

# Simulink® 3D Animation™

User's Guide



# MATLAB® & SIMULINK®

R2020b



## How to Contact MathWorks



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



Phone: 508-647-7000



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

### *Simulink® 3D Animation™ User's Guide*

© COPYRIGHT 2001–2020 by HUMUSOFT s.r.o. and The MathWorks, Inc.

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

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

### **Trademarks**

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

### **Patents**

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

## Revision History

August 2001	First printing	New for Version 2.0 (Release 12.1)
July 2002	Second printing	Revised for Version 3.0 (Release 13)
October 2002	Online only	Revised for Version 3.1 (Release 13)
June 2004	Third printing	Revised for Version 4.0 (Release 14)
October 2004	Fourth printing	Revised for Version 4.0.1 (Release 14SP1)
March 2005	Online only	Revised for Version 4.1 (Release 14SP2)
April 2005	Online only	Revised for Version 4.2 (Release 14SP2+)
September 2005	Online only	Minor revision for Version 4.2.1 (Release 14SP3)
March 2006	Online only	Revised for Version 4.3 (Release 2006a)
September 2006	Online only	Revised for Version 4.4 (Release 2006b)
March 2007	Online only	Revised for Version 4.5 (Release 2007a)
September 2007	Online only	Revised for Version 4.6 (Release 2007b)
March 2008	Online only	Revised for Version 4.7 (Release 2008a)
October 2008	Online only	Revised for Version 4.8 (Release 2008b)
March 2009	Online only	Revised for Version 5.0 (Release 2009a)
March 2010	Online only	Revised for Version 5.1.1 (Release 2010a)
September 2010	Online only	Revised for Version 5.2 (Release 2010b)
April 2011	Online only	Revised for Version 5.3 (Release 2011a)
September 2011	Online only	Revised for Version 6.0 (Release 2011b)
March 2012	Online only	Revised for Version 6.1 (Release 2012a)
September 2012	Online only	Revised for Version 6.2 (Release 2012b)
March 2013	Online only	Revised for Version 6.3 (Release 2013a)
September 2013	Online only	Revised for Version 7.0 (Release 2013b)
March 2014	Online only	Revised for Version 7.1 (Release 2014a)
October 2014	Online only	Revised for Version 7.2 (Release 2014b)
March 2015	Online only	Revised for Version 7.3 (Release 2015a)
September 2015	Online only	Revised for Version 7.4 (Release 2015b)
March 2016	Online only	Revised for Version 7.5 (Release 2016a)
September 2016	Online only	Revised for Version 7.6 (Release 2016b)
March 2017	Online only	Revised for Version 7.7 (Release 2017a)
September 2017	Online only	Revised for Version 7.8 (Release 2017b)
March 2018	Online only	Revised for Version 8.0 (Release 2018a)
September 2018	Online only	Revised for Version 8.1 (Release 2018b)
March 2019	Online only	Revised for Version 8.2 (Release 2019a)
September 2019	Online only	Revised for Version 8.3 (Release 2019b)
March 2020	Online only	Revised for Version 9.0 (Release 2020a)
September 2020	Online only	Revised for Version 9.1 (Release 2020b)



## Getting Started

### 1

<b>Simulink 3D Animation Product Description</b> .....	<b>1-2</b>
<b>Expected Background</b> .....	<b>1-3</b>
<b>Workflow for Building and Using Virtual Worlds</b> .....	<b>1-4</b>
Virtual Reality World Models of Dynamic Systems .....	<b>1-4</b>
Set up Your Working Environment .....	<b>1-4</b>
Build a Virtual Reality World .....	<b>1-5</b>
Link to a Virtual Reality World .....	<b>1-6</b>
View Dynamic System Simulations .....	<b>1-7</b>
Share Dynamic System Simulation Visualizations .....	<b>1-7</b>
<b>MATLAB Compiler Support</b> .....	<b>1-8</b>
<b>X3D Support</b> .....	<b>1-9</b>
X3D .....	<b>1-9</b>
Relationship of X3D and VRML .....	<b>1-9</b>
X3D Support in Simulink 3D Animation .....	<b>1-9</b>
Convert a VRML File to X3D Format .....	<b>1-10</b>
<b>Virtual Reality Modeling Language (VRML)</b> .....	<b>1-11</b>
Relationship of VRML and X3D .....	<b>1-11</b>
VRML .....	<b>1-11</b>
VRML Support .....	<b>1-11</b>
VRML Compatibility .....	<b>1-12</b>
Virtual World Coordinate System .....	<b>1-12</b>
VRML File Format .....	<b>1-13</b>
<b>Virtual Reality World and Dynamic System Examples</b> .....	<b>1-16</b>
Simulink Interface Examples .....	<b>1-16</b>
MATLAB Interface Examples .....	<b>1-25</b>

## Installation

### 2

<b>Set the Default Viewer</b> .....	<b>2-2</b>
<b>Set Simulink 3D Animation Preferences</b> .....	<b>2-5</b>
Simulink 3D Animation Preferences Dialog Box .....	<b>2-6</b>
3D World Editor Preferences Dialog Box .....	<b>2-7</b>
Canvas Preferences Dialog Box .....	<b>2-8</b>

Figure Appearance Preferences Dialog Box .....	2-8
Figure Rendering Preferences Dialog Box .....	2-9
Figure 2-D Recording Preferences Dialog Box .....	2-11
Figure Frame Capture Preferences .....	2-11
World Preferences Dialog Box .....	2-12
<b>Install V-Realm Editor .....</b>	<b>2-14</b>
V-Realm Editor Installation on Windows Platforms .....	2-14
V-Realm Builder Help .....	2-15
Uninstall V-Realm Builder .....	2-15
<b>Test the Viewer Installation .....</b>	<b>2-16</b>
Section Overview .....	2-16
Simulink Testing .....	2-16
MATLAB Testing .....	2-18

## Simulink Interface

### 3

<b>Connect Virtual Worlds and Models .....</b>	<b>3-2</b>
Output Simulation Data to a Virtual World .....	3-2
Input Virtual World Data to a Model .....	3-6
Change the Associated Virtual World .....	3-7
<b>Open a Viewer Window .....</b>	<b>3-9</b>
<b>Display Virtual World and Start Simulation .....</b>	<b>3-10</b>
<b>View Virtual World on Host Computer .....</b>	<b>3-12</b>
<b>View Virtual World Remotely .....</b>	<b>3-15</b>
<b>Modify Remote Virtual World Via Sensor Events .....</b>	<b>3-19</b>
<b>Interact with Generated Code .....</b>	<b>3-20</b>

## MATLAB Interface

### 4

<b>Create vrworld Object for a Virtual World .....</b>	<b>4-2</b>
<b>Open a Virtual World with MATLAB .....</b>	<b>4-3</b>
<b>Interact with a Virtual World with MATLAB .....</b>	<b>4-5</b>
Set Values for Nodes .....	4-5
Read Sensor Values Using MATLAB .....	4-7
<b>Close and Delete a vrworld Object .....</b>	<b>4-9</b>

<b>Animation Recording</b> .....	<b>4-10</b>
Recording Formats .....	<b>4-10</b>
Manual and Scheduled Animation Recording .....	<b>4-10</b>
<b>Define File Name Tokens</b> .....	<b>4-12</b>
Default File Name Format .....	<b>4-12</b>
Uses for File Name Tokens .....	<b>4-12</b>
<b>File Name Tokens</b> .....	<b>4-14</b>
<b>Manual 3-D Recording with MATLAB</b> .....	<b>4-16</b>
<b>Manual 2-D AVI Recording with MATLAB</b> .....	<b>4-18</b>
<b>Scheduled 3-D Recording with MATLAB</b> .....	<b>4-20</b>
<b>Scheduled 2-D AVI Recording with MATLAB</b> .....	<b>4-22</b>
<b>Record Animations for Unconnected Virtual Worlds</b> .....	<b>4-24</b>
<b>Play Animation Files</b> .....	<b>4-27</b>
Play Virtual World Animation Files .....	<b>4-27</b>
Play AVI Animation Files .....	<b>4-28</b>

## Build Virtual Reality Worlds

# 5

<b>Choose a Virtual World Editor</b> .....	<b>5-2</b>
Editors for Virtual Worlds .....	<b>5-2</b>
Set the Default Editor .....	<b>5-5</b>
<b>Build and Connect a Virtual World</b> .....	<b>5-8</b>
Introduction .....	<b>5-8</b>
Define the Problem .....	<b>5-8</b>
Add a Simulink 3D Animation Block .....	<b>5-9</b>
Open a New Virtual World .....	<b>5-10</b>
Add Nodes .....	<b>5-11</b>
Link to a Simulink Model .....	<b>5-17</b>
<b>Use Sensors</b> .....	<b>5-20</b>
Add Sensors to Virtual Worlds .....	<b>5-20</b>
Read Sensor Values .....	<b>5-21</b>
<b>Detect Object Collisions</b> .....	<b>5-23</b>
Set Up Collision Detection .....	<b>5-23</b>
Use Collision Detection Data in Models .....	<b>5-25</b>
Use Collision Detection in MATLAB .....	<b>5-27</b>
Use Collision Detection Data in Virtual Worlds .....	<b>5-27</b>
<b>Virtual World Data Types</b> .....	<b>5-30</b>
Field Data Types .....	<b>5-30</b>
Virtual World Data Class Types .....	<b>5-32</b>

<b>Simulink 3D Animation Textures</b> .....	<b>5-34</b>
<b>Add Sound to a Virtual World</b> .....	<b>5-35</b>
<b>Use CAD Models with the Simulink 3D Animation Product</b> .....	<b>5-36</b>
Use of CAD Designs .....	<b>5-36</b>
Import CAD Designs .....	<b>5-36</b>
Integrate the Imported Model Virtual World .....	<b>5-36</b>
<b>Import STL and Physical Modeling XML Files</b> .....	<b>5-38</b>
Results .....	<b>5-38</b>
<b>Import 3D Models from CAD Tools</b> .....	<b>5-40</b>
Level of Detail Considerations .....	<b>5-40</b>
Units Used in Exported Files .....	<b>5-40</b>
Coordinate System Used .....	<b>5-41</b>
Assembly Hierarchy .....	<b>5-41</b>
<b>Import VRML Models from CATIA Software</b> .....	<b>5-45</b>
CATIA Coordinate Systems .....	<b>5-45</b>
Settings That Affect the VRML Output .....	<b>5-45</b>
Level of Detail .....	<b>5-46</b>
VRML Export Filter Settings .....	<b>5-46</b>
VRML Models Exported from the CATIA Environment .....	<b>5-46</b>
Adjust the Resulting VRML Files .....	<b>5-48</b>
<b>Modify the CAD Model Virtual World</b> .....	<b>5-51</b>
Wrap Shape Objects with Transforms .....	<b>5-51</b>
Add DEF Names .....	<b>5-51</b>
Additional Virtual World Modifications .....	<b>5-52</b>
<b>Import Visual Representations of Robot Models</b> .....	<b>5-54</b>
Import a DAE File .....	<b>5-54</b>
Import a URDF File .....	<b>5-55</b>
Import an SDF File .....	<b>5-56</b>
Define Viewpoint to Make Imported Model Visible .....	<b>5-58</b>
Limitations .....	<b>5-58</b>
<b>Link to Simulink and Simscape Multibody Models</b> .....	<b>5-60</b>
Link the Virtual World to a Simulink Model .....	<b>5-60</b>
Initial Conditions .....	<b>5-61</b>
VR Placeholder and VR Signal Expander Blocks .....	<b>5-62</b>
Link to Simscape Multibody Models .....	<b>5-62</b>
Link to a MATLAB Model .....	<b>5-63</b>

## Using the 3D World Editor

### 6

<b>3D World Editor</b> .....	<b>6-2</b>
Supported Platforms .....	<b>6-2</b>
Use with Other Editors .....	<b>6-2</b>
VRML and X3D Support .....	<b>6-2</b>



Nodes, Library Objects, and Templates .....	6-2
<b>Open the 3D World Editor</b> .....	<b>6-5</b>
3D World Editor Is the Default Editor .....	6-5
Open an Empty Virtual World .....	6-5
Open a Saved Virtual World .....	6-5
3D World Editor Panes .....	6-6
Preferences for 3D World Editor Startup .....	6-7
<b>Create a Virtual World</b> .....	<b>6-9</b>
<b>Edit a Virtual World</b> .....	<b>6-11</b>
Add Objects .....	6-11
Copy and Paste a Node .....	6-12
Edit Object Properties .....	6-13
Document a Virtual World Using Comments .....	6-14
Display Event Fields .....	6-14
Expand and Collapse Nodes .....	6-14
Highlight Nodes and Virtual World Objects .....	6-15
Wrap Nodes as Children of Another Node .....	6-16
Remove Nodes .....	6-17
Save and Export Virtual World 3D Files .....	6-17
Edit VRML and X3D Scripts .....	6-18
<b>Reduce Number of Polygons for Shapes</b> .....	<b>6-20</b>
<b>Virtual World Navigation in 3D World Editor</b> .....	<b>6-21</b>
Specify Virtual World Rendering .....	6-21
Basic Navigation .....	6-21
Coordinate Axes Triad .....	6-21
View Panes .....	6-22
Pivot Point .....	6-23
View All/View Selected .....	6-23
<b>3D World Editor Library</b> .....	<b>6-26</b>
3D World Editor Library Objects .....	6-26
Add a Library Object .....	6-26
Guidelines for Using Custom Objects .....	6-27

## Viewing Virtual Worlds

# 7

<b>Virtual World Viewers</b> .....	<b>7-2</b>
Host and Remote Viewing .....	7-2
Comparison of Viewers .....	7-2
<b>Simulink 3D Animation Viewer</b> .....	<b>7-4</b>
What You Can Do with the Viewer .....	7-4
Viewer Uses MATLAB Figures .....	7-5
Set Viewer Appearance Preferences .....	7-6

<b>Open the Simulink 3D Animation Viewer</b> .....	7-7
Open from the VR Sink Block .....	7-7
Open from the Command Line .....	7-7
<b>Simulate with the Simulink 3D Animation Viewer</b> .....	7-8
Adjust Navigation Settings .....	7-8
<b>Specify Rendering Techniques</b> .....	7-9
Turn Off Rendering for Performance .....	7-14
<b>Navigate Using the Simulink 3D Animation Viewer</b> .....	7-15
Basic Navigation .....	7-15
Navigation Panel .....	7-16
Viewer Keyboard Shortcuts .....	7-18
Mouse Navigation .....	7-18
Navigation Control Menu .....	7-19
Change the Navigation Speed .....	7-19
Sensors Effect on Navigation .....	7-20
Display a Coordinate Axes Triad .....	7-20
Pivot Point .....	7-21
<b>Set Viewpoints</b> .....	7-23
Define Viewpoints .....	7-23
Reset Viewpoints .....	7-25
<b>Navigate Through Viewpoints</b> .....	7-26
<b>Record Offline Animations</b> .....	7-29
Animation Recording .....	7-29
Recording Formats .....	7-29
File Names .....	7-30
Start and Stop Animation Recording .....	7-30
Play Animation Files .....	7-31
Record 3-D Animation Files .....	7-31
Record in Audio Video Interleave (AVI) Format .....	7-31
Schedule Files for Recording .....	7-33
<b>Play Animations with Simulink 3D Animation Viewer</b> .....	7-35
<b>Configure Frame Capture Parameters</b> .....	7-36
<b>Capture Frames</b> .....	7-37
<b>Simulink 3D Animation Web Viewer</b> .....	7-38
<b>Open the Web Viewer</b> .....	7-39
Set up for Remote Viewing .....	7-39
Connect the Web Viewer .....	7-39
<b>Navigate Using the Web Viewer</b> .....	7-41
Display and Navigation .....	7-41
Keyboard Shortcuts .....	7-41
Web Viewer Preferences .....	7-42

<b>Listen to Sound in a Virtual World</b> .....	<b>7-43</b>
System Requirements for Sound .....	<b>7-43</b>
Listen to Sound .....	<b>7-43</b>
<b>View a Virtual World in Stereoscopic Vision</b> .....	<b>7-45</b>
Enable Stereoscopic Vision .....	<b>7-45</b>
Control Stereoscopic Effects .....	<b>7-45</b>
<b>Active Stereoscopic Vision Configuration</b> .....	<b>7-47</b>
Computer Platforms .....	<b>7-47</b>
Graphics Cards .....	<b>7-47</b>
Display Devices .....	<b>7-47</b>
Graphic Card Connection to Display Devices .....	<b>7-47</b>
Examples of Stereoscopic Vision Setups .....	<b>7-48</b>

## Simulink 3D Animation Stand-Alone Viewer

### 8

<b>Orbisnap Viewer</b> .....	<b>8-2</b>
What Is Orbisnap? .....	<b>8-2</b>
<b>Install Orbisnap</b> .....	<b>8-3</b>
Section Overview .....	<b>8-3</b>
System Requirements .....	<b>8-3</b>
Copying Orbisnap to Another Location .....	<b>8-3</b>
Adding Shortcuts or Symbolic Links .....	<b>8-3</b>
<b>Start Orbisnap</b> .....	<b>8-5</b>
<b>Orbisnap Interface</b> .....	<b>8-6</b>
Menu Bar .....	<b>8-7</b>
Toolbar .....	<b>8-7</b>
Navigation Panel .....	<b>8-8</b>
<b>Navigate Using Orbisnap</b> .....	<b>8-9</b>
<b>View Animations or Virtual Worlds with Orbisnap</b> .....	<b>8-12</b>
<b>View Virtual Worlds Remotely with Orbisnap</b> .....	<b>8-13</b>

## Blocks

### 9

## Functions

### 10



# Getting Started

---

- “Simulink 3D Animation Product Description” on page 1-2
- “Expected Background” on page 1-3
- “Workflow for Building and Using Virtual Worlds” on page 1-4
- “MATLAB Compiler Support” on page 1-8
- “X3D Support” on page 1-9
- “Virtual Reality Modeling Language (VRML)” on page 1-11
- “Virtual Reality World and Dynamic System Examples” on page 1-16

## **Simulink 3D Animation Product Description**

### **Visualize dynamic system behavior in a virtual reality environment**

Simulink 3D Animation links Simulink models and MATLAB® algorithms to 3D graphics objects in virtual reality scenes. You can animate a virtual world by changing position, rotation, scale, and other object properties during desktop or real-time simulation. You can also sense collisions and other events in the virtual world and feed them back into your MATLAB and Simulink algorithms. Video from virtual cameras can be streamed to Simulink for processing.

Simulink 3D Animation includes editors and viewers for rendering and interacting with virtual scenes. With the 3D World Editor, you can import CAD and URDF file formats as well as author detailed scenes assembled from 3D objects. The 3D world can be viewed immersively using stereoscopic vision. You can incorporate multiple 3D scene views inside MATLAB figures, and interact with the virtual world using a force-feedback joystick, space mouse, or other hardware device. Simulink 3D Animation supports X3D, an ISO standard file format and run-time architecture for representing and communicating with 3D scenes and objects.

## Expected Background

To help you effectively read and use this guide, here is a brief description of the chapters and a suggested reading path. Generally, you can assume that Simulink 3D Animation software on the Apple Mac OS X platform works as described for the UNIX®/Linux® platforms.

The documentation assumes that you are already familiar with:

- MATLAB product, to write scripts and functions with MATLAB code, and to use functions with the command-line interface
- Simulink and Stateflow charts products to create models as block diagrams and simulate those models
- X3D or VRML, to create or otherwise provide virtual worlds or three-dimensional scenes to connect to Simulink or MATLAB software

### See Also

### Related Examples

- “Workflow for Building and Using Virtual Worlds” on page 1-4

### More About

- “Simulink 3D Animation Product Description” on page 1-2

## Workflow for Building and Using Virtual Worlds

In this section...
“Virtual Reality World Models of Dynamic Systems” on page 1-4
“Set up Your Working Environment” on page 1-4
“Build a Virtual Reality World” on page 1-5
“Link to a Virtual Reality World” on page 1-6
“View Dynamic System Simulations” on page 1-7
“Share Dynamic System Simulation Visualizations” on page 1-7

### Virtual Reality World Models of Dynamic Systems

The Simulink 3D Animation product is a solution for interacting with virtual reality world models of dynamic systems over time. It extends the capabilities of your virtual world and Simulink, Simscape Multibody, and MATLAB software into the world of virtual reality graphics. The product provides a complete authoring, development, and working environment for carrying out 3-D visual simulations.

To use virtual reality worlds to visualize dynamic system simulations, perform the following tasks:

- “Set up Your Working Environment” on page 1-4
- “Build a Virtual Reality World” on page 1-5
- “Link to a Virtual Reality World” on page 1-6
- “View Dynamic System Simulations” on page 1-7
- “Share Dynamic System Simulation Visualizations” on page 1-7

As you refine your visualization, you often perform some of these tasks iteratively.

To work through an example that illustrates the building, linking, and viewing a virtual world, see “Build and Connect a Virtual World” on page 5-8.

### Set up Your Working Environment

Install the Simulink 3D Animation software in your MATLAB environment to build virtual reality worlds and to visualize dynamic simulations modeled in MATLAB, Simulink, or Simscape Multibody. If your computer does not already have a graphics card with hardware 3-D acceleration, consider installing such a card to enhance graphics performance.

You build and view the virtual reality world models using VRML (Virtual Reality Modeling Language) or X3D (**X**tensible **3D**).

In addition to the installed 3D World Editor (the default editor), you can configure your environment to use:

- The Ligos® V-Realm Builder, which is included in the Simulink 3D Animation software for Windows® platforms.
- Any third-party virtual world editor
- The MATLAB editor or a third-party text editor



In addition to the installed Simulink 3D Animation viewer (the default), you can use one of these viewers to display your virtual reality worlds:

- Simulink 3D Animation Web Viewer
- Orbisnap, on a client computer

To help decide which 3D virtual world editor and viewer to use, see “Choose a Virtual World Editor” on page 5-2 and “Virtual World Viewers” on page 7-2.

Use joystick and space mouse input devices to provide input for dynamic simulation visualizations.

### **TCP/IP Connection**

The Simulink 3D Animation product uses a TCP/IP connection to a virtual reality world client for communicating with the Simulink 3D Animation Viewer, as well as connecting to an HTML5-enabled web browser. You can verify the TCP/IP connection between the host and client computers by using the `ping` command from a command-line prompt. If there are problems, fix the TCP/IP protocol settings according to the documentation for your operating system.

### **LD\_LIBRARY\_PATH Environment Variable for UNIX**

If your system does not have the OpenGL® software properly installed when you run the Simulink 3D Animation Viewer, you can see a MATLAB error message like the following:

```
Invalid MEX-file 'matlab/toolbox/sl3d/sl3d/vrsfunc.mexglx':
libGL.so: cannot open shared object file
```

If you see an error like this, set the `LD_LIBRARY_PATH` environment variable.

If the `LD_LIBRARY_PATH` environment variable already exists, use a line similar to this code to add the new path to the existing one:

```
setenv LD_LIBRARY_PATH
matlabroot/sys/opengl/lib/<PLATFORM>:$LD_LIBRARY_PATH
```

If the `LD_LIBRARY_PATH` environment variable does not already exist, use a line similar to this code:

```
setenv LD_LIBRARY_PATH
matlabroot/sys/opengl/lib/<PLATFORM>
```

In both cases, `<PLATFORM>` is the UNIX platform you are using.

## **Build a Virtual Reality World**

Use the virtual world editor or other editor to build a virtual reality world. A non-VRML or non-X3D CAD model created with another tool can be a good basis for a virtual reality world to use with Simulink 3D Animation. You can convert some CAD models to a VRML or X3D model.

You can use advanced features of the Simulink 3D Animation product such as:

- Viewpoints, to highlight points of interest for quick browsing of a virtual reality world
- Sensors, to input virtual reality world values to Simulink models

For an overview of VRML and details about supported VRML features, see “Virtual Reality Modeling Language (VRML)” on page 1-11. You can also use X3D, which provides several extensions,

including additional nodes, fields, encoding, scene access interfaces, additional rendering control, and geospatial support. For details, see “X3D Support” on page 1-9.

As you add nodes with the 3D World Editor on page 6-2, you can use the viewer pane to see the virtual world that you are creating.

For a step-by-step example of building a virtual reality world with the 3D World Editor, see “Build and Connect a Virtual World” on page 5-8.

## Link to a Virtual Reality World

To use a dynamic system simulation to drive a virtual reality world, connect the virtual world to one of these systems or objects:

- Simulink model
- Simscape Multibody model
- MATLAB virtual world object

### Connect to Simulink Model

The Simulink 3D Animation library provides blocks to connect Simulink signals to virtual worlds. This connection lets you visualize your model as a three-dimensional animation. Simulink provides communication for control and manipulation of virtual reality objects, using Simulink 3D Animation blocks. For details, see “Connect Virtual Worlds and Models” on page 3-2.

After you include these blocks in a Simulink diagram, you can select a virtual world and connect Simulink signals to the virtual world. The software automatically scans a virtual world for available nodes that the Simulink software can drive.

All the node properties are listed in a hierarchical tree-style viewer. You select the degrees of freedom to control from within the Simulink interface. After you close a Block Parameters dialog box, the Simulink software updates the block with the inputs and outputs corresponding to selected nodes in the virtual world. After connecting these inputs to appropriate Simulink signals, you can view the simulation with a virtual world viewer.

### Connect to Simscape Multibody Model

You can use the Simulink 3D Animation product to view the behavior of a model created with the Simscape Multibody software. First, build a model of a machine in the Simulink interface using Simscape Multibody blocks. Then create a detailed picture of your machine in a virtual world. Connect this world to the Simscape Multibody body sensor outputs and view the behavior of the bodies in a virtual world viewer. For details, see “Link to Simulink and Simscape Multibody Models” on page 5-60.

### Connect to MATLAB Virtual World Object

Simulink 3D Animation software provides a flexible MATLAB interface to virtual reality worlds. After creating MATLAB objects and associating them with a virtual world, you can control the virtual world by using functions and methods. MATLAB provides communication for control and manipulation of virtual reality objects using MATLAB objects. For details about interacting between MATLAB and virtual reality worlds, see “Interact with Virtual Reality Worlds”.

In MATLAB, you can set positions and properties of virtual world objects, create callbacks from graphical interfaces, and map data to virtual world objects. You can also view the virtual world with a viewer, determine its structure, and assign new values to all available nodes and their fields.

The software includes functions for retrieving and changing the virtual world properties and for saving the virtual world 3D files corresponding to the actual structure of a virtual world.

## View Dynamic System Simulations

After you connect the virtual world to the model, use a virtual world viewer to view the virtual world representation of the dynamic system simulation.

- In Simulink and Simscape Multibody, simulate the model that is connected to the virtual reality world.
- In MATLAB, use the `view` function to view a `vrworld` object that the MATLAB code updates with data values.

While running a simulation, you can change the positions and properties of virtual world objects.

For information about using virtual world viewers to navigate a virtual reality world, see “View Dynamic System Simulations”.

## Share Dynamic System Simulation Visualizations

You can share dynamic system simulation results with others.

- Capture animation frame snapshots or record animations for video viewing. See “Capture Frames” on page 7-37 and “Share Visualizations”.
- Use a client-server configuration. In addition to the single computer configuration (when MATLAB, Simulink, and the virtual reality representations run on the same host computer). In a client-server configuration, an Orbisnap viewer on a remote client can connect to the server host on which Simulink 3D Animation software is running. This configuration allows others to view an animated virtual world remotely. Multiple clients can connect to one server. See “Orbisnap Viewer” on page 8-2.
- Use the MATLAB Compiler™ to take MATLAB files as input and generate redistributable, standalone applications that include Simulink 3D Animation functionality, including the Simulink 3D Animation Viewer. See “MATLAB Compiler Support” on page 1-8

## See Also

### Functions

`vredit` | `vrgetpref` | `vrjoystick` | `vrlib` | `vrsetpref` | `vrspacemouse`

### Blocks

VR Sink | VR Source

## Related Examples

- “Virtual Reality World and Dynamic System Examples” on page 1-16
- “Build Virtual Reality Worlds”

## MATLAB Compiler Support

To use the MATLAB Compiler to produce standalone applications, create a MATLAB file that uses the MATLAB interface for the Simulink 3D Animation product (for example, creating, opening, and closing a `vrworld` object). Then use the MATLAB Compiler product.

Standalone applications that include Simulink 3D Animation functionality have the following limitations:

- No Simulink software support, which results in no access to the Simulink 3D Animation Simulink library (`vrlib`).
- No Simulink 3D Animation server, which results in no remote connection for the Orbisnap viewer
- No animation recording ability
- No editing world ability
- The following Simulink 3D Animation Viewer features cannot be used in standalone applications:
  - **File > Open in Editor**
  - **Recording** menu
  - **Simulation** menu
  - **Help** access

### See Also

### Related Examples

- “Interact with Virtual Reality Worlds”

## X3D Support

### In this section...

“X3D” on page 1-9

“Relationship of X3D and VRML” on page 1-9

“X3D Support in Simulink 3D Animation” on page 1-9

“Convert a VRML File to X3D Format” on page 1-10

## X3D

The X3D (**X**tensible **3D**) ISO standard is an open standards file format and runtime architecture for representing and communicating 3D scenes and objects. X3D has a rich set of componentized features that you can customize. You can use X3D in applications such as engineering and scientific visualization, CAD and architecture, medical visualization, training and simulation, multimedia, entertainment, and education.

For information about supported X3D specification, see ISO/IEC 19775-1:2013. For information about supported X3D encoding, see ISO/IEC 19776-1.3:201x and ISO/IEC 19776-2.3:201x.

## Relationship of X3D and VRML

X3D is the successor of the VRML 97 standard (see “Virtual Reality Modeling Language (VRML)” on page 1-11). X3D and VRML share many similar approaches, such as their coordinate systems and the description of objects using nodes and their fields. X3D provides several extensions, including additional nodes, fields, encoding, scene access interfaces, additional rendering control, and geospatial support. VRML97 is still a widely supported 3D format for tools and viewers, and is a direct subset of X3D. Many CAD tools and 3D editors support import from and export to the X3D format.

Because many 3D virtual world tools and CAD tools have adopted X3D, Simulink 3D Animation software provides both X3D and VRML support. VRML97 is the default virtual world file format.

## X3D Support in Simulink 3D Animation

You can use XML encoded (.x3d files) and Classic VRML encoded (.x3dv files) X3D file formats. X3D support is for versions from version 3.0 up to version 3.3. Support is for X3D files that contain components that comply to the Immersive profile.

You can use Simulink blocks and MATLAB command-line interfaces to create and access virtual worlds.

### X3D Support Limitations

In the 3D World Editor, you can edit only VRML and VRML-compliant X3D files (files that contain only X3D features that have VRML97 counterparts).

Simulink 3D Animation does not support X3D for Ligos V-Realm Builder.

The X3D support has these limitations:

- No support for binary-encoded files (.x3db).
- The Simulink 3D Animation Web Viewer supports only X3D files that contain nodes complying to the HTML profile specified by the X3DOM developer community.
- You can use the `stl2vrm` function to import CAD models in STL format (.stl files) to X3D format (.x3d or .x3dv files). However, other methods that Simulink 3D Animation provides for converting CAD models do not support conversion to X3D format.
- You cannot inline X3D files (.x3d or .x3dv).
- No support for the scene-access interface specified by ISO/IEC 19775-2:201x. To access virtual worlds, use Simulink blocks or the MATLAB commands.
- `LineProperties` node support is limited to solid lines.
- The engine ignores `UNIT` and additional `COMPONENT` statements and elements.
- A `PROTO` node cannot have a VRML file (.wrl) that references an X3D file (.x3d or .x3dv).

## Convert a VRML File to X3D Format

You can save VRML (.wrl) files as X3D format files. The conversion process determines whether the X3D file is an .x3d or x3dv file.

This example code converts a VRML file to X3D format:

```
w = vrworld('octavia_scene.wrl');
open(w);

% save to XML encoding
save(w, 'octavia_scene.x3d');

% save to VRML syntax encoding
save(w, 'octavia_scene.x3dv');
```

## See Also

### Related Examples

- “Use CAD Models with the Simulink 3D Animation Product” on page 5-36
- “Workflow for Building and Using Virtual Worlds” on page 1-4
- “Virtual Reality World and Dynamic System Examples” on page 1-16

### More About

- “Virtual Reality Modeling Language (VRML)” on page 1-11
- “Expected Background” on page 1-3

# Virtual Reality Modeling Language (VRML)

## In this section...

“Relationship of VRML and X3D” on page 1-11  
 “VRML” on page 1-11  
 “VRML Support” on page 1-11  
 “VRML Compatibility” on page 1-12  
 “Virtual World Coordinate System” on page 1-12  
 “VRML File Format” on page 1-13

## Relationship of VRML and X3D

The X3D (**X**tensible **3D**) interface is the successor to the VRML (Virtual Reality Modeling Language) interface. The X3D interface supports VRML features. X3D also provides several extensions to VRML.

For details, see “X3D Support” on page 1-9.

## VRML

You can use the Virtual Reality Modeling Language (VRML) to display three-dimensional objects in a VRML viewer. Simulink 3D Animation supports VRML97.

VRML provides an open and flexible platform for creating interactive three-dimensional scenes (virtual worlds). Several VRML97-enabled browsers are available on several platforms. Also, you can choose from several VRML authoring tools. In addition, graphical software packages (CAD, visual art, and so on) offer VRML97 import/export features.

The Simulink 3D Animation product uses VRML97 technology for 3-D visualization.

## VRML Support

The Virtual Reality Modeling Language (VRML) is an ISO standard that is open, text-based, and uses a WWW-oriented format. You use VRML to define a virtual world that you can display with a virtual world viewer and connect to a Simulink model.

The Simulink 3D Animation software uses many of the advanced features defined in the current VRML97 specification. The standard is ISO/IEC 14772-1:1997, available from <http://www.web3d.org/documents/specifications/14772/V2.0/part1/javascript.html>. This format includes a description of 3-D scenes, sounds, internal actions, and WWW anchors.

The software analyzes the structure of the virtual world, determines what signals are available, and makes them available from the MATLAB and Simulink environment.

Simulink 3D Animation software ensures that the changes made to a virtual world are reflected in the MATLAB and Simulink interfaces. If you change the viewpoint in your virtual world, this change occurs in the `vrworld` object properties in MATLAB and Simulink interfaces.

The software includes functions for retrieving and changing virtual world properties.

**Note** Some VRML worlds are automatically generated in VRML1.0. However, the Simulink 3D Animation product does not support VRML1.0. Save these worlds in the current standard for VRML, VRML97.

For PC platforms, you can convert VRML1.0 worlds to VRML97 worlds by opening the worlds in Ligos V-Realm Builder and saving them. V-Realm Builder is shipped with the PC version of the software. Other commercially available software programs can also perform the VRML1.0 to VRML97 conversion.

---

## VRML Compatibility

The Simulink 3D Animation product currently supports most features of VRML97, with the following limitations:

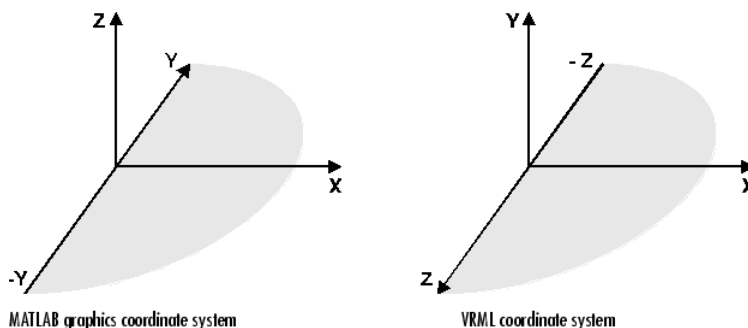
- The Simulink 3D Animation server ignores the VRML `Script` node, but it passes the node to the VRML Viewer. Passing the node allows you to run VRML scripts on the viewer side. You cannot run them on the Simulink 3D Animation server.
- In keeping with the VRML97 specification, the Simulink 3D Animation Viewer ignores BMP files. As a result, VRML scene textures sometimes display improperly in the Simulink 3D Animation Viewer. To display scene textures properly, replace all BMP texture files in a VRML scene with PNG, JPG, or GIF equivalents.

For a complete list of VRML97 nodes, refer to the VRML97 specification.

## Virtual World Coordinate System

Take coordinate systems into account when you want to:

- Display a virtual world object in a particular position.
- Move a virtual world.
- Export non-VRML models from CAD tools (including CATIA) and robot visual representations (URDF files) to use with Simulink 3D Animation .
- Have a virtual world interact with MATLAB or Simulink.

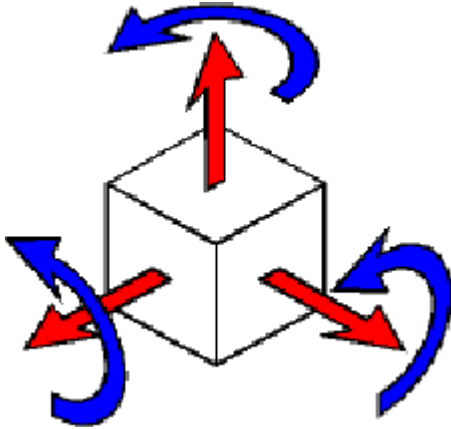


The VRML coordinate system is different from the MATLAB and Aerospace Blockset™ coordinate systems. VRML uses the *world coordinate system*: the y-axis points upward and the z-axis places objects nearer or farther from the front of the screen. The larger the z-axis value, the closer the object appears to the viewer. Understanding the coordinate system is important when you interact with different coordinate systems. Simscape Multibody uses the same coordinate system as VRML.



Here are some key VRML coordinate system concepts:

- Rotation angles — In VRML, rotation angles are defined using the *right-hand rule*. Imagine your right hand holding an axis while your thumb points in the direction of the axis toward its positive end. Your four remaining fingers point in a counterclockwise direction. This counterclockwise direction is the positive rotation angle of an object moving around that axis.



- Child objects — In the hierarchical structure of a VRML file, specify the position and orientation of child objects relative to the parent object. The parent object has its local coordinate space defined by its own position and orientation. Moving the parent object also moves the child objects relative to the parent object.
- Measurement units — All lengths and distances are measured in *meters*, and all angles are measured in *radians*.

Simulink 3D Animation provides a set of functions that can help you convert between different representations of orientation in space. An example of a coordinate conversion function is `vrrotmat2vec`, which converts a rotation from a matrix to an axis-angle representation.

For an example of using global coordinates in a Simulink 3D Animation model, see “Manipulator Moving a Load with Use of Global Coordinates”.

## VRML File Format

You need not have any substantial knowledge of the VRML format to use the VRML authoring tools to create virtual worlds. However, a basic knowledge of VRML scene description helps you create virtual worlds more effectively. A basic knowledge also gives you a good understanding of how you can control the virtual world elements using Simulink 3D Animation software.

For more information, see the VRML97 Reference at <https://www.web3d.org>. Many specialized VRML books can help you understand VRML concepts and create your own virtual worlds. For more information about the VRML, refer to an appropriate third-party VRML book.

VRML uses a hierarchical tree structure of objects (nodes) to describe a 3-D scene. Every node in the tree represents some functionality of the scene. There are many different types of nodes. Some of them are *shape nodes* (representing real 3-D objects), and some of them are *grouping nodes* used for holding child nodes. Here are some example nodes:

- Box — Represents a box in a scene.

- **Transform** — Defines position, scale, scale orientation, rotation, translation, and children of its subtree (grouping node).
- **Material** — Corresponds to material in a scene.
- **DirectionalLight**— Represents lighting in a scene.
- **Fog** — Allows you to modify the environment optical properties.
- **ProximitySensor** — Brings interactivity to VRML97. This node generates events when you enter, exit, and move within the defined region in space.

Each node contains a list of fields that hold values defining parameters for its function.

Nodes can be placed in the top level of a tree or as children of other nodes in the tree hierarchy. When you change a value in the field of a certain node, all nodes in its subtree are affected. This feature allows you to define relative positions inside complicated compound objects.

You can mark every node with a specific name by using the keyword **DEF** in the VRML scene code. For example, the statement **DEF MyNodeName Box** sets the name for this box node to **MyNodeName**. You can access the fields of only those nodes that you name in a virtual world.

In the following example of a simple VRML file, two graphical objects are modeled in a 3-D scene. A flat box with a red ball above it represents the floor. The VRML file is a readable text file that you can write in any text editor.

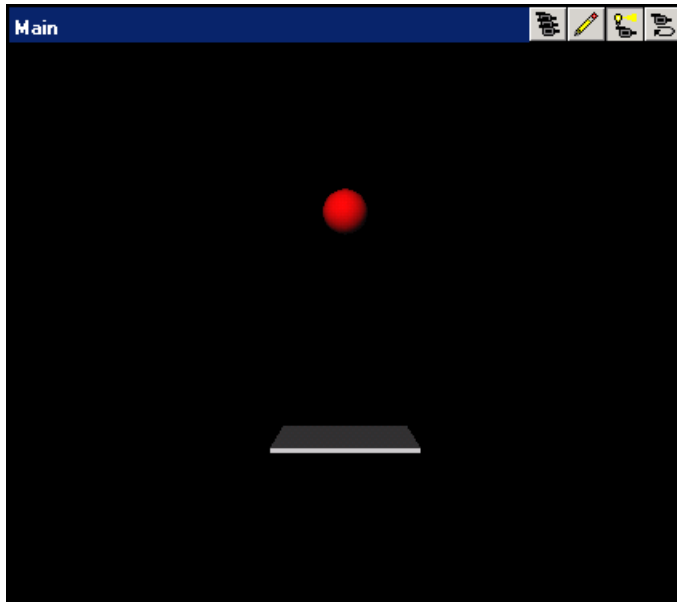
```
#VRML V2.0 utf8
# This is a comment line
WorldInfo {
  title "Bouncing Ball"
}
Viewpoint {
  position 0 5 30
  description "Side View"
}
DEF Floor Box {
  size 6 0.2 6
}
DEF Ball Transform {
  translation 0 10 0
  children Shape {
    appearance Appearance {
      material Material {
        diffuseColor 1 0 0
      }
    }
    geometry Sphere {
  }
}
}
```

The first line is the VRML header line. Every VRML file must start with this header line. It indicates that the file is a VRML 2 file and that the text objects in the file are encoded according to the UTF8 standard. You use the number sign (#) to comment VRML worlds. A VRML viewer ignores everything on a line after the # sign is ignored, except for the first header line.

Most of the box properties are left at their default values - distance from the center of the coordinate system, material, color, and so on. Only the name **Floor** and the dimensions are assigned to the box. To be able to control the position and other properties of the ball, it is defined as a child node of a

Transform type node. Here, the default unit sphere is assigned a red color and a position 10 m above the floor. In addition, the virtual world title is used by VRML viewers to distinguish between virtual worlds. A suitable initial viewpoint is defined in the virtual world VRML file.

When displayed in a VRML viewer, you see the floor and red ball.



## See Also

### Related Examples

- "Workflow for Building and Using Virtual Worlds" on page 1-4
- "Virtual Reality World and Dynamic System Examples" on page 1-16
- "Add Sensors to Virtual Worlds" on page 5-20

### More About

- "X3D Support" on page 1-9
- "Expected Background" on page 1-3

## Virtual Reality World and Dynamic System Examples

**In this section...**

“Simulink Interface Examples” on page 1-16

“MATLAB Interface Examples” on page 1-25

### Simulink Interface Examples

For all the examples that have a Simulink model, use the following procedure to run the example and view the model:

- 1 In the MATLAB Command Window, enter the name of a Simulink model. For example, enter:

```
vrbounce
```

A Simulink window opens with the block diagram for the model. By default, a virtual world also opens in the Simulink 3D Animation Viewer or your HTML5-enabled web browser. If you close the virtual world window, double-click the VR Sink block to display it again.

**Note** If the viewer does not open, double-click the VR Sink block in the Simulink model. In the Simulink 3D Animation Viewer, from the **Simulation** menu, click **Block Parameters**. A Block Parameters dialog box opens. The **Open viewer automatically** check box should be selected by default. When you double-click the VR Sink block, this selection enables the virtual world window to open.

- 2 In the Simulink window, from the **Simulation** menu, click **Run**. (Alternatively, in the Simulink 3D Animation Viewer, from the **Simulation** menu, click **Start**.)

A simulation starts running, and the virtual world is animated using signal data from the simulation.

The following table lists the Simulink examples provided with the Simulink 3D Animation product. Descriptions of the examples follow the table.

Example	Simulink Coder™ Ready	VR Sink	VR Source	Joystick	Space Mouse
sl3dex_rigidbodytree		X			
sl3dex_uav		X	X		X
vrbounce	X	X			
vrcrane_joystick		X		X	
vrcrane_panel		X	X		
vrcrane_traj	X	X			
vrlights	X	X			
vrmaglev		X	X		
vrmaglev_sldrt	X	X			
vrmanipul		X			X
vrmanipul_global		X	X		

Example	Simulink Coder™ Ready	VR Sink	VR Source	Joystick	Space Mouse
vrmemb1	X	X			
vrmorph	X	X			
vr_octavia	X	X			
vr_octavia_2cars		X			
vr_octavia_graphs		X			
vr_octavia_mirror		X			
vr_octavia_video		X			
vrdemo_panel		X	X		
vrpend	X	X			
vrplanets	X	X			
vrtkoff	X	X			
vrtkoff_trace		X			
vrtkoff_hud		X			
vrcollisions		X	X		
vrcollisions_lidar		X	X		
vrmaze		X	X		

### UAV Competition Example (sl3dex\_uav)

The sl3dex\_uav example shows how virtual collision sensors can be used to interactively control the simulation and to change the appearance of virtual world objects using Simulink® 3D Animation™. The example represents a simple unmanned aerial vehicle (UAV) challenge.

The UAV competition scene is based on the IMAV Flight Competition held in 2013 in Toulouse, France. ( <http://www.imav2013.org> )

### Rigid Body Tree Visualization (sl3dex\_rigidbodytree)

The sl3dex\_rigidbodytree example demonstrates the functionality of the Simulink 3D Animation VR RigidBodyTree block. This example requires Robotics System Toolbox™

The VR RigidBodyTree block inserts visual representation of a Robotics System Toolbox RigidBodyTree object in the virtual world and displays it in the virtual reality viewer. During simulation, the rigid body tree is subsequently animated according to the configuration defined in the Config input.

In this example, the manipulator configuration is provided by the Robotics System Toolbox Inverse Kinematics block. You can use the sliders to change the robot end-effector position and orientation about one axis.

### Bouncing Ball Example (vrbounce)

The vrbounce example represents a ball bouncing from a floor. The ball deforms as it hits the floor, keeping the volume of the ball constant. The deformation is achieved by modifying the scale field of the ball.

### **Portal Crane with Joystick Control (vrcrane\_joystick)**

The `vrcrane_joystick` example illustrates how a Simulink model can interact with a virtual world. The portal crane dynamics are modeled in the Simulink interface and visualized in virtual reality. The model uses the Joystick Input block to control the setpoint. Joystick 3 axes control the setpoint position and button 1 starts the crane. This example requires a standard joystick with at least three independent axes connected to the PC.

To minimize the number of signals transferred between the Simulink model and the virtual reality world, and to keep the model as simple and flexible as possible, only the minimum set of moving objects properties are sent from the model to the VR Sink block. All other values that are necessary to describe the virtual reality objects movement are computed from this minimum set using VRMLScript in the associated virtual world 3D file.

For details on how the crane model hierarchy and scripting logic is implemented, see the associated commented virtual world 3D file `portal_crane.wrl`.

### **Virtual Control Panel (vrdemo\_panel)**

The `vrdemo_panel` example shows the use of sensing objects that are available in the 3D World Editor Components library. These objects combine virtual world sensors with logic that changes their visual appearance based on user input. The sensor values can be read into Simulink by the VR Source block. The logic is implemented using VRML Scripts and Routes.

The control panel contains a pushbutton, switch button, toggle switch, and a 2-D setpoint selection area. Outputs of these elements are read into a Simulink model and subsequently displayed using standard sinks, or used as inputs of blocks that control back some objects in the virtual world.

Pushbutton, switch button, and toggle switches have the state outputs, which are of boolean type. Their values are displayed using the Scope.

Two outputs of the 2D setpoint area are used to achieve the following behavior. The value of the "SetPoint\_Changed" eventOut is continuously updated when the pointer is over the sensor area. This value is triggered by the second output - "isActive" that is true only on clicking the pointer button. Triggered value - coordinates of the active point on the sensor plane are displayed using the XY Graph and sent back to the virtual world in two ways: as a position of green cone marker and as text that the VR Text Output block displays on the control panel.

### **Portal Crane with Predefined Trajectory Example (vrcrane\_traj)**

The `vrcrane_traj` example is based on the `vrcrane_joystick` example, but instead of interactive control, it has a predefined load trajectory. The `vrcrane_traj` model illustrates a technique to create the visual impression of joining and splitting moving objects in the virtual world.

A crane magnet attaches the load box, moves it to a different location, then releases the box and returns to the initial position. This effect is achieved using an additional, geometrically identical shadow object that is placed as an independent object outside of the crane objects hierarchy. At any given time, only one of the Load or Shadow objects is displayed, using two Switch nodes connected by the ROUTE statement.

After the crane moves the load to a new position, at the time of the load release, a VRMLScript script assigns the new shadow object position according to the current Load position. The Shadow object becomes visible. Because it is independent from the rest of the crane moving parts hierarchy, it stays at its position as the crane moves away.

### Lighting Example (vrlights)

The `vrlights` example uses light sources. In the scene, you can move Sun (modeled as `DirectionalLight`) and Lamp (modeled as `PointLight`) objects around the Simulink model. This movement creates the illusion of changes between day and night, and night terrain illumination. The associated virtual world 3D file defines several viewpoints that allow you to observe gradual changes in light from various perspectives.

### Magnetic Levitation Model Example (vrmaglev)

The `vrmaglev` example shows the interaction between dynamic models in the Simulink environment and virtual worlds. The Simulink model represents the HUMUSOFT® CE 152 Magnetic Levitation educational/presentation scale model. The plant model is controlled by a PID controller with feed-forward to cope with the nonlinearity of the magnetic levitation system. To more easily observe and control the ball, set the virtual world viewer to the Camera 3 viewpoint.

You can set the ball position setpoint in two ways:

- Using a Signal Generator block
- Clicking in the virtual reality scene at a position that you want

To achieve a dragging effect, use the `PlaneSensor` attached to the ball geometry with its output restricted to  $\langle 0,1 \rangle$  in the vertical coordinate and processed by the VR Sensor Reader block. The `vrexтин` S-function provides the data connection.

For more details on how to read values from virtual worlds programmatically, see “Add Sensors to Virtual Worlds” on page 5-20.

### Magnetic Levitation Model for Simulink Desktop Real-Time Example (vrmaglev\_sldrt)

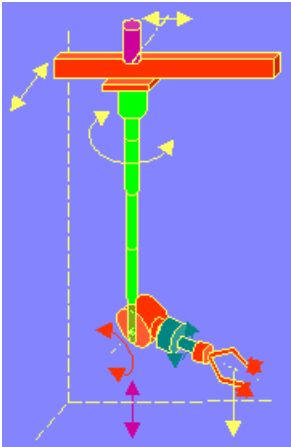
In addition to the `vrmaglev` example, the `vrmaglev_sldrt` example works directly with the actual CE 152 scale model hardware in real time. This model to work with the HUMUSOFT MF 624 data acquisition board, and Simulink Coder and Simulink Desktop Real-Time™ software. However, you can adapt this model for other targets and acquisition boards. A digital IIR filter, from the DSP System Toolbox™ library, filters the physical system output. You can bypass the physical system by using the built-in plant model. Running this model in real time is an example showing the capabilities of the Simulink product in control systems design and rapid prototyping.

After enabling the remote view in the VR Sink block dialog box, you can control the Simulink model even from another (remote) client computer. This control can be useful for distributing the computing power between a real-time Simulink model running on one machine and the rendering of a virtual reality world on another machine.

To work with this model, use as powerful a machine as possible or split the computing and rendering over two machines.

### Manipulator with Space Mouse Example (vrmanipul)

The `vrmanipul` example illustrates the use of Simulink 3D Animation software for virtual reality prototyping and testing the viability of designs before the implementation phase. Also, this example illustrates the use of a space mouse input for manipulating objects in a virtual world. You must have a space mouse input to run this example.



The virtual reality model represents a nuclear hot chamber manipulator. It is manipulated by a simple Simulink model containing the Space Mouse Input block. This model uses all six degrees of freedom of the space mouse for manipulating the mechanical arm, and uses mouse button 1 to close the grip of the manipulator jaws.

A space mouse is an input device with six degrees of freedom. It is useful for navigating and manipulating objects in a virtual world. A space mouse is also suitable as a general input device for Simulink models. You can use a space mouse for higher performance applications and user comfort. Space mouse input is supported through the Space Mouse Input block, which is included in the Simulink 3D Animation block library for the Simulink environment.

The Space Mouse Input block can operate in three modes to cover the most typical uses of such a device in a three-dimensional context:

- Speeds
- Positions
- Viewpoint coordinates

### **Manipulator Moving a Load with Use of Global Coordinates (vrmanipul\_global)**

The `vrmanipul_global` example illustrates the use of global coordinates in Simulink 3D Animation models. You can use global coordinates in a model in many ways, including:

- Object tracking and manipulation
- Simple collision detection
- Simulation of haptic effects

The VR Source block supports using global coordinates for objects in a virtual world. For each Transform in the scene, the tree view in the VR Source block parameter dialog box displays the **Extensions** branch. In that branch, you can select `translation_abs` and `rotation_abs` fields. Fields with the `_abs` suffix contain the object's global coordinates. The fields without the `_abs` suffix input their data into Simulink model object's local coordinates (relative to their parent objects in model hierarchy).

The virtual reality model represents a nuclear hot chamber manipulator. The manipulator moves the load from one gray cylindrical platform to another. The trajectory for the manipulator end-effector is predefined using the Signal Builder. Each part of manipulator arm is independently actuated using



decomposed trajectory components, with the help of VR Expander blocks (see the VR Transformations subsystem).

The VR Source block in the virtual scene tree on the left captures global coordinates of all objects important for load manipulation:

- Manipulator grip reference point (center of the clamp)
- Destination reference point
- Initial position of the load

The manipulator grip position results from complex movement of manipulator arm parts that form hierarchical structure. Generally it is very difficult to compute global coordinates for such objects affected by hierarchical relations in the scene. However, Simulink 3D Animation provides an easy way to read the global coordinates of objects affected by hierarchical relations into a Simulink model.

Based on having the global coordinates of all of the important objects, you can implement a simple manipulator control logic.

### **Rotating Membrane Example (vrmemb1)**

The vrmemb1 example is similar to the vrmemb example, but in the vrmemb1 example the associated virtual world is driven from a Simulink model.

### **Geometry Morphing Example (vrmorph)**

The vrmorph example illustrates how you can transfer matrix-type or variable-size signal data between the Simulink interface and a virtual reality world. With this capability, you can perform massive color changes or morphing. This model morphs a cube into an octahedron and then changes it back to a cube.

### **Vehicle Dynamics Visualization (vr\_octavia)**

The vr\_octavia example illustrates the benefits of the visualization of complex dynamic model in the virtual reality environment. It also shows the Simulink 3D Animation 3-D offline animation recording functionality.

### **Vehicle Dynamics Visualization - Simulation of Multiple Objects (vr\_octavia\_2cars)**

This example extends the vr\_octavia example to show multiple-object scenario visualizations.

The precomputed simulation data represents a standard double-lane-change maneuver conducted in two-vehicle configurations. One configuration engages the Electronic Stability Program control unit. The other configuration switches that control unit off. The example sends two sets of vehicle dynamics data in parallel to the virtual reality scene, to drive two different vehicles.

Models of the vehicles use the EXTERNPROTO mechanism. In the main virtual world associated with the VR Sink block, you can create several identical vehicles as instances of a common 3-D object. This approach greatly simplifies virtual world authoring. For instance, it is very easy to create a third vehicle to simultaneously visualize another simulation scenario. The octavia\_scene\_lchg\_2cars.wrl virtual world, the code after the definition of PROTO5 illustrates an approach for easy-to-define reusable objects.

In addition to vehicle properties controlled in the vr\_octavia example, vehicle prototypes also allow you to define vehicle color and scale. These properties distinguish individual car instances (color) and

avoid unpleasant visual interaction of two nearly-aligned 3-D objects (scale). Scaling one of the cars by a small amount, encompasses one car into another so that their faces do not clip randomly, based on the current simulation data in each simulation step.

To visualize vehicles side-by-side, add an offset to the position of one vehicle.

### **Vehicle Dynamics Visualization with Graphs (vr\_octavia\_graphs)**

The `vr_octavia_graphs` example extends the `vr_octavia` example by showing how to combine a virtual reality canvas in one figure with other graphical user interface objects. In this case, the virtual world displays three graphs that update at each major simulation time step.

### **Vehicle Dynamics Visualization with Live Rear Mirror Image (vr\_octavia\_mirror)**

The `vr_octavia_mirror` example extends the `vr_octavia` example by showing the capability of the VR Sink block to process video stream on input. In the virtual world, a `PixelTexture` texture map is defined at the point of the vehicle left rear mirror. The example places a 2-D image from a viewpoint at the same position (looking backward). That image is looped back into the same virtual world and projected on the rear mirror glass, creating the impression of a live reflection. Texture images can have different formats (corresponding to the available `SFImage` definitions according to the VRML97 standard). This example uses an RGB image that has the same format as the output from the VR to Video block. In the virtual world 3D file associated with the scene, you can define only a trivial texture (in this case, a 4x4 pixel checkerboard) that gets resized during simulation, according to the current size of the signal on the input. See the Plane Manipulation Using Space Mouse MATLAB Object example.

### **Vehicle Dynamics Visualization with Video Output Example (vr\_octavia\_video)**

The `vr_octavia_video` example illustrates how to use video output from the VR To Video block. This model performs simple operations on the video output. It requires the Computer Vision Toolbox™ product.

### **Inverted Pendulum Example (vrpend)**

The `vrpend` example illustrates the various ways a dynamic model in the Simulink interface can interact with a virtual reality scene. It is the model of a two-dimensional inverted pendulum controlled by a PID controller. What distinguishes this model from common inverted pendulum models are the methods for setting the set point. You visualize and interact with a virtual world by using a Trajectory Graph and VR Sink blocks. The Trajectory Graph block allows you to track the history of the pendulum position and change the set point in three ways:

- Mouse — Click and drag a mouse pointer in the **Trajectory Graph** two-dimensional window
- Input Signal — External Trajectory Graph input in this model (driven by a random number generator)
- VR Sensor — Activates the input from a VRML TouchSensor

When the pointing device in the virtual world viewer moves over an active TouchSensor area, the cursor shape changes. The triggering logic in this model is set to apply the new set point value with a left mouse button click.

Notice the pseudoorthographic view defined in the associated virtual world 3D file. You achieve this effect by creating a viewpoint that is located far from the object of interest with a very narrow view defined by the `FieldOfView` parameter. An orthographic view is useful for eliminating the panoramic distortion that occurs when you are using a wide-angle lens. The disadvantage of this

technique is that locating the viewpoint at a distance makes the standard viewer navigation tricky or difficult in some navigation modes, such as the Examine mode. If you want to navigate around the virtual pendulum bench, you should use some other viewpoint.

### **Solar System Example (vrplanets)**

The `vrplanets` example shows the dynamic representation of the first four planets of the solar system, Moon orbiting around Earth, and Sun itself. The model uses the real properties of the celestial bodies. Only the relative planet sizes and the distance between the Earth and the Moon are adjusted, to provide an interesting view.

Several viewpoints are defined in the virtual world, both static and attached to an observer on Earth. You can see that the planet bodies are not represented as perfect spheres. Using the `Sphere` graphic primitive, which is rendered this way, simplified the model. If you want to make the planets more realistic, you could use the more complex `IndexedFaceSet` node type.

Mutual gravity accelerations of the bodies are computed using Simulink matrix-type data support.

### **Plane Takeoff Example (vrtkoff)**

The `vrtkoff` example represents a simplified aircraft taking off from a runway. Several viewpoints are defined in this model, both static and attached to the plane, allowing you to see the takeoff from various perspectives.

The model shows the technique of combining several objects imported or obtained from different sources (CAD packages, general 3-D modelers, and so on) into a virtual reality scene. Usually it is necessary for you to wrap such imported objects with an additional `Transform` node. This wrapper allows you to set appropriately the scaling, position, and orientation of the objects to fit in the scene. In this example, the aircraft model from the Ligos V-Realm Builder Object Library is incorporated into the scene. The file `vrtkoff2.wrl` uses the same scene with a different type of aircraft.

### **Plane Take-Off with Trajectory Tracing Example (vrtkoff\_trace)**

The `vrtkoff_trace` is a variant of the `vrtkoff` example that illustrates how to trace the trajectory of a moving object (plane) in a scene. It uses a VR Tracer block. Using a predefined sample time, this block allows you to place markers at the current position of an object. When the simulation stops, the markers indicate the trajectory path of the object. This example uses an octahedron as a marker.

### **Plane Take-Off with HUD Text Example (vrtkoff\_hud)**

The `vrtkoff_hud` example illustrates how to display signal values as text in the virtual world and a simple Head-Up Display (HUD). It is a variant of the `vrtkoff` example.

The example sends the text to a virtual world using the VR Text Output block. This block formats the input vector using the format string defined in its mask (see `sprintf` for more information) and sends the resulting string to the `'string'` field of the associated Text node in the scene.

The example achieves HUD behavior (maintaining constant relative position between the user and the Text node) by defining a `ProximitySensor`. This sensor senses user position and orientation as it navigates through the scene and routes this information to the translation and rotation of the HUD object (in this case, a `Transform` that contains the Text node).

### **Collision Detection Using Line Sensor (vrcollisions)**

The `vrcollisions` example shows a simple way how to implement collision detection.

In the virtual world, an X3D LinePickSensor is defined. This sensor detects approximate collisions of several rays (modeled as IndexedLineSet) with arbitrary geometries in the scene. For geometric primitives, exact collisions are detected. One of LinePickSensor output fields is the `isActive` field, which becomes TRUE as soon as the collision between any of the rays and surrounding scene objects is detected.

The robot is inside a room with several obstacles. During the simulation, the robot moves forward as long as its sensor does not bounce into a wall or an obstacle. Use the **Left** and **Right** buttons to turn the robot so that there is a free path ahead, and the robot starts moving again.

The model defines both VR Sink and VR Source blocks, associated with the same virtual scene. The VR Source reads the sensor `isActive` signal and the current position of the robot. The VR Sink block sets the robot position, rotation, and color.

In the virtual world, there are two viewpoints defined - one static and one attached to the robot.

### **Differential Wheeled Robot with Lidar Sensor (vrcollisions\_lidar)**

The `vrcollisions_lidar` example shows how a LinePickSensor can be used to model lidar sensor behavior in Simulink 3D Animation.

In a simple virtual world, a wheeled robot with a lidar sensor mounted on its top is defined. This lidar sensor is implemented using the LinePickSensor that detects collisions of several rays (modeled as IndexedLineSet) with surrounding scene objects. Sensor `pickedRange` and `pickedPoint` fields are used in this model for visualization purposes only, but together with robot pose information they can be used for Simultaneous Localization and Mapping (SLAM) and other similar purposes.

The sensor sensing lines are visible, shown as transparent green lines. There are 51 sensing rays evenly spaced in the horizontal plane between -90 and 90 degrees. lidar range is 10 meters.

In order to visualize the lidar sensor output, there is a visualization proxy LineSet defined with lines identical to lines defined as the LinePickSensor sensing geometry. Visualization lines are blue. Combination of `pickedPoint` and `pickedRange` LinePickSensor outputs is used to visualize points of collision. The `pickedPoint` output contains coordinates of points that collided with surrounding objects. This output has variable size depending on how many sensor rays collided. The `pickedRange` output size is fixed, equal to the number of sensing rays. The output returns distance from lidar sensor origin to collision point for each sensing line. For rays that don't collide, this output returns -1. The `pickedRange` is used to determine the indices of lines for which the collision points are returned in the `pickedPoint` sensor output. In effect, the blue lines are shortened so that only the line segment between the ray fan origin and point of collision is displayed for each line.

Robot trajectory is modeled in a trivial way using the Signal Builder and the Ramp blocks. In the Signal Builder, a simple 1x1 meter square trajectory is defined for the first 40 seconds of simulation. After returning to its original position, the robot only rotates indefinitely.

In the model, there are both VR Sink and VR Source blocks defined, associated with the same virtual world. The VR Source is used to read the sensor signals. The VR Sink is used to set the Robot position / rotation and the coordinates of endpoints of the sensor visual proxy lines.

In the virtual world, there are several viewpoints defined, both static and attached to the robot, allowing to observe lidar visualization from different perspectives.

## Differential Wheeled Robot in a Maze (vrmaze)

The vrmaze example shows how you can use collision detection to simulate a differential wheeled robot that solves a maze challenge. The robot control algorithm uses information from virtual ultrasonic sensors that sense distance to surrounding objects.

A simple differential wheeled robot is equipped with two virtual ultrasonic sensors. One of the sensors looks ahead, and the other is directed to the left of the robot. Sensors are simplified, their active range is represented by green lines. The sensors are implemented as X3D LinePickSensor nodes. These sensors detect approximate collisions of rays (modeled as IndexedLineSet) with arbitrary geometries in the scene. For geometric primitives, exact collisions are detected. One of the LinePickSensor output fields is the isActive field, which becomes TRUE as soon as the collision between its ray and surrounding scene objects is detected. When activated, the sensor lines change their color from green to red using the script written directly in the virtual world.

In the model, there are both VR Sink and VR Source blocks defined, associated with the same virtual scene. The VR Source reads the sensors isActive signals. The VR Sink sets the robot position and rotation in the virtual world.

The robot control algorithm is implemented using a Stateflow® chart.

## MATLAB Interface Examples

The following table lists the MATLAB interface examples provided with the software. Descriptions of the examples follow the table. MATLAB interface examples display virtual worlds in your default viewer. If your default is the Simulink 3D Animation Viewer, some buttons are unavailable. In particular, the simulation buttons for simulation and recording are unavailable.

Example	Moving Objects	Morphing Objects	Text	Recording	vrml() Function Use	Space Mouse
vrcar	X					
vrheat		X	X			
vrheat_anim		X	X	X		
vrmemb	X		X		X	
vrterrain_simple		X				
vrtkoff_spacemouse			X			X

### Car in the Mountains Example (vrcar)

This example illustrates the use of the Simulink 3D Animation product with the MATLAB interface. In a step-by-step tutorial, it shows commands for navigating a virtual car along a path through the mountains.

- 1 In the MATLAB Command Window, type  

```
vrcar
```
- 2 A tutorial script starts running. Follow the instructions in the MATLAB Command Window.

### **Heat Transfer Example (vrheat)**

This example illustrates the use of the Simulink 3D Animation product with the MATLAB interface for manipulating complex objects.

In this example, matrix-type data is transferred between the MATLAB software and a virtual reality world. Using this feature, you can achieve massive color changes or morphing. This is useful for representing various physical processes. Precalculated data of time-based temperature distribution in an L-shaped metal block is used. The data is then sent to the virtual world. This forms an animation with relatively large changes.

This is a step-by-step example. Shown are the following features:

- Reshaping the object
- Applying the color palette to represent distributed parameters across an object shape
- Working with VRML or X3D text objects
- Animating a scene using the MATLAB interface
- Synchronization of multiple scene properties

At the end of this example, you can preserve the virtual world object in the MATLAB workspace, then save the resulting scene to a corresponding virtual world 3D file or carry out other subsequent operations on it.

### **Heat Transfer Visualization with 2-D Animation (vrheat\_anim)**

This example illustrates the use of the Simulink 3D Animation C interface to create 2-D offline animation files.

You can control the offline animation recording mechanism by setting the relevant `vrworld` and `vrfigure` object properties. You should use the Simulink 3D Animation Viewer to record animations. However, direct control of the recording is also possible.

This example uses the heat distribution data from the `vrheat` example to create an animation file. You can later distribute this animation file to be independently viewed by others. For this kind of visualization, where the static geometry represented by an `IndexedFaceSet` node is colored based on the simulation of some physical phenomenon, it is suitable to create 2-D `.avi` animation files. The software uses a `MATLAB VideoWriter` object to record 2-D animation exactly as it appears in the viewer figure.

There are several methods you can use to record animations. In this example, we use the scheduled recording. When scheduled recording is active, a time frame is recorded into the animation file with each setting of the virtual world `Time` property. Recording is completed when you set the scene time at the end or outside the predefined recording interval.

When using the Simulink 3D Animation MATLAB interface, you set the scene time as desired. This is typically from the point of view of the simulated phenomenon equidistant times. This is the most important difference from recording the animations for virtual worlds that are associated with Simulink models, where scene time corresponds directly to the Simulink time.

The scene time can represent any independent quantity along which you want to animate the computed solution.

This is a step-by-step example. Shown are the following features:

- Recording 2-D offline animations using the MATLAB interface
- Applying the color palette to visualize distributed parameters across an object shape
- Animating a scene
- Playing the created 2-D animation file using the system AVI player

At the end of this example, the resulting file `vrheat_anim.avi` remains in the working folder for later use.

### **Rotating Membrane with MATLAB Graphical Interface Example (vrmemb)**

The `vrmemb` example shows how to use a 3-D graphic object generated from the MATLAB environment with the Simulink 3D Animation product. The membrane was generated by the `logo` function and saved in the VRML format using the standard `vrml` function. You can save all Handle Graphics® objects this way and use them with the Simulink 3D Animation software as components of associated virtual worlds.

After starting the example, you see a control panel with two sliders and three check boxes. Use the sliders to rotate and zoom the membrane while you use the check boxes to determine the axis to rotate around.

In the virtual scene, notice the text object. It is a child of the `Billboard` node. You can configure this node so that its local *z*-axis turns to point to the viewer at all times. This can be useful for modeling virtual control panels and head-up displays (HUDs).

### **Terrain Visualization Example (vrterrain\_simple)**

This example illustrates converting available Digital Elevation Models into the VRML format, for use in virtual reality scenes.

As a source of terrain data, the South San Francisco DEM model (included in the Mapping Toolbox™ software) has been used. A simple Boeing® 747® model is included in the scene to show the technique of creating virtual worlds from several sources on-the-fly.

This example requires the Mapping Toolbox software from MathWorks®.

### **Plane Manipulation Using Space Mouse MATLAB Object**

This example illustrates how to use a space mouse using the MATLAB interface. After you start this example, a virtual world with an aircraft is displayed in the Simulink 3D Animation Viewer. You can navigate the plane in the scene using a space mouse input device. Press button 1 to place a marker at the current plane position.

This example requires a space mouse or compatible device.

## **See Also**

### **Related Examples**

- “Link to Models”





# Installation

---

The Simulink 3D Animation product provides the files you need for installation on both your host computer and client computer.

- “Set the Default Viewer” on page 2-2
- “Set Simulink 3D Animation Preferences” on page 2-5
- “Install V-Realm Editor” on page 2-14
- “Test the Viewer Installation” on page 2-16

## Set the Default Viewer

If you have an HTML5-enabled web browser, you can view virtual worlds with either the default Simulink 3D Animation Viewer or your web browser. You determine the viewer for displaying your scene using the `vrsetpref` and `vrgetpref` commands.

This procedure assumes that you are working with a PC platform.

- 1 When you install Simulink 3D Animation, the Simulink 3D Animation viewer is the default viewer. If you are not sure whether the default has been changed, you can determine your default viewer by typing:

```
vrgetpref
```

The MATLAB Command Window displays

```
ans =
```

```

        DataTypeBool: 'logical'
        DataTypeInt32: 'double'
        DataTypeFloat: 'double'
        DefaultCanvasNavPanel: 'opaque'
        DefaultCanvasUnits: 'normalized'
        DefaultEditorPosition: [822 123 661 703]
        DefaultEditorTriad: 'bottomleft'
        DefaultFigureAntialiasing: 'on'
        DefaultFigureCaptureFileFormat: 'tif'
        DefaultFigureCaptureFileName: '%f_anim_%n.tif'
        DefaultFigureDeleteFcn: ''
        DefaultFigureLighting: 'on'
        DefaultFigureMaxTextureSize: 'auto'
        DefaultFigureNavPanel: 'halfbar'
        DefaultFigureNavZones: 'off'
        DefaultFigurePosition: [5 92 576 380]
        DefaultFigureRecord2DCompressMethod: 'auto'
        DefaultFigureRecord2DCompressQuality: 75
        DefaultFigureRecord2DFileName: '%f_anim_%n.avi'
        DefaultFigureRecord2DFPS: 15
        DefaultFigureStatusBar: 'on'
        DefaultFigureTextures: 'on'
        DefaultFigureToolBar: 'on'
        DefaultFigureTooltips: 'on'
        DefaultFigureTransparency: 'on'
        DefaultFigureTriad: 'none'
        DefaultFigureWireframe: 'off'
        DefaultViewer: 'internal'
        DefaultWorldRecord3DFileName: '%f_anim_%n.%e'
        DefaultWorldRecordMode: 'manual'
        DefaultWorldRecordInterval: [0 0]
        DefaultWorldRemoteView: 'off'
        DefaultWorldTimeSource: 'external'
        Editor: '*BUILTIN'
        EditorPreserveLayout: 'off'
        EditorSavePosition: 'on'
        HttpPort: 8123
        TransportBuffer: 5
        TransportTimeout: 20
        VrPort: 8124

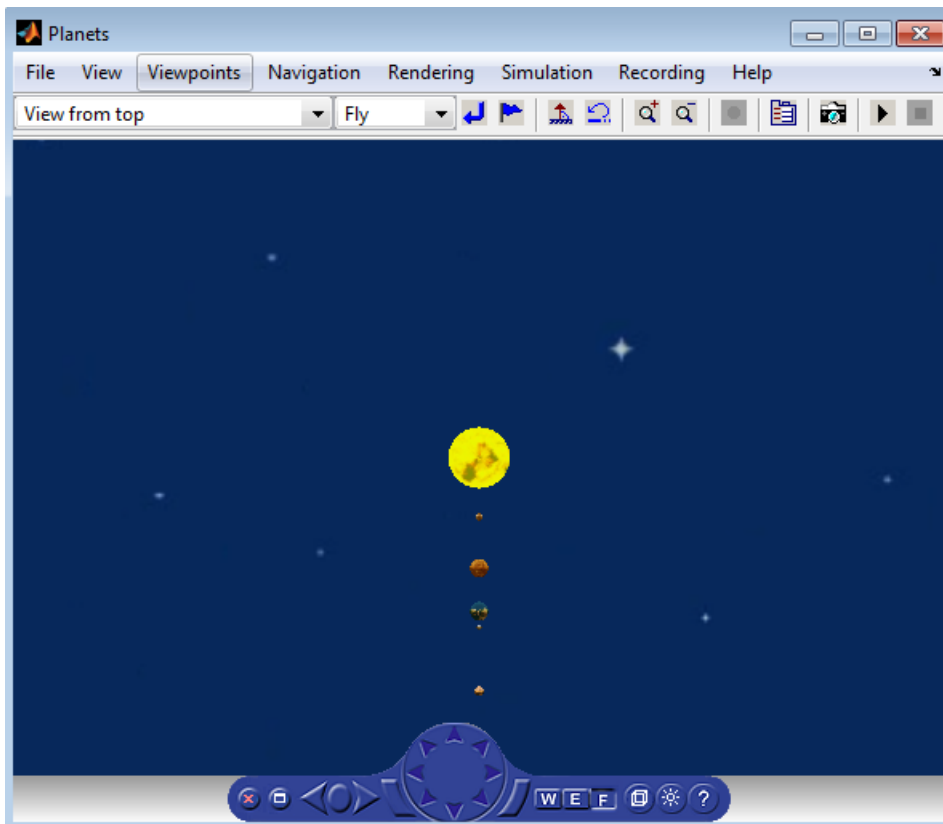
```

The `DefaultViewer` property is set to `'internal'`. The Simulink 3D Animation Viewer is the default viewer for viewing virtual worlds. Any virtual worlds that you open are displayed in the viewer.

- 1 For example, at the MATLAB command prompt, type

```
vrplanets
```

The Planets example is loaded and the virtual world is displayed in the Simulink 3D Animation Viewer.



- 2 To view the virtual world through the web browser, from the MATLAB command prompt, use the `view` and `vrview` commands.
- 3 Reset the Simulink 3D Animation Viewer as your default viewer by typing:

```
vrsetpref('DefaultViewer', 'factory')
```

Alternatively, you can use the MATLAB File menu Preferences dialog box. See “Set Simulink 3D Animation Preferences” on page 2-5.)

## See Also

### Functions

`vrgetpref` | `vrsetpref`

## **Related Examples**

- “Set Simulink 3D Animation Preferences” on page 2-5
- “Install V-Realm Editor” on page 2-14
- “Test the Viewer Installation” on page 2-16
- “Simulink 3D Animation Product Description” on page 1-2
- “Set the Default Editor” on page 5-5

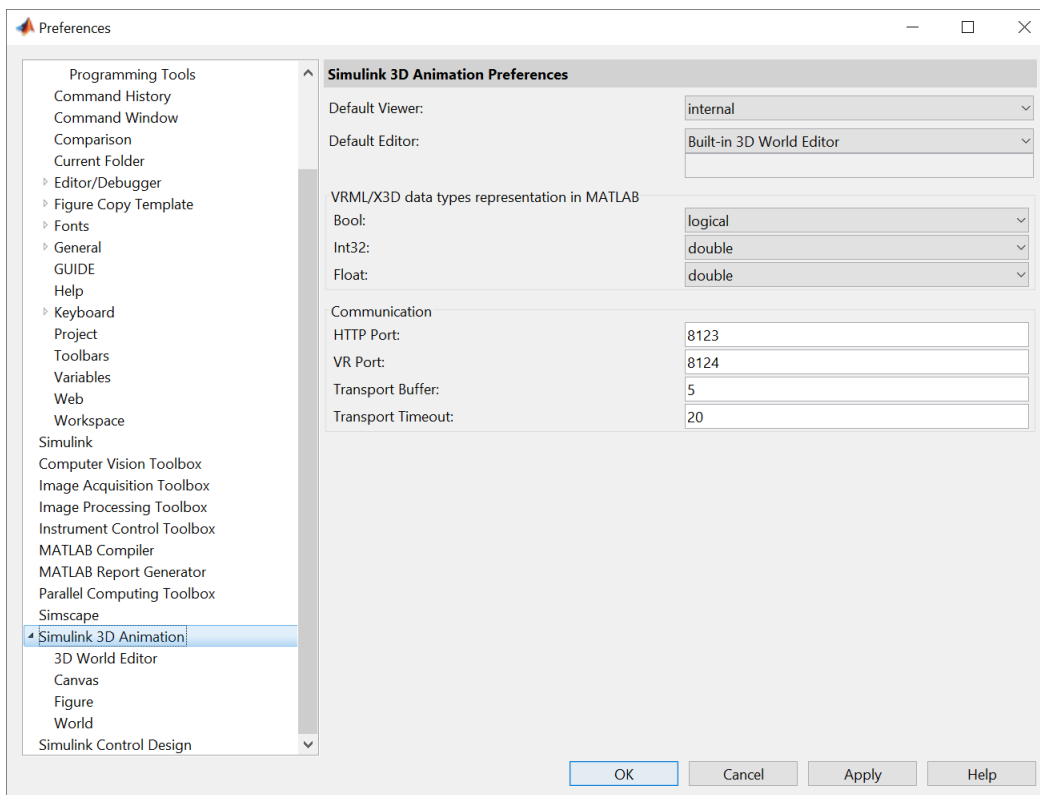
## Set Simulink 3D Animation Preferences

### In this section...

- “Simulink 3D Animation Preferences Dialog Box” on page 2-6
- “3D World Editor Preferences Dialog Box” on page 2-7
- “Canvas Preferences Dialog Box” on page 2-8
- “Figure Appearance Preferences Dialog Box” on page 2-8
- “Figure Rendering Preferences Dialog Box” on page 2-9
- “Figure 2-D Recording Preferences Dialog Box” on page 2-11
- “Figure Frame Capture Preferences” on page 2-11
- “World Preferences Dialog Box” on page 2-12

The Simulink 3D Animation software opens with default preference settings. You can change these settings so that the next time you open a Simulink 3D Animation interface, such as the 3D World Editor, the associated preferences take effect. Use one of these approaches:

- From the MATLAB Toolstrip, in the **Home** tab, in the **Environment** section, select **Preferences > Simulink 3D Animation**.



- At the MATLAB command line, use these functions:

**Tip** The preferences dialog box shows a subset of the preferences that you can set using MATLAB functions.

## Simulink 3D Animation Preferences Dialog Box

The top dialog box is for general Simulink 3D Animation preferences.

Preference	Value	Description
<b>Default Viewer</b>	internal   web Default: 'internal'	Specifies which viewer is used to view a virtual world. The default Simulink 3D Animation Viewer is used when the preference is set to <code>internal</code> . The web browser is used when this preference is set to <code>web</code> .
<b>Default Editor</b>	Built-in 3D World Editor   V-Realm Builder   MATLAB Editor   Custom	<p>Specifies which virtual world editor to use. Path to the virtual world editor. If this path is empty, the MATLAB editor is used.</p> <p>The path setting is active only if you select the Custom option.</p> <p>To open a virtual world file in a third-party editor, do not use the <code>vredit</code> command. For example, to open a virtual world in the Ligos V-Realm Builder editor:</p> <ol style="list-style-type: none"> <li>1 Set the default editor to V-Realm Builder. In MATLAB, enter: <pre>vrsetpref('Editor','*VREALM');</pre></li> <li>2 To open a file in the V-Realm editor, in MATLAB navigate to a virtual world file, right-click, and select <b>Edit</b>.</li> </ol> <p><b>Note</b> The <code>vredit</code> command opens the 3D World Editor, regardless of the default editor preference setting.</p>
<b>Bool</b>	logical   char Default: logical	Specifies the handling of the virtual world <code>Bool</code> data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . If set to <code>logical</code> , the virtual world <code>Bool</code> data type is returned as a logical value. If set to <code>char</code> , the <code>Bool</code> data type is returned 'on' or 'off'.
<b>Int32</b>	int32   double Default: double	Specifies handling of the virtual world <code>Int32</code> data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . If set to <code>int32</code> , the virtual world <code>Int32</code> data type is returned as 'int32'. If set to <code>double</code> , the <code>Int32</code> data type is returned as 'double'.
<b>Float</b>	'single'   'double' Default: 'double'	Specifies the handling of the virtual world <code>float</code> data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . If set to <code>single</code> , the virtual world <code>Float</code> and <code>Color</code> data types are returned as 'single'. If set to <code>double</code> , the <code>Float</code> and <code>Color</code> data types are returned as 'double'.

Preference	Value	Description
<b>HTTP Port</b>	Numeric Default: 8123	IP port number used to access the Simulink 3D Animation server over the web via HTTP. If you change this preference, restart the MATLAB software before the change takes effect.
<b>VR Port</b>	Numeric Default: 8124	IP port used for communication between the Simulink 3D Animation server and its clients. If you change this preference, restart the MATLAB software before the change takes effect.
<b>Transport Buffer</b>	Numeric Default: 5	Length of the transport buffer (network packet overlay) for communication between the Simulink 3D Animation server and its clients.
<b>Transport Timeout</b>	Numeric Default: 20	Amount of time, in seconds, that 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.

### 3D World Editor Preferences Dialog Box

The Simulink 3D Animation preferences include the following preferences for the 3D World Editor.

Property	Value	Description
<b>Position</b>	Specify the pixel location for the lower-left corner, the width, and the height (for example, [96 120 862 960]) Default: Depends on current screen resolution	Specifies the default location for the 3D World Editor. If you select <b>Save position on exit</b> , the default position changes to the position of the 3D World Editor used when you last exited it.
<b>Triad</b>	none   top left   top right   bottom left   bottom right   center Default: 'bottom left'	Specifies where in the virtual world display pane to locate a triad of coordinate axes.
<b>View pane mouse behavior</b>	navigate   select Default: navigate	Specifies whether the mouse in the view pane is in navigation mode or selection mode (for highlighting corresponding nodes in the tree view pane).
<b>Save position on exit</b>	off   on Default: on	Causes the 3D World Editor to open in the same location where the editor was when you last exited it.

Property	Value	Description
<b>Preserve Layout per Virtual Reality 3D File</b>	off   on Default: on	Specifies whether the 3D World Editor starts up either with the default virtual world display layout or with the layout as it was when you exited it previously. The saved layout includes settings for the view, viewpoints, navigation, and rendering. Simulink 3D Animation saves the layout in a separate virtual world 3D file for up to eight files.
<b>Highlight selected objects</b>	off   on Default: on	Specifies whether to highlight virtual world objects selected in the view pane.

## Canvas Preferences Dialog Box

The Simulink 3D Animation preferences include a **Navigation panel** preference. The canvas preferences apply to the 3D World Editor, Simulink 3D Animation Viewer, and Simulink 3D Animation Web Viewer.

Property	Value	Description
<b>Navigation panel</b>	none   minimized   translucent   opaque Default: none	Controls the appearance of the navigation panel in the canvas.

## Figure Appearance Preferences Dialog Box

The figure appearance preferences apply to the 3D World Editor and Simulink 3D Animation Viewer. Some of these preferences also apply to Simulink 3D Animation Web Viewer.

Property	Value	Description
<b>Toolbar</b>	on   off Default: on	Specifies whether the toolbar is displayed.
<b>Tooltips</b>	off   on Default: on	Specifies whether tooltips are displayed.
<b>Status bar</b>	off   on Default: on	Specifies whether status bar is displayed.  Also applies to the Simulink 3D Animation Web Viewer.
<b>Navigation zones</b>	off   on Default: on	Specifies whether navigation zones are displayed.  Also applies to the Simulink 3D Animation Web Viewer.



Property	Value	Description
<b>Navigation panel</b>	none   minimized   translucent   opaque Default: none	Controls the appearance of the navigation panel in the canvas.
<b>Triad</b>	none   top left   top right   bottom left   bottom right   center Default: bottom left	Specifies where in the virtual world display pane to locate a triad of coordinate axes.
<b>Position</b>	Matrix with upper-right and lower-left corner position. Default: [5 92 576 380]	Specifies the default location of the figure window.

## Figure Rendering Preferences Dialog Box

The figure rendering preferences specify how virtual worlds are displayed.

Property	Value	Description
<b>Antialiasing</b>	on   off Default: on	Determines whether antialiasing is used when rendering scene. Antialiasing smooths textures by interpolating values between texture points.
<b>Lighting</b>	off   on Default: on	Specifies whether the lighting is considered when rendering. If it is off, all the objects are drawn as if uniformly lit.
<b>Sound</b>	off   on Default: on	If a virtual world contains a Sound node and your computer supports sound, then you can listen to the sound in a virtual world.

Property	Value	Description
<b>Stereo 3D</b>	off   anaglyphactive Default: off	Specifies whether to use stereoscopic 3D vision.  For anaglyph viewing, use red/cyan 3D glasses. Viewing a virtual world in this mode causes the colors to appear as almost grayscale. This approach does not require any special computer hardware or software.  For active stereo viewing, use active shutter 3D glasses. This approach preserves color effects and produces more powerful 3D effects. Active stereo requires a specially configured computer and monitor setup.
<b>Stereo 3D Camera Offset</b>	Numeric Default: 0.1	Specifies the distance between the two points of view (cameras) that produce the 3D effect. The higher the offset, the further apart the cameras are, and thus the deeper the 3D effect.
<b>Stereo 3D Horizontal Image Translation</b>	Numeric value from 0 through 1, inclusive. The larger the value, the further back the background appears to be. Default: 0	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.
<b>Transparency</b>	off   on Default: on	Specifies whether transparency information is considered when rendering.
<b>Wireframe</b>	off   on Default: off	Specifies whether objects are drawn as solids or wireframes.
<b>Textures</b>	off   'on' Default: on	Turns texture rendering on or off.

Property	Value	Description
<b>Maximum texture size</b>	auto   32 <= x <= video card limit, where x is a power of 2 (video card limit is typically 1024 or 2048)	Sets the maximum pixel size of a texture used in rendering <code>vrfigure</code> objects. The smaller the size, the faster the texture can render. Increasing this value improves image quality but decreases performance. A value of 'auto' sets the maximum pixel size. If the value you enter is unsuitable, a warning can trigger. The software then automatically adjusts the property to the next smaller suitable value.

### Figure 2-D Recording Preferences Dialog Box

Property	Value	Description
<b>2-D animated file name</b>	Character vector Default: '%f_anim_%n.avi'	Specifies the 2-D offline animation file name. The name can contain tokens that are replaced by the corresponding information when the animation recording takes place. For further details, see "File Name Tokens" on page 4-14.
<b>Recording compression method</b>	' '   auto   lossless   codec_code Default: auto	Specifies the compression method for creating 2-D animation files. The codec code must be registered in the system. See the MATLAB documentation for <code>VideoWriter</code> .
<b>Recording compression quality</b>	Integer 0-100. Default: 75	Specifies the default quality of 2-D animation file compression for new <code>vrfigure</code> objects.
<b>Frames per second</b>	Numeric Default: 15	Specifies the default frames per second playback speed.

### Figure Frame Capture Preferences

Property	Value	Description
<b>CaptureFileFormat</b>	tif   png Default: tif	Specifies file format for a captured frame file.

Property	Value	Description
<b>CaptureFileName</b>	Character vector Default: '%f_anim_%n.tif'	Specifies the frame capture file name. The name can contain tokens that are replaced by the corresponding information when the animation recording takes place. For further details, see “Define File Name Tokens” on page 4-12.

## World Preferences Dialog Box

Property	Value	Description
<b>3-D animated file name</b>	character vector Default: '%f_anim_%n.%e'	3-D animation file name. The name can contain tokens that are replaced by the corresponding information when the animation recording takes place. For details, see “Define File Name Tokens” on page 4-12.
<b>Recording mode</b>	manual   scheduled Default: manual	Animation recording mode.
<b>Recording interval</b>	Vector of two doubles Default: [0 0]	Start and stop times for scheduled animation recording. Corresponds to the virtual world object Time property.
<b>Time source</b>	external   freerun Default: external	Source of the time for the virtual world. If set to <code>external</code> , time in the scene is controlled from the MATLAB software (by setting the Time property) or the Simulink software (simulation time). If set to <code>freerun</code> , time in the scene advances independently based on the system timer.
<b>Allowing viewing from the Internet</b>	off   on Default: off	Remote access flag. If the virtual world is enabled for remote viewing, it is set to <code>on</code> ; otherwise, it is set to <code>off</code> .
<b>Create or update world thumbnail file when world is opened</b>	off   on Default: off	Specify whether create world thumbnails when you open a virtual world.

## See Also

### Functions

vrgetpref | vrsetpref

## **Related Examples**

- “Set the Default Editor” on page 5-5
- “Set the Default Viewer” on page 2-2

## Install V-Realm Editor

### In this section...

“V-Realm Editor Installation on Windows Platforms” on page 2-14

“V-Realm Builder Help” on page 2-15

“Uninstall V-Realm Builder” on page 2-15

**Tip** The Simulink 3D Animation product includes the 3D World Editor for editing virtual worlds. You can use the 3D World Editor on all supported platforms for Simulink 3D Animation. The 3D World Editor is the default editor. For a comparison of editors, see “Choose a Virtual World Editor” on page 5-2.

### V-Realm Editor Installation on Windows Platforms

When you install the Simulink 3D Animation product, files are copied to your hard drive for the Ligos V-Realm Builder, which is an optional virtual world editor available on Windows platforms. However, the installation is not complete.

Installing the virtual world editor writes a key to the Microsoft® Windows registry, making extra V-Realm Builder library files available for you to use. The installation associates the **Edit** button in Simulink 3D Animation blocks with this editor:

- 1 From your desktop, right-click the MATLAB icon and select **Run as administrator**.
- 2 In the MATLAB Command Window, type

```
vrinstall -install
```

or type

```
vrinstall('-install')
```

The MATLAB Command Window displays the following messages:

```
Starting editor installation...
Done.
```

- 3 Type

```
vrinstall
```

If the editor installation was successful, the MATLAB Command Window displays this message:

```
Virtual World editor: installed
```

- 4 Exit MATLAB and restart MATLAB.
- 5 Set the default editor to V-Realm Builder. In MATLAB, enter:

```
vrsetpref('Editor','*VREALM');
```

- 6 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.

---

## V-Realm Builder Help

---

**Note** You cannot access the V-Realm Builder documentation from the web. If you are reading this page on the web, then open the MATLAB Help browser and navigate to the V-Realm Builder documentation.

---

To access V-Realm Builder help from the MATLAB Help browser, click V-Realm Builder help.

You can view the V-Realm Builder help even if you have not installed V-Realm Builder.

## Uninstall V-Realm Builder

Use the MathWorks uninstaller. Running this utility removes the Simulink 3D Animation and Ligos V-Realm Builder software from your system. It also restores your previous system configuration.

- 1 On the Windows task bar, click **Start**, point to **MATLAB**, and then click the uninstaller.

The MathWorks uninstaller begins running.

- 2 Select the **Simulink 3D Animation** check box.
- 3 Follow the remaining uninstall instructions.

## See Also

### Functions

`vrgetpref` | `vrinstall` | `vrsetpref`

## Related Examples

- “Test the Viewer Installation” on page 2-16
- “V-Realm Builder Help” on page 2-15

## Test the Viewer Installation

### In this section...

“Section Overview” on page 2-16

“Simulink Testing” on page 2-16

“MATLAB Testing” on page 2-18

### Section Overview

The Simulink 3D Animation product includes several Simulink models with the associated virtual worlds. These models are examples of what you can do with this software. You can use one of these examples to test the installation of the virtual world viewer.

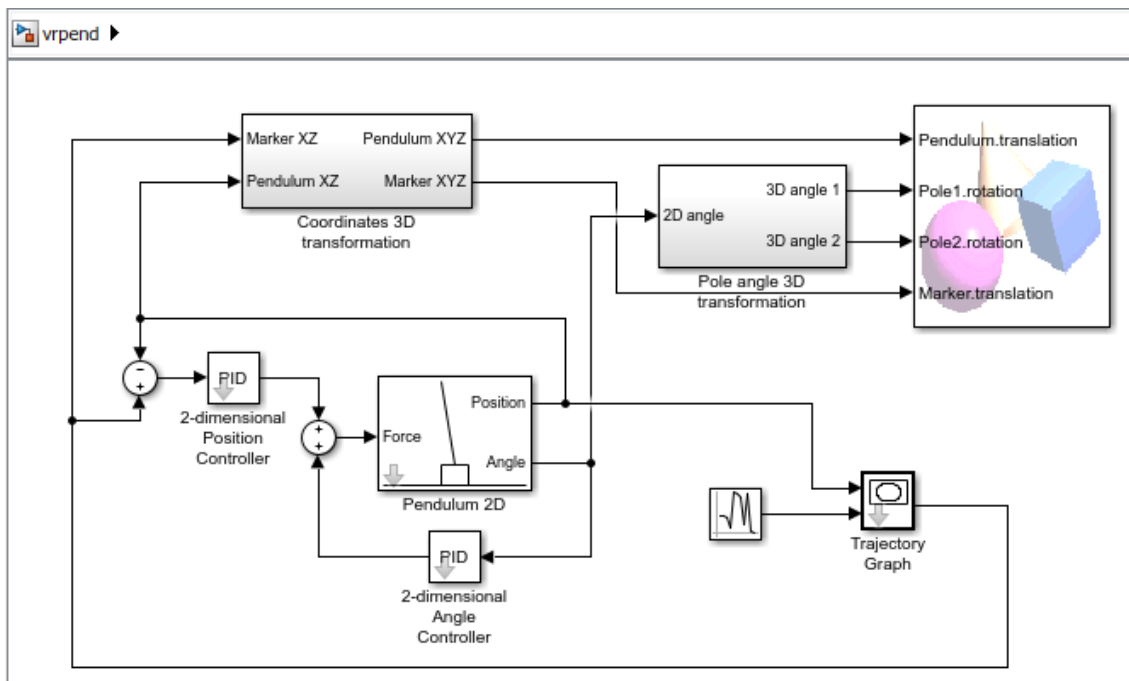
### Simulink Testing

Before you can run this example, install the MATLAB, Simulink, and Simulink 3D Animation products as follows:

- 1 In the MATLAB Command Window, type

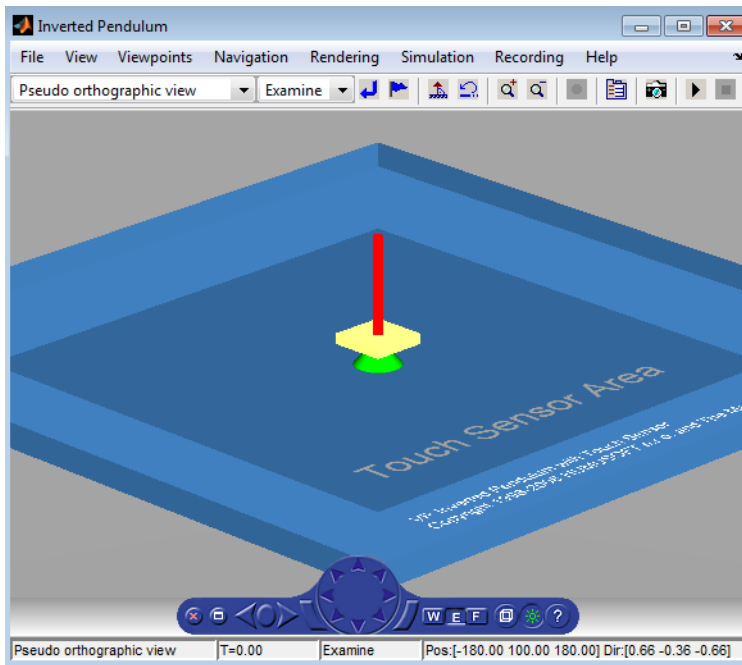
```
vrpend
```

A Simulink window opens with the model for an inverted pendulum. This model, which you can view in three dimensions with the software, has an interactive set point and trajectory graph.

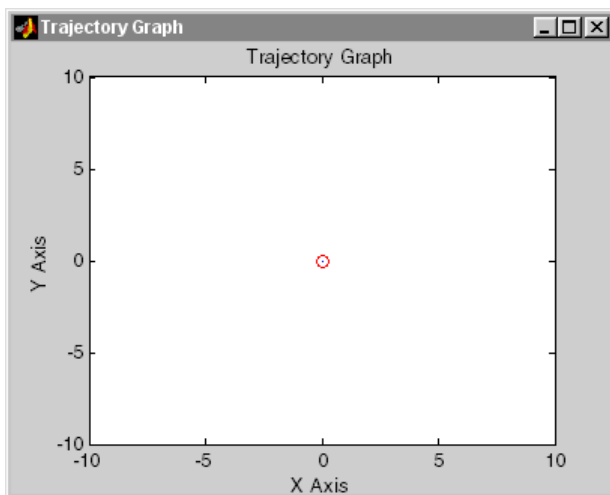


The Simulink 3D Animation Viewer opens with a 3-D model of the pendulum.





- 2 In the Simulink 3D Animation Viewer, from the **Simulation** menu, click **Run**. A **Trajectory Graph** window opens, and a simulation starts running.



- 3 In the Simulink 3D Animation Viewer, point to a position on the blue surface and left-click.

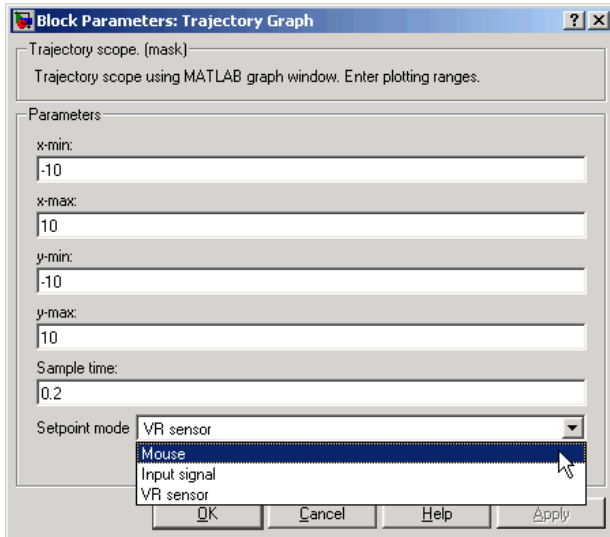
The pendulum set point, represented by the green cone, moves to a new location. Next, the path is drawn on the trajectory graph, and then the pendulum itself moves to the new location.

In the Simulink 3D Animation Viewer, you see the animated movement of the pendulum. Use the viewer controls to navigate through the virtual world, change the viewpoints, and move the set point. For more information about using the Simulink 3D Animation Viewer controls, see “Simulink 3D Animation Viewer” on page 7-4.

- 4 In the Simulink window, double-click the Trajectory Graph block.

The Block Parameters: Trajectory Graph dialog box opens.

- 5 From the **Stipend mode** list, choose **Mouse**, then click **OK**.



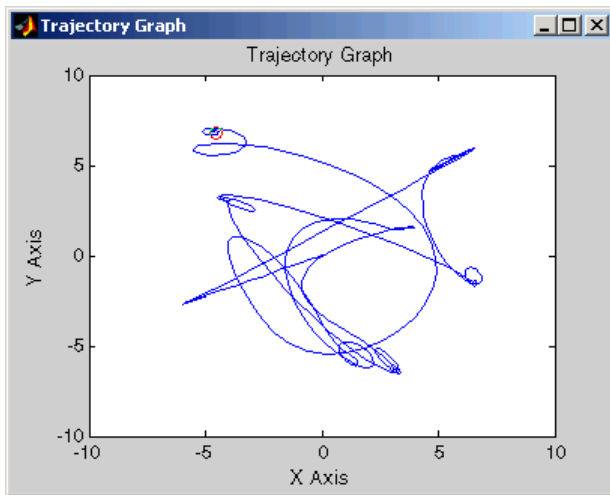
You can now use the trajectory graph as a 2-D input device to set the position of the pendulum.

- 6 Move the mouse pointer into the graph area and click.

The set point (red circle) for the pendulum position moves to a new location.

- 7 In the Simulink window, from the **Simulation** menu, click **Stop**.

The trajectory for the pendulum is displayed in the graph as a blue line.



- 8 Close the Simulink 3D Animation Viewer and close the Simulink window.

You can try other examples in “Simulink Interface Examples” on page 1-16, or you can start working on your own projects.

## MATLAB Testing

This model, which can be viewed in three dimensions with the software, has a MATLAB interface to control the figure in a virtual world viewer window.

Additional examples are listed in the table “MATLAB Interface Examples” on page 1-25.

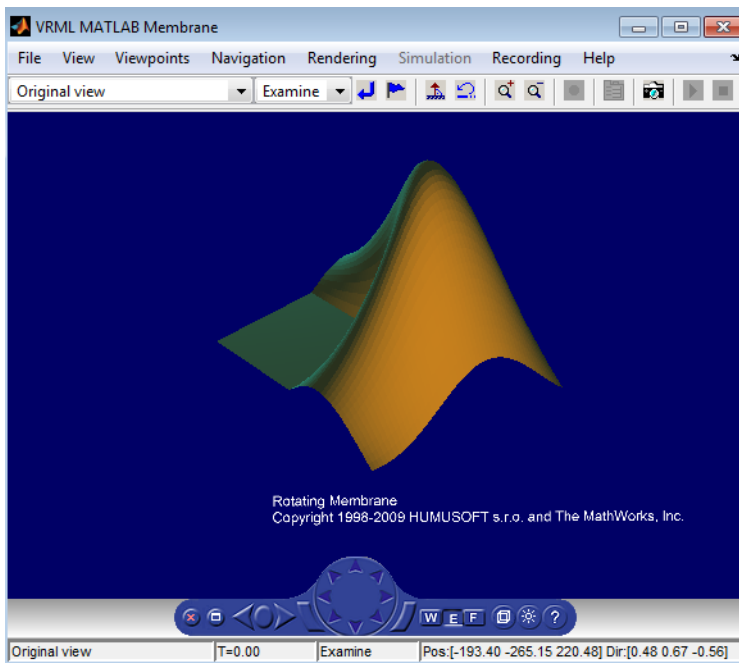
- 1 In the MATLAB window, type

```
vrmemb
```

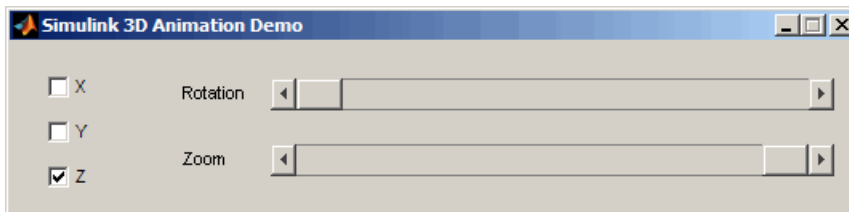
The MATLAB interface displays the following messages:

```
View the published version of this example to learn more about
"vrmemb.m".
```

The Simulink 3D Animation Viewer opens with a 3-D model.



- 2 Use the viewer controls to move within the virtual world, or use the example dialog box to rotate the membrane. Sometimes the Simulink 3D Animation example dialog box is hidden behind the viewer window.



## See Also

**Functions**  
vrinstall

## **Related Examples**

- “Set the Default Editor” on page 5-5
- “Set the Default Viewer” on page 2-2
- “Set Simulink 3D Animation Preferences” on page 2-5
- “Install V-Realm Editor” on page 2-14

# Simulink Interface

---

The Simulink 3D Animation product works with both the MATLAB and the Simulink products. However, the Simulink interface is the preferred way of working with the software. It is more straightforward to use and all the features are easily accessible through a graphical interface.

- “Connect Virtual Worlds and Models” on page 3-2
- “Open a Viewer Window” on page 3-9
- “Display Virtual World and Start Simulation” on page 3-10
- “View Virtual World on Host Computer” on page 3-12
- “View Virtual World Remotely” on page 3-15
- “Modify Remote Virtual World Via Sensor Events” on page 3-19
- “Interact with Generated Code” on page 3-20

## Connect Virtual Worlds and Models

In this section...
“Output Simulation Data to a Virtual World” on page 3-2
“Input Virtual World Data to a Model” on page 3-6
“Change the Associated Virtual World” on page 3-7

After you create a virtual world and a Simulink model, to have the virtual world interact with a dynamic system simulation, connect the model and the virtual world using Simulink 3D Animation blocks.

- To use simulation data from a model to interact with a virtual world, include a VR Sink block in the model. For details, see “Output Simulation Data to a Virtual World” on page 3-2.
- To use information from a virtual world to interact with a model, include a VR Source block in the model. For details, see “Input Virtual World Data to a Model” on page 3-6.

Simulating a Simulink model generates signal data for a dynamic system. To output data from the model to control and animate a virtual world, use a VR Sink block.

### Output Simulation Data to a Virtual World

This example shows how to use simulation data from a model to display a dynamic visualization of the simulation. The example simulates a plane takeoff and lets you view it in a virtual world. This example assumes that you are using the Simulink 3D Animation Viewer.

---

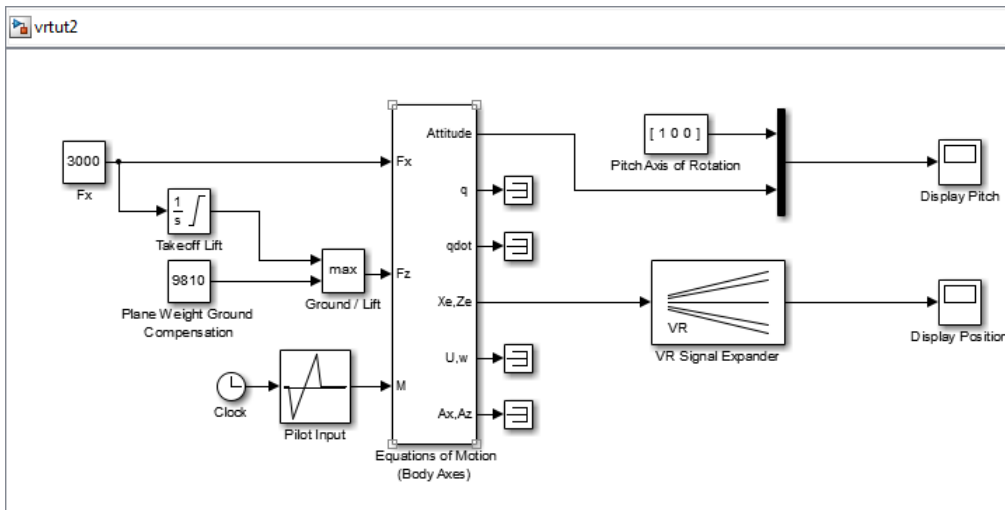
**Tip** For other examples of how to use the VR Sink block, see Magnetic Levitation Model and “Geometry Morphing”.

---

- 1 In the MATLAB Command Window, type

```
vrtut2
```

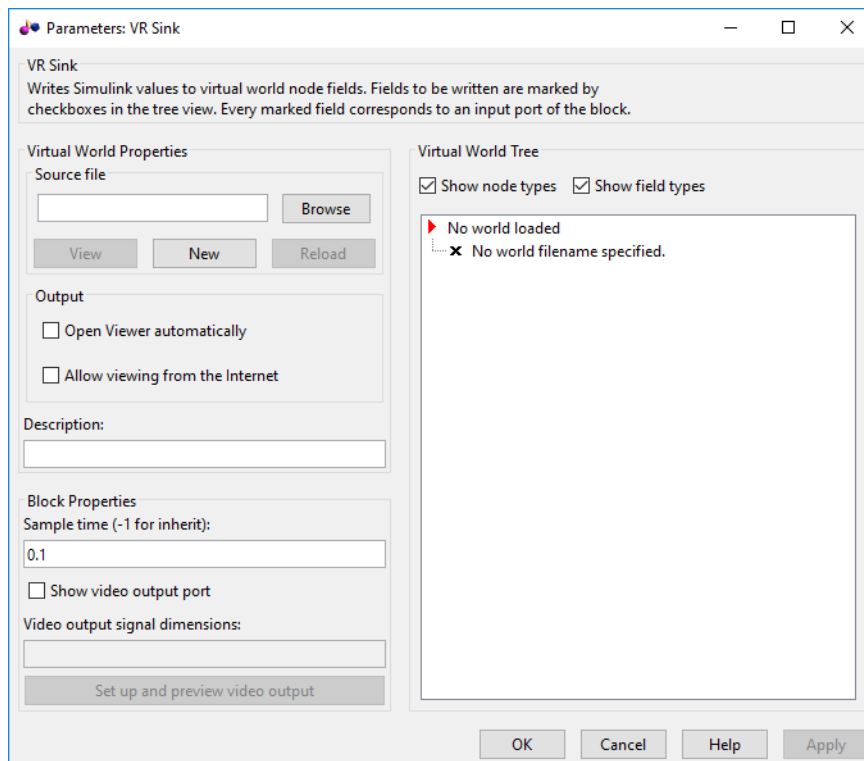
A Simulink model opens without a Simulink 3D Animation block that connects the model to a virtual world.



- 2 Simulate the model by clicking **Run** in the **Simulate** section of the **Simulation** tab in the Simulink toolstrip.

Observe the results of the simulation in the scope windows.

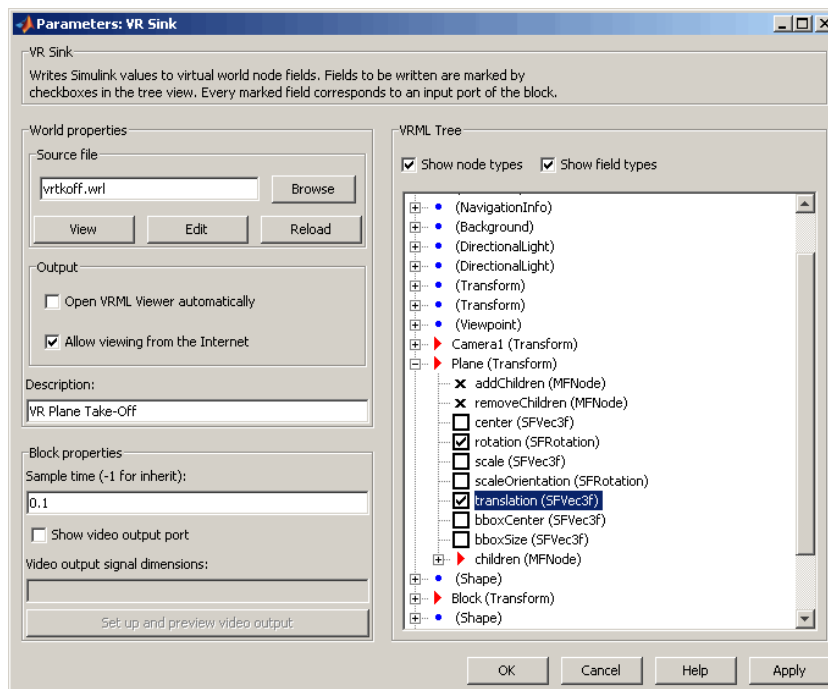
- 3 To the right of the model, left-click and type **VR Sink**. In the dialog box, select the VR Sink block.
- 4 Select a virtual world for the visualization of your simulation. Double-click the VR Sink block. Click **Browse** and select `vrtkoff.wrl`.



- 5 Associate a virtual world with the model. At the **Source File** text box, click the **Browse** button. The Select World dialog box opens. Find the folder `matlabroot\toolbox\s13d\s13ddemos`. Select the file `vrtkoff.wrl` and click **OK**.
- 6 In the **Description** text box, examine the brief description of the model. This description appears on the list of available worlds served by the Simulink 3D Animation server.
- 7 Select the **Open Viewer automatically** parameter and click **Apply**. The VR Sink dialog box displays the virtual scene node tree, showing the structure of the associate virtual world.
- 8 Expand the **Plane (Transform)** node.

The list of characteristics of the plane can be driven from the Simulink interface. This model computes the position and the pitch of the plane.

- 9 In the **Plane (Transform)** tree, select the **translation** and **rotation** fields, which represent the position and the pitch of the plane, respectively. Click **OK**.



In the Simulink diagram, the VR Sink block is updated with two inputs.



The first input is **Plane rotation**. Define the rotation with a four-element vector. The first three numbers define the axis of rotation. In this example, it is  $[1\ 0\ 0]$  for the  $x$ -axis (see the **Pitch Axis of Rotation** block in the model). The pitch of the plane is expressed by the rotation about the  $x$ -axis. The last number is the rotation angle around the  $x$ -axis, in radians. The rotation is in terms of the orientation of the object in space, relative to its parent node.

- 10 In the Simulink model, connect the line going to the Scope block labeled **Display Pitch** to the **Plane rotation** input.

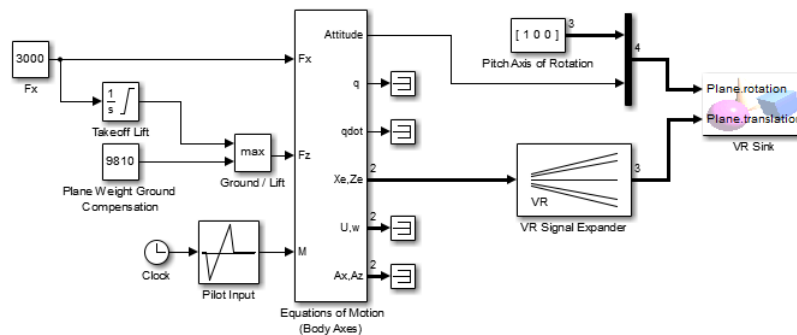


The second input is `Plane translation`. This input describes the position of the plane in the virtual world. This position consists of three coordinates,  $x, y, z$ . The connected vector must have three values. In this example, the runway is in the  $x$ - $z$  plane (using the VR Signal Expander block). The  $y$ -axis defines the altitude of the plane.

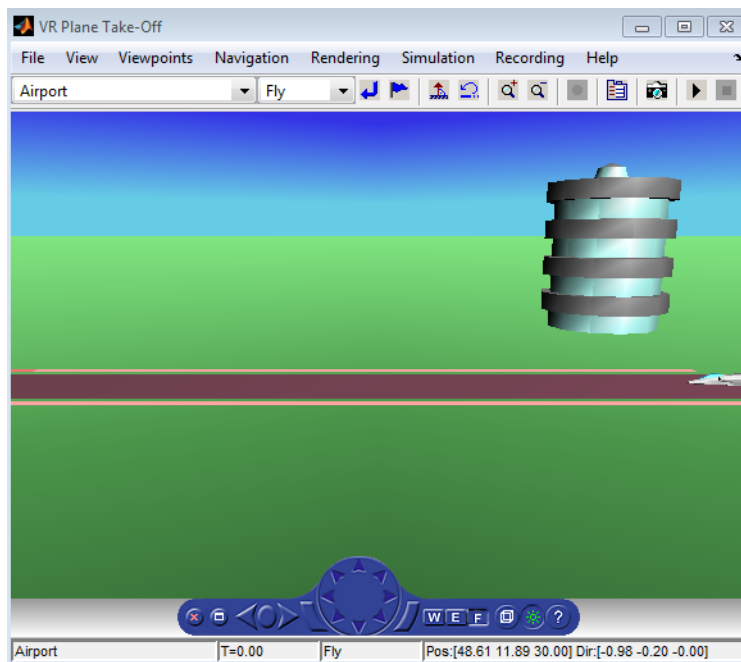
- 11 In the Simulink model, connect the line going to the Scope block labeled `Display Position` to the `Plane translation` input.

Remove the Scope blocks. Your model looks similar to the figure shown.

vrtut2



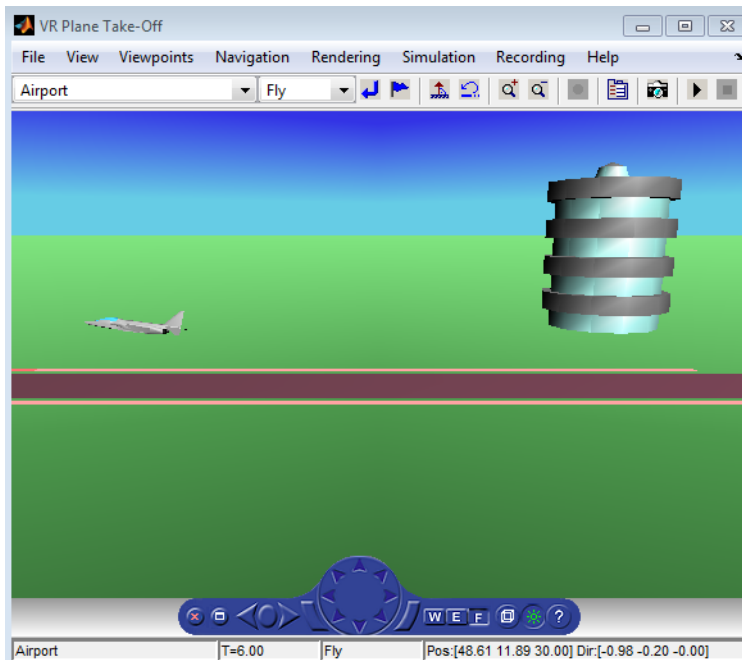
- 12 Double-click the VR Sink block. A viewer window containing the virtual world of the plane opens.



**Tip** When you next open the model, the associated virtual scene opens automatically. This behavior occurs even if the Simulink 3D Animation block associated with the virtual scene is in a subsystem of the model.

- 13 Run the simulation. In the Simulink 3D Animation Viewer, from the **Simulation** menu, click **Run**.

A plane, moving right to left, takes off .



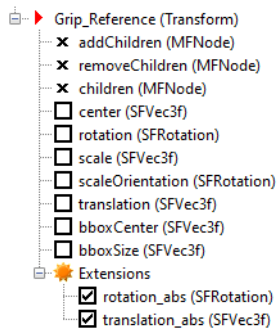
## Input Virtual World Data to a Model

You can use a VR Source block to provide interactivity between 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 block reads values from virtual world fields specified in the block dialog box and inputs their values to a model. Using the block in this way, you can:

- Use sensor data from a virtual world to control a simulation.
- 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 define 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 Magnetic Levitation Model and Virtual Control Panel.

To use global coordinates for a virtual world object, include a Transform node in that object. Open a second viewer window by double-clicking the VR Source block. In the second viewer window (which can overlap the first window), select **Simulation > Block Parameters**. For the Transform node of the object, select in the **Extensions** branch one or both of these Simulink 3D Animation extensions for converting rotation and translation values into global coordinates: `rotation_abs` and `translation_abs`.



See the Manipulator Moving a Load with Use of Global Coordinates example. For additional information about using the VR Source block and other approaches for provide interactivity in the model, see “Use Sensors” on page 5-20.

## Change the Associated Virtual World

You can associate a different virtual world with a Simulink model or connect different signals.

After you associate a virtual world with a Simulink model, you can select another virtual world or change signals connected to the virtual world. This example assumes that you have connected the `vrtut2` Simulink model with a virtual world. See “Input Virtual World Data to a Model” on page 3-6.

- 1 Double-click the VR Sink block in the model. The viewer opens.
- 2 Open the Block Parameters dialog box of the VR Sink block by selecting **Simulation > Block Parameters**.
- 3 At the **Source File** text box, click the **Browse** button. Find the folder `matlabroot\toolbox\sl3d\sl3ddemos`. Select the file `vrtkoff2.wrl`, and click **OK**. In the VR Sink dialog box, click **Apply**.

A virtual scene tree appears on the right side, associating a different virtual world with the model.

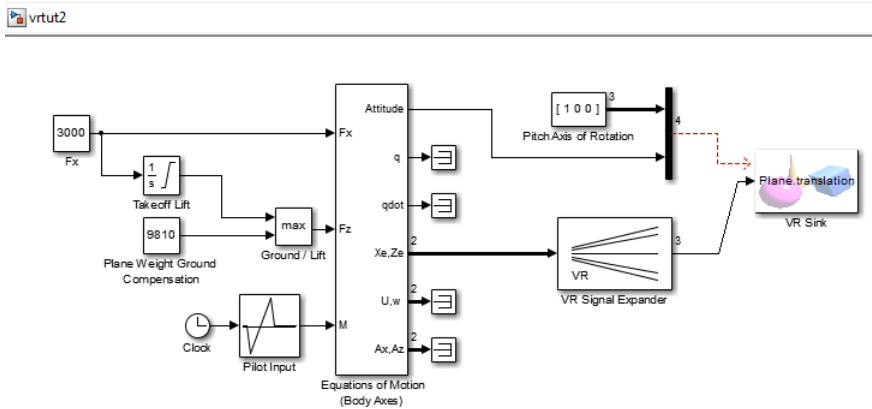
- 4 Expand the `Plane (Transform)` node.

The `Plane Transform` tree expands. Now you can see what characteristics of the plane you can drive from the Simulink interface. This model computes the position.

- 5 In the `Plane Transform` tree, select the `translation` field check box. Clear the `rotation` field check box. Click **OK**.

The VR Sink block is updated and changes to just one input, the `Plane translation`. The VR Sink block is ready to use with the new parameters defined.

- 6 Verify that the correct output is connected to your VR Sink block. Connect the output from the VR Signal Expander is connected to the single input.



- 7 In the Simulink 3D Animation Viewer, from the **Simulation** menu, run the simulation again and observe the simulation.

## See Also

### Functions

`vredit` | `vrjoystick` | `vrlib` | `vrspacemouse`

### Blocks

Joystick Input | Space Mouse Input | VR Sink | VR Source | VR Text Output | VR To Video | VR Tracer

## Related Examples

- “Add Sensors to Virtual Worlds” on page 5-20
- “Interact with Generated Code” on page 3-20
- “Link to Models”
- “Interact with Virtual Reality Worlds”

## Open a Viewer Window

When you simulate a model that contains a VR Sink block, your default viewer opens and displays the virtual scene. For more information on setting your default viewer, see “Set the Default Viewer” on page 2-2.

Multiple instances of the viewer can exist on your screen. A viewer appears each time you select the **File** menu **New Window** option in the Simulink 3D Animation Viewer. This feature is useful if you want to view one scene from many different viewpoints at the same time.

If you close the viewer window, you can reopen it. In the Simulink model window, double-click the VR Sink block.

### See Also

#### Functions

`vredit` | `vrgetpref` | `vrsetpref`

### Related Examples

- “Set the Default Viewer” on page 2-2
- “View Dynamic System Simulations”

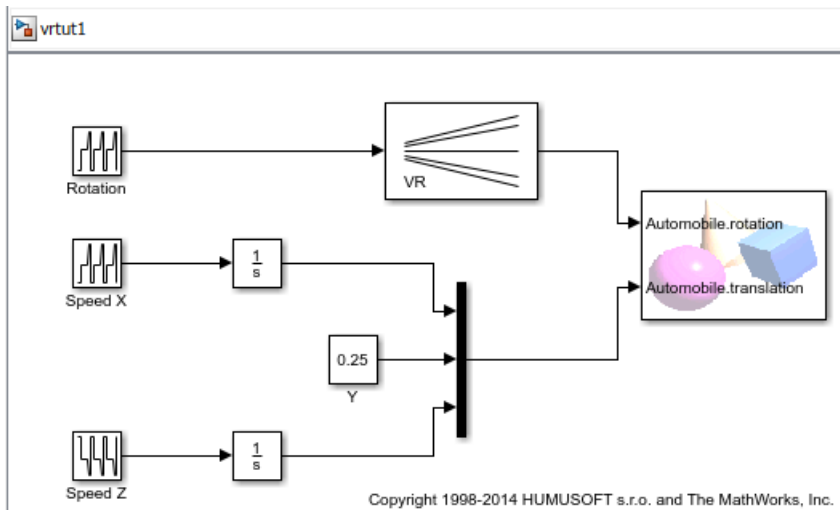
## Display Virtual World and Start Simulation

This example explains how to display a simulated virtual world using the Simulink 3D Animation Viewer on your host computer. The Simulink 3D Animation Viewer is the default and recommended method for viewing virtual worlds. A Simulink window opens with the model of a simple automobile. Automobile trajectory (vehicle position and angle) is viewed in virtual reality:

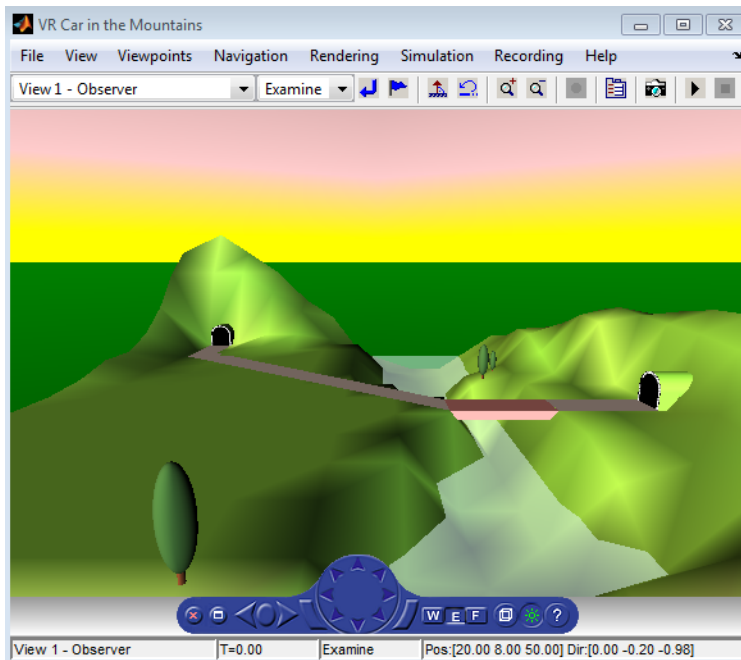
- 1 In the MATLAB Command Window, type

```
vrtut1
```

A Simulink window opens with the model of an automobile.



A virtual world viewer also opens with a 3-D model of the virtual world associated with the model.



- 2 In the Simulink 3D Animation Viewer, from the **Simulation** menu, click **Run**.

The simulation starts. In the Simulink 3D Animation Viewer, a car moves along the mountain road.

- 3 Use the Simulink 3D Animation Viewer controls to move the camera within this virtual world while the simulation is running. For more information on the Simulink 3D Animation Viewer controls, see “Simulink 3D Animation Viewer” on page 7-4.
- 4 In the Simulink 3D Animation Viewer, from the **Simulation** menu, click **Stop**.

## See Also

### Functions

`vredit` | `vrgetpref` | `vrsetpref`

### Blocks

VR Sink | VR Source

## Related Examples

- “Set the Default Viewer” on page 2-2
- “View Dynamic System Simulations”
- “View Virtual World on Host Computer” on page 3-12
- “View Virtual World Remotely” on page 3-15

## View Virtual World on Host Computer

Normally, you view a virtual world by double-clicking the VR Sink in the Simulink model. The virtual world opens in the Simulink 3D Animation Viewer or your HTML5-enabled web browser, depending on your `DefaultViewer` setting. For more information on setting your default viewer, see “Set the Default Viewer” on page 2-2.

Alternatively, you can view a virtual world in your web browser by selecting an open virtual world from a list in your web browser. You can display the HTML page that contains this list by connecting to the Simulink 3D Animation host. The host is the computer on which the Simulink 3D Animation software is running. You do not need an HTML5-enabled web browser to display this page.

A virtual world appears on this list in your web browser only if the `vrworld Description` property contains a string. If this property is empty for a virtual world, that world is not accessible from the remote host. The simplest way to set a world description is to define the virtual world 3D file `WorldInfo` node and fill in the `title` field for that node. You can set up the `WorldInfo` node to look like the following:

```
WorldInfo { title
  "My First World"
  info [ "Author: XY" ]
}
```

The `vrworld` object uses the `title` string in the virtual world 3D file for the `Description` property of the `vrworld` object. You can change this property with the Simulink 3D Animation MATLAB interface (`vrworld/set`).

The following procedure describes how to connect to the Simulink 3D Animation host:

- 1 At the MATLAB command prompt, type

```
vrbounce
```

The VR Bouncing Ball example is loaded and becomes active.

- 2 Open your HTML5-enabled web browser. In the address line of the browser, type

```
http://localhost:8123
```

---

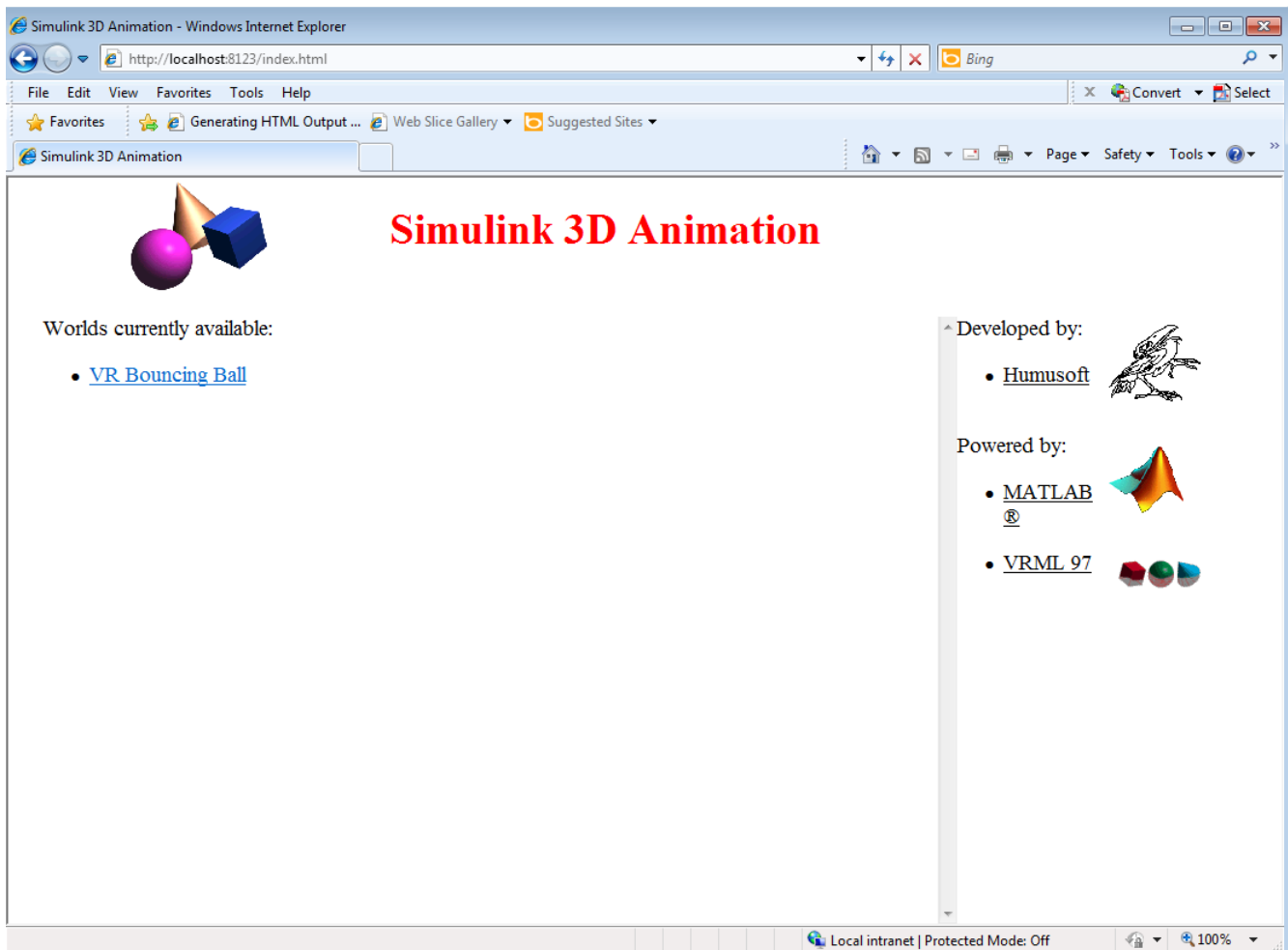
**Note** To connect to the main HTML page from a client computer, type `http://hostname:8123`, where `hostname` is the name of the computer on which the Simulink 3D Animation software is running.

---

The following page is loaded and becomes active.

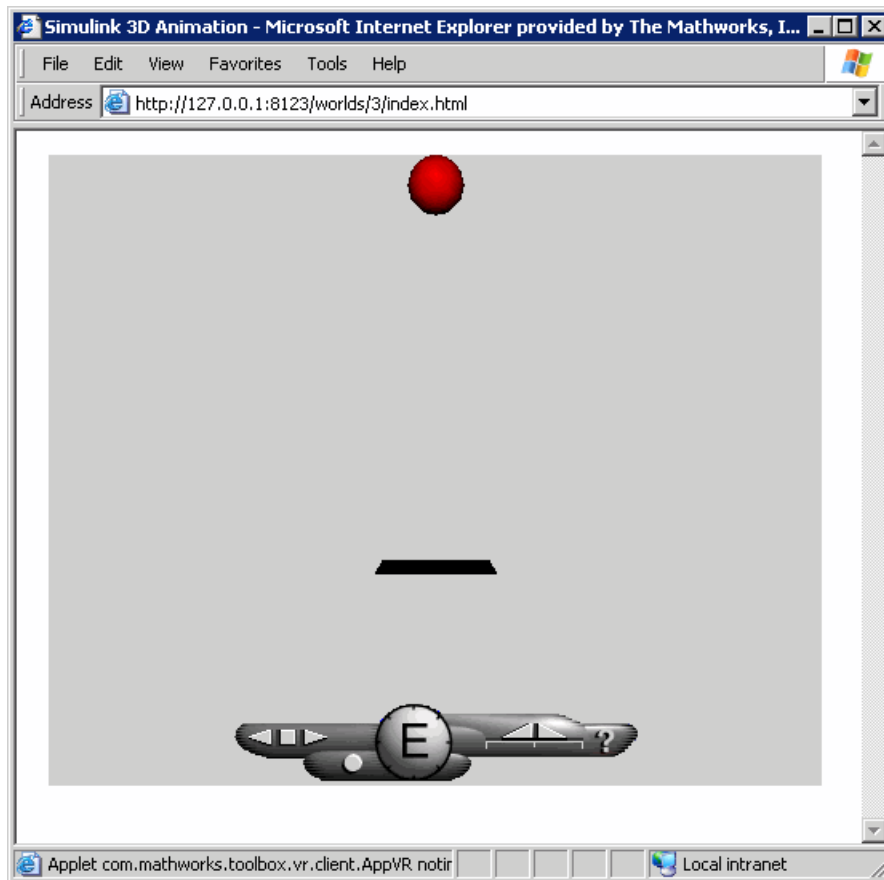
The main HTML page for the Simulink 3D Animation product lists the currently available (active) virtual worlds. In this example, the VR Bouncing Ball virtual world appears as a link.





**3 Click VR Bouncing Ball.**

The VR Bouncing Ball virtual world appears in your web browser.



From the main HTML page, select one of the listed worlds or click the **reload** link to update the status of the virtual worlds supported by the software. This page does not require the VRML or X3D capabilities from the browser; it is a standard HTML page. However, when you click one of the virtual world links in the list, the browser must be HTML5-enabled to display the virtual world correctly and to communicate with the Simulink 3D Animation product.

## See Also

### Related Examples

- “Display Virtual World and Start Simulation” on page 3-10
- “Set the Default Viewer” on page 2-2
- “View Dynamic System Simulations”
- “View Virtual World Remotely” on page 3-15

## View Virtual World Remotely

The Simulink 3D Animation software allows you to simulate a process on a host computer while running the visualization of the process on a client computer. You view the virtual world on the client computer using a web browser. This client computer is connected to the host computer through a network using the TCP/IP protocol. Setting up this configuration requires that you know the name or IP address of the host computer you want to access from the client computer.

Viewing a virtual world on a client computer is useful for:

- Remote computing
- Presentation of the results over the web
- Distribution of computing and graphical power

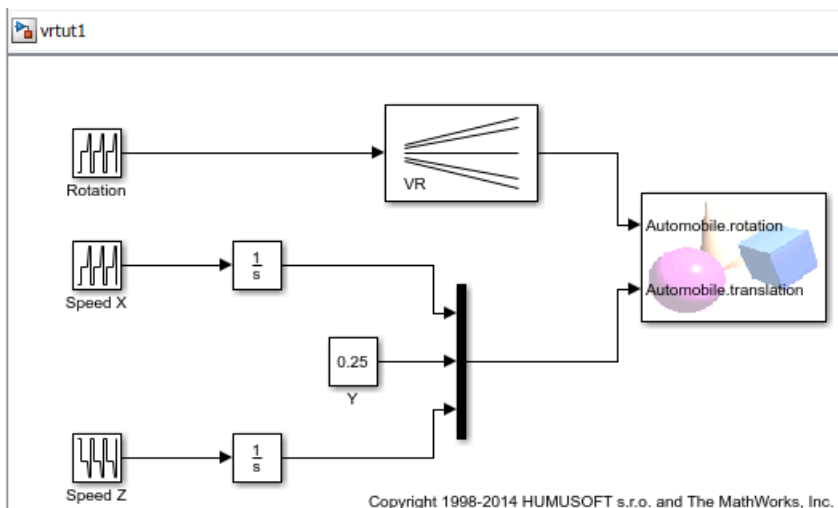
This example explains how to display a simulated virtual world on a client computer. In this case, the client computer is a PC platform with a Simulink 3D Animation Web Viewer. For a similar example using the Orbisnap viewer, see “View Virtual Worlds Remotely with Orbisnap” on page 8-13).

In the following example, a Simulink window opens with the model of a simple automobile. The automobile trajectory (vehicle position and angle) is viewed in virtual reality:

- 1 On the host computer, in the MATLAB Command Window, type

```
vrtut1
```

A Simulink window opens with the model of an automobile.

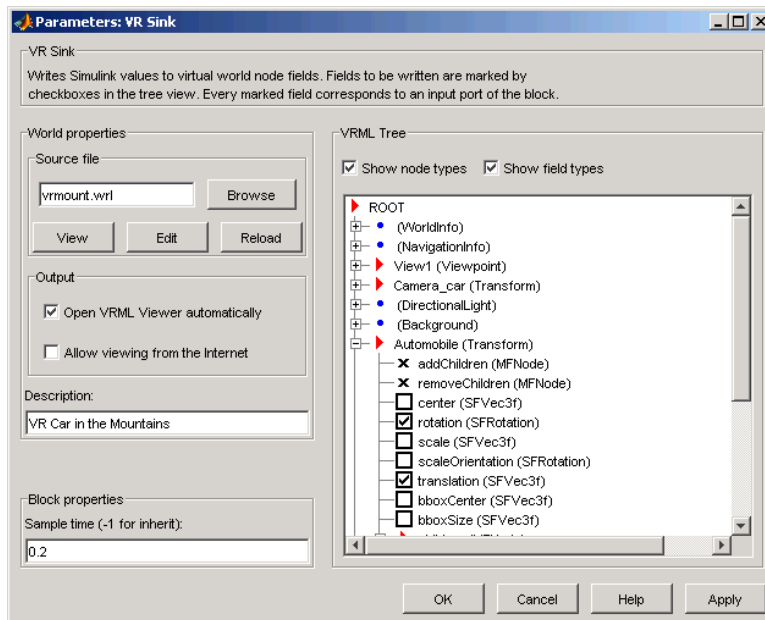


- 2 Double-click the VR Sink block. This block is in the right part of the model window.

A virtual world viewer also opens with a 3-D model of the virtual world associated with the model.

- 3 In the virtual world viewer, select the **Simulation** menu **Block Parameters** option.

A VR Sink block parameters dialog box opens.



- 4 Select the **Allow viewing from the Internet** check box.

---

**Note** This option allows any computer connected to the network to view your model. Do not select this box when you want your model to be private or confidential.

---

- 5 Click **OK**.
- 6 On the client computer, open your HTML5-enabled web browser. In the **Address** line, enter the address and Simulink 3D Animation port number for the host computer running the Simulink software. For example, if the IP address of the host computer is 192.168.0.1, enter:

http://192.168.0.1:8123

To determine your IP address on a Windows system, type `cmd`, and enter `ipconfig`.

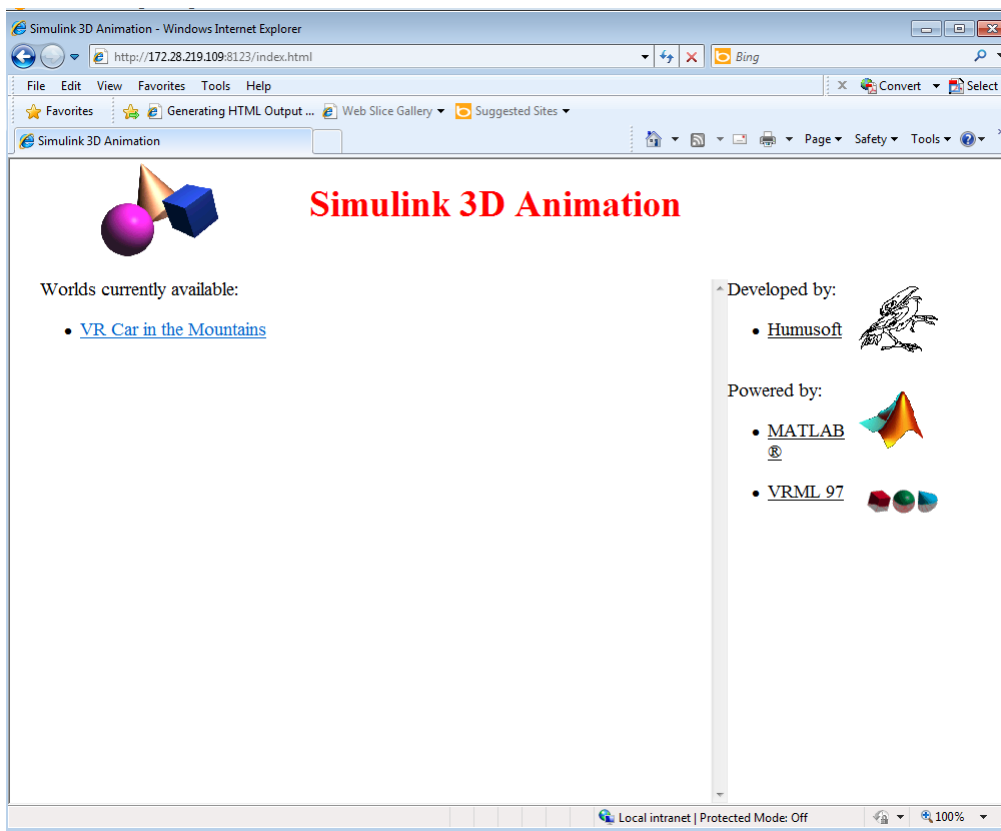
To determine your IP address on a UNIX system, type the command

```
ifconfig device_name
```

Click **OK**. An IP Configuration dialog box opens with a list of your IP, mask, and gateway addresses.

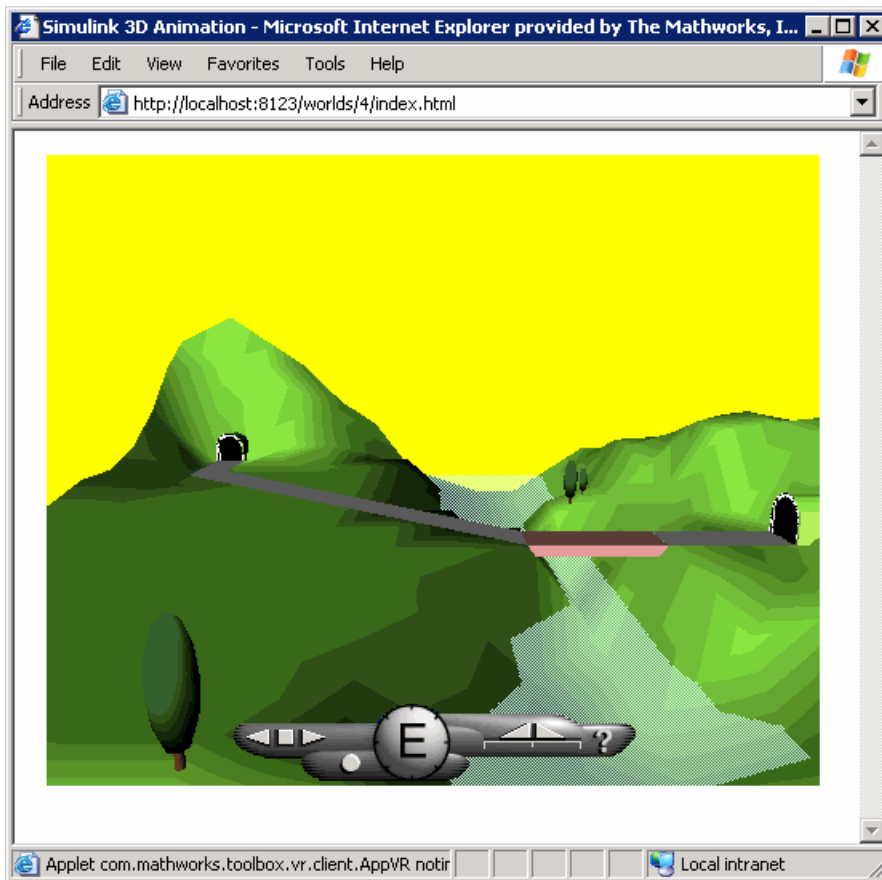
Alternatively, for Windows platforms, you can open a DOS shell and type `ipconfig`.

The web browser displays the main Simulink 3D Animation HTML page. Only one virtual world is in the list because you have only one Simulink model open.



**7 Click VR Car in the Mountains.**

The web browser displays a 3-D model of the virtual world associated with the model.



- 8 On the host computer, in the Simulink window, from the **Simulation** menu, click **Run**.

On the client computer, the animation of the scene reflects the process simulated in the Simulink diagram on the host computer.

You can tune communication between the host and the client computer by setting the **Sample time** and **Transport buffer size** parameters.

- 9 Use the web browser controls to move within this virtual world while the simulation is running.
- 10 On the host computer, in the Simulink window, from the **Simulation** menu, click **Stop**. On the client computer, close the web browser window.

## See Also

### Related Examples

- “Display Virtual World and Start Simulation” on page 3-10
- “Set the Default Viewer” on page 2-2
- “View Dynamic System Simulations”
- “View Virtual World on Host Computer” on page 3-12

## Modify Remote Virtual World Via Sensor Events

Interactive mode allows clients to modify a remote virtual world via events from sensor nodes defined in the virtual world. Interactive mode is useful when a virtual world includes a sensor.

Interactive mode is disabled by default on clients. You can enable (or later disable) interactive mode on a client via context menu in the Web Viewer or by pressing the **I** key shortcut.

You can disable interactive mode for a particular virtual world on the host computer. For details, see the `ClientUpdates` property, using `vrworld/get` or `vrworld/set`.

### See Also

#### Related Examples

- “Add Sensors to Virtual Worlds” on page 5-20
- “Read Sensor Values” on page 5-21

## Interact with Generated Code

You can have a virtual world that you create the Simulink 3D Animation product interact with code generated by the Simulink Coder product and compiled with a third-party C/C++ compiler in the Simulink Desktop Real-Time environment. To do so, use the Simulink External mode.

### See Also

### Related Examples

- “Connect Virtual Worlds and Models” on page 3-2



# MATLAB Interface

---

Although using the Simulink 3D Animation software with the Simulink interface is the preferred way of working with the Simulink 3D Animation software, you can also use the MATLAB interface. Enter commands directly in the MATLAB Command Window or use scripts to control virtual worlds.

- “Create vrworld Object for a Virtual World” on page 4-2
- “Open a Virtual World with MATLAB” on page 4-3
- “Interact with a Virtual World with MATLAB” on page 4-5
- “Close and Delete a vrworld Object” on page 4-9
- “Animation Recording” on page 4-10
- “Define File Name Tokens” on page 4-12
- “File Name Tokens” on page 4-14
- “Manual 3-D Recording with MATLAB” on page 4-16
- “Manual 2-D AVI Recording with MATLAB” on page 4-18
- “Scheduled 3-D Recording with MATLAB” on page 4-20
- “Scheduled 2-D AVI Recording with MATLAB” on page 4-22
- “Record Animations for Unconnected Virtual Worlds” on page 4-24
- “Play Animation Files” on page 4-27

## Create vrworld Object for a Virtual World

To connect MATLAB to a virtual world and to interact with that virtual world through the MATLAB command-line interface, create `vrworld` and `vrnode` objects. A virtual world 3D file defines a virtual world.

---

**Note** The Simulink interface and the MATLAB interface share virtual world objects. This sharing of objects enables you to use the MATLAB interface to change the properties of `vrworld` objects originally created by Simulink with Simulink 3D Animation blocks.

---

After you create a virtual world, you can create a `vrworld` object. This procedure uses the virtual world `vrmount.wrl` as an example.

- 1 Open MATLAB. In the MATLAB Command Window, type

```
myworld = vrworld('vrmount.wrl')
```

The MATLAB Command Window displays output like

```
myworld =  
  vrworld object: 1-by-1  
  
VR Car in the Mountains  
(matlabroot/toolbox/sl3d/vrdemos/vrmount.wrl)
```

- 2 Type

```
vrwhos
```

The MATLAB Command Window displays the messages

```
Closed, associated with  
'C:matlabroot\toolbox\sl3d\sl3ddemos\vrmount.wrl'.  
Visible for local viewers.  
No clients are logged on.
```

The `vrworld` object `myworld` is associated with the virtual world `vrmount.wrl`. You can think of the variable `myworld` as a handle to the `vrworld` object stored in the MATLAB workspace.

Your next step is to open a virtual world using the `vrworld` object. See “Open a Virtual World with MATLAB” on page 4-3.

## See Also

### Functions

`vrworld`

## Related Examples

- “Open a Virtual World with MATLAB” on page 4-3
- “Interact with a Virtual World with MATLAB” on page 4-5
- “Close and Delete a `vrworld` Object” on page 4-9

## Open a Virtual World with MATLAB

Open a virtual world to view the virtual world in a virtual world viewer, scan its structure, and change virtual world properties from the MATLAB Command Window.

After you create a `vrworld` object, you can open the virtual world by using the `vrworld` object associated with that virtual world. This procedure uses the `vrworld` object `myworld` associated with the virtual world `vrmount.wrl` as an example:

- 1 In the MATLAB Command Window, type

```
open(myworld);
```

The MATLAB Command Window opens the virtual world `vrmount.wrl`.

- 2 Type

```
set(myworld, 'Description', 'My first virtual world');
```

The `Description` property is changed to `My first virtual world`. This description displays in all Simulink 3D Animation 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.

- 3 Display the virtual world `vrmount.wrl`. Type

```
view(myworld)
```

The viewer that is set as the default viewer displays the virtual scene. The default viewer is typically the Simulink 3D Animation Viewer unless you have a different viewer set.

Alternatively, you can display the virtual world in an HTML5-enabled web browser.

- 1 Repeat steps 1 and 2 of the preceding procedure.
- 2 Open a web browser. In the **Address** box, type

```
http://localhost:8123
```

The browser displays the Simulink 3D Animation HTML page with a link to **My first virtual world**. The number `8123` is the default Simulink 3D Animation port number. If you set a different port number on your system, enter that number in place of `8123` and restart MATLAB. For more information on the Simulink 3D Animation HTML page, see “View Virtual World on Host Computer” on page 3-12.

- 3 If the web browser has the VRML or X3D plug-in installed, in the browser window, click **My first virtual world**.
- 4 Your default HTML5-enabled web browser displays the virtual world `vrmount.wrl`.

---

**Note** If your web browser is not HTML5-enabled, clicking a virtual world link such as **My first virtual world** results in a broken link message. The browser cannot display the virtual world.

---

For more information on changing your default viewer, see “Set the Default Viewer” on page 2-2.

## **See Also**

### **Functions**

vrworld

### **Related Examples**

- “Create vrworld Object for a Virtual World” on page 4-2
- “Interact with a Virtual World with MATLAB” on page 4-5
- “Close and Delete a vrworld Object” on page 4-9

## Interact with a Virtual World with MATLAB

### In this section...

“Set Values for Nodes” on page 4-5

“Read Sensor Values Using MATLAB” on page 4-7

### Set Values for Nodes

In the life cycle of a `vrworld` object you can set new values for all the available virtual world nodes and their fields using `vrnode` object methods. This way, you can change and control the degrees of freedom for the virtual world from within the MATLAB environment.

An object of type `vrworld` contains nodes named in the virtual world 3D file using the `DEF` statement. These nodes are of type `vrnode`. For more information, see `vrworld` and `vrnode` functions.

After you open a `vrworld` object, you can get a list of available nodes in the virtual world. This procedure uses the `vrworld` object `myworld` and the virtual world `vrmount.wrl` as an example. To create the `myworld`, see “Create `vrworld` Object for a Virtual World” on page 4-2.

- 1 In the MATLAB Command Window, type

```
nodes(myworld);
```

The MATLAB Command Window displays a list of the `vrnode` objects and their fields that are accessible from the Simulink 3D Animation software.

```
Tunnel (Transform) [My first virtual world]
Road (Shape) [My first virtual world]
Bridge (Shape) [My first virtual world]
River (Shape) [My first virtual world]
ElevApp (Appearance) [My first virtual world]
Canal (Shape) [My first virtual world]
Wood (Group) [My first virtual world]
Tree1 (Group) [My first virtual world]
Wheel (Shape) [My first virtual world]
Automobile (Transform) [My first virtual world]
VPfollow (Viewpoint) [My first virtual world]
Camera_car (Transform) [My first virtual world]
View1 (Viewpoint) [My first virtual world]
```

- 2 Type

```
mynodes = get(myworld, 'Nodes')
```

The MATLAB software creates an array of `vrnode` objects corresponding to the virtual world nodes and displays

```
mynodes =
```

```
vrnode object: 13-by-1
```

```
Tunnel (Transform) [My first virtual world]
Road (Shape) [My first virtual world]
Bridge (Shape) [My first virtual world]
```

```

River (Shape) [My first virtual world]
ElevApp (Appearance) [My first virtual world]
Canal (Shape) [My first virtual world]
Wood (Group) [My first virtual world]
Tree1 (Group) [My first virtual world]
Wheel (Shape) [My first virtual world]
Automobile (Transform) [My first virtual world]
VPfollow (Viewpoint) [My first virtual world]
Camera_car (Transform) [My first virtual world]
View1 (Viewpoint) [My first virtual world]

```

**3** Type

```
whos
```

The MATLAB Command Window displays the messages

Name	Size	Bytes	Class
ans	1x1	132	vrfigure object
mynodes	13x1	3564	vrnode object
myworld	1x1	132	vrworld object

Now you can get node characteristics and set new values for certain node properties. For example, you can change the position of the automobile by using `Automobile`, which is the fourth node in the virtual world.

**4** Access the fields of the Automobile node by typing

```
fields(myworld.Automobile)
```

or

```
fields(mynodes(10));
```

The MATLAB Command Window displays information about the Automobile node.

Field	Access	Type	Sync
addChildren	eventIn	MFNode	off
removeChildren	eventIn	MFNode	off
children	exposedField	MFNode	off
center	exposedField	SFVec3f	off
rotation	exposedField	SFRotation	off
scale	exposedField	SFVec3f	off
scaleOrientation	exposedField	SFRotation	off
translation	exposedField	SFVec3f	off
bboxCenter	field	SFVec3f	off
bboxSize	field	SFVec3f	off

The Automobile node is of type Transform. This node allows you to change its position by changing its `translation` field values. From the list, you can see that `translation` requires three values, representing the [x y z] coordinates of the object.

**5** Type

```
view(myworld)
```

Your default viewer opens and displays the virtual world `vrmount.wrl`.

- 6 Move the MATLAB window and the browser window side by side so you can view both at the same time. In the MATLAB Command Window, type

```
myworld.Automobile.translation = [15 0.25 20];
```

The MATLAB sets a new position for the `Automobile` node. You can observe that the car is repositioned in the virtual world browser window.

You can change the node fields listed by using the function `vrnode/setfield`.

---

**Note** The dot notation is the preferred method for accessing nodes.

---

## Read Sensor Values Using MATLAB

To read a value of a readable field (either `exposedField` or `eventOut`), first synchronize that field with the `vrnode/sync` method. After synchronization, each time the field changes in the scene, the field value updates on the host. You can then read the value of the field with the `vrnode/getfield` method or directly access the field value using dot notation.

The virtual scene for the Magnetic Levitation Model example, `maglev.wrl`, contains a `PlaneSensor` (with the DEF name `'Grab_Sensor'`). The `PlaneSensor` is attached to the ball geometry to register your attempts to move the ball up or down when grabbing it using the mouse. The example uses the sensor fields `minPosition` and `maxPosition` to restrict movement in other directions. You can use the output of the sensor translation field as the new setpoint for the ball position controller. You can read the sensor output value into a MATLAB variable setpoint.

- 1 Create the `vrworld` object and open the world.

```
wh = vrworld('maglev.wrl');
open(wh);
```

- 2 Get the node handle.

```
nh = vrnode(wh, 'Grab_Sensor');
```

- 3 Synchronize the translation field.

```
sync(nh, 'translation', 'on');
```

- 4 Read the synchronized field value, using one of these three alternatives:

```
setpoint = getfield(nh, 'translation');
setpoint = nh.translation;
setpoint = wh.Grab_Sensor.translation;
```

## Global Coordinates for Rotation and Translation

Rotation and translation values for a `Transform` object are specified in local coordinates, relative to the parent object of the object. Simulink 3D Animation provides two extensions for converting rotation and translation values into global coordinates: `rotation_abs` and `translation_abs`. To access these global coordinates, use dot notation with the translation or rotation field, adding `_abs` to the field name. This example shows the difference between the local and global coordinates for translation:

```
w = vrview('vrmanipul.wrl');  
n = get(w, 'Nodes');  
n = w.Grip_Reference;  
n.translation  
n.translation_abs  
  
ans =  
  
         0   -0.1000         0  
  
ans =  
  
   -3.0406   -3.0000    2.3334
```

## See Also

### Functions

vrworld

## Related Examples

- “Create vrworld Object for a Virtual World” on page 4-2
- “Open a Virtual World with MATLAB” on page 4-3
- “Close and Delete a vrworld Object” on page 4-9



## Close and Delete a vrworld Object

After you are finished with a session, close all open virtual worlds and remove them from memory:

- 1 In the MATLAB Command Window, type

```
close(myworld);  
delete(myworld);
```

The virtual world representation of the `vrworld` object `myworld` is removed from memory. All possible connections to the viewer and browser are closed and the virtual world name is removed from the list of available worlds.

---

**Note** Closing and deleting a virtual world does not delete the `vrworld` object handle `myworld` from the MATLAB workspace.

---

### See Also

#### Functions

`vrworld`

### Related Examples

- “Create `vrworld` Object for a Virtual World” on page 4-2
- “Open a Virtual World with MATLAB” on page 4-3
- “Interact with a Virtual World with MATLAB” on page 4-5

## Animation Recording

### In this section...

“Recording Formats” on page 4-10

“Manual and Scheduled Animation Recording” on page 4-10

The Simulink 3D Animation software enables you to record animations of virtual scenes that the Simulink or MATLAB product controls. You can record simulations through either the Simulink 3D Animation Viewer (described in “Simulink 3D Animation Viewer” on page 7-4) or the MATLAB interface. You can then play back these animations offline, in other words, independent of the MATLAB, Simulink, or Simulink 3D Animation products. You can generate such files for presentations, to distribute simulation results, or to generate archives.

---

**Note** If you are working with virtual scenes controlled from MATLAB, you can record virtual scenes through the MATLAB interface. Optimally, use the Simulink 3D Animation Viewer to record animations of virtual worlds associated with Simulink models. This method ensures that all necessary virtual world and `vrfigure` properties are properly set to record simulations. For details, see “Record Offline Animations” on page 7-29.

---

### Recording Formats

You can save the virtual world offline animation data in the following formats:

- 3-D virtual world file — The Simulink 3D Animation software traces object movements and saves that data into a virtual world 3D file using VRML97 standard interpolators. You can then view these files with the Simulink 3D Animation Viewer. 3-D VRML files typically use much less disk space than Audio Video Interleave (AVI) files. If you make any navigation movements in the Simulink 3D Animation Viewer while recording the animation, the Simulink 3D Animation software does not save any of these movements.

---

**Note** If you distribute virtual world 3D animation files, be sure to distribute all the inlined object and texture files referenced in the original virtual world 3D world file.

---

- 2-D Audio Video Interleave (AVI) file — The Simulink 3D Animation software writes animation data into an `.avi` file. The Simulink 3D Animation software uses `vrfigure` objects to record 2-D animation files. The recorded 2-D animation reflects exactly what you see in the viewer window. It includes any navigation movements you make during the recording.

---

**Note** While recording 2-D `.avi` animation data, always ensure that the Simulink 3D Animation Viewer is the topmost window and fully visible. Graphics acceleration limitations can prevent the proper recording of 2-D animation otherwise.

---

### Manual and Scheduled Animation Recording

You can use MATLAB to either manually record an animation or schedule a preset time interval for recording. For details, see:

- “Manual 3-D Recording with MATLAB” on page 4-16

- “Manual 2-D AVI Recording with MATLAB” on page 4-18
- “Scheduled 3-D Recording with MATLAB” on page 4-20
- “Scheduled 2-D AVI Recording with MATLAB” on page 4-22

## **See Also**

### **Functions**

`vrplay` | `vrview`

## **Related Examples**

- “Share Visualizations”
- “Define File Name Tokens” on page 4-12

## **More About**

- “File Name Tokens” on page 4-14

## Define File Name Tokens

### In this section...

“Default File Name Format” on page 4-12

“Uses for File Name Tokens” on page 4-12

### Default File Name Format

By default, the Simulink 3D Animation Viewer records simulations or captures virtual scene frames in a file named with the following format:

```
%f_anim_%n.%e
```

This format creates a unique file name each time you capture a frame or record the animation. The file name uses the %f, %n, and %e tokens.

The %f token is replaced with the name of the virtual world associated with the model. The %n token is a number that increments each time that you record a simulation for the same virtual world. For example, if the name of the virtual world file is `vrplanets.vrml` and you record a simulation for the first time, the animation file is `vrplanets_anim_1.vrml`. If you record the simulation a second time, the animation file name is `vrplanets_anim_2.vrml`. In the case of frame captures, capturing another frame of the scene increments the number.

The %e token represents the virtual world 3D file extension (`.vrml`, `.x3d`, or `.x3dv`) as the extension of the virtual world that drives the animation. By default, the %e token uses the file extension of the virtual world 3D file that drives the animation. The VR Sink and VR Source block **Source file** parameter specifies the file extension of the virtual world. You can specify a different extension. However, if the file extension in the **Source file** parameter is `.x3d` or `.x3dv`, you cannot set %e token to `.vrml` (VRML).

### Uses for File Name Tokens

You can use several tokens to customize the automated generation of frame capture or animation files. To use these tokens to create varying frame capture or animation file names, you can:

- Create files whose root names are the same as the root names of the virtual world. This option is useful if you use different virtual worlds for one model.
- Create files in directories relative to the virtual world location. This option is useful if you want to ensure that the virtual world file and frame capture or animation file are in the same folder.
- Create rolling numbered file names such that subsequent frame captures or runs of the model simulation create incrementally numbered file names. This approach is useful if you expect to create files of different parts of the model simulation. This feature allows you to capture a frame or run a Simulink model multiple times, but create a unique file each time.
- Create multiple file names with time or date stamps, with a unique file created each time.

See “File Name Tokens” on page 4-14 for a summary of the file name tokens.

## **See Also**

### **Related Examples**

- “Share Visualizations”

### **More About**

- “File Name Tokens” on page 4-14

## File Name Tokens

The software supports various file naming formats using file tokens. By default, the Viewer captures virtual scene frames or records simulations in a file named with the following format: `%f_anim_%n.%e`. This format creates a unique file name each time you capture a frame or record the animation.

The following tokens are the same for frame capture (`.tif` or `.png`) or animation (`.wrl`, `.x3d`, `.x3dv`, and `.avi`) files.

Token	Description
<code>%n</code>	The current incremental number replaces this token in the file name string. Each subsequent frame capture or run of the simulation increments the number. For example, the format <code>%f_anim_%n.wrl</code> saves the animation to <code>vrplanets_anim_1.wrl</code> on the first run, <code>vrplanets_anim_2.wrl</code> on the second run, and so forth.
<code>%f</code>	The virtual world file name replaces this token in the file name string. For example, the format <code>%f_anim_%D.wrl</code> saves the animation to <code>vrplanets_anim_29.wrl</code> .
<code>%e</code>	The virtual world file name replaces this token represents with the virtual world 3D file extension ( <code>.wrl</code> , <code>.x3d</code> , or <code>.x3dv</code> ). By default, the <code>%e</code> token uses the file extension of the virtual world 3D file that drives the animation. The VR Sink and VR Source block <b>Source file</b> parameter specifies the file extension of the virtual world.  You can specify a different extension. However, if the file extension in the <b>Source file</b> parameter is <code>.x3d</code> or <code>.x3dv</code> , you cannot set <code>%e</code> token to <code>.wrl</code> (VRML).
<code>%d</code>	The full path to the virtual world 3D file replaces this token in the file name string and creates files in directories relative to the virtual world file location. For example, the format <code>%d/animdir/%f_anim_%n.avi</code> saves the animation in the <code>animdir</code> subfolder of the folder containing the virtual world 3D file. If <code>animdir</code> subfolder does not exist, the software creates the <code>animdir</code> subfolder. This token is most helpful if you want to ensure that the virtual world file and animation file are in the same folder.
<code>%Y</code>	The current four-digit year replaces this token in the file name string. For example, the format <code>%f_anim_%Y.wrl</code> saves the animation to <code>vrplanets_anim_2015.wrl</code> for the year 2015.
<code>%M</code>	The current month replaces this token in the file name string. For example, the format <code>%f_anim_%M.wrl</code> saves the animation to <code>vrplanets_anim_4.wrl</code> for a start record time in April.
<code>%D</code>	The current day in the month replaces this token in the file name string. For example, the format <code>%f_anim_%D.wrl</code> saves the animation to <code>vrplanets_anim_29.wrl</code> for the 29th day of the month.
<code>%h</code>	The current hour replaces this token in the file name string. For example, the format <code>%f_anim_%h.wrl</code> saves the animation to <code>vrplanets_anim_14.wrl</code> for any time between 14:00 and 15:00.
<code>%m</code>	The current minute replaces this token in the file name string. For example, the format <code>%f_anim_%h%m.wrl</code> saves the animation to <code>vrplanets_anim_1434.wrl</code> for a start record time of 14:34.

Token	Description
%S	The current second replaces this token in the file name string. For example, the format %f_anim_%h%m%S.wrl saves the animation to vrplanets_anim_150430.wrl for a start record time of 15:04:30.

## See Also

### Related Examples

- “Define File Name Tokens” on page 4-12
- “Share Visualizations”

## Manual 3-D Recording with MATLAB

This topic describes how to record a 3-D animation manually using the MATLAB interface for a virtual world that is associated with a Simulink model. In this example, the timing of the animation file derives from the simulation time. One second of the recorded animation time corresponds to one second of Simulink time. You create and record the animation file by interactively starting and stopping the recording from the MATLAB Command Window.

This procedure uses the `vrplanets` example. It describes how to create a virtual world 3D animation file name with the default name format.

- 1 Run the Simulink model for `vrplanets`. In the MATLAB window, type

```
vrplanets
```

The Simulink model appears. Also by default, the Simulink 3D Animation Viewer for that model is loaded and becomes active. If the viewer does not appear, double-click the Simulink® 3D Animation block in the Simulink model.

- 2 To work with the virtual world associated with `vrplanets` from the MATLAB interface, retrieve the virtual world handle. Use the `vrwhos` command. Type

```
vrwhos
```

If the result shows that only one `vrworld` object is in the workspace, assign its handle directly to a variable. Type

```
myworld = vrwho;
```

If multiple virtual worlds are listed, select which of these virtual worlds you want to manipulate. To select the virtual world, you can use indexing or a selection method using a string comparison of virtual world descriptions. For the indexing method, type

```
worlds = vrwho;
myworld = worlds(1);
```

For the string comparison method, type

```
worlds = vrwho;
myworld =
worlds(strcmp('Planets',get(worlds,'Description')));
```

- 3 To have the Simulink 3D Animation software manually record the animation, set the `RecordMode` property to `manual`. Type

```
set(myworld,'RecordMode','manual');
```

- 4 Direct the Simulink 3D Animation software to record the animation to a virtual world 3D format file. Type

```
set(myworld,'Record3D','on');
```

- 5 Run the Simulink model. From the **Simulation** menu, select **Mode > Normal**, then click **Simulation > Run**. Alternatively, if you are using the Simulink 3D Animation default viewer, you can run the Simulink model with one of the following from the viewer.

- From the menu bar, select the **Simulation** menu **Start** option to start or stop the simulation.
- From the toolbar, click **Start/pause/continue simulation** to start the simulation.



- From the keyboard, press **Ctrl+T** to start the simulation.
- 6** As the simulation runs, start recording the animation by setting the virtual world **Recording** property. Type

```
set(myworld, 'Recording', 'on');
```

This setting turns on the recording state.

- 7** When you want to stop the recording operation, type:

```
set(myworld, 'Recording', 'off');
```

The Simulink 3D Animation software stops recording the animation. The Simulink 3D Animation software creates the file `vrplanets_anim_1.wrl` in the current working folder. If the simulation stops before you stop recording, the recording operation stops and creates the animation file.

- 8** Stop the simulation. You can use one of the following from the viewer.
- From the menu bar, select the **Simulation** menu **Stop** option to stop the simulation.
  - From the toolbar, click **Stop simulation** to stop the simulation.
  - From the keyboard, press **Ctrl+T** to stop the simulation.

You do not need to stop the recording manually before stopping the simulation. If you do not manually stop the recording, the recording operation does not stop and create the animation file when the simulation stops.

- 9** Close and delete the objects if you do not want to continue using them.

## See Also

### Related Examples

- “Manual 2-D AVI Recording with MATLAB” on page 4-18
- “Scheduled 3-D Recording with MATLAB” on page 4-20
- “Scheduled 2-D AVI Recording with MATLAB” on page 4-22
- “Record Animations for Unconnected Virtual Worlds” on page 4-24
- “Play Animation Files” on page 4-27

### More About

- “File Name Tokens” on page 4-14

## Manual 2-D AVI Recording with MATLAB

This topic describes how to record a 2-D animation manually using the MATLAB interface for a virtual world that is associated with a Simulink model. In this example, the timing of the animation file derives from the simulation time. One second of the recorded animation time corresponds to one second of Simulink time. You create and record the animation file by interactively starting and stopping the recording from the MATLAB Command Window.

This procedure uses the `vrplanets` example. It describes how to create an `.avi` animation file name with the default name format.

- 1 Run the Simulink model for `vrplanets`. In the MATLAB window, type

```
vrplanets
```

The Simulink model appears. Also by default, the Simulink 3D Animation Viewer for that model is loaded and becomes active. If the viewer does not appear, double-click the Simulink® 3D Animation block in the Simulink model.

- 2 To work with the virtual world associated with `vrplanets` from the MATLAB interface, retrieve the virtual world handle. Use the `vrwhos` command. Type

```
vrwhos
```

- 3 If the result indicates that only one `vrworld` object is in the workspace, assign its handle directly to a variable. Type

```
myworld = vrwho;
```

If multiple virtual worlds are listed, select which of these virtual worlds you want to manipulate. To select the virtual world, you can use indexing or a selection method using a string comparison of virtual world descriptions. For the indexing method, type

```
worlds = vrwho;
myworld = worlds(1);
```

For the string comparison method, type

```
worlds = vrwho;
myworld =
worlds(strcmp('Planets',get(worlds,'Description')));
```

If the description string is unique, `myworld` is assigned the correct virtual world.

- 4 To retrieve the handle to the currently displayed the Simulink 3D Animation Viewer figure, type

```
f=get(myworld,'Figures')
```

- 5 To have the software manually record the animation, set the `RecordMode` property to `manual`. Type

```
set(myworld,'RecordMode','manual');
```

- 6 Direct the Simulink 3D Animation software to record the animation as a `.avi` format file. Type

```
set(f,'Record2D','on');
```

- 7 Disable the navigation panel. The navigation panel appears at the bottom of the virtual scene view. You can turn off this panel for a cleaner view of the virtual scene. Type

```
set(f,'NavPanel','none');
```

- 8** Run the Simulink model. From the **Simulation** menu, select **Mode > Normal**, then click **Simulation > Run**. Alternatively, if you are using the Simulink 3D Animation default viewer, you can run the Simulink model with one of the following from the viewer:
- From the menu bar, select the **Simulation** menu **Start** option to start or stop the simulation.
  - From the toolbar, click **Start/pause/continue simulation** to start the simulation.
  - From the keyboard, press **Ctrl+T** to start the simulation.
- 9** As the simulation runs, start recording the animation by setting the virtual world **Recording** property. Type

```
set(myworld, 'Recording', 'on');
```

This setting turns on the recording state.

- 10** To stop the recording operation, type:

```
set(myworld, 'Recording', 'off');
```

The Simulink 3D Animation software stops recording the animation. The Simulink 3D Animation software creates the file `vrplanets_anim_1.avi` in the current working folder. If the simulation stops before you stop recording, the recording operation stops and creates the animation file.

- 11** Stop the simulation. You can use one of the following from the viewer.

- From the menu bar, select the **Simulation** menu **Stop** option to stop the simulation.
- From the toolbar, click **Stop simulation** to stop the simulation.
- From the keyboard, press **Ctrl+T** to stop the simulation.

You do not need to stop the simulation manually. If you do not manually stop the recording, the recording operation does not stop and create the animation file until the simulation stops.

- 12** If you want to enable the Navigation Panel again, type

```
set(f, 'NavPanel', 'halfbar');
```

- 13** Close and delete the objects if you do not want to continue using them.

## See Also

### Related Examples

- “Manual 3-D Recording with MATLAB” on page 4-16
- “Scheduled 3-D Recording with MATLAB” on page 4-20
- “Scheduled 2-D AVI Recording with MATLAB” on page 4-22
- “Record Animations for Unconnected Virtual Worlds” on page 4-24
- “Play Animation Files” on page 4-27

### More About

- “File Name Tokens” on page 4-14

## Scheduled 3-D Recording with MATLAB

This topic describes how to schedule the recording of a 3-D animation using the MATLAB interface for a virtual world that is associated with a Simulink model. You control the animation file recording by presetting a time interval. The Simulink 3D Animation software records the animation during this interval in the simulation. In this example, the timing of the recorded animation file derives from the simulation time. One second of the recorded animation time corresponds to one second of Simulink time.

This procedure uses the `vrplanets` example. It describes how to create a virtual world 3D animation file name with the default name format.

- 1 Run the Simulink model for `vrplanets`. In the MATLAB window, type

```
vrplanets
```

The Simulink model is displayed. Also by default, the Simulink 3D Animation Viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the Simulink® 3D Animation block in the Simulink model.

- 2 To work with the virtual world associated with `vrplanets` from the MATLAB interface, retrieve the virtual world handle. Use the `vrwhos` command. Type

```
vrwhos
```

- 3 If the result indicates that only one `vrworld` object is in the workspace, assign its handle directly to a variable. Type

```
myworld = vrwho;
```

If multiple virtual worlds are listed, select which of these virtual worlds you want to manipulate. To select the virtual world, you can use indexing or a selection method using a string comparison of virtual world descriptions. For the indexing method, type

```
worlds = vrwho;  
myworld = worlds(1);
```

For the string comparison method, type

```
worlds = vrwho;  
myworld =  
worlds(strcmp('Planets',get(worlds,'Description')));
```

- 4 Direct the Simulink 3D Animation software to record the animation on a schedule by setting the `RecordMode` property to `scheduled`. Type

```
set(myworld,'RecordMode','scheduled');
```

- 5 Direct the Simulink 3D Animation software to record the animation in a virtual world 3D format file.

```
set(myworld,'Record3D','on');
```

- 6 Select the start and stop times during which you want to record the animation. For example, enter 5 as the start time and 15 as the stop time.

```
set(myworld,'RecordInterval',[5 15]);
```

Ensure that the recording start time value is not earlier than the start time of the Simulink model; the recording operation cannot start in this instance. If the stop time exceeds the stop

time of the Simulink model, or if it is an out of bounds value such as a negative number, the recording operation stops when the simulation stops. The recording can be slow.

- 7 Run the Simulink model. From the **Simulation** menu, select **Mode > Normal**, then click **Simulation > Run**. Alternatively, if you are using the Simulink 3D Animation default viewer, you can run the Simulink model with one of the following from the viewer.
  - From the menu bar, select the **Simulation** menu **Start** option to start the simulation.
  - From the toolbar, click **Start/pause/continue simulation** to start the simulation.
  - From the keyboard, press **Ctrl+T** to start the simulation.

The simulation runs. The Simulink 3D Animation software starts recording when the simulation time reaches the specified start time. The software creates the file `vrplanets_anim_N.wrl` in the current working folder when finished, where N is either 1 or more, depending on how many file iterations you have.

- 8 When you are done, stop the simulation. You can use one of the following from the viewer.
  - From the menu bar, select the **Simulation** menu **Stop** option to stop the simulation.
  - From the toolbar, click **Stop simulation** to stop the simulation.
  - From the keyboard, press **Ctrl+T** to stop the simulation.
- 9 Close and delete the objects if you do not want to continue using them.

## See Also

### Related Examples

- “Manual 3-D Recording with MATLAB” on page 4-16
- “Manual 2-D AVI Recording with MATLAB” on page 4-18
- “Scheduled 2-D AVI Recording with MATLAB” on page 4-22
- “Record Animations for Unconnected Virtual Worlds” on page 4-24
- “Play Animation Files” on page 4-27

### More About

- “File Name Tokens” on page 4-14

## Scheduled 2-D AVI Recording with MATLAB

This topic describes how to schedule the recording of a 2-D animation using the MATLAB interface for a virtual world that is associated with a Simulink model. You control the animation file recording by presetting a time interval. The Simulink 3D Animation software records the animation during this interval in the simulation. In this example, the timing of the recorded animation file derives from the simulation time. One second of the recorded animation time corresponds to one second of Simulink time.

This procedure uses the `vrplanets` example. It describes how to create an `.avi` animation file name with the default name format.

- 1 Run the Simulink model for `vrplanets`. In the MATLAB window, type

```
vrplanets
```

The Simulink model is displayed. Also by default, the Simulink 3D Animation Viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the Simulink® 3D Animation block in the Simulink model.

- 2 To work with the virtual world associated with `vrplanets` from the MATLAB interface, retrieve the virtual world handle. Use the `vrwhos` command. Type

```
vrwhos
```

If the result indicates that only one `vrworld` object is in the workspace, assign its handle directly to a variable. Type

```
myworld = vrwho;
```

If multiple virtual worlds are listed, select which of these virtual worlds you want to manipulate. To select the virtual world, you can use indexing or a selection method using a string comparison of virtual world descriptions. For the indexing method, type

```
worlds = vrwho;  
myworld = worlds(1);
```

For the string comparison method, type

```
worlds = vrwho;  
myworld =  
worlds(strcmp('Planets',get(worlds,'Description')));
```

- 3 To retrieve the handle to the currently displayed Simulink 3D Animation Viewer figure, type  

```
f=get(myworld,'Figures')
```
- 4 To have the Simulink 3D Animation software manually record the animation, set the `RecordMode` property to `manual`. Type  

```
set(myworld,'RecordMode','scheduled');
```
- 5 Direct the Simulink 3D Animation software to record the animation as an `.avi` format file. Type  

```
set(f,'Record2D','on');
```
- 6 Select the start and stop times during which you want to record the animation. For example, enter 5 as the start time and 15 as the stop time.

```
set(myworld,'RecordInterval',[5 15]);
```

Ensure that the recording start time value is not earlier than the start time of the Simulink model; the recording operation cannot start in this instance. If the stop time exceeds the stop time of the Simulink model, or if it is an out of bounds value such as a negative number, the recording operation stops when the simulation stops. The recording can be slow.

- 7 Disable the Navigation Panel. The Navigation Panel appears at the bottom of the virtual scene view. You can turn off this panel for a cleaner view of the virtual scene. Type

```
set(f, 'NavPanel', 'none');
```

- 8 Ensure that the virtual reality figure window is the topmost window.
- 9 Run the Simulink model. From the **Simulation** menu, select **Mode > Normal**, then click **Simulation > Run**. Alternatively, if you are using the Simulink 3D Animation default viewer, you can run the Simulink model with one of the following from the viewer:
  - From the menu bar, select the **Simulation** menu **Start** option to start the simulation.
  - From the toolbar, click **Start/pause/continue simulation** to start the simulation.
  - From the keyboard, press **Ctrl+T** to start the simulation.

The simulation runs. The Simulink 3D Animation software starts recording when the simulation time reaches the specified start time. The software creates the file `vrplanets_anim_N.avi` in the current working folder when finished, where N is either 1 or more, depending on how many file iterations you have.

- 10 When you are done, stop the simulation. You can use one of the following from the viewer:
  - From the menu bar, select the **Simulation** menu **Stop** option to stop the simulation.
  - From the toolbar, click **Stop simulation** to stop the simulation.
  - From the keyboard, press **Ctrl+T** to stop the simulation.

- 11 If you want to enable the navigation panel again, type

```
set(f, 'NavPanel', 'halfbar');
```

- 12 Close and delete the objects if you do not want to continue using them.

## See Also

### Related Examples

- “Manual 3-D Recording with MATLAB” on page 4-16
- “Manual 2-D AVI Recording with MATLAB” on page 4-18
- “Scheduled 3-D Recording with MATLAB” on page 4-20
- “Record Animations for Unconnected Virtual Worlds” on page 4-24
- “Play Animation Files” on page 4-27

### More About

- “File Name Tokens” on page 4-14

## Record Animations for Unconnected Virtual Worlds

This topic describes how to record animation files programmatically for virtual worlds that are not associated with Simulink models (in other words, from the MATLAB interface). In this instance, you must specify the relationship between the events that change the virtual world state and the time in the animation file. This requirement is different from virtual worlds associated with Simulink models. Virtual worlds that are controlled completely from the MATLAB interface have no default, intuitive interpretation of time relation between MATLAB environment models and virtual scenes.

---

**Note** Many engineering time-dependent problems are modeled and solved in MATLAB. For those problems that have meaningful visual representation, you can create virtual reality models and animate their solutions. In addition, the offline animation time can represent any independent variable along which you can observe and visualize a model solution. Using offline animation files can bring the communication of such engineering problem resolutions to new levels. The Simulink 3D Animation example `vrheat` (heat transfer visualization) is an example of a time-dependent problem modeled and solved in MATLAB. Its modified version, `vrheat_anim`, shows the use of the programming technique described in this topic.

---

To record animation files for virtual worlds that are not associated with Simulink models, note the following guidelines, which require a strong understanding of the advanced Simulink 3D Animation software.

- Retrieve the `vrworld` object handle of the virtual scene that you want to record.
- To record 2-D animations,
  - 1 Retrieve the corresponding `vrfigure` object. For 2-D animations, the Simulink 3D Animation software records exactly what you see in the viewer window. Because 2-D animations record exactly what you see in the Simulink 3D Animation Viewer window, the properties that control 2-D file recording belong to `vrfigure` objects.
  - 2 Set the `Record2D` `vrfigure` property.
  - 3 To override default filenames for animation files, set the `vrfigure` `Record2DFileName` property.
- To create 3-D animation files,
  - 1 Retrieve the corresponding `vrworld` object.
  - 2 Set the `Record3D` `vrworld` property.
  - 3 To override default filenames for animation files, set the `vrworld` `Record3DFileName` property.
- Set the `RecordMode` `vrworld` object property to `manual` or `scheduled`. For optimal results, select `scheduled`.
- If you select `scheduled` for `RecordMode`, be sure to set the `vrworld` `RecordInterval` property to a desired time interval.
- To specify that the virtual world time source is an external one, set the `vrworld` property `TimeSource` to `external`. This setting ensures that the MATLAB software controls the virtual world scene time. Type

```
set(virtual_world, 'TimeSource', 'external')
```



- To specify time values at which you want to save animation frames, iteratively set the `vrworld` `Time` property. For a smoother animation, set the time at equal intervals, for example, every five seconds. Use a sequence like this one:

```
set(virtual_world,'Time',time_value)
```

For example, to set the `Time` property for `vrworld`, `w`, with values increasing by 10, enter

```
set(w,'Time',10);
set(w,'Time',20);
set(w,'Time',30);
set(w,'Time',40);
set(w,'Time',50);
set(w,'Time',60);
set(w,'Time',70);
set(w,'Time',80);
set(w,'Time',90);
set(w,'Time',100);
set(w,'Time',110);
set(w,'Time',120);
set(w,'Time',130);
set(w,'Time',140);
```

If you select a start time of 60 and a stop time of 120 (as described in “Scheduled 3-D Recording with MATLAB” on page 4-20), the Simulink 3D Animation software starts recording at 60 and stops at 120.

Because of the repetitive nature of the time interval setting, set the `Time` property in a loop from within a script or program.

- After you set the `vrworld` `Time` property, set the virtual scene object properties as necessary. Set these properties to values that correspond to the given time frame to achieve the desired animation effect.
- In each time frame, issue the `vrdrawnow` command for scene changes. This command renders and updates the scene.

The following code fragment contains a typical loop that iteratively sets the `Time` property, changes a virtual scene object property, and calls `vrdrawnow` to render the scene:

```
for time=StartTime:Step:StopTime
    % advance the time in the virtual scene
    set(myworld,'Time',time);
    % here we change node properties
    myworld.Car.translation = [ time*speed 0 0 ];
    % render the changed position
    vrdrawnow;
end
```

If you set the `Time` property at or outside the end boundary of `RecordInterval`, the Simulink 3D Animation software stops recording. You can then view the resulting animation file.

For a complete example of how to perform this kind of animation recording, refer to the Simulink 3D Animation `vrheat_anim` example.

## **See Also**

### **Related Examples**

- “Manual 3-D Recording with MATLAB” on page 4-16
- “Manual 2-D AVI Recording with MATLAB” on page 4-18
- “Scheduled 3-D Recording with MATLAB” on page 4-20
- “Scheduled 2-D AVI Recording with MATLAB” on page 4-22
- “Play Animation Files” on page 4-27

### **More About**

- “File Name Tokens” on page 4-14

## Play Animation Files

### In this section...

“Play Virtual World Animation Files” on page 4-27

“Play AVI Animation Files” on page 4-28

### Play Virtual World Animation Files

You can view virtual world animation files using one of these approaches:

- Open the 3D Animation Player from the MATLAB Toolstrip.

To open the 3D Animation Player from the MATLAB Toolstrip, in the **Apps** tab, in the **Simulation Graphics and Reporting** section, click **3D Animation Player**. Select or specify a virtual world 3D animation file.

- From the operating system, locate and double-click the VRML animation file

Double-click the virtual world 3D file. An HTML5-enabled web browser opens with the animation running. To view the resulting animation file, you must have an HTML5-enabled web browser installed on your system.

- Use `vrplay(filename)`, where `filename` is the name of your virtual world 3D file. This command opens the 3D Animation Player and your file. Using the player, you can control the playback of your file.

For example, play the animation file based on the `vr_octavia` example by running `vrplay('octavia_scene_anim.wrl')`.

`vrplay` works only with VRML animation files created using the Simulink 3D Animation recording functionality.

- In the **Current Folder** pane of MATLAB, double-click the animation file and from the context menu, select **Run**.
- At the MATLAB command line, use `vrview`.

A fourth option is to use the MATLAB command `vrview`. For example, enter:

```
w=vrview('vrplanets_anim_1.wrl');
set(w,'TimeSource','freerun');
```

The `vrview` command displays the default Simulink 3D Animation Viewer for the animation file. Setting the `TimeSource` property of the `set` method to `'freerun'` directs the viewer to advance its time independent of the MATLAB software.

To stop the animation, type:

```
set(w,'TimeSource','external');
```

To close the viewer and delete the world, get the handle of the `vrfigure` object and close it, as follows:

```
f=get(w,'Figures')
close(f);
delete(w);
```

Or, to close all `vrfigure` objects and delete the world, type

```
vrclose  
delete(w);
```

## Play AVI Animation Files

To view an AVI animation file, use *one* of these approaches:

- Double-click the AVI animation file. The program associated with `.avi` files in your system (for example, Windows Media® Player Media Player) opens for the `.avi` file. If your `.avi` file is not yet running, start it now from the application. The animation file runs.
- Use the MATLAB `VideoReader` function.

## See Also

### Functions

`vrplay` | `vrview`

# Build Virtual Reality Worlds

---

The Simulink 3D Animation product includes tools that you can use to edit and create virtual worlds. A basic understanding of these tools and how to use them helps you to get started quickly.

- “Choose a Virtual World Editor” on page 5-2
- “Build and Connect a Virtual World” on page 5-8
- “Use Sensors” on page 5-20
- “Detect Object Collisions” on page 5-23
- “Virtual World Data Types” on page 5-30
- “Simulink 3D Animation Textures” on page 5-34
- “Add Sound to a Virtual World” on page 5-35
- “Use CAD Models with the Simulink 3D Animation Product” on page 5-36
- “Import STL and Physical Modeling XML Files” on page 5-38
- “Import 3D Models from CAD Tools” on page 5-40
- “Import VRML Models from CATIA Software” on page 5-45
- “Modify the CAD Model Virtual World” on page 5-51
- “Import Visual Representations of Robot Models” on page 5-54
- “Link to Simulink and Simscape Multibody Models” on page 5-60

## Choose a Virtual World Editor

The primary way to create a virtual world is with a 3-D editing tool. These tools allow you to create complex virtual worlds without a deep understanding of the VRML or X3D language. These 3-D editing tools offer the power and versatility for creating many types of practical and technical models. For example, you can import 3-D objects from some CAD packages to make the authoring process easier and more efficient.

There is more than one way to create a virtual world defined with VRML or X3D code. You can use a virtual world editor to create a virtual world without knowing anything about the VRML or X3D language. Or you can use a text editor to write code directly.

The Simulink 3D Animation product includes the 3D World Editor for editing virtual worlds. You can use the 3D World Editor on all supported platforms for Simulink 3D Animation. The 3D World Editor is the default editor. For a comparison of editors, see “Editors for Virtual Worlds” on page 5-2 .

### Editors for Virtual Worlds

As you create a virtual world, you can use different editors for different phases of the process. Choose the editor that best meets your needs.

Some people prefer to create simple virtual worlds using MATLAB Editor or other text editor.

For Windows platforms, you can also use Ligos V-Realm Builder software to create and edit code. For information on using V-Realm Builder software with the Simulink 3D Animation product, see “Ligos V-Realm Builder” on page 5-4.

For details about specifying an editor, see “Set the Default Editor” on page 5-5.

For a description of the benefits and limitations of different types of editors, see the next section.

- “Text Editors” on page 5-2
- “General 3-D Editors” on page 5-3
- “Native VRML and X3D Editors” on page 5-3
- “3D World Editor” on page 5-3

### Text Editors

A virtual world 3D file uses a standard text format that you can read with any text editor. Reading the code in a text editor is useful for debugging and for directly changing the code, and for automated processing of the code. If you use the correct syntax, you can use the MATLAB Editor or any common text editor to create virtual worlds.

Consider using a text editor to work on a virtual world when you want to:

- Create a simple virtual world.
- Debug syntax and formatting errors in a virtual world 3D file. Corrupted files do not open in most 3-D tools.
- Learn about VRML and X3D syntax by using VRML and X3D syntax highlighting in the MATLAB Editor. For details, see “VRML and X3D Syntax Highlighting” on page 5-3.
- Perform global search editing operations across one or more virtual world 3D files.

- Combine several virtual world models. Combining models can involve temporary model inconsistencies, which most 3-D tools cannot handle.

### **VRML and X3D Syntax Highlighting**

You can display VRML, X3DV, and X3D syntax highlighting in the MATLAB Editor.

To set MATLAB Editor properties for VRML and syntax highlighting (for example, the color for highlighting comments or not using the smart indentation feature):

- 1 In MATLAB, select **Preferences > Editor/Debugger > Language**.
- 2 In the Editor/Debugger Language Preferences dialog box, set the **Language** field to VRML/X3DV.
- 3 Change the highlighting properties that you want.

For X3D syntax highlighting, set **Language** to XML/HTML.

### **General 3-D Editors**

General 3-D editors, such as 3D Studio, SolidWorks®, or Autodesk® Maya, do not use VRML or X3D as their native format. They export their formats to VRML or X3D. These tools have many features and are relatively easy to use.

General 3-D editing tools target specific types of work. For example, they can target visual art, animation, games, or technical applications. They offer different working environments depending on the application area for which they are designed. Some of these general 3-D editing tools are powerful, expensive, and complex to learn, but others are relatively inexpensive and can satisfy your specific needs.

The graphical user interfaces for many of the commercial general 3-D editors use features typical of the native VRML or X3D editing tools. For example, in addition to displaying 3-D scenes in various ways, they offer hierarchical tree styles, providing an overview of the model structure and a shortcut to nodes.

### **Native VRML and X3D Editors**

Native VRML and X3D editors use those languages as their native format. Native VRML editors support features that are unique to the VRML and X3D format, such as interpolators and sensors.

The Simulink 3D Animation software includes two native VRML and X3D editors:

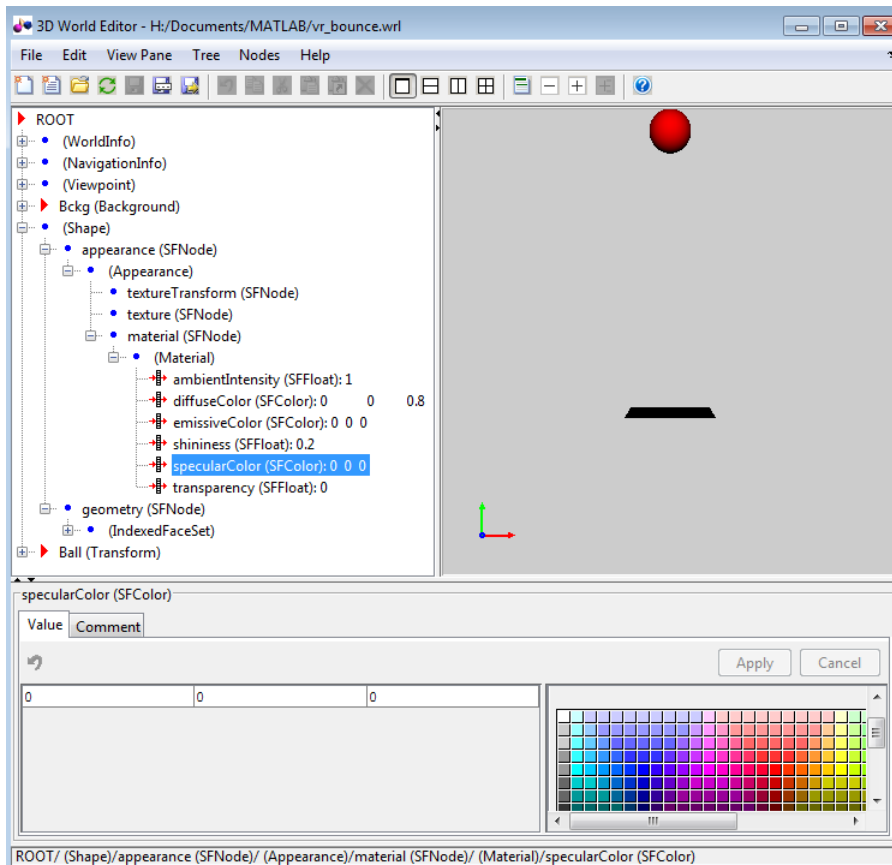
- “3D World Editor” on page 6-2, which works on all platforms supported for Simulink 3D Animation product
- The “Ligos V-Realm Builder” on page 5-4, which works on Windows platforms only

### **3D World Editor**

The 3D World Editor is installed as part of the Simulink 3D Animation installation. It is the default virtual world editor.

The 3D World Editor is a native VRML and X3D authoring tool that provides an interface to the syntax of those languages. The editor supports VRML97 types and language elements. For details on limitations, see “VRML Support” on page 1-11 and “X3D Support” on page 1-9.

The 3D World Editor interface provides three panes.



- **Tree structure** pane — View the hierarchy for the virtual world that you are editing. The 3D World Editor lists the nodes and their properties according to their respective node types. You can change the nesting levels of certain nodes to modify the virtual world. In the tree viewer, give the nodes unique names.
- **Virtual world display** pane — Observe the virtual world as you create it. The 3D World Editor renders inlined objects (grouped objects). It uses the same renderer as the Simulink 3D Animation viewer. Using the same renderer for the editor and the viewer provides consistent navigation and display throughout the development process.
- **Object property edit** pane — Change values for node items.

For details, see “Build and Connect a Virtual World” on page 5-8 and “3D World Editor” on page 6-2.

### Ligos V-Realm Builder

The Ligos V-Realm Builder interface is available only for Windows operating systems.

The V-Realm Builder application is a flexible, graphically oriented tool for 3-D editing. It provides similar functionality as the 3D World Editor.

The V-Realm Builder offers these features that the 3D World Editor does not:

- Manipulators — for dragging objects in the 3-D world
- Keyframe animation — animation involving interpolated linear movements



Compared to the 3D World Editor, the V-Realm Editor interface:

- Supports only VRML, not X3D
- Provides dialog boxes for editing properties, which can be less streamlined than the 3D World Editor **object properties edit** pane
- Does not always render virtual worlds the same way as the viewer
- Does not support rendering inlined objects

For more information about the V-Realm Editor, see “V-Realm Builder Help” on page 2-15.

## Set the Default Editor

- “Use Preferences to Set the Default Editor” on page 5-5
- “Use MATLAB Commands to Set the Default Editor” on page 5-6

---

**Tip** The Simulink 3D Animation product includes the 3D World Editor for editing virtual worlds. You can use the 3D World Editor on all supported platforms for Simulink 3D Animation. The 3D World Editor is the default editor. For a comparison of editors, see “Choose a Virtual World Editor” on page 5-2.

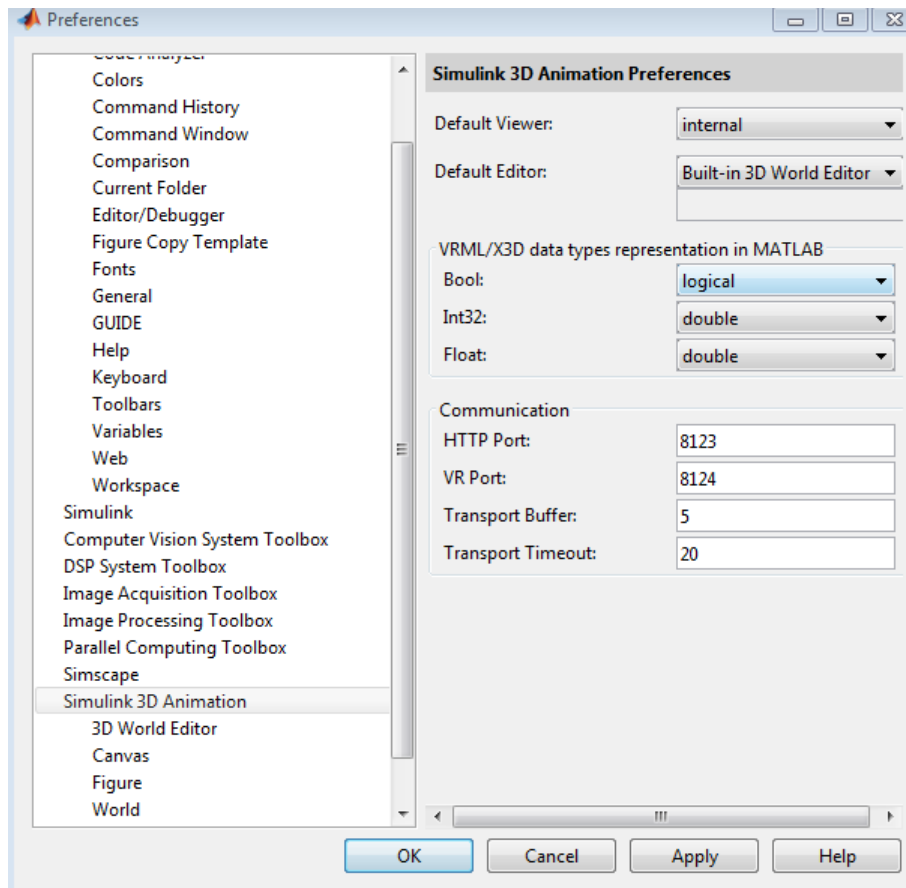
---

You can change your environment to use another editor. You can use the MATLAB Preferences menu or the MATLAB command line.

### Use Preferences to Set the Default Editor

To determine which virtual world editor is set up as the editor in your environment:

- 1 From the MATLAB Toolstrip, in the **Home** tab, in the **Environment** section, select **Preferences > Simulink 3D Animation**.
- 2 In the Simulink 3D Animation Preferences dialog box, examine the **3D World Editor** preference.



You can use the **3D World Editor** preference to select another editor: the V-Realm Builder, the MATLAB editor, or a third-party virtual world editor or text editor. To use a third-party editor, select the Custom option. In the text box that appears, enter the path to the editor.

### Use MATLAB Commands to Set the Default Editor

- 1 To determine which editor is installed, at the MATLAB command prompt, type:
 

```
vrgetpref('Editor')
```
- 2 The default is the 3D World Editor (\*BUILTIN). To change the editor, use the `vrsetpref` command, specifying the editor that you want. For example, to change to the V-Realm editor, type:
 

```
vrsetpref('Editor','*VREALM')
```
- 3 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.

---

## See Also

### Functions

vredit | vrgetpref | vrsetpref

### Related Examples

- “Set the Default Editor” on page 5-5
- “3D World Editor” on page 6-2
- “Edit a Virtual World” on page 6-11
- “Workflow for Building and Using Virtual Worlds” on page 1-4
- “Install V-Realm Editor” on page 2-14

## Build and Connect a Virtual World

In this section...
“Introduction” on page 5-8
“Define the Problem” on page 5-8
“Add a Simulink 3D Animation Block” on page 5-9
“Open a New Virtual World” on page 5-10
“Add Nodes” on page 5-11
“Link to a Simulink Model” on page 5-17

### Introduction

This example shows you how to create a simple virtual world using the 3D World Editor. The example does not show everything that you can do with the editor. However, the example does show you how to perform some basic tasks to get started.

This example assumes that you have set your default editor to be the 3D World Editor. For details, see “Set the Default Editor” on page 5-5.

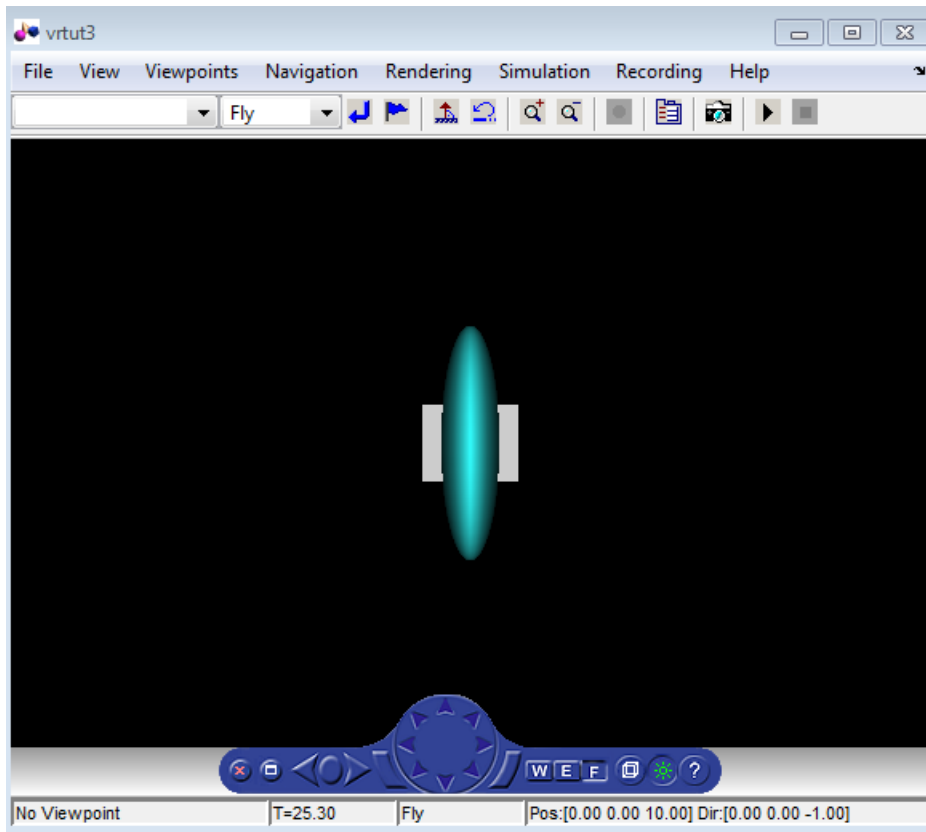
This example describes the steps to build a simplified version of the virtual world that you see if you enter this command in the MATLAB command window:

```
edit(vrworld('vrdeform.wrl'))
```

### Define the Problem

Suppose that you want to simulate and visualize in virtual reality the deformation of a sphere. In your virtual world, you want to have two boxes representing rigid plates (B1, B2) and an elastic sphere (S) between them. All three of the objects are center-aligned along the x-axis. The boxes B1 and B2 move toward S with identical velocities, but they move in opposite directions. As they reach the sphere S, they start to deform it by reducing its x dimension and stretching both its y and z dimensions.

Here is how this virtual world looks:



The following table lists the positions and dimensions of the objects that you create for this example.

Object	Center Position	Dimensions
B1	[3 0 0]	[0.3 1 1]
B2	[-3 0 0]	[0.3 1 1]
S	[0 0 0]	$r = 0.9$

The Simulink 3D Animation product includes the tutorial model `vrtut3`. This simplified model simulates the deformation of an elastic sphere. After collision with the rigid blocks, the sphere's  $x$  dimension decreases by a factor from 1 to 0.4. Also, the  $y$  and  $z$  dimensions expand to keep the volume of the deformed sphere-ellipsoid constant. Additional blocks in the model supply the correctly sized vectors to the Simulink 3D Animation block. The simulation stops when the sphere is deformed to 0.4 times its original size in the  $x$  direction.

Your first task is to open a Simulink model and add a Simulink 3D Animation block to your model.

## Add a Simulink 3D Animation Block

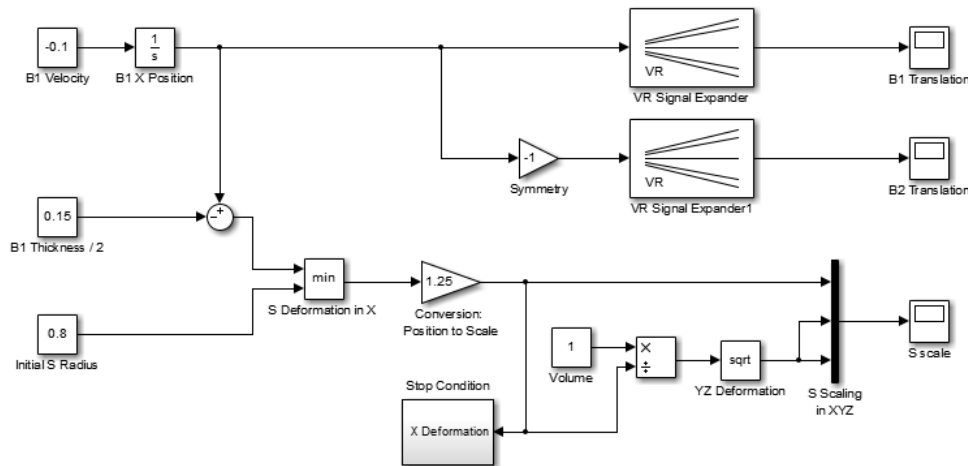
This procedure uses the Simulink model `vrtut3` to show how to add a Simulink 3D Animation block to your model. The model generates the values for the position of B1, the position of B2, and the dimensions of S (as described in “Define the Problem” on page 5-8).

- 1 Open the Tutorial #3. example.

- a At the top of the page that opens, select **Open Model**.
- b Save the `vrtut3` file to your MATLAB working folder.
- 2 In MATLAB, change the current folder to your MATLAB working folder.
- 3 In the MATLAB Command Window, type:

```
vrtut3
```

A Simulink window opens with a model that contains Simulink 3D Animation VR Signal Expander blocks, but no VR Sink block to write data from the model to Simulink 3D Animation. Instead, this model uses Scope blocks to monitor temporarily the relevant signals.



- 4 From the MATLAB Command Window, type

```
vrlib
```

The Simulink 3D Animation library opens.

- 5 From the Library window, drag and drop the VR Sink block to the Simulink diagram. You can then close the Library Browser window.

Your next task is to create a virtual world that you will associate with the VR Sink block. See “Open a New Virtual World” on page 5-10.

## Open a New Virtual World

You must create a virtual world to connect to a Simulink model for visualizing signals.

This procedure opens a new virtual world, in which you add nodes for visualizing the signals of the model `vrtut3`. The connection between the virtual world and the Simulink model requires that the model includes a VR Sink block, as described in “Add a Simulink 3D Animation Block” on page 5-9.

- 1 Start the 3D World Editor with an empty virtual world. From the MATLAB Toolstrip, in the **Apps** tab, in the **Simulation Graphics and Reporting** section, click **3D World Editor**.

The 3D World Editor displays:

- In the left pane, a virtual scene tree with only a `ROOT` node

- In the right pane, an empty virtual world
  - In the bottom pane, an empty pane for editing objects
- 2 You can save the virtual world at any point. Save the virtual world as `virtut3.wrl` in the same working folder where your `virtut3` file resides. Do not close the 3D World Editor.

Your next two tasks create a virtual world to use with the `virtut3` model:

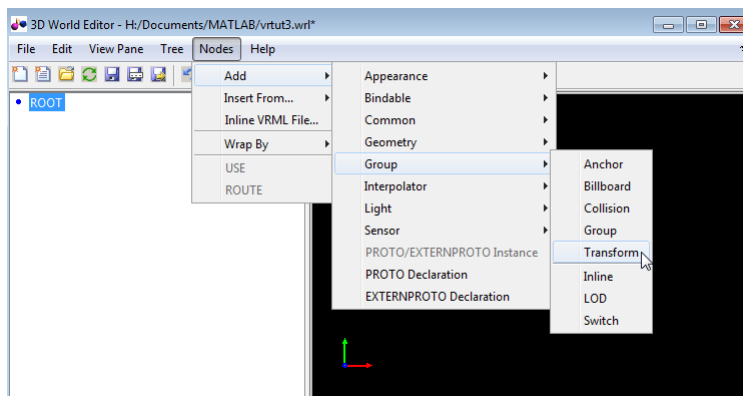
- “Add Nodes” on page 5-11
- “Build and Connect a Virtual World” on page 5-8

## Add Nodes

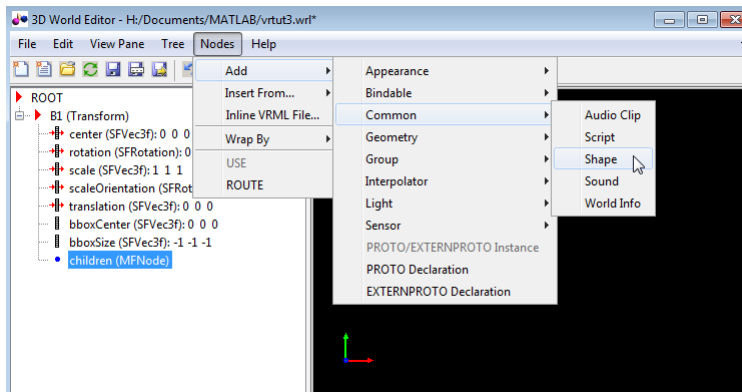
### Create Boxes

Defining virtual world objects involves defining a hierarchy of nodes. This example shows how to define Transform nodes under the ROOT node, with each Transform node including a hierarchy of children, Shape, Appearance, Geometry, and specific shape (in this case, a Box) nodes.

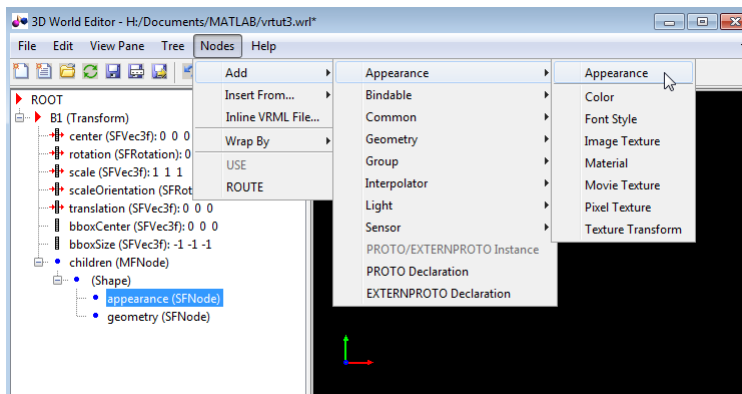
- 1 In the tree in the left pane, click ROOT (the topmost item).
- 2 Add a Transform node, using the following sequence of menu selections.



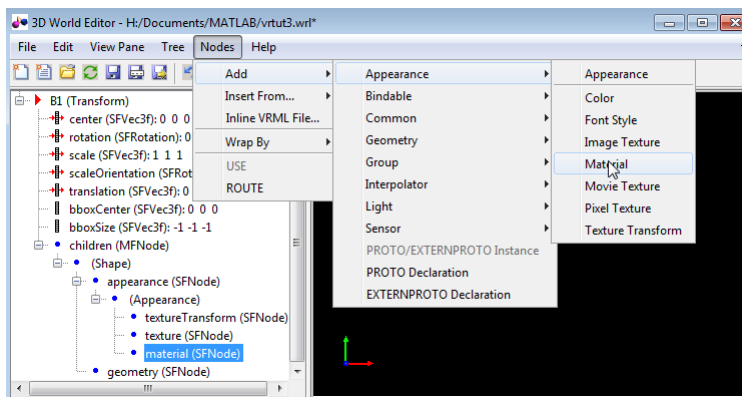
- 3 This Transform node is for the B1 box. To name the Transform node:
  - a Right-click the Transform node.
  - b Select the **Edit Name** menu item.
  - c In the edit box to the left of the Transform node, type B1.
- 4 Add a Shape node:
  - a Expand the B1 Transform node.
  - b Select the children node.
  - c Add a Shape node, using the following sequence of menu selections:



- 5 Add an Appearance node for the Shape node:
  - a Under the Shape node, select the appearance (SFNode) node.
  - b Add an Appearance node, using the following sequence of menu selections.



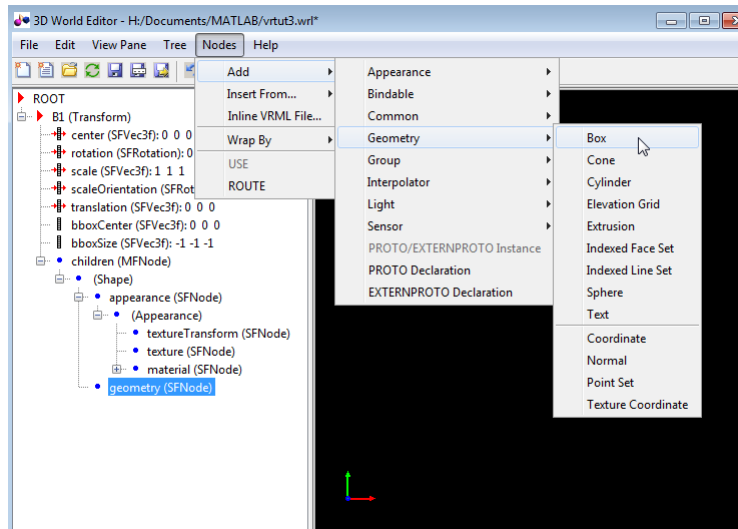
- 6 Add a Material node to the Appearance node:
  - a Expand the (Appearance) node and select the material (SFNode) node.
  - b Add a Material node, using the following sequence of menu selections.



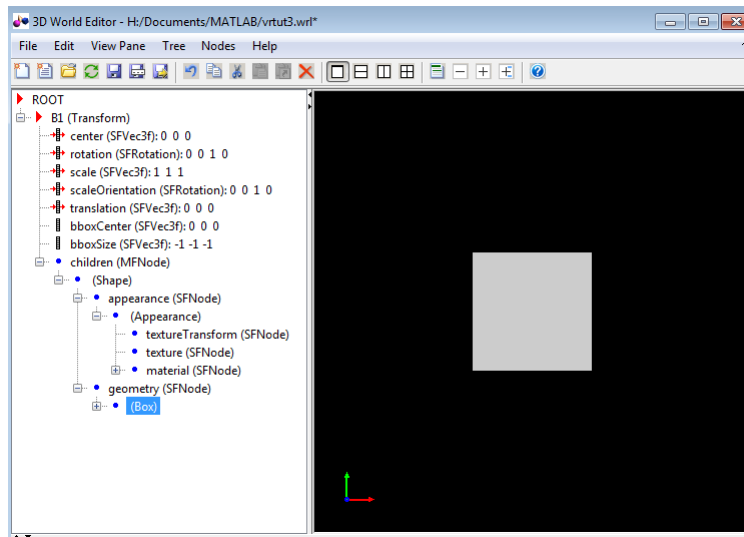
- 7 Add a Box node to the geometry node:
  - a Select the geometry (SFNode) node of the (Shape) node.



- b** Add a Box node, using the following sequence of menu selections.

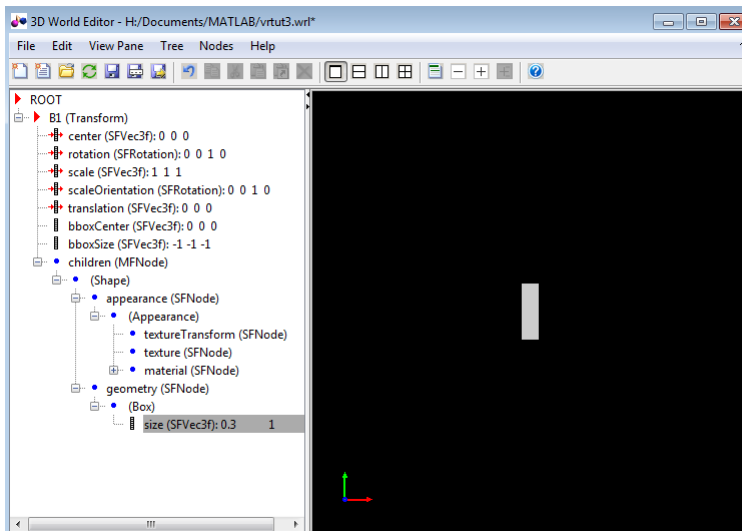


With all the nodes expanded, the 3D World Editor now displays a box in the **virtual world display** pane.

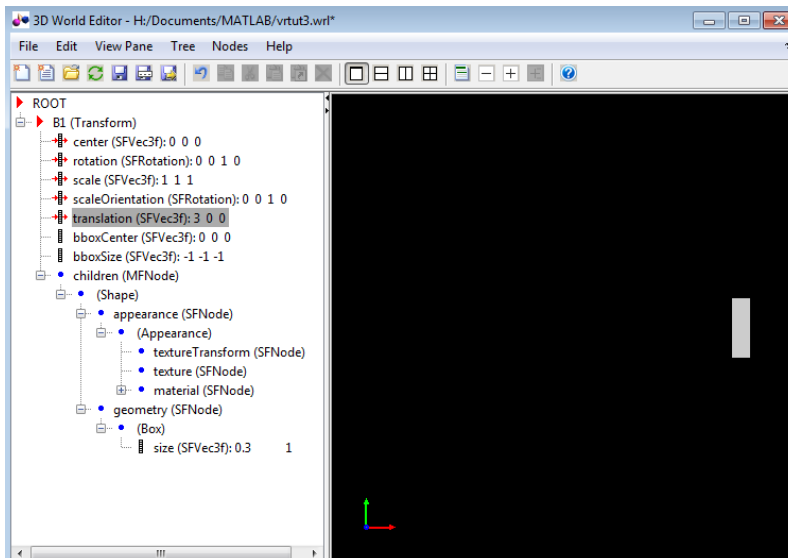


- 8** Make the box smaller by editing its size property:
- Select the size property of the **Box** node.
  - In the **object properties edit** pane at the bottom of the 3D World Editor, enter  $0.3$  in the first column, and  $1$  in the second and third columns.
  - Click **Apply**.

The box becomes smaller.

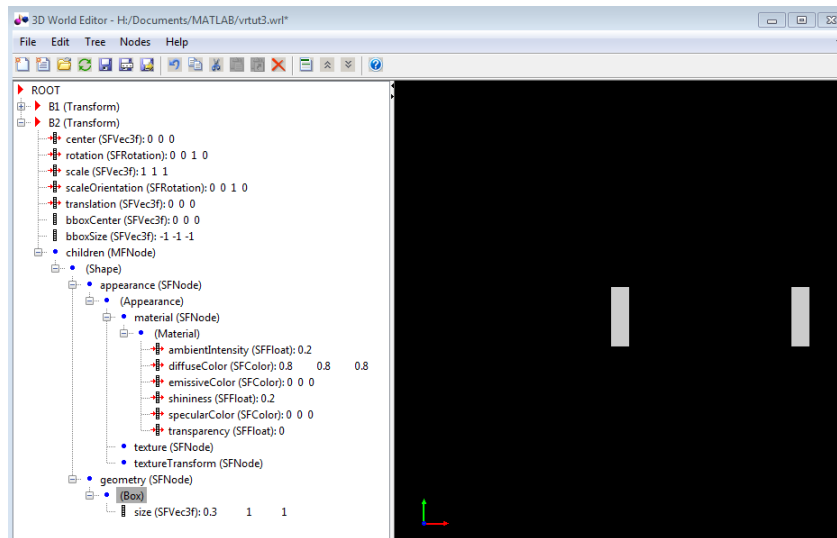


- 9 Move the box to the right by changing the translation(SFVec3f) property of the B1 (Transform) node. In the **object properties edit** pane, set the first column to 3 and leave the second and third columns set to 0.

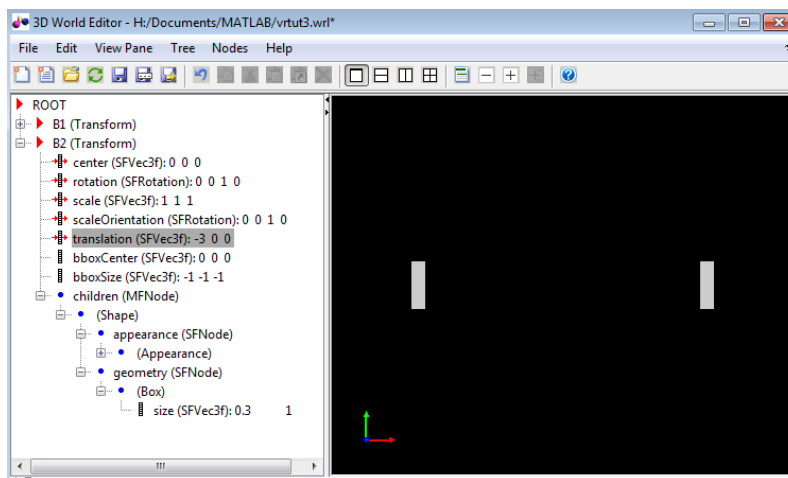


- 10 Add a second box that is similar to the first box.
  - a Under the ROOT node, add a Transform node (see step 2) and name it B2 (see step 3).
  - b Copy the Shape node. Under the B1 Transform node, right-click the Shape node in the B1 Transform node and select the **Copy** menu item.
  - c Paste the copied Shape node into the B2 Transform node. Under the B2 Transform node), right-click the children node and select the **Paste Node > Paste** menu item.

With the B1 node collapsed and the B2 node expanded, the 3D World Editor looks like the following graphic.



- 11 Move the box that you created to the left by changing the translation property of the B2 (Transform) node. In the **object properties edit** pane, set the first column to -3 and leave the second and third columns set to 0.

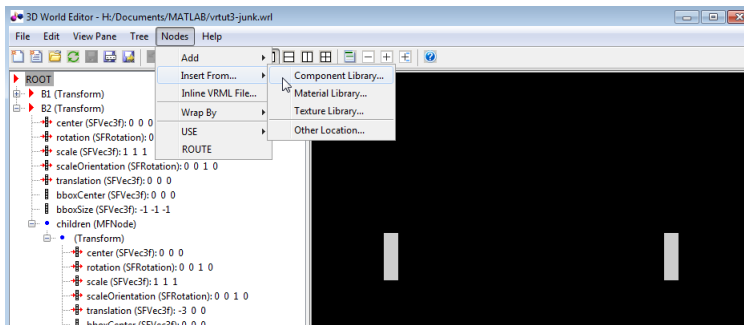


## Create a Sphere

Your next task is to add a sphere between the two boxes. This section assumes that you have completed the tasks described in “Add Nodes” on page 5-11.

- 1 To make it easier to focus the **tree structure** pane on the nodes that you want to add, collapse the B1 (Transform) and B2 (Transform) nodes.
- 2 In the tree in the left pane, click ROOT node.
- 3 Add a Sphere node. The 3D World Editor includes a library of objects for building a virtual world, including a Sphere object.

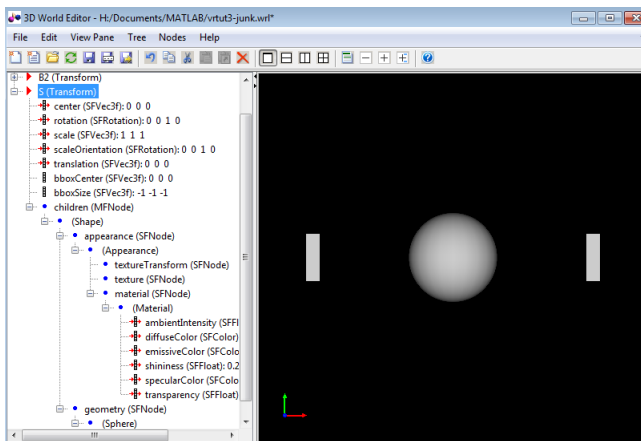
Add a Sphere library object using the following sequence of menu selections.



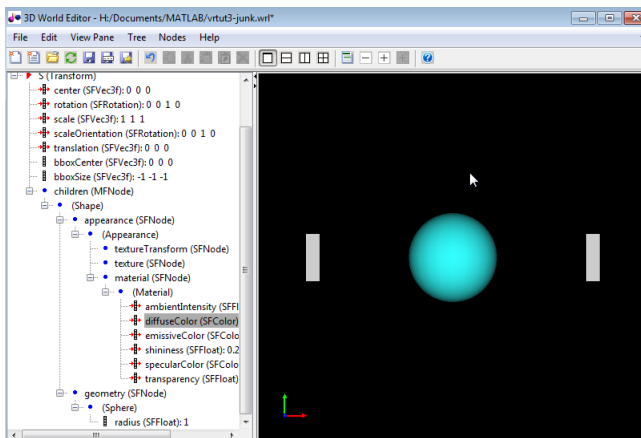
From the list of Component Library folders, select the Shapes folder, and then select the Sphere.wrl file.

- 4 Select the Transform node and name it S.

With the S Transform node fully expanded and the other Transform nodes collapsed, the 3D World Editor looks like the following graphic.



- 5 To make the sphere blue, under the Material node, select the diffuseColor property. In the **object properties edit** pane, change the first column value to 0.2, the second column to 1, and the third column to 1.



- 6 Save the virtual world file.

Your next task is to connect the model outputs to the Simulink 3D Animation block in your Simulink model. See “Link to a Simulink Model” on page 5-17.

## Link to a Simulink Model

After you create a virtual world and a Simulink model with a VR Sink block, define the associations between the model signals and the virtual world.

---

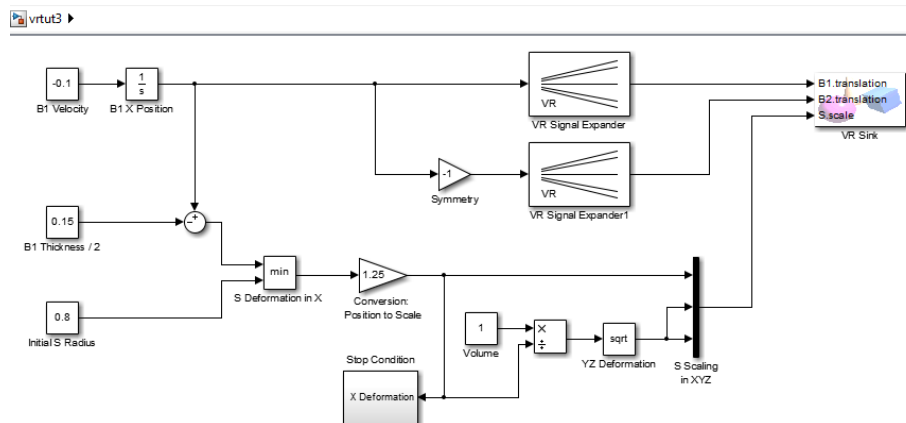
**Note** This procedure uses the model `vrtut3` as an example. It assumes that you have opened the model and that you have added a VR Sink block, and that you have created a virtual world called `vrtut3.wrl`. See the tutorial starting with “Add a Simulink 3D Animation Block” on page 5-9.

---

- 1 Open the VR Sink Block Parameters dialog box. In the Simulink Editor, double-click the VR Sink block.
- 2 Next to the **Source file** edit box, click **Browse**.
- 3 Select `vrtut3.wrl`, and then click **Open**.
- 4 In the **Output** pane, select **Open Viewer automatically**. This check box specifies that a viewer for the virtual world starts when you run the model.
- 5 For the **Description** parameter, type `vrtut3`.
- 6 In the VR Sink dialog box, click **Apply**.
- 7 In the **tree structure** pane, select the **B1 translation**, **B2 translation**, and **S scale** check boxes as the nodes that you want to connect to your model signals. Click **OK**.

The VR Sink block appears with corresponding inputs.

- 8 Delete the three Scope blocks and their associated input signal lines.
- 9 Connect the input lines from the two VR Signal Expander blocks and S Scaling in XYZ block to the appropriate ports in the VR Sink block.

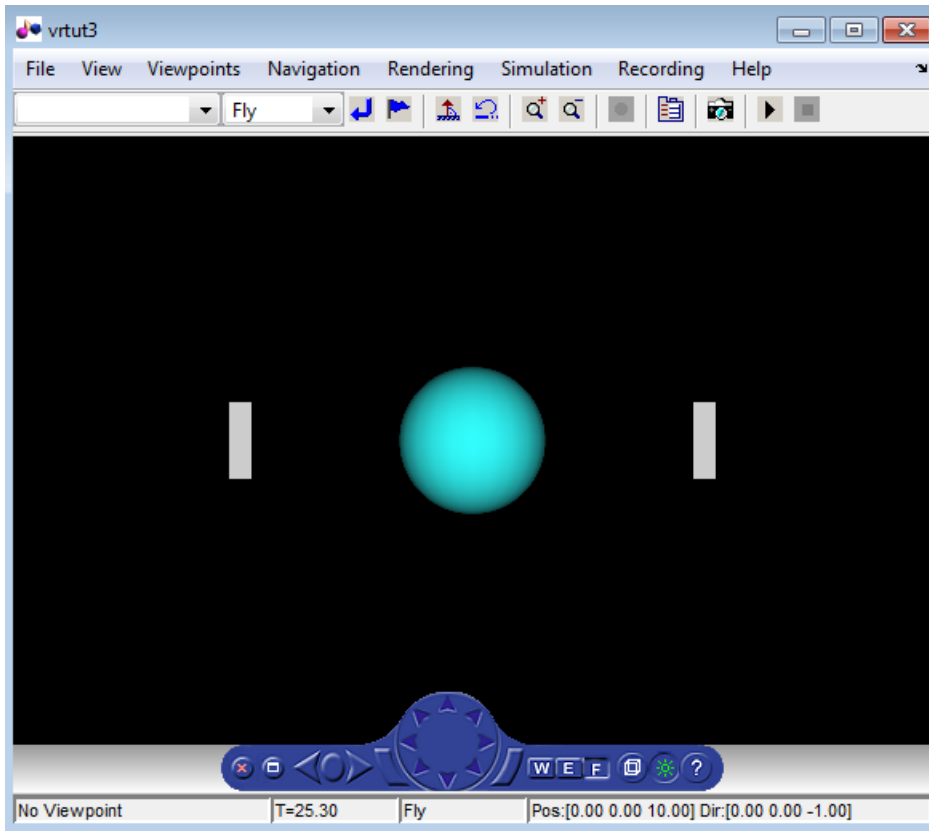


- 10 Double-click the VR Sink block.

The viewer appears.

- 11 In the viewer, select the **Simulation > Block Parameters** option. Your default viewer opens and displays the virtual world. For more information on changing your default viewer, see “Set the Default Viewer” on page 2-2.

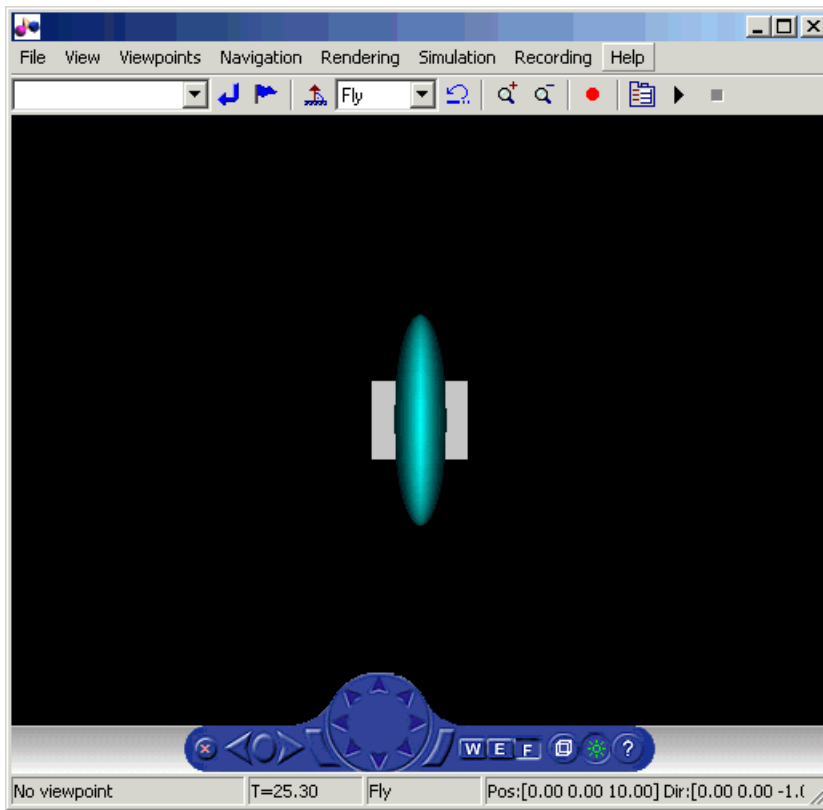
- 12 In the VR Sink Block Parameters dialog box, click the **View** button.



- 13 In the Simulink Editor, select **Simulation > Run**.

In your default viewer, you see a 3-D animation of the scene. Using the viewer controls, you can observe the action from various points.

When the width of the sphere is reduced to 0.4 of its original size, the simulation stops running.



This example shows you how to create and use a simple virtual reality model. Using the same method, you can create more complex models for solving the particular problems that you face.

## See Also

### Functions

`vredit` | `vrgetpref` | `vrjoystick` | `vrlib` | `vrsetpref` | `vrspacemouse`

### Blocks

VR Sink | VR Source

## Related Examples

- “Virtual Reality World and Dynamic System Examples” on page 1-16
- “Workflow for Building and Using Virtual Worlds” on page 1-4
- “Edit a Virtual World” on page 6-11
- “Connect Virtual Worlds and Models” on page 3-2
- “Display Virtual World and Start Simulation” on page 3-10

## Use Sensors

To interact with the simulation of a model based on user actions or events occurring in the virtual world, you can use virtual reality sensors. To move graphics objects around in a virtual world during simulation or to change their appearance, based on user actions or events, you can:

- 1 Define a sensor node, which generates events and output values depending on time, navigation, and actions and distance changes in the scene. For example, a `TouchSensor` node tracks the location and state of the pointing device. The sensor detects when you point at the geometry contained by the `TouchSensor` node parent group. See “Add Sensors to Virtual Worlds” on page 5-20.
- 2 Add a VR Source block and select the sensor properties to read. See “Read Sensor Values Using MATLAB” on page 4-7.

---

**Note** Instead of using a VR Source block to read sensor values, you can write an S-function or use a MATLAB Function block.

---

If you are working in MATLAB, you can read sensor values using `vrnode` object properties.

---

- 3 Read sensor values using a VR Source block, whose outputs can be used to drive simulation behavior.

## Add Sensors to Virtual Worlds

You can set up an interface in a Simulink block diagram to sensors in a virtual reality scene. You can also input signals programmatically from the virtual world into a simulation model.

Virtual reality scenes can contain sensors, which are nodes that generate events and output values depending on time, navigation, and actions and distance changes in the scene. These nodes add interactivity to the virtual world. The virtual world sensors resemble real world sensors, such as ultrasonic, lidar, and touch sensors. You can use Simulink 3D Animation functions to read sensor field values into simulation models and control simulation based on the user interaction with the virtual scene.

Ways you can use sensors include:

- Use sensor data from a virtual world to control a simulation.
- 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.

You can use collision detection to accurately model physical constraints of objects in the real world, where generally two objects cannot be in the same place at the same time. You can use collision detection node outputs to:

- Change the state of other virtual world nodes.
- Apply MATLAB algorithms to collision data.
- Drive Simulink models.



For example, you can use geometric sensors for robotics modeling. For more information, see “Detect Object Collisions” on page 5-23.

You can define these sensors in a scene.

Sensors	Description
CylinderSensor	Maps pointer motion (for example, a mouse) into a rotation on an invisible cylinder that is aligned with the y-axis of the local coordinate system.
PlaneSensor	Maps pointing device motion into two-dimensional translation in a plane parallel to the z=0 plane of the local coordinate system.
ProximitySensor	Generates events when the viewer enters, exits, and moves within a region in space (defined by a box)
SphereSensor	Maps pointing device motion into spherical rotation about the origin of the local coordinate system
TimeSensor	Generates events as time passes
TouchSensor	Tracks the location and state of the pointing device and detects when you point at geometry contained by the TouchSensor node parent group.
VisibilitySensor	Detects visibility changes of a rectangular box as you navigate the world.
PointPickSensor	Uses point clouds to detect which of the points are inside colliding geometries
LinePickSensor	Uses ray fans or other sets of lines that detect the distance to the colliding geometries
PrimitivePickSensor	Primitive geometries (such as a cone, sphere, or box) that detect colliding geometries

## Read Sensor Values

You can read values from sensor nodes in a virtual world by using:

- VR Source blocks on page 5-21
- S-Functions or MATLAB Function blocks on page 5-22
- `vrnode`

### Read Sensor Values Using VR Source Blocks

You can use the VR Source block for 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 block reads the values from the virtual world fields specified in the Block Parameters dialog box and inputs them to a model.

For example, you can specify setpoints (the desired positions) in the virtual world, so that a 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 that use a VR Source block, see Virtual Control Panel and Magnetic Levitation Model.

## Read Sensor Values Using S-Functions

To use the setpoint value in a Simulink model, you can write an S-function or a MATLAB Function block that reads the sensor output periodically. This example uses an S-function.

- 1 Right-click the VR Sensor Reader block of Magnetic Levitation Model (`vrmaglev`) model and select **Mask > Look Under Mask**.

The `vrmaglev`/VR Sensor Reader model displays. This model contains the `vrextin` block, which is an S-function block. The `vrextin` S-function synchronizes the sensor field in the `setup` method and periodically reads its value in the `mdlUpdate` method.

- 2 Examine the S-function parameters. Right-click `vrextin` and select **S-Function Parameters**.

The parameters defined in the mask supply the sample time, virtual world, and the node and field to read.

### Notes About the `vrextin` S-Function

- Instead of setting its own block outputs, the `vrextin` S-function sets the value of the adjacent Constant block `value_holder`. This setting makes the VR Sensor Reader block compatible with Simulink Coder code generation so that the model can run on Simulink Coder targets.
- The signal loop between user action (moving the ball to a desired position using a mouse) closes through the associated Simulink model `vrmaglev`. Grabbing the ball and moving it to a new position works only when the model is running and when the model sets the blue selection method switch to the virtual reality sensor signal path. To experience the behavior of the `PlaneSensor` using the virtual scene only, save the `maglev.wrl` file under a new name. Remove the comment symbol (`#`) to enable the last line of this file. These actions activate direct routing of sensor output to a ball translation. Then you can experiment with the newly created scene instead of the original `maglev.wrl` world.

```
ROUTE Grab_Sensor.translation_changed TO Ball.translation
```

- You can use this approach to input information from all node fields of the type `exposedField` or `eventOut`, not only a `Sensor eventOut` field. See [VRML Data Class Types](#) on page 5-32 for more information about virtual world data class types.
- For fields of class `exposedField`, you can use an alternate name using the field name with the suffix, `_changed`. For example, `translation` and `translation_changed` are alternate names for requesting the translation field value of the `Grab_Sensor` node.

## See Also

### Blocks

VR Sink | VR Source

## Related Examples

- “Connect Virtual Worlds and Models” on page 3-2
- “Detect Object Collisions” on page 5-23
- “Input Virtual World Data to a Model” on page 3-6

## Detect Object Collisions

You can use collision detection to model physical constraints of objects in the real world accurately, to avoid having two objects in the same place at the same time. You can use collision detection node outputs to:

- Change the state of other virtual world nodes.
- Apply MATLAB algorithms to collision data.
- Drive Simulink models.

For example, you can use geometric sensors for robotics modeling. For examples of using collision detection, see `vrcollisions` and `vrmaze`.

### Set Up Collision Detection

To set up collision detection, define collision (pick) sensors that detect when they collide with targeted surrounding scene objects. The virtual world sensors resemble real-world sensors, such as ultrasonic, lidar, and touch sensors. The Simulink 3D Animation sensors are based on X3D sensors (also supported for VRML), as described in the X3D picking component specification. For descriptions of pick sensor output properties that you can access with VR Source and VR Sink blocks, see “Use Collision Detection Data in Models” on page 5-25.

- `PointPickSensor` — Point clouds that detect which of the points are inside colliding geometries
- `LinePickSensor` — Ray fans or other sets of lines that detect the distance to the colliding geometries
- `PrimitivePickSensor` — Primitive geometries (such as a cone, sphere, or box) that detect colliding geometries

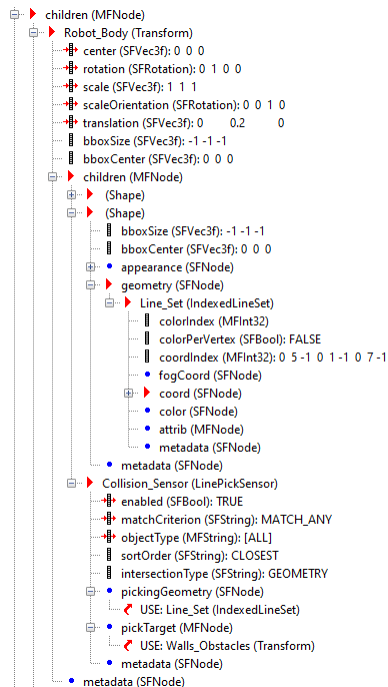
To add a collision detection sensor, use these general steps. For an example that reflects this workflow, see `vrcollisions`.

- 1 In the 3D World Editor tree structure pane, select the children node of the Transform node to which you want to add a pick sensor.
- 2 To create the picking geometry to use with the sensor, add a geometry node. Select **Nodes > Add > Geometry** and select a geometry appropriate to the type of pick sensor (for example, **Point Set**).
- 3 Add a pick sensor node by selecting **Nodes > Add > Pick Sensor Node**.
- 4 In the sensor node, right-click the `pickingGeometry` property and select **USE**. Specify the geometry node that you created for the sensor.
- 5 Also in the sensor node, right-click the `pickingTarget` property and select **USE**. Specify the target objects for which you want the sensor to detect collisions.

Instead of specifying the picking geometry with a **USE**, you can define the picking geometry directly. However, the directly defined geometry is invisible.

- 6 Optionally, change default property values or specify other values for sensor properties. For information about the `intersectionType`, see “Sensor Collisions with Multiple Object Pick Targets” on page 5-24. For descriptions of output properties that you can access with a VR Source block, see “Use Collision Detection Data in Models” on page 5-25.

Here is an example of the key nodes for defining a collision detection sensor for the robot in the `vrcollisions` virtual world:



- The `Robot_Body` node has the `Line_Set` node as one of its children. The `Line_Set` node defines the picking geometry for the sensor.
- The `Collision_Sensor` defines the collision detection sensor for the robot. The sensor node `pickingGeometry` specifies to use the `Line_Set` node as the picking geometry and the `Walls_Obstacles` node as the targets for collision detection.

### Sensor Collisions with Multiple Object Pick Targets

To control how a pick sensor behaves when it collides with a pick target geometry that consists of multiple objects, use the `intersectionType` property. Possible values are:

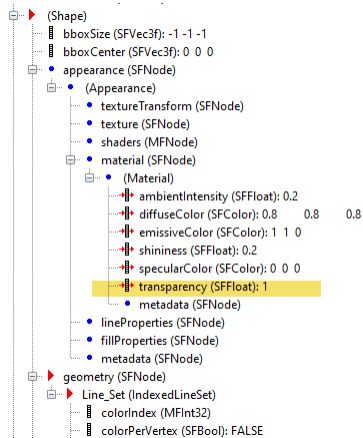
- **GEOMETRY** - The sensor collides with union of individual bounding boxes of all objects defined in the `pickTarget` field. In general, this setting produces more exact results.
- **BOUNDS** - (Default) The sensor collides with one large bounding box construed around all objects defined in the `pickTarget` field.

In the `vrcollisions` example, the `LinePickSensor` has the `intersectionType` field set to **GEOMETRY**. This setting means that the sensor that is inside the colliding geometry (consisting of the room walls), does not collide with the union of walls. A collision takes place only if sensor rays touch any of the walls. If the `intersectionType` is set to **BOUNDS**, collision detection works only for a sensor that approaches the room from the outside. The whole room is wrapped into one large bounding box that interacts with the sensor.

### Make Picking Geometry Transparent

You can make the picking geometry used for a pick sensor invisible in the virtual world. For the picking geometry, in its `Material` node, set the `Transparency` property to 1. For example, in the

vrCollisions virtual world, for the Collision\_Sensor picking geometry node (Line\_Set), in the Materials node, change the Transparency property to 1.



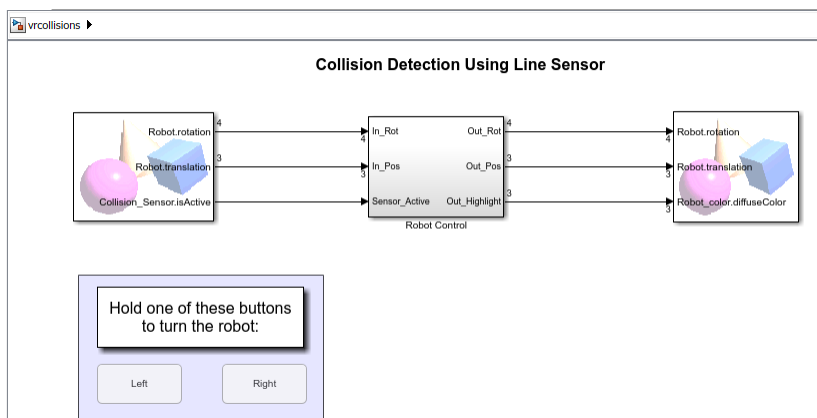
## Avoid Impending Collisions

To avoid an impending collision (before the collision actually occurs), you can use the `pickedRange` output property for a `LinePickSensor`. As part of the line set picking geometry, define one or more long lines that reflect your desired amount of advance notice of an impending collision. You can make those lines transparent. Then create logic based on the `pickedRange` value.

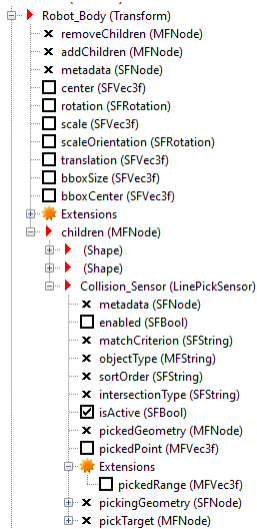
## Use Collision Detection Data in Models

The `isActive` output property of a sensor becomes TRUE when a collision occurs. To associate a model with the virtual reality scene, you can use a VR Source block to read the sensor `isActive` property and the current position of the object for which the sensor is defined. You can use a VR Sink block to define the behavior of the virtual world object, such as its position, rotation, or color.

For example, the VR Source block in the top left of the `vrCollisions` Simulink model gets data from the associated virtual world.



In the model, select the VR Source block, and then in the Simulink 3D Animation Viewer, select **Simulation > Block parameters**. This image shows some of the key selected properties.



For the LinePickSensor PointPickSensor, and PrimitivePickSensor, you can select these output properties for a VR Source block:

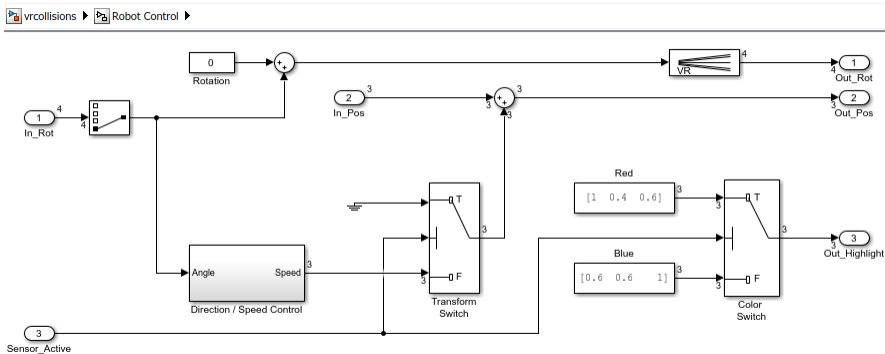
- enabled - Enables node operation.

**Note** The enabled property is the only property that you can select with a VR Sink block.

- isActive - Indicates when the intersecting object is picked by the picking geometry.
- pickedPoint - Displays the points on the surface of the underlying PickGeometry that are picked (in local coordinate system).
- pickedRange - Indicates range readings from the picking. For details, see “Avoid Impending Collisions” on page 5-25.

For a PointPickSensor, you can select the enabled, isActive, and pickedPoint outputs. For the PrimitivePickSensor, you can select the enabled and isActive outputs.

The Robot Control subsystem block includes the logic to change the color and position of the robot.



Based on the Robot Control subsystem output, the VR Sink block updates the virtual world to reflect the color and position of the robot.

---

**Tip** Consider adjusting the sample time for blocks for additional precision for collision detection.

---

## Use Collision Detection in MATLAB

You can use collision detection in a virtual world that you define in MATLAB. This example is based on the `vrcollisions` virtual world. It does not use a Simulink model.

- 1 Open and view the `vrcollisions` virtual world.

```
w = vrworld('vrcollisions');
open(w);
fig = view(w, '-internal');
```

- 2 Get the collision sensor and robot nodes of the virtual world.

```
col = vrnnode(w, 'Collision_Sensor')
rob = vrnnode(w, 'Robot')
color = vrnnode(w, 'Robot_color')
```

- 3 Move the robot, based on collision detection (when the `isActive` property is `TRUE`). At the default position, no collision is detected.

```
col.isActive

for ii = 1:30

    % Move robot
    rob.translation = rob.translation + [0.05 0 0];
    vrdrawnow

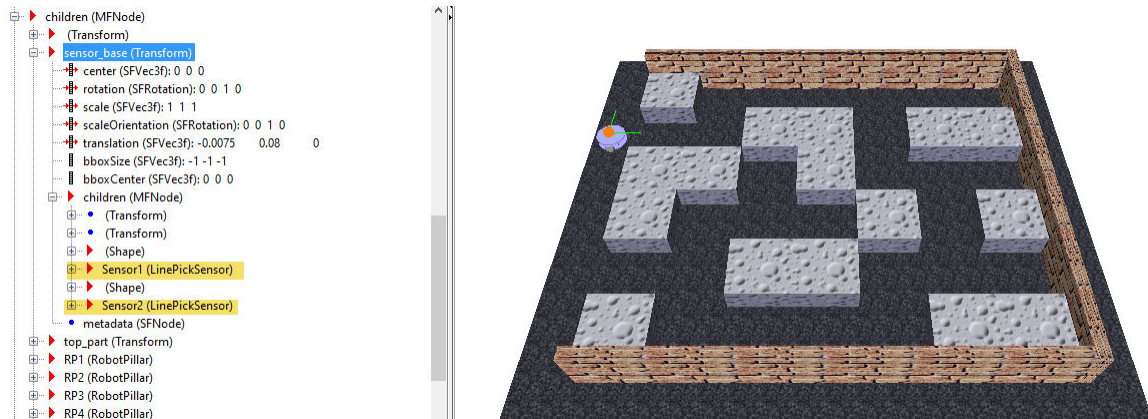
    % If collision is detected, change color to red.
    if col.isActive
        color.diffuseColor = [1 0 0];
    end

end
```

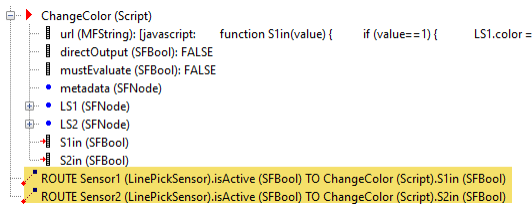
## Use Collision Detection Data in Virtual Worlds

You can use collision detection to manipulate virtual world objects, independently of a Simulink model or a virtual world object in MATLAB.

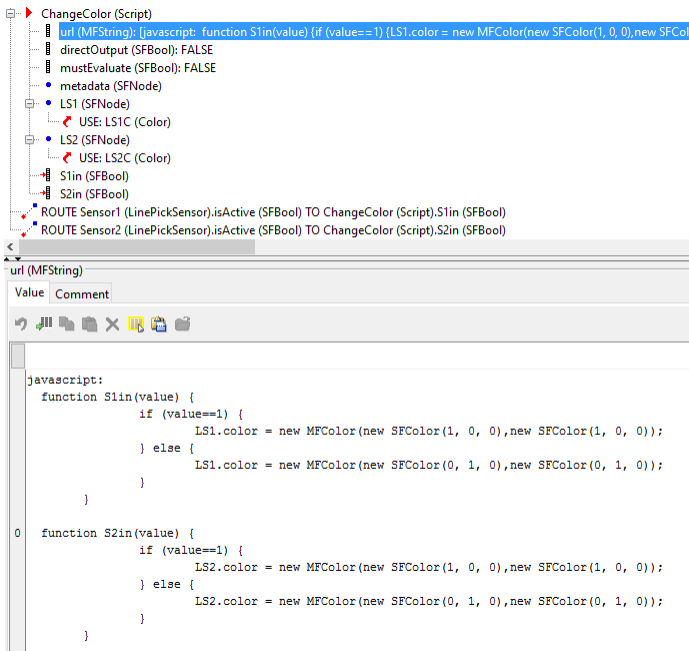
The `vrmaze` virtual world defines two green `IndexedLineSet` pick sensors (`Sensor1` and `Sensor2`) for the purple robot (`Robot` node).



The VRML code includes ROUTE nodes for each of the pick sensors.



The ROUTE nodes use logic defined in a Script node called ChangeColor.



## See Also

**Blocks**  
[VR Sink](#) | [VR Source](#)



## **Related Examples**

- “Add Sensors to Virtual Worlds” on page 5-20
- “Connect Virtual Worlds and Models” on page 3-2

## Virtual World Data Types

### In this section...

“Field Data Types” on page 5-30

“Virtual World Data Class Types” on page 5-32

Nodes use VRML and X3D virtual world data types to define objects and the types of data that can appear in the node fields and events.

This section explains these field data types and data class types.

### Field Data Types

The Simulink 3D Animation product provides an interface between the MATLAB and Simulink environment and virtual reality scenes. With this interface, you can set and get the scene node field values. Working with these values requires that you understand the relationship between virtual world data types and the corresponding MATLAB data types. The following table illustrates the virtual world data types and how they are converted to and from MATLAB types.

For a detailed description of the VRML fields, refer to the VRML97 Standard.

You can use MATLAB commands to read and save X3D files and to associate X3D files with Simulink models. For additional information about X3D support in Simulink 3D Animation, see “X3D Support” on page 1-9.

For information about the supported X3D specification, see ISO/IEC 19775-1:2013. For information about supported X3D encoding, see ISO/IEC 19776-1.3:201x and ISO/IEC 19776-2.3:201x.

VRML Type	Description	Simulink 3D Animation Type
SFBool	Boolean value true or false.	logical
SFFloat	32-bit, floating-point value.	single
SFInt32	32-bit, signed-integer value.	int32
SFTime	Absolute or relative time value.	double
SFVec2f	Vector of two floating-point values that you usually use for 2-D coordinates. For example, texture coordinates.	Single array (1-by-2)
SFVec3f	Vector of three floating-point values that you usually use for 3-D coordinates.	Single array (1-by-3)
SFColor	Vector of three floating-point values you use for RGB color specification.	Single array (1-by-3)
SFRotation	Vector of four floating-point values you use for specifying rotation coordinates (x, y, z) of an axis plus rotation angle around that axis.	Single array (1-by-4)

VRML Type	Description	Simulink 3D Animation Type
SFImage	Two-dimensional array represented by a sequence of floating-point numbers.	uint8 array (n-by-m-by-3)
SFString	String in UTF-8 encoding. Compatible with ASCII, allowing you to use Unicode® characters.	char
SFNode	Container for a node.	vrnode
MFFloat	Array of SFFloat values.	Single array (n-by-1)
MFInt32	Array of SFInt32 values.	int32 array (n-by-1)
MFFVec2f	Array of SFVec2f values.	Single array (n-by-2)
MFFVec3f	Array of SFVec3f values.	Single array (n-by-3)
MFColor	Array of SFColor values.	Single array (n-by-3)
MFRotation	Array of SFRotation values.	Single array (n-by-4)
MFString	Array of SFString values.	char array (n-by-1)
MFNode	Array of SFNode values.	vrnode

The Simulink 3D Animation software can work with various MATLAB data types, converting them if necessary:

- The inputs for the `setfield` function (and its dot notation form) and VR Sink and VR Source blocks, accept all meaningful data types on input. Both convert the data types into natural virtual world types as necessary. The data types include logicals, signed and unsigned integers, singles, and doubles.
- The `getfield` function and its dot notation form return their natural data types according to the table above.

To ensure backward compatibility with existing models and applications, use the Simulink 3D Animation `vrsetpref` function to define the data type support. Their names are as follows:

Property	Description
<code>DataTypeBool</code>	Specifies the boolean data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are 'logical' and 'char'. If set to 'logical', the virtual world boolean data type is returned as a logical value. If set to 'char', the virtual world boolean data type is returned 'on' or 'off'.
<code>DataTypeInt32</code>	Specifies the int32 data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . 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 virtual world int32 data type is returned as 'double'.
<code>DataTypeFloat</code>	Specifies the float data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are 'single' and 'double'. If set to 'single', the virtual world float and color data types (the types of most virtual world fields) are returned as 'single'. If set to 'double', the virtual world float and color data types are returned as 'double'.

## Virtual World Data Class Types

A node can contain four classes of data: `field`, `exposedField`, `eventIn`, and `eventOut`. These classes define the behavior of the nodes, how nodes are stored in memory, and how they can interact with other nodes and external objects.

VRML Data Class	Description
<code>eventIn</code>	An event that the node can receive
<code>eventOut</code>	An event that the node can send
<code>field</code>	A private node member, holding node data
<code>exposedField</code>	A public node member, holding node data

### **eventIn**

Usually, `eventIn` events correspond to a `field` in the node. Node fields are not accessible from outside the node. The only way you can change them is by having a corresponding `eventIn`.

Some nodes have `eventIn` events that do not correspond to any field of that node, but provide additional functionality for it. For example, the **Transform** node has an `addChilden eventIn`. When this event is received, the child nodes that are passed are added to the list of children of a given transform.

You use this class type for fields that are exposed to other objects.

### **eventOut**

This event is sent whenever the value of a corresponding node field that allows sending events changes its value.

You use this class type for fields that have this functionality.

### **field**

A field can be set to a particular value in the virtual world 3D file. Generally, the field is private to the node and its value can be changed only if its node receives a corresponding `eventIn`. It is important to understand that other nodes or the external authoring interface cannot change the field.

You use this class type for fields that are not exposed and do not have the `eventOut` functionality.

### **exposedField**

This powerful data class serves many purposes. You use this class type for fields that have both `eventIn` and `eventOut` functionality. The alternative name of the corresponding `eventIn` is always the field name with a `set_` prefix. The name of the `eventOut` is always the field name with a `_changed` suffix.

The `exposedField` class defines how the corresponding `eventIn` and `eventOut` behave. For all `exposedField` classes, when an event occurs, the field value is changed, with a corresponding change to the scene appearance, and an `eventOut` is sent with the new field value. These changes allow the chaining of events through many nodes.

The `exposedField` class is accessible to scripts, whereas the `field` class is not.

## **See Also**

### **More About**

- “Virtual Reality Modeling Language (VRML)” on page 1-11
- “X3D Support” on page 1-9
- “Simulink 3D Animation Textures” on page 5-34

## Simulink 3D Animation Textures

The following are texture file recommendations for Simulink 3D Animation models:

- Where possible, scale source texture files to a size equal to a power of 2 in both dimensions. Doing so ensures optimal performance for the Simulink 3D Animation viewer. If you do not scale the files, the Simulink 3D Animation viewer can attempt to descale the image or create textures with undesired resolutions.
- Use source texture files whose size and detail are no more than what you need for your application.
- Where possible, use the Portable Network Graphics (PNG) format as the static texture format. You can also use the GIF and JPG graphic formats.
- For movie textures, use the MPEG format. For optimal performance, be sure to scale source texture files to a size equal to the power of 2 in both dimensions.

### See Also

#### More About

- “Virtual Reality Modeling Language (VRML)” on page 1-11
- “X3D Support” on page 1-9
- “Virtual World Data Types” on page 5-30

## Add Sound to a Virtual World

To add sound to a virtual world, use a `Sound` node. You can include an `AudioClip` node in a `Sound` node. For an `AudioClip` node, use a mono or stereo WAV file in uncompressed PCM format.

To listen to the sound, use a computer that supports sound. For details, see “Listen to Sound in a Virtual World” on page 7-43.

---

**Note** A stereo sound source retains its channel separation during playback. Simulink 3D Animation attenuates the sound based on the distance of the viewer from the sound location. The relative position of the viewer to the sound location and the viewer direction in the virtual world do not affect the stereo channels. There is no impact even if the `Sound` node has the `spatialize` field set to `true`.

---

The following code adds to a virtual world a sound that switches on and off based on a logical signal.

```
DEF SoundSwitch Switch {
  choice [
    DEF MySound Sound {
      source DEF CraneNoise AudioClip {
        url "sound/crane_run.wav"
        loop TRUE
      }
    }
  ]
}
```

### See Also

### Related Examples

- “Listen to Sound in a Virtual World” on page 7-43

## Use CAD Models with the Simulink 3D Animation Product

In this section...
“Use of CAD Designs” on page 5-36
“Import CAD Designs” on page 5-36
“Integrate the Imported Model Virtual World” on page 5-36

---

**Note** These topics assume that the reader has a moderate knowledge of the Simulink 3D Animation product. For VRML-specific information, such as the description of nodes and their fields, refer to the VRML97 standard.

---

### Use of CAD Designs

When you work with models of dynamic systems, often it is helpful to visualize them in a three-dimensional virtual reality environment. If most of the 3D designs in your organization are created using CAD tools, convert these designs into forms that you can use with:

- Simulink models
- Simscape Multibody models
- MATLAB models

To adapt existing CAD designs for visualization using the Simulink 3D Animation software:

- Import CAD designs
- Integrate the virtual world created from the imported CAD model into the Simulink 3D Animation environment.

### Import CAD Designs

You can use the following techniques to import CAD designs for use in the Simulink 3D Animation product:

- Import STL and Physical Modeling file directly
- Import VRML models from CAD tools
- Import VRML models from CATIA® software

### Integrate the Imported Model Virtual World

After you import the CAD model, use the Simulink 3D Animation software to modify the resulting virtual world so that you can use it effectively. Also, create associations between the virtual world and Simulink and Simscape Multibody models.

### See Also

#### Functions

`stl2vrml` | `vrcadcleanup` | `vrphysmod`



## **Related Examples**

- “Import STL and Physical Modeling XML Files” on page 5-38
- “Import 3D Models from CAD Tools” on page 5-40
- “Import VRML Models from CATIA Software” on page 5-45
- “Modify the CAD Model Virtual World” on page 5-51
- “Link to Simulink and Simscape Multibody Models” on page 5-60

## Import STL and Physical Modeling XML Files

CAD models frequently use STL (STereoLithography) format files or Physical Modeling XML files.

You can use the 3D World Editor to import STL and Physical Modeling XML files directly into a VRML virtual scene. You can integrate content from those files into a virtual world, converting the structure into VRML.

---

**Note** You cannot use the 3D World Editor to import STL and Physical Modeling XML files directly into an X3D virtual scene.

---

To import an STL or Physical Modeling XML file:

- 1 In the 3D World Editor, select the **Root** node or an **MFNode** node (usually the `children` node of a **Transform** or **Group** node).
- 2 From the **Nodes > Import From** menu item, select either **STL File** or **Physical Modeling XML File**.

---

**Tip** Alternatively, you can right-click the **Root** node or an **MFNode** node and use the **Import From** menu item. To insert the new node in the middle of a node list, after the selected node, use the **Nodes > Import From** menu path.

---

---

**Note** To import a Physical Modeling XML file, the target folder must be writable.

---

Alternatively, you can use the `stl2vrml` function.

## Results

Importing an STL file creates a **Transform** node containing a **Shape** node with **IndexedFaceSet** geometry that represents the original STL shape. The new **Transform** is create in one of these locations:

- At the main level of the scene hierarchy (at the end of the file)
- As a child of a selected grouping node

Importing a Physical Modeling XML file creates an individual VRML file for each STL file that the XML file references. The created VRML files are stored in the folder where the edited VRML file is located.

In the edited VRML file, the newly created **Transform** contains hierarchical structure **Transforms** that correspond to the structure of objects described in the XML file. Individual VRML shape files in this structure are referenced using **Inline** nodes.

---

**Note** When importing a Physical Modeling XML file into a new, unsaved VRML model, children VRML files are created in the current working folder. If you save the file into a different folder, move the referenced VRML files to the new location.

---

## **See Also**

### **Related Examples**

- “Use CAD Models with the Simulink 3D Animation Product” on page 5-36
- “Import 3D Models from CAD Tools” on page 5-40
- “Import VRML Models from CATIA Software” on page 5-45
- “Modify the CAD Model Virtual World” on page 5-51
- “Link to Simulink and Simscape Multibody Models” on page 5-60

## Import 3D Models from CAD Tools

### In this section...

“Level of Detail Considerations” on page 5-40

“Units Used in Exported Files” on page 5-40

“Coordinate System Used” on page 5-41

“Assembly Hierarchy” on page 5-41

To import models from CAD tools, convert your product assembly model into the X3D format used by the Simulink 3D Animation software. Most CAD tools have X3D export filters. If the export filter is not directly available in the CAD tool, you can use conversion utilities available from third parties.

When exporting CAD models into the X3D format, you can set several options to customize the output. You can set options that are specific to the export filters or are general CAD file properties. Consult your CAD system documentation for specific details on how to set these properties. Some of the most typical and useful CAD file properties are:

- Level of Detail Considerations
- Units Used in Exported Files
- Coordinate System Used
- Assembly Hierarchy

### Level of Detail Considerations

Usually CAD models are parametric models that use proprietary object rendering methods to handle various contexts. During model export, the internal parametric model of the assembly is tessellated. In this process, the model surface is divided into triangular meshes, represented by the `IndexedFaceSet` nodes. Before tessellation, set the granularity of the mesh so that it is suitable for further use. Modifying the polygon count later is not practical. The resolution independent information of the object shape and structure is lost and cannot be reconstructed based on the tessellated model.

For effective rendering of moving parts, keep virtual world models as simple as possible, with minimal visible model degradation. Find the appropriate compromise between these two requirements.

Computers and graphic accelerators have a range of performance levels, so there is no firm recommendation for the number of polygons or triangles suitable for use. To assess the complexity of a model, you can display the resulting virtual world 3D file in the Simulink 3D Animation viewer and observe the viewer response to navigation. If you can navigate the virtual world without any significant delays, the model is suitable for further work. If you connect the virtual world to a Simulink model, you have access to more precise measures of suitability. For example, you can find the number of frames rendered per second during simulation.

### Units Used in Exported Files

X3D length units are in meters. To scale exported parts correctly in the virtual world, export the parts using meters. If the exported objects are small or large, consider creating your virtual world in some other scale. In this case, export the objects using units other than meters.

Virtual reality viewers measure using dimensions that are comparable to the dimensions of people, to achieve the immersion effect of virtual reality. Viewers assume that you prepared the scene so that it can be walked through or examined by a virtual visitor to the scene (an avatar). The physical dimensions of the avatar are used in calculations for purposes like collision detection, near-object clipping, or terrain following. To customize avatar dimensions and other navigation-specific parameters such as default navigation speed), use the `NavigationInfo` node. The Simulink 3D Animation viewer enables effective navigation in the virtual world, including scaled scenes. For example, you can use the viewer to inspect miniature objects or to visualize a large-scale aircraft operation in space.

## Coordinate System Used

X3D uses a Cartesian coordinate system with axes defined so that:

- +x points right
- +y points up
- +z points out of the screen

To avoid transforming object axes into the virtual world system later on, whenever possible, export CAD models using an identical coordinate system. If your CAD tool uses a different coordinate system and does not allow you to change it for the exported objects, note the system differences. Then implement axes transformations in your model later.

For example, if you export a vehicle model so that it points towards the +x axis on a road in the virtual world:

- Make the road also point towards the +x direction.
- Use the x coordinate for the model of vehicle dynamics.
- Make a note of the orientation of the parts in the coordinate system.

When the CAD tool allows you to animate parts and assemblies, reset their positions to the initial state before the export.

## Assembly Hierarchy

The export of assembly of parts varies based on the structure of the model, which usually comes in two forms:

- All parts are independent from each other, or objects in the scene are independent from each other at the same level of the scene hierarchy. The exported virtual world 3D file has a flat structure, with all part coordinates defined in global coordinates.
- Parts follow a hierarchy defined in the CAD tool. The exported virtual world 3D file uses this hierarchy via the `Transform-children` mechanism, to create a nested structure. In this case, usually part coordinates are defined in the local coordinate system of the parent of the part.

For example, you can export a robot with the following object hierarchy. The coordinates of each part are defined in the local coordinate system of the parent:

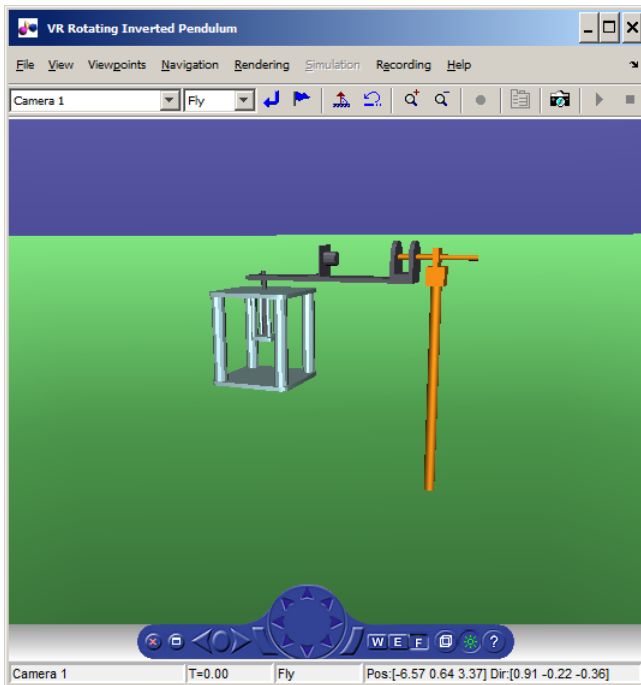
**rotating support — arm — wrist — hand — tool**

When the rotating support moves, all other parts are designed to move with it.

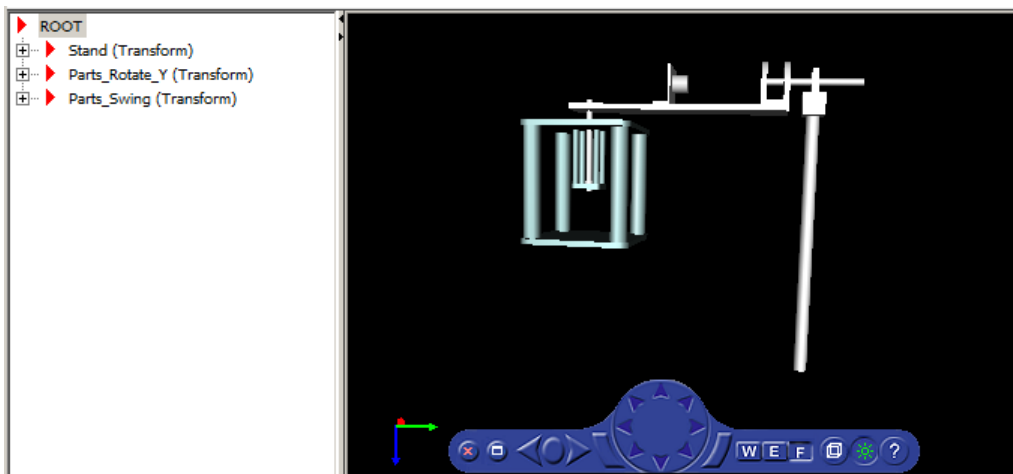
The hierarchy of the virtual world 3D file must correspond to the coordinates used in the dynamic model of the assembly as follows:

- If all parts in the Simulink or Simscape Multibody model are defined in global coordinates, use a flat virtual world structure.
- If all parts in the Simulink or Simscape Multibody model follow hierarchical relationships, use a nested virtual world structure.

To illustrate these two cases, imagine a rotating pendulum. As the gray arm rotates about the vertical axis, the orange pendulum swings about the z axis in local coordinates of the rotating gray arm.



If the pendulum dynamics model uses global coordinates for all moving parts, the virtual world model has a flat structure.



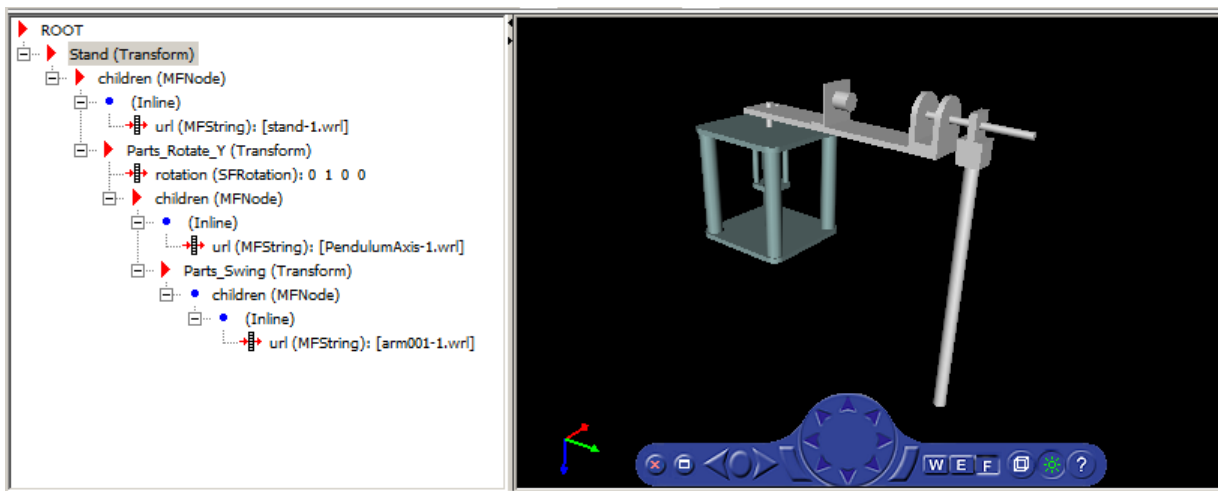
Here is the code for the flat structure.

```

1  #VRML V2.0 utf8
2
3  DEF Stand Transform {
4      children [
5          Inline {
6              url ["stand-1.wrl"]
7          }
8      ]
9  }
10
11 DEF Parts_Rotate_Y Transform {
12     children [
13         Inline {
14             url ["PendulumAxis-1.wrl"]
15         }
16     ]
17 }
18
19 DEF Parts_Swing Transform {
20     children [
21         Inline {
22             url ["arm001-1.wrl"]
23         }
24     ]
25 }

```

If the pendulum dynamics model uses local coordinates for moving parts, the corresponding virtual world model has a nested structure.



Here is the code for the nested structure.

```

1  #VRML V2.0 utf8
2
3  DEF Stand Transform {
4    children [
5      # Static parts of mechanism
6      Inline {
7        url ["stand-1.wrl"]
8      },
9
10     # Parts that rotate along Y axis
11     DEF Parts_Rotate_Y Transform {
12       # Initial rotation defined along the Y axis
13       # to facilitate using VR Expander block
14       rotation 0 1 0 0
15       children [
16         Inline {
17           url ["PendulumAxis-1.wrl"]
18         },
19
20         # Parts that rotate along Z axis
21         DEF Parts_Swing Transform {
22           # Initial rotation defined along the Z axis
23           # to facilitate using VR Expander block
24           rotation 0 0 1 0
25           children [
26             Inline {
27               url ["arm001-1.wrl"]
28             }
29           ]
30         }
31       ]
32     }
33   ]
34 }

```

Some third-party tools allow you to export each part of the assembly into separate virtual world 3D files. All parts are then referenced in one main file using the `InLine` mechanism. Referencing in this manner is the recommended way to work with assemblies, as the main file is small and easy to understand and modify.

## See Also

### Functions

`stl2vrml` | `vrcadcleanup` | `vrphysmod`

## Related Examples

- “Use CAD Models with the Simulink 3D Animation Product” on page 5-36
- “Import STL and Physical Modeling XML Files” on page 5-38
- “Import VRML Models from CATIA Software” on page 5-45
- “Modify the CAD Model Virtual World” on page 5-51
- “Link to Simulink and Simscape Multibody Models” on page 5-60



## Import VRML Models from CATIA Software

### In this section...

“CATIA Coordinate Systems” on page 5-45

“Settings That Affect the VRML Output” on page 5-45

“Level of Detail” on page 5-46

“VRML Export Filter Settings” on page 5-46

“VRML Models Exported from the CATIA Environment” on page 5-46

“Adjust the Resulting VRML Files” on page 5-48

You can use CAD designs created in the CATIA product to create Simulink 3D Animation virtual reality scenes. CATIA models are hierarchical trees comprised of products that contain parts.

To export CATIA parts or products to the VRML format, in the CATIA dialog box, select **File > Save as** and select VRML in the **Save as type** list.

**Note** You cannot use the Simulink 3D Animation to import CATIA models to X3D files in Simulink 3D Animation.

When exporting products, the CATIA software creates one compound VRML file that contains all the parts of the product.

To export each part of the assembly hierarchy into a separate VRML file, in the CATIA environment:

- 1 Save each part individually to a separate virtual world 3D file.
- 2 Create the main model virtual world 3D file manually, with `InLine` references to the part files.

### CATIA Coordinate Systems

Also, the CATIA software exports background color and viewpoints. The software exports individual parts without these properties.

By default, the CATIA software uses right-handed Cartesian coordinate system, identical to the MATLAB coordinate system on page 1-12. Account for the coordinate system when you export objects from the CATIA environment into virtual worlds. Also account for the coordinate system when you manipulate exported objects using the Simulink 3D Animation software.

You can also define a different coordinate system. Within the current geometrical set, create an axis system. Doing so sets this new system as a reference system that you can use to export the VRML virtual world. Consider creating such an axis system so that it corresponds to the virtual world coordinate system. This approach makes all the coordinates and orientations of objects compatible with other objects you combine into virtual worlds. See “Virtual World Coordinate System” on page 1-12.

### Settings That Affect the VRML Output

In the CATIA environment, the properties that affect the VRML output are available in two options dialog boxes:

- Display Performances dialog box
- VRML Compatibility dialog box

## Level of Detail

The level of detail of the exported VRML file (accuracy of the tessellation mesh of objects) corresponds to the setting of CATIA general visualization mesh. In the CATIA menu, select **Tools > Options > General > Display > Performances**. In the resulting dialog box, select the **3D Accuracy** options to control the visualization mesh detail.

Use the proportional method of tessellation (arcs are substituted by line segments based on their relative, not absolute, accuracy). This method works for models regardless their dimensions. For maximum accuracy of the exported virtual world model, set the slider at the rightmost position. If the resulting file is too complex to be handled effectively with VRML rendering tools, experiment with this accuracy setting. Find the setting that gives you the smallest possible virtual world model that is visually acceptable.

## VRML Export Filter Settings

The CATIA software enables you to tune some VRML export options.

- 1 Select **Tools > Options > General > Compatibility > VRML** options.
- 2 Select VRML97 as the export format.

The Simulink 3D Animation software uses VRML97 standard format.

- 3 Select the **Save normals** check box.

This option affects whether to export explicit face normals definitions.

- 4 Clear the **Save edges** check box

Clear this check box for optimum performance. Selecting this check box directs the CATIA software to export object edges (in the form of IndexedLineSets).

- 5 Set the appropriate **Save textures** check boxes to the desired settings.

In particular, if you want to save textures, select the **Save textures in external files** option. This option generates external JPG files for object textures.

- 6 Select the VRML model background color.

This option applies only to exporting products.

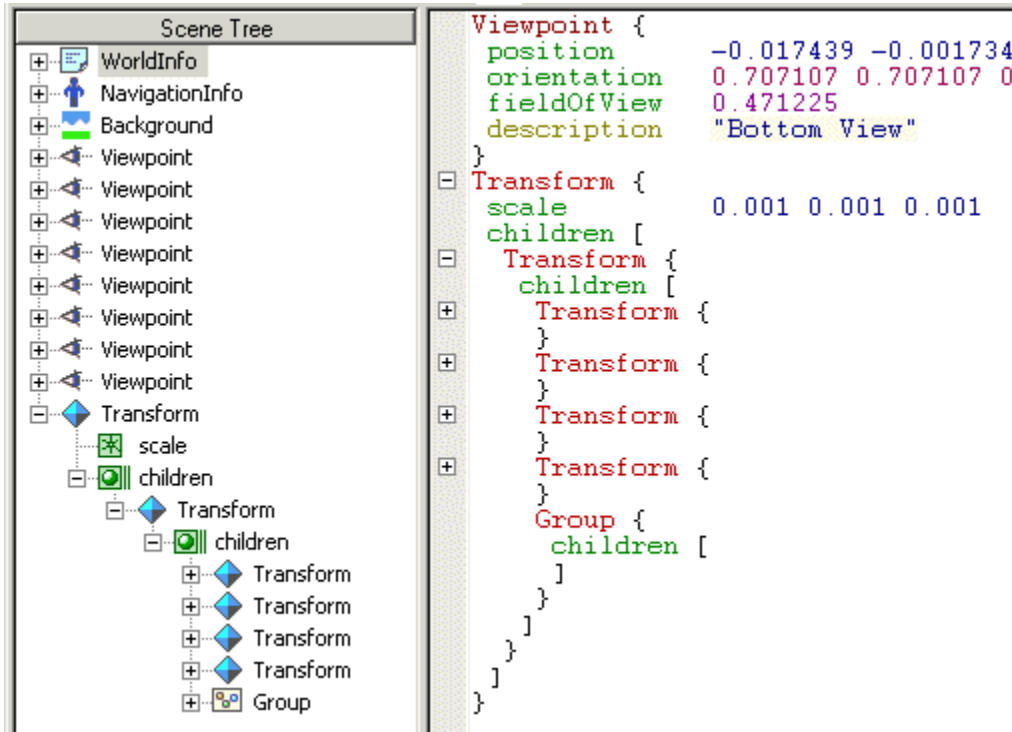
## VRML Models Exported from the CATIA Environment

The CATIA software exports CATProducts and their CATParts as VRML transforms. The structure of these transforms corresponds to the CATIA model hierarchy. In addition to transforms that represent physical elements, the CATIA software creates several transforms and groups in the VRML file. The transforms and groups represent relationships between objects and other model properties defined in the CATIA environment.

Some of these additional nodes can be empty. Many CATIA model properties do not have equivalents in the VRML language. Each part transform contains a hierarchy of nested transforms, groups, and

shapes that correspond to the part internal structure. Some of these elements have synthetic DEF names (for example, \_0161DC70). Usually, work with the main transforms that represent each part.

Here is the VRML model of a cylinder assembly consisting of four parts:



The left tree view illustrates the overall structure of the model.

- The CATIA software saves the general model information in the `WorldInfo`, `NavigationInfo`, and `Background` nodes.
- The software exports the default CATIA viewpoints (it does not export user-defined viewpoints).

Common to all products exported to VRML, there is a top-level transform node representing the `CATProduct`.

In the CATIA software, `Product CylinderAssembly1` consists of four parts:

- `CrankAssembly1`
- `CylinderSleeve1`
- `PistonAssembly1`
- `CrankshaftAssembly1`

The export does not preserve the `CATProduct` and `CATPart` names. You can identify these objects in the VRML file in the tree view and in the text mode. For clarity, in the figure, the contents of the part transforms are collapsed so that only the top-level objects are visible. After four transforms representing `CATParts`, the export adds an empty `Group` node for defining CATIA Constraints. You can delete such empty nodes from the VRML model.

The conversion scales contents of the `CATProduct` down by a factor of 1000 (conversion of units from millimeters to meters).

When you have VRML files created with the CATIA software, consider using these features with the Simulink 3D Animation software.

**CATIA Feature Support**

Feature	Conversion Notes
Object names	<p>Exporting to VRML does not preserve CATProduct and CATPart names. The CATIA environment only creates synthetic VRML DEF names for subparts, materials, and object coordinate fields. These synthetic names change between two or more consecutive export operations.</p> <p>To work with the Simulink 3D Animation software, provide meaningful DEF names for the objects that you want to control from the MATLAB /Simulink environment.</p>
Vertex coordinates	<p>The CATIA software saves all vertex coordinates for a part in one VRML coordinate field, which resides in the first exported IndexedFaceSet for the part. Several subparts throughout the file reference the VRML coordinate field with USE directives.</p> <p>Preserve this reference. Do not delete or rename the original Coordinate field DEF name.</p>
Materials	<p>The VRML file stores only one material per part. If the part consists of several subparts in VRML, their material also uses the USE reference to the material of the first subpart.</p>
Textures	<p>Textures are supported.</p>
Level of detail	<p>LOD (exporting parts in several levels of detail for more efficient visualization) is not supported.</p>
Units	<p>The CATIA software exports models in millimeters, VRML units are meters.</p> <p>Scale resulting objects to visualize them effectively. The conversion scales the main Transform representing the CATIA product by a factor of 0.001 (conversion from millimeters to meters). The scaling occurs regardless of the units used in the CATIA document.</p>
Viewpoints	<p>The VRML file does not save user-defined CATIA viewpoints.</p>

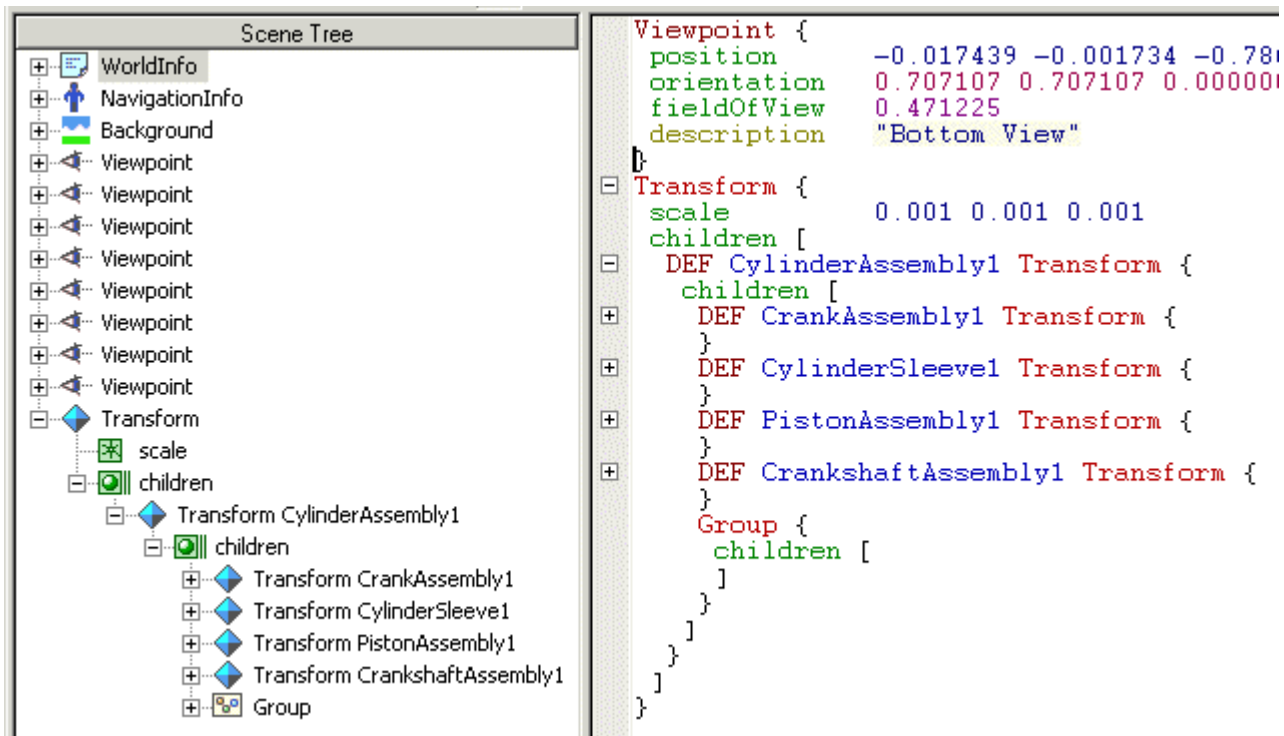
**Adjust the Resulting VRML Files**

To use the exported VRML models with the Simulink 3D Animation software, adjust exported VRML files. You can perform these adjustments manually, as described in this topic, or use the vrcadcleanup and vrphysmod functions to perform some of these tasks.

**Add DEF Names to Part Transforms.**

In the VRML file, assign a unique name for each VRML object. Add the DEF Object\_Name statement to each part Transform line.

This example shows a VRML file that has DEF names added to the cylinder assembly.



Do not adjust parts in the scene that you do not want to control from the MATLAB environment.

### Scale VRML Objects

To convert CATProduct size from millimeters to meters (VRML default units), the CATIA software wraps the transform corresponding to the CATProduct with an additional transform. In this transform, the scale field is defined. The preceding example illustrates this size conversion.

If you have a small object, or an object to place into an overall virtual world, adjust this scale.

If you leave the VRML object scale in the default state, the local part coordinates are still in millimeters. Remember this fact when controlling these parts from the MATLAB or Simulink environment. If your MATLAB or Simulink model units are meters, scale each part individually to achieve correct results. Delete the scale field from the top-level transform, and add it to each individual part transform. For example:

```
Transform {
  children [
    DEF CylinderAssembly1 Transform {
      children [
        DEF CrankAssembly1 Transform {
          scale 0.001 0.001 0.001
        }
        ..
      ]
    }
  ]
}
```

### See Also

#### Functions

stl2vrml | vrcadcleanup

## **Related Examples**

- “Use CAD Models with the Simulink 3D Animation Product” on page 5-36
- “Import STL and Physical Modeling XML Files” on page 5-38
- “Import 3D Models from CAD Tools” on page 5-40
- “Modify the CAD Model Virtual World” on page 5-51
- “Link to Simulink and Simscape Multibody Models” on page 5-60

## Modify the CAD Model Virtual World

### In this section...

“Wrap Shape Objects with Transforms” on page 5-51

“Add DEF Names” on page 5-51

“Additional Virtual World Modifications” on page 5-52

To modify the results of CAD tool export filters manually, you can use the 3D World Editor or other editor. For example, you can compose the converted model into an urban or manufacturing environment, or add objects such as viewpoints, backgrounds, and lights before using them in Simulink 3D Animation virtual worlds. Typically, adjusting exported files manually in an editor involves several modifications.

### Wrap Shape Objects with Transforms

CAD tools export parts into VRML or X3D as individual shapes using various object types such as `Shape` or `Inline` nodes. To control part positions and orientations, wrap each such `Shape` or `Inline` node with a node that allows for the changing of these properties. This wrapping node is the `Transform` node, which transforms the coordinates of its children. For instance, after wrapping with a `Transform` node, an `Inline` node has syntax similar to this syntax:

```
Transform {
  children [
    Inline {
      url ["robot_arm1.wrl"]
    }
  ]
}
```

To set the initial location of the entire assembly in the virtual world, consider wrapping all parts of the assembly with an additional `Transform` node.

### Add DEF Names

CAD export filters often export objects with no names or with synthetic nondescriptive names. To make export objects accessible in MATLAB, give each virtual world object a unique name in the virtual world 3D file. For example, name the object by adding a `DEF Object_Name` statement to the `Transform` line. After adding the `DEF Object_Name`, the `Robot_Arm1` definition in the main virtual world 3D file has the syntax similar to this syntax:

```
DEF Robot_Arm1 Transform {
  children [
    Inline {
      url ["robot_arm1.wrl"]
    }
  ]
}
```

The Simulink 3D Animation functions and in the user interface (such as the descriptions of inputs to the VR Sink block) use these object names. To help with managing the orientation in the object hierarchy, give the parts descriptive names.

**Note** Sometimes it is necessary to correct bugs introduced in the file by the CAD tool export filter. As the VRML and X3D format is a text-based format codified by an ISO® standard, these bugs are relatively easy to identify and correct. If problems occur when you are using exported VRML or X3D files in the Simulink 3D Animation software, consult MathWorks technical support.

---

## Additional Virtual World Modifications

To work with the virtual world effectively, you can make additional modifications to the scene file using a virtual world editor. Make these changes on an ongoing basis, in parallel with developing and using the dynamic model.

- Add a scene title by adding a `WorldInfo` node with a scene title. Simulink 3D Animation software uses the title as the virtual world description.
- Enhance the scene.
  - Add the `Background` node defining a color backdrop that simulates the ground and sky, and optional background textures, such as panoramas for the scene.
  - Add scene surroundings. This step is not crucial for the visualization of interactions between parts in a machine assembly, but is important for the visualization of simulations. For example, for aircraft and vehicle dynamics the position of one object relative to the scene in which it operates is important. Adding scene surroundings provides context.

For example, to visualize vehicle dynamics, place a virtual car on a virtual road. Make both objects to scale (the length units in the car and road models must match). Place the car in an appropriate position relative to the road. Set proper car scaling, placement, and orientation in the scene by defining corresponding fields of the `Transform` node main object (see “Wrap Shape Objects with Transforms” on page 5-51).

- Configure scene display and navigation.
  - Add several viewpoints to be able to observe the object conveniently from different positions. The viewpoints can be static or moving. Define a static viewpoint as an independent object at the top level of the scene hierarchy. To create a moving viewpoint, attached a viewpoint to objects that move in the scene during simulation. Such viewpoints are defined as siblings of moving objects in the scene hierarchy. For an example of a viewpoint moving with the object, see the viewpoint `Ride on the Plane` in the Simulink 3D Animation `vrtrkoff.wrl` example.
  - Illuminate a scene by adding lights to it. Although virtual world viewers always have a headlight available, consider defining lights in the scene so that it looks the same for everyone. The most useful type of light to illuminate a whole scene is the `DirectionalLight` node. To illuminate objects from several directions, consider using a combination of several such lights.
  - Add the `NavigationInfo` node defining the scene default navigation speed and avatar size that ensures correct display of the object from near and far distances.

For an example of a complete scene definition, see the `octavia_scene.wrl` file that is part of the Simulink 3D Animation `vr_octavia` example.

## See Also

### Related Examples

- “Use CAD Models with the Simulink 3D Animation Product” on page 5-36



- “Import STL and Physical Modeling XML Files” on page 5-38
- “Import 3D Models from CAD Tools” on page 5-40
- “Import VRML Models from CATIA Software” on page 5-45
- “Link to Simulink and Simscape Multibody Models” on page 5-60

## Import Visual Representations of Robot Models

### In this section...

“Import a DAE File” on page 5-54  
 “Import a URDF File” on page 5-55  
 “Import an SDF File” on page 5-56  
 “Define Viewpoint to Make Imported Model Visible” on page 5-58  
 “Limitations” on page 5-58

You can import into a virtual world in Simulink 3D Animation robot and robot environment 3-D models that are defined using unified robot description format (.urdf) or simulation description format (.sdf). The URDF file format is an XML file format that defines various model (typically robot) properties. When you import from a URDF file, only the visual representation of the model is imported. This representation typically refers to model component shape files, defined as STL or DAE files. You can import visual properties of the objects present in the URDF or SDF files into corresponding hierarchical structure of Transform nodes containing IndexedFaceSet objects with textures.

URDF and SDF files use Collada DAE and STL formats to define visual properties of scene objects.

To import visual representation of robots, in the 3D World Editor, select **Nodes > Import From**. Then select **STL File**, **Physical Modeling XML File**, **URDF File**, **SDF File**, or **COLLADA File**. From the MATLAB command line, you can use the `vimport` function to import .urdf, .dae, .stl, and .sdf files into a virtual world.

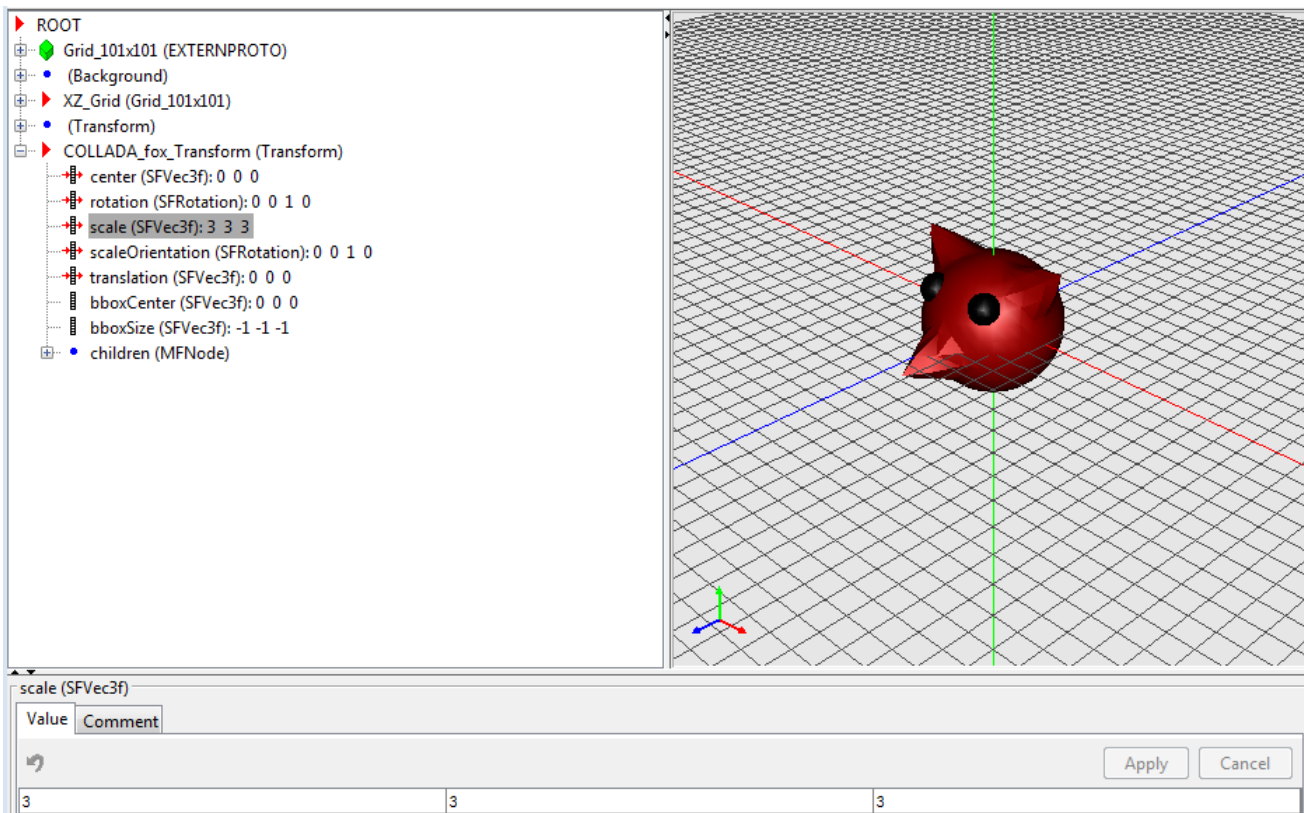
### Import a DAE File

The URDF and SDF import supports a subset of the Collada DAE 3D file format (objects that are typically present in URDF and SDF files: geometric shape objects with textures).

This example imports a Collada DAE file, `fox.dae`, into a scene with predefined grid, gray background, and viewpoint. The fox head is a sphere with 1-m radius, placed at the origin and looking in the direction of VRML +z-axis.

The `fox.dae` is shipped with the Simulink 3D Animation software. The path is `matlabroot/toolbox/sl3d/sl3ddemos/fox.dae`.

- 1 From the MATLAB Toolstrip, in the **Apps** tab, in the **Simulation Graphics and Reporting** section, click **3D World Editor**.
- 2 In the 3D World Editor, select **File > New from Template**.
- 3 In the file dialog box, select the `Grids` folder and then select the `XZGrid.wrl` virtual world file.
- 4 Save the virtual world in the current folder. Select **File > Save as** and name the virtual world file `fox_dae.wrl`.
- 5 Import the DAE model. Select **Nodes > Import From > COLLADA File**. Select the `fox.dae` file.
- 6 Open the `COLLADA_fox_Transform` node and set the `scale` field to 3 for each dimension.

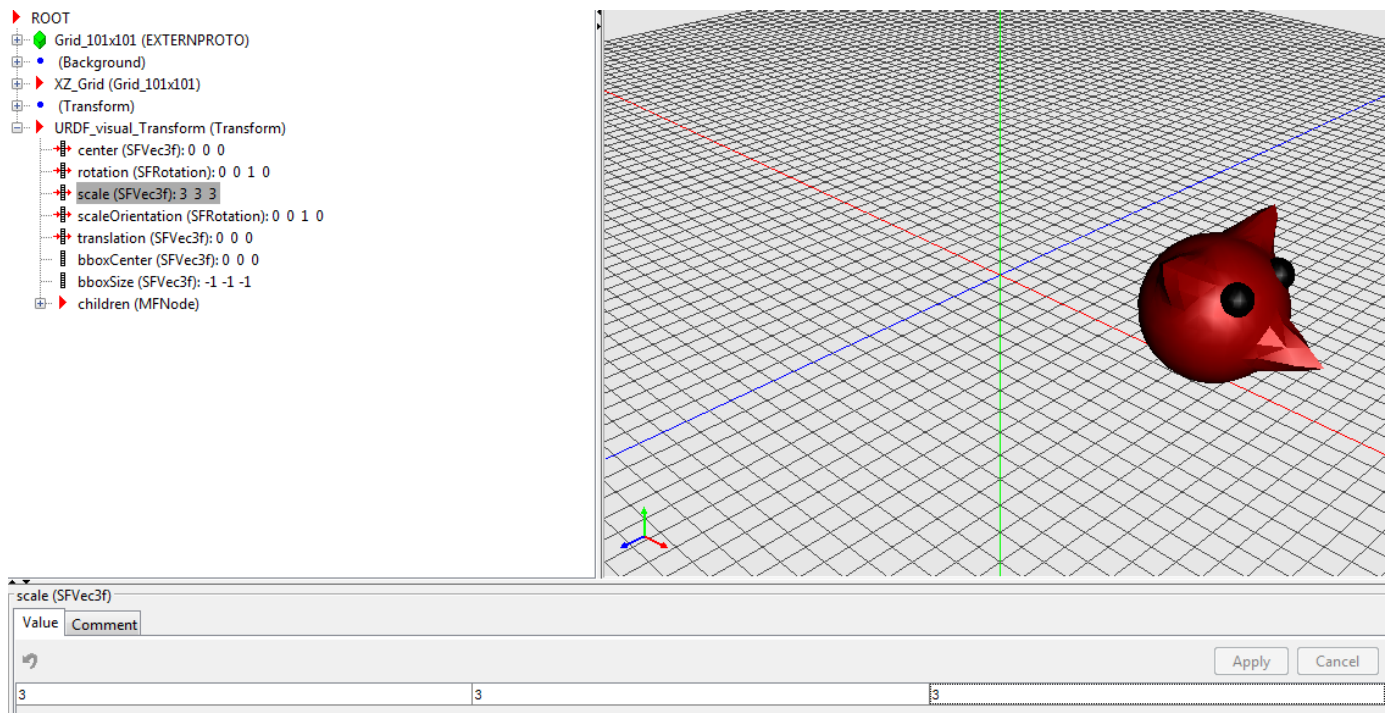


## Import a URDF File

In this example, the `fox.urdf` file refers to `fox.dae` file. The URDF file transforms the original DAE model so that the fox is translated to the position  $[4\ 1\ 0]$  and rotated so that it points to the VRML  $+x$ -axis direction.

The `fox.urdf` is shipped with the Simulink 3D Animation software. The path is `matlabroot/toolbox/sl3d/sl3ddemos/fox.urdf`.

- 1 From the MATLAB Toolstrip, in the **Apps** tab, in the **Simulation Graphics and Reporting** section, click **3D World Editor**.
- 2 In the 3D World Editor, select **File > New from Template**.
- 3 In the file dialog box, select the Grids folder and then select the `XZGrid.wrl` virtual world file.
- 4 Save the virtual world in the current folder. Select **File > Save as** and name the virtual world file `fox_urdf.wrl`.
- 5 Import the URDF model. Select **Nodes > Import From > URDF File**. Select the `fox.urdf` file.
- 6 Open the `URDF_visual_Transform` node and set the `scale` field to 3 for each dimension.

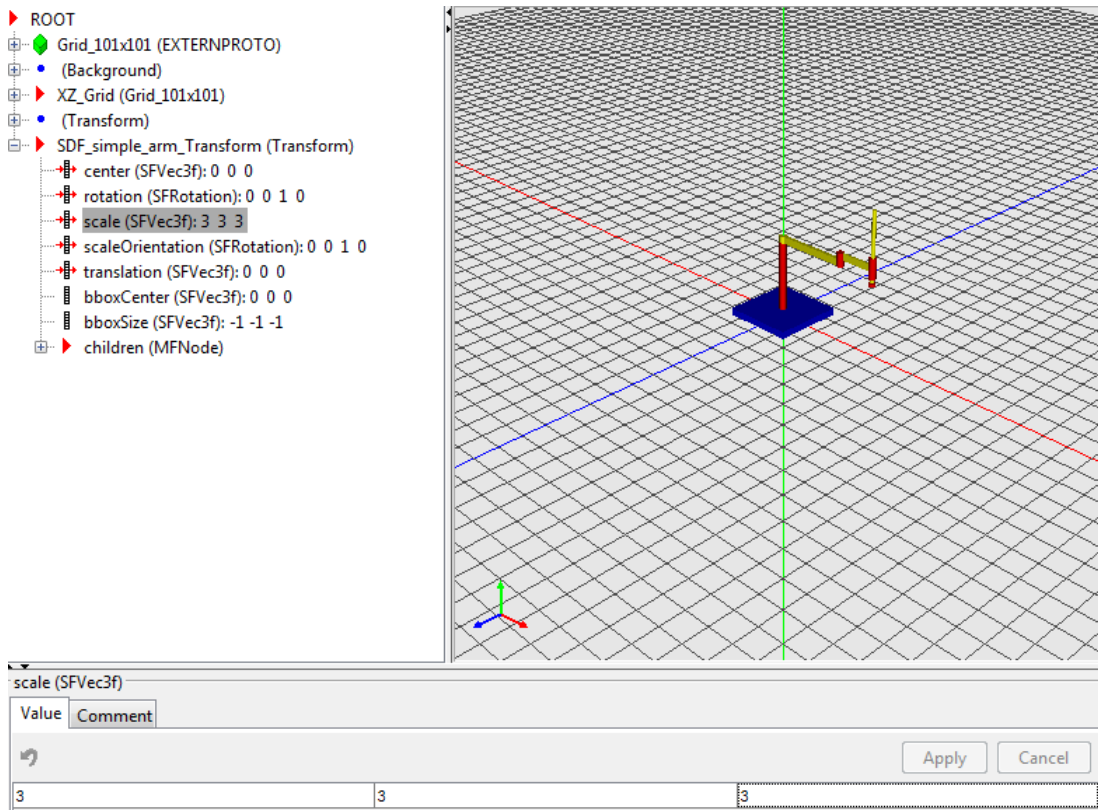


## Import an SDF File

SDF is an XML format that describes objects and environments for robot simulators, visualization, and control, originally developed as part of the Gazebo robot simulator. The folder in which the `model.sdf` file is located defines the SDF model name. This example shows how to import a simple arm link mechanism.

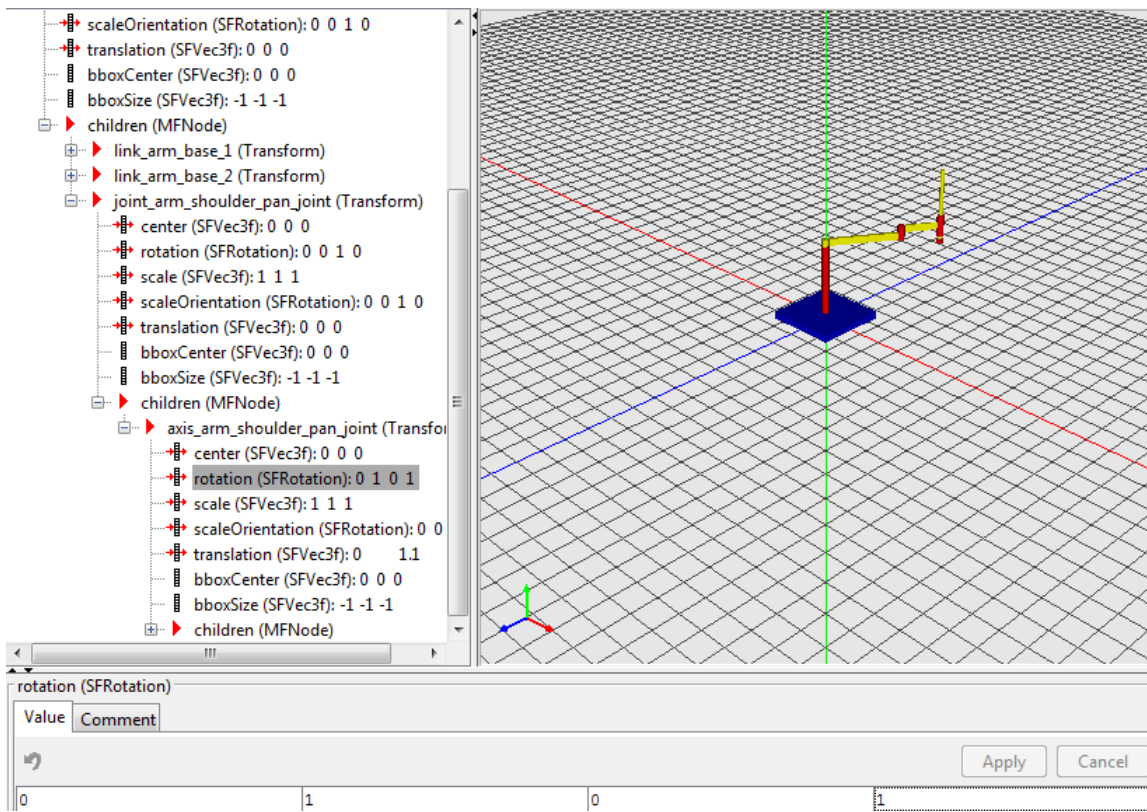
The example assumes that you have

- 1 Download the `simple_arm` model folder (including all its contents) from this Gazebo model site: [http://models.gazebosim.org/simple\\_arm/](http://models.gazebosim.org/simple_arm/). To download all the folder contents, download and unpack the `model.tar.gz` file.
- 2 From the MATLAB Toolstrip, in the **Apps** tab, in the **Simulation Graphics and Reporting** section, click **3D World Editor**.
- 3 In the 3D World Editor, select **File > New from Template**.
- 4 In the file dialog box, select the `Grids` folder and then select the `XZGrid.wrl` virtual world file.
- 5 Save the virtual world in the current folder. Select **File > Save as** and name the virtual world file `simple_arm.wrl`.
- 6 Import the SDF model. Select **Nodes > Import From > SDF File**. In the `simple_arm` folder, select the `model.sdf` file.
- 7 Open the `SDF_simple_arm_Transform` node and set the `scale` field to 3 for each dimension.



- The import process creates in the model hierarchy some special Transform nodes that make it easier to manipulate models. Each rotational joint in the SDF model is represented with a Transform node with the name starting with axis\_. These nodes have rotation axes defined so that the model elements move consistently if you set the rotation angle to a different value.

Change the rotation angle of a rotational joint node. For example, expand the children node, then the joint\_arm\_shoulder\_pan\_joint node, then the children, and finally, the axis\_arm\_shoulder\_pan\_joint. For the rotation field, set the fourth element to 1.



**Tip** If you associate this virtual world with a VR Sink block in a Simulink model, you can use the VR Signal Expander block to set only the rotation angle (the fourth element of the rotation vector) of the `axis_xxx Transform` node rotation. Leave the first three elements of the vector unchanged. This approach preserves robot consistency with the way model authors defined movement of robot parts.

## Define Viewpoint to Make Imported Model Visible

When you import a model, often it does not appear in the initial view of the virtual world. If necessary, create a viewpoint to make the imported model visible when the virtual world opens. If the imported model is not visible:

- 1 In the 3D World Editor, select the R00T node.
- 2 Select **Nodes > Add > Bindable > Viewpoint**.
- 3 To observe the imported model, navigate to a suitable location in the scene.
- 4 Populate viewpoint properties with current camera settings. In the tree hierarchy pane, right-click the viewpoint and select **Copy values from current camera**.
- 5 Save the virtual world.

## Limitations

URDF import transforms robot hierarchy of links and joints into a VRML or X3D hierarchy of transforms. It imports only visualization properties from each link. Joints are represented by

transforms that preserve rotation axes of the original revolute joints and axis alignment of prismatic joints.

SDF import transforms the hierarchy of SDF model objects into VRML or X3D hierarchy of transforms the same way as URDF import. It supports a subset of SDF format objects and is limited to visual objects under the Model element hierarchy. Supported Geometry elements are geometric primitives and meshes.

## See Also

### Functions

`stl2vrml` | `vrimport`

## Related Examples

- “Use CAD Models with the Simulink 3D Animation Product” on page 5-36
- “Import STL and Physical Modeling XML Files” on page 5-38
- “Import 3D Models from CAD Tools” on page 5-40
- “Modify the CAD Model Virtual World” on page 5-51
- “Link to Simulink and Simscape Multibody Models” on page 5-60

## Link to Simulink and Simscape Multibody Models

### In this section...

- “Link the Virtual World to a Simulink Model” on page 5-60
- “Initial Conditions” on page 5-61
- “VR Placeholder and VR Signal Expander Blocks” on page 5-62
- “Link to Simscape Multibody Models” on page 5-62
- “Link to a MATLAB Model” on page 5-63

To establish a live data connection between the model and the virtual world, create associations between dynamic model object quantities and corresponding virtual world object properties. For example, create associations with virtual world object properties such as position and rotations.

Although Simscape Multibody and Simulink are common platform for modeling mechanical systems, also you can use the Simulink 3D Animation product for the visualization of models implemented in MATLAB.

### Link the Virtual World to a Simulink Model

You associate Simulink model signals to virtual world object properties through the VR Sink block from the Simulink 3D Animation block library, `vrLib`.

To associate a Simulink signal to a virtual object property:

- 1 From the `vrLib` library, insert a VR Sink block into your Simulink model.
- 2 To define the virtual world, use the VR Sink block parameters dialog box. Enter the name of the virtual world 3D file in **Source file**, or click **Browse** to select the file interactively. To load the selected virtual reality scene, click **Apply**.
- 3 For smooth visualization of the movement, you can change the block **Sample time**. For example, to update the virtual world 25 times per simulation second, set the **Sample time** to `0.04`. Be careful when using the inherited sample time for the VR Sink block. Depending on the solver used, using inherited sample time can result in nonequidistant (in simulation time) updating of the virtual world. Nonequidistant updating gives a false impression of system dynamics to the person viewing the virtual world.
- 4 In **Virtual World Tree**, expand the main object **Transform** branch. In the scene object hierarchy, locate all parts you want to control from Simulink according to their names as given in “Add DEF Names” on page 5-51. Named **Transform** nodes represent each part. Select the check box next to the rotation and position fields of the **Transform** node. You can select other properties of virtual world objects, such as color, but rotations and positions are the ones most frequently controlled.
- 5 Click **OK**. For each selected field, the VR Sink block creates an input port. Increase the VR Sink block size as appropriate to accommodate the number of input ports.

After you associate the VR Sink block with a virtual world, you can double-click it to open the Simulink 3D Animation viewer. To access the block parameters, in the viewer, select **Simulation > Block Properties**.

VR Sink inputs take signals of the type corresponding to their virtual world representation. Position inputs are of type `SFVec3f`, which is the position represented in  $[x \ y \ z]$  coordinates. Rotation



inputs are of type `SFRotation`, the four-element vector defining rotation as `[axis angle]`, using the coordinate system described in “Coordinate System Used” on page 5-41, where the angle value is in radians.

Match the coordinate system used by the Simulink model to that of the virtual world. If the two coordinate systems are not identical, transform an axis.

Usually, object positions are available in the form required by virtual world (Cartesian coordinates). Often, object rotations are defined using the rotation matrix representation. To convert such rotations into the VRML format, use the `Rotation Matrix to VRML Rotation` block.

Object positions and rotations are treated differently depending on the virtual world hierarchy:

- To define all parts in a Simulink model in global coordinates, when the virtual world has a flat structure of independent objects, use these positions and rotations.

Object positions	Send to VR Sink all positions in global coordinates.
Object rotations	Send to VR Sink all rotations in global coordinates, with the center of rotation defined as the coordinate system origin. If the default center of rotation of <code>Transform</code> objects is <code>[0 0 0]</code> , you do not need to define the center for each part in the virtual world 3D file.

- When all parts in Simulink model follow hierarchical relations, and the virtual world has a nested structure, use these positions and rotations.

Object positions	Send to VR Sink all positions in local coordinates relative to their parents or predecessors in the object hierarchy. For example, send the robot tool position relative to the robot hand.
Object rotations	<p>Send to VR Sink all rotations in local coordinates relative to their parents or predecessors in the object hierarchy. For example, send the robot tool rotation relative to the robot hand.</p> <p>Match the positions of joints between objects visually by coinciding the centers of rotation in the virtual world and in the Simulink model. Coincide the center of rotation because when joints between parts are not positioned in the origin (<code>[0 0 0]</code>) of the coordinate system of the parent.</p> <p>To define a center of rotation different from the default value, <code>[0 0 0]</code>, define the <b>center</b> field of the child <code>Transform</code> node in the virtual world 3D file. For example, define the robot tool center of rotation to coincide with the joint connecting the hand and the tool in the hand local coordinates.</p>

In a hierarchical scene structure, when the parts are connected by revolving joints, it is easy to define the relative rotations between parts. The joint axis directly defines the virtual world rotation axis, so you can construct the `[axis angle]` four-element rotation vector.

## Initial Conditions

A Simulink model initial conditions must correspond to the initial object positions and rotations defined in the virtual world. Otherwise, the object controlled from Simulink jumps from the position

defined in the VRML file to the position dictated by Simulink for the simulation. To compensate for this offset, use one of these approaches:

- In the virtual world 3D file, define another level of nested `Transform` around the controlled object.
- In the Simulink mode, add the object initial position to the model calculations before sending to the VR Sink block.

Align the Simulink model initial conditions with the virtual world object positions, Maintain the correct position of the object relative to the surrounding scene. You can adjust the position of the surroundings of the object. For example, move the road position so that the car at position `[0 0 0]` stays on the road, without the wheels sinking or floating above the road.

## VR Placeholder and VR Signal Expander Blocks

The VR Sink block accepts only inputs that define fully qualified field values. Dynamic models that describe the system behavior in only one dimension still require full 3D positions for all controlled objects for their virtual reality visualization.

To simplify the modeling in such cases, you can use the VR Placeholder and VR Expander blocks of the Simulink 3D Animation library.

The VR Placeholder block sends out a special value that is interpreted as unspecified by the VR Sink block. When this placeholder value appears on a VR Sink input, as a single value or as vector element, the appropriate value in the virtual world remains unchanged.

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 control the position of a virtual object in a one-dimensional dynamic model, use the VR Signal Expander block with the controlled dimension as its input. For its output, use a three-component vector in the VR Sink block. The remaining vector elements are filled with placeholder signals.

Use of the VR Signal Expander block is also a possibility when defining rotations. When the axis of rotation is defined in the virtual world 3D file, you can send to the VR Sink block a virtual world rotation value. Use a value consisting of three placeholder signals and the computed angle. This rotation value forms a valid four-element `[axis angle]` vector.

## Link to Simscape Multibody Models

You can use the Simulink 3D Animation product to view the behavior of a model created with the Simscape Multibody software.

- 1 Build a model of a machine in the Simulink interface using Simscape Multibody blocks.
- 2 Create a detailed visual representation of your machine in a virtual world.
- 3 Connect the virtual world to the Simscape Multibody body sensor outputs.
- 4 View the behavior of the bodies in a virtual world viewer.

You can use the Simscape Multibody software for 3D visualizations using the Simulink 3D Animation product. In addition to the features that Simscape Multibody product offers for modeling mechanical assemblies, the following features simplify the visualization of Simscape Multibody models in virtual reality:

- Simscape Multibody and virtual world coordinate systems are identical.
- In the Simscape Multibody software, you can work with both global and local object coordinates. This flexibility makes it easy to adapt the model to the structure of the virtual world exported from the CAD tool.

The Simscape Multibody product also offers a convenient way of importing CAD assembly designs into Simscape Multibody machines through the Simscape Multibody Link interface. Alternatively, when you export a CAD assembly to the virtual world format, you can add virtual reality visualization to such assemblies.

The Simulink 3D Animation software includes the following functions for working with Simscape Multibody files: `vrcadcleanup`, `vrphysmod`, and `stl2vrml`.

Depending on the virtual world hierarchy, you can use one of two approaches to help visualize Simscape Multibody machines:

- When the virtual world has a flat structure of independent objects, you can obtain the positions and rotations of machine parts using Body Sensor blocks. Connect the Body Sensor block to appropriate coordinate systems attached to the bodies. Define positions and rotations using global coordinates. Usually, you can connect the sensor to a body coordinate system with origin at  $[\theta \ 0 \ 0]$  and with an initial rotation matrix defined as the identity matrix,  $[1 \ 0 \ 0; \ 0 \ 1 \ 0; \ 0 \ 0 \ 1]$ , in the global coordinates.
- When the virtual world has a hierarchical structure of nested objects, you can obtain the body positions and rotations. Use a Body Sensor block with its output set to use local body coordinates. In special cases, such as when two bodies are connected through a revolving joint, you can get the angle between the objects using a Joint Sensor block.

## Link to a MATLAB Model

To help you to interact with virtual worlds, the Simulink 3D Animation product offers a set of MATLAB functions and constructs referred to collectively as its “MATLAB interface.” Circumstances when this MATLAB functionality is appropriate for use with CAD-based designs include:

- Using customized GUIs to visualize static objects and their relations in a virtual environment, such as in interactive machine assembly instructions.
- Visualizing 3D information based on an independent quantity (not necessarily time).
- Using MATLAB interface functions in Simulink model callbacks.
- Visualizing systems whose dynamic models are available as MATLAB code.
- Visualizing systems where massive object changes, such as deformations, take place. In this case, send dynamically sized matrix-type data from the dynamic models to virtual worlds, which is not possible using just Simulink signals.

For information on setting object properties using the MATLAB interface, see “Interact with Virtual Reality Worlds”.

## See Also

## Related Examples

- “Use CAD Models with the Simulink 3D Animation Product” on page 5-36

- “Import STL and Physical Modeling XML Files” on page 5-38
- “Import 3D Models from CAD Tools” on page 5-40
- “Import VRML Models from CATIA Software” on page 5-45
- “Modify the CAD Model Virtual World” on page 5-51
- “Interact with Virtual Reality Worlds”

# Using the 3D World Editor

---

- “3D World Editor” on page 6-2
- “Open the 3D World Editor” on page 6-5
- “Create a Virtual World” on page 6-9
- “Edit a Virtual World” on page 6-11
- “Reduce Number of Polygons for Shapes” on page 6-20
- “Virtual World Navigation in 3D World Editor” on page 6-21
- “3D World Editor Library” on page 6-26

## 3D World Editor

In this section...
“Supported Platforms” on page 6-2
“Use with Other Editors” on page 6-2
“VRML and X3D Support” on page 6-2
“Nodes, Library Objects, and Templates” on page 6-2

The 3D World Editor is a native VRML and X3D editor.

For an example that shows how to see the 3D World Editor to create a virtual world, see “Build and Connect a Virtual World” on page 5-8.

### Supported Platforms

The 3D World Editor works on all supported platforms for the Simulink 3D Animation product.

The 3D World Editor is installed as part of the Simulink 3D Animation installation. It is the default virtual world editor.

### Use with Other Editors

As you create a virtual world, you can use a different editor for individual phases of the process.

Choose an editor that best meets your needs. For a description of the benefits and limitations of different types of editors, see “Choose a Virtual World Editor” on page 5-2.

### VRML and X3D Support

The file formats for the 3D World Editor are VRML and X3D.

The 3D World Editor supports all VRML97 types and language elements, except as noted.

For general VRML limitations relating to the Simulink 3D Animation software as a whole, see “VRML Compatibility” on page 1-12.

For general X3D limitations relating to the Simulink 3D Animation software as a whole, see “X3D Support” on page 1-9.

#### PixelTexture Nodes

You cannot create or edit PixelTexture node image contents. Existing PixelTexture node image contents are fully preserved

### Nodes, Library Objects, and Templates

Use the 3D World Editor to specify VRML or X3D to create 3-D virtual worlds that you can connect to a Simulink model.

Use the 3D World Editor to:

- To create a virtual world, assemble nodes. You can add nodes that specify many aspects of a virtual world, such as:
  - Appearance (for example, font style, color, and material)
  - Navigation information (for example, navigation mode and headlights)
  - Geometry (for example, boxes, text, and elevation grids)
  - Groups (for example, transforms)
  - Interpolators
  - Light
  - Sensors
- Select objects from a set of supplied libraries or from custom libraries for:
  - Components (for example, geometric objects, backgrounds, aircraft, vehicles, landscapes, and architecture)
  - Materials
  - Textures
- Use a supplied template as a starting point for a virtual world.

### Template Virtual World 3D Files

The 3D World Editor includes template virtual world 3D files that you can use as a starting point for creating virtual reality worlds. Some examples of templates are the Earth, road, sea, and terrain virtual world templates.

To access templates, use one of the following approaches:

- Select **File > New From Template**.
- Select the **New File From Template** button .

A template file name displayed in the 3D World Editor always starts with **Template:.**

To adapt the template world for your application, edit the file. To save your changes, use the **File > Save As** option. You cannot overwrite an existing template file.

You can create your own template files. Store them in a different folder than the folder used for template files provided with Simulink 3D Animation.

In virtual worlds that you create, you can reference nodes, such as texture files, that appear in the template files provided with Simulink 3D Animation.

## See Also

### Functions

`vredit` | `vrgetpref` | `vrsetpref`

### Related Examples

- “Edit a Virtual World” on page 6-11
- “Set the Default Editor” on page 5-5

- “Open the 3D World Editor” on page 6-5
- “Create a Virtual World” on page 6-9
- “Reduce Number of Polygons for Shapes” on page 6-20
- “Virtual World Navigation in 3D World Editor” on page 6-21
- “Workflow for Building and Using Virtual Worlds” on page 1-4



## Open the 3D World Editor

### In this section...

“3D World Editor Is the Default Editor” on page 6-5

“Open an Empty Virtual World” on page 6-5

“Open a Saved Virtual World” on page 6-5

“3D World Editor Panes” on page 6-6

“Preferences for 3D World Editor Startup” on page 6-7

### 3D World Editor Is the Default Editor

When you install the Simulink 3D Animation product, the 3D World Editor is configured to be the default editor. For details about changing the default editor, see “Set the Default Editor” on page 5-5. For an overview of the 3D World Editor, see “3D World Editor Panes” on page 6-6.

**Note** You can also use the V-Realm Editor. For more information, see “V-Realm Builder Help” on page 2-15.

### Open an Empty Virtual World

Use one of these approaches to open an empty virtual world in the 3D World Editor.

- From the Simulink Toolstrip, in the **Apps** tab, in the **Simulation Graphics and Reporting** section, click the app icon.
- From the MATLAB Toolstrip, in the **Apps** tab, in the **Simulation Graphics and Reporting** section, click **3D World Editor**.
- If the 3D World Editor is your default virtual world editor, open an empty virtual world from the MATLAB command line using the `edit` command.
 

```
edit(vrworld(''))
```
- Regardless of **Default Editor** preference setting, you can use the `vredit` command, without arguments.
- From within the 3D World Editor, select either **File > New** or **File > New From Template**. If there is already a file open in the 3D World Editor, these options open a new instance of the editor. Use multiple instances of the editor to work on multiple virtual worlds at the same time and to copy and paste from one virtual world to another.

### Open a Saved Virtual World

Use one of these approaches to open a saved virtual world.

- From the MATLAB Current Folder panel, right-click a virtual world 3D file and from the context menu, select **Edit**
- If 3D World Editor is your default virtual world editor, start it from the MATLAB command line using the `edit` command. For example:

```
edit(vrworld('myVRMLfile.wrl'))
```

- Regardless of what your default virtual world editor is, from the MATLAB command line, use the `vredit` command with the name of the virtual world 3D file. For example:

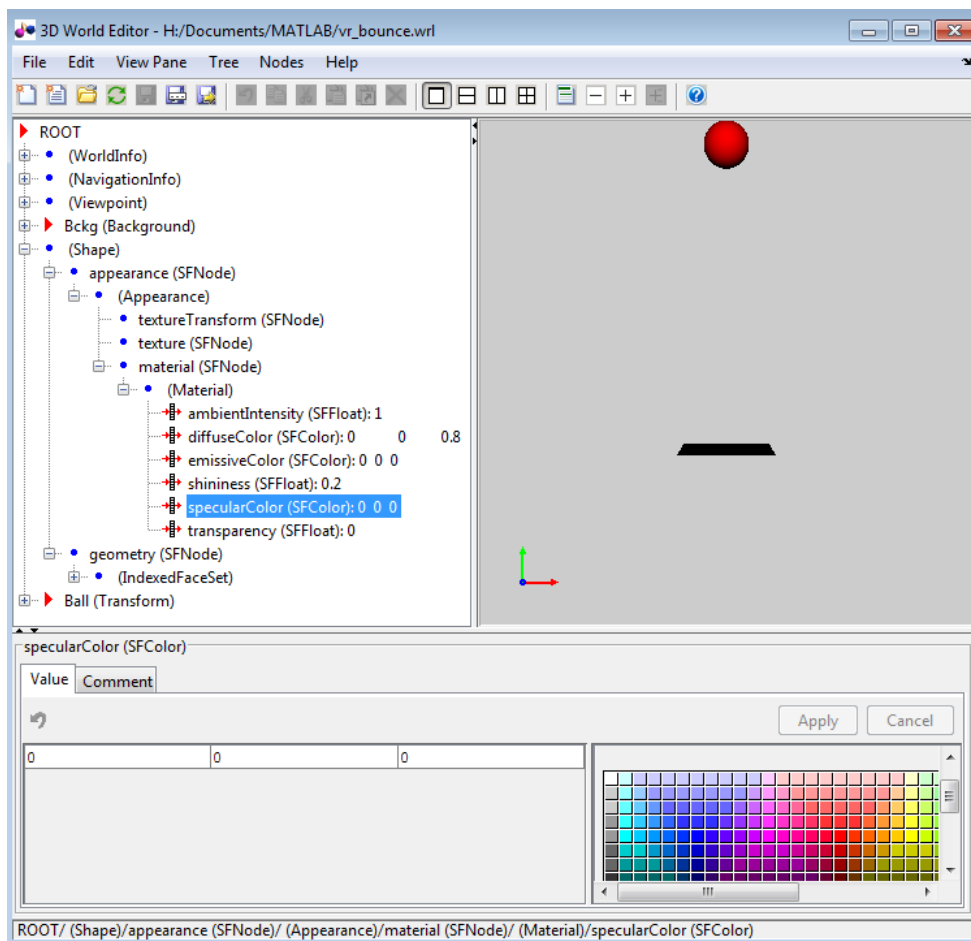
```
vredit('membrane.wrl')
```

- From within the 3D World Editor, select **File > Open**. If a file is already open in the 3D World Editor, this option opens a new instance of the editor.

## 3D World Editor Panes

The 3D World Editor provides three panes:

- **Tree structure** pane — View the hierarchy for the virtual world that you are editing.
- **Virtual world display** pane — Observe the virtual world as you create it.
- **Object property edit** pane — Change values for node items.









Use the **tree structure** pane interactively to create graphical virtual world elements and to view of all the virtual world structure elements present in the virtual world. These structure elements are called nodes. The 3D World Editor lists the nodes and their properties according to their respective virtual world node types. In the tree viewer, you give the nodes unique names.

Use the **virtual world display** pane to display a graphical representation of a 3-D scene.

Use the **object properties edit** pane to edit a selected property or add a comment to a selected node or property.

### Tree Structure Pane Icons

The **Tree structure** pane displays icons to help you visually distinguish node field types.

Node Field Type	3D World Editor Icon
field	
eventIn	
eventOut	
exposedField	
ROUTE	
USE	

## Preferences for 3D World Editor Startup

The **Simulink 3D Animation Preferences > 3D World Editor** pane includes the following options for specifying the startup position:

- For the default location, select **Position**. Then specify the pixel location for the lower-left corner, the width, and the height (for example, [96 120 862 960]).
- To open the 3D World Editor in the same location where you exited it, select **Save position on exit**.

You can specify whether the editor starts up with the default virtual world display layout or with the layout as it was when you exited it previously. The saved layout includes settings for the view, viewpoints, navigation, and rendering. Simulink 3D Animation saves the layout in a separate virtual world 3D file for up to eight files.

By default, the virtual world opens with the layout saved at exit. To have the virtual world open using the default layout, clear the **Preferences > Simulink 3D Animation > 3D World Editor > Preserve Layout per Virtual Reality 3D File** check box.

## See Also

### Functions

`vredit` | `vrgetpref` | `vrsetpref`

## Related Examples

- “3D World Editor” on page 6-2
- “Edit a Virtual World” on page 6-11
- “Set the Default Editor” on page 5-5
- “Set Simulink 3D Animation Preferences” on page 2-5
- “Create a Virtual World” on page 6-9

- “Workflow for Building and Using Virtual Worlds” on page 1-4

## Create a Virtual World

Creating a virtual world involves several tasks. You can use the 3D World Editor throughout the process of building a virtual world, and you can perform the tasks in many different ways.

For a step-by-step tutorial about building a virtual world using the 3D World Editor, see “Build and Connect a Virtual World” on page 5-8.

Here is an example of a common workflow for creating a virtual world using the 3D World Editor. This example workflow includes optional tasks and a small subset of the types of tasks that you can perform. For more information, see “Edit a Virtual World” on page 6-11.

- 1 Open a new virtual world 3D file.
- 2 Under the ROOT node, optionally add:
  - A `WorldInfo` node to document the virtual world.
  - A `NavigationInfo` node to define overall navigation characteristics of the virtual world, such as the Avatar size.
- 3 Under the ROOT node, add a `Transform` node in the virtual world for each object that you want to share properties with other object in that same `Transform` node.
- 4 Under the `Transform` node, include nodes in a hierarchy, such as:

```
children
  Shape
    appearance
      Appearance
        material
          Material
            texture
            textureTransform
          Geometry
            Box
```

- 5 Use the **object properties edit** pane to change default property values to create the effects that you want.
- 6 To define aspects (such as textures) for virtual world objects, insert 3D World Editor library objects. Give a `DEF` name to each object that you create, so that you can access them using Simulink 3D Animation.
- 7 In the **virtual world display** pane, use the context menu to specify display characteristics, such as:
  - View characteristics (for example, zooming and a navigation panel)
  - Viewpoints
  - Navigation characteristics (for example, methods (such as fly or walk) and speed)
  - Rendering techniques (for example, antialiasing, lighting, and transparency)
- 8 Save or export the virtual world 3D file.

### See Also

#### Functions

`vredit` | `vrgetpref` | `vrsetpref`

## **Related Examples**

- “Open the 3D World Editor” on page 6-5
- “Virtual Reality World and Dynamic System Examples” on page 1-16
- “Create vrworld Object for a Virtual World” on page 4-2

## Edit a Virtual World

### In this section...

“Add Objects” on page 6-11  
 “Copy and Paste a Node” on page 6-12  
 “Edit Object Properties” on page 6-13  
 “Document a Virtual World Using Comments” on page 6-14  
 “Display Event Fields” on page 6-14  
 “Expand and Collapse Nodes” on page 6-14  
 “Highlight Nodes and Virtual World Objects” on page 6-15  
 “Wrap Nodes as Children of Another Node” on page 6-16  
 “Remove Nodes” on page 6-17  
 “Save and Export Virtual World 3D Files” on page 6-17  
 “Edit VRML and X3D Scripts” on page 6-18

For information about opening a file in the editor, see “Open the 3D World Editor” on page 6-5.

For a step-by-step tutorial, see “Build and Connect a Virtual World” on page 5-8.

## Add Objects

Add virtual world objects (for example, the wing of an airplane) by adding nodes in the **tree structure** pane. The hierarchy of nodes controls the scope to which node properties apply.

---

**Note** Nodes must have unique names to work in the Simulink 3D Animation product.

---

### Approaches for Adding Objects

Use one of these approaches to add a node.

Approach	Procedure
Use the <b>Nodes</b> menu	<ol style="list-style-type: none"> <li><b>1</b> In the <b>tree structure</b> pane, select the parent node for the object that you want to add.</li> <li><b>2</b> Select <b>Nodes &gt; Add</b>.</li> <li><b>3</b> To add the node that you want, select appropriate submenus.</li> </ol>
Use a context menu for a node	<ol style="list-style-type: none"> <li><b>1</b> In the <b>tree structure</b> pane, right-click the parent node for the object that you want to add.</li> <li><b>2</b> To add the node that you want, select the <b>Add Node</b> menu, and then select appropriate submenus.</li> </ol>

Approach	Procedure
Insert an object from a library	For Material, Texture, and children nodes, select the <b>Insert From</b> menu option (from either the <b>Nodes</b> menu or from the context menu for a node).  For information about library objects, see “3D World Editor Library” on page 6-26.
Add an inlined virtual world 3D file	For a ROOT or children node, from the <b>Nodes</b> menu or the context menu for the node, select the <b>Inline Virtual Reality 3D File</b> menu item.  You can inline VRML files (.wrl) files, but not X3D files (.x3d or .x3dv).

The node that you add gets added to different locations in the hierarchy, depending on the node that you select to begin adding a node.

Selected Node	Location of Added Node
ROOT	At the bottom of the hierarchy
Node at the next level down from the ROOT node (for example, a Transform node).	Above the selected node
A children node	Under the children node (as a child node of the selected node)

## Copy and Paste a Node

You can copy a node below a top-level Transform node and paste that copied node to be a child of another node, including the ROOT node.

You can paste the copied node as either an explicit text copy (**Paste**) or as a referenced copy (**Paste As Reference**).

- An explicit text copy allows you to edit properties of that node, independently from the original node that you copied.
- A referenced copy node appears with the term USE. Referenced copies streamline the **tree structure** pane display. Edits that you make to the original (referenced) node are applied to the copied node, ensuring that the two nodes remain exact copies of each other.

To copy and paste a node:

- 1 In the **tree structure** pane, select the node that you want to copy.
- 2 Copy the node, using *one* of these techniques:
  - Select **Edit > Copy**.
  - Right-click the node and select **Copy**.
- 3 Under the appropriate node, paste the copied node.
  - Paste the node using *one* of the following techniques:
    - Select the **Edit > Paste** or the **Paste As Reference** menu item.



- Right-click the parent node and select **Paste Node**, and then select **Paste** or **Paste As Reference**.

### Copy and Paste Between Virtual Worlds

In the same editing session, you can copy nodes from a virtual world in one virtual world 3D file to another virtual world in a different file. After you copy the nodes from one virtual world, select **File > Open** to open the second file where you want to paste the nodes.

### Edit Object Properties

To define the characteristics of an object, in the **tree structure** pane, select the appropriate property. At the bottom of the 3D World Editor, use the **object properties edit** pane to change property values. Then click **Apply**.

The **tree structure** pane shows the current property values, which reflect your edits.

When you enter a numeric field value in the 3D World Editor, you can use MATLAB expressions and MATLAB variables. For example, to convert an angle from degrees to radians, enter a MATLAB expression such as  $25*\pi/180$ .

### Set Viewpoint Values Using Camera Position

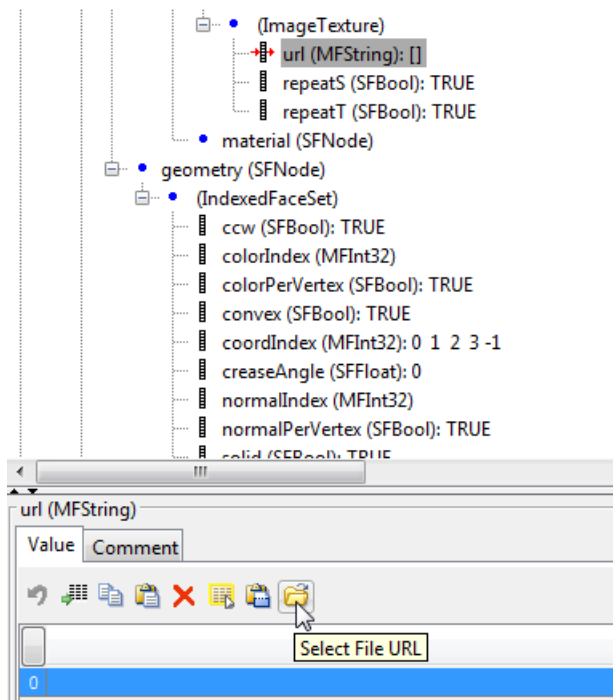
You can use the current camera position to specify interactively a viewpoint in the 3D World Editor.

- 1 Navigate to the position in the scene where you want the viewpoint.
- 2 In the **tree structure** pane, right-click a Viewpoint node.
- 3 Select **Copy values from current camera**.

### Specify a URL

For objects that have a URL field, to specify a URL, select the node and use one of these approaches.

- In the property edit box for the URL, enter the URL.
- Select the 0 on the left side of the property edit box and click the **Select File URL** button. Navigate to the file.




## Document a Virtual World Using Comments

To document a virtual world, in the **object property edit** pane, use the **Comments** tab for nodes and properties. Comments can help others understand the design of a virtual world.

Comments do not appear in the virtual world. They appear in the virtual world 3D file, next to the given node or property, on lines that begin with #.

## Display Event Fields

You can display eventIn and eventOut fields in the 3D World Editor tree pane. Either click the **Show Events** button  or select **Tree > Show Events**.

You can perform an IS mapping for events in a PROTO declaration.

## Expand and Collapse Nodes

To expand a node in the **tree structure** pane, click the plus (+) sign to the left of the node. To collapse a node, click the minus (-) sign to the left of the node.

To expand or collapse all nodes in one step, select **Tree > Expand All** or **Tree > Collapse All**.

To expand subtrees within a node:

- 1 In the tree structure pane, right-click a node.
- 2 From the context menu, select **Expand Subtree**.

Alternative approaches for expanding the subtree for a node are:


- Select **Tree > Expand Subtree**.
- Click the  button.

### Hide Default Values

To simplify the tree view, you can hide default values. Select **Tree > Hide Default Values**. To display default values, clear the **Hide Default Values** option.

### Highlight Nodes and Virtual World Objects

To select and highlight virtual world objects using the mouse pointer in the 3D World Editor view pane, use the select mode. Use that mode to highlight a node that defines a virtual world object or to highlight a virtual world object that a node defines. In the 3D World Editor display pane, the selected virtual world object is highlighted with an orange outline. For example:

- 1 Open the vrtkoff virtual world and select **File > Open in editor**.
- 2 In the 3D World Editor toolbar, click the  button.

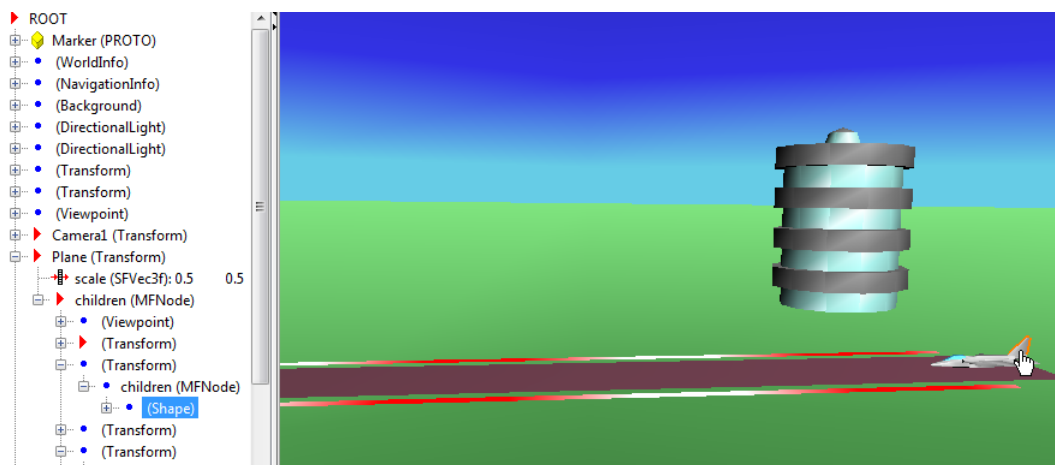
---

**Tip** Alternatively, you can select the **View Pane > Select** menu item.

---

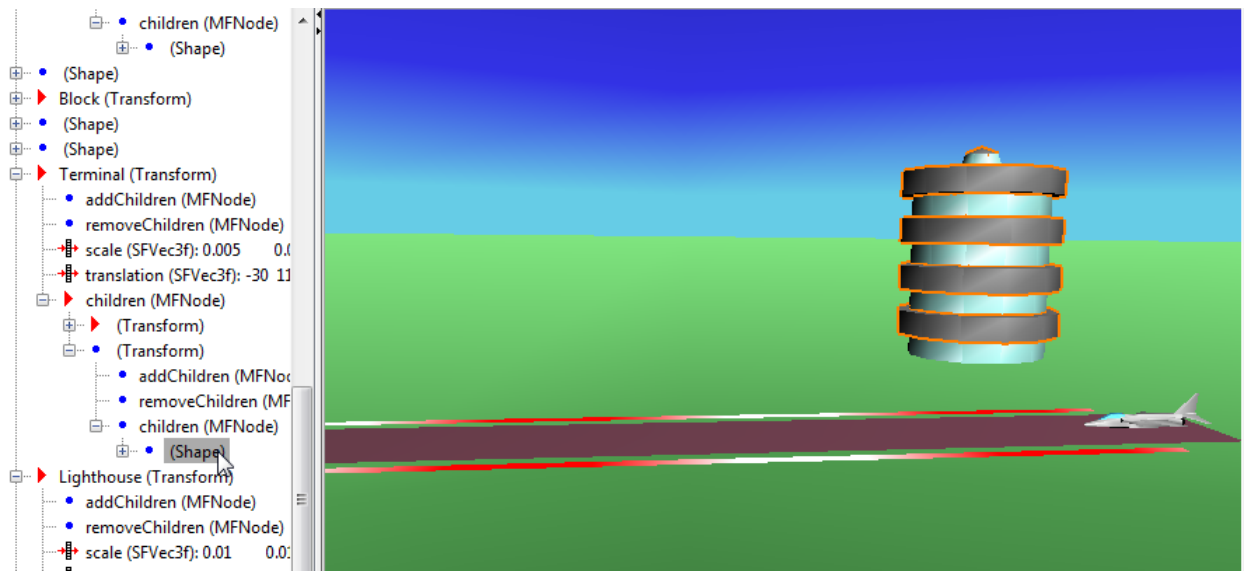
When the cursor hovers over a selectable object in the view pane, the cursor shape changes to a hand symbol.

- 3 In the virtual world display pane, click the tail of the plane.



The corresponding Shape node in the tree structure pane is highlighted.


- 4 In the tree structure pane, click in the Tower (Transform) node, select the bottom Shape node. The floors of the tower are highlighted.




---

**Tip** If you select a node in the tree structure pane but do not see anything highlighted in the virtual world display pane, adjust the viewing angle to make the highlighted object visible.

---

- 5 To return to navigation mode, in the 3D World Editor toolbar, click the  button.

---

**Tip** Alternatively, you can select the **View Pane > Navigate** menu item.

---

You can use **Shift**-click to select multiple objects in the tree structure pane or view pane.

If you do not want objects to be highlighted when you pick them, in the display pane right-click and clear the **Rendering > Highlight Selected Objects** option. Alternatively, you can use the **F4** key.

---

**Note** Compound virtual world objects, such as objects defined using `Inline` and `PROTO` nodes, are selected and highlighted as a whole. You cannot select individual components of these objects.

---

### Selection and Highlighting Preferences

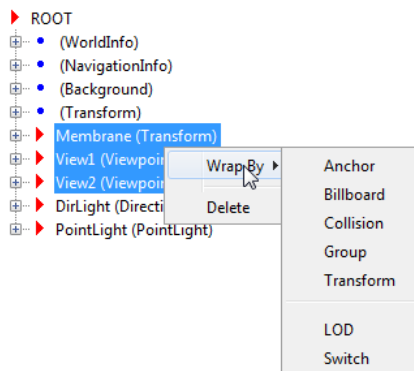
By default, the cursor in view pane navigates in the virtual world. To have the mouse cursor in the view pane behave in select mode by default, set the Simulink 3D Animation 3D World Editor preference **View pane mouse behavior** preference to `select`. As an alternative, use the `DefaultEditorMouseBehavior` parameter with the `vrsetpref` command.

By default, virtual world objects are highlighted when you select them using select mode. To have the default behavior for selected virtual world objects be to not highlight the objects, set the Simulink 3D Animation 3D World Editor preference **Highlight selected objects** preference to `off`. As an alternative, use the `DefaultEditorHighlighting` parameter with the `vrsetpref` command.

### Wrap Nodes as Children of Another Node

To wrap contiguous nodes as children of another node:

- 1 Select the nodes. You can use the **Shift** key to select contiguous nodes, and the **CTRL** key to select discontinuous nodes.
- 2 Right-click the selected nodes and from the context menu, select **Wrap By**.  
As an alternative, on the 3D World Editor menu bar, select **Nodes > Wrap By**.
- 3 From the list of nodes, select the node in which you want to wrap the selected nodes.



## Remove Nodes

To delete one or more nodes, select the nodes and use one of these methods:

- On the toolbar, click the red X button.
- Click the **Delete** button.
- Select **Edit > Delete**.
- Right-click the node and select **Delete**.

From the **Edit** menu, you can also delete a specific child node or all the children nodes of a selected parent node, without deleting the parent node.

To cut a node and save it to the clipboard, select the node and use one of these techniques:

- On the toolbar, click the scissors button.
- Select **Edit > Cut**.
- Right-click the node and select **Cut**.

## Save and Export Virtual World 3D Files

You can save your virtual world files as virtual world using the **File > Save** or **File > Save As** menu items.

If you use the **Save** option, the 3D World Editor renames the previous version of the file by appending **.bak** after the **.wrl**, **.x3dv**, or **.x3d** extension.

If you use the **Save As** option, the 3D World Editor saves the file using the new name that you specify. The file is saved in a form that the Simulink 3D Animation Viewer and 3D World Editor support (for example, the saved file preserves links to the library texture files).

Use the **File > Export** menu item to export a virtual world 3D file for use:

- With other VRML or X3D tools
- On different computers
- In previous versions of the Simulink 3D Animation (previously the Virtual Reality Toolbox) product (for VRML files)

---

**Note** You cannot save an X3D file (.x3d or .x3dv) file as a VRML (.wrl) file.

---

For exported files, the 3D World Editor copies referenced inlined virtual world 3D files and texture files to the <filename>\_files folder. It modifies the corresponding URLs for those files, so that they point to the <filename>\_files folder.

## Edit VRML and X3D Scripts

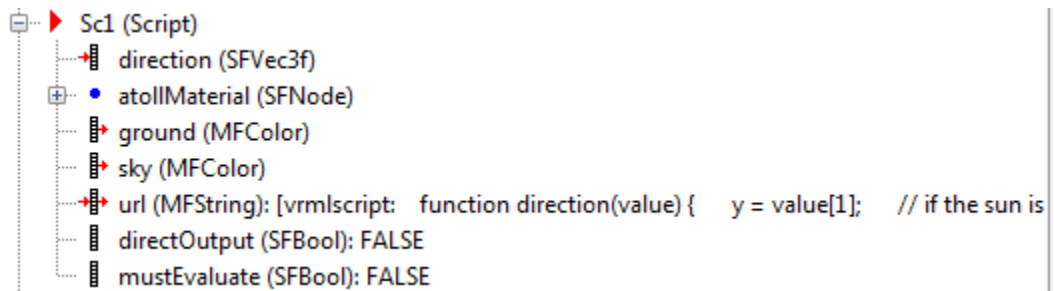
To add a VRML or X3D Script node:

- 1 In the **Tree structure** pane, select the ROOT node.
- 2 Select the appropriate type of script, using the **Node > Add > Common > Script** menu.

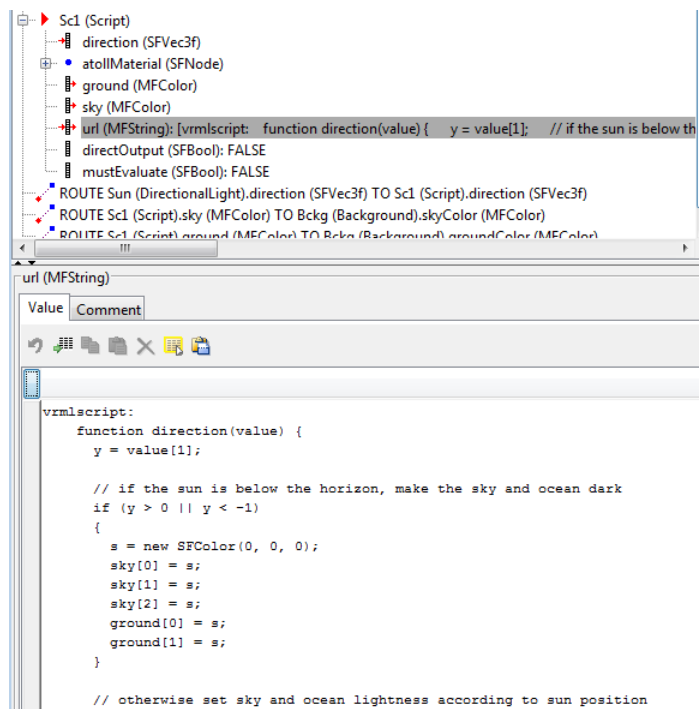
To add Script interface elements:

- 1 Right-click a Script node.
- 2 Select the appropriate **Add Interface Item** menu option.

The following is an example of a Script node in the **Tree structure** pane.



For a url node, click the node and either specify the path to a JavaScript® file or enter the URL code in the **Object property edit** pane.



## See Also

### Functions

vredit | vrgetpref | vrsetpref

## Related Examples

- “3D World Editor Library” on page 6-26
- “Set the Default Editor” on page 5-5
- “Open the 3D World Editor” on page 6-5
- “Create a Virtual World” on page 6-9
- “Reduce Number of Polygons for Shapes” on page 6-20
- “3D World Editor Library” on page 6-26
- “Virtual World Navigation in 3D World Editor” on page 6-21
- “Workflow for Building and Using Virtual Worlds” on page 1-4

## Reduce Number of Polygons for Shapes

For a node that is, or includes, an `IndexedFaceSet` node, you can improve rendering speed by reducing the number of polygons in the `IndexedFaceSet`. Choose the polygon reduction factor that produces your desired balance of rendering performance and quality.

To reduce the number of polygons:

- 1 Right-click an `IndexedFaceSet` node, or a node that includes one or more `IndexedFaceSet` nodes.
- 2 Select **Optimize Child Geometries**.
- 3 In the Geometries Optimization Preview dialog box, use the slider or enter a value in the **Polygon reduction factor** field to set the polygon reduction factor. A value of 0 performs the maximum reduction, and a value of 1 performs no reduction.
- 4 Click **OK**.

### See Also

### Related Examples

- “Edit a Virtual World” on page 6-11



## Virtual World Navigation in 3D World Editor

### In this section...

“Specify Virtual World Rendering” on page 6-21

“Basic Navigation” on page 6-21

“Coordinate Axes Triad” on page 6-21

“View Panes” on page 6-22

“Pivot Point” on page 6-23

“View All/View Selected” on page 6-23

**Note** Navigation in the 3D World Editor generally is the same as navigation in the Simulink 3D Animation Viewer

. For example, you can use stereoscopic viewing with the 3D World Editor. For more information about viewer navigation, see “Simulink 3D Animation Viewer”.

### Specify Virtual World Rendering

You can control the rendering used in the **virtual world display** pane of the 3D World Editor. Right-click in the virtual world display pane and set rendering options, such as antialiasing, lighting, and transparency.

### Basic Navigation

Use the **virtual world display** pane to visualize the virtual world as you create it.

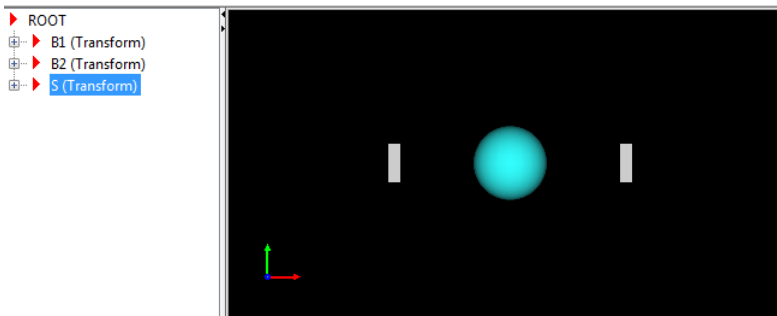
To control navigation, right-click anywhere in the pane and select the appropriate navigation options. You can control aspects such as navigation methods (for example, fly, or walk) and speed.

To save these navigation settings in a virtual world file, define the navigation properties in a `NavigationInfo` node.

To navigate in the virtual world, left-click in the pane. The cursor changes to a white cross hair. Moving the cursor changes the orientation of the virtual world.

### Coordinate Axes Triad

To help you visualize changes in the orientation (coordinate axes) of nodes in a virtual world, the **virtual world display** pane includes a triad of red, green, and blue arrows. These arrows are parallel with global x (red arrow), y (green arrow), and z (blue arrow) coordinate axes. As you navigate in a virtual world, the triad display changes to reflect changes in orientation.



To hide the triad for a virtual world, or to change the location of the triad in the **virtual world display** pane, right-click in the pane and select the appropriate option from the **View > Triad** menu.

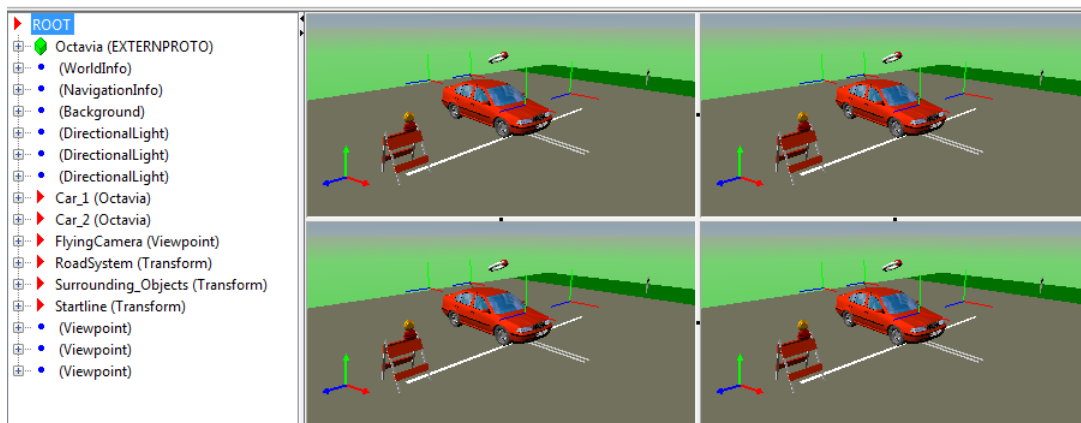
To change the default location or visibility of the triad:

- 1 From the MATLAB Toolstrip, in the **Home** tab, in the **Environment** section, select **Preferences**.
- 2 In the Preferences dialog box left pane, select **Simulink 3D Animation > 3D World Editor > Triad**.

## View Panes

You can view the virtual world in one pane, two horizontal panes, two vertical panes, or four panes.

For example, if you select **View Pane > Grid View**, the 3D World Editor displays four view panes:



You can also use toolbar buttons to set view panes:



When the grid view panes are open and you change the view pane option, the editor displays these view panes from the grid view.

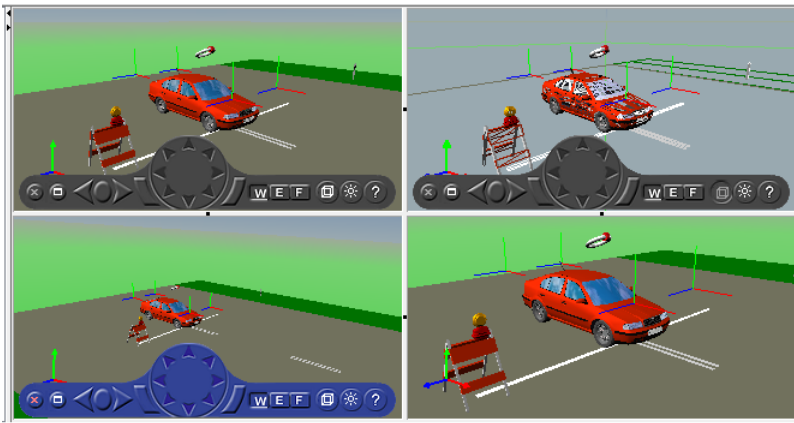
New View Pane Option	Grid View Panes Displayed
Horizontal Split	Left two panes
Vertical Split	Top two panes

New View Pane Option	Grid View Panes Displayed
Single View	Top left pane

To change the relative size of a pane, select one of the gray borders for a view pane and drag the cursor.

To delete a pane when there are multiple panes, click the dot in the border between view panes.

You can control the navigation of each pane independently. For example, you can set viewpoints, manipulate the triad, and specify rendering techniques independently for each pane. The navigation panel in the active pane is blue. For example:



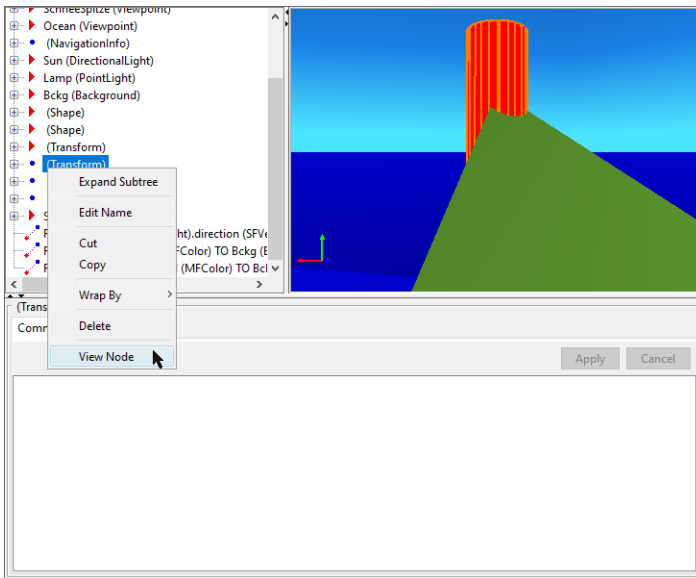
## Pivot Point

To use a mouse to rotate a virtual world around a point, In Examine mode, you can use a pivot point.

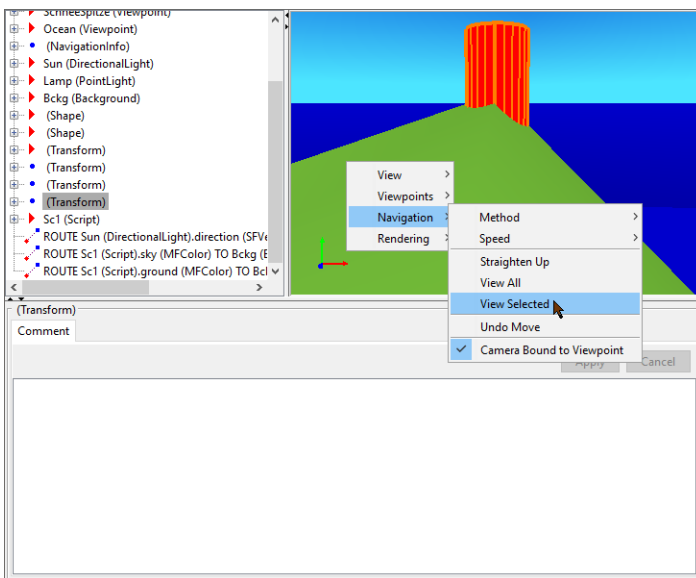
The default pivot point is  $0, 0, 0$  (world coordinates). To set a new pivot point, hold **Ctrl** and double-click the spot where you want the pivot point.

## View All/View Selected

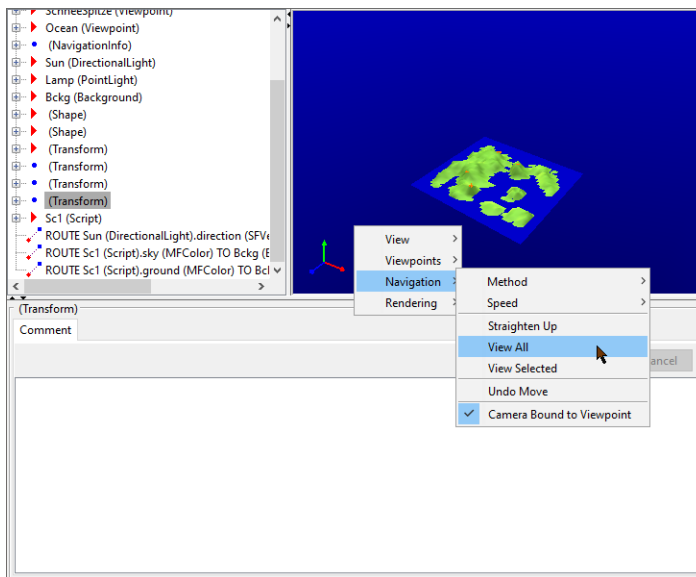
To bring a specific node into view in the **3D World Editor**, right-click its entry in the **Tree Structure** pane and select **View Node**.



You can also use the context menu in the **Virtual world display** pane to bring a selected node into view. Right-click the **Virtual world display** pane and select **Navigation > View Selected**.



To bring the entire world into view, right-click anywhere in the **Virtual world display** pane and select **Navigation > View All**.



## See Also

### Functions

`vrredit` | `vrgetpref` | `vrsetpref`

### Related Examples

- “3D World Editor” on page 6-2
- “Set Simulink 3D Animation Preferences” on page 2-5
- “Simulink 3D Animation Viewer”

## 3D World Editor Library

In this section...
"3D World Editor Library Objects" on page 6-26
"Add a Library Object" on page 6-26
"Guidelines for Using Custom Objects" on page 6-27

### 3D World Editor Library Objects

The 3D World Editor includes a library of virtual world objects, which you can insert into a virtual world. The library consists of component, texture, and material objects. These library objects supplement the default empty nodes available via the **Nodes** menu or the **Insert From** context menu item. The library objects are predefined with specific nodes and property settings to represent common virtual world effects.

Examples of component objects in the library include:

- Aircraft
- Vehicles
- Backgrounds
- Shapes

Examples of texture objects are:

- Surface of Earth
- Surface of the moon
- Water
- Surfaces of building materials

Examples of material objects are:

- Colored plastics
- Colored metals
- Sand

After you add a library object to a virtual world, you can modify its nodes and properties.

### Add a Library Object

To add a 3D World Editor library object to a virtual world:

- 1 Select the parent node under which you want to insert the library object.
- 2 Use one of these techniques to access the library objects:
  - Select the **Nodes > Insert From** menu item.
  - Right-click the parent node and select the **Insert From** menu item.

The set of objects displayed depends on the parent node that you select. For example, if you select a child node under a **Transform** node, you can choose among **Component** sublibrary

objects. To add a material object, select an `Appearance` or `Material` node or material field. To add a texture library object, select an `Appearance` node or texture field.

- 3 To find the object that you want to insert, follow the folder paths.

You can also insert objects from other locations, using the **Insert From > Other Location** menu item.

- The first `Transform` node of the file that you select from another location is inserted in place into the tree view of the virtual world that you are editing.
- If you want to insert a whole virtual world 3D file into the virtual world that you are editing, use the **Nodes > Inline Virtual Reality 3D file** menu option.

Before you add a custom library object from a location other than from the 3D World Editor object library, see “Guidelines for Using Custom Objects” on page 6-27.

## Guidelines for Using Custom Objects

If you use a VRML or X3D object from a source other than from the 3D World Editor object library, the object must comply with these rules.

- A component object must be in a file that contains at least one `Transform` node.
- A material object must be a file whose only content is a fully qualified `Material` node.
- A texture object must be in either:
  - A `.png`, `.jpg`, or `.gif` graphics file for use in the URL field of an `ImageTexture` node.
  - A file whose only content is a fully qualified `ImageTexture` node.

If you create VRML or X3D objects to use with the 3D World Editor, create your own folder for storing the custom objects. Avoid using the 3D World Editor library folder. Use your own folder so that you can:

- Edit the custom library object in the library folder; the 3D World Editor library folders are generally read-only.
- Update to a future version of the Simulink 3D Animation product without compatibility issues relating to mixing custom objects with the 3D World Editor objects.

## See Also

### Related Examples

- “Edit a Virtual World” on page 6-11
- “3D World Editor” on page 6-2





# Viewing Virtual Worlds

---

- “Virtual World Viewers” on page 7-2
- “Simulink 3D Animation Viewer” on page 7-4
- “Open the Simulink 3D Animation Viewer” on page 7-7
- “Simulate with the Simulink 3D Animation Viewer” on page 7-8
- “Specify Rendering Techniques” on page 7-9
- “Navigate Using the Simulink 3D Animation Viewer” on page 7-15
- “Set Viewpoints” on page 7-23
- “Navigate Through Viewpoints” on page 7-26
- “Record Offline Animations” on page 7-29
- “Play Animations with Simulink 3D Animation Viewer” on page 7-35
- “Configure Frame Capture Parameters” on page 7-36
- “Capture Frames” on page 7-37
- “Simulink 3D Animation Web Viewer” on page 7-38
- “Open the Web Viewer” on page 7-39
- “Navigate Using the Web Viewer” on page 7-41
- “Listen to Sound in a Virtual World” on page 7-43
- “View a Virtual World in Stereoscopic Vision” on page 7-45
- “Active Stereoscopic Vision Configuration” on page 7-47

## Virtual World Viewers

### In this section...

“Host and Remote Viewing” on page 7-2

“Comparison of Viewers” on page 7-2

After you create a virtual world in VRML or X3D (see “Build Virtual Reality Worlds”), you can visualize that virtual world with a virtual world viewer or HTML5-enabled web browser.

The Simulink 3D Animation product includes three viewers.

- Simulink 3D Animation Viewer (the default viewer)
- Simulink 3D Animation Web Viewer
- Orbisnap viewer

To determine which viewer to use, see “Comparison of Viewers” on page 7-2.

### Host and Remote Viewing

You can view a virtual world on your host computer, with either the Simulink 3D Animation Viewer, Orbisnap, or a web browser. For web browser support, Simulink 3D Animation provides the Simulink 3D Animation Web Viewer.

In most configurations, you do not need to install a viewer on a client computer because you can perform all the tasks on a host computer. However, if you have large models that consume considerable computational resources, you can use a client computer to run and view the virtual world.

To view a virtual world on a client computer, which does not need to have Simulink 3D Animation installed, you can use the Web Viewer, or Orbisnap. For remote viewing, use the Web Viewer or Orbisnap.

### Comparison of Viewers

Feature	Simulink 3D Animation Viewer	Simulink 3D Animation Web Viewer	Orbisnap
Platforms	Windows, Macintosh, and Linux	Windows, Macintosh, and Linux	Windows, Macintosh, and Linux
Web browser		HTML5-enabled web browser	
Installation	Installed with product	Host interface installed with product	Separate installation
Simulink 3D Animation required	Yes		
Remote viewing		Yes	Yes
Viewpoint creation	Yes		
Animation start/stop	Yes		Starts automatically

Feature	Simulink 3D Animation Viewer	Simulink 3D AnimationWeb Viewer	Orbisnap
Simulation start/stop	Yes		
Sound	Yes		Yes
Stereoscopic viewing	Yes		Yes

## See Also

### Functions

vrgetpref | vrsetpref

### Related Examples

- “Set the Default Viewer” on page 2-2
- “Set Simulink 3D Animation Preferences” on page 2-5
- “Open the Simulink 3D Animation Viewer” on page 7-7
- “Specify Rendering Techniques” on page 7-9
- “Navigate Using the Simulink 3D Animation Viewer” on page 7-15
- “Navigate Through Viewpoints” on page 7-26
- “Simulate with the Simulink 3D Animation Viewer” on page 7-8
- “Record Offline Animations” on page 7-29
- “Capture Frames” on page 7-37
- “Play Animations with Simulink 3D Animation Viewer” on page 7-35
- “Install V-Realm Editor” on page 2-14

### More About

- “Simulink 3D Animation Viewer” on page 7-4
- “Set Viewpoints” on page 7-23
- “Simulink 3D Animation Web Viewer” on page 7-38

## Simulink 3D Animation Viewer

In this section...
“What You Can Do with the Viewer” on page 7-4
“Viewer Uses MATLAB Figures” on page 7-5
“Set Viewer Appearance Preferences” on page 7-6

### What You Can Do with the Viewer

This section provides an overview of the features and controls of the viewer.

This section uses the `vrpend`, `vrplanets`, and `vrtut1` examples to illustrate the viewer features:

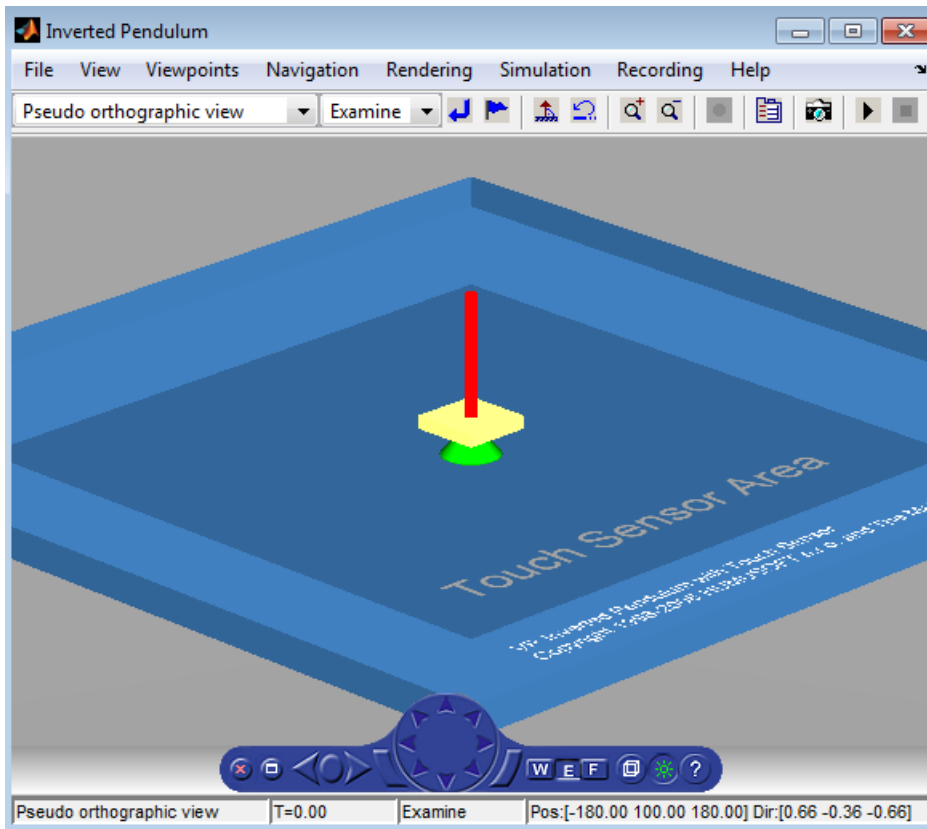
- 1 Select a Simulink 3D Animation example and type the example name in the MATLAB Command Window. For example:

```
vrpend
```

The Simulink model is displayed. By default, the Simulink 3D Animation viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the VR Sink block in the Simulink model.

- 2 Inspect the viewer window.

The Simulink 3D Animation viewer displays the virtual scene. The top of the viewer contains a menu bar and toolbar. By default, the Simulink 3D Animation viewer displays the virtual scene with a navigation panel at the bottom. These three areas of the viewer give you alternate ways to work with the virtual scene.




---

**Note** The Simulink 3D Animation viewer settings are saved when you save your model file.

---

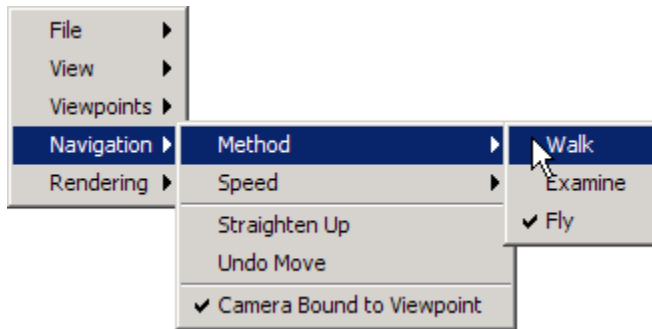
## Viewer Uses MATLAB Figures

The Simulink 3D Animation software contains a viewer as the default method for viewing virtual worlds. You can use this viewer on any supported operating system. This viewer uses MATLAB figures. It provides several capabilities, including:

- Treat the viewer window like a MATLAB figure. This ability enables you to perform MATLAB figure actions such as dock the viewer window in the MATLAB window. For example:



- To display a context menu that contains the viewer commands, right-click in the viewer window. For example:



- Combine multiple virtual reality viewer figures in several tiles of a MATLAB figure.

### Set Viewer Appearance Preferences

You can configure the appearance of the viewer using Simulink 3D Animation preferences. For details, see “Figure Appearance Preferences Dialog Box” on page 2-8.

### See Also

#### Functions

`vrgetpref` | `vrsetpref`

### Related Examples

- “Open the Simulink 3D Animation Viewer” on page 7-7
- “Set Simulink 3D Animation Preferences” on page 2-5

### More About

- “Virtual World Viewers” on page 7-2

## Open the Simulink 3D Animation Viewer

In this section...
“Open from the VR Sink Block” on page 7-7
“Open from the Command Line” on page 7-7

### Open from the VR Sink Block

You can open the Simulink 3D Animation Viewer by double-clicking a VR Sink block in the Simulink Editor.

You can configure a VR Sink block to automatically open the Simulink 3D Animation Viewer.

- 1 In the Simulink Editor, double-click the VR Sink block to open the Simulink 3D Animation Viewer.
- 2 From the Simulink 3D Animation Viewer **Simulation** menu, select **Block parameters**.
- 3 Select **Open Viewer automatically**.
- 4 Click **OK**.

### Open from the Command Line

Use `vrview`. For example, to open the `vrbounce` virtual world in the Simulink 3D Animation Viewer, use `vrview('vrbounce')`.

### See Also

#### Functions

`vrgetpref` | `vrsetpref`

### Related Examples

- “Set the Default Viewer” on page 2-2
- “Set Simulink 3D Animation Preferences” on page 2-5

### More About

- “Virtual World Viewers” on page 7-2

## Simulate with the Simulink 3D Animation Viewer

You can start and stop simulations of the virtual world from the Simulink 3D Animation viewer through the menu bar, toolbar, or keyboard.

- From the menu bar, select the **Simulation** menu **Start** or **Stop** option.
- From the toolbar, click **Start/pause/continue simulation** or **Stop simulation**.
- From the keyboard, press **Ctrl+T** to toggle between starting or stopping the simulation.

---

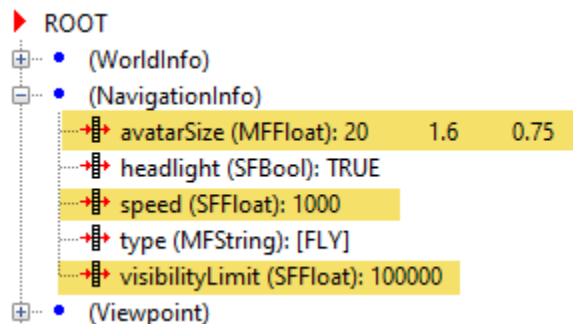
**Note** The **Ctrl+T** operation is available only if you started the viewer from a Simulink model. If you start the viewer through the MATLAB interface, no Simulink model is associated with the viewer. You cannot start and stop the simulation in this case.

---

### Adjust Navigation Settings

During simulation of a model, objects in the associated virtual world displayed in the Simulink 3D Animation can disappear as the objects get closer to the viewer or go away from the user. In the virtual world, you can adjust how the near and far clipping planes (which control when objects disappear as they move toward or away from the viewer) are calculated. In the `NavigationInfo` node, you can adjust the `avatarSize` and `VisibilityLimit` fields to adjust the clipping planes so that the objects are visible during the simulation.

For example, suppose the virtual world has a static large scene with no close object in view, and the simulation causes an airplane to move nearby from the left to the right of your view. This example shows some of the `NavigationInfo` fields that you can adjust:



### See Also

### Related Examples

- “Open the Simulink 3D Animation Viewer” on page 7-7

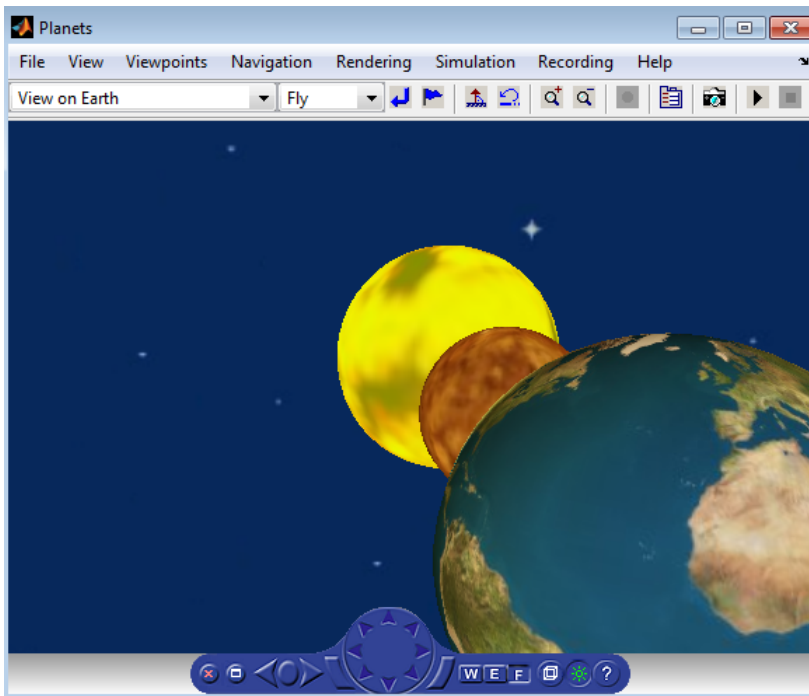


## Specify Rendering Techniques

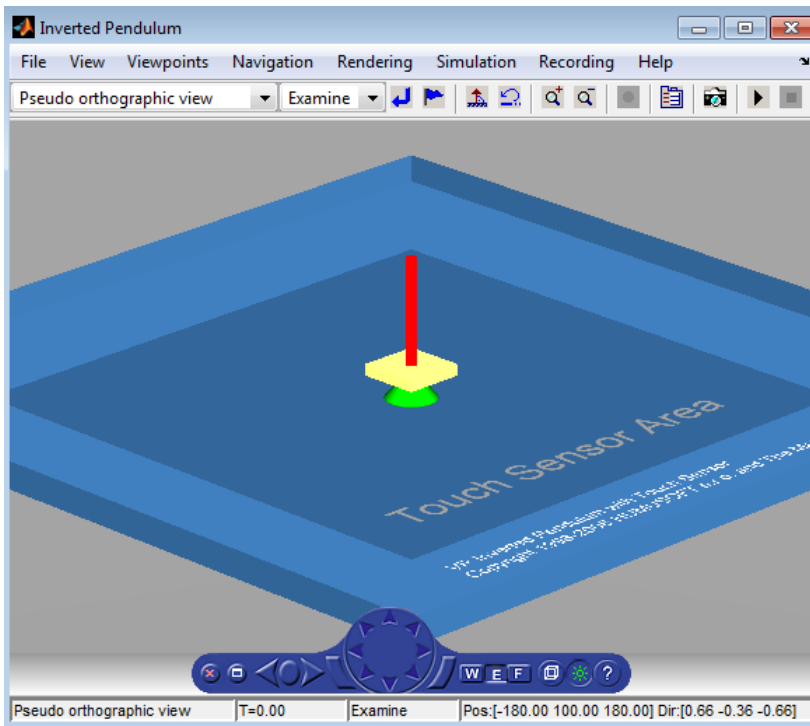
You can change the rendering of the scene through the controls on the navigation panel or options on the rendering menu. The `vrpend` and `vrplanets` examples are used to show the viewer functionality.

You can turn the antialiasing of the scene on or off. Antialiasing applies to the textures of a world. Antialiasing is a technique that attempts to smooth the appearance of jagged lines. These jagged lines are the result of a printer or monitor not having enough resolution to represent a line smoothly. When **Antialiasing** is on, the jagged lines are surrounded by shades of gray or color. Therefore, the lines appear smoother rather than jagged.

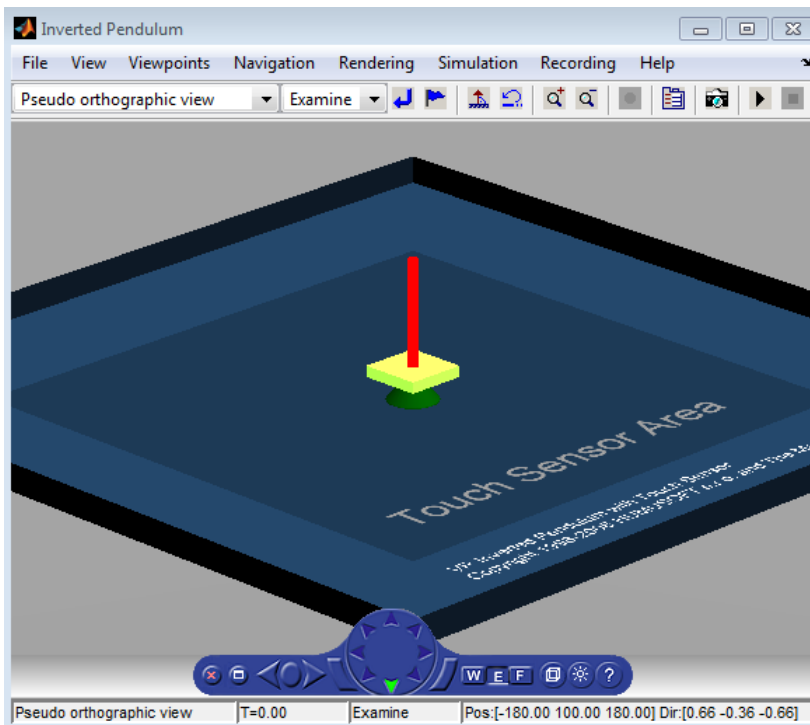
The following figure depicts the `vrplanets` example View on Earth viewpoint with **Antialiasing** on. To display the effects of antialiasing better, turn **Headlight** on. You can turn antialiasing on or off to observe the differences.



You can turn the camera headlight and the lighting of the scene on or off. When **Headlight** is off, the camera does not emit light. Therefore, the scene can appear dark. For example, the following figure depicts the `vrpend` example with **Headlight** on.

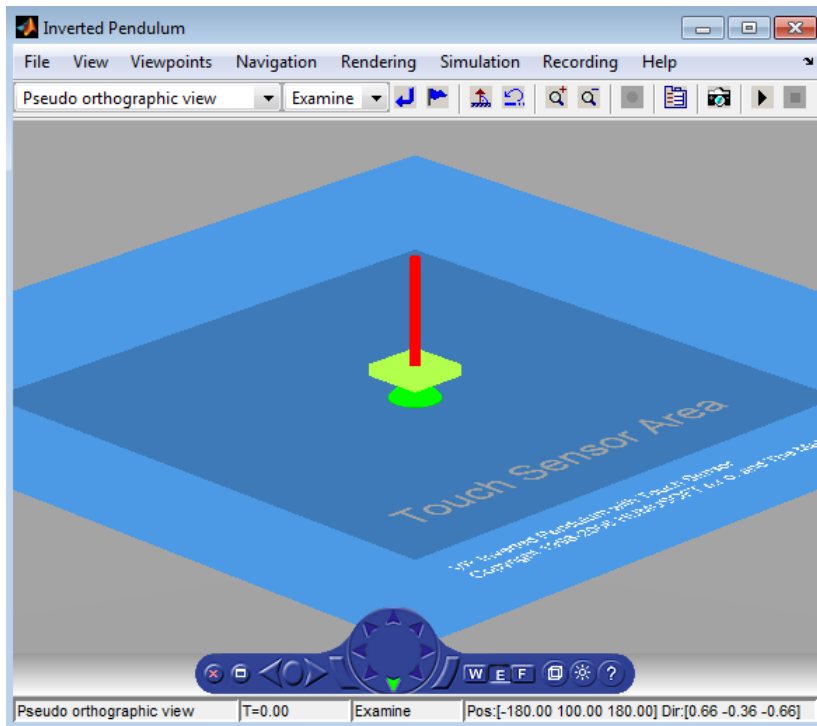


The scene looks darker when **Headlight** is set to off.

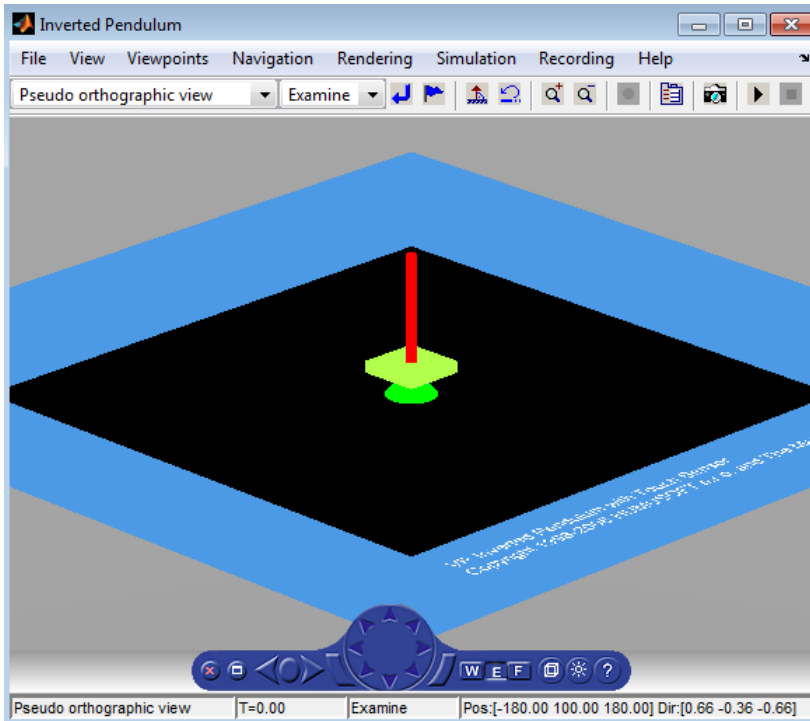


**Note** It is helpful to define enough lighting within the virtual scene so that it is lit regardless of the **Headlight** setting.

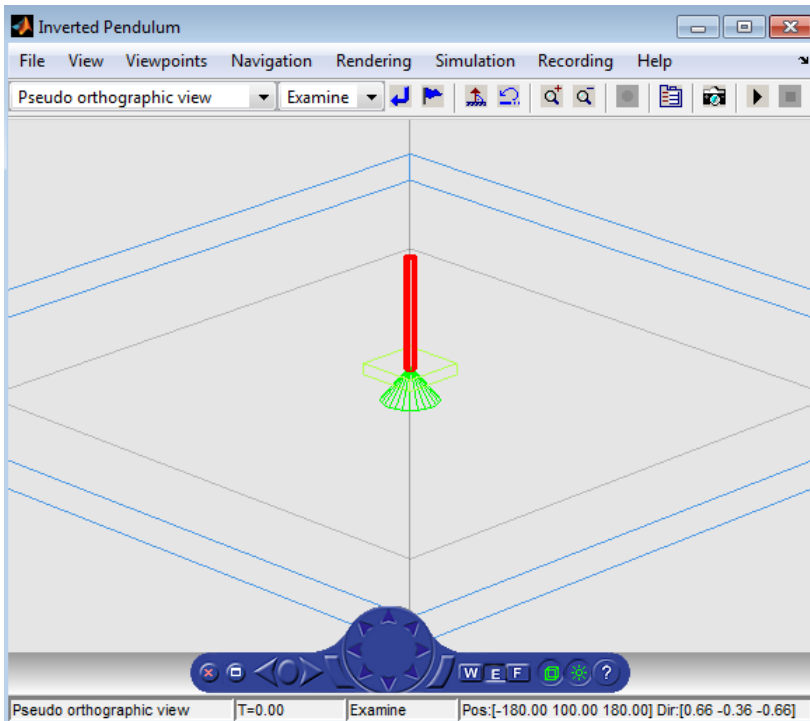
When **Lighting** is off, the virtual world appears as if lit in all directions. The Simulink 3D Animation viewer does not compute and render all the lighting effects at the surfaces of the objects. Shadows disappear and the scene loses some of its 3-D quality. The following is the `vrpend` example with **Lighting** off.



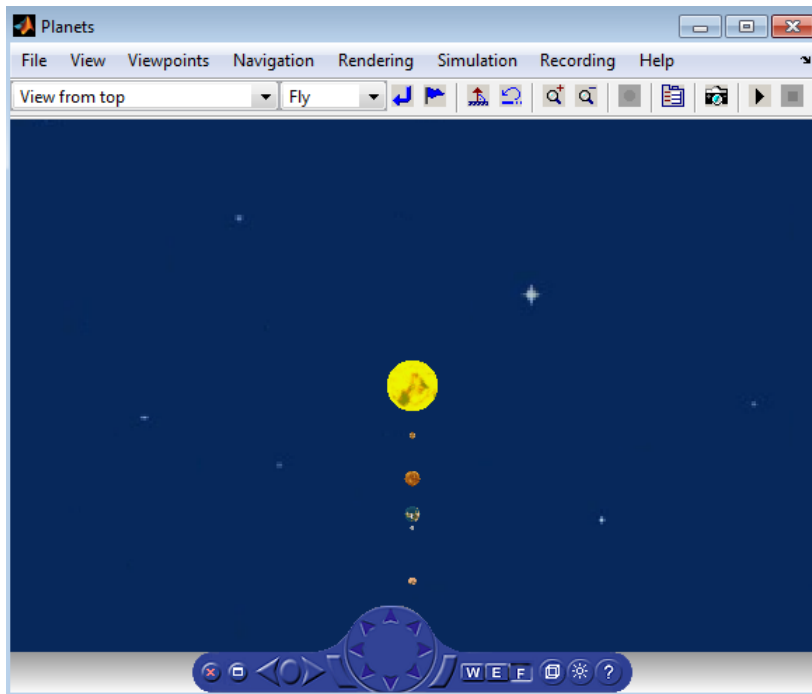
If **Transparency** is off, transparent objects are rendered as solid objects.



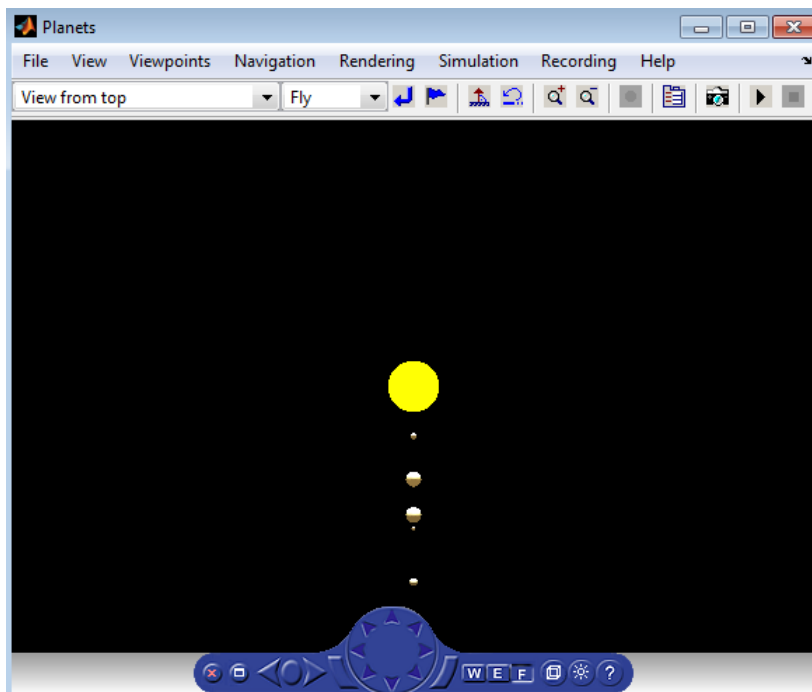
Turning **Wireframe** on changes the scene objects from solid to wireframe rendering. The following is the vrpend example with **Wireframe** on.



If the **Textures** option is on, objects have texture in the virtual scene. Here is the vrplanets example with **Textures** on:



If **Textures** is off, objects do not have texture in the virtual scene. The following is the `vrplanets` example with **Textures** off.



You can specify the maximum size of a texture used in rendering the `vrfigure` object. This option gives you a list of texture sizes to choose from. See the `vrfigure` `MaxTextureSize` property for further details.

## Turn Off Rendering for Performance

You can use the `Rendering` property for either a `vrfigure` or `vr.canvas` object to turn off rendering for the object in the Simulink 3D Animation Viewer. 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.

You can also use the new **Simulink 3D Animation > World > Default Figure Rendering** preference (`DefaultFigureRendering`) to specify whether to render a newly created virtual figure or canvas object.

## See Also

### Functions

`vr.canvas` | `vrgetpref` | `vrsetpref` | `vrworld`

## Related Examples

- “Set Simulink 3D Animation Preferences” on page 2-5

## More About

- “Virtual World Viewers” on page 7-2

## Navigate Using the Simulink 3D Animation Viewer

### In this section...

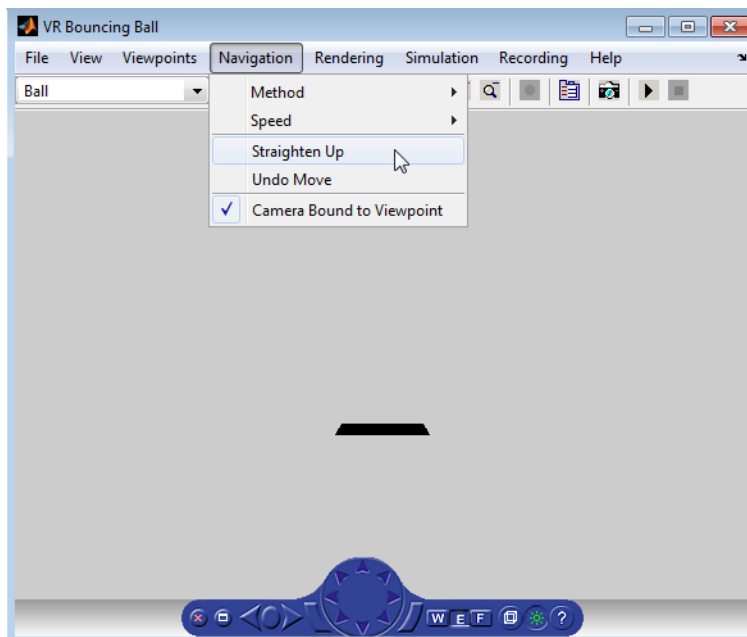
“Basic Navigation” on page 7-15  
 “Navigation Panel” on page 7-16  
 “Viewer Keyboard Shortcuts” on page 7-18  
 “Mouse Navigation” on page 7-18  
 “Navigation Control Menu” on page 7-19  
 “Change the Navigation Speed” on page 7-19  
 “Sensors Effect on Navigation” on page 7-20  
 “Display a Coordinate Axes Triad” on page 7-20  
 “Pivot Point” on page 7-21

### Basic Navigation

You can navigate in a virtual scene using the menu bar, toolbar, navigation panel, mouse, and keyboard. The `vrbounce` example illustrates some key features of the viewer.

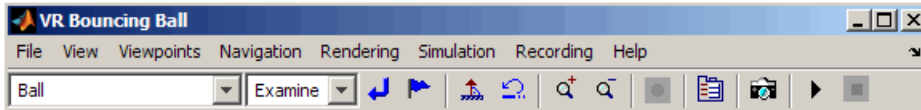
**Navigation Panel** — The center navigation wheel and two curved buttons on either side allow you to move about in the scene. Experiment by moving backward and forward and side to side.

**Navigation view** — You can change the camera position. From the menu bar, select the **Navigation** menu **Straighten Up** option. Alternatively, from the toolbar, click the **Straighten Up** control, or on the keyboard, you can press **F9**. This option resets the camera so that it points straight ahead.



**Navigation methods** — Navigation with the mouse depends on the navigation method that you select and the navigation zone that you are in when you first click and hold down the mouse button. You can set the navigation method using one of these approaches:

- From the menu bar, select the **Navigation** menu **Method** option. This option provides three choices: Walk, Examine, or Fly. See “Mouse Navigation” on page 7-18.
- From the toolbar, select the drop-down list that displays the navigation options Walk, Examine, and Fly.

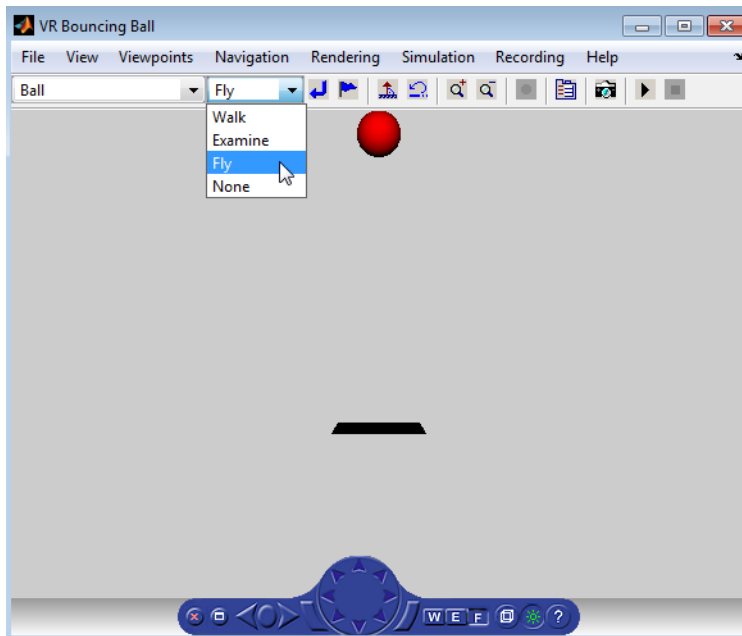


- From the navigation panel, click the **W**, **E**, or **F** buttons.
- On the keyboard, press **Shift+W**, **Shift+E**, **Shift+F**, or **Shift+N**.

**Navigation zones** — You can view the navigation zones for a scene by using the menu bar or keyboard.

From the menu bar, select the **View** menu **Navigation Zones** option. The virtual scene changes as the navigation zones are toggled on and appear in the virtual scene. Alternatively, on the keyboard, press the **F7** key.

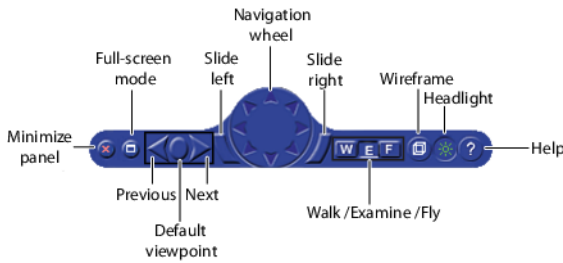
The vrbounce example with **Method** set to **Fly** has three navigation zones.



## Navigation Panel

The Simulink 3D Animation viewer navigation panel has navigation controls for some of the more commonly used navigation operations available from the menu bar.





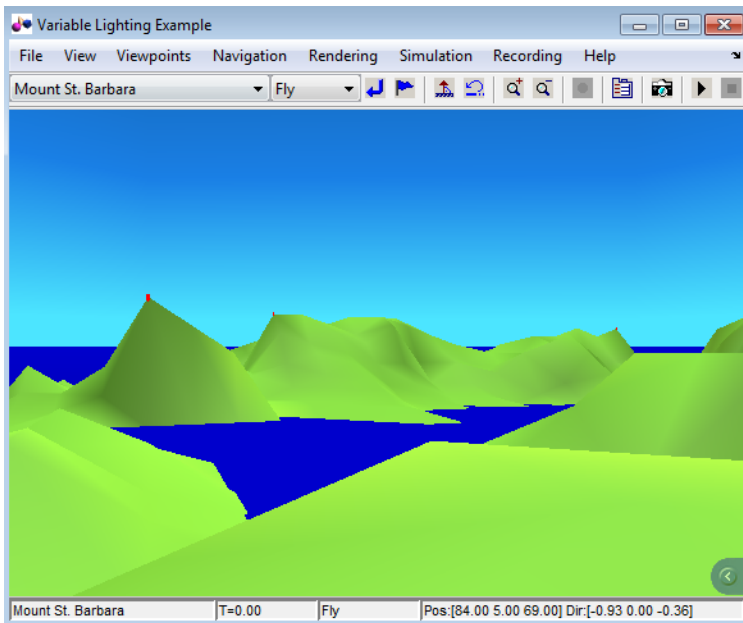
If you have multiple viewers open, the navigation panel in the active viewer is blue.

### Minimize the Navigation Panel

You can minimize the navigation panel using *either* of these approaches:

- Click the red x control on the left side of the navigation panel.
- Select **View > Navigation Panel > Minimized**

The minimized navigation panel appears as an icon in the lower right corner of the viewer.



To display the navigation panel again, click the **Show Panel** left arrow on the minimized navigation panel icon.

To minimize the navigation panel by default, from the MATLAB Toolstrip, set the **Preferences > Simulink 3D Animation > Canvas > Navigation panel** preference to minimized.

The minimized navigation panel is blue for the active viewer and gray for the inactive viewers.

## Viewer Keyboard Shortcuts

Navigation Function	Keyboard Shortcut
Use full-screen mode.	<b>Ctrl+f</b>
Undo move.	<b>Backspace</b>
Start or stop recording.	<b>Ctrl+r</b>
Capture frame.	<b>Ctrl+i</b>
Start or stop simulation.	<b>Ctrl+t</b>
Straighten up and make the camera stand on the horizontal plane of its local coordinates.	<b>F9</b>
Zoom in and out.	<b>+/-</b>
Toggle the headlight on and off.	<b>F6</b>
Toggle the navigation zones on and off.	<b>F7</b>
Toggle the wireframe option on and off.	<b>F5</b>
Toggle the antialiasing option on and off.	<b>F8</b>
Go to default viewpoint.	<b>Esc</b>
Return to current viewpoint.	<b>Home</b>
Go to previous viewpoint.	<b>Page Up</b>
Go to next viewpoint.	<b>Page Down</b>
Camera is bound/unbound from the viewpoint.	<b>F10</b>
Set the navigation method to Walk.	<b>Shift+w</b>
Set the navigation method to Examine.	<b>Shift+e</b>
Set the navigation method to Fly.	<b>Shift+f</b>
Move the camera forward and backward.	<b>Shift Up/Down Arrow</b>
Pan the camera up and down.	<b>Up/Down Arrow</b>
Pan the camera right and left.	<b>Left/Right Arrow, Shift+Left/Right Arrow</b>
Slide up and down.	<b>Alt+Up/Down Arrow</b>
Slide left and right.	<b>Alt+Left/Right Arrow</b>
Pressing <b>Ctrl</b> alone acquires the examine lock at the point of intersection between the line perpendicular to the screen, coming through the center of the viewer window, and the closest visible surface to the camera. Pressing the arrow keys without releasing <b>Ctrl</b> rotates the viewpoint about the acquired center point.	<b>Ctrl+Left/Right/ Up/Down Arrow</b>
Tilt the camera right and left.	<b>Shift+Alt+Left/ Right Arrow</b>

## Mouse Navigation

When you use your mouse to navigate through a virtual world, the behavior depends on the movement modes and navigation zones. Turn on the navigation zones and experiment by clicking and dragging your mouse in the different zones of a virtual world.

## Simulink 3D Animation Viewer Mouse Navigation

Movement Mode	Zone and Description
Walk	<p><b>Outer</b> — Click and drag the mouse up, down, left, or right to slide the camera in any of these directions in a single plane.</p> <p><b>Inner</b> — Click and drag the mouse up and down to move forward and backward. Drag the mouse left and right to turn left or right.</p>
Examine	<p><b>Outer</b> — Click and drag the mouse up and down to move forward and backward. Drag the mouse left and right to slide left or right.</p> <p><b>Inner</b> — Click and drag the mouse to rotate the viewpoint around the origin of the scene.</p>
Fly	<p><b>Outer</b> — Click and drag the mouse to tilt the view either left or right.</p> <p><b>Inner</b> — Click and drag the mouse to pan the camera up, down, left, or right within the scene.</p> <p><b>Center</b> — Click and drag the mouse up and down to move forward and backward. Move the mouse left or right to turn in either of these directions.</p>

If your virtual world contains sensors, these sensors take precedence over mouse navigation at the sensor location. For more information, see “Sensors Effect on Navigation” on page 7-20.

## Navigation Control Menu

Access the control menu by right-clicking in the viewer window. You can use the control menu to specify a predefined viewpoint or change the appearance of the control panel. You can also control the navigation method, speed, and rendering of the virtual world. For more information about navigation methods, see “Navigate Using the Simulink 3D Animation Viewer” on page 7-15. For more information about rendering, see “Specify Rendering Techniques” on page 7-9.

## Change the Navigation Speed

You can change the speed at which you navigate around the view.

- 1 In the menu bar, select the **Navigation** menu.
- 2 Select the **Speed** option.
- 3 Select the specific speed that you want.
- 4 Navigate the virtual world.

---

**Note** Your navigation speed controls the distance that you move with each keystroke. It does not affect rendering speed.

---

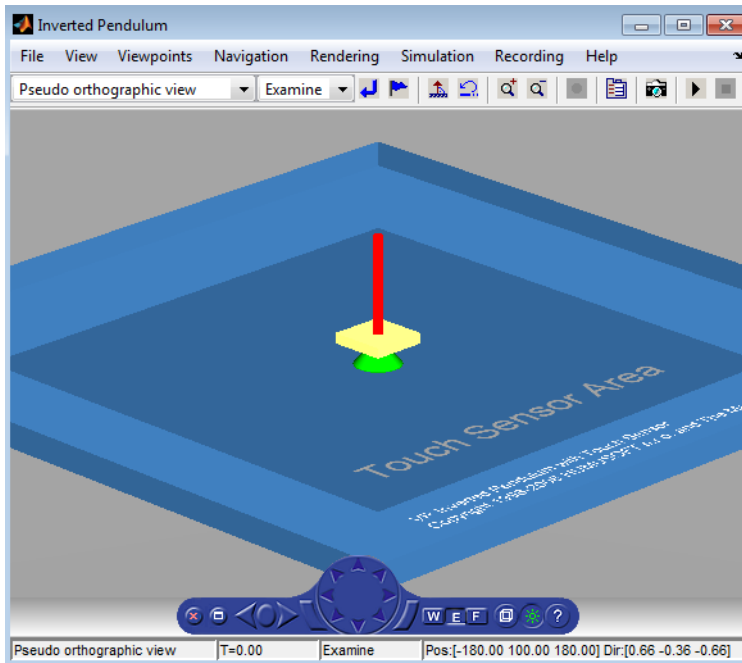
Consider setting a higher speed for large scenes and a slower speed for more controlled navigation in smaller scenes.

To change the default navigation speed for a virtual scene, modify the **speed** field of the `NavigationInfo` node in the scene virtual world 3D file.

## Sensors Effect on Navigation

- 1 At the MATLAB command prompt, type  
vrpend

The Inverted Pendulum example starts, and the viewer displays this scene.



- 2 In the Simulink Editor window, from the **Simulation** menu, choose **Run**.

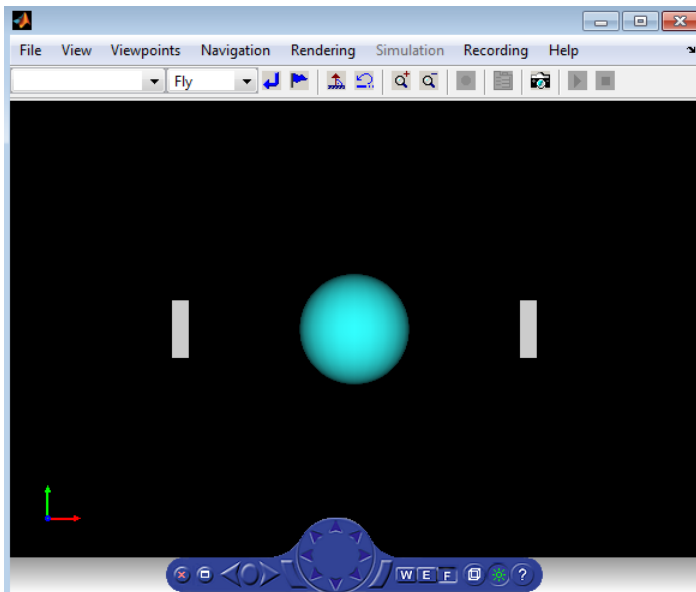
The example starts running.

- 3 Click inside and outside the sensor area of the viewer window. The sensor takes precedence over navigation with the left mouse button. The shape of your pointer changes when it is located over the sensor area.

If the sensor covers the entire navigable area, mouse navigation is effectively disabled. In this case, use the control panel or the keyboard to move about the scene. For a three-button mouse or a mouse with a clickable wheel, you can use the middle button or the wheel to move about the scene. The middle mouse button and wheel do not trigger sensors within the virtual world.

## Display a Coordinate Axes Triad

To help you visualize changes in the orientation (coordinate axes) of nodes in a virtual world, display a triad of red, green, and blue arrows. These arrows are always parallel with global  $x$ ,  $y$ , and  $z$  coordinate axes. As you navigate in a virtual world, the triad display changes to reflect changes in orientation.



To display a triad in the viewer, or to change the location of the triad, use *either* of these approaches:

- Right-click in the virtual world. Select the appropriate option from the **View > Triad** menu.
- In the viewer menu bar, select the appropriate option from the **View > Triad** menu.

To change the default location or visibility of the triad:

- 1 From the MATLAB Toolstrip, in the **Home** tab, in the **Environment** section, select **Preferences**.
- 2 In the Preferences dialog box, select **Simulink 3D Animation > Figure > Triad**.

## Pivot Point

To use a mouse to rotate a virtual world around a point, In Examine mode, you can use a pivot point.

To set the pivot point in a virtual world, hold **Ctrl** and double-click the spot where you want the pivot point.

---

**Note** On Macintosh platforms, use the command key instead of **Ctrl**.

---

## See Also

### Functions

vrgetpref | vrsetpref

## Related Examples

- “Define Viewpoints” on page 7-23
- “Navigate Through Viewpoints” on page 7-26

### **More About**

- “Virtual World Viewers” on page 7-2
- “Simulink 3D Animation Viewer” on page 7-4

## Set Viewpoints

### In this section...

“Define Viewpoints” on page 7-23

“Reset Viewpoints” on page 7-25

Visitors to your virtual world navigate in an environment that you create for them, using navigation methods allowed by the viewer (Walk, Examine, Fly). It is useful to set up in the world several locations, places of interest you want to point the visitors to. These locations are called viewpoints. Visitors can browse through them, carrying out a guided tour you prepared for them, gaining the visual information you consider important in your model.

When entering a world, you are placed at the first **Viewpoint** node encountered in the file. It is especially important to define this viewpoint carefully as the most interesting entry point.

Each virtual world has as many viewpoints as you define for it. You can define viewpoints in the virtual world through your chosen editor or through the Simulink 3D Animation viewer.

Defined viewpoints can be:

- Static — usually created at the top level of the virtual world object hierarchy or as children of static objects (Transforms).
- Dynamic — created as children of moving objects (objects driven from MATLAB/Simulink) or linked to them using the VRML ROUTE mechanism.

Dynamic viewpoints allow you to create interesting effects like view at the driving range from the view of the driver.

## Define Viewpoints

You can add new viewpoints to the virtual world through the menu bar or toolbar. You can start the simulation before creating viewpoints. This procedure assumes that the model is not currently running.

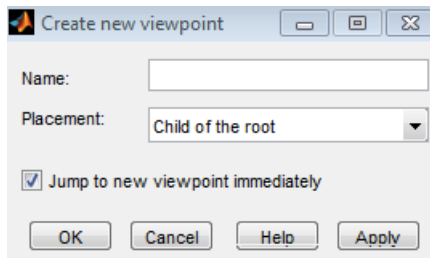
- 1 Select a Simulink 3D Animation example and type that example name in the MATLAB Command Window. For example:

```
vrplanets
```

The Simulink model is displayed. Also, by default, the Simulink 3D Animation viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the VR Sink block in the Simulink model.

In the Simulink 3D Animation viewer, the default viewpoint for this model is **View from top**.

- 2 From the menu bar, choose the **Viewpoints > View on Earth**.
- 3 In the viewer window, navigate to a random position in the scene.
- 4 Select the **Viewpoints>Create Viewpoint**.



- 5 For the **Name** parameter, enter a unique and descriptive name for the viewpoint.
- 6 Specify the placement of the viewpoint. Set **Placement** to **Child of the root**. This option makes the viewpoint a static one.

The availability of the **Placement** parameter depends on the current viewpoint. If the current viewpoint is at the top hierarchy level in the virtual world (one of the children of the root), the parameter is grayed out. In this case, it is only meaningful to create the viewpoint at the same top hierarchy level.

- 7 To make the new viewpoint the current viewpoint for the view, select **Jump to new viewpoint immediately** and click **OK**. If you do not select this option, you still create a viewpoint, but you remain bound to the current viewpoint, not to the new viewpoint.
- 8 Save the file with the new viewpoint, using **FileSave As**. If you do not save the file, the new viewpoint is lost during simulation.
- 9 Simulate the model by selecting **Simulation Start**. Observe the motion of the planets from the new, static viewpoint. Then stop the simulation.
- 10 Create another viewpoint.
- 11 Create a viewpoint at the same level in the virtual world object hierarchy as the child of the parent transform of the current viewpoint. Set **Placement** to **Sibling of the current viewpoint**. The local coordinate system of the parent transform defines the new viewpoint coordinates. As a result, the new viewpoint moves with the parent transform. The new viewpoint also keeps the position relative to the transform (offset) that you first defined by navigating somewhere in the space from the current viewpoint.

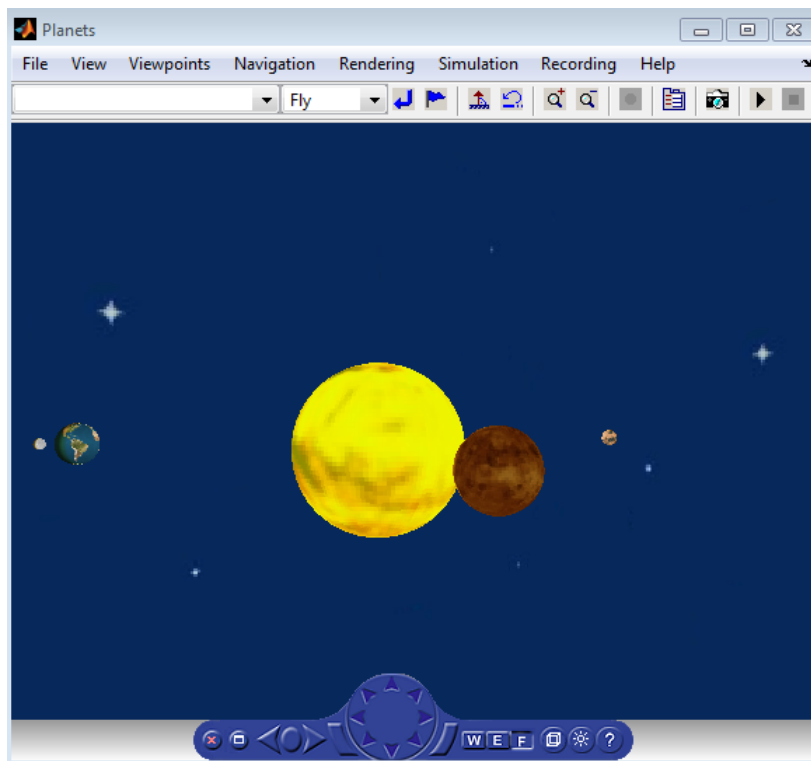
---

**Note** If the current viewpoint is at the top hierarchy level in the virtual world (one of the children of the root), the **Placement** field is grayed out. In this case, it is only meaningful to create the viewpoint as a static one at the same top hierarchy level.

---

- 12 Make the new viewpoint the current viewpoint for the viewer. Select **Jump to new viewpoint immediately** and click **OK**. If you do not select this option, you still create a viewpoint, but you remain bound to the current viewpoint, not to the new viewpoint.
- 13 Save the file with the new viewpoint. If you do not save the file, the new viewpoint is lost during simulation.
- 14 Simulate the model. Observe that the relative position between the new viewpoint and Earth remains the same. The new viewpoint moves together with its parent object Earth transform.





## Reset Viewpoints

You can reset your position in a scene to the initial default or current viewpoint position through the menu bar, toolbar, navigation panel, or keyboard shortcut keys.

- From the menu bar, use the **Viewpoints** menu **Return to viewpoint** option to return to the initial position of the current viewpoint. Alternatively, from the toolbar, select **Return to viewpoint** button to return to the initial position of the current viewpoint.
- From the navigation panel, click the **Go to default viewpoint** control to return to the default viewpoint of the virtual world. Alternatively, from the menu bar, use the **Viewpoints** menu **Go to Default Viewpoint** option to return to the default viewpoint of the virtual world.
- From the keyboard:
  - To return to the default viewpoint of the virtual world, press the **Esc** key.
  - To return to the initial position of the current viewpoint, press the **Home** key.

## Navigate Through Viewpoints

You can navigate through viewpoints using the menu bar, toolbar, navigation panel, or keyboard shortcut keys. These navigation methods are inactive if the author has specified no or only one viewpoint in the virtual world.

- From the menu bar, use the **Viewpoints** menu to move between viewpoints. Use the **Previous Viewpoint** and **Next Viewpoint** options to move up and down the list of existing viewpoints sequentially. To move focus to a particular viewpoint, choose a viewpoint from the list of viewpoints.
- From the toolbar, use the drop-down list of viewpoints to select a particular viewpoint.
- From the navigation panel, use the **Previous Viewpoint** and **Next Viewpoint** controls to move up and down the list of existing viewpoints sequentially.
- From the keyboard, press **Page Up** and **Page Down**.

To reset a camera to the initial position of the current viewpoint, use one of the methods listed in “Reset Viewpoints” on page 7-25. Resetting the viewpoint is useful to reorient yourself when you have been moving about the scene.

This topic illustrates viewpoints using the `vrplanets` example.

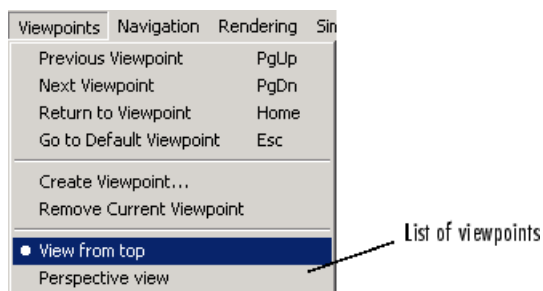
- 1 Select a Simulink 3D Animation example and type that example name in the MATLAB command window. For example:

```
vrplanets
```

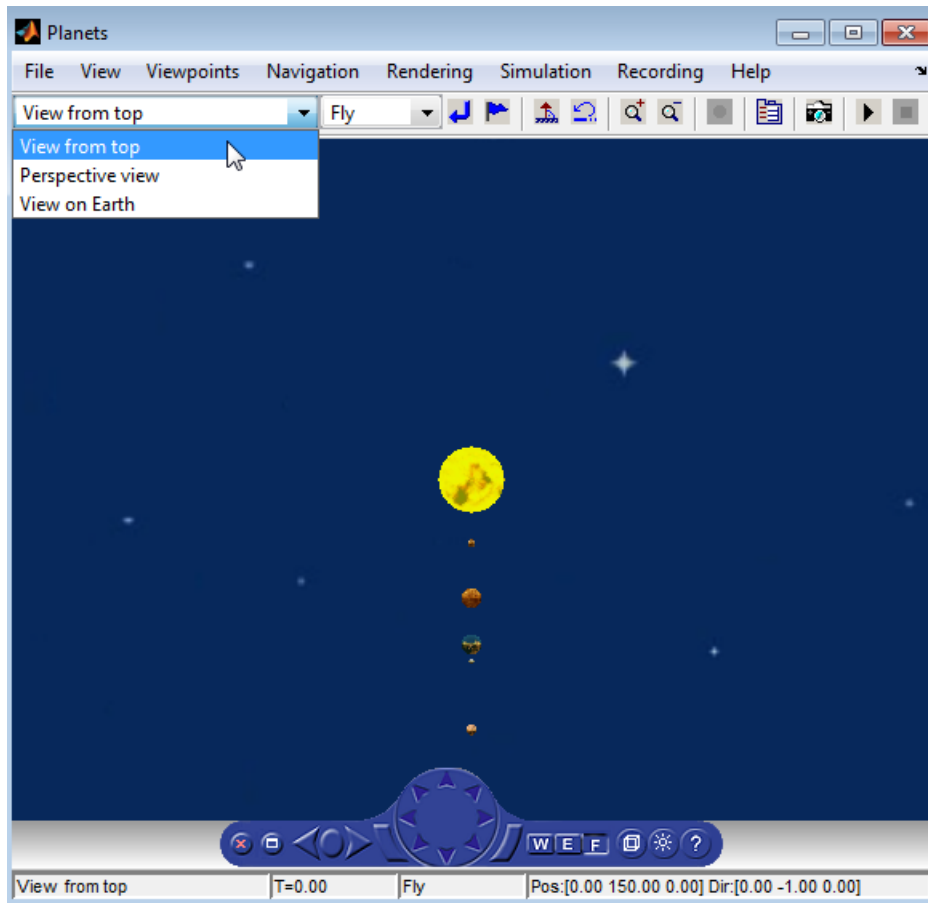
The Simulink model is displayed. By default, the Simulink 3D Animation viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the VR Sink block in the Simulink model.

- 2 From the menu bar, select the **Viewpoints** menu.

A menu of the viewpoint options is displayed. Included is a list of the existing viewpoints.



- 3 To see the list of existing viewpoints from the toolbar, select the drop-down list on the leftmost side of the toolbar. The status bar at the bottom of the viewer displays the current viewpoint.



When you add new viewpoints to the world, these lists are updated to reflect the new viewpoints.

The current viewpoint is also displayed in the left pane of the status bar.

You manage and navigate through viewpoints from the menu bar, toolbar, navigation panel, and keyboard. In particular, you can

- Navigate to a previous or next viewpoint
- Return to the initial position of the current viewpoint
- Go to the default viewpoint
- Create and remove viewpoints
- Navigate to an existing viewpoint

## See Also

## Related Examples

- “Define Viewpoints” on page 7-23

## **More About**

- “Set Viewpoints” on page 7-23

## Record Offline Animations

### In this section...

“Animation Recording” on page 7-29  
 “Recording Formats” on page 7-29  
 “File Names” on page 7-30  
 “Start and Stop Animation Recording” on page 7-30  
 “Play Animation Files” on page 7-31  
 “Record 3-D Animation Files” on page 7-31  
 “Record in Audio Video Interleave (AVI) Format” on page 7-31  
 “Schedule Files for Recording” on page 7-33

### Animation Recording

The Simulink 3D Animation software enables you to record animations of virtual scenes that the Simulink or MATLAB product controls. You can record simulations using the Simulink 3D Animation Viewer. You can then play back these animations offline, in other words, independent of the MATLAB, Simulink, or Simulink 3D Animation products. You can generate such files for presentations, to distribute simulation results, or to generate archives.

---

**Note** Optimally, use the Simulink 3D Animation Viewer to record animations of virtual worlds associated with Simulink models. This method ensures that all necessary virtual world and `vrfigure` properties are properly set to record simulations. If you are working with virtual scenes controlled from MATLAB, you can record virtual scenes through the MATLAB interface. For details, see “Animation Recording” on page 4-10.

---

You can save a frame snapshot (capture) of the current Simulink 3D Animation viewer scene. You can save this frame as either a TIF or PNG format file. For details, see “Capture Frames” on page 7-37.

### Recording Formats

You can save the virtual world offline animation data in the following formats:

- 3D file — The Simulink 3D Animation software traces object movements and saves that data into a virtual world 3D file using standard interpolators. You can then view these files with the Simulink 3D Animation Viewer. 3-D files typically use much less disk space than Audio Video Interleave (AVI) files. If you make any navigation movements in the Simulink 3D Animation Viewer while recording the animation, the Simulink 3D Animation software does not save any of these movements.

---

**Note** If you distribute virtual world 3D animation files, distribute all the inlined object and texture files referenced in the original virtual world 3D file.

---

- 2-D Audio Video Interleave (AVI) file — The Simulink 3D Animation software writes animation data into an `.avi` file. The Simulink 3D Animation software uses `vrfigure` objects to record 2-D animation files. The recorded 2-D animation reflects exactly what you see in the viewer window. It includes any navigation movements you make during the recording.

---

**Note** While recording 2-D .avi animation data, always ensure that the Simulink 3D Animation Viewer is the topmost window and fully visible. Graphics acceleration limitations can prevent the proper recording of 2-D animation otherwise.

---

See the following topics:

- “Record 3-D Animation Files” on page 7-31 — Describes how to configure the record simulation parameters to create 3-D format animation files.
- “Record in Audio Video Interleave (AVI) Format” on page 7-31 — Describes how to configure the record simulation parameters to create 2-D format animation files.
- “Schedule Files for Recording” on page 7-33 — Describes how to schedule record simulation operations to occur automatically.

## File Names

By default, the Simulink 3D Animation Viewer records simulations or captures virtual scene frames in a file named with the following format:

```
%f_anim_%n.%e
```

This format creates a unique file name each time you capture a frame or record the animation. The file name uses the %f, %n, and %e tokens.

The %f token is replaced with the name of the virtual world associated with the model. The %n token is a number that increments each time that you record a simulation for the same virtual world. For example, if the name of the virtual world file is `vrplanets.wrl` and you record a simulation for the first time, the animation file is `vrplanets_anim_1.wrl`. If you record the simulation a second time, the animation file name is `vrplanets_anim_2.wrl`. In the case of frame captures, capturing another frame of the scene increments the number.

The %e token represents the virtual world 3D file extension (`.wrl`, `.x3d`, or `.x3dv`) as the extension of the virtual world that drives the animation. By default, the %e token uses the file extension of the virtual world 3D file that drives the animation. The VR Sink and VR Source block **Source file** parameter specifies the file extension of the virtual world.

You can specify other file name tokens. For details, see “File Name Tokens” on page 4-14.

## Start and Stop Animation Recording

You can start or stop recording animations of the virtual world from the Simulink 3D Animation viewer through the menu bar, toolbar, or keyboard. This section assumes that you have specified animation files for recording the animation.

- From the menu bar, choose the **Simulation** menu, **Run** option to start recording the animation (select **Stop** to stop the recording).
- From the toolbar, click the **Start/stop recording** button to start or stop recording the animation (select **Stop** to stop the recording). Alternatively, you can use the **Recording** menu **Start Recording** and **Stop Recording** options. From the keyboard, press **Ctrl+R** to toggle between starting or stopping the animation recording.
- Stop the simulation or let the model simulate until the defined simulation stop time.

---

**Note** If you stop the simulation while recording is enabled, the viewer also stops recording the animation.

---

## Play Animation Files

You can view animation files using the 3D Animation Player or `vrplay`. For details, see “Play Animation Files” on page 4-27.

## Record 3-D Animation Files

To create a 3-D animation files from a Simulink model execution, set recording parameters. You can start the simulation before setting up the recording.

- 1 In the MATLAB Command Window, type the model name. For example:

```
vrplanets
```

The Simulink model is displayed. Also, by default, the Simulink 3D Animation viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the VR Sink block in the Simulink model.

- 2 From the **Recording** menu, choose **Capture and Recording Parameters**.

The Capture and Recording Parameters dialog box is displayed.

- 3 Find the **Recording** section of the dialog. This is located under the Frame Capture dialog.
- 4 Select the **Record to 3D** file check box.

The **File** text field becomes active and the default file name, `%f_anim_%n.wrl`, appears in the text field.

To save files to other file names, see “File Name Tokens” on page 4-14.

- 5 Click **OK**.

After you define an animation file, you can manually record simulations. See “Start and Stop Animation Recording” on page 7-30. If you want to record simulations on a schedule, see “Schedule Files for Recording” on page 7-33.

## Record in Audio Video Interleave (AVI) Format

To create a 2-D AVI format file from a Simulink model execution, set recording parameters. You can start the simulation before setting up the recording.

- 1 In the MATLAB Command Window, type the model name. For example:

```
vrplanets
```

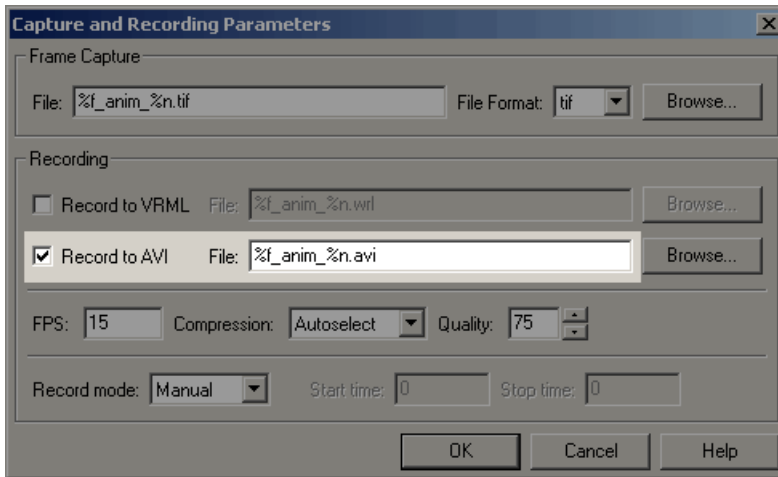
The Simulink model is displayed. Also, by default, the Simulink 3D Animation viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the VR Sink block in the Simulink model.

- 2 From the **Recording** menu, choose **Capture and Recording Parameters**.

The Capture and Recording Parameters dialog box is displayed.

- 3 Find the **Recording** section of the dialog box, located under the Frame Capture dialog box.
- 4 Select the **Record to AVI** file check box.

The **File** text field and Compression selection area become active, and the default file name, `%f_anim_%n.avi`, appears in the text field.



To save files to other file names, see “File Name Tokens” on page 4-14.

- 5 Set the **FPS** (Frames Per Second) to an appropriate value.

To use the sample time of the associated VR Sink block to make the file playback correspond to the model simulation time, set **FPS** to **auto**.

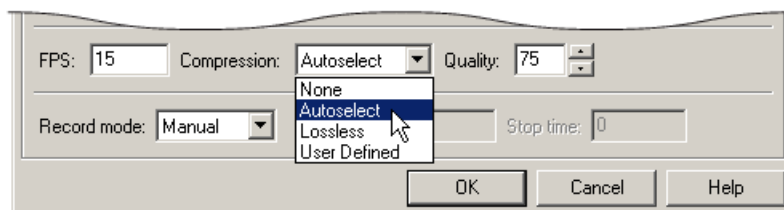
For example, to record a Simulink simulation with 25 frames per second (of the simulation time), in the VR Sink block set **Sample time** to be `0.04`. In that situation, to create an AVI file where one second of simulation time corresponds to one second of AVI file playback time, set the **FPS** parameter to **auto**. Simulink 3D Animation saves the value 25 into the AVI file **FPS** parameter.

---

**Note** An **FPS** setting of 15 is used, even if you set **FPS** to **auto**:

- For a virtual world not associated with a Simulink model.
- If the sample time of the associated VR Sink block cannot be determined at simulation start time.

- 6 From the **Compression** list, select a compression method for the `.avi` file. Because `.avi` files can become large, you can compress the `.avi` file.



Choose from



- **Autoselect** — Allows the Simulink 3D Animation software to select the most appropriate compression codec. This option allows you to specify a quality setting that is a number from 0 through 100. Higher-quality numbers result in higher video quality and larger file sizes. Lower quality numbers result in lower video quality and smaller file sizes.
  - **Lossless** — Forces the Simulink 3D Animation software to compress the animation file without loss of data. (Typically, the compression of files sacrifices some data.)
  - **User Defined** — Enables you to specify a particular compression codec. This option allows you to specify a quality setting that is a number from 0 through 100. Higher-quality numbers result in higher video quality and larger file sizes. Lower quality numbers result in lower video quality and smaller file sizes. Specify an identification string of a codec that your system supports.
  - **None** — Prevents any compression for the animation file.
- 7** Disable the navigation panel. The navigation panel appears at the bottom of the virtual scene view. You can turn off this panel for a cleaner view of the virtual scene. Choose **View > Navigation Panel > None**.

You can reenable the Navigation Panel (for example, choose **View > Navigation Panel > Halfbar**) after you are finished recording the .avi file.

- 8** Click **OK**.

After you define an animation file, you can record animations. See “Start and Stop Animation Recording” on page 7-30. If you want to record animations on a schedule, see “Schedule Files for Recording” on page 7-33.

## Schedule Files for Recording

This topic describes how to schedule the recording of an animation using the MATLAB interface for a virtual world that is associated with a Simulink model. In this case, the timing in an animation file derives from the simulation time. One second of the recorded animation time corresponds to one second of Simulink time. To schedule the recording of an animation file, you preset the simulation time interval during which the animation recording occurs. This procedure uses the `vrplanets` example. It assumes that you have already configured the recording parameters for an animation file.

- 1** In the MATLAB Command Window, type the model name. For example:

```
vrplanets
```

The Simulink model is displayed. Also, by default, the Simulink 3D Animation viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the VR Sink block in the Simulink model.

- 2** From the **Recording** menu, choose **Capture and Recording Parameters**.

The Capture and Recording Parameters dialog box is displayed. In the **Recording** section, this dialog box contains the **Record mode** list. The **Record mode** list is enabled only if you also select either or both of the **Record to 3D** and **Record to AVI** check boxes.

- 3** From the **Record mode** list, choose **Scheduled**.

The **Start time** and **Stop time** text fields are enabled.

- 4** Enter in **Start time** and **Stop time** the start and stop times during which you want to record the animation. For example, enter 0 as the start time and 100 as the stop time.

Ensure that the recording start time value is not earlier than the start time of the Simulink model; the recording operation cannot start in this instance. If the stop time exceeds the stop time of the Simulink model, or if it is an out of bounds value such as a negative number, the recording operation stops when the simulation stops.

---

**Note** You can also set the stop time before the start time to allow for a scenario where the simulation starts first and you manually start recording. The recording then automatically stops at stop time and automatically restarts at start time.

---

**5** Click **OK**.

After you define the schedule, you can record simulations. See “Start and Stop Animation Recording” on page 7-30.

---

**Note** You can override the recording schedule by starting or stopping the recording interactively.

---

## See Also

### Functions

vrplay

## Related Examples

- “Play Animations with Simulink 3D Animation Viewer” on page 7-35
- “Configure Frame Capture Parameters” on page 7-36
- “Capture Frames” on page 7-37
- “Define File Name Tokens” on page 4-12

## More About

- “Animation Recording” on page 4-10
- “File Name Tokens” on page 4-14

## Play Animations with Simulink 3D Animation Viewer

To play animation files, you can use a web browser or you can use the Simulink 3D Animation Viewer using one of these approaches:

At the MATLAB command line, use `vrview`. For example, enter:

```
w=vrview('vrplanets_anim_1.wrl');  
set(w, 'TimeSource', 'freerun');
```

The `vrview` command displays the default Simulink 3D Animation Viewer for the animation file. Setting the `TimeSource` property of the `set` method to `'freerun'` directs the viewer to advance its time independent of the MATLAB software.

To stop the animation, type:

```
set(w, 'TimeSource', 'external');
```

To close the viewer and delete the world, get the handle of the `vrfigure` object and close it:

```
f=get(w, 'Figures')  
close(f);  
delete(w);
```

Or, to close all `vrfigure` objects and delete the world, type

```
vrclose  
delete(w);
```

### See Also

#### Functions

`vrplay`

### Related Examples

- “Record Offline Animations” on page 7-29
- “Configure Frame Capture Parameters” on page 7-36
- “Capture Frames” on page 7-37
- “Define File Name Tokens” on page 4-12

### More About

- “Animation Recording” on page 4-10
- “File Name Tokens” on page 4-14

## Configure Frame Capture Parameters

This topic describes how to configure and capture a frame, using the `vrplanets` example as the example.

- 1 In the MATLAB Command Window, type

```
vrplanets
```

at the MATLAB command prompt. The Planets example starts.

- 2 From the **Recording** menu, choose **Capture and Recording Parameters**.

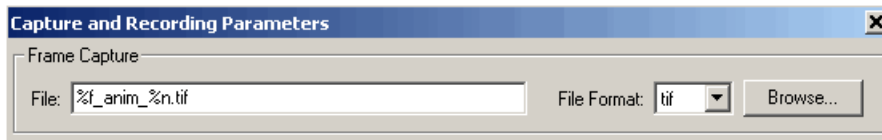
The Capture and Recording Parameters dialog box is displayed.

- 3 Find the **Frame Capture** section at the top of the dialog box.

The file name `%f_anim_%n.tif` appears in the first text field, **File**.

- 4 Leave this file name as is.

- 5 In the **File Format** list, `tif` or `png` specify the graphic file format for the captured frame. The default is `tif`. For this procedure, leave this format setting at `tif`.



- 6 You can disable the navigation panel. The navigation panel appears at the bottom of the virtual scene view. You can turn off this panel for a cleaner view of the virtual scene. Choose **View > Navigation Panel > None**.

You can reenable the Navigation Panel (for example, choose **View > Navigation Panel > Halfbar**) after you finish recording the `.tif` file.

- 7 Click **OK**.

With this configuration, each subsequent capture of a scene in the same world increments the file name number (`%n`) and saves it in a `tif` file.

### See Also

#### Related Examples

- “Record Offline Animations” on page 7-29
- “Capture Frames” on page 7-37
- “Define File Name Tokens” on page 4-12

## Capture Frames

The Simulink 3D Animation product allows you to save a frame snapshot (capture) of the current Simulink 3D Animation viewer scene. You can save this frame as either a TIF or PNG format file. You can later view this scene offline (in other words, without the Simulink 3D Animation viewer). You can treat this frame capture file like any other TIF or PNG file, such as print it, include it in other files.

After you complete the steps in “Configure Frame Capture Parameters” on page 7-36, you can capture frames of a virtual scene.

You can capture frames of the virtual world from the Simulink 3D Animation viewer through the menu bar, toolbar, or keyboard. This section assumes that you have specified frame capture file formats.

These actions save the captures in files in the current folder.

- From the menu bar, choose **Recording > Capture Frame** to capture a frame.
- From the toolbar, click the **Capture a frame screenshot** button to capture a frame.
- From the keyboard, press **Ctrl+I** to capture a frame.

You can view the frame capture files using any tool that reads `tif` or `png` files, including the MATLAB `imread` function. For example,

```
image(imread('vrplanets_anim_1.tif'))
```

### See Also

#### Related Examples

- “Record Offline Animations” on page 7-29
- “Configure Frame Capture Parameters” on page 7-36
- “Define File Name Tokens” on page 4-12

## Simulink 3D Animation Web Viewer

Use the Simulink 3D Animation Web Viewer to access virtual worlds with an HTML5-enabled web browser. You can open a virtual world in Simulink 3D Animation on a host computer and then view it remotely in a web browser on another computer. You do not need to install Simulink 3D Animation on the remote computer.

The Web Viewer supplements the Simulink 3D Animation Viewer. It also supplements the Orbisnap viewer, which comes with Simulink 3D Animation. Some benefits of the Web Viewer include:

- Multiplatform remote viewing of Simulink 3D Animation virtual worlds
  - Support for HTML5-enabled browsers on Microsoft Windows, Macintosh, and Linux platforms
- No additional software installation required on client computers
- Access to HTML5 browser features for creating customized pages with virtual reality visualization

---

**Note** Because the Web Viewer accesses an HTML version of the virtual world, you cannot use it to modify the virtual world (for example, to create viewpoints).

---

### See Also

#### Functions

`vrview`

#### Related Examples

- “Open the Web Viewer” on page 7-39
- “Navigate Using the Web Viewer” on page 7-41
- “Listen to Sound in a Virtual World” on page 7-43
- “View a Virtual World in Stereoscopic Vision” on page 7-45

#### More About

- “Virtual World Viewers” on page 7-2
- “Active Stereoscopic Vision Configuration” on page 7-47

## Open the Web Viewer

### In this section...

“Set up for Remote Viewing” on page 7-39

“Connect the Web Viewer” on page 7-39

### Set up for Remote Viewing

To enable a person on a remote computer to use a web browser to view a virtual world that is connected to a Simulink model:

- 1 In the Simulink 3D Animation Viewer, select **Simulation > Block parameters > Allow viewing from the Internet**.

If the Simulink 3D Animation Viewer is the default viewer, you can open that viewer from the Simulink Editor by double-clicking the VR Sink block.

- 2 Enable the **Allow viewing from the Internet** parameter.

Alternatively, to enable remote computers to view all opened virtual worlds with a web browser, in the MATLAB **Home** tab, in the **Environment** section, use the **Preferences > Simulink 3D Animation > Allow viewing from the Internet** preference.

### Connect the Web Viewer

- 1 In Simulink 3D Animation, open the virtual world that you want to view.
- 2 On the computer from which you want to open the Web Viewer, open an HTML5-enabled browser (supporting WebGL and WebSocket). The browser must have Javascript enabled.

---

**Note** For a list of HTML5-enabled browsers, see [https://www.x3dom.org/?page\\_id=9](https://www.x3dom.org/?page_id=9).

---

- 3 Connect to the local host HTTP port. To open the Web Viewer on:
  - The same computer on which Simulink 3D Animation runs, enter the following in your web browser: `http://localhost:8123/`
  - A remote computer that does not have Simulink 3D Animation, in the `http://localhost:8123` URL, replace `localhost` with the HTTP port of the host computer.

---

**Note** To change the HTTP port (for example, if a firewall blocks a port), set a different port number and restart MATLAB. To change the HTTP port, in the MATLAB **Home** tab, in the **Environment** section, use the **Preferences > Simulink 3D Animation > HTTP Port** preference.

---

- 4 In the Web Viewer, from the list of open virtual worlds, select the one that you want to view.

### See Also

#### Functions

`vrview`

### **Related Examples**

- “Navigate Using the Web Viewer” on page 7-41
- “Listen to Sound in a Virtual World” on page 7-43
- “View a Virtual World in Stereoscopic Vision” on page 7-45

### **More About**

- “Simulink 3D Animation Web Viewer” on page 7-38
- “Virtual World Viewers” on page 7-2
- “Active Stereoscopic Vision Configuration” on page 7-47



## Navigate Using the Web Viewer

### In this section...

“Display and Navigation” on page 7-41

“Keyboard Shortcuts” on page 7-41

“Web Viewer Preferences” on page 7-42

### Display and Navigation

Most of the navigation features for the Web Viewer are the same as for the Simulink 3D Animation Viewer and the 3D World Editor **Virtual world display** pane. For details about Simulink 3D Animation Viewer navigation, see “Navigate Using the Simulink 3D Animation Viewer” on page 7-15.

Some differences between the two viewers include:

- The Web Viewer does not include a menu bar.
- For virtual worlds with undefined background colors, the Web Viewer uses the default canvas color of the browser; the Simulink 3D Animation Viewer uses a black background.

The supported navigation features depend on what web browser you use. For example, when R2013b was released, Firefox® was the only supported web browser that supports context menu options that are specific to Simulink 3D Animation, such as viewpoints options.

To determine whether the HTML5-enabled browser that you want to use supports WebGL (Web Graphics Library), see <https://www.khronos.org/webgl/>.

### Keyboard Shortcuts

Navigation Function	Keyboard Shortcut
Straighten up and make the camera stand on the horizontal plane of its local coordinates.	U
Toggle the headlight on and off.	H
Toggle the navigation zones on and off.	Z
Go to default viewpoint.	Esc
Return to current viewpoint.	R
Go to previous viewpoint.	Page Up
Go to next viewpoint.	Page Down
Set the navigation method to Walk.	W
Set the navigation method to Examine.	E
Set the navigation method to Fly.	F
Move the camera forward and backward.	Shift Up/Down Arrow
Pan the camera up and down.	Up/Down Arrow
Pan the camera right and left.	Left/Right Arrow, Shift+Left/Right Arrow

Navigation Function	Keyboard Shortcut
Tilt the camera right and left.	<b>Shift+Alt+Left/ Right Arrow</b>
Slide up and down.	<b>Alt+Up/Down Arrow</b>
Slide left and right.	<b>Alt+Left/Right Arrow</b>
Set pivot point	Double-click
Orbit around selected item (pivot point)	<b>Ctrl+Left/Right/Up/Down Arrow</b>
Turn interactive mode on or off	I
Set navigation mode to none	N
Cycle through navigation speed presets	G
Show or hide information panel	D
Show or hide status bar	S
Show or hide rendering information panel	<b>Spacebar</b>
Cycle through navigation panel modes	P

## Web Viewer Preferences

The following Simulink 3D Animation preferences apply to the Web Viewer, and to the 3D World Editor.

- **Canvas > Navigation panel**
- **Figure > Appearance > Status bar**
- **Figure > Appearance > Navigation zones**

To access the preferences, from the MATLAB Toolstrip, in the **Home** tab, in the **Environment** section, select **Preferences > Simulink 3D Animation**.

## See Also

### Functions

`vrgetpref` | `vrsetpref` | `vrview`

## Related Examples

- “Open the Web Viewer” on page 7-39
- “Set Simulink 3D Animation Preferences” on page 2-5
- “Listen to Sound in a Virtual World” on page 7-43
- “View a Virtual World in Stereoscopic Vision” on page 7-45

## More About

- “Simulink 3D Animation Web Viewer” on page 7-38
- “Active Stereoscopic Vision Configuration” on page 7-47

## Listen to Sound in a Virtual World

### In this section...

“System Requirements for Sound” on page 7-43

“Listen to Sound” on page 7-43

If a virtual world contains a Sound node and your computer supports sound, then you can listen to the sound using these Simulink 3D Animation components:

- Simulink 3D Animation Viewer
- 3D Animation Player
- `vr.canvas` on a figure window

### System Requirements for Sound

To listen to virtual world sound, use a computer setup that supports sound, including having:

- A sound card
- Speakers
- Operating system support, such as ALSA (Advanced Linux Sound Architecture) on Linux platforms

For an AudioClip node, use a mono or stereo WAV file in uncompressed PCM format.

---

**Note** A stereo sound source retains its channel separation during playback. Simulink 3D Animation attenuates the sound based on the distance of the viewer from the sound location. The relative position of the viewer to the sound location and the viewer direction in the virtual world do not affect the stereo channels. There is no impact even if the Sound node has the `spatialize` field set to `true`.

---

### Listen to Sound

Simulink 3D Animation enables sound by default.

To change the default behavior so that sound is disabled, set the Simulink 3D Animation **Figure > Rendering > Sound** preference to `off`.

In the Simulink 3D Animation Viewer or 3D Animation Player, to disable sound, right-click in the virtual world and clear the **Rendering > Sound** option.

For a `vr.canvas` object, to disable sound, set the Sound property to `'off'`.

To control volume, use your computer volume controls.

### See Also

### Related Examples

- “Add Sound to a Virtual World” on page 5-35

## **More About**

- “Simulink 3D Animation Web Viewer” on page 7-38

## View a Virtual World in Stereoscopic Vision

In this section...
“Enable Stereoscopic Vision” on page 7-45
“Control Stereoscopic Effects” on page 7-45

You can view a virtual world using 3D effects, so that elements in the virtual world appear to come forward or back from the plane of the monitor.

You can use stereoscopic vision with these Simulink 3D Animation components:

- 3D World Editor
- Simulink 3D Animation Viewer
- 3D Animation Player
- `vr.canvas` on a figure window
- Orbisnap

Simulink 3D Animation supports two stereoscopic vision approaches:

- Anaglyph — Use red/cyan 3D glasses. Viewing a virtual world in this mode causes the colors to appear as almost grayscale. This approach does not require any special computer hardware or software.
- Active stereo — Use active shutter 3D glasses. This approach preserves color effects and produces more powerful 3D effects. Active stereo requires a specially configured computer and monitor setup. For details, see “Active Stereoscopic Vision Configuration” on page 7-47.

### Enable Stereoscopic Vision

By default, virtual worlds display without stereoscopic vision effects.

- 1 Right-click in the virtual world.
- 2 From the **Rendering > Stereo 3D** menu, select either **Anaglyph** or **Active**.

---

**Note** To enable stereoscopic vision by default, set the Simulink 3D Animation **Figure > Rendering > Stereo 3D** preference to anaglyph or active.

---

### Control Stereoscopic Effects

You can control the following stereoscopic effects interactively or using preferences.

Stereo 3D Effect	Description	Keyboard Shortcut	Figure > Rendering Preference
Camera offset	Distance between the two points of view (cameras) that produce the 3D effect. The higher the offset, the further apart the cameras are, and thus the deeper the 3D effect.	<b>Shift+K</b> increases the offset.  <b>Shift+J</b> decreases the offset.	<b>Stereo 3D Camera Offset</b>  The default value is 0.1.
Horizontal image translation (HIT)	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.  You can specify a value from 0 through 1, inclusive. The larger the value, the further back the background appears to be.	<b>Shift+O</b> increases the distance back for the background image.  <b>Shift+P</b> decreases the distance back for the background image.	<b>Stereo 3D Horizontal Image Translation</b>  The default value is 0.

You can also control the camera offset and horizontal image translation programmatically, using `vr.canvas`, `vr.figure`, and `vr.utils.stereo3d`. If you use a `vr.utils.stereo3d` object, you can also control the color filters for the left and right cameras.

## See Also

### Functions

`vr.utils.stereo3d`

## More About

- “Active Stereoscopic Vision Configuration” on page 7-47

# Active Stereoscopic Vision Configuration

## In this section...

“Computer Platforms” on page 7-47

“Graphics Cards” on page 7-47

“Display Devices” on page 7-47

“Graphic Card Connection to Display Devices” on page 7-47

“Examples of Stereoscopic Vision Setups” on page 7-48

This section identifies system requirements for active stereoscopic vision configuration. For detailed information to determine whether a system meets the Simulink 3D Animation active stereoscopic vision requirements, consult the documentation for your systems.

## Computer Platforms

You can use stereoscopic vision on properly configured Windows and Linux platforms. You cannot use active stereoscopic vision on Macintosh platforms.

## Graphics Cards

Your computer must have a stereo 3D graphic card that supports OpenGL-based stereoscopic vision, together with appropriate system driver, such as:

- AMD® FirePro “W” series of cards (for example, W5000) that support HD3D Pro technology
- NVIDIA® Quadro cards that support 3D Vision Pro technology

## Display Devices

To display the stereoscopic video output of a graphic card, use one of these 3D display devices.

- 3D-compliant monitor synchronized with active shutter glasses. Depending on the display technology, enable synchronization using Infrared emitters, cables, or RF hubs. Some monitors include an infrared (IR) emitter. Other monitors require a separate IR emitter.
- 3D television set that displays 3D content. For stereoscopic vision, you typically use active shutter glasses or passive polarized glasses.
- Auto-stereoscopic display (monitor, display containing pair of video projectors, etc.).

## Graphic Card Connection to Display Devices

Connect 3D graphic cards to 3D display devices using an interface such as DVI, HDMI, or DisplayPort.

HDMI 1.4a and DisplayPort display interfaces natively expose the ability to transmit stereo images using schemes described in their specifications. These interfaces allow for plug-and-play capability. It is up to the display device to decode the image pairs and present them according to the presentation technology they implement (active, passive, auto-stereoscopic).

The DVI interface does not offer native stereoscopic image transfer. To transfer and identify stereoscopic images correctly, synchronize the graphic card output with the display device, using synchronization signals transmitted through an additional cable, an IR emitter, or an RF hub.

## **Examples of Stereoscopic Vision Setups**

Here are two possible configurations for using stereoscopic vision with Simulink 3D Animation:

- AMD FirePro “W” series of cards (for example, W5000) connected with an HDMI 1.4 cable to a 3D television set
- NVIDIA Quadro cards (for example, Quadro K4000), a 3D vision-ready monitor connected using dual DVI cable, and a 3D Vision Pro kit (an RF hub and active shutter glasses)

## **See Also**

### **Related Examples**

- “View a Virtual World in Stereoscopic Vision” on page 7-45



# Simulink 3D Animation Stand-Alone Viewer

---

The Simulink 3D Animation stand-alone viewer, Orbisnap, allows you to visualize virtual worlds or prerecorded animation files without running the MATLAB or Simulink 3D Animation products.

- “Orbisnap Viewer” on page 8-2
- “Install Orbisnap” on page 8-3
- “Start Orbisnap” on page 8-5
- “Orbisnap Interface” on page 8-6
- “Navigate Using Orbisnap” on page 8-9
- “View Animations or Virtual Worlds with Orbisnap” on page 8-12
- “View Virtual Worlds Remotely with Orbisnap” on page 8-13

## Orbisnap Viewer

### What Is Orbisnap?

The Simulink 3D Animation product includes Orbisnap. Orbisnap is a free, optional, stand-alone virtual world viewer that does not require you to have either the MATLAB or Simulink 3D Animation products running. You can use Orbisnap to:

- View prerecorded WRL animation files. For example, you can show prerecorded animation files in a meeting at which you do not have access to the MATLAB or Simulink 3D Animation products.
- Remotely view, from a client machine, a virtual world loaded in a current session of the Simulink 3D Animation product. For example, to visualize a virtual world active in a Simulink 3D Animation session that is running on a computer in another room, or across the network. This functionality allows you to view a simulation remotely, but not control it.
- View and navigate, but not simulate, a virtual world. You can navigate, render, and otherwise visualize a virtual world without simulating it.
- View virtual worlds using stereoscopic vision.

Orbisnap is multiplatform. You can run Orbisnap on any of the platforms that the Simulink 3D Animation product supports. You do not need a MathWorks license to run Orbisnap.

### See Also

#### Related Examples

- “Set the Default Viewer” on page 2-2
- “Install Orbisnap” on page 8-3
- “Start Orbisnap” on page 8-5
- “Navigate Using Orbisnap” on page 8-9
- “View Animations or Virtual Worlds with Orbisnap” on page 8-12
- “View Virtual Worlds Remotely with Orbisnap” on page 8-13

#### More About

- “Orbisnap Interface” on page 8-6
- “Virtual World Viewers” on page 7-2

# Install Orbisnap

## In this section...

“Section Overview” on page 8-3  
 “System Requirements” on page 8-3  
 “Copying Orbisnap to Another Location” on page 8-3  
 “Adding Shortcuts or Symbolic Links” on page 8-3

## Section Overview

The collection of Orbisnap files includes the Orbisnap starter file, Orbisnap executable file, and supporting files. These files are located under the Simulink 3D Animation orbisnap folder (for example, *matlabroot\toolbox\sl3d\orbisnap\bin* for the Windows platform). No further installation is necessary. You can copy the Orbisnap files to another location or create shortcuts or symbolic links to the Orbisnap starter file for convenience.

## System Requirements

Orbisnap has the same hardware and software requirements as MATLAB. It is a multiplatform product that can run on PC-compatible computers with Windows or Linux. See the following page on the MathWorks website:

<https://www.mathworks.com/products/availability.html#ML>

## Copying Orbisnap to Another Location

Orbisnap runs independently of the MATLAB and Simulink 3D Animation products. This independence means that you can copy Orbisnap to another location or even another machine. The following is a general procedure on how to copy Orbisnap to another location:

- 1 From a command line or a graphical interface such as Windows Explorer, create a folder into which you can copy Orbisnap.
- 2 Copy all the files in the Orbisnap folder and its subfolders. These files are likely located in the Simulink 3D Animation orbisnap folder, for example, *matlabroot\toolbox\sl3d\orbisnap* for the Windows platform.
- 3 Paste the files into the folder you created in step 1.

## Adding Shortcuts or Symbolic Links

For convenience, you can create a shortcut (Windows) or symbolic link (UNIX) to the Orbisnap starter file.

- In Windows Explorer, right-click *orbisnap.bat* and select **Properties**. You can start Orbisnap from either the shortcut or the original starter file.
- In UNIX, use the `ln -s` command to create a symbolic link to *orbisnap*.

## **See Also**

### **Related Examples**

- “Start Orbisnap” on page 8-5
- “Navigate Using Orbisnap” on page 8-9
- “View Animations or Virtual Worlds with Orbisnap” on page 8-12
- “View Virtual Worlds Remotely with Orbisnap” on page 8-13

### **More About**

- “Orbisnap Viewer” on page 8-2
- “Orbisnap Interface” on page 8-6

## Start Orbisnap

You can start Orbisnap from any command line with the following:

```
orbisnap
orbisnap -f vr_filename
orbisnap -c hostname -w "vrworld" -t http -v vrport -q=end_time
orbisnap -t http -v vrport vr_filename_or_hostname -q=end_time
orbisnap -h
```

No arguments -- Starts the default Orbisnap. There is no loaded `vrworld` file and no connection to a Simulink 3D Animation server.

`-f vr_filename` — (Optional) Orbisnap starts and loads the `vrworld` contained in `vr_filename`. If you do not provide `vr_filename`, Orbisnap prompts you for the file name.

`-c hostname` — (Optional) Orbisnap starts and connects to the Simulink 3D Animation server at `hostname`. `hostname` can be a hostname or IP address. If you do not provide `hostname`, Orbisnap prompts you for the hostname.

`-w vrworld` — (Optional) Orbisnap starts, connects to the Simulink 3D Animation server, and loads the virtual world associated with the title "`vrworld`". If "`vrworld`" is not currently active in the Simulink 3D Animation server, the connection to the server does not succeed and the default Orbisnap starts.

`-t http` — (Optional) Orbisnap starts and connects to the Simulink 3D Animation server at this HTTP port (default 8123).

`-t vrport` — (Optional) Orbisnap starts and connects to the Simulink 3D Animation server listening at this port (default 8124).

`vr_filename_or_hostname` — (Optional) Orbisnap starts and interprets this string first as a `vrworld` file name (for example, `vrbounce.wrl`). If the string is not a valid `vrworld` file name, Orbisnap tries to interpret the string as the name of the host that is running the Simulink 3D Animation server.

`-q=end_time` — (Optional) Orbisnap ends when virtual scene time equals `end_time`.

`-h` — (Optional) Orbisnap displays the Orbisnap command-line help.

## See Also

### Related Examples

- "Install Orbisnap" on page 8-3
- "Start Orbisnap" on page 8-5
- "Navigate Using Orbisnap" on page 8-9

### More About

- "Orbisnap Viewer" on page 8-2
- "Orbisnap Interface" on page 8-6

## Orbisnap Interface

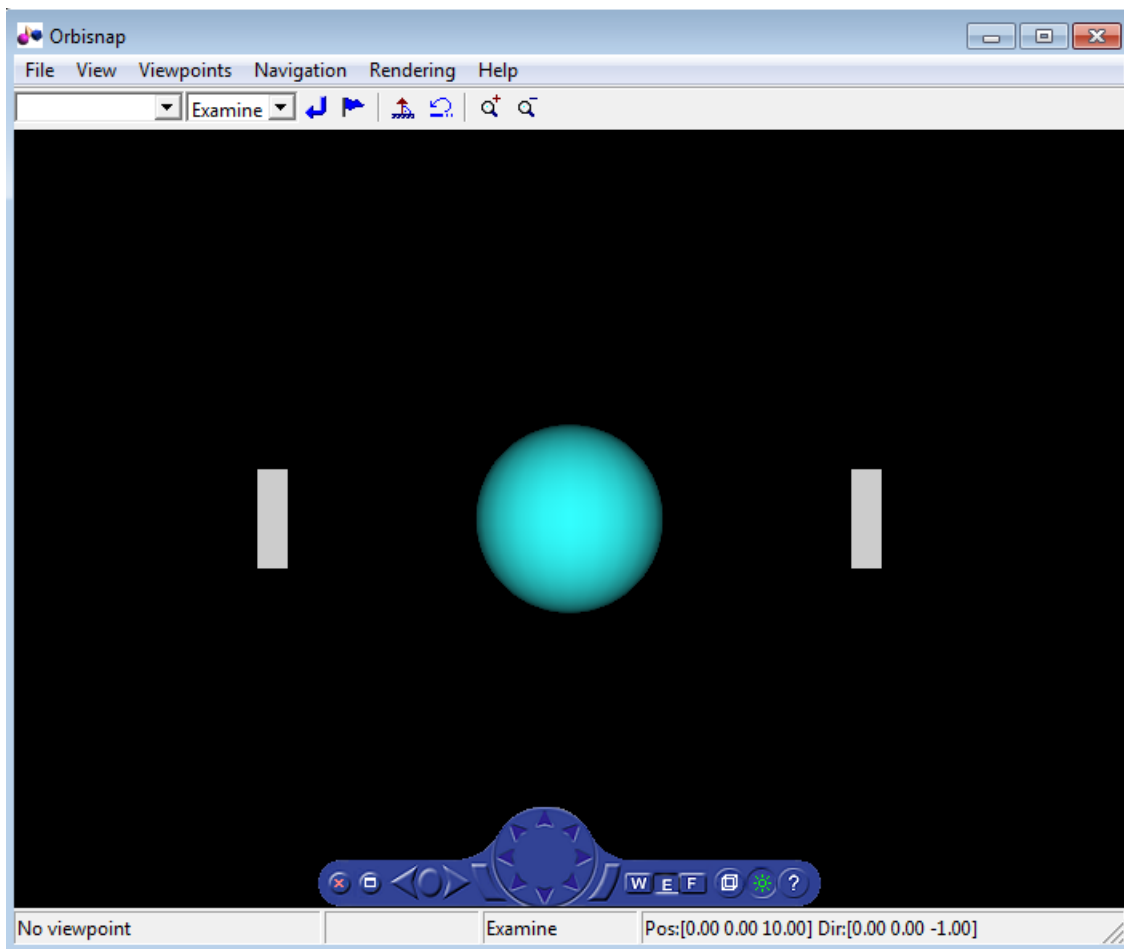
### In this section...

“Menu Bar” on page 8-7

“Toolbar” on page 8-7

“Navigation Panel” on page 8-8

Here is Orbisnap with a virtual world displayed.



Orbisnap provides much of the functionality of the Simulink 3D Animation Viewer. Using the menu bar, toolbar, and navigation panel, you can:

- Customize the Orbisnap window
- Manage virtual world viewpoints
- Manage scene rendering

You cannot

- Open an editor for the virtual world

- Open another window for the virtual world
- Simulate the world (start/stop the model)
- Record or manage animation files


## Menu Bar





The Orbisnap menu bar has the following menus:

- **File** — General file operation options, including,
  - **Open** — Invokes a browser that you can use to browse to the virtual world you want to visualize.
  - **Connect to server** -- Allows you to connect to a Simulink 3D Animation server. Enter the IP address or hostname of the host computer running the Simulink 3D Animation server (127.0.0.1 by default) and the port number at which the Simulink 3D Animation server is listening (8124 by default).
  - **Reload** — Reloads the saved virtual world. If you have created any viewpoints in this session, they are not retained unless you have saved those viewpoints with the **Save As** option.
  - **Save As** — Allows you to save the virtual world.
  - **Close** — Closes the Orbisnap window.
- **View** — Enables you to customize Orbisnap, including,
  - **Toolbar** — Toggles the toolbar display.
  - **Status Bar** — Toggles the status bar display at the bottom of Orbisnap. This display includes the current viewpoint, simulation time, navigation method, and the camera position and direction.
  - **Navigation Zones** — Toggles the navigation zones on/off (see “Navigate Using Orbisnap” on page 8-9 for a description of how to use navigation zones).
  - **Navigation Panel** — Controls the display of the navigation panel, including toggling it.
  - **Triad** — Shows red, green, and blue arrows that are parallel to the orientation of global x, y, and z coordinate axes.
  - **Zoom In/Out** — Zooms in or out of the world view.
  - **Normal (100%)** — Returns the zoom to normal (initial viewpoint setting).
  - **Fullscreen Mode** — Displays the viewer in full-screen mode.
- **Viewpoints** — Manages the virtual world viewpoints.
- **Navigation** — Manages scene navigation.
- **Rendering** — Manages scene rendering.
- **Help** — Displays the Help browser for Orbisnap.

## Toolbar

The Orbisnap toolbar has buttons for some of the more commonly used operations available from the menu bar. These buttons include:

- Drop-down list that displays all the viewpoints in the virtual world
- Return to viewpoint button 

- Create viewpoint button 
- Straighten up button 
- Drop-down list that displays the navigation options Walk, Examine, and Fly
- Undo move button 
- Zoom in/out buttons 

## Navigation Panel

The Orbisnap navigation panel has navigation controls for some of the more commonly used navigation operations available from the menu bar.

The navigation panel controls include from left to right:

- Hide panel — Toggles the navigation panel.
- Full-screen mode— Uses the whole screen for Orbisnap.
- Next/previous viewpoint — Left and right arrows toggles through the list of viewpoints.
- Return to default viewpoint — Returns focus to original default viewpoint.
- Slide left/right — Buttons to the left and right of the navigation wheel slide the view left or right.
- Navigation wheel — Moves view in one of eight directions.
- Navigation method — Manages scene navigation (walk, examine, or fly).
- Wireframe toggle — Toggles scene wireframe rendering.
- Headlight toggle — Toggles camera headlight.
- Help — Invokes the Orbisnap online help.

## See Also

### Related Examples

- “Install Orbisnap” on page 8-3
- “Start Orbisnap” on page 8-5
- “Navigate Using Orbisnap” on page 8-9
- “View Animations or Virtual Worlds with Orbisnap” on page 8-12

### More About

- “Orbisnap Viewer” on page 8-2



## Navigate Using Orbisnap

You can navigate around a virtual world using the menu bar, toolbar, navigation panel, mouse, and keyboard.

**Navigation view** — You can change the camera position. From the menu bar, select the **Navigation** menu **Straighten Up** option. This option resets the camera so that it points straight ahead.

**Navigation methods** — Navigation with the mouse depends on the navigation method you select and the navigation zone you are in when you first click and hold down the mouse button. You can set the navigation method using one of the following:

- From the menu bar, select the **Navigation** menu **Method** option. This option provides three choices, **Walk**, **Examine**, or **Fly**. See the table Orbisnap Mouse Navigation.
- From the toolbar, select the drop-down menu that displays the navigation options **Walk**, **Examine**, and **Fly**.
- From the navigation panel, click the **W**, **E**, or **F** buttons.
- From the keyboard, press **Shift+W**, **Shift+E**, or **Shift+F**.

**Navigation zones** — You can view the navigation zones for a virtual world through the menu bar or keyboard.

From the menu bar, select the **View** menu **Navigation Zones** option. The virtual world changes as the navigation zones are toggled on and appear in the virtual world. Alternatively, from the keyboard, press the **F7** key.

The following table summarizes the behavior associated with the movement modes and navigation zones when you use your mouse to navigate through a virtual world. Turn on the navigation zones and experiment by clicking and dragging your mouse in the different zones of a virtual world.

### Orbisnap Mouse Navigation

Movement Mode	Zone and Description
Walk	<p><b>Outer</b> -- Click and drag the mouse up, down, left, or right to slide the camera in any of these directions in a single plane.</p> <p><b>Inner</b> -- Click and drag the mouse up and down to move forward and backward. Drag the mouse left and right to turn left or right.</p>
Examine	<p><b>Outer</b> -- Click and drag the mouse up and down to move forward and backward. Drag the mouse left and right to slide left or right.</p> <p><b>Inner</b> -- Click and drag the mouse to rotate the viewpoint around the origin of the scene.</p>
Fly	<p><b>Outer</b> -- Click and drag the mouse to tilt the view either left or right.</p> <p><b>Inner</b> -- Click and drag the mouse to pan the camera up, down, left, or right within the scene.</p> <p><b>Center</b> -- Click and drag the mouse up and down to move forward and backward. Move the mouse left or right to turn in either of these directions.</p>

If your virtual world contains sensors, these sensors take precedence over mouse navigation at the sensor location. In this case, mouse navigation is still possible through the right or middle mouse buttons.

**Keyboard** — You can also use the keyboard to navigate through a virtual world. It can be faster and easier to issue a keyboard command, especially if you want to move the camera repeatedly in a single direction. The following table summarizes the keyboard commands and their associated navigation functions. You do not have to capitalize the letters to perform their intended function.

### Orbisnap Keyboard Navigation

Keyboard Command	Navigation Function
<b>Backspace</b>	Undo move.
<b>F9</b>	Straighten up and make the camera stand on the horizontal plane of its local coordinates.
<b>+/-</b>	Zoom in/out.
<b>F6</b>	Toggle the headlight on/off.
<b>F7</b>	Toggle the navigation zones on/off.
<b>F5</b>	Toggle the wireframe option on/off.
<b>F8</b>	Toggle the antialiasing option on/off.
<b>Esc</b>	Go to default viewpoint.
<b>Home</b>	Return to current viewpoint.
<b>Page Up, Page Down</b>	Move between preset viewpoints.
<b>F10</b>	Toggle camera binding from the viewpoint.
<b>Shift+W</b>	Set the navigation method to Walk.
<b>Shift+E</b>	Set the navigation method to Examine.
<b>Shift+F</b>	Set the navigation method to Fly.
<b>Shift Up/Down Arrow</b>	Move the camera forward and backward.
<b>Up/Down Arrow</b>	Pan the camera up and down.
<b>Left/Right Arrow, Shift+Left/Right Arrow</b>	Pan the camera right and left.
<b>Alt+Up/Down Arrow</b>	Slide up and down.
<b>Alt+Left/Right Arrow</b>	Slide left and right.
<b>Ctrl+Left/Right/Up/Down Arrow</b>	Pressing <b>Ctrl</b> alone acquires the examine lock at the point of intersection between the line perpendicular to the screen, coming through the center of the Orbisnap window, and the closest visible surface to the camera. Pressing the arrow keys without releasing <b>Ctrl</b> rotates the viewpoint about the acquired center point.
<b>Shift+Alt+Left/Right Arrow</b>	Tilt the camera right and left.

## **See Also**

### **Related Examples**

- “Install Orbisnap” on page 8-3
- “Start Orbisnap” on page 8-5
- “View Animations or Virtual Worlds with Orbisnap” on page 8-12
- “View Virtual Worlds Remotely with Orbisnap” on page 8-13

### **More About**

- “Orbisnap Viewer” on page 8-2
- “Orbisnap Interface” on page 8-6

## View Animations or Virtual Worlds with Orbisnap

This topic assumes that you have a prerecorded WRL animation file or an existing virtual world file. This procedure uses a file named `vr_bounce_anim.wrl`.

- 1 Start Orbisnap. For example, in Windows double-click `orbisnap.bat` in `matlabroot\toolbox\sl3d\orbisnap\bin`.

This file is an Orbisnap starter file that calls the Orbisnap executable. Orbisnap is displayed.

- 2 In Orbisnap, select **File > Open**.

A file browser is displayed.

- 3 Browse to the folder that contains the prerecorded WRL animation file or virtual world you want to view.
- 4 Select the virtual world or prerecorded WRL file you want to view.
- 5 Click **Open**.

The file is displayed. If the file is an animation file, the simulation begins.

- 6 To close Orbisnap, select **File > Close**.

Using the menus, toolbar, and navigation panel, you can perform many of the same operations on the virtual world that you can with the Simulink 3D Animation Viewer. See “Orbisnap Interface” on page 8-6 for an overview of the Orbisnap interface. See “Start Orbisnap” on page 8-5 for a description of the Orbisnap command-line options.

### See Also

#### Related Examples

- “Install Orbisnap” on page 8-3
- “Start Orbisnap” on page 8-5
- “View Virtual Worlds Remotely with Orbisnap” on page 8-13

#### More About

- “Orbisnap Viewer” on page 8-2
- “Orbisnap Interface” on page 8-6

## View Virtual Worlds Remotely with Orbisnap

To view virtual worlds from the Simulink 3D Animation server in Orbisnap, you must have

- The MATLAB software running a Simulink 3D Animation server session
- The version of the Simulink 3D Animation server to which you want to connect must be compatible with the Orbisnap version you are running. For example, you cannot connect Orbisnap to Simulink 3D Animation software Version 3.1.
- Network access between the client computer (running Orbisnap) and host computer (running MATLAB and Simulink 3D Animation server)

---

**Note** If you expect Orbisnap to access a virtual world on the Simulink 3D Animation server from a remote computer, make that virtual world available for internet viewing. In the Simulink 3D Animation Viewer for the virtual world you want to make available, select **Simulation > Block Parameters**, select the **Allow viewing from the Internet** check box, then click **OK**.

---

Note the following when using Orbisnap remotely:

- Although you can visualize a virtual world from the Simulink 3D Animation server in Orbisnap, any navigation or rendering in one viewer is not reflected in the other. For example, any navigation you do on the virtual world in Orbisnap is not reflected in the virtual world in the Simulink 3D Animation Viewer.
- You cannot start or stop a simulation of the virtual world in Orbisnap. You can see a simulation on Orbisnap only if the virtual world is simulated in the Simulink 3D Animation server.
- The simulation can be slow when you connect Orbisnap remotely to the Simulink 3D Animation server.

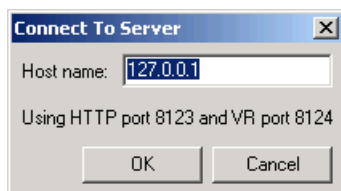
- 1 Start Orbisnap. For example, in Windows, double-click `orbisnap.bat` in `matlabroot\toolbox\sl3d\orbisnap\bin`.

This file is an Orbisnap starter file that calls the Orbisnap executable. Orbisnap is displayed.

- 2 In Orbisnap, select **File > Connect to Server**.

The Connect to Server dialog box is displayed.

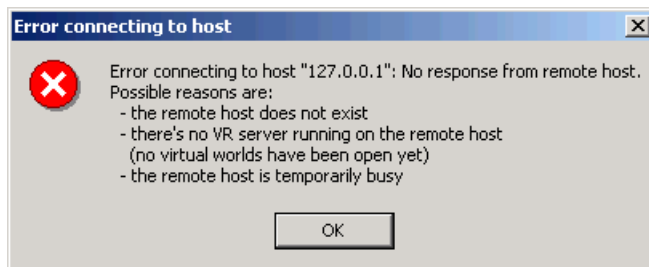
- 3 Enter the IP address or hostname of the host computer running the Simulink 3D Animation server (127.0.0.1 by default). The HTTP port number is 8123 by default and the port number at which the Simulink 3D Animation server is listening is 8124 by default.



The Choose a world dialog box is displayed. This dialog box lists all the virtual worlds that are currently active on the Simulink 3D Animation server.



If no virtual world has ever been opened in this session of the Simulink 3D Animation server, Orbisnap displays a message. If you see this message, contact your counterpart running the Simulink 3D Animation server to better synchronize your activities. A virtual world must be fully active on the Simulink 3D Animation server for Orbisnap to access it remotely.



- 4 Select a virtual world.
- 5 Click **OK**.

Orbisnap displays the selected virtual world of the remote Simulink 3D Animation server.

- 6 Navigate and render the virtual world as you want.
- 7 To close Orbisnap, select **File > Close**.

Using the menus, toolbar, and navigation panel, you can perform many of the same operations on the virtual world that you can with the Simulink 3D Animation Viewer. See “Orbisnap Interface” on page 8-6. See “Start Orbisnap” on page 8-5 for a description of the Orbisnap command-line options.

## See Also

### Related Examples

- “Install Orbisnap” on page 8-3
- “Start Orbisnap” on page 8-5
- “View Animations or Virtual Worlds with Orbisnap” on page 8-12

### More About

- “Orbisnap Viewer” on page 8-2
- “Orbisnap Interface” on page 8-6

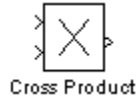
# Blocks

---

# Cross Product

Cross product of two 3-D vectors

**Library:** 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

## 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” on page 3-2

**Introduced in R2006a**

## Joystick Input

Process input from asynchronous joystick device

**Library:** 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

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.

## **See Also**

Space Mouse Input | vrjoystick | vrspace mouse

## **Topics**

“Connect Virtual Worlds and Models” on page 3-2

**Introduced before R2006a**

## MATLAB to VR Coordinates

Convert MATLAB coordinates to VR coordinates

**Library:** 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” on page 1-12.

### 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`

### 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**

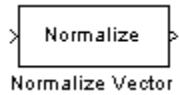
“Virtual World Coordinate System” on page 1-12

**Introduced in R2019a**

# Normalize Vector

Output unit vector parallel to input vector

**Library:** 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.

## See Also

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

## Topics

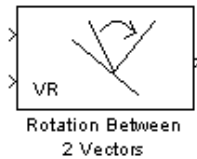
“Connect Virtual Worlds and Models” on page 3-2

## Introduced in R2006a

# Rotation Between 2 Vectors

Virtual world rotation between two 3-D vectors

**Library:** 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

## 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” on page 3-2

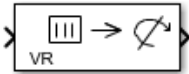
**Introduced in R2006a**



# Rotation Matrix to VR Rotation

Convert rotation matrix to axis/angle rotation

**Library:** 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  $R$ :  $R^T R = R R^T = I$ , 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  $\epsilon = 1e-12$ .

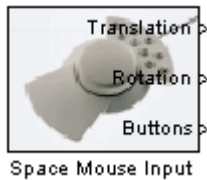
**See Also**

VR Rotation to Rotation Matrix

**Introduced in R2019a**

# Space Mouse Input

Process input from space mouse device



## Library

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.

## Data Type Support

The Space Mouse Input block outputs signals of type double.

## Parameters

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

**Output Type** — 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** — 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 rotation movement** — Fixes the positions at the initial values, allowing you to change rotations only.

**Disable position movement** — Fixes the rotations at initial values, allowing you to change positions only.

**Normalize output angle** — 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** — Determines whether you can limit the upper and lower positions of the mouse.

**Position sensitivity** — Mouse sensitivity for translations. Higher values correspond to higher sensitivity.

**Rotation sensitivity** — Mouse sensitivity for rotations. Higher values correspond to higher sensitivity.

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

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

**Lower position limit** — Position coordinates for the lower limit of the mouse.

**Upper position limit** — Position coordinates for the upper limit of the mouse.

## **See Also**

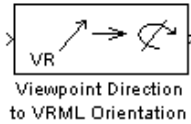
- Space Mouse Input
- Manipulator with SpaceMouse
- “Connect Virtual Worlds and Models” on page 3-2
- `vrjoystick`
- `vrspacemouse`

**Introduced in R2007b**

# Viewpoint Direction to VR Orientation

Convert viewpoint direction to virtual world orientation

**Library:** 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.

## 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

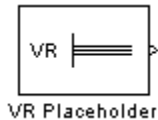
[Manipulator with SpaceMouse](#)

“Connect Virtual Worlds and Models” on page 3-2

**Introduced in R2006a**

# VR Placeholder

Send unspecified value to Simulink 3D Animation block



## Library

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.

## Data Type Support

A VR Placeholder block outputs signals of type `double`.

## Parameters

**Output Width** — Length of the vector containing placeholder signal values.

## See Also

- VR Signal Expander

**Introduced before R2006a**

## VR RigidBodyTree

Visualize Robotics System Toolbox RigidBodyTree objects in Simulink

**Library:** 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” on page 6-9.

#### 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.

**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.

## 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

**Introduced in R2018b**

## VR to MATLAB Coordinates

Convert VR coordinates to MATLAB coordinates

**Library:** 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” on page 1-12.

### 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`

### 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**

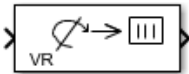
“Virtual World Coordinate System” on page 1-12

**Introduced in R2019a**

## VR Rotation to Rotation Matrix

Convert axis/angle rotation to rotation matrix

**Library:** Simulink 3D Animation / Utilities



### Description

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

### Ports

#### Input

##### Input 1 – Axis/Angle rotation

4-element vector

Input rotation, specified 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.

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  $R$ :  $R^T R = R R^T = I$ , 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  $\epsilon = 1e-12$ .

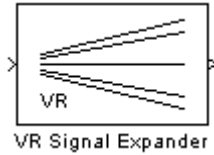
**See Also**

Rotation Matrix to VR Rotation

**Introduced in R2019a**

## 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

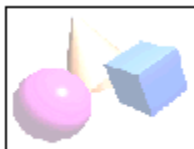
- VR Placeholder

**Introduced before R2006a**

## VR Sink

Write data from Simulink model to virtual world

**Library:** Simulink 3D Animation



VR Sink

### 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

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.

Example:

### 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.

## See Also

[VR Source](#) | [VR To Video](#)

### Topics

“Connect Virtual Worlds and Models” on page 3-2

“Detect Object Collisions” on page 5-23

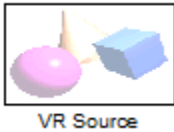
Foucault Pendulum Model with Virtual Reality Scene

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

**Introduced before R2006a**

## VR Source

Read data from virtual world to Simulink model



## Library

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” on page 5-20 and “Detect Object Collisions” on page 5-23.
- 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 `vr_crane_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.

## Data Type Support

A VR Source block outputs signals of type double.

## Parameters

**Source file** — Virtual world 3D file name specifying the virtual world that connects to this block. 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 the **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 the **Edit** button to launch the default virtual world editor with the source file open.
- Click the **View** button to view the world in the Simulink 3D Animation Viewer or a Web browser.
- Click the **Reload** button reloads the world after you change it.

**Open Viewer automatically** — If you select this check box, the default virtual world viewer displays the virtual world after loading the Simulink model.

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

**Description** — Description that 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** — Enter the sample time or -1 for inherited sample time.

---

**Note** To achieve a smooth simulation, MathWorks recommends that you explicitly set the **Sample time** parameter. You can change the value of this parameter to achieve the specific visual experience you want.

---

**Allow variable-size output signals** — Specify the type of signals allowed out of this port.

By default, the VR Source block does not allow variable-size signals. If you enable this parameter, then the VR Source block allows variable-size signals for fields that can change dimensions during simulation. These fields include MFxxx fields that can have a variable number of elements (typically, MFFloat or MFVec3f). The SFImage is the only SFxxx field that can map to a variable-size signal. For details about these data types, see “Field Data Types” on page 5-30.

---

**Note** The signal dimensions of a variable-size output signal of a VR Source block must be the same size as, or smaller than, the initial state of the signal.

---

**Virtual World Tree** — 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 MATLAB 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 readable values have check boxes. Use these check boxes to select the fields that you want the Simulink software to monitor and to use to input values. For each field that you select in the **Virtual World Tree** box, Simulink creates an output port in the VR Source block. Simulink creates the output ports in the same order as the selected fields appear in the virtual world 3D file.

For Transform nodes, the tree includes an **Extensions** branch that contains two Simulink 3D Animation extensions for converting rotation and translation values into global coordinates: `rotation_abs` and `translation_abs`. These fields are read-only, and do not appear in the 3D World Editor **tree structure** pane. For more information, see “Input Virtual World Data to a Model” on page 3-6.

Fields whose values cannot be read (because their parent nodes do not have names, or because their values cannot be imported to Simulink) have an X-shaped icon.

**Show node types** — If you select this check box, node types are shown in the virtual scene tree.

**Show field types** — If you select this check box, field types are shown in the virtual scene tree.

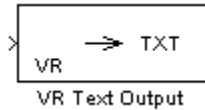
## See Also

- VR Sink
- VR To Video
- “Read Sensor Values Using VR Source Blocks” on page 5-21
- “Input Virtual World Data to a Model” on page 3-6
- “Detect Object Collisions” on page 5-23
- “Virtual World Data Types” on page 5-30
- `vrcrane_panel`

**Introduced in R2011b**

# 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

## **Extended Capabilities**

### **C/C++ Code Generation**

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

Actual data type or capability support depends on block implementation.

**Introduced in R2006b**

## VR To Video

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

**Library:** 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.

Example:

**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.

**See Also**

VR Sink | VR To Video

**Topics**

“Virtual World Data Types” on page 5-30

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

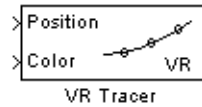
**Introduced in R2007b**



# VR Tracer

Trace trajectory of object in associated virtual scene

**Library:** 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

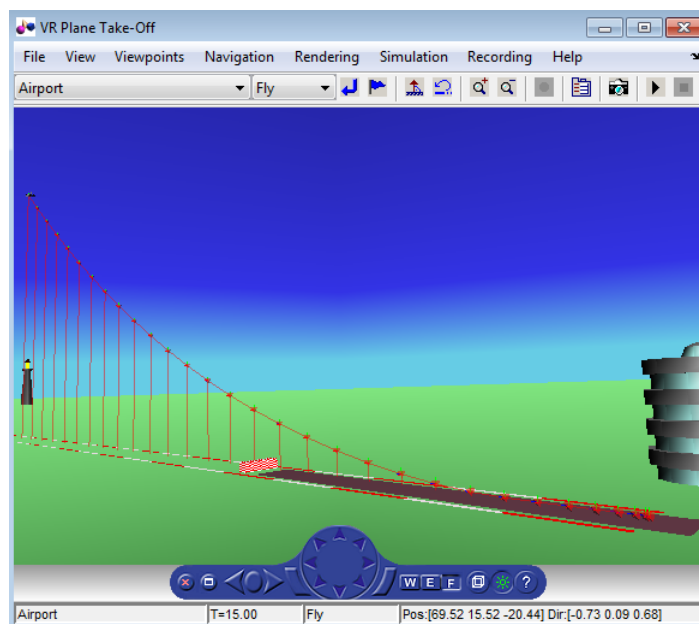
off (default) | on

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.

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

**Introduced in R2008b**

# Functions

---

## stl2vrm1

Convert STL file to virtual world file

### Syntax

```
stl2vrm1(source)
stl2vrm1(source,destination)
stl2vrm1(source,destination,format)
```

### Description

`stl2vrm1(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 `stl2vrm1` 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 `stl2vrm1` function.

---

`stl2vrm1(source,destination)` creates the converted VRML file in the `destination` folder.

`stl2vrm1(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 `fourbar-Bar1-1.STL` (in `matlab/toolbox/phymod/mech/mechdemos`) to a VRML file and place the resulting file in the current folder.

```
stl2vrm1('fourbar-Bar1-1.STL')
ls

.      ..      fourbar-Bar1-1.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 `fourbar-Bar1-1.STL` (in `matlab/toolbox/phymod/mech/mechdemos`) to an XML-encoded virtual world file and place the resulting file in a folder called `virtualworlds`.

```
stl2vrm1('fourbar-Bar1-1.STL','virtualworlds','x3d')
ls

.          ..      fourbar-Bar1-1.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, `stl2vrm1` 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 `stl2vrm1` 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 `stl2vrm1` 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.

## See Also

`vrcadcleanup` | `vrimport` | `vrphysmod`

## Topics

“Import STL and Physical Modeling XML Files” on page 5-38

“Link to Simulink and Simscape Multibody Models” on page 5-60

## Introduced in R2010b

## 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
<code>'solidworks'</code>	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');
```

## See Also

“Import STL and Physical Modeling XML Files” on page 5-38 | “Link to Simulink and Simscape Multibody Models” on page 5-60 | `stl2vrm` | `vrphysmod`

**Introduced in R2009a**

## 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” on page 7-18.

**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 — 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” on page 7-18.

**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
<code>left</code>	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.
<code>bottom</code>	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.
<code>width</code>	Distance between the right and left inner edges of the canvas.
<code>height</code>	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**

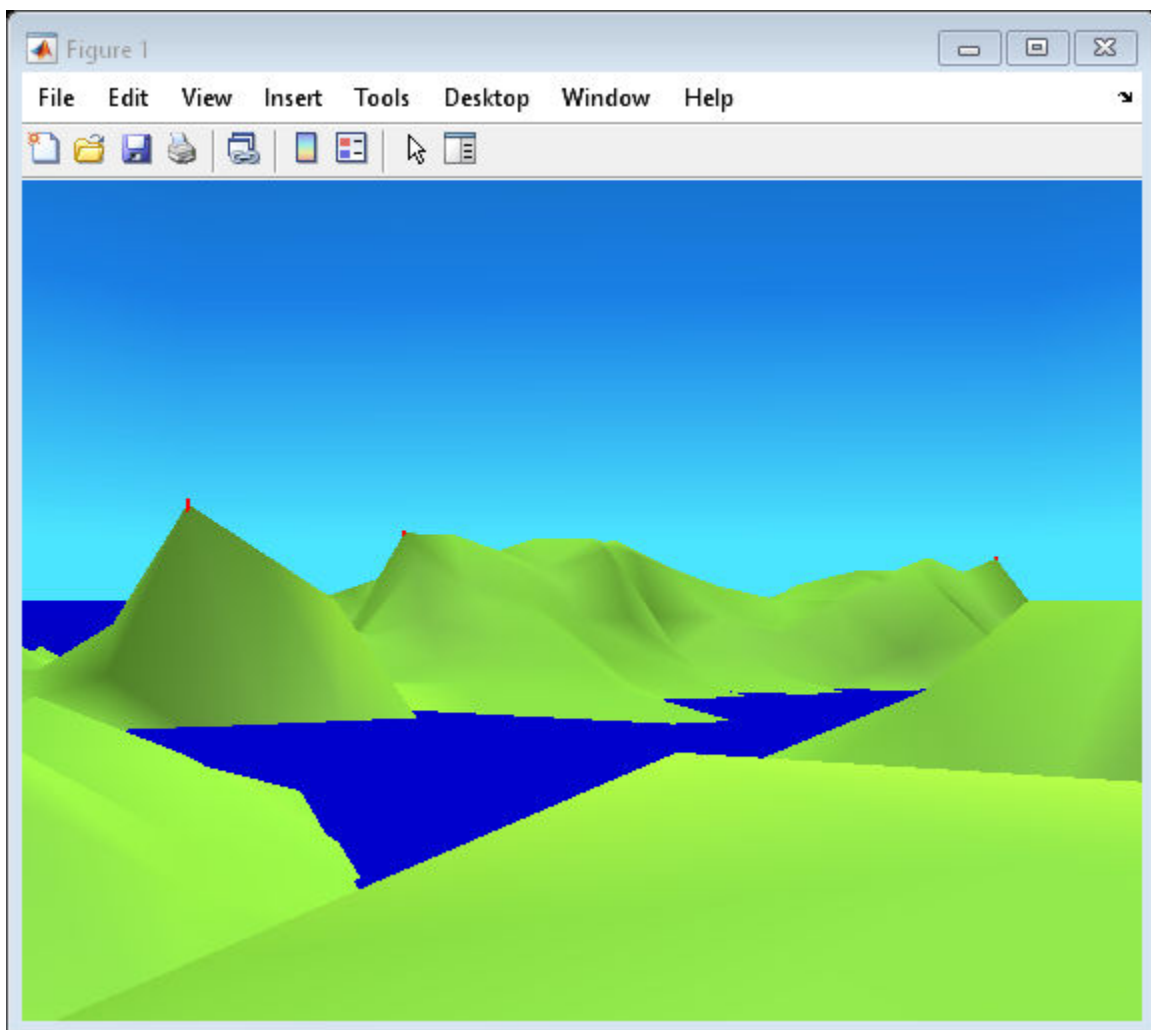
## Create a Canvas That Displays in a Figure

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.

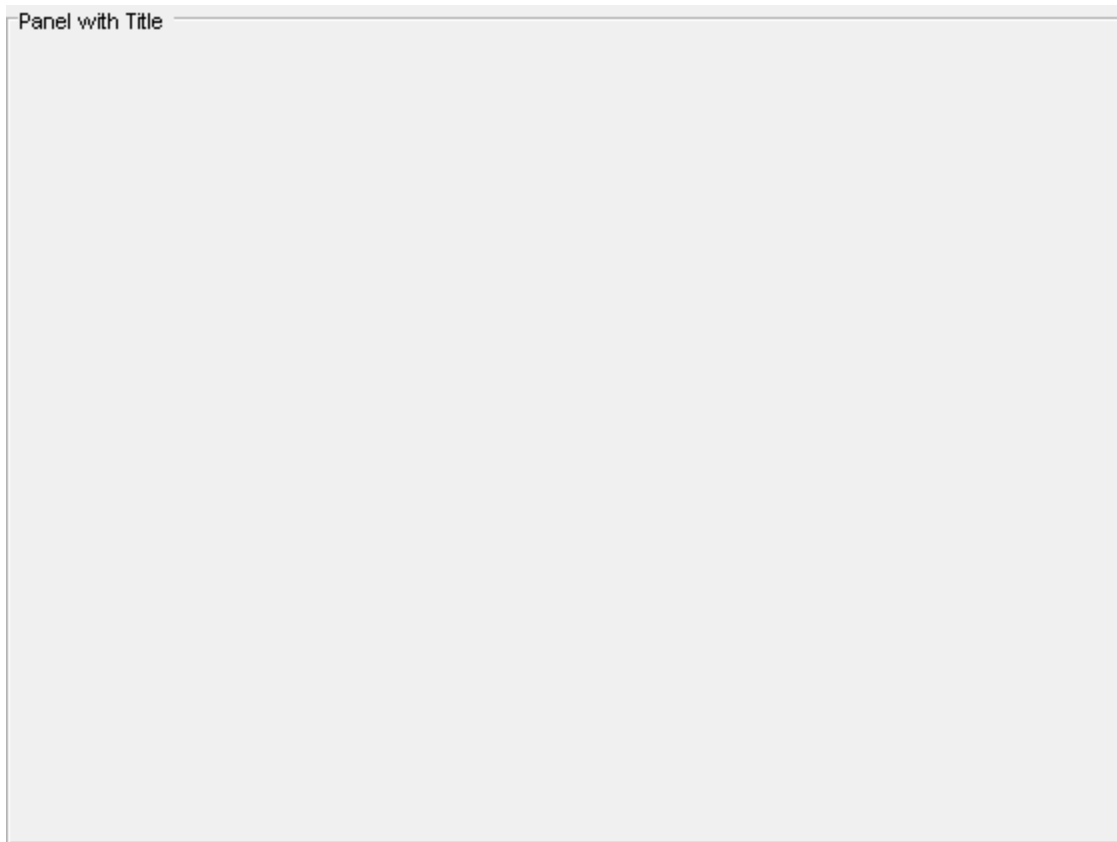
```
fig = figure;  
myCanvas = vr.canvas(myWorld, 'Parent', fig, 'Units', ...  
    'normalized', 'Position', [0 0 1 1]);
```



## 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');
```

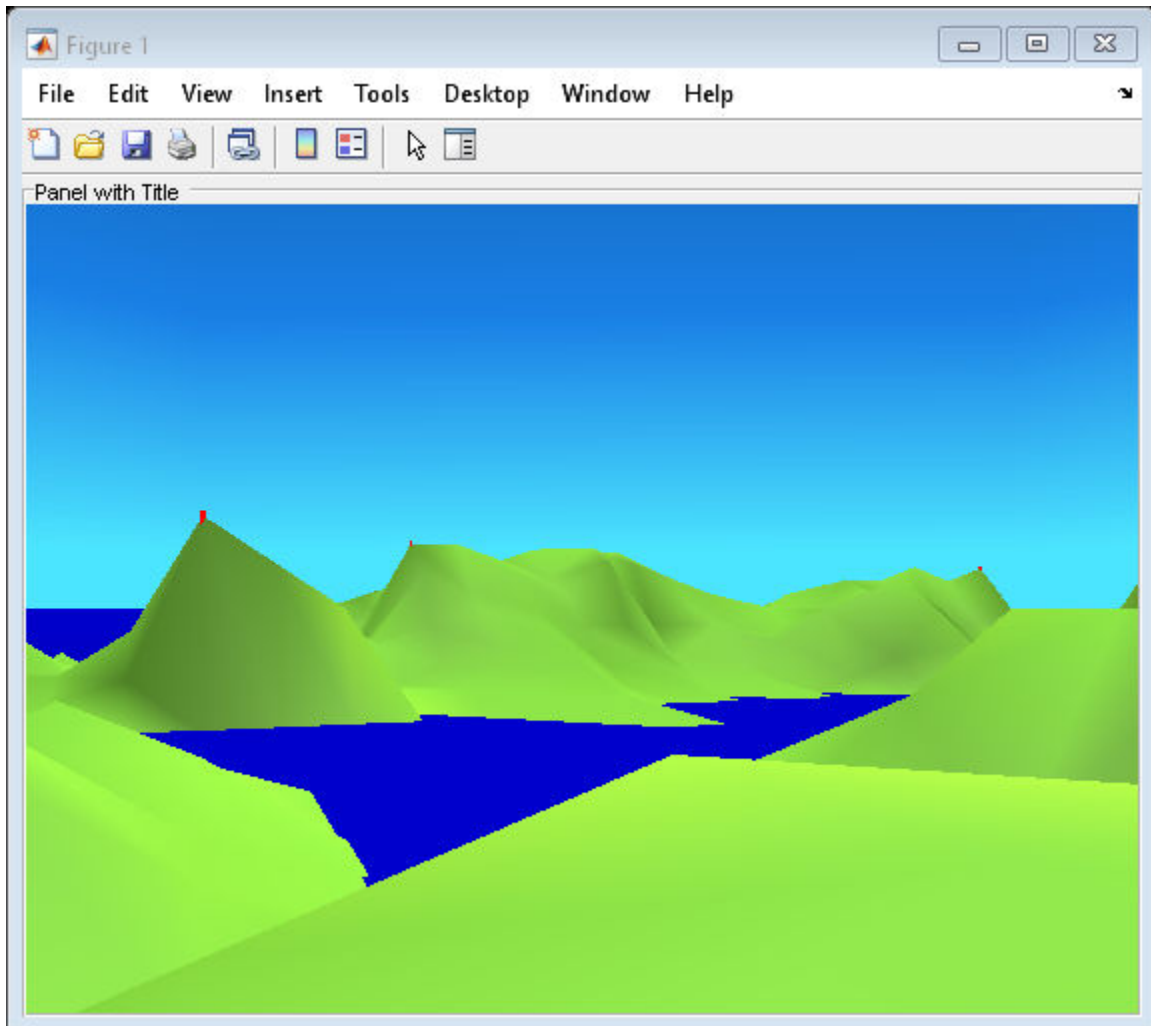


Create and open a virtual world.

```
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.wrl');
open(vrmountWorld);
```

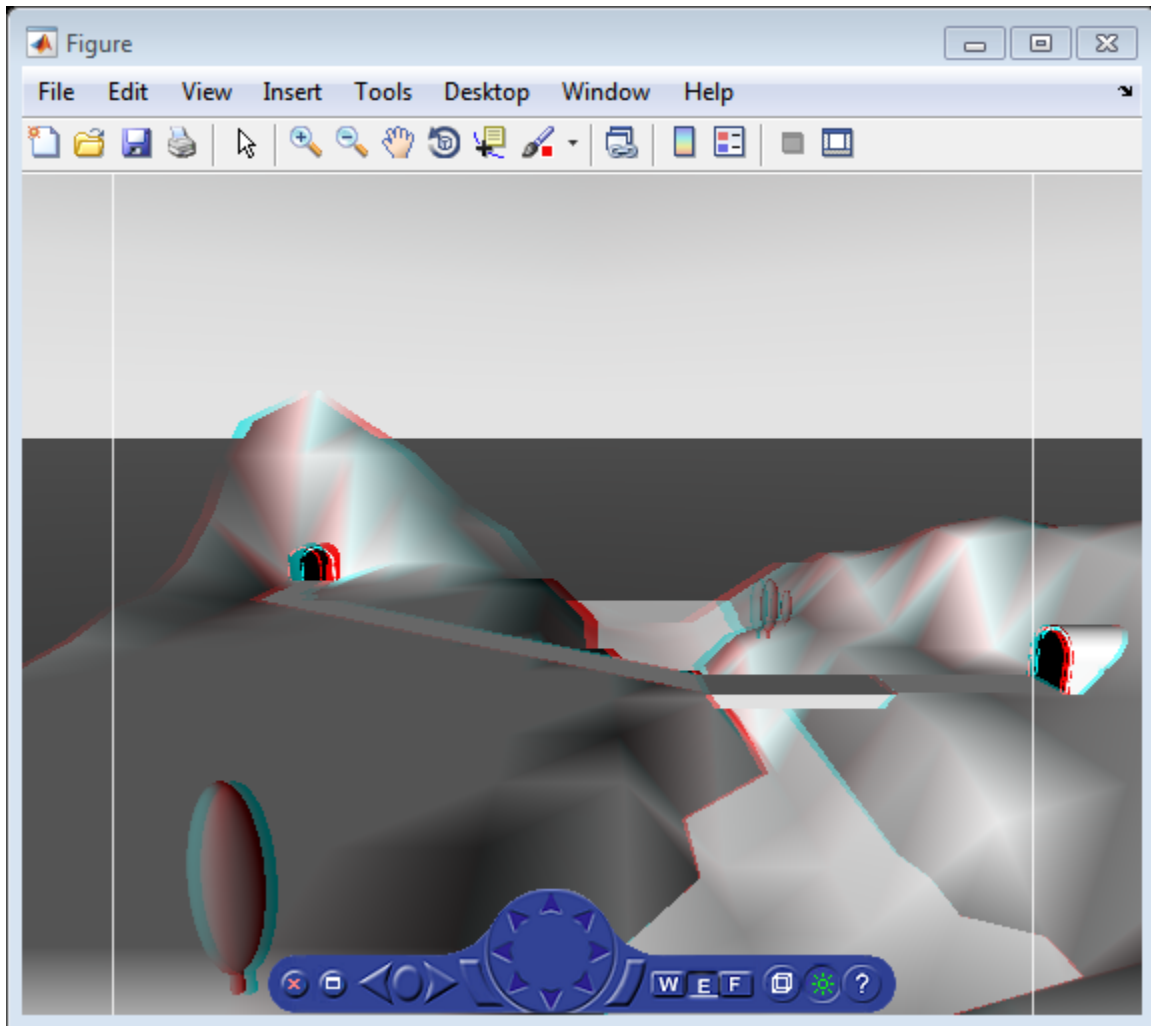
Create a `vr.utils.stereo3d` object to use to specify stereoscopic vision properties.

```
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'  
    Lighting: 'on'  
    MaxTextureSize: 'auto'  
    NavPanel: 'opaque'  
    NavMode: 'examine'  
    NavSpeed: 'normal'  
    NavZones: 'on'  
    Position: [0 0 1 1]  
    Sound: 'on'  
    Stereo3D: 'anaglyph'  
    Stereo3DCameraOffset: 0.2500  
    Stereo3DHIT: 0.0200  
    Textures: 'on'  
    Tooltips: 'on'  
    Transparency: 'on'  
    Triad: 'none'  
    Units: 'normalized'  
    Viewpoint: 'View 1 - Observer'  
    Wireframe: 'off'  
    ZoomFactor: 1  
    DeleteFcn: []  
    CameraDirectionAbs: [0 -0.1987 -0.9801]  
    CameraPositionAbs: [20.2500 8 50]  
    CameraUpVectorAbs: [0 0.9801 -0.1987]  
    Parent: [1x1 Figure]  
    World: [1x1 vrworld]
```



## See Also

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

## Topics

"Create `vrworld` Object for a Virtual World" on page 4-2

"Interact with Virtual Reality Worlds"

"View a Virtual World in Stereoscopic Vision" on page 7-45

**Introduced before R2006a**

## 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

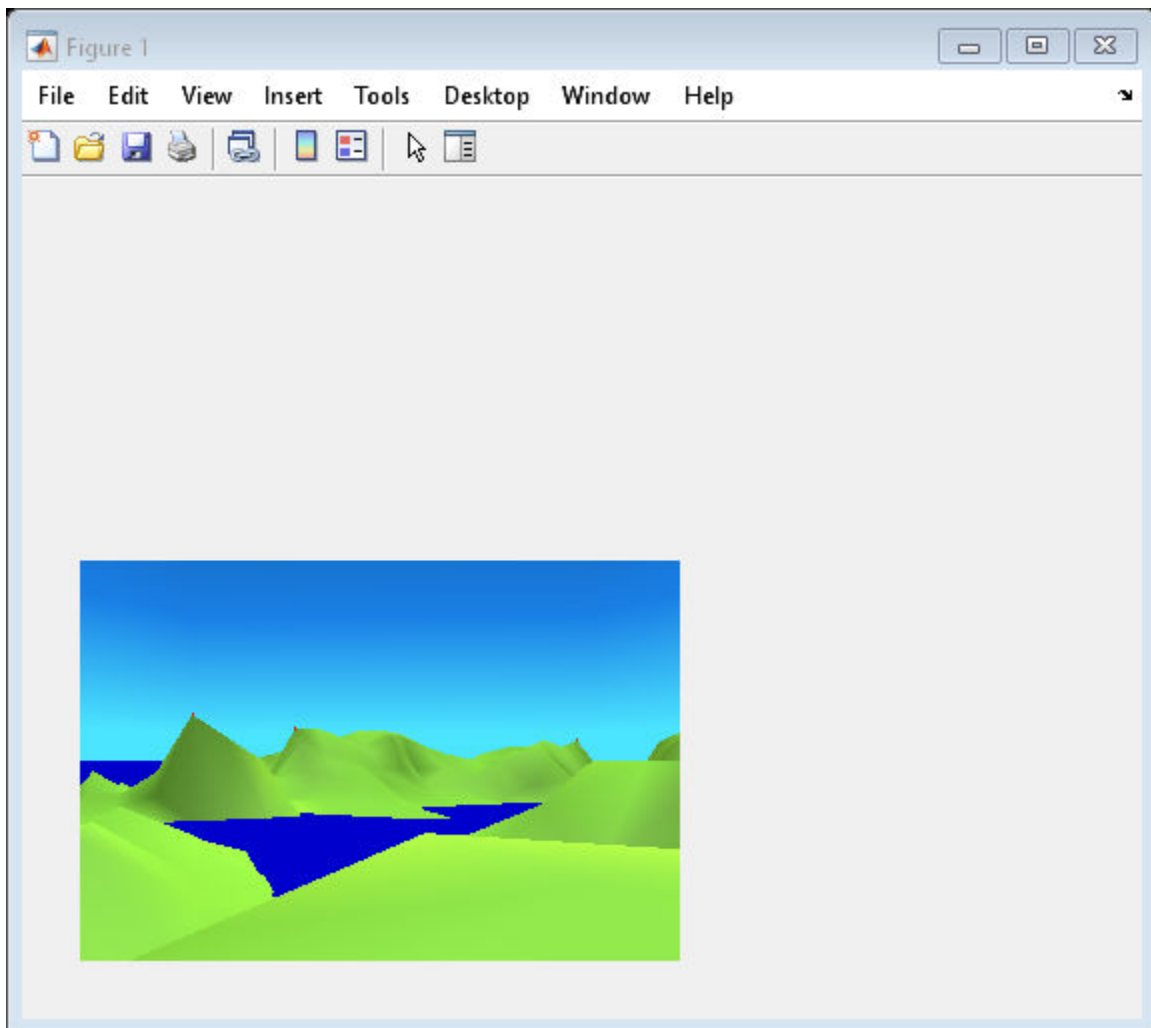
Create and open a `vrworld` object and associate it with the virtual world `vr\lights.wrl`.

```
lights_world = vrworld('vr\lights');  
open(lights_world);
```

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

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





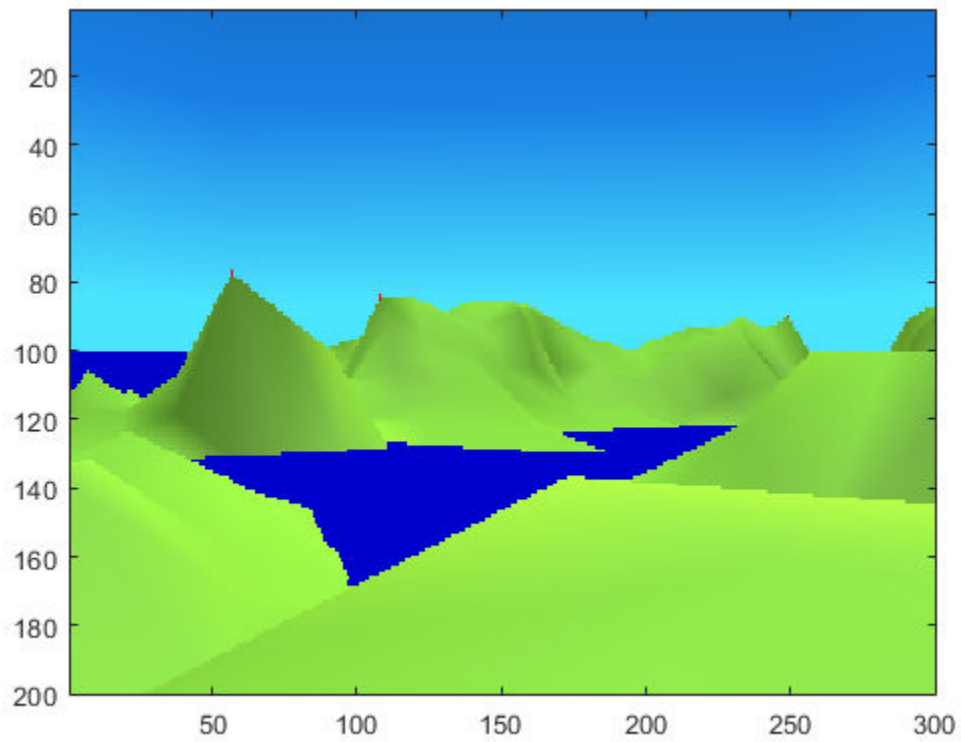
```
vrdrawnow;
```

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);
```



**See Also**

`vrworld | image`

**Topics**

“Interact with Virtual Reality Worlds”

“Create `vrworld` Object for a Virtual World” on page 4-2

**Introduced before R2006a**

## 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.

### See Also

`vrworld` | `vrworld/delete` | “Close and Delete a `vrworld` Object” on page 4-9

**Introduced before R2006a**

## **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.

### **See Also**

`close` | “Close and Delete a vrworld Object” on page 4-9

**Introduced before R2006a**

## 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`

## 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” on page 1-12

## Introduced in R2019a

## 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.



**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” on page 1-12

**Introduced in R2019a**

## **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`).

### **See Also**

`vrori2dir` on page 10-96 | `vrrotmat2vec` on page 10-108 | `vrrotvec` on page 10-107 | `vrrotvec2mat` on page 10-109

**Introduced in R2007b**

# 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.

## See Also

vrworld/edit | vrworld/open | “Open a Virtual World with MATLAB” on page 4-3 | “Interact with a Virtual World with MATLAB” on page 4-5

**Introduced before R2006a**

## 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')
```

### See Also

`vrworld/edit` | `vrworld/open` | “Open a Virtual World with MATLAB” on page 4-3 | “Interact with a Virtual World with MATLAB” on page 4-5

**Introduced in R2012b**

## 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

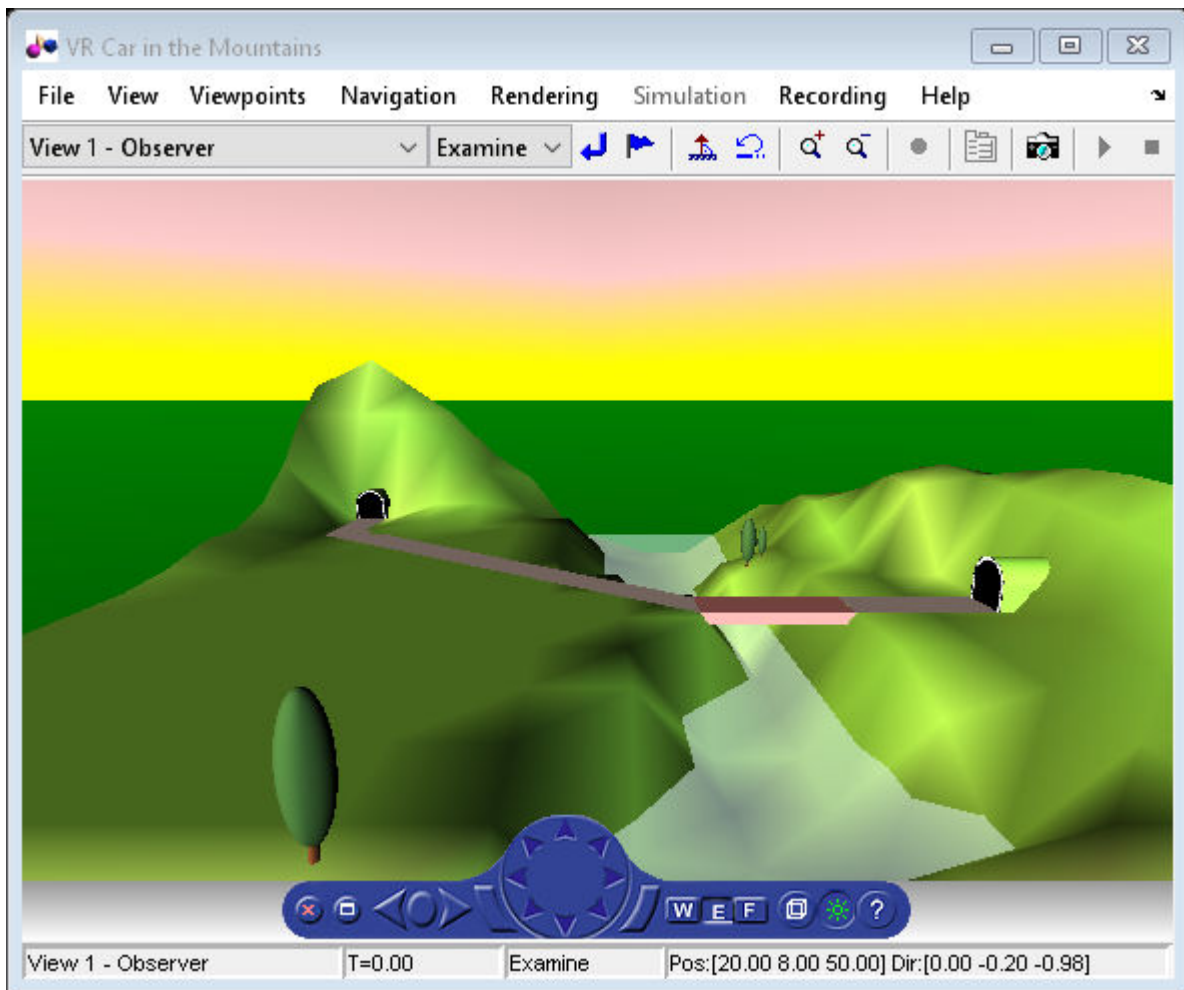
<code>capture</code>	Capture virtual reality figure image
<code>capture</code>	Capture virtual reality figure image
<code>close</code>	Close virtual reality figure
<code>get</code>	Return property value of <code>vrfigure</code> object
<code>isvalid</code>	Check validity of <code>vrfigure</code> object handles
<code>set</code>	Set property values of <code>vrfigure</code> object
<code>set</code>	Set property values of <code>vrfigure</code> object

### Examples

#### Create and Display a `vrworld` Object

Create a `vrworld` object that is associated with the virtual world `vrmount.wr1`. Open and view the virtual world.

```
myworld = vrworld('vrmount');
open(myworld);
f = vrfigure(myworld);
```



## See Also

`vr.utils.stereo3d` | `vr.canvas`

## Topics

“Create vrworld Object for a Virtual World” on page 4-2

“Interact with Virtual Reality Worlds”

“View a Virtual World in Stereoscopic Vision” on page 7-45

**Introduced before R2006a**

## 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

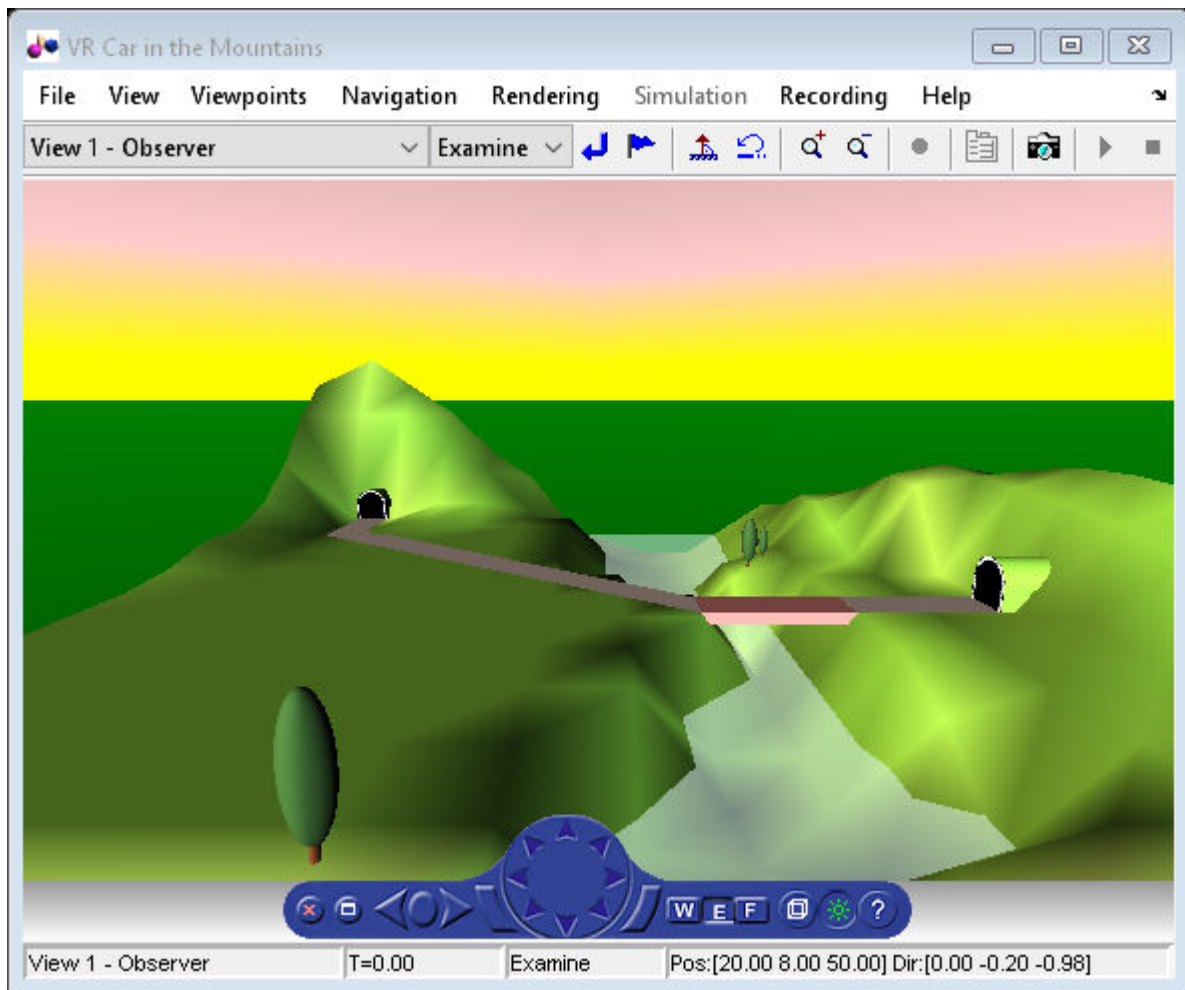
Create and open a `vrworld` object and associate it with the virtual world `vrmount.wrl`.

```
myworld = vrworld('vrmount');  
open(myworld);
```

View the virtual world in the Simulink® 3D Animation™ Viewer.

```
f = vrfigure(myworld);
```



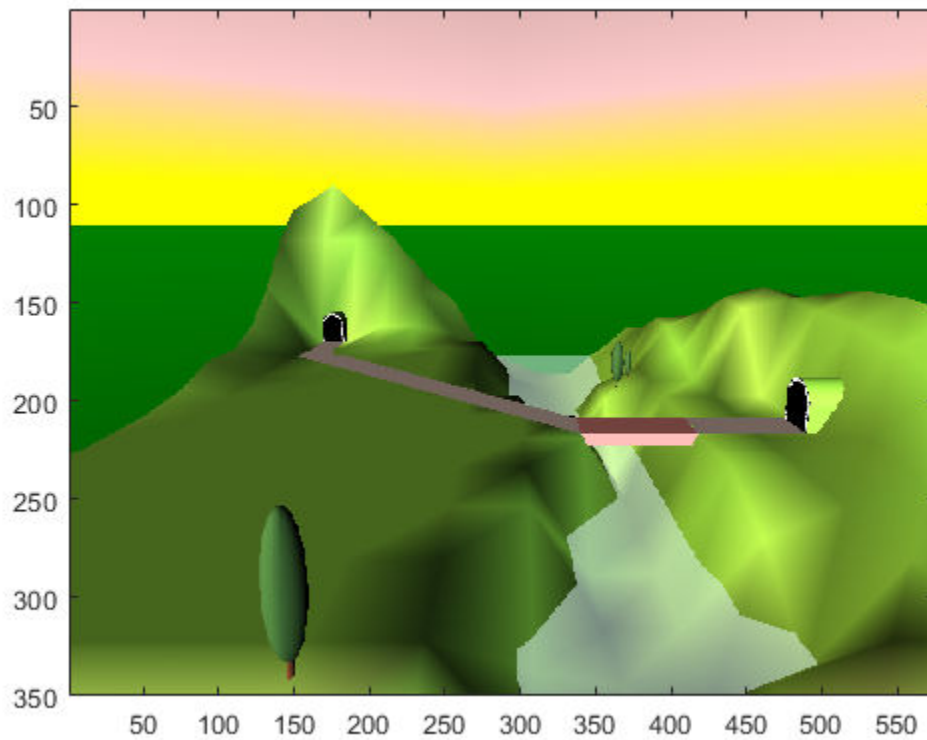


Create an RGB image of the figure.

```
image_capture = capture(f);
```

Display the RGB figure image in a MATLAB® figure window.

```
image(image_capture);
```



**See Also**

`vrfigure` | `vrworld` | `isvalid` | `vrnode/isvalid` | `image`

**Topics**

“Interact with Virtual Reality Worlds”

“Create `vrworld` Object for a Virtual World” on page 4-2

**Introduced before R2006a**

# 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)
```

## See Also

`vrfigure` | `vrworld`

### Topics

“Interact with Virtual Reality Worlds”

“Create `vrworld` Object for a Virtual World” on page 4-2

**Introduced before R2006a**

## 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.

<b>vrfigure Property</b>	<b>Meaning</b>
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" on page 7-18.
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" on page 4-14.
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.

<b>vrfigure Property</b>	<b>Meaning</b>
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.
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

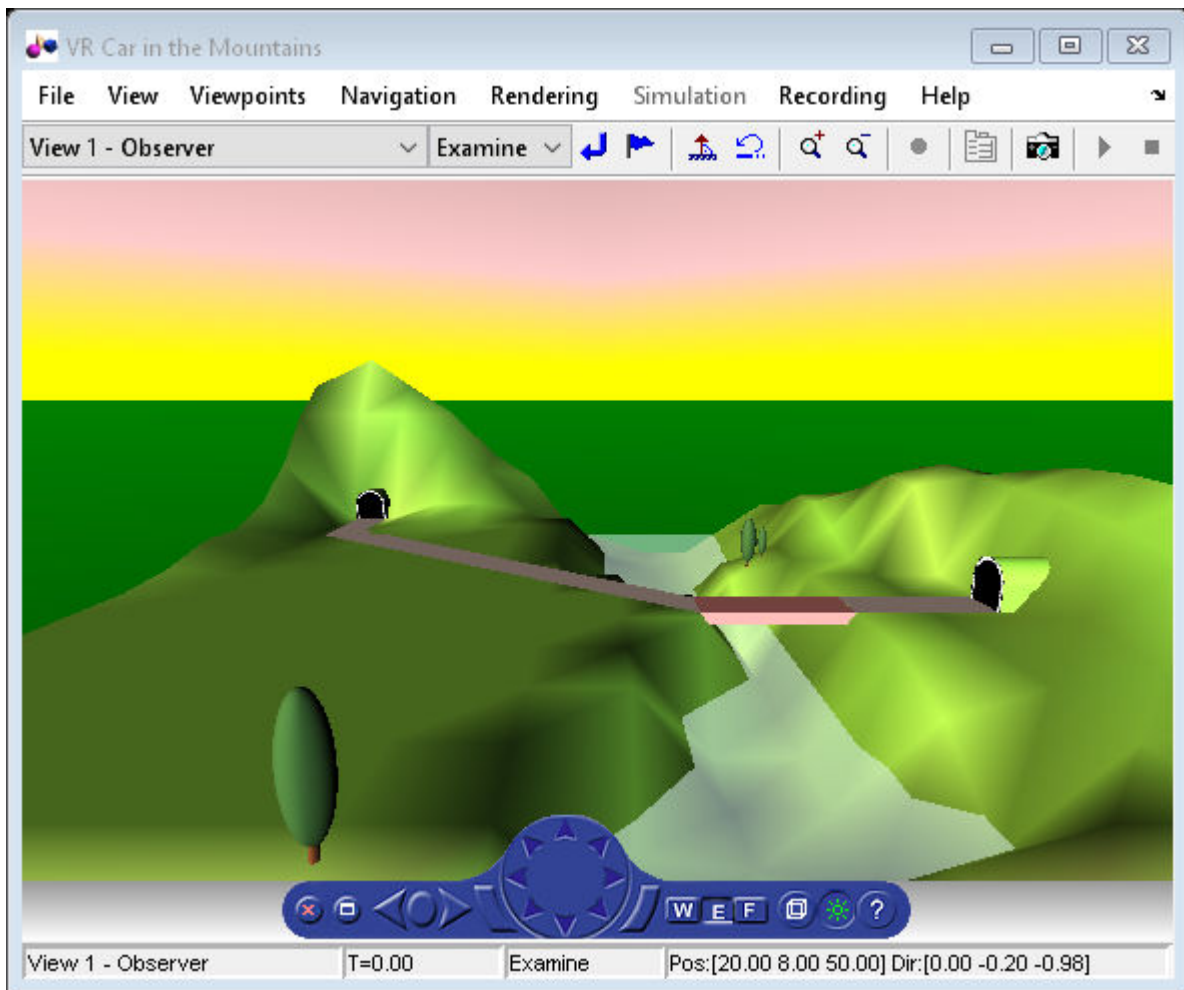
Virtual reality figure property, returned as a string or vector.

## 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'

```

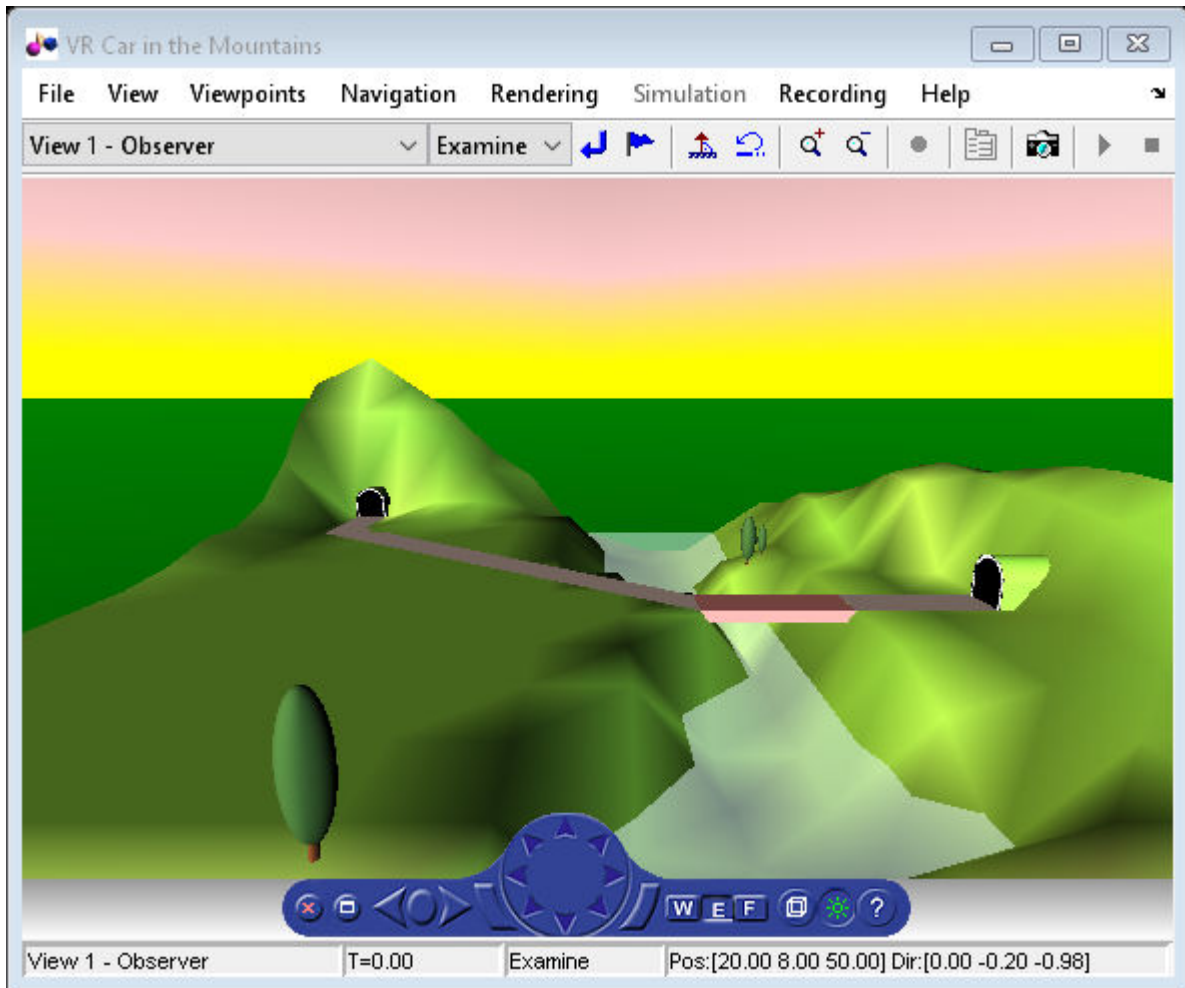
```
NavPanel = 'halfbar'  
NavSpeed = 'normal'  
NavZones = 'off'  
Position = [5 92 576 350]  
Record2D = 'off'  
Record2DCompressMethod = 'auto'  
Record2DCompressQuality = 75  
Record2DFPS = 'auto'  
Record2DFileName = '%f_anim_%n.avi'  
Rendering = 'on'  
Sound = 'on'  
StatusBar = 'on'  
Stereo3D = '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'
```

## See Also

`vrfigure` | `vr.utils.stereo3d`

## Topics

“Interact with Virtual Reality Worlds”

“Create `vrworld` Object for a Virtual World” on page 4-2

“View a Virtual World in Stereoscopic Vision” on page 7-45

**Introduced before R2006a**

## 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

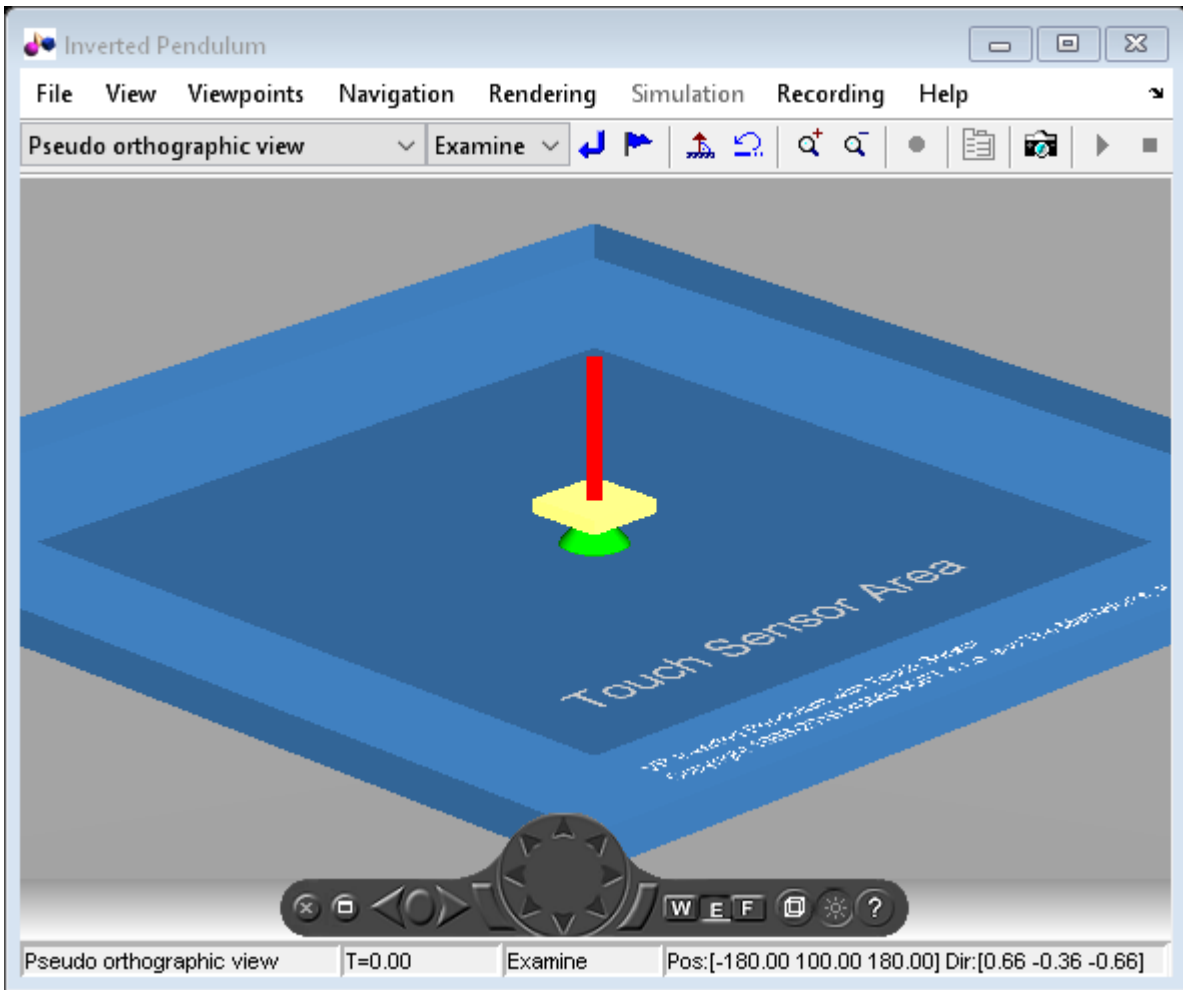
Virtual reality figure image, captured as a logical array. The array that contains a 1 where the `vrfigure` handles are valid and returns a 0 where they are not.

### 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
```

**See Also**

vrfigure | vrworld

**Topics**

- “Interact with Virtual Reality Worlds”
- “Create vrworld Object for a Virtual World” on page 4-2

**Introduced before R2006a**

# 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” on page 4-12.

**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” on page 7-18.

**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” on page 4-14.

**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.

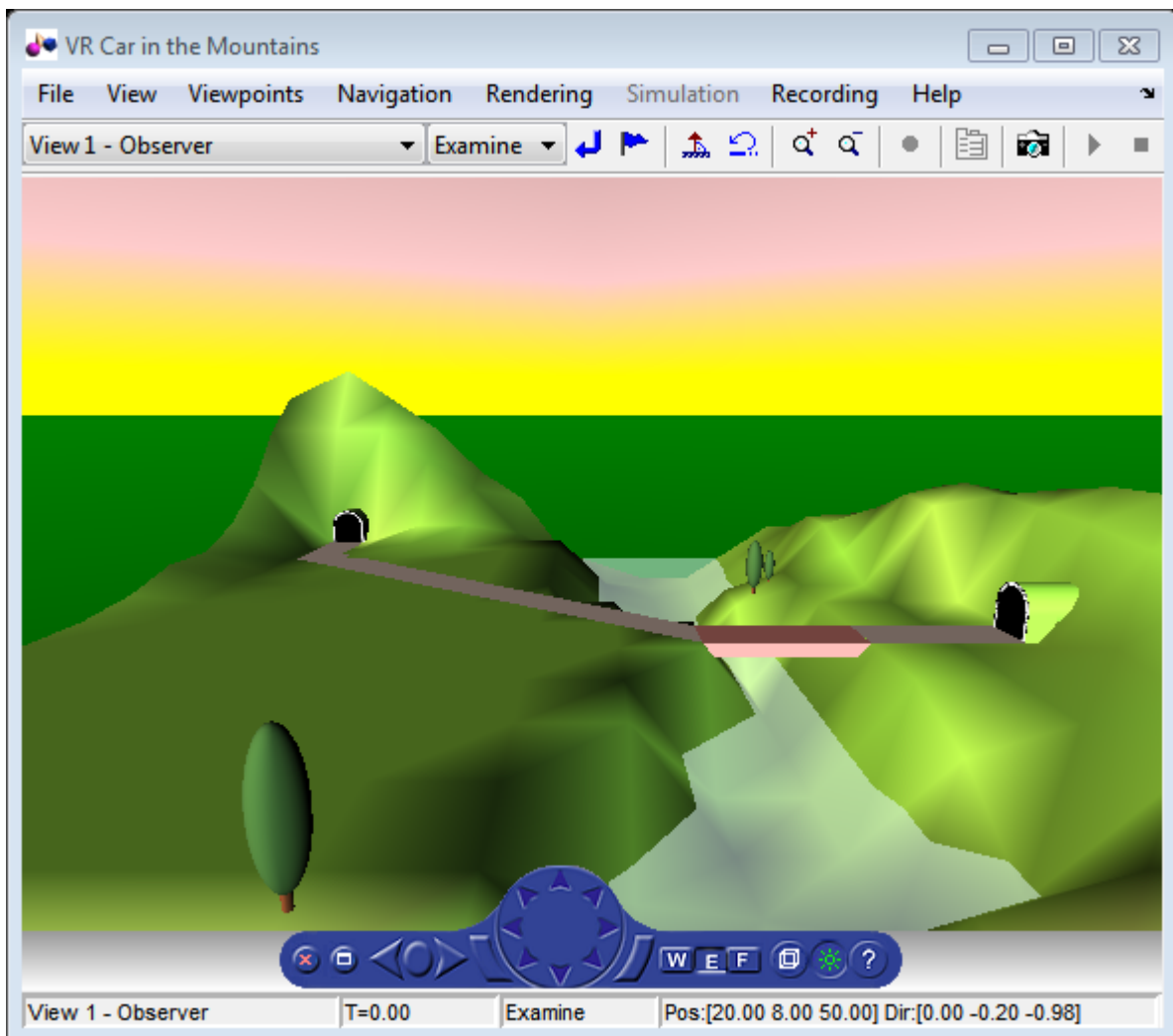
## Examples

### 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.wrl');  
open(myworld);  
virtual_fig = vrfigure(myworld);
```

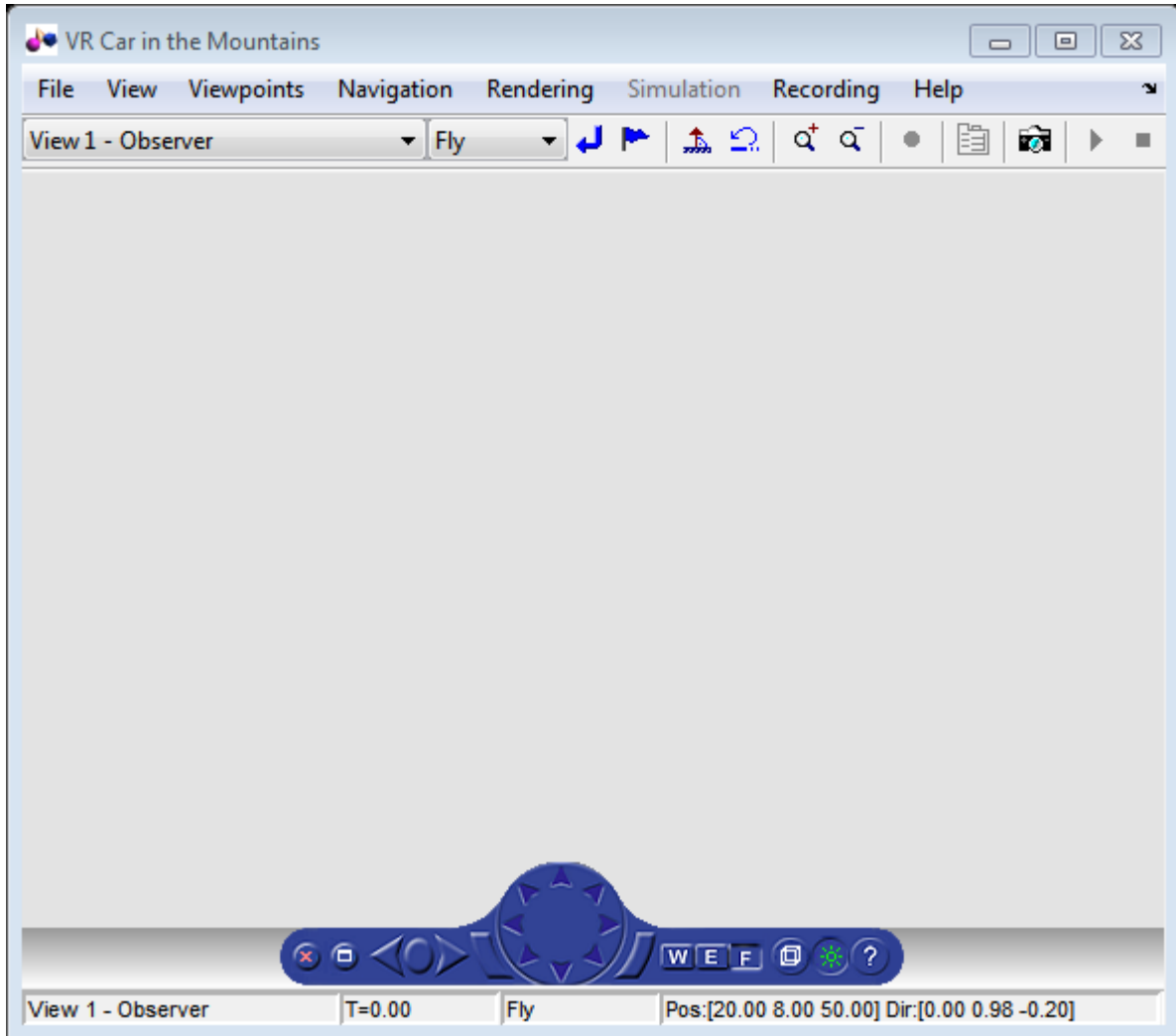


Create a `vr.utils.stereo3d` object to use to specify stereoscopic vision properties.

```
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 92 576 380]
Record2D = 'off'
Record2DCompressMethod = 'auto'
Record2DCompressQuality = 75
Record2DFPS = 'auto'
Record2DFileName = '%f_anim_%n.avi'
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
```

## See Also

`vrfigure | get | vr.utils.stereo3d`

## Topics

“Interact with Virtual Reality Worlds”

“Create vrworld Object for a Virtual World” on page 4-2

“View a Virtual World in Stereoscopic Vision” on page 7-45

**Introduced before R2006a**

## 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.

### See Also

`vrfigure` | `vr.canvas` | “Create `vrworld` Object for a Virtual World” on page 4-2 | “View a Virtual World in Stereoscopic Vision” on page 7-45

**Introduced in R2008b**

## **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.

### **See Also**

`vrfigure`

**Introduced in R2008b**

## 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
<code>AutoCreateThumbnail</code>	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
<code>DataTypeBool</code>	Specifies the handling of the virtual world <code>Bool</code> data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are <code>'logical'</code> and <code>'char'</code> . If set to <code>'logical'</code> , the virtual world <code>Bool</code> data type is returned as a logical value. If set to <code>'char'</code> , the <code>Bool</code> data type is returned <code>'on'</code> or <code>'off'</code> . Default is <code>'logical'</code> .
<code>DataTypeInt32</code>	Specifies handling of the virtual world <code>Int32</code> data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are <code>'int32'</code> and <code>'double'</code> . If set to <code>'int32'</code> , the virtual world <code>Int32</code> data type is returned as <code>int32</code> . If set to <code>'double'</code> , the <code>Int32</code> data type is returned as <code>'double'</code> . Default is <code>'double'</code> .
<code>DataTypeFloat</code>	Specifies the handling of the virtual world <code>float</code> data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are <code>'single'</code> and <code>'double'</code> . If set to <code>'single'</code> , the virtual world <code>Float</code> and <code>Color</code> data types are returned as <code>'single'</code> . If set to <code>'double'</code> , the <code>Float</code> and <code>Color</code> data types are returned as <code>'double'</code> . Default is <code>'double'</code> .
<code>DefaultCanvasNavPanel</code>	Controls the appearance of the control panel in the <code>vr.canvas</code> object. Values are: <ul style="list-style-type: none"> <li>• <code>'none'</code> Panel is not visible.</li> <li>• <code>'minimized'</code> Panel appears as a minimized icon in the right-hand corner of the viewer.</li> <li>• <code>'translucent'</code> Panel floats half transparently above the scene.</li> <li>• <code>'opaque'</code> Panel floats above the scene.</li> </ul> Default: <code>'none'</code>
<code>DefaultCanvasUnits</code>	Specifies default units for new <code>vr.canvas</code> objects. See <code>vr.canvas</code> for detailed description. Default is <code>'normalized'</code> .
<code>DefaultEditorMouseBehavior</code>	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 <code>'navigate'</code> .
<code>DefaultEditorHighlighting</code>	Specifies whether to highlight virtual world objects selected in the view pane. The default is <code>'on'</code> .



Preference	Description
DefaultFigureAntiAliasing	Determines whether antialiasing is used by default for new <code>vrfigure</code> objects. This preference also applies to new <code>vr.canvas</code> objects. Valid values are 'off' and 'on'.
DefaultFigureCaptureFileName	Specifies default file name for <code>vr.capture</code> files. See <code>get</code> for detailed description. Default is '%f_anim_%n.tif'.
DefaultFigureDeleteFcn	Specifies the default callback invoked when closing a <code>vrfigure</code> object.
DefaultFigureLighting	Specifies whether the lights are rendered by default for new <code>vrfigure</code> objects. This preference also applies to new <code>vr.canvas</code> objects. Valid values are 'off' and 'on'.
DefaultFigureMaxTextureSize	Specifies the default maximum size of a texture used in rendering new <code>vrfigure</code> objects. This preference also applies to new <code>vr.canvas</code> objects. Valid values are 'auto' and $32 \leq x \leq \text{video card limit}$ , 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 <code>vrfigure</code> objects. This preference also applies to new <code>vr.canvas</code> 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 <code>vrfigure</code> objects. Valid values are '', 'auto', 'lossless', and 'codec_code'.
DefaultFigureRecord2DCompressQuality	Specifies the default quality of 2-D animation file compression for new <code>vrfigure</code> objects. Valid values are 0-100.
DefaultFigureRecord2DFileName	Specifies the default 2-D offline animation file name for new <code>vrfigure</code> 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 <code>vrfigure</code> or <code>vr.canvas</code> 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 <code>vrfigure</code> objects. Valid values are 'off' and 'on'.
DefaultFigureTextures	Specifies whether textures should be rendered by default for new <code>vrfigure</code> objects. This preference also applies to new <code>vr.canvas</code> objects. See <code>get</code> for detailed description. Default is 'on'.
DefaultFigureToolBar	Specifies whether the toolbar appears by default on the Simulink 3D Animation Viewer for new <code>vrfigure</code> objects. Valid values are 'off' and 'on'.
DefaultFigureTransparency	Specifies whether or not transparency information is taken into account when rendering for new <code>vrfigure</code> objects. This preference also applies to new <code>vr.canvas</code> objects. Valid values are 'off' and 'on'.
DefaultFigureWireframe	Specifies whether objects are drawn as solids or wireframes by default for new <code>vrfigure</code> objects. This preference also applies to new <code>vr.canvas</code> objects. Valid values are 'off' and 'on'.
DefaultViewer	Specifies which viewer is used to view a virtual scene. <ul style="list-style-type: none"> <li>'internal' Default Simulink 3D Animation Viewer.</li> <li>'web' Web browser becomes viewer. This is the current Web browser virtual world plug-in.</li> </ul>
DefaultWorldRecord3DFileName	Specifies the default 3-D animation file name for new <code>vrworld</code> objects.
DefaultWorldRecordMode	Specifies the default animation recording mode for new <code>vrworld</code> objects. Valid values are 'manual' and 'scheduled'.
DefaultWorldRecordInterval	Specifies the default start and stop times for scheduled animation recording for new <code>vrworld</code> objects. Valid value is a vector of two doubles.
DefaultWorldRemoteView	Specifies whether the virtual world is enabled by default for remote viewing for new <code>vrworld</code> 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	<p>Specifies which virtual world editor to use. Path to the virtual world editor. If this path is empty, the MATLAB editor is used.</p> <p>The path setting is active only if you select the Custom option.</p> <p>To open a virtual world file in a third-party editor, do not use the <code>vredit</code> command. For example, to open a virtual world in the Ligos V-Realm Builder editor:</p> <ol style="list-style-type: none"> <li>1 Set the default editor to V-Realm Builder. In MATLAB, enter: <pre>vrsetpref('Editor','*VREALM');</pre></li> <li>2 To open a file in the V-Realm editor, in MATLAB navigate to a virtual world file, right-click, and select <b>Edit</b>.</li> </ol> <p><b>Note</b> The <code>vredit</code> command opens the 3D World Editor, regardless of the default editor preference setting.</p>
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
<code>'internal'</code>	Default Simulink 3D Animation Viewer.
<code>'web'</code>	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:

<code>%matlabroot</code>	Refers to the MATLAB root folder
<code>%file</code>	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.

**See Also**

“Set Simulink 3D Animation Preferences” on page 2-5

**Introduced before R2006a**

## 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

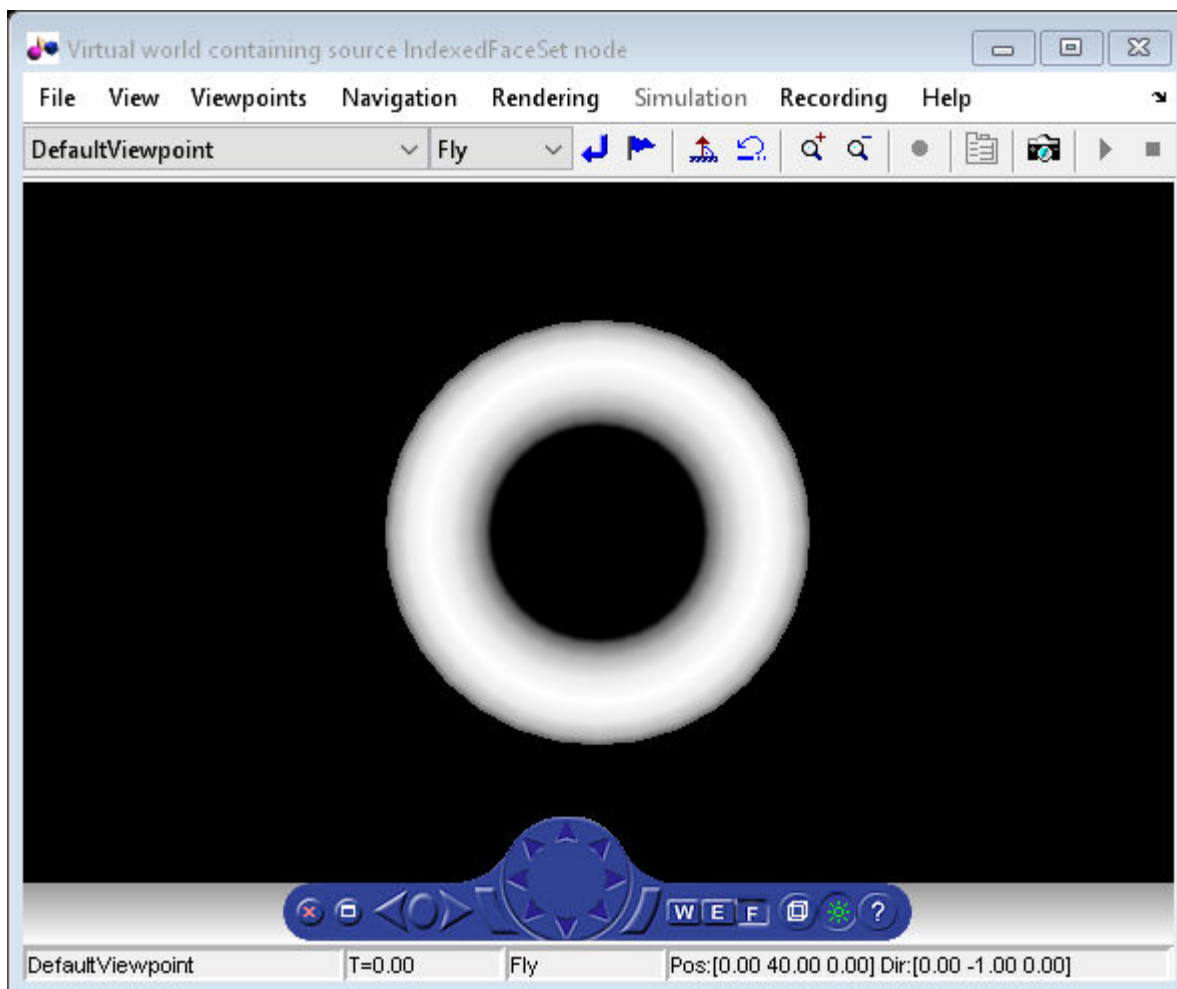
This command converts three IndexedFaceSet nodes to MATLAB® patch objects.

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.

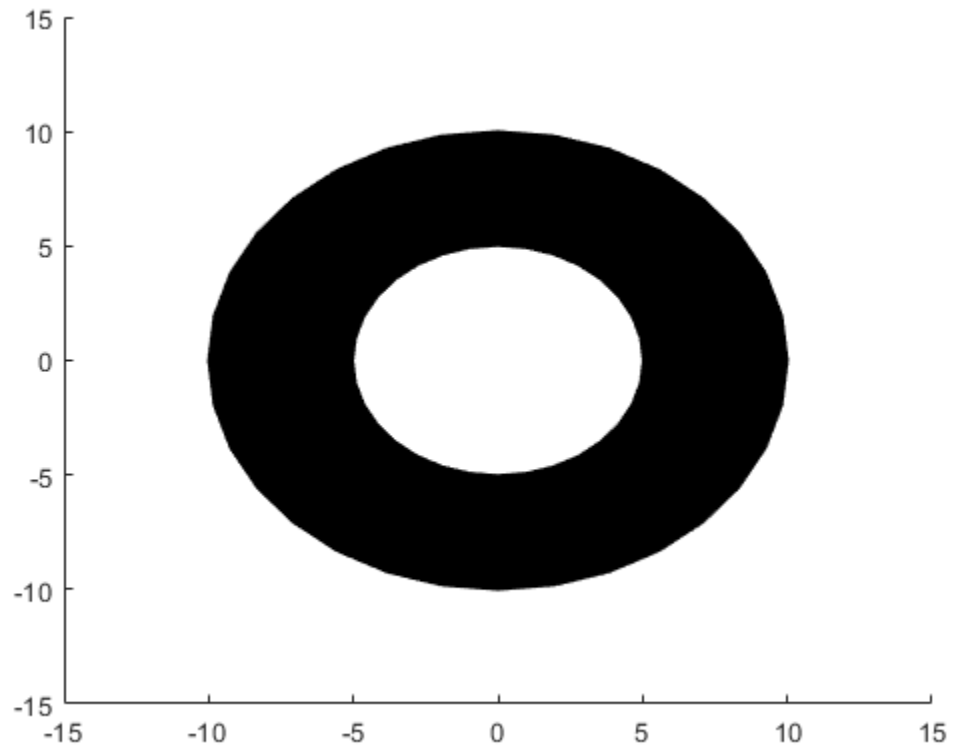
```
vrfig1 = vrfigure(w1, ...  
    'Name', 'Virtual world containing source IndexedFaceSet node', ...  
    'CameraBound', 'off', ...  
    'CameraPosition', [0 40 0], ...  
    'CameraDirection', [0 -1 0], ...  
    'CameraUpVector', [0 0 -1]);
```



```
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.

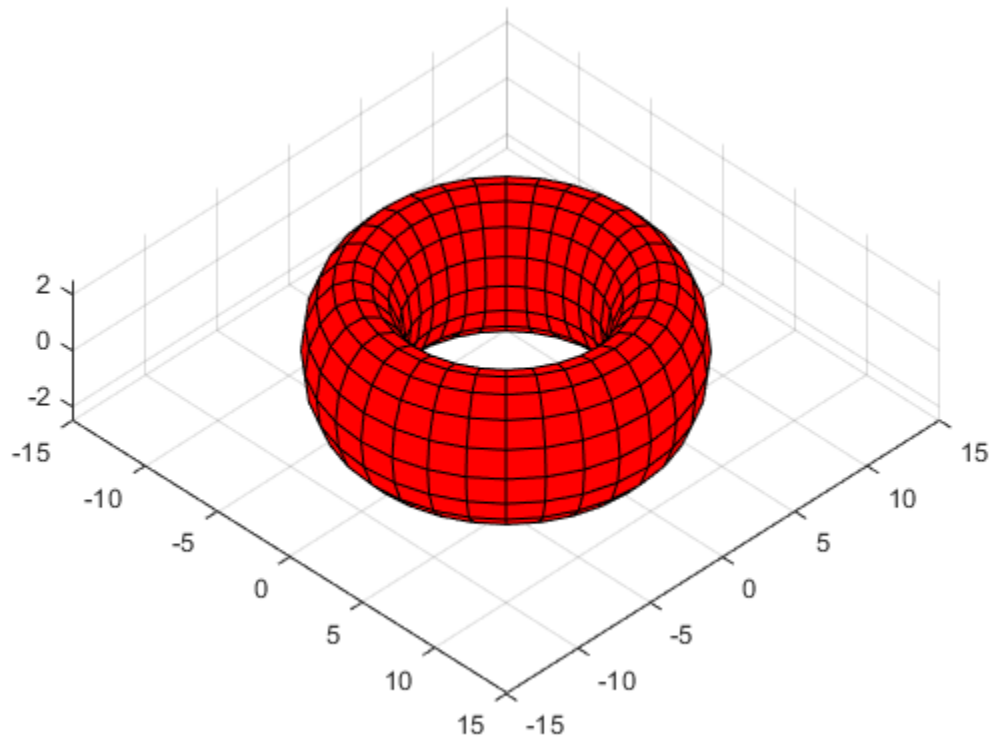
```
tp.FaceColor = 'red';
```

```
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.

## See Also

patch | vrpach2ifs

## Topics

“Introduction to Patch Objects”

**Introduced in R2015a**

## 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)

---

**Tip** To import Physical Modeling XML files, use the `stl2vrm` function instead of `vrimport`.

---

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.

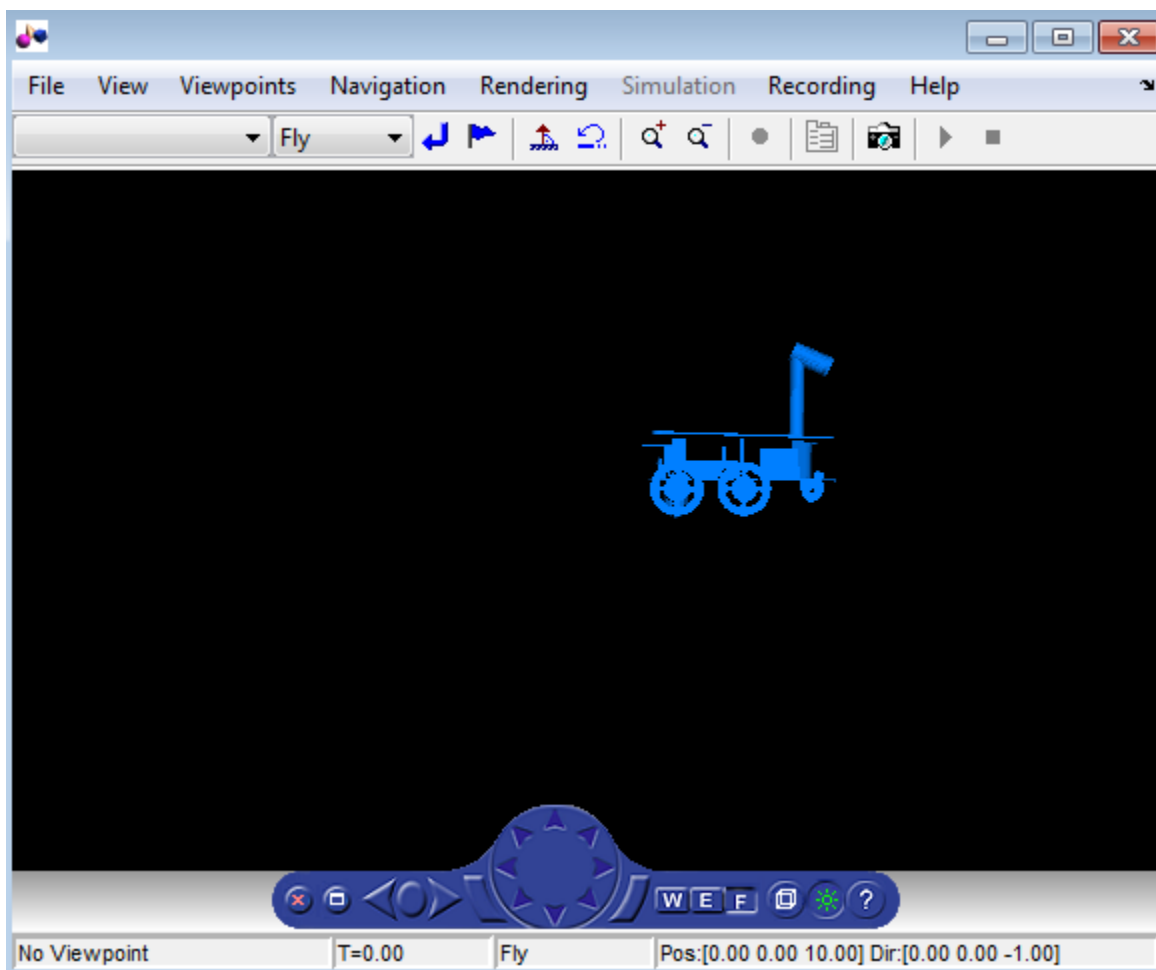
```
get(w, 'Nodes')
```

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 = vrnnode(w.Rover_Shape, 'appearance', 'Rover_App', 'Appearance');
mat = vrnnode(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.wrl')
```

### **Import a DAE file**

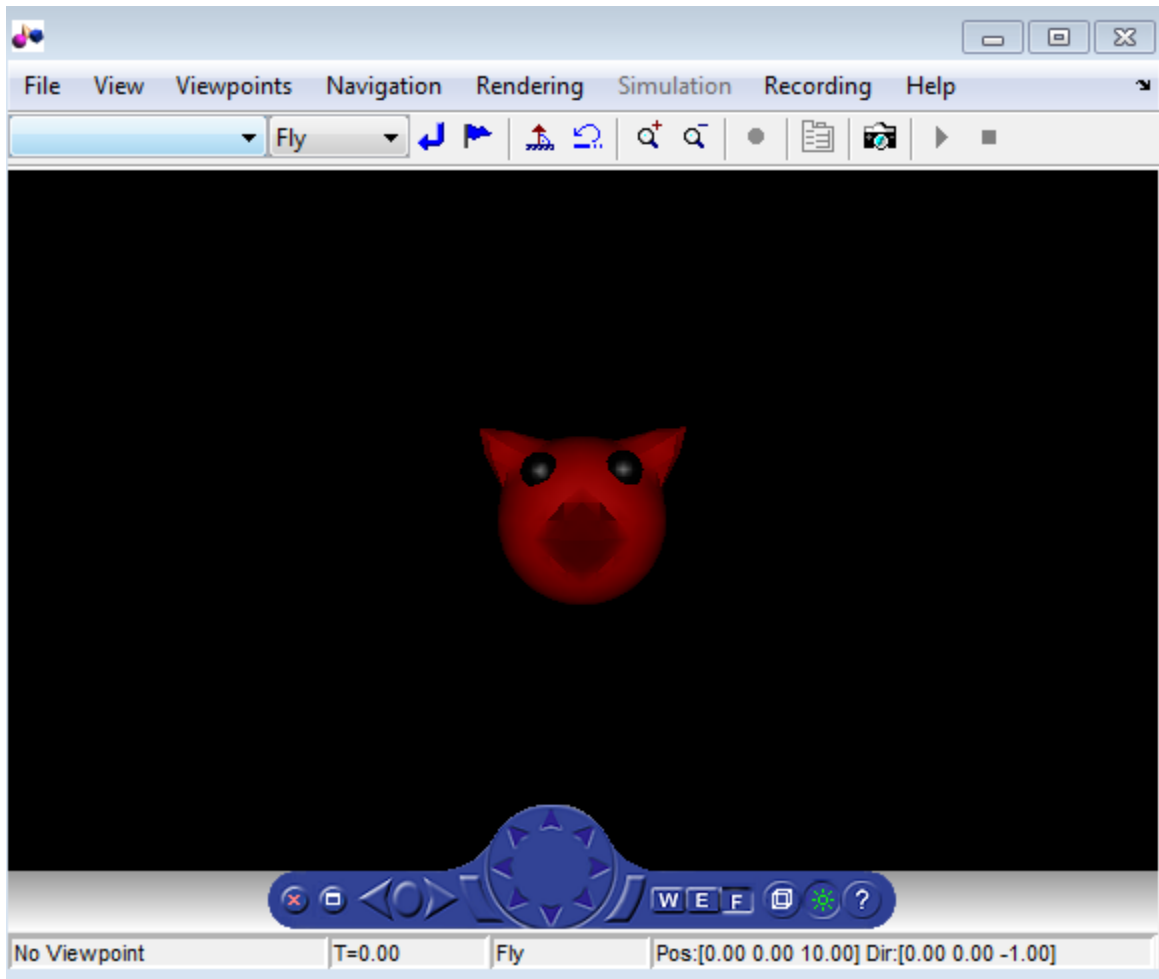
This example imports a .dae format file into a virtual world.

Import the fox.dae file to a node in a virtual world.

```
[n,w] = vrimport(which('fox.dae'))  
n =  
    vrnnode 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.wrl')
```

## 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 | vrnnode 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.

## See Also

`stl2vrml` | `vrcadcleanup` | `vrphysmod`

## Topics

“Import STL and Physical Modeling XML Files” on page 5-38

“Import Visual Representations of Robot Models” on page 5-54

“Link to Simulink and Simscape Multibody Models” on page 5-60

## Introduced in R2016b

# 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

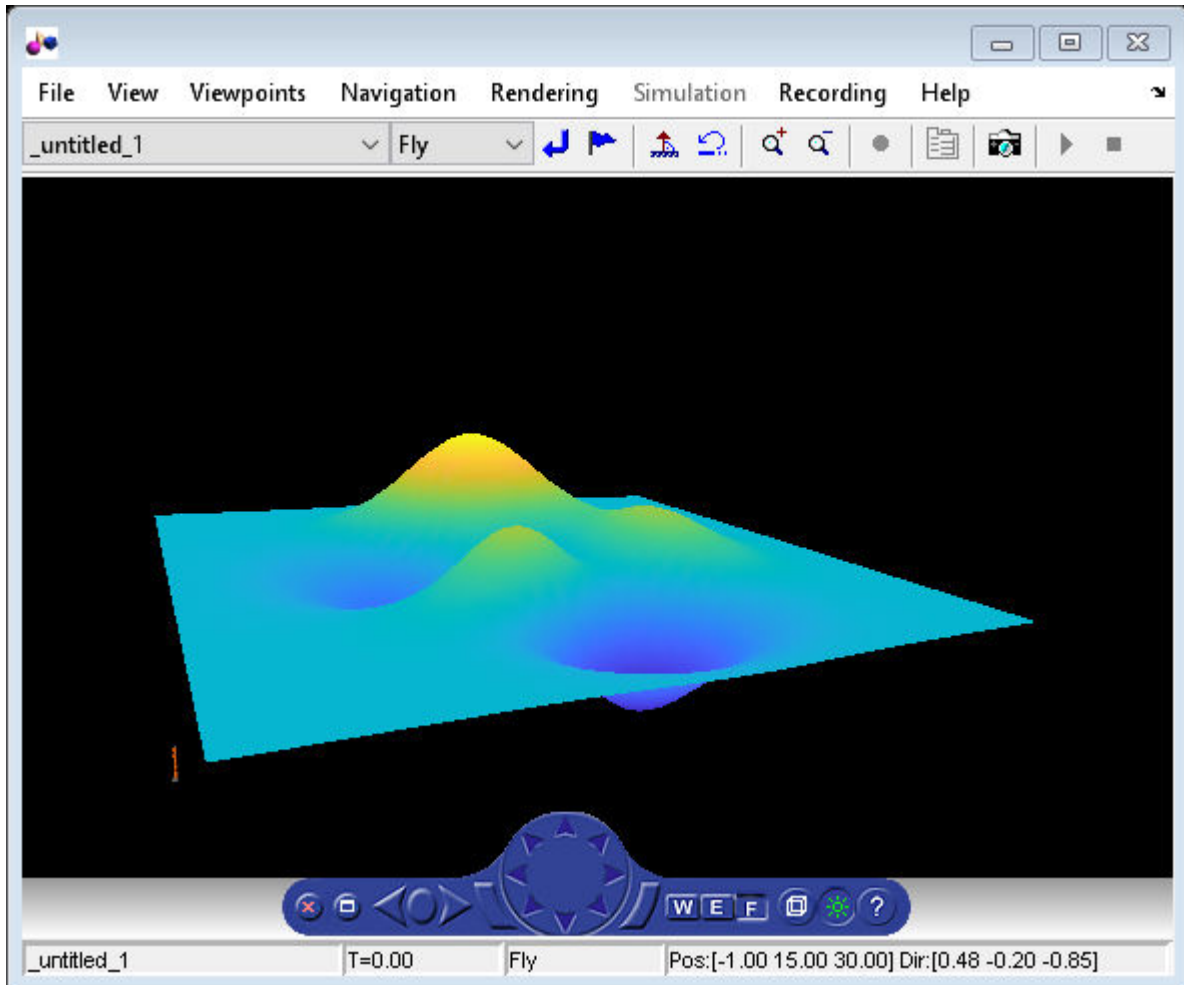
```
RBT = importrobot('iiwa7.urdf');
RBT.DataFormat = 'Row';
```

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” on page 6-9



```
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.

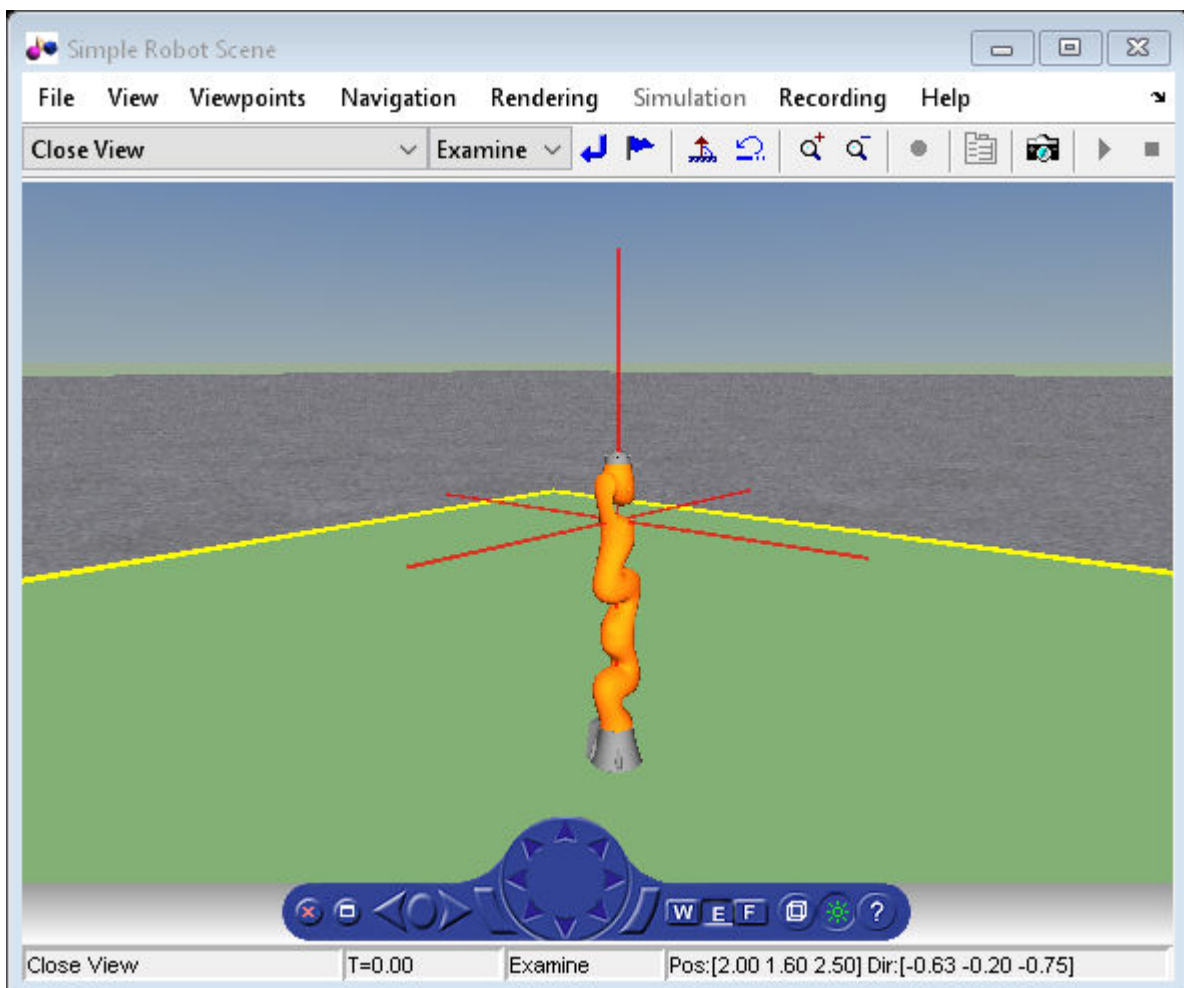
```
rbt = importrobot('iiwa7.urdf');
rbt.DataFormat = 'row';
```

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

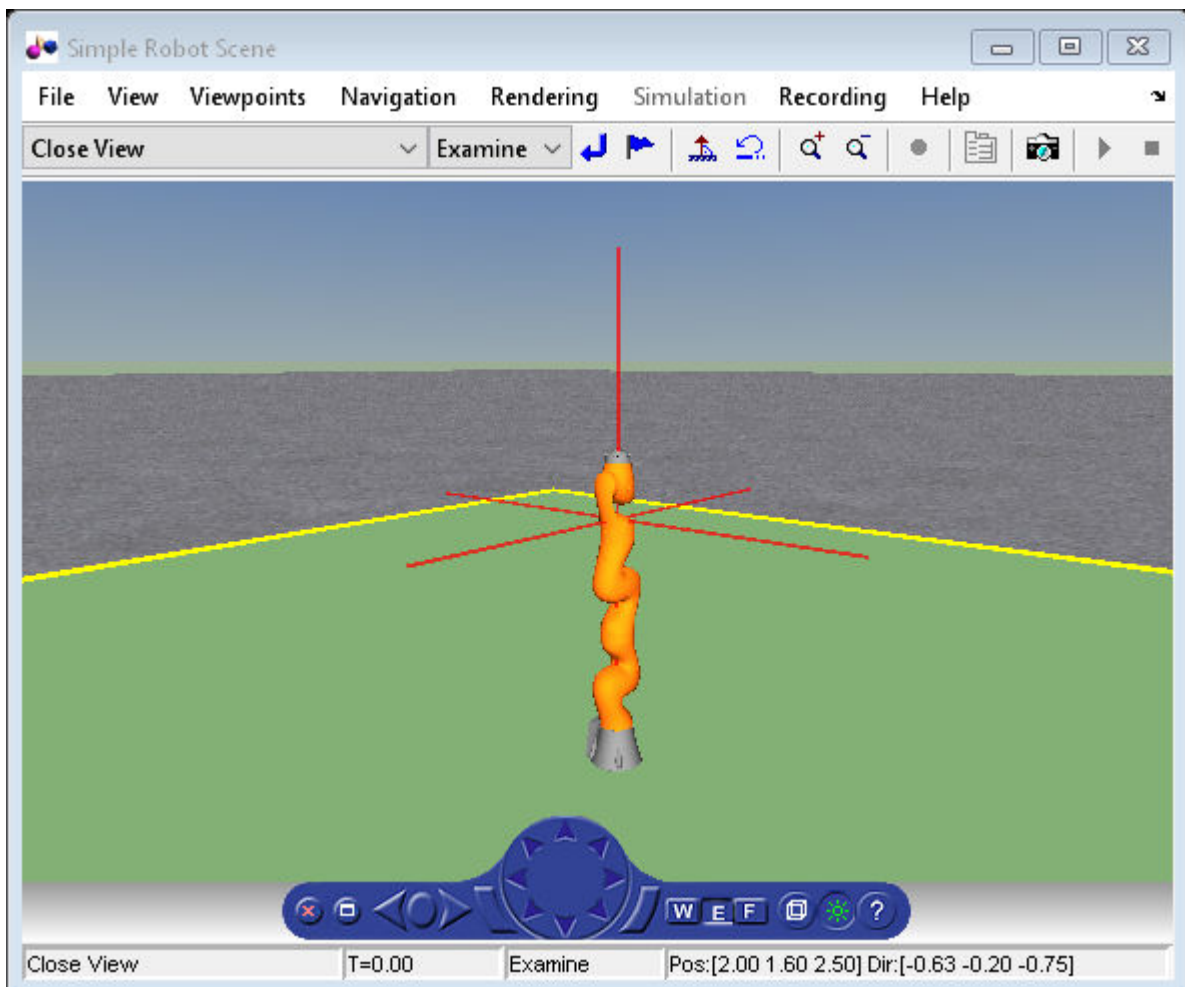
Import the KUKA LFR iiwa robot from its URDF definition and insert it to the virtual world created from `robot_scene.wrl`.

```
RBT = importrobot('iiwa7.urdf');
RBT.DataFormat = 'row';
robotWorld = vrworld('robot_scene');
open(robotWorld);
```

### Get Transforms of Current Pose of the Robot

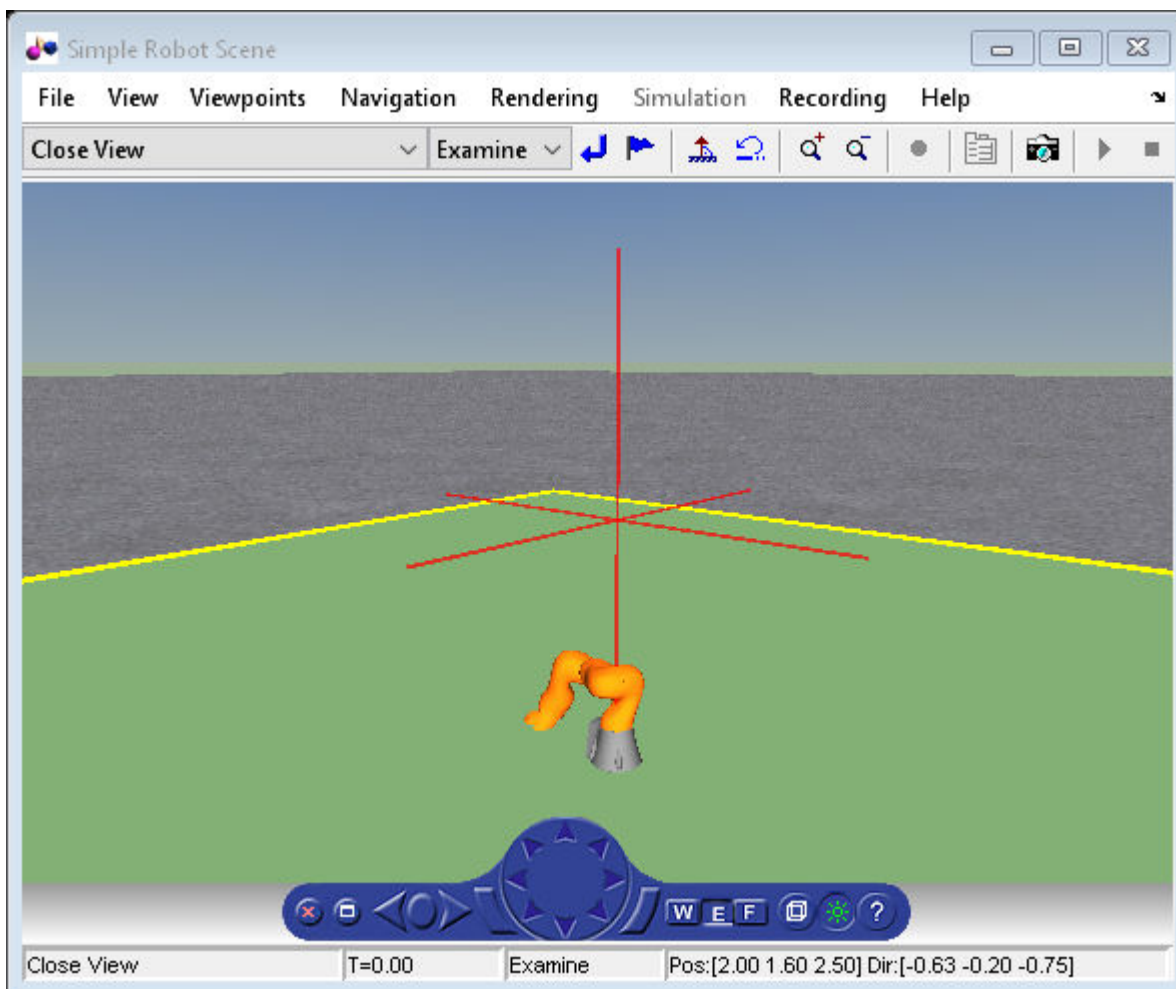
The `tforms` output argument contains a list of transforms that describe the robot pose in its initial or 'home' configuration.

```
[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.

## See Also

`vrworld` | `vrimport` | `importrobot` (Robotics System Toolbox) | `rigidBodyTree` (Robotics System Toolbox)

**Introduced in R2018b**

# 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” on page 2-14.

---

**Note** The `vrinstall` command does no perform any action on a Linux platform.

---

The actions you can perform

Action Value	Description
<code>-selftest</code>	Checks the integrity of the current installation. If this function reports an error, you should reinstall the Simulink 3D Animation software. The function <code>vrinstall</code> automatically does a self-test with any other actions.
<code>-interactive</code>	Checks for the installed components, and then displays a list of uninstalled components you can choose to install.
<code>-check</code>	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.
<code>-install</code>	Installs optional components. This action requires you to specify the component name.
<code>-uninstall</code>	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`

**Introduced before R2006a**

# vrjoystick

Create joystick object

## Syntax

```
joy = vrjoystick(id)
```

```
joy = vrjoystick(id, 'forcefeedback')
```

## 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.

## Methods

Method	Description
<code>axis</code>	<code>a = axis(joy, n)</code> reads the status of joystick with axis number <code>n</code> . Axis status is returned in the range of -1 to 1. The <code>n</code> parameter may be a vector to return multiple buttons.
<code>button</code>	<code>b = button(joy, n)</code> reads the status of joystick button number <code>n</code> . Button status is returned as logical 0 if not pressed and logical 1 if pressed. The <code>n</code> parameter may be a vector to return multiple buttons.
<code>caps</code>	<code>c = caps(joy)</code> returns joystick capabilities, such as the number of axes, buttons, POVs, and force-feedback axes. The return value is a structure with fields named <code>Axes</code> , <code>Buttons</code> , <code>POVs</code> , and <code>Forces</code> .
<code>close</code>	<code>close(joy)</code> closes and invalidates the joystick object. The object cannot be used once it is closed.
<code>force</code>	<code>force(joy, n, f)</code> applies force feedback to joystick axis <code>n</code> . The <code>n</code> parameter can be a vector to affect multiple axes. <code>f</code> values should be in range of -1 to 1, and the number of elements in <code>f</code> should either match the number of elements of <code>n</code> , or <code>f</code> can be a scalar to be applied to all the axes specified by <code>n</code> .
<code>pov</code>	<code>p = pov(joy, n)</code> reads the status of joystick POV (point of view) of control number <code>n</code> . <code>pov</code> is usually returned in degrees, with -1 meaning "not selected." <code>n</code> can be a vector to return multiple POVs.

<b>Method</b>	<b>Description</b>
read	[axes, buttons, povs] = read(joy) reads the status of axes, buttons, and POVs of the specified joystick. [axes, buttons, povs] = read(joy, forces) applies feedback forces, in addition, to a force-feedback joystick.

where joy is the handle to the joystick object.

### **See Also**

“Set Simulink 3D Animation Preferences” on page 2-5

**Introduced in R2007b**



## 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.

**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

<code>vrworld_object</code>	Name of a <code>vrworld</code> object representing a virtual world.
<code>node_name</code>	Name of the node.
<code>node_type</code>	Type of the node.
<code>parent_node</code>	Name of the parent node that is a <code>vrnode</code> object.
<code>parent_field</code>	Name of the field of the parent node.
<code>'USE'</code>	Enables a USE reference to another node.
<code>othernode</code>	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
<code>delete</code>	Remove <code>vrnode</code> object
<code>fields</code>	Virtual world field summary of node object
<code>get</code>	Property value of <code>vrnode</code> object
<code>getfield</code>	Field value of <code>vrnode</code> object
<code>isvalid</code>	1 if <code>vrnode</code> object is valid, 0 if not
<code>set</code>	Change property of virtual world node
<code>setfield</code>	Change field value of <code>vrnode</code> object
<code>sync</code>	Enable or disable synchronization of virtual world fields with client

## See Also

`vrnode/delete` | `vrnode/get` | `vrnode/getfield` | `vrnode/set` | `vrnode/setfield` | `vrworld`

**Introduced before R2006a**

## **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.

### **See Also**

vrworld/delete

**Introduced before R2006a**

## 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`.

### See Also

`vrnode/get` | `vrnode/set`

**Introduced before R2006a**

## vrnode/get

Property value of vrnode object

### Syntax

```
get(vrnode_object)
```

```
x = get(vrnode_object)
```

```
x = get(vrnode_object, 'property_name')
```

### Arguments

<code>vrnode_object</code>	Name of a vrnode object representing the node to be queried.
<code>property_name</code>	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.

### See Also

[vrnode](#) | [vrnode/getfield](#) | [vrnode/set](#) | [vrnode/setfield](#)

**Introduced before R2006a**

## vrnode/getfield

Field value of vrnode object

### Syntax

```
getfield(vrnode_object)
```

```
x = getfield(vrnode_object)
```

```
x = getfield(vrnode_object, 'fieldname')
```

### Arguments

<code>vrnode_object</code>	Name of a vrnode object representing the node to be queried.
<code>fieldname</code>	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
```

---

### See Also

[vrnode](#) | [vrnode/get](#) | [vrnode/set](#) | [vrnode/setfield](#)

**Introduced before R2006a**

## **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.

### **See Also**

`isvalid` | `vrworld/isvalid`

**Introduced before R2006a**



## vrnode/set

Change property of virtual world node

### Syntax

```
x = set(vrnode_object, 'property_name', 'property_value')
```

### Arguments

<code>vrnode_object</code>	Name of a <code>vrnode</code> object representing a node in the virtual world.
<code>property_name</code>	Name of a property.
<code>property_value</code>	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 <code>vrworld</code> object. This is a <code>vrworld</code> object that represents the node's parent world. Read only.

Currently, nodes have no settable properties.

### See Also

`vrnode` | `vrnode/get` | `vrnode/getfield` | `vrnode/setfield`

**Introduced before R2006a**

## vrnode/setfield

Change field value of vrnode object

### Syntax

```
x = setfield(vrnode_object, 'fieldname', 'fieldvalue')
```

### Arguments

<code>vrnode_object</code>	Name of a vrnode object representing the node to be changed.
<code>fieldname</code>	Name of the vrnode object field whose values you want to set.
<code>fieldvalue</code>	Value of <code>fieldname</code> .

### 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;
```

---

### See Also

`vrnode` | `vrnode/get` | `vrnode/getfield` | `vrnode/set`

**Introduced before R2006a**

## vrnode/sync

Enable or disable synchronization of fields with client

### Syntax

```
sync(vrnode_object, 'field_name', 'action')
```

### Arguments

<code>vrnode_object</code>	Name of a <code>vrnode</code> object representing the node.
<code>field_name</code>	Name of the field to be synchronized.
<code>action</code>	The action parameter determines what should be done: <ul style="list-style-type: none"><li>• 'on' enables synchronization of this field.</li><li>• 'off' disables synchronization of this field.</li></ul>

### 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.

---

### See Also

`vrnode` | `vrnode/get`

Introduced before R2006a

## **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`).

### **See Also**

`vrdir2ori` on page 10-30 | `vrrotmat2vec` on page 10-108 | `vrrotvec` on page 10-107 | `vrrotvec2mat` on page 10-109

**Introduced in R2007b**

## 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.

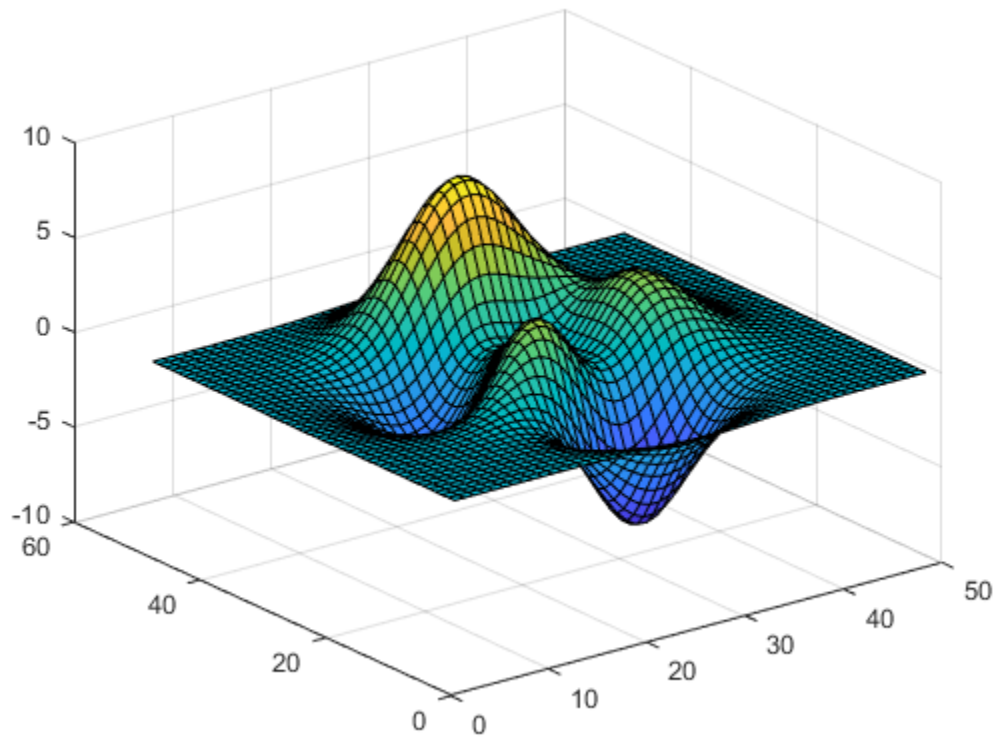
### Examples

#### Convert MATLAB Patches to IndexedFaceSet Nodes

This command converts three MATLAB® patches to `IndexedFaceSet` nodes.

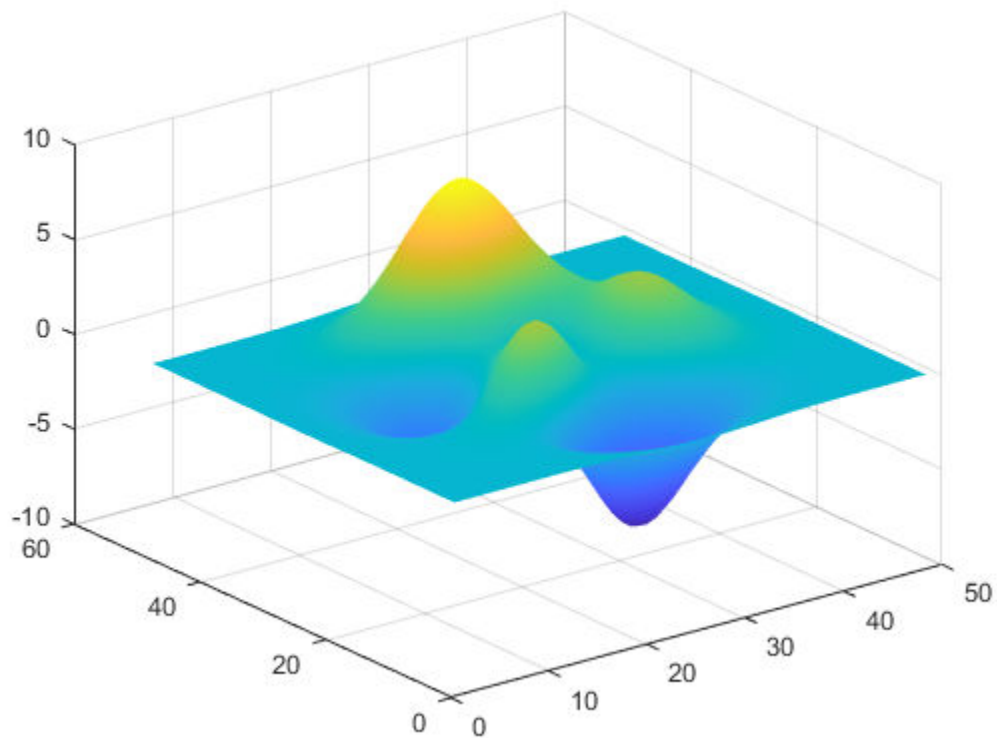
Create surface using MATLAB `peaks` function.

```
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.

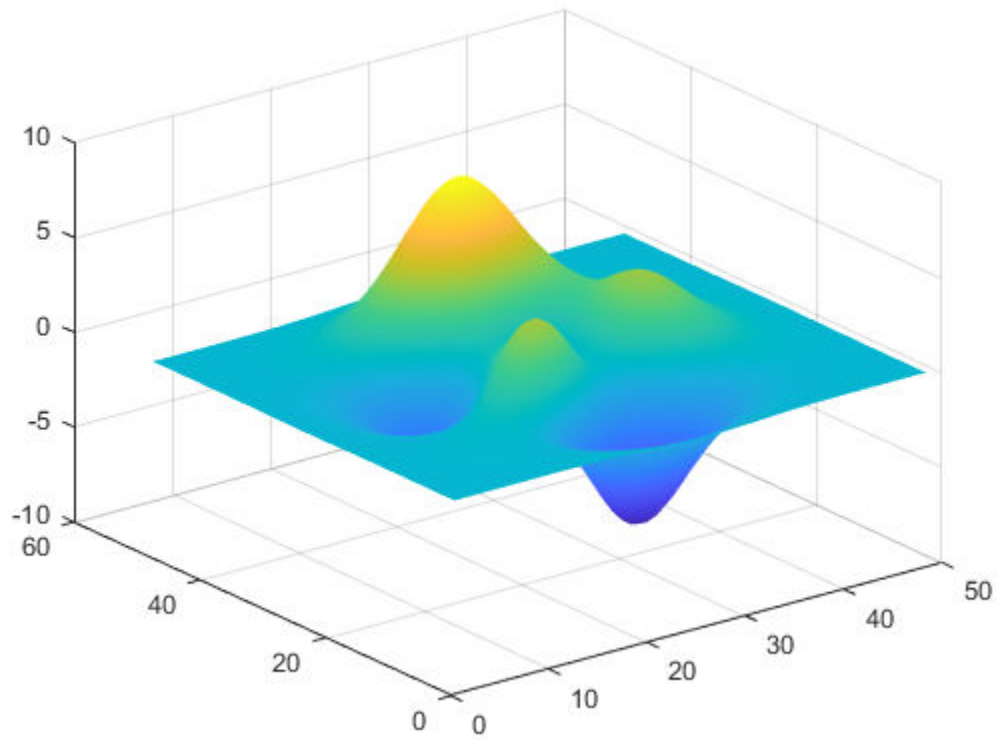
```
w2 = vrworld('');
open(w2);
```

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)

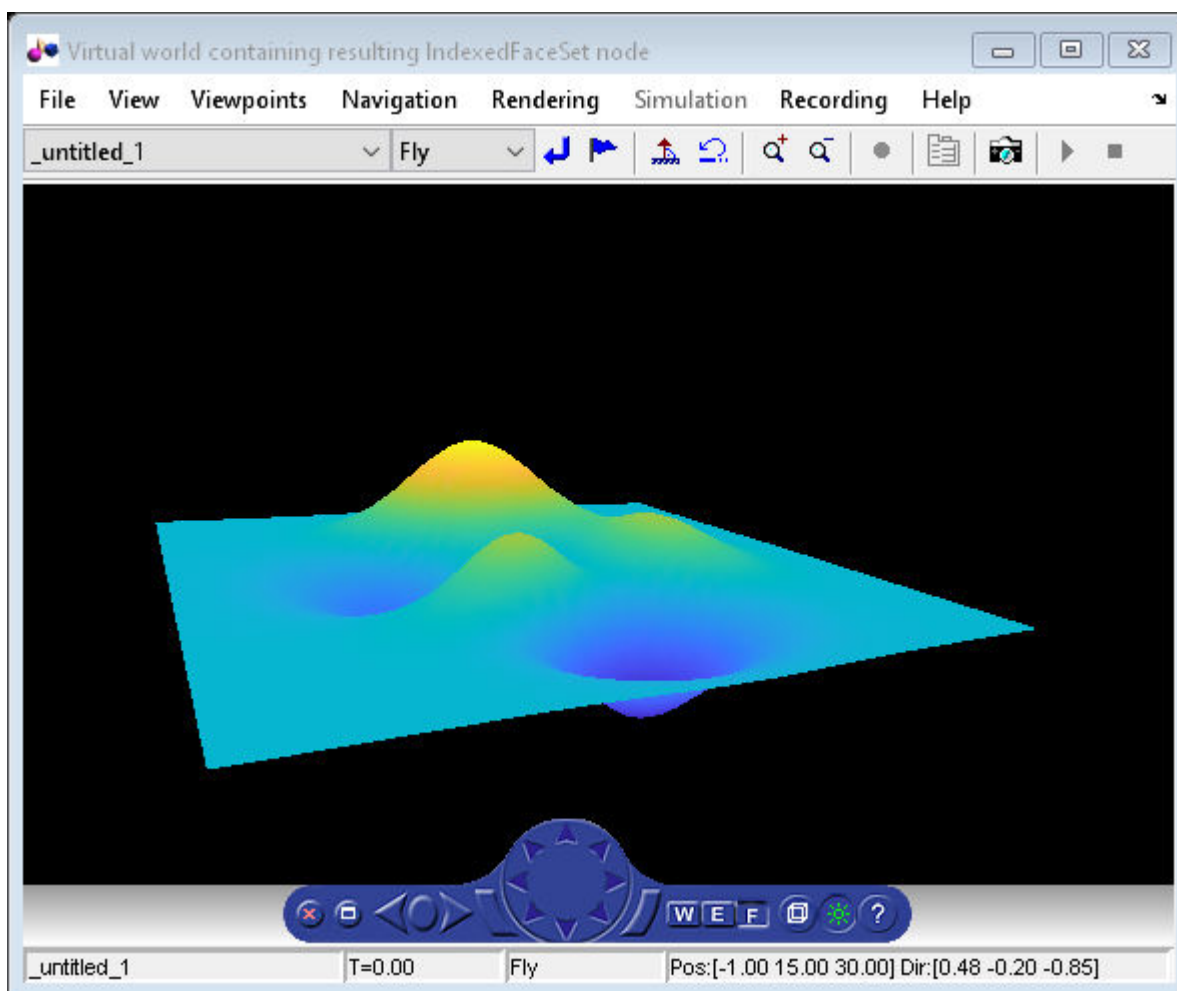
```
vrpatch2ifs(peaksPatch,w2);
```



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.

**See Also**

patch | vrifs2patch

**Topics**

“Introduction to Patch Objects”

**Introduced in R2015a**

# 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.

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.

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.

---

**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.

---

## Examples

To update the model `four_link` using the file `four_link.wrl`:

```
vrphysmod('four_link.wrl', 'four_link');
```

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');
```

To update the current system using the file `four_link.wrl`:

```
vrphysmod('four_link.wrl', gcs);
```

**See Also**

`smimport` | `stl2vrm` | `vrcadcleanup`

**Introduced in R2009a**

## vrplay

Play VRML animation file

### Syntax

```
vrplay
vrplay(filename)
x=vrplay(filename)
```

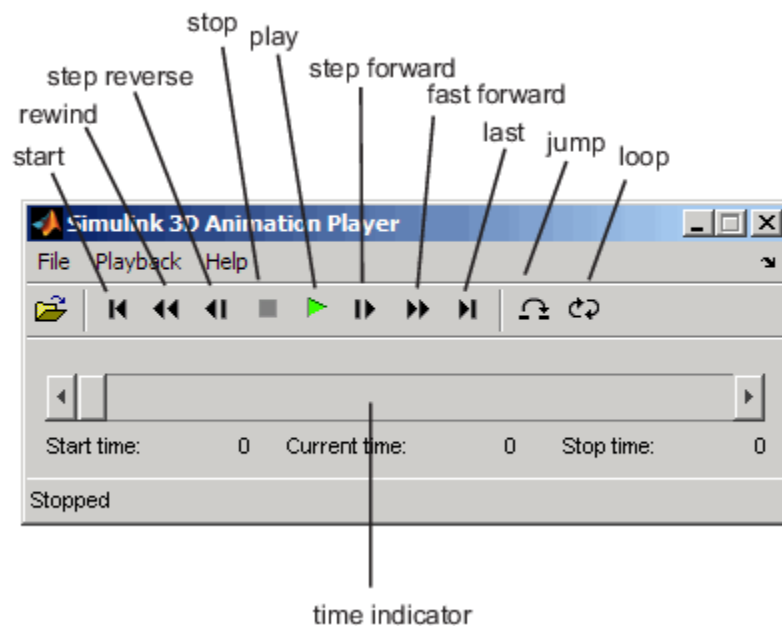
### Description

`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')`.

## See Also

`vrview`

## Topics

“Record Offline Animations” on page 7-29

**Introduced in R2006a**

## 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`.

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.

### See Also

`vrrotmat2vec` on page 10-108 | `vrrotvec2mat` on page 10-109

**Introduced in R2007b**

## **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.

### **See Also**

`vrrotvec` on page 10-107 | `vrrotvec2mat` on page 10-109

**Introduced in R2007b**



## 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.

### See Also

`vrrotmat2vec` on page 10-108 | `vrrotvec` on page 10-107

**Introduced in R2007b**

## 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
<code>AutoCreateThumbnail</code>	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.
<code>DataTypeBool</code>	Specifies the handling of the virtual world <code>Bool</code> data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are 'logical' and 'char'. If set to 'logical', the virtual world <code>Bool</code> data type is returned as a logical value. If set to 'char', the <code>Bool</code> data type is returned 'on' or 'off'. Default is 'logical'.
<code>DataTypeInt32</code>	Specifies handling of the virtual world <code>Int32</code> data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are 'int32' and 'double'. If set to 'int32', the virtual world <code>Int32</code> data type is returned as <code>int32</code> . If set to 'double', the <code>Int32</code> data type is returned as 'double'. Default is 'double'.

Preference	Description
DataTypeFloat	Specifies the handling of the virtual world float data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . 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 <code>vr.canvas</code> object. Values are: <ul style="list-style-type: none"> <li>• 'none' Panel is not visible.</li> <li>• 'minimized' Panel appears as a minimized icon in the right-hand corner of the viewer.</li> <li>• 'translucent' Panel floats half transparently above the scene.</li> <li>• 'opaque' Panel floats above the scene.</li> </ul> Default: 'none'
DefaultCanvasUnits	Specifies default units for new <code>vr.canvasobjects</code> . See <code>vr.canvas</code> 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'.
DefaultFigureAntiAliasing	Determines whether antialiasing is used by default for new <code>vrfigure</code> objects. This preference also applies to new <code>vr.canvasobjects</code> . Valid values are 'off' and 'on'.
DefaultFigureCaptureFileName	Specifies default file name for capturing viewer figures. See <code>get</code> for detailed description. Default is '%f_anim_%n.tif'.
DefaultFigureDeleteFcn	Specifies the default callback invoked when closing a <code>vrfigure</code> object.

Preference	Description
DefaultFigureLighting	Specifies whether the lights are rendered by default for new <code>vrfigure</code> objects. This preference also applies to new <code>vr.canvas</code> objects. Valid values are 'off' and 'on'.
DefaultFigureMaxTextureSize	Specifies the default maximum size of a texture used in rendering new <code>vrfigure</code> objects. This preference also applies to new <code>vr.canvas</code> objects. Valid values are 'auto' and $32 \leq x \leq \text{video card limit}$ , 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 <code>vrfigure</code> objects. This preference also applies to new <code>vr.canvasobjects</code> . 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 <code>vrfigure</code> objects. Valid values are '', 'auto', 'lossless', and 'codec_code'.
DefaultFigureRecord2DCompressQuality	Specifies the default quality of 2-D animation file compression for new <code>vrfigure</code> objects. Valid values are 0-100.
DefaultFigureRecord2DFileName	Specifies the default 2-D offline animation file name for new <code>vrfigure</code> 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 <code>vrfigure</code> or <code>vr.canvas</code> 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 <code>vrfigure</code> objects. Valid values are 'off' and 'on'.
DefaultFigureTextures	Specifies whether textures should be rendered by default for new <code>vrfigure</code> objects. This preference also applies to new <code>vr.canvas</code> objects. See <code>get</code> for detailed description. Default is 'on'.

Preference	Description
DefaultFigureToolBar	Specifies whether the toolbar appears by default on the Simulink 3D Animation Viewer for new <code>vrfigure</code> objects. Valid values are 'off' and 'on'.
DefaultFigure Transparency	Specifies whether or not transparency information is taken into account when rendering for new <code>vrfigure</code> objects. This preference also applies to new <code>vr.canvas</code> objects. Valid values are 'off' and 'on'.
DefaultFigureWireframe	Specifies whether objects are drawn as solids or wireframes by default for new <code>vrfigure</code> objects. This preference also applies to new <code>vr.canvas</code> objects. Valid values are 'off' and 'on'.
DefaultViewer	Specifies which viewer is used to view a virtual scene. <ul style="list-style-type: none"> <li>• 'internal' Default Simulink 3D Animation Viewer.</li> <li>• 'web' Web browser becomes viewer. This is the current Web browser virtual world plug-in.</li> </ul>
DefaultWorldRecord3D FileName	Specifies the default 3-D animation file name for new <code>vrworld</code> objects.
DefaultWorldRecordMode	Specifies the default animation recording mode for new <code>vrworld</code> objects. Valid values are 'manual' and 'scheduled'.
DefaultWorldRecord Interval	Specifies the default start and stop times for scheduled animation recording for new <code>vrworld</code> objects. Valid value is a vector of two doubles.
DefaultWorldRemoteView	Specifies whether the virtual world is enabled by default for remote viewing for new <code>vrworld</code> objects. Valid values are 'off' and 'on'.
DefaultWorldTimeSource	Specifies the default source of the time for new <code>vrworld</code> objects. Valid values are 'external' and 'freerun'.

Preference	Description
Editor	<p>Specifies which virtual world editor to use. Path to the virtual world editor. If this path is empty, the MATLAB editor is used.</p> <p>The path setting is active only if you select the Custom option.</p> <p>To open a virtual world file in a third-party editor, do not use the <code>vredit</code> command. For example, to open a virtual world in the Ligos V-Realm Builder editor:</p> <ol style="list-style-type: none"> <li>1 Set the default editor to V-Realm Builder. In MATLAB, enter: <code>vrsetpref('Editor','VREALM');</code></li> <li>2 To open a file in the V-Realm editor, in MATLAB navigate to a virtual world file, right-click, and select <b>Edit</b>.</li> </ol> <p><b>Note</b> The <code>vredit</code> command opens the 3D World Editor, regardless of the default editor preference setting.</p>
EditorPreserveLayout	<p>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).</p>
HttpPort	<p>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.</p>
TransportBuffer	<p>Length of the transport buffer (network packet overlay) for communication between the Simulink 3D Animation server and its clients.</p>
TransportTimeout	<p>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.</p>
VrPort	<p>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.</p>

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
<code>'internal'</code>	Default Simulink 3D Animation Viewer.
<code>'web'</code>	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:

<code>%matlabroot</code>	Refers to the MATLAB root folder
<code>%file</code>	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.

**See Also**

`vrgetpref`

**Introduced before R2006a**



# 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 <code>UpperPositionLimit</code> and <code>LowerPositionLimit</code> 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 <code>Output Type</code> as <code>Speed</code> .
InitialPosition	Initial condition for integrated translations. This is not used when you set the <code>Output Type</code> to <code>Speed</code> .

Property	Description
InitialRotation	Initial condition for integrated rotations. This is not used when you set the Output Type to Speed.

**Methods**

Method	Description
button	<code>b = button(mouse, n)</code> reads the status of space mouse button number <code>n</code> . Button status is returned as logical 0 if not pressed and logical 1 if pressed. <code>n</code> can be a vector to return multiple buttons.
close	<code>close(mouse)</code> closes and invalidates the space mouse object. The object cannot be used once it is closed.
position	<code>p = position(mouse, n)</code> reads the position of space mouse axis number <code>n</code> . <code>n</code> 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	<code>s = speed(mouse, n)</code> reads the speed of space mouse axis number <code>n</code> . <code>n</code> can be a vector to return the speeds of multiple axes. No transformations are done. Outputs are the translation and rotation speeds.
viewpoint	<code>p = viewpoint(mouse)</code> 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.

**See Also**

“Set Simulink 3D Animation Preferences” on page 2-5

**Introduced in R2007b**

## 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];`

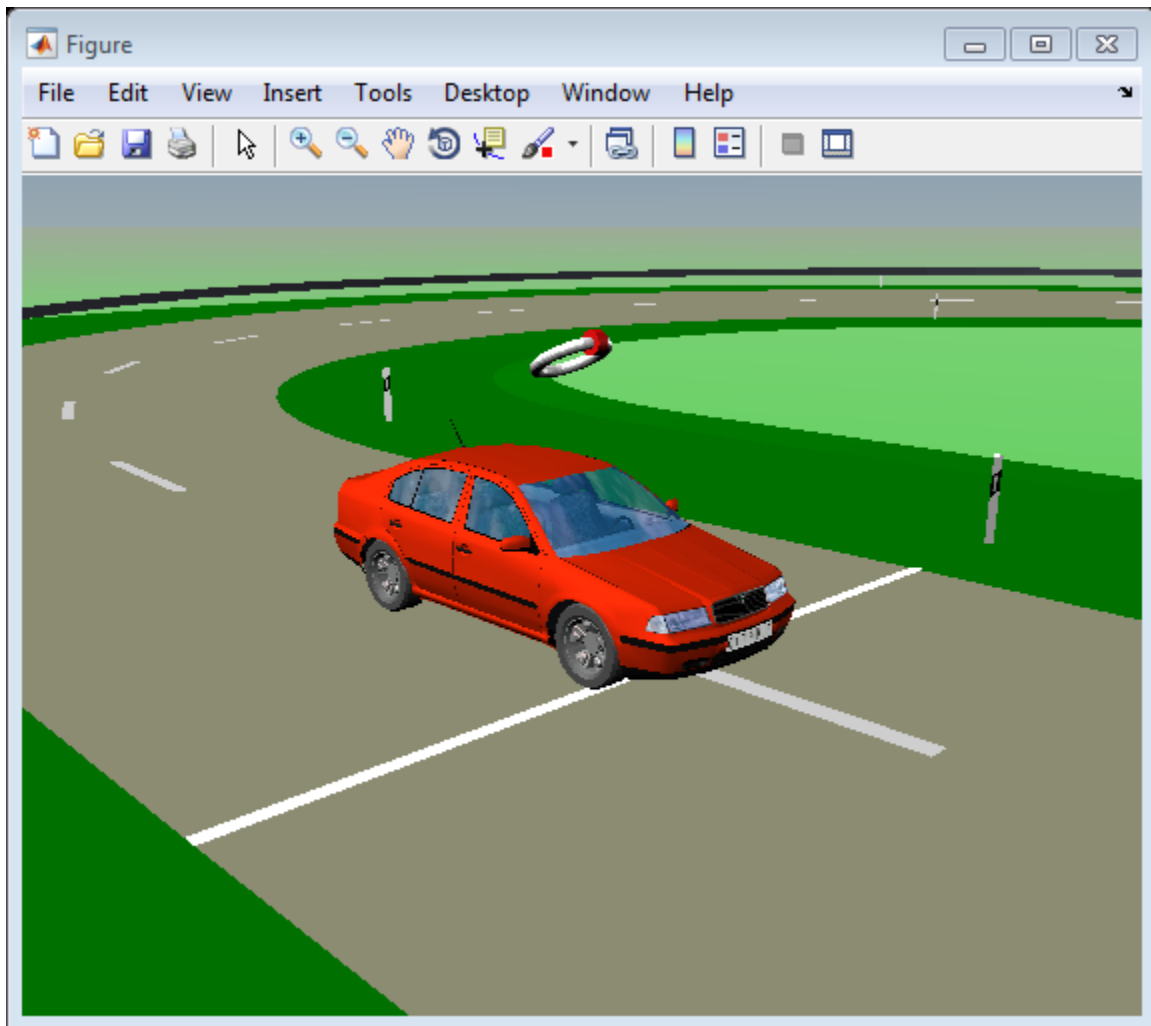
```
Example: stereo3d_object.RightCameraFilter = stereo3d_object.CAMERA_FILTER_RED
```

## 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)
```

## See Also

`vr.canvas` | `vrfigure`

## Topics

“View a Virtual World in Stereoscopic Vision” on page 7-45

## Introduced in R2015a

## 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.wrl`.

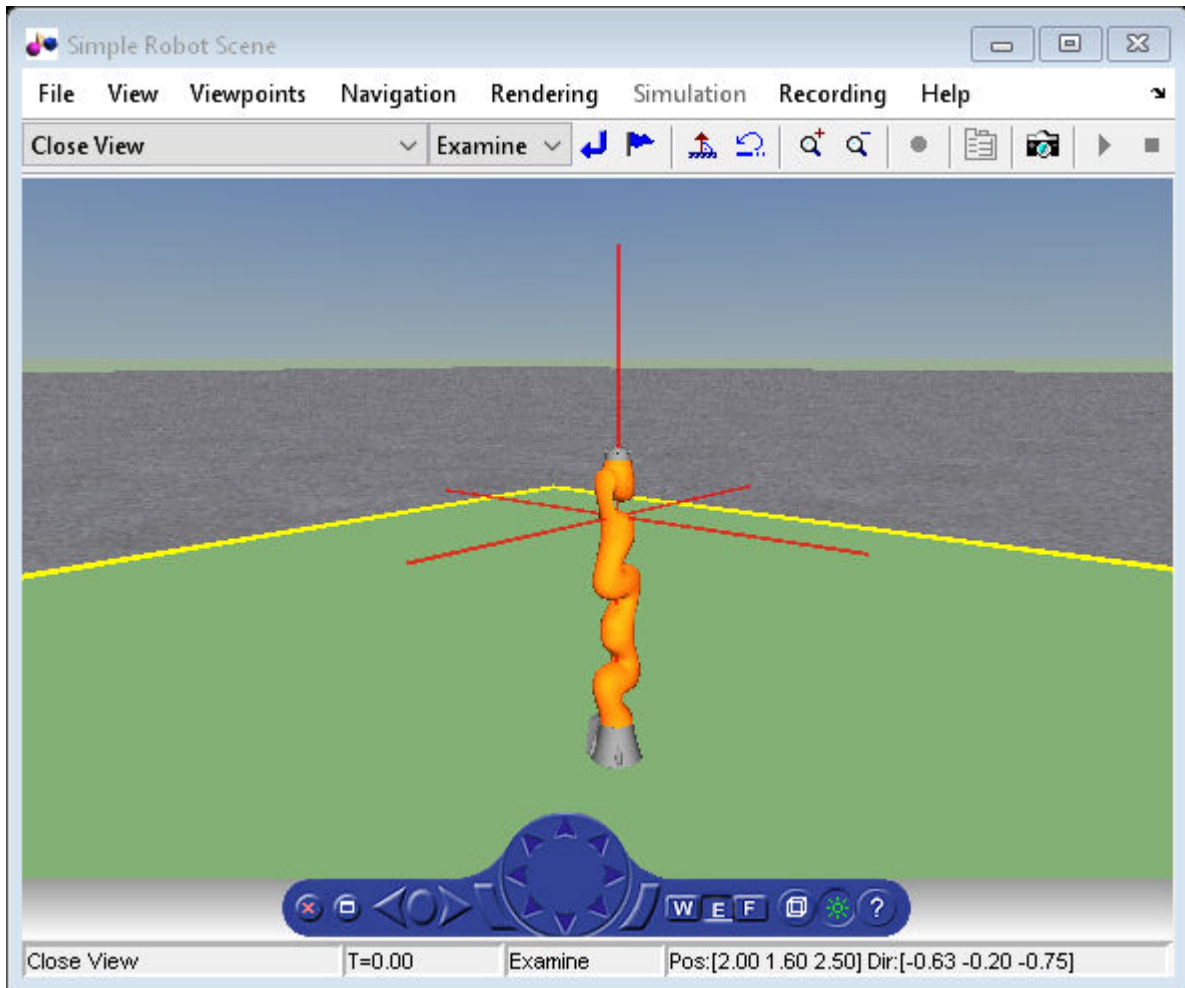
```
RBT = importrobot('iiwa7.urdf');  
RBT.DataFormat = 'row';  
robotWorld = vrworld('robot_scene');  
open(robotWorld);
```

#### 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.

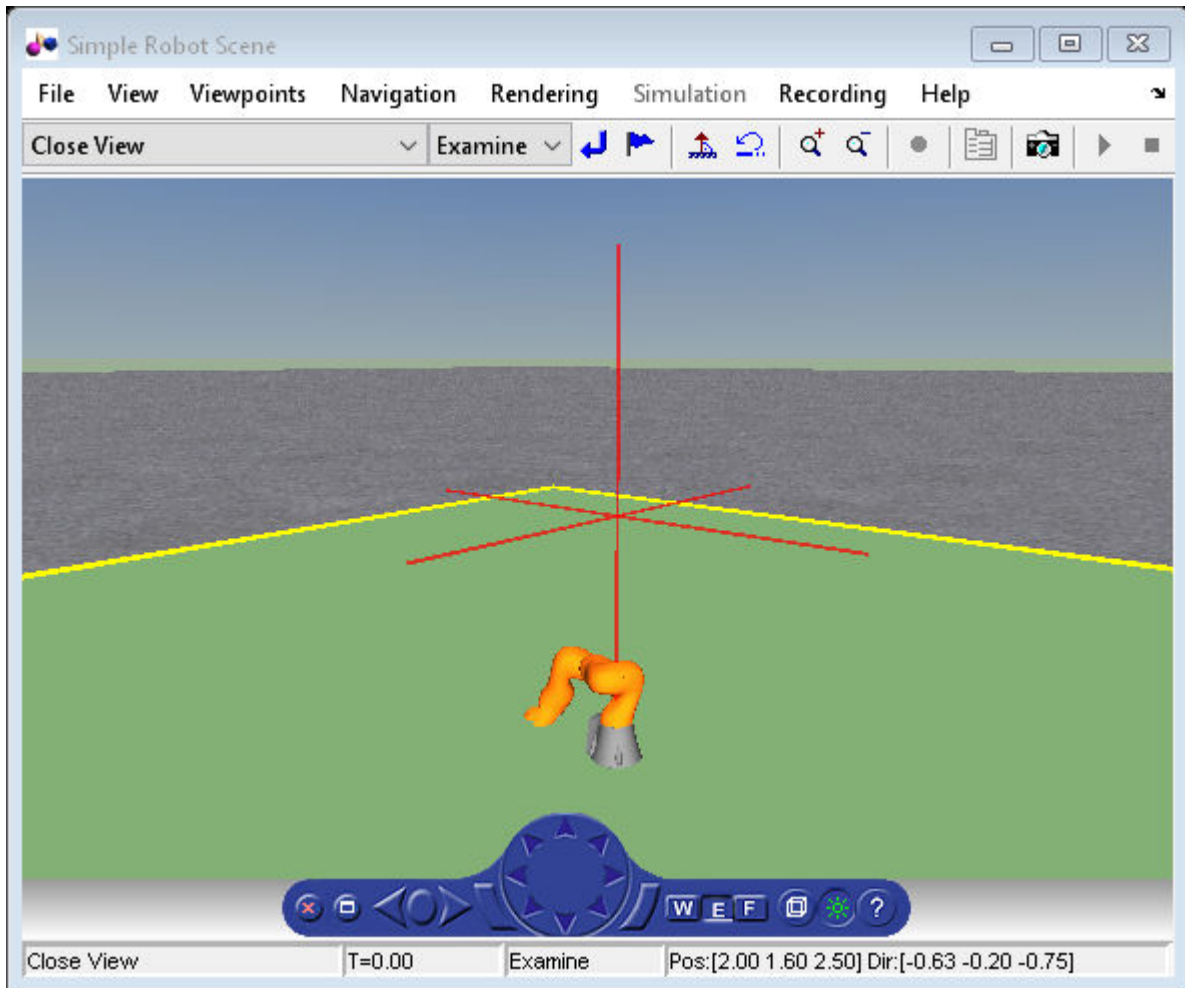
```
[node, W, tforms] = vrinsertrobot(robotWorld, RBT);  
vrfigure(robotWorld);
```





### 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).

### t forms — Robot transforms

cell array

List of robot transforms, specified as a cell array of `vnode` 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`

## **See Also**

`vrinsertrobot` | `rigidBodyTree` (Robotics System Toolbox)

**Introduced in R2018b**

## 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.

### See Also

`vrplay` | `vrworld` | `vrworld/open` | `vrworld/view`

**Introduced before R2006a**

# 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.

## See Also

`vrclear` | `vrwhos` | `vrworld`

**Introduced before R2006a**

## **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`.

### **See Also**

`vrclear` | `vrwho`

**Introduced before R2006a**

## 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 <code>.wrl</code> , <code>.x3d</code> , or <code>.x3dv</code> . If no file extension is specified, the file extension <code>.wrl</code> 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
<code>addexternproto</code>	Add <code>externproto</code> declaration to virtual world.
<code>close</code>	Close virtual world
<code>delete</code>	Remove virtual world from memory
<code>edit</code>	Open virtual world file in external virtual world editor
<code>get</code>	Property value of <code>vrworld</code> object
<code>isvalid</code>	1 if <code>vrworld</code> object is valid, 0 if not
<code>nodes</code>	List nodes available in virtual world
<code>open</code>	Open virtual world
<code>reload</code>	Reload virtual world from virtual world 3D file
<code>save</code>	Write virtual world to virtual world 3D file
<code>set</code>	Change property values of <code>vrworld</code> object
<code>view</code>	View virtual world

## See Also

`vrworld/close` | `vrworld/delete` | `vrworld/open` | “Create `vrworld` Object for a Virtual World” on page 4-2 | “Open a Virtual World with MATLAB” on page 4-3

**Introduced before R2006a**



## vrworld/addexternproto

Add externproto declaration to virtual world

### Syntax

```
addexternproto(vrworld_object, protofile, protoname)
```

```
addexternproto(vrworld_object, protofile, protoname, protodef)
```

### Arguments

<code>vrworld_object</code>	A <code>vrworld</code> object representing the virtual world.
<code>protofile</code>	String containing the name of the prototype file from which the EXTERNPROTO declaration is added.
<code>protoname</code>	String containing the name of the EXTERNPROTO declaration.
<code>protodef</code>	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 EXTERNPROTOs 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 EXTERNPROTOs 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.

### See Also

`vrworld/close` | `vrworld/delete` | `vrworld/open` | “Create `vrworld` Object for a Virtual World” on page 4-2 | “Open a Virtual World with MATLAB” on page 4-3

**Introduced in R2008b**

## **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)
```

### **See Also**

`vrworld` | `vrworld/delete` | `vrworld/open` | “Close and Delete a `vrworld` Object” on page 4-9

**Introduced before R2006a**

## 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.

### See Also

`vrworld` | `vrclear` | `vrworld/delete` | `vrworld/open` | “Close and Delete a `vrworld` Object” on page 4-9

**Introduced before R2006a**

## **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.

### **See Also**

`vrworld/reload` | `vrworld/save` | `vrworld/delete` | `vrworld/open` | “Close and Delete a `vrworld` Object” on page 4-9 | “Create `vrworld` Object for a Virtual World” on page 4-2

**Introduced before R2006a**

## 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.
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.

Property	Value	Description
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" on page 4-14. Read/write.
Recording	'off'   'on' Default: 'off'	Animation recording toggle. This property acts as the master 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.
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.

Property	Value	Description
View	'off'   'on' Default: 'on'	Indicates an unviewable or viewable virtual world. Read/write.

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.

## See Also

vrworld | vrworld/set

**Introduced before R2006a**

## **vrworld/invalid**

1 if vrworld object is valid, 0 if not

### **Syntax**

```
x = invalid(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 = invalid(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.

### **See Also**

`invalid | vrnode/invalid`

**Introduced before R2006a**



## vrworld/nodes

List nodes available in virtual world

### Syntax

```
nodes(vrworld_object, '-full')
```

```
x = nodes(vrworld_object, '-full')
```

### Arguments

<code>vrworld_object</code>	A <code>vrworld</code> object representing a virtual world.
<code>'-full'</code>	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.

### See Also

`vrworld` | `vrworld/open` | `vrnode`

**Introduced before R2006a**

## 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.

### See Also

`vrworld` | `vrworld/close` | “Open a Virtual World with MATLAB” on page 4-3

**Introduced before R2006a**

## 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.

### See Also

`vrworld/edit` | `vrworld/open` | `vrworld/save` | “Open a Virtual World with MATLAB” on page 4-3

**Introduced before R2006a**

## 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

<code>vrworld_object</code>	vrworld object representing a virtual world
<code>file</code>	Name of virtual world 3D file, specified as a string. You can specify a <code>.wrl</code> (VRML), <code>.x3dv</code> (XML encoded) or <code>.x3d</code> (X3D in classic or XML format) file.
<code>'-export'</code>	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.
<code>'-nothumbnail'</code>	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 CR-LF or LF, according to the local system default. Values are separated by spaces.

You can use the optional `'-export'` and `'-nothumbnail'` arguments either by themselves or together, in addition to the required `vrworld_object` and `file` arguments.

### See Also

`vrworld/edit` | `vrworld/open` | `vrworld/reload` | “Close and Delete a vrworld Object” on page 4-9

**Introduced before R2006a**

## vrworld/set

Change property values of `vrworld` object

### Syntax

```
set(vrworld_object, 'property_name', property_value)
```

### Arguments

<code>vrworld_object</code>	Name of a <code>vrworld</code> object representing a virtual world.
<code>property_name</code>	Name of the property.
<code>property_value</code>	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 <code>vrfigure</code> 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 <code>vrnode</code> objects	Vector of <code>vrnode</code> 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" on page 4-14. Read/write.
Recording	'off'   'on' Default: 'off'	Animation recording toggle. This property acts as the master 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.

## See Also

vrworld | vrworld/get

Introduced before R2006a

## 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('vrend.wrl')
open(myworld)
view(myworld)
```

### See Also

`vrview` | `vrworld`

**Introduced before R2006a**



# Simulink 3D Animation Player

Play recorded 3D animation files

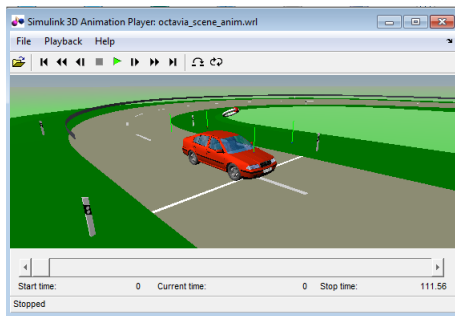
## Description

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**.

## **See Also**

### **Apps**

**3D World Editor**

### **Functions**

vrsetpref | vrview

### **Topics**

“Play Animation Files” on page 4-27

“Play Animation Files” on page 7-31

“Play Animations with Simulink 3D Animation Viewer” on page 7-35

“Animation Recording” on page 4-10

“File Name Tokens” on page 4-14

### **Introduced in R2006a**

# 3D World Editor

Edit virtual worlds for 3D animation

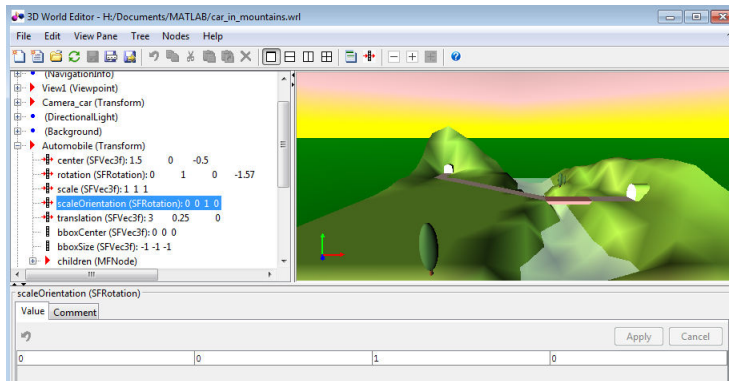
## Description

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” on page 6-9
- “Build and Connect a Virtual World” on page 5-8
- “Edit a Virtual World” on page 6-11
- “Reduce Number of Polygons for Shapes” on page 6-20
- “Add Sound to a Virtual World” on page 5-35
- “View a Virtual World in Stereoscopic Vision” on page 7-45
- “Virtual Reality World and Dynamic System Examples” on page 1-16

## See Also

### Apps

**Simulink 3D Animation Player**

### Functions

`vrsetpref` | `vrview`

### Topics

- “Create a Virtual World” on page 6-9
- “Build and Connect a Virtual World” on page 5-8
- “Edit a Virtual World” on page 6-11
- “Reduce Number of Polygons for Shapes” on page 6-20
- “Add Sound to a Virtual World” on page 5-35
- “View a Virtual World in Stereoscopic Vision” on page 7-45
- “Virtual Reality World and Dynamic System Examples” on page 1-16
- “3D World Editor” on page 6-2
- “X3D Support” on page 1-9
- “Virtual Reality Modeling Language (VRML)” on page 1-11
- “3D World Editor Library” on page 6-26
- “Virtual World Navigation in 3D World Editor” on page 6-21

### Introduced in R2010b