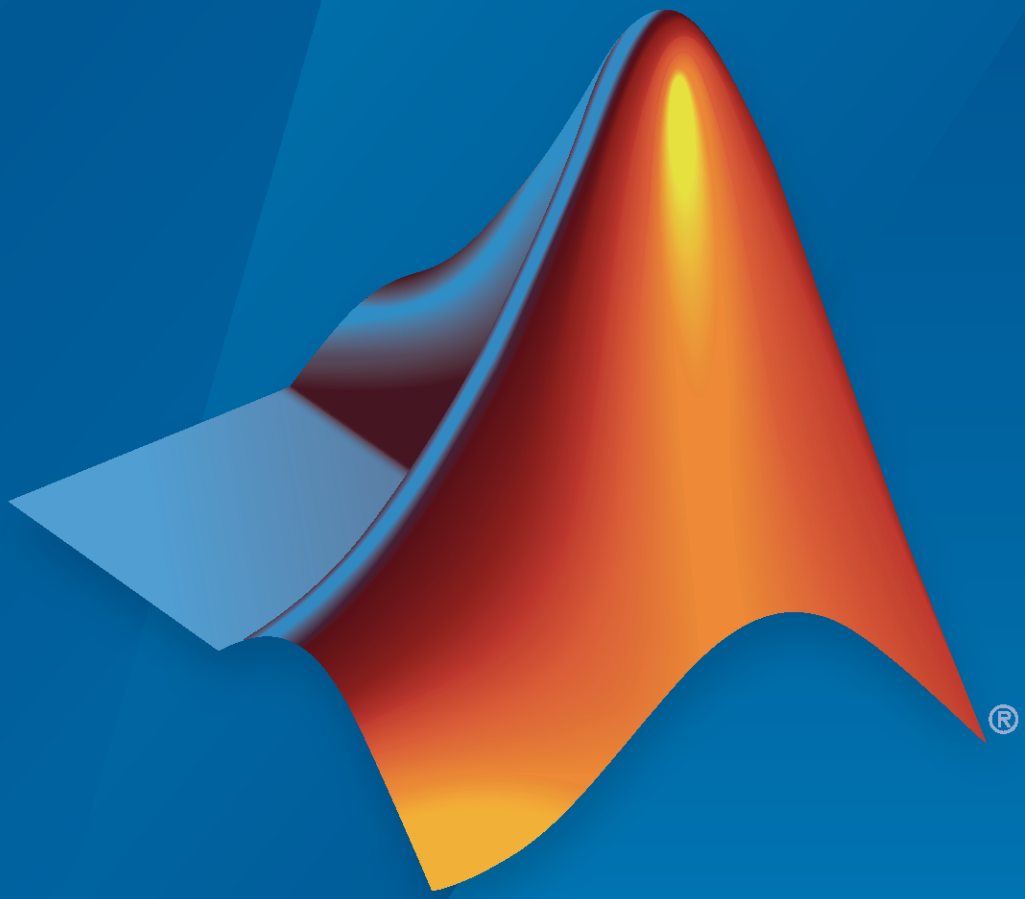
Simulink® 3D Animation™

Reference



MATLAB® & SIMULINK®

R2023a

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`

Phone: 508-647-7000

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

Simulink® 3D Animation™ Reference

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `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` for more information.

Revision History

March 2023	Online only	New for Version 9.6 (Release 2023a)

Contents

Scenes

Empty Scene

Empty environment

Description

The **Empty Scene** contains a 3D environment of an empty world that contains no objects and vehicles. The scene is rendered using the Unreal Engine® from Epic Games®.

Setup

To simulate in this scene:

1. Add a block to your Simulink model.
2. In this block, set the **Scene source** parameter to `Default Scenes`.
3. There is one scene available in the Simulink 3D Animation toolbox, so **Scene name** will be set to `Empty scene`.

Layout

The scene uses the coordinate system of the world to locate the objects.

Version History

Introduced in R2022b

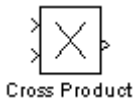
See Also

Simulation 3D Actor | Simulation 3D Scene Configuration

Blocks

Cross Product

Cross product of two 3-D vectors



Libraries:
Simulink 3D Animation / Utilities

Description

Return the cross product–or vector product–of two 3-by-1 vectors. Each input is a vector of the form $a_1\hat{i} + a_2\hat{j} + a_3\hat{k}$ where i, j, and k are unit vectors parallel to the x, y, and z coordinate axes. The output vector $\vec{y} = \vec{a} \times \vec{b}$ is a 3 element vector orthogonal to the input vectors $\vec{a}$ and $\vec{b}$

Ports

Input

Port 1 (a) — 3-element vector
vector

Input vector $\vec{a}$, where the elements represent the magnitude of the vector parallel to the x, y, and z coordinate axes.

Data Types: `double`

Port 2 (b) — 3-element vector
vector

Input vector $\vec{b}$, where the elements represent the magnitude of the vector parallel to the x, y, and z coordinate axes.

Data Types: `double`

Output

Port 1 (y) — Resultant vector
vector

Output vector $\vec{y} = \vec{a} \times \vec{b}$, which is orthogonal to $\vec{a}$ and $\vec{b}$

Data Types: `double`

Version History

Introduced in R2006a

Extended Capabilities

C/C++ Code Generation

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

Actual data type or capability support depends on block implementation.

See Also

Normalize Vector | Rotation Between 2 Vectors | Rotation Matrix to VR Rotation | Viewpoint Direction to VRML Orientation

Topics

“Connect Virtual Worlds and Models”

Joystick Input

Process input from asynchronous joystick device

Libraries:
Simulink 3D Animation

Description

The Joystick Input block provides interaction between a Simulink model and the virtual world associated with a Simulink 3D Animation block.

The Joystick Input block uses axes, buttons, and the point-of-view selector, if present. You can use this block as you would use any other Simulink source block. Its output ports reflect the status of the joystick controls for axes and buttons.

The Joystick Input block also supports force-feedback devices.

Ports

Input

Force — Force feedback input
vector

Provide the force-feedback to be applied along supported joystick axes.

The length of the Force vector corresponds to the number of joystick axes that support force-feedback.

To enable this port, you must first select the **Enable force-feedback input** parameter.

Data Types: `double`

Output

Axes — Joystick position along any given axis
vector with each element in the range [-1,1]

The first joystick axes element is `x`, the second element is `y`, and so on up to the total number of axes. What the `x` axis represents depends on the type and shape of the joystick. The Joystick Input block uses the mapping between the joystick driver and the joystick.

Data Types: `double`

Buttons — Status of joystick buttons
vector of 0 and 1

Status of joystick buttons.

Data Types: `double`

Point of view — Current status of the joystick point-of-view selector
-1 (selector inactive) | scalar

The output signal is the angle of the point-of-view selector, or POV Hat, in degrees from 0 to 360. If the selector is inactive, the signal is -1.

Data Types: `double`

Parameters

Joystick ID — The ID assigned to given joystick device
`1` (default)

You can find the properties of the joystick that is connected to the system in the Game Controllers section of the system Control Panel.

Adjust I/O ports according to joystick capabilities — Dynamically adjust ports to correspond to joystick capabilities
`on` (default) | `off`

If you enable this parameter, the Simulink 3D Animation software dynamically adjusts the ports to correspond to the capabilities of the connected joystick each time that you open the model. If the connected device does not have force-feedback capability, selecting this check box causes the removal of the force-feedback input from the block, even if you enable the **Enable force-feedback input** parameter.

The block ports do not have the full widths provided by the Windows® Game Controllers interface.

Enable force-feedback input — Support joysticks with force-feedback
`on` (default) | `off`

If you select this check box, the Simulink 3D Animation software can support force-feedback joystick, steering wheel, and haptic (one that enables tactile feedback) devices.

Output Ports — Enable output ports for joystick commands
`off` (default) | `on`

When the **Adjust I/O ports according to joystick capabilities** parameter is enabled, the output ports change to correspond to the actual capabilities of the connected joystick. On Windows platforms, the output ports have fixed maximum width provided by the system Game Controllers interface.

Version History

Introduced before R2006a

See Also

Space Mouse Input | `vrjoystick` | `vrspacemouse`

Topics
“Connect Virtual Worlds and Models”

MATLAB to VR Coordinates

Convert MATLAB coordinates to VR coordinates

Libraries:
Simulink 3D Animation / Utilities

Description

The MATLAB to VR Coordinates block converts a point with coordinates in the MATLAB® coordinate system to the VRML coordinate system.

The following relation holds between the two coordinate systems:

$[x_m, y_m, z_m] = [x_v, z_v, -y_v]$

where MATLAB coordinates are denoted with the *m* subscript and Virtual World coordinates are denoted with the *v* subscript. For more information on the two coordinate systems, see “Virtual World Coordinate System”.

Ports

Input

M — Coordinates in MATLAB notation
3-element vector

Coordinates of a point in MATLAB notation, specified as a 3-element row vector.

Data Types: `single` | `double`

Output

VR — Coordinates in VRML notation
3-element vector

Coordinates of a point in VRML notation, returned as a 3-element row vector.

Data Types: `single` | `double`

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

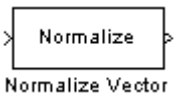
`vrcoordm2vr` | `vrcoordvr2m` | VR to MATLAB Coordinates | VR Rotation to Rotation Matrix | Rotation Matrix to VR Rotation | `vrrotmat2vec` | `vrrotvec2mat`

Topics

“Virtual World Coordinate System”

Normalize Vector

Output unit vector parallel to input vector



Libraries:
Simulink 3D Animation / Utilities

Description

Use the Normalize Vector block to obtain a unit vector parallel to a given vector.

Ports

Input

Input 1 — Input signal
vector

Vector of arbitrary size.

Data Types: `single` | `double`

Output

Output 1 — Unit vector
vector

Unit vector parallel to the vector provided by the input signal.

Data Types: `single` | `double`

Parameters

Maximum modulus to treat vector as zero — Input signal threshold

`0` (default)

The output is set to zeroes if the modulus of the input is equal to or lower than this value.

Version History

Introduced in R2006a

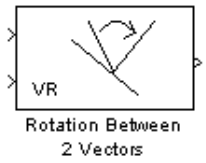
See Also

Cross Product | Rotation Between 2 Vectors | Rotation Matrix to VR Rotation | Viewpoint Direction to VRML Orientation

Topics
“Connect Virtual Worlds and Models”

Rotation Between 2 Vectors

Virtual world rotation between two 3-D vectors



Libraries:
Simulink 3D Animation / Utilities

Description

The Rotation Between 2 Vectors takes the input of two 3-by-1 vectors and returns a virtual world rotation (specified as a 4-element vector defining the axis and angle) that is needed to transform the first input vector to the second input vector.

Ports

Input

Port 1 — Input signal
3-element vector

The input signal is a 3-element vector whose elements correspond to its magnitudes along the $\hat{i}$, $\hat{j}$, $\hat{k}$ unit vectors, respectively.

Data Types: `double`

Port 2 — Input signal
3-element vector

The input signal is a 3- element vector whose elements correspond to its magnitudes along the $\hat{i}$, $\hat{j}$, $\hat{k}$ unit vectors, respectively.

Data Types: `double`

Output

Output 1 — Axis-Angle rotation
4-element vector

The output of the block is an axis-angle representation of the rotation needed to transform the first input vector to the second input vector.

Data Types: `double`

Version History

Introduced in R2006a

Extended Capabilities

C/C++ Code Generation

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

Actual data type or capability support depends on block implementation.

See Also

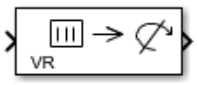
Cross Product | Normalize Vector | Rotation Matrix to VR Rotation | Viewpoint Direction to VRML Orientation

Topics

“Connect Virtual Worlds and Models”

Rotation Matrix to VR Rotation

Convert rotation matrix to axis/angle rotation



Libraries:
Simulink 3D Animation / Utilities

Description

The Rotation Matrix to VR Rotation converts Rotation Matrix (defined columnwise as 3-by-3 matrix or as a 9-element column vector) into the Axis / Angle rotation representation used for defining rotations in VR.

Ports

Input

input 1 — Rotation matrix
3-by-3 matrix

3D rotation, specified as a 3-by-3 columnwise-defined matrix, also known as a direction cosine matrix.

A representation of a three-dimensional spherical rotation as a 3-by-3 real, orthogonal matrix R: $R^{\mathrm{T}}R = RR^{\mathrm{T}} = I$, where I is the 3-by-3 identity and R^{T} is the transpose of R. This matrix is also known as the direction cosine matrix (DCM). The DCM is the orientation of the object in space, relative to its parent node.

$$R = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix} = \begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix}$$

Data Types: `single` | `double`

Output

Port 1 — Axis/Angle Rotation
4 element vector

Output rotation, returned as a 4-element vector in axis/angle notation, The first three elements specify the axis of rotation and the fourth element specifies the angle.

Parameters

Maximum value to treat input value as zero — Effective zero value

`1e-12` (default) | scalar

Input signal value is considered to be zero if it is equal to or lower than the value set in this parameter. By default, the parameter is set to ε = 1e-12.

Version History

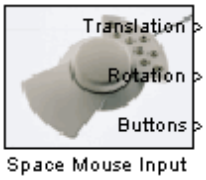
Introduced in R2019a

See Also

VR Rotation to Rotation Matrix

Space Mouse Input

Process input from space mouse device



Libraries:
Simulink 3D Animation

Description

A space mouse is a device similar to a joystick in purpose, but it also provides movement control with six degrees of freedom. This block reads the status of the space mouse and provides some commonly used transformations of the input. The Space Mouse Input block supports current models of 3–D navigation devices manufactured by 3Dconnexion (`https://www.3dconnexion.com`). Contact MathWorks® Technical Support (`https://www.mathworks.com/support`) for further information on the support of older 3Dconnexion devices.

To open the Block Parameters dialog box, double-click the block.

Ports

Output

Translation — Status of object translation
`1` or `true` | `0` or `false`

Status of object translation, returned as `0` or `false` if not pressed and `1` or `true` if pressed.

Data Types: `Boolean`

Rotation — Status of object rotation
`1` or `true` | `0` or `false`

Status of object rotation, returned as `0` or `false` if not pressed and `1` or `true` if pressed.

Data Types: `Boolean`

Buttons — Status of button
`1` or `true` | `0` or `false`

Status of button, returned as `0` or `false` if not pressed and `1` or `true` if pressed.

Data Types: `Boolean`

Parameters

Port — Serial port where mouse is connected
`COM1` (default) | `COM2` | `COM3` | `COM4` | `USB1` | `USB2` | `USB3` | `USB4` | `USB`

Serial port to which the space mouse is connected. Possible values are `USB1`...`USB4` and `COM1`...`COM4`.

Output type — Type of output
`Speed` (default) | `Position` | `Viewpoint coordinates`

This field specifies how the inputs from the device are transformed:

- `Speed` — No transformations are done. Outputs are translation and rotation speeds.
- `Position` — Translations and rotations are integrated. Outputs are position and orientation in the form of roll/pitch/yaw angles.
- `Viewpoint coordinates` — Translations and rotations are integrated. Outputs are position and orientation in the form of an axis and an angle. You can use these values as viewpoint coordinates in a virtual world.

Dominant mode — Option to accept only prevailing movement and rotation
`off` (default) | `on`

If this check box is selected, the mouse accepts only the prevailing movement and rotation and ignores the others. This mode is very useful for beginners using space mouse input.

Disable position movement — Option to fix rotations at initial values
`off` (default) | `on`

Fixes the rotations at initial values, allowing you to change positions only.

Disable rotation movement — Option to fix positions at initial values
`off` (default) | `on`

Fixes the positions at the initial values, allowing you to change rotations only.

Normalize output angle — Option to determine whether rotation angles should wrap in a full circle
`off` (default) | `on`

Determines whether the integrated rotation angles should wrap on a full circle (360°) or not. This is not used when you set the **Output Type** to `Speed`.

Limit position — Option to limit position of mouse
`off` (default) | `on`

Determines whether you can limit the upper and lower positions of the mouse.

Position sensitivity — Mouse sensitivity for translations
`0.0001` (default)

Mouse sensitivity for translations. Higher values correspond to higher sensitivity.

Rotation sensitivity — Mouse sensitivity for rotations
`0.00001` (default)

Mouse sensitivity for rotations. Higher values correspond to higher sensitivity.

Initial position — Initial condition for translations
`[0 0 0]` (default)

Initial condition for integrated translations. This is not used when you set the **Output Type** to `Speed`.

Initial rotation — Initial condition for rotations
`[0 0 0]` (default)

Initial condition for integrated rotations. This is not used when you set the **Output Type** to `Speed`.

Lower position limit — Lower limit of mouse
`[-100 -100 -100]` (default)

Position coordinates for the lower limit of the mouse.

Upper position limit — Upper limit of mouse
`[100 100 100]` (default)

Position coordinates for the upper limit of the mouse.

Version History

Introduced in R2007b

See Also

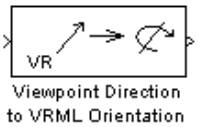
`vrspacemouse` | `vrjoystick`

Topics

“Connect Virtual Worlds and Models”
“Manipulator with SpaceMouse”

Viewpoint Direction to VR Orientation

Convert viewpoint direction to virtual world orientation



Libraries:
Simulink 3D Animation / Utilities

Description

The Viewpoint Direction to VR Orientation takes a viewpoint direction (a 3 element vector) as input and outputs the corresponding virtual world viewpoint orientation (a 4-element rotation vector).

To open the Block Parameters dialog box, double-click the block.

Ports

Input

Input 1 — Viewpoint direction
3-element vector

Viewpoint direction, specified as a 3-element vector.

Data Types: `single` | `double`

Output

Port 1 — Axis/Angle Rotation
4 element vector

Output rotation, returned as a 4-element vector in axis/angle notation, The first three elements specify the axis of rotation and the fourth element specifies the angle.

Version History

Introduced in R2006a

Extended Capabilities

C/C++ Code Generation

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

Actual data type or capability support depends on block implementation.

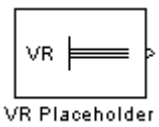
See Also

Cross Product | Normalize Vector | Rotation Between 2 Vectors | Rotation Matrix to VR Rotation

Topics
“Manipulator with SpaceMouse”
“Connect Virtual Worlds and Models”

VR Placeholder

Send unspecified value to Simulink 3D Animation block



Libraries:
Simulink 3D Animation

Description

The VR Placeholder block sends out a special value that is interpreted as “unspecified” by the VR Sink block. When this value appears on the VR Sink input, whether as a single value or as an element of a vector, the appropriate value in the virtual world stays unchanged. Use this block to change only one value from a larger vector. For example, use this block to change just one coordinate from a 3-D position.

The value output by the VR Placeholder block should not be modified before being used in other VR blocks.

To open the Block Parameters dialog box, double-click the block.

Ports

Output

Output 1 — Output signal
real scalar | real vector

Output signal returned as unspecified, that drives the virtual reality visualization.

Data Types: `double`

Parameters

Output width — Length of vector
1 (default)

Length of the vector containing placeholder signal values.

Version History

Introduced before R2006a

See Also

VR Signal Expander

VR RigidBodyTree

Visualize Robotics System Toolbox RigidBodyTree objects in Simulink

Libraries:
Simulink 3D Animation

Description

Use the VR RigidBodyTree block to visualize RigidBodyTree objects from Robotics System Toolbox™ in the Simulink 3D Animation viewer.

Ports

Input

Input 1 — Joint Positions
scalar | vector

Robot configuration that solves the desired end-effector pose, specified as a vector. A robot configuration is a vector of joint positions for the `rigidBodyTree` model. The number of positions is equal to the number of non-fixed joints in the `rigidBodyTree` parameter.

Data Types: `single` | `double`

Parameters

Associated VRML File — 3D World
3D world file name

Specify the virtual world in which the `rigidBodyTree` is visualized

Parent node (leave empty for root) — Scene hierarchy location

character vector | string

Specify the location of the `rigidBodyTree` object in the scene hierarchy. For more information on scene hierarchy, see “Create a Virtual World”.

Rigid Body Tree — robot pose

`rigidBodyTree`

Specify the name of the Robotics System Toolbox `rigidBodyTree` object to be used in the virtual world. If a robot with an identical name is already present in the virtual world, it is used for visualization by default.

You can enable the **Always use robot definition from the RigidBodyTree object** parameter to overwrite the existing robot, if present, with the robot specified by the `rigidBodyTree` object.

Always use robot definition from the RigidBodyTree object — create robot

`'off'` (default) | `'on'`

Enable this parameter to always create a robot from the `rigidBodyTree` object specified by the **Rigid Body Tree** parameter.

By default, the virtual world uses an existing robot by the same name, if it exists.

Sample time — Block sample time for simulation
0.1 (default) | scalar | vector

Specify the sample time for the block, or specify `-1` to inherit the sample time.

Ensure that a viewer window is open during simulation — Open 3D world viewer

`'off'` (default) | `'on'`

Enable this parameter to ensure that the Simulink 3D Animation Viewer is open during simulation.

Version History

Introduced in R2018b

Extended Capabilities

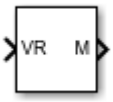
C/C++ Code Generation

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

Actual data type or capability support depends on block implementation.

VR to MATLAB Coordinates

Convert VR coordinates to MATLAB coordinates



Libraries:
Simulink 3D Animation / Utilities

Description

The VR to MATLAB Coordinates block converts a point with coordinates in the Virtual World coordinate system (Znear) to the MATLAB coordinate system (Zup).

The following relation holds between the two coordinate systems:

$[x_m, y_m, z_m] = [x_v, -z_v, y_v]$

where MATLAB coordinates are denoted with the *m* subscript and Virtual World coordinates are denoted with the *v* subscript. For more information on the two coordinate systems, see “Virtual World Coordinate System”.

Ports

Input

VR — Coordinates in the Virtual World coordinate system
3-element vector

Coordinates of a point in VRML notation, specified as a 3-element row vector.

Data Types: `single` | `double`

Output

M — Coordinates in the MATLAB coordinate system
3-element vector

Coordinates of a point in MATLAB notation, returned as a 3-element row vector.

Data Types: `single` | `double`

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

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

See Also

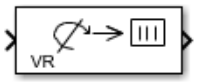
`vrcoordm2vr` | `vrcoordvr2m` | VR Rotation to Rotation Matrix | Rotation Matrix to VR Rotation | `vrrotmat2vec` | `vrrotvec2mat`

Topics

“Virtual World Coordinate System”

VR Rotation to Rotation Matrix

Convert array/angle rotation to rotation matrix



Libraries:
Simulink 3D Animation / Utilities

Description

The VR Rotation to Rotation Matrix block converts the array / angle rotation representation used for defining rotations in virtual reality to a 3-by-3 rotation matrix

Ports

Input

Input 1 — array/Angle rotation
4-element vector

Input rotation, specified as a 4-element vector in array/angle notation,. The first three elements specify the array of rotation and the fourth element specifies the angle.

Data Types: `single` | `double`

Output

Output 1 — Rotation matrix
3-by-3 matrix

3D rotation, returned as a 3-by-3 columnwise defined matrix, also known as a direction cosine matrix.

A representation of a three-dimensional spherical rotation as a 3-by-3 real, orthogonal matrix R: $R^{\mathrm{T}}R = RR^{\mathrm{T}} = I$, where I is the 3-by-3 identity and R^{T} is the transpose of R. This matrix is also known as the direction cosine matrix (DCM). The DCM is the orientation of the object in space, relative to its parent node.

$$R = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix} = \begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix}$$

Data Types: `single` | `double`

Parameters

Maximum value to treat input value as zero — Effective zero value

`1e-12` (default) | scalar

Input signal value is considered to be zero if it is equal to or lower than the value set in this parameter. By default, the parameter is set to ε = 1e-12.

Version History

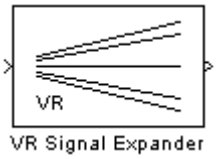
Introduced in R2019a

See Also

Rotation Matrix to VR Rotation

VR Signal Expander

Expand input vectors into fully qualified virtual world field vectors



Library

Simulink 3D Animation

Description

The VR Signal Expander block creates a vector of predefined length, using some values from the input ports and filling the rest with placeholder signal values.

To open the Block Parameters dialog box, double-click the block.

Data Type Support

A VR Signal Expander block accepts and outputs signals of type `double`.

Parameters

Output width — How long the output vector should be.

Output signal indices — Vector indicating the position at which the input signals appear at the output. The remaining positions are filled with VR Placeholder signals.

For example, suppose you want an input vector with two signals and an output vector with four signals, with the first input signal in position 2 and the second input signal in position 4. In the **Output width** box, enter 4 and in the **Output signal indices** box, enter `[2,4]`. The first and third output signals are unspecified.

See Also

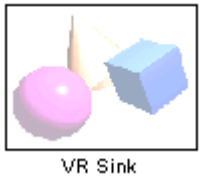
- VR Placeholder

Version History

Introduced before R2006a

VR Sink

Write data from Simulink model to virtual world



Libraries:
Simulink 3D Animation

Description

To output data from the model to control and animate a virtual world, use a VR Sink block. The VR Sink block writes values from its ports to virtual world fields specified in the Block Parameters dialog box.

The VR Sink block is equivalent to the VR To Video block, except that the **Show video output port** parameter for the VR Sink block is cleared by default.

The VR Sink block cannot be compiled by the Simulink Coder™ software, but it can be used as a SimViewing device on the host computer.

Note The current internal viewer window (`vrfigure`) properties are saved together with the Simulink model. Next time you open the model, the internal viewer window opens with the same parameters that were last saved, such as position, size, and navigation mode. When you close the viewer window, the Simulink software does not alert you if these properties have changed.

The VR Sink block is a Sim Viewing Device. You can include it in models that you compile with Simulink Coder software. If you use External mode to compile, build, and deploy the model on a target platform, such as Simulink Real-Time™ or Simulink Desktop Real-Time™, some sink blocks and Sim Viewing Device blocks stay in normal mode during simulation, receive data from the target, and display that data. For more information, see “Use C/C++ S-Functions as Sim Viewing Devices in External Mode” (Simulink).

Ports

Input

Input 1 — Input signal
real scalar | real vector

Input signal to drive the virtual reality visualization of nodes selected in the **Virtual World Tree**.

Data Types: `double`

Output

Output 1 — Output video stream
3-element vector of video signal dimensions

Use the output port to access the RGB video stream of the VR signal input.

Data Types: `double`

Parameters

Source file — File name specifying virtual world that connects to block
string scalar | character vector

By default, the full path to the associated virtual world 3D file appears in this text box. If you enter only the file name in this box, the software assumes that the virtual world 3D file resides in the same folder as the model file. You can specify a VRML file or an X3D file.

- Click **New** to open an empty default virtual world editor. When you either enter a source file name or use the **Browse** button, the **New** button becomes an **Edit** button.
- Click **Edit** to launch the default virtual world editor with the source file open.
- Click **View** to view the world in the Simulink 3D Animation Viewer or a Web browser.
- Click **Reload** to reload the world after you change it.

Open Viewer automatically — Display virtual world on model load
off | on

Enable this parameter to display the virtual world after loading the Simulink model.

Allow viewing from the Internet — View virtual world over network
off (default) | on

Enable this parameter to make the virtual world accessible for viewing on a client computer. If you do not select this check box, then the world is visible only on the host computer. This parameter is equivalent to the `RemoteView` property of a `vrworld` object.

Description — Virtual reality object description
string scalar | character vector

The description is displayed in all virtual reality object listings, in the title bar of the Simulink 3D Animation Viewer, and in the list of virtual worlds on the Simulink 3D Animation HTML page. This parameter is equivalent to the `Description` property of a `vrworld` object.

Sample time — Block sample time for simulation
0.1 (default) | scalar | vector

Specify the sample time for the block, or specify `-1` to inherit the sample time.

Show video output port — Output VR signal to video
off (default) | on

Enable a port to output an RGB video stream for further 2D video processing.

Video output signal dimensions — Specify video output size
`[200 320]` (default) | 2-element vector

Specify the dimensions ([height width]) of the video output signal in pixels.

Virtual World Tree — View structure of virtual world
node

This box shows the structure of the virtual world 3D file and the virtual world itself.

Nodes that have names are marked with red arrows. You can access them from the Simulink 3D Animation interface. Nodes without names but whose children are named are also marked with red arrows. This marking scheme makes it possible for you to find all accessible nodes by traversing the tree using arrows. Other nodes have a blue dot before their names.

Fields with values that you set have check boxes. Use these check boxes to select the fields whose values you want the Simulink software to update. For every field that you select, an input port is created in the block. Input ports are assigned to the selected nodes and fields in the order that corresponds to the virtual world 3D file.

Fields whose values cannot be written (because their parent nodes do not have names, or because they are not of virtual world data class `eventIn` or `exposedField`) have an X-shaped icon.

Show node types — Display node types in virtual world tree
off (default) | on

Enable this parameter to show node types in the virtual world tree.

Show field types — Display field types in virtual world tree
off (default) | on

Enable this parameter to show field types in the virtual scene tree.

Version History

Introduced before R2006a

See Also

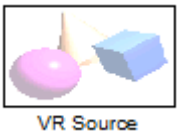
VR Source | VR To Video

Topics

“Connect Virtual Worlds and Models”
“Detect Object Collisions”
“Foucault Pendulum Model with VRML Visualization”
“Use C/C++ S-Functions as Sim Viewing Devices in External Mode” (Simulink)

VR Source

Read data from virtual world to Simulink model



Libraries:
Simulink 3D Animation

Description

Use the VR Source block to provide interactivity between a user navigating the virtual world and the simulation of a Simulink model. The VR Source block registers user interactions with the virtual world and passes that data to the model to affect the simulation of the model. The VR Source reads values from virtual world fields specified in the Block Parameters dialog box and inputs their values to a model.

Examples of some ways that you can use a VR Source block to input data from a virtual world to a Simulink model include:

- Use sensor data from a virtual world to control a simulation. For details, see “Add Sensors to Virtual Worlds” and “Detect Object Collisions”.
- Provide interactivity between user navigation and interaction in a virtual world and the simulation of the model.
- Have a simulation react to virtual world events, such as time ticks or outputs from scripts.
- Use static information from the virtual world, such as the size of a box, to control a simulation.

For example, you can specify setpoints in the virtual world, so that user can specify the location of a virtual world object interactively. The simulation then responds to the changed location of the object. The VR Source block can read into the model events from the virtual world, such as time ticks or outputs from scripts. The VR Source block can also read into the model static information about the virtual world (for example, the size of a box defined in the virtual world 3D file). For examples of models that use the VR Source block, see “Virtual Control Panel” and the `Set the Setpoint` subsystem in the “Portal Crane with Control Panel” example.

Note The current internal viewer window (`vrfigure`) properties are saved together with the Simulink model. The next time that you open the model, the internal viewer window opens with the same parameters that were saved, such as position, size, and navigation mode. When closing the viewer window, the Simulink software does not alert you if these properties have changed.

To open the Block Parameters dialog box VR Source block:

- When you first add a VR Source block and it is still not associated with a virtual world, double-click the block.
- Otherwise, in the Simulink 3D Animation Viewer, select **SimulationBlock parameters**. If the viewer is not already open, you can open it by double-clicking the VR Source block.

You cannot use the Simulink Coder software to compile a model that includes a VR Source block.

Ports

Output

Output 1 — Output signal
real scalar | real vector

Output signal that drives the virtual reality visualization of nodes selected in the **Virtual World Tree**.

Data Types: `double`

Parameters

Source file — File name specifying virtual world that connects to block
string scalar | character vector

By default, the full path to the associated virtual world 3D file appears in this text box. If you enter only the file name in this box, the software assumes that the virtual world 3D file resides in the same folder as the model file. You can specify a VRML file or an X3D file.

- Click **New** to open an empty default virtual world editor. When you either enter a source file name or use the **Browse** button, the **New** button becomes an **Edit** button.
- Click **Edit** to launch the default virtual world editor with the source file open.
- Click **View** to view the world in the Simulink 3D Animation Viewer or a Web browser.
- Click **Reload** to reload the world after you change it.

Open Viewer automatically — Display virtual world on model load
off | on

Enable this parameter to display the virtual world after loading the Simulink model.

Allow viewing from the Internet — View virtual world over network
off (default) | on

Enable this parameter to make the virtual world accessible for viewing on a client computer. If you do not select this check box, then the world is visible only on the host computer. This parameter is equivalent to the `RemoteView` property of a `vrworld` object.

Description — Virtual reality object description
string scalar | character vector

The description is displayed in all virtual reality object listings, in the title bar of the Simulink 3D Animation Viewer, and in the list of virtual worlds on the Simulink 3D Animation HTML page. This parameter is equivalent to the `Description` property of a `vrworld` object.

Sample time — Block sample time for simulation
0.1 (default) | scalar | vector

Specify the sample time for the block, or specify `-1` to inherit the sample time.

Virtual World Tree — View structure of virtual world
node

This box shows the structure of the virtual world 3D file and the virtual world itself.

Nodes that have names are marked with red arrows. You can access them from the Simulink 3D Animation interface. Nodes without names but whose children are named are also marked with red arrows. This marking scheme makes it possible for you to find all accessible nodes by traversing the tree using arrows. Other nodes have a blue dot before their names.

Fields with values that you set have check boxes. Use these check boxes to select the fields whose values you want the Simulink software to update. For every field that you select, an input port is created in the block. Input ports are assigned to the selected nodes and fields in the order that corresponds to the virtual world 3D file.

Fields whose values cannot be written (because their parent nodes do not have names, or because they are not of virtual world data class `eventIn` or `exposedField`) have an X-shaped icon.

Show node types — Display node types in virtual world tree
off (default) | on

Enable this parameter to show node types in the virtual world tree.

Show field types — Display field types in virtual world tree
off (default) | on

Enable this parameter to show field types in the virtual scene tree.

Version History

Introduced in R2011b

See Also

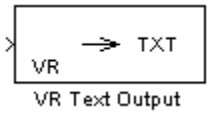
VR Sink | VR To Video

Topics

“Connect Virtual Worlds and Models”
“Detect Object Collisions”
“Foucault Pendulum Model with VRML Visualization”
“Use C/C++ S-Functions as Sim Viewing Devices in External Mode” (Simulink)

VR Text Output

Allows display of Simulink signal values as text in virtual reality scene



Library

Simulink 3D Animation

Description

The VR Text Output can display Simulink values of signal as text in a virtual reality scene.

Text rendering is a demanding task for virtual world viewers, so there is generally be a decrease in rendering speed when outputting text. This effect increases with the complexity of the text output. You can improve the performance if you limit the output from the Simulink model to only the values of signals that change (e.g., modeling captions) or use more static-text nodes.

To open the Block Parameters dialog box, double-click the block.

Parameters

Associated VRML file — Virtual world 3D file specifying the virtual world to which text is output.

Associated Text node — Text node within the virtual world to which text is output.

Format string — Format used for output text. This block uses `sprintf()` to format the output strings. Like `sprintf()`, it works in a vectorized fashion, where the format string is recycled through the components of the input vector. This block does not support the `%c` and `%s` conversion formats, as signals in the Simulink product cannot have both characters and strings.

Sample time — Enter the sample time or `-1` for inherited sample time.

Ensure that a viewer window is open during simulation — Select this check box to ensure that the Simulink 3D Animation Viewer is open during simulation.

See Also

- VR Sink
- VR Source
- VR To Video
- VR Tracer

Version History

Introduced in R2006b

Extended Capabilities

C/C++ Code Generation

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

Actual data type or capability support depends on block implementation.

VR To Video

Write data from Simulink model to virtual world (video output port enabled)

Libraries:
Simulink 3D Animation

Description

The VR to Video block is equivalent to the VR Sink block, except that its **Show video output port** is selected by default.

To open the Block Parameters dialog box, double-click the block.

See the VR Sink block for details.

Ports

Input

Input 1 — Input signal
real scalar | real vector

Input signal to drive the virtual reality visualization of nodes selected in the **Virtual World Tree**.

Data Types: `double`

Output

Output 1 — Output video stream
3-element vector of video signal dimensions

Use the output port to access the RGB video stream of the VR signal input.

Data Types: `double`

Parameters

Source file — File name specifying virtual world that connects to block
string scalar | character vector

By default, the full path to the associated virtual world 3D file appears in this text box. If you enter only the file name in this box, the software assumes that the virtual world 3D file resides in the same folder as the model file. You can specify a VRML file or an X3D file.

- Click **New** to open an empty default virtual world editor. When you either enter a source file name or use the **Browse** button, the **New** button becomes an **Edit** button.

- Click **Edit** to launch the default virtual world editor with the source file open.
- Click **View** to view the world in the Simulink 3D Animation Viewer or a Web browser.
- Click **Reload** to reload the world after you change it.

Open Viewer automatically — Display virtual world on model load
off | on

Enable this parameter to display the virtual world after loading the Simulink model.

Allow viewing from the Internet — View virtual world over network
off (default) | on

Enable this parameter to make the virtual world accessible for viewing on a client computer. If you do not select this check box, then the world is visible only on the host computer. This parameter is equivalent to the `RemoteView` property of a `vrworld` object.

Description — Virtual reality object description
string scalar | character vector

The description is displayed in all virtual reality object listings, in the title bar of the Simulink 3D Animation Viewer, and in the list of virtual worlds on the Simulink 3D Animation HTML page. This parameter is equivalent to the `Description` property of a `vrworld` object.

Show video output port — Output VR signal to video
on (default) | off

Enables a port to output an RGB video stream for further 2D video processing.

Video output signal dimensions — Specify video output size
`[200 320]` (default) | 2-element vector

Specify the dimensions ([height width]) of the video output signal in pixels.

Show node types — Display node types in virtual world tree
off (default) | on

Enable this parameter to show node types in the virtual world tree.

Show field types — Display field types in virtual world tree
off (default) | on

Enable this parameter to show field types in the virtual scene tree.

Version History

Introduced in R2007b

See Also

VR Sink

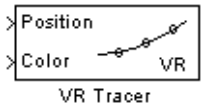
Topics

“Virtual World Data Types”

“Use C/C++ S-Functions as Sim Viewing Devices in External Mode” (Simulink)

VR Tracer

Trace trajectory of object in associated virtual scene



Libraries:
Simulink 3D Animation

Description

The VR Tracer block allows you to trace the trajectory of an object in the associated virtual scene.

This block creates marker nodes in regular time steps either as children of the specified parent node (**Parent node** parameter) or at the top level of scene hierarchy (root).

You can specify one of three types of markers:

- General shape
- Line segments connecting object positions in every time step
- Axis-aligned triads for orienting the trajectory in the 3-D space

You can also project traced object positions to a plane or to a point.

Object position input must correspond to the placement of the object in the scene hierarchy. If the traced object resides as a child of a parent object, define the parent object DEF name in the parent node field. If the traced object resides at the top of the scene hierarchy (its position is defined in global scene coordinates), leave this field empty.

The first block input vector determines the position of the marker. The second block input (if enabled by the **Marker color selection** parameter) represents the marker color. The second or third block input vector (depending on whether the marker color input vector is enabled) specifies the project point coordinates.

To open the Block Parameters dialog box, double-click the block.

Ports

Input

Position — Position coordinates of node
3-element vector

Object position input corresponding to the placement of the object in the scene hierarchy. If the traced object resides as a child of a parent object, define the parent object name in the **Parent node (leave empty for root)** parameter.

Data Types: `double`

Color — Marker color
3-element vector

Note This port is enabled when the **Marker color selection** parameter is set to `Block input`

Provide the color to be used for the tracer markers as a 3-element vector of R, G, and B values.

Data Types: `double`

Parameters

Associated VRML file — Virtual world 3D file
string scalar | character vector

Specify the virtual world file used for the 3D viewer.

Parent node (leave empty for root) — — Select node from hierarchy
VR node

Select the node to be traced from the scene hierarchy.

Marker shape — Select marker shape
`None` (default) | `Tetrahedron` | `Pyramid` | `Box` | `Octahedron` | `Sphere`

Select a shape from the provided options to mark the signal trace.

Connect markers with line segments — Display traced path
`on` (default) | `off`

Enable this parameter to connect the markers on the traced object's path.

Place a triad at each marker position — Provide orientation information
`on` (default) | `off`

Enable this parameter to place a triad at each marker position. A triad helps you orient the object trajectory in the *x*-*y*-*z* plane.

Marker scale — Specify size of marker
`[1 1 1]` (default)

Specify a 3-element vector that defines the scaling of predefined marker shapes and triads. This parameter allows accommodation for scenes of various sizes.

Marker color selection — Specify source of marker colors
`Block input` (default) | `Selected from color list` | `Defined as RGB values`

- `Block input` — Disables **Marker color** parameter and relies on the second block input to define the marker color. Selecting this option enables the second block input, to which you can connect a signal for the marker color.
- `Selected from color list` — Enables the **Marker color** parameter. You can select one color from a list for the marker.
- `Defined as RGB values` — Enables **Marker color** parameter to accept RGB values for the marker color.

Marker color — Specify tracer color
`yellow` (default) | `magenta` | `cyan` | `red` | `green` | `blue` | `white` | `black`

Set the tracer marker color from the provided options. This parameter is enabled when you set **Marker color selection** to `Selected from color list`.

Marker color (RGB) — Specify tracer color
`[1 0 0]` (default) | 3-element vector

Set the tracer marker color as a 3-element vector of RGB values, each ranging from 0-255.

Sample time — Block sample time for simulation
0.1 (default) | scalar | vector

Specify the sample time for the block, or specify `-1` to inherit the sample time.

Ensure that a viewer window is open during simulation — Keep viewer open
`off` (default) | `on`

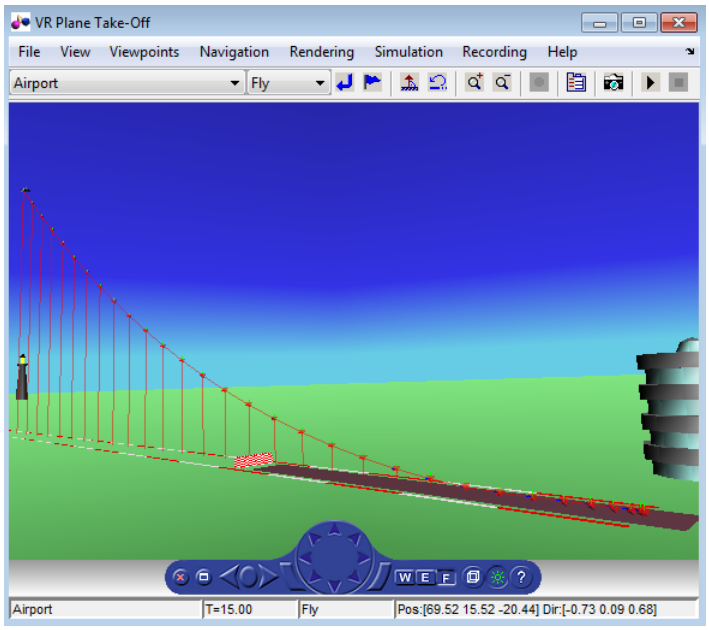
Select this check box to ensure that the Simulink 3D Animation Viewer is open during simulation.

Project positions on a plane — Project path onto plane
`off` (default) | `on`

Specify whether to display line segments from an object onto a plane to approximate the trajectory of the object.

Projection plane equation coefficients (ax+by+cz+d=0) — Specify projection plane as coefficients of vector
`[0 1 0 0]` (default)

When the **Project positions on a plane** parameter is enabled, specify the plane onto which to project the position of the object. The coefficients are in the form `ax+by+cz+d=0`. For example, if you use the default plane equation coefficients to `[0 1 0 0]` for the `vrtkoff_trace` model, then after you simulate the model, the object positions project to the `y=0` plane.



Project positions to a point — Display line segments from marker to projection
`None` (default) | `Defined in the block mask` | `Defined in the block input`

Displays line segments from an object to a point to approximate the trajectory of the object.

- `None` — (Default) No projection to a point.
- `Defined in block mask` — If you select this option, enter coordinates in the **Projection point coordinates** edit box.
- `Defined in the block input` — If you select this option, specify the coordinates of the point in the output of a block that inputs to the VR Tracer block.

Version History

Introduced in R2008b

Extended Capabilities

C/C++ Code Generation

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

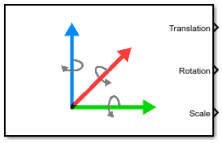
Actual data type or capability support depends on block implementation.

See Also

VR Sink | VR Source | VR To Video | VR Text Output

Simulation 3D Actor Transform Get

Get actor translation, rotation, scale

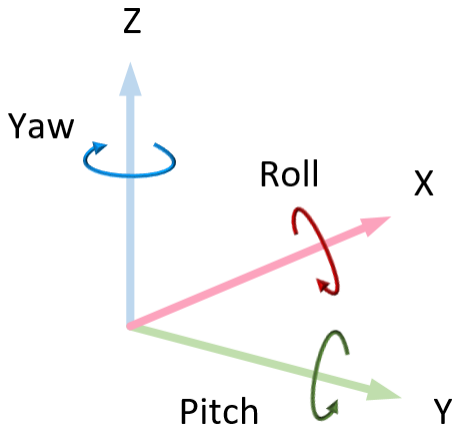


Libraries:
Vehicle Dynamics Blockset / Vehicle Scenarios / Sim3D / Sim3D Core
Aerospace Blockset / Animation / Simulation 3D
Simulink 3D Animation / Simulation 3D

Description

The Simulation 3D Actor Transform Get block provides the actor translation, rotation, and scale for the Simulink simulation environment.

The block uses a vehicle-fixed coordinate system that is initially aligned with the inertial world coordinate system.



Axis	Description
X	Forward direction of the vehicle Roll — Right-handed rotation about *X*-axis
Y	Extends to the right of the vehicle, initially parallel to the ground plane Pitch — Right-handed rotation about *Y*-axis
Z	Extends upwards Yaw — Left-handed rotation about *Z*-axis

Actors are scene objects that support 3D translation, rotation, and scale. Parts are actor components. Components do not exist by themselves; they are associated with an actor.

Tip Verify that the Simulation 3D Scene Configuration block executes before the Simulation 3D Actor Transform Get block. That way, the Unreal Engine 3D visualization environment prepares the data

before the Simulation 3D Actor Transform Get block receives it. To check the block execution order, right-click the blocks and select **Properties**. On the **General** tab, confirm these **Priority** settings:

- Simulation 3D Scene Configuration — `0`
- Simulation 3D Actor Transform Get — `1`

For more information about execution order, see “Control and Display Execution Order” (Simulink).

Ports

Output

Translation — Actor translation
`array`

Actor translation, in m. Array dimensions are number of parts per actor-by-`3`.

- `Translation(1,1)`, `Translation(1,2)`, and `Translation(1,3)` — Vehicle displacement along world *X*-, *Y*, and *Z*- axes, respectively.
- `Translation(...,1)`, `Translation(...,2)`, and `Translation(...,3)` — Actor displacement relative to vehicle, in vehicle-fixed coordinate system initially aligned with world *X*-, *Y*, and *Z*- axes, respectively.

For example, consider a vehicle actor with a vehicle body and four wheels. The `Translation` signal:

- Dimensions are `[5x3]`.
- Contains translation information according to the axle and wheel locations, relative to vehicle.

$$Translation = \begin{bmatrix} X_v & Y_v & Z_v \\ X_{FL} & Y_{FL} & Z_{FL} \\ X_{FR} & Y_{FR} & Z_{FR} \\ X_{RL} & Y_{RL} & Z_{RL} \\ X_{RR} & Y_{RR} & Z_{RR} \end{bmatrix}$$

Translation	Array Element
Vehicle, X_v	`Translation(1,1)`
Vehicle, Y_v	`Translation(1,2)`
Vehicle, Z_v	`Translation(1,3)`
Front left wheel, X_{FL}	`Translation(2,1)`
Front left wheel, Y_{FL}	`Translation(2,2)`
Front left wheel, Z_{FL}	`Translation(2,3)`
Front right wheel, X_{FR}	`Translation(3,1)`
Front right wheel, Y_{FR}	`Translation(3,2)`
Front right wheel, Z_{FR}	`Translation(3,3)`
Rear left wheel, X_{RL}	`Translation(4,1)`
Rear left wheel, Y_{RL}	`Translation(4,2)`

Translation	Array Element
Rear left wheel, Z_{RL}	`Translation(4,3)`
Rear right wheel, X_{RR}	`Translation(5,1)`
Rear right wheel, Y_{RR}	`Translation(5,2)`
Rear right wheel, Z_{RR}	`Translation(5,3)`

Rotation — Actor rotation
`array`

Actor rotation across a [-pi/2, pi/2] range, in rad. Array dimensions are number of parts per actor-by-3.

- `Rotation(1,1)`, `Rotation(1,2)`, and `Rotation(1,3)` — Vehicle rotation about vehicle-fixed pitch, roll, and yaw *Y*-, *Z*-, and *X*- axes, respectively.
- `Rotation(...,1)`, `Rotation(...,2)`, and `Rotation(...,3)` — Actor rotation about vehicle-fixed pitch, roll, and yaw *Y*-, *Z*-, and *X*- axes, respectively.

For example, consider a vehicle actor with a vehicle body and four wheels. The `Rotation` signal:

- Dimensions are `[5x3]`.
- Contains rotation information according to the axle and wheel locations.

$$Rotation = \begin{bmatrix} Pitch_v & Roll_v & Yaw_v \\ Pitch_{FL} & Roll_{FL} & Yaw_{FL} \\ Pitch_{FR} & Roll_{FR} & Yaw_{FR} \\ Pitch_{RL} & Roll_{RL} & Yaw_{RL} \\ Pitch_{RR} & Roll_{RR} & Yaw_{RR} \end{bmatrix}$$

Rotation	Array Element
Vehicle, $Pitch_v$	`Rotation(1,1)`
Vehicle, $Roll_v$	`Rotation(1,2)`
Vehicle, Yaw_v	`Rotation(1,3)`
Front left wheel, $Pitch_{FL}$	`Rotation(2,1)`
Front left wheel, $Roll_{FL}$	`Rotation(2,2)`
Front left wheel, Yaw_{FL}	`Rotation(2,3)`
Front right wheel, $Pitch_{FR}$	`Rotation(3,1)`
Front right wheel, $Roll_{FR}$	`Rotation(3,2)`
Front right wheel, Yaw_{FR}	`Rotation(3,3)`
Rear left wheel, $Pitch_{RL}$	`Rotation(4,1)`
Rear left wheel, $Roll_{RL}$	`Rotation(4,2)`
Rear left wheel, Yaw_{RL}	`Rotation(4,3)`
Rear right wheel, $Pitch_{RR}$	`Rotation(5,1)`
Rear right wheel, $Roll_{RR}$	`Rotation(5,2)`

Rotation	Array Element
Rear right wheel, Yaw_{RR}	`Rotation(5,3)`

Scale — Actor scale
`array`

Actor scale. Array dimensions are number of number of parts per actor-by-3.

- `Scale(1,1)`, `Scale(1,2)`, and `Scale(1,3)` — Vehicle scale along world *X*-, *Y*-, and *Z*- axes, respectively.
- `Scale(...,1)`, `Scale(...,2)`, and `Scale(...,3)` — Actor scale along world *X*-, *Y*-, and *Z*- axes, respectively.

For example, consider a vehicle actor with a vehicle body and four wheels. The `Scale` signal:

- Dimensions are `[5x3]`.
- Contains scale information according to the axle and wheel locations.

$$Scale = \begin{bmatrix} X_{V_{scale}} & Y_{V_{scale}} & Z_{V_{scale}} \\ X_{FL_{scale}} & Y_{FL_{scale}} & Z_{FL_{scale}} \\ X_{FR_{scale}} & Y_{FR_{scale}} & Z_{FR_{scale}} \\ X_{RL_{scale}} & Y_{RL_{scale}} & Z_{RL_{scale}} \\ X_{RR_{scale}} & Y_{RR_{scale}} & Z_{RR_{scale}} \end{bmatrix}$$

Scale	Array Element
Vehicle, $X_{v_{scale}}$	`Scale(1,1)`
Vehicle, $Y_{v_{scale}}$	`Scale(1,2)`
Vehicle, $Z_{v_{scale}}$	`Scale(1,3)`
Front left wheel, $X_{FL_{scale}}$	`Scale(2,1)`
Front left wheel, $Y_{FL_{scale}}$	`Scale(2,2)`
Front left wheel, $Z_{FL_{scale}}$	`Scale(2,3)`
Front right wheel, $X_{FR_{scale}}$	`Scale(3,1)`
Front right wheel, $Y_{FR_{scale}}$	`Scale(3,2)`
Front right wheel, $Z_{FR_{scale}}$	`Scale(3,3)`
Rear left wheel, $X_{RL_{scale}}$	`Scale(4,1)`
Rear left wheel, $Y_{RL_{scale}}$	`Scale(4,2)`
Rear left wheel, $Z_{RL_{scale}}$	`Scale(4,3)`
Rear right wheel, $X_{RR_{scale}}$	`Scale(5,1)`
Rear right wheel, $Y_{RR_{scale}}$	`Scale(5,2)`
Rear right wheel, $Z_{RR_{scale}}$	`Scale(5,3)`

Parameters

Tag for actor in 3D scene, ActorTag — Name
`SimulinkActor1` (default) | character vector

Actor name.

Actors are scene objects that support 3D translation, rotation, and scale. Parts are actor components. Components do not exist by themselves; they are associated with an actor.

The block does not support multiple instances of the same actor tag. To refer to the same scene actor when you use the 3D block pairs (e.g. Simulation 3D Actor Transform Get and Simulation 3D Actor Transform Set), specify the same **Tag for actor in 3D scene, ActorTag** parameter.

Number of parts per actor to get, NumberOfParts — Name
`1` (default) | `scalar`

Number of parts per actor. Actors are scene objects that support 3D translation, rotation, and scale. Parts are actor components. Components do not exist by themselves; they are associated with an actor. Typically, a vehicle actor with a body and four wheels has 5 parts.

The block does not support multiple instances of the same actor tag. To refer to the same scene actor when you use the 3D block pairs (e.g. Simulation 3D Actor Transform Get and Simulation 3D Actor Transform Set), specify the same **Tag for actor in 3D scene, ActorTag** parameter.

Sample time — Sample time
`-1` (default) | `scalar`

Sample time, T_s. The graphics frame rate is the inverse of the sample time.

Version History

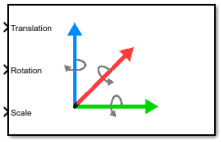
Introduced in R2022b

See Also

Simulation 3D Actor Transform Set | Simulation 3D Camera Get | Simulation 3D Scene Configuration

Simulation 3D Actor Transform Set

Set actor translation, rotation, scale

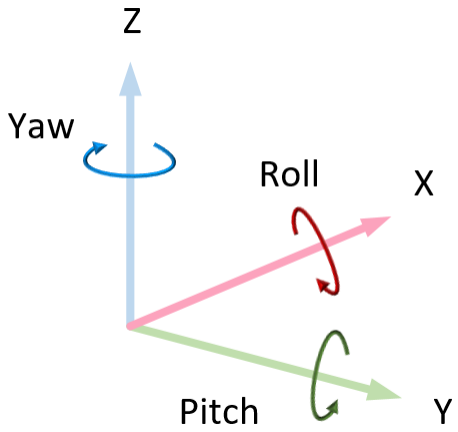


Libraries:
Vehicle Dynamics Blockset / Vehicle Scenarios / Sim3D / Sim3D Core
Aerospace Blockset / Animation / Simulation 3D
Simulink 3D Animation / Simulation 3D

Description

The Simulation 3D Actor Transform Set block sets the actor translation, rotation, and scale in the 3D visualization environment.

The block uses a vehicle-fixed coordinate system that is initially aligned with the inertial world coordinate system.



Axis	Description
X	Forward direction of the vehicle Roll — Right-handed rotation about *X*-axis
Y	Extends to the right of the vehicle, initially parallel to the ground plane Pitch — Right-handed rotation about *Y*-axis
Z	Extends upwards Yaw — Left-handed rotation about *Z*-axis

Actors are scene objects that support 3D translation, rotation, and scale. Parts are actor components. Components do not exist by themselves; they are associated with an actor.

Tip Verify that the Simulation 3D Actor Transform Set block executes before the Simulation 3D Scene Configuration block. That way, Simulation 3D Actor Transform Set prepares the signal data

before the Unreal Engine 3D visualization environment receives it. To check the block execution order, right-click the blocks and select **Properties**. On the **General** tab, confirm these **Priority** settings:

- Simulation 3D Scene Configuration — `0`
- Simulation 3D Actor Transform Set — `-1`

For more information about execution order, see “Control and Display Execution Order” (Simulink).

Ports

Input

Translation — Actor translation
`array`

Actor translation, in m. Array dimensions are number of parts per actor-by-`3`.

- `Translation(1,1)`, `Translation(1,2)`, and `Translation(1,3)` — Vehicle displacement along world *X*-, *Y*, and *Z*- axes, respectively.
- `Translation(...,1)`, `Translation(...,2)`, and `Translation(...,3)` — Actor displacement relative to vehicle, in vehicle-fixed coordinate system initially aligned with world *X*-, *Y*, and *Z*- axes, respectively.

For example, consider a vehicle actor with a vehicle body and four wheels. The `Translation` signal:

- Dimensions are `[5x3]`.
- Contains translation information according to the axle and wheel locations, relative to vehicle.

$$Translation = \begin{bmatrix} X_v & Y_v & Z_v \\ X_{FL} & Y_{FL} & Z_{FL} \\ X_{FR} & Y_{FR} & Z_{FR} \\ X_{RL} & Y_{RL} & Z_{RL} \\ X_{RR} & Y_{RR} & Z_{RR} \end{bmatrix}$$

Translation	Array Element
Vehicle, X_v	`Translation(1,1)`
Vehicle, Y_v	`Translation(1,2)`
Vehicle, Z_v	`Translation(1,3)`
Front left wheel, X_{FL}	`Translation(2,1)`
Front left wheel, Y_{FL}	`Translation(2,2)`
Front left wheel, Z_{FL}	`Translation(2,3)`
Front right wheel, X_{FR}	`Translation(3,1)`
Front right wheel, Y_{FR}	`Translation(3,2)`
Front right wheel, Z_{FR}	`Translation(3,3)`
Rear left wheel, X_{RL}	`Translation(4,1)`

Translation	Array Element
Rear left wheel, Y_{RL}	`Translation(4,2)`
Rear left wheel, Z_{RL}	`Translation(4,3)`
Rear right wheel, X_{RR}	`Translation(5,1)`
Rear right wheel, Y_{RR}	`Translation(5,2)`
Rear right wheel, Z_{RR}	`Translation(5,3)`

Rotation — Actor rotation
`array`

Actor rotation across a [-pi/2, pi/2] range, in rad. Array dimensions are number of parts per actor-by-3.

- `Rotation(1,1)`, `Rotation(1,2)`, and `Rotation(1,3)` — Vehicle rotation about vehicle-fixed pitch, roll, and yaw *Y*-, *Z*-, and *X*- axes, respectively.
- `Rotation(...,1)`, `Rotation(...,2)`, and `Rotation(...,3)` — Actor rotation about vehicle-fixed pitch, roll, and yaw *Y*-, *Z*-, and *X*- axes, respectively.

For example, consider a vehicle actor with a vehicle body and four wheels. The `Rotation` signal:

- Dimensions are `[5x3]`.
- Contains rotation information according to the axle and wheel locations.

$$Rotation = \begin{bmatrix} Pitch_v & Roll_v & Yaw_v \\ Pitch_{FL} & Roll_{FL} & Yaw_{FL} \\ Pitch_{FR} & Roll_{FR} & Yaw_{FR} \\ Pitch_{RL} & Roll_{RL} & Yaw_{RL} \\ Pitch_{RR} & Roll_{RR} & Yaw_{RR} \end{bmatrix}$$

Rotation	Array Element
Vehicle, $Pitch_v$	`Rotation(1,1)`
Vehicle, $Roll_v$	`Rotation(1,2)`
Vehicle, Yaw_v	`Rotation(1,3)`
Front left wheel, $Pitch_{FL}$	`Rotation(2,1)`
Front left wheel, $Roll_{FL}$	`Rotation(2,2)`
Front left wheel, Yaw_{FL}	`Rotation(2,3)`
Front right wheel, $Pitch_{FR}$	`Rotation(3,1)`
Front right wheel, $Roll_{FR}$	`Rotation(3,2)`
Front right wheel, Yaw_{FR}	`Rotation(3,3)`
Rear left wheel, $Pitch_{RL}$	`Rotation(4,1)`
Rear left wheel, $Roll_{RL}$	`Rotation(4,2)`
Rear left wheel, Yaw_{RL}	`Rotation(4,3)`
Rear right wheel, $Pitch_{RR}$	`Rotation(5,1)`

Rotation	Array Element
Rear right wheel, $Roll_{RR}$	`Rotation(5,2)`
Rear right wheel, Yaw_{RR}	`Rotation(5,3)`

Scale — Actor scale
`array`

Actor scale. Array dimensions are number of number of parts per actor-by-3.

- `Scale(1,1)`, `Scale(1,2)`, and `Scale(1,3)` — Vehicle scale along world *X*-, *Y*-, and *Z*- axes, respectively.
- `Scale(...,1)`, `Scale(...,2)`, and `Scale(...,3)` — Actor scale along world *X*-, *Y*-, and *Z*- axes, respectively.

For example, consider a vehicle actor with a vehicle body and four wheels. The `Scale` signal:

- Dimensions are `[5x3]`.
- Contains scale information according to the axle and wheel locations.

$$Scale = \begin{bmatrix} X_{V_{scale}} & Y_{V_{scale}} & Z_{V_{scale}} \\ X_{FL_{scale}} & Y_{FL_{scale}} & Z_{FL_{scale}} \\ X_{FR_{scale}} & Y_{FR_{scale}} & Z_{FR_{scale}} \\ X_{RL_{scale}} & Y_{RL_{scale}} & Z_{RL_{scale}} \\ X_{RR_{scale}} & Y_{RR_{scale}} & Z_{RR_{scale}} \end{bmatrix}$$

Scale	Array Element
Vehicle, $X_{v_{scale}}$	`Scale(1,1)`
Vehicle, $Y_{v_{scale}}$	`Scale(1,2)`
Vehicle, $Z_{v_{scale}}$	`Scale(1,3)`
Front left wheel, $X_{FL_{scale}}$	`Scale(2,1)`
Front left wheel, $Y_{FL_{scale}}$	`Scale(2,2)`
Front left wheel, $Z_{FL_{scale}}$	`Scale(2,3)`
Front right wheel, $X_{FR_{scale}}$	`Scale(3,1)`
Front right wheel, $Y_{FR_{scale}}$	`Scale(3,2)`
Front right wheel, $Z_{FR_{scale}}$	`Scale(3,3)`
Rear left wheel, $X_{RL_{scale}}$	`Scale(4,1)`
Rear left wheel, $Y_{RL_{scale}}$	`Scale(4,2)`
Rear left wheel, $Z_{RL_{scale}}$	`Scale(4,3)`
Rear right wheel, $X_{RR_{scale}}$	`Scale(5,1)`
Rear right wheel, $Y_{RR_{scale}}$	`Scale(5,2)`
Rear right wheel, $Z_{RR_{scale}}$	`Scale(5,3)`

Parameters

Actor Setup

Tag for actor in 3D scene, ActorTag — Name
`SimulinkActor1` (default) | character vector

Actor name.

Actors are scene objects that support 3D translation, rotation, and scale. Parts are actor components. Components do not exist by themselves; they are associated with an actor.

The block does not support multiple instances of the same actor tag. To refer to the same scene actor when you use the 3D block pairs (e.g. Simulation 3D Actor Transform Get and Simulation 3D Actor Transform Set), specify the same **Tag for actor in 3D scene, ActorTag** parameter.

Number of parts per actor to set, NumberOfParts — Name
`1` (default) | `scalar`

Number of parts per actor. Actors are scene objects that support 3D translation, rotation, and scale. Parts are actor components. Components do not exist by themselves; they are associated with an actor. Typically, a vehicle actor with a body and four wheels has 5 parts.

The block does not support multiple instances of the same actor tag. To refer to the same scene actor when you use the 3D block pairs (e.g. Simulation 3D Actor Transform Get and Simulation 3D Actor Transform Set), specify the same **Tag for actor in 3D scene, ActorTag** parameter.

Initial Values

Initial array values to translate actor per part, Translation — Actor initial position
`[0 0 0]` (default) | `array`

Actor initial position, along world *X*-, *Y*-, and *Z*- axes, in m.

Array dimensions are number of parts per actor-by-3.

- `Translation(1,1)`, `Translation(1,2)`, and `Translation(1,3)` — Vehicle displacement along world *X*-, *Y*, and *Z*- axes, respectively.
- `Translation(...,1)`, `Translation(...,2)`, and `Translation(...,3)` — Actor displacement relative to vehicle, in vehicle-fixed coordinate system initially aligned with world *X*-, *Y*, and *Z*- axes, respectively.

For example, consider a vehicle actor with a vehicle body and four wheels. The parameter:

- Dimensions are `[5x3]`.
- Contains translation information according to the axle and wheel locations, relative to vehicle.

$$Translation = \begin{bmatrix} X_v & Y_v & Z_v \\ X_{FL} & Y_{FL} & Z_{FL} \\ X_{FR} & Y_{FR} & Z_{FR} \\ X_{RL} & Y_{RL} & Z_{RL} \\ X_{RR} & Y_{RR} & Z_{RR} \end{bmatrix}$$

Translation	Array Element
Vehicle, X_v	`Translation(1,1)`
Vehicle, Y_v	`Translation(1,2)`
Vehicle, Z_v	`Translation(1,3)`
Front left wheel, X_{FL}	`Translation(2,1)`
Front left wheel, Y_{FL}	`Translation(2,2)`
Front left wheel, Z_{FL}	`Translation(2,3)`
Front right wheel, X_{FR}	`Translation(3,1)`
Front right wheel, Y_{FR}	`Translation(3,2)`
Front right wheel, Z_{FR}	`Translation(3,3)`
Rear left wheel, X_{RL}	`Translation(4,1)`
Rear left wheel, Y_{RL}	`Translation(4,2)`
Rear left wheel, Z_{RL}	`Translation(4,3)`
Rear right wheel, X_{RR}	`Translation(5,1)`
Rear right wheel, Y_{RR}	`Translation(5,2)`
Rear right wheel, Z_{RR}	`Translation(5,3)`

Initial array values to rotate actor per part, Rotation — Actor initial rotation
`[0 0 0]` (default) | `array`

Actor initial rotation about world *X*-, *Y*-, and *Z*- axes across a [-pi/2, pi/2] range, in rad.

Array dimensions are number of parts per actor-by-3.

- `Rotation(1,1)`, `Rotation(1,2)`, and `Rotation(1,3)` — Vehicle rotation about vehicle-fixed pitch, roll, and yaw *Y*-, *Z*-, and *X*- axes, respectively.
- `Rotation(...,1)`, `Rotation(...,2)`, and `Rotation(...,3)` — Actor rotation about vehicle-fixed pitch, roll, and yaw *Y*-, *Z*-, and *X*- axes, respectively.

For example, consider a vehicle actor with a vehicle body and four wheels. The parameter:

- Dimensions are `[5x3]`.
- Contains rotation information according to the axle and wheel locations.

$$Rotation = \begin{bmatrix} Pitch_v & Roll_v & Yaw_v \\ Pitch_{FL} & Roll_{FL} & Yaw_{FL} \\ Pitch_{FR} & Roll_{FR} & Yaw_{FR} \\ Pitch_{RL} & Roll_{RL} & Yaw_{RL} \\ Pitch_{RR} & Roll_{RR} & Yaw_{RR} \end{bmatrix}$$

Rotation	Array Element
Vehicle, $Pitch_v$	`Rotation(1,1)`
Vehicle, $Roll_v$	`Rotation(1,2)`
Vehicle, Yaw_v	`Rotation(1,3)`

Rotation	Array Element
Front left wheel, $Pitch_{FL}$	`Rotation(2,1)`
Front left wheel, $Roll_{FL}$	`Rotation(2,2)`
Front left wheel, Yaw_{FL}	`Rotation(2,3)`
Front right wheel, $Pitch_{FR}$	`Rotation(3,1)`
Front right wheel, $Roll_{FR}$	`Rotation(3,2)`
Front right wheel, Yaw_{FR}	`Rotation(3,3)`
Rear left wheel, $Pitch_{RL}$	`Rotation(4,1)`
Rear left wheel, $Roll_{RL}$	`Rotation(4,2)`
Rear left wheel, Yaw_{RL}	`Rotation(4,3)`
Rear right wheel, $Pitch_{RR}$	`Rotation(5,1)`
Rear right wheel, $Roll_{RR}$	`Rotation(5,2)`
Rear right wheel, Yaw_{RR}	`Rotation(5,3)`

Initial array values to scale actor per part, Scale — Actor initial scale
`[1 1 1]` (default) | `array`

Actor initial scale.

Array dimensions are number of number of parts per actor-by-3.

- `Scale(1,1)`, `Scale(1,2)`, and `Scale(1,3)` — Vehicle scale along world *X*-, *Y*, and *Z*- axes, respectively.
- `Scale(...,1)`, `Scale(...,2)`, and `Scale(...,3)` — Actor scale along world *X*-, *Y*, and *Z*- axes, respectively.

For example, consider a vehicle actor with a vehicle body and four wheels. The parameter:

- Dimensions are `[5x3]`.
- Contains scale information according to the axle and wheel locations.

$$Scale = \begin{bmatrix} X_{V_{scale}} & Y_{V_{scale}} & Z_{V_{scale}} \\ X_{FL_{scale}} & Y_{FL_{scale}} & Z_{FL_{scale}} \\ X_{FR_{scale}} & Y_{FR_{scale}} & Z_{FR_{scale}} \\ X_{RL_{scale}} & Y_{RL_{scale}} & Z_{RL_{scale}} \\ X_{RR_{scale}} & Y_{RR_{scale}} & Z_{RR_{scale}} \end{bmatrix}$$

Scale	Array Element	Scale Axis
Vehicle, $X_{v_{scale}}$	`Scale(1,1)`	World *X*-axis
Vehicle, $Y_{v_{scale}}$	`Scale(1,2)`	World *Y*-axis
Vehicle, $Z_{v_{scale}}$	`Scale(1,3)`	World *Z*-axis
Front left wheel, $X_{FL_{scale}}$	`Scale(2,1)`	World *X*-axis
Front left wheel, $Y_{FL_{scale}}$	`Scale(2,2)`	World *Y*-axis

Scale	Array Element	Scale Axis
Front left wheel, $Z_{FL_{scale}}$	`Scale(2,3)`	World *Z*-axis
Front right wheel, $X_{FR_{scale}}$	`Scale(3,1)`	World *X*-axis
Front right wheel, $Y_{FR_{scale}}$	`Scale(3,2)`	World *Y*-axis
Front right wheel, $Z_{FR_{scale}}$	`Scale(3,3)`	World *Z*-axis
Rear left wheel, $X_{RL_{scale}}$	`Scale(4,1)`	World *X*-axis
Rear left wheel, $Y_{RL_{scale}}$	`Scale(4,2)`	World *Y*-axis
Rear left wheel, $Z_{RL_{scale}}$	`Scale(4,3)`	World *Z*-axis
Rear right wheel, $X_{RR_{scale}}$	`Scale(5,1)`	World *X*-axis
Rear right wheel, $Y_{RR_{scale}}$	`Scale(5,2)`	World *Y*-axis
Rear right wheel, $Z_{RR_{scale}}$	`Scale(5,3)`	World *Z*-axis

Sample time — Sample time
`-1` (default) | `scalar`

Sample time, T_s. The graphics frame rate is the inverse of the sample time.

Version History

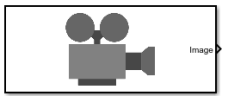
Introduced in R2022b

See Also

Simulation 3D Actor Transform Get | Simulation 3D Camera Get | Simulation 3D Scene Configuration

Simulation 3D Camera Get

Camera image



Libraries:
Vehicle Dynamics Blockset / Vehicle Scenarios / Sim3D / Sim3D Core
Aerospace Blockset / Animation / Simulation 3D
Simulink 3D Animation / Simulation 3D

Description

The Simulation 3D Camera Get block provides an interface to an ideal camera in the 3D visualization environment. The image output is a red, green, and blue (RGB) array.

If you set the sample time to `-1`, the block uses the sample time specified in the Simulation 3D Scene Configuration block. To use this sensor, ensure that the Simulation 3D Scene Configuration block is in your model.

Tip Verify that the Simulation 3D Scene Configuration block executes before the Simulation 3D Camera Get block. That way, the Unreal Engine 3D visualization environment prepares the data before the Simulation 3D Camera Get block receives it. To check the block execution order, right-click the blocks and select **Properties**. On the **General** tab, confirm these **Priority** settings:

- Simulation 3D Scene Configuration — `0`
- Simulation 3D Camera Get — `1`

For more information about execution order, see “Control and Display Execution Order” (Simulink).

Ports

Output

Image — 3D output camera image
m-by-*n*-by-3 array of RGB triplet values

3D output camera image, returned as an *m*-by-*n*-by-3 array of RGB triplet values. *m* is the vertical resolution of the image, and *n* is the horizontal resolution of the image.

Data Types: `int8` | `uint8`

Parameters

Mounting

Sensor identifier — Number to identify unique sensor
`0` (default) | positive integer

Unique sensor identifier, specified as a positive integer. This number is used to identify a specific sensor. The sensor identifier distinguishes between sensors in a multi-sensor system.

Example: 2

Vehicle name — Name of a vehicle
`Scene Origin` (default) | character vector

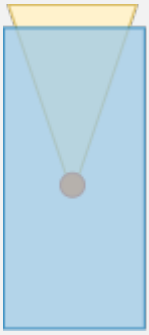
Vehicle name. Block provides a list of vehicles in the model. If you select `Scene Origin`, the block places a sensor at the scene origin.

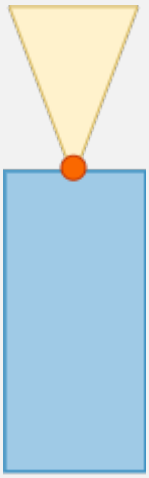
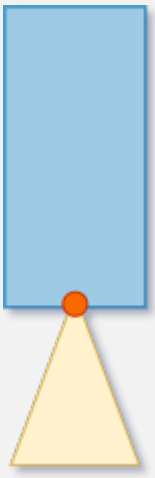
Example: `SimulinkVehicle1`

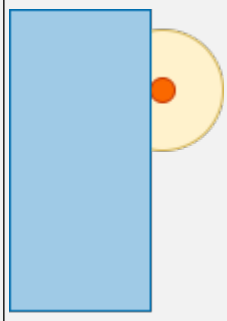
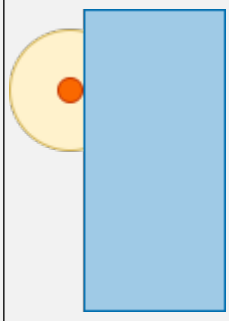
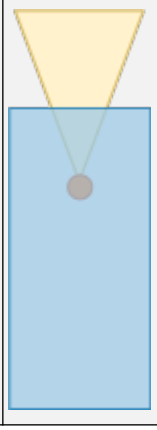
Vehicle mounting location — Sensor mounting location
`Origin` (default) | `Front bumper` | `Rear bumper` | `Right mirror` | `Left mirror` | `Rearview mirror` | `Hood center` | `Roof center`

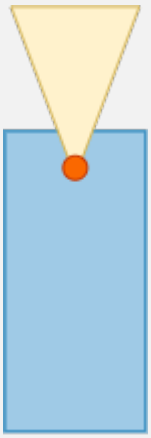
Sensor mounting location.

- When **Vehicle name** is `Scene Origin`, the block mounts the sensor to the origin of the scene, and **Mounting location** can be set to `Origin` only. During simulation, the sensor remains stationary.
- When **Vehicle name** is the name of a vehicle (for example, `SimulinkVehicle1`) the block mounts the sensor to one of the predefined mounting locations described in the table. During simulation, the sensor travels with the vehicle.

Vehicle Mounting Location	Description	Orientation Relative to Vehicle Origin [Roll, Pitch, Yaw] (deg)
`Origin`	Forward-facing sensor mounted to the vehicle origin, which is on the ground and at the geometric center of the vehicle 	[0, 0, 0]

Vehicle Mounting Location	Description	Orientation Relative to Vehicle Origin [Roll, Pitch, Yaw] (deg)
`Front bumper`	Forward-facing sensor mounted to the front bumper 	[0, 0, 0]
`Rear bumper`	Backward-facing sensor mounted to the rear bumper 	[0, 0, 180]

Vehicle Mounting Location	Description	Orientation Relative to Vehicle Origin [Roll, Pitch, Yaw] (deg)
`Right mirror`	Downward-facing sensor mounted to the right side-view mirror 	[0, –90, 0]
`Left mirror`	Downward-facing sensor mounted to the left side-view mirror 	[0, –90, 0]
`Rearview mirror`	Forward-facing sensor mounted to the rearview mirror, inside the vehicle 	[0, 0, 0]

Vehicle Mounting Location	Description	Orientation Relative to Vehicle Origin [Roll, Pitch, Yaw] (deg)
`Hood center`	Forward-facing sensor mounted to the center of the hood 	[0, 0, 0]
`Roof center`	Forward-facing sensor mounted to the center of the roof 	[0, 0, 0]

The (*X*, *Y*, *Z*) location of the sensor relative to the vehicle depends on the vehicle type. To specify the vehicle type, use the **Type** parameter of the Simulation 3D Scene Configuration block to which you are mounting. The tables show the *X*, *Y*, and *Z* locations of sensors in the vehicle coordinate system. In this coordinate system:

- The *X*-axis points forward from the vehicle.
- The *Y*-axis points to the left of the vehicle, as viewed when facing forward.
- The *Z*-axis points up from the ground.
- Roll, pitch, and yaw are clockwise-positive when looking in the positive direction of the *X*-axis, *Y*-axis, and *Z*-axis, respectively. When looking at a vehicle from the top down, then the yaw angle (that is, the orientation angle) is counterclockwise-positive, because you are looking in the negative direction of the axis.

Box Truck — Sensor Locations Relative to Vehicle Origin

Mounting Location	X (m)	Y (m)	Z (m)
`Front bumper`	5.10	0	0.60
`Rear bumper`	–5	0	0.60
`Right mirror`	2.90	1.60	2.10
`Left mirror`	2.90	–1.60	2.10
`Rearview mirror`	2.60	0.20	2.60
`Hood center`	3.80	0	2.10
`Roof center`	1.30	0	4.20

Hatchback — Sensor Locations Relative to Vehicle Origin

Mounting Location	X (m)	Y (m)	Z (m)
`Front bumper`	1.93	0	0.51
`Rear bumper`	–1.93	0	0.51
`Right mirror`	0.43	–0.84	1.01
`Left mirror`	0.43	0.84	1.01
`Rearview mirror`	0.32	0	1.27
`Hood center`	1.44	0	1.01
`Roof center`	0	0	1.57

Muscle Car — Sensor Locations Relative to Vehicle Origin

Mounting Location	X (m)	Y (m)	Z (m)
`Front bumper`	2.47	0	0.45
`Rear bumper`	–2.47	0	0.45
`Right mirror`	0.43	–1.08	1.01
`Left mirror`	0.43	1.08	1.01
`Rearview mirror`	0.32	0	1.20
`Hood center`	1.28	0	1.14
`Roof center`	–0.25	0	1.58

Sedan — Sensor Locations Relative to Vehicle Origin

Mounting Location	X (m)	Y (m)	Z (m)
`Front bumper`	2.42	0	0.51
`Rear bumper`	–2.42	0	0.51
`Right mirror`	0.59	–0.94	1.09
`Left mirror`	0.59	0.94	1.09
`Rearview mirror`	0.43	0	1.31
`Hood center`	1.46	0	1.11
`Roof center`	–0.45	0	1.69

Small Pickup Truck — Sensor Locations Relative to Vehicle Origin

Mounting Location	X (m)	Y (m)	Z (m)
`Front bumper`	3.07	0	0.51
`Rear bumper`	–3.07	0	0.51
`Right mirror`	1.10	–1.13	1.52
`Left mirror`	1.10	1.13	1.52
`Rearview mirror`	0.85	0	1.77
`Hood center`	2.22	0	1.59
`Roof center`	0	0	2.27

Sport Utility Vehicle — Sensor Locations Relative to Vehicle Origin

Mounting Location	X (m)	Y (m)	Z (m)
`Front bumper`	2.42	0	0.51
`Rear bumper`	–2.42	0	0.51
`Right mirror`	0.60	–1	1.35
`Left mirror`	0.60	1	1.35
`Rearview mirror`	0.39	0	1.55
`Hood center`	1.58	0	1.39
`Roof center`	–0.56	0	2

Example: `Origin`

Specify offset — Specify offset from mounting location
`off` (default) | `on`

Select this parameter to specify an offset from the mounting location.

Relative translation [X, Y, Z] — Translation offset from mounting location
`[0,0,0]` (default) | real-valued 1-by-3 vector

Specify a translation offset from the mount location, about the vehicle coordinate system X, Y, and Z axes. Units are in meters.

- The *X*-axis points forward from the vehicle.
- The *Y*-axis points to the left of the vehicle, as viewed when facing forward.
- The *Z*-axis points up.

Example: `[0,0,0.01]`

Dependencies

To enable this parameter, select **Specify offset**.

Relative rotation [Roll, Pitch, Yaw] — Rotational offset from mounting location
`[0,0,0]` (default) | real-valued 1-by-3 vector

Specify a rotational offset from the mounting location, about the vehicle coordinate system *X*, *Y*, and *Z* axes. Units are in degrees.

- Roll angle is the angle of rotation about the *X*-axis of the vehicle coordinate system. A positive roll angle corresponds to a clockwise rotation when looking in the positive direction of the *X*-axis.
- Pitch angle is the angle of rotation about the *Y*-axis of the vehicle coordinate system. A positive pitch angle corresponds to a clockwise rotation when looking in the positive direction of the *Y*-axis.
- Yaw angle is the angle of rotation about the *Z* of the vehicle coordinate system. A positive yaw angle corresponds to a clockwise rotation when looking in the positive direction of the *Z*-axis.

Example: `[0,0,10]`

Dependencies

To enable this parameter, select **Specify offset**.

Sample time — Sample time
`-1` (default) | positive scalar

Sample time of the block in seconds. The 3D simulation environment frame rate is the inverse of the sample time.

If you set the sample time to `-1`, the block uses the sample time specified in the Simulation 3D Scene Configuration block.

Parameter

Horizontal resolution — Pixels
`uint32(1280)` (default) | `scalar`

Horizontal image resolution, in pixels.

Vertical resolution — Pixels
`uint32(720)` (default) | `scalar`

Vertical image resolution, in pixels.

Horizontal field of view — Field of view
`single(60)` (default) | `scalar`

Horizontal field of view (FOV), in deg.

Version History

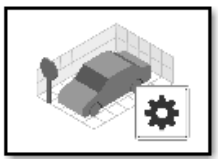
Introduced in R2022b

See Also

Simulation 3D Actor Transform Get | Simulation 3D Actor Transform Set | Simulation 3D Scene Configuration

Simulation 3D Scene Configuration

Scene configuration for 3D simulation environment



Libraries:
Vehicle Dynamics Blockset / Vehicle Scenarios / Sim3D / Sim3D Core
Aerospace Blockset / Animation / Simulation 3D
Automated Driving Toolbox / Simulation 3D
UAV Toolbox / Simulation 3D
Simulink 3D Animation / Simulation 3D

Description

The Simulation 3D Scene Configuration block implements a 3D simulation environment that is rendered by using the Unreal Engine from Epic Games. integrates the 3D simulation environment with Simulink so that you can query the world around the vehicle and virtually test perception, control, and planning algorithms. Using this block, you can also control the position of the sun and the weather conditions of a scene. For more details, see Sun Position and Weather on page 2-77.

You can simulate from a set of prebuilt scenes or from your own custom scenes. Scene customization requires the support package. For more details, see .

Note The Simulation 3D Scene Configuration block must execute after blocks that send data to the 3D environment and before blocks that receive data from the 3D environment. To verify the execution order of such blocks, right-click the blocks and select **Properties**. Then, on the **General** tab, confirm these **Priority** settings:

- For blocks that send data to the 3D environment, such as Simulation 3D Vehicle with Ground Following blocks, **Priority** must be set to `-1`. That way, these blocks prepare their data before the 3D environment receives it.
- For the Simulation 3D Scene Configuration block in your model, **Priority** must be set to `0`.
- For blocks that receive data from the 3D environment, such as blocks, **Priority** must be set to `1`. That way, the 3D environment can prepare the data before these blocks receive it.

For more information about execution order, see “Control and Display Execution Order” (Simulink).

Parameters

Scene

Scene Selection

Scene source — Source of scene

`Default Scenes` (default) | `Unreal Executable` | `Unreal Editor`

Source of the scene in which to simulate, specified as one of the options in the table.

Option	Description
`Default Scenes`	Simulate in one of the default, prebuilt scenes specified in the **Scene name** parameter.
`Unreal Executable`	Simulate in a scene that is part of an Unreal Engine executable file. Specify the executable file in the **Project name** parameter. Specify the scene in the **Scene** parameter. Select this option to simulate in custom scenes that have been packaged into an executable for faster simulation.
`Unreal Editor`	Simulate in a scene that is part of an Unreal Engine project (`.uproject`) file and is open in the Unreal® Editor. Specify the project file in the **Project** parameter. Select this option when developing custom scenes. By clicking **Open Unreal Editor**, you can co-simulate within Simulink and the Unreal Editor and modify your scenes based on the simulation results.

Scene name — Name of prebuilt 3D scene

`Empty scene` (default)

Name of the prebuilt 3D scene in which to simulate, specified as `Empty Scene` (Empty Scene).

Dependencies

To enable this parameter, set **Scene source** to `Default Scenes`.

Project name — Name of Unreal Engine executable file

`VehicleSimulation.exe` (default) | valid executable file name

Name of the Unreal Engine executable file, specified as a valid executable project file name. You can either browse for the file or specify the full path to the project file, using backslashes. To specify a scene from this file to simulate in, use the **Scene** parameter.

By default, **Project name** is set to `VehicleSimulation.exe`, which is on the MATLAB search path.

Example: `C:\Local\WindowsNoEditor\AutoVrtlEnv.exe`

Dependencies

To enable this parameter, set **Scene source** to `Unreal Executable`.

Scene — Name of scene from executable file

`/Game/Maps/HwStrght` (default) | path to valid scene name

Name of a scene from the executable file specified by the **Project name** parameter, specified as a path to a valid scene name.

When you package scenes from an Unreal Engine project into an executable file, the Unreal Editor saves the scenes to an internal folder within the executable file. This folder is located at the path `/Game/Maps`. Therefore, you must prepend `/Game/Maps` to the scene name. You must specify this path using forward slashes. For the file name, do not specify the `.umap` extension. For example, if the scene from the executable in which you want to simulate is named `myScene.umap`, specify **Scene** as `/Game/Maps/myScene`.

Alternatively, you can browse for the scene in the corresponding Unreal Engine project. These scenes are typically saved to the `Content/Maps` subfolder of the project. This subfolder contains all the scenes in your project. The scenes have the extension `.umap`. Select one of the scenes that you packaged into the executable file specified by the **Project name** parameter. Use backward slashes and specify the `.umap` extension for the scene.

By default, **Scene** is set to `/Game/Maps/HwStrght`, which is a scene from the default `VehicleSimulation.exe` executable file specified by the **Project name** parameter. This scene corresponds to the prebuilt **Straight Road** scene.

Example: `/Game/Maps/scene1`

Example: `C:\Local\myProject\Content\Maps\scene1.umap`

Dependencies

To enable this parameter, set **Scene source** to `Unreal Executable`.

Project — Name of Unreal Engine project file

valid project file name

Name of the Unreal Engine project file, specified as a valid project file name. You can either browse for the file or specify the full path to the file, using backslashes. The file must contain no spaces. To simulate scenes from this project in the Unreal Editor, click **Open Unreal Editor**. If you have an Unreal Editor session open already, then this button is disabled.

To run the simulation, in Simulink, click **Run**. Before you click **Play** in the Unreal Editor, wait until the Diagnostic Viewer window displays this confirmation message:

```
In the Simulation 3D Scene Configuration block, you set the scene source to 'Unreal Editor'.
In Unreal Editor, select 'Play' to view the scene.
```

This message confirms that Simulink has instantiated the scene actors, including the vehicles and cameras, in the Unreal Engine 3D environment. If you click **Play** before the Diagnostic Viewer window displays this confirmation message, Simulink might not instantiate the actors in the Unreal Editor.

Dependencies

To enable this parameter, set **Scene source** to `Unreal Editor`.

Scene Parameters

Scene view — Configure placement of virtual camera that displays scene

`Scene Origin` | vehicle name

Configure the placement of the virtual camera that displays the scene during simulation.

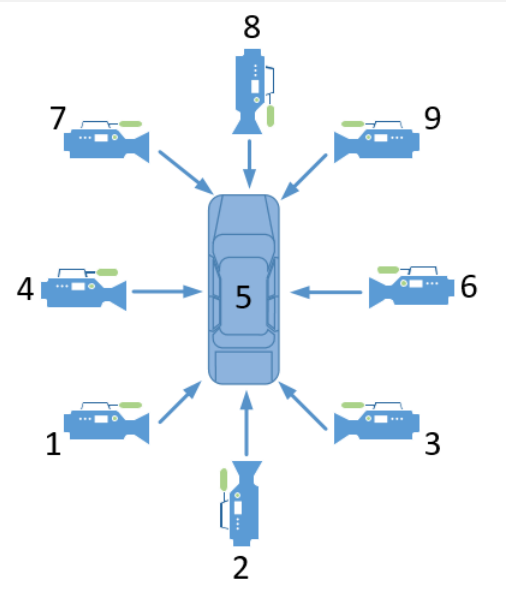
- If your model contains no blocks, then during simulation, you view the scene from a camera positioned at the scene origin.
- If your model contains at least one vehicle block, then by default, you view the scene from behind the first vehicle that was placed in your model. To change the view to a different vehicle, set **Scene view** to the name of that vehicle. The **Scene view** parameter list is populated with all the **Name** parameter values of the vehicle blocks contained in your model.

If you add a Simulation 3D Scene Configuration block to your model before adding any vehicle blocks, the virtual camera remains positioned at the scene. To reposition the camera to follow a vehicle, update this parameter.

When **Scene view** is set to a vehicle name, during simulation, you can change the location of the camera around the vehicle.

To smoothly change the camera views, use these key commands.

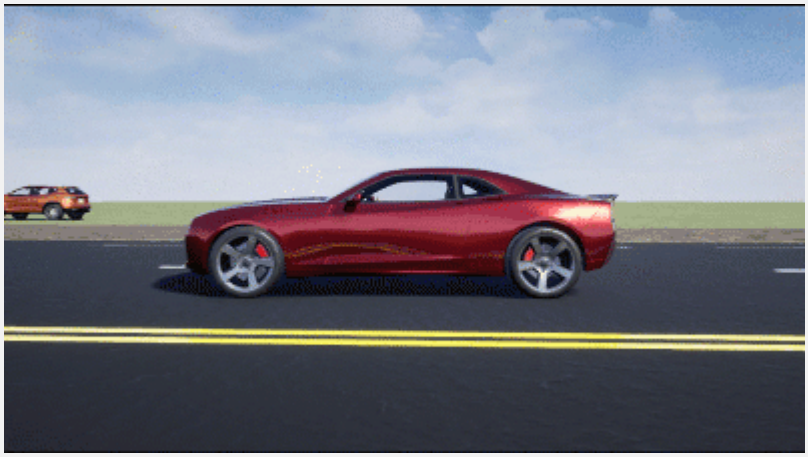
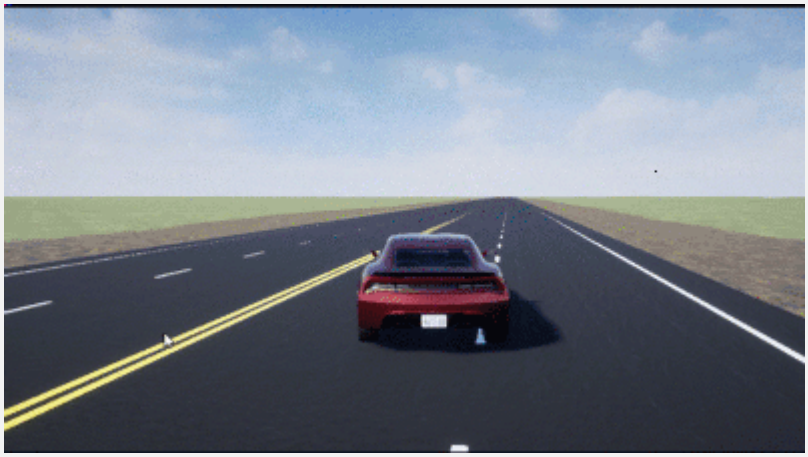
Key	Camera View
1	Back left
2	Back
3	Back right
4	Left
5	Internal
6	Right
7	Front left
8	Front
9	Front right

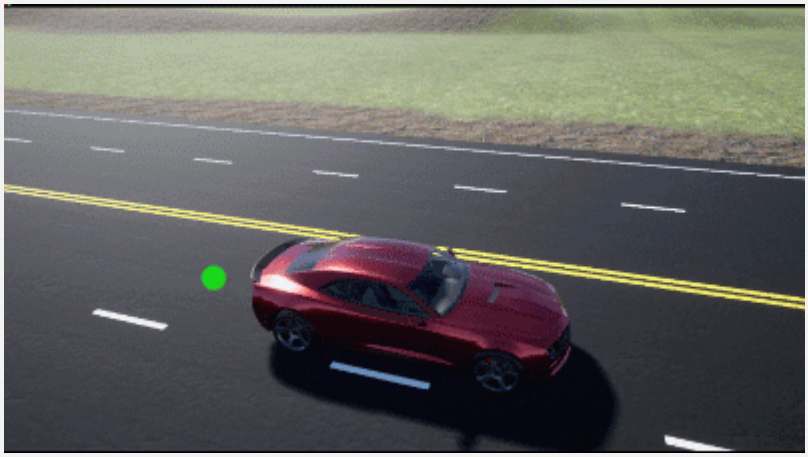


Key	Camera View	
0	Overhead	**View Animated GIF** 

For additional camera controls, use these key commands.

Key	Camera Control
Tab	Cycle the view between all vehicles in the scene. **View Animated GIF** 

Key	Camera Control
Mouse scroll wheel	Control the camera distance from the vehicle. **View Animated GIF** 
L	Toggle a camera lag effect on or off. When you enable the lag effect, the camera view includes: • Position lag, based on the vehicle translational acceleration • Rotation lag, based on the vehicle rotational velocity This lag enables improved visualization of overall vehicle acceleration and rotation. **View Animated GIF** 

Key	Camera Control
F	Toggle the free camera mode on or off. When you enable the free camera mode, you can use the mouse to change the pitch and yaw of the camera. This mode enables you to orbit the camera around the vehicle. **View Animated GIF** 

Sample time — Sample time of visualization engine

(default) | scalar greater than or equal to 0.01

Sample time, T_s, of the visualization engine, specified as a scalar greater than or equal to 0.01. Units are in seconds.

The graphics frame rate of the visualization engine is the inverse of the sample time. For example, if **Sample time** is `1/60`, then the visualization engine solver tries to achieve a frame rate of 60 frames per second. However, the real-time graphics frame rate is often lower due to factors such as graphics card performance and model complexity.

By default, blocks that receive data from the visualization engine, such as blocks, inherit this sample rate.

Display 3D simulation window — Unreal Engine visualization

`on` (default) | `off`

Select whether to run simulations in the 3D visualization environment without visualizing the results, that is, in headless mode.

Consider running in headless mode in these cases:

- You want to run multiple 3D simulations in parallel to test models in different Unreal Engine scenarios.

Dependencies

To enable this parameter, set **Scene source** to `Default Scenes` or `Unreal Executable`.

Weather

Override scene weather — Control the scene weather and sun position

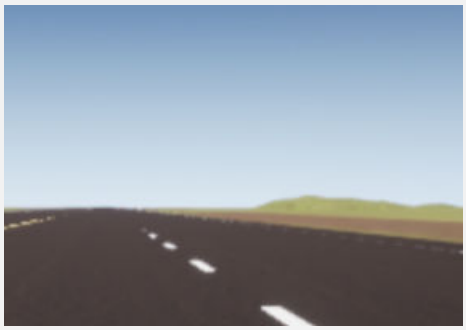
`off` (default) | `on`

Select whether to control the scene weather and sun position during simulation. Use the enabled parameters to change the sun position, clouds, fog, and rain.

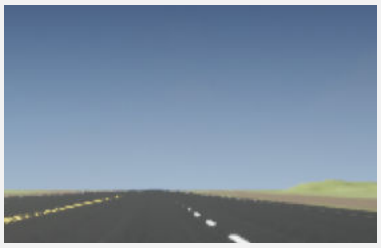
This table summarizes sun position settings for specific times of day.

Time of Day	Settings	Unreal Editor Environment
Midnight	**Sun altitude**: -90 **Sun azimuth**: 180	
Sunrise in the north	**Sun altitude**: 0 **Sun azimuth**: 180	
Noon	**Sun altitude**: 90 **Sun azimuth**: 180	

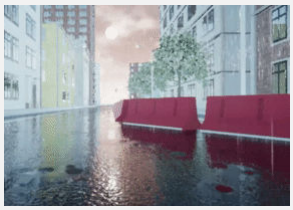
This table summarizes settings for specific cloud conditions.

Cloud Condition	Settings	Unreal Editor Environment
Clear	**Cloud opacity**: 0	
Heavy	**Cloud opacity**: 85	

This table summarizes settings for specific fog conditions.

Fog Condition	Settings	Unreal Editor Environment
None	**Fog density**: 0	
Heavy	**Fog density**: 100	

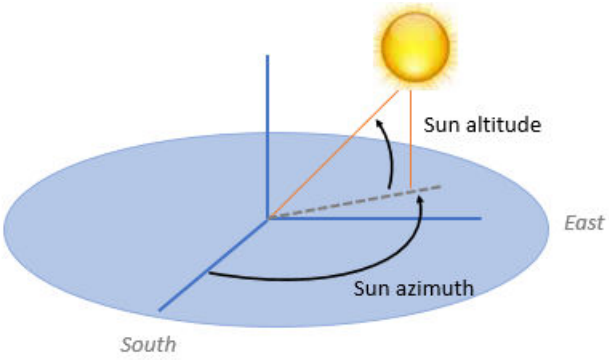
This table summarizes settings for specific rain conditions.

Rain Condition	Settings	Unreal Editor Environment
Light	**Cloud opacity**: 10 **Rain density**: 25	
Heavy	**Cloud opacity**: 10 **Rain density**: 80	

Sun altitude — Altitude angle between sun and horizon

`40` (default) | any value between `-90` and `90`

Altitude angle in a vertical plane between the sun's rays and the horizontal projection of the rays, in deg.



Use the **Sun altitude** and **Sun azimuth** parameters to control the time of day in the scene. For example, to specify sunrise in the north, set **Sun altitude** to 0 deg and **Sun azimuth** to 180 deg.

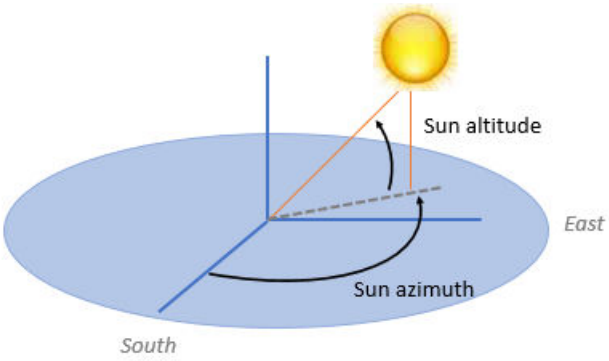
Dependencies

To enable this parameter, select **Override scene weather**.

Sun azimuth — Azimuth angle from south to horizontal projection of the sun ray

`90` (default) | any value between `0` and `360`

Azimuth angle in the horizontal plane measured from the south to the horizontal projection of the sun rays, in deg.



Use the **Sun altitude** and **Sun azimuth** parameters to control the time of day in the scene. For example, to specify sunrise in the north, set **Sun altitude** to 0 deg and **Sun azimuth** to 180 deg.

Dependencies

To enable this parameter, select **Override scene weather**.

Cloud opacity — Unreal Editor Cloud Opacity global actor target value

`10` (default) | any value between `0` and `100`

Parameter that corresponds to the Unreal Editor **Cloud Opacity** global actor target value, in percent. Zero is a cloudless scene.

Use the **Cloud opacity** and **Cloud speed** parameters to control clouds in the scene.

Dependencies

To enable this parameter, select **Override scene weather**.

Cloud speed — Unreal Editor Cloud Speed global actor target value

`1` (default) | any value between `-100` and `100`

Parameter that corresponds to the Unreal Editor **Cloud Speed** global actor target value. The clouds move from west to east for positive values and east to west for negative values.

Use the **Cloud opacity** and **Cloud speed** parameters to control clouds in the scene.

Dependencies

To enable this parameter, select **Override scene weather**.

Fog density — Unreal Editor Set Fog Density and Set Start Distance target values

`0` (default) | any value between `0` and `100`

Parameter that corresponds to the Unreal Editor **Set Fog Density** and **Set Start Distance** target values, in percent.

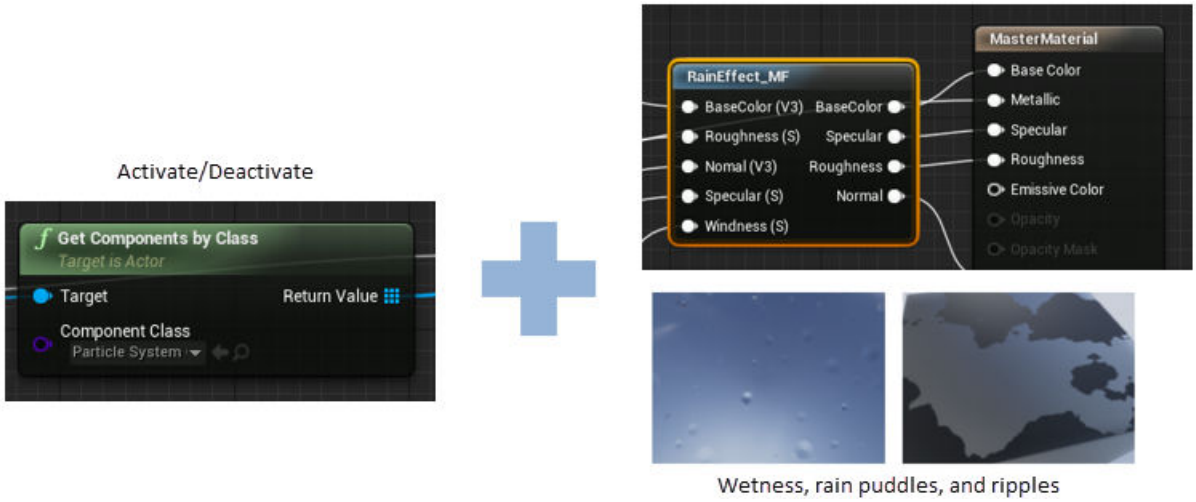
Dependencies

To enable this parameter, select **Override scene weather**.

Rain density — Unreal Editor local actor controlling rain density, wetness, rain puddles, and ripples

`0` (default) | any value between `0` and `100`

Parameter corresponding to the Unreal Editor local actor that controls rain density, wetness, rain puddles, and ripples, in percent.



Wetness, rain puddles, and ripples

Use the **Cloud opacity** and **Rain density** parameters to control rain in the scene.

Dependencies

To enable this parameter, select **Override scene weather**.

More About

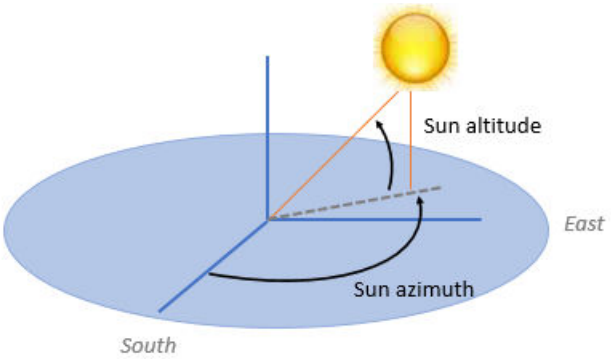
Sun Position and Weather

To control the scene weather and sun position, on the **Weather** tab, select **Override scene weather**. Use the enabled parameters to change the sun position, clouds, fog, and rain during the simulation.

Sun Position

Use **Sun altitude** and **Sun azimuth** to control the sun position.

- **Sun altitude** — Altitude angle in a vertical plane between the sun rays and the horizontal projection of the rays.
- **Sun azimuth** — Azimuth angle in the horizontal plane measured from the south to the horizontal projection of the sun rays.



This table summarizes sun position settings for specific times of day.

Time of Day	Settings	Unreal Editor Environment
Midnight	**Sun altitude**: -90 **Sun azimuth**: 180	
Sunrise in the north	**Sun altitude**: 0 **Sun azimuth**: 180	
Noon	**Sun altitude**: 90 **Sun azimuth**: 180	

Clouds

Use **Cloud opacity** and **Cloud speed** to control clouds in the scene.

- **Cloud opacity** — Unreal Editor **Cloud Opacity** global actor target value. Zero is a cloudless scene.
- **Cloud speed** — Unreal Editor **Cloud Speed** global actor target value. The clouds move from west to east for positive values and east to west for negative values.

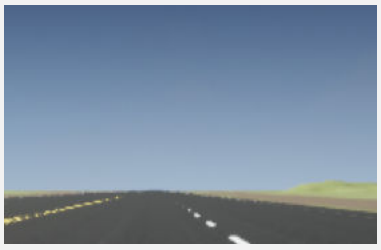
This table summarizes settings for specific cloud conditions.

Cloud Condition	Settings	Unreal Editor Environment
Clear	**Cloud opacity**: 0	
Heavy	**Cloud opacity**: 85	

Fog

Use **Fog density** to control fog in the scene. **Fog density** corresponds to the Unreal Editor **Set Fog Density**.

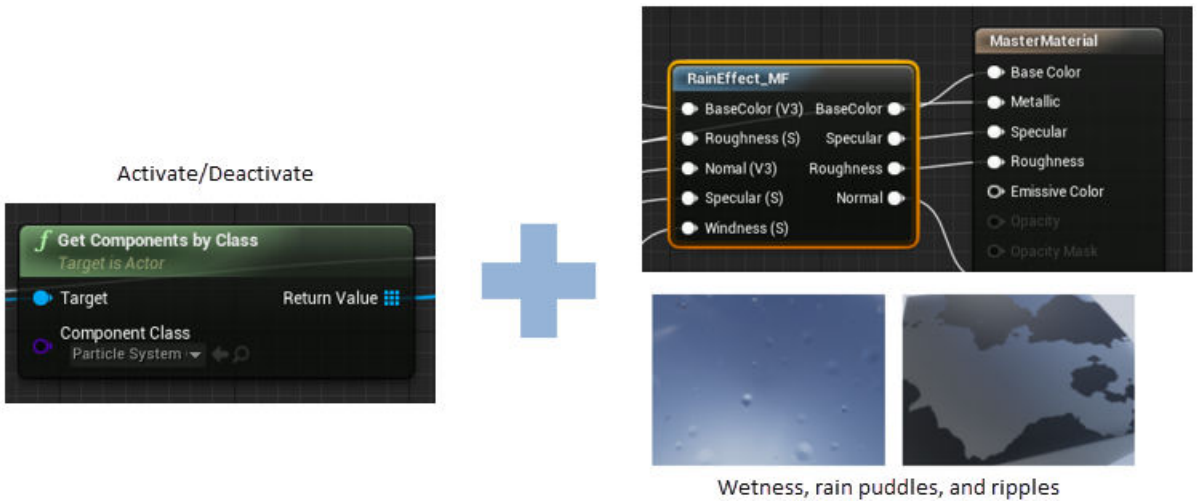
This table summarizes settings for specific fog conditions.

Fog Condition	Settings	Unreal Editor Environment
None	**Fog density**: 0	
Heavy	**Fog density**: 100	

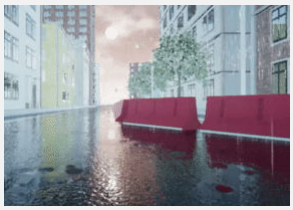
Rain

Use **Cloud opacity** and **Rain density** to control rain in the scene.

- **Cloud opacity** — Unreal Editor **Cloud Opacity** global actor target value.
- **Rain density** — Unreal Editor local actor that controls rain density, wetness, rain puddles, and ripples.



This table summarizes settings for specific rain conditions.

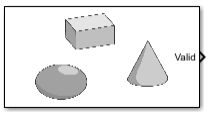
Rain Condition	Settings	Unreal Editor Environment
Light	**Cloud opacity**: 10 **Rain density**: 25	
Heavy	**Cloud opacity**: 10 **Rain density**: 80	

Version History

Introduced in R2022b

Simulation 3D Actor

Define actors in Unreal Engine viewer



Libraries:
Simulink 3D Animation / Simulation 3D

Description

The Simulation 3D Actor block implements a generic actor in the Unreal Engine viewer. You can use this block to specify actor name and source file, initialize the actor, and define how the actor is created and behaves during simulation.

Ports

Input

Instance — Instance number of actor
real integer

Instance number of actor, specified as a real integer.

- For positive values, a new actor is created if it has not been created already. The reference block refers to this actor if the actor is present in the world.
- For negative values, actor instances are deleted, if they exist. For example, instance 1 is deleted if the value of this port is `-1` at a given time step.

Dependencies

This input is visible when the operating mode of the block is set to either `Create at step` or `Reference by instance number`.

Output

Valid — Validity of specified actor
`false` (default) | `true`

Validity of the specified actor, returned as `false` if the specified actor cannot be found in the world or `true` if the actor is found.

Parameters

Main

Actor Name — Name of actor
`Sim3dActor1` (default) | string

Name of the root actor for the current Simulation 3D Actor block.

Parent Name — Parent of actor
`Scene Origin` (default) | string

Name of the parent actor of the actor, specified as a string.

Operation — Operating mode of block
`Create at setup` (default) | `Create at step` | `Reference by name` | `Reference by instance number`

Operating mode that determines block behavior, specified as one of these options.

- `Create at setup` – Use this option to create actors in the scene. The actors are created during game setup, before the simulation runs. You can also control the simulation of these actors using the **Input**, **Output**, and **Event** ports.
- `Create at step` – Use this option to dynamically create actors when the simulation is running. An **Instance** input port of the type `int` is created in this mode, and the actors are dynamically generated based on the values input to the block. If the value of **Instance** is greater than zero, an actor with the name `'Actor name + Instance value'` is created. The name of the actor is derived from `'Actor Name'` in the block mask. The instance value is the current value of the port. Unlike the `Create at setup` option, you cannot control the actors created using with this option selected in the block mask. You can only create actors in the scene.
- `Reference by name` – Use this option to refer to an actor in the scene using `Actor Name`, and control this actor's simulation using the inputs of the block. Get data of the actor using the outputs or events. This also provides a **Valid** output port.

  If an actor with the specified name is not present, the **Valid** port will return `false` or `0`. In this case, default values will be passed as output for rest of the out ports (if any). Use the `Valid` port to verify whether the other outputs generated are currently valid or not.

  Note **Source file** and **Initialization script** parameters are unavailable in the `Reference by name` operating mode.

- `Reference by instance number` – Use this option to refer to an instance created via a different actor block. Similar to the `Create at step` operating mode, this option creates an **Instance** input in the block, and the effective actor name `'Actor name + Instance value'` is used to reference the actor.

  If an actor with the generated name is not present, the **Valid** port returns `false` or `0`. In this case, default values are passed as output for rest of the output ports, if any. Use the **Valid** port to verify whether the other outputs generated are currently valid or not.

  Note **Source file** and **Initialization script** parameters are unavailable in the `Reference by instance number` operating mode.

Dependencies

The **Transform** tab is available if you set **Operation** as either `Create at setup` or `Create at step`.

Source file — File from which actors are sourced
string

File from which actors are sourced, specified as a string.

This option loads a 3D file or imports a `sim3d.Actor` object, including child actor objects from MAT or STL files.

Alternatively, the **Select** button can also be used to select a file from the Windows file browser.

Initialization script — Text area to set properties of actor
MATLAB script

Text area to set properties of the actor created by the block, specified as a MATLAB script. The order of actor creation is:

1. Root actor with specified name is created.
2. The **Source file** is loaded into the root actor.
3. The **Initialization script** runs.

Use the name `Actor` when modifying the root actor of the block. `Actor` is a reserved handle in the initialization script. Otherwise, use the `findBy` function to get the actor handle of a different name, for example, one of the child actors for example. Then use the retrieved actor handle to initialize that specific actor.

Sample time — Sample time
`-1` (default) | `scalar`

Sample time, T_s. The graphics frame rate is the inverse of the sample time.

Transform

The **Transform** tab is available if you set **Operation** as either `Create at setup` or `Create at step`.

Translation — Relative translation
`[0,0,0]` (default) | real 1-by-3 vector

Translation (x,y,z) of the actor object relative to its parent actor, specified as a real 1-by-3 vector, in meters.

Rotation — Relative rotation
`[0,0,0]` (default) | real 1-by-3 vector

Rotation (roll, pitch, yaw) of the actor object relative to its parent actor, specified as a real 1-by-3 vector, in radians.

Scale — Relative scaling
`[1,1,1]` (default) | real 1-by-3 vector

Relative scaling in x, y and z coordinates, specified as a real 1-by-3 vector.

Inputs

Input ports — List of user-selected input ports
text area

This text area lists the user-selected input ports of the block and provides a button that opens the port selector app, which you can use to modify the input ports. Each line in the text area represents one block port.

You can manually modify the ports using the text area, but using the port selector app is recommended for choosing the ports.

Outputs

Output ports — List of user-selected output ports
text area

This text area lists the user-selected output ports of the block and provides a button that opens the port selector app, which you can use to modify the output ports. Each line in the text area represents one block port.

You can manually modify the ports using the text area, but using the port selector app is recommended for choosing the ports.

More About

Operating Modes

- `Create at setup` – Use this operating mode to create actors before simulation begins and control the actors when the simulation is running. You can control actors in a simulation by adding input ports to the block using the **Inputs** tab in the block dialog. Similarly, you can read specific actor properties using the **Outputs** tab to create custom output ports for the block.

  The Simulation 3D Actor block implements these steps during actor creation:

  1. An empty root actor is created with an `Actor Name` equal to the name specified in the block dialog box.
  2. The source file is used to load actors, meshes, or 3D objects into the root actor.
  3. The initialization script is run to initialize the actors that have been created.

  Note The initialization script and the source file are optional. However, if both are empty, the block will only create an empty actor in the scene.

  You can use the `load` function in the initialization script as an alternative to the source file. This command has the same effect as specifying a source file path.

  ```
  load(actor, "source_file_path")
  ```

- `Create at step` – Use this operating mode to create actors dynamically during simulation runtime. When the **Instance** port is a nonzero number, an instance of the specified actors is created. If the instance input is unique, it will only create a new instance for the first step in which it is equal to the input number.

  When this operating mode is selected, the block creates actor instances by creating an empty root actor is created with name equal to specified values of `Actor Name` + **Instance**.

  So, if the actor name specified in the dialog box is `ActorInst`, and value of `Instance` is specified as `1`, then the `ActorName` for the new instance will be equal to `ActorInst1`. This process is followed for every new instance that is created.

  Note The **Valid** output port for this mode returns `0` (`false`) by default and returns `1` (`true`) after an instance has been created. The value remains `true` even if all instances created by this block

are deleted. So, you can use the valid port to check if an instance has been created. Otherwise this output can be ignored.

- `Reference by name` – Use this operating mode to control an actor that is already present in the scene. The block looks for an actor with the actor name specified in the block dialog. If found, the block controls that specific actor using the specified input ports. You can control the actor in a simulation by adding input ports to the block using the **Inputs** tab in the block dialog.
- `Reference by instance number` – Use this operating mode to control an actor that is already present in the scene. The block looks for an actor with the actor name specified in the block dialog. If found, the block controls that specific actor using the specified input ports. You can control the actor in a simulation by adding input ports to the block using the **Inputs** tab in the block dialog.

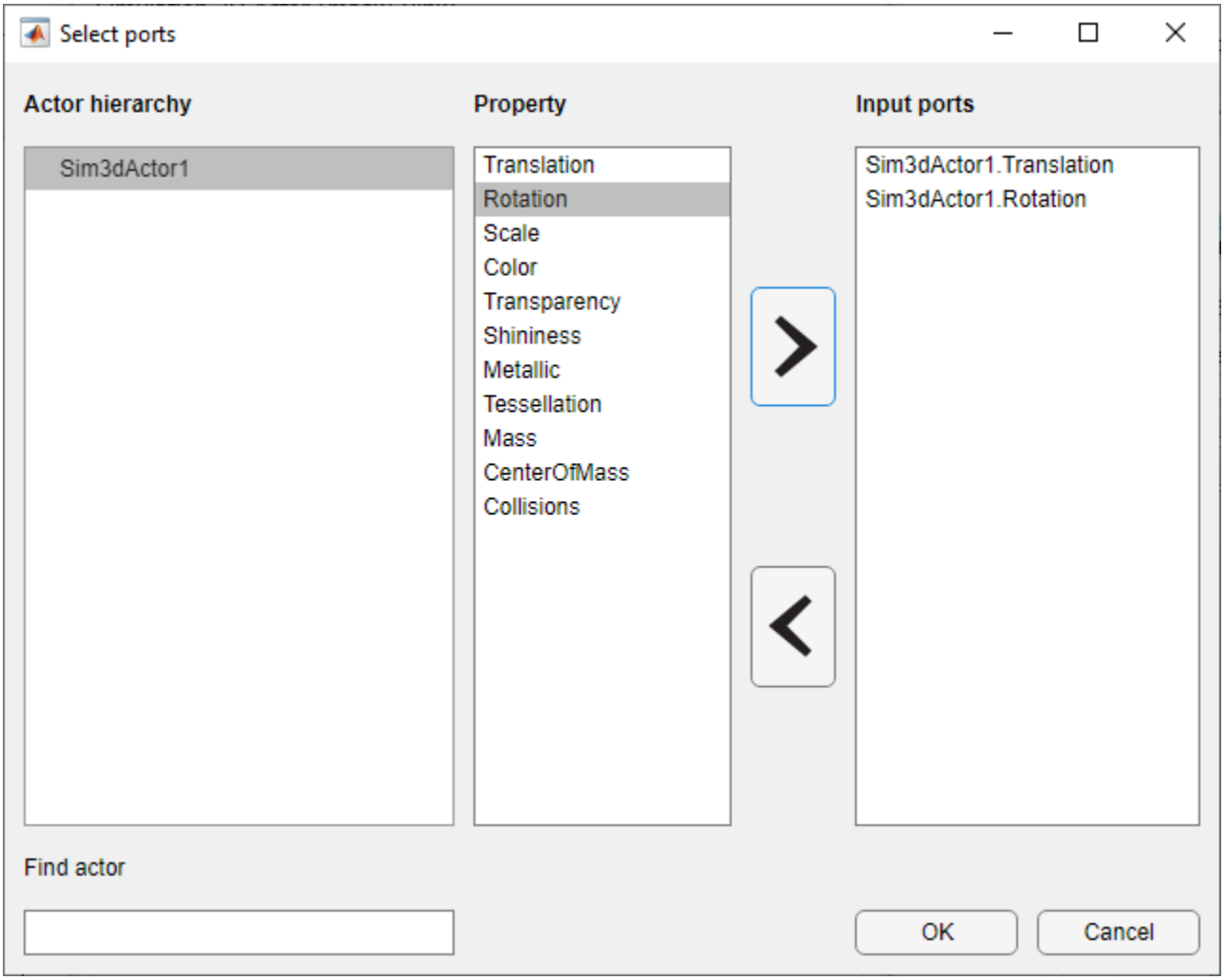
The difference between the `Reference by name` and `Reference by instance number` operating modes is that, based on the value of the **Instance** input port, the Simulation 3D Actor block can control multiple actors during simulation. However, during individual time steps, the block still controls a single actor. In the `Reference by name` operating mode, the block can control only a single actor during the entire simulation.

Input, Output, and Event Port Modification

Use the Simulation 3D Actor block to control simulation during runtime by modifying the block ports. To control actors during simulation, you can use the **Inputs** tab of the block mask to add input ports to the block which can then be used to control specific actors.

Similarly, you can use the output and event ports of the block mask to get data about the actors from Unreal Engine. The output ports can be modified from the **Outputs** and **Event** tabs in the dialog. Selecting event ports on the **Event** tab creates corresponding output ports on the block.

The **Browse** buttons in `Inputs`, `Outputs`, and `Events` text area opens up the port selection app, which you can use to add or delete input and output ports. Although ports can be modified manually, the suggested method is to use the port selection app.



This table provides the possible input, output, and event "Properties" on page 3-179 of `sim3d.Actor`.

Type	Properties
Input and output properties	• `Translation` • `Rotation` • `Scale` • `Color` • `Transparency` • `Shininess` • `Metallic` • `Tessellation` • `Mass` • `CenterOfMass` • `Collisions`

Type	Properties

`Events` properties	• Hit Event
	  • `HitEvent`
	  • `HitSelfID`
	  • `HitLocation`
	  • `HitOtherID`
	  • `HitOtherActorName`
	• Begin Overlap Event
	  • `BeginOverlapEvent`
	  • `BeginOverlapSelfID`
	  • `BeginOverlapOtherID`
	  • `BeginOverlapOtherActorName`
	• End Overlap Event
	  • `EndOverlapEvent`
	  • `EndOverlapSelfID`
	  • `EndOverlapOtherID`
	  • `EndOverlapOtherActorName`
	• Click Event
	  • `ClickEvent`
	  • `ClickActorID`
	  • `ClickLocation`
	  • `ClickActorName`

Actors can be controlled for all operating modes, except `Create at step`. For all other operating modes, these are some special considerations.

- `Create at setup` – The port selection app in this mode lists the entire tree hierarchy of actors created in the actor block, and you can use the block ports to control the simulation of the parent as well as child actors. These actors are present throughout the entire simulation, and the outputs of the blocks are always valid.
- `Reference by name` – In this operating mode, the block can only control a single actor during the simulation, specified using the actor name. If the actor is not present in the world, the block inputs are ignored, the **Valid** output is `false`, and the remaining outputs are default values (invalid).
- `Reference by instance number` – In this operating mode, the Simulation 3D Actor block only controls one actor per time step. In different time steps the block can control different actor based on the value of the **Instance** input. The port selection app for this operating mode uses * as a placeholder for the actor name. The actor being controlled has the name as a concatenated string: actor name (in block dialog) + current instance input value. The * wildcard only applies to the `Reference by instance number` operating mode. Similar to `Reference by name` operating mode, the **Valid** output is `false` whenever the actor currently being referenced is not present in the world.

Version History

Introduced in R2022b

R2023a: Communicate Unreal Engine Events

Behavior changed in R2023a

Use the Simulation 3D Actor block event ports to communicate events in the Unreal Engine simulation 3D environment, including when:

- You click on an actor.
- An actor collides or overlaps with another actor.

See Also

`sim3d.World` | `sim3d.Actor` | `sim3d.sensors.IdealCamera` | `sim3d.sensors.MainCamera`

Functions

stl2vrml

Convert STL file to virtual world file

Syntax

```
stl2vrml(source)
stl2vrml(source,destination)
stl2vrml(source,destination,format)
```

Description

`stl2vrml(source)` converts an ASCII or binary STL file that you specify with `source` to a VRML97-compliant, UTF-8 encoded text file.

The converted VRML file has the same name as the source STL file, except that the extension is `.wrl` instead of `.stl`. The `stl2vrml` function places the VRML file in the current folder.

Tip You can also use the `vrimport` function to import STL files. However, to import Physical Modeling XML files, use the `stl2vrml` function.

`stl2vrml(source,destination)` creates the converted VRML file in the `destination` folder.

`stl2vrml(source,destination,format)` creates the converted virtual world file in the specified format.

Examples

Convert STL File to VRML File

This example uses an STL file in the Simscape™ Multibody™ product.

Convert the STL file `Bar1_Default_sldprt.STL` (in `matlab/toolbox/physmod/sm/smdemos/import/four_bar`) to a VRML file and place the resulting file in the current folder.

```
stl2vrml('Bar1_Default_sldprt.STL')
ls
```

```
.       ..   Bar1_Default_sldprt.wrl
% Other files and folders in the current folder appear.
```

Convert STL File to X3D File

This example uses an STL file in the Simscape Multibody product.

Convert the STL file `Bar1_Default_sldprt.STL` (in `matlab/toolbox/physmod/sm/smdemos/import/four_bar`) to an XML-encoded virtual world file and place the resulting file in a folder called `virtualworlds`.

```
stl2vrml('Bar1_Default_sldprt.STL','virtualworlds','x3d')
ls

.       ..   Bar1_Default_sldprt.x3d
% Other files and folders in the current folder appear, as well.
```

Input Arguments

`source` — STL source file path

character vector

STL source file path, specified as a string. The STL file can be either ASCII or binary.

If the source file is a Physical Modeling XML file, `stl2vrml` converts all STL files referenced in the XML file. It also creates a main assembly VRML file that contains `Inline` references to all converted individual VRML files. Inlines are wrapped by `Transform` nodes with `DEF` names corresponding to the part names defined in their respective STL source files.

`destination` — Path to folder for converted file

character vector

Path to the destination folder for converted file, specified as a string. If the destination folder does not exist, the `stl2vrml` function attempts to create it.

`format` — File format for converted virtual world file

`'wrl'` (VRML) (default) | `'x3d'` (XML-encoded X3D file) | `'x3dv'` (Classic VRML-encoded X3D file)

File format for converted virtual world file, specified as a string.

Tips

- Use the created assembly virtual world files as templates for creating virtual scenes. Edit the scenes. For example, add lights, viewpoints, or surrounding objects, modify part materials, define navigation speeds, and so on.
- The `stl2vrml` function places assembly parts in the global coordinate system. If the `source` is a physical modeling XML file, the resulting virtual world assembly file reflects the initial positions of parts defined in the XML file.
- To use the tree structure of the related SolidWorks® source file in the assembly virtual world file, avoid spaces in assembly and component names. To process the assembly VRML files (but not X3D files), you can use the `vrphysmod` function to obtain a Simulink model with VRML visualization.

Version History

Introduced in R2010b

See Also

`vrimport` | `vrcadcleanup` | `vrphysmod`

Topics

“Import STL and Physical Modeling XML Files”
“Link to Simulink and Simscape Multibody Models”

vrcadcleanup

Clean up virtual world 3D file exported from CAD tools

Syntax

```
vrcadcleanup('filename')
vrcadcleanup('filename', 'hint')
```

Description

`vrcadcleanup('filename')` copies the specified file to a backup file with the extension `bak`. It then modifies the virtual world 3D file exported from Pro/ENGINEER™ or SolidWorks. This cleanup enables the Simulink 3D Animation software to use these files.

`vrcadcleanup` performs the following modifications to VRML files:

- Removal of everything except inlines, viewpoints, and transforms
- Provision of names for inline transforms

Note You can use `vrcadcleanup` with VRML files (`.wrl`), but not with X3D files (`.x3d` or `.x3dv`).

`vrcadcleanup('filename', 'hint')` takes in account the value of `'hint'` during conversion. Possible value of `'hint'` includes:

Argument	Description
`'solidworks'`	Assumes that the software is exporting the original set of virtual world 3D files from SolidWorks. This option adds or increments the numerical suffix to the node names to match the part names that exist in the corresponding physical modeling XML file.

This function expects the input file structure to correspond to the typical output of the specified CAD tools. The typical input file should contain:

- A structure of viewpoints and inline nodes (possibly contained in one layer of transform nodes)
- One inline node for each part of the exported assembly

The function also performs the following:

- Upon output, discards any additional nodes, including transform nodes, that do not contain inline nodes.
- Processes hierarchically organized assemblies, where inline files instead of part geometries contain additional groups of nested node inline nodes. In such subassembly files, copies all inline references to the main virtual world 3D file. The function wraps these inline references with a `Transform` node, using a name that corresponds to the subassembly name.

Note If you call this function for a file that is not a product of a CAD export filter, the output file might be corrupted.

Examples

To clean up the VRML file `four_link.wrl`:

```
vrcadcleanup('four_link.wrl');
```

Version History

Introduced in R2009a

See Also

`stl2vrml` | `vrphysmod` | “Import STL and Physical Modeling XML Files” | “Link to Simulink and Simscape Multibody Models”

vr.canvas class

Create virtual reality canvas

Description

Create a virtual reality canvas.

Construction

`virtualCanvas = vr.canvas(world)` creates a virtual reality canvas showing the specified virtual world.

`virtualCanvas = vrfigure(world,parent)` creates a virtual reality canvas in the specified parent figure or panel. A panel arranges user interface components into groups. By visually grouping related controls, panels can make the user interface easier to understand. A panel can have a title and various borders.

`virtualCanvas = vr.canvas(world,parent,position)` creates a virtual reality canvas in a figure or panel at the specified position.

`virtualCanvas = vr.canvas(world,PropertyName,Value,...,PropertyName,Value)` sets the values of the `vr.canvas` properties specified by one or more `PropertyName,Value` pair arguments.

Input Arguments

`world` — Virtual world

`vrworld` object

Virtual world, specified as a `vrworld` object.

Note Open the virtual world before you create a `vr.canvas` object using that virtual world.

`parent` — Figure for displaying canvas

`figure` object | `uipanel` object

Figure for displaying the canvas, specified as a MATLAB `figure` or `uipanel` object

`position` — Canvas location and size

`vector with four elements`

Location and size of virtual canvas, specified as the vector, in the form `[left bottom width height]`. Specify measurements in pixels.

Note On Windows systems, figure windows cannot be less than 104 pixels wide, regardless of the value of the `position` argument.

Element	Description
`left`	Distance from the left edge of the primary display to the inner left edge of the canvas. This value can be negative on systems that have more than one monitor.
`bottom`	Distance from the bottom edge of the primary display to the inner bottom edge of the canvas. This value can be negative on systems that have more than one monitor.
`width`	Distance between the right and left inner edges of the canvas.
`height`	Distance between the top and bottom inner edges of the canvas.

Example: `[230 250 570 510]`

Data Types: `double`

PropertyName-Value Pair Arguments

Specify optional comma-separated pairs of `PropertyName,Value` arguments. `PropertyName` is the argument name and `Value` is the corresponding value. `PropertyName` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `PropertyName1,Value1,...,PropertyNameN,ValueN`.

Example: `set(myFigure,'Antialiasing','on','CameraPosition',[0 100 100])`

`Antialiasing` — Smooth textures using antialiasing
`'off'` (default) | `'on'`

Smooth textures using antialiasing, specified as `'on'` or `'off'`. Antialiasing smooths textures by interpolating values between texture points.

`CameraBound` — Camera movement with current viewpoint
`'on'` (default) | `'off'`

Camera movement with the current viewpoint, specified as `'on'` or `'off'`.

`CameraDirection` — Camera direction in the current viewpoint local coordinates
vector of three doubles

Camera direction in the current viewpoint local coordinates, specified as a vector of three doubles. The doubles represent the x, y, and z vectors in current viewpoint local coordinates.

`CameraPosition` — Camera position in the current viewpoint local coordinates
vector of three doubles

Camera position in the current viewpoint local coordinates, specified as a vector of three doubles. The doubles represent the x, y, and z vectors in the current viewpoint local coordinates.

`CameraUpVector` — Camera up vector
vector of three doubles

Camera up vector, specified as a vector of three doubles. The doubles represent the x, y, and z vectors in the current viewpoint local coordinates.

`DeleteFcn` — Callback invoked when closing `vr.canvas` object
string

Callback invoked when closing the `vr.canvas` object, specified as a string.

`ExaminePivotPoint` — Pivot point about which camera rotates in `examine` navigation mode
vector of three doubles

Pivot point about which camera rotates in `examine` navigation mode, specified as a vector of three doubles in world coordinates.

`Headlight` — Headlight from camera
`'on'` (default) | `'off'`

Headlight from camera, specified as `'on'` or `'off'`. If you specify `'off'`, the camera does not emit light and the scene can appear dark.

`Lighting` — Lighting effect
`'on'` (default) | `'off'`

Lighting effect, specified as `'on'` or `'off'`. If you specify `'off'`, the camera does not emit light and the scene can appear dark.

`MaxTextureSize` — Maximum pixel size of textures
`'auto'` (default) | integer in a power of 2

Maximum pixel size of textures, specified as `'auto'` or integer in a power of 2. The value of `'auto'` sets the maximum texture pixel size. Otherwise, specify an integer in a power of two that is equal to or less than the video card limit (typically `1024` or `2048`).

The smaller the size, the faster the texture renders. Increasing the size improves image quality but decreases performance.

Note Specifying a value that is unsuitable causes a warning. The Simulink 3D Animation software then adjusts the property to the next smaller suitable value.

Data Types: `int32`

`NavMode` — Navigation mode
`'fly'` (default) | `'examine'` | `'walk'` | `'none'`

Navigation mode, specified as `'fly'`, `'examine'`, `'walk'`, or `'none'`. See “Mouse Navigation”.

`NavPanel` — Navigation panel appearance
`'none'` (default) | `'halfbar'` | `'bar'` | `'opaque'` | `'translucent'`

Navigation panel appearance, specified as one of `'none'`, `'halfbar'`, `'bar'`, `'opaque'`, or `'translucent'`.

`Navspeed` — Navigation speed
`'normal'` (default) | `'slow'` | `'veryslow'` | `'fast'` | `'veryfast'`

Navigation speed, specified as `'normal'`, `'slow'`, `'veryslow'`, `'fast'`, or `'veryfast'`.

`NavZones` — Display navigation zones
`'off'` (default) | `'on'`

Navigation zones display, specified as `'on'` or `'off'`.

`Position` — Canvas location and size
`vector with four doubles`

Location and size of virtual canvas, specified as the vector in the form `[left bottom width height]`. Specify measurements in pixels or normalized, based on the `Units` property setting.

Element	Description
`left`	Distance from the left edge of the primary display to the inner left edge of the canvas. You can specify a negative value on systems that have more than one monitor.
`bottom`	Distance from the bottom edge of the primary display to the inner bottom edge of the canvas. You can specify a negative value on systems that have more than one monitor.
`width`	Distance between the right and left inner edges of the canvas.
`height`	Distance between the top and bottom inner edges of the canvas.

Example: `[230 250 570 510]`

`Sound` — Sound effects
`'on'` (default) | `'off'`

Sound effects, specified as `'on'` or `'off'`.

`Stereo3D` — Stereoscopic vision mode
`'off'` (default) | `'anaglyph'` | `'active'` | `vr.utils.stereo3d` object

Stereoscopic vision mode, specified as `'off'`, `'anaglyph'`, `'active'` or a `vr.utils.stereo3d` object.

Specifying a `vr.utils.stereo3d` object sets the `Stereo3D`, `Stereo3DCameraOffset`, and `Stereo3DHIT` properties. Specifying a `vr.utils.stereo3d` object also sets color filters for the left and right cameras.

Data Types: `int32`

`Stereo3DCameraOffset` — Distance of left and right camera for stereoscopic vision
non-negative floating-point double-precision number

Distance of left and right camera from parallax for stereoscopic vision, specified as a non-negative floating-point double-precision number.

Specifying a `vr.utils.stereo3d` object for the `Stereo3D` property also sets the `Stereo3DCameraOffset` and `Stereo3DHIT` properties and sets color filters for the left and right cameras.

`Stereo3DHIT` — Horizontal image translation (HIT) of two stereoscopic images
double from 0 to 1

Horizontal image translation (HIT) of two stereoscopic images, specified as a double from 0 through 1, inclusive. The larger the value, the further back the background appears.

Specifying a `vr.utils.stereo3d` object for the `Stereo3D` property also sets the `Stereo3DCameraOffset` and `Stereo3DHIT` properties and sets color filters for the left and right cameras.

`Textures` — Texture use
`'on'` (default) | `'off'`

Texture use, specified as `'on'` or `'off'`.

`Tooltips` — Tooltips display
`'on'` (default) | `'off'`

Tooltips display, specified as `'on'` or `'off'`.

`Transparency` — Transparency effect
`'on'` (default) | `'off'`

Transparency effect, specified as `'on'` or `'off'`.

`Triad` — Triad location
`'bottomleft'` (default) | `'bottomright'` | `'center` | `'topleft'` | `'topright'` | `'none'`

Triad location, specified `'bottomleft'`, `'bottomright'`, `'center`, `'topleft'`, `'topright'`, or `'none'`.

`Units` — Units for `Position` property
`'pixels'` (default) | `'normalized'`

Units for `Position` property, specified as `'pixels'` or `'normalized'`.

`Viewpoint` — Active viewpoint of figure
string

Active viewpoint of a figure, specified as a string. If the active viewpoint has no description, use an empty string.

`Wireframe` — Wireframe display
`'off'` (default) | `'on'`

Wireframe display, specified as `'on'` or `'off'`.

`ZoomFactor` — Camera zoom factor
1 (default) | floating-point number

Camera zoom factor, specified as a floating-point number. A zoom factor of `2` makes the scene look twice as large. A zoom factor of `0.1` makes it look 10 times smaller, and so forth.

Output Arguments

`virtualCanvas` — Virtual reality canvas
`vr.canvas` object

Virtual reality canvas, represented by a `vr.canvas` object

Properties

`Antialiasing` — Smooth textures using antialiasing

`'off'` (default) | `'on'`

Smooth textures using antialiasing, returned as `'on'` or `'off'`. Antialiasing smooths textures by interpolating values between texture points.

`CameraBound` — Camera movement with current viewpoint

`'on'` (default) | `'off'`

Camera movement with the current viewpoint, returned as `'on'` or `'off'`.

`CameraDirection` — Camera direction in the current viewpoint local coordinates

vector of three doubles

Camera direction in the current viewpoint local coordinates, specified as a vector of three doubles. The doubles represent the x, y, and z vectors in current viewpoint local coordinates.

`CameraDirectionAbs` — Camera direction in world coordinates

vector of three doubles

Camera direction in world coordinates, returned as a vector of three doubles (read-only property).

`CameraPosition` — Current camera position in the current viewpoint local coordinates

vector of three doubles

Camera position in the current viewpoint local coordinates, returned as a vector of three doubles. The doubles represent the x, y, and z vectors in the current viewpoint local coordinates.

`CameraPositionAbs` — Camera position in world coordinates

vector of three doubles

Camera direction in world coordinates, represented by a vector of three doubles (read-only property).

`CameraUpVector` — Camera up vector

vector of three doubles

Camera up vector, returned as a vector of three doubles. The doubles represent the x, y, and z vectors in the current viewpoint local coordinates.

`CameraUpVectorAbs` — Camera up vector in world coordinates

vector of three doubles

Camera up vector in world coordinates, represented by a vector of three doubles (read-only property).

`DeleteFcn` — Callback invoked when closing `vr.canvas` object

string

Callback invoked when closing the `vr.canvas` object, returned as a string.

`ExaminePivotPoint` — Pivot point about which camera rotates in `examine` navigation mode

vector of three doubles

Pivot point about which camera rotates in `examine` navigation mode, returned as a vector of three doubles in world coordinates.

`Headlight` — Headlight from camera
`'on'` (default) | `'off'`

Headlight from camera, returned as `'on'` or `'off'`. If set to `'off'`, the camera does not emit light and the scene can appear dark.

`Lighting` — Lighting effect
`'on'` (default) | `'off'`

Lighting effect, returned as `'on'` or `'off'`. If set to `'off'`, the camera does not emit light and the scene can appear dark.

`MaxTextureSize` — Maximum pixel size of textures
`'auto'` (default) | integer in a power of 2

Maximum pixel size of a texture used. The smaller the size, the faster the texture can render. A value of `'auto'` means the texture is set to the maximum pixel size.

Data Types: `int32`

`NavMode` — Navigation mode
`'fly'` (default) | `'examine'` | `'walk'` | `'none'`

Navigation mode, returned as `'fly'`, `'examine'`, `'walk'`, or `'none'`. See “Mouse Navigation”.

`NavPanel` — Navigation panel appearance
`'none'` (default) | `'halfbar'` | `'bar'` | `'opaque'` | `'translucent'`

Navigation panel appearance, returned as `'none'`, `'halfbar'`, `'bar'`, `'opaque'`, or `'translucent'`.

`Navspeed` — Navigation speed
`'normal'` (default) | `'slow'` | `'veryslow'` | `'fast'` | `'veryfast'`

Navigation speed, returned as `'normal'`, `'slow'`, `'veryslow'`, `'fast'`, or `'veryfast'`.

`NavZones` — Display navigation zones
`'off'` (default) | `'on'`

Navigation zones display, returned as `'on'` or `'off'`.

`Parent` — Handle of parent of virtual reality canvas object
double

Handle of parent of virtual reality canvas object, represented by a double (read-only property).

`Position` — Canvas location and size
`vector with four doubles`

Location and size of virtual canvas, returned as the vector in the form `[left bottom width height]`. Specify measurements in pixels or normalized, based on the `Units` property setting.

Note On Windows systems, figure windows cannot be less than 104 pixels wide, regardless of the value of the `Position` property.

Element	Description
`left`	Distance from the left edge of the primary display to the inner left edge of the canvas. You can specify a negative value on systems that have more than one monitor.
`bottom`	Distance from the bottom edge of the primary display to the inner bottom edge of the canvas. You can specify a negative value on systems that have more than one monitor.
`width`	Distance between the right and left inner edges of the canvas.
`height`	Distance between the top and bottom inner edges of the canvas.

Example: `[230 250 570 510]`

Sound — Sound effects
`'on'` (default) | `'off'`

Sound effects, returned as `'on'` or `'off'`.

`Stereo3D` — Stereoscopic vision mode
`'off'` (default) | `'anaglyph'` | `'active'` | `vr.utils.stereo3d` object

Stereoscopic vision mode, returned as `'off'`, `'anaglyph'`, `'active'` or a `vr.utils.stereo3d` object.

Specifying a `vr.utils.stereo3d` object sets the `Stereo3D`, `Stereo3DCameraOffset`, and `Stereo3DHIT` properties. Specifying a `vr.utils.stereo3d` object also sets color filters for the left and right cameras.

`Stereo3DCameraOffset` — Distance of left and right camera for stereoscopic vision
non-negative floating-point double-precision number

Distance of left and right camera from parallax for stereoscopic vision, specified as a non-negative floating-point double-precision number.

Specifying a `vr.utils.stereo3d` object for the `Stereo3D` property also sets the `Stereo3DCameraOffset` and `Stereo3DHIT` properties and sets color filters for the left and right cameras.

`Stereo3DHIT` — Horizontal image translation (HIT) of two stereoscopic images
double from 0 to 1

Horizontal image translation (HIT) of two stereoscopic images, returned as a double from 0 through 1, inclusive. The larger the value, the further back the background appears. By default, the background image is at zero and the foreground image appears to pop out from the monitor toward the person viewing the virtual world.

Specifying a `vr.utils.stereo3d` object for the `Stereo3D` property also sets the `Stereo3DCameraOffset` and `Stereo3DHIT` properties and sets color filters for the left and right cameras.

`Textures` — Texture use
`'on'` (default) | `'off'`

Texture use, returned as `'on'` or `'off'`.

`Tooltips` — Tooltips display
`'on'` (default) | `'off'`

Tooltips display, returned as `'on'` or `'off'`.

`Transparency` — Transparency effect
`'on'` (default) | `'off'`

Transparency effect, returned as `'on'` or `'off'`.

`Triad` — Triad location
`'bottomleft'` (default) | `'bottomright'` | `'center` | `'topleft'` | `'topright'` | `'none'`

Triad location, returned as `'bottomleft'`, `'bottomright'`, `'center`, `'topleft'`, `'topright'`, or `'none'`.

`Units` — Units for `Position` property
`'pixels'` (default) | `'normalized'`

Units for `Position` property, returned as `'pixels'` or `'normalized'`.

`Viewpoint` — Active viewpoint of figure
string

Active viewpoint of a figure, returned as a string.

`Wireframe` — Wireframe display
`'off'` (default) | `'on'`

Wireframe display, returned as `'on'` or `'off'`.

`World` — World containing canvas
`vrworld` object

World containing canvas, represented by a `vrworld` object (read-only property).

`ZoomFactor` — Camera zoom factor
1 (default) | floating-point number

Camera zoom factor, returned as a floating-point number. A zoom factor of 2 makes the scene look twice as large. A zoom factor of `0.1` makes it look 10 times smaller, and so forth.

Methods

capture	Capture virtual reality canvas image

Examples

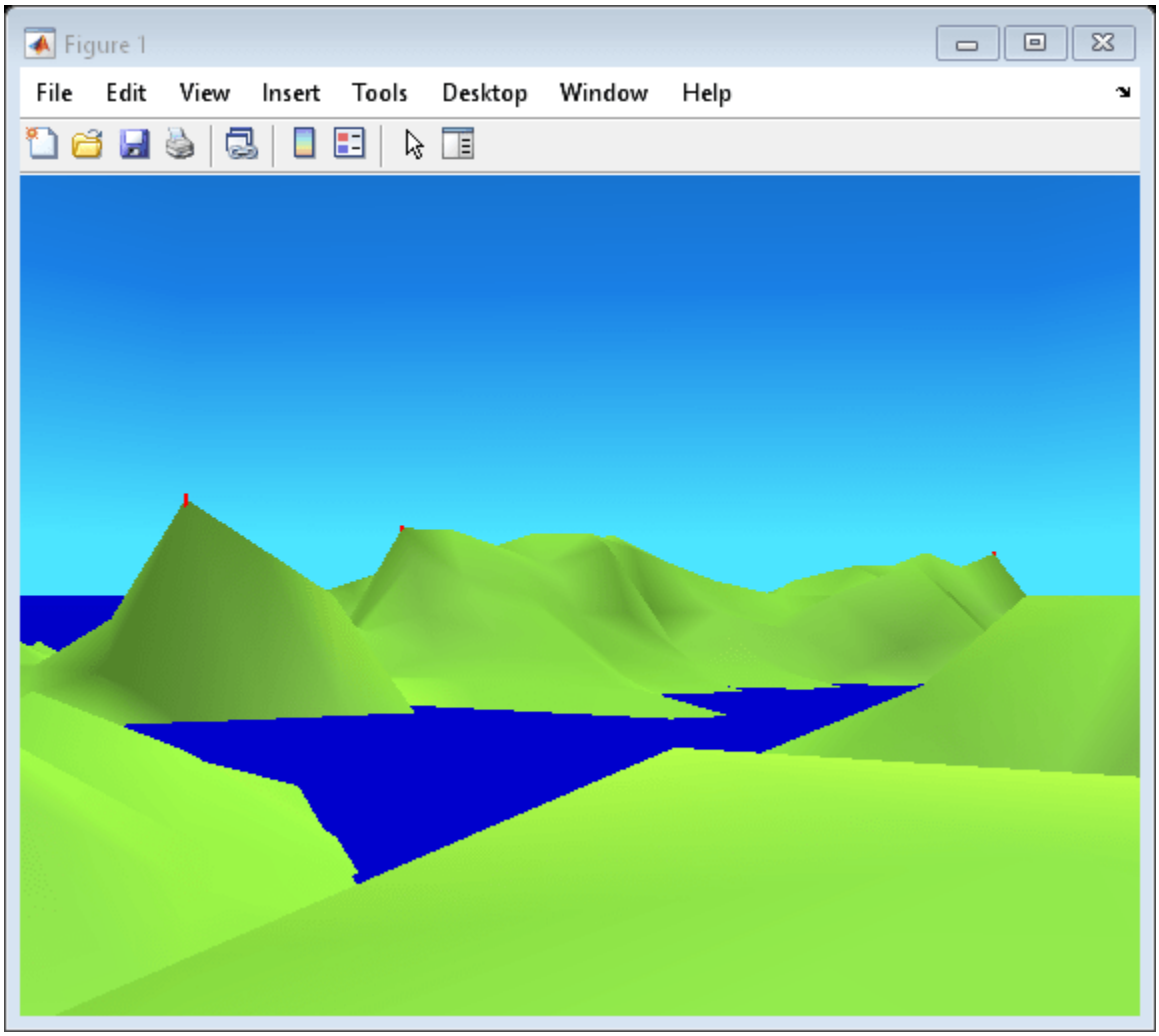
Create a Canvas That Displays in a Figure

Create and open a `vrworld` object.

```
myWorld = vrworld('vrlights');
open(myWorld);
```

Create a figure to use as the parent of the canvas. Create a canvas. Use a figure as the parent and specify the position.

```
fig = figure;
myCanvas = vr.canvas(myWorld,'Parent',fig,'Units',...
'normalized','Position',[0 0 1 1]);
```



Create a Virtual World in a Canvas

Create a figure. Create a canvas in the figure and specify a title.

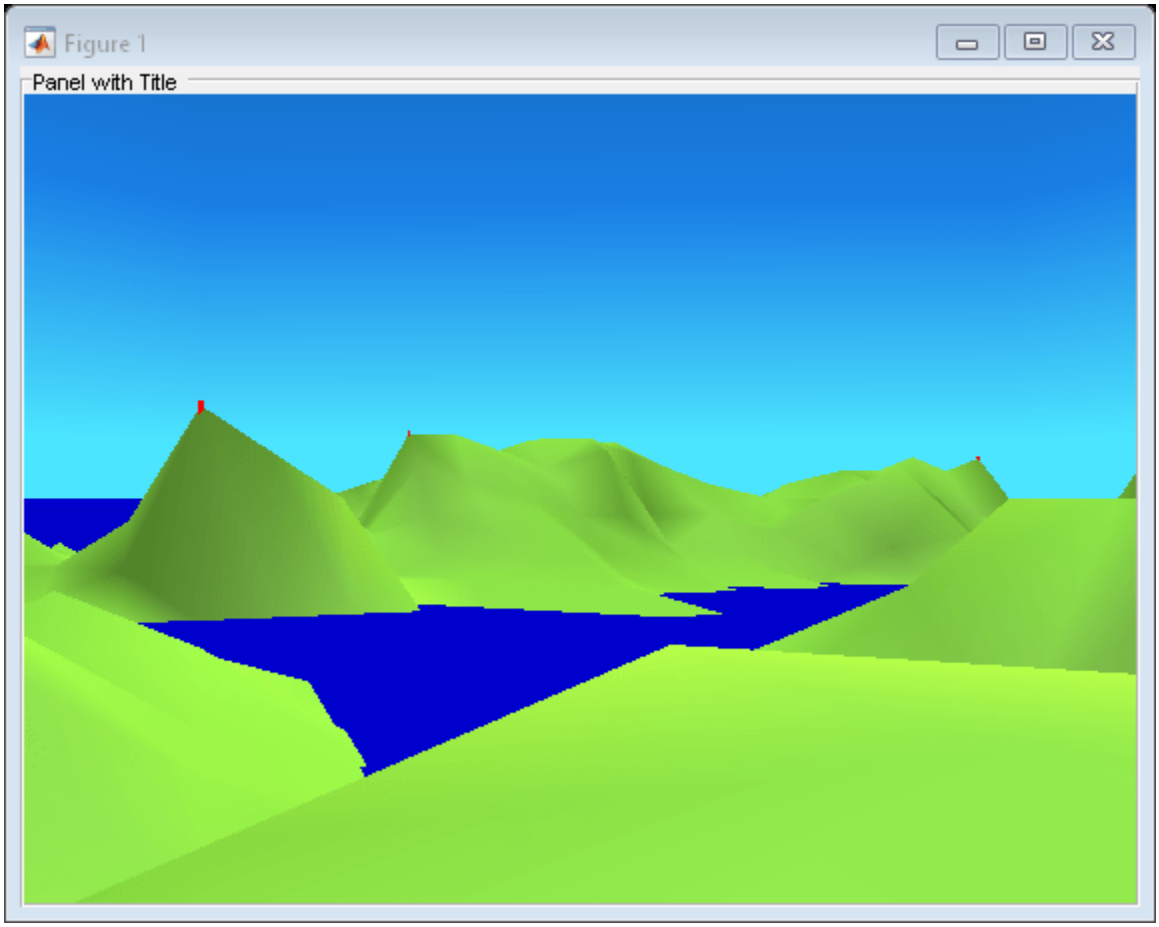
```
pf = figure;
pp1 = uipanel('Parent',pf,'Title','Panel with Title');
```

Create and open a virtual world.

```
w = vrworld('vrlights');
open(w);
```

Create a canvas in the virtual world.

```
c = vr.canvas(w,pp1);
```



Set Property Values of Canvas

Set the camera direction, navigation mode, and stereoscopic vision properties of a canvas.

Create and open a `vrworld` object.

```
vrmountWorld = vrworld('vrmount.x3d');
open(vrmountWorld);
```

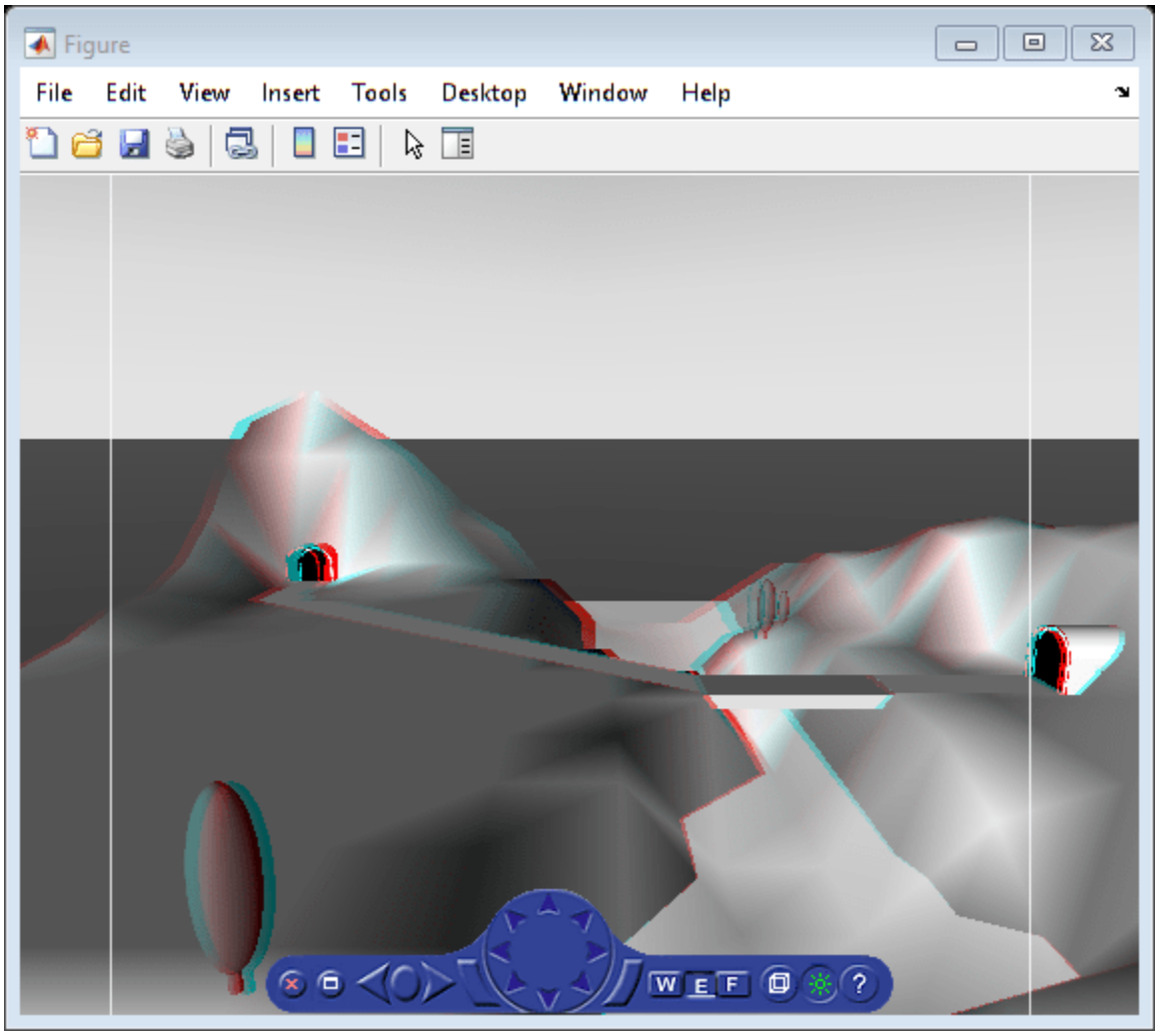
Create a `vr.utils.stereo3d` object to use to specify stereoscopic vision properties.

```
myStereo3D = vr.utils.stereo3d.ANAGLYPH_RED_CYAN;
```

Create a canvas. Define non-default values for some properties.

```
myCanvas = vr.canvas(vrmountWorld,'Antialiasing','on',...
           'NavPanel','opaque','NavZones','on','Stereo3D',...
           myStereo3D,'Stereo3DCameraOffset',0.25,...
           'Stereo3DHIT',0.02)


myCanvas =
```

```
canvas with properties:

          Antialiasing: 'on'
           CameraBound: 'on'
       CameraDirection: [0 0 -1]
        CameraPosition: [0 0 0]
        CameraUpVector: [0 1 0]
     ExaminePivotPoint: [0 0 0]
             Headlight: 'on'
          Highlighting: 'on'
              Lighting: 'on'
        MaxTextureSize: 'auto'
               NavPanel: 'opaque'
                NavMode: 'examine'
               NavSpeed: 'normal'
               NavZones: 'on'
               Position: [0 0 1 1]
              Rendering: 'on'
                  Sound: 'on'
               Stereo3D: 'anaglyph'
  Stereo3DCameraOffset: 0.2500
           Stereo3DHIT: 0.0200
              Textures: 'on'
              Tooltips: 'off'
          Transparency: 'on'
                 Triad: 'none'
                 Units: 'normalized'
             Viewpoint: 'View 1 - Observer'
             Wireframe: 'off'
            ZoomFactor: 1
            DeleteFcn: []
    CameraDirectionAbs: [0 -0.1987 -0.9801]
     CameraPositionAbs: [20 8 50]
     CameraUpVectorAbs: [0 0.9801 -0.1987]
                Parent: [1×1 Figure]
                 World: [1×1 vrworld]
```



Version History

Introduced before R2006a

See Also

`vr.utils.stereo3d` | `vrfigure` | `vrworld` | `figure`

Topics

“Create vrworld Object for a Virtual World”
“Interact with Virtual Reality Worlds”
“View a Virtual World in Stereoscopic Vision”

capture

Class: `vr.canvas`

Capture virtual reality canvas image

Syntax

```
image_capture = capture(canvas)
```

Description

`image_capture = capture(canvas)` captures a virtual reality canvas into a TrueColor RGB image. You can display this image using the `image` command.

Input Arguments

`canvas` — Virtual reality canvas

`vr.canvas` object

Virtual reality canvas, specified as a `vr.canvas` object.

Output Arguments

`image_capture` — Virtual reality canvas image

array

Virtual reality canvas image, captured as an array. The array is an m-by-n-by-3 data array that defines red, green, and blue color components for each individual pixel.

Examples

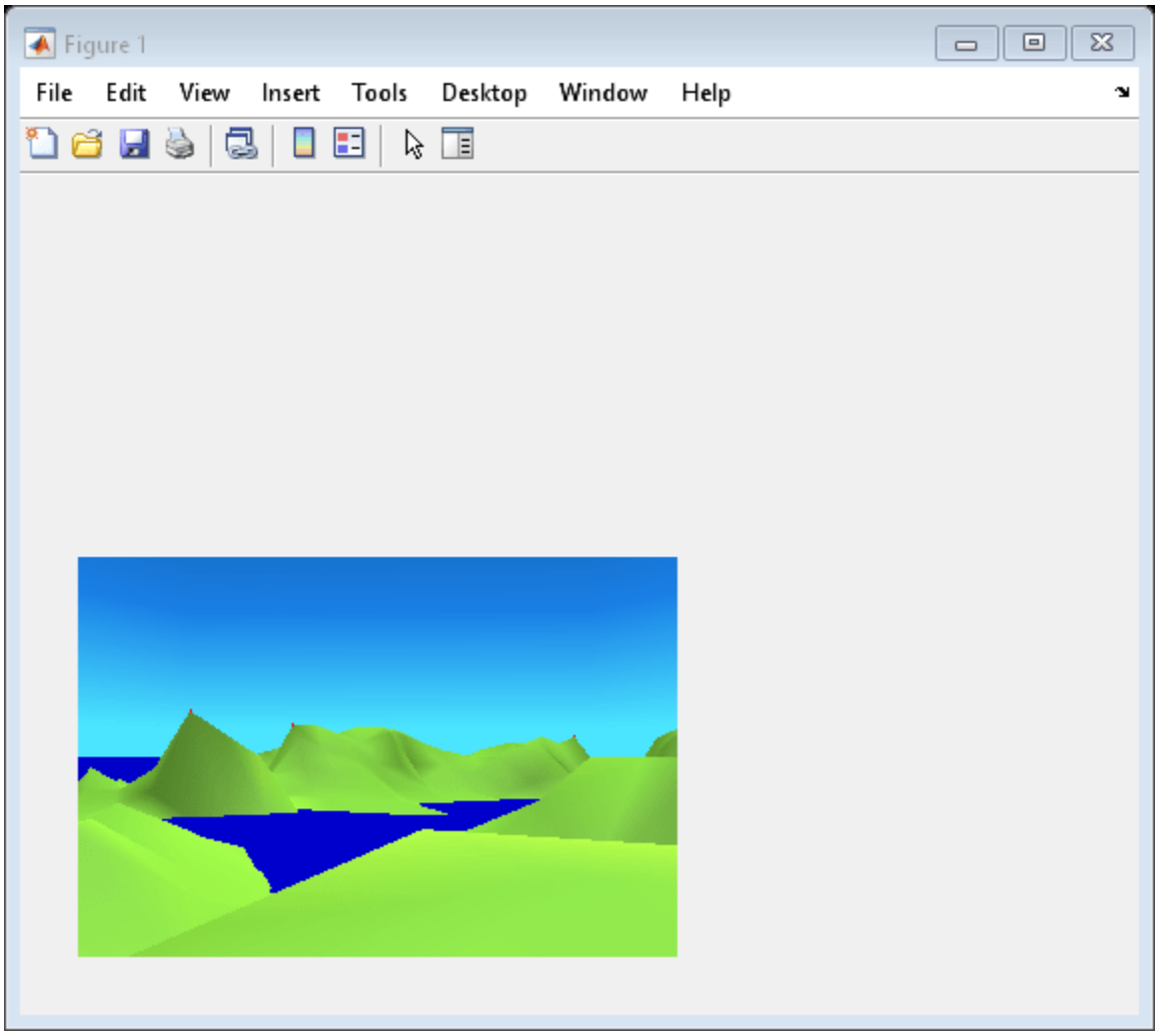
Capture an RGB Image of a Figure in a Canvas

Create and open a `vrworld` object and associate it with the virtual world `vrlights.x3d`.

```
lights_world = vrworld('vrlights');
open(lights_world);
```

Create a `vr.canvas` object for `lights_world`.

```
f = figure;
c = vr.canvas(lights_world, f, [30 30 300 200]);
```



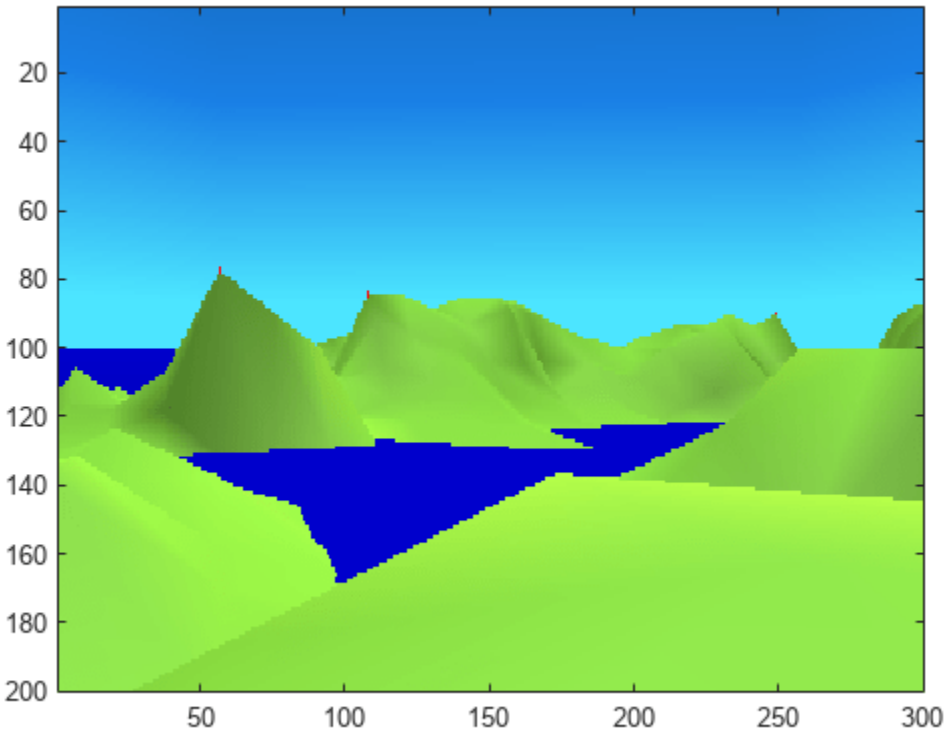

```
vrdrawnow;
```

Capture an image of the canvas.

```
image_capture = capture(c);
```

Display an RGB image of the canvas in a MATLAB® figure window.

```
figure;
image(image_capture);
```



Version History

Introduced before R2006a

See Also

`vrworld` | `image`

Topics

“Interact with Virtual Reality Worlds”
“Create vrworld Object for a Virtual World”

vrclear

Remove all closed virtual worlds from memory

Syntax

```
vrclear
```

```
vrclear('-force')
```

Description

The `vrclear` function removes from memory all virtual worlds that are closed and invalidates all `vrworld` objects related to them. This function does not affect open virtual worlds. Open virtual worlds include those loaded from the Simulink interface. You use this command to

- Ensure that the maximum amount of memory is freed before a memory-consuming operation takes place.
- Perform a general cleanup of memory.

The **`vrclear('-force')`** command removes all virtual worlds from memory, including worlds opened from the Simulink interface.

Version History

Introduced before R2006a

See Also

`vrworld` | `vrworld/delete` | “Close and Delete a vrworld Object”

vrclose

Close virtual reality figure windows

Syntax

```
vrclose
vrclose all
```

Description

`vrclose` and `vrclose all` close all the open virtual reality figures.

Examples

Open a series of virtual reality figure windows by typing

```
vrpend
vrbounce
vrlights
```

Arrange the viewer windows so they are all visible. Type

```
vrclose
```

All the virtual reality figure windows disappear from the screen.

Version History

Introduced before R2006a

See Also

`close` | “Close and Delete a vrworld Object”

vrcoordm2vr

Convert MATLAB coordinates to VR coordinates

Syntax

```
vr = vrcoordm2vr(m)
```

Description

`vr = vrcoordm2vr(m)` converts a point with coordinates in the MATLAB coordinate system to the Virtual World coordinate system.

Examples

Translate an object along a path

This example is a variation of the “Car in the Mountains” example, with the coordinates for translation specified in MATLAB coordinate system.

Create a `vrworld` object representing the virtual world and open it.

```
world = vrworld('vrmount');
open(world);
view(world);
```

Identify the nodes in the virtual world using the `nodes` command

```
nodes(world)
```

```
View1 (Viewpoint) [VR Car in the Mountains]
    Camera_car (Transform) [VR Car in the Mountains]
    VPfollow (Viewpoint) [VR Car in the Mountains]
    Automobile (Transform) [VR Car in the Mountains]
    Wheel (Shape) [VR Car in the Mountains]
    Tree1 (Group) [VR Car in the Mountains]
    Wood (Group) [VR Car in the Mountains]
    Canal (Shape) [VR Car in the Mountains]
    ElevApp (Appearance) [VR Car in the Mountains]
    River (Shape) [VR Car in the Mountains]
    Bridge (Shape) [VR Car in the Mountains]
    Road (Shape) [VR Car in the Mountains]
    Tunnel (Transform) [VR Car in the Mountains]
```

Access the `Automobile` `vrnode` object by assigning it to a handle

```
car = world.Automobile
```

```
car =

    vrnode object: 1-by-1

    Automobile (Transform) [VR Car in the Mountains]
```

Move the car along the first section of the road.

```
xz_my = zeros(12,3);

xz_my(:,2) = 1:12;
xz_my(:,1) = 3;
xz_my(:,3) = -0.25;

for idx = 1:length(xz_my)
   car.translation = vrcoordm2vr(xz_my(idx,:));
   vrdrawnow;
   pause(0.1);
end
```

Rotate the car a little to get to the second part of the road. This is done by setting the `rotation` property of the `Automobile` node.

```
car.rotation = [0 1 0 -0.7];
vrdrawnow;
```

Move the car through the second section of the road

```
z2 = 12:26;
x2 = 3:1.4285:23;
y2 = -0.25 + zeros(size(z2));
xz_my2 = [x2' z2' y2'];

for idx = 1:length(xz_my2)
   car.translation = vrcoordm2vr(xz_my2(idx,:));
   vrdrawnow;
   pause(0.1);
end
```

Rotate the car once again to face the third stretch of the road and continue to the end.

```
car.rotation = [0 1 0 0];
x3 = 23:43;
z3 = 26 + zeros(size(x3));
y3 = -0.25 + zeros(size(z3));
xz_my3 = [x3' z3' y3'];
for idx = 1:length(xz_my3)
   car.translation = vrcoordm2vr(xz_my3(idx,:));
   vrdrawnow;
   pause(0.1);
end
```

Input Arguments

`m` — Coordinates in MATLAB notation

3-element vector

Coordinates of a point in MATLAB notation, specified as a 3-element row vector.

Data Types: `single` | `double`

Output Arguments

`vr` — Coordinates in VRML notation

3-element vector

Coordinates of a point in VRML notation, returned as a 3-element row vector.

Data Types: `single` | `double`

Version History

Introduced in R2019a

See Also

`vrcoordvr2m` | MATLAB to VR Coordinates | VR to MATLAB Coordinates | VR Rotation to Rotation Matrix | Rotation Matrix to VR Rotation | `vrrotmat2vec` | `vrrotvec2mat`

Topics

“Virtual World Coordinate System”

vrcoordvr2m

Convert VR coordinates to MATLAB coordinates

Syntax

```
m = vrcoordvr2m(vr)
```

Description

`m = vrcoordvr2m(vr)` converts a point with coordinates in the Virtual World coordinate system to the MATLAB coordinate system.

Examples

Take-off Trajectory

This example creates a simple plane take-off trajectory in the virtual reality coordinate system. and plots it in MATLAB using `plot3`

The plane starts in the +x direction and goes up (positive values of y coordinate) in the z=0 plane. We then convert the trajectory to MATLAB coordinates, so that it can be displayed in a 3D figure using the MATLAB `plot3` command.

Define a simple take-off trajectory

```
vrpath = [0 0 0; 1 0 0; 2 0.2 0; 3 0.5 0; 4 1 0];
```

Convert the path from virtual reality coordinates to MATLAB coordinates

```
mpath = vrcoordvr2m(vrpath);
```

Display the path in a 3D plot using the plot3 command

```
plot3 (mpath(:,1), mpath(:,2), mpath(:,3))
```

Input Arguments

`vr` — Coordinates in VRML notation

3-element vector

Coordinates of a point in VRML notation, specified as a 3-element row vector.

Data Types: `double`

Output Arguments

`m` — Coordinates in MATLAB notation

3-element vector

Coordinates of a point in MATLAB notation, returned as a 3 element row vector.

Version History

Introduced in R2019a

See Also

`vrcoordm2vr` | MATLAB to VR Coordinates | VR to MATLAB Coordinates | VR Rotation to Rotation Matrix | Rotation Matrix to VR Rotation | `vrrotmat2vec` | `vrrotvec2mat`

Topics

“Virtual World Coordinate System”

vrdir2ori

Convert viewpoint direction to orientation

Syntax

```
vrdir2ori(d)
vrdir2ori(d,options)
```

Description

`vrdir2ori(d)` converts the viewpoint direction, specified by a vector of three elements, to an appropriate orientation (virtual world rotation vector).

`vrdir2ori(d,options)` converts the viewpoint direction with the default algorithm parameters replaced by values defined in `options`.

The `options` structure contains the parameter `epsilon` that represents the value below which a number will be treated as zero (default value is 1e-12).

Version History

Introduced in R2007b

See Also

`vrori2dir` on page 3-108 | `vrrotmat2vec` on page 3-120 | `vrrotvec` | `vrrotvec2mat` on page 3-121

vrdrawnow

Update virtual world

Syntax

```
vrdrawnow
```

Description

`vrdrawnow` removes from the queue pending changes to the virtual world and makes these changes to the scene in the viewer.

Changes to the scene are normally queued and the views are updated when

- The MATLAB software is idle for some time (no Simulink model is running and no script is being executed).
- A Simulink step is finished.

Version History

Introduced before R2006a

See Also

`vrworld/edit` | `vrworld/open` | “Open a Virtual World with MATLAB” | “Interact with a Virtual World with MATLAB”

vredit

Open 3D World Editor

Syntax

```
w = vredit
w = vredit(filename)
```

Description

`w = vredit` opens the 3D World Editor with an empty virtual world.

`w = vredit(filename)` opens a virtual world file in the 3D World Editor, based on the specified `filename`. It returns the `vrworld` handle of the virtual world.

To open a virtual world file in a third-party editor, do not use the `vredit` command. For example, to open a virtual world in the Ligos® V-Realm Builder editor:

1. Set the default editor to V-Realm Builder. In MATLAB, enter:

   ```
   vrsetpref('Editor','*VREALM');
   ```

2. To open a file in the V-Realm editor, in MATLAB navigate to a virtual world file, right-click, and select **Edit**.

Note The `vredit` command opens the 3D World Editor, regardless of the default editor preference setting.

Examples

Open New Virtual World in 3D World Editor

```
vredit
```

Open Existing Virtual World in 3D World Editor

Open the `membrane` virtual world in the 3D World Editor.

```
myworld = vredit('membrane.wrl')
```

Version History

Introduced in R2012b

See Also

`vrworld/edit` | `vrworld/open` | “Open a Virtual World with MATLAB” | “Interact with a Virtual World with MATLAB”

vrfigure class

Create virtual reality figure

Description

Creates a virtual reality figure.

To access `vrfigure` properties, use the `vrfigure/get` method. To change properties, use the `vrfigure/set` method.

If you create a `vrfigure` object by specifying a virtual world, the virtual figure displays in the viewer specified in the `vrsetpref` `DefaultViewer` property.

Construction

`virtual_figure = vrfigure(world)` creates a virtual reality figure showing the specified virtual world.

`virtual_figure = vrfigure(world,position)` creates a virtual reality figure at the specified position.

`virtual_figure = vrfigure([])` returns an empty `vrfigure` object that does not have a visual representation.

`virtual_figure = vrfigure` returns an empty vector of type `vrfigure`.

Input Arguments

`world` — Virtual world

`vrworld` object

Virtual world, specified as a `vrworld` object.

Note Open the virtual world that you specify before you create a `vrfigure` object using that virtual world.

`Position` — Figure location and size

`vector with four elements`

Location and size of virtual figure, specified as the vector in the form `[left bottom width height]`. Specify measurements in pixels.

Note On Windows systems, figure windows cannot be less than 104 pixels wide, regardless of the value of the `Position` property.

Element	Description
`left`	Distance from the left edge of the primary display to the inner left edge of the figure window. This value can be negative on systems that have more than one monitor.
`bottom`	Distance from the bottom edge of the primary display to the inner bottom edge of the figure window. This value can be negative on systems that have more than one monitor.
`width`	Distance between the right and left inner edges of the figure.
`height`	Distance between the top and bottom inner edges of the figure.

Example: `[230 250 570 510]`

Data Types: `double`

Output Arguments

`virtual_figure` — Virtual reality figure
`vrfigure` object | empty vector of type `vrfigure`

If you use a `vrworld` object as an input argument, `virtual_figure` is a virtual reality figure, represented by a `vrfigure` object.

If you use an empty array as an input argument, the `vrfigure` constructor returns a vector of type `vrfigure`.

If you do not use an input argument, the `vrfigure` constructor returns an empty vector of type `vrfigure`.

Methods

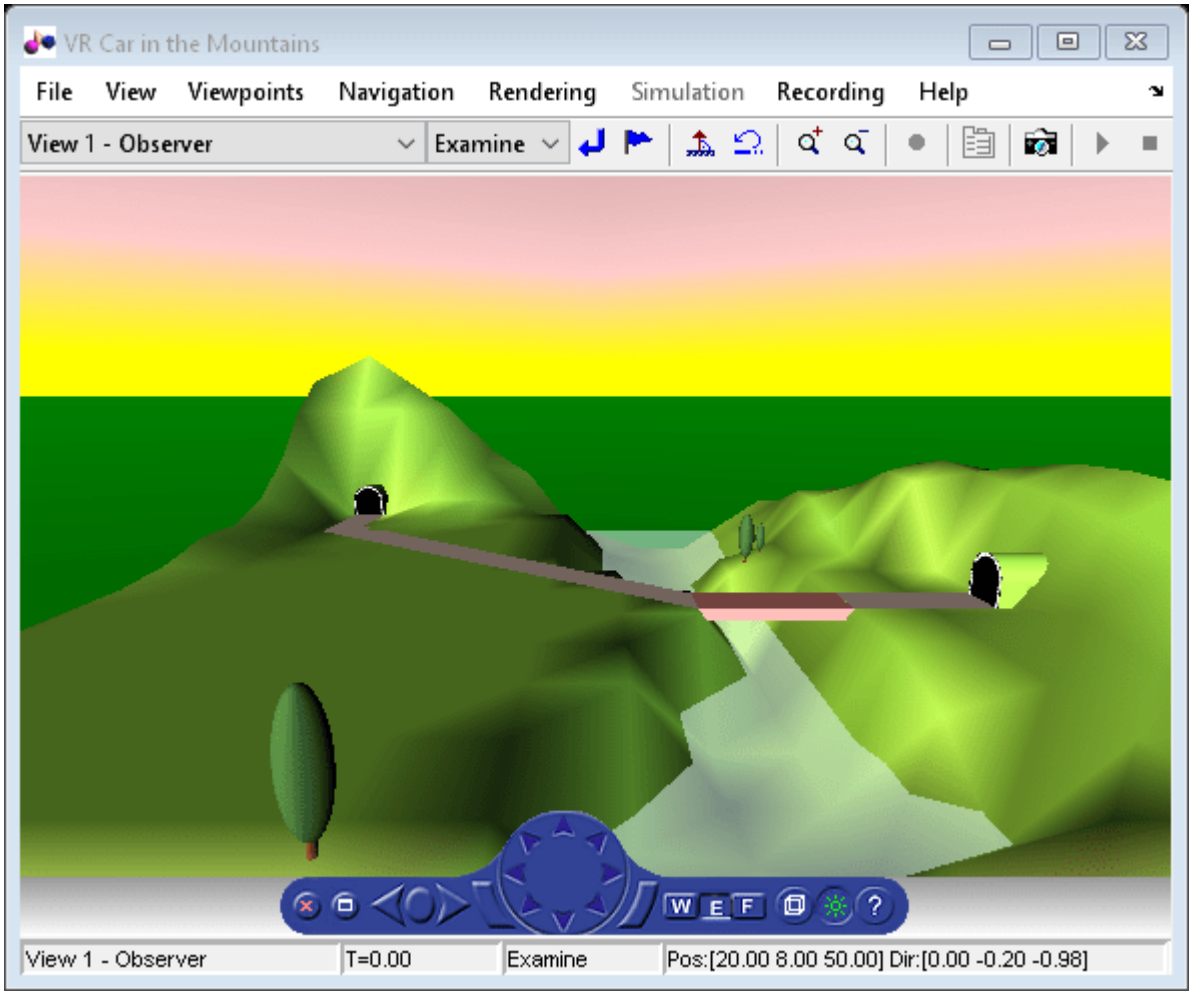
capture	Capture virtual reality figure image
capture	Capture virtual reality figure image
close	Close virtual reality figure
get	Return property value of `vrfigure` object
isvalid	Check validity of `vrfigure` object handles
set	Set property values of `vrfigure` object
set	Set property values of `vrfigure` object

Examples

Create and Display a vrworld Object

Create a `vrworld` object that is associated with the virtual world `vrmount.wrl`. Open and view the virtual world.

```
myworld = vrworld('vrmount');
open(myworld);
f = vrfigure(myworld);
```



Version History

Introduced before R2006a

See Also

`vr.utils.stereo3d` | `vr.canvas`

Topics

“Create vrworld Object for a Virtual World”
“Interact with Virtual Reality Worlds”
“View a Virtual World in Stereoscopic Vision”

capture

Class: `vrfigure`

Capture virtual reality figure image

Syntax

```
image_capture = capture(figure)
```

Description

`image_capture = capture(figure)` captures a virtual reality figure into a TrueColor RGB image. You can display this image using the `image` command. You can then print the figure.

Input Arguments

`figure` — Virtual reality figure

`vrfigure` object

Virtual reality figure, specified as a `vrfigure` object.

Output Arguments

`image_capture` — Virtual reality figure image

array

Virtual reality figure image, captured as an array. The array is an m-by-n-by-3 data array that defines red, green, and blue color components for each individual pixel.

Examples

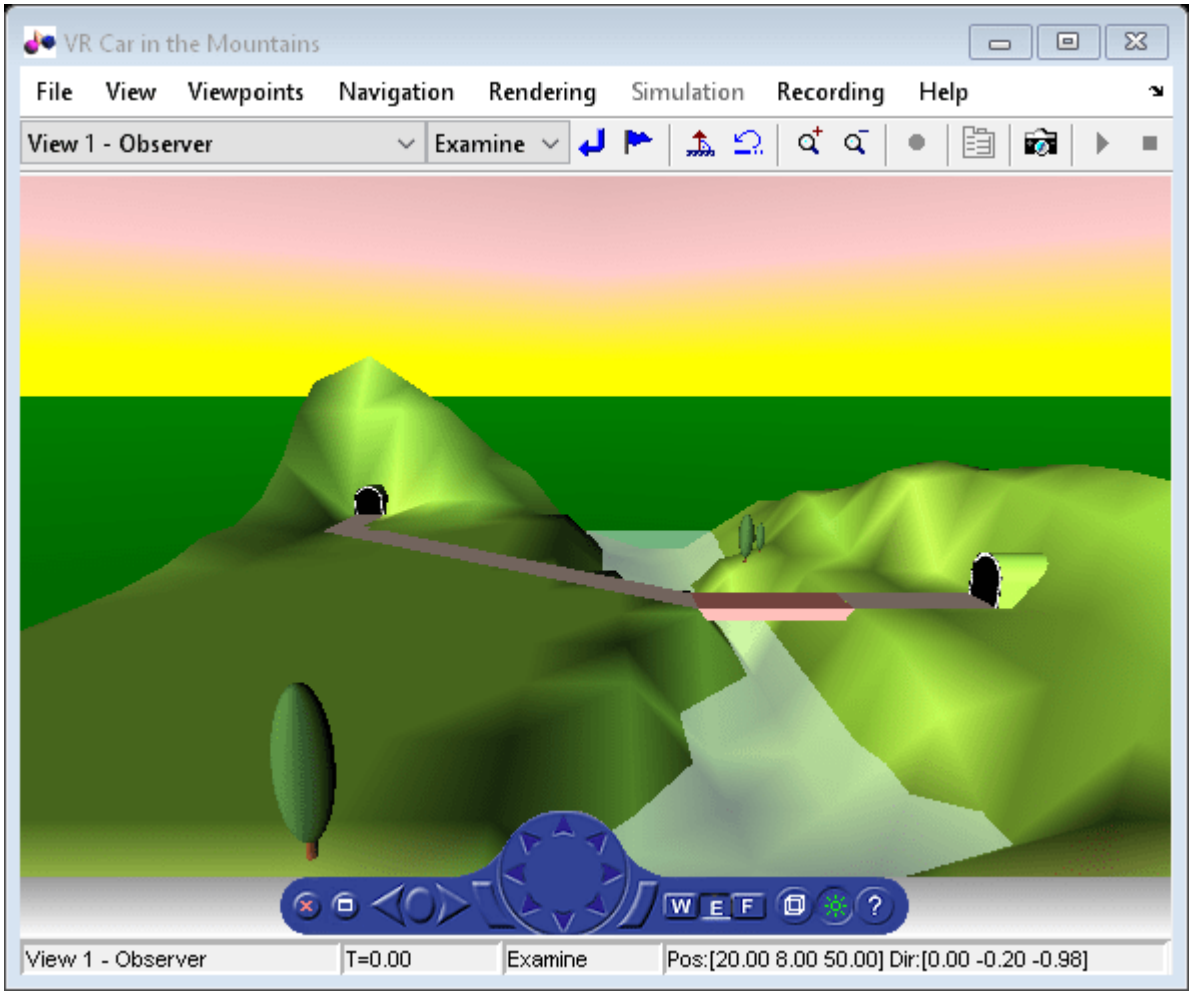
Capture an RGB Image of a Figure

Create and open a `vrworld` object and associate it with the virtual world `vrmount.x3d`.

```
myworld = vrworld('vrmount');
open(myworld);
```

View the virtual world in the Simulink® 3D Animation™ Viewer.

```
f = vrfigure(myworld);
```

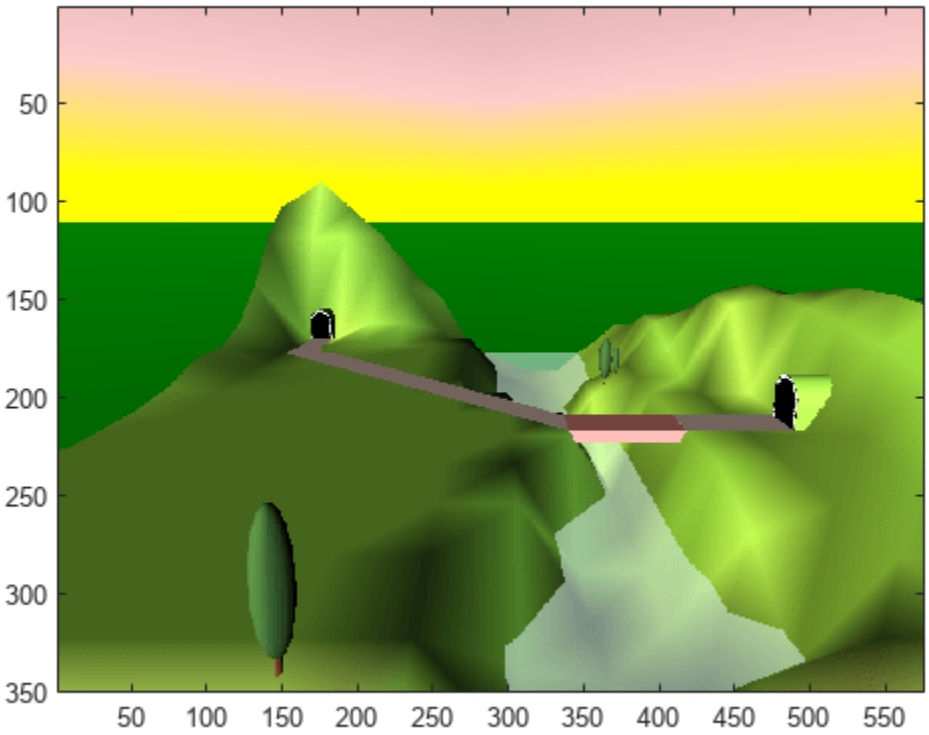


Create an RGB image of the figure.

```
image_capture = capture(f);
```

Display the RGB figure image in a MATLAB® figure window.

```
image(image_capture);
```



Version History

Introduced before R2006a

See Also

`vrfigure` | `vrworld` | `image` | `isvalid` | `vrnode/isvalid`

Topics

“Interact with Virtual Reality Worlds”
“Create vrworld Object for a Virtual World”

close

Class: `vrfigure`

Close virtual reality figure

Syntax

```
close(figure)
```

Description

`close(figure)` closes the virtual reality figure referenced by `figure`. If `figure` is a vector of `vrfigure` object handles, then the method closes multiple figures.

Input Arguments

`figure` — Virtual reality figure

`vrfigure` object

Virtual reality figure, specified as a `vrfigure` object.

Examples

Create a Figure

```
myworld = vrworld('vrpend');
open(myworld);
f = vrfigure(myworld);
close(f)
```

Version History

Introduced before R2006a

See Also

`vrfigure` | `vrworld`

Topics

“Interact with Virtual Reality Worlds”
“Create vrworld Object for a Virtual World”

get

Class: `vrfigure`

Return property value of `vrfigure` object

Syntax

```
get(figure)
figureProp = get(figure,propertyName)
```

Description

`get(figure)` lists the values of all the properties of the `vrfigure` object.

`figureProp = get(figure,propertyName)` returns the value of the specified property of the `vrfigure` object.

Input Arguments

`figure` — Virtual reality figure

`vrfigure` object

Virtual reality figure, specified as a `vrfigure` object.

`property_name` — Virtual reality figure object property

string

Virtual reality figure property, specified as one of these.

vrfigure Property	Meaning
`Antialiasing`	Smooth textures using antialiasing, which interpolates values between texture points.
`CameraBound`	Camera movement with the current viewpoint.
`CameraDirection`	Camera direction in the current viewpoint local coordinates.
`CameraDirectionAbs`	Camera direction in the world coordinates. (read-only property).
`CameraPosition`	Camera position in the current viewpoint local coordinates.
`CameraPositionAbs`	Camera position in world coordinates (read-only property).
`CameraUpVector`	Camera up vector.
`CameraUpVectorAbs`	Camera up vector in world coordinates (read-only property).
`CaptureFileFormat`	File format for a captured frame file.

vrfigure Property	Meaning
`CaptureFileName`	Frame capture file name.
`DeleteFcn`	Callback invoked when closing the `vrfigure` object.
`ExaminePivotPoint`	Pivot point about which camera is rotated in `examine` navigation mode, in world coordinates.
`Fullscreen`	Full screen display of figure.
`Headlight`	Headlight from camera.
`Lighting`	Lighting effect.
`MaxTextureSize`	Maximum pixel size of a texture used. The smaller the size, the faster the texture can render. A value of `'auto'` means the texture is set to the maximum pixel size.
`Name`	Name of figure.
`NavMode`	Navigation mode. See “Mouse Navigation”.
`NavPanel`	Navigation panel appearance.
`NavSpeed`	Navigation speed.
`NavZones`	Navigation zones display.
`Position`	Screen coordinates of figure.
`Record2D`	2-D offline animation file recording.
`Record2DCompress Method`	Compression method for creating 2-D animation files. See `profile` in the MATLAB `VideoWriter` documentation.
`Record2DCompress Quality`	Quality of 2-D animation file compression. See the MATLAB `VideoWriter` documentation.
`Record2DFileName`	Name of 2-D offline animation file. The string can contain tokens that animation recording replaces with information. See “File Name Tokens”.
`Record2DFPS`	Rate of playback for the 2-D offline animation video in frames per second (fps).
`Rendering`	Specifies whether to render a `vrfigure` object. Turning off rendering improves performance. For example, if your code does batch operations on a virtual figure, you can turn off rendering during that processing and then turn it back on after the processing.
`Sound`	Sound effects.
`StatusBar`	Status bar display.
`Stereo3D`	Stereoscopic vision mode.
`Stereo3DCameraOffset`	Distance in virtual world units of left and right camera from parallax for stereoscopic vision. Parallax is the difference in the apparent position of an object viewed from two cameras.

vrfigure Property	Meaning
`Stereo3DHIT`	Horizontal image translation (HIT) of the two stereo images in stereoscopic vision, represented by a value from 0 to 1, inclusive. The larger the value, the further back the background appears.
`Textures`	Texture use.
`ToolBar`	Toolbar display.
`Tooltips`	Tooltips display in navigation panel.
`Transparency`	Transparency effect.
`Triad`	Location of the triad.
`Viewpoint`	Active viewpoint of figure.
`Wireframe`	Wireframe display.
`World`	Virtual world that the figure displays (read-only property).
`ZoomFactor`	Camera zoom factor.

Output Arguments

`figureProp` — Virtual reality figure property

string | vector

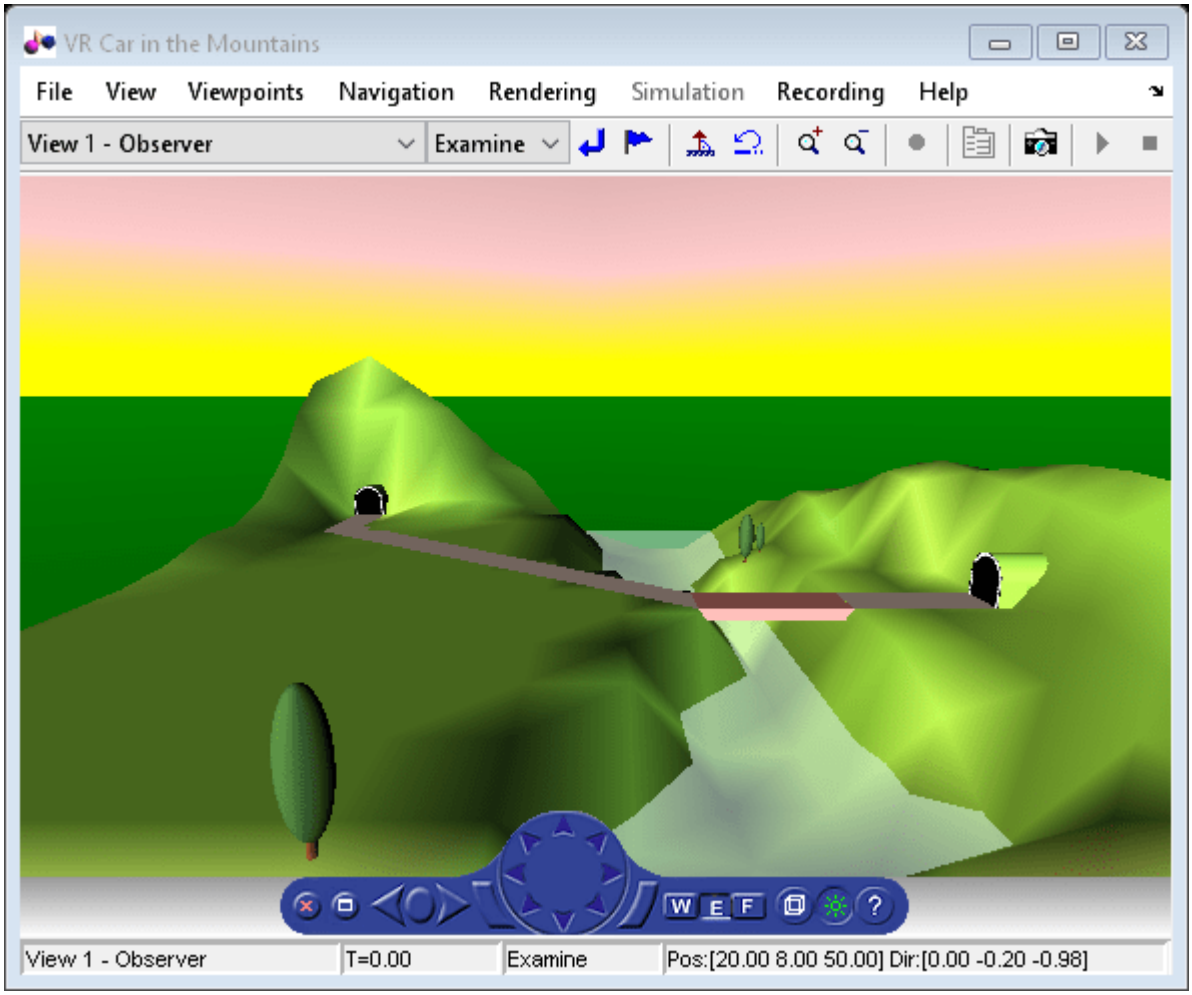
Virtual reality figure property, returned as a string or vector.

Examples

Return All Property Values of a Figure

Create a `vrfigure` object.

```
myworld = vrworld('vrmount');
open(myworld);
virtual_fig = vrfigure(myworld);
```



Return the properties of the virtual figure `virtual_fig`.

```
get(virtual_fig)
```

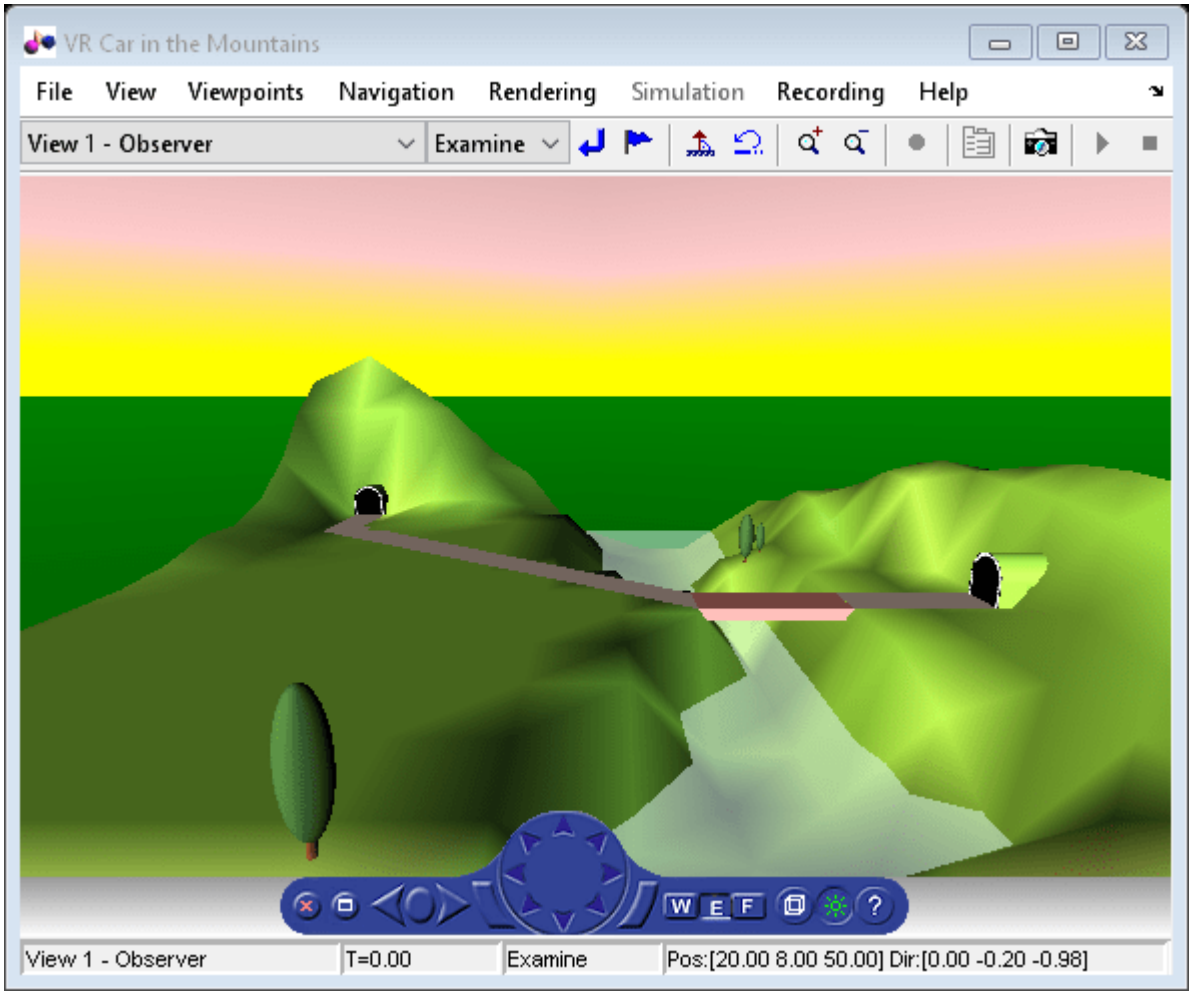
```
    Antialiasing = 'on'
    CameraBound = 'on'
    CameraDirection = [0 0 -1]
    CameraDirectionAbs = [0 -0.198669 -0.980067]
    CameraPosition = [0 0 0]
    CameraPositionAbs = [20 8 50]
    CameraUpVector = [0 1 0]
    CameraUpVectorAbs = [0 0.980067 -0.198669]
    CaptureFileFormat = 'tif'
    CaptureFileName = '%f_anim_%n.tif'
    DeleteFcn = ''
    ExaminePivotPoint = [0 0 0]
    Fullscreen = 'off'
    Headlight = 'on'
    Lighting = 'on'
    MaxTextureSize = 'auto'
    Name = 'VR Car in the Mountains'
    NavMode = 'examine'
```

```
    NavPanel = 'halfbar'
    NavSpeed = 'normal'
    NavZones = 'off'
    Position = [5 92 576 350]
    Record2D = 'off'
    Record2DCompressMethod = 'auto'
    Record2DCompressQuality = 75
    Record2DFPS = 'auto'
    Record2DFileName = '%f_anim_%n.avi'
    Rendering = 'on'
    Sound = 'on'
    StatusBar = 'on'
    Stereo3D = 'off'
    Stereo3DCameraOffset = 0.1
    Stereo3DHIT = 0
    Textures = 'on'
    ToolBar = 'on'
    Tooltips = 'on'
    Transparency = 'on'
    Triad = 'none'
    Viewpoint = 'View 1 - Observer'
    Wireframe = 'off'
    World = vrworld object: 1-by-1
    ZoomFactor = 1
```

Return Name of a Figure

Create a `vrfigure` object.

```
myworld = vrworld('vrmount');
open(myworld);
virtual_fig = vrfigure(myworld);
```



Return the properties of the virtual figure `virtual_fig`.

```
figure_name = get(virtual_fig,'Name')
```

```
figure_name =
'VR Car in the Mountains'
```

Version History

Introduced before R2006a

See Also

`vrfigure` | `vr.utils.stereo3d`

Topics

“Interact with Virtual Reality Worlds”
“Create vrworld Object for a Virtual World”
“View a Virtual World in Stereoscopic Vision”

isvalid

Class: `vrfigure`

Check validity of `vrfigure` object handles

Syntax

```
valid_handles = isvalid(vrfigure_vector)
```

Description

`valid_handles = isvalid(vrfigure_vector)` detects whether the `vrfigure` handles are valid.

Input Arguments

`figure_vector` — Virtual reality figure vector

array of `vrfigure` object handles

Virtual reality figure vector, specified as a `vrfigure` object.

Output Arguments

`valid_handles` — Valid `vrfigure` object handles

logical array

Virtual reality figure image, captured as a logical array. The array that contains a `1` where the `vrfigure` handles are valid and returns a `0` where they are not.

Examples

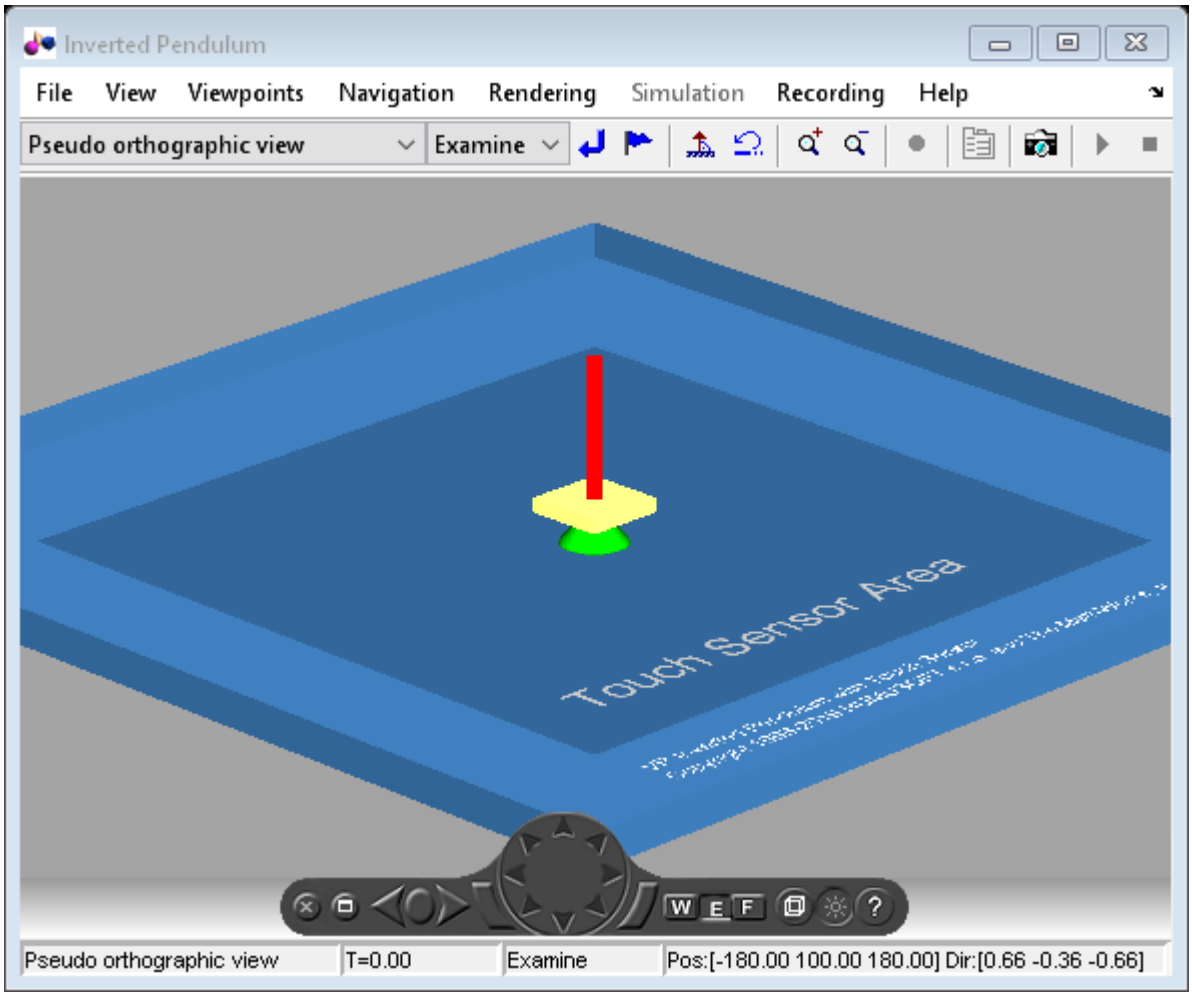
Check Validity of Figure Handles

Check whether the figure handles of the `vrfigure` object are valid. The first check shows that the figure handle is valid, but the second check shows that the handle is invalid because the figure is closed.

```
myworld = vrview('vrpend');
f = vrfigure(myworld);
firstCheck = isvalid(f)
```

```
firstCheck = logical
   1
```

```
close(f)
```




```
secondCheck = isvalid(f)
```

```
secondCheck = logical
   0
```

Version History

Introduced before R2006a

See Also

`vrfigure` | `vrworld`

Topics

“Interact with Virtual Reality Worlds”
“Create vrworld Object for a Virtual World”

set

Class: `vrfigure`

Set property values of `vrfigure` object

Syntax

```
set(figure,PropertyName,Value,...,PropertyName,Value)
```

Description

`set(figure,PropertyName,Value,...,PropertyName,Value)` sets the values of the `vrfigure` properties specified by one or more `PropertyName,Value` pair arguments.

Input Arguments

`figure` — Virtual reality figure

`vrfigure` object

Virtual reality figure, specified as a `vrfigure` object.

PropertyName-Value Pair Arguments

Specify comma-separated pairs of `PropertyName,Value` arguments. `PropertyName` is the argument name and `Value` is the corresponding value. `PropertyName` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `PropertyName1,Value1,...,PropertyNameN,ValueN`.

Example: `set(myFigure,'Antialiasing','on','CameraPosition',[0 100 100])`

`Antialiasing` — Smooth textures using antialiasing

`'off'` (default) | `'on'`

Smooth textures using antialiasing, specified as `'on'` or `'off'`. Antialiasing smooths textures by interpolating values between texture points.

`CameraBound` — Camera movement with current viewpoint

`'on'` (default) | `'off'`

Camera movement with the current viewpoint, specified as `'on'` or `'off'`.

`CameraDirection` — Camera direction in the current viewpoint local coordinates

vector of three doubles

Camera direction in the current viewpoint local coordinates, specified as a vector of three doubles. The doubles represent the x, y, and z vectors in the current viewpoint local coordinates.

`CameraPosition` — Camera position in the current viewpoint local coordinates

vector of three doubles

Camera position in the current viewpoint local coordinates, specified as a vector of three doubles. The doubles represent the x, y, and z vectors in the current viewpoint local coordinates.

`CameraUpVector` — Camera up vector
vector of three doubles

Camera up vector, specified as a vector of three doubles. The doubles represent the x, y, and z vectors in the current viewpoint local coordinates.

`CaptureFileFormat` — File format for captured frame file
`'tif'` (default) | `'png'`

File format for a captured frame file, specified as `'tif'` for Tagged Image Format or `'png'` for Portable Network Graphics format.

`CaptureFileName` — Frame capture file name
`'%f_anim_%n.ext'` (default) | string

Frame capture file name, specified as a string. The string can contain tokens that the frame capture replaces with the corresponding information. See “Define File Name Tokens”.

`DeleteFcn` — Callback invoked when closing `vrfigure` object
string

Callback invoked when closing the `vrfigure` object, specified as a string.

`Fullscreen` — Fullscreen display of figure
`'off'` (default) | `'on'`

Fullscreen display of figure, specified as `'on'` or `'off'`.

`Headlight` — Headlight from camera
`'on'` (default) | `'off'`

Headlight from camera, specified as `'on'` or `'off'`. If you specify `'off'`, the camera does not emit light and the scene can appear dark.

`Lighting` — Lighting effect
`'on'` (default) | `'off'`

Lighting effect, specified as `'on'` or `'off'`. If you specify `'off'`, the camera does not emit light and the scene can appear dark.

`MaxTextureSize` — Maximum pixel size of textures
`'auto'` (default) | integer in a power of 2

Maximum pixel size of textures, specified as `'auto'` or integer in a power of 2. The value of `'auto'` sets the maximum texture pixel size. Otherwise, specify an integer in a power of two that is equal to or less than the video card limit (typically `1024` or `2048`).

The smaller the size, the faster the texture renders. Increasing the size improves image quality but decreases performance.

Note Specifying a value that is unsuitable causes a warning. The Simulink 3D Animation software then adjusts the property to the next smaller suitable value.

Data Types: `int32`

`Name` — Name of figure
string

Name of figure, specified as a string.

`NavMode` — Navigation mode
`'examine'` (default) | `'fly'` | `'walk'` | `'none'`

Navigation mode, specified as `'examine'`, `'fly'`, `'walk'`, or `'none'`. See “Mouse Navigation”.

`NavPanel` — Navigation panel appearance
`'none'` (default) | `'halfbar'` | `'bar'` | `'opaque'` | `'translucent'`

Navigation panel appearance, specified as `'none'`, `'halfbar'`, `'bar'`, `'opaque'`, or `'translucent'`.

`Navspeed` — Navigation speed
`'normal'` (default) | `'slow'` | `'veryslow'` | `'fast'` | `'veryfast'`

Navigation speed, specified as `'normal'`, `'slow'`, `'veryslow'`, `'fast'`, or `'veryfast'`.

`NavZones` — Navigation zones display
`'off'` (default) | `'on'`

Navigation zones display, specified as `'on'` or `'off'`.

`Position` — Figure location and size
`vector with four doubles`

Location and size of virtual figure, specified as the vector in the form `[left bottom width height]`. Specify measurements in pixels.

Element	Description
`left`	Distance from the left edge of the primary display to the inner left edge of the figure window. You can specify a negative value on systems that have more than one monitor.
`bottom`	Distance from the bottom edge of the primary display to the inner bottom edge of the figure window. You can specify a negative value on systems that have more than one monitor.
`width`	Distance between the right and left inner edges of the figure.
`height`	Distance between the top and bottom inner edges of the figure.

All measurements are in units specified in pixels.

Example: `[230 250 570 510]`

Data Types: `double`

`Record2D` — 2-D offline animation file recording
`'off'` (default) | `'on'`

2-D offline animation file recording, specified as `'on'` or `'off'`.

`Record2DCompressMethod` — Compression method for creating 2-D animation files
`'auto'` (default) | `''` | `'lossless'` | `'none'` | string with name of a compression method

Compression method for creating 2-D animation files, specified as `''`, `'lossless'`, `'none'`, or a string specifying the name of a compression method. See `profile` in the MATLAB `VideoWriter` documentation.

`Record2DCompressQuality` — Quality of 2-D animation file compression
75 (default) | floating point number from 0 through 100, inclusive

Quality of 2-D animation file compression, specified as a floating-point number from 0 through 100, inclusive. See the MATLAB `VideoWriter` documentation.

Data Types: `int32`

`Record2DFileName` — Name of 2-D offline animation file
string

Name of 2-D offline animation file, specified as a string. The string can contain tokens that animation recording replaces with the corresponding information. See “File Name Tokens”.

`Record2DFPS` — Playback rate for 2-D offline animation file
`'auto'` (default) | scalar

Playback rate for 2-D offline animation file, specified as `'auto'` or as a scalar. The `'auto'` setting aligns simulation time with actual time and uses an appropriate frame rate.

Data Types: `int32`

`Rendering` — Render `vrfigure` object in Simulink 3D Animation Viewer
`'on'` (default) | `'off'`

Render `vrfigure` object in the Simulink 3D Animation Viewer, by specifying `'on'` or `'off'`. Turning off rendering improves performance. For example, if your code does batch operations on a virtual figure, you can turn off rendering during that processing and then turn it back on after the processing.

`Sound` — Sound effects
`'on'` (default) | `'off'`

Sound effects, specified as `'on'` or `'off'`.

`StatusBar` — Status bar display
`'on'` (default) | `'off'`

Status bar display, specified as `'on'` or `'off'`.

`Stereo3D` — Stereoscopic vision mode
`'off'` (default) | `'anaglyph'` | `'active'` | `vr.utils.stereo3d` object

Stereoscopic vision mode, specified as `'off'`, `'anaglyph'`, `'active'` or a `vr.utils.stereo3d` object.

Specifying a `vr.utils.stereo3d` object sets the `Stereo3D`, `Stereo3DCamaraOffset`, and `Stereo3DHIT` properties. Specifying a `vr.utils.stereo3d` object also sets color filters for the left and right cameras.

`Stereo3DCameraOffset` — Distance of left and right camera for stereoscopic vision
vector of three doubles

Distance of left and right camera from parallax for stereoscopic vision, specified as a vector of three doubles representing virtual world units or as a `vr.utils.stereo3d` object.

Specifying a `vr.utils.stereo3d` object sets the `Stereo3D`, `Stereo3DCamaraOffset`, and `Stereo3DHIT` properties. Specifying a `vr.utils.stereo3d` object also sets color filters for the left and right cameras.

`Stereo3DHIT` — Horizontal image translation (HIT) of two stereoscopic images
double from 0 to 1

Horizontal image translation (HIT) of two stereoscopic images, specified as a double from 0 through 1, inclusive. The larger the value, the further back the background appears. By default, the background image is at zero and the foreground image appears to pop out from the monitor toward the person viewing the virtual world.

Specifying a `vr.utils.stereo3d` object sets the `Stereo3D`, `Stereo3DCamaraOffset`, and `Stereo3DHIT` properties. Specifying a `vr.utils.stereo3d` object also sets color filters for the left and right cameras.

`Textures` — Texture use
`'on'` (default) | `'off'`

Texture use, specified as `'on'` or `'off'`.

`Toolbar` — Toolbar display
`'on'` (default) | `'off'`

Toolbar display, specified as `'on'` or `'off'`.

`Tooltips` — Tooltips display
`'on'` (default) | `'off'`

Tooltips display, specified as `'on'` or `'off'`.

`Transparency` — Transparency effect
`'on'` (default) | `'off'`

Transparency effect, specified as `'on'` or `'off'`.

`Viewpoint` — Active viewpoint of figure
string

Active viewpoint of a figure, specified as a string. If the active viewpoint has no description, use an empty string.

`Wireframe` — Wireframe display
`'off'` (default) | `'on'`

Wireframe display, specified as `'on'` or `'off'`.

`ZoomFactor` — Camera zoom factor
1 (default) | floating-point number

Camera zoom factor, specified as a floating-point number. A zoom factor of 2 makes the scene look twice as large. A zoom factor of `0.1` makes it look 10 times smaller, and so forth.

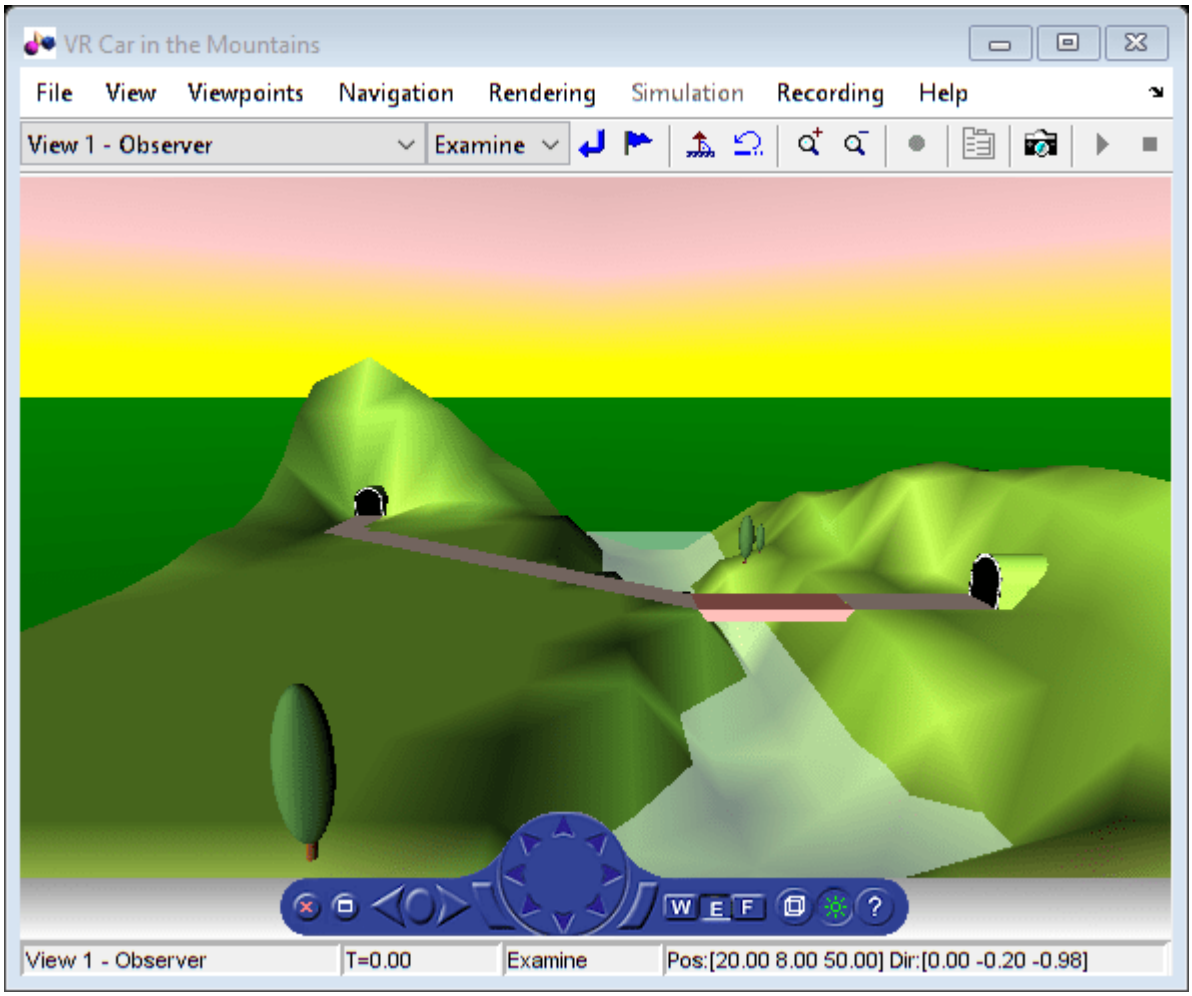
Examples

Set Property Values of Figure

Set the camera direction, navigation mode, and stereoscopic vision properties of a virtual figure.

Create a `vrfigure` object.

```
myworld = vrworld('vrmount.x3d');
open(myworld);
virtual_fig = vrfigure(myworld);
```

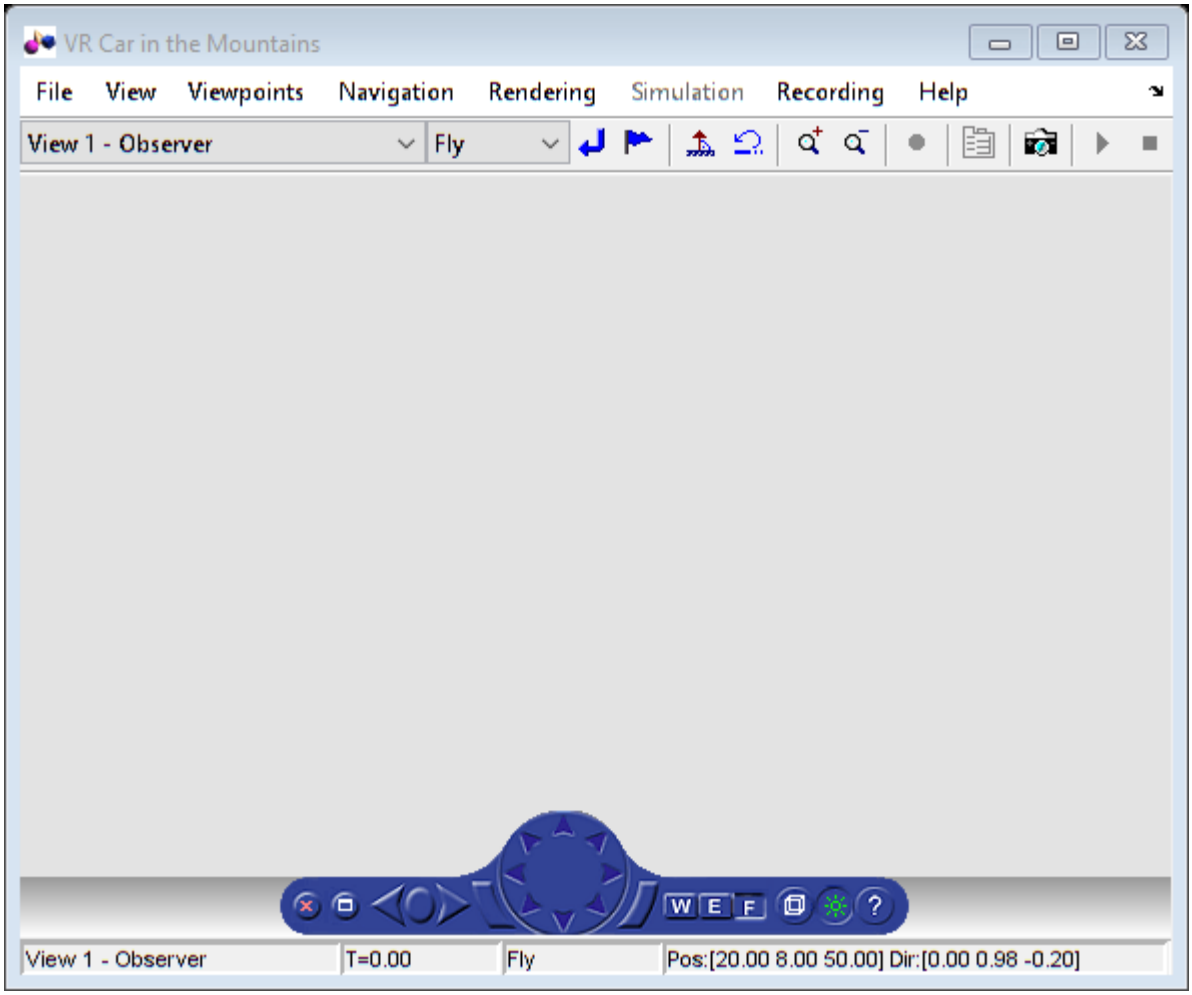


Create a `vr.utils.stereo3d` object to use to specify stereoscopic vision properties.

```
myStereo3D = vr.utils.stereo3d.ANAGLYPH_RED_CYAN;
```

Set the properties for a figure.

```
set(virtual_fig,'CameraDirection',[0 1 0],'NavMode','fly',...
    'Stereo3D',myStereo3D);
```



View the figure properties.

```
get(virtual_fig)
```

```
    Antialiasing = 'on'
    CameraBound = 'on'
    CameraDirection = [0 1 0]
    CameraDirectionAbs = [0 0.980067 -0.198669]
    CameraPosition = [0 0 0]
    CameraPositionAbs = [20 8 50]
    CameraUpVector = [0 1 0]
    CameraUpVectorAbs = [0 0.980067 -0.198669]
    CaptureFileFormat = 'tif'
    CaptureFileName = '%f_anim_%n.tif'
    DeleteFcn = ''
    ExaminePivotPoint = [0 0 0]
    Fullscreen = 'off'
    Headlight = 'on'
    Lighting = 'on'
```

```
MaxTextureSize = 'auto'
Name = 'VR Car in the Mountains'
NavMode = 'fly'
NavPanel = 'halfbar'
NavSpeed = 'normal'
NavZones = 'off'
Position = [5 92 576 350]
Record2D = 'off'
Record2DCompressMethod = 'auto'
Record2DCompressQuality = 75
Record2DFPS = 'auto'
Record2DFileName = '%f_anim_%n.avi'
Rendering = 'on'
Sound = 'on'
StatusBar = 'on'
Stereo3D = 'anaglyph'
Stereo3DCameraOffset = 0.1
Stereo3DHIT = 0
Textures = 'on'
ToolBar = 'on'
Tooltips = 'on'
Transparency = 'on'
Triad = 'none'
Viewpoint = 'View 1 - Observer'
Wireframe = 'off'
World = vrworld object: 1-by-1
ZoomFactor = 1
```

Version History

Introduced before R2006a

See Also

`vrfigure` | `get` | `vr.utils.stereo3d`

Topics

“Interact with Virtual Reality Worlds”
“Create vrworld Object for a Virtual World”
“View a Virtual World in Stereoscopic Vision”

vrgcbf

Current callback `vrfigure` object

Syntax

```
f = vrgcbf
```

Description

`f = vrgcbf` returns a `vrfigure` object representing the virtual reality figure that contains the callback currently being executed.

When no virtual reality figure callbacks are executing, `vrgcbf` returns an empty array of `vrfigure` objects.

Version History

Introduced in R2008b

See Also

`vrfigure` | `vr.canvas` | “Create vrworld Object for a Virtual World” | “View a Virtual World in Stereoscopic Vision”

vrgcf

Handle for active virtual reality figure

Syntax

```
h = vrgcf
```

Description

`h = vrgcf` returns the handle of the current virtual reality figure. The current virtual reality figure is the currently active virtual reality figure window in which you can get and set the viewer properties. If no virtual reality figure exists, the MATLAB software returns an empty `vrfigure` object.

This method is most useful to query and set virtual reality figure properties.

Version History

Introduced in R2008b

See Also

`vrfigure`

vrgetpref

Values of Simulink 3D Animation preferences

Syntax

```
x = vrgetpref
x = vrgetpref('preference_name')
x = vrgetpref('preference_name','factory')
x = vrgetpref('factory')
```

Arguments

preference_name	Name of the preference to read.

Description

`x = vrgetpref` returns the values of all the Simulink 3D Animation preferences in a structure array.

`x = vrgetpref('preference_name')` returns the value of the specified preference. If *preference_name* is a cell array of preference names, a cell array of corresponding preference values is returned.

`x = vrgetpref('preference_name','factory')` returns the default value for the specified preference.

`x = vrgetpref('factory')` returns the default values for all the preferences.

The following preferences are defined. For preferences that begin with the string `DefaultFigure` or `DefaultWorld`, these values are the default values for the corresponding `vrfigure` or `vrworld` property:

Preference	Description
`AutoCreateThumbnail`	Creates a thumbnail of a virtual world when you open a virtual world. The default is `'off'`. Setting this preference to `'on'` can be helpful if you download multiple virtual worlds from the Internet, without saving them. Creating thumbnails on file open provides thumbnails the next time someone browses through the downloaded worlds.

Preference	Description
`DataTypeBool`	Specifies the handling of the virtual world `Bool` data type for `vrnode/setfield` and `vrnode/getfield`. Valid values are `'logical'` and `'char'`. If set to `'logical'`, the virtual world `Bool` data type is returned as a logical value. If set to `'char'`, the `Bool` data type is returned `'on'` or `'off'`. Default is `'logical'`.
`DataTypeInt32`	Specifies handling of the virtual world `Int32` data type for `vrnode/setfield` and `vrnode/getfield`. Valid values are `'int32'` and `'double'`. If set to `'int32'`, the virtual world `Int32` data type is returned as `int32`. If set to `'double'`, the `Int32` data type is returned as `'double'`. Default is `'double'`.
`DataTypeFloat`	Specifies the handling of the virtual world `float` data type for `vrnode/setfield` and `vrnode/getfield`. Valid values are `'single'` and `'double'`. If set to `'single'`, the virtual world `Float` and `Color` data types are returned as `'single'`. If set to `'double'`, the `Float` and `Color` data types are returned as `'double'`. Default is `'double'`.
`DefaultCanvasNavPanel`	Controls the appearance of the control panel in the `vr.canvas` object. Values are: • `'none'` Panel is not visible. • `'minimized'` Panel appears as a minimized icon in the right-hand corner of the viewer. • `'translucent'` Panel floats half transparently above the scene. • `'opaque'` Panel floats above the scene. Default: `'none'`
`DefaultCanvasUnits`	Specifies default units for new `vr.canvas` objects. See `vr.canvas` for detailed description. Default is `'normalized'`.
`DefaultEditorMouseBehavior`	Specifies whether the mouse in the view pane is in navigation mode or selection mode (for highlighting corresponding nodes in the tree view pane). The default is `'navigate'`.
`DefaultEditorHighlighting`	Specifies whether to highlight virtual world objects selected in the view pane. The default is `'on'`.

Preference	Description
`DefaultFigureAntiAliasing`	Determines whether antialiasing is used by default for new `vrfigure` objects. This preference also applies to new `vr.canvas` objects. Valid values are `'off'` and `'on'`.
`DefaultFigureCaptureFileName`	Specifies default file name for `vr.capture` files. See `get` for detailed description. Default is `'%f_anim_%n.tif'`.
`DefaultFigureDeleteFcn`	Specifies the default callback invoked when closing a `vrfigure` object.
`DefaultFigureLighting`	Specifies whether the lights are rendered by default for new `vrfigure` objects. This preference also applies to new `vr.canvas` objects. Valid values are `'off'` and `'on'`.
`DefaultFigureMaxTextureSize`	Specifies the default maximum size of a texture used in rendering new `vrfigure` objects. This preference also applies to new `vr.canvas` objects. Valid values are `'auto' and 32` <= x <= video card limit, where x is a power of 2.
`DefaultFigureNavPanel`	Specifies the default appearance of the control panel in the viewer. Valid values are `'opaque'`, `'translucent'`, `'none'`, `'halfbar'`, `'bar'`, and `'factory'`. Default is `'halfbar'`.
`DefaultFigureNavZones`	Specifies whether the navigation zone is on or off by default for new `vrfigure` objects. This preference also applies to new `vr.canvas` objects. Valid values are `'off'` and `'on'`.
`DefaultFigurePosition`	Sets the default initial position and size of the Simulink 3D Animation Viewer window. Valid value is a vector of four doubles.
`DefaultFigureRecord2DCompressMethod`	Specifies the default compression method for creating 2-D animation files for new `vrfigure` objects. Valid values are `''`, `'auto'`, `'lossless'`, and `'codec_code'`.
`DefaultFigureRecord2DCompressQuality`	Specifies the default quality of 2-D animation file compression for new `vrfigure` objects. Valid values are 0-100.
`DefaultFigureRecord2DFileName`	Specifies the default 2-D offline animation file name for new `vrfigure` objects.
`DefaultFigureRecord2DFPS`	Specifies the default frames per second playback speed. To have the 2D AVI animation play back at approximately the same playback speed as the 3D virtual world animation, set this preference to `auto`.

Preference	Description
`DefaultFigureRendering`	Specifies whether to render a `vrfigure` or `vr.canvas` object. Turning off rendering improves performance. For example, if your code does batch operations on a virtual figure, you can turn off rendering during that processing and then turn it back on after the processing.
`DefaultFigureStatusBar`	Specifies whether the status bar appears by default at the bottom of the Simulink 3D Animation Viewer for new `vrfigure` objects. Valid values are `'off'` and `'on'`.
`DefaultFigureTextures`	Specifies whether textures should be rendered by default for new `vrfigure` objects. This preference also applies to new `vr.canvas` objects. See `get` for detailed description. Default is `'on'`.
`DefaultFigureToolBar`	Specifies whether the toolbar appears by default on the Simulink 3D Animation Viewer for new `vrfigure` objects. Valid values are `'off'` and `'on'`.
`DefaultFigure Transparency`	Specifies whether or not transparency information is taken into account when rendering for new `vrfigure` objects. This preference also applies to new `vr.canvas` objects. Valid values are `'off'` and `'on'`.
`DefaultFigureWireframe`	Specifies whether objects are drawn as solids or wireframes by default for new `vrfigure` objects. This preference also applies to new `vr.canvas` objects. Valid values are `'off'` and `'on'`.
`DefaultViewer`	Specifies which viewer is used to view a virtual scene. • `'internal'` Default Simulink 3D Animation Viewer. • `'web'` Web browser becomes viewer. This is the current Web browser virtual world plug-in.
`DefaultWorldRecord3D FileName`	Specifies the default 3-D animation file name for new `vrworld` objects.
`DefaultWorldRecordMode`	Specifies the default animation recording mode for new `vrworld` objects. Valid values are `'manual'` and `'scheduled'`.
`DefaultWorldRecord Interval`	Specifies the default start and stop times for scheduled animation recording for new `vrworld` objects. Valid value is a vector of two doubles.
`DefaultWorldRemoteView`	Specifies whether the virtual world is enabled by default for remote viewing for new `vrworld` objects. Valid values are `'off'` and `'on'`.

Preference	Description
`DefaultWorldTimeSource`	Specifies the default source of the time for new `vrworld` objects. Valid values are `'external'` and `'freerun'`.
`Editor`	Specifies which virtual world editor to use. Path to the virtual world editor. If this path is empty, the MATLAB editor is used. If you set the `Editor` as `Builtin`, then MATLAB uses the built-in graphical Virtual Reality 3D file editor. The path setting is active only if you select the `Custom` option.
`EditorPreserveLayout`	Specifies whether the 3D World Editor starts up with a saved version of the layout of a virtual world when you exited it or reverts to the default layout. The layout of the virtual world display pane includes settings for the view, viewpoints, navigation, and rendering. Valid values are `'off'` and `'on'`. The default is on (use saved layout).
`HttpPort`	For remote access, IP port number used to access the Simulink 3D Animation server over the Web via HTTP. If you change this preference, you must restart the MATLAB software before the change takes effect.
`TransportBuffer`	For remote access, length of the transport buffer (network packet overlay) for communication between the Simulink 3D Animation server and its clients.
`TransportTimeout`	Amount of time the Simulink 3D Animation server waits for a reply from the client. If there is no response from the client, the Simulink 3D Animation server disconnects from the client.
`VrPort`	For remote access, IP port used for communication between the Simulink 3D Animation server and its clients. If you change this preference, you must restart the MATLAB software before the change takes effect.

The `HttpPort`, `VrPort`, and `TransportBuffer` preferences affect Web-based remote viewing of virtual worlds. `DefaultFigurePosition` and `DefaultNavPanel` affect the Simulink 3D Animation Viewer.

`DefaultFigureNavPanel` — Controls the appearance of the navigation panel in the Simulink 3D Animation Viewer. For example, setting this value to `'translucent'` causes the navigation panel to appear translucent.

`DefaultViewer` — Determines whether the virtual scene appears in the default Simulink 3D Animation Viewer or in your Web browser.

`DefaultViewer` Setting	Description
`'internal'`	Default Simulink 3D Animation Viewer.

DefaultViewer Setting	Description
'web'	Viewer is your Web browser.

Editor — Contains a path to the virtual world editor executable file. When you use the edit command, Simulink 3D Animation runs the virtual world editor executable with all parameters required to edit the virtual world file.

When you run the editor, Simulink 3D Animation uses the Editor preference value as if you typed it into a command line. The following tokens are interpreted:

%matlabroot	Refers to the MATLAB root folder
%file	Refers to the virtual world file name

For instance, a possible value for the Editor preference is

```
`%matlabroot\bin\win64\meditor.exe %file'
```

If this preference is empty, the MATLAB editor is used.

HttpPort -- Specifies the network port to be used for Web access. The port is given in the Web URL as follows:

```
http://server.name:port_number
```

The default value of this preference is 8123.

TransportBuffer — Defines the size of the message window for client-server communication. This value determines how many messages, at a maximum, can travel between the client and the server at one time.

Generally, higher values for this preference make the animation run more smoothly, but with longer reaction times. (More messages in the line create a buffer that compensates for the unbalanced delays of the network transfer.)

The default value is 5, which is optimal for most purposes. You should change this value only if the animation is significantly distorted or the reaction times are very slow. On fast connections, where delays are introduced more by the client rendering speed, this value has very little effect. Viewing on a host computer is equivalent to an extremely fast connection. On slow connections, the correct value can improve the rendering speed significantly but, of course, the absolute maximum is determined by the maximum connection throughput.

VrPort — Specifies the network port to use for communication between the Simulink 3D Animation server (host computer) and its clients (client computers). Normally, this communication is completely invisible to the user. However, if you view a virtual world from a client computer, you might need to configure the security network system (firewall) so that it allows connections on this port. The default value of this preference is 8124.

Version History

Introduced before R2006a

See Also

“Set Simulink 3D Animation Preferences”

vrifs2patch

Convert virtual world `IndexedFaceSet` nodes to MATLAB patches

Syntax

```
vrifs2patch(ifs)
```

Description

`vrifs2patch(ifs)` converts the `ifs` array of existing `IndexedFaceSet` nodes to MATLAB patch objects.

Note This function converts only geometry and color data of the source `IndexedFaceSet` node.

Examples

Convert IndexedFaceSet Nodes to MATLAB Patches

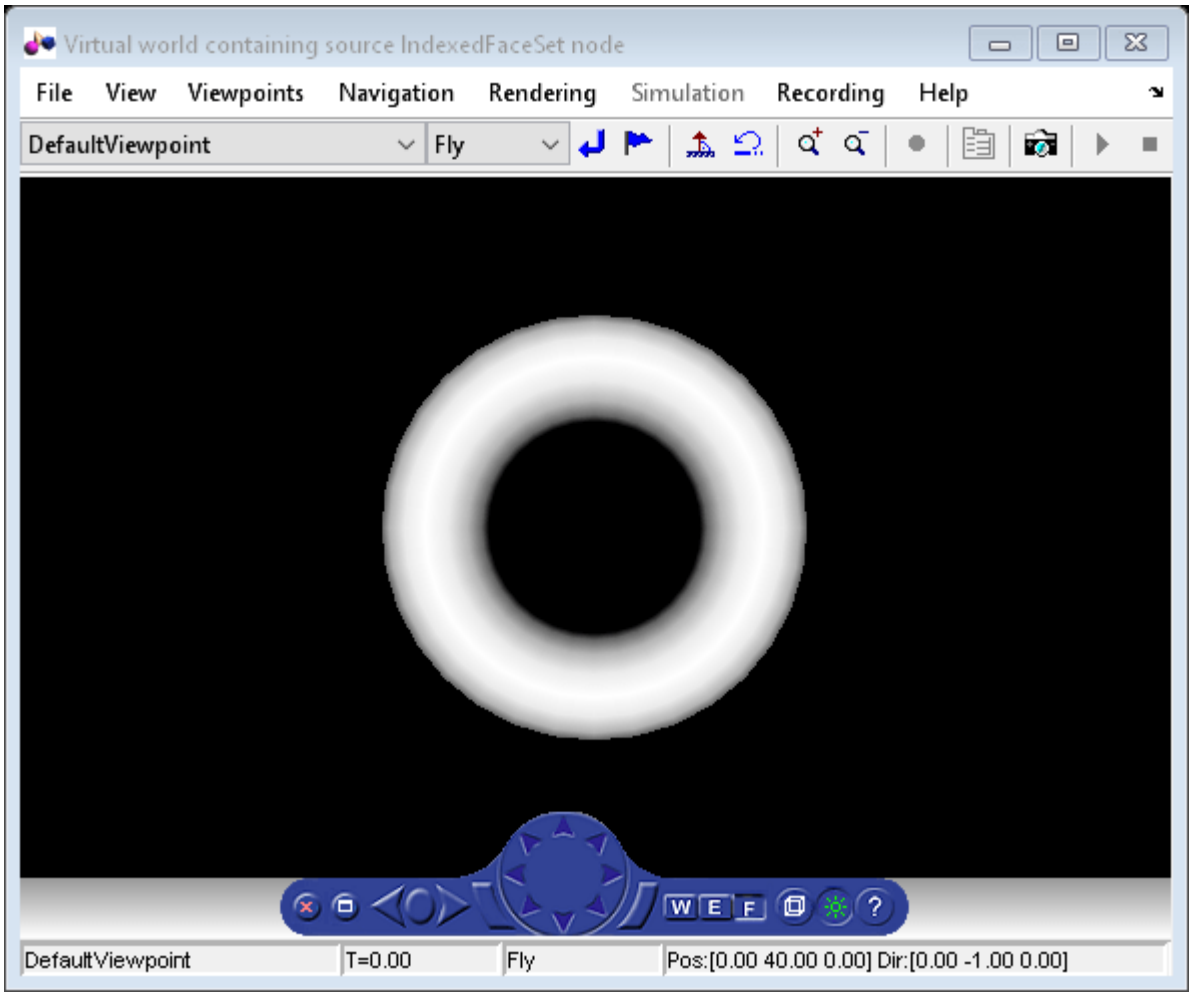
This command converts three `IndexedFaceSet` nodes to MATLAB® patch objects.

Open virtual world containing an `IndexedFaceSet` node.

```
w1 = vrworld('*sl3dlib/objects/Components/Shapes/Torus_High.wrl');
open(w1);
```

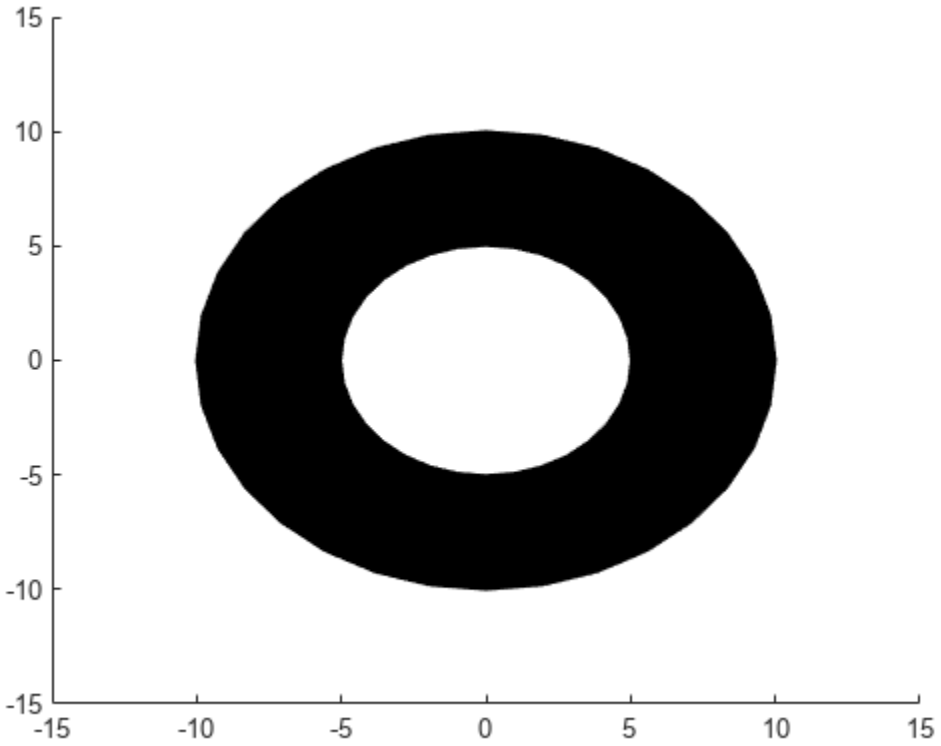
View the virtual world as a virtual figure.

```
vrfig1 = vrfigure(w1, ...
           'Name', 'Virtual world containing source IndexedFaceSet node', ...
           'CameraBound', 'off', ...
           'CameraPosition',[0 40 0], ...
           'CameraDirection',[0 -1 0], ...
           'CameraUpVector',[0 0 -1]);
vrdrawnow;
```



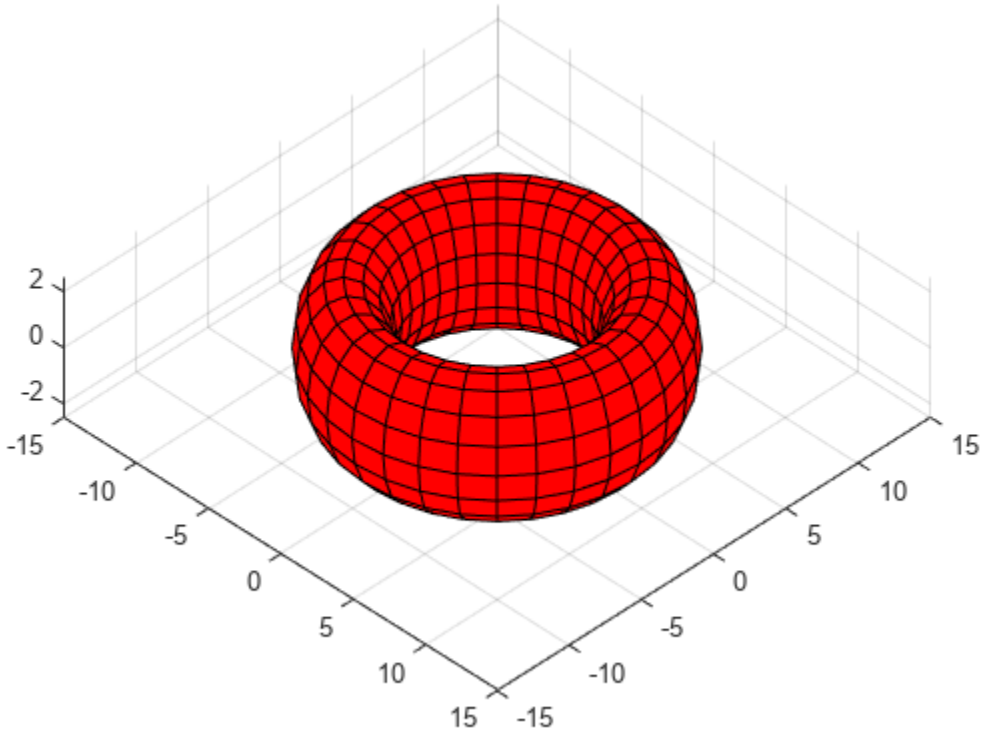
Convert the `IndexedFaceSet` a MATLAB patch and show it.

```
figure('Name', 'Resulting patch');
tp = vrifs2patch(w1.torushi.children.geometry);
```



Change the patch color, show the axes grid, rotate the camera, and enable mouse rotation.

```
tp.FaceColor = 'red';

axs = gca;
axs.XGrid = 'on';
axs.YGrid = 'on';
axs.ZGrid = 'on';

camorbit(45, -20);

rotate3d on
```



Input Arguments

`ifs` — `IndexedFaceSet` nodes to convert
array

`IndexedFaceSet` nodes, specified as an array.

Version History

Introduced in R2015a

See Also

`vrpatch2ifs` | `patch`

Topics
“Introduction to Patch Objects”

vrimport

Import 3D file into virtual world or node

Syntax

```
node = vrimport(source)
node = vrimport(parent,source)
node = vrimport( ___ ,format)
[node,virtualWorld] = vrimport( ___ )
```

Description

`node = vrimport(source)` creates an empty VRML virtual world and imports the `source` 3D file into it. The format of the 3D file is detected automatically. You can import these file formats:

- FBX (Autodesk® FilmBoX format)
- DAE (Collada digital asset exchange)
- SDF (simulation description format)
- STL (STereoLithography)
- URDF (unified robot description file)
- XML (Physical Modeling XML files)

The function returns a handle to the newly created node.

`node = vrimport(parent,source)` specifies the existing virtual world or node to import the 3D source file into.

`node = vrimport( ___ ,format)` explicitly specifies the file format of the 3D source file (for example, `'urdf'`). If the format of the source file does not match the format specified in the `format` argument, the function returns an error.

`[node,virtualWorld] = vrimport( ___ )` returns the handle of the new node and the handle of the virtual world that contains that node.

Examples

Import STL File Into an Empty Virtual World

This example imports an STL file `rover_1.stl`, a model of a simple wheeled robot. The example also shows how to add visual appearance and material nodes to the imported model in the virtual world.

Create a virtual world with the imported model.

```
[n,w] = vrimport(which('Rover_1.stl'));
```

View the virtual world with the imported shape.

```
view(w)
```

Scale the imported model from mm to dm to see it in the view.

```
n.scale = [0.01 0.01 0.01]
```

Rotate the rover around the x-axis.

```
w.Rover_Transform.rotation = [1 0 0 -pi/2]
```

Explore the virtual world structure.

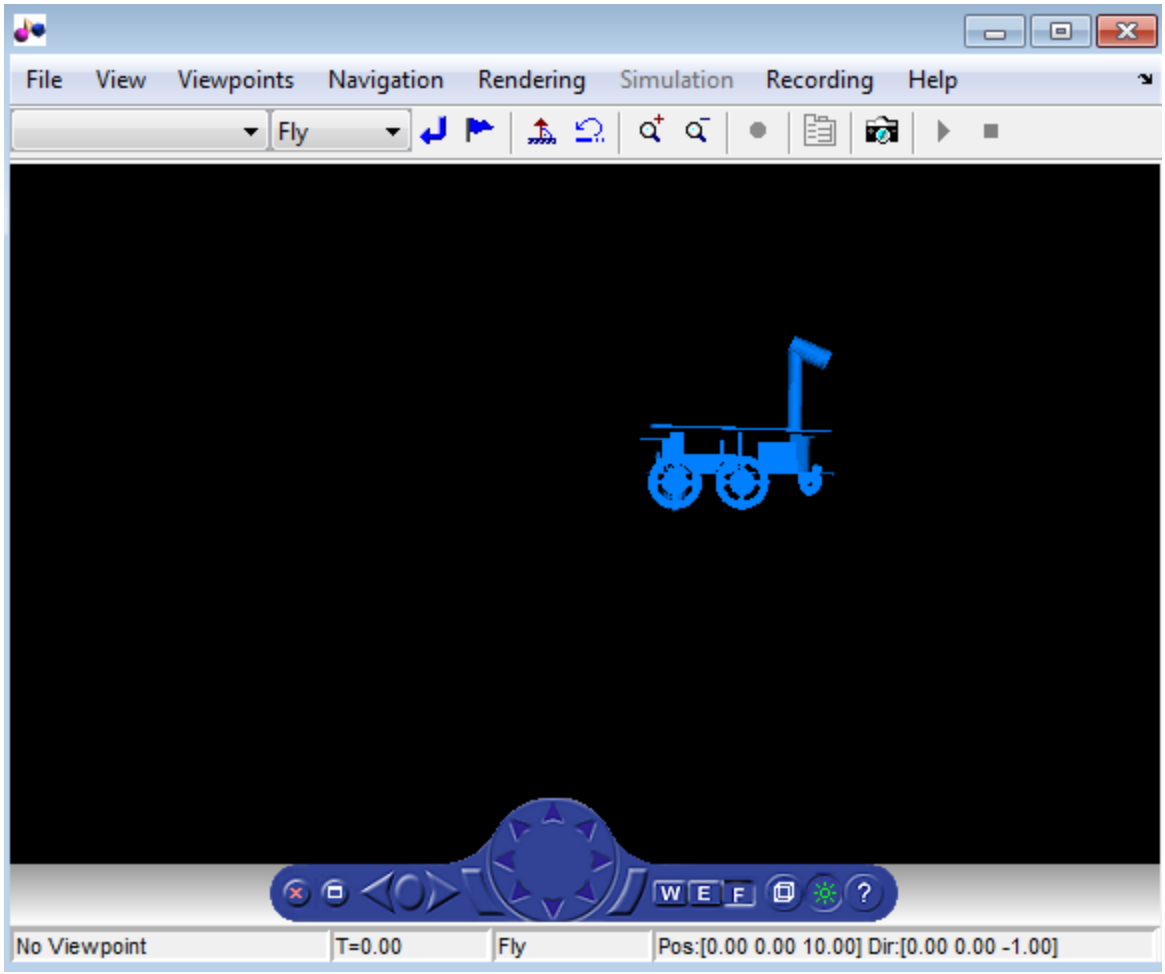
```
get(w,'Nodes')
```

STL imported shapes have no visual properties. Add an `Appearance` and a `Materials` node to the shape. The `Appearance` node is created in the `appearance` field of the `Shape`. The `Material` node is create in the `material` field of the `Appearance` node.

```
app = vrnode(w.Rover_Shape,'appearance','Rover_App','Appearance');
mat = vrnode(w.Rover_App,'material','Rover_Mat','Material');
```

Set the diffuse color to a shade of blue.

```
w.Rover_Mat.diffuseColor = [0 0.5 1]
```



Save the virtual world.

```
save(w,'Rover_1.x3d')
```

Import a DAE file

This example imports a `.dae` format file into a virtual world.

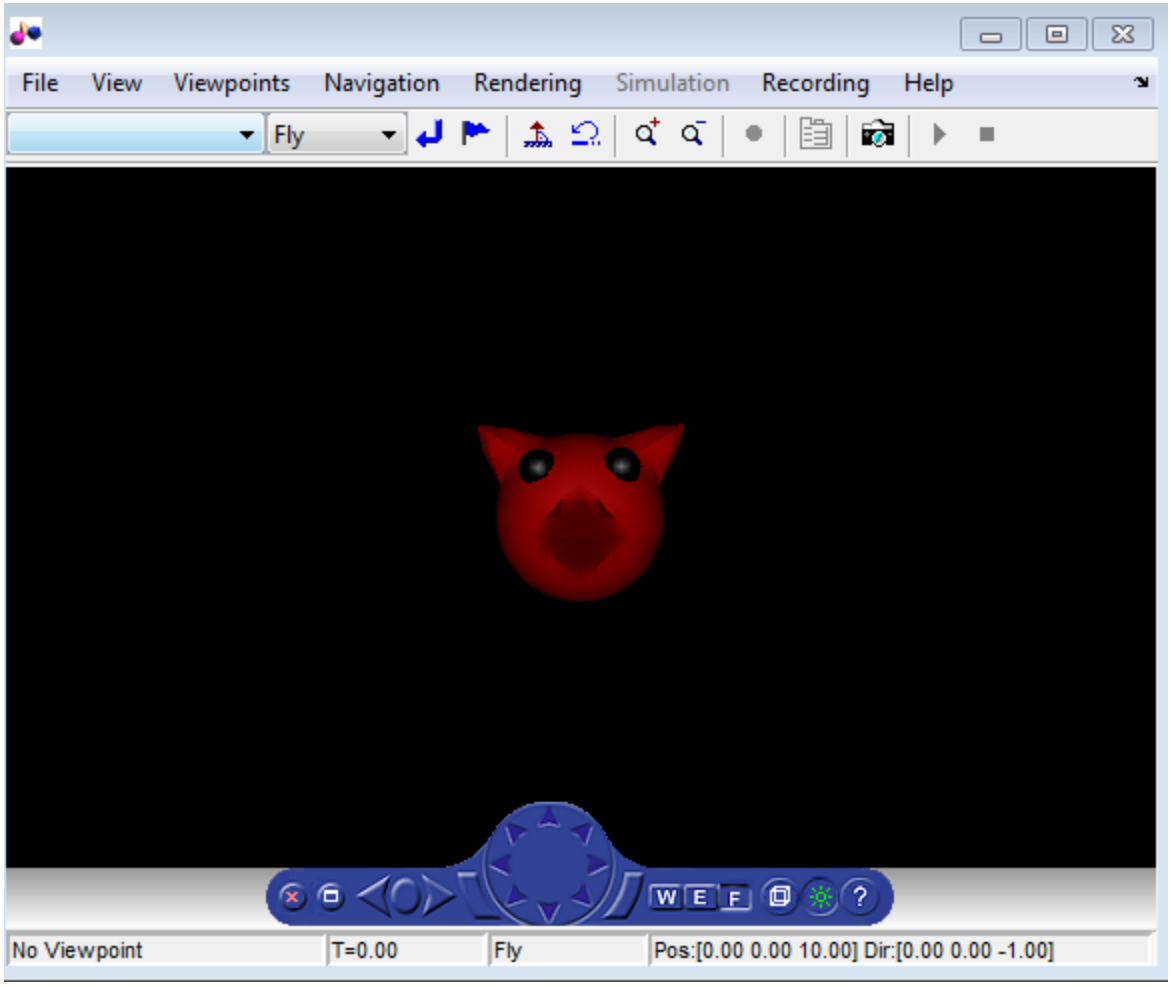
Import the `fox.dae` file to a node in a virtual world.

```
[n,w] = vrimport(which('fox.dae'))
```

```
n =

    vrnode object: 1-by-1

    COLLADA_fox_Transform_0001 (Transform) []


w =

    vrworld object: 1-by-1

     (No Virtual Reality 3D File Associated)
```

View the imported visual representation.

```
view(w)
```



Save the virtual world.

```
save(w,'fox.x3d')
```

Input Arguments

`source` — 3D source file

character vector

3D source file path, specified as a character vector. The 3D file can be in DAE, SDF, STL, or URDF format.

`format` — File format of source 3D file

`'fbx'` | `'dae'` | `'sdf'` | `'stl'` | `'urdf'`

File format of source 3D file, specified as a character vector. Use this argument to specify explicitly the required format for the source 3D file.

`parent` — Virtual world or node to import 3D source file to

`vrworld` object | `vrnode` object

Virtual world or node to import 3D source file into, specified as a virtual world handle or node handle.

- If the parent is a virtual world, the imported node is placed at the `ROOT` node of the parent.
- If the parent is a node in a virtual world, the imported node is placed in the `children` field of the node.

Output Arguments

`node` — New node
`vrnode` object

New node, returned as a `vrnode` object.

`virtualWorld` — Virtual world containing new node
`vrworld` object

Virtual world containing new node, returned as a `vrworld` object.

Version History

Introduced in R2016b

See Also

`vrcadcleanup` | `vrphysmod`

Topics
“Import STL and Physical Modeling XML Files”
“Import Visual Representations of Robot Models”
“Link to Simulink and Simscape Multibody Models”

vrinsertrobot

Add robot to virtual world

Syntax

```
node = vrinsert(RBT)
node = vrinsertrobot(parent,RBT)
[node, W] = vrinsertrobot(...)
[node, W, tforms] = vrinsertrobot(...)
```

Description

`node = vrinsert(RBT)` creates an empty virtual world and inserts the visual representation of the Robotics System Toolbox `rigidBodyTree` object `RBT` into it. It then returns a handle to the newly created `node` in the virtual world.

`node = vrinsertrobot(parent,RBT)` inserts the visual representation of the Robotics System Toolbox `rigidBodyTree` object `RBT` into an existing virtual world or node specified by `parent`. If `parent` is a virtual world, object specified by `RBT` is placed at its root. If `parent` is a node within a virtual world, the inserted object is placed as a direct child of `parent`.

`[node, W] = vrinsertrobot(...)` also returns a handle to the virtual world `W` in addition to the visualization of the `rigidBodyTree` object represented by `node`.

`[node, W, tforms] = vrinsertrobot(...)` also returns a handle to the appropriate transforms `tforms`, which can be used to make additional changes to the robot pose.

Examples

Import Robot to an Empty World

This example shows you how to import and insert a `rigidBodyTree` object for the KUKA LBR iiwa robot manipulator into a newly created world.

Import Robot

Create the `rigidBodyTree` object from the URDF file of the associated robot

```
RBT = importrobot('iiwa7.urdf');
RBT.DataFormat = 'Row';
```

For more information on the `rigidBodyTree` structure, see `rigidBodyTree` (Robotics System Toolbox).

Insert and View Robot

Create an empty world and open it.

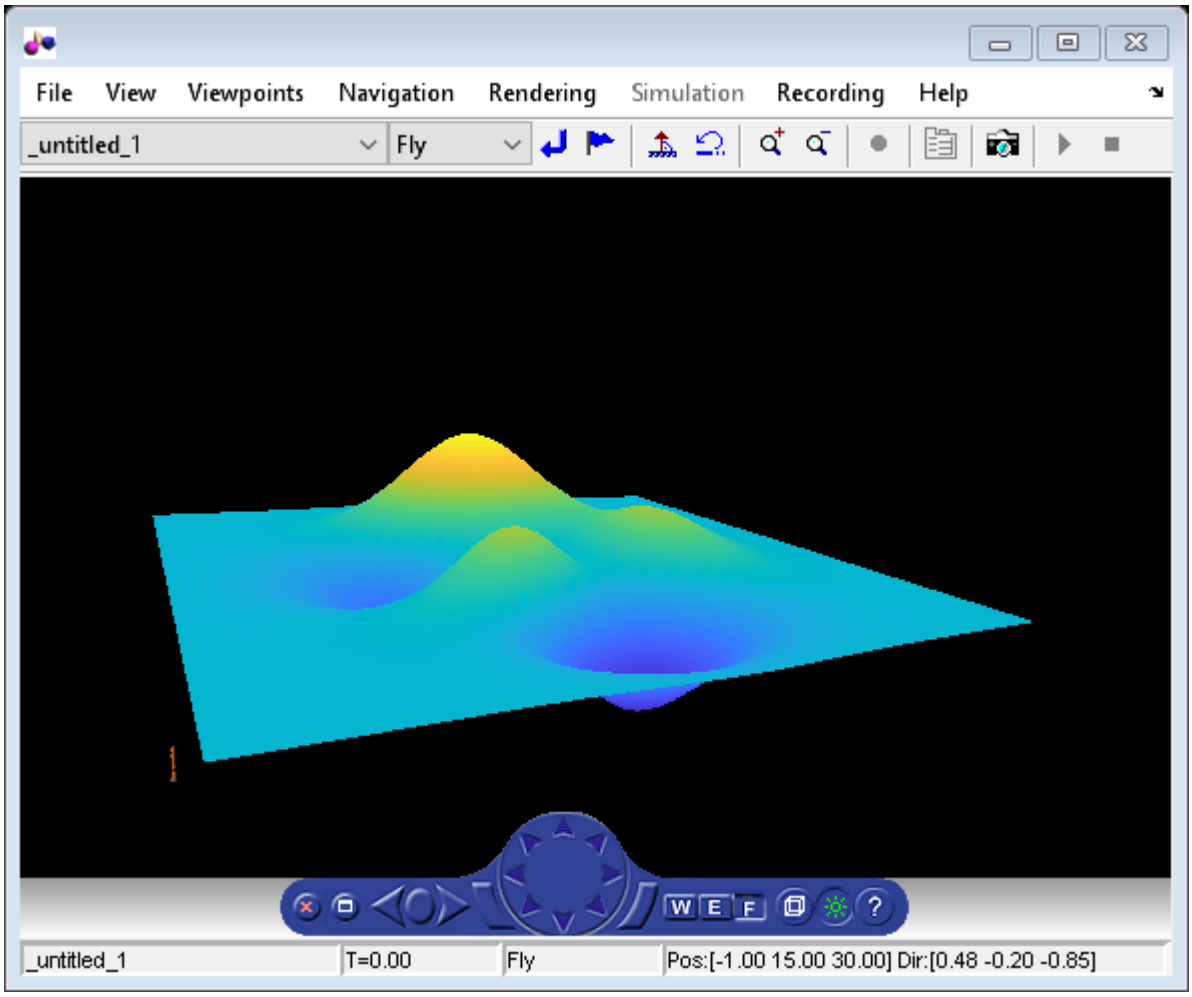
```
w = vrworld('');
open(w);
view(w);
```

Create a node in an empty world using `vrinsertrobot`.

```
node = vrinsertrobot(w,RBT);
```

View the created world in the Simulink 3D Animation™ internal viewer.

```
vrdrawnow
```



Insert Robot into an Existing World

This example shows how to insert a `rigidBodyTree` object to an existing world and update the viewer.

Open a Virtual World

Open up a virtual world in the Simulink 3D Animation™ viewer. This example uses the `robot_scene.wrl` world. To create your own virtual world, see “Create a Virtual World”

```
robotWorld = vrworld('robot_scene','new');
open(robotWorld);
```

Add Robot to the Existing World

Import the KUKA LBR iiwa robot from its URDF definition into a `rigidBodyTree` object.

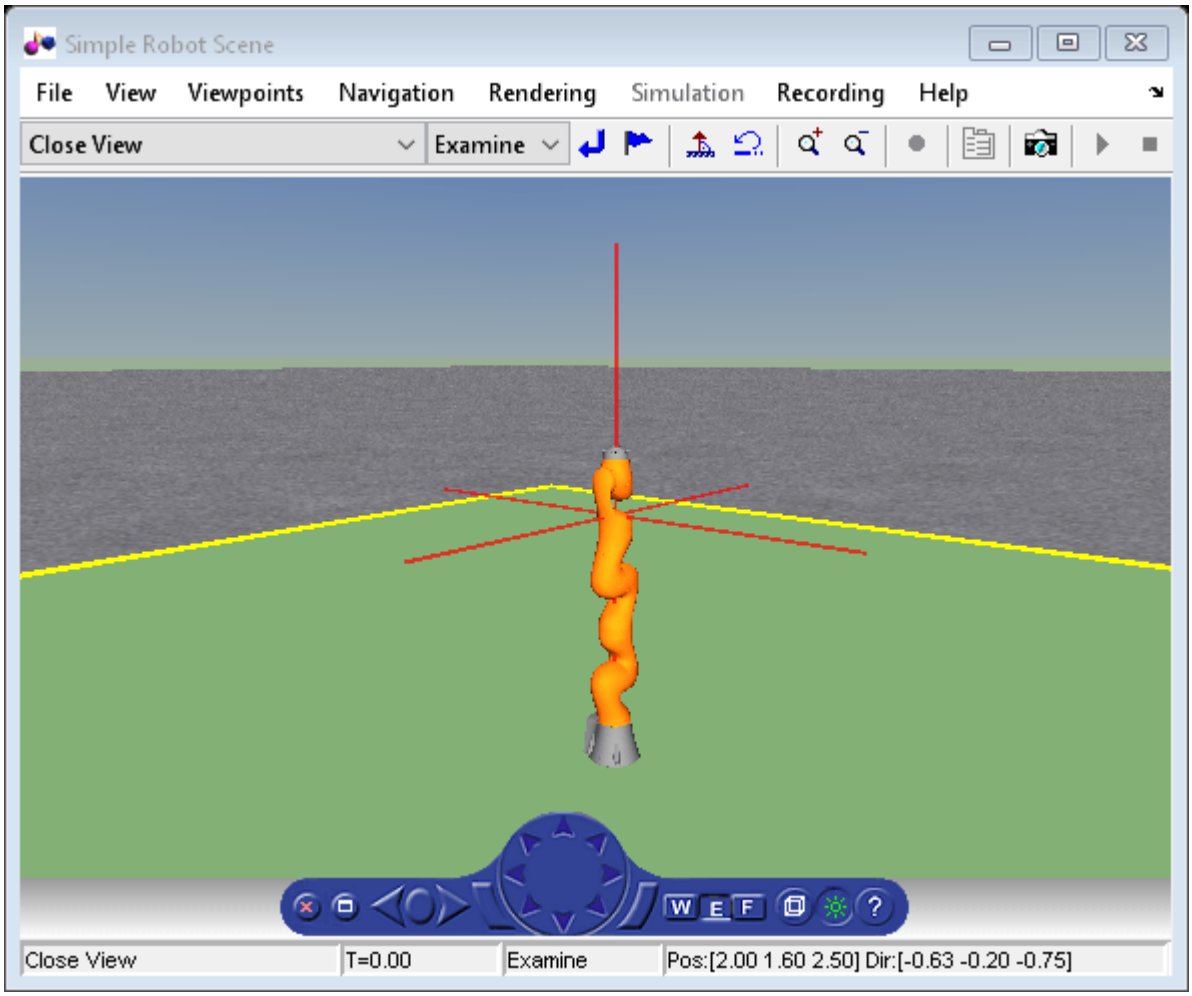
```
rbt = importrobot('iiwa7.urdf');
rbt.DataFormat = 'row';
```

Add the robot to the `robotWorld` world object created in the previous step.

```
n = vrinsertrobot(robotWorld,rbt);
```

Update the scene, even if the viewer is closed. Open the updated world and scene in the internal viewer.

```
vrdrawnow
view(robotWorld);
```



Add Robot to World and Change its Pose

This example shows you how to insert a robot into a virtual world and update its pose

Import the Robot and Setup the World

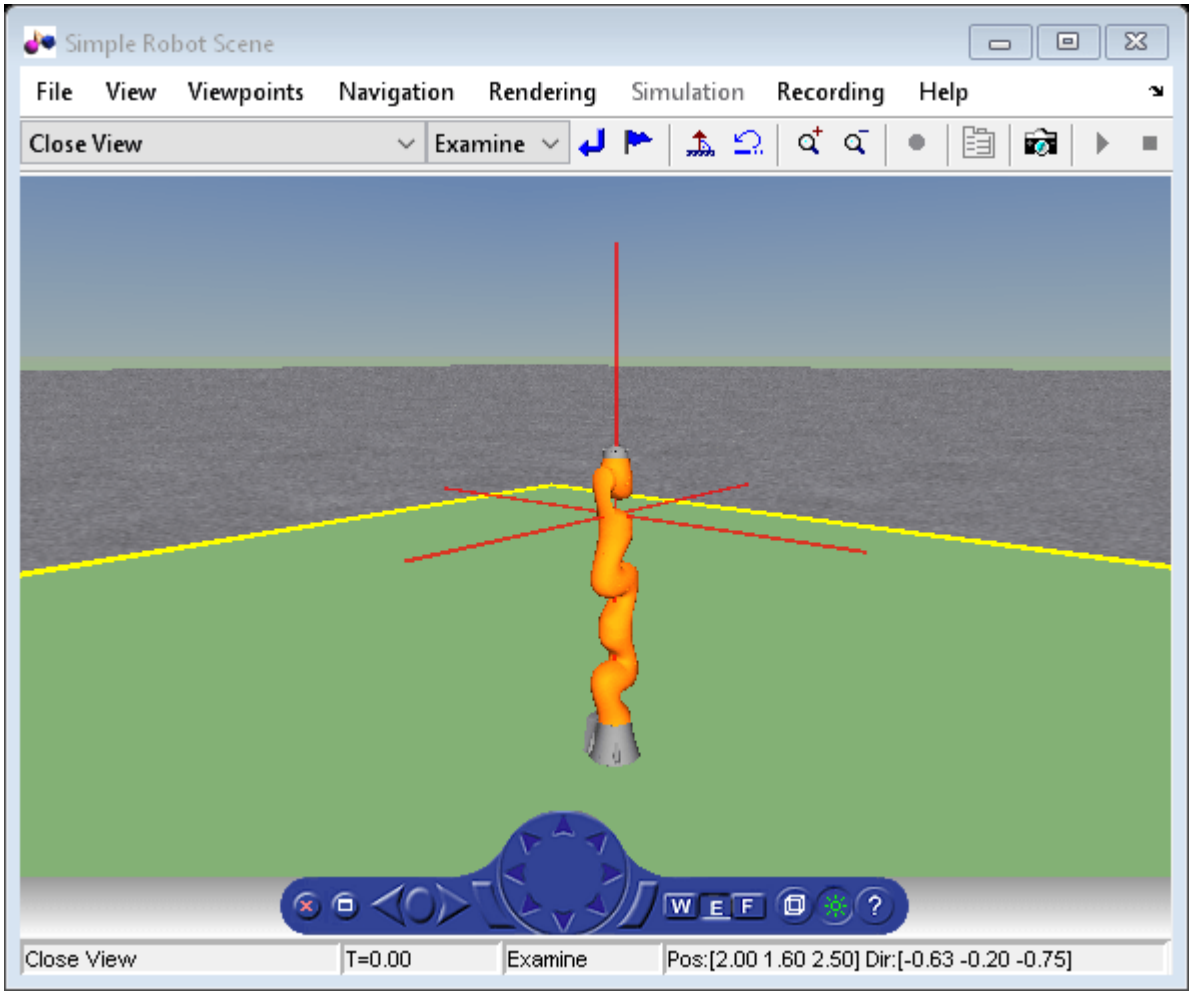
Import the KUKA LFR iiwa robot from its URDF definition and insert it to the virtual world created from `robot_scene.x3d`.

```
RBT = importrobot('iiwa7.urdf');
RBT.DataFormat = 'row';
robotWorld = vrworld('robot_scene');
open(robotWorld);
```

Get Transforms of Current Pose of the Robot

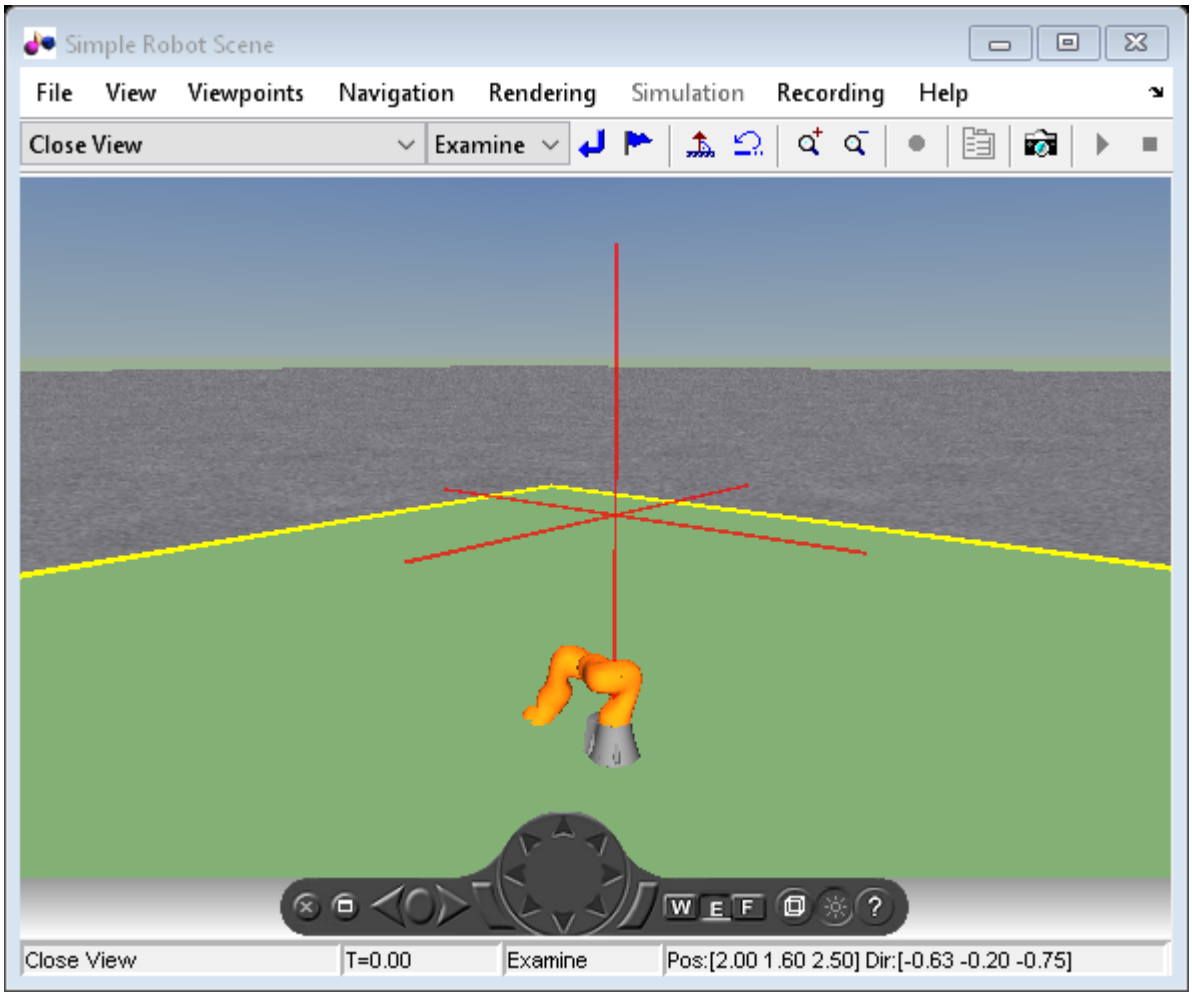
The `tforms` output argument contains a list of transforms that describe the robot pose in its initial or 'home' configuration.

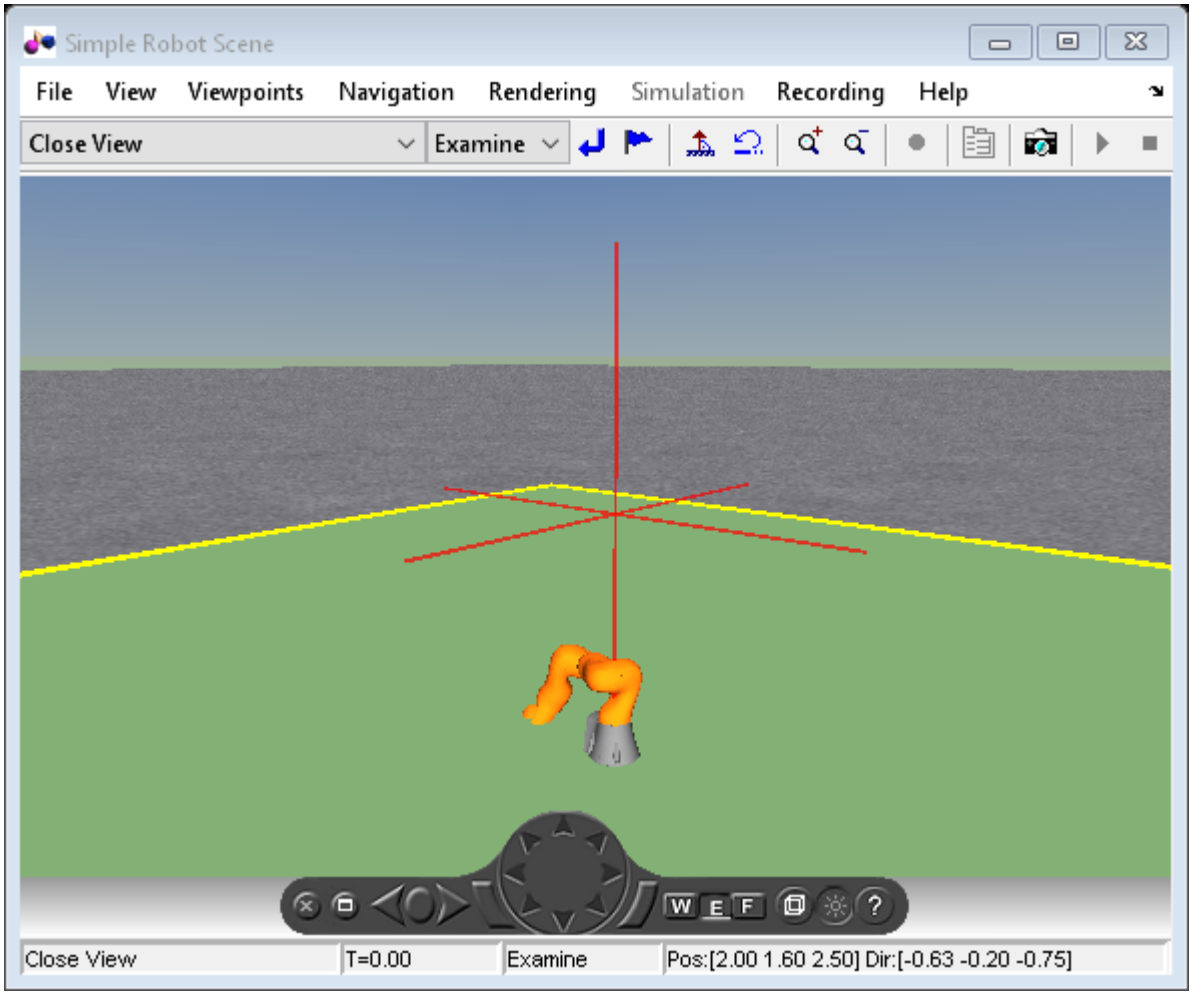
```
[node, W, tforms] = vrinsertrobot(robotWorld, RBT);
vrfigure(robotWorld);
```



Change the Pose of the Robot

```
vrupdaterobot(RBT, tforms, randomConfiguration(RBT));
vrdrawnow;
vrfigure(robotWorld);
```





Input Arguments

RBT — Robot description

`rigidBodyTree` object

Robotics System Toolbox `rigidBodyTree` object. For more information, see `rigidBodyTree` (Robotics System Toolbox).

`parent` — Virtual world node

`vrworld` object | `vrnode` object

Node in the virtual world hierarchy under which to insert the robot specified by `RBT`. If a `vrworld` object is provided, the robot is inserted at the `ROOT` node of the world.

Output Arguments

node — Robot object handle

`vrnode` object

Handle to the newly inserted robot in the virtual world, returned as a `vrnode` object. For more information, see `vrnode`.

`W` — Virtual world handle
`vrworld` object

Handle to the virtual world containing the robot, returned as a `vrworld` object. For more information, see `vrworld`.

`tforms` — List of transforms for the robot
cell array

List of transformations applied to the robot, returned as a cell array of `vrnode` objects.

Version History

Introduced in R2018b

See Also

`vrworld` | `vrimport` | `importrobot` (Robotics System Toolbox) | `rigidBodyTree` (Robotics System Toolbox)

vrinstall

Install and check Simulink 3D Animation components

Syntax

```
vrinstall('action')

vrinstall action

x = vrinstall('action')
```

Arguments

action	Type of action for this function. Values are `-interactive`, `-selftest`, `-check`, `-install`, and `-uninstall`.

Description

You use this function to install on Windows platforms the Ligos V-Realm Builder. The V-Realm Builder is an optional virtual world editor. For details, see “Install V-Realm Editor”.

Note The `vrinstall` command does no perform any action on a Linux® platform.

The actions you can perform

Action Value	Description
`-selftest`	Checks the integrity of the current installation. If this function reports an error, you should reinstall the Simulink 3D Animation software. The function `vrinstall` automatically does a self-test with any other actions.
`-interactive`	Checks for the installed components, and then displays a list of uninstalled components you can choose to install.
`-check`	Checks the installation of optional components. If the given component is installed, returns 1. If the given component is not installed, returns 0. If you do not specify a component, displays a list of components and their status.
`-install`	Installs optional components. This action requires you to specify the component name.
`-uninstall`	Uninstalls optional components. This option is currently available for the editor only. Note that this action does not remove the files for the editor from the installation folder. It removes the editor registry information.

Examples

Install the virtual world editor. This command associates V-Realm Builder with the **Edit** button in the Block Parameters dialog boxes.

```
vrinstall -install
```

Version History

Introduced before R2006a

vrjoystick

Create joystick object

Description

Create a joystick object.

Creation

Description

`joy = vrjoystick(id)` creates a joystick object capable of interfacing with a joystick device. The `id` parameter is a one-based joystick ID. The joystick ID is the system ID assigned to the given joystick device. You can find the properties of the joystick that is connected to the system in the Game Controllers section of the system Control Panel.

`joy = vrjoystick(id,'forcefeedback')` enables force feedback if the joystick supports this capability.

Object Functions

axis	Read status of joystick
button	Read status of joystick
caps	Joystick capabilities
close	Close and invalidate joystick
force	Apply force feedback to joystick axis
pov	Apply force feedback to joystick axis
read	Read status of joystick button, axes and pov

Version History

Introduced in R2007b

See Also

“Set Simulink 3D Animation Preferences”

axis

Read status of joystick

Syntax

```
a = axis(joy, n)
```

Description

`a = axis(joy, n)` reads the status of joystick with axis number `n`.

Input Arguments

`joy` — Joystick object

`vrjoystick` object

Joystick, specified as a `vrjoystick` object.

`n` — Axis number

real positive scalar | real positive vector

Axis number of joystick, specified as a scalar or a vector.

Output Arguments

`a` — Status of joystick

real positive scalar in the range [-1, 1]

Status of joystick axis, returned as a real positive scalar in the range [-1,1]. If `n` is a vector, multiple buttons are returned.

Version History

Introduced in R2007b

button

Read status of joystick

Syntax

```
b = button(joy,n)
```

Description

`b = button(joy,n)` reads the status of joystick button number `n`.

Input Arguments

`joy` — Joystick object

`vrjoystick` object

Joystick, specified as a `vrjoystick` object.

`n` — Button number

real positive scalar | real positive vector

Button number of joystick, specified as a scalar or a vector.

Output Arguments

`b` — Status of button

`0` | `1`

Status of joystick button, returned as either `0` or `1`. If `n` is a vector, multiple buttons are returned.

Version History

Introduced in R2007b

See Also

`vrjoystick`

caps

Joystick capabilities

Syntax

```
c = caps(joy)
```

Description

`c = caps(joy)` returns joystick capabilities, such as the number of axes, buttons, POVs, and force-feedback axes. The return value is a structure with fields named `Axes`, `Buttons`, `POVs`, and `Forces`.

Input Arguments

`joy` — Joystick object

`vrjoystick` object

Joystick, specified as a `vrjoystick` object.

Output Arguments

`c` — Joystick capabilities

structure

Joystick capabilities, returned as a structure.

Version History

Introduced in R2007b

close

Close and invalidate joystick

Syntax

```
close(joy)
```

Description

`close(joy)` closes and invalidates joystick object.

Input Arguments

`joy` — Joystick object

`vrjoystick` object

Joystick, specified as a `vrjoystick` object.

Version History

Introduced in R2007b

See Also

`vrjoystick`

force

Apply force feedback to joystick axis

Syntax

```
force(joy,n,f)
```

Description

`force(joy,n,f)` applies force feedback `f` to joystick axis `n`.

Input Arguments

`joy` — Joystick object

`vrjoystick` object

Joystick, specified as a `vrjoystick` object.

`n` — Axis number

real positive scalar | real positive vector

Axis number of joystick, specified as a scalar or a vector.

`f` — Force feedback to be applied

scalar | vector

Force feedback to be applied to joystick axis, specified as a scalar or a vector in the range [-1,1]. Values of `f` should be in range of -1 to 1, and the number of elements in `f` should either match the number of elements of `n`, or `f` can be a scalar to be applied to all the axes specified by `n`.

Version History

Introduced in R2007b

See Also

`vrjoystick`

pov

Apply force feedback to joystick axis

Syntax

```
p = pov(joy,n)
```

Description

`p = pov(joy,n)` reads the status of joystick point of view (POV) of axis number `n`.

Input Arguments

`joy` — Joystick object

`vrjoystick` object

Joystick, specified as a `vrjoystick` object.

`n` — Axis number

real positive scalar | real positive vector

Axis number of joystick, specified as a scalar or a vector.

Output Arguments

`p` — Point of view

real scalar

Point of view, returned as a real scalar in degrees. When a value of -1 is no axis is selected.

Version History

Introduced in R2007b

See Also

`vrjoystick`

read

Read status of joystick button, axes and pov

Syntax

```
[a,b,p] = read(joy)
[a,b,p] = read(joy,f)
```

Description

`[a,b,p] = read(joy)` reads the status of axes, buttons, and point of views (POVs) of the specified joystick.

`[a,b,p] = read(joy,f)` additionally applies feedback forces to a force-feedback joystick.

Input Arguments

`joy` — Joystick object

`vrjoystick` object

Joystick, specified as a `vrjoystick` object.

`n` — Axis number

real positive scalar | real positive vector

Axis number of joystick, specified as a scalar or a vector.

`f` — Force feedback to be applied

scalar | vector

Force feedback to be applied to joystick axis, specified as a scalar or a vector in the range [-1,1]. Values of `f` should be in range of -1 to 1, and the number of elements in `f` should either match the number of elements of `n`, or `f` can be a scalar to be applied to all the axes specified by `n`.

Output Arguments

`a` — Status of joystick

real positive scalar in the range [-1, 1]

Status of joystick axis, returned as a real positive scalar in the range [-1,1]. If `n` is a vector, multiple buttons are returned.

`b` — Status of button

`0` | `1`

Status of joystick button, returned as either `0` or `1`. If `n` is a vector, multiple buttons are returned.

`p` — Point of view

real scalar

Point of view, returned as a real scalar in degrees. When a value of -1 is no axis is selected.

Version History

Introduced in R2007b

See Also

`vrjoystick`

vrlib

Open Simulink block library for Simulink 3D Animation

Syntax

```
vrlib
```

Description

The Simulink library for the Simulink 3D Animation product has a number of blocks and utilities. You can access these blocks in one of the following ways:

- In the MATLAB Command Window, type `vrlib`.
- From a Simulink block diagram, select the **Library Browser** from the **Simulation** tab of the toolstrip.
- In the MATLAB Command Window, click the Simulink icon.

Version History

Introduced before R2006a

vrnode

Create node or handle to existing node

Syntax

```
mynode = vrnode

mynode = vrnode([])

mynode = vrnode(vrworld_object,'node_name')

mynode = vrnode(vrworld_object, 'node_name','node_type')

mynode = vrnode(vrworld_object, 'USE', othernode)

mynode = vrnode(parent_node,'parent_field', 'node_name',
'node_type')

mynode = vrnode(parent_node,'parent_field', 'USE',
'othernode')
```

Arguments

`vrworld_object`	Name of a `vrworld` object representing a virtual world.
`node_name`	Name of the node.
`node_type`	Type of the node.
`parent_node`	Name of the parent node that is a `vrnode` object.
`parent_field`	Name of the field of the parent node.
`'USE'`	Enables a `USE` reference to another node.
`othernode`	Name of another node for a `USE` reference.

Description

`mynode = vrnode` creates an empty `vrnode` handle that does not reference any node.

`mynode = vrnode([])` creates an empty array of `vrnode` handles.

`mynode = vrnode(vrworld_object,'node_name')` creates a handle to an existing named node in the virtual world.

`mynode = vrnode(vrworld_object, 'node_name','node_type')` creates a new node called *`node_name`* of type *`node_type`* on the root of the virtual world. It returns the handle to the newly created node.

`mynode = vrnode(vrworld_object, 'USE', othernode)` creates a `USE` reference to the node `othernode` on the root of the world `vrworld_object`. It returns the handle to the virtual world to the original node.

`mynode = vrnode(parent_node,'parent_field', 'node_name','node_type')` creates a new node called `node_name` of type `node_type` that is a child of the `parent_node` and resides in the field `parent_field`. It returns the handle to the newly created node.

`mynode = vrnode(parent_node,'parent_field', 'USE', 'othernode')` creates a `USE` reference to the node `othernode` as a child of node `parentnode` and resides in the field `parentfield`. It returns the handle to the original node.

A `vrnode` object identifies a virtual world node in a way very similar to a handle. If you apply the `vrnode` method to a node that does not exist, the method creates a node, the `vrnode` object, and returns the handle to the `vrnode` object. If you apply the `vrnode` method to an existing node, the method returns the handle to the `vrnode` object associated with this node.

Method Summary

Method	Description
`delete`	Remove `vrnode` object
`fields`	Virtual world field summary of node object
`isfield`	Returns true if field is name of `vrnode` object field
`get`	Property value of `vrnode` object
`getfield`	Field value of `vrnode` object
`isvalid`	`1` if `vrnode` object is valid, `0` if not
`interpolate`	Interpolate field value of `vrnode` object
`set`	Change property of virtual world node
`setfield`	Change field value of `vrnode` object
`sync`	Enable or disable synchronization of virtual world fields with client
`optimize`	Change geometries to reduce number of vertices

Version History

Introduced before R2006a

See Also

`vrnode/delete` | `vrnode/get` | `vrnode/getfield` | `vrnode/set` | `vrnode/setfield` | `vrworld` | `optimize`

vrnode/delete

Remove `vrnode` object

Syntax

```
delete(vrnode_object)
```

```
delete(n)
```

Arguments

`vrnode_object`	Name of a `vrnode` object.

Description

`delete(vrnode_object)` deletes the virtual world node.

`delete(n)` deletes the `vrnode` object referenced by the `vrnode` handle `n`. If `n` is a vector of `vrnode` handles, multiple nodes are deleted.

As soon as a node is deleted, it and all its child objects are removed from all clients connected to the virtual world.

Version History

Introduced before R2006a

See Also

`vrworld/delete`

vrnode/interpolate

Interpolate field value

Syntax

```
interpolate(Node,FieldName,NewValue)
```

```
interpolate(Node,FieldName, NewValue,Shape)
```

```
interpolate(Node,FieldName,NewValue,Shape, Duration)
```

```
interpolate(Node,FieldName,NewValue,Shape,Duration,FPS)
```

Arguments

`Node`	Node with field, can be a vector.
`FieldName`	String array or cell array of character vectors.
`NewValue`	String array or cell array of character vectors.
`Shape`	Shape of the interpolation curve: • `'sweep'` – Smoothly increasing and decreasing interpolation speed • `'linear'` – constant interpolation speed • `'hill'` – Double-sweep shape with return to the original value • `'triangle'` – Double linear shape with return to the original value
`Duration`	Interpolation duration, in seconds. Default is `1`.
`FPS`	Number of interpolation points per one second of duration time.

Description

`interpolate(Node,FieldName,NewValue)` interpolates any writable numerical field specified by the `FieldName` from its current value to `NewValue`. If `Node` is a vector, then `FieldName` and `NewValue` must be cell arrays of the same size. The software simultaneously interpolates multiple node fields.

`interpolate(Node,FieldName,NewValue,Shape)` specifies the shape of the interpolation curve.

`interpolate(Node,FieldName,NewValue,Shape,Duration)` specifies the interpolation duration.

`interpolate(Node,FieldName,NewValue,Shape,Duration,FPS)` specifies the number of interpolation points per second.

Version History

Introduced in R2023a

See Also

`vrnode/get` | `vrnode/set`

vrnode/isfield

Returns true if field is name of `vrnode` object field

Syntax

```
x = fields(vrnode_object,fieldnames)
```

Arguments

`vrnode_object`	Name of a `vrnode` object representing the node to be queried.
`fieldnames`	String array or cell array of character vectors.

Description

`x = fields(vrnode_object,fieldnames)` returns a logical array, *x*, the same size as that of `fieldnames`. *x* contains true for the elements of `fieldnames` that are the names of fields in the `vrnode` and false otherwise.

Version History

Introduced in R2023a

See Also

`vrnode/get` | `vrnode/set`

vrnode/fields

virtual world field summary of node object

Syntax

```
fields(vrnode_object)
```

```
x = fields(vrnode_object)
```

Arguments

`vrnode_object`	Name of a `vrnode` object representing the node to be queried.

Description

`fields(vrnode_object)` displays a list of fields of the node associated with the `vrnode` object in the MATLAB Command Window.

`x = fields(vrnode_object)` returns the fields of the node associated with the `vrnode` object in a structure array. The resulting structure contains a field for every field with the following subfields:

- `Type` is the name of the field type, for example, `'MFString'`, `'SFColor'`.
- `Access` is the accessibility description of the data class, for example, `'eventIn'`, `'exposedField'`.
- `Sync` is the synchronization status `'on'` or `'off'`. See also `vrnode/sync`.

Version History

Introduced before R2006a

See Also

`vrnode/get` | `vrnode/set`

vrnode/get

Property value of `vrnode` object

Syntax

```
get(vrnode_object)

x = get(vrnode_object)

x = get(vrnode_object, 'property_name')
```

Arguments

`vrnode_object`	Name of a `vrnode` object representing the node to be queried.
`property_name`	Name of the property to be read.

Description

`get(vrnode_object)` lists all `vrnode` properties in the MATLAB Command Window.

`x = get(vrnode_object)`, where `vrnode_object` is a scalar, returns a structure where each field name is the name of a property and each field contains the value of that property.

`x = get(vrnode_object, '`*`property_name`*`')` returns the value of given property.

If `vrnode_object` is a vector of `vrnode` handles, `get` returns an M-by-1 cell array of values, where M is equal to `length(vrnode_object)`.

The `vrnode` property values are case sensitive. Property names are not case sensitive.

The `vrnode` object properties allow you to control the behavior and appearance of objects. The `vrnode` objects have the following properties. All these properties are read only.

Property	Value	Description
`Fields`	Cell array	Valid field names for the node.
`Name`	String	Name of the node.
`Type`	String	Type of the node. The value is a string (for example, `'Transform'`, `'Shape'`).
`World`	Handle	Handle of the parent `vrworld` object. This is a `vrworld` object that represents the node's parent world.

Version History

Introduced before R2006a

See Also

`vrnode` | `vrnode/getfield` | `vrnode/set` | `vrnode/setfield`

vrnode/getfield

Field value of `vrnode` object

Syntax

```
getfield(vrnode_object)

x = getfield(vrnode_object)

x = getfield(vrnode_object,'fieldname')
```

Arguments

`vrnode_object`	Name of a `vrnode` object representing the node to be queried.
`fieldname`	Name of the `vrnode` object field whose values you want to query.

Description

`getfield(vrnode_object)` displays all the field names and their current values for the respective node.

`x = getfield(vrnode_object)`, where `vrnode_object` is a scalar, returns a structure where each field name is the name of a `vrnode` field and each field contains the value of that field.

`x = getfield(vrnode_object,'fieldname')`returns the value of the specified field for the node referenced by the `vrnode_object` handle. If `vrnode_object` is a vector of `vrnode` handles, `getfield` returns an M-by-1 cell array of values, where M is equal to `length(vrnode_object)`.

If `'fieldname'` is a 1-by-N or N-by-1 cell array of strings containing field names, `getfield` returns an M-by-N cell array of values.

Tip Using dot notation is the recommended approach for accessing nodes.

Note For `Transform` nodes, the `getfield` function does not list the Simulink 3D Animation extensions `rotation_abs` and `translation_abs`. To access those fields, use dot notation. For example:

```
gcoords = myWorld.Arm.rotation_abs
```

Version History

Introduced before R2006a

See Also

`vrnode` | `vrnode/get` | `vrnode/set` | `vrnode/setfield`

vrnode/isvalid

`1` if `vrnode` object is valid, `0` if not

Syntax

```
x = isvalid(vrnode_object_vector)
```

Arguments

`vrnode_object_vector`	Name of an array of `vrnode` objects to be queried.

Description

This method returns an array that contains `1` when the elements of `vrnode_object_vector` are valid `vrnode` objects, and `0` when they are not.

The `vrnode` object is considered valid if the following conditions are met:

- The parent world of the node exists.
- The parent world of the node is open.
- The node with the given `vrnode` handle exists in the parent world.

Version History

Introduced before R2006a

See Also

`isvalid` | `vrworld/isvalid`

vrnode/set

Change property of virtual world node

Syntax

```
x = set(vrnode_object, 'property_name','property_value')
```

Arguments

`vrnode_object`	Name of a `vrnode` object representing a node in the virtual world.
`property_name`	Name of a property.
`property_value`	Value of a property.

Description

`x = set(vrnode_object, 'property_name','property_value')` changes the specified property of the `vrnode` object to the specified value.

The `vrnode` property values are case sensitive, while property names are not case sensitive.

The `vrnode` property values are case sensitive, while property names are not case sensitive.

The `vrnode` objects have the following properties. All these properties are read only.

Property	Value	Description
`Fields`	Cell array	Valid field names for the node. Read only.
`Name`	String	Name of the node. Read only.
`Type`	String	Type of the node. The value is a string (for example, `'Transform'`, `'Shape'`). Read only.
`World`	Handle	Handle of the parent `vrworld` object. This is a `vrworld` object that represents the node's parent world. Read only.

Currently, nodes have no settable properties.

Version History

Introduced before R2006a

See Also

`vrnode` | `vrnode/get` | `vrnode/getfield` | `vrnode/setfield`

vrnode/setfield

Change field value of `vrnode` object

Syntax

```
x = setfield(vrnode_object,'fieldname','fieldvalue')
```

Arguments

`vrnode_object`	Name of a `vrnode` object representing the node to be changed.
`fieldname`	Name of the `vrnode` object field whose values you want to set.
`fieldvalue`	Value of `fieldname`.

Description

`x = setfield(vrnode_object,'fieldname','fieldvalue')` changes the specified field of the `vrnode` object to the specified value. You can specify multiple field names and field values in one line of code by grouping them in pairs. For example, `x = setfield(vrnode_object, 'fieldname1', 'fieldvalue1', 'fieldname2', 'fieldvalue2', ...)`.

Note that field names are case sensitive, while property names are not.

Note The dot notation is the preferred method for accessing nodes. For example:

```
vrnode_object.fieldname=fieldvalue;
```

Version History

Introduced before R2006a

See Also

`vrnode` | `vrnode/get` | `vrnode/getfield` | `vrnode/set`

vrnode/sync

Enable or disable synchronization of fields with client

Syntax

```
sync(vrnode_object, 'field_name', 'action')
```

Arguments

`vrnode_object`	Name of a `vrnode` object representing the node.
`field_name`	Name of the field to be synchronized.
`action`	The action parameter determines what should be done: • `'on'` enables synchronization of this field. • `'off'` disables synchronization of this field.

Description

The `sync` method controls whether the value of a field is synchronized.

If you set the field to be synchronized to `'on'`, the field value is updated every time it is changed on the client computer. If you set the field to `'off'`, the host computer ignores the changes on the client computer.

Synchronized fields add more traffic to the network line because the value of the field must be resent by the client any time it is changed. Because of this, mark for synchronization only the fields you need to scan for changes made on clients (typically sensors). By default, fields are not synchronized and their values reflect only settings from MATLAB or the Simulink software.

Note Synchronization is meaningful only for readable fields. Readable fields are of data class `eventOut` and `exposedField`. You cannot enable synchronization for `eventIn` or nonexposed fields.

Version History

Introduced before R2006a

See Also

`vrnode` | `vrnode/get`

vrori2dir

Convert viewpoint orientation to direction

Syntax

```
vrori2dir(r)
vrori2dir(r,options)
```

Description

`vrori2dir(r)` converts the viewpoint orientation, specified by a rotation vector, `r`, to a direction the viewpoint points to.

`vrori2dir(r,options)` converts the viewpoint orientation with the default algorithm parameters replaced by values defined in `options`.

The `options` structure contains the parameter `epsilon` that represents the value below which a number will be treated as zero (default value is 1e-12).

Version History

Introduced in R2007b

See Also

`vrdir2ori` on page 3-30 | `vrrotmat2vec` on page 3-120 | `vrrotvec` | `vrrotvec2mat` on page 3-121

vrpatch2ifs

Convert MATLAB patches to `IndexedFaceSet` nodes

Syntax

```
node = vrpatch2ifs(patches,world)
node = vrpatch2ifs(patches,shape)
node = vrpatch2ifs(patches,parent)
vrpatch2ifs(patches,ifs)
```

Description

`node = vrpatch2ifs(patches,world)` converts the `patches` array and saves the result into the `vrnode` array `node`. Each resulting `IndexedFaceSet` node in `node` is wrapped by the created `Shape` node residing in a root level of the `world` virtual world.

`node = vrpatch2ifs(patches,shape)` converts the `patches` array and saves the result into the `vrnode` array `node`. Each resulting `IndexedFaceSet` node in node is a child of the respective Shape node in the shape array. If the Shape node already contains an `IndexedFaceSet` node, that `IndexedFaceSet` is overwritten. The number of patches must equal the number of `Shape` nodes.

Note This function converts only geometry and color data of the patch.

`node = vrpatch2ifs(patches,parent)` converts the `patches` array and saves the result into the `vrnode` array `node`. Each resulting `IndexedFaceSet` node in `node` is wrapped by the created `Shape` node that is a child of the `parent` node.

`vrpatch2ifs(patches,ifs)` converts the `patches` array and saves the result into `ifs` array of existing `IndexedFaceSet` nodes, overwriting the `IndexedFaceSet` nodes. The number of patches must equal the number of `IndexedFaceSet` nodes.

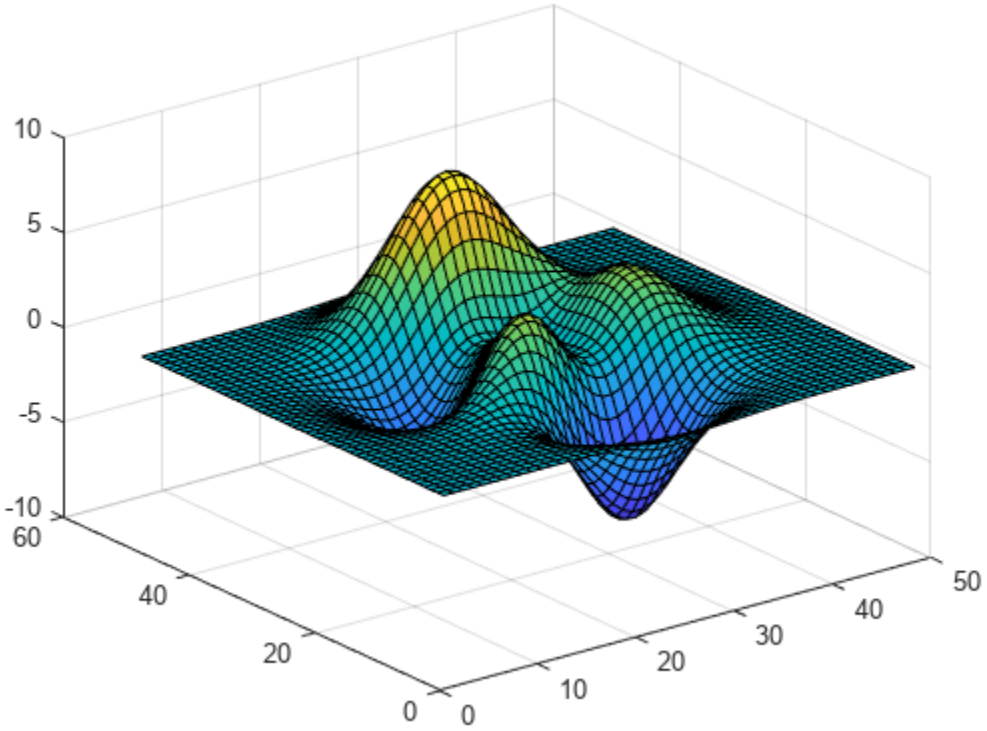
Examples

Convert MATLAB Patches to IndexedFaceSet Nodes

This command converts three MATLAB® patches to `IndexedFaceSet` nodes.

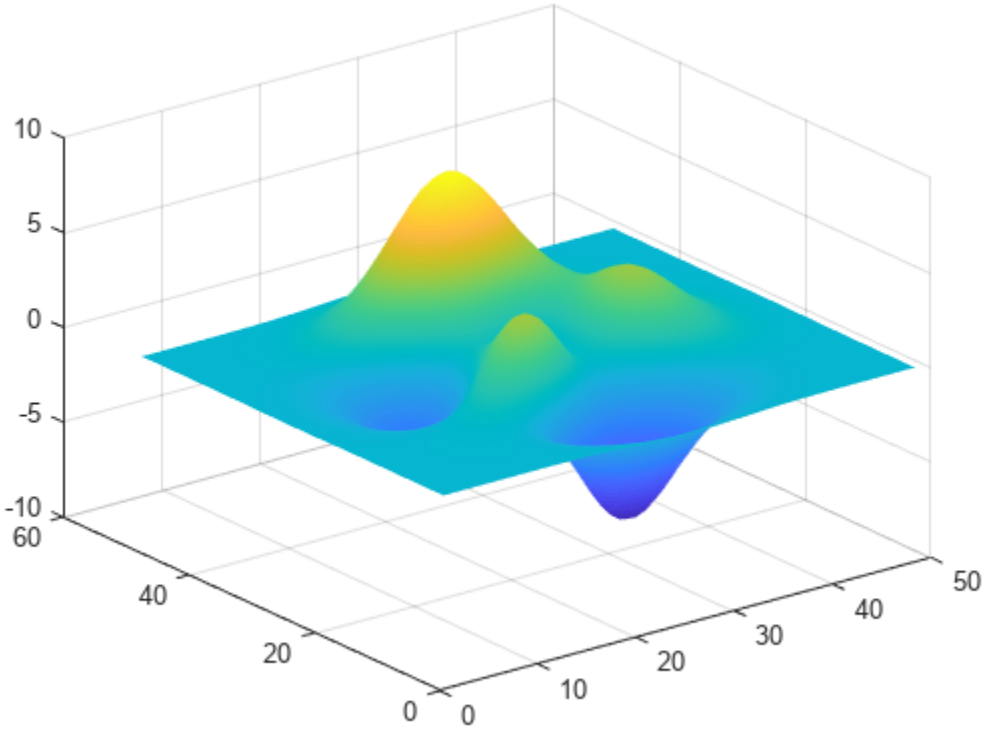
Create surface using MATLAB `peaks` function.

```
fig = figure('Name','Source peaks surface');
s = surf(peaks);
```



Convert the peaks surface to a patch.

```
peaksPatch = patch(surf2patch(s));
delete(s);
shading interp;
```



Create and open an empty virtual world.

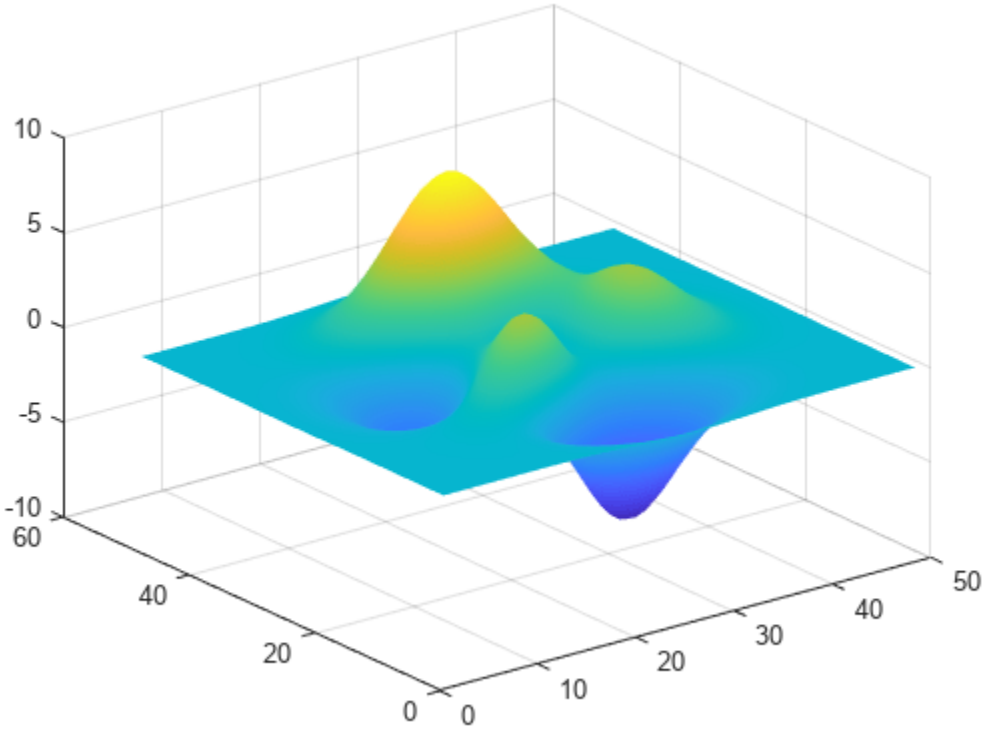
```
w2 = vrworld('');
open(w2);
```

Create and bind viewpoint

```
dv = vrnode(w2, 'DefaultViewpoint','Viewpoint');
dv.position  = [-1 15 30];
dv.orientation = [-0.38 -0.93 0 0.55];
setfield(dv,'set_bind',true); %#ok<STFLD,SFLD>
```

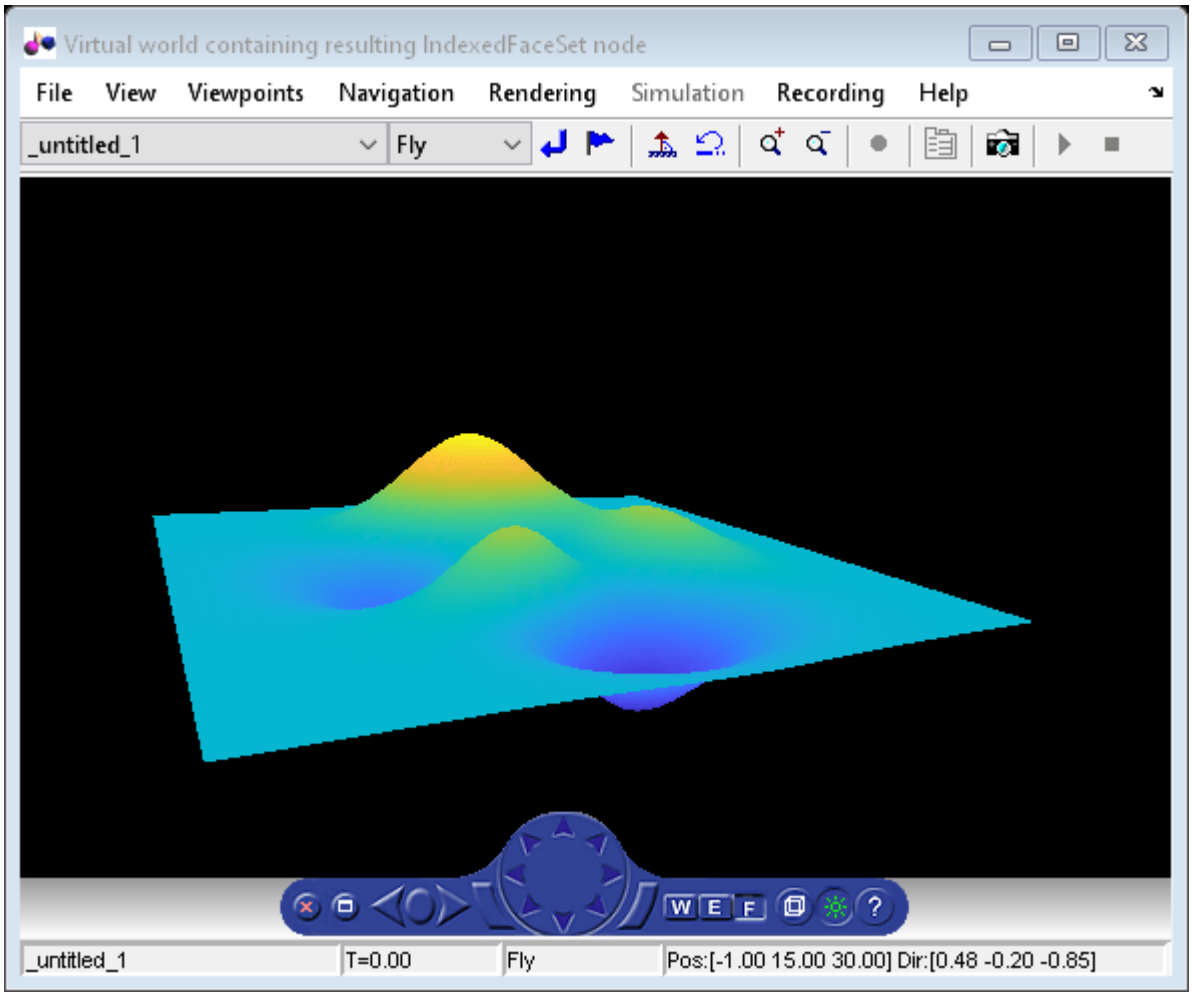
Convert the patch to an `IndexedFaceSet` nodes. The resulting nodes are created in the root level of supplied `vrworld` object)

```
vrpatch2ifs(peaksPatch,w2);
```



Show the result.

```
vrfig2 = vrfigure(w2,'Name',...
         'Virtual world containing resulting IndexedFaceSet node');
```



Input Arguments

`patches` — MATLAB patches to convert
array

MATLAB patches, specified as an array.

`world` — Virtual world that contains Shape nodes
`vrworld` object

Virtual world that contains `Shape` nodes, specified as a `vrworld` object.

`parent` — Parent grouping node
`vrnode` object

Parent grouping node, specified as a `vrnode` object.

`shape` — Shape array
array of `Shape` nodes

Shape array, specified as an array of `Shape` nodes.

`ifs` — `IndexedFaceSet` nodes
array

`IndexedFaceSet` nodes, specified as an array.

Output Arguments

`node` — Conversion result
`vrnode` array

Conversion result, returned as a `vrnode` array.

Version History

Introduced in R2015a

See Also

`vrifs2patch` | `patch`

Topics
“Introduction to Patch Objects”

vrphysmod

Add virtual reality visualization framework to block diagrams

Syntax

```
vrphysmod(virtualWorldFile,system)
```

Description

`vrphysmod(virtualWorldFile,system)` updates the Simulink system (model or subsystem) that the Simscape Multibody `smimport` function generates.

The model must be on the MATLAB path or already open prior to calling the `vrphysmod` function.

You can then save, rename, modify, and run the model. When you save the resulting model, be sure to preserve the relative path between the Simulink system and the virtual world 3D file.

Examples

Update Model

To update the model `four_link` using the file `four_link.wrl`:

```
vrphysmod('four_link.wrl', 'four_link');
```

Update Subsystem

To update the subsystem `four_link/FOURLINK_ASM` using the VRML file `four_link.wrl`, ensure that the model that contains the subsystem is open, then:

```
vrphysmod('four_link.wrl', 'four_link/FOURLINK_ASM');
```

Update Current System

To update the current system using the file `four_link.wrl`:

```
vrphysmod('four_link.wrl', gcs);
```

Input Arguments

`virtualWorldFile` — Virtual world file
`.wrl` file | `.x3d` file | `.x3dv` file

Virtual world file, specified as either `.wrl`, `.x3d` or `.x3dv` files.

The `.wrl` extension is optional for a VRML virtual world file. If the specified system was created with Simscape Multibody First Generation `smimport` function, you can specify also an `.x3d` or `.x3dv` file for the `virtualWorldFile`.

As necessary, `vrphysmod` adds additional blocks to visualize the mechanical system in virtual reality. The association between mechanical system bodies and corresponding nodes found in the virtual world 3D file is based on the name correspondence.

If your model contains several VR Sink blocks that refer to the same `virtualWorldFile`, this function attempts to consolidate the animation signals of that virtual scene into one VR Sink block.

`system` — Model or subsystem to be updated
Simulink model

Model or subsystem to be updated, created by the `smimport` function.

Note The SolidWorks VRML export filter does not preserve part instance names and the part order in the resulting virtual world 3D file. Therefore, the association between such parts and the corresponding bodies in the block diagram is not always an exact match. In such cases, the function identifies nodes with partial matches and issues warnings. To prevent these warnings, ensure that node `DEF` names in the virtual world 3D file are identical to their corresponding bodies in the Simulink model before running this function.

If you receive this warning and the set of virtual world 3D files does not originate in the SolidWorks product, ignore the message. Other supported CAD tools also generate part names with similar names, but preserve them across different export formats.

Version History

Introduced in R2009a

See Also

`stl2vrml` | `vrcadcleanup` | `smimport`

vrplay

Play VRML animation file

Syntax

```
vrplay
vrplay(filename)
x=vrplay(filename)
```

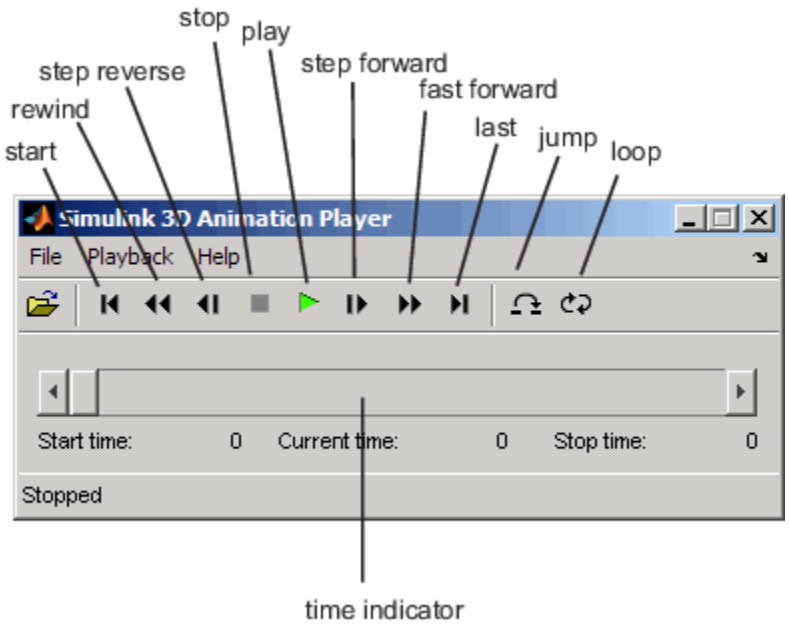
Description

`vrplay` opens the 3D Animation Player, which you use to open and play virtual world animation files.

`vrplay(filename)` opens the 3D Animation Player and loads the virtual world `filename`.

`x=vrplay(filename)` also returns a 3D Animation Player figure handle.

`vrplay` works only with VRML animation files created using the Simulink 3D Animation virtual world recording functionality.



When you create additional `vrplay` windows using the **File > New Window** command, the window respects the current setting of the `DefaultViewer` property. By default, the **File > New Window** command creates the new player window implemented as a MATLAB figure.

Simulink 3D Animation Player App

You can open the Simulink 3D Animation Player from the **Apps** tab in the MATLAB toolstrip as well as the Simulink toolstrip. In the tab, scroll to the **Simulation Graphics and Reporting** section and click **3D Animation Player**.

Keyboard Support

The playback controls can also be accessed from the keyboard.

Key	Function
F, Page Down	Fast forward
J	Jump to time
L	Loop
P	Play/pause toggle
S	Stop
R, Page Up	Rewind
Right arrow key	Step forward
Left arrow key	Step reverse
Up arrow key	First
Down arrow key	Last

Examples

To play the animation file based on the `vr_octavia` example, run `vrplay('octavia_scene_anim.wrl')`.

Version History

Introduced in R2006a

See Also

`vrview`

Topics

“Record Offline Animations”

vrrotvec

Calculate rotation between two vectors

Syntax

```
r = vrrotvec(a,b)
r = vrrotvec(a,b,options)
```

Description

`r = vrrotvec(a,b)` calculates a rotation needed to transform the 3D vector `a` to the 3D vector `b`.

`r = vrrotvec(a,b,options)` calculates the rotation with the default algorithm parameters replaced by values defined in `options`.

Input Arguments

a, b — 3-D vector

vector

3-D vectors between which rotation is being calculated.

Data Types: `single` | `double`

options — Structure containing epsilon

structure

Structure containing the parameter `epsilon` that represents the value below which a number will be treated as zero (default value is 1e-12). Default value of `epsilon` is `1e-12`.

Data Types: `struct`

Output Arguments

r — axis angle rotation vector

row vector

Axis-angle rotation, returned as a four element row vector. The first three elements specify the rotation axis, and the last element defines the angle of rotation.

Version History

Introduced in R2007b

See Also

`vrrotmat2vec` on page 3-120 | `vrrotvec2mat` on page 3-121

vrrotmat2vec

Convert rotation from matrix to axis-angle representation

Syntax

```
r = vrrotmat2vec(m)
r = vrrotmat2vec(m,options)
```

Description

`r = vrrotmat2vec(m)` returns an axis-angle representation of rotation defined by the rotation matrix `m`.

`r = vrrotmat2vec(m,options)` converts the rotation with the default algorithm parameters replaced by values defined in `options`.

The `options` structure contains the parameter `epsilon` that represents the value below which a number will be treated as zero (default value is 1e-12).

The result `r` is a four-element axis-angle rotation row vector. The first three elements specify the rotation axis, and the last element defines the angle of rotation.

Version History

Introduced in R2007b

See Also

`vrrotvec` | `vrrotvec2mat` on page 3-121

vrrotvec2mat

Convert rotation from axis-angle to matrix representation

Syntax

```
m = vrrotvec2mat(r)
m = vrrotvec2mat(r,options)
```

Description

`m = vrrotvec2mat(r)` returns a matrix representation of the rotation defined by the axis-angle rotation vector, `r`.

`m = vrrotvec2mat(r,options)` returns a matrix representation of rotation defined by the axis-angle rotation vector `r`, with the default algorithm parameters replaced by values defined in `options`.

The `options` structure contains the parameter `epsilon` that represents the value below which a number will be treated as zero (default value is 1e-12).

The rotation vector, `r`, is a row vector of four elements, where the first three elements specify the rotation axis, and the last element defines the angle.

To rotate a column vector of three elements, multiply it by the rotation matrix. To rotate a row vector of three elements, multiply it by the transposed rotation matrix.

Version History

Introduced in R2007b

See Also

`vrrotvec` | `vrrotmat2vec` on page 3-120

vrsetpref

Change Simulink 3D Animation preferences

Syntax

```
vrsetpref('preference_name', 'preference_value')
```

```
vrsetpref('factory')
```

Arguments

preference_name	Name of the preference.
preference_value	New value of the preference.

Description

This function sets the given Simulink 3D Animation preference to a given value. The following preferences are defined. For preferences that begin with the string `DefaultFigure` or `DefaultWorld`, these values are the default values for the corresponding `vrfigure` or `vrworld` property:

Preference	Description
`AutoCreateThumbnail`	Creates a thumbnail of a virtual world when you open a virtual world. The default is `'off'`. Setting this preference to `'on'` can be helpful if you download multiple virtual worlds from the Internet, without saving them. Creating thumbnails on file open provides thumbnails the next time someone browses through the downloaded worlds.
`DataTypeBool`	Specifies the handling of the virtual world `Bool` data type for `vrnode/setfield` and `vrnode/getfield`. Valid values are `'logical'` and `'char'`. If set to `'logical'`, the virtual world `Bool` data type is returned as a logical value. If set to `'char'`, the `Bool` data type is returned `'on'` or `'off'`. Default is `'logical'`.
`DataTypeInt32`	Specifies handling of the virtual world `Int32` data type for `vrnode/setfield` and `vrnode/getfield`. Valid values are `'int32'` and `'double'`. If set to `'int32'`, the virtual world `Int32` data type is returned as `int32`. If set to `'double'`, the `Int32` data type is returned as `'double'`. Default is `'double'`.

Preference	Description
`DataTypeFloat`	Specifies the handling of the virtual world `float` data type for `vrnode/setfield` and `vrnode/getfield`. Valid values are `'single'` and `'double'`. If set to `'single'`, the virtual world `Float` and `Color` data types are returned as `'single'`. If set to `'double'`, the `Float` and `Color` data types are returned as `'double'`. Default is `'double'`.
`DefaultCanvasNavPanel`	Controls the appearance of the control panel in the `vr.canvas` object. Values are: • `'none'` Panel is not visible. • `'minimized'` Panel appears as a minimized icon in the right-hand corner of the viewer. • `'translucent'` Panel floats half transparently above the scene. • `'opaque'` Panel floats above the scene. Default: `'none'`
`DefaultCanvasUnits`	Specifies default units for new `vr.canvas`objects. See `vr.canvas` for detailed description. Default is `'normalized'`.
`DefaultEditorMouseBehavior`	Specifies whether the mouse in the view pane is in navigation mode or selection mode (for highlighting corresponding nodes in the tree view pane). The default is `'navigate'`. To make selection mode the default, set the preference to `'select'`.
`DefaultEditorHighlighting`	Specifies whether to highlight virtual world objects selected in the view pane. The default is `'on'`. To avoid highlighting selected virtual objects by default, set the preference to `'off'`.
`DefaultFigureAnti Aliasing`	Determines whether antialiasing is used by default for new `vrfigure` objects. This preference also applies to new `vr.canvas`objects. Valid values are `'off'` and `'on'`.
`DefaultFigureCapture FileName`	Specifies default file name for capturing viewer figures. See `get` for detailed description. Default is `'%f_anim_ %n.tif'`.
`DefaultFigureDeleteFcn`	Specifies the default callback invoked when closing a `vrfigure` object.

Preference	Description
DefaultFigureLighting	Specifies whether the lights are rendered by default for new vrfigure objects. This preference also applies to new vr.canvas objects. Valid values are 'off' and 'on'.
DefaultFigureMax TextureSize	Specifies the default maximum size of a texture used in rendering new vrfigure objects. This preference also applies to new vr.canvas objects. Valid values are 'auto' and 32 <= x <= video card limit, where x is a power of 2.
DefaultFigureNavPanel	Specifies the default appearance of the control panel in the viewer. Valid values are 'opaque', 'translucent', 'none', 'halfbar', 'bar', and 'factory'. Default is 'halfbar'.
DefaultFigureNavZones	Specifies whether the navigation zone is on or off by default for new vrfigure objects. This preference also applies to new vr.canvasobjects. Valid values are 'off' and 'on'.
DefaultFigurePosition	Sets the default initial position and size of the Simulink 3D Animation Viewer window. Valid value is a vector of four doubles.
DefaultFigureRecord2D CompressMethod	Specifies the default compression method for creating 2-D animation files for new vrfigure objects. Valid values are '', 'auto', 'lossless', and 'codec_code'.
DefaultFigureRecord2D CompressQuality	Specifies the default quality of 2-D animation file compression for new vrfigure objects. Valid values are 0-100.
DefaultFigureRecord2D FileName	Specifies the default 2-D offline animation file name for new vrfigure objects.
DefaultFigureRecord2DFPS	Specifies the default frames per second playback speed. To have the 2D AVI animation play back at approximately the same playback speed as the 3D virtual world animation, set this preference to auto.
DefaultFigureRendering	Specifies whether to render a vrfigure or vr.canvas object. Turning off rendering improves performance. For example, if your code does batch operations on a virtual figure, you can turn off rendering during that processing and then turn it back on after the processing.
DefaultFigureStatusBar	Specifies whether the status bar appears by default at the bottom of the Simulink 3D Animation Viewer for new vrfigure objects. Valid values are 'off' and 'on'.
DefaultFigureTextures	Specifies whether textures should be rendered by default for new vrfigure objects. This preference also applies to new vr.canvas objects. See get for detailed description. Default is 'on'.

Preference	Description
`DefaultFigureToolBar`	Specifies whether the toolbar appears by default on the Simulink 3D Animation Viewer for new `vrfigure` objects. Valid values are `'off'` and `'on'`.
`DefaultFigure Transparency`	Specifies whether or not transparency information is taken into account when rendering for new `vrfigure` objects. This preference also applies to new `vr.canvas` objects. Valid values are `'off'` and `'on'`.
`DefaultFigureWireframe`	Specifies whether objects are drawn as solids or wireframes by default for new `vrfigure` objects. This preference also applies to new `vr.canvas` objects. Valid values are `'off'` and `'on'`.
`DefaultViewer`	Specifies which viewer is used to view a virtual scene. • `'internal'` Default Simulink 3D Animation Viewer. • `'web'` Web browser becomes viewer. This is the current Web browser virtual world plug-in.
`DefaultWorldRecord3D FileName`	Specifies the default 3-D animation file name for new `vrworld` objects.
`DefaultWorldRecordMode`	Specifies the default animation recording mode for new `vrworld` objects. Valid values are `'manual'` and `'scheduled'`.
`DefaultWorldRecord Interval`	Specifies the default start and stop times for scheduled animation recording for new `vrworld` objects. Valid value is a vector of two doubles.
`DefaultWorldRemoteView`	Specifies whether the virtual world is enabled by default for remote viewing for new `vrworld` objects. Valid values are `'off'` and `'on'`.
`DefaultWorldTimeSource`	Specifies the default source of the time for new `vrworld` objects. Valid values are `'external'` and `'freerun'`.

Preference	Description
`Editor`	Specifies which virtual world editor to use. Path to the virtual world editor. If this path is empty, the MATLAB editor is used. The path setting is active only if you select the `Custom` option. To open a virtual world file in a third-party editor, do not use the `vredit` command. For example, to open a virtual world in the Ligos V-Realm Builder editor: **1** Set the default editor to V-Realm Builder. In MATLAB, enter: `vrsetpref('Editor','*VREALM');` **2** To open a file in the V-Realm editor, in MATLAB navigate to a virtual world file, right-click, and select **Edit**. **Note** The `vredit` command opens the 3D World Editor, regardless of the default editor preference setting.
`EditorPreserveLayout`	Specifies whether the 3D World Editor starts up with a saved version of the layout of a virtual world when you exited it or reverts to the default layout. The layout of the virtual world display pane includes settings for the view, viewpoints, navigation, and rendering. Valid values are `'off'` and `'on'`. The default is on (use saved layout).
`HttpPort`	IP port number used to access the Simulink 3D Animation server over the Web via HTTP. If you change this preference, you must restart the MATLAB software before the change takes effect.
`TransportBuffer`	Length of the transport buffer (network packet overlay) for communication between the Simulink 3D Animation server and its clients.
`TransportTimeout`	Amount of time the Simulink 3D Animation server waits for a reply from the client. If there is no response from the client, the Simulink 3D Animation server disconnects from the client.
`VrPort`	IP port used for communication between the Simulink 3D Animation server and its clients. If you change this preference, you must restart the MATLAB software before the change takes effect.

When you use `'factory'` as a single argument, all preferences are reset to their default values. If you use `'factory'` for a preference value, that single preference is reset to its default.

The `HttpPort`, `VrPort`, and `TransportBuffer` preferences affect Web-based viewing of virtual worlds. `DefaultFigurePosition` and `DefaultNavPanel` affect the Simulink 3D Animation Viewer. Changes to the `HttpPort` or `VrPort` preferences take effect only after you restart the MATLAB software.

`DefaultFigureNavPanel` — Controls the appearance of the navigation panel in the Simulink 3D Animation Viewer. For example, setting this value to `'translucent'` causes the navigation panel to appear translucent.

`DefaultViewer` — Determines whether the virtual scene appears in the default Simulink 3D Animation Viewer or in your Web browser.

`DefaultViewer` Setting	**Description**
`'internal'`	Default Simulink 3D Animation Viewer.
`'web'`	Viewer is the default Web browser with the virtual world plug-in.

`Editor` — Contains a path to the virtual world editor executable file. When you use the `edit` command, Simulink 3D Animation runs the virtual world editor executable with all parameters required to edit the virtual world file.

When you run the editor, Simulink 3D Animation uses the `Editor` preference value as if you typed it into a command line. The following tokens are interpreted:

`%matlabroot`	Refers to the MATLAB root folder
`%file`	Refers to the virtual world 3D file name

For instance, a possible value for the `Editor` preference is

```
`%matlabroot\bin\win64\meditor.exe %file'
```

If this preference is empty, the MATLAB editor is used.

`HttpPort` -- Specifies the network port to be used for remote Web access. The port is given in the Web URL as follows:

```
http://server.name:port_number
```

The default value of this preference is `8123`.

`TransportBuffer` — Defines the size of the message window for client-server communication. This value determines how many messages, at a maximum, can travel between the client and the server at one time.

Generally, higher values for this preference make the animation run more smoothly, but with longer reaction times. (More messages in the line create a buffer that compensates for the unbalanced delays of the network transfer.)

The default value is `5`, which is optimal for most purposes. You should change this value only if the animation is significantly distorted or the reaction times are very slow. On fast connections, where delays are introduced more by the client rendering speed, this value has very little effect. Viewing on a host computer is equivalent to an extremely fast connection. On slow connections, the correct value can improve the rendering speed significantly but, of course, the absolute maximum is determined by the maximum connection throughput.

`VrPort` — Specifies the network port to use for communication between the Simulink 3D Animation server (host computer) and its clients (client computers). Normally, this communication is completely invisible to the user. However, if you view a virtual world from a client computer, you might need to configure the security network system (firewall) so that it allows connections on this port. The default value of this preference is `8124`.

Version History

Introduced before R2006a

See Also

`vrgetpref`

vrspacemouse

Create space mouse object

Syntax

```
mouse = vrspacemouse(id)
```

Description

`mouse = vrspacemouse(id)` creates a space mouse object capable of interfacing with a space mouse input device. The `id` parameter is a string that specifies the space mouse connection: `COM1`, `COM2`, `COM3`, `COM4`, `USB1`, `USB2`, `USB3`, or `USB4`.

The `vrspacemouse` object has several properties that influence the behavior of the space mouse input device. The properties can be read or modified using dot notation (e.g., `mouse.DominantMode = true;`).

Properties

Valid properties are (property names are case-sensitive):

Property	Description
`PositionSensitivity`	Mouse sensitivity for translations. Higher values correspond to higher sensitivity.
`RotationSensitivity`	Mouse sensitivity for rotations. Higher values correspond to higher sensitivity.
`DisableRotation`	Fixes the rotations at initial values, allowing you to change positions only.
`DisableTranslation`	Fixes the positions at the initial values, allowing you to change rotations only.
`DominantMode`	If this property is true, the mouse accepts only the prevailing movement and rotation and ignores the others. This mode is very useful for beginners using a space mouse.
`UpperPositionLimit`	Position coordinates for the upper limit of the mouse.
`LimitPosition`	Enables mouse position limits. If false, the object ignores the `UpperPositionLimit` and `LowerPositionLimit` properties.
`LowerPositionLimit`	Position coordinates for the lower limit of the mouse.
`NormalizeOutputAngle`	Determines whether the integrated rotation angles should wrap on a full circle (360°). This is not used when you read the `Output Type` as `Speed`.
`InitialPosition`	Initial condition for integrated translations. This is not used when you set the `Output Type` to `Speed`.

Property	Description
`InitialRotation`	Initial condition for integrated rotations. This is not used when you set the `Output Type` to `Speed`.

Methods

Method	Description
`button`	`b = button(mouse, n)` reads the status of space mouse button number `n`. Button status is returned as logical 0 if not pressed and logical 1 if pressed. `n` can be a vector to return multiple buttons.
`close`	`close(mouse)` closes and invalidates the space mouse object. The object cannot be used once it is closed.
`position`	`p = position(mouse, n)` reads the position of space mouse axis number `n`. `n` can be a vector to return positions of multiple axes. Translations and rotations are integrated. Outputs are the position and orientation in the form of roll/pitch/yaw angles.
`speed`	`s = speed(mouse, n)` reads the speed of space mouse axis number `n`. `n` can be a vector to return the speeds of multiple axes. No transformations are done. Outputs are the translation and rotation speeds.
`viewpoint`	`p = viewpoint(mouse)` reads the space mouse coordinates in virtual world viewpoint format. Translations and rotations are integrated. Outputs are the position and orientation in the form of an axis and an angle. You can use these values as viewpoint coordinates in virtual world.

Version History

Introduced in R2007b

See Also

“Set Simulink 3D Animation Preferences”

vr.utils.stereo3d class

Stereoscopic vision settings for `vr.canvas` and `vr.figure` objects

Description

Tip Use the `vr.utils.stereo3d` class for advanced tuning of stereoscopic viewer and canvas properties. You can select and use basic stereoscopic settings from the Viewer menu.

Specifies these stereoscopic vision properties:

- Active, anaglyph, or no stereoscopic vision
- Camera offset
- Camera angle
- Color filter for the left and right cameras
- Horizontal image translation (HIT)

Use a `vr.utils.stereo3d` object to set the `Stereo3D`, `Stereo3DCameraOffset`, and `Stereo3DHIT` stereoscopic vision properties of `vrfigure` and `vr.canvas` objects. Specifying a `vr.utils.stereo3d` object to set one `vrfigure` and `vr.canvas` property also sets the other stereoscopic vision properties. Using a `vr.utils.stereo3d` object also specifies color filters for the left and right cameras. You cannot set camera color filters directly using the `vrfigure/set` method or `vr.canvas` properties.

Construction

`stereoVision = vr.utils.stereo3d.OFF` disables stereoscopic vision.

`stereoVision = vr.utils.stereo3d.ACTIVE` enables active stereoscopic vision.

`stereoVision = vr.utils.stereo3d.ANAGLYPH` enables red-cyan anaglyph stereoscopic vision.

`stereoVision = vr.utils.stereo3d.RED_CYAN` enables red-cyan anaglyph stereoscopic vision.

`stereoVision = vr.utils.stereo3d.ANAGLYPH_GREEN_MAGENTA` enables green-magenta anaglyph stereoscopic vision.

`stereoVision = vr.utils.stereo3d.ANAGLYPH_RED_GREEN` enables red-green anaglyph stereoscopic vision.

`stereoVision = vr.utils.stereo3d.ANAGLYPH_RED_BLUE` enables red-blue anaglyph stereoscopic vision.

`stereoVision = vr.utils.stereo3d.ANAGLYPH_YELLOW_BLUE` enables yellow-blue anaglyph stereoscopic vision.

Output Arguments

`stereoVision` — Stereoscopic vision settings for `vr.canvas` and `vrfigure` objects
`vr.utils.stereo3d` object

Stereoscopic vision settings for `vr.canvas` and `vrfigure` objects, represented by a `vr.utils.stereo3d` object.

Properties

`CameraAngle` — Camera angle
`vr.utils.stereo3D.DEFAULT_CAMERA_ANGLE` | radians

Camera angle, specified using the predefined `DEFAULT_CAMERA_ANGLE` or in radians. This property is in effect when you enable stereoscopic vision.

This property does not apply to `vr.canvas` or `vrfigure` objects.

`CameraOffset` — Camera offset
`0.1` (default) | floating-point number between 0 and 1

Camera offset, specified as a number representing the distance in virtual world units of left/right camera from parallax. The parallax is the difference in the apparent position of an object viewed from two cameras.

This property sets the `Stereo3DCameraOffset` property of a `vr.canvas` or `vrfigure` object.

`HIT` — Horizontal image translation
predefined `DEFAULT_HIT` | floating-point number

Horizontal image translation, specified as either the predefined `DEFAULT_HIT` or as a floating-point number from 0 through 1, inclusive. The number of pixels for stereo 3D horizontal image translation (HIT) derives from this number. Horizontal image translation is the horizontal relationship of the two stereo images. By default, the background image is at zero and the foreground image appears to pop out from the monitor toward the person viewing the virtual world. The larger the value, the further back the background appears to be.

This property sets the `Stereo3DHIT` property of a `vr.canvas` or `vrfigure` object.

`LeftCameraFilter` — Color filter of left camera
row vector of nine floating-point numbers | predefined filter

Color filter of the left camera, specified as a row vector of nine floating-point numbers or using a predefined filter.

If you specify a row vector, use floating-point numbers from 0 through 1. The first three numbers represent the red value, the second three numbers represent the green value, and the last three numbers represent the blue value. For example, specifying `1` for the first three numbers and zeros for the other numbers produces a pure red filter.

The predefined filters are:

- `CAMERA_FILTER_FULL`
- `CAMERA_FILTER_RED`

- CAMERA_FILTER_CYAN
- CAMERA_FILTER_GREEN
- CAMERA_FILTER_MAGENTA
- CAMERA_FILTER_YELLOW
- CAMERA_FILTER_BLUE

This property specifies the left camera filter for `vr.canvas` or `vrfigure` objects.

Example: `stereo3d_object.LeftCameraFilter = [0.1 0.5 0.5 0.0 0.0 0.0 1.0 0.5 0.5];`

Example: `stereo3d_object.LeftCameraFilter = stereo3d_object.CAMERA_FILTER_RED`

Mode — Stereoscopic vision mode
read only

Stereoscopic vision mode. Read only.

- STEREO3D_OFF — No stereoscopic vision.
- STEREO3D_ACTIVE — Active stereoscopic vision. Stereoscopic vision uses quad-buffered rendering. You can use a graphics card driver to output stereoscopic vision. This mode allows active stereoscopic vision via shutter glasses.
- STEREO3D_ANAGLYPH — Anaglyph stereoscopic vision. Stereoscopic vision is enabled using red-cyan anaglyph. Use appropriate anaglyph 3D glasses to see the effect.

This property sets the `Stereo3D` property of a `vr.canvas` or `vrfigure` object.

RightCameraFilter — Color filter of right camera
row vector of nine floating-point numbers | predefined filter

Color filter of the right camera, specified as a row vector of nine floating-point numbers or using a predefined filter.

If you specify a row vector, use floating-point numbers from 0 through 1. The first three numbers represent the red value, the second three numbers represent the green value, and the last three numbers represent the blue value. For example, specifying 1 for the first three numbers and zeros for the other numbers produces a pure red filter.

The predefined filters are:

- CAMERA_FILTER_FULL
- CAMERA_FILTER_RED
- CAMERA_FILTER_CYAN
- CAMERA_FILTER_GREEN
- CAMERA_FILTER_MAGENTA
- CAMERA_FILTER_YELLOW
- CAMERA_FILTER_BLUE

This property specifies the right camera filter for `vr.canvas` or `vrfigure` objects.

Example: `stereo3d_object.RightCameraFilter = [0.1 0.5 0.5 0.0 0.0 0.0 1.0 0.5 0.5];`

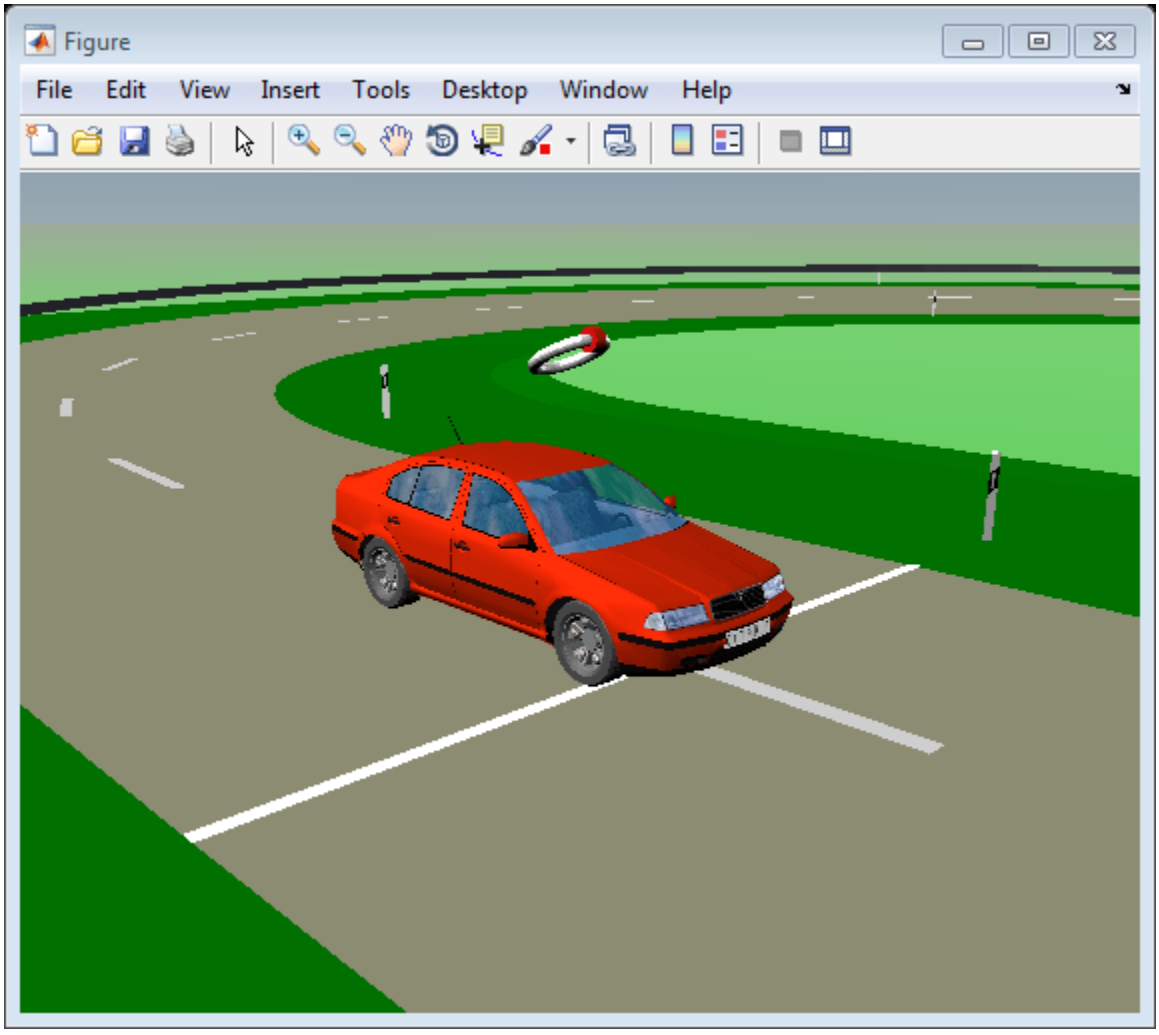
Example: `stereo3d_object.RightCameraFilter = stereo3d_object.CAMERA_FILTER_RED`

Examples

Define and Apply Stereoscopic Vision Settings

Create a virtual world.

```
w = vrworld('octavia_scene');
open(w);
c = vr.canvas(w);
```



Specify stereoscopic vision settings.

```
s3d = vr.utils.stereo3d.ANAGLYPH_RED_CYAN;
s3d.CameraOffset = 0.05;
s3d.CameraAngle = pi/128;
```

Modify the red component of filter for the left camera.

```
s3d.LeftCameraFilter(1:3) = s3d.LeftCameraFilter(1:3)...
                          + [0.1 -0.05 -0.05];
```

Apply stereoscopic vision settings of `vr.utils.stereo3d` object `s3d` to `vr.canvas` object `c`.

```
set(c,'Stereo3D',s3d)
```

Version History

Introduced in R2015a

See Also

`vr.canvas` | `vrfigure`

Topics

“View a Virtual World in Stereoscopic Vision”

vrupdaterobot

Update RigidBodyTree robot pose

Syntax

```
vrupdaterobot(RBT, tforms, config)
```

Description

`vrupdaterobot(RBT, tforms, config)` updates the robot pose from its current configuration using the `config` argument.

Examples

Add Robot to World and Change its Pose

This example shows you how to insert a robot into a virtual world and update its pose

Import the Robot and Setup the World

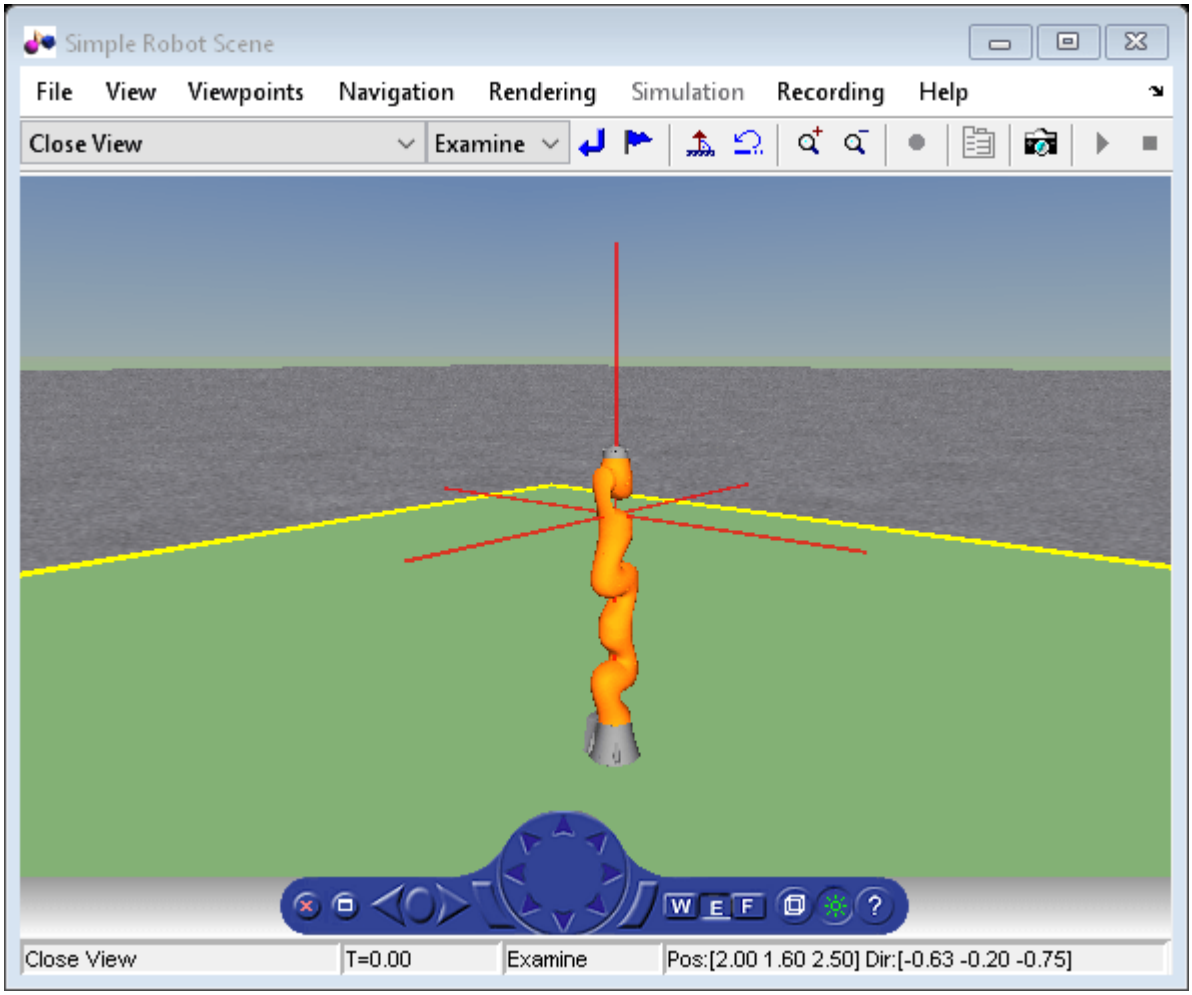
Import the KUKA LFR iiwa robot from its URDF definition and insert it to the virtual world created from `robot_scene.x3d.`

```
RBT = importrobot('iiwa7.urdf');
RBT.DataFormat = 'row';
robotWorld = vrworld('robot_scene');
open(robotWorld);
```

Get Transforms of Current Pose of the Robot

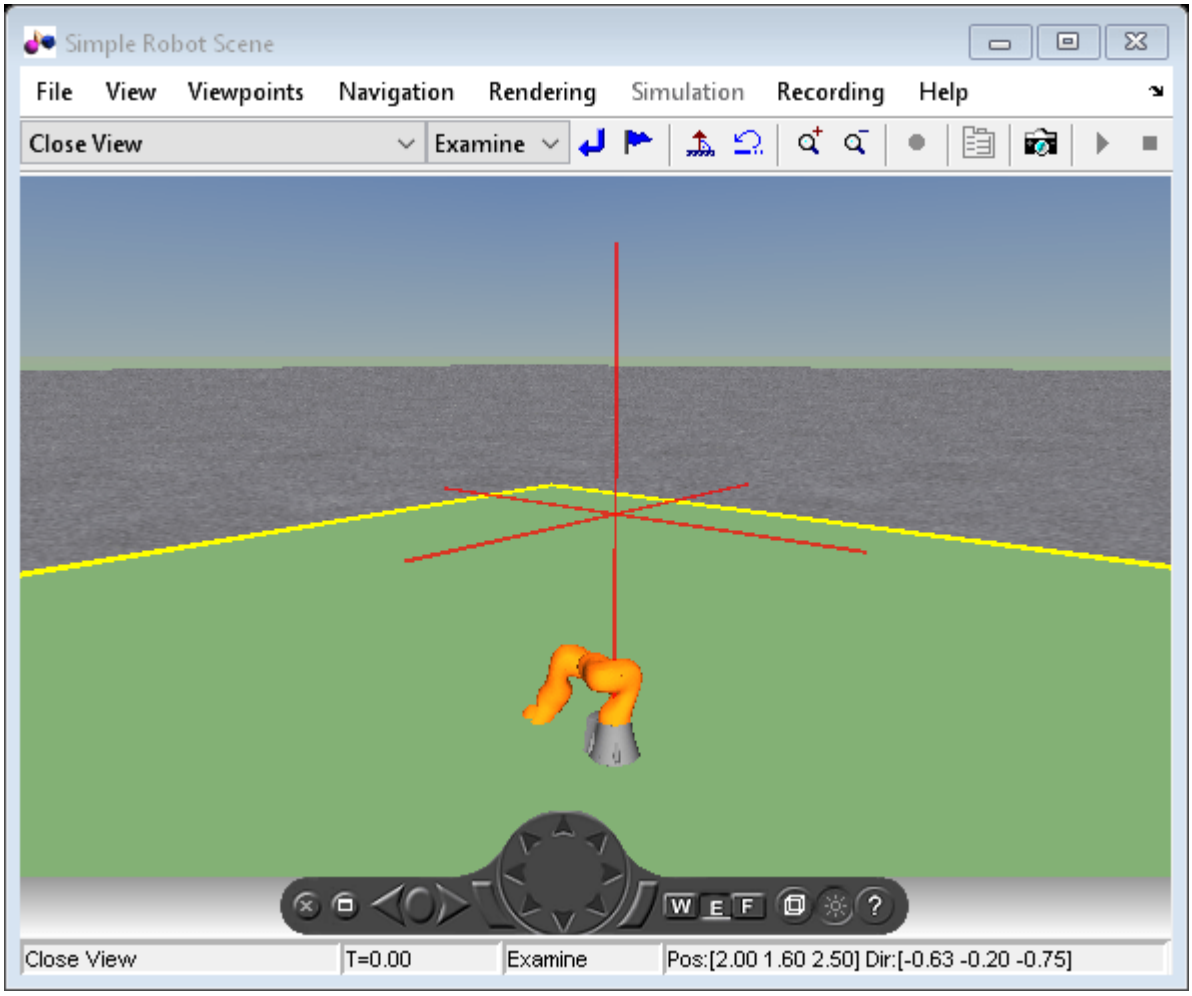
The `tforms` output argument contains a list of transforms that describe the robot pose in its initial or 'home' configuration.

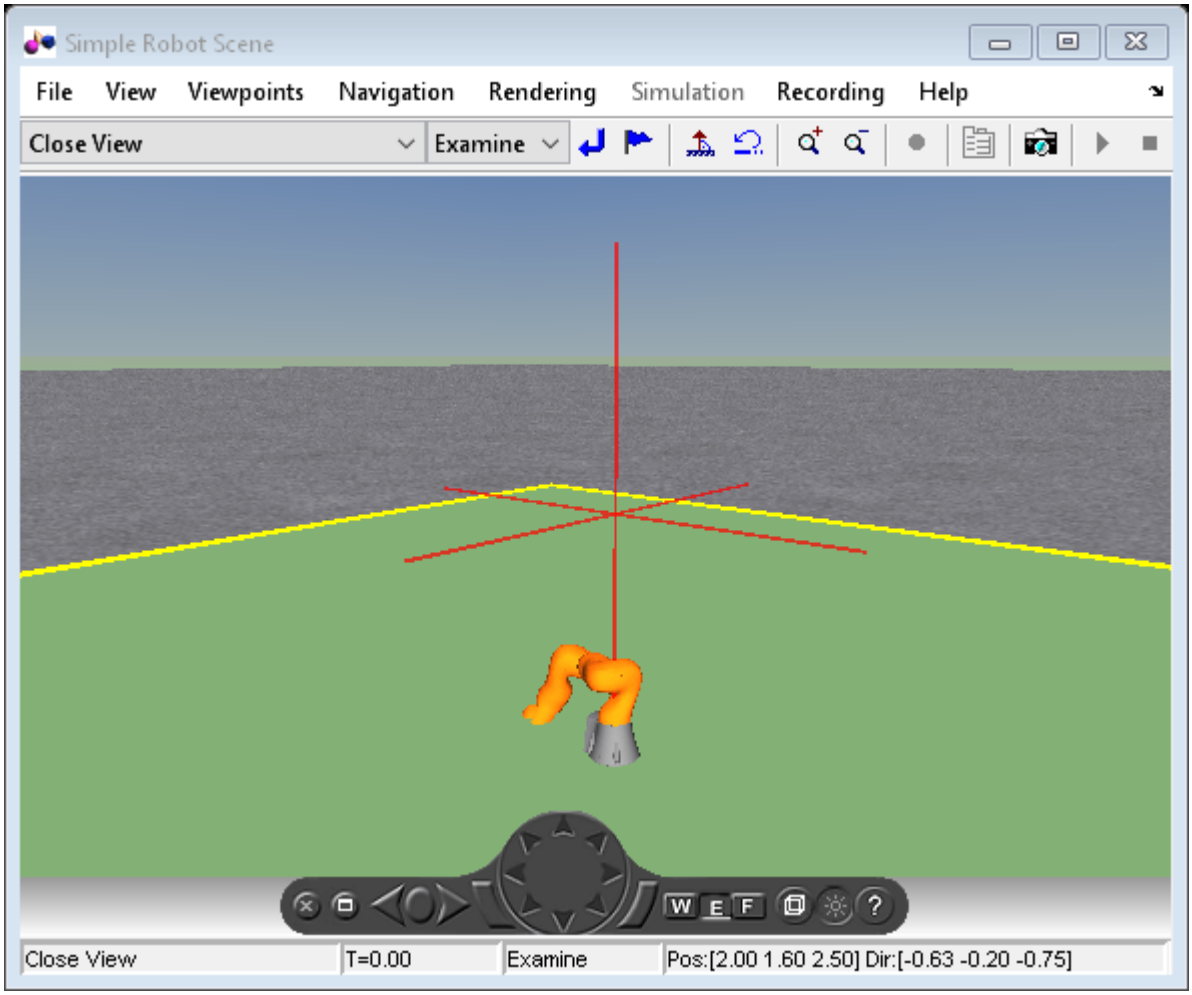
```
[node, W, tforms] = vrinsertrobot(robotWorld, RBT);
vrfigure(robotWorld);
```



Change the Pose of the Robot

```
vrupdaterobot(RBT, tforms, randomConfiguration(RBT));
vrdrawnow;
vrfigure(robotWorld);
```


Simple Robot Scene
File
View
Viewpoints
Navigation
Rendering
Simulation
Recording
Help
Close View
Examine
W
E
F
Close View
T=0.00
Examine
Pos:[2.00 1.60 2.50] Dir:[-0.63 -0.20 -0.75]



Input Arguments

RBT — Robot description

`rigidBodyTree` object

Robotics System Toolbox `rigidBodyTree` object. For more information, see `rigidBodyTree` (Robotics System Toolbox).

`tforms` — Robot transforms

cell array

List of robot transforms, specified as a cell array of `vrnode` objects.

`config` — Desired end configuration

structure | vector

Desired pose of the robot, specified in the same format as the `RBT.DataFormat` field of the `rigidBodyTree` object.

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

Version History

Introduced in R2018b

See Also

`vrinsertrobot` | `rigidBodyTree` (Robotics System Toolbox)

vrview

View virtual world using Simulink 3D Animation viewer or Web browser

Syntax

```
vrview

x = vrview('filename')

x = vrview('filename','-internal')

x = vrview('filename','-web')
```

Description

`vrview` opens the default Web browser and loads the Simulink 3D Animation software Web page containing a list of virtual worlds available for viewing.

`x = vrview('filename')` creates a virtual world associated with the `.wrl` file, opens the virtual world, and displays it in the Simulink 3D Animation Viewer or the Web browser depending on the value of the `DefaultViewer` preference. The handle to the virtual world is returned.

`x = vrview('filename','-internal')` creates a virtual world associated with the `wrl` file, opens the virtual world, and displays it in the Simulink 3D Animation Viewer.

`x = vrview('filename','-web')` creates a virtual world associated with the `.wrl` file, opens the virtual world, and displays it in your Web browser.

`vrview('filename#viewpointname')` specifies a default viewpoint.

Version History

Introduced before R2006a

See Also

`vrplay` | `vrworld` | `vrworld/open` | `vrworld/view`

vrwho

List virtual worlds in memory

Syntax

```
vrwho
```

```
x = vrwho
```

Description

If you do not specify an output parameter, `vrwho` displays a list of virtual worlds in memory in the MATLAB Command Window.

If you specify an output parameter, `vrwho` returns a vector of handles to existing `vrworld` objects, including those opened from the Simulink interface.

Version History

Introduced before R2006a

See Also

`vrclear` | `vrwhos` | `vrworld`

vrwhos

List details about virtual worlds in memory

Syntax

```
vrwhos
```

Description

`vrwhos` displays a list of virtual worlds currently in memory, with a description, in the MATLAB Command Window. The relation between `vrwho` and `vrwhos` is similar to the relation between `who` and `whos`.

Version History

Introduced before R2006a

See Also

`vrclear` | `vrwho`

vrworld

Create new `vrworld` object associated with virtual world

Syntax

```
myworld = vrworld(filename)

myworld = vrworld(filename,'reuse')

myworld = vrworld(filename, 'new')

myworld = vrworld

myworld = vrworld('')

myworld = vrworld([])
```

Arguments

`filename`	String containing the name of the virtual world 3D file from which the virtual world is loaded. You can specify `.wrl`, `.x3d`, or `.x3dv`). If no file extension is specified, the file extension `.wrl` is assumed.
`'new'`	Argument to create a virtual world associated with `filename`.

Description

`myworld = vrworld(filename)` creates a virtual world associated with the virtual world 3D file `filename` and returns its handle. If the virtual world already exists, a handle to the existing virtual world is returned. Specify the file name as a string.

`myworld = vrworld(filename,'reuse')` has the same functionality as `myworld = vrworld(filename)`.

`myworld = vrworld('filename', 'new')` creates a virtual world associated with the virtual world 3D file `filename` and returns its handle. It always creates a new virtual world object, even if another `vrworld` object associated with the same file already exists.

`myworld = vrworld` creates an invalid `vrworld` handle

`myworld = vrworld('')` creates an empty `vrworld` object that is not associated with any virtual world 3D file

`myworld = vrworld([])` returns an empty array of returns an empty array of `vrworld` handles.

A `vrworld` object identifies a virtual world in a way very similar to a handle. All functions that affect virtual worlds accept a `vrworld` object as an argument to identify the virtual world.

If the given virtual world already exists in memory, the handle to the existing virtual world is returned. A second virtual world is not loaded into memory. If the virtual world does not exist in memory, it is loaded from the associated virtual world 3D file. The newly loaded virtual world is closed and must be opened before you can use it.

The `vrworld` object associated with a virtual world remains valid until you use either `delete` or `vrclear`.

Examples

```
myworld = vrworld('vrpend.wrl')
```

Method Summary

Method	Description
`addexternproto`	Add `externproto` declaration to virtual world.
`close`	Close virtual world
`delete`	Remove virtual world from memory
`edit`	Open virtual world file in external virtual world editor
`get`	Property value of `vrworld` object
`isvalid`	`1` if `vrworld` object is valid, `0` if not
`nodes`	List nodes available in virtual world
`open`	Open virtual world
`reload`	Reload virtual world from virtual world 3D file
`save`	Write virtual world to virtual world 3D file
`set`	Change property values of `vrworld` object
`view`	View virtual world
`optimize`	Change geometries to reduce number of vertices

Version History

Introduced before R2006a

See Also

`vrworld/close` | `vrworld/delete` | `vrworld/open` | “Create vrworld Object for a Virtual World” | “Open a Virtual World with MATLAB”

vrworld/addexternproto

Add `externproto` declaration to virtual world

Syntax

```
addexternproto(vrworld_object, protofile, protoname)

addexternproto(vrworld_object, protofile, protoname, protodef)
```

Arguments

`vrworld_object`	A `vrworld` object representing the virtual world.
`protofile`	String containing the name of the prototype file from which the `EXTERNPROTO` declaration is added.
`protoname`	String containing the name of the `EXTERNPROTO` declaration.
`protodef`	String containing a new name for the `EXTERNPROTO` declaration.

Description

`addexternproto(vrworld_object, protofile, protoname)` adds an `EXTERNPROTO` declaration from file `protofile` to the virtual world. The handle `vrworld_object` refers to the virtual world. The `EXTERNPROTO` declaration is identified as `protoname`. If `protoname` is a cell array of identifiers, the function adds multiple `EXTERNPROTO`s from one file to the virtual world.

`addexternproto(vrworld_object, protofile, protoname, protodef)` adds an `EXTERNPROTO` declaration from file `protofile` to the virtual world. The handle `vrworld_object` refers to the virtual world. The `EXTERNPROTO` declaration is identified as `protoname`. If `protoname` is a cell array of identifiers, the function adds multiple `EXTERNPROTO`s from one file to the virtual world. This command then renames the new `EXTERNPROTO` declaration to `protodef`.

In both cases, the `EXTERNPROTO` declaration becomes equivalent to the `PROTO` declaration. In other words, `protoname` or `protodef` becomes an internal `PROTO` type in the virtual scene associated with `vrworld_object`. After you save the virtual world, these `PROTO` declarations no longer require a reference to the original file, `protofile`, that contains the `EXTERNPROTO` declarations.

Version History

Introduced in R2008b

See Also

`vrworld/close` | `vrworld/delete` | `vrworld/open` | “Create vrworld Object for a Virtual World” | “Open a Virtual World with MATLAB”

vrworld/close

Close virtual world

Syntax

```
close(vrworld_object)
```

Arguments

`vrworld_object`	A `vrworld` object representing the virtual world.

Description

This method changes the virtual world from an opened to a closed state:

- If the world was opened more than once, you must use an appropriate number of `close` calls before the virtual world closes.
- If `vrworld_object` is a vector of `vrworld` objects, all associated virtual worlds close.
- If the virtual world is already closed, `close` does nothing.

Opening and closing virtual worlds is a mechanism of memory management. When the system needs more memory and the virtual world is closed, you can discard its contents at any time.

Generally, you should close a virtual world when you no longer need it. This allows you to reuse the memory it occupied. The `vrworld` objects associated with this virtual world stay valid after it is closed, so the virtual world can be opened again without creating a new `vrworld` object.

Examples

```
myworld = vrworld('vrpend.wrl')
open(myworld)
close(myworld)
```

Version History

Introduced before R2006a

See Also

`vrworld` | `vrworld/delete` | `vrworld/open` | “Close and Delete a vrworld Object”

vrworld/delete

Remove virtual world from memory

Syntax

```
delete(vrworld_object)
```

Arguments

`vrworld_object`	A `vrworld` object representing a virtual world.

Description

The `delete` method removes from memory the virtual world associated with a `vrworld` object. The virtual world must be closed before you can delete it.

Deleting a virtual world frees the virtual world from memory and invalidates all existing `vrworld` objects associated with the virtual world.

If `vrworld_object` is a vector of `vrworld` objects, all associated virtual worlds are deleted.

You do not commonly use this method. One of the possible reasons to use this method is to ensure that a large virtual world is removed from memory before another memory-consuming operation starts.

Version History

Introduced before R2006a

See Also

`vrworld` | `vrclear` | `vrworld/open` | “Close and Delete a vrworld Object”

vrworld/edit

Open virtual world file in virtual world editor

Syntax

```
edit(vrworld_object)
```

Arguments

`vrworld_object`	A `vrworld` object representing a virtual world.

Description

The `edit` method opens the virtual world 3D file associated with the `vrworld` object in a virtual world editor. The `Editor` preference specifies the editor to use. See `vrsetpref` for details on setting preferences.

The editor saves any changes you make directly to a virtual world file. If the virtual world is open,

- Use the `save` command in the virtual world editor to save the changes to a virtual world file. In the MATLAB interface, the changes appear after you reload the virtual world.
- Use the `save` method in the MATLAB software to replace the modified virtual world 3D file. Any changes you made in the editor are lost.

Version History

Introduced before R2006a

See Also

`vrworld/reload` | `vrworld/save` | `vrworld/delete` | `vrworld/open` | “Close and Delete a vrworld Object” | “Create vrworld Object for a Virtual World”

vrworld/get

Property value of `vrworld` object

Syntax

```
get(vrworld_object)

x = get(vrworld_object)

x = get(vrworld_object, 'property_name')
```

Arguments

`vrworld_object`	A `vrworld` object representing a virtual world.
`property_name`	Name of the property.

Description

`get(vrworld_object)` displays all the virtual world properties and their values.

`x = get(vrworld_object)` returns an M-by-1 structure where the field names are the names of the virtual world properties. Each field contains the associated property value. M is equal to `length(vrworld_object)`.

`x = get(vrworld_object, '`*`property_name`*`')` returns the value of the specified property.

- If `vrworld_object` is a vector of `vrworld` handles, the `get` method returns an M-by-1 cell array of values where M is equal to `length(vrworld_object)`.
- If *`property_name`* is a 1-by-N or N-by-1 cell array of strings containing field names, the `get` method returns an M-by-N cell array of values.

The following are properties of `vrworld` objects. Names are not case sensitive.

Property	Value	Description
`Clients`	Scalar	Number of clients currently viewing the virtual world. Read only.
`ClientUpdates`	`'off'` | `'on'` Default: `'on'`	Client cannot or can update the virtual scene. Read/write.
`Canvases`	Vector of canvases	Vector of handles of canvases currently open for this world.
`Description`	String. Default: automatically taken from the virtual world 3D file property `title`	Description of the virtual world as it appears on the main Web page. Read/write.

Property	Value	Description
`Figures`	Vector of `vrfigure` objects	Vector of handles to Simulink 3D Animation Viewer windows currently viewing the virtual world. Read only.
`FileName`	String	Name of the associated virtual world 3D file. Read only.
`Nodes`	Vector of `vrnode` objects	Vector of `vrnode` objects for all named nodes in the virtual world. Read only.
`Open`	`'off' | 'on'` Default: `'off'`	Indicates a closed or open virtual world. Read only.
`Record3D`	`'off' |'on'` Default: `'off'`	Enables 3-D animation recording. Read/write.
`Record3DFileName`	String. Default: `'%f_anim_%n.wrl'`	3-D animation file name. The string can contain tokens that are replaced by the corresponding information when the animation recording takes place. For details, see “File Name Tokens”. Read/write.
`Recording`	`'off' | 'on'` Default: `'off'`	Animation recording toggle. This property acts as the main recording switch. Read/write.
`RecordMode`	`'manual' | 'scheduled'` Default: `'manual'`	Animation recording mode. Read/write.
`RecordInterval`	Vector of two doubles Default: `[0 0]`	Start and stop times for scheduled animation recording. Corresponds to the virtual world object `Time` property. Read/write.
`RemoteView`	`'off' | 'on'` Default: `'off'`	Remote access flag. If the virtual world is enabled for remote viewing, it is set to `'on'`; otherwise, it is set to `'off'`. Read/write.
`Rendering`	`'off' | 'on'` Default: `'on'`	Render `vrworld` object in the Simulink 3D Animation Viewer, specifying `'on'` or `'off'`. Turning off rendering improves performance. For example, if your code does batch operations on a virtual world, you can turn off rendering during that processing and then turn it back on after the processing.
`Time`	Double	Current time in the virtual world. Read/write.

Property	Value	Description
`TimeSource`	`'external' | 'freerun'` Default: `'external'`	Source of the time for the virtual world. If set to `'external'`, time in the scene is controlled from the MATLAB interface (by setting the `Time` property) or the MATLAB interface (simulation time). If set to `'freerun'`, time in the scene advances independently based on the system timer. Read/write.
`View`	`'off' | 'on'` Default: `'on'`	Indicates an unviewable or viewable virtual world. Read/write.
`Viewpoints`	Vector of `vrnode` objects	Vector of `vrnode` objects representing viewpoints defined in the virtual world. Read only.

The `ClientUpdates` property is set to `'on'` by default and can be set by the user. When it is set to `'off'`, the viewers looking at this virtual world should not update the view according to the virtual world changes. That is, the view is frozen until this property is changed to `'on'`. This is useful for preventing tearing effects with complex animations. Before every animation frame, set `ClientUpdates` to `'off'`, make the appropriate modifications to the object positions, and then switch `ClientUpdates` back to `'on'`.

The `Description` property defaults to `'(untitled)'` and can be set by the user. If the virtual world is loaded from a virtual world 3D file containing a **WorldInfo** node with a `title` property, the `Description` property is loaded from the virtual world 3D file instead.

The `Nodes` property is valid only when the virtual world is open. If the virtual world is closed, `Nodes` always contains an empty vector.

The `RemoteView` property is set to `'off'` by default and can be set by the user. If it is set to `'on'`, all viewers can access the virtual world through the Web interface. If it is set to `'off'`, only host viewers can access it.

The `View` property is set to `'on'` by default and can be set by the user. When it is set to `'off'`, the virtual world is not accessible by the viewer. You rarely use this property.

Version History

Introduced before R2006a

See Also

`vrworld` | `vrworld/set`

vrworld/isvalid

`1` if `vrworld` object is valid, `0` if not

Syntax

```
x = isvalid(vrworld_object)
```

Arguments

`vrworld_object`	A `vrworld` object representing a virtual world.

Description

A `vrworld` object is considered valid if its associated virtual world still exists.

`x = isvalid(vrworld_object)` returns an array that contains a `1` when the elements of `vrworld_object` are valid `vrworld` objects, and returns a `0` when they are not.

You use this method to check whether the `vrworld` object is still valid. Using a `delete` or `vrclear` command can make a `vrworld` object invalid.

Version History

Introduced before R2006a

See Also

`isvalid` | `vrnode/isvalid`

vrworld/nodes

List nodes available in virtual world

Syntax

```
nodes(vrworld_object, '-full')
```

```
x = nodes(vrworld_object, '-full')
```

Arguments

`vrworld_object`	A `vrworld` object representing a virtual world.
`'-full'`	Optional switch to obtain a detailed list of nodes and fields.

Description

If you give an output argument, the method `nodes` returns a cell array of the names of all available nodes in the world. If you do not give an output argument, the list of nodes is displayed in the MATLAB window.

You can use the `'-full'` switch to obtain a detailed list that contains not only the nodes, but also all their fields. This switch affects only the output to the MATLAB Command Window.

The virtual world must be open for you to use this method.

Version History

Introduced before R2006a

See Also

`vrworld` | `vrworld/open` | `vrnode`

vrworld/open

Open virtual world

Syntax

```
open(vrworld_object)
```

Arguments

`vrworld_object`	A `vrworld` object representing a virtual world.

Description

The `open` method opens the virtual world. When the virtual world is opened for the first time, the virtual world internal representation is created based on the associated virtual world 3D file.

If the input argument is an array of virtual world handles, all the virtual worlds associated with those handles are opened.

The virtual world must be open for you to use it. You can close the virtual world with the method `close`.

You can call the method `open` more than once, but you must use an appropriate number of `close` calls before the virtual world returns to a closed state.

Examples

Create two `vrworld` objects by typing

```
myworld1 = vrworld('vrmount.wrl')
myworld2 = vrworld('vrpend.wrl')
```

Next, create an array of virtual world handles by typing

```
myworlds = [myworld1 myworld2];
```

`open(myworlds)` opens both of these virtual worlds.

Version History

Introduced before R2006a

See Also

`vrworld` | `vrworld/close` | “Open a Virtual World with MATLAB”

vrworld/reload

Reload virtual world from virtual world 3D file

Syntax

```
reload(vrworld_object)
```

Arguments

`vrworld_object`	A `vrworld` object representing a virtual world.

Description

The `reload` method reloads the virtual world from the virtual world 3D file associated with the `vrworld` object. If the input argument is an array of virtual world handles, all the virtual worlds associated with those handles are reloaded. The virtual world must be open for you to use this method.

`reload` forces all the clients currently viewing the virtual world to reload it. This is useful when there are changes to the virtual world 3D file.

Version History

Introduced before R2006a

See Also

`vrworld/edit` | `vrworld/open` | `vrworld/save` | “Open a Virtual World with MATLAB”

vrworld/save

Write virtual world to virtual world 3D file

Syntax

```
save(vrworld_object,file)

save(vrworld_object,file,'-export')

save(vrworld_object,file,'-nothumbnail')

save(vrworld_object,file,'-export','-nothumbnail')
```

Arguments

`vrworld_object`	`vrworld` object representing a virtual world
`file`	Name of virtual world 3D file, specified as a string. You can specify a `.wrl` (VRML), `.x3dv` (XML encoded) or `.x3d` (X3D in classic or XML format) file.
`'-export'`	Saves a complete copy of the virtual world, including all resources used by the world, located relative to the exported virtual world location. Resources include virtual world elements such as textures and resources from the Simulink 3D Editor library. This option supports using a Simulink 3D Animation virtual world outside of Simulink 3D Animation.
`'-nothumbnail'`	Suppress creating a thumbnail image used for virtual world preview.

Description

The `save` method saves the current virtual world to a VRML97 file or X3D file, based on the file extension (`.wrl` , `.x3dv`, or `.x3d`) that you specify. The virtual world must be open for you to use this method.

If the virtual world is associated to a VRML file, it can be saved to the VRML or X3D file formats. If the virtual world is associated to an X3D file, it can be saved only to one of the X3D file formats.

If you specify a VRML file, the resulting file is a VRML97 compliant UTF-8 encoded text file.

Lines are indented using spaces. Line ends are encoded as LF on all platforms to ensure cross-platform compatibility.

You can use the optional `'-export'` and `'-nothumbnail'` arguments either by themselves or together, in addition to the required `vrworld_object` and `file` arguments.

Version History

Introduced before R2006a

See Also

`vrworld/edit` | `vrworld/open` | `vrworld/reload` | “Close and Delete a vrworld Object”

vrworld/set

Change property values of `vrworld` object

Syntax

```
set(vrworld_object, 'property_name', property_value)
```

Arguments

`vrworld_object`	Name of a `vrworld` object representing a virtual world.
`property_name`	Name of the property.
`property_value`	New value of the property.

Description

You can change the values of the read/write virtual world properties. The following are properties of `vrworld` objects. Names are not case sensitive.

Property	Value	Description
`Clients`	Scalar	Number of clients currently viewing the virtual world. Read only.
`ClientUpdates`	`'off'` | `'on'` Default: `'on'`	Client cannot or can update the virtual scene. Read/write.
`Description`	String. Default: automatically taken from the virtual world 3D file property `title`	Description of the virtual world as it appears on the main Web page. Read/write.
`Figures`	Vector of `vrfigure` objects	Vector of handles to Simulink 3D Animation viewer windows currently viewing the virtual world. Read only.
`FileName`	String	Name of the associated virtual world 3D file. Read only.
`Nodes`	Vector of `vrnode` objects	Vector of `vrnode` objects for all named nodes in the virtual world. Read only.
`Open`	`'off' | 'on'` Default: `'off'`	Indicates a closed or open virtual world. Read only.
`Record3D`	`'off'` | `'on'` Default: `'off'`	Enables 3-D animation recording. Read/write.

Property	Value	Description
`Record3DFileName`	String. Default: `'%f_anim_%n.%e'`	3D animation file name. The string can contain tokens that are replaced by the corresponding information when the animation recording takes place. For details, see “File Name Tokens”. Read/write.
`Recording`	`'off' |'on'` Default: `'off'`	Animation recording toggle. This property acts as the main recording switch. Read/write.
`RecordMode`	`'manual' | 'scheduled'` Default: `'manual'`	Animation recording mode. Read/write.
`RecordInterval`	Vector of two doubles Default: `[0 0]`	Start and stop times for scheduled animation recording. Corresponds to the virtual world object `Time` property. Read/write.
`RemoteView`	`'off' | 'on'` Default: `'off'`	Remote access flag. If the virtual world is enabled for remote viewing, it is set to `'on'`; otherwise, it is set to `'off'`. Read/write.
`Time`	Double	Current time in the virtual world. Read/write.
`TimeSource`	`'external' | 'freerun'` Default: `'external'`	Source of the time for the virtual world. If set to `'external'`, time in the scene is controlled from the MATLAB interface (by setting the `Time` property) or the Simulink interface (simulation time). If set to `'freerun'`, time in the scene advances independently based on the system timer. Read/write.
`View`	`'off'| 'on'` Default: `'on'`	Indicates an unviewable or viewable virtual world. Read/write.

Version History

Introduced before R2006a

See Also

`vrworld` | `vrworld/get`

vrworld/view

View virtual world

Syntax

```
view(vrworld_object)

x = view(vrworld_object)

x = view(vrworld_object,'-internal')

x = view(vrworld_object,'-web')
```

Arguments

`vrworld_object`	A `vrworld` object representing a virtual world.

Description

The `view` method opens the default virtual world viewer on the host computer and loads the virtual world associated with the `vrworld` object into the viewer window. You specify the default virtual world viewer using the `DefaultViewer` preference. The virtual world must be open for you to use this method.

x = view(`vrworld_object`) opens the default virtual world viewer on the host computer and loads the virtual world associated with the `vrworld` object into the viewer window. If the Simulink 3D Animation Viewer is used, `view` also returns the `vrfigure` handle of the viewer window. If a Web browser is used, `view` returns an empty array of `vrfigure` handles.

`x = view(vrworld_object,'-internal')` opens the virtual world in the Simulink 3D Animation Viewer.

x = view(`vrworld_object`,'**`-web`**') opens the virtual world in the Web browser.

If the virtual world is disabled for viewing (that is, the `View` property for the associated `vrworld` object is set to `'off'`), the `view` method does nothing.

Examples

```
myworld = vrworld('vrpend.x3d')
open(myworld)
view(myworld)
```

Version History

Introduced before R2006a

See Also

`vrview` | `vrworld`

sim3d.World

Object used to define virtual reality world in Unreal Engine viewer

Description

Use the `sim3d.World` object to create and define virtual reality worlds and run the cosimulation using Unreal Engine.

- Add or remove custom actors to or from the world. Only the actors added to the world object are displayed.
- Store the virtual reality scene, the root `sim3d.Actor` object, and the hierarchical structure of all actors in the scene.
- Optionally control the simulation logic by specifying custom functions. The output function passes data to the Unreal Engine to change actor properties in the engine.
- Control the cosimulation between MATLAB and the Unreal Engine.
- Define the virtual world global coordinate system.
- Control scene environment settings.
- Control virtual world timing.

Creation

Syntax

```
world = sim3d.World()
world = sim3d.World('Name',name)
world = sim3d.World('RenderOffScreen',true)
world = sim3d.World('Setup',@setupFcn)
world = sim3d.World('Update',@updateFcn)
world = sim3d.World('Output',@outputFcn)
world = sim3d.World('Release',@releaseFcn)
```

Description

`world = sim3d.World()` launches the default Unreal Engine executable in a new window with a default name.

`world = sim3d.World('Name',name)` launches the Unreal Engine executable in a new window with the name specified.

`world = sim3d.World('RenderOffScreen',true)` runs the cosimulation while running the Unreal Engine executable in the background.

`world = sim3d.World('Setup',@setupFcn)` modifies the cosimulation by executing `@setupFcn` before setup. This custom function can be used to import additional data which required during simulation runtime.

`world = sim3d.World('Update',@updateFcn)` modifies the cosimulation by executing `@updateFcn` at each simulation step. This custom function can be used to read data from the Unreal Engine.

`world = sim3d.World('Output',@outputFcn)` modifies the cosimulation by executing `@outputFcn` at each simulation step. This custom function can be used to send data about the specified `sim3d.Actor` object to the Unreal Engine.

`world = sim3d.World('Release',@releaseFcn)` modifies the cosimulation by executing `@releaseFcn` at each simulation step. This function can be used to delete additional data you may import during setup.

Input Arguments

Name-Value Pair Arguments

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

Example: `World = sim3d.World('Name',world1)`

`Name` — Name of world
string

Name of the world, specified as a string.

`ExecutablePath` — Path to executable file
string

Path to executable file that the world runs, specified as a string.

`Map` — Path to Unreal Engine map
string

Path to Unreal Engine map that is loaded when the executable file runs, specified as a string.

`RenderOffScreen` — Option to run simulation in background
`0` (`false`) | `1` (`true`)

Option to run simulation in the background, specified as either `0`(`false`) or `1`(`true`).

`Update` — Custom update function
handle to user-defined function

Custom update function that reads data about specified actors from Unreal Engine, specified as the handle to the user-defined function.

`Output` — Custom output function
handle to user-defined function

Custom output function that returns information, such as transform and color, for the specified actors to the Unreal Engine, specified as the handle to the user-defined function. This allows Unreal to control the simulation.

`Setup` — Custom setup function
handle to user-defined function

Custom function that reads to run additional commands during setup, specified as the handle to the user defined function. This can be used to load data required for simulation.

`Release` — Custom release function
handle to user-defined function

Custom release function that runs additional commands after simulation ends, specified as the handle to the user-defined function. This setting can be used to delete additional data imported for the setup.

Properties

`'Name'` — Name of world
string

Name of the world, specified as a string.

`Actors` — All actors in world
structure

All actors in the world, specified as a structure whose fields are the names of the actors.

`UserData` — Data specified by user
structure

Data specified by user that may be required during simulation runtime, specified as a structure.

You can use `UserData` to store the data needed by the `Update`, `Output`, `Setup`, and `Release` custom functions. `UserData` ensures that the same data is accessible to all these functions.

`Viewports` — Perspective of main camera
empty structure (default) | structure

Perspective of the main camera, defined as a structure containing a single field, `Main`. `Main` contains a `sim3d.sensors.MainCamera` object with these properties:

- `Parent` — Parent of actor object
- `Children` — Children of actor object
- `ParentWorld` — Handle to parent world
- `SensorIdentifier` — Unique ID of the main camera, with data type `uint32`
- `Translation` — Translation of the main camera relative to the parent vehicle, in meters
- `Rotation` — Rotation of the sensor relative to the parent vehicle, in radians

`HitActors` — Event container
empty array (default) | array

Event container for the handles for actors hit at a time step, specified as an array. After each time step, the event container resets to empty.

`BeginOverlappedActors` — Event container
empty array (default) | array

Event container for the handles for actors that start overlapping at a time step, specified as an array. After each time step, the event container resets to empty.

`EndOverlappedActors` — Event container
empty array (default) | array

Event container for the handles for actors that end overlapping at a time step, specified as an array. After each time step, the event container resets to empty.

`ClickedActors` — Event container
empty array (default) | array

Event container for the handles for actors that are interactively clicked at a time step, specified as an array. After each time step, the event container resets to empty.

Object Functions

createViewport	Create viewport for world
add	Add actor to virtual reality world
run	Run cosimulation in virtual reality world
remove	Remove actor added to world or remove all actors in world

Examples

Create World and Actor

This example shows how to use the `sim3d.World` and `sim3d.Actor` objects to create an actor in a world scene to display in the Simulation 3D Viewer window. This process does not build an actor object appearance, preventing the Simulation 3D Viewer from rendering a visualization of the actor object. For an example building an appearance for an actor, see “Build Actor from Mesh Data and Apply Texture”.

Create World

Create a world scene.

```
world = sim3d.World();
```

Create Actor

Add an actor to the world.

```
add(world,sim3d.Actor());
```

Run Simulation

Run a simulation set for 10 seconds with a sample time of 0.02 seconds.

```
run(world,0.02,10)
```

Delete World

Delete the world object.

```
delete(world);
```

Version History

Introduced in R2022b

R2023a: Event Attributes

Use `sim3d.World` object properties to read event information, including:

- **HitActors**: Actors hit at a time step.
- **BeginOverlappedActors**: Actors that starts overlapping at a time step.
- **EndOverlappedActors**: Actors that ends overlapping at a time step.
- **ClickedActors**: Actors that are interactively clicked at a time step.

See Also

Classes
`sim3d.Actor` | `sim3d.sensors.IdealCamera` | `sim3d.sensors.MainCamera`

Blocks
Simulation 3D Actor Transform Get | Simulation 3D Actor Transform Set | Simulation 3D Camera Get | Simulation 3D Scene Configuration

takeSnapshot

Take snapshot of selected properties

Syntax

```
takeSnapshot(actor)
takeSnapshot(actor,Name)
takeSnapshot(actor,Name,Properties)
takeSnapshot(actor)
takeSnapshot(actor,Name)
takeSnapshot(actor,Name,Properties)
takeSnapshot(actor,Name,Properties,IncludeChildren)
```

Description

`takeSnapshot(actor)` takes a snapshot of the actor specified by `actor`.

`takeSnapshot(actor,Name)` takes a snapshot of the `actor` called `Name`.

`takeSnapshot(actor,Name,Properties)` takes a snapshot of the `actor` called `Name` by storing the values of the properties specified by `Properties`.

`takeSnapshot(actor)` takes a snapshot of `actor`.

`takeSnapshot(actor,Name)` takes a snapshot of `actor` called `Name`.

`takeSnapshot(actor,Name,Properties)` takes a snapshot of the `actor` called `Name` by storing the values of the properties contained in `Properties`.

`takeSnapshot(actor,Name,Properties,IncludeChildren)` takes a snapshot of `actor` by storing the values of `Properties` mentioned.

Input Arguments

`actor` — Actor class whose property is returned

`sim3d.Actor` object

Actor class whose property is returned, specified as a `sim3d.Actor` object.

`Name` — Name of snapshot

string | character array

Name of snapshot, specified as a string or character array. This name can be used later to retrieve the snapshot. If no name is specified, a default name is generated.

`Properties` — Names of properties whose values are returned

`{'Translation', 'Rotation'}` (default) | cell array of `sim3d.Actor` object properties

Names of properties whose values are returned from `actor`, specified a cell array of Properties of the `sim3d.Actor` object properties.

`Properties` should be a cell array of strings with the list of all properties to be included in the snapshot.

`IncludeChildren` — Whether to include children of actor
`true` or `1` (default) | `false` or `0`

Whether to include children of in search, specified as either `true` (`1`) or `false` (`0`).

- `true` – Include all children.
- `false` – Do not include children.

Version History

Introduced in R2022b

See Also

`copy` | `findBy` | `propagate` | `gather` | `restoreSnapshot`

restoreSnapshot

Restore actor to state of properties saved in specified snapshot

Syntax

```
restoreSnapshot(actor)
restoreSnapshot(actor,SnapID)
```

Description

`restoreSnapshot(actor)` restores properties of the actor specified by `actor` to the values stored in the snapshot. If `IncludeChildren` is set to `true` while taking the snapshot, all the properties of the children of the actor restore to their original state.

`restoreSnapshot(actor,SnapID)` additionally specifies the name or time of the snapshot specified by `SnapID`.

Input Arguments

`actor` — Actor class whose properties are restored
`sim3d.Actor` object

Actor class whose properties are restored, specified as a `sim3d.Actor` object.

`SnapID` — Name of snapshot
character array

Name of snapshot being restored, specified as a character array. If `SnapID` is empty or incorrect or non existing snapshot ID is provided, it will result in an error and no snapshot will be restored.

Version History

Introduced in R2022b

See Also

`copy` | `findBy` | `propagate` | `gather` | `takeSnapshot`

createMesh

Create new mesh with specified values

Syntax

```
createMesh(actor,vertices,normals,faces)
createMesh( ___ ,tcoords,vcolor)
```

Description

`createMesh(actor,vertices,normals,faces)` creates a mesh defined by `vertices`, `normals` and `faces`.

`createMesh( ___ ,tcoords,vcolor)` creates a mesh additionally defined by the texture coordinates `tcoords` and the vertex colors `vcolor`.

Examples

Build Actor from Mesh Data and Apply Texture

This example shows how to build an actor from mesh data using the `createMesh` function and how to use the texture property of the `sim3d.Actor` object. First, you create a world scene and an actor object. Next, you build the appearance of the actor from a mesh and apply a texture. Then, you add the actor to the world, transform the actor, and set a view in the scene. Finally, you view the actor in the Simulation 3D Viewer window.

A mesh is a 3D build of a model consisting of polygons and is defined by vertices, normals, and faces. If you require detailed control over the geometry of an actor, create an actor from a mesh. This example uses sphere geometry for the mesh data to build the actor.

Build Actor from Mesh in World

Create a world scene.

```
world = sim3d.World();
```

Instantiate an actor object named `sphere`. Use the `sim3d.utils.Geometry` function to create a sphere. Obtain the sphere vertices `V`, normals `N`, faces `F`, texture coordinates `T`, and vertex colors `C`.

```
ActObj = sim3d.Actor('ActorName', 'sphere');
[V, N, F, T, C] = sim3d.utils.Geometry.sphere([0.5 0.5 0.5]);
```

Create a mesh using the actor object and sphere geometry. Set a texture provided by the image file. Add the actor object to the world.

```
createMesh(ActObj, V, N, F, T, C);
ActObj.Texture = fullfile(pwd,"image.png");
add(world,ActObj);
```

Set Actor Transformation

Use the actor object translation, rotation, and scale properties to orient the actor relative to the world origin.

```
ActObj.Translation = [0 0 0];
ActObj.Rotation = [0, 0, 0];
ActObj.Scale = [1, 1, 1];
```

Set Viewer Window Point of View

If you do not create a viewport, then the point of view is set to 0, 0 ,0, and you can use the arrow keys and pointer to navigate in the Simulation 3D Viewer window.

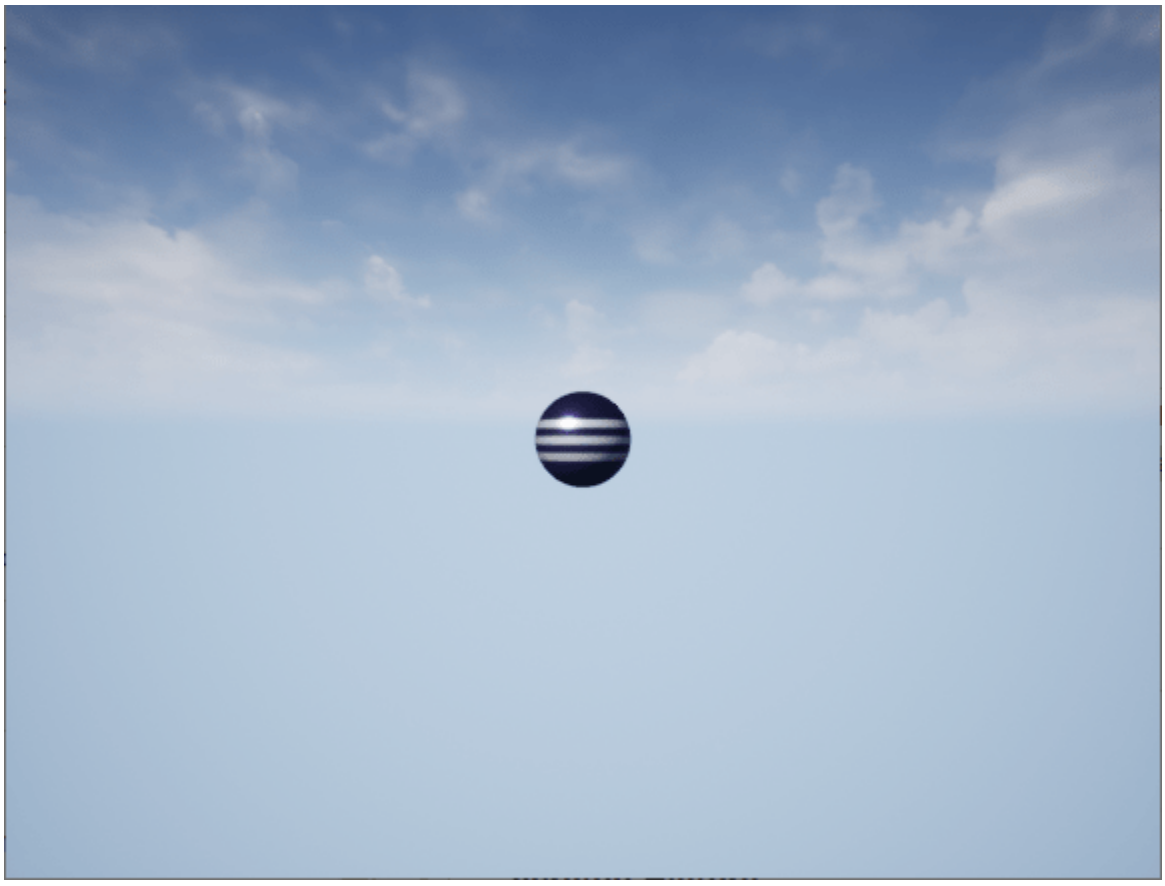
For this example, use the `createViewport` function to create a viewport with a single field, `Main`, that contains a `sim3d.sensors.MainCamera` object.

```
viewport = createViewport(world);
viewport.Translation = [-3, 0, 0];
```

Run Animation

Run a simulation set for 10 seconds with a sample time of 0.02 seconds.

```
run(world,0.02,10)
```



Delete World

Delete the world object.

```
delete(world)
```

Input Arguments

`vertices` — Vertex positions

real positive (*N*,3) vector

Vertex positions, specified as a real positive (*N*,*3*) vector. This vector includes all vertex positions to be used for the mesh geometry.

Example: `vertices = reshape(1: 6, 2, 3)`

Data Types: `double`

`actor` — Actor class where mesh is being created

`sim3d.Actor` object

Actor class where mesh is created, specified as a `sim3d.Actor` object.

`faces` — Faces of actor shape

real positive (*N*,3) vector

Faces of actor shape, specified as a real positive (N,3) vector. This vector defines how each triangle of the mesh is drawn. Length must be a multiple of 3.

Example: `faces = [1: 3; 4 : 6]`

Data Types: `double`

`normals` — Normal vectors

real positive (*N*,3) vector

Normal vectors for each vertex, specified as a real positive (*N*,3) vector. This vector must be the same length as `vertices` vector.

Example: `normals = reshape(7: 12, 2, 3)`

Data Types: `double`

`vcolor` — Vertex colors

real positive (*N*,3) vector

Vertex colors, specified a real positive (*N*,3) vector. This vector must be the same length as `vertices` vector.

Example: `vertexcolor = reshape(25 : 33, 3, 3)`

Data Types: `double`

`tcoords` — Texture coordinates

real positive (*N*,2) vector

Texture coordinates of each vertex, specified a real positive (*N*,2) vector. This must be the same length as `vertices` array.

Example: `texturecoord = reshape(21 : 24, 2, 2)`

Data Types: `double`

Version History

Introduced in R2022b

See Also

`copy` | `findBy` | `propagate` | `gather` | `takeSnapshot` | `restoreSnapshot`

addMesh

Add mesh on top of current mesh

Syntax

```
addMesh(actor,vertices,normals,faces)
addMesh( ___ ,tcoords,vcolor)
```

Description

`addMesh(actor,vertices,normals,faces)` adds a partial mesh defined by `vertices`, `normals`, and `faces`. The resulting geometric mesh creates a single solid body.

`addMesh( ___ ,tcoords,vcolor)` adds a mesh additionally defined by the texture coordinates `tcoords` and the vertex colors `vcolor`.

Input Arguments

`vertices` — Vertex positions

real positive (*N*,3) vector

Vertex positions, specified as a real positive (*N*,*3*) vector. This vector includes all vertex positions to be used for the mesh geometry.

Example: `vertices = reshape(1: 6, 2, 3)`

Data Types: `double`

`actor` — Actor class where property is being added

`sim3d.Actor` object

Actor class where mesh is being added, specified as a `sim3d.Actor` object.

`faces` — Faces of actor shape

real positive (*N*,3) vector

Faces of actor shape, specified as a real positive (N,3) vector. This vector defines how each triangle of the mesh is drawn. Length must be a multiple of 3.

Example: `faces = [1: 3; 4 : 6]`

Data Types: `double`

`normals` — Normal vectors

real positive (*N*,3) vector

Normal vectors for each vertex, specified as a real positive (*N*,3) vector. This vector must be the same length as `vertices` vector.

Example: `normals = reshape(7: 12, 2, 3)`

Data Types: `double`

`vcolor` — Vertex colors
real positive (*N*,3) vector

Vertex colors, specified a real positive (*N*,3) vector. This vector must be the same length as `vertices` vector.

Example: `vertexcolor = reshape(25 : 33, 3, 3)`

Data Types: `double`

`tcoords` — Texture coordinates
real positive (*N*,2) vector

Texture coordinates of each vertex, specified a real positive (*N*,2) vector. This must be the same length as `vertices` array.

Example: `texturecoord = reshape(21 : 24, 2, 2)`

Data Types: `double`

Version History

Introduced in R2022b

sim3d.Actor

Object used to define actors in the Unreal Engine viewer

Description

Use the `sim3d.Actor` object for user-defined Unreal Engine C++ or blueprint actors.

Creation

Syntax

```
actor = sim3d.Actor
actor = sim3d.Actor( ___ ,Name,Value)
```

Description

`actor = sim3d.Actor` creates a default actor object that represents all items in the virtual reality world.

`actor = sim3d.Actor( ___ ,Name,Value)` specifies additional options using one or more name-value arguments.

Input Arguments

Name-Value Pair Arguments

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

Example: `actor1 = sim3d.Actor('ActorName','vehicle1','Translation', [3, 4, 3], 'Rotation', [], 'Scale',[1 2 1], 'OnHit', @hitcallback)`

`ActorName` — Name of actor
autogenerated name (default) | character array | string

Name of actor, specified as a character array or string. If an actor name is not specified, then the actor is assigned an autogenerated name.

Note If you specify the same name as an actor that already exists, then the actor name you specify is appended with a unique identifier to differentiate it.

`Translation` — Relative translation
`[0,0,0]` (default) | real 1-by-3 vector

Relative translation (*x*,*y*,*z*) of the actor object to its parent actor, specified as a real 1-by-3 vector, in meters. Use this argument to set the `Translation` property of the `sim3d.Actor` object.

Data Types: `double`

`Rotation` — Relative rotation
`[0,0,0]` (default) | real 1-by-3 vector

Relative rotation (roll, pitch, yaw) of the actor object to its parent actor, specified as a real 1-by-3 vector, in radians. Use this argument to set the `Rotation` property of the `sim3d.Actor` object.

Data Types: `double`

`Scale` — Relative scaling
`[1,1,1]` (default) | real 1-by-3 vector

Relative scaling in *x*, *y* and *z* coordinates, specified as a real 1-by-3 vector. Use this name value argument to set the `Scale` property of the `sim3d.Actor` object.

Data Types: `double`

`ActorClassId` — Semantic segmentation map of object class identifiers
`0` (default) | *m*-by-*n* array

Semantic segmentation map of object class identifiers, specified as an array.

`OnHit` — Event callback function
function handle

Use the `@hitcallback` function and pass an input argument to store data in a structure for a `sim3d.Actor` hit event.

Example: `function hitcallback(EventData)`

`OnBeginOverlap` — Event callback function
function handle

Use the `@beginoverlapcallback` function and pass an input argument to store data in a structure for a `sim3d.Actor` begin overlap event.

Example: `function beginoverlapcallback(EventData)`

`OnEndOverlap` — Event callback function
function handle

Use the `@endoverlapcallback` function and pass an input argument to store data in a structure for a `sim3d.Actor` end overlap event.

Example: `function endoverlapcallback(EventData)`

`OnClick` — Event callback function
function handle

Use the `@clickcallback` function and pass an input argument to store data in a structure for a `sim3d.Actor` click event.

Example: `function clickcallback(EventData)`

In addition to these name-value arguments, you also can specify any of the properties on this page as name-value arguments.

Output Arguments

`actor` — Actor object
`sim3d.Actor` object

Actor object, returned as a `sim3d.Actor` object. A unique identifier and unique name is assigned to the actor. You can change the actor name, but the unique identifier is associated with the actor until the actor is deleted. The virtual reality world is composed of a hierarchical structure of actors.

Properties

Base Attributes

`UserData` — Structure to hold actor data
structure

Structure to hold actor data, specified as a structure. You can update and maintain this data and use it to update the fields of the structure during simulation or in the setup method.

`Parent` — Parent of actor
handle to parent `sim3d.actor` object

Parent of actor, specified as a handle to the parent `sim3d.Actor` object. You can set this property using the `Parent` argument.

`Children` — Children of actor
handle to children `sim3d.actor` object

This property is read-only.

Children of actor, specified as a handle to the `sim3d.Actor` object.

`ParentWorld` — Parent world method handle
handle to parent `sim3d.World` object

This property is read-only.

Parent world method handle, specified as a handle to the parent `sim3d.Actor` object.

`Translation` — Cartesian coordinates for actor position
`[0,0,0]` (default) | real 1-by-3 vector

Cartesian coordinates for actor position in the Unreal Editor scene, specified as a real 1-by-3 vector. You can set this property using the `Translation` argument.

`Rotation` — Rotation of actor
`[0,0,0]` (default) | real 1-by-3 vector

Rotation of actor in the Unreal Editor scene, specified as a three element vector. You can set this property using the `Rotation` name value argument.

`Scale` — Scale of actor
`[1,1,1]` (default) | real 1-by-3 vector

Scale of actor used for scaling in each of three directions in the Unreal Editor scene, specified as a real 1-by-3 vector. You can set this property using the `Scale` argument.

`Mobility` — Type of actor mobility

`'sim3d.utils.MobilityTypes.Movable'` (default) | `'sim3d.utils.MobilityTypes.Static'`

Type of actor mobility to respond to physics and/or move the actor during simulation, specified as `'sim3d.utils.MobilityTypes.Movable'` or `'sim3d.utils.MobilityTypes.Static'`.

Data Types: `string`

Mesh Attributes

`Vertices` — Vertices of mesh geometry

empty array (default) | real *n*-by-3 vector

Vertices of the mesh geometry, specified as a real *n*-by-3 vector, where *n* is the number of vertices.

Example: `actor.Vertices = [1, 2, 3; 43, 54, 65]`

Data Types: `double`

`Faces` — Vertices of each triangle

empty array (default) | real *n*-by-3 vector

Vertices of each triangle, specified as a real *n*-by-3 vector, where *n* is the number of vertices. You can use this property to define how each triangle of the mesh is drawn.

Example: `actor.Faces = [1, 2, 3; 4, 5, 6]`

Data Types: `double`

`Normals` — Normals of each vertex

empty array (default) | real *n*-by-3 vector

Normals of each vertex, specified as a real *n*-by-3 vector, where *n* is the number of vertices. `Normals` must be of the same length as `Vertices`.

Example: `actor.Normals = [1, 2, 3; 43, 54, 65]`

Data Types: `double`

`TextureCoordinates` — Texture coordinates

empty array (default) | real *n*-by-2 vector

Texture coordinates for each vertex, specified as a real *n*-by-2 vector, where *n* is the number of vertices. This argument is optional, and if specified, must be the same length as `Vertices`.

Example: `actor.TextureCoordinates = [1 2;3 4]`

`VertexColors` — Vertex colors

empty array (default) | real *n*-by-3 vector

Vertex colors, specified as a real *n*-by-3 vector, where *n* is the number of vertices. This argument is optional, and if specified, must be the same length as `Vertices`.

Example: `actor.VertexColors = [1, 2,3; 4, 5, 6]`

Material Attributes

`Color` — Base color of actor

`[1,1,1]` (default) | real 1-by-3 vector

Base color of actor, specified as a real 1-by-3 vector. `Color` consists of a vector of color RGB components `[red green blue]`. Values greater than `1` cause glowing and can be used for various indicators. If the `sim3d.Actor` object has a texture, the `TextureMapping.Blend` coefficient can be used to control how it blends with the base color.

Example: `actor.Color = [0.3 0.27 0.9]`

`Transparency` — Transparency of actor

`0` (default) | real positive number in the range (0,1)

Transparency of actor, specified as a real positive number in the range (0,1), where 0 indicates a nontransparent object and 1 indicates a completely transparent object.

Example: `actor.Transparency = 0.8`

Data Types: `double`

`Shininess` — Shininess of actor

`0.7` (default) | real positive number in the range (0,1)

Shininess of actor, specified as a real positive number in the range (0,1), where 0 indicates a nonshiny object and 1 indicates a completely shiny object.

Example: `actor.Shininess = 0.3`

Data Types: `double`

`Metallic` — Metallic look of actor

`0.7` (default) | real positive number in the range (0,1)

Metallic look of actor, specified as a real positive number in the range (0,1), where 0 indicates a plastic surface and 1 indicates a metallic surface.

Example: `actor.Metallic = 0.1`

Data Types: `double`

`Flat` — Flat shading factor of actor

`0` (default) | real positive number in the range (0,1)

Flat shading factor of actor, specified as a real positive number in the range (0,1), where 0 indicates a smooth surface and 1 indicates a faceted surface.

Example: `actor.Flat = 0.7`

Data Types: `double`

`Tessellation` — Tessellation factor of actor

0 (default) | real positive integer in the range (0,8)

Tessellation factor of actor, specified as a real positive integer in the range (0,8). Use this property to specify the coefficient of automatic geometry refinement. This property can be useful when texture displacement mapping is used.

Example: `actor.Tessellation = 5`

`VertexBlend` — Color blending coefficient of vertices

0 (default) | real positive number in the range (0,1)

Color blending coefficient of vertices, specified as a real positive number in the range (0,1). This value can be larger to achieve the effect of glowing colors.

Example: `actor.VertexBlend = 0.5`

`TextureTransform` — Texture transformations applied to actor texture

real positive vector

Texture transformations applied to actor texture, specified as a real positive scalar. Use `TextureTransform` to define texture position, velocity, scale, and angle.

Transformation Properties

Property	Detail	Value	Value Example
Position	Use this property to set the position of the texture. You can use `Position` to move the texture along *U* and *V* coordinates.	• Value – Real positive (1,2) vector • Default Value – (0,0) • Data Type – `double`	`actor.TextureTransform.Position = [2, 3]`
Velocity	Use this property to set the velocity of the texture movement. velocity is expressed as a change of *U* and *V* coordinates per second of simulation time. The animated surface has no effect on physical interactions.	• Value – Real positive (1,2) vector • Default Value – (0,0) • Data Type – `double`	`actor.TextureTransform.Velocity = [2, 3]`
Scale	Use this property to set the relative texture scale. If the texture is smaller than the surface to be covered, the scale repeats automatically in all directions. Negative values can be used to flip the texture in corresponding coordinate.	• Value – Real positive (1,2) vector • Default Value – (0,0) • Data Type – `double`	`actor.TextureTransform.Scale = [2, 2]`

`TextureMapping` — Texture mapping parameters applied to actor texture

real positive vector

Texture mapping parameters applied to actor texture, specified as a real positive vector. Use `TextureMapping` to define texture blend, displacement, bump factor, and roughness.

Mapping Properties

Property	Detail	Value	Value Example
Blend	Use this property to set the blend ratio of the texture. You can use `Blend` to set the ratio of texture mixing with the base color of the surface using linear interpolation. To create an effect of glowing texture, use a black base color ([0 0 0]) and blend values greater than 1. You can set the value independently for each color channel ([red green blue]).	• Value – Real positive (1,3) vector • Default Value – (0,0,0) • Data Type – `double`	`actor.TextureMapping.Blend = [1 1 0.5]`
Displacement	Use this property to set the displacement of the texture. You can use `Displacement` to move the vertex of a geometric mesh in the direction of the normal depending on the color of the texture at that location. This deformation is only visual and does not affect physical interactions. You can set the value independently for each color channel ([red green blue]) and its range is unlimited.	• Value – Real positive (1,3) vector • Default Value – (0,0,0) • Data Type – `double`	`actor.TextureMapping.Displacement = [1 2 5]`

Property	Detail	Value	Value Example
Bumps	Use this property to set the bump factor of the texture. You can use `Bump` to create the effect of a bumpy surface (such as a stone wall) by making small manipulations with normals and lighting. You can set the value independently for each color channel ([red green blue]) and its range is unlimited.	• Value – Real positive (1,3) vector • Default Value – (0,0,0) • Data Type – `double`	`actor.TextureMapping.Bumps = [10 10 0.5]`
Roughness	Use this property to set the roughness factor of the texture. You can use `Roughness` to locally manipulate the global surface shininess specified by the Shininess property. You can set the value independently for each color channel ([red green blue]) and its range is unlimited.	• Value – Real positive (1,3) vector • Default Value – (0,0,0) • Data Type – `double`	`actor.TextureMapping.Roughness = [1 1 0.5]`

`Texture` — Source file for actor shape
'' (default) | character array

Source file for actor shape, specified as a character array. The supported file types are JPEG, PNG, and BMP. The filepath should be absolute.

Note For BMP files, only 8-bit files are supported.

Example: `actor.Texture = "file.jpg"`

Physical Attributes

`Shadows` — Actor shadows
`0` or `false` (default) | `1` or `true`

Actor shadows, specified as `0` (`false`) if the actor does not cast shadows or `1` (`true`) if it does.

`LinearVelocity` — Linear velocity of actor
`(0,0,0)` (default) | real positive vector

Linear velocity of actor in local coordinates, specified as a real positive vector, in meters per second.

Example: `actor.LinearVelocity = [1 1 1]`

Dependencies

`Mobility` should be set to `'sim3d.utils.MobilityTypes.Movable'` for velocities to work. Otherwise the actor will not move, and you will see a warning.

`AngularVelocity` — Angular velocity of actor

`(0,0,0)` (default) | real positive vector

Angular velocity of actor in local coordinates, specified as a real positive vector, in radians per second.

Example: `actor.AngularVelocity = [2 2 2]`

Dependencies

`Mobility` should be set to `'sim3d.utils.MobilityTypes.Movable'` for velocities to work. Otherwise the actor will not move, and you will see a warning.

`Mass` — Mass of actor

`0` (default) | real positive scalar

Mass of actor, specified as a real positive scalar, in kilograms.

Example: `actor.Mass = 12`

`CenterOfMass` — Center of mass of actor

`(0,0,0)` (default) | real positive vector

Center of mass of `sim3d.Actor` object, specified as a real positive vector. Use this property to shift the center of gravity from the origin of the local coordinate system.

Example: `actor.CenterOfMass = [1 0 1 ]`

`Gravity` — Application of gravity to actor

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

Application of gravity to actor, specified as `0` (`false`) if no gravity is applied or `1` (`true`) if gravity is applied.

Example: `actor.Gravity = false`

Dependencies

- This property works when:
  - `Physics` property is set to `1` or `true`.
  - `PreciseContact` property is set to `0` or `false`.
- `Mobility` should be set to `'sim3d.utils.MobilityTypes.Movable'` for the actor to experience the effects of gravity. Otherwise the actor will not move, and you will see a warning.

`Physics` — Reaction of actor to physical forces

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

Reaction of actor to physical forces such as gravity and collision, specified as `0` (`false`) or `1` (`true`).

If `Physics` is enabled, the actor moves independently of its parent actor object but together with its children, unless the children also have `Physics` enabled.

Example: `actor.Physics = true`

Dependencies

This property works when `PreciseContact` property is set to `0` or `false`.

`Collisions` — Object collision

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

Object collision, specified as `0` (`false`) for no collision or `1` (`true`) if objects will collide.

Example: `actor.Collisions = true`

`PreciseContacts` — Precise contacts

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

Precise contacts during collisions, specified as `0` (`false`) or `1` (`true`). Setting value to `true` allows Unreal Engine to precisely render collisions.

Example: `actor.PreciseContacts = true`

Dependencies

Setting `PreciseContacts` property to `true` for an actor disables `Physics` and `Gravity`.

`LocationLocked` — Stationary translational motion

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

Stationary translational motion, specified as either `0` (`false`) or `1` (`true`). If this property is enabled, the actor is fixed in place. If the actor has defined a nonzero linear velocity, it interacts with other objects as if it were moving itself. You can use this property to model belt conveyors.

Example: `actor.LocationLocked = true`

`RotationLocked` — Stationary rotational motion

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

Stationary rotational motion, specified as either `0` (`false`) or `1` (`true`). If this property is enabled, the `sim3d.Actor` object is fixed in place. If the actor has defined a nonzero angular velocity, it interacts with other objects as if it were moving itself. You can use this property to model circular conveyors (carousels).

Example: `actor.RotationLocked = true`

`Friction` — Dynamic friction

0.7 (default) | scalar

Dynamic friction, dimensionless, specified as a scalar.

Example: `actor.Friction = 0.7`

`Restitution` — Coefficient of restitution

0.3 (default) | scalar

Coefficient of restitution, dimensionless, specified as a scalar.

Example: `actor.Restitution = 0.3`

Event Attributes

`Events` — Option to report actor events

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

Option to report actor events, specified as `0` (`false`) or `1` (`true`). This table provides the required events and collision property settings to report events.

Event	Report Event Description	Actor `Events` Property	Actor `Collision` Property
`Hit`	Actor 1 collides with Actor 2	Actor 1 – `true` Actor 2 – `true` or `false`	Actor 1 – `true` Actor 2 – `true`
`BeginOverlap`	Actor 1 starts to overlap with Actor 2	Actor 1 – `true` Actor 2 – `true` or `false`	Actor 1 – `false` Actor 2 – `false`
`EndOverlap`	Actor 1 stops overlapping with Actor 2	Actor 1 – `true` Actor 2 – `true` or `false`	Actor 1 – `false` Actor 2 – `false`
`ClickEvent`	You click Actor 1	Actor 1 – `true`	NA

Example: `actor.Events = true`

`HitEvent` — Hit event

`0` or `false` | `1` or `true`

This property is read-only.

Hit event for `sim3d.Actor` object, specified as `0` (`false`) or `1` (`true`).

`HitSelfID` — Hit `sim3d.Actor` object identifier

real positive integer

This property is read-only.

Hit `sim3d.Actor` object identifier, specified as a real positive integer.

`HitOtherID` — Identifier of actor that collides with `sim3d.Actor` object

real positive integer

This property is read-only.

Identifier of actor that collides with the `sim3d.Actor` object, specified as a real positive integer.

`HitLocation` — Hit event location

`[0,0,0]` (default) | real 1-by-3 vector

This property is read-only.

Hit event location in the Unreal Editor scene, specified as a real 1-by-3 vector.

Data Types: `double`

`HitOtherActorName` — Name of actor that collides with `sim3d.Actor` object
character array | string

This property is read-only.

Name of actor that collides with the `sim3d.Actor` object, specified as a character array or string.

`BeginOverlapEvent` — Overlap begin event
`0` or `false` | `1` or `true`

This property is read-only.

Overlap begin event for `sim3d.Actor` object, specified as `0` (`false`) or `1` (`true`).

`BeginOverlapSelfID` — Identifier of overlapped `sim3d.Actor` object
real positive integer

This property is read-only.

Identifier of the overlapped `sim3d.Actor` object, specified as a real positive integer.

`BeginOverlapOtherID` — Identifier of actor that overlaps with `sim3d.Actor` object
real positive integer

This property is read-only.

Identifier of the actor that overlaps with the `sim3d.Actor` object, specified as a real positive integer.

`BeginOverlapOtherActorName` — Name of actor that overlaps with `sim3d.Actor` object
character array | string

This property is read-only.

Name of actor that overlaps with the `sim3d.Actor` object, specified as a character array or string.

`EndOverlapEvent` — Overlap end event
`0` or `false` | `1` or `true`

This property is read-only.

Overlap end event for `sim3d.Actor` object, specified as `0` (`false`) or `1` (`true`).

`EndOverlapSelfID` — Identifier of `sim3d.Actor` object ending overlap
real positive integer

This property is read-only.

Identifier of the `sim3d.Actor` object that is ending overlap, specified as a real positive integer.

`EndOverlapOtherID` — Identifier of actor that ends overlap with `sim3d.Actor` object
real positive integer

This property is read-only.

Identifier of the actor that ends overlap with the `sim3d.Actor` object, specified as a real positive integer.

`EndOverlapOtherActorName` — Name of actor that ends overlap with `sim3d.Actor` object
character array | string

This property is read-only.

Name of actor that ends overlap with the `sim3d.Actor` object, specified as a character array or string.

`ClickEvent` — Click event
`0` or `false` | `1` or `true`

This property is read-only.

Click event for `sim3d.Actor` object, specified as `0` (`false`) or `1` (`true`).

`ClickActorID` — Identifier of `sim3d.Actor` object that is clicked
real positive integer

This property is read-only.

Identifier of the `sim3d.Actor` object that is clicked, specified as a real positive integer.

`ClickLocation` — Cartesian coordinates of the click event location
`[0,0,0]` (default) | real 1-by-3 vector

This property is read-only.

Cartesian coordinates of the click event location in the Unreal Editor scene, specified as a real 1-by-3 vector.

Data Types: `double`

`ClickActorName` — Name of clicked actor
character array | string

This property is read-only.

Name of clicked actor, specified as a character array or string.

Object Functions

copy	Copy all properties from another actor
findBy	Find all actors that match specified criteria
propagate	Propagate value of selected property to actor and its children
gather	Return values of selected property from all objects in selected branch
takeSnapshot	Take snapshot of selected properties
restoreSnapshot	Restore actor to state of properties saved in specified snapshot
createMesh	Create new mesh with specified values
addMesh	Add mesh on top of current mesh
load	Load or import 3D file
save	Save actor and children to a MAT file
createShape	Create geometry for basic primitives

Examples

Create World and Actor

This example shows how to use the `sim3d.World` and `sim3d.Actor` objects to create an actor in a world scene to display in the Simulation 3D Viewer window. This process does not build an actor object appearance, preventing the Simulation 3D Viewer from rendering a visualization of the actor object. For an example building an appearance for an actor, see “Build Actor from Mesh Data and Apply Texture”.

Create World

Create a world scene.

```
world = sim3d.World();
```

Create Actor

Add an actor to the world.

```
add(world,sim3d.Actor());
```

Run Simulation

Run a simulation set for 10 seconds with a sample time of 0.02 seconds.

```
run(world,0.02,10)
```

Delete World

Delete the world object.

```
delete(world);
```

Version History

Introduced in R2022b

R2023a: Communicate Unreal Engine Events

Use `sim3d.Actor` object properties to communicate events in the Unreal Engine simulation 3D environment, including when:

- You click an actor.
- An actor collides or overlaps with another actor.
- You initialize an event callback for an actor.

R2023a: Precise Contacts

Use the `PreciseContacts` property to allow Unreal Engine to precisely render collisions.

R2023a: Physical properties

The `sim3d.Actor` object has these new physical properties:

- `Friction`
- `Restitution`

See Also

`sim3d.World` | `sim3d.sensors.IdealCamera` | `sim3d.sensors.MainCamera` | Simulation 3D Actor

copy

Copy all properties from another actor

Syntax

```
copy(self,other)
copy(self,other,copyChildren)
copy(self,other,copyChildren,useSourcePosition)
```

Description

`copy(self,other)` copies the contents of the actor specified by `other` to the actor object `self`.

`copy(self,other,copyChildren)` additionally copies the children of the actor.

`copy(self,other,copyChildren,useSourcePosition)` also provides an option for the source actor to retain its location.

Input Arguments

`self` — Actor object

`sim3d.Actor` object

Actor object, specified as a `sim3d.Actor` object, which is the destination actor to which a copy is made.

Example: `copy(actor1,actor2)`

`other` — Actor object

`sim3d.Actor` object

Actor object, specified as a `sim3d.Actor` object. This object is the source actor from which properties are copied.

`copyChildren` — Option to copy children

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

Option to copy children from `other` to `self`, specified as either `1` (`true`) or `0` (`false`). If `true`, the copy function copies all children and their properties from the specified source. If `false`, the function does not copy these properties.

Data Types: `logical`

`useSourcePosition` — Option for source actor to retain its location

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

Option for source actor to retain its location, specified as either `1` (`true`) or `0` (`false`). When `true`, the `copy` function leaves the position of `self` (translation, rotation) as-is.

Example: `copy(actor1, actor2, false, true)`

Data Types: `logical`

Version History

Introduced in R2022b

See Also

`findBy` | `propagate` | `gather` | `takeSnapshot` | `restoreSnapshot`

save

Save actor and children to a MAT file

Syntax

```
save(actor,dest)
```

Description

`save(actor,dest)` saves the actor class specified by `actor` and its children to the `dest` MAT file.

Input Arguments

`actor` — Actor class that is saved

`sim3d.Actor` object

Actor class that is saved, specified as a `sim3d.Actor` object.

`dest` — Name of destination where actor is saved

character array

Name of destination where actor is saved, specified as a character array. The destination can be a file path or file name.

Version History

Introduced in R2022b

See Also

`copy` | `findBy` | `propagate` | `gather` | `takeSnapshot` | `restoreSnapshot`

propagate

Propagate value of selected property to actor and its children

Syntax

```
propagate(actor,PropertyName,PropertyValue)
propagate(actor,PropertyName,PropertyValue,Condition)
```

Description

`propagate(actor,PropertyName,PropertyValue)` propagates the property value specified by `PropertyValue` to the `PropertyName` property of `actor`.

`propagate(actor,PropertyName,PropertyValue,Condition)` restricts propagation according to the value of `Condition`.

Input Arguments

`actor` — Actor class where property is propagated
`sim3d.Actor` object

Actor class where property is propagated, specified as a `sim3d.Actor` object.

`PropertyName` — Name of property propagated
`sim3d.Actor` object property

Name of property being propagated to `actor`, specified as one of the properties of the `sim3d.Actor` object.

`PropertyValue` — Value of property propagated
`sim3d.Actor` object property

Value of property propagated to `actor`, specified as one of the properties of the `sim3d.Actor` object.

`Condition` — Where to propagate property value
`'all'` | `'children'` | `'selected'`

Where to propagate property value, specified as `'all'`, `'children'`, or `'selected'`, where

- `'all'` – Propagate value to `actor` and all its children.
- `'children'` – Propagate value only to children but not to `actor` itself.
- `'selected'` – Propagate value only to selected `sim3d.Actor` objects.

Version History

Introduced in R2022b

See Also

`copy` | `findBy` | `gather` | `takeSnapshot` | `restoreSnapshot`

gather

Return values of selected property from all objects in selected branch

Syntax

```
res = gather(actor,PropertyName)
res = gather(actor,PropertyName,IncludeChildren)
```

Description

`res = gather(actor,PropertyName)` returns a cell array with objects found in the actor class specified by `actor` and current property values for `PropertyName`.

`res = gather(actor,PropertyName,IncludeChildren)` specifies whether the children of `actor` should be searched too.

Input Arguments

`actor` — Actor class whose property is returned
`sim3d.Actor` object

Actor class whose property is returned, specified as a `sim3d.Actor` object.

`PropertyName` — Name of property returned
`sim3d.Actor` property

Name of property returned from `actor`, specified as one of the Properties of the `sim3d.Actor` object.

`IncludeChildren` — Whether to include children of actor
`true` or `1` (default) | `false` or `0`

Whether to include children of in search, specified as either `true` (`1`) or `false` (`0`).

- `true` – Include all children.
- `false` – Do not include children.

Output Arguments

`res` — Table of results
cell array

Table of results, returned as a cell array. The first column contains references to found actors. The second column contains the current value of the property specified by `PropertyName`.

Version History

Introduced in R2022b

See Also

`copy` | `propagate` | `findBy` | `takeSnapshot` | `restoreSnapshot`

createShape

Create geometry for basic primitives

Syntax

```
createShape(actor,type)
createShape(actor,type,inputspec)
```

Description

`createShape(actor,type)` creates the actor `actor` in the shape defined by `type`.

`createShape(actor,type,inputspec)` provides additional details specified by `inputspec` for the shape specified by `type`.

Examples

Build Actor from 3D Graphic Primitives

This example shows how to build an actor from a 3D graphic primitive using the `createShape` function. First, you create a world scene and an actor object. Next, you build the appearance of the actor from a 3D graphic. Then, you add the actor to the world, transform the actor, and set a view in the scene. Finally, you view the actor in the Simulation 3D Viewer window.

If you do not require detailed control over the geometry of an actor, use a predefined 3D graphic from the list:

- Arrow
- Box
- Cone
- Cylinder
- Checker
- Extrusion
- Icosphere
- Plane
- Prism
- Pyramid
- Revolution
- Sphere
- Surf
- Terrain
- Tile
- Torus

- Tube
- Voxel

Build Actor from a 3D Graphic Primitive in World

Create a world scene.

```
world = sim3d.World();
```

Instantiate an actor object named `Cylinder`.

```
ActObj = sim3d.Actor('ActorName','Cylinder');
```

Build a cylinder shape for the actor object. Specify a color. Add the actor object to the world.

```
createShape(ActObj,'cylinder', [0.5, 0.5, .75]);
ActObj.Color = [1, 0, 1];
add(world,ActObj);
```

Set Actor Transformation

Use the actor object translation, rotation, and scale properties to orient the actor relative to the world origin.

```
ActObj.Translation = [0 0 0];
ActObj.Rotation = [0, 0, 0];
ActObj.Scale = [1, 1, 1];
```

Set Viewer Window Point of View

If you do not create a viewport, then the point of view is set to 0, 0 ,0, and you can use the arrow keys and pointer to navigate in the Simulation 3D Viewer window.

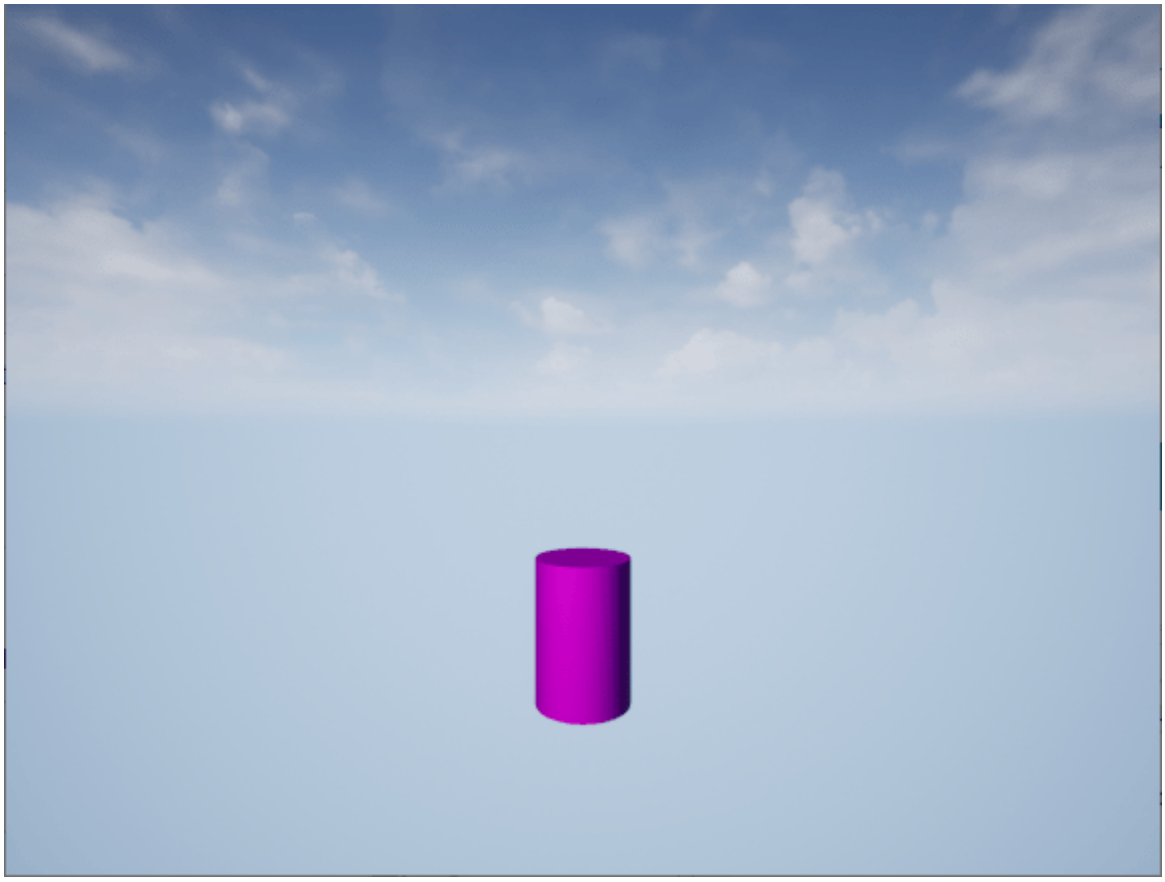
For this example, use the `createViewport` function to create a viewport with a single field, `Main`, that contains a `sim3d.sensors.MainCamera` object.

```
viewport = createViewport(world);
viewport.Translation = [-3, 0, 1];
```

Run Animation

Run a simulation set for 10 seconds with a sample time of 0.02 seconds.

```
run(world,0.02,10)
```



Delete World

Delete the world object.

```
delete(world)
```

Input Arguments

`actor` — Actor class where new geometry is created

`sim3d.Actor` object

Actor class where the new geometry is created, specified as a `sim3d.Actor` object.

`type` — Name of imported 3D file

`'arrow'` | `'box'` | `'cone'` | `'cylinder'` | `'checker'` | `'extrusion'` | `'icosphere'` | `'plane'` | `'prism'` | `'pyramid'` | `'revolution'` | `'sphere'` | `'surf'` | `'terrain'` | `'tile'` | `'torus'` | `'triad'` | `'tube'` | `'voxel'`

Name of imported 3D file, specified as one of these values:

- `'arrow'`
- `'box'`
- `'cone'`

- 'cylinder'
- 'checker'
- 'extrusion'
- 'icosphere'
- 'plane'
- 'prism'
- 'pyramid'
- 'revolution'
- 'sphere'
- 'surf'
- 'terrain'
- 'tile'
- 'torus'
- 'triad'
- 'tube'
- 'voxel'

`inputspec` — Optional parameters of each shape type
details of shape type

Optional parameters of each shape type, specified as details of the shape type.

Version History

Introduced in R2022b

See Also

`copy` | `findBy` | `propagate` | `gather` | `takeSnapshot` | `restoreSnapshot`

load

Load or import 3D file

Syntax

```
load(actor,source)
load(actor,source,scale)
```

Description

`load(actor,source)` loads the 3D file called `source` to the actor specified by `actor`.

`load(actor,source,scale)` specifies the overall scale applied to all the actors in `source`.

Input Arguments

`actor` — Actor class where 3D file is loaded

`sim3d.Actor` object

Actor class where the 3D file is loaded, specified as a `sim3d.Actor` object.

`source` — Name of location from where actor is loaded

character array

Name of location from where actor is loaded, specified as a character array. The source can be a file path or file name. `load` supports these as sources.

File Formats

- MAT
- STL
- FBX
- URDF
- x3d

MATLAB Objects

- RigidBodyTree
- surf (only 3D shapes based on polygons)
- patch (only 3D shapes based on polygons)

`scale` — Scale applied to loaded model

real positive scalar

Overall scale applied to the entire loaded model, specified as a real positive scalar. Scale parameters of individual objects are not affected by using this parameter.

Tips

You can load STL files even with `sim3d.Actor` objects that are not currently added to the World.

Version History

Introduced in R2022b

See Also

`copy` | `findBy` | `propagate` | `gather` | `takeSnapshot` | `restoreSnapshot`

read

Return image captured with camera

Syntax

```
image = read(camera)
```

Description

`image = read(camera)` returns an image that is captured with the ideal camera specified by `camera` in the Simulation 3D Environment (game engine)

Input Arguments

`camera` — Virtual camera that captures image

`sim3d.sensors.IdealCamera` object

Virtual camera that captures the image, specified as a `sim3d.sensors.IdealCamera` object.

Output Arguments

`image` — Image returned by camera

real positive array of integers

Image returned by camera, returned as a real positive array of integers. The size of `image` is vertical-resolution-by-horizontal-resolution-by-3.

Data Types: `uint8`

Version History

Introduced in R2022b

See Also

`sim3d.World` | `sim3d.sensors.MainCamera`

run

Run cosimulation in virtual reality world

Syntax

```
run(world,sampleTime)
run(world,sampleTime,simulationTime)
```

Description

`run(world,sampleTime)` runs the cosimulation with sampling period specified in `sampleTime`.

`run(world,sampleTime,simulationTime)` additionally specifies the simulation time in `simulationTime`.

Input Arguments

`world` — World object
`sim3d.World` object

The `sim3d.World` object where cosimulation is being run.

`sampleTime` — Sampling period of world
0.02 | scalar

Sampling period of the virtual reality world, specified as a real positive scalar.

Data Types: `single`

`simulationTime` — Length of time of simulation
inf | scalar

Length of time the simulation runs, specified as a real positive scalar.

Data Types: `single`

Version History

Introduced in R2022b

See Also

`sim3d.World` | `sim3d.Actor` | `findBy` | `propagate` | `gather` | `takeSnapshot` | `restoreSnapshot`

add

Add actor to virtual reality world

Syntax

```
act = add(world,actor)
act = add(world,actor,parent)
```

Description

`act = add(world,actor)` adds `actor` to `world` and returns the handle for `actor` in `act`.

`act = add(world,actor,parent)` additionally updates the world scene graph by attaching `actor` to `parent`. It also returns the handle for `actor` in `act`.

Input Arguments

`actor` — Actor object
`sim3d.Actor` object

The `sim3d.Actor` object being added to the 3D world.

`world` — World object
`sim3d.World` object

The `sim3d.World` object to which the actor object is being added.

`parent` — Parent actor object
`sim3d.Actor` object

Handle to the parent actor where `actor` is being attached.

Version History

Introduced in R2022b

See Also

`sim3d.World` | `sim3d.Actor` | `findBy` | `propagate` | `gather` | `takeSnapshot` | `restoreSnapshot`

remove

Remove actor added to world or remove all actors in world

Syntax

```
remove(world,actor)
remove(world)
```

Description

`remove(world,actor)` removes the actor specified by `actor` from the world specified by `world`.

`remove(world)` removes all actors from `world`.

Input Arguments

`world` — World object
`sim3d.World` object

World object from which the actor object is being removed, specified as a `sim3d.World` object.

`actor` — Actor object
`sim3d.Actor` object

Actor object being removed from the 3D world, specified as a `sim3d.Actor` object.

Version History

Introduced in R2023a

See Also

`sim3d.World` | `sim3d.Actor`

findBy

Find all actors that match specified criteria

Syntax

```
actorList = findBy(actor,PropertyName,PropertyValue)
actorList = findBy(actor,PropertyName,PropertyValue,SearchMode)
```

Description

`actorList = findBy(actor,PropertyName,PropertyValue)` finds within `actor` and its children, all actors that match `PropertyName` with specified `PropertyValue`.

`actorList = findBy(actor,PropertyName,PropertyValue,SearchMode)` finds within `actor` and its children, all actors that match `PropertyName` with specified `PropertyValue`.

Input Arguments

`actor` — Actor being searched

`sim3d.Actor` object

Actor being searched, specified as a `sim3d.Actor` object.

`PropertyName` — Name of property

character array

Name of property of `sim3d.Actor` object whose value is being matched, specified as a character array.

Example: `actors = findBy(Actor, 'Physics', true)` finds all actors with activated physics

`PropertyValue` — Value of property being matched

value of `sim3d.Actor` object property

Value of property being matched specified as the value of a `sim3d.Actor` object property.

Example: `a = findBy(Actor1, 'Color', [1 0 0])`

`SearchMode` — Method of actor tree search

`'full'` (default) | `'first'` | `'last'`

Method of actor tree search, specified as `'full'`, `'first'`, or .`'last'`.

- `'full'` – Search the entire tree branch.
- `'first'` – Search the first actor object.
- `'last'` – Search the last actor object.

Output Arguments

`actorList` — List of actors matching specified criteria
array of `sim3d.Actor` objects

List of actors matching specified criteria, specified as an array of `sim3d.Actor` objects.

Version History

Introduced in R2022b

See Also

`copy` | `propagate` | `gather` | `takeSnapshot` | `restoreSnapshot`

Simulink 3D Animation Player

Play recorded 3D animation files

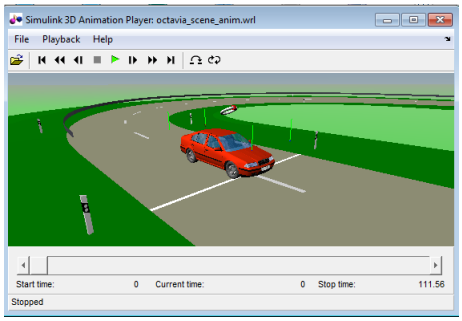
Description

Play recorded 3D animation files

The **Simulink 3D Animation Player** app plays 3D animation files created using the Simulink 3D Animation animation recording functionality.

You can control the playing of the animation using toolbar buttons or **Playback** menu options. For example, you can step forward or reverse, fast forward, or jump. For keyboard shortcuts, see `vrplay`.

To create an additional Simulink 3D Animation Player window, in the Simulink 3D Animation Player, select **File > New Window**.



Open the Simulink 3D Animation Player App

- Simulink Toolstrip: On the **Apps** tab, under **Simulation Graphics and Reporting**, click the app icon.
- MATLAB Toolstrip: On the **Apps** tab, under **Simulation Graphics and Reporting**, click the app icon.
- MATLAB command prompt: Enter `vrplay`.

Examples

Play an Animation File

To play the animation file based on the `vr_octavia` example, run `vrplay('octavia_scene_anim.wrl')`.

1. In the MATLAB **Apps** tab, in the **Simulation Graphics and Reporting** section, click **3D Animation Player**.
2. In the Simulink 3D Animation Player, select **File > Open**. Navigate to `matlab/toolbox/sl3d/sl3ddemos/octavia_scene_anim.wrl`.
3. Select **Playback > Play**.

Version History

Introduced in R2006a

See Also

Apps

3D World Editor

Functions

`vrview` | `vrsetpref` | `vrplay`

Topics

“Play Animation Files”
“Play Animation Files”
“Play Animations with Simulink 3D Animation Viewer”
“Animation Recording”
“File Name Tokens”

sim3d.sensors.IdealCamera

Capture an image and make it available in MATLAB

Description

Use the `sim3d.sensors.IdealCamera` class to capture an image with the virtual ideal camera in the 3D simulation environment and return the image to MATLAB.

Creation

Syntax

```
camera = sim3d.sensors.IdealCamera('ActorName',actor)
camera = sim3d.sensors.IdealCamera('ImageSize',[VerticalResolution,
HorizontalResolution])
camera = sim3d.sensors.IdealCamera('HorizontalFieldOfView',fieldOfView)
```

Description

`camera = sim3d.sensors.IdealCamera('ActorName',actor)` returns a `sim3d.sensors.IdealCamera` object with the actor specified by `ActorName`.

`camera = sim3d.sensors.IdealCamera('ImageSize',[VerticalResolution, HorizontalResolution])` returns an image of the size specified by `'ImageSize'`.

`camera = sim3d.sensors.IdealCamera('HorizontalFieldOfView',fieldOfView)` returns the horizontal field of view in degrees, specified by `'HorizontalFieldOfView'`s.

Input 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: `Camera = sim3d.sensors.IdealCamera('ActorName',Camera2)`

`'ActorName'` — Name of camera actor
`'Camera1'` (default) | character vector | string

Name of camera actor, specified as a character vector or string.

Example: `Camera = sim3d.sensors.IdealCamera('ActorName',Camera2)`

`'ImageSize'` — Image size returned by camera
`[768,1024]` (default) | 1-by-2 vector of real positive integers

Image size returned by camera, specified as a 1-by-2 vector of real positive integers. The unit is in Pixels.

`'HorizontalFieldOfView'` — Horizontal field of view in degrees
`90` (default) | real nonnegative scalar

Horizontal field of view in degrees, specified as a real nonnegative scalar.

Properties

`Parent` — Parent of actor
handle to parent `sim3d.actor` object

Parent of actor, specified as a handle to the parent `sim3d.Actor` object.

`Children` — Children of actor
handle to `sim3d.actor` children object

Children of actor, specified as a handle to the children `sim3d.Actor` object.

`ParentWorld` — Parent world method handle
handle to parent `sim3d.World` object

This property is read only.

Parent world method handle, specified as a handle to the `sim3d.World` object.

`SensorIdentifier` — Unique ID of sensor
real positive scalar

Unique ID of the sensor, specified as a real positive scalar.

Data Types: `uint32`

`Translation` — Translation of the sensor
`[0,0,0]` (default) | real 1-by-3 vector

Translation of sensor relative to the parent vehicle, specified as a real 1-by-3 vector, in meters.

`Rotation` — Rotation of the sensor
`[0,0,0]` (default) | real 1-by-3 vector

Rotation (roll, pitch, yaw) of sensor relative to the parent vehicle, specified as a real 1-by-3 vector, in radians.

Object Functions

read Return image captured with camera

Version History

Introduced in R2022b

See Also

`sim3d.World` | `sim3d.sensors.MainCamera` | `sim3d.Actor`

sim3d.sensors.MainCamera

Define perspective in Unreal executable window

Description

Use the `sim3d.sensors.MainCamera` class to define a perspective in the Unreal executable window using APlayerController::SetViewTarget.

Creation

The `sim3d.sensors.MainCamera` class is created by the `createViewport` function of the `sim3d.World` class and stored in the `Viewport` property.

Properties

`Parent` — Parent of actor
handle to parent `sim3d.actor` object

Parent of actor, specified as a handle to the parent `sim3d.Actor` object.

`Children` — Children of actor
handle to `sim3d.actor` children object

Children of actor, specified as a handle to the children `sim3d.Actor` object.

`ParentWorld` — Parent world method handle
handle to parent `sim3d.World` object

This property is read only.

Parent world method handle, specified as a handle to the `sim3d.World` object.

`SensorIdentifier` — Unique ID of sensor
real positive scalar

Unique ID of the sensor, specified as a real positive scalar.

Data Types: `uint32`

`Translation` — Translation of the sensor
`[0,0,0]` (default) | real 1-by-3 vector

Translation of sensor relative to the parent vehicle, specified as a real 1-by-3 vector, in meters.

`Rotation` — Rotation of the sensor
`[0,0,0]` (default) | real 1-by-3 vector

Rotation (roll, pitch, yaw) of sensor relative to the parent vehicle, specified as a real 1-by-3 vector, in radians.

Version History

Introduced in R2022b

See Also

`sim3d.World` | `sim3d.sensors.IdealCamera` | `sim3d.Actor`

optimize

Change geometries to reduce number of vertices

Syntax

```
optimize(world,ratio)
optimize(node,ratio)
```

Description

`optimize(world,ratio)` reduces the number of faces in all `IndexedFaceSet` nodes of `world` while trying to preserve the overall shape of these nodes.

`optimize(node,ratio)` reduces the number of faces in the geometry of `node` while trying to preserve the overall shape of the node. If `node` is a grouping node, the function optimizes all `IndexedFaceSet` children of that node.

Input Arguments

`world` — Virtual World
`vrworld` object

Virtual World, specified as a `vrworld` object.

`node` — Virtual Node
`vrnode` object

Virtual Node, specified as a `vrnode` object.

`ratio` — Fraction of intended number of faces relative to the original number of faces
real number

Fraction of intended number of faces relative to the original number of faces, specified as a real number. If `ratio` is less than 1, then it is interpreted as a fraction of the resulting number of faces relative to the original number of faces. If `ratio` is equal to 1, number of faces is unchanged and only shared vertices are merged (gluing faces).

Version History

Introduced in R2021a

See Also

`vrworld/edit` | `vrworld/reload` | `vrworld` | `vrnode` | `vrnode/getfield` | `vrnode/delete`

createViewport

Create viewport for world

Syntax

```
createViewport(world)
```

Description

`createViewport(world)` creates a viewport with a single field, `Main` that contains a `sim3d.sensors.MainCamera` object.

If you do not create a viewport, then the point of view is set to 0, 0 ,0, and you can use the arrow keys and pointer to navigate in the game.

Input Arguments

`world` — World where viewport is added

`sim3d.World` object

World where viewport is added, specified as a `sim3d.World` object. You can view the fields in the structure from the `Viewport` property of the `sim3d.World` object.

Version History

Introduced in R2022b

3D World Editor

Edit virtual worlds for 3D animation

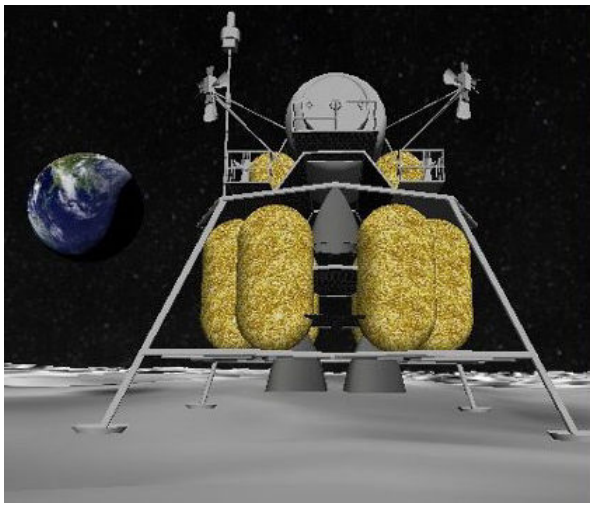
Description

Edit virtual worlds for 3D animation

The **3D World Editor** app creates virtual worlds for visualizing and verifying dynamic system behavior using Simulink 3D Animation. Build virtual worlds with Virtual Reality Modeling Language (VRML) or X3D (Extensible 3D).

Use the 3D World Editor to:

- Create objects in the virtual world from scratch using X3D or VRML node types.
- Create objects using templates available in the 3D World Editor object library.
- Import objects exported from CAD tools.
- Simplify geometries of imported objects.
- Create or modify hierarchy of objects in the scene.
- Give objects in the virtual world unique names in order to access them from MATLAB and Simulink.
- Set scene background, lighting and navigation properties.
- Define suitable viewpoints that are significant for working with the virtual world .



Open the 3D World Editor App

- Simulink Toolstrip: On the **Apps** tab, under **Simulation Graphics and Reporting**, click the app icon.
- MATLAB Toolstrip: On the **Apps** tab, under **Simulation Graphics and Reporting**, click the app icon.

- MATLAB command prompt: Enter `vredit`.

Examples

- “Create a Virtual World”
- “Build and Connect a Virtual World”
- “Edit a Virtual World”
- “Reduce Number of Polygons for Shapes”
- “Add Sound to a Virtual World”
- “View a Virtual World in Stereoscopic Vision”
- “Virtual Reality World and Dynamic System Examples”

Version History

Introduced in R2010b

See Also

Apps

Simulink 3D Animation Player

Functions

`vrsetpref` | `vrview` | `vredit`

Topics

“Create a Virtual World”
“Build and Connect a Virtual World”
“Edit a Virtual World”
“Reduce Number of Polygons for Shapes”
“Add Sound to a Virtual World”
“View a Virtual World in Stereoscopic Vision”
“Virtual Reality World and Dynamic System Examples”
“3D World Editor”
“X3D Support”
“Virtual Reality Modeling Language (VRML)”
“3D World Editor Library”
“Virtual World Navigation in 3D World Editor”