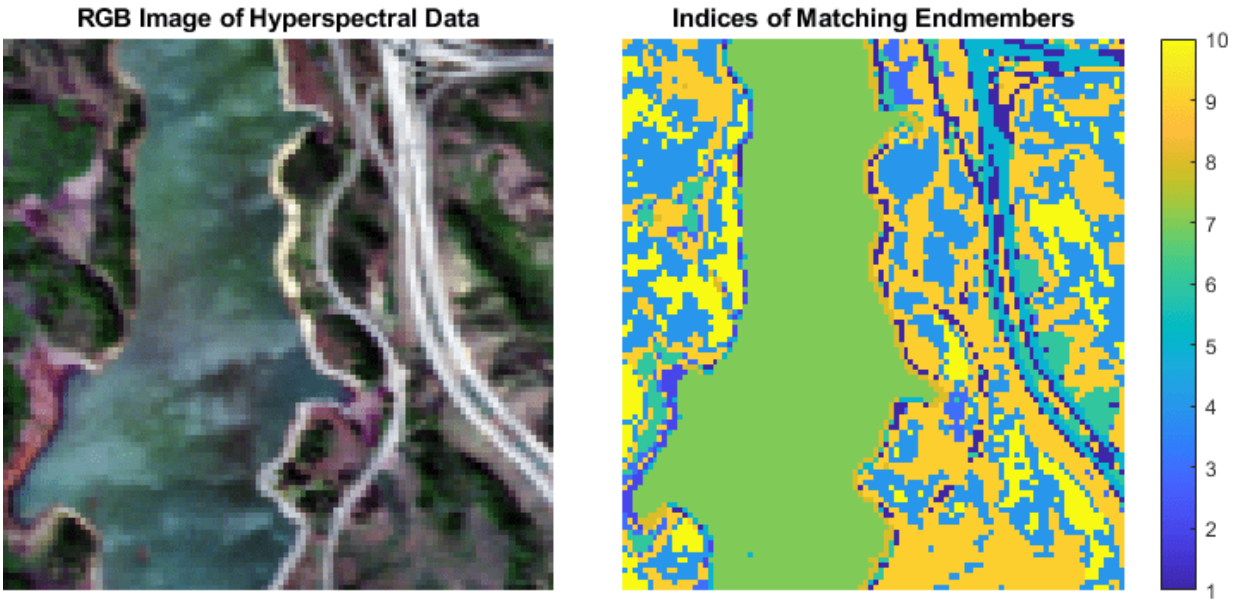
Estimate an RGB image of the input data by using the `colorize` function.

```
rgbImg = colorize(hcube,'Method','rgb','ContrastStretching',true);
```

Display both the RGB image and the matrix of matched index values.

```
figure('Position',[0 0 800 400])
subplot('Position',[0 0.1 0.4 0.8])
imagesc(rgbImg)
```

```
axis off
title('RGB Image of Hyperspectral Data')
subplot('Position',[0.45 0.1 0.45 0.8])
imagesc(matchingIdx)
axis off
title('Indices of Matching Endmembers')
colorbar
```



Determine Similarity of Endmember Spectra Using SID-SAM

Read hyperspectral data into the workspace.

```
hcube = hypercube('jasperRidge2_R198.hdr');
```

Find the first 10 endmembers of the hyperspectral data.

```
numEndmembers = 10;
endmembers = nfindr(hcube,numEndmembers);
```

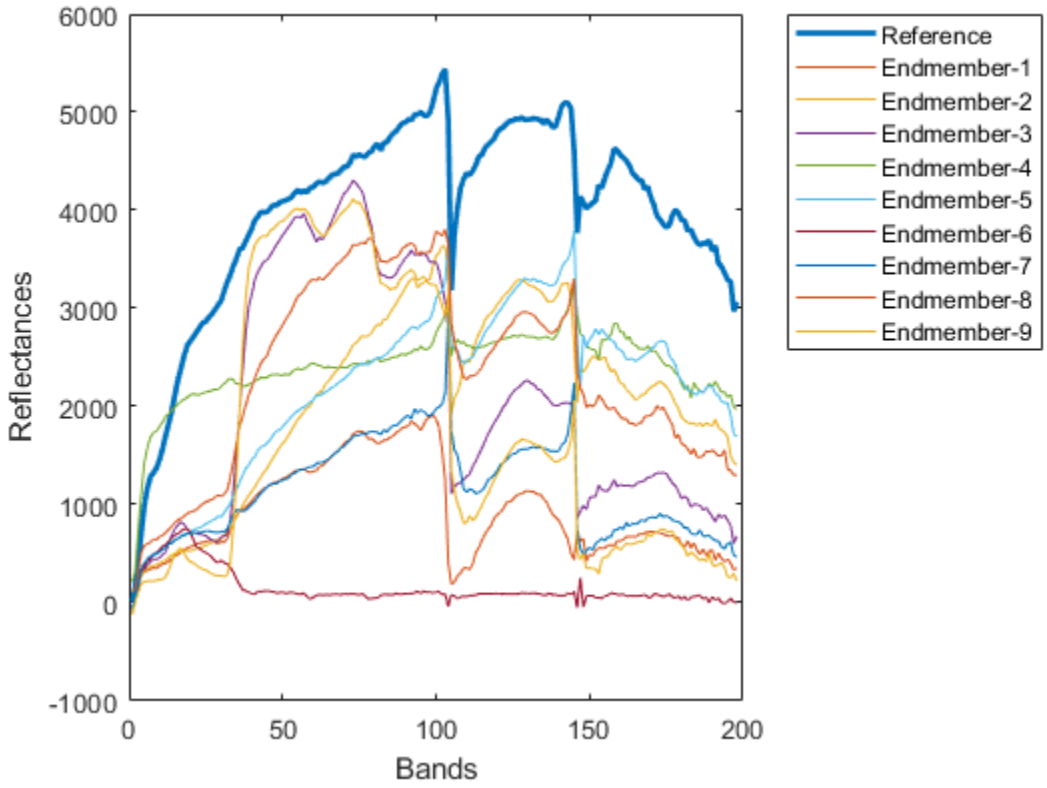
Consider the first endmember as the reference spectrum and the rest of the endmembers as the test spectrum.

```
refSpectrum = endmembers(:,1);
testSpectra = endmembers(:,2:end);
```

Plot the reference spectrum and other endmember spectra.

```
figure
plot(refSpectrum,'LineWidth',2);
hold on
```

```
plot(testSpectra);
hold off
label = cell(1,numEndmembers-1);
label{1} = 'Reference';
for itr = 1:numEndmembers-1
    label{itr+1} = ['Endmember-' num2str(itr)];
end
xlabel('Bands')
ylabel('Reflectances')
legend(label,'Location','Bestoutside');
```



Compute the SID-SAM score between the reference and test spectra.

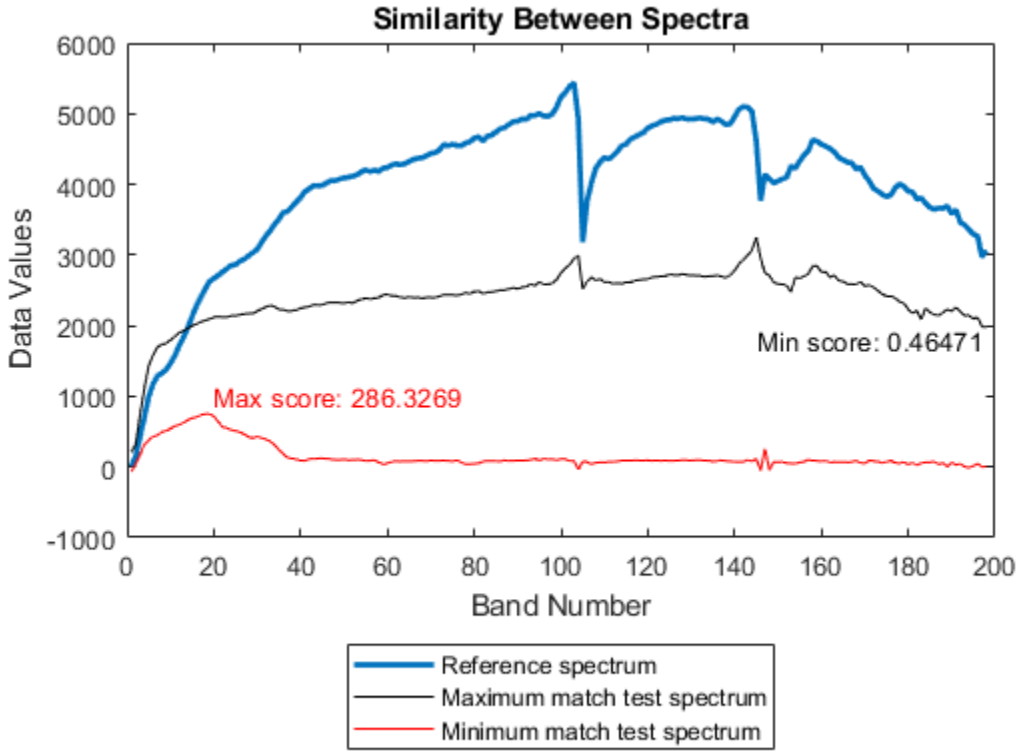
```
score = zeros(1,numEndmembers-1);
for itr = 1:numEndmembers-1
    testSpectrum = testSpectra(:,itr);
    score(itr) = sidsam(testSpectrum,refSpectrum);
end
```

Find the test spectrum that exhibit maximum similarity (minimum distance) to the reference spectrum. Then find the test spectrum that exhibit minimum similarity (maximum distance) to the reference spectrum.

```
[minval,minidx] = min(score);
maxMatch = testSpectra(:,minidx);
[maxval,maxidx] = max(score);
minMatch = testSpectra(:,maxidx);
```

Plot the reference spectrum, the maximum similarity, and the minimum similarity test spectra. The test spectrum with the minimum score value indicates highest similarity to the reference endmember. On the other hand, the test spectrum with the maximum score value has the highest spectral variability and characterises the spectral behaviour of two different materials.

```
figure
plot(refSpectrum,'LineWidth',2)
hold on
plot(maxMatch,'k')
plot(minMatch,'r')
xlabel('Band Number')
ylabel('Data Values')
legend('Reference spectrum','Maximum match test spectrum','Minimum match test spectrum',...
       'Location','Southoutside')
title('Similarity Between Spectra')
text(20,1000,['Max score: ' num2str(maxval)],'Color','r')
text(145,1800,['Min score: ' num2str(minval)],'Color','k')
```



Input Arguments

`inputData` — Input hyperspectral data

`hypercube` object | 3-D numeric array

Input hyperspectral data, specified as a `hypercube` object or a 3-D numeric array containing the data cube. If the input is a `hypercube` object, the data is read from the `DataCube` property of the object.

`testSpectrum` — Test spectrum

C-element vector

Test spectrum, specified as a *C*-element vector. The test spectrum is the spectral signature of an unknown region or material.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`refSpectrum` — Reference spectrum

C-element vector

Reference spectrum, specified as a *C*-element vector. The reference spectrum is the spectral signature of a known region or material. The function matches the test spectrum against these values.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`score` — SID-SAM score

scalar | matrix

SID-SAM score, returned as a scalar or matrix. The output is a

- scalar — If you specify the `testSpectrum` input argument. The function matches the test spectral signature against the reference spectral signature and returns a scalar value. Both the test and the reference spectra must be vectors of same length.
- matrix — If you specify the `inputData` input argument. The function matches the spectral signature of each pixel in the data cube against the reference spectral signature and returns a matrix. If the data cube is of size *M*-by-*N*-by-*C* and the reference spectra is a vector of length *C*, the output matrix is of size *M*-by-*N*.

A smaller SID-SAM score indicates a strong match between the test signature and the reference signature.

Data Types: `single` | `double`

More About

SID-SAM

Given the test spectra t and a reference spectra r of length C, the SAM score α is calculated as

Compute the SID value by using the probability distributions of the reference and the test spectra:

where,

Then, compute the SID-SAM hybrid score as

References

[1] Chang, Chein-I. “New Hyperspectral Discrimination Measure for Spectral Characterization.” *Optical Engineering* 43, no. 8 (August 1, 2004): 1777. https://doi.org/10.1117/1.1766301.

See Also

`spectralMatch` | `readEcostressSig` | `sid` | `hypercube` | `jmsam` | `ns3` | `sam`

Introduced in R2020b

smileMetric

Compute spectral smile metrics of hyperspectral data

Syntax

```
[oxystd,carbonstd,oxyderiv,carbonderiv] = smileMetric(hcube)
```

Description

`[oxystd,carbonstd,oxyderiv,carbonderiv] = smileMetric(hcube)` computes the column mean derivatives, and their standard deviations, for the oxygen and carbon-dioxide absorption features of a hyperspectral data set. You can use these values to detect the spectral smile effect in the hyperspectral data set. For more information, see “Smile Indicators” on page 1-3390.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Compute Derivative Values for Oxygen and Carbon-Dioxide Absorption Features

Load the hyperspectral data into the workspace.

```
hcube = hypercube('EO1H0440342002212110PY_cropped.dat');
```

Compute the column mean derivative values, and their standard deviations, for the oxygen and carbon-dioxide absorption features of the hyperspectral dataset `hcube`.

```
[oxystd,carbonstd,oxyderiv,carbonderiv] = smileMetric(hcube);
```

Perform spectral smile reduction using the maximum noise fraction (MNF) transform-based method.

```
correctedData = reduceSmile(hcube,'Method','MNF');
```

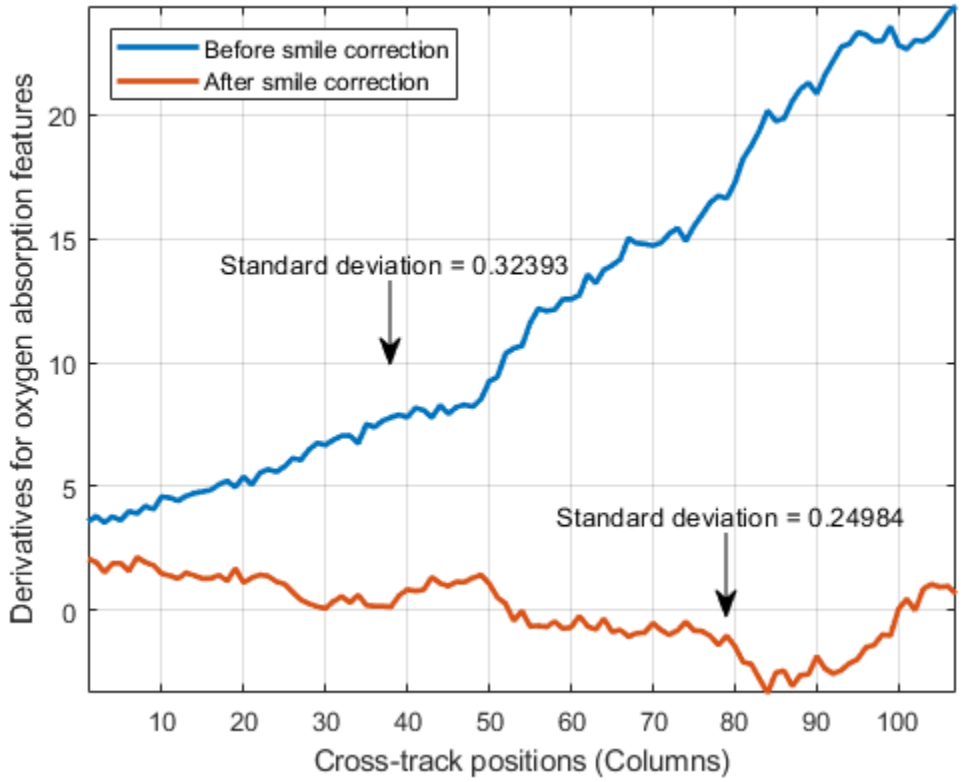
Compute the column mean derivative values, and their standard deviations, for the oxygen and carbon-dioxide absorption features of the smile-corrected hyperspectral dataset `correctedData`.

```
[noxystd,ncarbonstd,noxyderiv,ncarbonderiv] = smileMetric(correctedData);
```

Plot the column mean derivative values of the oxygen absorption feature for both the uncorrected hypercube `hcube` and the smile-corrected hypercube `correctedData`, and display their standard deviations.

```
figure
plot(oxyderiv,'LineWidth',2)
hold on
plot(noxyderiv,'LineWidth',2)
hold off
```

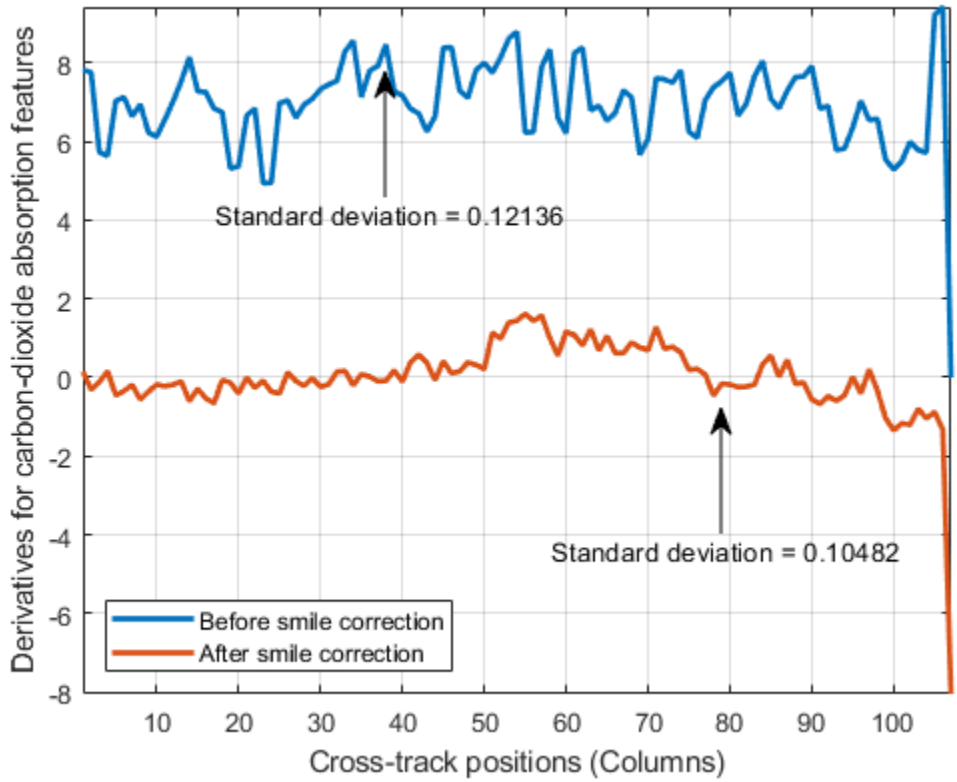
```
axis tight
grid on
xlabel('Cross-track positions (Columns)')
ylabel('Derivatives for oxygen absorption features')
legend({'Before smile correction','After smile correction'},'Location','northwest');
annotation(gcf,'textarrow',[0.4 0.4],[0.6 0.5],...
    'String',['Standard deviation = ' num2str(oxystd)]);
annotation(gcf,'textarrow',[0.7 0.7],[0.3 0.2],...
    'String',['Standard deviation = ' num2str(noxystd)]);
```



Plot the column mean derivative values of the carbon-dioxide absorption feature for both the uncorrected hypercube `hcube` and the smile-corrected hypercube `correctedData`, and display their standard deviations.

```
figure
plot(carbonderiv,'LineWidth',2)
hold on
plot(ncarbonderiv,'LineWidth',2)
hold off
axis tight
grid on
xlabel('Cross-track positions (Columns)')
ylabel('Derivatives for carbon-dioxide absorption features')
legend({'Before smile correction','After smile correction'},'Location','southwest');
annotation(gcf,'textarrow',[0.4 0.4],[0.7 0.85],...
    'String',['Standard deviation = ' num2str(carbonstd)]);
```

```
annotation(gcf,'textarrow',[0.7 0.7],[0.3 0.45],...
    'String',['Standard deviation = ' num2str(ncarbonstd)]);
```



Input Arguments

hcube — Input hyperspectral data

`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `DataCube` property of the `hypercube` object stores the hyperspectral data cube. To calculate the column mean of oxygen and carbon-dioxide absorption feature derivatives, the `hypercube` object must have the full width half maximum (`FWHM`) values in the `Metadata` property.

Note

- To compute the column mean of oxygen absorption feature derivatives, the input hyperspectral data must contain data in the visible and near-infrared (VNIR) wavelength range 760 - 785 nm.
- To compute the column mean of carbon-dioxide absorption feature derivatives, the input hyperspectral data must contain data in the short-wave-infrared (SWIR) wavelength range 2010 - 2025 nm.

Output Arguments

`oxyderiv` — Column mean derivatives for oxygen absorption features

N-element row vector

Column mean derivatives for the oxygen absorption features, returned as a *N*-element row vector. *N* is the number of columns in the input hyperspectral data cube. If the input hyperspectral data cube is of type `double`, then the output vector is of data type `double`. Otherwise, the data type of the output vector is `single`.

`carbonderiv` — Column mean derivative for carbon-dioxide absorption features

N-element row vector

Column mean derivatives for the carbon-dioxide absorption features, returned as a *N*-element row vector. *N* is the number of columns in the input hyperspectral data cube. If the input hyperspectral data cube is of type `double`, then the output vector is of data type `double`. Otherwise, the data type of the output vector is `single`.

`oxystd` — Standard deviation of column mean derivatives for oxygen absorption features

scalar

Standard deviation of the column mean derivatives for oxygen absorption features, returned as a scalar. You can use this scalar to detect the presence of the spectral smile effect in hyperspectral data. If the value of `oxystd` is low, then the chances of the data having a smile effect is less in the VNIR range.

`carbonstd` — Standard deviation of column mean derivatives for carbon-dioxide absorption features

scalar

Standard deviation of the column mean derivatives for carbon-dioxide absorption features, returned as a scalar. You can use this scalar to detect the presence of the spectral smile effect in hyperspectral data. If the value of `carbonstd` is low, then the chances of the data having a smile effect is less in the SWIR range.

More About

Smile Indicators

The smile effect occurs when hyperspectral data contains significant cross-track curvature with nonlinear disturbances along the spectral dimension. These nonlinear disturbances occur only in data captured using push-broom hyperspectral sensors, such as the Hyperion EO-1 and SEBASS. Based on [1], you can detect cross-track variation in the oxygen and carbon-dioxide absorption features, due to a possible smile effect, by calculating the first derivatives of the oxygen and carbon-dioxide band images. The first derivative of the adjacent bands B' is calculated using the absorption band image B_1 and the image of the subsequent band B_2, using the equation:

where, $\overline{FWHM}$ is the average FWHM of the two bands B_1 and B_2. This derivative calculation is applicable to both the oxygen and carbon-dioxide absorption band images. The column mean values of the oxygen and carbon-dioxide derivatives can indicate cross-track nonlinearity caused by the spectral smile effect.

- The nonlinear, cross-track column mean of oxygen absorption feature derivative values indicates a spectral smile effect in the VNIR spectrum.
- The nonlinear, cross-track column mean of carbon-dioxide absorption feature derivative values indicates a spectral smile effect in the SWIR spectrum.

References

[1] Dadon, Alon, Eyal Ben-Dor, and Arnon Karnieli. “Use of Derivative Calculations and Minimum Noise Fraction Transform for Detecting and Correcting the Spectral Curvature Effect (Smile) in Hyperion Images.” *IEEE Transactions on Geoscience and Remote Sensing* 48, no. 6 (June 2010): 2603–12. https://doi.org/10.1109/TGRS.2010.2040391.

See Also

`reduceSmile` | `hypercube`

Introduced in R2021a

subtractDarkPixel

Subtract dark pixel value from hyperspectral data cube

Syntax

```
correctedData = subtractDarkPixel(inputData)
correctedData = subtractDarkPixel(inputData,darkPixels)
correctedData = subtractDarkPixel( ___ ,'BlockSize',blocksize)
```

Description

`correctedData = subtractDarkPixel(inputData)` subtracts the minimum pixel value of each band from all pixels in that band of the hyperspectral data, `inputData`. The pixels with minimum intensity values are the dark pixels of the hyperspectral data.

`correctedData = subtractDarkPixel(inputData,darkPixels)` subtracts the specified value, `darkPixels`, from all pixels in each hyperspectral band. You can specify a single value to subtract across all bands of the data cube or a separate value for each band. After subtraction, the function sets all negative pixel values to `0`.

`correctedData = subtractDarkPixel( ___ ,'BlockSize',blocksize)` specifies the block size for block processing of the hyperspectral data cube by using the name-value pair argument `'BlockSize'`. You can specify the `'BlockSize'` name-value pair argument in addition to the input arguments in the previous syntaxes.

The function divides the input image into distinct blocks, processes each block, and then concatenates the processed output of each block to form the output matrix. Hyperspectral images are multi-dimensional data sets that can be too large to fit in system memory in their entirety. This can cause the system to run out of memory while running the `subtractDarkPixel` function. If you encounter such an issue, perform block processing by using this syntax.

For example, `subtractDarkPixel(inputData,darkPixels,'BlockSize',[50 50])` divides the input image into non-overlapping blocks of size 50-by-50 and then performs dark pixel subtraction on each block.

Note To perform block processing by specifying the `'BlockSize'` name-value pair argument, you must have MATLAB R2021a or a later release.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Subtract Band Minimum Pixel Values from Hyperspectral Data

Read hyperspectral data into the workspace.

```
hcube = hypercube('paviaU');
```

Subtract the minimum pixel value of each band from all pixels in that band.

```
hcubeCorrected = subtractDarkPixel(hcube);
```

Input Arguments

`inputData` — Input hyperspectral data

`hypercube` object | *M*-by-*N*-by-*C* numeric array

Input hyperspectral data, specified as one of the following.

- `hypercube` object. The `DataCube` property of the `hypercube` object stores the hyperspectral data cube.
- *M*-by-*N*-by-*C* numeric array — *M* and *N* are the number of rows and columns in each band of hyperspectral data. *C* is the number of spectral bands in the hyperspectral dataset.

`darkPixels` — Value to subtract from pixels of each band

numeric scalar | *C*-element numeric vector

Value to subtract from the pixels of each band, specified as a numeric scalar or a *C*-element numeric vector. *C* is the number of bands in the hyperspectral dataset. If you specify a scalar, the function subtracts that value from the pixels of all bands in the dataset.

`blocksize` — Size of data blocks

2-element vector of positive integers

Size of the data blocks, specified as a 2-element vector of positive integers. The elements of the vector correspond to the number of rows and columns in each block, respectively. The size of the data blocks must be less than the size of the input image. Dividing the hyperspectral images into smaller blocks enables you process large data sets without running out of memory.

- If the `blocksize` value is too small, the memory usage of the function reduces at the cost of increased execution time.
- If the `blocksize` value is large or equal to the input image size, the execution time reduces at the cost of increased memory usage.

Example: `'BlockSize',[20 20]` specifies the size of each data block as 20-by-20.

Output Arguments

`correctedData` — Corrected hyperspectral data

`hypercube` object | *M*-by-*N*-by-*C* numeric array

Corrected hyperspectral data, returned as a `hypercube` object or *M*-by-*N*-by-*C* numeric array with data cube dimensions equal to those of the input data `inputData`.

References

[1] Souri, A. H. and M. A. Sharifi. "Evaluation of Scene-Based Empirical Approaches for Atmospheric Correction of Hyperspectral Imagery." Paper presented at the *33rd Asian Conference on Remote Sensing*, Pattaya, Thailand, November 2012.

See Also

`hypercube` | `iarr` | `flatField` | `logResiduals` | `empiricalLine` | `reduceSmile` | `sharc`

Introduced in R2020b

spectralIndices

Compute hyperspectral indices

Syntax

```
indices = spectralIndices(hcube)
indices = spectralIndices(hcube,indexNames)
indices = spectralIndices(hcube,'all')
indices = spectralIndices( ___ ,'BlockSize',blocksize)
```

Description

`indices = spectralIndices(hcube)` computes the greenness indices: enhanced vegetation index (EVI), modified chlorophyll absorption ratio index (MCARI), and simple ratio (SR) index of a hyperspectral data. The function reads the data cube and the wavelength values stored in the hypercube object `hcube` to compute the greenness indices.

`indices = spectralIndices(hcube,indexNames)` computes one or more spectral indices specified by `indexNames`.

`indices = spectralIndices(hcube,'all')` computes all the supported spectral indices.

`indices = spectralIndices( ___ ,'BlockSize',blocksize)` specifies the block size for block processing of the hyperspectral data cube by using the name-value pair argument `'BlockSize'`. You can specify the `'BlockSize'` name-value pair argument in addition to the input arguments in the previous syntaxes.

The function divides the input image into distinct blocks, processes each block, and then concatenates the processed output of each block to form the output matrix. Hyperspectral images are multi-dimensional data sets that can be too large to fit in system memory in their entirety. This can cause the system to run out of memory while running the `spectralIndices` function. If you encounter such an issue, perform block processing by using this syntax.

For example, `spectralIndices(hcube,'BlockSize',[50 50])` divides the input image into non-overlapping blocks of size 50-by-50 and then computes the spectral indices for pixels in each block.

Note To perform block processing by specifying the `'BlockSize'` name-value pair argument, you must have MATLAB R2021a or a later release.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Compute Spectral Indices for Hyperspectral Data

Read hyperspectral data into the workspace.

```
hcube = hypercube('indian_pines.dat');
```

Compute the spectral indices value for each pixel in the data cube. By default, the `spectralIndices` function returns the simple ratio (SR) index, enhanced vegetation index (EVI), and modified chlorophyll absorption ratio index (MCARI).

```
indices = spectralIndices(hcube);
```

Inspect the index names in the output struct `indices`. Read the corresponding index images returned at the output.

```
indices.IndexName
```

```
ans =
"Simple Ratio (SR)"

ans =
"Enhanced Vegetation Index (EVI)"

ans =
"Modified Chlorophyll Absorption Ratio Index (MCARI)"
```

```
srImg = indices(1).IndexImage;
eviImg = indices(2).IndexImage;
mcariImg = indices(3).IndexImage;
```

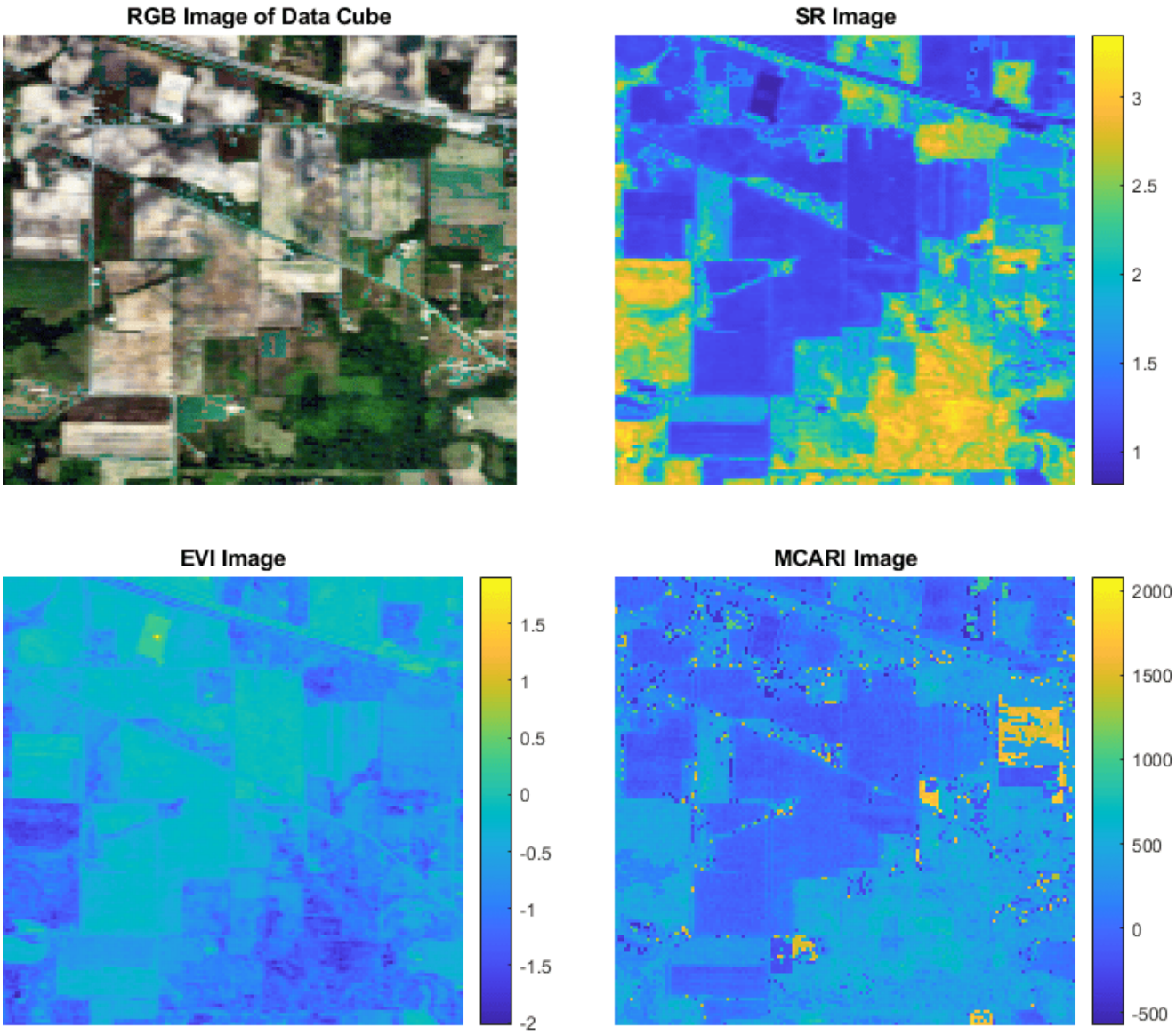
Estimate a contrast-stretched RGB image from the original data cube by using the `colorize` function.

```
rgbImg = colorize(hcube,'Method','RGB','ContrastStretching',true);
```

Display the original and the computed index images. The SR index value greater than 3 signifies vegetation. The EVI index identifies dense vegetation and the typical EVI index value for healthy vegetation lie between 0.2 and 0.8. MCARI index signifies abundance of chlorophyll in a region.

```
fig = figure('Position',[0 0 800 700]);

axes1 = axes('Parent',fig,'Position',[0 0.54 0.42 0.42]);
imagesc(rgbImg,'Parent',axes1);
axis off
title('RGB Image of Data Cube')

axes2 = axes('Parent',fig,'Position',[0.5 0.54 0.45 0.42]);
imagesc(srImg,'Parent',axes2);
axis off
title('SR Image')
colorbar

axes3 = axes('Parent',fig,'Position',[0 0.035 0.45 0.42]);
imagesc(eviImg,'Parent',axes3);
axis off
title('EVI Image')
colorbar

axes4 = axes('Parent',fig,'Position',[0.5 0.035 0.45 0.42]);
```

```
imagesc(mcariImg,'Parent',axes4);
axis off
title('MCARI Image')
colorbar
```



Detect Water Regions Using MNDWI

Read hyperspectral data into the workspace.

```
hcube = hypercube('jasperRidge2_R198.img');
```

Compute the MNDWI value for each pixel in the data cube and read the water index image.

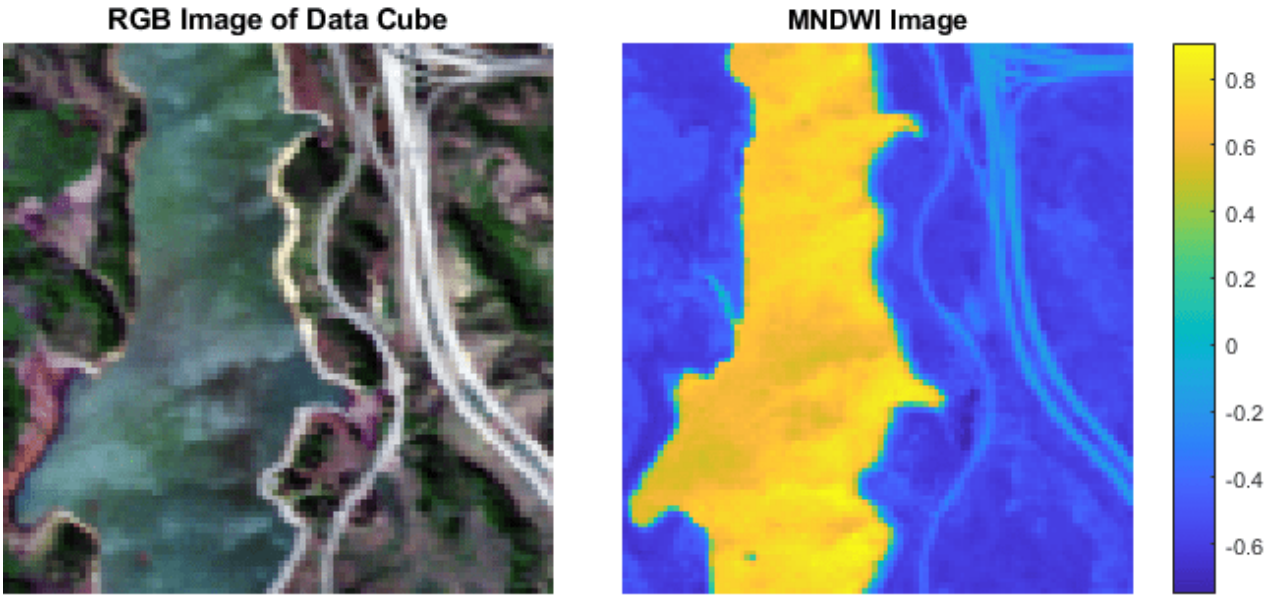
```
indices = spectralIndices(hcube,'MNDWI');
mndwiImg = indices.IndexImage;
```

Estimate a contrast-stretched RGB image from the original data cube by using the `colorize` function.

```
rgbImg = colorize(hcube,'Method','RGB','ContrastStretching',true);
```

Display the original and the MNDWI image.

```
fig = figure('Position',[0 0 700 400]);
axes1 = axes('Parent',fig,'Position',[0 0.1 0.4 0.8]);
imshow(rgbImg,'Parent',axes1)
title('RGB Image of Data Cube')
axes2 = axes('Parent',fig,'Position',[0.45 0.15 0.47 0.7]);
imagesc(mndwiImg,'Parent',axes2)
colorbar
axis off
title('MNDWI Image')
```



Water regions typically have MNDWI values greater than 0.09. Perform thresholding of MNDWI image to segment the water regions. Specify the threshold value.

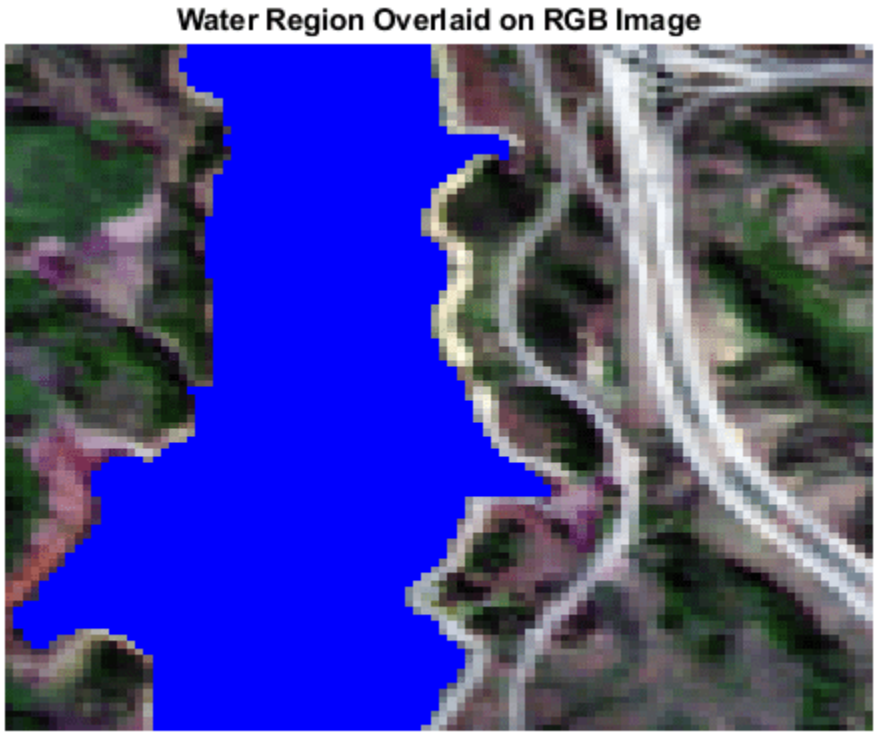
```
threshold = 0.09;
```

Generate a binary image with a intensity value 1 for pixels with a score greater than or equal to the specified threshold. All other pixels have a value 0. The regions in the binary image with a value of 1 correspond to the water regions in the data cube with MNDWI values greater than the threshold.

```
bw = mndwiImg > threshold;
```

Overlay the binary image on to the RGB image and display the overlaid image.

```
overlayImg = imoverlay(rgbImg,bw,[0 0 1]);
figure
imagesc(overlayImg)
axis off
title('Water Region Overlaid on RGB Image')
```



Input Arguments

`hcube` — Input hyperspectral data

`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `DataCube` property of the `hypercube` object contains the hyperspectral data cube.

`indexNames` — Name of spectral index

character vector | string scalar | cell array of character vectors | cell array of string scalars

Name of spectral index to compute, specified as a character vector or string scalar. You can also specify the names of multiple spectral indices as a cell array of either character vectors or string scalars. The value of the `indexNames` must be one of the names listed in this table.

Supported spectral indices

indexNames	Description
'CAI'	Cellulose absorption index
'CMR'	Clay minerals ratio
'EVI'	Enhanced vegetation index
'GVI'	Green vegetation index
'MCARI'	Modified chlorophyll absorption ratio index
'MTVI'	Modified triangular vegetation index
'MNDWI'	Modified normalized difference water index
'MSI'	Moisture stress index
'NBR'	Normalized burn ratio
'NDBI'	Normalized difference built-up index
'NDMI'	Normalized difference mud index
'NDNI'	Normalized difference nitrogen index
'NDVI'	Normalized difference vegetation index
'OSAVI'	Optimized soil adjusted vegetation index
'PRI'	Photochemical reflectance index
'SR'	Simple ratio

Example: `indexNames = 'PRI'indexNames = "PRI"indexNames = {'NDVI,'OSAVI'}indexNames = {"GVI","NDMI"}`

Data Types: `char` | `string`

`blocksize` — Size of data blocks

2-element vector of positive integers

Size of the data blocks, specified as a 2-element vector of positive integers. The elements of the vector correspond to the number of rows and columns in each block, respectively. The size of the data blocks must be less than the size of the input image. Dividing the hyperspectral images into smaller blocks enables you process large data sets without running out of memory.

- If the `blocksize` value is too small, the memory usage of the function reduces at the cost of increased execution time.
- If the `blocksize` value is large or equal to the input image size, the execution time reduces at the cost of increased memory usage.

Example: `'BlockSize',[20 20]` specifies the size of each data block as 20-by-20.

Output Arguments

`indices` — Spectral index values

struct

Spectral index values of the hyperspectral data, returned as a structure with two fields: `IndexName` and `IndexImage`.

Fields	Description
`IndexName`	Names of the spectral indices computed for the hyperspectral data, returned as a string.
`IndexImage`	Index image returned as a matrix. Each pixel value is the spectral index value computed across all the spectral bands. If the size of the hyperspectral data cube specified at the input is *M*-by-*N*-by-*C*, the size of the index image is *M*-by-*N*.

The size of the output structure depends on the number of spectral indices computed for the hyperspectral data.

- If the second input argument `indexNames` is not specified, the output is a structure array of size 1-by-3. The structure array contains the index images corresponding to EVI, MCARI, and SR index.
- If the second input argument `indexNames` is specified and is of length 1-by-*k*, the output is a structure array of size 1-by-*k*. You can use dot notation to read the outputs obtained for each spectral index specified at the input.
- If the second input argument is `'all'`, the output is a structure array of size 1-by-16. The structure array contains the index images corresponding to all the supported spectral indices.

Data Types: `struct`

See Also

`hypercube` | `ndvi`

Introduced in R2020b

spectralMatch

Identify unknown regions or materials using spectral library

Syntax

```
score = spectralMatch(libData,hcube)
score = spectralMatch(libData,reflectance,wavelength)
score = spectralMatch( ___ ,Name,Value)
```

Description

`score = spectralMatch(libData,hcube)` identifies regions in a hyperspectral data cube by matching spectra signature of each pixel to the spectral data read from the ECOSTRESS spectral library `libData`.

`score = spectralMatch(libData,reflectance,wavelength)` identifies a region or material by matching its spectral reflectance values, specified as `reflectance` and `wavelength`, with the values available in the ECOSTRESS spectral library `libData`.

`score = spectralMatch( ___ ,Name,Value)` specifies options using one or more name-value pair arguments in addition to any combination of input arguments in previous syntaxes.

Note This function requires the Image Processing Toolbox Hyperspectral Imaging Library. You can install the Image Processing Toolbox Hyperspectral Imaging Library from Add-On Explorer. For more information about installing add-ons, see Get and Manage Add-Ons.

Examples

Segment Vegetation Regions in Hyperspectral Data Using Spectral Matching

The spectral matching method compares the spectral signature of each pixel in the hyperspectral data cube with a reference spectral signature for vegetation from an ECOSTRESS spectrum file.

Read the spectral signature of vegetation from the ECOSTRESS spectral library.

```
fileroot = matlabshared.supportpkg.getSupportPackageRoot();
filename = fullfile(fileroot,'toolbox','images','supportpackages','hyperspectral','hyperdata',...
               'ECOSTRESSSpectraFiles','vegetation.tree.tsuga.canadensis.vswir.tsca-1-47.ucsb.asc
libData = readEcostressSig(filename);
```

Read the hyperspectral data into the workspace.

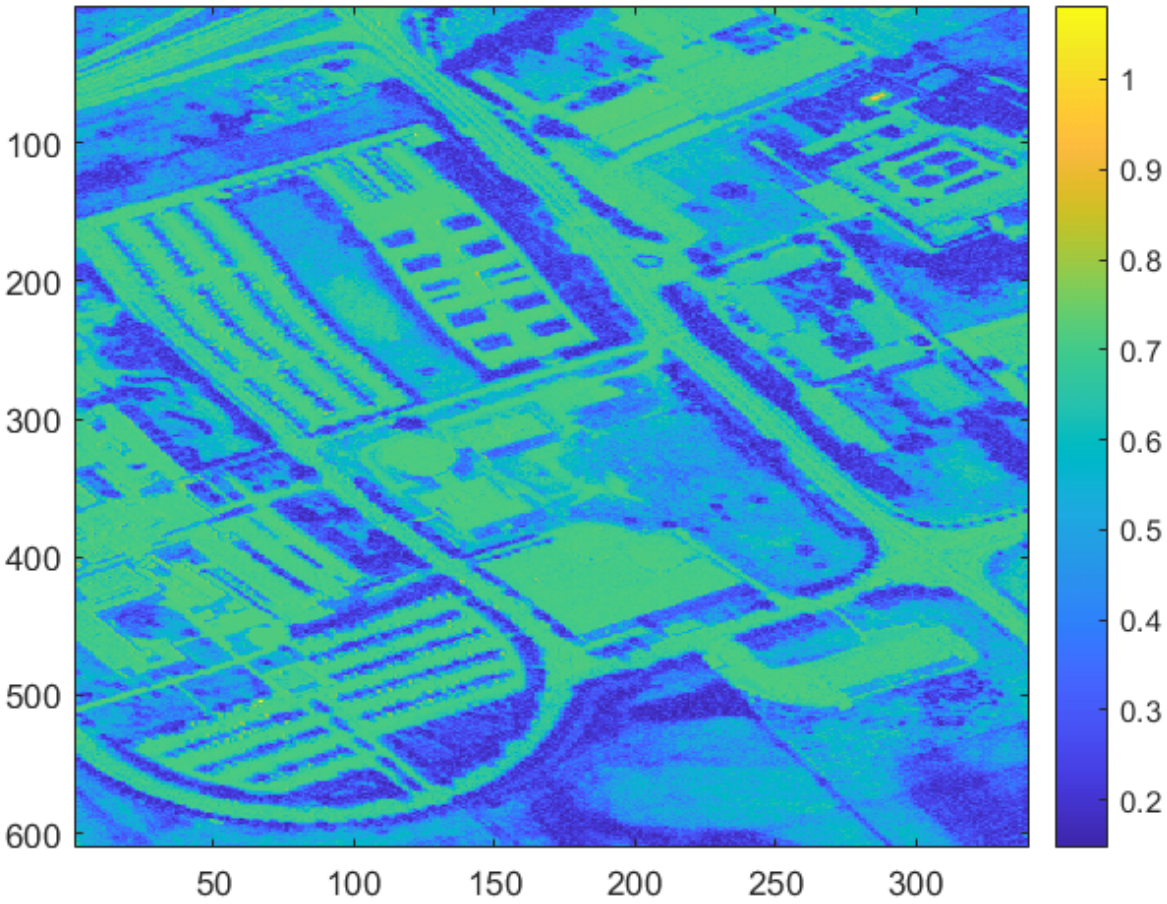
```
hcube = hypercube('paviaU.hdr');
```

Compute the distance scores of the spectrum of the hyperspectral data pixels with respect to the reference spectrum.

```
score = spectralMatch(libData,hcube);
```

Display the distance scores. The pixels with low distance scores are stronger matches to the reference spectrum and are more likely to belong to the vegetation region.

```
figure
imagesc(score)
colorbar
```



Define a threshold for detecting distance scores that correspond to the vegetation region.

```
threshold = 0.3;
```

Generate a binary image by assigning a intensity value 1 for pixels with score less than a specified threshold. Other regions are assigned the intensity value 0. The maximum intensity regions in the binary image correspond to the vegetation regions in the hyperspectral data cube.

```
bw = score < threshold;
```

Segment the vegetation regions of the hyperspectral data cube by using the indices of the maximum intensity regions in the binary image.

```
T = reshape(hcube.DataCube,[size(hcube.DataCube,1)*size(hcube.DataCube,2) size(hcube.DataCube,3)]
Ts = zeros(size(T));
```

```
Ts(bw == 1,:) = T( bw==1 ,:);
Ts = reshape(Ts,[size(hcube.DataCube,1) size(hcube.DataCube,2) size(hcube.DataCube,3)]);
```

. Create a new `hypercube` object that contains only the segmented vegetation regions.

```
segmentedDataCube = hypercube(Ts,hcube.Wavelength);
```

Estimate the RGB colour image of the original data cube and the segmented data cube by using the `colorize` function.

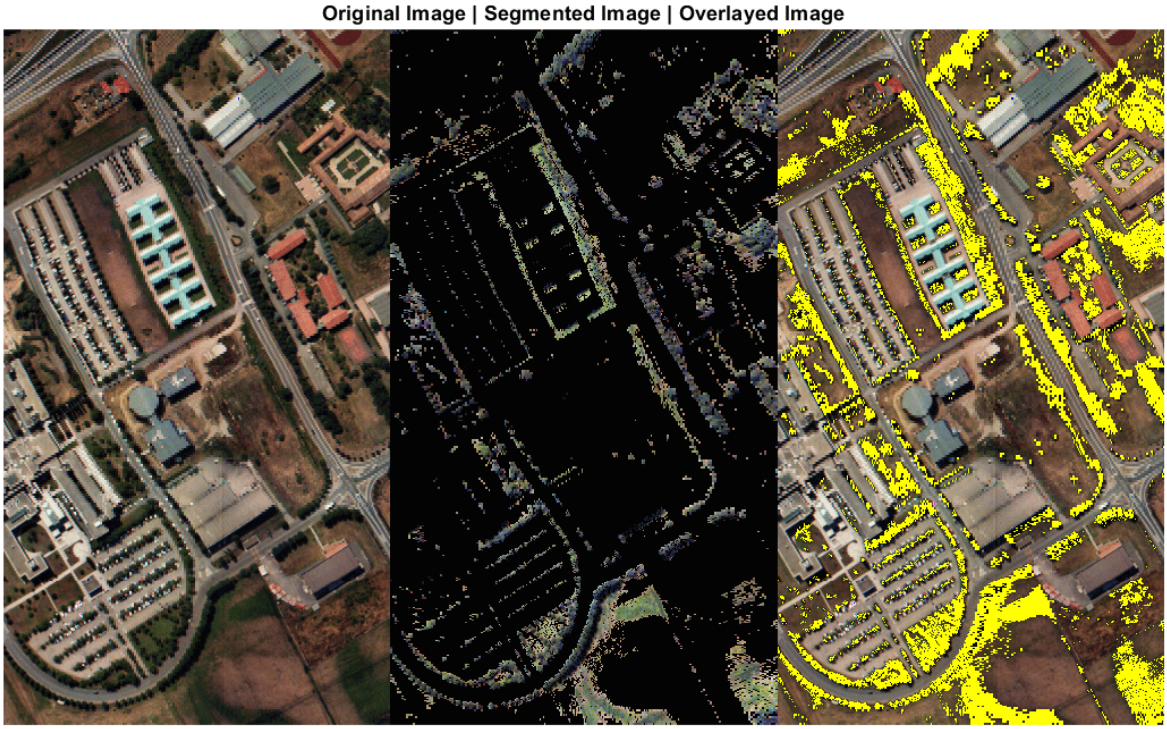
```
rgbImg = colorize(hcube,'Method','rgb','ContrastStretching',true);
segmentedImg = colorize(segmentedDataCube,'Method','rgb','ContrastStretching',true);
```

Overlay the binary image on the RGB version of the original data cube by using the `imoverlay` function.

```
B = imoverlay(rgbImg,bw,'Yellow');
```

Display the RGB colour images of the original data cube and the segmented data cube along with the overlaid image. The segmented image contains only the vegetation regions that are segmented from the original data cube.

```
figure
montage({rgbImg segmentedImg B},'Size',[1 3])
title(['Original Image | ' 'Segmented Image | ' 'Overlayed Image'])
```



Identify Unknown Spectral Signature Using Spectral Matching

Read reference spectral signatures from the ECOSTRESS spectral library. The library consists of 15 spectral signatures belonging to manmade materials, soil, water, and vegetation. The output is a structure array that stores the spectral data read from ECOSTRESS library files.

```
fileroot = matlabshared.supportpkg.getSupportPackageRoot();
dirname = fullfile(fileroot,'toolbox','images','supportpackages','hyperspectral','hyperdata','EC
libData = readEcostressSig(dirname);
```

Load a `.mat` file that contains the reflectance and the wavelength values of an unknown material into the workspace. The reflectance and the wavelength values together comprise the test spectrum.

```
load spectralData 'reflectance' 'wavelength'
```

Compute the spectral match between the reference spectrum and test spectrum using spectral information divergence (SID) method. The function computes the distance score for only those reference spectra that have bandwidth overlap with the test spectrum. The function displays a warning message for all other spectra.

```
score = spectralMatch(libData,reflectance,wavelength,'Method','SID');
```

```
Warning: Unable to find overlapping wavelengths between test spectra and library signature numbe

Warning: Unable to find overlapping wavelengths between test spectra and library signature numbe

Warning: Unable to find overlapping wavelengths between test spectra and library signature numbe
```

Display the distance scores of the test spectrum. The pixels with lower distance scores are stronger matches to the reference spectrum. A distance score value of `NaN` indicates that the corresponding reference spectrum and the test spectrum do not meet the overlap bandwidth threshold.

```
score
```

```
score = 1×15

  297.8016  122.5567  203.5864  103.3351  288.7747  275.5321  294.2341       NaN       NaN  290.4
```

Find the minimum distance score and the corresponding index. The returned index value indicates the row of the structure array `libData` that contains the reference spectrum that most closely matches a test spectrum.

```
[value,ind] = min(score);
```

Find the matching reference spectrum by using the index of the minimum distance score, and display the details of the matching spectral data in the ECOSTRESS library. The result shows that the test spectrum match most closely with the spectral signature of sea water.

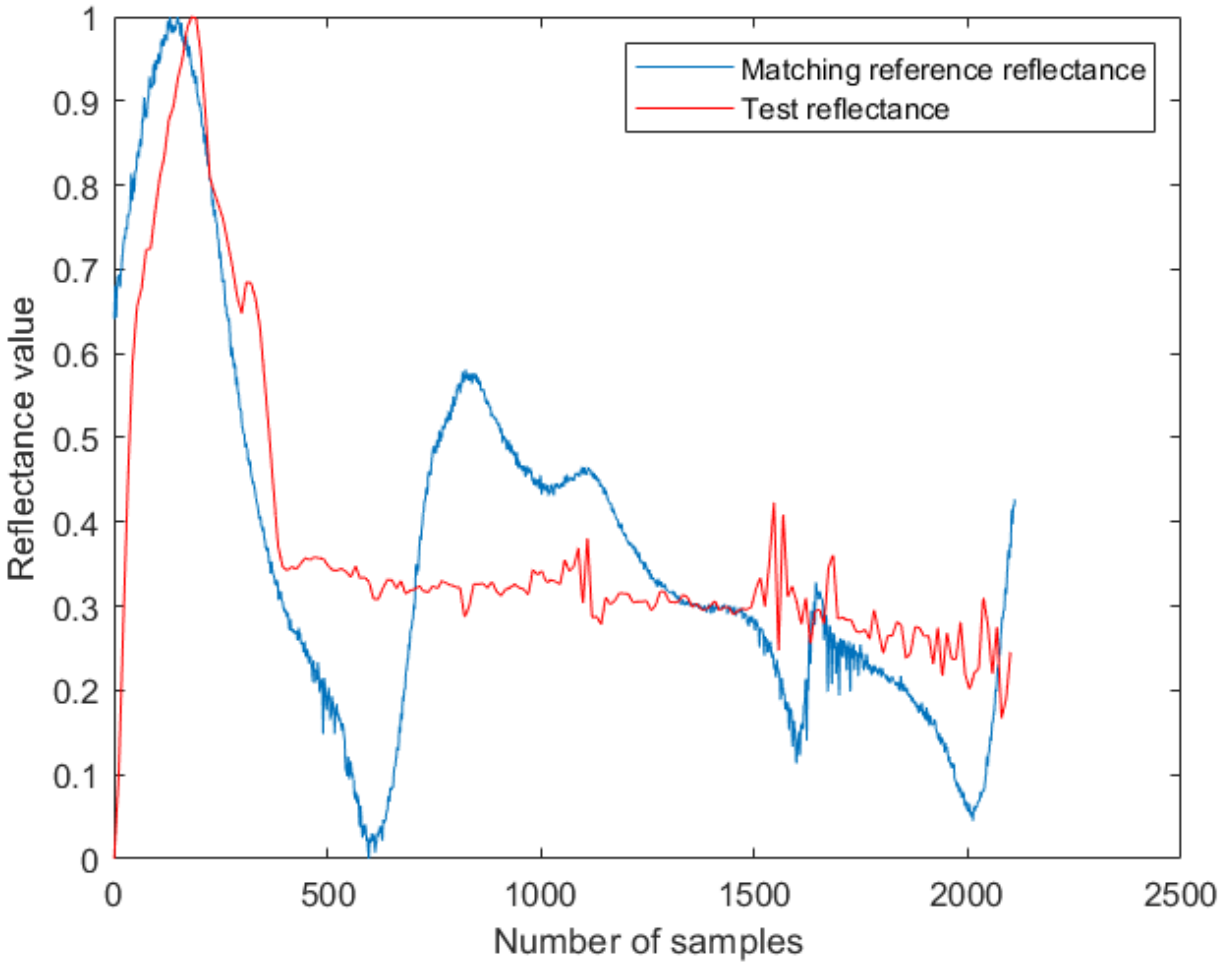
```
matchingSpectra = libData(ind)
```

```
matchingSpectra = struct with fields:
                   Name: "Sea Foam"
                   Type: "Water"
                  Class: "Sea Water"
               SubClass: "none"
           ParticleSize: "Liquid"
                  Genus: [0×0 string]
```

```
                Species: [0×0 string]
              SampleNo: "seafoam"
                 Owner: "Dept. of Earth and Planetary Science, John Hopkins University"
        WavelengthRange: "TIR"
                 Origin: "JHU IR Spectroscopy Lab."
         CollectionDate: "N/A"
            Description: "Sea foam water. Original filename FOAM Original ASTER Spectral Libra
            Measurement: "Directional (10 Degree) Hemispherical Reflectance"
            FirstColumn: "X"
           SecondColumn: "Y"
         WavelengthUnit: "micrometer"
               DataUnit: "Reflectance (percent)"
            FirstXValue: "14.0112"
             LastXValue: "2.0795"
        NumberOfXValues: "2110"
    AdditionalInformation: "none"
              Wavelength: [2110×1 double]
             Reflectance: [2110×1 double]
```

Plot the reflectance values of the test spectrum and the corresponding reference spectrum. For the purpose of plotting and visualizing the shape of the reflectance curves, rescale the reflectance values to the range [0, 1] and interpolate test reflectance values to match the reference reflectance values in number.

```
figure
testReflectance = rescale(reflectance,0,1);
refReflectance = rescale(matchingSpectra.Reflectance,0,1);
testLength = length(testReflectance);
newLength = length(testReflectance)/length(refReflectance);
testReflectance = interp1(1:testLength,testReflectance,1:newLength:testLength);

plot(refReflectance)
hold on
plot(testReflectance,'r')
hold off
legend('Matching reference reflectance','Test reflectance')
xlabel('Number of samples')
ylabel('Reflectance value')
```



Input Arguments

`libData` — ECOSTRESS Spectral data

structure

Spectral data from ECOSTRESS files, returned as a 1-by-*K* structure array. *K* is the number of spectrum files read by the function. Each element of the structure array has 24 fields that contain the header information of the spectrum files.

Field Names	Description
`Name`	Name of the measured sample or material
`Type`	Type of sample, such as `"mineral"`, `"rock"`, `"tree"`, or `"manmade"`

`Class`	Class of the sample type For example, if the sample type is `"mineral"` then the class can be: `"native elements"`,`"silicates"`, `"oxides"`, `"sulfides"`, `"sulfates"`, `"halides"`, `"carbonates"`, `"phosphates"`, or `"mineraloids"`.
`SubClass`	Subclass of the sample type This field contains an empty array or `"none"`, unless the `Type` value is `"mineral"`, `"rock"`, `"manmade"`, `"soil"`, `"lunar"`, or `"meteorite"`.
`ParticleSize`	Particle size of the sample type This field contains an empty array unless the `Type` value is `"mineral"`, `"rock"`, `"manmade"`, `"soil"`, `"lunar"`, or `"meteorite"`.
`Genus`	Genus of the sample This field contains an empty array unless the `Type` value is `"vegetation"` or `"nonphotosynthetic"`.
`Species`	Species of the sample This field contains an empty array unless the `Type` value is `"vegetation"` or `"nonphotosynthetic"`.
`SampleNo`	Sample number This value is an identifier for the associated sample.
`Owner`	Owner of the sample
`WavelengthRange`	Wavelength range of the measured sample The value must be `"All"`, `"TIR"`, or `"VSWIR"`.
`Origin`	Location from which the data was obtained
`CollectionDate`	Date on which the sample was collected This value is in `mm/dd/yy` format.
`Description`	Description of the measured sample This field provides additional information about the characteristics of the sample.
`Measurement`	Spectral measurement mode used to measure the sample
`FirstColumn`	First column of data values in the spectrum file

`SecondColumn`	Second column of data values in the spectrum file
`WavelengthUnit`	Measuring unit for the spectral wavelengths of the samples The value for every sample type is `"micrometer"`. This field corresponds to the `X Units` field of the header data in the ECOSTRESS spectrum file.
`DataUnit`	Unit of the spectral measurement mode Spectral measurement mode includes reflectance, transmittance, and transmission. The unit is percentage. This field corresponds to the `Y Units` field of the header data in the ECOSTRESS spectrum file.
`FirstXValue`	First value in the first column of data values in the spectrum file
`LastXValue`	Last value in the first column of data values in the spectrum file
`NumberofXValues`	Total number of data values in the first column of the spectrum file
`AdditionalInformation`	Additional information about the sample This field includes information that is not part of the spectral data.
`Wavelength`	Wavelength values at which the reflectances were measured
`Reflectance`	Reflectance values measured at each wavelengths

`hcube` — Input hyperspectral data
`hypercube` object

Input hyperspectral data, specified as a `hypercube` object. The `DataCube` property of the `hypercube` object contains the hyperspectral datacube.

`reflectance` — Reflectance values
C-element vector

Reflectance values, specified as a *C*-element vector. *C* is the number of wavelengths for which the reflectance values have been measured.

`wavelength` — Wavelength values
C-element vector

Wavelength values, specified as a *C*-element vector. *C* is the number of wavelengths for which the reflectance values have been measured.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `spectralMatch(libData,hcube,'MinBandWidth',0.5)`

`Method` — Spectral matching method

`'sam'` (default) | `'sid'` | `'sidsam'` | `'jmsam'` | `'ns3'`

Spectral matching method, specified as the comma-separated pair consisting of `'Method'` and one of these values:

- `'sam'` — Spectral angle mapper (SAM) method, which measures the similarity between two spectra by computing the angular distance between them.
- `'sid'` — Spectral information divergence (SID) method, which measures the similarity between two spectra by computing the difference between their probability distribution values.
- `'sidsam'` — Mixed spectral similarity method, which measures the similarity between two spectra by combining the SID and SAM distance measures.
- `'jmsam'` — Jeffries Matusita-Spectral Angle Mapper (JMSAM), which measures the similarity between two spectra by combining the Jeffries Matusita (JM) and SAM distance measures.
- `'ns3'` — Normalized spectral similarity score (NS3) method, which measures the similarity between two spectra by combining the Euclidean and SAM distance measures.

For details about these spectral matching methods, see “More About” on page 1-3411.

Data Types: `char` | `string`

`MinBandWidth` — Minimum overlap bandwidth

`300` (default) | positive scalar

Minimum overlap bandwidth, specified as the comma-separated pair consisting of `'MinBandWidth'` and a positive scalar in nanometers. The overlap bandwidth between the reference spectrum and the test spectra is defined as:

$$BW_{\text{overlap}} = W_{\max} - W_{\min}$$

$W_{\min}$ is the maximum of minimum wavelengths in the reference and test spectra.

$W_{\max}$ is the maximum of maximum wavelengths in the reference and test spectra.

The `'MinBandWidth'` argument defines the minimum expected value for the overlap bandwidth between the spectral values of the test material and the ECOSTRESS spectral data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`score` — Distance scores

3-D numeric array | matrix | *K*-element column vector | scalar

Distance scores, returned as a 3-D numeric array, matrix, *K*-element column vector, or scalar. The dimensions of the output score depend on the dimensions of the `libData` and whether the test data is a `hypercube` object or a `wavelength` and `reflectance` pair.

If the test spectral signatures are specified as a `hypercube` object, `hcube` and the data cube is of size *M*-by-*N*-by-*C*:

Dimension of input argument, `libData`	**Dimension of output**, `score`
1-by-*K*, containing *K* reference signatures read from *K* number of spectrum files	3-D numeric array of size *M*-by-*N*-by-*K* containing the distance score for each pixel with respect to *K* reference signatures The values in each channel of *K* are the distance scores of the spectra of each pixel with respect to the spectral data in the corresponding row of `libData`. Similarly, the values in the second channel relate to the spectral data in the second row of `libData`.
1-by-1, containing reference signature read from one spectrum file (*K* = 1)	matrix of size *M*-by-*N*, The matrix contains the distance score for each pixel's spectra with respect to a reference signature.

If the test spectral signature is specified as `reflectance` and `wavelength` values:

Dimension of input argument, `libData`	**Dimension of output**, `score`
1-by-*K*, containing *K* reference signatures read from *K* number of spectrum files	*K*-element vector containing the distance score of the test spectra with respect to *K* reference signatures. Each element of the vector is the distance score of the test reflectance values with respect to the spectral data in the corresponding row of `libData`.
1-by-1, containing reference signature read from one spectrum file (*K* = 1)	scalar

Data Types: `double`

More About

Spectral Angle Mapper (SAM)

Given the test spectra t and a reference spectra r of length C, the SAM score α is calculated as

Spectral Information Divergence (SID)

The spectral information divergence (SID) method computes spectral similarity based on the divergence between the probability distributions of the two spectra. Let r and t be the reference and test spectra respectively. Calculate the distribution values for the reference spectra as:

.

Calculate the distribution values for the test spectra as:

.

Then, compute the SID value by using the probability distributions of the reference and the test spectra:

SID-SAM

The SID-SAM method computes spectral similarity as:

Jeffries Matusita-Spectral Angle Mapper (JMSAM)

The JMSAM method computes spectral similarity based on the Jeffries Matusita (JM) and SAM distances between two spectra. Let r and t be the reference and test spectra respectively.

First, compute the JM distance,

where B is the Bhattacharyya distance,

$$B = \frac{1}{8}(\mu_t - \mu_r)^T\left[\frac{\sigma_t + \sigma_r}{2}\right]^{-1}(\mu_t - \mu_r) + \frac{1}{2}\ln\left[\frac{\left|\frac{\sigma_t + \sigma_r}{2}\right|}{\sqrt{|\sigma_t||\sigma_r|}}\right]$$

μ_r and μ_t are the mean values of the reference and test spectra respectively. σ_r and σ_t are the covariance values of the reference and test spectra respectively.

Then, compute the SAM value α by using the test spectra t and the reference spectra r of length C,

Finally, compute the JMSAM score as:

$$JMSAM = JM_{distance} \times \tan(\alpha)$$

Normalized Spectral Similarity Score (NS3)

The NS3 method computes spectral similarity based on the Euclidean and SAM distances between two spectra. Let r and t be the reference and test spectra respectively. Compute the Euclidean distance between two spectra as:

Then, compute the SAM value α

Finally, compute the NS3 score as:

See Also

`sam` | `sid` | `hypercube` | `readEcostressSig` | `ns3` | `sidsam` | `jmsam`

Introduced in R2020a