

# Medical Imaging Toolbox™

Reference



# MATLAB®

R2023a



# How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

## *Medical Imaging Toolbox™ Reference*

© COPYRIGHT 2022–2023 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

### **Trademarks**

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

### **Patents**

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

### **Revision History**

September 2022	Online only	New for Version 1.0 (Release 2022b)
March 2023	Online Only	Revised for Version 1.1 (Release 2023a)

<b>1</b>	<b>Functions</b>
----------	------------------



# Functions

---

# Medical Image Labeler

Interactively explore, label, and publish animations of 2-D or 3-D medical image data

## Description

The **Medical Image Labeler** app enables you to label ground truth data in medical images. Using the app, you can:

- Import multiple 2-D images or 3-D image volumes.
- Explore images as slice planes or volumes with anatomical orientation markers and scale bars. Navigate volume slices using crosshairs.
- Publish 2-D and 3-D PNG or PDF image snapshots and GIF animations.
- Create multiple pixel label definitions to label regions of interest. Label pixels using automatic algorithms such as flood fill, semi-automatic techniques such as interpolation, and manual techniques such as painting by superpixels.
- Write, import, and use your own custom automation algorithm to automatically label ground truth data.
- Export the labeled ground truth data as a `groundTruthMedical` object. You can use this object to share labels with colleagues or for training semantic segmentation deep learning networks.

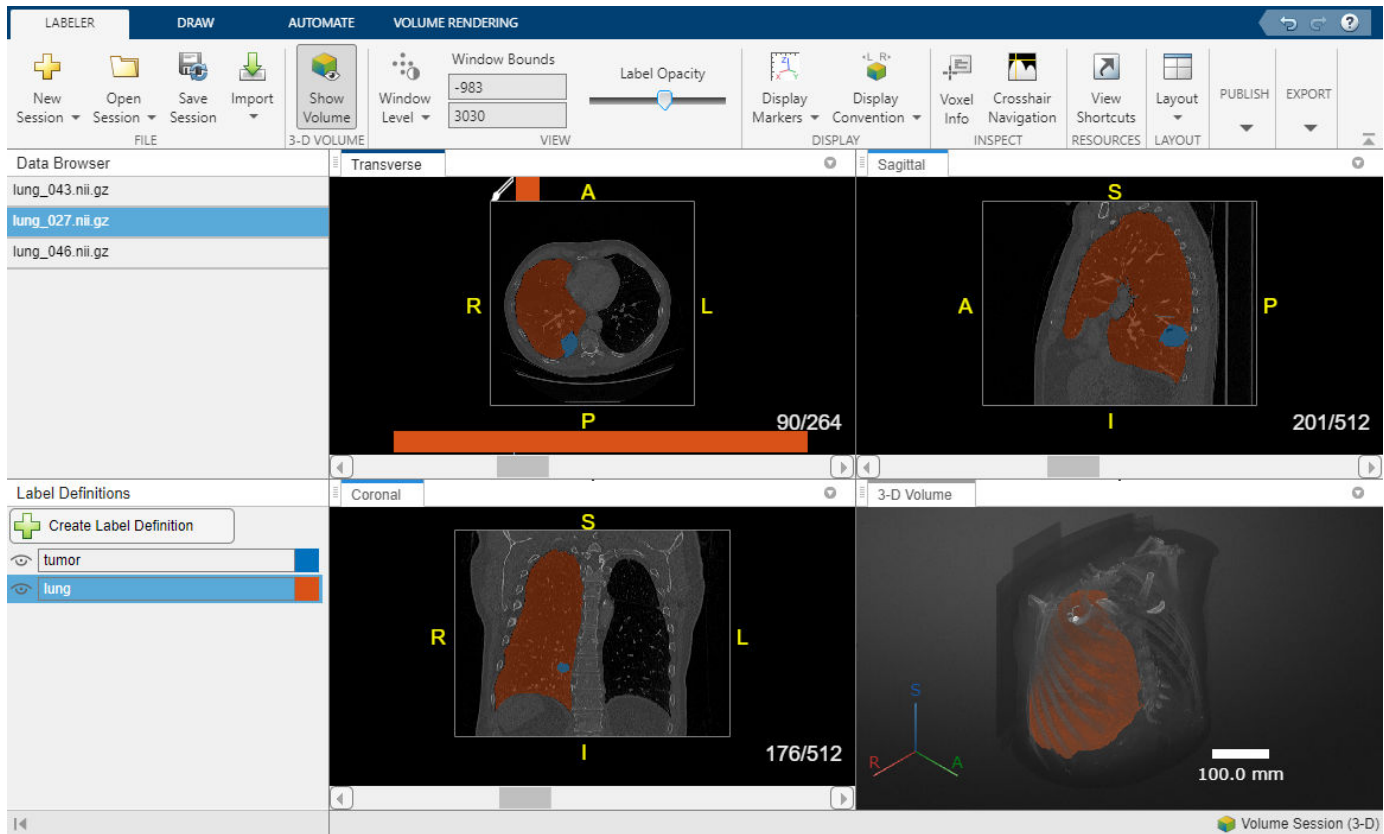
The **Medical Image Labeler** app supports 2-D images and image sequences stored in the DICOM and NIfTI file formats. An image sequence is a series of images related by time, such as ultrasound data. The app supports 3-D image volume data stored in the DICOM (single or multiframe volume), NIfTI, and NRRD file formats.

To learn more about this app, see “Get Started with Medical Image Labeler”.

---

**Note** The app is not supported in MATLAB® Online™. For details, see Specifications and Limitations.

---



## Open the Medical Image Labeler App

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

## Examples

- “Get Started with Medical Image Labeler”
- “Visualize 3-D Medical Image Data Using Medical Image Labeler”
- “Label 2-D Ultrasound Series Using Medical Image Labeler”
- “Label 3-D Medical Image Using Medical Image Labeler”
- “Automate Labeling in Medical Image Labeler”
- “Collaborate on Multi-Labeler Medical Image Labeling Projects”

## Programmatic Use

`medicalImageLabeler` opens the **Medical Image Labeler** app, which enables you to start a new session to label 2-D or 3-D medical image data.

`medicalImageLabeler(gTruthFile)` opens the app and loads the image and label data stored in the file `gTruthFile` into the app. `gTruthFile` is the full path to a MAT file containing a `groundTruthMedical` object, specified as a string scalar or character vector.

`medicalImageLabeler(gTruth)` opens the app and loads the image and label data stored in the `groundTruthMedical` object `gTruth` from the workspace into the app.

`medicalImageLabeler(sessionFolder)` opens the app and loads a saved labeling session into the app, where `sessionFolder` is the full path to a session folder created using the **Medical Image Labeler** app.

`medicalImageLabeler(sessionType)` opens the app and creates a new session of the specified type, where `sessionType` is "Volume" or "Image". For more information about session types, see "Get Started with Medical Image Labeler".

## Version History


### Introduced in R2022b

#### R2023a: Generate animations and display slice crosshairs

The app includes these new capabilities:

- Animation Generator — Configure, preview, and export animations from the app. In a volume session, generate animations that loop through slices in the coronal, sagittal, or transverse plane, or show a rotating 3-D volume. In an image session, generate animations that loop through the frames of an image series, such as an ultrasound video.

Optionally include labeled regions in animations, and customize parameters such as step size, loop count, and total animation length in seconds. To open the **Animation** pane, in the app toolbar, click **Generate Animations**.

- Crosshair Navigation — In a volume session, view and navigate 2-D slice positions using crosshairs. To turn on the crosshair indicators, in the app toolbar, select **Crosshair Navigation**. The indicators show the relative positions of the slices in the other 2-D views. To navigate slices, pause on a crosshair until the cursor changes to the fleur shape, , and then click and drag to a new position. The other slice views update automatically. To hide the crosshairs, in the app toolbar, clear **Crosshair Navigation**.

## See Also

`groundTruthMedical` | **Image Labeler**

### Topics

"Get Started with Medical Image Labeler"

"Visualize 3-D Medical Image Data Using Medical Image Labeler"

"Label 2-D Ultrasound Series Using Medical Image Labeler"

"Label 3-D Medical Image Using Medical Image Labeler"

"Automate Labeling in Medical Image Labeler"

"Collaborate on Multi-Labeler Medical Image Labeling Projects"



# extractIsosurface

Extract isosurface from volume using marching cubes algorithm

## Syntax

```
[faces,verts] = extractIsosurface(V,isovalue)
```

## Description

An isosurface is a 3-D surface representation of points with equal values in a 3-D intensity volume. The `extractIsosurface` function returns the face and vertex data of the isosurface extracted by connecting points of a constant value within a volume of space. The `extractIsosurface` function uses the marching cubes algorithm to extract isosurface data as arrays faster than the corresponding syntax of the `isosurface` function, without compromising resolution. For additional options, you must use the `isosurface` function.

`[faces,verts] = extractIsosurface(V,isovalue)` extracts an isosurface from the intensity volume `V` by determining where the values of `V` are equal to the specified `isovalue`. The function returns the face and vertex data of the isosurface in `faces` and `verts`, respectively.

## Examples

### Plot Isosurface

Load the intensity volume data into the workspace.

```
load(fullfile(toolboxdir("images"),"imdata","BrainMRILabeled","images","vol_001.mat"));
V = vol;
```

Specify the `isovalue` for isosurface extraction.

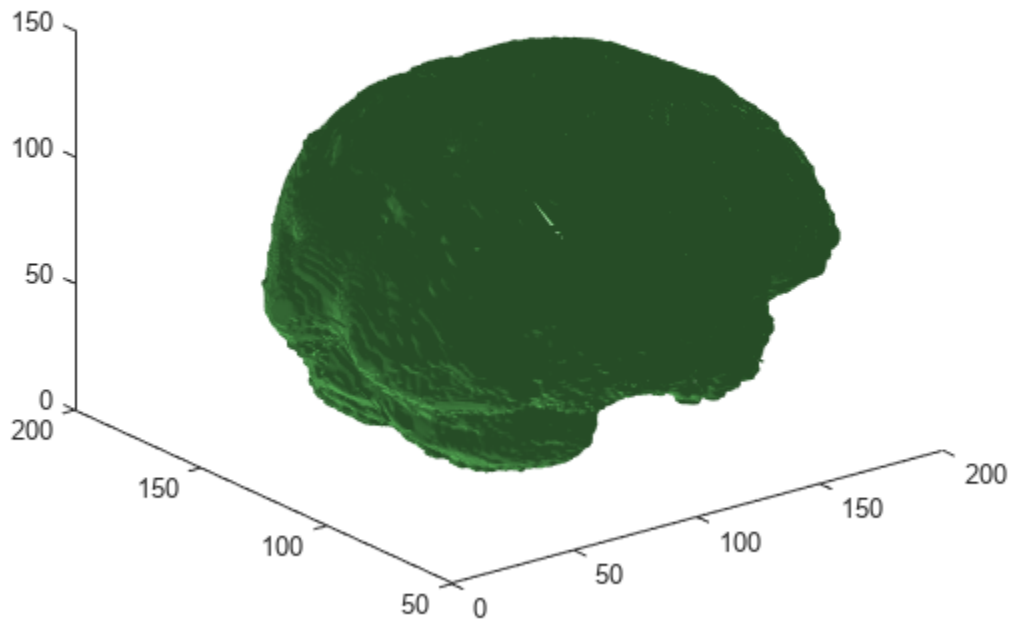
```
isovalue = 100;
```

Extract the isosurface of the input volume at the specified `isovalue`.

```
[faces,verts] = extractIsosurface(V,isovalue);
```

Plot the extracted isosurface.

```
figure
p = patch(Faces=faces,Vertices=verts);
isonormals(V,p)
view(3)
set(p,FaceColor=[0.5 1 0.5])
set(p,EdgeColor="none")
camlight
lighting gouraud
```



### Create Point Cloud from Isosurface

Load the intensity volume data into the workspace.

```
load(fullfile(toolboxdir("images"),"imdata","BrainMRILabeled","labels","label_001.mat"));  
V = label;
```

Specify the isovalue for isosurface extraction.

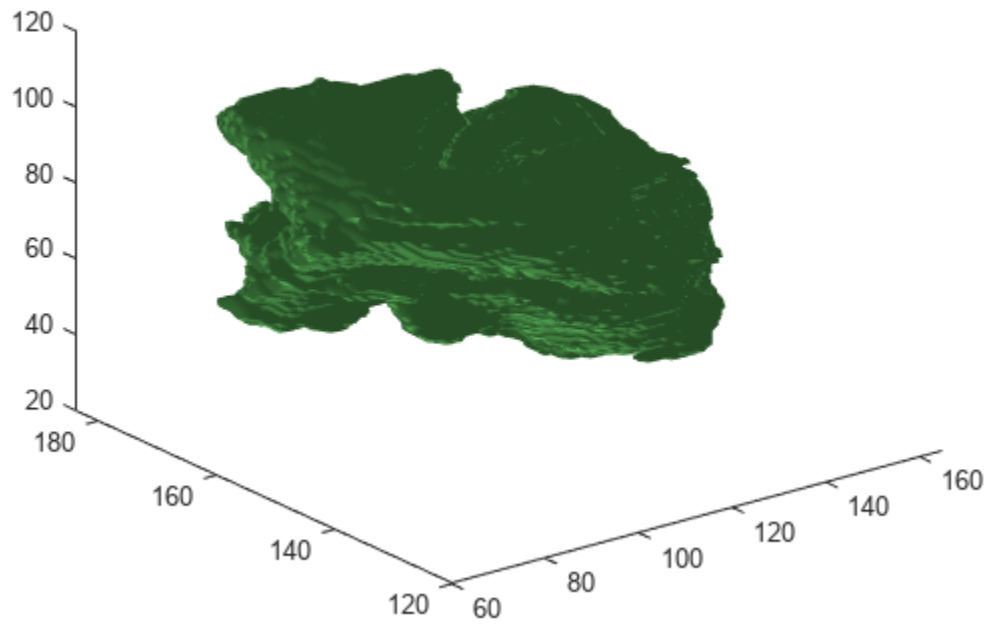
```
isovalue = 0.05;
```

Extract the isosurface of the input volume at the specified isovalue.

```
[faces,verts] = extractIsosurface(V,isovalue);
```

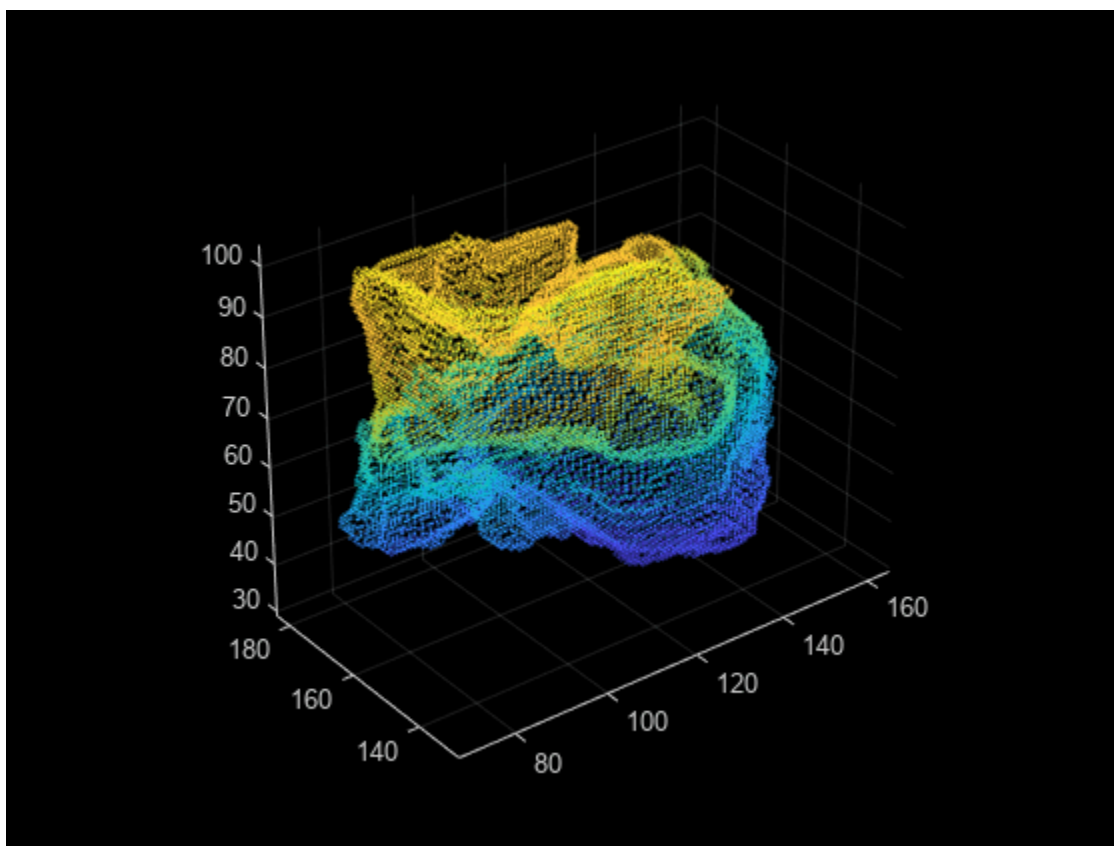
Display and inspect the extracted isosurface.

```
figure  
p = patch(Faces=faces,Vertices=verts);  
isonormals(V,p)  
view(3)  
set(p,FaceColor=[0.5 1 0.5])  
set(p,EdgeColor="none")  
camlight  
lighting gouraud
```



Create a point cloud from the vertices of the extracted isosurface. Display the point cloud.

```
ptCloud = pointCloud(verts);  
figure  
pcshow(ptCloud)
```



### Create STL File for 3-D Printing from Isosurface

Load the intensity volume data into the workspace.

```
load(fullfile(toolboxdir("images"),"imdata","BrainMRILabeled","labels","label_002.mat"));  
V = label;
```

Specify the isovalue for isosurface extraction.

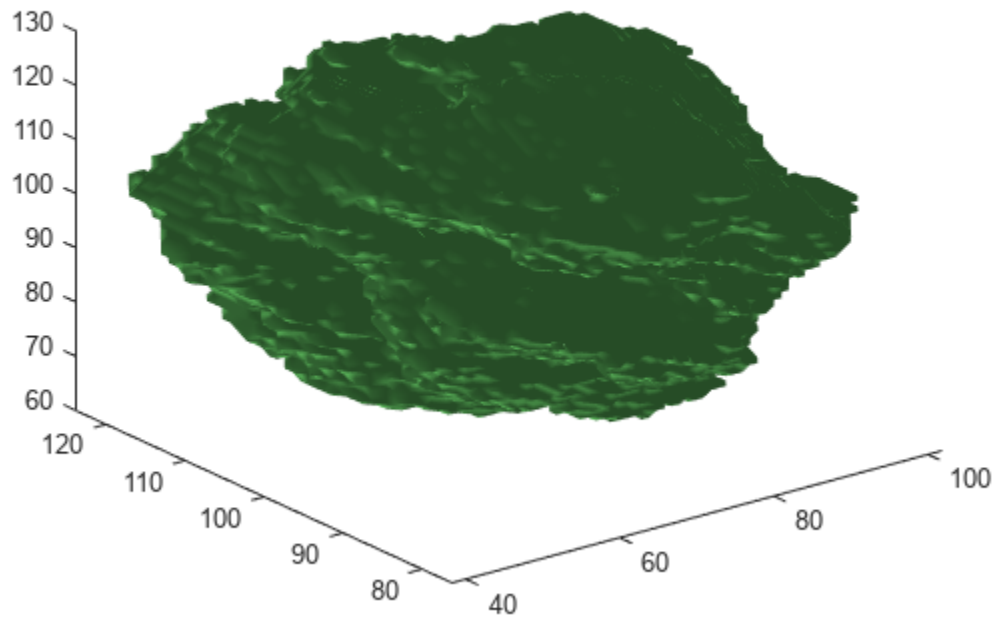
```
isovalue = 0.05;
```

Extract the isosurface of the input volume at the specified isovalue.

```
[faces,verts] = extractIsosurface(V,isovalue);
```

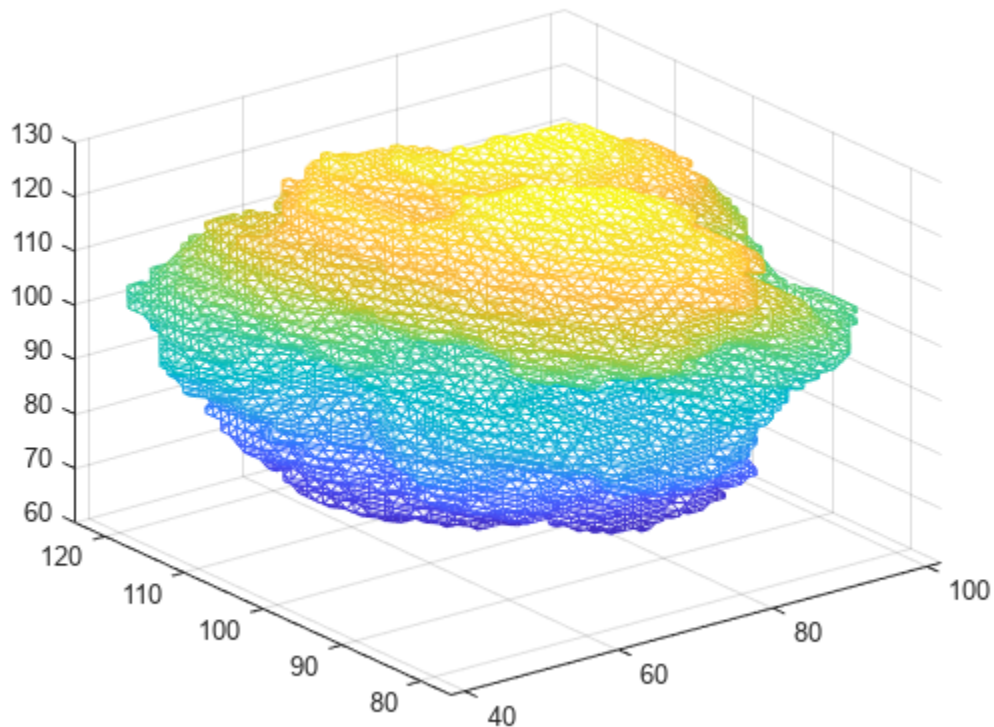
Display and inspect the extracted isosurface.

```
figure  
p = patch(Faces=faces,Vertices=verts);  
isonormals(V,p)  
view(3)  
set(p,FaceColor=[0.5 1 0.5])  
set(p,EdgeColor="none")  
camlight  
lighting gouraud
```



Triangulate the extracted isosurface. Display the triangulation as a mesh.

```
T = triangulation(double(faces),double(verts));  
figure  
trimesh(T)
```



Create an STL file for 3-D printing, using the triangulation of the extracted isosurface.

```
stlwrite(T, "brain.stl")
```

## Input Arguments

### **V** — Intensity volume data

3-D numeric array | 3-D logical array

Intensity volume data, specified as a 3-D numeric or logical array.

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

### **isoValue** — Isovalue

numeric scalar

Isovalue at which to compute the isosurface, specified as a numeric scalar.

Data Types: `single` | `double`

## Output Arguments

### **faces** — Face data of computed isosurface

*M*-by-3 matrix

Face data of the computed isosurface, returned as an  $M$ -by-3 matrix.  $M$  is the number of faces in the isosurface. Three vertices form a triangular face. The 3-D index coordinates of all the vertices of the surface are returned as the rows of `verts`. Each row of the face matrix contains the row indices of the three vertices from `verts` that form the triangular face.

Data Types: `single`

### **verts — Vertex data of computed isosurface**

$N$ -by-3 matrix

Vertex data of the computed isosurface, returned as an  $N$ -by-3 matrix.  $N$  is the number of vertices in the isosurface. Each row of the matrix contains the 3-D index coordinates of one vertex of the isosurface.

Data Types: `single`

## **Algorithms**

The `extractIsosurface` function uses the marching cubes algorithm to extract the isosurface of a volume  $V$  at the specified isovalue `isoValue`. The marching cubes algorithm uses lookup tables to obtain information about the faces and vertices of the isosurface. The lookup tables enable the `extractIsosurface` function to extract face and vertex data as arrays faster than the `isosurface` function without compromising resolution for large intensity volumes, such as those typically used in medical imaging. Though the purposes of `extractIsosurface` and `isosurface` are similar, there are certain differences in their implementation and output.

- The behavior of `extractIsosurface` and `isosurface` differs for the edge case when an intensity value is equal to the specified isovalue. The `isosurface` function generates a surface between regions with intensities less than or equal to the isovalue and regions with intensities greater than the isovalue. The `extractIsosurface` function generates a surface between regions with intensities less than the isovalue and regions with intensities greater than or equal to the isovalue.
- Except for the edge case, both `extractIsosurface` and `isosurface` generate the same number of vertices with the index coordinates of the vertices matching within a small tolerance. However, the order of the vertices in the output `verts` can be different.
- Except for the edge case, both `extractIsosurface` and `isosurface` generate the same number of faces. However, the actual faces in the output `faces` are different because both algorithms create the same surface using different triangulations of the vertices.

## **Version History**

Introduced in R2022b

## **References**

- [1] Lorensen, William E., and Harvey E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm." *ACM SIGGRAPH Computer Graphics* 21, no. 4 (August 1987): 163-69. <https://doi.org/10.1145/37402.37422>.

## **See Also**

`isosurface`

**Topics**

“Connecting Equal Values with Isosurfaces”



# imregdeform

Deformable registration of grayscale images or intensity volumes using total variation method

## Syntax

```
[dispField,reg] = imregdeform(moving, fixed)
[dispField,reg] = imregdeform(moving, fixed, Name=Value)
```

## Description

The `imregdeform` function uses the total variation method to perform deformable registration of grayscale images or intensity volumes. You can use this function to register medical images or volumes deformed due to local transformations.

`[dispField,reg] = imregdeform(moving, fixed)` transforms the grayscale image or intensity volume `moving`, so that it is registered with the reference image or volume `fixed`, and returns the displacement field `dispField` and the registered image or volume `reg`.

`[dispField,reg] = imregdeform(moving, fixed, Name=Value)` specifies options for the total variation method using one or more optional name-value arguments.

## Examples

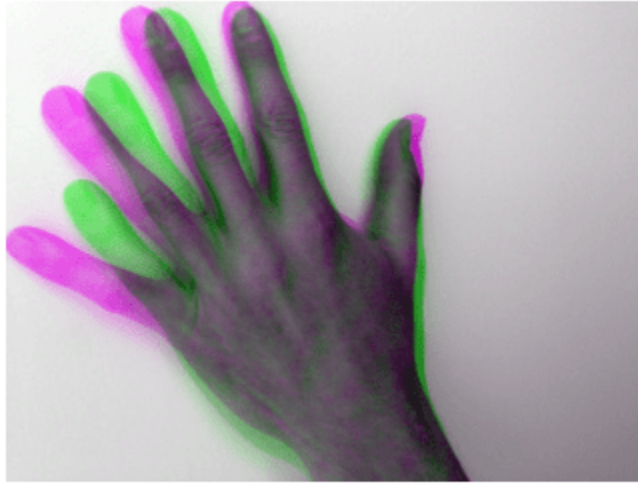
### Deformable Registration of Images

Load a reference image and an image to be registered into the workspace. Convert the images to grayscale.

```
fixedImg = imread("hands1.jpg");
fixed = im2gray(fixedImg);
movingImg = imread("hands2.jpg");
moving = im2gray(movingImg);
```

Display the fixed image and the moving image. Observe the deformation in the alignment of the images.

```
figure
imshowpair(fixed,moving)
```



Register the moving image to the fixed image.

```
[dispField,reg] = imregdeform(moving, fixed, NumPyramidLevels=6, GridRegularization=0.6);
```

```
----- Pyramid Level = 6 -----
```

Iteration	Normalized rmse	Function local minima	Step-size	Closeness to optimal solution
1	1.64147	37024.85	4.19	49.62783
2	1.48633	31590.32	24.34	48.04233
3	1.63155	30339.11	18.00	46.47121
4	1.33748	29085.22	7.83	60.12520
5	1.29691	28578.16	10.20	74.34988
6	1.35689	28002.80	14.17	78.68985
7	1.36835	27701.23	7.44	65.67400
8	1.17573	27384.96	8.51	35.42655
9	1.08997	26944.82	13.44	53.24290
10	0.95153	26374.55	5.58	29.71761
11	0.91025	26169.77	2.55	26.32110
12	0.91195	25891.90	5.95	34.23311
13	1.14507	25726.65	14.50	41.34727
14	1.22759	25281.23	11.61	33.04872
15	0.97035	24703.21	7.40	34.35227
16	1.03194	24307.52	9.90	34.04009
17	1.03114	24077.92	7.69	41.98692
18	0.98282	23736.04	14.83	46.11518
19	0.95928	23317.39	14.87	37.88543
20	0.92435	22891.46	17.10	45.98425
21	0.81884	22398.19	11.15	37.09230
22	0.70854	22097.88	6.20	23.16606
23	0.65405	21979.78	3.92	20.64745
24	0.58632	21890.52	3.22	17.18440
25	0.59774	21767.81	7.38	17.62879
26	0.63929	21698.98	8.36	22.69476
27	0.50519	21574.89	2.62	15.70961

28	0.50145	21491.14	3.01	12.74984
29	0.50583	21419.94	3.16	14.86162
30	0.48495	21390.37	4.85	13.89448
31	0.46792	21363.09	1.43	13.67136
32	0.40962	21356.92	1.65	9.81103
33	0.40800	21356.83	0.18	9.50099

----- Pyramid Level = 5 -----

Iteration	Normalized rmse	Function local minima	Step-size	Closeness to optimal solution
1	0.44400	23959.26	5.30	22.96675
2	0.39054	23254.00	9.46	18.54226
3	0.36834	23134.24	5.02	22.51686
4	0.33450	23051.56	1.59	12.51227
5	0.33048	23004.59	2.58	10.00633
6	0.32744	22960.51	3.02	8.53760
7	0.35590	22882.32	6.67	25.78811
8	0.32620	22788.98	1.99	10.72627
9	0.31109	22735.69	1.69	7.66283
10	0.30765	22695.63	2.81	6.17570
11	0.30926	22686.24	1.23	7.16721
12	0.31211	22682.41	0.80	6.14514
13	0.31531	22680.85	0.64	4.67112
14	0.31613	22680.72	0.19	4.48773

----- Pyramid Level = 4 -----

Iteration	Normalized rmse	Function local minima	Step-size	Closeness to optimal solution
1	0.28145	28405.52	4.99	21.00908
2	0.26476	27597.03	9.61	12.38693
3	0.24598	27383.91	4.02	15.36024
4	0.23363	27324.84	1.62	7.73944
5	0.23838	27210.31	2.64	5.38586
6	0.23927	27069.79	3.11	8.21936
7	0.24557	26887.38	5.87	15.04406
8	0.23496	26800.32	1.50	8.34007
9	0.23145	26768.71	1.57	4.85061
10	0.23168	26719.01	2.55	6.54151
11	0.23681	26642.15	5.61	15.08834
12	0.23464	26630.58	0.24	13.59409
13	0.23266	26610.13	0.46	10.57861
14	0.23255	26601.49	0.22	9.64829

----- Pyramid Level = 3 -----

Iteration	Normalized rmse	Function local minima	Step-size	Closeness to optimal solution
1	0.16744	30346.55	7.61	9.96176
2	0.16423	30102.90	4.35	8.32035
3	0.16156	29980.10	2.71	5.30050
4	0.16202	29875.25	3.51	5.31045
5	0.16292	29776.72	4.77	7.43417
6	0.16392	29666.28	4.26	6.45342
7	0.16464	29634.29	3.28	6.47225
8	0.16468	29602.16	3.41	9.24666
9	0.16277	29597.64	2.84	4.73214
10	0.16176	29572.86	3.45	4.35966
11	0.16176	29572.86	0.00	4.35966

```
----- Pyramid Level = 2 -----
```

Iteration	Normalized rmse	Function local minima	Step-size	Closeness to optimal solution
1	0.12139	29164.12	9.06	7.14654
2	0.11765	28862.99	5.23	7.51096
3	0.11638	28735.95	3.36	5.13431
4	0.11604	28645.66	3.50	6.77245
5	0.11627	28571.44	4.39	7.44123
6	0.11643	28484.52	3.61	6.90509
7	0.11650	28430.70	2.00	4.35053
8	0.11647	28378.27	2.95	6.94753
9	0.11613	28331.36	3.26	7.69636
10	0.11584	28303.83	2.11	6.46119
11	0.11552	28262.96	2.64	4.12788
12	0.11530	28249.84	2.80	11.04358
13	0.11536	28165.72	3.59	7.07860
14	0.11575	28115.74	1.89	4.88700
15	0.11620	28076.12	2.02	7.58377
16	0.11622	28074.33	0.16	7.56145

```
----- Pyramid Level = 1 -----
```

Iteration	Normalized rmse	Function local minima	Step-size	Closeness to optimal solution
1	0.08227	31990.04	11.46	9.08693
2	0.08242	31703.92	3.95	6.42661
3	0.08259	31673.28	2.83	5.67247
4	0.08258	31592.70	1.62	2.86767
5	0.08248	31472.07	3.63	3.94959
6	0.08236	31394.89	4.08	8.77327
7	0.08236	31318.72	3.15	6.37404
8	0.08241	31284.27	2.45	7.80319
9	0.08246	31261.99	2.13	5.12846
10	0.08250	31229.43	2.73	9.01950
11	0.08251	31184.45	2.76	3.64364
12	0.08250	31134.16	2.36	4.62543
13	0.08248	31093.56	2.19	4.72333
14	0.08242	31058.02	3.06	4.89104
15	0.08241	31054.33	0.65	4.79401
16	0.08241	31043.47	4.64	4.55497
17	0.08246	30993.96	1.31	4.77195
18	0.08250	30974.60	1.80	2.70112
19	0.08252	30930.28	3.55	2.98517
20	0.08254	30888.53	3.00	6.74025
21	0.08255	30859.24	0.88	3.84303
22	0.08255	30832.77	0.33	4.47997
23	0.08256	30830.20	0.29	4.08082
24	0.08256	30822.42	0.76	3.60574
25	0.08256	30821.96	0.31	3.39634
26	0.08256	30818.17	0.43	3.08191

Display the fixed image and the registered image. Observe the alignment of the images.

```
figure
imshowpair(fixed,reg)
```



### Deformable Registration of MRI Volume

Load a MAT file containing a reference volume into the workspace. Convert the reference volume to data type double.

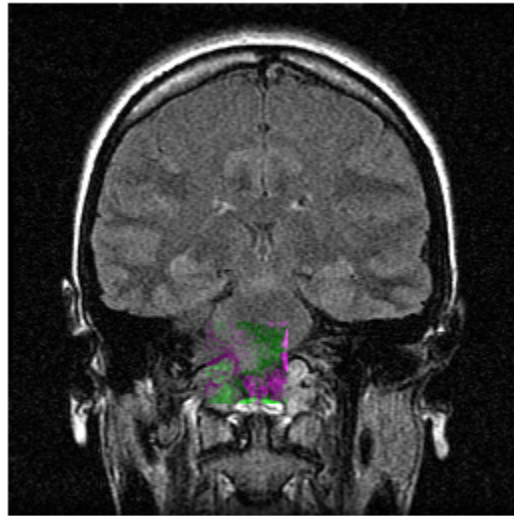
```
load mrystack.mat  
fixed = im2double(squeeze(mrystack));
```

Create a deformed volume using local transformations.

```
local = fixed(160:200,100:140,:);  
local = imrotate(local,90);  
moving = fixed;  
moving(160:200,100:140,:) = local;
```

Display a slice of the fixed volume and a corresponding slice of the moving volume. Observe the deformation in the alignment of the volumes.

```
figure  
imshowpair(fixed(:,:,10),moving(:,:,10))
```



Register the moving volume to the fixed volume.

```
[dispField,reg]=imregdeform(moving, fixed, GridRegularization=0.001);
```

----- Pyramid Level = 3 -----

Iteration	Normalized rmse	Function local minima	Step-size	Closeness to optimal solution
1	0.00016	34081.23	4.60	79.85176
2	0.00016	27591.18	16.75	70.06876
3	0.00019	26948.09	23.68	65.83620
4	0.00018	26498.21	5.02	59.19717
5	0.00018	25872.69	8.68	43.63346
6	0.00019	25774.23	7.70	46.14477
7	0.00019	25774.23	0.00	46.14477

----- Pyramid Level = 2 -----

Iteration	Normalized rmse	Function local minima	Step-size	Closeness to optimal solution
1	0.00009	24832.60	6.00	46.80394
2	0.00012	24491.93	29.78	58.92927
3	0.00014	21486.74	9.76	59.40296
4	0.00010	20100.15	7.82	36.86658
5	0.00010	19653.02	5.40	32.11689
6	0.00009	19344.57	6.25	27.88253
7	0.00009	19179.94	6.32	26.61265
8	0.00008	19105.13	6.35	25.79584
9	0.00008	19105.13	0.00	25.79584

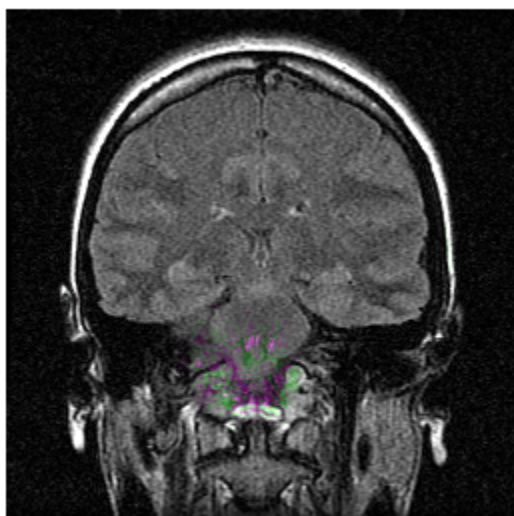
----- Pyramid Level = 1 -----

Iteration	Normalized rmse	Function local minima	Step-size	Closeness to optimal solution
1	0.00005	22834.26	6.43	25.64158
2	0.00006	21498.26	20.85	63.44257

3	0.00007	20173.48	11.87	81.18027
4	0.00006	20039.11	21.12	51.90494
5	0.00006	19909.26	3.44	51.33763
6	0.00005	19738.00	4.92	45.68723
7	0.00005	19542.03	6.75	37.99851
8	0.00005	19430.34	7.47	30.56819
9	0.00005	19396.99	4.20	26.59914

Display the same slice of the fixed volume and the corresponding slice of the registered volume. Observe the improved alignment of the volumes.

```
figure
imshowpair(fixed(:,:,10),reg(:,:,10))
```



## Input Arguments

### **moving** — Image or volume to be registered

2-D numeric matrix | 3-D numeric array

Image or volume to be registered, specified as a 2-D numeric matrix or 3-D numeric array, respectively. The size of **moving** must be the same as the size of **fixed**.

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

### **fixed** — Reference image or volume

2-D numeric matrix | 3-D numeric array

Reference image or volume, specified as a 2-D numeric matrix or 3-D numeric array, respectively. The size of **fixed** must be the same as the size of **moving**.

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: `[dispField,reg] = imregdeform(moving,fixed,NumPyramidLevels=6)` registers moving to fixed using six pyramid levels.

### **GridSpacing — Grid spacing**

two-element numeric vector | three-element numeric vector

Grid spacing, specified as a two-element or three-element numeric vector.

If `moving` and `fixed` are 2-D grayscale images, `GridSpacing` must be a two-element vector, and its default value is `[4 4]`. If `moving` and `fixed` are 3-D intensity volumes, `GridSpacing` must be a three-element vector, and its default value is `[4 4 4]`. Smaller values of `GridSpacing` specify a finer grid resolution.

Data Types: double

### **PixelResolution — Pixel size**

two-element numeric vector | three-element numeric vector

Pixel size, specified as a two-element or three-element numeric vector. Values are in millimeters.

If `moving` and `fixed` are 2-D grayscale images, `PixelResolution` must be a two-element vector, and its default value is `[1 1]`. If `moving` and `fixed` are 3-D intensity volumes, `PixelResolution` must be a three-element vector, and its default value is `[1 1 1]`.

Data Types: double

### **NumPyramidLevels — Number of multiresolution pyramid levels**

3 (default) | positive integer

Number of multiresolution pyramid levels, specified as a positive integer.

- If `moving` and `fixed` are 2-D grayscale images of size  $M$ -by- $N$ , then the value of `NumPyramidLevels` must satisfy the condition  $\min([M\ N]) > (2^{\text{NumPyramidLevels}}) * 0.7$ .
- If `moving` and `fixed` are 3-D intensity volumes of size  $M$ -by- $N$ -by- $P$ , then the value of `NumPyramidLevels` must satisfy the condition  $\min([M\ N\ P]) > (2^{\text{NumPyramidLevels}}) * 0.7$ .

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

### **GridRegularization — Weighing factor for grid displacement regularization**

0.11 (default) | nonnegative scalar

Weighing factor for grid displacement regularization, specified as a nonnegative scalar.

A large value for `GridRegularization` can create a smooth output displacement field, whereas a small value can create more localized displacements.

Data Types: double

### **DisplayProgress — Progress information output**

true or 1 (default) | false or 0



Progress information output, specified as a numeric or logical 1 (true) or 0 (false). Specify `DisplayProgress` as true to display information such as the number of iterations, normalized root mean square error (RMSE), function local minima, step size, and closeness to optimal solution.

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

## Output Arguments

### **dispField** — Displacement field

3-D numeric array | 4-D numeric array

Displacement field, returned as a 3-D or 4-D numeric array.

If `moving` and `fixed` are 2-D grayscale images of size  $M$ -by- $N$ , the displacement field `dispField` is a 3-D numeric array of size  $M$ -by- $N$ -by-2, where `dispField(:, :, 1)` and `dispField(:, :, 2)` contain the displacements in the  $X$ - and  $Y$ - directions, respectively. If `moving` and `fixed` are 3-D intensity volumes of size  $M$ -by- $N$ -by- $P$ , the displacement field `dispField` is a 4-D numeric array of size  $M$ -by- $N$ -by- $P$ -by-3, where `dispField(:, :, 1)`, `dispField(:, :, 2)`, and `dispField(:, :, 3)` contain the displacements in the  $X$ -,  $Y$ - and  $Z$ - directions, respectively.

Data Types: `double`

### **reg** — Registered image or volume

2-D numeric matrix | 3-D numeric array

Registered image or volume, returned with the same size and data type as `moving`.

## Version History

**Introduced in R2022b**

## References

- [1] Vishnevskiy, Valery, Tobias Gass, Gabor Szekely, Christine Tanner, and Orcun Goksel. "Isotropic Total Variation Regularization of Displacements in Parametric Image Registration." *IEEE Transactions on Medical Imaging* 36, no. 2 (February 2017): 385–95. <https://doi.org/10.1109/TMI.2016.2610583>.

## See Also

### Functions

`imregdemons` | `imregicp` | `imregmoment` | `imreggroupwise`

### Topics

"Medical Image Registration"

# imreggroupwise

Groupwise deformable registration

## Syntax

```
[dispField,reg] = imreggroupwise(moving)
[dispField,reg] = imreggroupwise(moving,Name=Value)
```

## Description

The `imreggroupwise` function uses the total variation method to perform deformable registration of slices in a series of grayscale images. You can use this function to reduce sliding motion between slices in a series of medical images, such as a timeseries. Registering all slices of the series to one of the slices using deformable registration in a `for` loop can introduce bias towards the artifacts of one slice in all the slices. In contrast, the `imreggroupwise` function reduces the overall range of sliding motion across all slices.

`[dispField,reg] = imreggroupwise(moving)` transforms the slices in the image series `moving` so that they are groupwise registered, and returns the displacement field `dispField` and the registered image series `reg`.

`[dispField,reg] = imreggroupwise(moving,Name=Value)` specifies options for the total variation method using one or more optional name-value arguments.

## Examples

### Groupwise Registration of MRI Image Series

Load an MRI volume into the workspace.

```
imgs = load("mristack.mat");
imgs = squeeze(imgs.mristack);
```

Pad the slices of the MRI volume to allow for rotation.

```
imgs = padarray(imgs,[10 10],"both");
```

Extract the 10th slice of the MRI volume, to use to generate the moving image series.

```
paddedSlice = imgs(:,:,10);
```

Generate the moving image series by rotating the slice by different angles.

```
moving = zeros(276,276,27);
for i = 1:27
    moving(:,:,i) = imrotate(paddedSlice,i,"crop");
end
```

Perform groupwise registration of the slices of the moving image series.

```
[dispField,reg] = imreggroupwise(moving,GridRegularization=0.05);
```

----- Pyramid Level = 3 -----

Iteration	f(x)	Step-size	Closeness to optimal solution
1	160720.92	11.50	18.19431
2	146622.17	41.61	31.55621
3	137582.30	54.80	37.51630
4	131892.89	21.32	30.22352
5	129146.80	12.05	27.98123
6	126260.51	18.80	34.57209
7	123292.77	21.42	32.76271
8	119756.72	34.44	26.53227
9	117145.16	27.64	33.13160
10	114257.37	28.72	39.06536
11	111212.99	50.30	35.02401
12	108848.93	28.47	27.27366
13	106838.15	16.68	21.59996
14	104141.37	48.40	29.34133
15	103034.70	43.98	36.78617
16	101421.24	11.33	21.98658
17	99846.91	25.65	19.28558
18	98950.73	20.29	22.02194
19	97683.82	55.50	23.47823
20	97058.47	35.79	56.52787
21	95693.10	12.29	39.94096
22	94308.28	31.93	50.77296
23	93620.93	21.01	23.42839
24	92756.49	43.04	26.44428
25	92160.48	39.07	20.02188
26	91649.86	20.14	23.32969
27	90848.45	52.28	12.75790
28	90785.24	35.18	50.98939
29	89193.25	13.04	24.02603
30	87913.57	26.84	23.86608
31	87261.74	13.25	27.18752
32	86609.40	28.99	20.23212
33	86403.18	36.29	19.85028
34	86171.97	12.29	14.73627
35	85991.59	43.41	19.17823
36	85880.43	31.97	52.82440
37	84848.06	22.60	38.11074
38	83896.58	24.78	26.63796
39	83221.23	17.79	31.42078
40	82422.75	15.26	17.38349
41	82386.70	42.07	30.26392
42	82118.45	10.10	20.60638
43	81714.94	31.01	27.63559
44	81475.85	25.61	35.72504
45	81219.57	11.81	30.83450
46	80900.59	42.13	21.99698
47	80701.15	27.56	46.08761
48	79735.79	18.84	29.77860
49	78885.47	19.72	21.33987
50	78443.85	14.00	33.84345
51	78025.38	19.61	21.81821
52	77820.46	10.80	27.58759
53	77566.30	14.89	14.12064
54	77356.17	16.30	24.02455
55	77210.20	17.62	13.38399

56	77175.39	16.57	13.65389
57	77139.47	16.67	14.31732
58	77103.48	16.91	14.75186
59	77052.18	18.54	13.45865
60	77007.15	17.58	11.45708
61	76995.53	6.92	12.12129

----- Pyramid Level = 2 -----

Iteration	f(x)	Step-size	Closeness to optimal solution
1	90397.57	9.98	16.07776
2	88598.63	13.10	23.03916
3	86659.87	23.18	36.97704
4	85346.69	26.94	46.28403
5	83821.96	9.70	20.80432
6	82514.48	16.43	25.34133
7	81522.62	18.21	27.43884
8	80714.91	45.26	44.56642
9	79515.86	16.33	17.96243
10	78667.76	10.10	25.12929
11	77756.18	24.19	27.73775
12	77571.40	32.06	51.91825
13	76629.12	7.86	22.85993
14	75929.67	19.14	31.05180
15	75590.59	16.23	14.57161
16	75563.08	47.79	52.11658
17	74964.02	19.25	34.85287
18	74383.90	8.57	36.63964
19	73887.61	25.26	20.34162
20	73676.18	8.94	12.66080
21	73360.07	26.43	19.09864
22	73190.90	31.92	33.94272
23	73032.36	10.59	20.21763
24	72764.07	32.08	14.04518
25	72626.99	24.21	27.84766
26	72493.83	30.79	28.66531
27	72090.89	18.12	17.14031
28	71660.52	18.11	11.76067
29	71269.90	15.98	24.21175
30	71067.73	22.38	15.58385
31	70985.12	8.50	14.58411
32	70888.81	11.41	13.56274
33	70796.71	14.63	12.23691
34	70729.12	15.27	12.57831
35	70721.71	14.16	12.59730
36	70716.14	5.36	12.87683

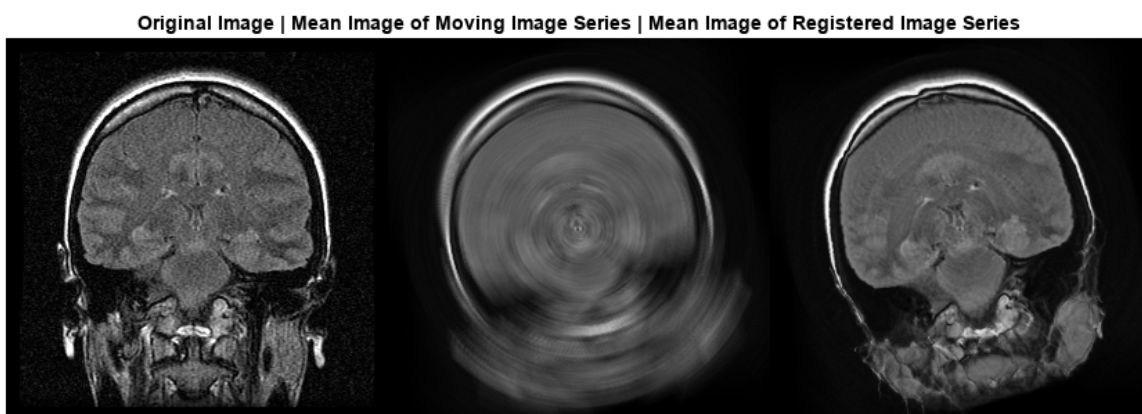
----- Pyramid Level = 1 -----

Iteration	f(x)	Step-size	Closeness to optimal solution
1	80123.51	13.71	13.80900
2	78453.64	14.18	11.56824
3	76729.32	27.24	26.34447
4	75799.10	26.53	37.72128
5	74508.05	9.21	19.73064
6	73460.13	20.96	27.40239
7	72800.90	16.67	26.18356
8	71919.69	32.43	14.43351

9	71545.75	30.56	32.57132
10	70717.01	15.97	18.49375
11	70052.45	19.04	22.48044
12	69587.50	17.23	23.88982
13	69094.55	28.76	22.76715
14	68720.48	24.05	24.80507
15	68212.79	16.41	14.46844
16	67766.80	23.74	14.20388
17	67589.19	23.70	29.27166
18	67121.86	14.29	13.96609
19	66779.83	23.35	12.63738
20	66611.24	19.61	21.53575
21	66390.03	23.81	15.41449
22	66226.75	9.53	10.36505
23	66174.37	35.33	14.71976
24	66046.98	6.29	11.75380
25	65777.74	20.41	21.22998
26	65694.36	26.64	37.03862
27	65478.70	9.46	33.20081
28	65330.62	28.51	30.64879
29	65193.94	14.45	30.87433
30	64969.61	20.04	27.73873
31	64863.63	7.47	23.45422
32	64742.49	10.56	18.98134
33	64655.24	11.63	19.72796
34	64589.34	12.86	11.98109
35	64575.96	6.52	13.75064
36	64563.45	8.01	13.59998
37	64558.37	3.78	12.33042

Visualize the output of the groupwise registration. The mean image of the moving image series shows that the slices are misaligned. In contrast, the mean image of the registered image series indicates alignment across slices.

```
figure
imshow([paddedSlice,mean(moving,3),mean(reg,3)])
title("Original Image | Mean Image of Moving Image Series " + ...
      "| Mean Image of Registered Image Series")
```



## Input Arguments

### **moving** — Image series to be registered

3-D numeric array

Image series to be registered, specified as a 3-D numeric array. The slices in the image series must capture the same anatomical slice of the body. For example, the image series can be a collection of the same slice imaged at different times.

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: `[dispField,reg] = imreggroupwise(moving,NumPyramidLevels=6)` registers the slices of `moving` using six pyramid levels.

### **GridSpacing** — Grid spacing

[4 4] (default) | two-element numeric vector

Grid spacing, specified as a two-element numeric vector. Smaller values of `GridSpacing` specify a finer grid resolution.

Data Types: `double`

### **PixelResolution** — Pixel size

[1 1] (default) | two-element numeric vector

Pixel size, specified as a two-element numeric vector. Values are in millimeters.

Data Types: `double`

### **NumPyramidLevels** — Number of multiresolution pyramid levels

3 (default) | positive integer

Number of multiresolution pyramid levels, specified as a positive integer.

If `moving` is of size  $M$ -by- $N$ -by- $P$ , then the value of `NumPyramidLevels` must satisfy the condition  $\min([M\ N\ P]) > (2^{\text{NumPyramidLevels}}) * 0.7$ .

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

### **GridRegularization** — Weighing factor for grid displacement regularization

0.11 (default) | nonnegative scalar

Weighing factor for grid displacement regularization, specified as a nonnegative scalar.

A large value for `GridRegularization` can create a smooth output displacement field, whereas a small value can create more localized displacements.

Data Types: `double`

### **DisplayProgress** — Progress information output

`true` or `1` (default) | `false` or `0`

Progress information output, specified as a numeric or logical 1 (`true`) or 0 (`false`). Specify `DisplayProgress` as `true` to display information such as the number of iterations, normalized root mean square error (RMSE), function local minima, step size, and closeness to optimal solution.

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

## Output Arguments

### **dispField** — Displacement field

4-D numeric array

Displacement field, returned as a 4-D numeric array.

If `moving` is of size  $M$ -by- $N$ -by- $P$ , the displacement field `dispField` is a 4-D numeric array of size  $M$ -by- $N$ -by-2-by- $P$ , where `dispField(:, :, 1, :)` and `dispField(:, :, 2, :)` contain the displacements for each of the slices in the  $X$ - and  $Y$ - directions, respectively.

### **reg** — Registered image series

3-D numeric array

Registered image series, returned with the same size and data type as `moving`.

## Version History

Introduced in R2022b

## References

- [1] Vishnevskiy, Valery, Tobias Gass, Gabor Szekely, Christine Tanner, and Orcun Goksel. "Isotropic Total Variation Regularization of Displacements in Parametric Image Registration." *IEEE Transactions on Medical Imaging* 36, no. 2 (February 2017): 385–95. <https://doi.org/10.1109/TMI.2016.2610583>.

## See Also

### Functions

`imregdemons` | `imregicp` | `imregmoment` | `imregdeform`

### Topics

"Medical Image Registration"

## imregicp

Surface registration using iterative closest point algorithm

### Syntax

```
regSurface = imregicp(movingSurface, fixedSurface)
regSurface = imregicp(movingSurface, fixedSurface, Name=Value)
[regSurface, tform] = imregicp(____)
[regSurface, tform, rmse] = imregicp(____)
```

### Description

The `imregicp` function uses the iterative closest point (ICP) algorithm for rigid registration of surfaces. Use this function to register surfaces extracted from medical volumes.

`regSurface = imregicp(movingSurface, fixedSurface)` transforms the surface `movingSurface`, so that it is registered with the reference surface `fixedSurface` using the ICP algorithm. The function returns the registered surface `regSurface`.

`regSurface = imregicp(movingSurface, fixedSurface, Name=Value)` specifies options for the ICP algorithm using one or more optional name-value arguments.

`[regSurface, tform] = imregicp(____)` returns the transformation `tform` between the moving surface and the registered surface, in addition to any combination of input arguments from previous syntaxes.

`[regSurface, tform, rmse] = imregicp(____)` returns the root mean squared error (RMSE) `rmse` of the Euclidean distance between the inlier points of the aligned surfaces `regSurface` and `fixedSurface`, in addition to any combination of input arguments from previous syntaxes.

### Examples

#### Surface Registration of Isosurfaces

Load a MAT file containing intensity volume data into the workspace. Extract the volume data.

```
load(fullfile(toolboxdir("images"), "imdata", "BrainMRILabeled", "images", "vol_001.mat"));
V = vol;
```

Specify an isovalue for isosurface extraction.

```
isovalue = 0.5;
```

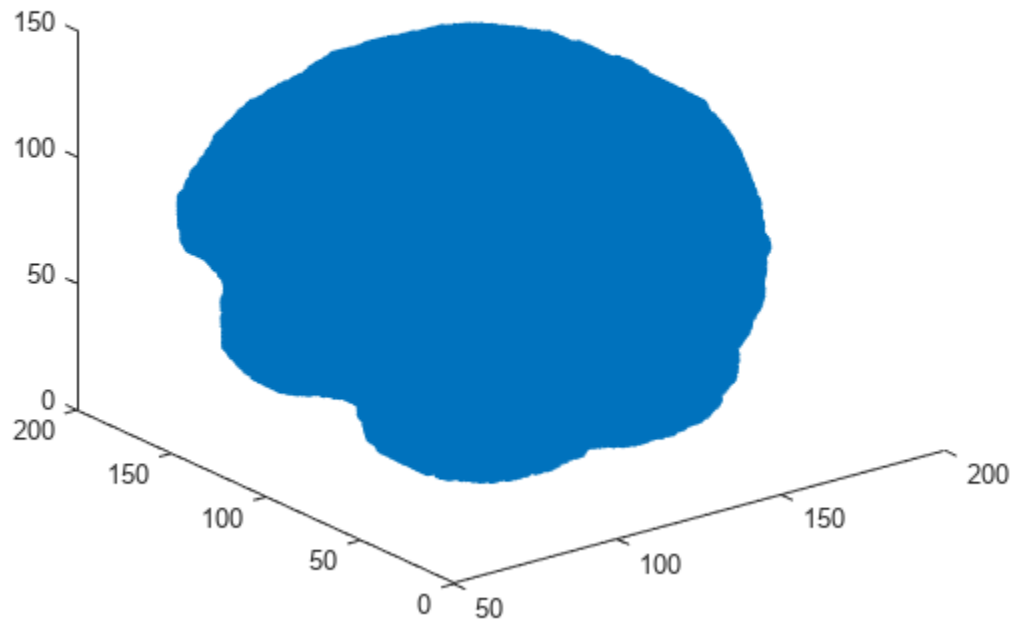
Extract the isosurface of the reference volume at the specified isovalue.

```
[~, fixedSurface] = extractIsosurface(V, isovalue);
```

Display and inspect the reference isosurface.

```
figure
plot3(fixedSurface(:,2), fixedSurface(:,1), fixedSurface(:,3), "o")
```





Create a rigid transformation, as an `affinetform3d` object, for the reference volume.

```
theta = pi/18;
affineMatrix = [cos(theta)  sin(theta)  0  5; ...
               -sin(theta) cos(theta)  0  5; ...
                0           0          1 10; ...
                0           0          0  1];
tform_rigid = affinetform3d(affineMatrix);
```

Transform the reference volume using the rigid transformation.

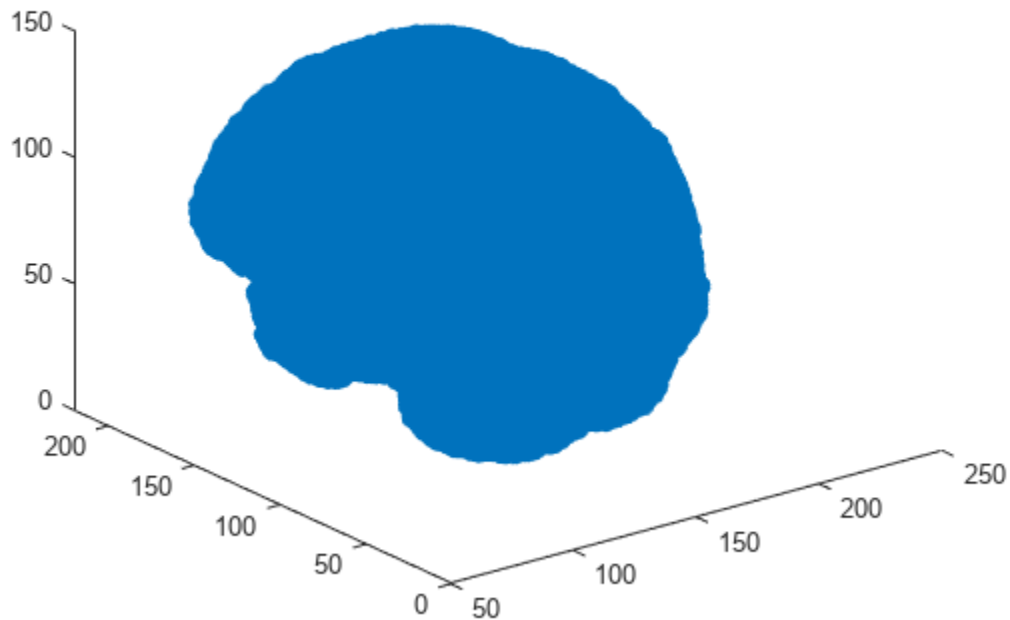
```
tformV = imwarp(V,tform_rigid);
```

Extract the isosurface of the transformed volume at the specified isovalue.

```
[~,movingSurface] = extractIsosurface(tformV,isovalue);
```

Display and inspect the transformed isosurface.

```
figure
plot3(movingSurface(:,2),movingSurface(:,1),movingSurface(:,3),"o")
```



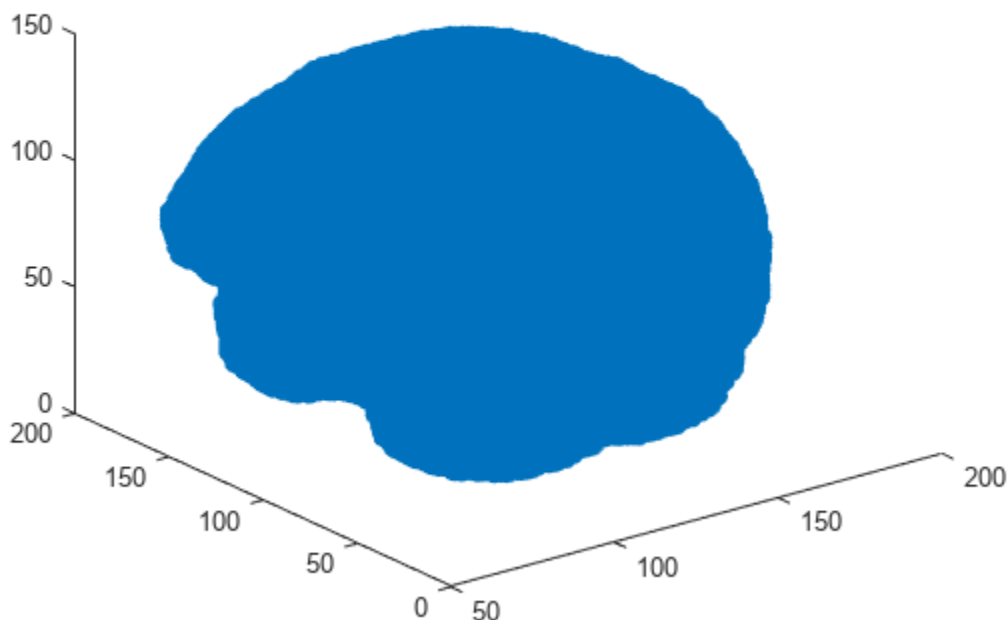
Register the transformed isosurface with respect to the reference isosurface using surface registration.

```
regSurface = imregicp(movingSurface, fixedSurface, DistanceThreshold=30);
```

iterations = 1	Fitness = 1.0000	inlierRmse = 11.2527
iterations = 2	Fitness = 1.0000	inlierRmse = 8.7565
iterations = 3	Fitness = 1.0000	inlierRmse = 6.9034
iterations = 4	Fitness = 1.0000	inlierRmse = 5.5111
iterations = 5	Fitness = 1.0000	inlierRmse = 4.5172
iterations = 6	Fitness = 1.0000	inlierRmse = 3.7946
iterations = 7	Fitness = 1.0000	inlierRmse = 3.2685
iterations = 8	Fitness = 1.0000	inlierRmse = 2.8860
iterations = 9	Fitness = 1.0000	inlierRmse = 2.6006
iterations = 10	Fitness = 1.0000	inlierRmse = 2.3830
iterations = 11	Fitness = 1.0000	inlierRmse = 2.2119
iterations = 12	Fitness = 1.0000	inlierRmse = 2.0698
iterations = 13	Fitness = 1.0000	inlierRmse = 1.9481
iterations = 14	Fitness = 1.0000	inlierRmse = 1.8440
iterations = 15	Fitness = 1.0000	inlierRmse = 1.7480
iterations = 16	Fitness = 1.0000	inlierRmse = 1.6617
iterations = 17	Fitness = 1.0000	inlierRmse = 1.5824
iterations = 18	Fitness = 1.0000	inlierRmse = 1.5087
iterations = 19	Fitness = 1.0000	inlierRmse = 1.4391
iterations = 20	Fitness = 1.0000	inlierRmse = 1.3745

Display and inspect the registered isosurface.

```
figure
plot3(regSurface(:,2),regSurface(:,1),regSurface(:,3),"o")
```



## Input Arguments

### **movingSurface** — Surface to be registered

*M*-by-3 numeric matrix

Surface to be registered, specified as an *M*-by-3 numeric matrix. *M* is the number of points in the surface. Each row contains the 3-D coordinates of a surface point.

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

### **fixedSurface** — Reference surface

*M*-by-3 numeric matrix

Reference surface, specified as an *M*-by-3 numeric matrix. *M* is the number of points in the surface. Each row contains the 3-D coordinates of a surface point.

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: `regSurface = imregicp(movingSurface, fixedSurface, Metric="pointToPlane")` registers the surfaces using the "pointToPlane" minimization metric for the ICP algorithm.

**Metric — Minimization metric**

"pointToPoint" (default) | "pointToPlane"

Minimization metric, specified as "pointToPoint" or "pointToPlane".

When registering planar surfaces, you can use the "pointToPlane" metric to reduce the number of iterations in the algorithm, but at a higher computational complexity for each iteration.

Data Types: `char` | `string`

**MaxIterations — Maximum number of iterations**

20 (default) | positive integer

Maximum number of iterations for the ICP algorithm, specified as a positive integer.

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

**InitialTransform — Initial rigid transformation**

`affinetform3d()` (default) | `affinetform3d` object

Initial rigid transformation, specified as an `affinetform3d` object.

**OutlierRemoval — Parameters for outlier removal**

[20 2] (default) | two-element vector

Parameters for outlier removal, specified as a two-element vector of the form [ $n$   $r$ ].

The function considers the surface points with fewer than  $n$  neighbors within a sphere of radius  $r$  to be outliers, and removes them from consideration for the registered surface.  $n$  must be a positive integer, and  $r$  must be positive. Specifying a high value for  $n$  with a low value for  $r$  causes the function to remove outliers aggressively.

Data Types: `single` | `double`

**DistanceThreshold — Distance threshold for closest point**

0.1 (default) | positive scalar

Distance threshold for closest point, specified as a positive scalar. The `DistanceThreshold` is the maximum distance at which the k-nearest neighbor search considers a point in `movingSurface` as a possible correspondent for a point in `fixedSurface`. Increasing the distance threshold can enable you to detect a correspondence in images with sparse surface points, but can also cause the algorithm to return false positives.

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

**NumPoints — Number of points used for local plane fitting**

6 (default) | positive integer greater than or equal to 3

Number of points used for local plane fitting, specified as a positive integer greater than or equal to 3. To specify this argument, you must specify `Metric` as "pointToPlane". Smaller values of `NumPoints` can result in faster computation, but can reduce accuracy.

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

**Verbose — Progress information output**

true or 1 (default) | false or 0

Progress information output, specified as a numeric or logical 1 (true) or 0 (false). Specify `Verbose` as true to display information such as the number of iterations, fitness score, and inlier RMSE value.

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

**Output Arguments****regSurface — Registered surface**

$M$ -by-3 numeric matrix

Registered surface, returned as an  $M$ -by-3 numeric matrix.  $M$  is the number of points in the surface. Each row contains the 3-D coordinates of a surface point.

Data Types: double | single

**tform — Rigid transformation**

affinetform3d object

Rigid transformation, returned as an `affinetform3d` object.

**rmse — Root mean square error**

numeric scalar

Root mean square error of the Euclidean distance between the inlier points of the aligned surfaces `regSurface` and `fixedSurface`, returned as a numeric scalar.

Data Types: double

**Limitations**

- The `imregicp` function is not supported on Mac computers with Apple silicon chips.

**Version History**

Introduced in R2022b

**References**

- [1] Besl, P.J., and Neil D. McKay. "A Method for Registration of 3-D Shapes." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14, no. 2 (February 1992): 239-56. <https://doi.org/10.1109/34.121791>.

**See Also****Functions**

`imregister` | `imregmoment` | `imregdeform` | `imreggroupwise`

**Objects**

affinetform3d

**Topics**

“Medical Image Registration”

# imregmoment

Fast registration of grayscale images or intensity volumes using moment of mass method

## Syntax

```
[tform,reg] = imregmoment(moving, fixed)
[tform,reg] = imregmoment(moving, movingRef, fixed, fixedRef)
[tform,reg] = imregmoment( ___, MedianThresholdBitmap=mtb)
```

## Description

The `imregmoment` function uses the moment of mass method for fast, similarity registration of grayscale images or intensity volumes. You can use this function to register multimodal medical images or volumes as a preprocessing step before multimodal medical image fusion.

`[tform,reg] = imregmoment(moving, fixed)` transforms the grayscale image or intensity volume `moving`, so that it is registered with the reference image or volume `fixed`, and returns the transformation `tform` and the registered image or volume `reg`.

`[tform,reg] = imregmoment(moving, movingRef, fixed, fixedRef)` specifies the spatial referencing information `movingRef` and `fixedRef` for `moving` and `fixed`, respectively.

`[tform,reg] = imregmoment( ___, MedianThresholdBitmap=mtb)` specifies whether to threshold `moving` and `fixed` before registration, in addition to any combination of input arguments from previous syntaxes.

## Examples

### Fast Registration of Multimodal Medical Images

The data used in this example is a modified version of the 3-D CT and MRI datasets provided by Dr. Michael Fitzpatrick as part of The Retrospective Image Registration Evaluation (RIRE) Dataset. The modified dataset contains the CT and MRI scans stored in the NRRD file format. The size of the entire data set is approximately 35 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", ...
    "MedicalRegistrationNRRDdata.zip");
filepath = fileparts(zipFile);
unzip(zipFile, filepath);
```

Read an MRI volume into the workspace as the reference volume. Get the spatial referencing information for the MRI volume.

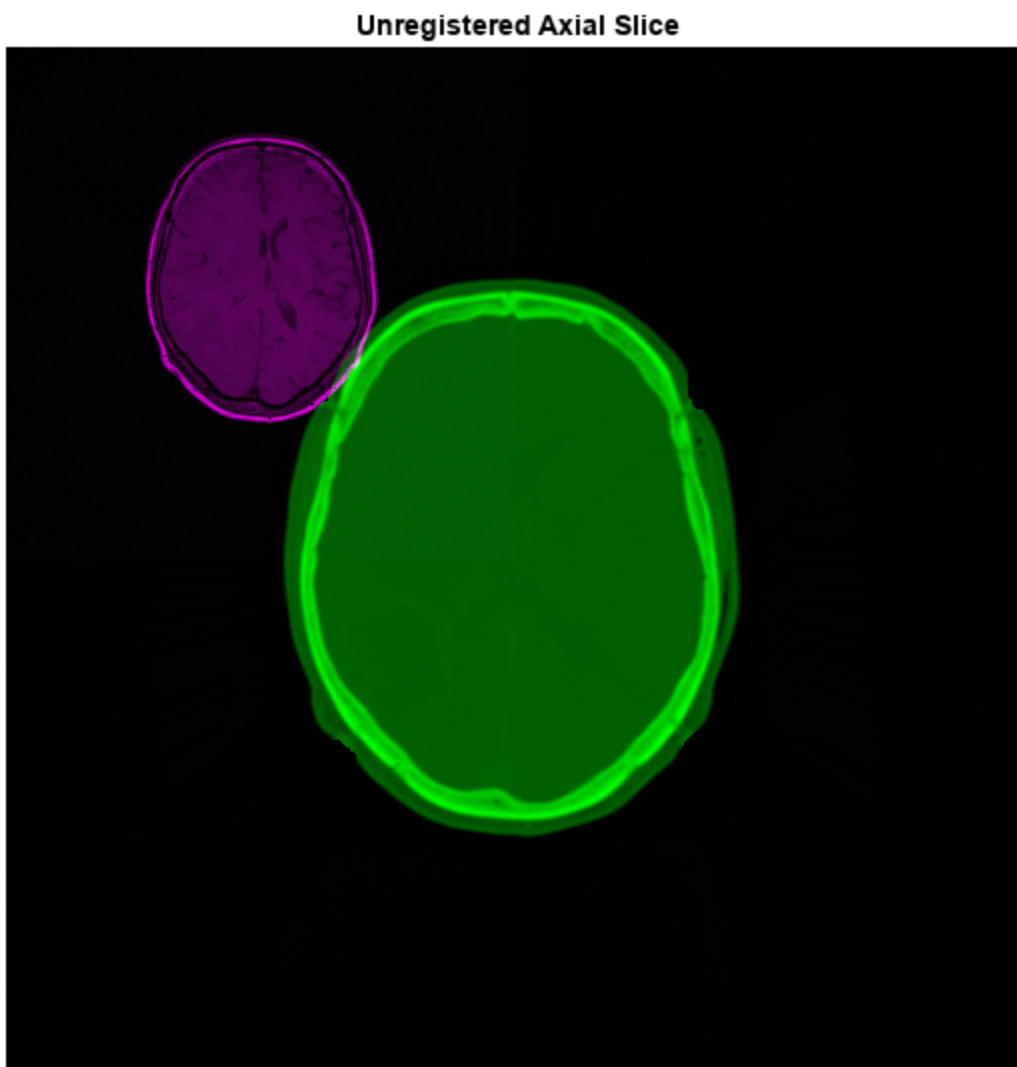
```
fixedFile = fullfile(filepath, "supportfilesNRRD", "Patient007MRT1.nrrd");
fixedVol = nrrdinfo(fixedFile);
fixed = nrrdread(fixedFile);
fixedRef = imref3d(size(fixed), fixedVol.PixelDimensions(2), ...
    fixedVol.PixelDimensions(1), fixedVol.PixelDimensions(3));
```

Read a CT volume into the workspace as the volume to be registered. Get the spatial referencing information for the CT volume.

```
movingFile = fullfile(filepath,"supportfilesNRRD","Patient007CT.nrrd");  
movingVol = nrrdinfo(movingFile);  
moving = nrrdread(movingFile);  
movingRef = imref3d(size(moving),movingVol.PixelDimensions(2), ...  
    movingVol.PixelDimensions(1),movingVol.PixelDimensions(3));
```

Compare a slice of the unregistered volume to a corresponding slice of the reference volume.

```
centerFixed = size(fixed)/2;  
centerMoving = size(moving)/2;  
figure  
imshowpair(moving(:,:,centerMoving(3)),fixed(:,:,centerFixed(3)))  
title("Unregistered Axial Slice")
```



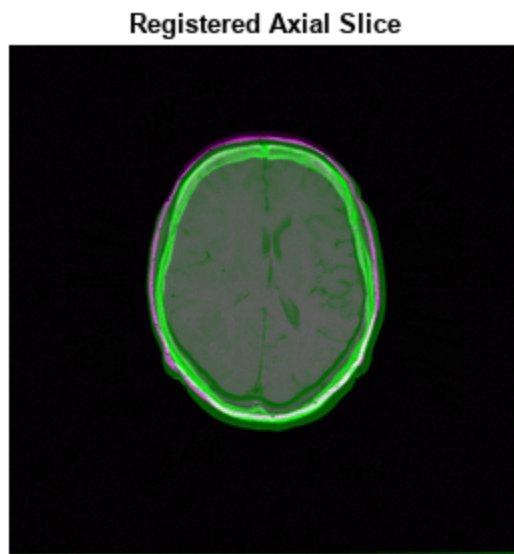


Register the volumes, and return the registered volume and the transformation between the moving and fixed volumes.

```
[tform,reg] = imregmoment(moving,movingRef,fixed,fixedRef,MedianThresholdBitmap=true);
```

Compare the same slice of the registered volume to the corresponding slice of the reference volume.

```
figure
imshowpair(reg(:,:,centerFixed(3)),fixed(:,:,centerFixed(3)))
title("Registered Axial Slice")
```



## Input Arguments

### **moving** — Image or volume to be registered

2-D numeric matrix | 3-D numeric array

Image or volume to be registered, specified as a 2-D numeric matrix or 3-D numeric array, respectively.

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

### **fixed** — Reference image or volume

2-D numeric matrix | 3-D numeric array

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

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

### **movingRef** — Spatial referencing information for image or volume to be registered

`imref2d` object | `imref3d` object

Spatial referencing information for the image or volume to be registered, specified as an `imref2d` object or an `imref3d` object, respectively. Use the spatial referencing inputs when the images or volumes differ in size by a scaling factor.

**fixedRef — Spatial referencing information for reference image or volume**

`imref2d` object | `imref3d` object

Spatial referencing information for the reference image or volume, specified as an `imref2d` object or an `imref3d` object, respectively. Use the spatial referencing inputs when the images or volumes differ in size by a scaling factor.

**mtb — Median threshold bitmap processing**

`false` or `0` (default) | `true` or `1`

Median threshold bitmap processing, specified as a numeric or logical `0` (`false`) or `1` (`true`). Specify `MedianThresholdBitmap` as `true` to threshold moving and fixed. This can improve registration if the images have been captured using different sensors or have different intensity levels.

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

## Output Arguments

**tform — Similarity transformation**

`affinetform2d` object | `affinetform3d` object

Similarity transformation, returned as an `affinetform2d` object or an `affinetform3d` object.

**reg — Registered image or volume**

2-D numeric matrix | 3-D numeric array

Registered image or volume, returned with the same size as `fixed` and same data type as `moving`.

## Version History

**Introduced in R2022b**

## See Also

**Functions**

`imregmtb` | `imregicp` | `imregdeform` | `imreggroupwise`

**Objects**

`imref2d` | `imref3d` | `affinetform2d` | `affinetform3d`

**Topics**

“Medical Image Registration”

# isnrrd

Check if file is valid NRRD file

## Syntax

```
tf = isnrrd(filename)
```

## Description

`tf = isnrrd(filename)` checks if the specified file is a valid nearly raw raster data (NRRD) file.

## Examples

### Check If File Is Valid NRRD File

Check whether a file is a valid NRRD format file. The file is part of a data set containing the 3-D CT and MRI scans from The Retrospective Image Registration Evaluation (RIRE) Dataset, converted to the NRRD file format. The original data set was provided by Dr. Michael Fitzpatrick. For more information, see the RIRE Project homepage. The size of the entire data set is approximately 35 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", ...
    "MedicalRegistrationNRRDdata.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

Specify the name of the NRRD file to check.

```
filename = fullfile(filepath,"supportfilesNRRD","Patient007CT.nrrd");
```

Check if `filename` is a valid NRRD file. The `isnrrd` function determines that the file is valid.

```
isnrrd(filename)
```

```
ans = logical
     1
```

## Input Arguments

### **filename** — Name of NRRD file

string scalar | character vector

Name of the NRRD file, specified as a string scalar or a character vector. The file must contain a valid NRRD file header. `filename` can contain the absolute path to the file, a relative path from the current directory, or a relative path from a directory on the MATLAB path.

Data Types: char | string

## Output Arguments

### **tf** — File is valid NRRD file

true or 1 | false or 0

File is a valid NRRD file, returned as a logical 1 (**true**) or 0 (**false**). The output value is 1 if `filename` contains a valid NRRD header. The function can determine the validity of NRRD files that contain image data or are detached header files.

## Version History

**Introduced in R2022b**

### **See Also**

`nrrdinfo` | `nrrdread`

# specklefilt

Filter image using speckle-reducing anisotropic diffusion

## Syntax

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

## Description

Coherent imaging modalities, such as ultrasound images, are prone to degradation because of the interference of the transmitted waveform and its echoes. This degradation is called speckle, and is a form of multiplicative noise. The `specklefilt` function uses a speckle-reducing anisotropic diffusion (SRAD) algorithm to reduce the speckle in an image.

`J = specklefilt(I)` filters the image `I` using SRAD and returns the filtered image `J`.

`J = specklefilt(I,Name=Value)` fine-tunes the behavior of the SRAD algorithm using one or more optional name-value arguments.

## Examples

### Adjust Parameters for Different Smoothing Levels

Import an ultrasound image into the workspace.

```
I = imread("heartUltrasoundImage.png");
```

Filter the image using the speckle-reducing anisotropic diffusion algorithm with default parameters.

```
J1 = specklefilt(I);
```

You can increase the smoothing the filter performs on the image by increasing the degree of smoothing of the algorithm. Specify a degree of smoothing of 0.6, increased from the default value of 0.2.

```
J2 = specklefilt(I,DegreeOfSmoothing=0.6);
```

You can also increase the smoothing by increasing the number of iterations performed by the algorithm. Specify for the algorithm to perform 50 iterations, increased from the default value of 30.

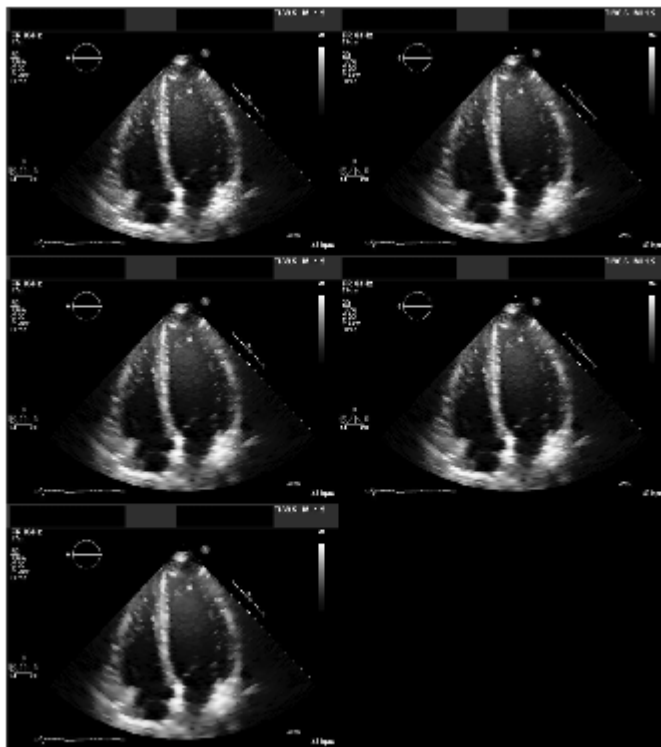
```
J3 = specklefilt(I,NumIterations=50);
```

Filter the image using the SRAD algorithm with both an increased degree of smoothing and greater number of iterations. Set the degree of smoothing to 0.6 and the number of iterations to 50.

```
J4 = specklefilt(I,DegreeOfSmoothing=0.6,NumIterations=50);
```

Display the original image alongside the denoised images, and inspect the different smoothing levels.

```
figure  
montage({I J1 J2 J3 J4})
```



### Filter Speckle Generated from Different Noise Models

Import an ultrasound image into the workspace.

```
I = imread("heartUltrasoundImage.png");
```

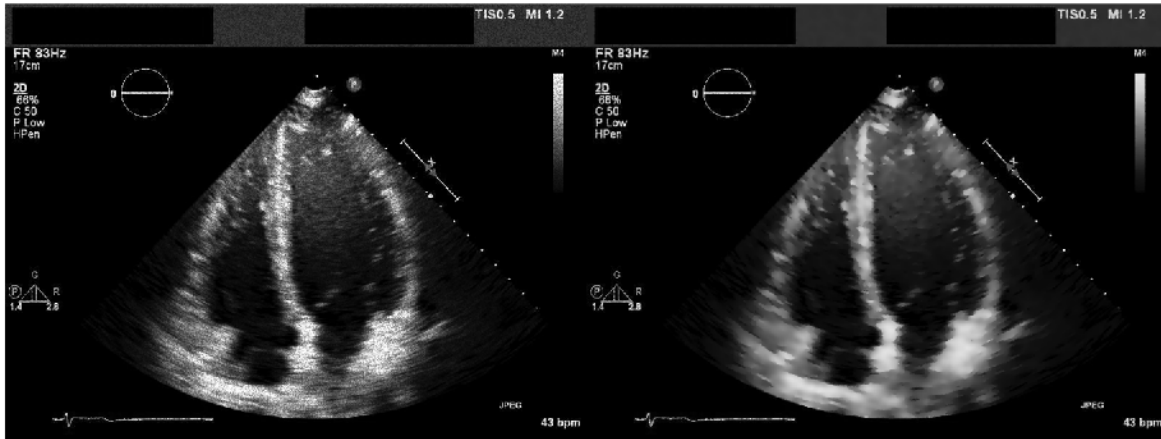
Corrupt the image with multiplicative speckle drawn from a uniform distribution.

```
Inoise = imnoise(I,"speckle");
```

Filter the corrupted image to remove speckle. Display the corrupted and denoised images.

```
J = specklefilt(Inoise,DegreeOfSmoothing=0.6,NumIterations=50);
```

```
figure  
montage({Inoise J})
```



Compute the structural similarity of the denoised image J with the original image I.

```
disp(ssim(I,J))
```

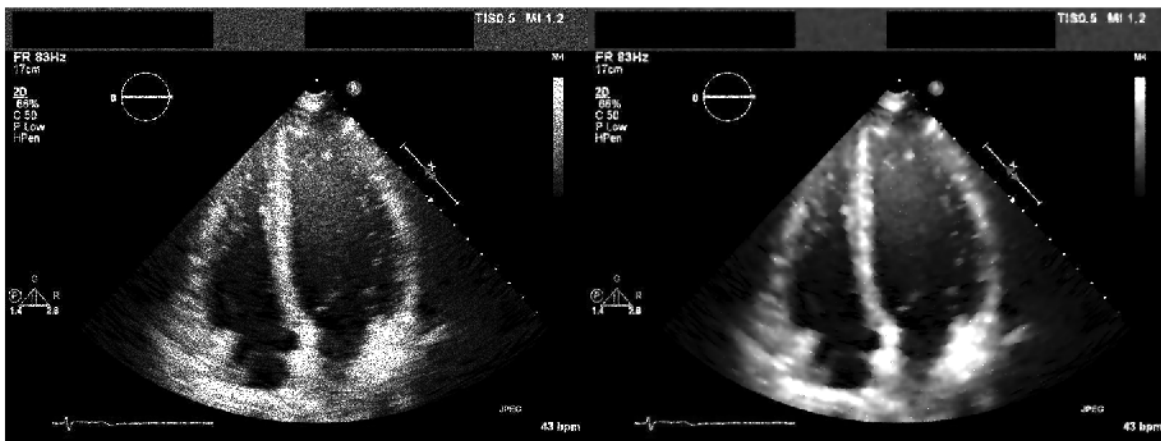
```
0.9330
```

Corrupt the image with multiplicative speckle drawn from a Rayleigh distribution.

```
I = im2double(I);
Inoise = I.*raylrnd(1,size(I));
```

Filter the corrupted image to remove speckle. Display the corrupted and denoised images.

```
J = specklefilt(Inoise,DegreeOfSmoothing=0.6,NumIterations=50);
figure
montage({Inoise J})
```



Compute the structural similarity between the denoised image J and the original image I.

```
disp(ssim(I,J))
0.8868
```

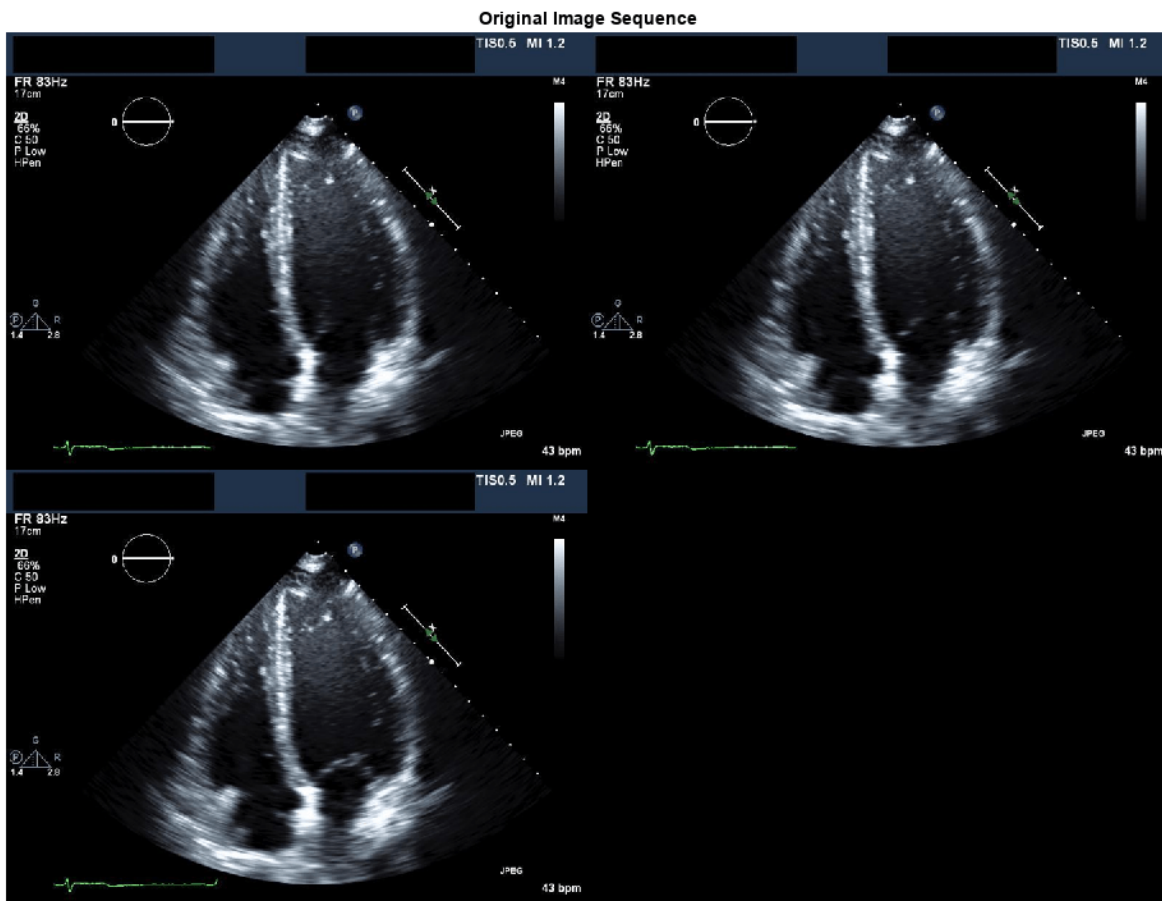
### Filter Image Sequence

Import an ultrasound image sequence into the workspace.

```
I = dicomread("heartUltrasoundSequence.dcm");
[h,w,c,d] = size(I);
```

View the original image sequence.

```
figure
montage(I)
title("Original Image Sequence")
```



Convert each image in the sequence to grayscale. Filter the grayscale images.

```
J = zeros(h,w,d);
for i = 1:d
```



```

Ii = im2double(im2gray(I(:,:,i)));
J(:,:,i) = specklefilt(Ii,DegreeOfSmoothing=0.6,NumIterations=50);
end

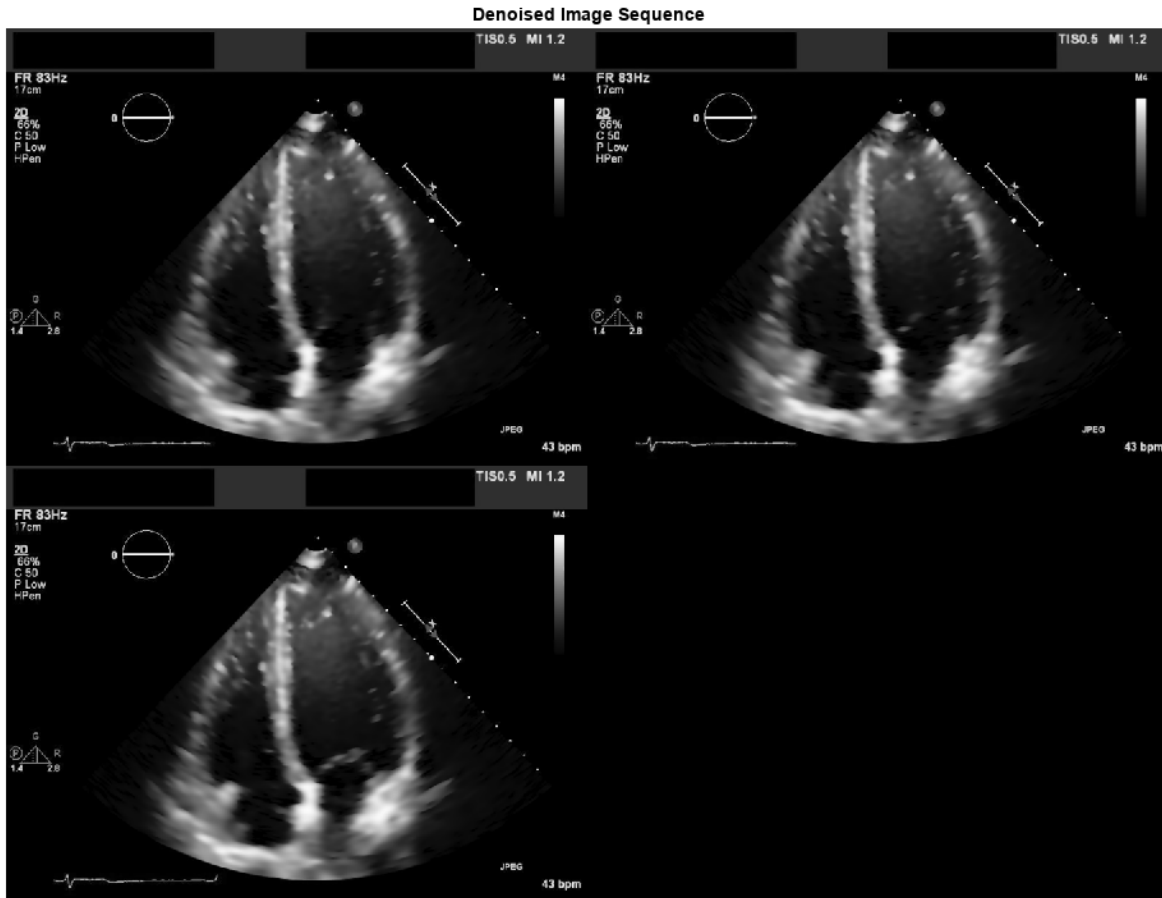
```

View the denoised image sequence.

```

figure
montage(J)
title("Denoised Image Sequence")

```



## Input Arguments

### **I** — Image to be filtered

2-D numeric matrix

Image to be filtered, specified as a 2-D numeric matrix.

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

## Name-Value 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: `J = specklefilt(I, DegreeOfSmoothing=0.6, NumIterations=50)` applies the SRAD filter to image `I` with a degree of smoothing of 0.6 for 50 iterations, and returns the filtered image `J`.

### DegreeOfSmoothing — Degree of smoothing

0.2 (default) | scalar in the range (0,1]

Degree of smoothing of the SRAD filter, specified as a scalar in the range (0, 1]. Increasing the value of `DegreeOfSmoothing` denoises the image to a greater degree.

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

### NumIterations — Number of iterations

30 (default) | positive integer

Number of iterations of the SRAD filter, specified as a positive integer. Increasing the value of `NumIterations` denoises the image to a greater degree.

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

## Output Arguments

### J — SRAD-filtered image

2-D numeric matrix

SRAD-filtered image, returned as a 2-D numeric matrix of the same size and data type as the input image `I`.

## Algorithms

The speckle-reducing anisotropic diffusion used in `specklefilt` combines elements of speckle-reducing filters with the edge-preserving anisotropic diffusion filter used in `imdiffusefilt`. This makes `specklefilt` useful for edge-preserving denoising of images corrupted with speckle.

## Version History

Introduced in R2022b

## References

- [1] Yongjian Yu, and S.T. Acton. "Speckle Reducing Anisotropic Diffusion." *IEEE Transactions on Image Processing* 11, no. 11 (November 2002): 1260-70. <https://doi.org/10.1109/TIP.2002.804276>.
- [2] Aja-Fernandez, S., and C. Alberola-Lopez. "On the Estimation of the Coefficient of Variation for Anisotropic Diffusion Speckle Filtering." *IEEE Transactions on Image Processing* 15, no. 9 (September 2006): 2694-2701. <https://doi.org/10.1109/TIP.2006.877360>.

## **See Also**

imdiffusefilt

## **Topics**

“Read, Process, and View Ultrasound Data”

## jitterIntensity

Randomly augment intensity of grayscale image or intensity volume

### Syntax

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

### Description

You can use data augmentation to increase the variety and quantity of training data in deep learning applications, especially when available training data is limited, as is typical in medical imaging. Data augmentation can be intensity augmentation, geometric augmentation, or color augmentation. The `jitterIntensity` function performs intensity augmentation of grayscale images and intensity volumes by randomly augmenting their brightness, contrast, and gamma correction.

`J = jitterIntensity(I,Name=Value)` jitters the intensity of grayscale image or intensity volume `I` by randomly selecting brightness (shifting of intensity), contrast (scaling of intensity), and gamma correction values. To specify ranges for these values, use the corresponding name-value arguments.

### Examples

#### Jitter Intensity of 2-D Computed Tomography (CT) Image

Import a grayscale CT image into the workspace. Crop the image to retain only the object of interest.

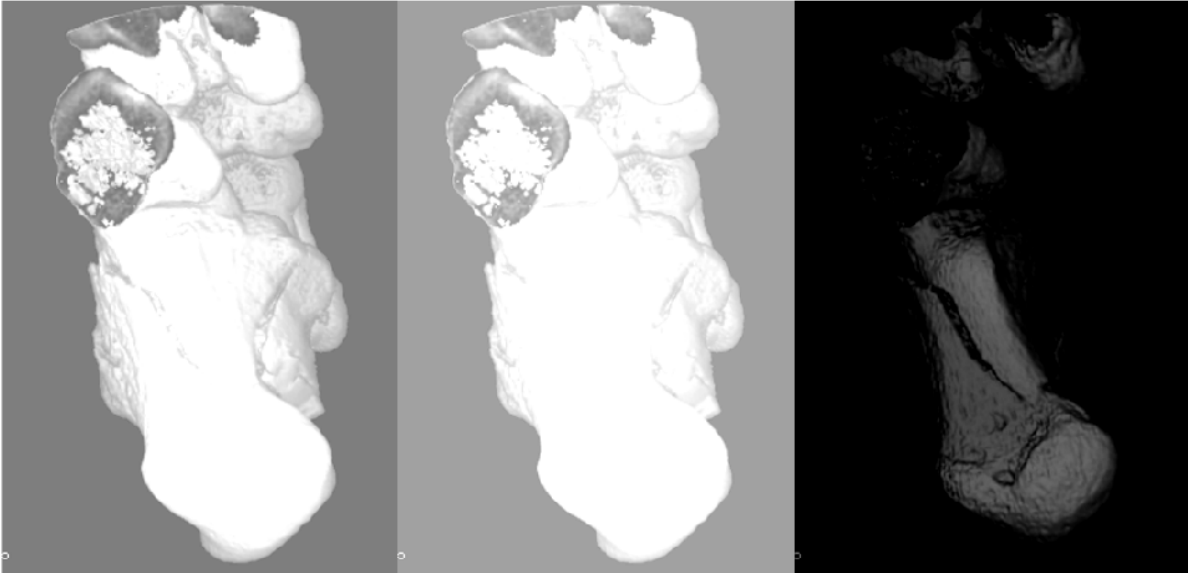
```
I = dicomread("CT-M0N02-16-ankle.dcm");  
I = imcrop(I,[101 51 290 420]);
```

Randomly shift the intensity of the image `I` multiple times.

```
J1 = jitterIntensity(im2single(I),Brightness=0.75);  
J2 = jitterIntensity(im2single(I),Brightness=0.75);  
J3 = jitterIntensity(im2single(I),Brightness=0.75);
```

Visualize the augmented CT images to observe the impact of the randomly selected brightness value.

```
figure  
montage({J1 J2 J3})
```



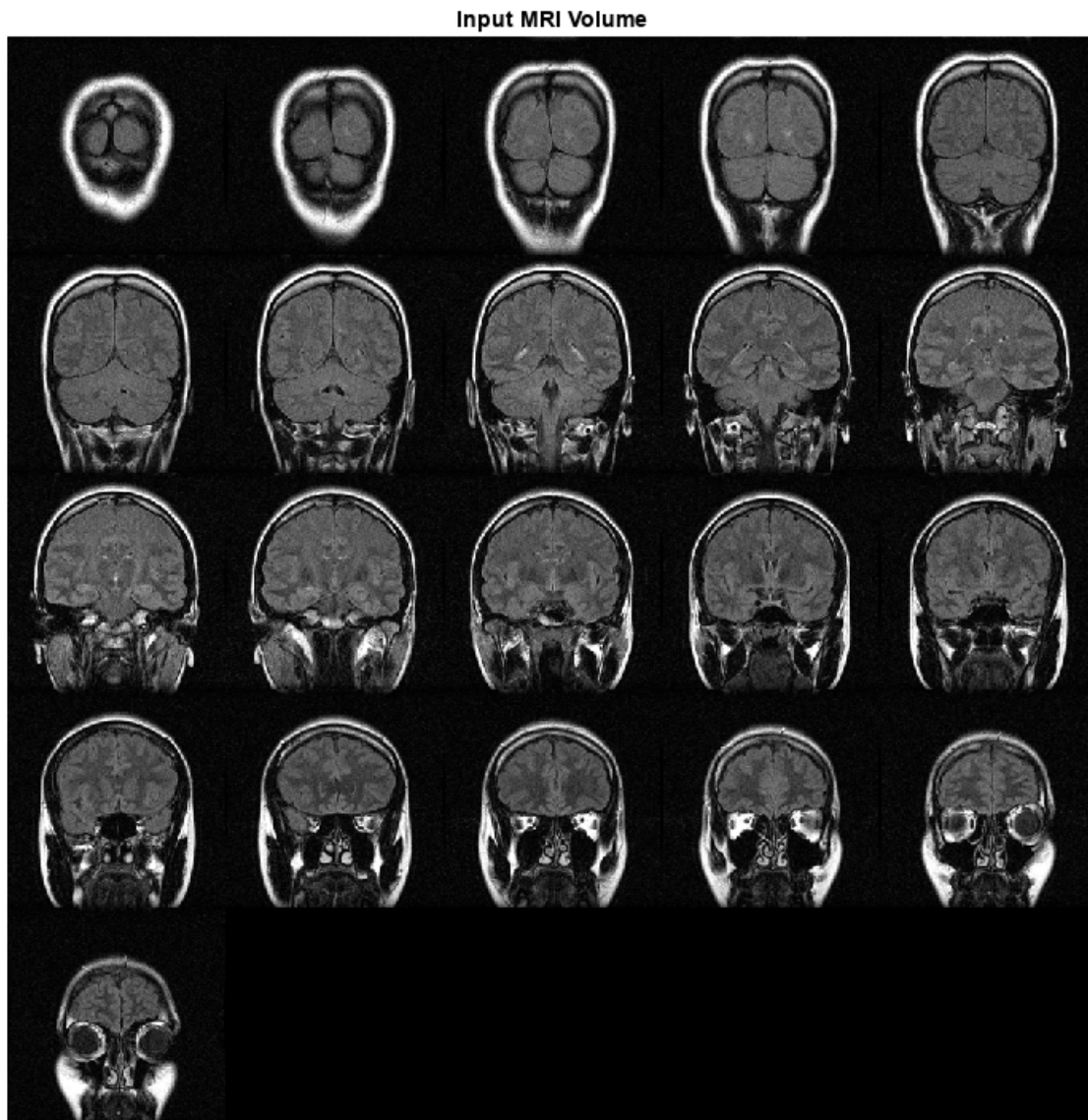
### Jitter Intensity of 3-D Magnetic Resonance Imaging (MRI) Volume

Load an MRI intensity volume into the workspace.

```
load("mristack.mat","mristack");  
V = mristack;
```

Visualize the MRI volume.

```
figure  
montage(V)  
title("Input MRI Volume")
```



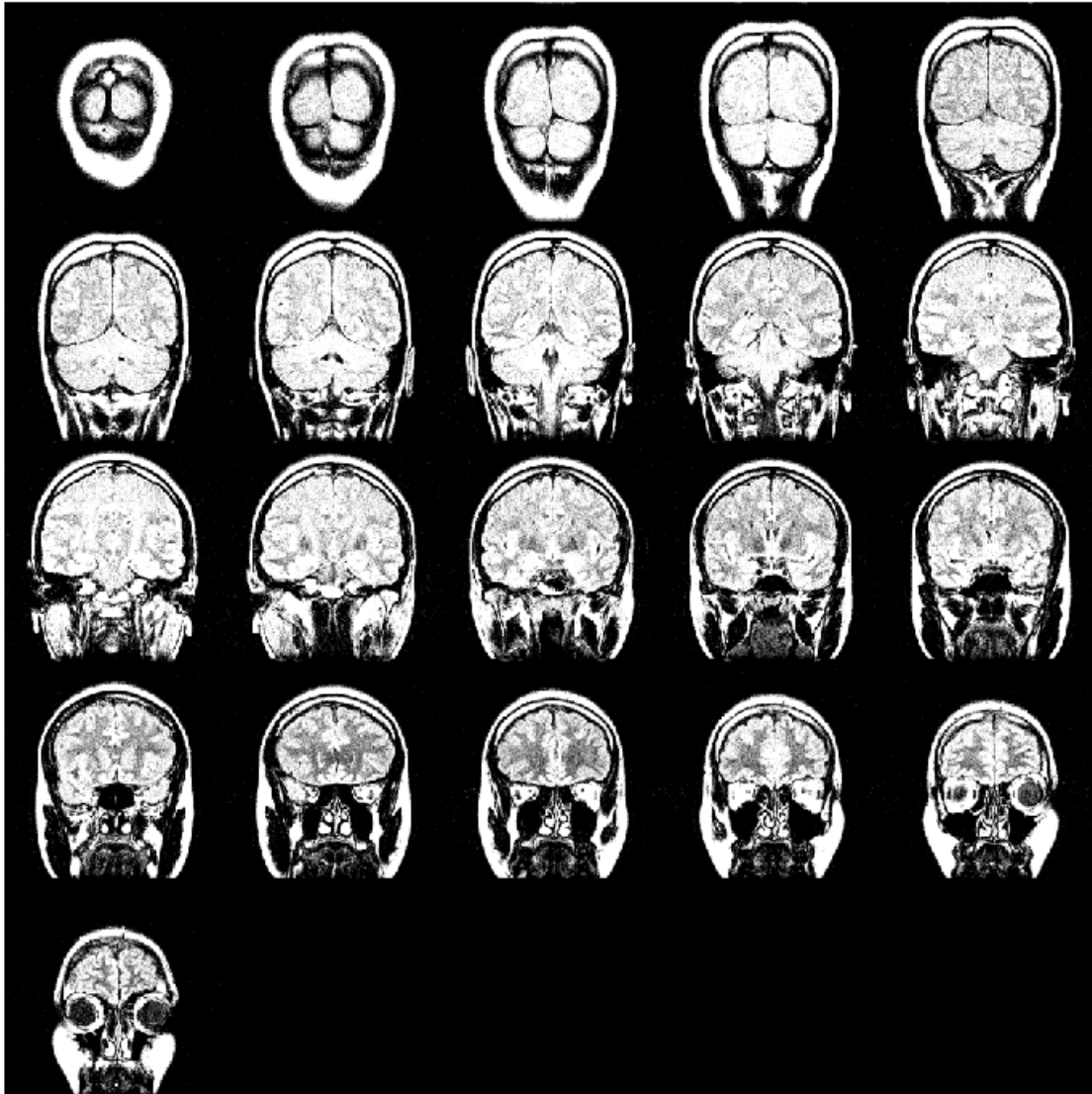
Randomly shift and scale the intensity of the volume  $V$ .

```
jitterV = jitterIntensity(V,Brightness=[-0.1 0.2],Contrast=3);
```

Visualize the augmented MRI volume.

```
figure  
montage(jitterV)  
title("Augmented MRI Volume")
```

Augmented MRI Volume



### Augment Datastore for Deep Learning

Specify the location of a directory containing DICOM image files. Create a datastore for deep learning from the DICOM files.

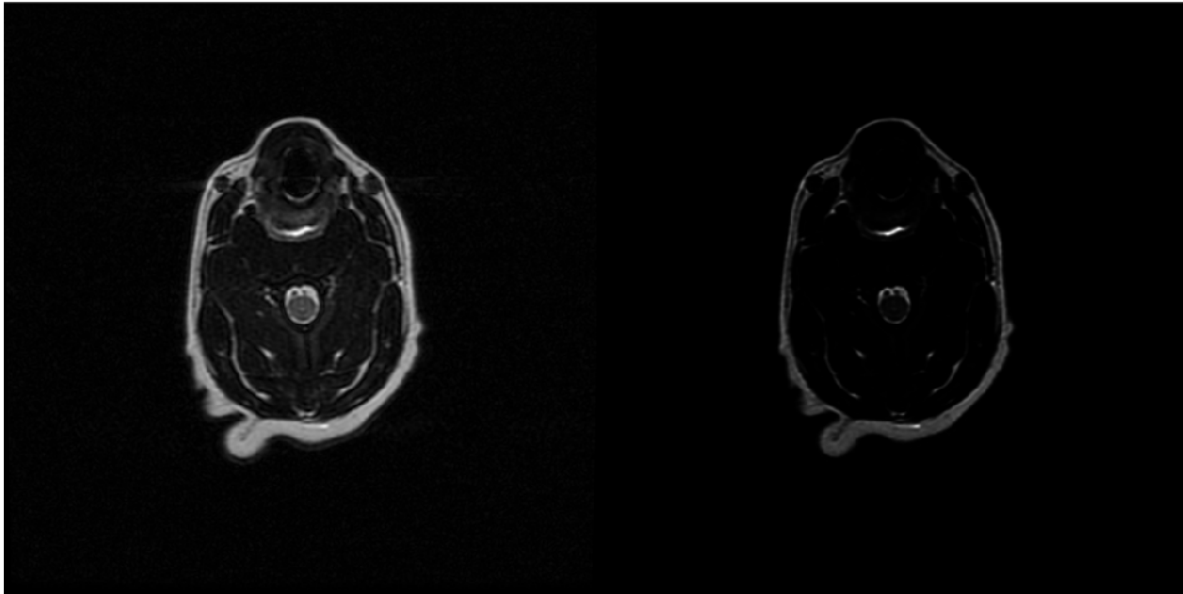
```
dicomDir = "dog";  
dicomds = imageDatastore(dicomDir,FileExtensions=".dcm",ReadFcn=@(x)dicomread(x));
```

Transform the datastore by gamma-correcting the intensity of the images with a random gamma value.

```
jitterds = transform(dicomds,@(x)jitterIntensity(im2single(x),Gamma=[2 3]));
```

Visualize the original and augmented datastore images.

```
dicomImage = read(dicomds);  
jitterImage = read(jitterds);  
figure  
imshowpair(dicomImage,jitterImage,"montage")
```



## Input Arguments

### **I** — Grayscale image or intensity volume

2-D numeric matrix | 3-D numeric array

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

The function does not support 3-D RGB images and 4-D RGB volumes. For color augmentation of 3-D RGB images, see `jitterColorHSV`.

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

### Name-Value 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: `J = jitterIntensity(I,Brightness=0.75,Contrast=3,Gamma=[2 3])` augments the intensity of `I` by randomly selecting parameters from the ranges specified in the name-value arguments, and returns the augmented image `J`.



**Brightness — Brightness range**

0 (default) | scalar in the range [0,1] | two-element vector in the range [-1, 1]

Brightness range, specified as a scalar in the range [0,1] or a two-element vector with elements in the range [-1, 1]. If you specify this value as a scalar,  $b$ , `jitterIntensity` shifts the intensity of the image or volume by a randomly selected value from the range  $[-b, b]$ . If you specify this value as a vector,  $[b_1 b_2]$ , `jitterIntensity` shifts the intensity of the image or volume by a randomly selected value from the range  $[b_1, b_2]$ . You must specify values for  $b_1$  and  $b_2$  such that  $b_2 \geq b_1$ .

Example: `Brightness=0.75` shifts the intensity by a randomly selected value from the range  $[-0.75, 0.75]$ .

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

**Contrast — Contrast range**

0 (default) | nonnegative scalar | two-element nonnegative vector

Contrast range, specified as a nonnegative scalar or two-element nonnegative vector. If you specify this value as a scalar,  $c$ , `jitterIntensity` scales the intensity of the image or volume by a randomly selected value from the range  $[min, 1+c]$ , where  $min$  is the higher of  $1-c$  and 0. If you specify the value as a vector,  $[c_1 c_2]$ , `jitterIntensity` scales the intensity of the image or volume by a randomly selected value from the range  $[c_1, c_2]$ .

Example: `Contrast=3` scales the intensity by a randomly selected value from the range  $[0, 4]$ .

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

**Gamma — Gamma range**

0 (default) | nonnegative scalar | two-element nonnegative vector

Gamma range, specified as a nonnegative scalar or two-element nonnegative vector. If you specify the value as a scalar,  $g$ , `jitterIntensity` gamma-corrects the intensity of the image or volume with a randomly selected gamma value from the range  $[min, 1+g]$ , where  $min$  is the higher of  $1-g$  and 0. If you specify the value as a vector,  $[g_1 g_2]$ , `jitterIntensity` gamma-corrects the intensity of the image or volume with a randomly selected gamma value from the range  $[g_1, g_2]$ .

Example: `Gamma=[2 3]` gamma-corrects the intensity with a randomly selected gamma value from the range  $[2, 3]$ .

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

**Output Arguments****J — Jittered image or volume**

2-D numeric matrix | 3-D numeric array

Jittered image or volume, returned as a numeric matrix or array of the same size and data type as the input image or volume `I`. If the input image or volume `I` is of the data type `single` or `double`, the function rescales the intensities in `J` to the range  $[0, 1]$ .

**Version History**

Introduced in R2022b

**See Also**

`jitterColorHSV`

# nrrdinfo

Read metadata from NRRD file

## Syntax

```
info = nrrdinfo(filename)
```

## Description

`info = nrrdinfo(filename)` reads the metadata from the nearly raw raster data (NRRD) image file `filename` and returns the metadata structure `info`.

## Examples

### Read Metadata from NRRD File

Read the metadata from an NRRD format file. The file is part of a data set containing the 3-D CT and MRI scans from The Retrospective Image Registration Evaluation (RIRE) Dataset, converted to the NRRD file format. The original data set was provided by Dr. Michael Fitzpatrick. For more information, see the RIRE Project homepage. The size of the entire data set is approximately 35 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", ...
    "MedicalRegistrationNRRDdata.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

Specify the name of the NRRD file to read.

```
filename = fullfile(filepath,"supportfilesNRRD","Patient007CT.nrrd");
```

Read the metadata from `filename`.

```
info = nrrdinfo(filename);
```

The `ImageSize` metadata field contains the image volume size, in pixels.

```
info.ImageSize
```

```
ans = 1×3
```

```
    512    512    28
```

The `PixelDimensions` metadata field contains the voxel size in each dimension, in real world units such as millimeters.

```
info.PixelDimensions
```

```
ans = 1×3
```

```
0.6536    0.6536    4.0000
```

The `SpatialMapping` metadata field contains an `affinetform3d` object that defines the transformation between the intrinsic and patient coordinate systems for the volume.

```
info.SpatialMapping
```

```
ans =  
  affinetform3d with properties:  
  
    Dimensionality: 3  
           A: [4x4 double]
```

The `RawAttributes` metadata field contains the raw metadata extracted from the NRRD header information.

```
info.RawAttributes
```

```
ans = struct with fields:  
    dimension: '3'  
    sizes: '512 512 28 '  
    type: 'float'  
    encoding: 'raw'  
    endian: 'little'  
    spacedimension: '3'  
    spaceorigin: '(1.653595, 1.653595, 5.000000)'  
    spacedirections: '(0.653595,0.000000,0.000000) (0.000000,0.653595,0.000000) (0.000000,0.000000,0.000000)'
```

## Input Arguments

**filename** — Name of NRRD file

string scalar | character vector

Name of the NRRD file, specified as a string scalar or a character vector. The file must contain a valid NRRD file header. The `nrrdinfo` function supports NRRD files containing header and image data (`.nrrd`) as well as detached header files (`.nhdr`). Specify `filename` as the absolute path to the file, a relative path from the current directory, or a relative path from a directory on the MATLAB path.

Data Types: `char` | `string`

## Output Arguments

**info** — NRRD metadata

structure

NRRD metadata, returned as a structure.

## Version History

Introduced in R2022b

**See Also**

nrrdread | isnrrd

## nrrdread

Read NRRD image

### Syntax

```
V = nrrdread(filename)
```

### Description

`V = nrrdread(filename)` reads the nearly raw raster data (NRRD) image file specified by `filename`, and returns the volumetric image data `V`.

### Examples

#### Read Image Data from NRRD File

Read the image data from an NRRD format file. The file is part of a data set containing the 3-D CT and MRI scans from The Retrospective Image Registration Evaluation (RIRE) Dataset, converted to the NRRD file format. The original data set was provided by Dr. Michael Fitzpatrick. For more information, see the RIRE Project homepage. The size of the entire data set is approximately 35 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", ...  
    "MedicalRegistrationNRRDdata.zip");  
filepath = fileparts(zipFile);  
unzip(zipFile,filepath)
```

Specify the name of the NRRD file to read.

```
filename = fullfile(filepath,"supportfilesNRRD","Patient007CT.nrrd");
```

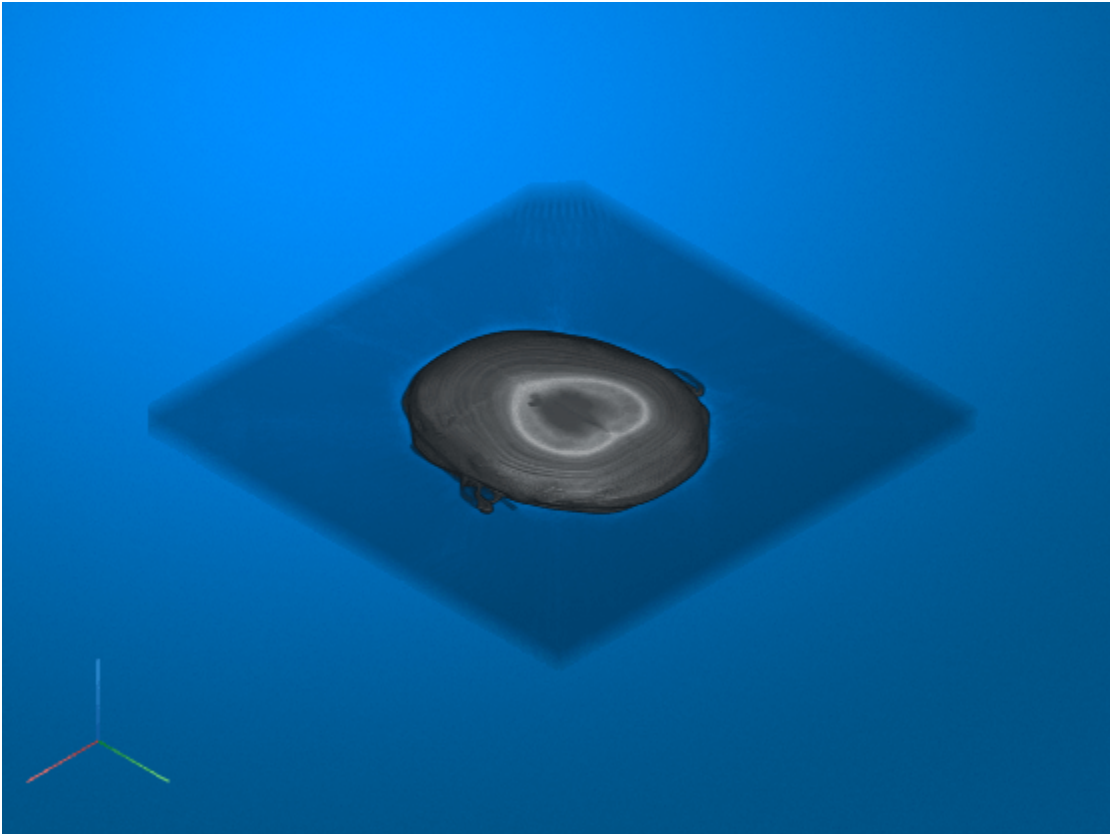
Read the image data from `filename`. The image data, `V`, is a 3-D array of intensity values.

```
V = nrrdread(filename);  
whos V
```

Name	Size	Bytes	Class	Attributes
V	512x512x28	29360128	single	

Display the image data stored in the array `V`.

```
volshow(V);
```



## Input Arguments

### **filename** — Name of NRRD file

string scalar | character vector

Name of the NRRD file, specified as a string scalar or a character vector. The file must be a valid NRRD file (.nrrd) that contains header information and image data. Specify `filename` as the absolute path to the file, a relative path from the current directory, or a relative path from a directory on the MATLAB path.

Data Types: `char` | `string`

## Output Arguments

### **V** — Volumetric image data

numeric array

Volumetric image data, returned as a numeric array.

## Version History

Introduced in R2022b

**See Also**

nrrdinfo | isnrrd



# getAttribute

Get value of specified DICOM attribute

## Syntax

```
data = getAttribute(dFile,name)
data = getAttribute(dFile,group,element)
```

## Description

`data = getAttribute(dFile,name)` returns the value `data` of the attribute name from the DICOM file specified by the `dicomFile` object `dFile`.

`data = getAttribute(dFile,group,element)` specifies the attribute to get as a DICOM group number `group` and element number `element`.

## Examples

### Find Modality of DICOM File

Import a DICOM file into the workspace. The DICOM file is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
datapath = fullfile(filepath,"MedicalVolumeDICOMData/LungCT01/CT000000.dcm");
dFile = dicomFile(datapath);
```

Get the value of the Modality attribute of the DICOM file.

```
modality = getAttribute(dFile,"Modality")
```

```
modality =
'CT'
```

Use the group and element numbers of the Modality attribute, in decimal form, to get its value from the DICOM file.

```
modality = getAttribute(dFile,8,96)
```

```
modality =
'CT'
```

Use the group and element numbers of the Modality attribute, in hexadecimal form, to get its value from the DICOM file.

```
modality = getAttribute(dFile,"0008","0060")
```

```
modality =
'CT'
```

## Input Arguments

### **dFile** — DICOM file

dicomFile object

DICOM file from which to retrieve the attribute value, specified as a `dicomFile` object.

### **name** — Name of attribute

string scalar | character vector

Name of the attribute, specified as a string scalar or character vector. This argument must exactly match the DICOM attribute name, and is case sensitive. You can use the property `AttributeNames` of the `dicomFile` object, that contains the list of attributes present in the DICOM file, to look for the name of the attribute.

Data Types: `char` | `string`

### **group** — Group number of attribute

numeric scalar | string scalar | character vector

Group number of the attribute, specified in decimal form as a numeric scalar or in the hexadecimal form as a string scalar or character vector. For more information on the group numbers of DICOM attributes, see Registry of DICOM Data Elements.

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

### **element** — Element number of attribute

numeric scalar | string scalar | character vector

Element number of the attribute, specified in decimal form as a numeric scalar or in the hexadecimal form as a string scalar or character vector. For more information on the element numbers of DICOM attributes, see Registry of DICOM Data Elements.

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

## Output Arguments

### **data** — Value of DICOM attribute

string scalar | character vector | numeric scalar | numeric vector | structure

Value of the DICOM attribute, returned as a string scalar, character vector, numeric scalar, numeric vector, or structure, depending upon the value representation (VR) of the specified DICOM attribute.

## Version History

Introduced in R2023a

### See Also

`dicomFile` | `isAttribute` | `getPixelData` | `write` | `dicominfo`

**External Websites**

Registry of DICOM Data Elements

## getPixelData

Get pixel data of DICOM file

### Syntax

```
pixelData = getPixelData(dFile)  
pixelData = getPixelData(dFile,Rescale=rescale)
```

### Description

`pixelData = getPixelData(dFile)` gets the pixel data `pixelData` of the DICOM file specified by the `dicomFile` object `dFile`.

`pixelData = getPixelData(dFile,Rescale=rescale)` specifies whether to rescale the pixel data. By default, the function rescales the pixel data.

### Examples

#### Get Pixel Data of DICOM File

Import a DICOM file into the workspace. The DICOM file is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

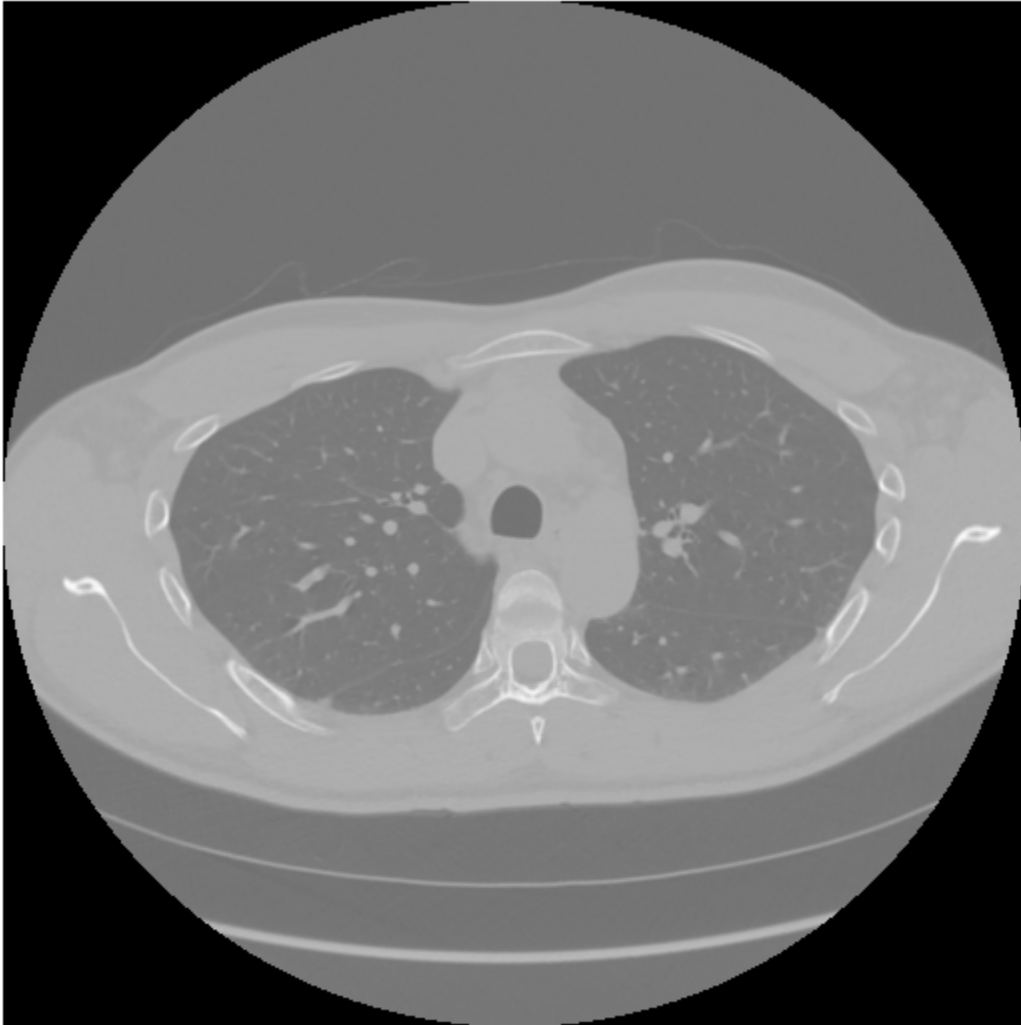
```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeDICOMData.zip");  
filepath = fileparts(zipFile);  
unzip(zipFile,filepath)  
datapath = fullfile(filepath, "MedicalVolumeDICOMData/LungCT01/CT000000.dcm");  
dFile = dicomFile(datapath);
```

Get the raw pixel data of the DICOM file.

```
rawPixelData = getPixelData(dFile,Rescale=false);
```

Visualize the raw pixel data.

```
figure  
imshow(rawPixelData,[])
```



Get the rescaled pixel data of the DICOM file.

```
rescaledPixelData = getPixelData(dFile);
```

Visualize the rescaled pixel data.

```
figure  
imshow(rescaledPixelData, [])
```



Observe that rescaling changes the range of the pixel data.

```
[min(rowPixelData,[],"all") max(rowPixelData,[],"all")]
```

```
ans = 1x2 int16 row vector
```

```
-2000    2520
```

```
[min(rescaledPixelData,[],"all") max(rescaledPixelData,[],"all")]
```

```
ans = 1x2 int16 row vector
```

```
-3024    1496
```

## Input Arguments

### **dFile — DICOM file**

dicomFile object

DICOM file from which to retrieve the pixel data, specified as a dicomFile object.

### **rescale — Rescale pixel data**

true or 1 (default) | false or 0

Rescale pixel data, specified as a logical 1 (true) or 0 (false). If rescale is true, the getPixelData function rescales the pixel data using the rescaleSlope and rescaleIntercept attributes of the DICOM file. Rescaling linearly transforms the on-disk representation of the pixel data to its in-memory representation. For example, for CT images, rescaling transforms the on-disk representation of intensity values to the standard Hounsfield units in the in-memory representation. If the rescaleSlope and rescaleIntercept attributes are not present in the DICOM file or if you specify this argument as false, the function returns the raw pixel data. You can choose to get the raw pixel data if you want to modify the pixel data and write it to a new DICOM file, to prevent repeated rescaling when reading pixel data from the new DICOM file.

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

## Output Arguments

### **pixelData — Pixel data of DICOM file**

2-D numeric matrix | 3-D numeric array

Pixel data of the DICOM file, returned as a 2-D numeric matrix or 3-D numeric array.

## Version History

Introduced in R2023a

## See Also

dicomFile | isAttribute | getAttribute | write | dicomread

## isAttribute

Check if specified attribute is present in DICOM file

### Syntax

```
TF = isAttribute(dFile,name)
TF = isAttribute(dFile,group,element)
```

### Description

`TF = isAttribute(dFile,name)` returns a logical `1` (`true`) if the attribute name is present in the DICOM file specified by the `dicomFile` object `dFile`. Otherwise, it returns a logical `0` (`false`).

`TF = isAttribute(dFile,group,element)` returns a logical `1` (`true`) if the attribute specified by the DICOM group number `group` and element number `element` is present in the DICOM file specified by the `dicomFile` object `dFile`. Otherwise, it returns a logical `0` (`false`).

### Examples

#### Check If Modality Is Attribute of DICOM File

Import a DICOM file into the workspace. The DICOM file is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
datapath = fullfile(filepath,"MedicalVolumeDICOMData/LungCT01/CT000000.dcm");
dFile = dicomFile(datapath);
```

Use the attribute name to check if `Modality` is an attribute of the DICOM file.

```
TF = isAttribute(dFile,"Modality")
```

```
TF = logical
    1
```

Use the group and element numbers of the `Modality` attribute, in decimal form, to check if it is an attribute of the DICOM file.

```
TF = isAttribute(dFile,8,96)
```

```
TF = logical
    1
```

Use the group and element numbers of the `Modality` attribute, in hexadecimal form, to check if it is an attribute of the DICOM file.



```
TF = isAttribute(dFile, "0008", "0060")
```

```
TF = logical  
    1
```

## Input Arguments

### **dFile** — DICOM file

dicomFile object

DICOM file in which to search for the attribute, specified as a `dicomFile` object. The attribute to be searched, may or may not be a property of the `dicomFile` object.

### **name** — Name of attribute

string scalar | character vector

Name of the attribute, specified as a string scalar or character vector. This argument must exactly match the DICOM attribute name, and is case sensitive. You can use the property `AttributeNames` of the `dicomFile` object, that contains the list of attributes present in the DICOM file, to look for the name of the attribute.

Data Types: `char` | `string`

### **group** — Group number of attribute

numeric scalar | string scalar | character vector

Group number of the attribute, specified in decimal form as a numeric scalar or in the hexadecimal form as a string scalar or character vector. For more information on the group numbers of DICOM attributes, see Registry of DICOM Data Elements.

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

### **element** — Element number of attribute

numeric scalar | string scalar | character vector

Element number of the attribute, specified in decimal form as a numeric scalar or in the hexadecimal form as a string scalar or character vector. For more information on the element numbers of DICOM attributes, see Registry of DICOM Data Elements.

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

## Version History

**Introduced in R2023a**

### See Also

`dicomFile` | `getAttribute` | `getPixelData` | `write`

### External Websites

Registry of DICOM Data Elements

## write

Create selectively modified copy of DICOM file

### Syntax

```
write(dFile, filename)
write(dFile, filename, pixelData)
write(dFile, filename, metadata)
write(dFile, filename, pixelData, metadata)
```

### Description

`write(dFile, filename)` writes the contents of the `dicomFile` object `dFile` to the new DICOM file specified by `filename`.

`write(dFile, filename, pixelData)` writes `pixelData` as the pixel data of the new DICOM file.

`write(dFile, filename, metadata)` modifies the attributes of the new DICOM file as specified by `metadata`.

`write(dFile, filename, pixelData, metadata)` modifies the pixel data and the attributes of the new DICOM file as specified by `pixelData` and `metadata`, respectively. You can specify `pixelData` and `metadata` in any order.

### Examples

#### Create Selectively Modified Copy of DICOM File

Import a DICOM file into the workspace. The DICOM file is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile, filepath)
datapath = fullfile(filepath, "MedicalVolumeDICOMData/LungCT01/CT000000.dcm");
dFile = dicomFile(datapath);
```

Get the raw pixel data of the DICOM file by specifying `rescale` as `false`.

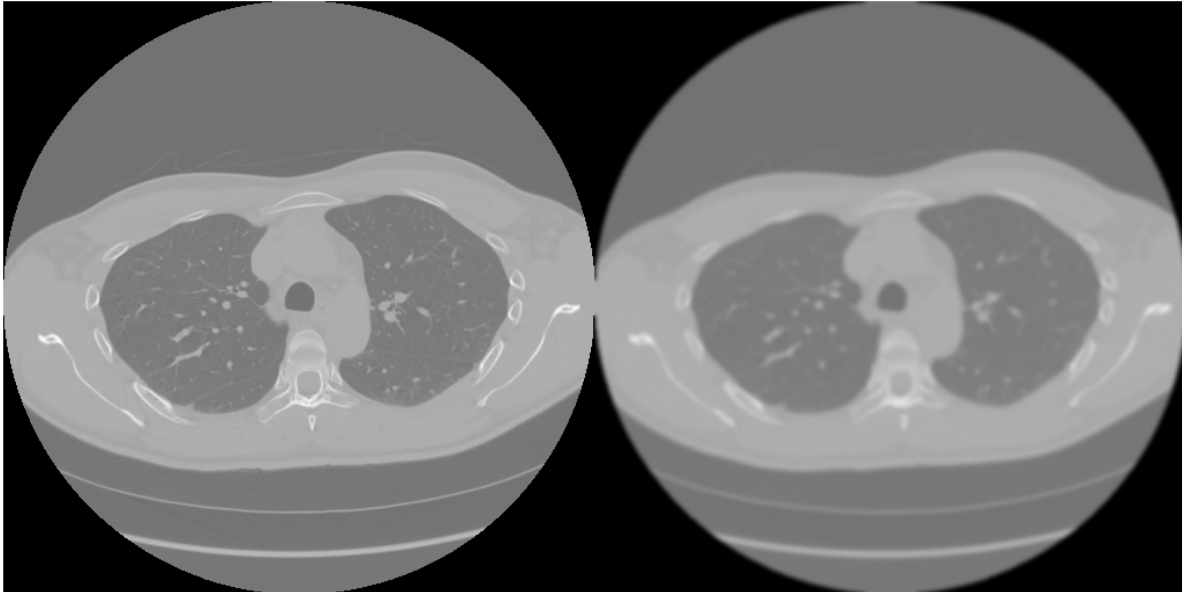
```
pixelData = getPixelData(dFile, rescale=false);
```

Filter the pixel data. The filtered pixel data is the pixel data for the new DICOM file. Making the modifications on the raw pixel data instead of the rescaled pixel data ensures that the pixel data is not rescaled repeatedly while getting pixel data from the new DICOM file.

```
filteredPixelData = imgaussfilt(pixelData, 2);
```

Visualize and compare the original and filtered pixel data.

```
figure
imshow([pixelData filteredPixelData],[])
```



Create a study description for the new DICOM file. Then, create a structure metadata with a `StudyDescription` field that contains your new study description.

```
newStudyDescription = 'CT CARDIAC CALCIUM SCORING FILTERED';
metadata = struct("StudyDescription",newStudyDescription)
```

```
metadata = struct with fields:
    StudyDescription: 'CT CARDIAC CALCIUM SCORING FILTERED'
```

Create a modified copy of the CT image DICOM file, replacing its pixel data with the filtered pixel data and its metadata with the structure containing your new study description.

```
write(dFile,"CT00000_filtered.dcm",filteredPixelData,metadata)
```

## Input Arguments

### **dFile** — DICOM file

dicomFile object

DICOM file from which to create the selectively modified copy, specified as a `dicomFile` object.

### **filename** — Name of modified DICOM file

string scalar | character vector

Name of the modified DICOM file, specified as a string scalar or character vector.

Data Types: `char` | `string`

**pixelData — Pixel data for modified DICOM file**

2-D numeric matrix | 3-D numeric array

Pixel data for the modified DICOM file, specified as a 2-D numeric matrix or 3-D numeric array.

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

**metadata — Metadata for modified DICOM file**

structure

Metadata for the modified DICOM file, specified as a structure. The names of the fields of the structure must match the names of the attributes to be modified.

Data Types: `struct`

## Version History

Introduced in R2023a

**See Also**

`dicomFile` | `isAttribute` | `getAttribute` | `getPixelData` | `dicomwrite`

# changeFilePaths

Change file paths in ground truth data for medical images

## Syntax

```
unresolvedFilePaths = changeFilePaths(gTruthMed,alternateFilePaths)
unresolvedFilePaths = changeFilePaths(gTruthMed,
alternateFilePaths,ChangeProperty=propertyName)
```

## Description

`unresolvedFilePaths = changeFilePaths(gTruthMed,alternateFilePaths)` changes the file paths stored in the `DataSource` and `LabelData` properties of the `groundTruthMedical` object `gTruthMed`. The `alternateFilePaths` argument specifies pairs of current paths in `gTruthMed` and alternative paths that point to the new data location on your machine. The function returns any unresolved paths in `unresolvedPaths`. An unresolved path is a current path not found in `gTruthMed` or an alternative path not found on your machine. In both cases, `unresolvedPaths` returns only the corresponding current path.

Use this function to update a `groundTruthMedical` object if you move the data source or label image files to a new folder. If you receive a `groundTruthMedical` object that was created on a different computer, use this function to point to the data location on your local machine.

`unresolvedFilePaths = changeFilePaths(gTruthMed,alternateFilePaths,ChangeProperty=propertyName)` replaces only the file paths stored in the specified property, `DataSource` or `LabelData`, of `gTruthMed`.

## Examples

### Change File Paths in Medical Ground Truth Data

Change the file paths in a medical ground truth data object to point to a subset of the Medical Segmentation Decathlon data set [1 on page 1-74]. The subset of data includes two CT chest volumes and corresponding label images, stored in the NIfTI file format. Download the `MedicalVolumNIfTIData.zip` file from the MathWorks® website, then unzip the file. The size of the data file is approximately 76 MB.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeNIfTIData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
dataFolder = fullfile(filepath,"MedicalVolumeNIfTIData");
```

Load a `groundTruthMedical` object containing ground truth data into the workspace. The data source and label data of the object specify file paths to the data at a different location than `dataFolder`. MATLAB® displays a warning that the path to the data source cannot be found. This can occur when a `groundTruthMedical` object is created on one computer and loaded on a different computer that stores the image data in a different location.

```
load("gTruthMed-NIfTI-Chest.MAT")
```

Warning: Unable to find one or more 'DataSource' or 'LabelData' files. Use the `changeFilePaths` object function.

Display the current path to the data source and label data.

```
gTruthMed.DataSource.Source
```

```
ans=2x1 cell array
    {"C:\CFP\lung_027.nii.gz"}
    {"C:\CFP\lung_043.nii.gz"}
```

```
gTruthMed.LabelData
```

```
ans = 2x1 string
    "C:\CFP\LabelData\lung_027.nii.gz"
    "C:\CFP\LabelData\lung_043.nii.gz"
```

Specify the current path to the data source and an alternative path and store these paths in a string array `alternativePaths`.

```
currentPathDataSource = "C:\CFP";
newPathDataSource = dataFolder;
alternativePaths = [currentPathDataSource newPathDataSource];
```

Use the `changeFilePaths` object function to update the data source file paths, based on the paths in the string array. Because the function resolves all paths, it returns an empty array of unresolved paths

```
unresolvedPaths = changeFilePaths(gTruthMed,alternativePaths)
```

```
unresolvedPaths =
    0x1 empty string array
```

Verify that the new data source and label data paths are stored in the `DataSource` and `LabelData` properties of `gTruthMed`.

```
gTruthMed.DataSource;
gTruthMed.LabelData;
```

## References

[1] Medical Segmentation Decathlon. "Lung." Tasks. Accessed May 10, 2018. <http://medicaldecathlon.com/>.

The Medical Segmentation Decathlon data set is provided under the CC-BY-SA 4.0 license. All warranties and representations are disclaimed. See the license for details.

## Input Arguments

**gTruthMed** — Ground truth data  
groundTruthMedical object

Ground truth data, specified as a `groundTruthMedical` object. You can export a `groundTruthMedical` object from the **Medical Image Labeler** app or create one programmatically by using the `groundTruthMedical` function.

### **alternateFilePaths** — Alternative file paths

*n*-by-2 string array

Alternative file paths, specified as an *n*-by-2 string array, where *n* is the number of paths to update. Each row specifies a current location and its corresponding new location in the format [ $p_{\text{current}}$   $p_{\text{new}}$ ].

- $p_{\text{current}}$  is a current file path in `gTruthMed`. Specify  $p_{\text{current}}$  using the same file separators, either forward slashes or backslashes, present in `gTruthMed`.
- $p_{\text{new}}$  is the alternative path replacing  $p_{\text{current}}$ . Specify  $p_{\text{new}}$  using either forward slashes or backslashes as the path separators.

The function updates the properties of `gTruthMed` based on the types of files specified:

- Data sources — The `DataSource` property of `gTruthMed` contains a `VolumeSource` or `ImageSource` object. The `changeFilePaths` function updates the paths in the `Source` property of these objects.
- Label data — The `changeFilePaths` function updates the paths in the `LabelData` property of `gTruthMed`.

You can specify paths as full path names to specific files, or as the beginning portion of a file path. If you specify a pair of partial paths, [ $p_{\text{currentPartial}}$   $p_{\text{newPartial}}$ ], the function determines the full current and alternate paths using these steps:

- 1 The function searches `gTruthMed` for file names that start with  $p_{\text{currentPartial}}$ . The final list of current paths consists of all full file paths starting with  $p_{\text{currentPartial}}$ .
- 2 The final list of alternative paths is the list of current full file paths with the portion  $p_{\text{currentPartial}}$  replaced with  $p_{\text{newPartial}}$ .

Example: [`"C:\VolumeData"`, `"C:\VolumeData2"`] changes the path of the data directory. The function updates the path for all files in `gTruthMed` in the folder `C:\VolumeData` and its subfolders to begin with `C:\VolumeData2`.

Example: [`"C:\VolumeData"`, `"B:\VolumeData"`] changes the drive letter in the file paths in `gTruthMed` that begin with `C:\VolumeData` from `C` to `B`.

Data Types: `string`

### **propertyName** — Property for which to change file paths

`"Auto"` (default) | `"DataSource"` | `"LabelData"`

Property for which to change file paths, specified as one of these values:

- `"Auto"` — Update file paths stored in both the `DataSource` and `LabelData` properties.
- `"DataSource"` — Update file paths stored in the `DataSource` property.
- `"LabelData"` — Update file paths stored in the `LabelData` property.

Data Types: `string`

## Output Arguments

### **unresolvedFilePaths — Unresolved file paths**

string array

Unresolved file paths, returned as a string array. If the function cannot find either the specified current path in `gTruth` or the specified alternative path location, then it returns the corresponding current path.

If the function finds and resolves all file paths, then it returns `unresolvedFilePaths` as an empty string array.

## Version History

**Introduced in R2022b**

### **See Also**

`groundTruthMedical`



## merge

Merge two or more `groundTruthMedical` objects

### Syntax

```
gTruthMerged = merge(gTruth1,gTruth2,...,gTruthn)
```

### Description

`gTruthMerged = merge(gTruth1,gTruth2,...,gTruthn)` merges two or more `groundTruthMedical` objects into a new `groundTruthMedical` object, `gTruthMerged`.

### Examples

#### Merge Medical Ground Truth Data

Merge two compatible `groundTruthMedical` objects present in the MATLAB workspace.

The ground truth objects, `gTruthMed1` and `gTruthMed2`, specify the volumetric data source and label image for the CT lung volumes, `lung_001.nii.gz` and `lung_003.nii.gz`, respectively.

```
gTruthMed1
```

```
gTruthMed1 =
```

```
groundTruthMedical with properties:
```

```
DataSource: [1x1 medical.labeler.loading.VolumeSource]
LabelData: "C:\Merge\Label Data\lung_001.nii.gz"
LabelDefinitions: [1x3 table]
```

```
gTruthMed2
```

```
gTruthMed2 =
```

```
groundTruthMedical with properties:
```

```
DataSource: [1x1 medical.labeler.loading.VolumeSource]
LabelData: "C:\Merge\Label Data\lung_003.nii.gz"
LabelDefinitions: [1x3 table]
```

Merge the `groundTruthMedical` objects. The merged object specifies both lung CT volumes.

```
gTruthMerged = merge(gTruthMed1,gTruthMed2)
```

```
gTruthMerged =
```

```
groundTruthMedical with properties:
```

```
DataSource: [1x1 medical.labeler.loading.VolumeSource]
```

```
LabelData: [2×1 string]  
LabelDefinitions: [1×3 table]
```

## Input Arguments

### **gTruth1, gTruth2, . . . , gTruthn — Ground truth data to merge**

`groundTruthMedical` objects

Ground truth data to merge, specified as a comma-separated list of two or more `groundTruthMedical` objects. You can export the `groundTruthMedical` objects from the **Medical Image Labeler** app or create them programmatically by using the `groundTruthMedical` function.

The specified `groundTruthMedical` objects must all have the same data source type, specified by the `DataSource` property. For example, if one object has a data source of type `VolumeSource`, all `groundTruthMedical` objects you merge it with must also have a data source of type `VolumeSource`.

You cannot merge `groundTruthMedical` objects that have one or more `DataSource` entries pointing to the same file location. Each `groundTruthMedical` object must specify a unique set of image sequences or image volumes

If the `LabelDefinition` properties of two or more objects specify the same label name, they must also specify the same color and pixel label ID for that label name. The label name, color, and pixel label ID are specified by the `LabelName`, `LabelColor`, and `PixelLabelID` columns of the `LabelDefinition` property.

## Output Arguments

### **gTruthMerged — Merged ground truth data**

`groundTruthMedical` object

Merged ground truth data, returned as a `groundTruthMedical` object. The `DataSource` property of `gTruthMerged` is the same as that of the input `groundTruthMedical` objects.

## Version History

**Introduced in R2022b**

### **See Also**

`groundTruthMedical`

## extractFrame

Extract pixel data for one frame of 2-D medical image series

### Syntax

```
X = extractFrame(medImage, frame)
```

### Description

`X = extractFrame(medImage, frame)` extracts the pixel data for the frame at index `frame` of the `medicalImage` object `medImage`.

### Examples

#### Extract Frame from Medical Image Series

Specify the name of an echocardiogram series stored as a DICOM file.

```
fileName = "heartUltrasoundSequence.dcm";
```

Create a medical image object for the echocardiogram series.

```
medImage = medicalImage(fileName);
```

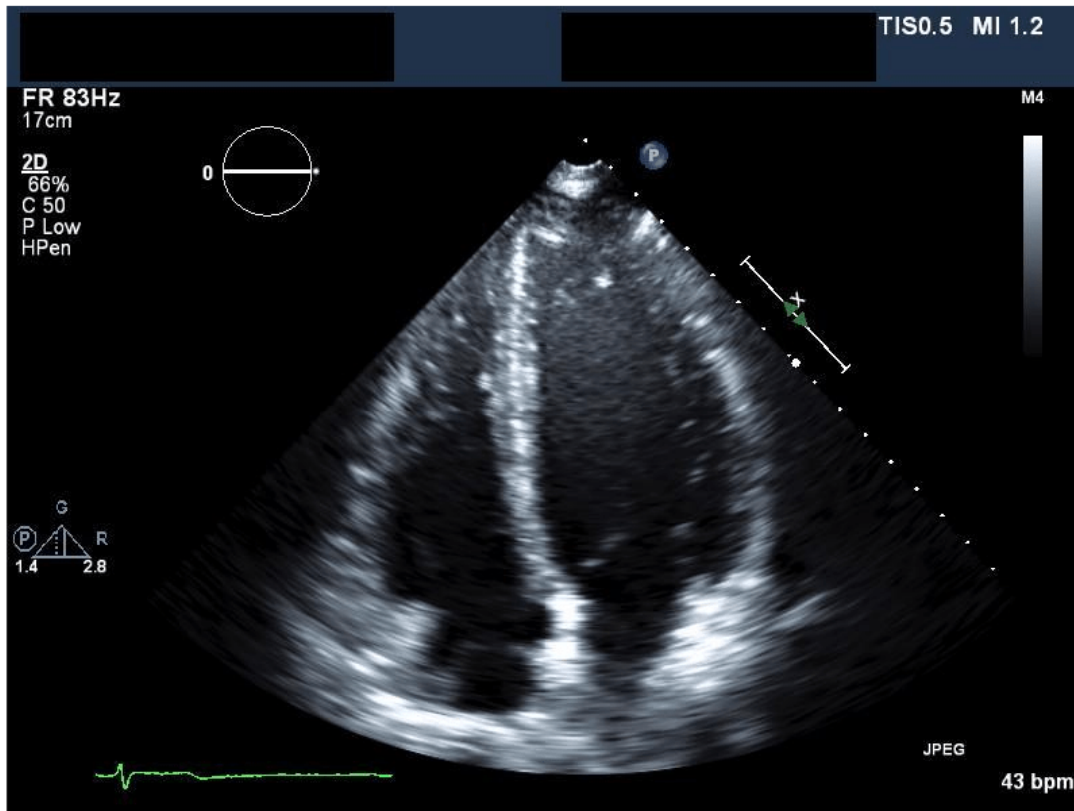
Check the `NumFrames` property value to determine the number of frames in the series. `medImage` contains three frames.

```
numFrames = medImage.NumFrames
```

```
numFrames = 3
```

Extract the second frame in the image series. Display the extracted frame image.

```
X = extractFrame(medImage,2);  
imshow(X)
```



## Input Arguments

### **medImage** – Medical image

medicalImage object

Medical image, specified as a medicalImage object.

### **frame** – Frame index

positive integer scalar in range [1, numFrames]

Frame index, specified as a positive integer scalar in the range [1, numFrames], where numFrames is the number of frames in the image or series of images in medImage. The number of frames is specified by the NumFrames property of medImage.

Data Types: double

## Output Arguments

### **X** – Pixel data of extracted frame

m-by-n numeric matrix

Pixel data of the extracted frame, returned as an  $m$ -by- $n$  numeric matrix, where  $m$  and  $n$  are the first two dimensions of the image frame extracted from `medImage`. The X output matrix is the same data type as the `Pixels` property of `medImage`.

## **Version History**

**Introduced in R2022b**

## **See Also**

`medicalImage`

## **Topics**

“Read, Process, and View Ultrasound Data”

# implay

View 2-D medical image series in Video Viewer app

## Syntax

```
implay(medImage)
```


```
implay(filename)  
implay(I)
```

```
implay( ___, fps)
```

## Description

Medical Imaging Toolbox™ extends the functionality of the `implay` (Image Processing Toolbox™) function to display a `medicalImage` object. The function uses the `medicalImage` properties to set the frame rate of the **Video Viewer** app. If you do not have Medical Imaging Toolbox installed, see **Video Viewer**.

### Medical Image Object

`implay(medImage)` opens the **Video Viewer** app and loads the image series in the `medicalImage` object `medImage`. The app displays the first frame of the image series. Click the Play button  to view the image series as a video. The app automatically sets the frame rate using the `FrameTime` property of `medImage`.

### Nonmedical Image Formats

`implay(filename)` opens the **Video Viewer** app and loads the content of the Audio Video Interleaved (AVI) file with filename `filename`.

`implay(I)` opens the **Video Viewer** app and displays the first frame in the multiframe image sequence `I`, specified as a numeric array or MATLAB movie structure.

### Additional Options

`implay( ___, fps)` specifies the frame rate `fps` in frames per second, in addition to any input argument from a previous syntax.

## Examples

### Play Medical Image Series as Video

Specify the filename of an echocardiogram ultrasound series. The DICOM file is attached to this example as a supporting file.

```
filename = "heartUltrasoundSequenceVideo.dcm";
```

Read the metadata and image data from the file by creating a `medicalImage` object. The `FrameTime` property indicates that each frame has a duration of 33.333 milliseconds. The `NumFrames` property indicates that the series has a total of 116 image frames.

```

medImg = medicalImage(filename)

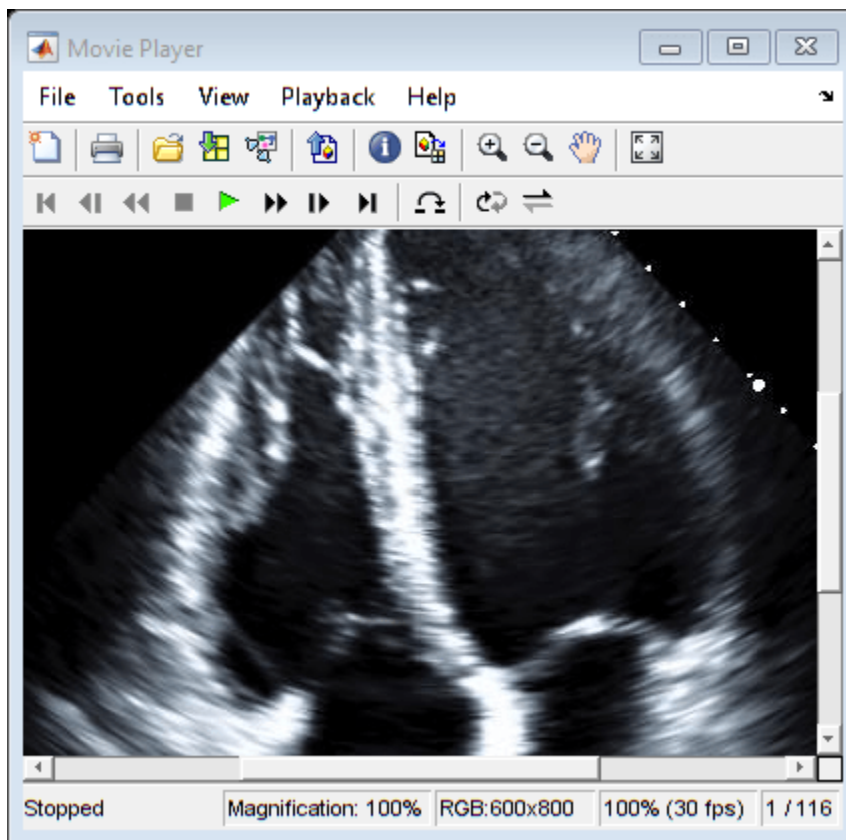
medImg =
    medicalImage with properties:

        Pixels: [600x800x116x3 uint8]
        Colormap: []
        SpatialUnits: "unknown"
        FrameTime: 33.3330
        NumFrames: 116
        PixelSpacing: [1 1]
        Modality: 'US'
        WindowCenter: []
        WindowWidth: []

```

View the ultrasound series as a video by using the Video Viewer app. By default, the app plays the video at the frame rate specified by the `FrameTime` property value.

```
implay(medImg)
```



## Input Arguments

**medImage** — Medical image  
 medicalImage object

Medical image, specified as a `medicalImage` object. The video plays at the frame rate specified by the `FrameTime` property value of `medImage`. If the `FrameTime` property is empty, the default frame rate is 20 fps.

**filename — Name of Audio Video Interleaved (AVI) file**

string scalar | character vector

Name of an Audio Video Interleaved (AVI) file, specified as a string scalar or character vector. The video plays at the frame rate specified by the file metadata. If the file metadata does not specify a frame rate, the default frame rate is 20 fps.

Data Types: `char` | `string`

**I — Multiframe image sequence**

numeric array | structure

Multiframe image sequence, specified as one of these values:

- An  $m$ -by- $n$ -by- $k$  numeric or logical array representing a grayscale or binary image sequence, respectively, of  $k$  frames.
- An  $m$ -by- $n$ -by-1-by- $k$  numeric or logical array representing a grayscale or binary image sequence, respectively, of  $k$  frames.
- 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`.

**fps — Frame rate in frames per second**

20 (default) | positive numeric scalar

Frame rate in frames per second, specified as a positive numeric scalar. If the input is a `medicalImage` object or AVI filename, with a frame rate specified by the `FrameTime` property or AVI file metadata, respectively, `implay` uses the specified frame rate. If the frame rate is not specified by the `FrameTime` property or file metadata, or if the input is a numeric array, the default value is 20 frames per second.

Data Types: `single` | `double`

## Version History

Introduced in R2023a

### See Also

`Video Viewer` | `medicalImage` | `montage`

### Topics

“Choose Approach for Medical Image Visualization”  
“Read, Process, and View Ultrasound Data”



# intrinsicToWorldMapping

Geometric transform between intrinsic and patient coordinates of medical image volume

## Syntax

```
tform = intrinsicToWorldMapping(R)
```

## Description

`tform = intrinsicToWorldMapping(R)` computes the geometric transformation `tform` between the intrinsic and patient coordinate systems for the medical image volume defined by `R`. If the volume specified by `R` is non-affine, then this function computes the transformation for only the first slice along the third dimension.

## Examples

### Get Geometric Transform Between Intrinsic and Patient Coordinate Systems

Get the geometric transformation between the intrinsic and patient coordinate systems for a chest CT volume saved as a directory of DICOM files. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile, filepath)
```

Specify the directory of the DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath, "MedicalVolumeDICOMData", "LungCT01");
```

Create a medical volume object that contains the image and spatial metadata for the CT volume.

```
medVol = medicalVolume(dataFolder);
```

The `VolumeGeometry` property of a medical volume object contains a `medicalref3d` object that specifies the spatial referencing for the volume. Extract the `medicalref3d` object for the chest CT.

```
R = medVol.VolumeGeometry;
```

Calculate the transformation that maps between the intrinsic and patient coordinate systems by using the `intrinsicToWorldMapping` object function. The function returns an `affinetform3d` object, `tform`.

```
tform = intrinsicToWorldMapping(R)
```

```
tform =
    affinetform3d with properties:
```

```
    Dimensionality: 3
```

```
A: [4x4 double]
```

The `A` property of the `affinetform3d` object contains a 4-by-4 geometric transformation matrix.

```
tform.A
```

```
ans = 4x4
```

```
      0      0.7285      0 -187.2285
0.7285      0      0      0 -187.2285
      0      0      2.5000 -283.7500
      0      0      0      1.0000
```

## Input Arguments

### **R** – Spatial referencing information

`medicalref3d` object

Spatial referencing information, specified as a `medicalref3d` object.

## Output Arguments

### **tform** – Geometric transformation

`affinetform3d` object

Geometric transformation, returned as an `affinetform3d` object. If the volume specified by `R` is affine, then `tform` describes the transformation between the intrinsic and patient coordinate systems for the entire volume. If the volume is non-affine, then `tform` describes the transformation between the intrinsic and patient coordinate systems for the points in the first slice of the volume along the third dimension.

An image volume is affine if these conditions are met:

- All slices are parallel to each other.
- The spacing between slices in each dimension is uniform.
- The upper-left voxels of all slices are collinear.
- No two slices are coincident, meaning no two slices are located at the same position in space.

The `A` property of `tform` contains a 4-by-4 3-D transformation matrix that maps triplets of voxel indices in the order (*column*, *row*, *slice*) to (*x*, *y*, *z*) triplets of patient coordinates in real-world units.

## Version History

**Introduced in R2022b**

## See Also

`medicalref3d` | `oneSliceIntrinsicToWorldMapping`

## Topics

“Display Medical Image Volume in Patient Coordinate System”

“Display Labeled Medical Image Volume in Patient Coordinate System”

## contains

Determine if affine image volume contains points specified in patient coordinate system

### Syntax

```
tf = contains(R,xyzWorld)
```

### Description

`tf = contains(R,xyzWorld)` returns a logical vector, `tf`, that indicates whether each of the specified 3-D coordinates points `xyzWorld` falls within the bounds of the affine medical image volume defined by `R`.

### Examples

#### Determine If Medical Image Volume Contains Patient Coordinates

Determine if a chest CT volume, saved as a directory of DICOM files, contains a set of coordinates specified in the patient coordinate system. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");  
filepath = fileparts(zipFile);  
unzip(zipFile,filepath);
```

Specify the directory of the DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData","LungCT01");
```

Create a medical volume object that contains the image and spatial metadata for the CT volume.

```
medVol = medicalVolume(dataFolder);
```

The `VolumeGeometry` property of the medical volume object contains a `medicalref3d` object that specifies the spatial referencing for the volume. Extract the `medicalref3d` object for the chest CT.

```
R = medVol.VolumeGeometry;
```

Specify the patient coordinates, in millimeters, of three sample points.

```
xyzWorld = [-100 -100 -200; 0 0 -100; 300 200 -80]
```

```
xyzWorld = 3x3
```

```
-100  -100  -200  
   0    0  -100  
  300   200  -80
```

Check whether the sample points are inside the image boundary. The values of `tf` indicate that the third point lies outside the image.

```
tf = contains(R,xyzWorld)
```

```
tf = 3×1 logical array
```

```
1  
1  
0
```

## Input Arguments

### **R** — Spatial referencing information

medicalref3d object

Spatial referencing information, specified as a `medicalref3d` object. R must specify an affine image volume. An image volume is affine if these conditions are met:

- All slices are parallel to each other.
- The spacing between slices in each dimension is uniform.
- The upper-left voxels of all slices are collinear.
- No two slices are coincident, meaning no two slices are located at the same position in space.

### **xyzWorld** — Patient coordinates of points to query

*n*-by-3 numeric matrix

Patient coordinates of points to query, specified as an *n*-by-3 numeric matrix, where *n* is the number of points. The patient coordinates are in real-world units defined by the patient coordinate system.

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

## Output Arguments

### **tf** — Image volume contains specified points

*n*-element logical vector

Image volume contains the specified points, returned as an *n*-element logical vector, where *n* is the number of points. A value of 1 (`true`) indicates that the corresponding `xyzWorld` point exists in the image volume, and a value of 0 (`false`) indicates that it does not.

## Version History

Introduced in R2022b

### See Also

`medicalref3d`

## intrinsicToWorld

Map points from intrinsic coordinates to patient coordinates

### Syntax

```
[X,Y,Z] = intrinsicToWorld(R,I,J,K)
```

### Description

`[X,Y,Z] = intrinsicToWorld(R,I,J,K)` maps points from the intrinsic coordinate system to the patient coordinate system using the spatial referencing information, `R`. The intrinsic coordinates `I`, `J`, and `K` are defined by axes aligned with the row, column, and slice subscripts of the image data array, respectively. The patient coordinates `X`, `Y`, and `Z` are defined by the patient coordinate system axes.

### Examples

#### Map 3-D Intrinsic Coordinates to Patient Coordinates

Map 3-D intrinsic coordinates from a chest CT volume, saved as a directory of DICOM files, to patient coordinates. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");  
filepath = fileparts(zipFile);  
unzip(zipFile,filepath)
```

Specify the directory of the DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData","LungCT01");
```

Create a medical volume object that contains the image and spatial metadata for the CT volume.

```
medVol = medicalVolume(dataFolder);
```

The `VolumeGeometry` property of the medical volume object contains a `medicalref3d` object that specifies the spatial referencing for the volume. Extract the `medicalref3d` object for the chest CT.

```
R = medVol.VolumeGeometry;
```

Select three sample points, and store their  $(i, j, k)$  intrinsic coordinates, in voxels. The points must fall within the image boundary.

```
I = [54 200 512];  
J = [46 48 79];  
K = [1 13 88];
```

Convert the intrinsic coordinates to patient coordinates. The output vectors provide the  $(x, y, z)$  patient coordinates, in millimeters.

```
[X,Y,Z] = intrinsicToWorld(R,I,J,K)
```

$X = 1 \times 3$

-147.8887 -41.5253 185.7717

$Y = 1 \times 3$

-153.7168 -152.2597 -129.6758

$Z = 1 \times 3$

-281.2500 -251.2500 -63.7500

## Input Arguments

### **R — Spatial referencing information**

medicalref3d object

Spatial referencing information, specified as a `medicalref3d` object.

### **I — Coordinates along *i*-dimension in intrinsic coordinate system**

numeric array

Coordinates along the *i*-dimension in the intrinsic coordinate system, specified as a numeric array. The *i*-axis is aligned with the first dimension of the spatial volume specified by R.

I, J, and K must be the same size.

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

### **J — Coordinates along *j*-dimension in intrinsic coordinate system**

numeric array

Coordinates along the *j*-dimension in the intrinsic coordinate system, specified as a numeric array. The *j*-axis is aligned with the second dimension of the spatial volume specified by R.

I, J, and K must be the same size.

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

### **K — Coordinates along *k*-dimension in intrinsic coordinate system**

numeric array

Coordinates along the *k*-dimension in the intrinsic coordinate system, specified as a numeric array. The *k*-axis is aligned with the third dimension of the spatial volume specified by R.

I, J, and K must be the same size.

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

## Output Arguments

### **X — Coordinates along *x*-dimension in patient coordinate system**

numeric array

Coordinates along the x-dimension in the patient coordinate system, returned as a numeric array. X is the same size as I.

Data Types: double

**Y – Coordinates along y-dimension in patient coordinate system**

numeric array

Coordinates along the y-dimension in the patient coordinate system, returned as a numeric array. Y is the same size as I.

Data Types: double

**Z – Coordinates along z-dimension in patient coordinate system**

numeric array

Coordinates along the z-dimension in the patient coordinate system, returned as a numeric array. Z is the same size as I.

Data Types: double

## **Version History**

**Introduced in R2022b**

### **See Also**

`medicalref3d` | `worldToIntrinsic` | `worldToSubscript`

### **Topics**

“Create STL Surface Model of Femur Bone for 3-D Printing”



# oneSliceIntrinsicToWorldMapping

Geometric transform between intrinsic and patient coordinates of medical image volume slice

## Syntax

```
tform = oneSliceIntrinsicToWorldMapping(R,slice)
```

## Description

`tform = oneSliceIntrinsicToWorldMapping(R,slice)` computes the geometric transformation, `tform`, between the intrinsic and patient coordinate systems for one slice of the medical image volume defined by `R`. The output `tform` maps the geometric transformation for the specified slice `slice` along the third dimension.

## Examples

### Get Mapping Between Intrinsic and Patient Coordinate Systems for One Slice

Get the geometric transform between the intrinsic and patient coordinate systems for one slice of a chest CT volume, saved as a directory of DICOM files. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

Specify the directory of the DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData","LungCT01");
```

Create a medical volume object that contains the image and spatial metadata for the CT volume.

```
medVol = medicalVolume(dataFolder);
```

The `VolumeGeometry` property of the medical volume object contains a `medicalref3d` object that specifies the spatial referencing for the volume. Extract the `medicalref3d` object for the chest CT.

```
R = medVol.VolumeGeometry;
```

Calculate the transformation that maps between the intrinsic and patient coordinate systems for the 20th slice along the third dimension of the volume. The `oneSliceIntrinsicToWorldMapping` function returns an `affinetform3d` object, `tform`.

```
tform = oneSliceIntrinsicToWorldMapping(R,20)
```

```
tform =
    affinetform3d with properties:
```

```
    Dimensionality: 3
```

A: [4×4 double]

The `A` property of the `affinetform3d` object contains a 4-by-4 geometric transformation matrix.

`tform.A`

ans = 4×4

```
      0      0.7285      0 -187.2285
0.7285      0      0 -187.2285
      0      0 2.5000 -236.2500
      0      0      0  1.0000
```

## Input Arguments

### **R** — Spatial referencing information

`medicalref3d` object

Spatial referencing information, specified as a `medicalref3d` object. The volume specified by `R` can be affine or non-affine.

### **slice** — Slice index

positive integer in range [1, *p*]

Slice index, specified as a positive integer in the range [1, *p*], where *p* is the number of slices in the image volume along the third dimension.

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

## Output Arguments

### **tform** — Geometric transformation

`affinetform3d` object

Geometric transformation, returned as an `affinetform3d` object.

The `A` property of `tform` contains a 4-by-4 3-D transformation matrix that maps triplets of pixel indices in the order (*column*, *row*, *slice*) to (*x*, *y*, *z*) triplets of patient coordinates in real-world units.

## Version History

Introduced in R2022b

### See Also

`medicalref3d` | `intrinsicToWorldMapping`

# orient

Update patient coordinate system convention

## Syntax

```
orientR = orient(R,targetSpace)
```

## Description

`orientR = orient(R,targetSpace)` updates the patient coordinate system convention of the spatial referencing information `R` to the specified convention `targetSpace` and returns an equivalent spatial referencing information object, `orientR`, that uses the specified patient coordinate system convention. Use this function to set or update the patient coordinate system orientation, such as from LPS+ to RAS+.

## Examples

### Update Orientation of Medical Spatial Referencing Object

Update the orientation of the medical spatial referencing object for a chest CT volume saved as a directory of DICOM files. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

Specify the directory of the DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData","LungCT01");
```

Create a medical volume object that contains the image and spatial metadata for the CT volume.

```
medVol = medicalVolume(dataFolder);
```

The `VolumeGeometry` property of the medical volume object contains a `medicalref3d` object that specifies the spatial referencing for the volume. Extract the `medicalref3d` object for the chest CT.

```
R = medVol.VolumeGeometry
```

```
R =
```

```
medicalref3d with properties:
```

```

    VolumeSize: [512 512 88]
      Position: [88x3 double]
  VoxelDistances: {[88x3 double] [88x3 double] [88x3 double]}
PatientCoordinateSystem: "LPS+"
    PixelSpacing: [88x2 double]
      IsAffine: 1
    IsAxesAligned: 1
```

```
IsMixed: 0
```

The `PatientCoordinateSystem` property of the `medicalref3d` object specifies the patient coordinate system orientation. The initial orientation, based on the DICOM file metadata, is LPS+. An LPS+ orientation indicates that the positive  $x$ -,  $y$ -, and  $z$ -axes of the patient coordinate system point in the left, posterior, and superior directions, respectively.

```
R.PatientCoordinateSystem
```

```
ans =  
"LPS+"
```

Update the patient coordinate system orientation to the RAS+ orientation. An RAS+ orientation indicates that the positive  $x$ -,  $y$ -, and  $z$ -axes of the patient coordinate system point in the right, anterior, and superior directions, respectively. The `orient` object function returns a new `medicalref3d` object, `orientR`. In addition to the `PatientCoordinateSystem` property, the `Position` and `VoxelDistances` property values reflect the new orientation of `orientR`.

```
orientR = orient(R, "RAS+")
```

```
orientR =  
medicalref3d with properties:
```

```
    VolumeSize: [512 512 88]  
    Position: [88x3 double]  
    VoxelDistances: {[-0.7285 0 0] [0 -0.7285 0] [0 0 2.5000]}  
    PatientCoordinateSystem: "RAS+"  
    PixelSpacing: [0.7285 0.7285]  
    IsAffine: 1  
    IsAxesAligned: 1  
    IsMixed: 0
```

## Input Arguments

### R — Spatial referencing information

`medicalref3d` object

Spatial referencing information, specified as a `medicalref3d` object.

### targetSpace — Target orientation convention of the patient coordinate system

"LPS+" | "LAS+" | "RAS+"

Target orientation convention of the patient coordinate system, specified as "LPS+", "LAS+", or "RAS+". The first three characters indicate the positive direction of the  $x$ -,  $y$ -, and  $z$ -axes of the patient coordinate system, respectively.

- The positive direction of the  $x$ -axis points left ("L") or right ("R").
- The positive direction of the  $y$ -axis points anterior ("A") or posterior ("P").
- The positive direction of the  $z$ -axis points inferior ("I") or superior ("S").
- "+" indicates that values increase in the stated direction.

For example, "LPS+" specifies a patient coordinate system with the  $x$ -,  $y$ -, and  $z$ -axes positive in the left, posterior, and superior directions, respectively.

Data Types: char | string

## Output Arguments

### **orientR — Spatial referencing information with target patient coordinate system convention**

`medicalref3d` object

Spatial referencing information with the target patient coordinate system convention, returned as a `medicalref3d` object. Which properties of this object the `orient` function updates from R depends on the `PatientCoordinateSystem` property of R.

- If `PatientCoordinateSystem` is "unknown", then `orient` updates only the `PatientCoordinateSystem` of `orientR`.
- If `PatientCoordinateSystem` is not "unknown", then `orient` updates the `VoxelDistances` and `Position` properties of `orientR`, in addition to the `PatientCoordinateSystem` property, to maintain the correct mapping between the intrinsic and patient coordinate system axes.

## Version History

Introduced in R2022b

### See Also

`medicalref3d`

## sliceCorners

Extract patient coordinates of corner voxels for one slice

### Syntax

```
xyzCorners = sliceCorners(R,slice)
```

### Description

`xyzCorners = sliceCorners(R,slice)` extracts the *xyz*-coordinates of the four corner voxels for one slice of an image volume.

### Examples

#### Extract Corner Coordinates for One Slice of Medical Spatial Referencing Object

Extract the corner coordinates for one slice of the medical spatial referencing object of a chest CT volume, saved as a directory of DICOM files. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");  
filepath = fileparts(zipFile);  
unzip(zipFile,filepath)
```

Specify the directory of the DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData","LungCT01");
```

Create a medical volume object that contains the image and spatial metadata for the CT volume.

```
medVol = medicalVolume(dataFolder);
```

The `VolumeGeometry` property of the medical volume object contains a `medicalref3d` object that specifies the spatial referencing for the volume. Extract the `medicalref3d` object for the chest CT.

```
R = medVol.VolumeGeometry;
```

Extract the *xyz*-coordinates, in millimeters, of the corner voxels for the first slice along the third dimension of the volume.

```
xyzCorners = sliceCorners(R,1)
```

```
xyzCorners = 4×3
```

```
-186.5000 -186.5000 -281.2500  
-186.5000 185.7717 -281.2500  
185.7717 185.7717 -281.2500  
185.7717 -186.5000 -281.2500
```

## Input Arguments

### **R** — Spatial referencing information

`medicalref3d` object

Spatial referencing information, specified as a `medicalref3d` object.

### **slice** — Slice index

positive integer scalar in range  $[1, p]$

Slice index, specified as a positive integer scalar in the range  $[1, p]$ , where  $p$  is the number of slices in the image volume along the third dimension.

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

## Output Arguments

### **xyzCorners** — Coordinates of four corner voxels in patient coordinate system

4-by-3 matrix

Coordinates of the four corner voxels in the patient coordinate system, returned as a 4-by-3 numeric matrix. Each row contains xyz-coordinates for one corner of the slice, returned in clockwise order:

- 1 Pixel in the first row and first column of the data array.
- 2 Pixel in the first row and last column of the data array.
- 3 Pixel in the last row and last column of the data array.
- 4 Pixel in the last row and first column of the data array.

## Version History

Introduced in R2022b

### See Also

`medicalref3d`

## worldToIntrinsic

Map points from patient coordinates to intrinsic coordinates

### Syntax

```
[I,J,K] = worldToIntrinsic(R,X,Y,Z)
```

### Description

`[I,J,K] = worldToIntrinsic(R,X,Y,Z)` maps points from the patient coordinate system to the intrinsic coordinate system using the spatial referencing information, `R`. The intrinsic coordinates `I`, `J`, and `K` are defined by axes aligned with the row, column, and slice subscripts of the image data array, respectively. The patient coordinates `X`, `Y`, and `Z` are defined by the real-world patient coordinate system axes.

For a point,  $n$ , if the input coordinates  $(X_n, Y_n, Z_n)$  fall outside the image bounds, `worldToIntrinsic` extrapolates  $I_n$ ,  $J_n$ , and  $K_n$  outside the image bounds in the intrinsic coordinate system.

### Examples

#### Map 3-D Patient Coordinates to Intrinsic Coordinates

Map 3-D patient coordinates from a chest CT volume, saved as a directory of DICOM files, to intrinsic coordinates. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

Specify the directory of the DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData","LungCT01");
```

Create a medical volume object that contains the image and spatial metadata for the CT volume.

```
medVol = medicalVolume(dataFolder);
```

The `VolumeGeometry` property of the medical volume object contains a `medicalref3d` object that specifies the spatial referencing for the volume. Extract the `medicalref3d` object for the chest CT.

```
R = medVol.VolumeGeometry;
```

Select three sample points, and store their  $(x, y, z)$  patient coordinates, in millimeters. For example, the first point has patient coordinates of  $(100, 101, -200)$ , in mm. The third point is outside the image boundary.

```
X = [100 100 190];
Y = [101 101.2 -190];
Z = [-200 -100 -300];
```



Convert the world coordinates to intrinsic coordinates. The `worldToIntrinsic` function extrapolates the intrinsic coordinates of the third point outside the image boundary. The output vectors provide the  $(i, j, k)$  intrinsic coordinates, in voxels. Note that the intrinsic coordinate system is continuous, and the intrinsic coordinates can have noninteger values.

If you receive a warning that an approximate mapping is being used, then the image volume is nearly but not perfectly affine. This might be due to small numeric precision errors in how the data was encoded in the file, or due to the discrete step sizes of motors used to move the patient through the scanner. If an approximate mapping is used, you might expect small errors, on the order of millimeters in patient coordinates.

```
[I,J,K] = worldToIntrinsic(R,X,Y,Z)
```

```
I = 1×3
```

```
394.2652 394.2652 517.8040
```

```
J = 1×3
```

```
395.6379 395.9124 -3.8043
```

```
K = 1×3
```

```
33.5000 73.5000 -6.5000
```

## Input Arguments

### **R — Spatial referencing information**

`medicalref3d` object

Spatial referencing information, specified as a `medicalref3d` object.

### **X — Coordinates along x-dimension in patient coordinate system**

numeric array

Coordinates along the x-dimension in the patient coordinate system, specified as a numeric array.

X, Y, and Z must be the same size.

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

### **Y — Coordinates along y-dimension in patient coordinate system**

numeric array

Coordinates along the y-dimension in the patient coordinate system, specified as a numeric array.

X, Y, and Z must be the same size.

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

### **Z — Coordinates along z-dimension in patient coordinate system**

numeric array

Coordinates along the  $z$ -dimension in the patient coordinate system, specified as a numeric array.

$X$ ,  $Y$ , and  $Z$  must be the same size.

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

## Output Arguments

### **I** — Coordinates along $i$ -dimension in intrinsic coordinate system

numeric array

Coordinates along the  $i$ -dimension in the intrinsic coordinate system, returned as a numeric array. The  $i$ -axis is aligned with the first dimension of the spatial volume specified by  $R$ .  $I$  is the same size as  $X$ .

Data Types: `double`

### **J** — Coordinates along $j$ -dimension in intrinsic coordinate system

numeric array

Coordinates along the  $j$ -dimension in the intrinsic coordinate system, returned as a numeric array. The  $j$ -axis is aligned with the second dimension of the spatial volume specified by  $R$ .  $J$  is the same size as  $X$ .

Data Types: `double`

### **K** — Coordinates along $k$ -dimension in intrinsic coordinate system

numeric array

Coordinates along the  $k$ -dimension in the intrinsic coordinate system, returned as a numeric array. The  $k$ -axis is aligned with third dimension of the spatial volume specified by  $R$ .  $K$  is the same size as  $X$ .

Data Types: `double`

## Version History

Introduced in R2022b

### See Also

`medicalref3d` | `intrinsicToWorld` | `worldToSubscript`

# worldToSubscript

Convert from patient coordinates to row and column subscripts

## Syntax

```
[row,col,slice] = worldToSubscript(R,X,Y,Z)
```

## Description

`[row,col,slice] = worldToSubscript(R,X,Y,Z)` maps points from the patient coordinate system to the nearest subscript indices `row`, `col`, and `slice`, using the spatial referencing information, `R`.

For a point,  $n$ , if the input coordinates  $(X_n, Y_n, Z_n)$  fall outside the image bounds, `worldToSubscript` extrapolates `rown`, `coln`, and `slicen` outside the image bounds in the intrinsic coordinate system and rounds to the nearest integer values.

## Examples

### Map 3-D Patient Coordinates to Image Subscripts

Map 3-D patient coordinates from a chest CT volume, saved as a directory of DICOM files, to image subscripts. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile, filepath)
```

Specify the directory of the DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath, "MedicalVolumeDICOMData", "LungCT01");
```

Create a medical volume object that contains the image and spatial metadata for the CT volume.

```
medVol = medicalVolume(dataFolder);
```

The `VolumeGeometry` property of the medical volume object contains a `medicalref3d` object that specifies the spatial referencing for the volume. Extract the `medicalref3d` object for the chest CT.

```
R = medVol.VolumeGeometry;
```

Select three sample points, and store their  $(x,y,z)$  patient coordinates, in millimeters. For example, the first point has patient coordinates of  $(100, 101, -200)$ , in mm. The third point is outside the image boundary.

```
X = [100 100 190];
Y = [101 101.2 -190];
Z = [-200 -100 -300];
```

Convert the world coordinates to row, column, and slice indices. The `worldToSubscript` function rounds the transformed world coordinates to integer values. The function extrapolates the subscripts of the point outside the image boundary.

If you receive a warning that an approximate mapping is being used, then the image volume is nearly but not perfectly affine. This might be due to small numeric precision errors in how the data was encoded in the file, or due to the discrete step sizes of motors used to move the patient through the scanner. If an approximate mapping is used, you might expect small errors, on the order of millimeters in patient coordinates.

```
[row,col,slice] = worldToSubscript(R,X,Y,Z)
```

```
row = 1×3
```

```
    394    394    518
```

```
col = 1×3
```

```
    396    396     -4
```

```
slice = 1×3
```

```
    34    74    -7
```

## Input Arguments

### **R — Spatial referencing information**

`medicalref3d` object

Spatial referencing information, specified as a `medicalref3d` object.

### **X — Coordinates along x-dimension in patient coordinate system**

numeric array

Coordinates along the x-dimension in the patient coordinate system, specified as a numeric array.

X, Y, and Z must be the same size.

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

### **Y — Coordinates along y-dimension in patient coordinate system**

numeric array

Coordinates along the y-dimension in the patient coordinate system, specified as a numeric array.

X, Y, and Z must be the same size.

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

### **Z — Coordinates along z-dimension in patient coordinate system**

numeric array

Coordinates along the z-dimension in the patient coordinate system, specified as a numeric array.

X, Y, and Z must be the same size.

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

## Output Arguments

### **row** — Row subscript indices

positive integer array

Row subscript indices, returned as a positive integer array. `row` is the same size as X.

Data Types: `double`

### **col** — Column subscript indices

positive integer array

Column subscript indices, returned as a positive integer array. `col` is the same size as X.

Data Types: `double`

### **slice** — Slice subscript indices

positive integer array

Slice subscript indices, returned as a positive integer array. `slice` is the same size as X.

Data Types: `double`

## Version History

Introduced in R2022b

### See Also

`medicalref3d` | `worldToIntrinsic` | `intrinsicToWorld`

## extractSlice

Extract voxels and spatial details for one slice of medical volume

### Syntax

```
[X,position,spacings] = extractSlice(medVol,slice,direction)
```

### Description

`[X,position,spacings] = extractSlice(medVol,slice,direction)` extracts the voxel data and spatial information for one slice, `slice`, of the `medicalVolume` object `medVol` along the specified direction, `direction`.

### Examples

#### Extract Slice from Medical Image Volume

Extract a slice from a medical image volume created using a chest CT volume saved as a directory of DICOM files. The CT volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

Specify the directory of DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData","LungCT01");
```

Create a medical volume object for the CT volume.

```
medVol = medicalVolume(dataFolder);
```

Extract the third slice in the transverse direction.

```
[X,position,spacings] = extractSlice(medVol,3,"transverse");
```

The `X` output contains the voxel data for the extracted slice.

```
whos X
```

Name	Size	Bytes	Class	Attributes
X	512x512	524288	int16	

The `position` output provides the patient coordinates of the first voxel in the slice, in millimeters.

```
position
```

```
position = 1x3
```

```
-186.5000 -186.5000 -276.2500
```

The `spacings` output provides the in-plane pixel spacing within the extracted slice, in millimeters.

```
spacings
```

```
spacings = 1×2
```

```
    0.7285    0.7285
```

## Input Arguments

### **medVol** — Medical volume

medicalVolume object

Medical volume, specified as a medicalVolume object.

---

**Note** The `extractSlice` function is not valid if the `Orientation` property value of `medVol` is "mixed", "oblique", or "unknown". In these cases, access the voxel data stored in the `Voxels` property directly using array indexing.

---

### **slice** — Slice index

positive integer scalar in range [1, *numSlices*]

Slice index, specified as a positive integer scalar in the range [1, *numSlices*], where *numSlices* is the number of slices in the volume along the direction specified by `direction`.

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

### **direction** — Direction along which to extract slice

"coronal" | "sagittal" | "transverse"

Direction along which to extract the slice information, specified as "coronal", "sagittal", or "transverse".

Data Types: char | string

## Output Arguments

### **X** — Voxel data of extracted slice

*m*-by-*n* numeric matrix

Voxel data of the extracted slice, returned as an *m*-by-*n* numeric matrix, where *m* and *n* are the number of rows and columns in the image slice in the plane specified by `direction`. The `X` output array is the same data type as the `Voxels` property of `medVol`.

### **position** — Position of upper-left pixel of extracted slice

3-element numeric vector

Position of the upper-left pixel of the extracted slice, returned as a 3-element numeric vector of the form  $[x\ y\ z]$ . The coordinates are in the patient coordinate system, in the units specified by the `SpatialUnits` property of `medVol`.

**spacings — Spacing between voxel centers in first two dimensions of extracted slice**

2-element numeric vector

Spacing between voxel centers in the first two dimensions of the extracted slice, returned as a 2-element numeric vector of the form  $[dim1_{spacing}\ dim2_{spacing}]$ . The values are in the units specified by the `SpatialUnits` property of `medVol`.

## Version History

Introduced in R2022b

### See Also

`medicalVolume` | `medicalref3d` | `replaceSlice` | `sliceCorners` | `sliceLimits`

### Topics

“Read, Process, and Write 3-D Medical Images”



# replaceSlice

Replace voxel values for one slice of medical volume

## Syntax

```
medVolUpdated = replaceSlice(medVol,slice,direction,sliceValues)
```

## Description

`medVolUpdated = replaceSlice(medVol,slice,direction,sliceValues)` replaces the voxel values for one slice, `slice`, of the `medicalVolume` object `medVol` along the specified direction, `direction`. The `replaceSlice` function returns a new `medicalVolume` object with the updated values, specified by `sliceValues`.

## Examples

### Replace Slice in Medical Image Volume

Replace one slice of a medical image volume created using a chest CT volume saved as a directory of DICOM files. The CT volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

Specify the directory of DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData","LungCT01");
```

Create a medical volume object for the CT volume.

```
medVol = medicalVolume(dataFolder);
```

Extract the second slice in the transverse direction. The size of the extracted slice, `X`, is 512-by-512.

```
[X,position,spacings] = extractSlice(medVol,2,"transverse");
whos X
```

Name	Size	Bytes	Class	Attributes
X	512x512	524288	int16	

Specify a new 512-by-512 matrix to replace the extracted slice.

```
newX = ones(512);
```

Create a new `medicalVolume` object that replaces the extracted slice with the new slice.

```
medVolUpdated = replaceSlice(medVol,2,"transverse",newX);
```

## Input Arguments

### **medVol** — Medical volume

medicalVolume object

Medical volume, specified as a medicalVolume object.

### **slice** — Slice index

positive integer scalar in range [1, numSlices]

Slice index, specified as a positive integer scalar in the range [1, numSlices], where numSlices is the number of slices in the volume along the direction specified by direction.

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

### **direction** — Direction along which to update slice

"coronal" | "sagittal" | "transverse"

Direction along which to update the slice information, specified as "coronal", "sagittal", or "transverse".

Data Types: char | string

### **sliceValues** — New voxel values

numeric array

New voxel values, specified as a numeric array. The size of sliceValues must be the same size as the original slice in medVol specified by slice.

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

## Output Arguments

### **medVolUpdated** — Medical volume with updated slice

medicalVolume object

Medical volume with updated slice, returned as a medicalVolume object.

## Version History

Introduced in R2022b

## See Also

medicalVolume | extractSlice

# resample

Resample medical image volume in different patient coordinate system

## Syntax

```
medVolResampled = resample(medVol,R)
medVolResampled = resample(medVol,R,Name=Value)
```

## Description

`medVolResampled = resample(medVol,R)` resamples the voxel data stored in the `medicalVolume` object `medVol` in the patient coordinate system specified by `R`. The updated voxel data is returned in a new `medicalVolume` object, `medVolResampled`.

`medVolResampled = resample(medVol,R,Name=Value)` specifies additional options for resampling the voxel data using name-value arguments. For example, `Method="linear"` specifies a linear method for resampling.

## Examples

### Resample Medical Image Volume to Target Voxel Size

Resample a chest CT volume to a target voxel size, without changing the spatial limits in the patient coordinate system. This can be useful to standardize the voxel spacing in a data set of scans acquired using different settings.

The CT volume is saved as a directory of DICOM files. The volume is part of a data set containing three CT scans. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

Specify the directory of DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData","LungCT01");
```

Create a medical volume object, `medVol`, for the CT volume. The voxel size is 0.7285-by-0.7825-by-2.5 mm.

```
medVol = medicalVolume(dataFolder);
medVol.VoxelSpacing
```

```
ans = 1×3
```

```
    0.7285    0.7285    2.5000
```

Resample `medVol` so that it has a voxel size of 0.5-by-0.5-by-1.25 mm. First, calculate the ratio between the original voxel size and the target voxel size, in millimeters, in each dimension.

```
targetVoxelSize = [0.5 0.5 1.25];
ratios = targetVoxelSize ./ medVol.VoxelSpacing;
```

Calculate the target dimensions, in voxels, of the resampled volume.

```
origSize = size(medVol.Voxels);
newSize = round(origSize ./ ratios);
```

Define the spatial referencing for the resampled volume. First, get the mapping between the intrinsic and patient coordinate systems of `medVol` by using the `intrinsicToWorldMapping` object function. The output, `origMapping`, is an `affinetform3d` object that contains a transformation matrix in its `A` property.

```
origRef = medVol.VolumeGeometry;
origMapping = intrinsicToWorldMapping(origRef);
tform = origMapping.A;
```

Transform the matrix `tform` so that it corresponds to the target voxel size. Create an `affinetform3d` object, `newMapping`, that contains the transformed matrix.

```
newMapping4by4 = tform.* [ratios([2 1 3]) 1];
newMapping = affinetform3d(newMapping4by4);
```

Create a `medicalref3d` object that describes the target spatial referencing for the resampled volume.

```
newRef = medicalref3d(newSize,newMapping);
```

To maintain the mapping between the patient coordinate axes and the anatomical planes, use the `orient` object function to set the `PatientCoordinateSystem` property of `newRef` to match `origRef`.

```
newRef = orient(newRef,origRef.PatientCoordinateSystem);
```

Resample `medVol` by using the `resample` object function. The voxel size of the new medical volume matches the target size.

```
newVol = resample(medVol,newRef)
```

```
newVol =
```

```
    medicalVolume with properties:
```

```
        Voxels: [746×746×176 int16]
    VolumeGeometry: [1×1 medicalref3d]
      SpatialUnits: "unknown"
      Orientation: "transverse"
    VoxelSpacing: [0.5000 0.5000 1.2500]
    NormalVector: [0 0 1]
    NumCoronalSlices: 746
    NumSagittalSlices: 746
    NumTransverseSlices: 176
      PlaneMapping: ["sagittal"    "coronal"    "transverse"]
      Modality: "unknown"
    WindowCenters: []
    WindowWidths: []
```

## Input Arguments

### **medVol** — Medical volume

medicalVolume object

Medical volume, specified as a medicalVolume object.

### **R** — Spatial referencing object

medicalref3d object

Spatial referencing object, specified as a medicalref3d object. R defines the spatial details for the resampled volume.

### 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: `resample(medVol,R,Method="linear")` calculates the voxel values in `medVolResampled` by linearly interpolating voxel values at the corresponding locations in `medVol`.

### **FillValue** — Fill value

0 (default) | numeric scalar

Fill value, specified as a numeric scalar. The value of `FillValue` must be supported by the data type of the voxel data stored in the `Voxels` property of R. If an output voxel in the volume defined by R falls outside the input volume, `medVol`, `resample` sets the corresponding value of `medVolResampled` to `FillValue`.

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

### **Method** — Resampling method

"cubic" (default) | "linear" | "nearest"

Resampling method, specified as "cubic", "linear", or "nearest".

Data Types: `char` | `string`

## Output Arguments

### **medVolResampled** — Resampled medical volume

medicalVolume object

Resampled medical volume, returned as a medicalVolume object. The `VolumeGeometry` property of `medVolResampled` contains the spatial referencing information specified by R. The resampled voxel data is stored in the `Voxels` property of `medVolResampled`.

## Algorithms

The `resample` function calculates the voxel values in the output volume, `medVolResampled`, by mapping locations in the output volume to the corresponding locations in the input volume (an inverse mapping). When the center of the voxel in the output volume does not map to the center of a voxel in the input volume, `resample` interpolates the input voxel values to calculate the output values.

`resample` performs the mapping in the patient coordinate systems of each volume. The function assumes that the patient coordinate systems of the input and output volumes have the same orientation convention, specified by the `PatientCoordinateSystem` property of a `medicalref3d` object, and that the patient coordinate systems have the same origin.

## **Version History**

**Introduced in R2022b**

### **See Also**

`medicalVolume` | `medicalref3d`

### **Topics**

“Medical Image Coordinate Systems”

# sliceCorners

Extract coordinates of corner voxels for one slice of medical volume

## Syntax

```
xyzCorners = sliceCorners(medVol,slice,direction)
```

## Description

`xyzCorners = sliceCorners(medVol,slice,direction)` extracts the *xyz*-coordinates of the four corner voxels for one slice, *slice*, in the specified direction *direction* of the `medicalVolume` object `medVol`. The function returns the corner coordinates in the patient coordinate system.

## Examples

### Extract Corner Coordinates for One Slice of Medical Volume

Extract corner coordinates for a slice of a medical volume created using a chest CT volume saved as a directory of DICOM files. The CT volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

The `dataFolder` folder contains the downloaded CT volume.

```
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData","LungCT01");
```

Create a medical volume object for the CT volume.

```
medVol = medicalVolume(dataFolder);
```

Extract the *xyz*-coordinates, in millimeters, of the corner voxels for the second slice in the coronal direction.

```
xyzCorners = sliceCorners(medVol,2,"coronal")
```

```
xyzCorners = 4x3
```

```
-186.5000 -185.7715 -281.2500
-186.5000 -185.7715 -63.7500
 185.7717 -185.7715 -63.7500
 185.7717 -185.7715 -281.2500
```

## Input Arguments

### **medVol** — Medical volume

medicalVolume object

Medical volume, specified as a medicalVolume object.

### **slice** — Slice index

positive integer scalar in range [1, numSlices]

Slice index, specified as a positive integer scalar in the range [1, numSlices], where numSlices is the number of slices in the volume along the direction specified by direction.

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

### **direction** — Direction along which to extract slice limits

"coronal" | "sagittal" | "transverse"

Direction along which to extract the slice limits, specified as "coronal", "sagittal", or "transverse".

Data Types: char | string

## Output Arguments

### **xyzCorners** — Coordinates of four corner voxels in patient coordinate system

4-by-3 numeric matrix

Coordinates of the four corner voxels in the patient coordinate system, returned as a 4-by-3 numeric matrix. Each row contains the xyz-coordinates for one corner, returned in clockwise order from the upper-left corner.

## Version History

Introduced in R2022b

### See Also

medicalVolume | sliceLimits



# sliceLimits

Extract X-, Y-, Z-limits for one slice of medical volume

## Syntax

```
[XLim,YLim,ZLim] = sliceLimits(medVol,slice,direction)
```

## Description

`[XLim,YLim,ZLim] = sliceLimits(medVol,slice,direction)` extracts the X-, Y-, and Z-limits for one slice, `slice`, in the specified direction `direction` of the `medicalVolume` object `medVol`. The slice limits are in the patient coordinate system.

## Examples

### Get Limits for One Slice of Medical Volume

Get the limits for one slice of a medical volume created using a chest CT volume saved as a directory of DICOM files. The CT volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

Specify the directory of DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData","LungCT01");
```

Create a medical volume object for the CT volume.

```
medVol = medicalVolume(dataFolder);
```

Extract the x-, y-, and z-axis limits, in millimeters, of the second slice in the coronal direction.

```
[XLim,YLim,ZLim] = sliceLimits(medVol,2,"coronal")
```

```
XLim = 1×2
```

```
-186.5000 185.7717
```

```
YLim = 1×2
```

```
-185.7715 -185.7715
```

```
ZLim = 1×2
```

```
-281.2500 -63.7500
```

## Input Arguments

### **medVol** — Medical volume

medicalVolume object

Medical volume, specified as a medicalVolume object.

### **slice** — Slice index

positive integer scalar in range [1, numSlices]

Slice index, specified as a positive integer scalar in the range [1, numSlices], where numSlices is the number of slices in the volume along the direction specified by direction.

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

### **direction** — Direction along which to extract slice limits

"coronal" | "sagittal" | "transverse"

Direction along which to extract the slice limits, specified as "coronal", "sagittal", or "transverse".

Data Types: char | string

## Output Arguments

### **XLim** — Limits of volume in x-dimension of patient coordinate system

2-element numeric vector

Limits of the volume in the x-dimension of the patient coordinate system, returned as a 2-element numeric vector of the form [xMin xMax]. The limits are in the units specified by the SpatialUnits property of medVol.

### **YLim** — Limits of volume in y-dimension of patient coordinate system

2-element numeric vector

Limits of the volume in the y-dimension of the patient coordinate system, returned as a 2-element numeric vector of the form [yMin yMax]. The limits are in the units specified by the SpatialUnits property of medVol.

### **ZLim** — Limits of volume in z-dimension of patient coordinate system

2-element numeric vector

Limits of the volume in the z-dimension of the patient coordinate system, returned as a 2-element numeric vector of the form [zMin zMax]. The limits are in the units specified by the SpatialUnits property of medVol.

## Version History

Introduced in R2022b

**See Also**

medicalVolume | sliceCorners

## volshow

Display medical volume in patient coordinates

### Syntax

```
vol = volshow(medVol)  
vol = volshow(medVol,Name=Value)
```

```
vol = volshow(V)  
vol = volshow(V,config)  
vol = volshow(V,Name=Value)
```

```
bVol = volshow(bim)  
bVol = volshow(bim,Name=Value)
```

### Description

Medical Imaging Toolbox extends the functionality of the `volshow` (Image Processing Toolbox) function to display a `medicalVolume` object in the patient coordinate system. The function uses the `medicalVolume` properties to set the `Transformation` property of the displayed `Volume` object. If you do not have Medical Imaging Toolbox installed, see `volshow` (Image Processing Toolbox).

#### Medical Volume Object

`vol = volshow(medVol)` creates a `Volume` object that displays the 3-D image volume in the `medicalVolume` object `medVol` in the patient coordinate system. You can rotate and zoom in and out on the display interactively using the mouse. Use `vol` to access and modify properties of the `Volume` object after creation. For a list of properties, see `Volume`.

`vol = volshow(medVol,Name=Value)` modifies the appearance of the volume using one or more name-value arguments. For example, `RenderingStyle="Isosurface"` specifies the rendering style of the volume as "Isosurface".

#### Numeric Array

`vol = volshow(V)` creates a `Volume` object that displays the 3-D volume `V`. Use `vol` to query and modify properties of the `Volume` object after you create the object. For a list of properties, see `Volume`. Use numeric array inputs to display images in file formats not supported by `medicalVolume`, or to view a volume in the intrinsic coordinate system.

`vol = volshow(V,config)` displays the 3-D volume `V` using the values of the `Volume` object properties specified by the `config` structure.

`vol = volshow(V,Name=Value)` modifies the appearance of the volume using one or more name-value arguments. For example, `RenderingStyle="Isosurface"` specifies the rendering style of the volume as "Isosurface".

#### Blocked Image Volume

`bVol = volshow(bim)` creates a `BlockedVolume` object that displays the 3-D blocked image `bim`. You can rotate and zoom in and out on the display interactively using the mouse. Use `bVol` to query

and modify properties of the `BlockedVolume` object after you create the object. For a list of properties, see `BlockedVolume` Properties.

`bVol = volshow(bim,Name=Value)` modifies the appearance of the blocked volume using one or more name-value arguments. For example, `ResolutionLevel="coarse"` specifies the resolution level to display as the coarsest resolution level.

## Examples

### Display Medical Image Volume in Patient Coordinates

Display a chest CT volume saved as a directory of DICOM files. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB.

Run this code to download the data set from the MathWorks® website and unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

The `dataFolder` folder contains the downloaded CT scan used by this example.

```
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData","LungCT01");
```

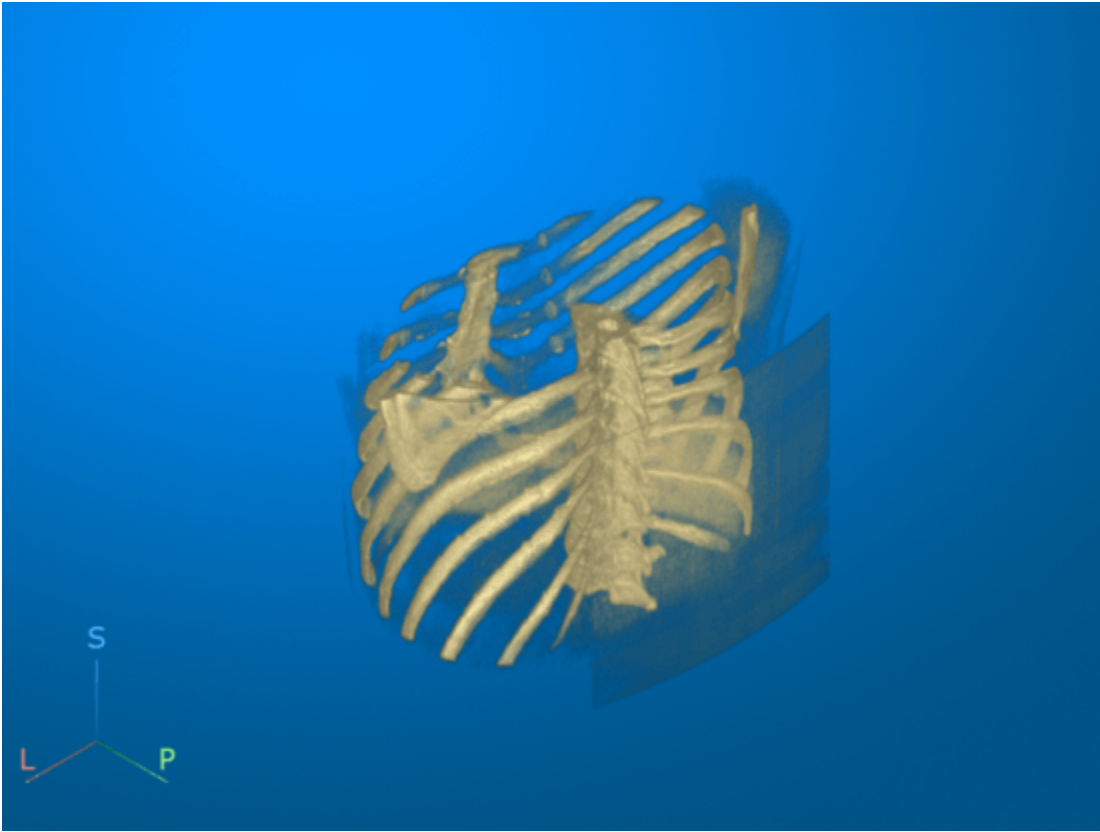
Create a medical image volume object that contains the image data and spatial referencing information for the CT volume.

```
medVol = medicalVolume(dataFolder);
```

Display the image stored in the `Voxels` property of `medVol`. Specify a custom colormap and transparency map. The `volshow` object function displays the volume in the patient coordinate system defined by the spatial properties of `medVol`. The axes display indicators label the inferior/superior (**S**), left/right (**L**), and anterior/posterior (**P**) anatomical axes.

```
alpha = [0 0 0.72 0.72];
color = [0 0 0; 231 208 141; 231 208 141; 255 255 255]/255;
intensity = [-3024 50 1400 1499];
queryPoints = linspace(min(intensity),max(intensity),256);
alphamap = interp1(intensity,alpha,queryPoints)';
colormap = interp1(intensity,color,queryPoints);

volshow(medVol,Colormap=colormap,Alphamap=alphamap)
```



### Display Medical Image Volume in Intrinsic Coordinates

Display a chest CT volume saved as a directory of DICOM files. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB.

Run this code to download the data set from the MathWorks® website and unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeDICOMData.zip");  
filepath = fileparts(zipFile);  
unzip(zipFile, filepath)
```

The dataFolder folder contains the downloaded CT scan.

```
dataFolder = fullfile(filepath, "MedicalVolumeDICOMData", "LungCT01");
```

Create a medical volume object that contains the image data and spatial referencing information for the CT volume. The `Voxels` property contains a numeric array of the voxel intensities. The `VoxelSpacing` property indicates that the voxels are anisotropic, with a size of 0.7285-by-0.7285-by-2.5 mm in the patient coordinate system.

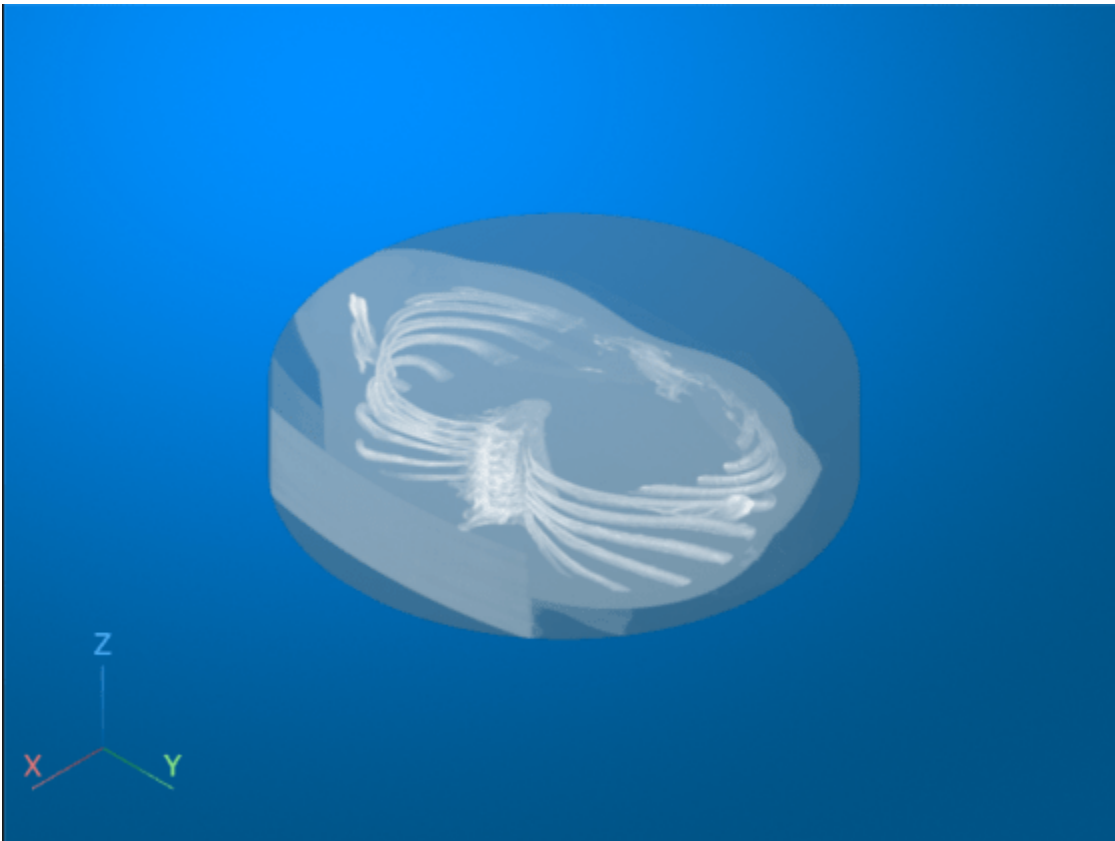
```
medVol = medicalVolume(dataFolder)
```

```
medVol =  
    medicalVolume with properties:
```

```
Voxels: [512x512x88 int16]
VolumeGeometry: [1x1 medicalref3d]
SpatialUnits: "mm"
Orientation: "transverse"
VoxelSpacing: [0.7285 0.7285 2.5000]
NormalVector: [0 0 1]
NumCoronalSlices: 512
NumSagittalSlices: 512
NumTransverseSlices: 88
PlaneMapping: ["sagittal" "coronal" "transverse"]
Modality: "CT"
WindowCenters: [88x1 double]
WindowWidths: [88x1 double]
```

Display the volume in the intrinsic coordinate system by using `volshow` with the numeric array from the `Voxels` property as input. The intrinsic coordinate system assumes cubic voxels with uniform dimensions. Therefore, this anisotropic volume appears squished. The x-, y- and z-axis display markers align with the first, second, and third dimensions of the image array, respectively, rather than the x-, y-, and z-axes of the patient coordinate system.

```
vol = volshow(medVol.Voxels,RenderingStyle="MaximumIntensityProjection");
```



## Input Arguments

### **medVol — Medical volume**

medicalVolume object

Medical volume, specified as a medicalVolume object. The function uses the spatial referencing information stored in the VolumeGeometry property of medVol to define the Transformation property of vol and display the image in patient coordinates.

### **V — 3-D volume**

numeric array

3-D volume, specified as a numeric array.

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

### **config — Rendering information**

structure

Rendering information exported by **Volume Viewer**, specified as a structure.

Data Types: struct

### **bim — Blocked image volume**

blockedImage object

Blocked image volume, specified as a blockedImage object that reads 3-D blocks of grayscale, RGB, or RGBA data. The blocked image can have a single resolution level or multiple resolution levels.

## 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: volshow(medVol, RenderingStyle="Isosurface") displays the image data in medVol as an isosurface in the patient coordinate system.

---

**Note** The properties listed here are only a subset. For a full list of properties that you can modify after object creation, see Volume or BlockedVolume Properties.

---

### **AnatomicalConvention — Anatomical display orientation convention**

"radiological" (default) | "neurological"

Anatomical display orientation convention, specified as one of these values:

- "radiological" — Displays the left side of the patient on the right side of the image when viewing the volume head on.
- "neurological" — Displays the left side of the patient on the left side of the image when viewing the volume head on.

---

**Note** To use the AnatomicalConvention name-value argument, you must specify vol as a medicalVolume object.

---



Data Types: char | string

### Parent — Parent

Viewer3D object

Parent of the Volume object, specified as a Viewer3D object. You can create a Viewer3D object using the viewer3d function. When you call volshow without specifying a parent, the function creates a new Viewer3D object and sets that object as the parent. You cannot reparent a Volume object.

### RenderingStyle — Rendering style

"VolumeRendering" (default) | "MaximumIntensityProjection" |  
"MinimumIntensityProjection" | "GradientOpacity" | "Isosurface" | "SlicePlanes"

Rendering style, specified as one of the values in the table.

Value	Description
"VolumeRendering"	View the volume based on the specified color and transparency for each voxel.
"MaximumIntensityProjection"	View the voxel with the largest intensity value for each ray projected through the data. For RGB volumes, view the voxel with the largest luminance in the CIE 1976 L*a*b* color space.
"MinimumIntensityProjection"	View the voxel with the smallest intensity value for each ray projected through the data. For RGB volumes, view the voxel with the smallest luminance in the CIE 1976 L*a*b* color space.
"GradientOpacity"	View the volume based on the specified color and transparency with an additional transparency applied if the voxel is similar in intensity (for grayscale volumes) or luminance (for RGB volumes) to the previous voxel along the viewing ray.  When you render a volume with uniform intensity using "GradientOpacity", the internal portion of the volume appears more transparent than in the "VolumeRendering" rendering style, enabling you to better visualize the intensity or luminance gradients in the volume.
"Isosurface"	View an isosurface of the volume, as specified by the value in the IsosurfaceValue property.
"SlicePlanes"	View three orthogonal slice planes.

### Alphamap — Transparency map for volume content

linspace(0,1,256)' (default) | *n*-element column vector

Transparency map for the volume content, specified as an *n*-element column vector with values in the range [0, 1]. The maximum length of the vector is 256. When viewing RGB volumes, the object uses the luminance of the voxel in the CIE 1976 L\*a\*b\* color space to assign a transparency from the transparency map. When the AlphaData property is nonempty, the Alphamap property has no effect.

**Colormap — Colormap**gray(256) (default) |  $n$ -by-3 numeric matrix

Colormap of grayscale volume content, specified as an  $n$ -by-3 numeric matrix with values in the range [0, 1]. The maximum number of colors  $n$  is 256. This property has no effect when viewing RGB volumes.

**OverlayData — Overlay data**

[] (default) | numeric array | blockedImage object

Overlay data to blend with the object data during rendering, specified as one of these values:

- When displaying a `medicalVolume` object `vol` or numeric array  $V$ , specify `OverlayData` as a numeric array.
- When displaying a blocked image volume, `bim`, specify `OverlayData` as a `blockedImage` object that reads blocks of 3-D grayscale data.

The object shows the overlay only when the `RenderingStyle` property value is "SlicePlanes", "VolumeRendering", or "GradientOpacity". You can modify the appearance of the overlay by changing the `OverlayRenderingStyle`, `OverlayColormap`, and `OverlayAlphamap` properties.

**OverlayRenderingStyle — Overlay rendering style**

"LabelOverlay" (default) | "VolumeOverlay" | "GradientOverlay"

Overlay rendering style, specified as one of the values in the table.

Value	Description
"LabelOverlay"	View the overlay based on the color and transparency of each labeled region. Use this rendering style to visualize ordinal data, like binary or semantic segmentation results, on top of your data.
"VolumeOverlay"	View the overlay based on the specified color and transparency for each voxel.
"GradientOverlay"	View the overlay based on the color and transparency for each voxel with an additional transparency applied based on the difference between the current voxel and the previous voxel along the viewing ray.

**Output Arguments****vol — Volume**

Volume object

Volume, returned as a `Volume` object. For `medicalVolume` object inputs, `volshow` sets the `Transformation` property of `vol` to the 4-by-4 transformation matrix defining the patient coordinate system.

For more information about modifying aspects of the volume display, see `Volume`.

## bVol — Blocked Volume

BlockedVolume object

Blocked volume, returned as a BlockedVolume object. For more information about modifying aspects of the volume, see BlockedVolume Properties.

## More About

### Events

To receive notification from the Volume object when certain events happen, set up listeners for these events. You can specify a callback function to call when one of these events occurs. When the Volume object notifies your application through the listener, MATLAB returns data specific to the event. Look at the event class for the specific event to see what the callback function returns.

Event Name	Trigger	Event Data	Event Attributes
ClippingPlanesChanging	Interactively moving an object clipping plane. This event does not execute if the clipping plane is programmatically moved.	images.ui.graphics 3d.events.ClippingPlanesChangedEventData	NotifyAccess: private  ListenAccess: public
ClippingPlanesChanged	Ceasing to interactively move an object clipping plane. This event does not execute if the clipping plane is programmatically moved.	images.ui.graphics 3d.events.ClippingPlanesChangedEventData	NotifyAccess: private  ListenAccess: public
SlicePlanesChanging	Interactively moving an object slice plane. This event does not execute if the slice plane is programmatically moved.	images.ui.graphics 3d.events.SlicePlanesChangedEventData	NotifyAccess: private  ListenAccess: public
SlicePlanesChanged	Ceasing to interactively move an object slice plane. This event does not execute if the slice plane is programmatically moved.	images.ui.graphics 3d.events.SlicePlanesChangedEventData	NotifyAccess: private  ListenAccess: public
DataReadStarted	A BlockedVolume object is sending blocks of data to be rendered in the scene. This event is not applicable for Volume objects.	event.EventData	NotifyAccess: private  ListenAccess: public

<b>Event Name</b>	<b>Trigger</b>	<b>Event Data</b>	<b>Event Attributes</b>
DataReadFinished	The BlockedVolume object has finished sending all blocks of data that are visible in the scene. This event is not applicable for Volume objects.	event.EventData	NotifyAccess: private  ListenAccess: public

## Version History

Introduced in R2023a

### See Also

#### Objects

sliceViewer | medicalVolume | medicalref3d | Volume | Surface | Viewer3D Properties

#### Functions

montage | viewer3d

#### Topics

“Choose Approach for Medical Image Visualization”

“Display Medical Image Volume in Patient Coordinate System”

“Display Labeled Medical Image Volume in Patient Coordinate System”

# write

Write affine medical volume data to NIFTI file

## Syntax

```
write medVol, filename)
write medVol, filename, info)
```

## Description

`write medVol, filename)` writes the voxel data and spatial information of the affine `medicalVolume` object `medVol` to the file `filename` in the Neuroimaging Informatics Technology Initiative (NIFTI) file format. The `write` function creates a combined NIFTI file that contains both metadata and volumetric data. The object function populates the metadata using appropriate default values and volume properties, such as size and data type.

`write medVol, filename, info)` sets metadata attributes by using the metadata structure `info`. If the specified metadata structure does not match the image contents and size, then `write` returns an error.

## Examples

### Write Medical Volume Object Data to NIFTI File

Write data from a medical volume object, created using a chest CT volume saved as a directory of DICOM files. The CT volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile, filepath)
```

Specify the directory of DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath, "MedicalVolumeDICOMData", "LungCT01");
```

Create a medical volume object for the CT volume.

```
medVol = medicalVolume(dataFolder);
```

The `Voxels` property contains the intensity values of each voxel. The `VolumeGeometry` property contains a `medicalref3d` object defining the spatial referencing for the image volume.

```
V = medVol.Voxels;
R = medVol.VolumeGeometry;
```

Modify the voxel data by applying a 3-D Gaussian filter.

```
sigma = 2;
filterV = imgaussfilt3(V, sigma);
```

Create a new `medicalVolume` object that contains the smoothed voxel values. To maintain the same spatial referencing as the original volume, specify the original `medicalref3d` object `R`.

```
medVolSmooth = medicalVolume(filterV,R);
```

Write the smoothed image data to a new NIfTI file.

```
niftiFilename = "LungCT01_smoothed.nii";  
write(medVolSmooth,niftiFilename)
```

## Input Arguments

### **medVol** — Medical volume

`medicalVolume` object

Medical volume, specified as a `medicalVolume` object. `medVol` must be an affine image volume.

An image volume is affine if these conditions are met:

- All slices are parallel to each other.
- The spacing between slices in each dimension is uniform.
- The upper-left voxels of all slices are collinear.
- No two slices are coincident, meaning no two slices are located at the same position in space.

### **filename** — Name of NIfTI file

string scalar | character vector

Name of the NIfTI file, specified as a string scalar or a character vector. The `write` function creates a combined NIfTI file that contains both metadata and volumetric data and has the file extension `.nii`.

### **info** — NIfTI file metadata

structure

NIfTI file metadata, specified as a structure in the format returned by the `niftiinfo` function.

## Version History

**Introduced in R2022b**

### See Also

`medicalVolume` | `niftiinfo` | `niftiwrite`

# montage

Display medical image slices or frames as montage in patient coordinates

## Syntax

```
montage(medImage)  
montage(medVolume)  
  
montage(I)  
montage(imagelist)  
montage(filenames)  
montage(imds)  
  
montage(___,map)  
montage(___,Name=Value)  
img = montage(___)
```

## Description

Medical Imaging Toolbox extends the functionality of the `montage` (Image Processing Toolbox) function to display data in a `medicalImage` or `medicalVolume` object. If you do not have Medical Imaging Toolbox installed, see `montage` (Image Processing Toolbox).

### Medical Objects

`montage(medImage)` displays all of the frames in a `medicalImage` object as a rectangular montage. The function automatically adjusts display settings using `medicalImage` properties.

`montage(medVolume)` displays all of the slices in a `medicalVolume` object as a rectangular montage. The function automatically adjusts display settings using `medicalVolume` properties.

### Nonmedical Image Formats

`montage(I)` displays all of the frames of the numeric array `I`.

`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 the specified filenames `filenames`. Files must be in standard image formats supported by `imread`.

`montage(imds)` displays a montage of the images specified in the image datastore `imds`.

### Additional Options

`montage( ___ ,map)` treats grayscale and binary images as indexed images and displays them with the specified colormap `map`, in addition to any input argument from a previous syntax. If you specify images using a `medicalImage` object, `medicalVolume` object, `filenames`, or an image datastore, then `map` overrides any internal colormap present in the object or 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( ____, Name=Value)` returns a handle to the single image object that contains all of the displayed images.

## Examples

### Display Medical Image Series as Montage

Specify the name of a DICOM file containing an echocardiogram ultrasound series. The DICOM file is attached to this example as a supporting file.

```
filename = "heartUltrasoundSequenceVideo.dcm";
```

Read the metadata and image data from the file by creating a `medicalImage` object. The `FrameTime` property indicates that each frame has a duration of 33.333 milliseconds. The `NumFrames` property indicates that the series has a total of 116 image frames.

```
medImg = medicalImage(filename)
```

```
medImg =  
    medicalImage with properties:  
  
        Pixels: [600x800x116x3 uint8]  
        Colormap: []  
        SpatialUnits: "unknown"  
        FrameTime: 33.3330  
        NumFrames: 116  
        PixelSpacing: [1 1]  
        Modality: 'US'  
        WindowCenter: []  
        WindowWidth: []
```

Display the frames of the image series stored in the `Pixels` property of `medImg` as a montage.

```
montage(medImg)
```





### Display Medical Image Volume Slices as Montage

Display a montage of slices from a chest CT volume saved as a directory of DICOM files. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB.

Run this code to download the data set from the MathWorks® website and unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeDICOMData.zip");  
filepath = fileparts(zipFile);  
unzip(zipFile, filepath)
```

The dataFolder folder contains the downloaded CT volume.

```
dataFolder = fullfile(filepath, "MedicalVolumeDICOMData", "LungCT01");
```

Create a medical image volume object that contains the image data and spatial referencing information for the CT volume.

```
medVol = medicalVolume(dataFolder)
```

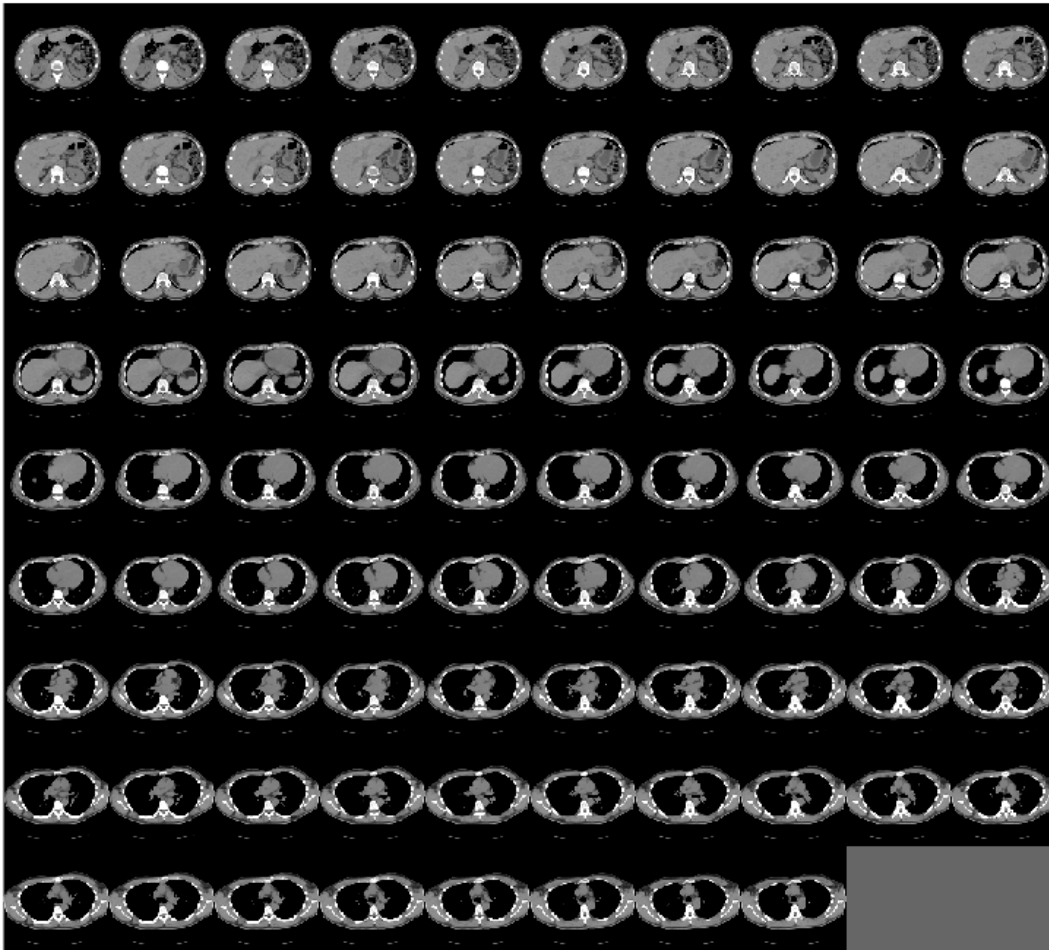
```
medVol =
```

```
    medicalVolume with properties:
```

```
        Voxels: [512x512x88 int16]
    VolumeGeometry: [1x1 medicalref3d]
        SpatialUnits: "mm"
        Orientation: "transverse"
        VoxelSpacing: [0.7285 0.7285 2.5000]
        NormalVector: [0 0 1]
    NumCoronalSlices: 512
    NumSagittalSlices: 512
    NumTransverseSlices: 88
        PlaneMapping: ["sagittal"    "coronal"    "transverse"]
        Modality: "CT"
    WindowCenters: [88x1 double]
    WindowWidths: [88x1 double]
```

Display the slices of the images stored in `medVol` as a montage. The `Orientation` property of `medVol` determines the slice plane, which is transverse in this example. The `montage` object function automatically sets the intensity display range and orients slices in the patient coordinate system using the property values of `medVol`.

```
montage(medVol)
```



## Input Arguments

### **medImage — Medical image**

medicalImage object

Medical image, specified as a medicalImage object. The montage object function displays frames stored in the Pixels property of medImage and uses these properties of medImage to set display settings:

- WindowCenter and WindowWidth — If specified, these properties set the DisplayRange name-value argument.
- Colormap — If specified, this property sets the map input argument.

### **medVolume — Medical volume**

medicalVolume object

Medical volume, specified as a `medicalVolume` object. The `montage` object function displays slices stored in the `Voxels` property of `medVolume`, and uses these properties of `medVolume` to set display settings:

- `WindowCenters` and `WindowWidths` — If specified, these properties set the `DisplayRange` name-value argument.
- `Orientation` — Specifies the plane in which the function displays slices:
  - If the `Orientation` property value is "transverse", then `montage` displays transverse slices with the anterior side of the patient on the top of the image and the right side of the patient on the left side of the image. To display the right side of the patient on the right side of the image, set the `AnatomicalConvention` name-value argument to "neurological".
  - If the `Orientation` property value is "coronal", then `montage` displays coronal slices with the superior end of the patient on the top of the image and the right side of the patient on the left side of the image. To display the right side of the patient on the right side of the image, set the `AnatomicalConvention` name-value argument to "neurological".
  - If the `Orientation` property value is "sagittal", then `montage` displays sagittal slices with the superior end of the patient on the top of the image and the anterior side of the patient on the left side of the image.

If the patient coordinate system is undefined, then `montage` displays the data in the `Voxels` property of the `medicalVolume` object as a numeric array with default display settings. The patient coordinate system is undefined if the `PatientCoordinateSystem` property of the `medicalref3d` object in the `VolumeGeometry` property of the `medicalVolume` object is "unknown".

### **I — Multiframe image array**

numeric array

Multiframe image array, specified as one of these options:

- *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 (RGB) 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. Each matrix is of size *m-by-n* for binary or grayscale images or *m-by-n-by-3* for truecolor (RGB) images.

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.

If you specify a `medicalImage` object with a defined `Colormap` property as input, `montage`, by default, applies the colormap from this property to the image frames. Specify `map` to override the default behavior.

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: `montage(medVol, AnatomicalConvention="radiological")` displays the left side of the patient on the right side of the image.

**AnatomicalConvention — Anatomical display orientation convention**`"radiological"` (default) | `"neurological"`

Anatomical display orientation convention, specified as one of these values:

- `"radiological"` — Displays the left side of the patient on the right side of the image.
- `"neurological"` — Displays the left side of the patient on the left side of the image.

---

**Note** To use the `AnatomicalConvention` name-value argument, you must specify a `medicalVolume` object as the input.

---

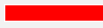



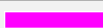
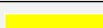


Data Types: char | string

**BackgroundColor — Background color**`"black"` (default) | RGB triplet | color name | short color name


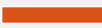





Background color, specified as specified as an RGB triplet, a color name, or a short color name. The `montage` object function fills all blank spaces with the background color, including the space specified by `BorderSize`. If you specify a background color, then the `montage` function renders the output 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 — Intensity display range**

1-by-2 vector

Intensity display range, specified as a two-element vector of the form [low high]. Intensity values less than or equal to `low` appear black. Intensity values greater than or equal to `high` appear white. If you specify an empty matrix ([ ]), then `montage` uses the minimum and maximum values in the image. The default display range depends on the input image:

- `medicalImage` object — `montage` uses the `WindowCenter` and `WindowWidth` properties to set the display range. If `WindowCenter` and `WindowWidth` are empty, then `montage` uses the minimum and maximum values in the image.

- `medicalVolume` object — `montage` uses the `WindowCenters` and `WindowWidths` properties to set the display range. If `WindowCenters` and `WindowWidths` are empty, then `montage` uses the minimum and maximum values in the image.
- Numeric array — The default is the range of the datatype of the array.

`DisplayRange` has no effect when displaying indexed images with colormaps or RGB images.

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

### Indices — Frames or slices to display

array of positive integers

Frames or slices to display in the montage, specified as an array of positive integers. If you specify a `medicalImage` object as input, then `montage` displays the corresponding frames of the image series. If you specify a `medicalVolume` object, `montage` displays the corresponding slices in the displayed plane. For nonmedical inputs, the `montage` function interprets the values as indices into the array `I` or into the cell array `filenames` or `imagelist`.

By default, the `montage` function displays all frames or image files.

Example: `Indices=1:4` creates a montage of the first four frames.

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 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]`. By default, `montage` arranges the images in the number of rows and columns that roughly form a square.

If you specify `NaN` or `Inf` for a particular dimension, then the `montage` function calculates the value of that dimension required to display all images in the montage, while preserving the specified dimension. For example, if `Size` is `[2 NaN]`, then the montage has two rows and the minimum number of columns required to display all images. When there is a mismatch between `Size` and the number of images (frames) specified, then the `montage` function displays as many of the images as possible while preserving the specified `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 function preserves the aspect ratio of each image, and fills any blank space with the background color `BackgroundColor`. By default, `montage` sets the thumbnail size based on your screen size and the number of frames to display.

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 so as to preserve the aspect ratio of the first image, given the specified size of the other dimension.

Data Types: `single` | `double`

## **Output Arguments**

### **img — Montage image**

Image object

Montage image, returned as an `Image` object. For more information on `Image` objects, see `Image Properties`.

## **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.
- Figure titles can appear cut off in the **Live Editor**. To ensure the whole title is visible, set the `PositionConstraint` property of the parent `Axes` object to `"outerposition"`. Update the property value after you call the `montage` function and before you call the `title` function.

```
I = imread("peppers.png");
montage({I,I})
ax = gca;
ax.PositionConstraint = "outerposition";
title("Peppers");
```

If you specify the parent axes using the `Parent` name-value argument, set the `PositionConstraint` property of the specified parent `Axes` object. For more details about axes position properties, see `"Control Axes Layout"`.

## **Version History**

**Introduced in R2023a**

## **See Also**

### **Objects**

`medicalImage` | `medicalVolume` | `sliceViewer`



**Functions**

imshow | volshow

**Topics**

“Choose Approach for Medical Image Visualization”

“Read, Process, and View Ultrasound Data”

“Display Medical Image Volume in Patient Coordinate System”

“Visualize 3-D Medical Image Data Using Medical Image Labeler”

## getAxesHandle

Get handle to axes in slice viewer

### Syntax

```
hAx = getAxesHandle(s)
```

### Description

`hAx = getAxesHandle(s)` returns the axes in the `sliceViewer` object `s`.

### Examples

#### Get Handle to Axes in Slice Viewer

Run this code to download a data set from the MathWorks® website and unzip the downloaded folder. The data set contains three CT volumes that are each saved as a directory of DICOM files. The size of the entire data set is approximately 81 MB. The scan used by this example, LungCT01, is stored in the folder specified by `dataFolder`.

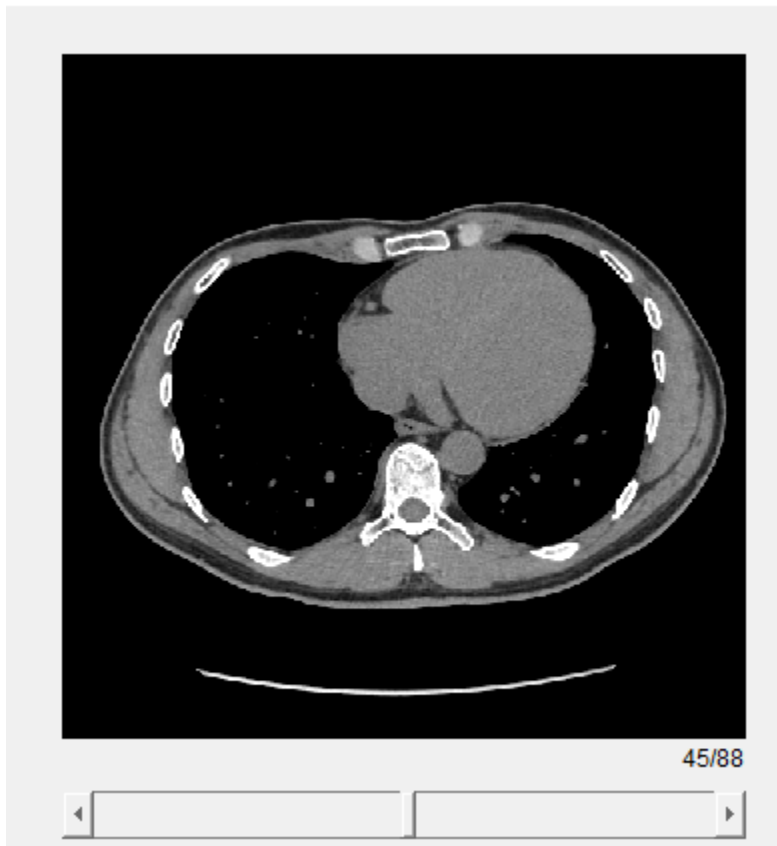
```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeDICOMData.zip");  
filepath = fileparts(zipFile);  
unzip(zipFile, filepath)  
dataFolder = fullfile(filepath, "MedicalVolumeDICOMData", "LungCT01");
```

Create a medical image volume object that contains the image data and spatial referencing information for the CT volume. The `Orientation` property indicates that the primary slice direction is "transverse".

```
medVol = medicalVolume(dataFolder);
```

View the transverse slices of the volume in the slice viewer. By default, the viewer uses the properties of `medVol` to scale anisotropic voxels, set the intensity display range, and orient the slices. The viewer opens on the center slice. Use the scroll bar to navigate to other slices.

```
sv = sliceViewer(medVol);
```



Get the handle to the axes that contains the image slices in the slice viewer.

```
hAx = getAxesHandle(sv)
```

```
hAx =
```

```
  Axes with properties:
```

```
      XLim: [0.5000 512.5000]
      YLim: [0.5000 512.5000]
      XScale: 'linear'
      YScale: 'linear'
      GridLineStyle: '-'
      Position: [30 57.5000 512 512]
      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 the slice viewer window, returned as an Axes object.

## **Version History**

**Introduced in R2023a**

### **See Also**

sliceViewer

# dicomFile

Process DICOM file

## Description

The `dicomFile` object and its object functions enable you to selectively read, parse, access, modify, and write the metadata of a DICOM file. You can also simultaneously access the pixel data of the DICOM file. Compared to the `dicominfo` and `dicomwrite` functions, which cannot parse and modify DICOM metadata selectively, and the `dicomread` function, which can read only pixel data from a DICOM file, the flexibility of the `dicomFile` object and its object functions makes it more efficient than these functions when processing a large number of DICOM files.

## Creation

### Syntax

```
dFile = dicomFile(filename)
dFile = dicomFile(filename,Dictionary=dict)
```

### Description

`dFile = dicomFile(filename)` creates a `dicomFile` object for the DICOM file `filename`. You can use the object to read, parse, access, modify, and write the metadata and pixel data of the DICOM file. `filename` is the absolute or relative path to the DICOM file, and sets the `Filename` property.

`dFile = dicomFile(filename,Dictionary=dict)` specifies the DICOM data dictionary `dict` to use to parse the DICOM metadata. `dict` is the absolute or relative path to the DICOM data dictionary, and sets the `Dictionary` property.

## Properties

### Filename — Name of DICOM file

string scalar | character vector

This property is read-only.

Name of the DICOM file, stored as a string scalar.

Data Types: `char` | `string`

### AttributeNames — List of DICOM file attribute names

vector of strings

This property is read-only.

List of DICOM file attribute names, stored as a vector of strings.

Data Types: `string`

**Dictionary — Path of DICOM data dictionary**

`fullfile(matlabroot,"toolbox","images","iptformats","dicom-dict.txt")` (default) | string scalar | character vector

This property is read-only.

Path of DICOM data dictionary, stored as a character vector.

Data Types: char | string

**DICOM Attribute Properties**

Some of the dependent attributes of a DICOM file are properties of the `dicomFile` object that are updated automatically when you modify the pixel data. Other attributes of the DICOM file are not properties of the `dicomFile` object.

**TransferSyntaxUID — TransferSyntaxUID attribute of DICOM file**

string scalar | character vector

TransferSyntaxUID attribute of the DICOM file, stored as a character vector.

Data Types: string | char

**Modality — Modality attribute of DICOM file**

string scalar | character vector

Modality attribute of the DICOM file, stored as a character vector.

Data Types: string | char

**MediaStorageSOPClassUID — MediaStorageSOPClassUID attribute of DICOM file**

string scalar | character vector

MediaStorageSOPClassUID attribute of the DICOM file, stored as a character vector.

Data Types: string | char

**SOPClassUID — SOPClassUID attribute of DICOM file**

string scalar | character vector

SOPClassUID attribute of the DICOM file, stored as a character vector.

Data Types: string | char

**SOPInstanceUID — SOPInstanceUID attribute of DICOM file**

string scalar | character vector

SOPInstanceUID attribute of the DICOM file, stored as a character vector.

Data Types: string | char

**FileMetaInformationVersion — FileMetaInformationVersion attribute of DICOM file**

two-element numeric vector

FileMetaInformationVersion attribute of the DICOM file, stored as a two-element numeric vector.

Data Types: uint8

**MediaStorageSOPInstanceUID — MediaStorageSOPInstanceUID attribute of DICOM file**

string scalar | character vector

MediaStorageSOPInstanceUID attribute of the DICOM file, stored as a character vector.

Data Types: string | char

**ImplementationClassUID — ImplementationClassUID attribute of DICOM file**

string scalar | character vector

ImplementationClassUID attribute of the DICOM file, stored as a character vector.

Data Types: string | char

**ImplementationVersionName — ImplementationVersionName attribute of DICOM file**

string scalar | character vector

ImplementationVersionName attribute of the DICOM file, stored as a character vector.

Data Types: string | char

**SamplesPerPixel — SamplesPerPixel attribute of DICOM file**

numeric scalar

SamplesPerPixel attribute of the DICOM file, stored as a numeric scalar.

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

**PhotometricInterpretation — PhotometricInterpretation attribute of DICOM file**

string scalar | character vector

PhotometricInterpretation attribute of the DICOM file, stored as a character vector.

Data Types: string | char

**Rows — Rows attribute of DICOM file**

numeric scalar

Rows attribute of the DICOM file, stored as a numeric scalar.

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

**Columns — Columns attribute of DICOM file**

numeric scalar

Columns attribute of the DICOM file, stored as a numeric scalar.

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

**BitsAllocated — BitsAllocated attribute of DICOM file**

numeric scalar

BitsAllocated attribute of the DICOM file, stored as a numeric scalar.

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

**BitsStored — BitsStored attribute of DICOM file**

numeric scalar

BitsStored attribute of the DICOM file, stored as a numeric scalar.

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

### **HighBit — HighBit attribute of DICOM file**

numeric scalar

HighBit attribute of the DICOM file, stored as a numeric scalar.

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

### **PixelRepresentation — PixelRepresentation attribute of DICOM file**

numeric scalar

PixelRepresentation attribute of the DICOM file, stored as a numeric scalar.

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

### **SmallestImagePixelValue — SmallestImagePixelValue attribute of DICOM file**

numeric scalar

SmallestImagePixelValue attribute of the DICOM file, stored as a numeric scalar.

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

### **LargestImagePixelValue — LargestImagePixelValue attribute of DICOM file**

numeric scalar

LargestImagePixelValue attribute of the DICOM file, stored as a numeric scalar.

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

## **Object Functions**

isAttribute	Check if specified attribute is present in DICOM file
getAttribute	Get value of specified DICOM attribute
getPixelData	Get pixel data of DICOM file
write	Create selectively modified copy of DICOM file

## **Examples**

### **Create Denoised Image DICOM File from Ultrasound DICOM File**

#### **Load and Inspect Ultrasound DICOM File**

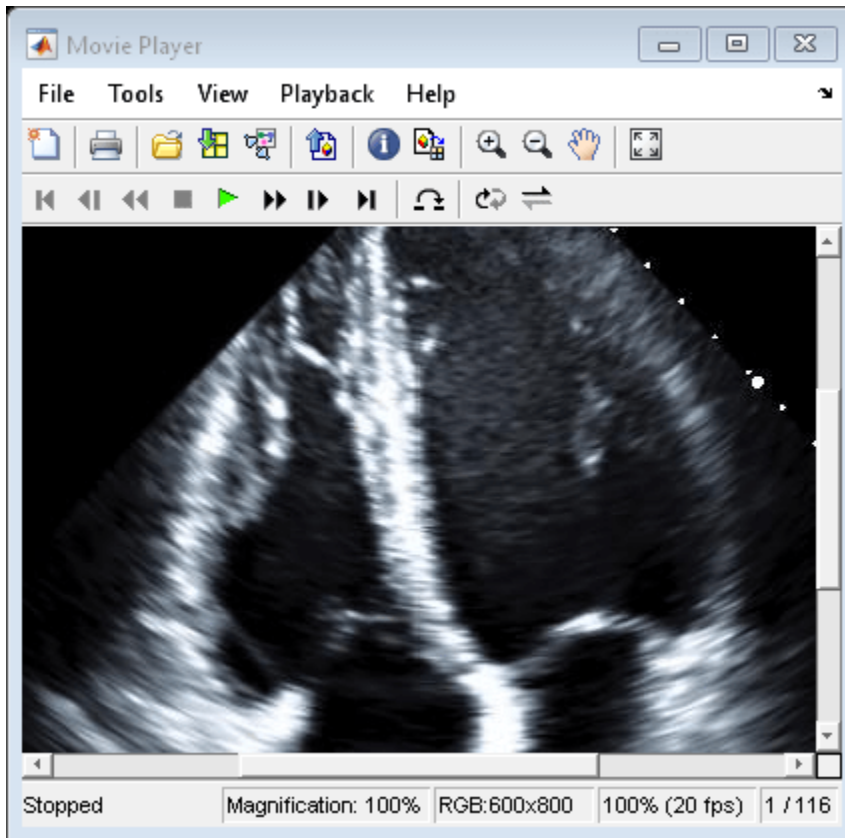
Import an ultrasound sequence DICOM file into the workspace.

```
dFile = dicomFile("heartUltrasoundSequenceVideo.dcm");
```

Get the pixel data of the ultrasound sequence. Visualize and inspect the pixel data.

```
pixelData = getPixelData(dFile);  
figure  
imshow(pixelData)
```





Observe that the ultrasound sequence DICOM file has 600 rows and 800 columns.

```
getAttribute(dFile, "Rows")
```

```
ans = uint16
      600
```

```
getAttribute(dFile, "Columns")
```

```
ans = uint16
      800
```

### Modify Pixel Data and Metadata of Ultrasound Sequence

Crop the pixel data to reduce its size.

```
cropPixelData = pixelData(51:end,51:end,:,:);
```

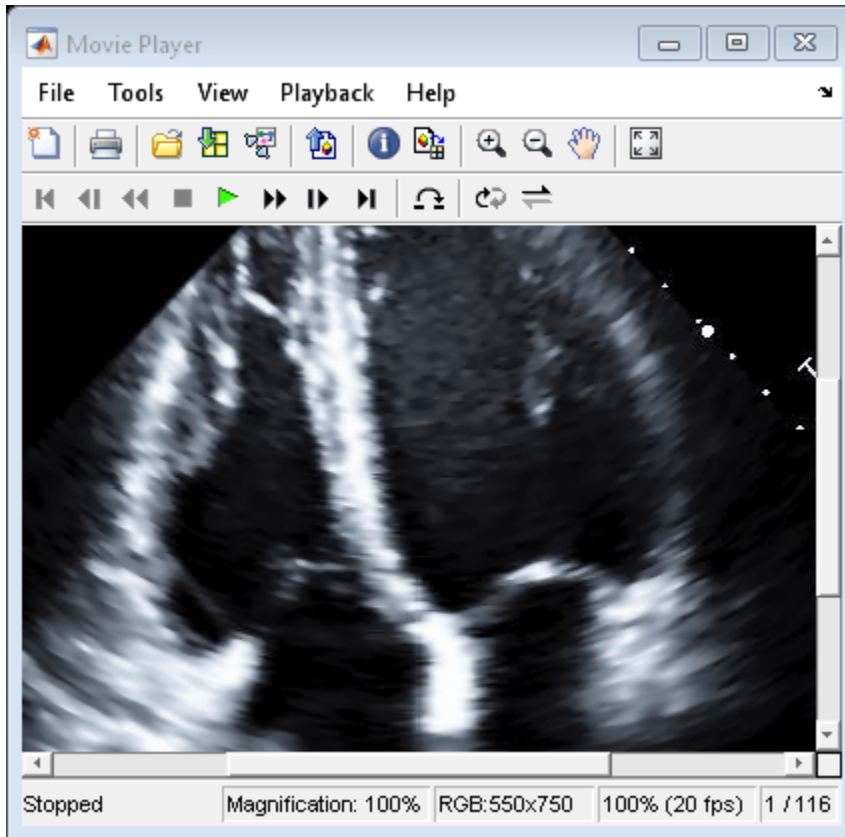
Denoise the cropped pixel data using a speckle filter. For denoising, convert each frame to HSV color space, denoise the value (V) channel of the frame, and then convert the denoised HSV image back to RGB color space. Visualize and inspect the denoised pixel data.

```
[h,w,c,d] = size(cropPixelData);
denoisedPixelData = zeros(h,w,c,d);
for i = 1:d
    [H,S,V] = rgb2hsv(cropPixelData(:,:,i));
    denoisedV = specklefilt(V);
    denoisedPixelData(:,:,i) = hsv2rgb(H,S,denoisedV);
```

```

end
figure
imshow(denoisedPixelData)

```



Check if `StudyDescription` is an attribute of the ultrasound sequence DICOM file.

```
isAttribute(dFile,"StudyDescription")
```

```
ans = logical
      1
```

Get the value of the `StudyDescription` attribute of the ultrasound sequence DICOM file.

```
studyDescription = getAttribute(dFile,"StudyDescription")
```

```
studyDescription =
'ADULT ECHO TTE'
```

Create a study description for the new DICOM file based on `studyDescription`. Then create a structure named `metadata` with a `StudyDescription` field that contains your new study description.

```
denoisedStudyDescription = [studyDescription ' DENOISED'];
metadata = struct("StudyDescription",denoisedStudyDescription)
```

```
metadata = struct with fields:
  StudyDescription: 'ADULT ECHO TTE DENOISED'
```

### Write Modified Pixel Data and Metadata to New DICOM File

Create a modified copy of the ultrasound sequence DICOM file, replacing its pixel data with the denoised pixel data and its metadata with the structure containing your new study description.

```
write(dFile, "heartUltrasoundSequenceVideo_denoised.dcm", denoisedPixelData, metadata)
```

### Verify New DICOM File

To check whether the new DICOM file contains the denoised pixel data and the updated metadata, import it into the workspace.

```
dFileDenoised = dicomFile("heartUltrasoundSequenceVideo_denoised.dcm");
```

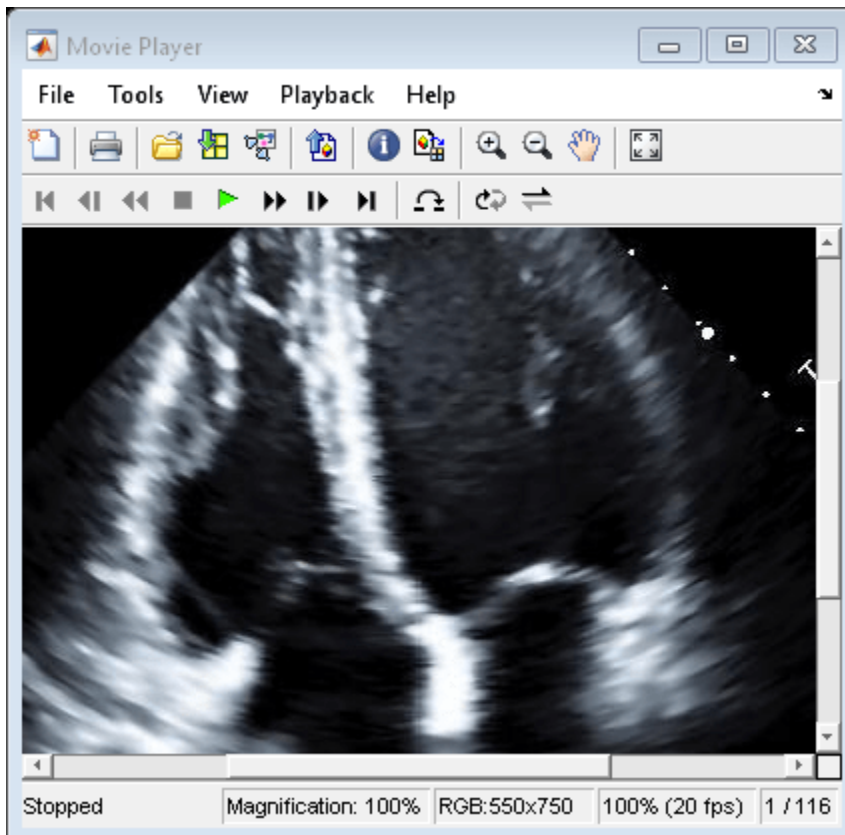
Get the study description, to verify that it matches your updated study description.

```
getAttribute(dFileDenoised, "StudyDescription")
```

```
ans =  
'ADULT ECHO TTE DENOISED'
```

Get the pixel data of the new DICOM file. Visualize the pixel data to verify that it matches the denoised pixel data of the ultrasound sequence.

```
pixelDataDenoised = getPixelData(dFileDenoised);  
figure  
imshow(pixelDataDenoised)
```



Check that the rows and columns of the new DICOM file have automatically updated to reflect the size of the cropped image.

```
getAttribute(dFileDenoised, "Rows")
```

```
ans = uint16  
    550
```

```
getAttribute(dFileDenoised, "Columns")
```

```
ans = uint16  
    750
```

## Limitations

Noncompliant DICOM files that switch value representation (VR) modes incorrectly cannot be read using the `dicomFile` object.

## Version History

**Introduced in R2023a**

### See Also

`dicominfo` | `dicomread` | `dicomwrite`

### External Websites

DICOM Standard

# groundTruthMedical

Ground truth label data for medical images

## Description

The `groundTruthMedical` object contains information about the data sources, label definitions, and label data for a collection of medical image data. You can import or export a `groundTruthMedical` object to or from the **Medical Image Labeler** app.

## Creation

The **Medical Image Labeler** app automatically creates a `groundTruthMedical` object in the Session folder for an app session. The app saves the object as a MAT file. To manually export a `groundTruthMedical` object to a specific file location, on the **Home** tab of the app toolstrip, select **Export** and, under **Ground Truth**, select **To File**. To create a `groundTruthMedical` object programmatically, use the `groundTruthMedical` function.

## Syntax

```
gTruthMed = groundTruthMedical(dataSource, labelDefinitions, labelData)
```

### Description

`gTruthMed = groundTruthMedical(dataSource, labelDefinitions, labelData)` creates a `groundTruthMedical` object, `gTruthMed`, that you can import into the **Medical Image Labeler** app.

- `dataSource` specifies the source of the unlabeled ground truth images and sets the `DataSource` property.
- `labelDefs` specifies the label definitions of the ground truth data and sets the `LabelDefinitions` property.
- `labelData` specifies the location of image label data and sets the `LabelData` property.

## Properties

### **DataSource** — Source of ground truth data

`VolumeSource` object | `ImageSource` object

This property is read-only.

Source of the ground truth data, specified as a `VolumeSource` object or an `ImageSource` object. The data source object specifies the locations of the unlabeled medical image files from which the ground truth data is labeled. You must set this property at object creation by using the `dataSource` input argument.

### **LabelDefinitions** — Label definitions

table

This property is read-only.

Label definitions, specified as a table. To create this table, use one of these options:

- In the **Medical Image Labeler** app, create label definitions, and then export them as a MAT file by clicking **Export** and, under **Label Definitions**, selecting **To File**.
- Manually create the label definitions table at the MATLAB command line.

This table describes the required and optional columns of the `LabelDefinitions` table.

Column	Description	Required or Optional
Name	String scalars or character vectors, each specifying the name of a label definition.	Required  All names must be unique and valid MATLAB variable names. For more details about valid variable names, see “Variable Names”.
PixelLabelID	Numeric integer scalars in the range [1, 255], each specifying a numeric label ID.	Required  All pixel label IDs must be unique.
LabelColor	RGB triplets that specify the label colors. Values must be in the range [0, 1].	Optional  When you define labels in the <b>Medical Image Labeler</b> app, you must specify a color. Therefore, an exported label definitions table always includes this column.  When you create label definitions programmatically, you can exclude the <code>LabelColor</code> column. When you create a <code>groundTruthMedical</code> object, this column is automatically added with default values. The default colors are assigned in the same order as in the <b>Medical Image Labeler</b> app.

You must set this property at object creation by using the `labelDefs` input argument.

Data Types: `table`

### **LabelData — Label data file names**

*n*-by-1 string array

This property is read-only.

Label data file names, specified as an *n*-by-1 string array, where *n* is the number of images or image volumes specified by `DataSource`. Each element of `LabelData` contains the name of the label data file for the corresponding image or volume in the data source.

- If the data source is a `VolumeSource` object, then the label data must be stored as NIfTI files.
- If the data source is an `ImageSource` object, then the label data must be stored as MAT files.

- If no labels exist for an image or image volume, specify the corresponding element of `LabelData` as an empty string, "".

You must set this property at object creation by using the `labelData` input argument.

Data Types: `string`

## Object Functions

`changeFilePaths` Change file paths in ground truth data for medical images  
`merge` Merge two or more `groundTruthMedical` objects

## Examples

### Create Ground Truth for Medical Image Data

Create ground truth data for three 2-D X-rays of forearms, stored as DICOM files, and their corresponding label images, stored as MAT files. The images are attached to this example as supporting files. Specify the data directory as the current example directory.

```
dataFolder = pwd;
```

Create a table of the X-ray image file information by using the `dicomCollection` function, and specify the table as the `sourceTable` for an `ImageSource` object.

```
sourceTable = dicomCollection(dataFolder);
dataSource = medical.labeler.loading.ImageSource(sourceTable);
```

Define the labels used to specify the ground truth. Specify the name, display color, and numeric label ID for the bone label data.

```
labelName = "radiusBone";
labelColor = [1 0 0];
labelId = 1;
variableNames = ["Name", "LabelColor", "PixelLabelID"];
labelDefs = table(labelName, labelColor, labelId, VariableNames=variableNames)
```

```
labelDefs=1x3 table
      Name      LabelColor      PixelLabelID
      _____      _____      _____
      "radiusBone"      1    0    0      1
```

Specify the file paths to the label images for each of the images in the data source as a string array. For ground truth data containing an `ImageSource` object, the label images must be in the MAT file format.

```
labelData = [fullfile(dataFolder, "forearmXrayLabels1.mat");
             fullfile(dataFolder, "forearmXrayLabels2.mat");
             fullfile(dataFolder, "forearmXrayLabels3.mat")];
```

Create a `groundTruthMedical` object.

```
gTruthMed = groundTruthMedical(dataSource, labelDefs, labelData)
```

```
gTruthMed =
  groundTruthMedical with properties:

    DataSource: [1x1 medical.labeler.loading.ImageSource]
    LabelData: [3x1 string]
    LabelDefinitions: [1x3 table]
```

### Create Ground Truth for Medical Image Volume Data

Create ground truth data for two chest CT volumes and their corresponding label images, stored in the NIfTI file format. The files are a subset of the Medical Segmentation Decathlon data set [1 on page 1-157]. Download the `MedicalVolumeNIFTIData.zip` file from the MathWorks® website, then unzip the file. The size of the data file is approximately 76 MB.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeNIFTIData.zip");
filePath = fileparts(zipFile);
unzip(zipFile,filePath)
dataFolder = fullfile(filePath,"MedicalVolumeNIFTIData");
```

Create a `VolumeSource` object specifying the two CT volumes.

```
filePath1 = fullfile(dataFolder,"lung_027.nii.gz");
filePath2 = fullfile(dataFolder,"lung_043.nii.gz");
source = {filePath1; filePath2};
dataSource = medical.labeler.loading.VolumeSource(source);
```

Define the labels used to specify the ground truth. Specify the name, display color, and numeric label ID for the tumor label data.

```
labelName = "tumor";
labelColor = [1 0 0];
labelId = 1;
variableNames = ["Name","LabelColor","PixelLabelID"];
labelDefs = table(labelName,labelColor,labelId,VariableNames=variableNames)
```

```
labelDefs=1x3 table
    Name      LabelColor      PixelLabelID
    _____  _____  _____
    "tumor"    1     0     0          1
```

Specify the file paths to the label images for each of the images in the data source as a string array. If a data source image does not have label data, specify the corresponding element of the label data array as an empty string.

```
labelDataFile = fullfile(dataFolder,"LabelData","lung_027.nii.gz");
labelData = [labelDataFile ""];
```

Create a `groundTruthMedical` object.

```
gTruthMed = groundTruthMedical(dataSource,labelDefs,labelData)
```

```
gTruthMed =
  groundTruthMedical with properties:
```



```
DataSource: [1×1 medical.labeler.loading.VolumeSource]  
LabelData: [2×1 string]  
LabelDefinitions: [1×3 table]
```

[1] Medical Segmentation Decathlon. "Lung." Tasks. Accessed May 10, 2018. <http://medicaldecathlon.com/>.

The Medical Segmentation Decathlon data set is provided under the CC-BY-SA 4.0 license. All warranties and representations are disclaimed. See the license for details.

## **Version History**

**Introduced in R2022b**

### **See Also**

ImageSource | VolumeSource

# ImageSource

Source of 2-D medical image data for `groundTruthMedical` object

## Description

An `ImageSource` object defines the source of ground truth data for medical images or series of images related by time, such as ultrasound data. Use this object to specify the data sources for a `groundTruthMedical` object. Each source must be a single DICOM or NIFTI file.

## Creation

When you export labels from a **Medical Image Labeler** app image session, the `DataSource` property of the exported `groundTruthMedical` object contains an `ImageSource` object.

To create an `ImageSource` object programmatically, such as when programmatically creating a `groundTruthMedical` object, use the `medical.labeler.loading.ImageSource` function.

## Syntax

```
imgSource = medical.labeler.loading.ImageSource(source)
imgSeqSource = medical.labeler.loading.ImageSource(sourceTable)
```

### Description

`imgSource = medical.labeler.loading.ImageSource(source)` creates an `ImageSource` object for loading the 2-D medical image data stored in the files specified by `source`.

`imgSeqSource = medical.labeler.loading.ImageSource(sourceTable)` creates an `ImageSource` object for loading the image files specified in a table, `sourceTable`, returned by the `dicomCollection` function.

### Input Arguments

#### **source** — Source file names

*n*-by-1 string array | *n*-by-1 cell array of character vectors

Source file names, specified as an *n*-by-1 string array or an *n*-by-1 cell array of character vectors. *n* is the total number of images or series of images related by time to store in the `groundTruthMedical` object. Each name specifies a single DICOM or NIFTI file containing 2-D image data that is readable by a `medicalImage` object.

Data Types: `string` | `cell`

#### **sourceTable** — Table of source file names

table

Table of source file names, specified as a table returned by the `dicomCollection` function. Each row in `sourceTable` must specify a valid DICOM series that contains a 2-D image or series of

images related by time that is readable by a `medicalImage` object. The file name for each DICOM series is stored in the `FileNames` column of the table returned by `dicomCollection`.

Data Types: `table`

## Properties

### Source — Source of ground truth data

*n*-by-1 string array

This property is read-only.

Source of the ground truth data, specified as an *n*-by-1 string array, where *n* is the total number of images or series of images related by time to store in the `groundTruthMedical` object. Each element in the string array specifies the file name for one image or image series.

Data Types: `string`

## Examples

### Create Medical Image Data Source from File

Specify the name of a 2-D X-ray image. The file is attached to this example as a supporting file.

```
filename = fullfile(pwd, "forearmXrayImage1.dcm");
```

Create an `ImageSource` object that specifies the X-ray image.

```
imgSource = medical.labeler.loading.ImageSource(filename);
```

Verify that the filename is stored in the `Source` property of the data source object.

```
imgSource.Source;
```

### Create Image Data Source from DICOM Collection

Specify the name of the directory containing this example, which includes three 2-D X-ray images of a forearm stored as DICOM files.

```
dataFolder = pwd;
```

Gather the details about the DICOM files in the directory into a table by using the `dicomCollection` function.

```
sourceTable = dicomCollection(dataFolder);
```

Create an `ImageSource` object that specifies the files in `sourceTable`.

```
imgSource = medical.labeler.loading.ImageSource(sourceTable);
```

Verify that each element of the `Source` property contains a filename from the `FileNames` column of `sourceTable`.

`imgSource.Source;`

## **Version History**

**Introduced in R2022b**

### **See Also**

`groundTruthMedical` | `VolumeSource`

# medicalImage

2-D medical image pixel data and file metadata

## Description

A `medicalImage` object stores the pixel data and metadata for a 2-D medical image or image sequence contained in a single DICOM file. An image sequence is a series of 2-D image frames related by time, such as an ultrasound video.

## Creation

### Syntax

```
medImage = medicalImage(dirname)
medImage = medicalImage(filename)
medImage = medicalImage(sourceTable)
medImage = medicalImage(sourceTable, rowname)
medImage = medicalImage(imds)
medImage = medicalImage(pixels, info)
```

### Description

`medImage = medicalImage(dirname)` creates a `medicalImage` object for the single DICOM file in the directory `dirname`.

`medImage = medicalImage(filename)` creates a `medicalImage` object for the image contained in the single DICOM file `filename`.

`medImage = medicalImage(sourceTable)` creates a `medicalImage` object for the image listed in `sourceTable`. The table must contain only one row that specifies the metadata for a 2-D DICOM image.

`medImage = medicalImage(sourceTable, rowname)` creates a `medicalImage` object for the image listed in the row `rowname` of `sourceTable`. Use this syntax to specify one row of a multirow table by name or by its numeric index.

`medImage = medicalImage(imds)` creates a `medicalImage` object for the image specified by the image datastore object `imds`.

`medImage = medicalImage(pixels, info)` creates a `medicalImage` object by specifying the image data `pixels` and `info`, a metadata structure returned by the `dicominfo` function. `pixels` sets value of the `Pixels` property.

### Input Arguments

**dirname** — Name of directory containing medical image data

string scalar | character vector

Name of the directory containing the medical image data, specified as a string scalar or a character vector. `dirname` is the name of a directory that contains one DICOM file that specifies one 2-D image or sequence of 2-D images related by time, such as an ultrasound video.

**filename — Name of file containing medical image data**

string scalar | character vector

Name of the file containing the medical image data, specified as a string scalar or a character vector. `filename` can specify a single DICOM file that contains a 2-D image or series of 2-D images related by time, such as an ultrasound video.

**sourceTable — Collection of DICOM file metadata**

table

Collection of DICOM file metadata, specified as a table returned by the `dicomCollection` function.

**rowname — Name or index of table row**

string scalar | character vector | positive integer

Name or index of the table row to read from `sourceTable`, specified as a string scalar, character vector, or positive integer scalar. Specify `rowname` as a string scalar or character vector to specify the row by name. Specify `rowname` as a positive integer scalar to specify the row by its numeric index.

**imds — Datastore containing one DICOM file**

ImageDatastore object

Datastore containing one DICOM file, specified as an `imageDatastore` object.

**info — DICOM file metadata**

structure

DICOM file metadata, specified as a structure returned by the `dicominfo` function.

## Properties

**Pixels — Image pixel values**

*m*-by-*n*-by-*t*-by-*c* numeric array

Image pixel values, specified as an *m*-by-*n*-by-*t*-by-*c* numeric array, where *m* and *n* are the spatial dimensions of the 2-D image or image frames, *t* is the number of image frames related by time, and *c* is the number of color channels in each frame. If the file contains the `RescaleIntercept` and `RescaleSlope` metadata attributes, then `medicalImage` rescales the intensity values based on them.

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

**Colormap — Colormap**

*j*-by-3 numeric matrix | []

Colormap, specified as one of these options:

- If the source is an indexed image, then specify `Colormap` as a *j*-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 the source is a grayscale or RGB image, then `ColorMap` is empty (`[]`).

Data Types: `double`

### **SpatialUnits — Real-world spatial units**

`"mm" | "unknown"`

This property is read-only.

Real-world spatial units, specified as one of these options:

- If the pixel spacing is specified in the file metadata, then the spatial units value is `"mm"`.
- If the metadata does not specify the pixel spacing information, then `SpatialUnits` is `"unknown"`.

Data Types: `string`

### **FrameTime — Number of milliseconds between frames**

numeric scalar | `[]`

This property is read-only.

Number of milliseconds between frames, specified as one of these options:

- If the `Frame Time` metadata attribute is specified in the file, then the frame time value is specified as a numeric scalar extracted from the metadata.
- If the metadata does not specify the frame time, then `FrameTime` is empty (`[]`).

Data Types: `double`

### **NumFrames — Number of frames**

numeric scalar

This property is read-only.

Number of frames in the image data, specified as a numeric scalar.

Data Types: `double`

### **PixelSpacing — Spacing between pixel centers**

`[1 1]` (default) | 2-element row vector

This property is read-only.

Spacing between pixel centers, specified as a 2-element vector of the form  $[\Delta x \Delta y]$ . The  $\Delta x$  and  $\Delta y$  values are the distances between adjacent pixels in the  $x$ - and  $y$ -directions, respectively, in the units specified by the `SpatialUnits` property. If the file metadata does not specify the pixel spacing, the default value is `[1 1]`.

Data Types: `double`

### **Modality — Imaging modality used to capture image data**

`"unknown"` (default) | string scalar

This property is read-only.

Imaging modality used to capture image data, specified as a string scalar. The modality is extracted from the file metadata if it is present. Common values include, but are not limited to, "DX" for digital radiography, "US" for ultrasound, "XA" for X-ray angiography, and "MG" for mammography. If the file metadata does not specify the modality, then the default value is "unknown".

Data Types: `string`

### **WindowCenter — Center of display range window**

numeric scalar | []

This property is read-only.

Center of the display range window, specified as one of these options:

- If the `Window Center` attribute is specified in the file metadata, then the `WindowCenter` value is specified as a numeric scalar extracted from the metadata.
- If the metadata does not specify the window center, then `WindowCenter` is empty ( []).

Data Types: `double`

### **WindowWidth — Width of display range window**

numeric scalar | []

This property is read-only.

Width of the of display range window, specified as one of these options:

- If the `Window Width` attribute is specified in the file metadata, then the `WindowWidth` value is specified as a numeric scalar extracted from the metadata.
- If the metadata does not specify the window width, then `WindowWidth` is empty ( []).

Data Types: `double`

## **Object Functions**

<code>extractFrame</code>	Extract pixel data for one frame of 2-D medical image series
<code>implay</code>	View 2-D medical image series in Video Viewer app
<code>montage</code>	Display multiple image frames as rectangular montage

## **Examples**

### **Create Medical Image Object from Filename**

Specify the name of a 2-D X-ray image. The image is attached to this example as a supporting file.

```
filename = "forearmXrayImage1.dcm";
```

Create a `medicalImage` object for the file.

```
medImage = medicalImage(filename)
```

```
medImage =  
    medicalImage with properties:
```

```
        Pixels: [1540x1250 uint16]
```



```
Colormap: []  
SpatialUnits: "mm"  
FrameTime: []  
NumFrames: 1  
PixelSpacing: [0.1390 0.1390]  
Modality: 'DX'  
WindowCenter: 2048  
WindowWidth: 4096
```

The image pixel data is stored in the `Pixels` property. Display the image stored in `medImage`.

```
imshow(medImage.Pixels, [])
```



### Create Medical Image Object from DICOM Collection

Specify the name of the directory containing this example, which includes an echocardiogram image series stored as a DICOM file.

```
dirname = pwd;
```

Gather the details about the DICOM files in the directory into a table by using the `dicomCollection` function.

```
sourceTable = dicomCollection(dirname);
```

Create a `medicalImage` object for the echocardiogram sequence.

```
medImage = medicalImage(sourceTable)
```

```
medImage =
  medicalImage with properties:
        Pixels: [600x800x3x3 uint8]
        Colormap: []
        SpatialUnits: "unknown"
        FrameTime: []
        NumFrames: 3
        PixelSpacing: [1 1]
        Modality: 'OT'
        WindowCenter: []
        WindowWidth: []
```

### Create Medical Image Object from Specified Row of DICOM Collection

Specify the name of the directory containing this example, which includes three 2-D X-ray images of a forearm.

```
dataFolder = pwd;
```

Gather the details about the DICOM files in the directory into a table by using the `dicomCollection` function. The table contains three rows corresponding to the three DICOM series.

```
sourceTable = dicomCollection(dataFolder);
```

Create a medical image object for the second X-ray image by specifying the DICOM collection table and a numeric row index.

```
medImage = medicalImage(sourceTable,2)
```

```
medImage =
  medicalImage with properties:
        Pixels: [1932x1492 uint16]
        Colormap: []
        SpatialUnits: "mm"
        FrameTime: []
        NumFrames: 1
        PixelSpacing: [0.1390 0.1390]
```

```
Modality: 'DX'  
WindowCenter: 2048  
WindowWidth: 4096
```

### Create Medical Image Object from Image Datastore

Specify the name of the directory containing this example, which includes an echocardiogram image series stored as a single DICOM file.

```
dataFolder = pwd;
```

Create an image datastore containing the DICOM file in the `dataFolder` directory. Specify a custom read function to read the DICOM file.

```
dicomds = imageDatastore(dataFolder, ...  
    FileExtensions=".dcm",ReadFcn=@(x) dicomread(x));
```

Create a medical image object for the echocardiogram series.

```
medImage = medicalImage(dicomds)
```

```
medImage =  
    medicalImage with properties:  
  
        Pixels: [600x800x3x3 uint8]  
        Colormap: []  
        SpatialUnits: "unknown"  
        FrameTime: []  
        NumFrames: 3  
        PixelSpacing: [1 1]  
        Modality: 'OT'  
        WindowCenter: []  
        WindowWidth: []
```

### Create Medical Image Object from Pixels and Metadata

Specify the name of a 2-D X-ray image. The file is attached to this example as a supporting file.

```
filename = "forearmXrayImage1.dcm";
```

Read the metadata from the DICOM file by using the `dicominfo` function.

```
info = dicominfo(filename);
```

Create a `medicalImage` object for the file.

```
medImage = medicalImage(filename)
```

```
medImage =  
    medicalImage with properties:
```

```

    Pixels: [1540x1250 uint16]
    Colormap: []
    SpatialUnits: "mm"
    FrameTime: []
    NumFrames: 1
    PixelSpacing: [0.1390 0.1390]
    Modality: 'DX'
    WindowCenter: 2048
    WindowWidth: 4096

```

The image pixel data is stored in the `Pixels` property. Apply a smoothing filter to the pixel values and save the smoothed values as a new image array, `imSmooth`.

```

sigma = 2;
imSmooth = imgaussfilt(medImage.Pixels,sigma);

```

Create a new `medicalImage` object that contains the smoothed pixel values. To maintain the same spatial information as the original image, specify the metadata structure `info`.

```

medImageSmoothed = medicalImage(imSmooth,info)

```

```

medImageSmoothed =
    medicalImage with properties:

        Pixels: [1540x1250 uint16]
        Colormap: []
        SpatialUnits: "mm"
        FrameTime: []
        NumFrames: 1
        PixelSpacing: [0.1390 0.1390]
        Modality: 'DX'
        WindowCenter: 2048
        WindowWidth: 4096

```

## Version History

Introduced in R2022b

### See Also

`ImageSource` | `medicalVolume` | `specklefilt`

### Topics

“Choose Approach for Medical Image Visualization”  
 “Read, Process, and View Ultrasound Data”

## medicalref3d

Spatial referencing information for 3-D medical image volumes

### Description

A `medicalref3d` object stores the relationship between the intrinsic coordinate system anchored to the columns, rows, and planes of a medical image volume and the real-world patient coordinate system.

The `medicalref3d` object handles affine and non-affine medical image volumes. For non-affine volumes, the object stores information about the spacing and orientation of each slice in each dimension. You can also use the `medicalref3d` object to access and update the mapping between the patient coordinate system axes and the left-right, superior-inferior, and anterior-posterior anatomical axes.

### Creation

You can create a `medicalref3d` object in these ways:

- `medicalVolume` — Creates a `medicalref3d` object, stored in the `VolumeGeometry` property.
- The `medicalref3d` function described here.

### Syntax

```
R = medicalref3d(volumeSize,tform)
R = medicalref3d(volumeSize,position,pixelSpacing,cosines)
R = medicalref3d(volumeSize,position,voxelDistances)
```

### Description

`R = medicalref3d(volumeSize,tform)` creates a `medicalref3d` object and sets the `VolumeSize` property. `tform` is an `affinetform3d` object that specifies an affine image volume.

`R = medicalref3d(volumeSize,position,pixelSpacing,cosines)` specifies the spatial referencing information for an affine or non-affine image volume by setting the `Position` and `PixelSpacing` properties, and specifies the orientation of each slice using the `cosines` argument.

`R = medicalref3d(volumeSize,position,voxelDistances)` specifies the spatial referencing information for an affine or non-affine image volume by setting the `Position` and `VoxelDistances` properties.

### Input Arguments

**tform — Geometric transformation between image data array and patient coordinate system**

`affinetform3d` object

Geometric transformation between the image data array and the patient coordinate system, specified as an `affinetform3d` object. The `A` property of `tform` contains a 4-by-4 3-D transformation matrix

that maps triplets of pixel indices in the order (*column, row, slice*) to (*x, y, z*) triplets of patient coordinates in real-world units.

### **cosines — Orientation of each slice in patient coordinate system**

2-by-3-by-*p* numeric array

Orientation of each slice in the patient coordinate system, specified as a 2-by-3-by-*p* numeric array, where *p* is the number of slices in the image volume along the third dimension. Each 2-by-3 matrix specifies the orientation of one slice:

- The first row specifies the direction cosines of the first dimension, which is aligned with the rows of the data array, relative to the *x*-, *y*-, and *z*-axes of the patient coordinate system.
- The second row specifies the direction cosines of the second dimension, which is aligned with the columns of the data array, relative to the *x*-, *y*-, and *z*-axes of the patient coordinate system.

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

## **Properties**

### **VolumeSize — Number of voxels in each dimension of data array**

3-element row vector of positive integers

Number of voxels in each dimension of the data array, specified as a 3-element row vector of positive integers. The three elements of `VolumeSize` specify the number of rows, columns, and slices, respectively, in the image data. You must set this property at object creation by using the `volumeSize` input argument. Otherwise, this property is read-only.

Data Types: `double`

### **Position — Patient coordinates of pixel in first index of each slice in data array**

*p*-by-3 numeric matrix

Patient coordinates of the pixel in the first index of each slice in the data array, specified as a *p*-by-3 numeric matrix. *p* is the number of slices in the image volume along the third dimension. Each row specifies the *xyz*-coordinates, in real-world units, of the center of the first pixel of the corresponding slice. The first pixel is the pixel in the first row and column of the data array. You can set this property at object creation directly, by using the `position` input argument, or indirectly, by using the `transform` input argument. Otherwise, this property is read-only.

Data Types: `double`

### **VoxelDistances — 3-D displacement vectors between adjacent voxels**

3-element cell array of *p*-by-3 numeric matrices | 3-element cell array of 1-by-3 numeric matrices

3-D displacement vectors between adjacent voxels, specified as a 3-element cell array of *p*-by-3 numeric matrices or 1-by-3 numeric matrices. *p* is the number of slices in the image volume along the third dimension. `VoxelDistances` defines the 3-D mapping between the intrinsic coordinate system, in voxels, and the patient coordinate system, in real-world units. Displacements can be positive or negative.

- `VoxelDistances{1}` specifies the real-world displacements  $[\Delta x \ \Delta y \ \Delta z]$  corresponding to a  $[1 \ 0 \ 0]$  voxel displacement.
- `VoxelDistances{2}` specifies the real-world displacements  $[\Delta x \ \Delta y \ \Delta z]$  corresponding to a  $[0 \ 1 \ 0]$  voxel displacement.

- `VoxelDistances{3}` specifies the real-world displacements  $[\Delta x \Delta y \Delta z]$  corresponding to a  $[0 \ 0 \ 1]$  voxel displacement.

For example, `VoxelDistances{2}(1, :)` specifies the vector between voxels of the first slice when moving along the second dimension of the data array.

If the image volume is non-affine, or if the `medicalref3d` object was created as the `VolumeGeometry` property value of a `medicalVolume` object for a DICOM file, then `VoxelDistances` specifies displacement vectors for each slice in a separate row. Otherwise, each element of `VoxelDistances` simplifies to a 1-by-3 matrix.

You can set this property at object creation directly, by using the `voxelDistances` input argument, or indirectly, by using the `tform`, `position`, `pixelSpacing`, or `cosines` input argument. Otherwise, this property is read-only.

Data Types: `cell`

### **PatientCoordinateSystem — Orientation convention of patient coordinate system relative to anatomical axes**

"unknown" (default) | "LPS+" | "LAS+" | "RAS+"

Orientation convention of the patient coordinate system relative to the anatomical axes, specified as "LPS+", "LAS+", "RAS+", or "unknown". For "LPS+", "LAS+", and "RAS+", the first three characters indicate the positive direction of the *x*-, *y*-, and *z*-axes of the patient coordinate system, respectively.

- The positive direction of the *x*-axis points left ("L") or right ("R").
- The positive direction of the *y*-axis points anterior ("A") or posterior ("P").
- The positive direction of the *z*-axis points inferior ("I") or superior ("S").
- "+" indicates that values increase in the stated direction.

For example, "LPS+" specifies a patient coordinate system with the *x*-, *y*-, and *z*-axes positive in the left, posterior, and superior directions, respectively.

The default value is "unknown". For a `medicalref3d` object created by the `medicalVolume` function, the `PatientCoordinateSystem` is set based on the corresponding attribute of the file metadata, if present.

Data Types: `char` | `string`

### **PixelSpacing — Spacing between voxel centers in first two dimensions of each slice**

*p*-by-2 matrix of positive values | 2-element vector of positive values

Spacing between voxel centers in the first two dimensions of each slice, specified as a *p*-by-2 matrix or 2-element vector of positive values. *p* is the number of slices in the image volume along the third dimension. Each row specifies the distances between voxel centers, for one slice, in the form  $[xSpacing \ ySpacing]$ . Distances are in real-world units such as millimeters.

If the image volume is non-affine, or if the `medicalref3d` object was created as the `VolumeGeometry` property value of a `medicalVolume` object for a DICOM file, then the `PixelSpacing` property specifies spacing for each slice in a separate row. Otherwise, `PixelSpacing` simplifies to a 2-element vector.



You can set this property at object creation directly, by using the `pixelSpacing` input argument, or indirectly, by using the `tform` or `voxelDistances` input argument. Otherwise, this property is read-only.

Data Types: `double`

### **IsAffine — Spatial referencing system is affine**

`true` or `1` | `false` or `0`

This property is read-only.

Spatial referencing system is affine, specified as a logical `1` (`true`) or `0` (`false`). An image volume is affine if these conditions are met:

- All slices are parallel to each other.
- The spacing between slices in each dimension is uniform.
- The upper-left voxels of all slices are collinear.
- No two slices are coincident, meaning no two slices are located at the same position in space.

Data Types: `logical`

### **IsAxesAligned — Image data array is aligned with patient coordinate system**

`true` or `1` | `false` or `0`

This property is read-only.

Image data array is aligned with the patient coordinate system, specified as a logical `1` (`true`) or `0` (`false`). A `true` value indicates that the directions along which the row, column, and slice indices change are aligned with the *x*-, *y*-, and *z*-axes of the patient coordinate system.

Data Types: `logical`

### **IsMixed — Orientation of image slices is mixed**

`true` or `1` | `false` or `0`

This property is read-only.

Orientation of the image slices is mixed, specified as a logical `1` (`true`) or `0` (`false`). A `false` value indicates that all of the slices in the image volume are parallel and have a similar orientation. A `true` value indicates that one or more slices are not parallel and similarly oriented to other slices.

Data Types: `logical`

## **Object Functions**

<code>intrinsicToWorldMapping</code>	Geometric transform between intrinsic and patient coordinates of medical image volume
<code>contains</code>	Determine if affine image volume contains points specified in patient coordinate system
<code>intrinsicToWorld</code>	Map points from intrinsic coordinates to patient coordinates
<code>oneSliceIntrinsicToWorldMapping</code>	Geometric transform between intrinsic and patient coordinates of medical image volume slice
<code>orient</code>	Update patient coordinate system convention
<code>sliceCorners</code>	Extract patient coordinates of corner voxels for one slice
<code>worldToIntrinsic</code>	Map points from patient coordinates to intrinsic coordinates

worldToSubscript

Convert from patient coordinates to row and column subscripts

## Examples

### Create `medicalref3d` Object Using Volume Size and Geometric Transformation

Specify the size of the target image volume array, in voxels.

```
volumeSize = [256 256 100];
```

Specify the geometric transformation that describes the mapping between the intrinsic and patient coordinates for the target image volume. Assume the patient coordinate system has units of millimeters.

Specify the scale factors that correspond to voxel dimensions of 0.5 mm, 0.5 mm, and 1.0 mm.

```
sx = 0.5;  
sy = 0.5;  
sz = 1;
```

Specify the position of the first voxel of the first transverse slice, in millimeters. The position defines the translation between the intrinsic coordinate origin and the patient coordinate origin.

```
tx = 1000;  
ty = 50;  
tz = 75;
```

Define a 4-by-4 geometric transformation matrix that performs the voxel scaling and translation. The first column corresponds to the y-dimension of the image, and the second column corresponds to the x-dimension.

```
A = [0 sx 0 tx;  
     sy 0 0 ty;  
     0 0 sz tz;  
     0 0 0 1];
```

Create an `affinetform3d` object that performs the transformation.

```
tform = affinetform3d(A);
```

Create a `medicalref3d` object with the specified array size and geometric transformation.

```
R = medicalref3d(volumeSize,tform)
```

```
R =  
  medicalref3d with properties:  
    VolumeSize: [256 256 100]  
    Position: [100x3 double]  
    VoxelDistances: {[0.5000 0 0] [0 0.5000 0] [0 0 1]}  
    PatientCoordinateSystem: "Unknown"  
    PixelSpacing: [0.5000 0.5000]  
    IsAffine: 1  
    IsAxesAligned: 1  
    IsMixed: 0
```

## **Version History**

**Introduced in R2022b**

### **See Also**

#### **Topics**

“Read, Process, and Write 3-D Medical Images”

“Display Medical Image Volume in Patient Coordinate System”

“Display Labeled Medical Image Volume in Patient Coordinate System”

# medicalVolume

3-D medical image voxel data and spatial referencing information

## Description

A `medicalVolume` object stores the voxel data and spatial referencing information for the medical image volume contained in a single DICOM, NIFTI, or NRRD file, or in a directory of DICOM files. The `medicalVolume` object specifies the mapping between the intrinsic image coordinate system, the patient coordinate system, and the anatomical planes. The `medicalVolume` object and its object functions provide a standardized interface for accessing voxel data, spatial referencing, and intensity scaling information.

## Creation

### Syntax

```
medVol = medicalVolume(dirname)
medVol = medicalVolume(filenamees)
medVol = medicalVolume(sourceTable)
medVol = medicalVolume(sourceTable, rowname)
medVol = medicalVolume(imds)
medVol = medicalVolume(voxels, VolumeGeometry)
```

### Description

`medVol = medicalVolume(dirname)` creates a `medicalVolume` object for the image volume contained in a multifile DICOM series in the directory `dirname`.

`medVol = medicalVolume(filenamees)` creates a `medicalVolume` object for the image volume stored in the single DICOM, NIFTI, or NRRD file or list of DICOM files specified by `filenamees`.

`medVol = medicalVolume(sourceTable)` creates a `medicalVolume` object for the image volume listed in `sourceTable`. The table must contain only one row that specifies the metadata for a DICOM volume.

`medVol = medicalVolume(sourceTable, rowname)` creates a `medicalVolume` object for the image volume listed in the row `rowname` of `sourceTable`. Use this syntax to specify one row of a multirow table by name or by numeric index.

`medVol = medicalVolume(imds)` creates a `medicalVolume` object for the image volume specified by the image datastore object `imds`.

`medVol = medicalVolume(voxels, VolumeGeometry)` creates a `medicalVolume` by directly specifying the Voxels and "VolumeGeometry" on page 1-0 properties.

## Input Arguments

### **dirname** — Name of directory containing medical image volume data

string scalar | character vector

Name of the directory containing the medical image volume data, specified as a string scalar or a character vector. Specify `dirname` as the name of a directory containing multiple DICOM files corresponding to one image volume.

### **filenames** — Name of file or files containing medical image volume data

string scalar | character vector | string array

Name of the file or files containing the medical image volume data, specified as a string scalar, character vector, or string array. Specify `filenames` as a single DICOM, NIFTI, or NRRD file or as a list of DICOM files corresponding to one image volume.

### **sourceTable** — Collection of DICOM file metadata

table

Collection of DICOM file metadata, specified as a table returned by the `dicomCollection` function.

### **rowname** — Name or index of table row

string scalar | character vector | positive integer

Name or index of table row, specified as a string scalar, character vector, or positive integer. Specify `rowname` as a string scalar or character vector to specify a row of `sourceTable` by name. Specify `rowname` as a positive integer to specify a row by its numeric index.

### **imds** — Datastore specifying list of DICOM files containing medical image volume data

ImageDatastore object

Datastore specifying a list of DICOM files containing medical image volume data, specified as an `ImageDataStore` object. The list of files must correspond to one medical image volume.

## Properties

### **Voxels** — Image voxel values

*m-by-n-by-p* numeric array | *N-D* numeric array

Image voxel values, specified as an *m-by-n-by-p* numeric array or an *N-D* numeric array. The first three dimensions of `Voxels` correspond to the spatial dimensions of the patient coordinate system.

For DICOM files, `medicalVolume` rescales the intensity values using the `RescaleIntercept` and `RescaleSlope` metadata attributes, if they are present in the file. Additionally, `medicalVolume` converts DICOM intensities of data type `uint16` to data type `int16` or `single`.

- If the `RescaleSlope` attribute value is unspecified or an integer, then `medicalVolume` converts the intensity values from `uint16` to `int16`.
- If the `RescaleSlope` attribute value is fractional, then `medicalVolume` converts the intensity values from `uint16` to `single`.

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

**VolumeGeometry — Spatial referencing information**`medicalref3d` object

Spatial referencing information, specified as a `medicalref3d` object. The coordinates and distances specified by `VolumeGeometry` are in the units specified by the `SpatialUnits` property, if available in the file metadata.

**SpatialUnits — Real-world spatial units**`"unknown"` (default) | string scalar

Real-world spatial units, specified as a string scalar or a character vector. If the data source is a DICOM file, then the `SpatialUnits` value is `"mm"` (millimeters). If the data source is a NIfTI or NRRD file, the object extracts the spatial units from the `SpaceUnits` or `spaceunits` metadata attribute, respectively, if present in the file.

Data Types: `string`**VoxelSpacing — Distances between voxel centers in each dimension**

1-by-3 numeric vector

This property is read-only.

Distances between voxel centers in each dimension, specified as a 1-by-3 numeric vector. The values of `VoxelSpacing` are in the units specified by the `SpatialUnits` property. If `VolumeGeometry.IsAffine` is `false`, then the object calculates the spacing for each dimension as the average spacing across all slices.

Data Types: `double`**Orientation — Slice plane with greatest spatial resolution**`"coronal"` | `"mixed"` | `"oblique"` | `"sagittal"` | `"transverse"` | `"unknown"`

This property is read-only.

Slice plane with the greatest spatial resolution, specified as one of these values:

- `"coronal"` — Coronal plane.
- `"sagittal"` — Sagittal plane.
- `"transverse"` — Transverse plane.
- `"mixed"` — Image volume slices are not parallel. This value occurs when the `IsMixed` property of the `VolumeGeometry` object is `true`.
- `"oblique"` — Image slices are not aligned with the anatomical axes.
- `"unknown"` — Mapping between image coordinate system and anatomical axes is unknown. This value occurs when the `PatientCoordinateSystem` property of the `VolumeGeometry` object is `"unknown"`.

Data Types: `string`**NumCoronalSlices — Number of slices in coronal direction**

numeric scalar | []

This property is read-only.

Number of slices in the coronal direction, specified as a numeric scalar or empty array. `NumCoronalSlices` is empty in these cases:

- PatientCoordinateSystem property of the VolumeGeometry object is "unknown".
- Image volume is oblique.
- Image volume contains a temporal dimension with multiple slices at each spatial location.

Data Types: double

### **NumSagittalSlices — Number of slices in sagittal direction**

numeric scalar | []

This property is read-only.

Number of slices in the sagittal direction, specified as a numeric scalar or empty array. NumCoronalSlices is empty in these cases:

- PatientCoordinateSystem property of the VolumeGeometry object is "unknown".
- Image volume is oblique.
- Image volume contains a temporal dimension with multiple slices at each spatial location.

Data Types: double

### **NumTransverseSlices — Number of slices in transverse direction**

numeric scalar | []

This property is read-only.

Number of slices in the transverse direction, specified as a numeric scalar or empty array. NumCoronalSlices is empty in these cases:

- PatientCoordinateSystem property of the VolumeGeometry object is "unknown".
- Image volume is oblique.
- Image volume contains a temporal dimension with multiple slices at each spatial location.

Data Types: double

### **PlaneMapping — Mapping between image data dimensions and anatomical planes**

1-by-3 string array

This property is read-only.

Mapping between image data dimensions and anatomical planes, specified as a 1-by-3 string array. If the spatial referencing information is available in the source file, PlaneMapping contains the strings "transverse", "coronal", and "sagittal". For example, if the first element of PlaneMapping is "coronal", then moving along the first dimension of Voxels corresponds to moving between slices in the coronal plane.

If the PatientCoordinateSystem property of the VolumeGeometry object is "unknown", or if the volume is oblique, the values of PlaneMapping are "unknown" or "oblique", respectively.

Example: ["transverse", "coronal", "sagittal"]

Data Types: string

### **NormalVector — Unit vector normal to first slice of image volume**

1-by-3 numeric vector

This property is read-only.

Unit vector normal to the first slice of the image volume, specified as a 1-by-3 numeric vector. The first slice of the volume corresponds to the points stored in `Voxels(:, :, 1)`.

Data Types: `double`

### **Modality — Imaging modality used to capture image volume data**

"unknown" (default) | string scalar

This property is read-only.

Imaging modality used to capture the image volume data, specified as a string scalar. The modality is extracted from the file metadata, if present. Common values include, but are not limited to, "CT" for computed tomography, "MR" for magnetic resonance, "NM" for nuclear medicine, and "US" for ultrasound. If the modality is not specified in the file metadata, the default value is "unknown".

Data Types: `string`

### **WindowCenters — Center of display range window of each slice**

[] (default) |  $p$ -by-1 numeric vector

This property is read-only.

Center of the display range window for each slice, specified as a  $p$ -by-1 numeric vector, where  $p$  is the number of slices in the image volume along the third dimension. The object extracts the `WindowCenters` property value from the file metadata, if available. If the display window metadata is not available, then `WindowCenters` is empty.

Data Types: `double`

### **WindowWidths — Width of display window of each slice**

[] (default) |  $p$ -by-1 numeric vector

This property is read-only.

Width of display window of each slice, specified as a  $p$ -by-1 numeric vector, where  $p$  is the number of slices in the image volume along the third dimension. The object extracts the `WindowWidths` property value from the file metadata, if available. If the display window metadata is not available, then `WindowWidths` is empty.

Data Types: `double`

## **Object Functions**

<code>extractSlice</code>	Extract voxels and spatial details for one slice of medical volume
<code>replaceSlice</code>	Replace voxel values for one slice of medical volume
<code>resample</code>	Resample medical image volume in different patient coordinate system
<code>sliceCorners</code>	Extract coordinates of corner voxels for one slice of medical volume
<code>sliceLimits</code>	Extract X-, Y-, Z-limits for one slice of medical volume
<code>volshow</code>	Display medical volume in patient coordinates
<code>montage</code>	Display medical image slices or frames as montage in patient coordinates
<code>write</code>	Write affine medical volume data to NIFTI file

## **Examples**



## Create Medical Volume Object for Multifile DICOM File

Create a medical volume object using a chest CT volume saved as a directory of DICOM files. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile, filepath)
```

Specify the directory of DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath, "MedicalVolumeDICOMData", "LungCT01");
```

Create a medical volume object for the CT volume.

```
medVol = medicalVolume(dataFolder)
```

```
medVol =
```

```
    medicalVolume with properties:
```

```

        Voxels: [512x512x88 int16]
    VolumeGeometry: [1x1 medicalref3d]
      SpatialUnits: "mm"
      Orientation: "transverse"
    VoxelSpacing: [0.7285 0.7285 2.5000]
    NormalVector: [0 0 1]
    NumCoronalSlices: 512
    NumSagittalSlices: 512
    NumTransverseSlices: 88
      PlaneMapping: ["sagittal"    "coronal"    "transverse"]
      Modality: "CT"
    WindowCenters: [88x1 double]
    WindowWidths: [88x1 double]
```

## Create Medical Volume Object from Filename

Create a medical volume object using a CT chest volume from the Medical Segmentation Decathlon data set [1 on page 1-182]. Download the `MedicalVolumNIftIData.zip` file from the MathWorks website, then unzip the file. The file contains two CT chest volumes and corresponding label images, stored in the NIfTI file format. The size of the data file is approximately 76 MB.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeNIftIData.zip");
filepath = fileparts(zipFile);
unzip(zipFile, filepath)
dataFolder = fullfile(filepath, "MedicalVolumeNIftIData");
```

Specify the file name of the first CT volume.

```
filePath = fullfile(dataFolder, "lung_027.nii.gz");
```

Create a medical volume object for the CT volume.

```
medVol = medicalVolume(filePath)
```

```
medVol =
  medicalVolume with properties:

        Voxels: [512x512x264 single]
  VolumeGeometry: [1x1 medicalref3d]
    SpatialUnits: "mm"
    Orientation: "transverse"
    VoxelSpacing: [0.8594 0.8594 1.2453]
    NormalVector: [0 0 -1]
  NumCoronalSlices: 512
  NumSagittalSlices: 512
  NumTransverseSlices: 264
    PlaneMapping: ["sagittal"    "coronal"    "transverse"]
    Modality: "unknown"
  WindowCenters: 0
  WindowWidths: 0
```

[1] Medical Segmentation Decathlon. "Lung." Tasks. Accessed May 10, 2018. <http://medicaldecathlon.com/>.

The Medical Segmentation Decathlon data set is provided under the CC-BY-SA 4.0 license. All warranties and representations are disclaimed. See the license for details.

### Create Medical Volume Object from DICOM Collection

Create a medical volume object using a chest CT volume saved as a directory of DICOM files. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile, filepath)
```

Specify the directory of DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath, "MedicalVolumeDICOMData", "LungCT01");
```

Gather the details about the DICOM files in the `dataFolder` directory into a table by using the `dicomCollection` function. The files belong to one CT volume series, so the table has one row.

```
sourceTable = dicomCollection(dataFolder);
```

Create a medical volume object for the CT volume by specifying the single-row DICOM collection table.

```
medVol = medicalVolume(sourceTable)
```

```
medVol =
  medicalVolume with properties:

        Voxels: [512x512x88 int16]
  VolumeGeometry: [1x1 medicalref3d]
```

```

    SpatialUnits: "mm"
    Orientation: "transverse"
    VoxelSpacing: [0.7285 0.7285 2.5000]
    NormalVector: [0 0 1]
    NumCoronalSlices: 512
    NumSagittalSlices: 512
    NumTransverseSlices: 88
    PlaneMapping: ["sagittal" "coronal" "transverse"]
    Modality: "CT"
    WindowCenters: [88x1 double]
    WindowWidths: [88x1 double]

```

### Create Medical Volume Object from Specified Row of DICOM Collection

Create a medical volume object using a data set containing three chest CT scans. Each CT scan is saved as a directory of DICOM files. The size of the data set is approximately 81 MB. Download the data set from the MathWorks website, then unzip the folder.

```

zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData");

```

Gather the details about the DICOM files in the `dataFolder` directory into a table by using the `dicomCollection` function. The table contains three rows, each corresponding to one of three CT DICOM series.

```
sourceTable = dicomCollection(dataFolder);
```

Create a medical volume object for the second CT volume by specifying the DICOM collection table and a numeric row index.

```
medVol = medicalVolume(sourceTable,2)
```

```
medVol =
```

```
    medicalVolume with properties:
```

```

        Voxels: [512x512x88 int16]
    VolumeGeometry: [1x1 medicalref3d]
    SpatialUnits: "mm"
    Orientation: "transverse"
    VoxelSpacing: [0.7617 0.7617 2.5000]
    NormalVector: [0 0 1]
    NumCoronalSlices: 512
    NumSagittalSlices: 512
    NumTransverseSlices: 88
    PlaneMapping: ["sagittal" "coronal" "transverse"]
    Modality: "CT"
    WindowCenters: [88x1 double]
    WindowWidths: [88x1 double]

```

## Create Medical Volume Object from Image Datastore

Create a medical volume object using a chest CT volume saved as a directory of DICOM files. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile, filepath)
```

Specify the directory of DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath, "MedicalVolumeDICOMData", "LungCT01");
```

Create an image datastore containing the DICOM files in the `dataFolder` directory. Specify a custom read function to read the DICOM files.

```
dicomds = imageDatastore(dataFolder, ...
    FileExtensions=".dcm", ReadFcn=@(x) dicomread(x));
```

Create a medical volume object for the CT volume.

```
medVol = medicalVolume(dicomds)
```

```
medVol =
    medicalVolume with properties:
```

```

        Voxels: [512x512x88 int16]
    VolumeGeometry: [1x1 medicalref3d]
      SpatialUnits: "mm"
      Orientation: "transverse"
    VoxelSpacing: [0.7285 0.7285 2.5000]
    NormalVector: [0 0 1]
    NumCoronalSlices: 512
    NumSagittalSlices: 512
    NumTransverseSlices: 88
      PlaneMapping: ["sagittal" "coronal" "transverse"]
      Modality: "CT"
    WindowCenters: [88x1 double]
    WindowWidths: [88x1 double]
```

## Create Medical Volume Object from Voxels and Spatial Details

Create a medical volume object using a chest CT volume saved as a directory of DICOM files. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile, filepath)
```

Specify the directory of DICOM files for the first CT volume in the data set.

```
dataFolder = fullfile(filepath, "MedicalVolumeDICOMData", "LungCT01");
```

Create a medical volume object for the CT volume.

```
medVol = medicalVolume(dataFolder);
```

The `Voxels` property contains the intensity values of each voxel. The `VolumeGeometry` property contains a `medicalref3d` object defining the spatial referencing for the image volume.

```
V = medVol.Voxels;
R = medVol.VolumeGeometry;
```

Modify the voxel data by applying a 3-D Gaussian filter.

```
sigma = 2;
filterV = imgaussfilt3(V,sigma);
```

Create a new `medicalVolume` object that contains the smoothed voxel values. To maintain the same spatial referencing as the original volume, specify the original `medicalref3d` object `R`.

```
medVolFiltered = medicalVolume(filterV,R);
```

### Display Medical Image Volume in Slice Viewer

Run this code to download a data set from the MathWorks® website and unzip the downloaded folder. The data set contains three CT volumes that are each saved as a directory of DICOM files. The size of the entire data set is approximately 81 MB.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

The folder `dataFolder` contains the downloaded scan used by this example, `LungCT01`.

```
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData","LungCT01");
```

Create a medical image volume object that contains the image data and spatial referencing information for the CT volume. The `Orientation` property indicates that the primary slice direction is "transverse".

```
medVol = medicalVolume(dataFolder)
```

```
medVol =
  medicalVolume with properties:
        Voxels: [512x512x88 int16]
  VolumeGeometry: [1x1 medicalref3d]
    SpatialUnits: "mm"
    Orientation: "transverse"
    VoxelSpacing: [0.7285 0.7285 2.5000]
    NormalVector: [0 0 1]
  NumCoronalSlices: 512
  NumSagittalSlices: 512
  NumTransverseSlices: 88
    PlaneMapping: ["sagittal" "coronal" "transverse"]
    Modality: "CT"
  WindowCenters: [88x1 double]
```

```
WindowWidths: [88x1 double]
```

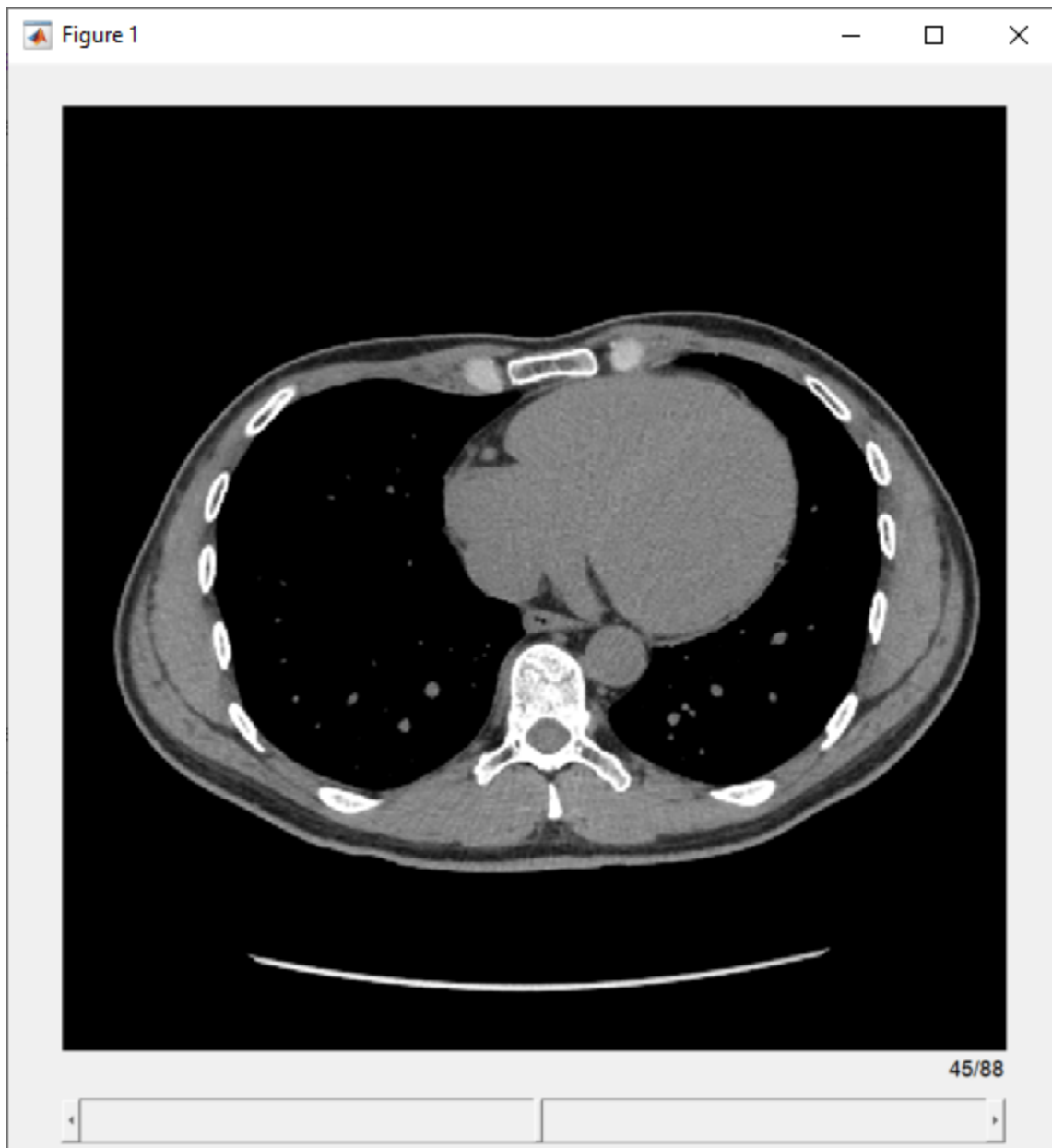
View the transverse slices of the volume in the slice viewer. By default, the viewer uses the properties of `medVol` to scale anisotropic voxels, set the intensity display range, and orient slices. The viewer opens on the center slice. Use the scroll bar to navigate to other slices.

```
sv = sliceViewer(medVol)
```

```
sv =
```

```
sliceViewer with properties:
```

```
    SliceDirection: [0 0 1]
      SliceNumber: 45
        Parent: [1x1 Panel]
      Colormap: [256x3 double]
    DisplayRange: [-160 240]
      ScaleFactors: [1 1 1]
DisplayRangeInteraction: 'on'
```



## Version History

Introduced in R2022b

**R2023a: The `medicalVolume` object converts DICOM intensities from unsigned to signed values**

*Behavior changed in R2023a*

When reading DICOM files, the `medicalVolume` object converts intensity values of data type `uint16` to data type `int16` or `single`. If the DICOM file specifies the `RescaleIntercept` and `RescaleSlope` metadata attributes, `medicalVolume` converts the values before rescaling them. By

converting unsigned `uint16` values to signed values before rescaling, the new behavior preserves the full data range of the rescaled intensities, including negative values.

- If the `RescaleSlope` attribute value is unspecified or an integer, then `medicalVolume` converts the intensities from `uint16` to `int16`.
- If the `RescaleSlope` attribute value is fractional, then `medicalVolume` converts the intensities from `uint16` to `single`.

In R2022b, `medicalVolume` stored intensity values as the same data type as in the DICOM file. For DICOM files that stored intensities as data type `uint16`, `medicalVolume` limited rescaled values to the range  $[0, 2^{16}-1]$ , and set all negative intensities to 0 before storing them in the `Voxels` property.

## See Also

`medicalref3d` | `medicalImage` | `sliceViewer`

### Topics

“Choose Approach for Medical Image Visualization”

“Display Medical Image Volume in Patient Coordinate System”

“Display Labeled Medical Image Volume in Patient Coordinate System”

“Create STL Surface Model of Femur Bone for 3-D Printing”

“Medical Image-Based Finite Element Analysis of Spine”



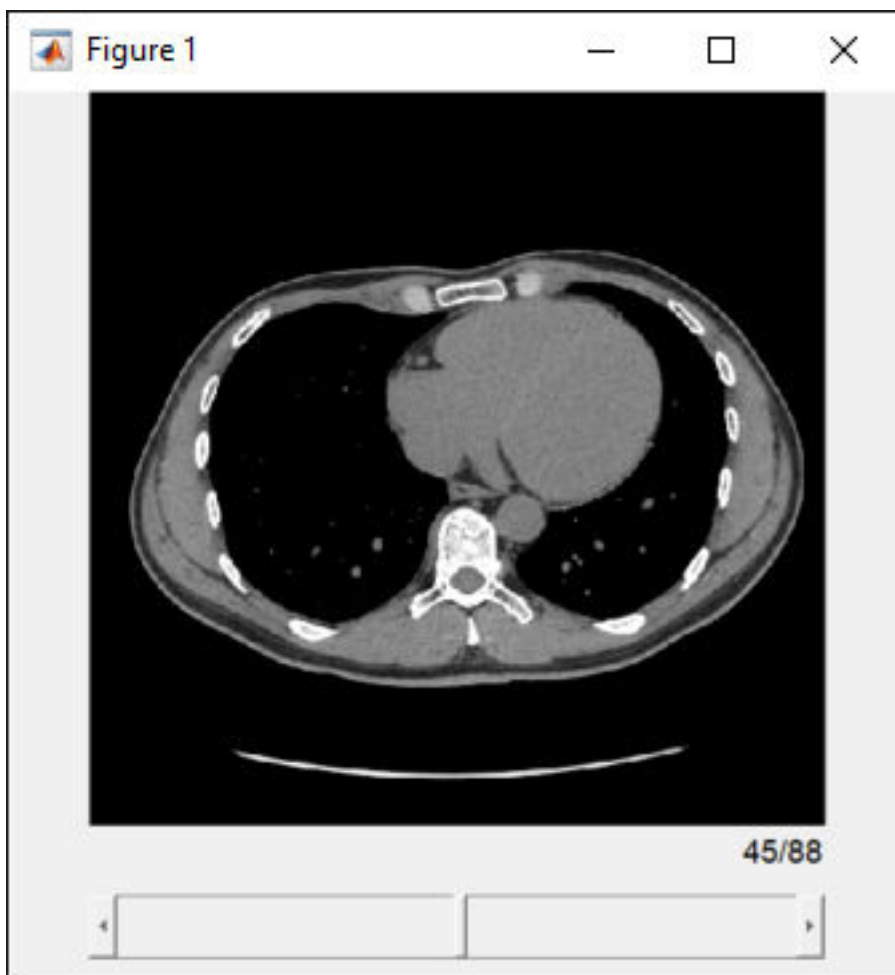
## sliceViewer

Browse slices of medical image volume in patient coordinates

### Description

Medical Imaging Toolbox extends the functionality of the `sliceViewer` (Image Processing Toolbox) object to display the slices of a `medicalVolume` object in the patient coordinate system. The function uses the `medicalVolume` properties to orient slices, set the intensity display range, and scale anisotropic voxels. If you do not have Medical Imaging Toolbox installed, see `sliceViewer` (Image Processing Toolbox).

When it opens a figure, the `sliceViewer` object displays the middle image in the stack. The viewer displays slices along the primary direction of the volume, specified by the `Orientation` property of the `medicalVolume` object. 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 behavior. The `sliceViewer` object can send notifications when certain events occur, such as the slider moving. For more information, see “Events” on page 1-196.

---

**Note** By default, clicking and dragging the mouse in any slice view changes the brightness and contrast, a technique called window/level. Drag the mouse left and right to change the contrast. Drag the mouse up and down to change the brightness. Hold **Ctrl** while you click and drag to accelerate changes. Hold **Shift** while you click and drag to slow the rate of change. To control this behavior, use the `DisplayRangeInteraction` property.

---

## Creation

### Syntax

```
sliceViewer(medVol)  
sliceViewer(medVol, AnatomicalConvention=convention)
```

```
sliceViewer(V)
```

```
sliceViewer( ____, Name=Value)
```

### Description

#### Medical Volume Object

`sliceViewer(medVol)` displays the slices of the `medicalVolume` object `medVol` in a scrollable figure window. The `sliceViewer` object uses the `medicalVolume` properties to orient slices, set the intensity display range, and scale anisotropic voxels. See the `medVol` argument description for details.

`sliceViewer(medVol, AnatomicalConvention=convention)` orients the medical volume according to the display convention `convention`.

#### Numeric Array

`sliceViewer(V)` displays the grayscale or RGB volume `V`, where `V` is a numeric array. Use this syntax to display image file formats not supported by `medicalVolume`.

#### Additional Options

`sliceViewer( ____, Name=Value)` sets Properties using one or more name-value arguments, in addition to any combination of input arguments from previous syntaxes. For example, `sliceViewer(medVol, Colormap=cmap)` specifies the colormap used to display the volume.

`sv = sliceViewer( ____, Name=Value)` returns a handle to the `sliceViewer` object, `sv`. Use `sv` to access and modify properties that control the visualization of the slice images.

### Input Arguments

#### **medVol** — Medical volume

`medicalVolume` object

Medical volume, specified as a `medicalVolume` object. The `sliceViewer` object displays the slices using the properties of `medVol`:

- The `sliceViewer` uses the `WindowCenters` and `WindowWidths` properties, if specified, to set the `DisplayRange` property.

- The `sliceViewer` uses the `VoxelSpacing` property to resize the display axes to accurately display anisotropic voxels.
- The `sliceViewer` displays slices in the plane specified by the `Orientation` property of `medVol`:
  - If the `Orientation` property value is `"transverse"`, the viewer displays transverse slices with the anterior side of the patient on the top of the image and the right side of the patient on the left side of the image. To display the right side of the patient on the right side of the image, specify the convention argument as `"neurological"`.
  - If the `Orientation` property value is `"coronal"`, the viewer displays coronal slices with the superior end of the patient on the top of the image and the right side of the patient on the left side of the image. To display the right side of the patient on the right side of the image, specify the convention argument as `"neurological"`.
  - If the `Orientation` property value is `"sagittal"`, the viewer displays sagittal slices with the superior end of the patient on the top of the image and the anterior side of the patient on the left side of the image.

If the patient coordinate system is undefined, then `sliceViewer` displays the data in the `Voxels` property of `medVol` as a numeric array with default display settings. The patient coordinate system is undefined if the `PatientCoordinateSystem` property of the `medicalref3d` object in the `VolumeGeometry` property of `medVol` is `"unknown"`.

#### **convention — Anatomical display orientation convention**

`"radiological"` (default) | `"neurological"`

Anatomical display orientation convention, specified as one of these values:

- `"radiological"` — Display the left side of the patient on the right side of transverse and coronal slice images.
- `"neurological"` — Display the left side of the patient on the left side of transverse and coronal slice images.

The convention argument has no effect when displaying sagittal slices.

Data Types: `char` | `string`

#### **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 must be of data type `uint8`, `uint16`, `single`, or `double`.

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

## **Properties**

#### **Colormap — Colormap**

`gray(256)` (default) | *m-by-3* numeric array

Colormap, specified as an *m-by-3* numeric array with values in the range `[0 1]`. The `Colormap` property has no effect when `medVol` contains an RGB image volume.

#### **DisplayRange — Intensity display range**

two-element vector

Intensity display range, specified as a two-element vector of the form `[low high]`. Intensity values less than or equal to `low` appear black, or as the first color in the colormap. Intensity values greater than or equal to `high` appear white, or as the last color in the colormap. If you specify an empty matrix (`[]`), then `sliceViewer` uses the minimum and maximum values in the image. The default display range depends on the input image:

- If you specify a `medicalVolume` with the `WindowCenters` and `WindowWidths` properties specified, then `sliceViewer` uses the property values to set the display range. If `WindowCenters` and `WindowWidths` are empty, then `sliceViewer` uses the minimum and maximum values in the image.
- If you specify a numeric array, `V`, as input, `sliceViewer` uses the minimum and maximum values in the image.

`DisplayRange` has no effect when displaying RGB image data.

### DisplayRangeInteraction — 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 1-196.

Value	Description
"on" (default for grayscale intensity volumes)	Control the display range of an 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 volume

`[1 1 1]` (default) | 3-element positive numeric row vector

Scale factors used to rescale the volume, specified as a 3-element positive numeric row vector. The values in the vector specify the scale factor applied in the *x*-, *y*-, and *z*-directions, respectively.

---

**Note** If you specify a `medicalVolume` with anisotropic voxels as input, the slice viewer scales the display axes to accurately display voxel sizes. In this case, `sliceViewer` does not update the `ScaleFactors` value.

---

### SliceDirection — Direction in which to browse image stack

`[0 0 1]` (default) | `[1 0 0]` | `[0 1 0]` | "X" | "Y" | "Z"

Direction in which to browse the image stack, specified as one of the logical vectors or strings in the table. If you specify a `medicalVolume` object as input, `sliceViewer` does not support setting the `SliceDirection` property. The `Orientation` property of `medVol` determines the slice direction.

For numeric array inputs, the values correspond to the dimensions of the input array  $V$ , not the patient coordinate system axes.

String	Logical Vector	Description
"X"	[1 0 0]	Browse in the first direction
"Y"	[0 1 0]	Browse in the second direction
"Z" (default)	[0 0 1]	Browse in the third direction

### SliceNumber — Index of slice to display

positive numeric scalar

Index of the slice to display when `sliceViewer` opens the figure, 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

### Display Medical Image Volume in Slice Viewer

Run this code to download a data set from the MathWorks® website and unzip the downloaded folder. The data set contains three CT volumes that are each saved as a directory of DICOM files. The size of the entire data set is approximately 81 MB.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile, filepath)
```

The folder `dataFolder` contains the downloaded scan used by this example, `LungCT01`.

```
dataFolder = fullfile(filepath, "MedicalVolumeDICOMData", "LungCT01");
```

Create a medical image volume object that contains the image data and spatial referencing information for the CT volume. The `Orientation` property indicates that the primary slice direction is "transverse".

```
medVol = medicalVolume(dataFolder)
```

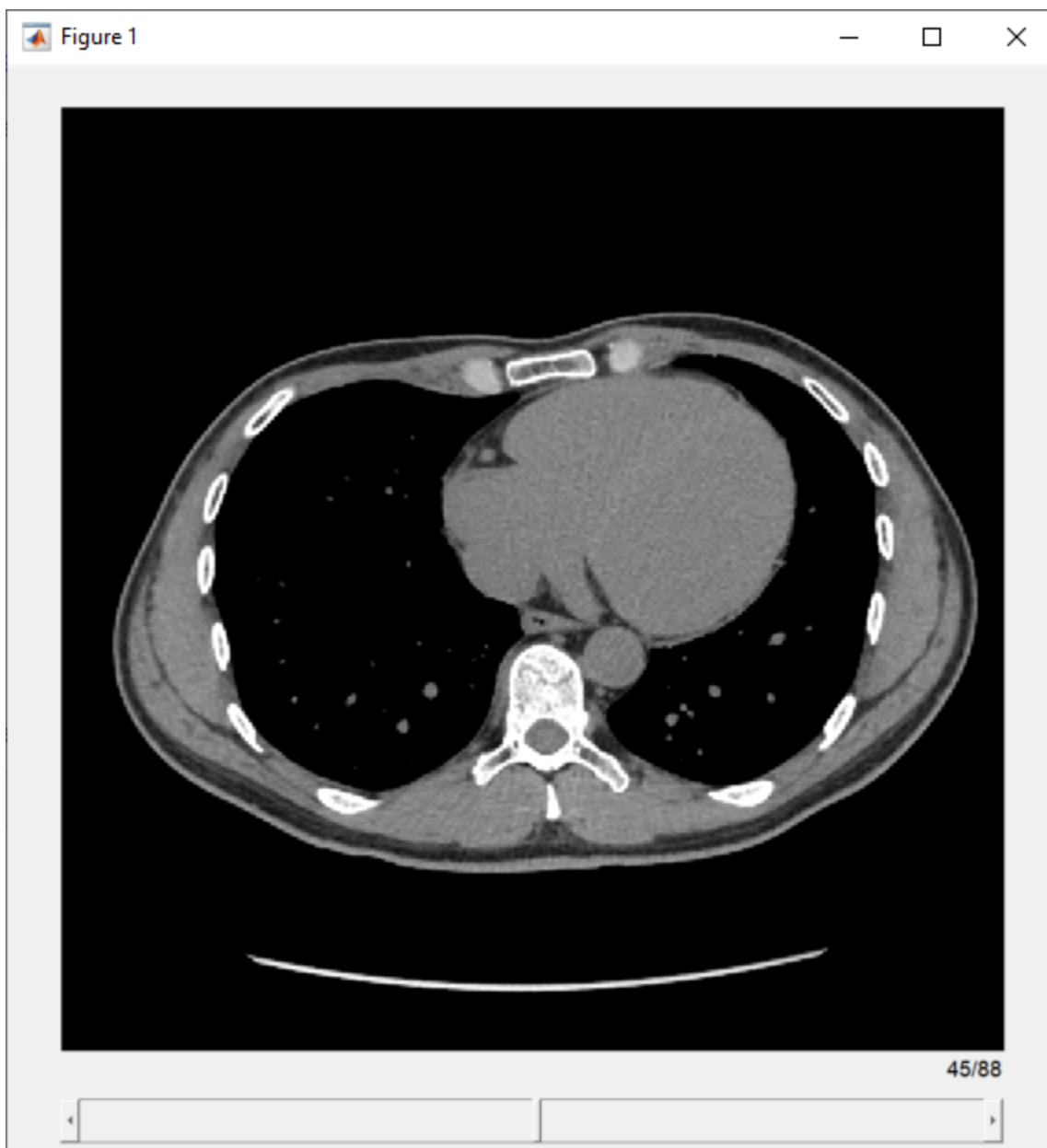
```
medVol =
    medicalVolume with properties:
        Voxels: [512x512x88 int16]
        VolumeGeometry: [1x1 medicalref3d]
        SpatialUnits: "mm"
        Orientation: "transverse"
        VoxelSpacing: [0.7285 0.7285 2.5000]
        NormalVector: [0 0 1]
```

```
    NumCoronalSlices: 512
    NumSagittalSlices: 512
    NumTransverseSlices: 88
      PlaneMapping: ["sagittal" "coronal" "transverse"]
      Modality: "CT"
    WindowCenters: [88x1 double]
    WindowWidths: [88x1 double]
```

View the transverse slices of the volume in the slice viewer. By default, the viewer uses the properties of `medVol` to scale anisotropic voxels, set the intensity display range, and orient slices. The viewer opens on the center slice. Use the scroll bar to navigate to other slices.

```
sv = sliceViewer(medVol)
```

```
sv =
  sliceViewer with properties:
    SliceDirection: [0 0 1]
    SliceNumber: 45
    Parent: [1x1 Panel]
    Colormap: [256x3 double]
    DisplayRange: [-160 240]
    ScaleFactors: [1 1 1]
    DisplayRangeInteraction: 'on'
```



### Display Medical Image Stored as Numeric Array in Slice Viewer

If you are working with 3-D medical image data stored as a DICOM, NiftI, or NRRD file, create a `medicalVolume` object before displaying the volume. Passing a `medicalVolume` object as input allows `sliceViewer` to use the `medicalVolume` properties to automatically adjust display settings. If your medical image volume data is stored in a format that is not supported by `medicalVolume`, you can pass a numeric array as input to the `sliceViewer` object. The viewer displays the rows, columns, and slices of the data without any knowledge of a patient coordinate system.

Load a MAT file that contains an MRI image volume. The file loads the 3-D numeric array `mrstack` into the workspace.

```
load mrstack.mat
```

Display the slices of the MRI volume. By default, `sliceViewer` displays slices along the third direction of the numeric array.

```
sliceViewer(mrstack)
```

```
ans =  
  sliceViewer with properties:  
  
    SliceDirection: [0 0 1]  
    SliceNumber: 11  
    Parent: [1x1 Panel]  
    Colormap: [256x3 double]  
    DisplayRange: [0 255]  
    ScaleFactors: [1 1 1]  
    DisplayRangeInteraction: 'on'
```



## 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 to call when the event occurs. This table lists events supported by the `sliceViewer` object.



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

## Version History

Introduced in R2023a

### See Also

#### Objects

medicalVolume | medicalref3d

#### Functions

volshow | montage

#### Topics

“Choose Approach for Medical Image Visualization”

“Visualize 3-D Medical Image Data Using Medical Image Labeler”

“Display Medical Image Volume in Patient Coordinate System”

# VolumeSource

Source of 3-D medical image data for `groundTruthMedical` object

## Description

A `VolumeSource` object defines the source of ground truth data for 3-D medical image volumes. Use this object to specify the volume data sources for a `groundTruthMedical` object. Each source must be a single DICOM, NIFTI, or NRRD file or multiple DICOM files comprising one 3-D volume.

## Creation

When you export labels from a **Medical Image Labeler** app volume session, the `DataSource` property of the exported `groundTruthMedical` object contains a `VolumeSource` object.

To create a `VolumeSource` object programmatically, such as when programmatically creating a `groundTruthMedical` object, use the `medical.labeler.loading.VolumeSource` function.

## Syntax

```
volSrc = medical.labeler.loading.VolumeSource(source)
volSource = medical.labeler.loading.VolumeSource(sourceTable)
```

### Description

`volSrc = medical.labeler.loading.VolumeSource(source)` creates a `VolumeSource` object for loading the 3-D medical image data stored in the files or directory specified by `source`.

The data source for a `VolumeSource` object must be readable by a `medicalVolume` object and have a primary slice direction of "sagittal", "coronal", "transverse", or "oblique", which indicates that all slices are parallel. The `Orientation` property of the `medicalVolume` object specifies the primary slice direction. `VolumeSource` does not support volumes with a primary orientation that is "mixed" or "unknown".

`VolumeSource` does not support volumes with more than three dimensions.

`volSource = medical.labeler.loading.VolumeSource(sourceTable)` creates a `VolumeSource` object for loading the image data specified in a table, `sourceTable`, returned by the `dicomCollection` function.

### Input Arguments

#### **source** — Source file names

*n*-by-1 cell array

Source file names, specified as an *n*-by-1 cell array, where *n* is the total number of image volumes to store in the `groundTruthMedical` object.

Specify each element of `source` as one of these options:

- Name of a single DICOM, NIfTI, or NRRD file defining one image volume, specified as a string scalar.
- List of file names defining one multi-file DICOM image volume, specified as a string array.
- Name of directory containing files defining one multi-file DICOM image volume, specified as a string scalar.

Data Types: `cell`

### **sourceTable — Table of source file names**

table

Table of source file names, specified as a table returned by the `dicomCollection` function. Each row in `sourceTable` must specify a valid DICOM series that contains a 3-D image volume readable by a `medicalVolume` object. The file name or list of file names for each DICOM series is stored in the `FileNames` column of the table returned by `dicomCollection`.

Data Types: `table`

## **Properties**

### **Source — Source of ground truth data**

*n*-by-1 cell array

This property is read-only.

Source of the ground truth data, specified as an *n*-by-1 cell array, where *n* is the total number of image volumes to store in the `groundTruthMedical` object. Each element contains the file path or file paths corresponding to one image volume, stored as a string scalar for single files or a string array for multi-file DICOM volumes.

Data Types: `cell`

## **Examples**

### **Create Volume Data Source from Medical Image Files**

Create a volume data source using a subset of the Medical Segmentation Decathlon data set [1 on page 1-200]. The subset of data includes two CT chest volumes and corresponding label images stored in the NIfTI file format. Download the `MedicalVolumeNiftIData.zip` file from the MathWorks® website, then unzip the file. The size of the data file is approximately 76 MB.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeNiftIData.zip");
filepath = fileparts(zipFile);
unzip(zipFile, filepath)
dataFolder = fullfile(filepath, "MedicalVolumeNiftIData");
```

Create a `VolumeSource` object specifying the two CT volumes.

```
filePath1 = fullfile(dataFolder, "lung_027.nii.gz");
filePath2 = fullfile(dataFolder, "lung_043.nii.gz");
source = {filePath1; filePath2};
dataSource = medical.labeler.loading.VolumeSource(source);
```

Verify that the filenames are stored in the `Source` property of the data source object.

```
dataSource.Source;
```

[1] Medical Segmentation Decathlon. "Lung." Tasks. Accessed May 10, 2018. <http://medicaldecathlon.com/>.

The Medical Segmentation Decathlon data set is provided under the CC-BY-SA 4.0 license. All warranties and representations are disclaimed. See the license for details.

### **Create Volume Data Source from DICOM Collection**

Create a volume data source using a data set containing three chest CT scans. Each CT scan is saved as a directory of DICOM files. The size of the data set is approximately 81 MB. Download the data set from the MathWorks website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");  
filepath = fileparts(zipFile);  
unzip(zipFile,filepath)  
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData");
```

Gather the details about the DICOM files in the `dataFolder` directory into a table by using the `dicomCollection` function.

```
sourceTable = dicomCollection(dataFolder);
```

Create a volume data source specifying the files in `sourceTable`.

```
dataSource = medical.labeler.loading.VolumeSource(sourceTable);
```

Verify that each element of the `Source` property value contains a string array of filenames for one CT volume. The file names are extracted from the `Filenames` column of `sourceTable`.

```
dataSource.Source;
```

## **Version History**

**Introduced in R2022b**

### **See Also**

`groundTruthMedical` | `ImageSource`