

# Simulink® 3D Animation™

## User's Guide

**R2012b**

**MATLAB®  
& SIMULINK®**

## How to Contact MathWorks



[www.mathworks.com](http://www.mathworks.com) Web  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab) Newsgroup  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html) Technical Support



[suggest@mathworks.com](mailto:suggest@mathworks.com) Product enhancement suggestions  
[bugs@mathworks.com](mailto:bugs@mathworks.com) Bug reports  
[doc@mathworks.com](mailto:doc@mathworks.com) Documentation error reports  
[service@mathworks.com](mailto:service@mathworks.com) Order status, license renewals, passcodes  
[info@mathworks.com](mailto:info@mathworks.com) Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



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

For contact information about worldwide offices, see the MathWorks Web site.

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

© COPYRIGHT 2001–2012 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)



## Getting Started

**1**

<b>Product Description</b> .....	<b>1-2</b>
Key Features .....	1-2
<b>Expected Background</b> .....	<b>1-3</b>
<b>Build Virtual Worlds to Visualize Dynamic</b>	
<b>Simulations</b> .....	<b>1-4</b>
Virtual Reality World Models of Dynamic Systems .....	1-4
Set up Your Working Environment .....	1-5
Build a Virtual Reality World .....	1-5
Link to a Virtual Reality World .....	1-6
View Dynamic System Simulations .....	1-8
Share Dynamic System Simulation Visualizations .....	1-8
MATLAB Compiler Support .....	1-9
<b>Virtual Reality Modeling Language (VRML)</b> .....	<b>1-10</b>
VRML History .....	1-10
VRML Support .....	1-11
VRML Compatibility .....	1-12
VRML Coordinate System .....	1-13
VRML File Format .....	1-15
<b>Virtual Reality World and Dynamic System</b>	
<b>Examples</b> .....	<b>1-18</b>
Simulink Interface Examples .....	1-18
MATLAB Interface Examples .....	1-31

<b>Required Products</b> .....	2-2
Section Overview .....	2-2
MATLAB Product .....	2-2
VRML Viewer .....	2-3
<b>Recommended Product</b> .....	2-4
Simulink Product .....	2-4
<b>Related Products</b> .....	2-5
<b>System Requirements</b> .....	2-6
Section Overview .....	2-6
Supported Computer Platforms .....	2-6
Host Computer .....	2-8
Client Computer .....	2-10
<b>Installing Simulink 3D Animation Software on the Host</b>	
<b>Computer</b> .....	2-12
Section Overview .....	2-12
Components on a Host Computer .....	2-12
Installing from a DVD (Windows) .....	2-13
Installing from a DVD (UNIX/Linux) .....	2-14
<b>Blaxxun Contact Host Computer Installation</b> .....	2-15
Section Overview .....	2-15
Simulink 3D Animation Viewer .....	2-15
Installing a VRML Plug-In (Windows) .....	2-16
Installing a VRML Plug-In (UNIX and Linux) .....	2-19
Set the Default Viewer .....	2-20
<b>Installing the VRML Editor on the Host Computer</b> ....	2-25
V-Realm Editor Installation (Windows) .....	2-25
V-Realm Builder Help .....	2-26
Set the Default Editor .....	2-26
<b>Set Simulink 3D Animation Preferences</b> .....	2-29
Section Overview .....	2-29

Simulink 3D Animation Preferences .....	2-30
3D World Editor Preferences .....	2-33
Canvas Preferences .....	2-34
Figure Preferences .....	2-34
Figure Rendering Preferences .....	2-35
Figure 2-D Recording Preferences .....	2-37
Figure Frame Capture Preferences .....	2-38
Virtual World Preferences .....	2-39
<b>Removing Components (Windows) .....</b>	<b>2-42</b>
Section Overview .....	2-42
Uninstall V-Realm Builder .....	2-42
Uninstall Blaxxun Contact from Host Computer .....	2-43
<b>Blaxxun Contact Client Computer Installation .....</b>	<b>2-44</b>
Section Overview .....	2-44
Installing a VRML Plug-In (Windows) .....	2-44
<b>Testing the Viewer Installation .....</b>	<b>2-45</b>
Section Overview .....	2-45
Simulink Testing .....	2-45
MATLAB Testing .....	2-50

## Simulink Interface

# 3

<b>Virtual World Connection to a Model .....</b>	<b>3-2</b>
Add a Simulink 3D Animation Block .....	3-2
Changing the Virtual World Associated with a Simulink Block .....	3-8
<b>Using the Simulink Interface .....</b>	<b>3-11</b>
Section Overview .....	3-11
Displaying a Virtual World and Starting Simulation .....	3-11
View Virtual World on Host Computer .....	3-14
View Virtual World Remotely .....	3-18
<b>Working with VRML Sensors .....</b>	<b>3-25</b>

Add VRML Sensors to Virtual Worlds .....	3-25
Read VRML Sensor Values .....	3-26
<b>VR Source Block Input to Simulink Models .....</b>	<b>3-29</b>
<b>Interact with Generated Code .....</b>	<b>3-30</b>

## MATLAB Interface

# 4

<b>Using the MATLAB Interface .....</b>	<b>4-2</b>
Create a vrworld Object .....	4-2
Open a Virtual World .....	4-3
Interact with a Virtual World .....	4-5
Close and Delete a vrworld Object .....	4-9
<b>Recording Offline Animations .....</b>	<b>4-10</b>
Animation Recording .....	4-10
Manual and Scheduled Animation Recording .....	4-11
Animation Recording File Tokens .....	4-12
Manual 3-D VRML Animation Recording .....	4-14
Manual 2-D AVI Animation Recording .....	4-16
Scheduled 3-D VRML Animation Recording .....	4-20
Scheduled 2-D AVI Animation Recording .....	4-22
Viewing Animation Files .....	4-25
Animation Recording of Unconnected Virtual Worlds .....	4-27

## Build Virtual Reality Worlds

# 5

<b>VRML Editors .....</b>	<b>5-2</b>
Editors for Virtual Worlds .....	5-2
<b>Build and Connect a Virtual World .....</b>	<b>5-7</b>
Introduction .....	5-7



Define the Problem .....	5-7
Add a Simulink 3D Animation Block .....	5-9
Open a New Virtual World .....	5-11
Add VRML Nodes .....	5-11
Link to a Simulink Model .....	5-20
<b>VRML Data Types .....</b>	<b>5-24</b>
Section Overview .....	5-24
VRML Field Data Types .....	5-24
VRML Data Class Types .....	5-27
<b>Simulink 3D Animation Textures .....</b>	<b>5-29</b>
<b>Using CAD Models with the Simulink 3D Animation</b>	
<b>Product .....</b>	<b>5-30</b>
Section Overview .....	5-30
Export VRML Models from CAD Tools .....	5-30
CAD Virtual World Modeling .....	5-37
Link to CAD Virtual Worlds .....	5-40
Export VRML Models from CATIA Software .....	5-46

## Using the 3D World Editor

# 6

<b>3D World Editor .....</b>	<b>6-2</b>
Supported Platforms .....	6-2
Use with Other Editors .....	6-2
VRML Support .....	6-2
VRML Nodes, Library Objects, and Templates .....	6-3
3D World Editor Interface .....	6-4
<b>Getting Started with the 3D World Editor .....</b>	<b>6-7</b>
3D World Editor Is the Default Editor .....	6-7
Open the Editor .....	6-7
Create a Virtual World .....	6-8
<b>Basic Editing .....</b>	<b>6-12</b>
Add Objects .....	6-12

Copy and Paste a Node .....	6-14
Edit Object Properties .....	6-15
Document a Virtual World Using Comments .....	6-15
Expand and Collapse Nodes .....	6-16
Wrap Nodes as Children of Another Node .....	6-16
Remove Nodes .....	6-17
Save and Export Virtual World Files .....	6-17
Navigate a Virtual World .....	6-18
Specify Virtual World Rendering .....	6-19
Edit VRML Scripts .....	6-20
<b>3D World Editor Library .....</b>	<b>6-22</b>
3D World Editor Library Objects .....	6-22
Add a Library Object .....	6-22
Guidelines for Using Custom Objects .....	6-23

## Viewing Virtual Worlds

# 7

<b>VRML Viewers .....</b>	<b>7-2</b>
<b>Simulink 3D Animation Viewer .....</b>	<b>7-4</b>
Interface .....	7-4
Menu Bar .....	7-8
Toolbar .....	7-9
Navigation Panel .....	7-10
Setting Viewer Appearance Preferences .....	7-13
Starting and Stopping Simulations .....	7-13
Navigate a Virtual World .....	7-14
Define File Name Tokens .....	7-22
File Name Tokens .....	7-24
Capture Frames .....	7-25
Animation Recording File Formats .....	7-27
Record Files in the VRML Format .....	7-28
Record Files in the Audio Video Interleave (AVI) Format ..	7-29
Schedule Files for Recording .....	7-32
Start and Stop Animation Recording .....	7-35
View Animation Files .....	7-35
Viewpoints .....	7-37
Navigate Through Viewpoints .....	7-38

Reset Viewpoints .....	7-41
Define Viewpoints .....	7-41
Specify Rendering Techniques .....	7-45
<b>Blaxxun Contact VRML Plug-In .....</b>	<b>7-54</b>
Blaxxun Contact Interface .....	7-54
Navigate Through Viewpoints .....	7-55
Control Menu .....	7-55
Navigate a Virtual World .....	7-56
Movement Modes .....	7-57
Blaxxun Contact Settings .....	7-57
Specify Rendering Techniques .....	7-58
Stereoscopic Vision .....	7-58
Using Blaxxun Contact on a Client Computer .....	7-59
<b>Legacy Simulink 3D Animation Viewer .....</b>	<b>7-61</b>
Introduction .....	7-61
Starting the Legacy Viewer .....	7-61
Differences Between the Default and Legacy Viewer .....	7-63
Differences When Setting the DefaultViewer Property to 'internalv4' .....	7-64

## Simulink 3D Animation Stand-Alone Viewer

# 8

<b>What Is Orbisnap? .....</b>	<b>8-2</b>
<b>Orbisnap Installation .....</b>	<b>8-3</b>
Section Overview .....	8-3
System Requirements .....	8-3
Copying Orbisnap to Another Location .....	8-3
Adding Shortcuts or Symbolic Links .....	8-4
<b>Using Orbisnap .....</b>	<b>8-5</b>
Overview .....	8-5
View WRL Animations or Virtual Worlds .....	8-6
View Virtual Worlds Remotely .....	8-7

<b>Orbisnap Interface</b> .....	8-10
Menu Bar .....	8-10
Toolbar .....	8-11
Navigation Panel .....	8-11
 <b>Start Orbisnap</b> .....	 8-17

## Block Reference

### 9

<b>Control Input Devices</b> .....	9-2
<b>Utilities</b> .....	9-3
<b>Virtual Worlds</b> .....	9-4
<b>VRML-Related Signals</b> .....	9-5

## Blocks — Alphabetical List

### 10

## Function Reference

### 11

<b>MATLAB Interface</b> .....	11-2
<b>vr.canvas Object Methods</b> .....	11-4
<b>vrworld Object Methods</b> .....	11-5
<b>vrnode Object Methods</b> .....	11-6

<b>vrfigure Object Methods</b> .....	<b>11-7</b>
<b>Input Device Objects</b> .....	<b>11-8</b>

## **Functions — Alphabetical List**

**12**

---

**Glossary**

---

**Index**



# Getting Started

---

- “Product Description” on page 1-2
- “Expected Background” on page 1-3
- “Build Virtual Worlds to Visualize Dynamic Simulations” on page 1-4
- “Virtual Reality Modeling Language (VRML)” on page 1-10
- “Virtual Reality World and Dynamic System Examples” on page 1-18

## Product Description

### **Animate and visualize models in three dimensions**

Simulink® 3D Animation™ provides an interface linking Simulink models and MATLAB® algorithms to 3D graphics objects. It lets you visualize and verify dynamic system behavior in a virtual reality environment. Objects are represented in the Virtual Reality Modeling Language (VRML), an open 3D modeling standard. You can animate a 3D world by changing object properties such as position, rotation, and scale during desktop or real-time simulation. You can also access 3D animation data in Simulink or MATLAB for post-processing.

Simulink 3D Animation includes a viewer for rendering and recording high-quality animations. With the 3D World Editor, you can author detailed scenes assembled from 3D models exported from CAD- or Web-based sources. You can incorporate multiple 3D scene views inside MATLAB figures and interact with these views via a force-feedback joystick, space mouse, or other hardware device.

### **Key Features**

- Simulink blocks and MATLAB functions for connecting models to virtual reality worlds
- 3D World Editor for authoring 3D worlds
- Video recording and animation playback
- Visualization of real-time simulations
- Client/server architecture
- Interaction with 3D views via a joystick, space mouse, or other hardware device



## Expected Background

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

This guide 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<sup>®</sup> charts products to create models as block diagrams and simulate those models
- VRML, to create or otherwise provide virtual worlds or three-dimensional scenes to connect to Simulink or MATLAB software

## Build Virtual Worlds to Visualize Dynamic Simulations

In this section...
“Virtual Reality World Models of Dynamic Systems” on page 1-4
“Set up Your Working Environment” on page 1-5
“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 1-8
“Share Dynamic System Simulation Visualizations” on page 1-8
“MATLAB® Compiler™ Support” on page 1-9

### 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 and Simulink, SimMechanics™, 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, you perform the following tasks:

- “Set up Your Working Environment” on page 1-5
- “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 1-8
- “Share Dynamic System Simulation Visualizations” on page 1-8

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 5-7.

## Set up Your Working Environment

Installing the Simulink 3D Animation software in your MATLAB environment provides the tools that you need to build virtual reality worlds and to visualize dynamic simulations modeled in MATLAB, Simulink, or SimMechanics.

You build and view the virtual reality world models using VRML (Virtual Reality Modeling Language).

In addition to the installed VRML editor, 3D World 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 VRML editor
- The MATLAB editor or a third-party text editor

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

- The Blaxxun Contact® VRML plug-in, on either the host or a client computer
- Orbisnap, on a client computer

To help decide which VRML editor and viewer to use, see “VRML Editors” on page 5-2 and “VRML Viewers” on page 7-2. For more information about installing a VRML editor or VRML viewer, see “Install VRML Viewer”.

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

## Build a Virtual Reality World

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

The Simulink 3D Animation product uses many of the advanced features defined in the current VMRL97 specification, including:

- 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 more an overview of VRML and details about supported VRML features, see “Virtual Reality Modeling Language (VRML)” on page 1-10.

As you add VRML nodes with the 3D World Editor, 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 5-7.

## **Link to a Virtual Reality World**

To use a dynamic system simulation to drive a virtual reality world, you need to connect the virtual world to one of the following:

- Simulink model
- SimMechanics model
- MATLAB virtual world object

## **Simulink**

The Simulink 3D Animation library provides blocks to directly connect Simulink signals with virtual worlds. This connection lets you visualize your model as a three-dimensional animation.

You can implement most of the software features with Simulink blocks. Once 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 VRML nodes that the Simulink software can drive.

All the VRML 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 VRML viewer.

The Simulink product provides communication for control and manipulation of virtual reality objects, using Simulink 3D Animation blocks.

For details, see “Virtual World Connection to a Model” on page 3-2.

### **SimMechanics**

You can use the Simulink 3D Animation product to view the behavior of a model created with the SimMechanics software. First, you build a model of a machine in the Simulink interface using SimMechanics blocks. Then, create a detailed picture of your machine in a virtual world, connect this world to the SimMechanics body sensor outputs, and view the behavior of the bodies in a VRML viewer.

For details, see “Link to a SimMechanics Model” on page 5-45.

### **MATLAB**

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.

From the MATLAB software, you can set positions and properties of VRML objects, create callbacks from graphical user interfaces (GUIs), and map data to virtual objects. You can also view the world with a VRML 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 VRML files corresponding to the actual structure of a virtual world.

The MATLAB software provides communication for control and manipulation of virtual reality objects using MATLAB objects.

For details about interacting between MATLAB and virtual reality worlds, see “Using the MATLAB Interface” on page 4-2.

## **View Dynamic System Simulations**

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

- In Simulink and SimMechanics, 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 in a virtual world.

For information about using VRML 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 by:

- Capture animation frame snapshots or record animations for video viewing. See “Capture Frames” on page 7-25 and “Recording Offline Animations” on page 4-10.
- In addition to the single computer configuration (when MATLAB, Simulink, and the virtual reality representations run on the same host computer), Simulink 3D Animation software also allows a client-server configuration. In this configuration, an Orbisnap or Blaxxun Contact VRML viewer on a remote client can connect to the server host on which Simulink 3D Animation software is running. This allows others to view an animated virtual world remotely. Multiple clients can connect to one server. See “Blaxxun Contact VRML Plug-In” on page 7-54 and “Using Orbisnap” on page 8-5.
- 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 1-9

## **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 (`vr1ib`).
- No Simulink 3D Animation server, which results in no remote connection for the Orbisnap or Blaxxun viewers
- 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

## Virtual Reality Modeling Language (VRML)

In this section...
“VRML History” on page 1-10
“VRML Support” on page 1-11
“VRML Compatibility” on page 1-12
“VRML Coordinate System” on page 1-13
“VRML File Format” on page 1-15

### VRML History

The Virtual Reality Modeling Language (VRML) is the language you use to display three-dimensional objects with a VRML viewer.

Since people started to publish their documents on the World Wide Web (WWW), there has been an effort to enhance the content of Web pages with advanced three-dimensional graphics and interaction with those graphics.

The term Virtual Reality Markup Language (VRML) was first used by Tim Berners-Lee at a European Web conference in 1994 when he talked about a need for a 3-D Web standard. Soon afterward, an active group of artists and engineers formed around a mailing list called `www-vrml`. They changed the name of the standard to Virtual Reality Modeling Language to emphasize the role of graphics. The result of their effort was to produce the VRML 1 specification. As a basis for this specification, they used a subset of the Inventor file format from Silicon Graphics.

The VRML 1 standard was implemented in several VRML browsers, but it allowed you to create only static virtual worlds. This limitation reduced the possibility of its widespread use. Quickly it became clear that the language needed a robust extension to add animation and interactivity, and bring life to a virtual world. The VRML 2 standard was developed, and in the year 1997 it was adopted as International Standard ISO/IEC 14772-1:1997. Since then it is referred to as VRML97.



VRML97 represents an open and flexible platform for creating interactive three-dimensional scenes (virtual worlds). As computers improve in computational power and graphic capability, and communication lines become faster, the use of 3-D graphics becomes more popular outside the traditional domain of art and games. There are now a number of VRML97-enabled browsers available on several platforms. Also, there are an increasing number of VRML authoring tools from which to choose. In addition, many traditional graphical software packages (CAD, visual art, and so on) offer VRML97 import/export features.

The Simulink 3D Animation product uses VRML97 technology to deliver a unique, open 3-D visualization solution for MATLAB users. It is a useful contribution to a wide use of VRML97 in the field of technical and scientific computation and interactive 3-D animation.

The VRML97 standard continues to be improved by the Web 3D Consortium. The newly released X3D (eXtensible 3D) standard is the successor to VRML97. X3D is an extensible standard that provides compatibility with existing VRML content and browsers. For more information, see <http://www.web3d.org> and <http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/>.

## **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 VRML 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 term VRML, in this guide, always refers to VRML as defined in the VRML97 standard ISO/IEC 14772-1:1997, available from <http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/>. 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** Since some VRML worlds are automatically generated in VRML1.0, and the Simulink 3D Animation product does not support VRML1.0, you need to 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

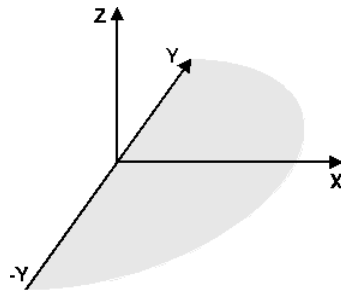
The Simulink 3D Animation product currently supports most features of VRML97, with the following limitations:

- The Simulink 3D Animation server ignores the VRML Script node, but it passes the node to the VRML viewer. This allows you to run VRML scripts on the viewer side. You cannot run them on the Simulink 3D Animation server.
- The Simulink 3D Animation viewer does not support the Sound node. (The Blaxxun Contact viewer does.)
- In keeping with the VRML97 specification, the Simulink 3D Animation viewer ignores BMP files. As a result, VRML scene textures might not display properly in the Simulink 3D Animation viewer. To properly display scene textures, replace all BMP texture files in a VRML scene with PNG,

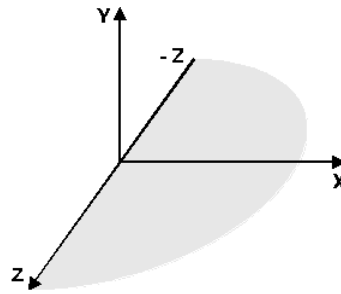
JPG, or GIF equivalents. Note that Blaxxun Contact supports BMP files in addition to the standard VRML texture file formats.

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

## VRML Coordinate System



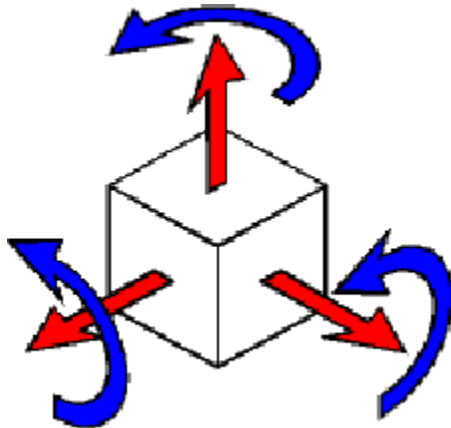
MATLAB graphics coordinate system



VRML coordinate system

The VRML coordinate system is different from the MATLAB and Aerospace Blockset™ coordinate systems. VRML uses the *world coordinate system* in which the *y*-axis points upward and the *z*-axis places objects nearer or farther from the front of the screen. It is important to realize this fact in situations involving the interaction of these different coordinate systems. SimMechanics uses the same coordinate system as VRML.

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, the position and orientation of child objects are specified 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*.

## 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, it is useful to have a basic knowledge of VRML scene description. This helps you create virtual worlds more effectively, and gives you a good understanding of how the virtual world elements can be controlled using Simulink 3D Animation software.

This section introduces VRML. For more information, see the VRML97 Reference. This reference is available online at <http://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.

In VRML, a 3-D scene is described by a hierarchical tree structure of objects (nodes). Every node in the tree represents some functionality of the scene. There are 54 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 examples:

- Box node — Represents a box in a scene.
- Transform node — Defines position, scale, scale orientation, rotation, translation, and children of its subtree (grouping node).
- Material node — Corresponds to material in a scene.
- DirectionalLight node — Represents lighting in a scene.
- Fog node — Allows you to modify the environment optical properties.
- ProximitySensor node — Brings interactivity to VRML97. This node generates events when the user enters, exits, and moves 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

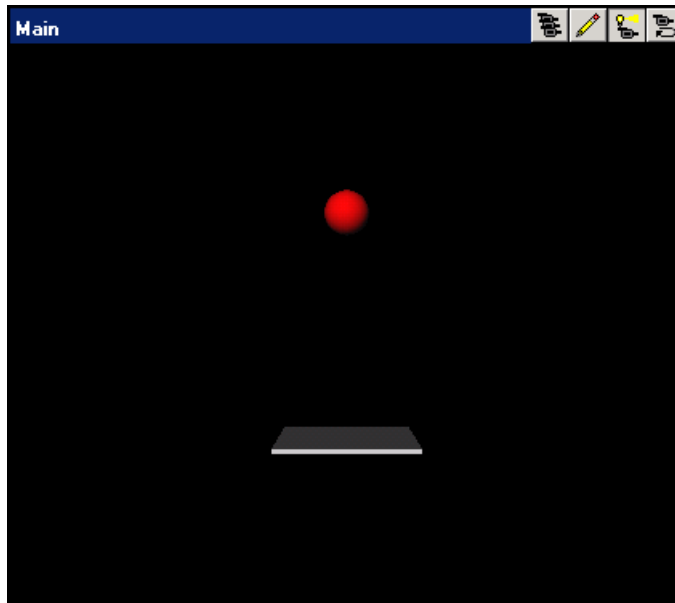
In the following example of a simple VRML file, two graphical objects are modeled in a 3-D scene: A floor is represented by a flat box with a red ball above it. The VRML file is a readable text file that you can write in any text editor.

```
#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 this 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. Everything on a line after the # sign is ignored by a VRML viewer, with the exception of 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, the floor and red ball look like this:



## Virtual Reality World and Dynamic System Examples

In this section...
“Simulink Interface Examples” on page 1-18
“MATLAB Interface Examples” on page 1-31

### 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 VRML-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 VRML 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.

<b>Example</b>	<b>Simulink Coder Ready</b>	<b>VR Sink</b>	<b>VR Source</b>	<b>Joystick</b>	<b>Space Mouse</b>
vrbounce	X	X			
vrcrane_joystick		X		X	
vrcrane_panel		X	X		
vrcrane_traj	X	X			
vrlights	X	X			
vrmaglev		X	X		
vrmaglev_rtwin	X	X			
vrmanipul		X			X
vrmanipul_global		X	X		
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			

## **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 VRML file.

For details on how the crane model hierarchy and scripting logic is implemented, see the associated commented VRML file `portal_crane.wr1`.

## **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 VRML sensors with logic that changes their visual appearance based on user input. The VRML 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 (vr crane\_traj)**

The vr crane\_traj example is based on the vr crane\_joystick example, but instead of interactive control, it has a predefined load trajectory. The vr crane\_traj model illustrates a technique to create the visual impression of joining and splitting moving objects in the VRML 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 VRML 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 VRML 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 VRML 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 VRML 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 `vrexin` S-function provides the data connection.

For more details on how to read values from virtual worlds programmatically, see “Working with VRML Sensors” on page 3-25.

### **Magnetic Levitation Model for Real-Time Windows Target Example (vrmaglev\_rtwin)**

In addition to the `vrmaglev` example, the `vrmaglev_rtwin` example works directly with the actual CE 152 scale model hardware in real time. MathWorks created this model to work with the HUMUSOFT MF 624 data acquisition board, and Simulink Coder™ and Real-Time Windows Target™ 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.

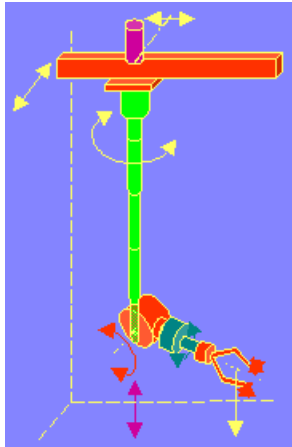
After enabling the remote view in the VR Sink block dialog box, you can control the Simulink model even from another (remote) client computer. This control can be useful for distributing the computing power between a real-time Simulink model running on one machine and the rendering of a virtual reality 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 VRML 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 VRML 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 VRML 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 VRML97 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\_1chg\_2cars.wrl virtual world, the VRML after the definition of PROTOS 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 `PixelFormat` 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 VRML 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 System 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 VRML 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 VRML 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 VRML **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 VRML 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 VRML 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 VRML 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 VRML Transform that contains the Text node).

## 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. “S-Function Limitations”

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				
vrtekoff_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 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 VRML 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 MATLAB 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 VRML `IndexedFaceSet` is colored based on the simulation of some physical phenomenon, it is suitable to create 2-D `.avi` animation files. The software uses the `avifile` function 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 GUI 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 `vrm1` 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 VRML scene, notice the text object. It is a child of the VRML 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.



# Installation

---

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

- “Required Products” on page 2-2
- “Recommended Product” on page 2-4
- “Related Products” on page 2-5
- “System Requirements” on page 2-6
- “Installing Simulink® 3D Animation™ Software on the Host Computer” on page 2-12
- “Blaxxun Contact Host Computer Installation” on page 2-15
- “Installing the VRML Editor on the Host Computer” on page 2-25
- “Set Simulink® 3D Animation™ Preferences” on page 2-29
- “Removing Components (Windows)” on page 2-42
- “Blaxxun Contact Client Computer Installation” on page 2-44
- “Testing the Viewer Installation” on page 2-45

## Required Products

In this section...
“Section Overview” on page 2-2
“MATLAB Product” on page 2-2
“VRML Viewer” on page 2-3

### Section Overview

The Simulink 3D Animation product is part of a family of MathWorks products. You need to install some of these products and other third-party products to use Simulink 3D Animation.

### MATLAB Product

The MATLAB software provides the tools to write scripts and functions in MATLAB code. You can use your scripts to set positions and properties of VRML objects, create callbacks from GUIs, and map data to virtual objects.

The MATLAB documentation explains how to work with data and how to use the functions supplied with the MATLAB software. For a reference describing the functions specific to the Simulink 3D Animation software, see “MATLAB Interaction”.

## VRML Viewer

You use a VRML viewer to visualize and explore virtual worlds described with VRML. The following are descriptions of VRML viewers:

- Simulink 3D Animation viewer — This viewer is installed with the Simulink 3D Animation product and is the default viewer for virtual worlds. You can access this viewer from either a Simulink block in your Simulink model, or by using the MATLAB command line.

The Simulink 3D Animation viewer is a client to the Simulink 3D Animation server. It does not require a Web browser. It is supported on Windows, Mac OS X, UNIX, and Linux platforms. The viewer is the recommended method for viewing virtual worlds on a host computer.

- Blaxxun Contact Version 4.4 — VRML plug-in shipped with the PC version of the Simulink 3D Animation product. This VRML plug-in allows you to view virtual worlds in your Web browser. The Blaxxun Contact plug-in is a supported VRML plug-in.

You can view a virtual world in the Simulink 3D Animation viewer. If you want to view the virtual world in your Web browser, you need to use the `vrinstall` command to install the Blaxxun Contact plug-in. See “Installing a VRML Plug-In (Windows)” on page 2-16.

For information on using a Web browser to view virtual worlds, see “Testing the Viewer Installation” on page 2-45. The Blaxxun Contact installation executable files are located at `C:\matlabroot\toolbox\s13d\blaxxun`.

Every VRML plug-in installs Sun™ Java™ classes into the Web browser. Limit the number of plug-ins you use to avoid Java errors and conflicts. For this reason, use only the Simulink 3D Animation viewer or the Blaxxun Contact VRML plug-in on PC platforms. On UNIX and Linux platforms, use only the Simulink 3D Animation viewer.

### **Recommended Product**

Optionally, you can install the Simulink product to use the Simulink 3D Animation product.

#### **Simulink Product**

The Simulink product provides an environment where you model your dynamic system and controller as a block diagram. You create the block diagram interactively to by connecting blocks and editing block parameters.

With Simulink 3D Animation, you can connect and animate 3-D worlds associated with your dynamic systems modeled in Simulink.

## Related Products

Several MathWorks products are especially relevant to the kinds of tasks you can perform with the Simulink 3D Animation product.

For more information about any of these products, see either of the following:

- Online documentation for that product, if it is installed on your system
- The MathWorks Web site, at  
<http://www.mathworks.com/products/3d-animation/related.html>

## System Requirements

In this section...
“Section Overview” on page 2-6
“Supported Computer Platforms” on page 2-6
“Host Computer” on page 2-8
“Client Computer” on page 2-10

### Section Overview

The Simulink 3D Animation product has the same hardware requirements as the MATLAB product. It is a multiplatform product that runs on PC-compatible computers with Microsoft® Windows or Linux operating systems. It runs on SGI, Solaris™, and Alpha hardware running UNIX, and also on Apple Power Macintosh hardware running Mac OS X.

### Supported Computer Platforms

The Simulink 3D Animation server is the part of the Simulink 3D Animation software that interfaces with your Simulink models. It stores information about the current state of virtual worlds and manages connections to VR clients. The VR client is a VRML viewer that displays a virtual world. The VR client can be either the Simulink 3D Animation viewer or a Web browser with a VRML plug-in.

The following table summarizes the supported computer platforms and the viewer and editor that are provided for each of them.

<b>Platform/Product</b>	<b>VR Server</b>	<b>Simulink 3D Animation Viewer</b>	<b>VRML Editor</b>	<b>VRML Browser Plug-In</b>
Windows	Yes	Yes	3D World Editor and Ligos V-Realm Builder	Blaxxun Contact
Windows x64	Yes	Yes	3D World Editor and Ligos V-Realm Builder	Blaxxun Contact
Linux kernels	Yes	Yes	3D World Editor	No
Linux x86-64 kernels	Yes	Yes	3D World Editor	No
Sun Solaris 64	Yes	Yes	3D World Editor	No
Mac OS X	Yes	Yes	3D World Editor	No

## Host Computer

The host computer is a desktop computer where you install the MATLAB, Simulink, and Simulink 3D Animation products, a VRML editor and, optionally, a Web browser with a VRML plug-in. You can also install the Simulink Coder product with Real-Time Windows Target or xPC Target™ software to run and view a real-time application.

The following table lists the minimum resources the software requires on the host computer.

### Host Computer Hardware Requirements

Hardware	Description
CPU	Intel® Pentium, Athlon or higher (PC)
Graphics card	Graphics card with hardware 3-D acceleration
RAM	128 Mbytes or more
Peripherals	Hard disk drive with 45 Mbytes of free space DVD-ROM drive
TCP/IP communication	If you want to allow a connection from a client computer, you need a network connection between the host computer and the client computer.

The following table lists the minimum software the software requires on your host computer. For a list of optional software products, see <http://www.mathworks.com/products/3d-animation/related.html>.

### Host Computer Software Requirements

Software	Description
MATLAB product	Version 7.9.
Simulink product	Version 7.4. The Simulink product is not required, but highly recommend.
Simulink 3D Animation product	Version 5.1.



**Host Computer Software Requirements (Continued)**

<b>Software</b>	<b>Description</b>
VRML editor	<p>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 VRML editor.</p> <p>The V-Realm Builder is also included with the Simulink 3D Animation software, on Windows platforms. See “V-Realm Editor Installation (Windows)” on page 2-25.</p>
Web browser	<p>On PC platforms, you can use a Web browser and the Blaxxun Contact plug-in to view virtual worlds. This is an alternative to using the Simulink 3D Animation viewer.</p> <p>Use Internet Explorer® software Version 4.0 or later, or Netscape Navigator® 4.0 or later with Sun Java enabled.</p>
VRML plug-in	<p>If you are using a Web browser instead of the Simulink 3D Animation viewer, you need to install a VRML97 plug-in with External Authoring Interface (EAI) support. If you have Blaxxun Contact (Windows) on your computer, you have already installed a VRML plug-in.</p> <p><b>Windows platforms</b> — You can install the Blaxxun Contact 4.4 plug-in provided with the Simulink 3D Animation product.</p> <p>For information on how to install the Blaxxun Contact plug-in, see “Installing a VRML Plug-In (Windows)” on page 2-16.</p>

### **LD\_LIBRARY\_PATH Environment Variable (UNIX)**

If your system does not have the OpenGL<sup>®</sup> software properly installed when you run the Simulink 3D Animation viewer, you might see an error message like the following in the MATLAB window:

```
Invalid MEX-file 'matlab/toolbox/s13d/s13d/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 like the following 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 like the following:

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

In both cases, <PLATFORM> is the UNIX platform you are working in.

## **Client Computer**

You can use a client computer to view and control a virtual world. Because the MATLAB or Simulink product does not run on this computer, you must connect to a host computer running a simulation or executable code. The host computer, through the Simulink 3D Animation server, provides the values needed to animate a virtual world.

The client computer communicates with the host computer over TCP/IP, and it displays the virtual world using a VR client. In this case, the VR client is a VRML-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, you must first fix the TCP/IP protocol settings according to the documentation for your operating system.

The following table lists the minimum hardware resources the Simulink 3D Animation product needs on the client computer.

## Client Computer Hardware Requirements

Hardware	Description
Graphics card	Graphics card with hardware 3-D acceleration.
TCP/IP communication	If you want to allow a connection from a client computer, you need a network connection between the host computer and the client computer.

The following table lists the software the Simulink 3D Animation installation requires on the client computer. You do not need to install the software on the client computer.

Because the only component required for the client computer is standard VRML97 viewing software, it is possible that different configurations will work. For example, you might be able to run an operating system not listed in the table “Supported Computer Platforms” on page 2-6. However, these configurations have not been tested and they are not supported.

## Client Computer Software Requirements

Software	Description
Operating system	Windows XP (the TCP/IP protocol must be installed).
Web browser	Use Internet Explorer 4.0 or later, or Netscape Navigator 4.0 or later with Java enabled.
VRML plug-in	<p>VRML97 plug-in with External Authoring Interface support. If you have the Blaxxun Contact software (Windows) on your computer, you have already installed a VRML plug-in.</p> <p><b>Windows platforms</b> -- You can install the Blaxxun Contact 4.4 plug-in provided with the Simulink 3D Animation product.</p> <p>For information on how to install the Blaxxun Contact plug-in, see “Installing a VRML Plug-In (Windows)” on page 2-16.</p>

# Installing Simulink 3D Animation Software on the Host Computer

**In this section...**

“Section Overview” on page 2-12

“Components on a Host Computer” on page 2-12

“Installing from a DVD (Windows)” on page 2-13

“Installing from a DVD (UNIX/Linux)” on page 2-14

## Section Overview

You may install the Simulink 3D Animation software from a DVD or from the MathWorks Web site. Before you install the software for a standard installation, you need your online MathWorks Account. For detailed information about the installation process, see the MathWorks installation documentation.

## Components on a Host Computer

This section introduces you to the individual components of the Simulink 3D Animation software: what they are, what they are used for, and when they should or should not be installed. If you are not interested, you can skip this section, or you can simply accept the defaults at the component selection screen, and the recommended default components are installed.

- Simulink 3D Animation software — This component contains the core files that interconnect the MATLAB and Simulink interfaces to VRML. This component is required for the software to operate, and you must install it on the host computer. This component is *not* used on a client computer.
- Simulink 3D Animation viewer — This is a multiplatform VRML viewer that is included with the Simulink 3D Animation software, and it is set as the default viewer for displaying virtual worlds.
- VRML plug-in — Optionally, you can use a VRML plug-in for a Web browser to view virtual reality worlds. The Blaxxun Contact plug-in is included with the Simulink 3D Animation product for Microsoft Windows

platforms. A VRML plug-in is the only component that you need to install on a client computer.

- VRML editor — If you are going to create and modify virtual worlds, you need a VRML97-compatible editor. Your choices are:
  - The 3D World Editor, which Simulink 3D Animation software installs on all supported platforms
  - The Ligos V-Realm Builder, for Windows platforms,
  - Any third-party VRML editor
  - The MATLAB editor or a third-party text editor
- Example models — These are MATLAB and Simulink programs and models connected to prebuilt virtual reality worlds. You can use these models and virtual reality worlds both for discovering the capabilities of the Simulink 3D Animation product and as templates for building your own projects. This component is not used on the client computer.
- Online documentation — This component contains the guide you are reading now. You can access the online version through the MATLAB Help browser. An Adobe® Acrobat® PDF file is available on the MathWorks Web site at <http://www.mathworks.com>. Follow the links to product documentation. This documentation can be read using the Adobe Acrobat Reader. If you do not have this reader installed on your computer, you can download it from <http://www.adobe.com>.

## Installing from a DVD (Windows)

To install the Simulink 3D Animation software from a DVD on a Windows platform:

- 1** Insert the DVD into your host DVD-ROM drive.

The installation program should start automatically after a few seconds. If the installation program does not start automatically, run `setup.exe` on the DVD.

- 2** Follow the instructions on each of the screens to complete the installation.

The Simulink 3D Animation viewer is installed with the software. For PC platforms, you have the option of installing a VRML plug-in for your browser

as an alternative to the viewer. See “Installing a VRML Plug-In (Windows)” on page 2-16.

If you are using a PC platform, you must complete additional steps for installing the VRML editor. See “V-Realm Editor Installation (Windows)” on page 2-25.

## **Installing from a DVD (UNIX/Linux)**

The following is an overview of how to install the Simulink 3D Animation software on a UNIX or Linux platform. For a comprehensive description of the installation process, see the MathWorks installation guide .

- 1** Log in to your system.
- 2** As necessary, mount the DVD-ROM drive.
- 3** Run the appropriate installation script for your platform.
- 4** During the installation process, a dialog box allows you to select the products to install.

This dialog box lists all the products you are licensed to install in the **Items to Install** box. Make sure the Simulink 3D Animation product is listed in this box.

- 5** Follow the instructions on each of the remaining screens to complete the installation.

The Simulink 3D Animation viewer is the default viewer for UNIX platforms. For more information, see “Simulink® 3D Animation™ Viewer” on page 2-15.

# Blaxxun Contact Host Computer Installation

## In this section...

“Section Overview” on page 2-15

“Simulink® 3D Animation™ Viewer” on page 2-15

“Installing a VRML Plug-In (Windows)” on page 2-16

“Installing a VRML Plug-In (UNIX and Linux)” on page 2-19

“Set the Default Viewer” on page 2-20

## Section Overview

You can use the Simulink 3D Animation viewer or VRML-enabled Web browser to view virtual worlds.

The Simulink 3D Animation viewer is the only viewer that can be used on all supported platforms. The Blaxxun Contact plug-in is available for PC platforms only.

## Simulink 3D Animation Viewer

The Simulink 3D Animation viewer is the recommended method of viewing a virtual world. The viewer can be used on any supported operating system. It is installed and set as the default viewer when you install the software. You can view virtual worlds as soon as the software is installed on your machine.

---

**Note** It is possible to view virtual worlds with a Web browser that contains a VRML plug-in. Every VRML plug-in installs Java classes into the Web browser. It is best to limit the number of plug-ins you install on your machine to avoid Sun Java errors and conflicts. For this reason, use only the Simulink 3D Animation viewer and the Blaxxun Contact VRML plug-in on PC platforms. On UNIX and Linux platforms, use only the Simulink 3D Animation viewer.

---

## Installing a VRML Plug-In (Windows)

When you install the Simulink 3D Animation software, the Simulink 3D Animation viewer is set as the default viewer. If you want to use a Web browser as a VRML viewer, use the following procedure to install the Blaxxun Contact plug-in. You can use this plug-in with either the Internet Explorer or Netscape Navigator browser. The Blaxxun Contact plug-in is the only supported VRML plug-in.

---

**Note** The Blaxxun Contact installer installs the plug-in for the current default browser only. If you change the default browser, you need to complete the install procedure a second time. The Blaxxun Contact installation executable files are located at `C:\matlabroot\toolbox\s13d\blaxxun`.

---

You must use Blaxxun Contact 4.4 with the Simulink 3D Animation product. This version of the Blaxxun Contact VRML plug-in is distributed with the software. The following procedure describes how to install the plug-in.

If you have the MATLAB Web Server installed on your machine, make sure that the Web Server is stopped before you install the Blaxxun Contact plug-in. Also, verify that you are connected to the Internet before starting this installation procedure:

- 1 Start the MATLAB software.
- 2 In the MATLAB Command Window, type

```
vrinstall -install viewer
```

The MATLAB interface displays the message

```
Do you want to use OpenGL or Direct3d  
acceleration? (o/d)
```

- 3 Check the graphics card manual to determine the acceleration method to select. If you are not sure, select Direct 3d by typing

```
d
```

The Blaxxun installer starts running and displays the following dialog box.





**4** Follow the instructions on the remaining screens.

**5** In the MATLAB Command Window, type

```
vrinstall
-check
```

If the viewer installation was successful, the MATLAB Command Window displays the following message:

```
External VRML viewer:    installed
```

If the viewer installation was unsuccessful, the MATLAB Command Window displays the message

```
VRML
viewer:    not installed
```

### **Known Issue with the Blaxxun Contact Plug-In**

The Blaxxun Contact VRML plug-in can fail to update the virtual world when used with the Simulink 3D Animation and Internet Explorer 5.5 and later products. Netscape Navigator users do not experience this problem.

If you are using Internet Explorer 5.5 or later, you must manually change a network security setting before you can use Blaxxun Contact 4.4 with Simulink 3D Animation Version 3.0 or later. Upgrading your version of the Blaxxun Contact plug-in does not resolve this problem.

The Blaxxun Contact VRML plug-in does not work reliably on Microsoft Windows 7 platforms.

### **Changing the Default Network Security Setting**

You must change your default network security setting before using the Blaxxun Contact plug-in with Internet Explorer 5.5 and later to ensure that the virtual world is updated appropriately. You can use this workaround for the following:

- PC platform is Windows XP Service Pack 1.
- The PC platform is not one of the above, but you have installed the Microsoft Java Virtual Machine (JVM) on the PC.

**1** Open Internet Explorer.

**2** From the **Tools** menu, choose **Internet Options**.

The Internet Options dialog box opens.

**3** Select the **Local Intranet** icon.

**4** Click the **Security** tab.

**5** Click the **Custom Level** button.

The Security Settings dialog box opens.

**6** Scroll down until you see **Microsoft VM**. The first subheading is **Java permissions**.

**7** Select **Custom**.

The **Java Custom Settings** button appears in the lower left of the Security Settings dialog box.

**8** Click **Java Custom Settings**.

The Local intranet dialog box opens.

**9** Click the **Edit Permissions** tab.

**10** Scan the main headings and subheadings (marked with a lock icon) until you see **Run Unsigned Content**.

**11** Under **Run Unsigned Content**, find **Access to all Network Addresses**.

**12** Under **Access to all Network Addresses**, select **Enable**.

**13** Click **OK**.

The Local intranet dialog box closes.

**14** In the Security Settings dialog box, click **OK**.

You are asked if you want to change the security settings for this zone.

**15** Select **Yes**.

**16** In the Internet Options dialog box, click **OK**.

## **Installing a VRML Plug-In (UNIX and Linux)**

If you want to use a Web browser instead of the Simulink 3D Animation viewer to view virtual worlds, you need to install a VRML97 plug-in with External Authoring Interface (EAI) support. This requirement is met by the Blaxxun Contact plug-in for Microsoft platforms. If you are using any other operating system, you need to use the Simulink 3D Animation viewer to view virtual worlds.

---

**Note** The Blaxxun Contact interface is the only supported VRML plug-in.

---

## Set the Default Viewer

If you install a VRML plug-in in your Web browser, it is possible to view virtual worlds with either the Simulink 3D Animation viewer, legacy Simulink 3D Animation viewer, or your Web browser. You determine the viewer used to display your scene using the `vrsetpref` and `vrgetpref` commands. (Alternatively, if you want to use the MATLAB File menu Preferences dialog box, see “Set Simulink® 3D Animation™ Preferences” on page 2-29.)

The following procedure describes how to set the viewer to the Simulink 3D Animation viewer or the Web browser. It assumes that you are working with a PC platform. If you want to use the legacy Simulink 3D Animation viewer, see “Legacy Simulink® 3D Animation™ Viewer” on page 7-61.

- 1 At the MATLAB command prompt, type

```
vrinstall -check
```

to determine whether the Blaxxun Contact software is installed.

The MATLAB Command Window displays

```
VRML  
viewer:    installed VRML editor:    installed
```

The viewer and editor are installed. If the viewer is not installed, see “Installing a VRML Plug-In (Windows)” on page 2-16.

- 2 Determine your default viewer by typing

```
vrgetpref
```

The MATLAB Command Window displays

```
ans =  
  
          DataTypeBool: 'logical'  
          DataTypeInt32: 'double'  
          DataTypeFloat: 'double'  
DefaultCanvasNavPanel: 'none'  
          DefaultCanvasUnits: 'normalized'  
DefaultEditorPosition: [325 216 763 901]
```

```

        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'
        DefaultFigureTransparency: 'on'
        DefaultFigureTriad: 'none'
        DefaultFigureWireframe: 'off'
        DefaultViewer: 'internal'
DefaultWorldRecord3DFileName: '%f_anim_%n.wrl'
        DefaultWorldRecordMode: 'manual'
DefaultWorldRecordInterval: [0 0]
        DefaultWorldRemoteView: 'off'
        DefaultWorldTimeSource: 'external'
        Editor: '*BUILTIN'
        EditorSavePosition: 'on'
        HttpPort: 8123
        TransportBuffer: 5
        TransportTimeout: 20
        VrPort: 8124

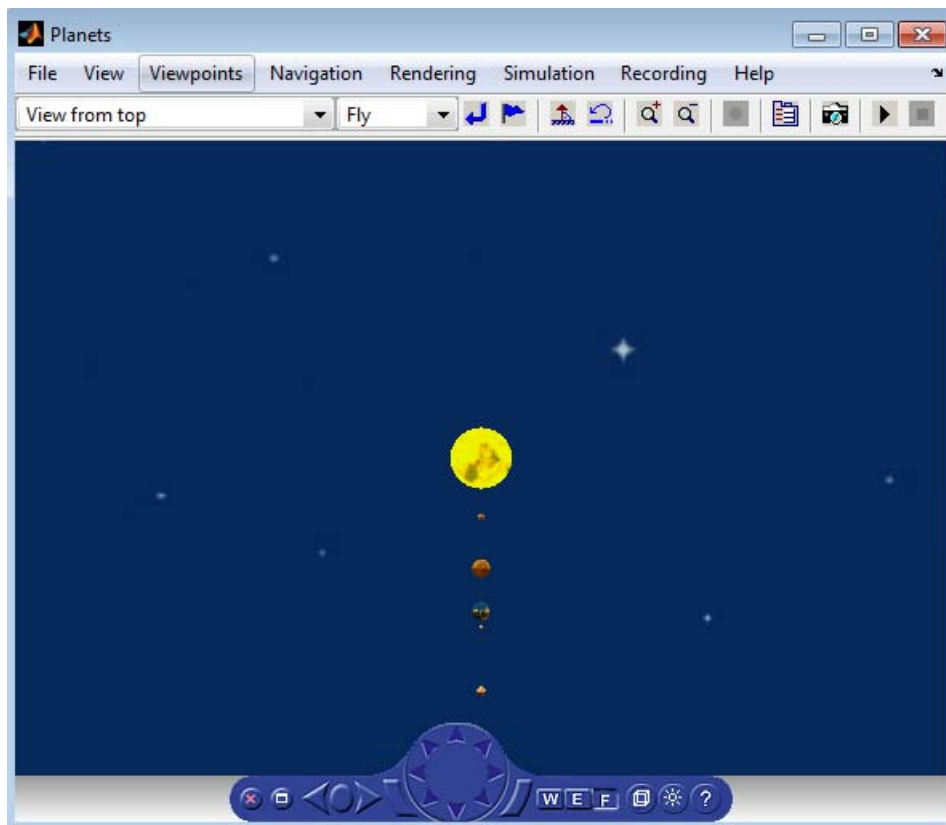
```

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.

**3** 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.



- 4** Change the default viewer to your Web browser by typing

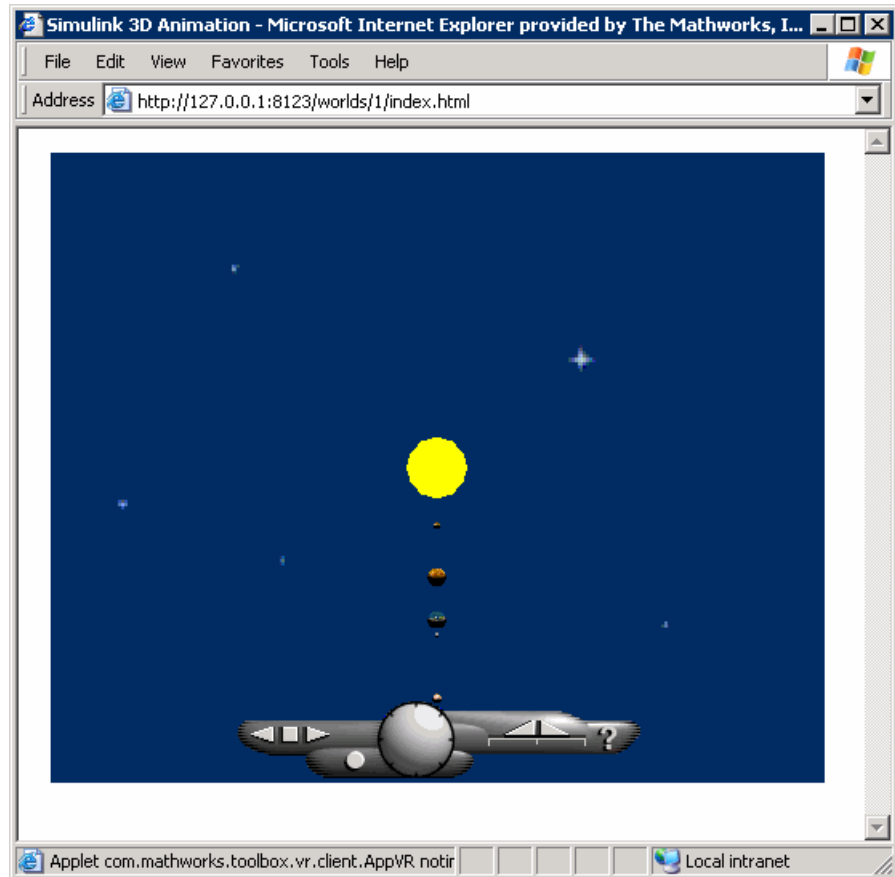
```
vrsetpref('DefaultViewer','web')
```

The default Windows system VRML plug-in is used. The Blaxxun Contact VRML plug-in sets itself as the default VRML plug-in during its installation.

- 5** At the MATLAB command prompt, type

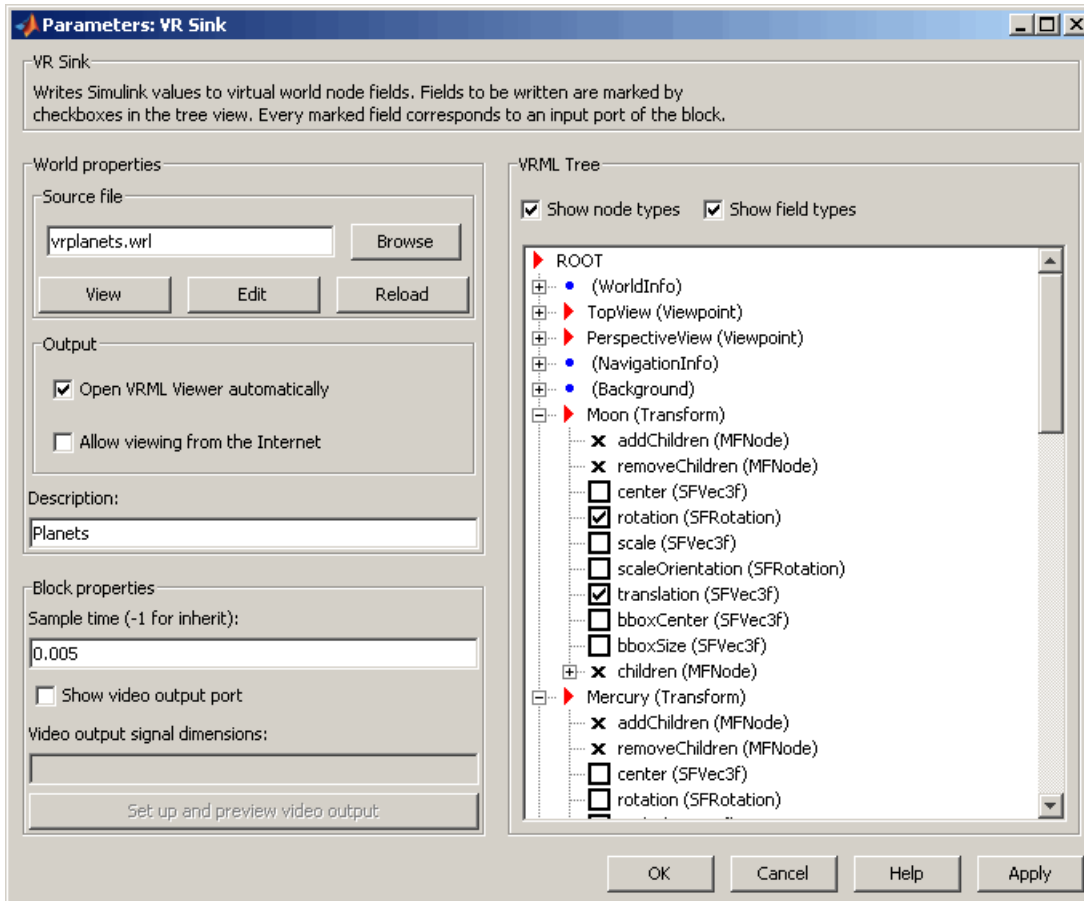
```
vrplanets
```

The Planets example is loaded and the virtual world is displayed in your Web browser.



- 6 Reset the Simulink 3D Animation viewer as your default viewer by typing  
`vrsetpref('DefaultViewer','factory')`
- 7 In the Simulink 3D Animation viewer for `vrplanets`, from the **Simulation** menu, select **Block Parameters**.

A Parameters: VR Sink dialog box opens.



The target of the **View** button is determined by the `DefaultViewer` property. If the `DefaultViewer` property is set to 'internal', clicking the **View** button opens the virtual world in the Simulink 3D Animation viewer. If the `DefaultViewer` property is set to 'web', clicking the **View** button opens the virtual world in your Web browser.



## Installing the VRML Editor on the Host Computer

### In this section...

“V-Realm Editor Installation (Windows)” on page 2-25

“V-Realm Builder Help” on page 2-26

“Set the Default Editor” on page 2-26

### V-Realm Editor Installation (Windows)

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 VRML editor available on Windows platforms. However, the installation is not complete.

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

- 1 Start the MATLAB software.
- 2 In the MATLAB Command Window, type

```
vrinstall  
-install editor
```

or type

```
vrinstall('-install','editor')
```

The MATLAB Command Window displays the following messages:

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

- 3 Type

```
vrinstall  
-check
```

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

VRML editor:      installed

## **V-Realm Builder Help**

To access V-Realm Builder help, view Doc Center by clicking the **MATLAB Help** button. You can view the V-Realm Builder help even if you have not installed V-Realm Builder.

If you are reading this in the MATLAB Help browser, see the V-Realm Builder help.

You cannot access the V-Realm Builder documentation from the Web. If you are reading this page on the Web, then you need to open the MATLAB Help browser and navigate to this page. If you have MATLAB open:

- 1** From the MATLAB Toolstrip, click the **Help** button. Doc Center opens.
- 2** In Doc Center, browse to **Install Editor and Viewer > Install V-Realm Builder > V-Realm Builder Help**.
- 3** See the V-Realm Builder help.

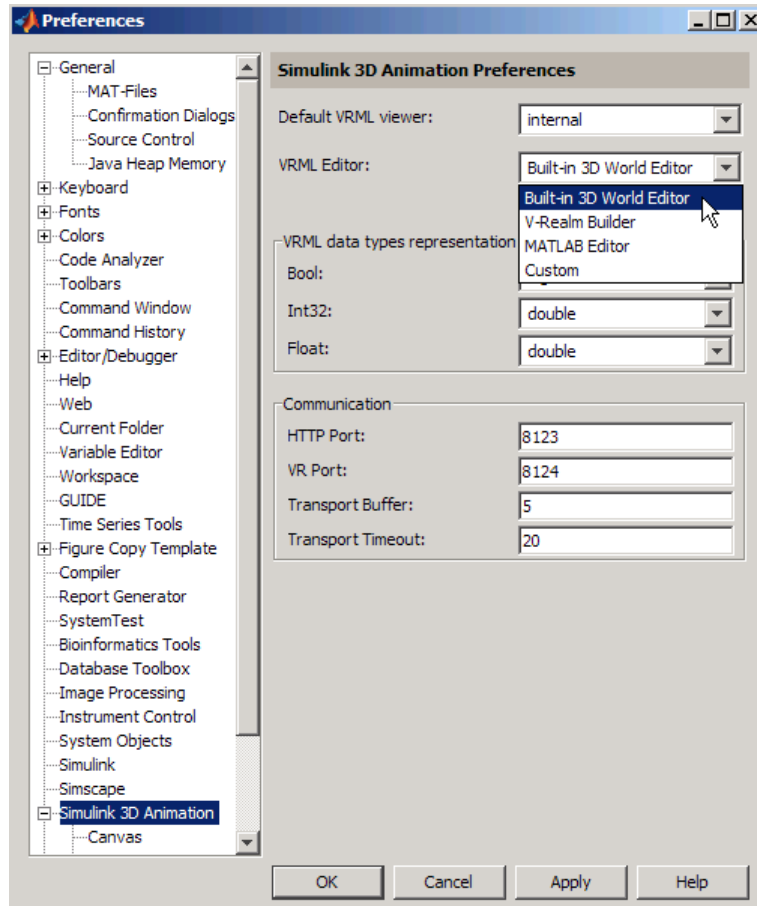
## **Set the Default Editor**

The default VRML editor is the 3D World Editor. You can change your environment to use another editor. You can use the MATLAB Preferences menu or the MATLAB command line.

### **Using the Preferences Menu to Set the Default Editor**

To determine which VRML 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 **VRML Editor** preference.



You can use the **VRML Editor** preference to select another editor: the V-Realm Builder, the MATLAB editor, or a third-party VRML 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.

## Using 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')
```

---

**Tip** The `vredit` command opens the 3D World Editor, regardless of the default editor preference setting.

---

## Set Simulink 3D Animation Preferences

### In this section...

“Section Overview” on page 2-29

“Simulink® 3D Animation™ Preferences” on page 2-30

“3D World Editor Preferences” on page 2-33

“Canvas Preferences” on page 2-34

“Figure Preferences” on page 2-34

“Figure Rendering Preferences” on page 2-35

“Figure 2-D Recording Preferences” on page 2-37

“Figure Frame Capture Preferences” on page 2-38

“Virtual World Preferences” on page 2-39

### Section Overview

The topics in this section describe how to set the Simulink 3D Animation preferences. To access those preferences, from the MATLAB Toolstrip, in the **Home** tab, in the **Environment** section, select **Preferences > Simulink 3D Animation**. The list of settable preferences is a subset of those available through the MATLAB interface functions.

The Simulink 3D Animation software installs with default preference settings. You can change these settings with

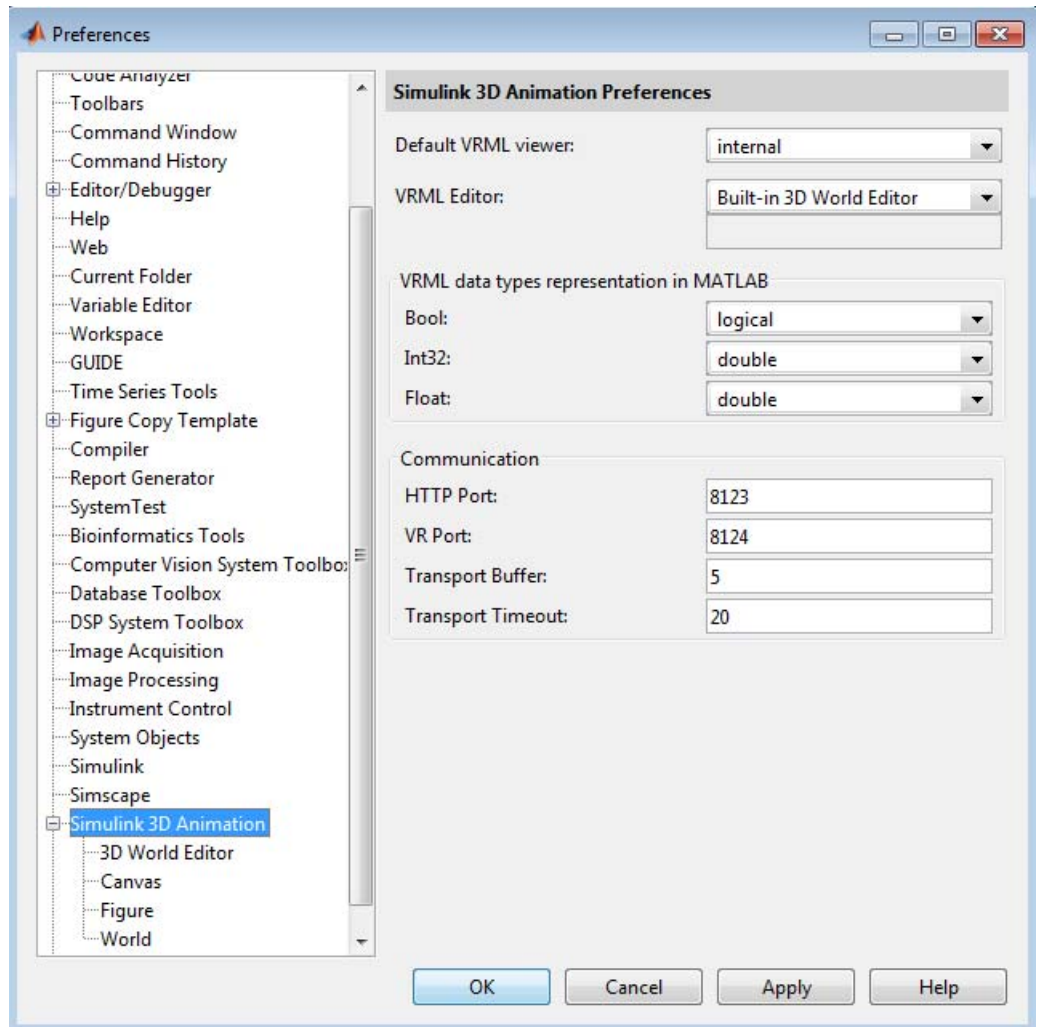
- **MATLAB File > Preferences** dialog box — This GUI has preference dialog boxes for the MATLAB product and its related products, including the Simulink 3D Animation product.
- Simulink 3D Animation MATLAB interface functions

## **Simulink 3D Animation Preferences**

To access the Simulink 3D Animation preferences GUI:

- 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**.

The Simulink 3D Animation Preferences dialog box opens in the right pane.



- 3** Set the preferences that you want. See the following table for the preferences that you can change. Click **OK** to save the settings.

<b>Preference</b>	<b>Value</b>	<b>Description</b>
Bool	'logical'   'char' Default: 'logical'	Specifies the handling of the VRML Bool data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . If set to 'logical', the VRML Bool data type is returned as a logical value. If set to 'char', the Bool data type is returned 'on' or 'off'.
Default VRML Viewer	'internal'   'internalv4'   'internalv5'   '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 'internal' or 'internalv5'. The legacy viewer is used when this preference is set to 'internalv4'. The Web browser is used when this preference is set to 'web'.
Float	'single'   'double' Default: 'double'	Specifies the handling of the VRML float data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . If set to 'single', the VRML Float and Color data types are returned as 'single'. If set to 'double', the Float and Color data types are returned as 'double'.
Int32	'int32'   'double' Default: 'double'	Specifies handling of the VRML Int32 data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . If set to 'int32', the VRML Int32 data type is returned as int32. If set to 'double', the Int32 data type is returned as 'double'.
HTTP Port	Numeric Default: 8123	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.
Transport Buffer	Numeric Default: 5	Length of the transport buffer (network packet overlay) for communication between the Simulink 3D Animation server and its clients.
Transport Timeout	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.



Preference	Value	Description
VRML Editor	Built-in 3D World Editor   V-Realm Builder   MATLAB Editor   Custom	Specifies which VRML editor to use. Path to the VRML editor. If this path is empty, the MATLAB editor is used.  The path setting is active only if you select the Custom option.
VR Port	Numeric Default: 8124	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.

### 3D World Editor Preferences

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

Simulink 3D Animation 3D World Editor Preferences

Position: [143 105 1202 960]

Triad: bottom left

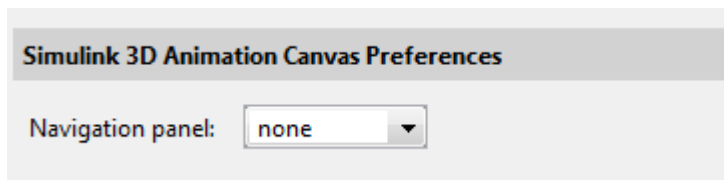
Save position on exit

Property	Value	Description
Position	Two pairs of pixel locations, establishing the upper-left and lower-right corners for the 3D World Editor	Specifies the default location for the 3D World Editor. If you select <b>Save position on exit</b> , then the default position changes to the

Property	Value	Description
	Default: Depends on current screen resolution	position of the 3D World Editor when you exited that editor.
Save position on exit	'off'   'on' Default: 'on'	Causes the 3D World Editor to open in the same location where the editor was when you exited that editor.
Triad	'none'   'top left'   'top right'   'bottom left'   'bottom right'   'center' Default: 'bottom left'	Specifies where in the <b>Tree structure</b> pane to display a triad of coordinate axes.

## Canvas Preferences

The Simulink 3D Animation preferences include a **Navigation panel** preference.



Property	Value	Description
Navigation panel	'none'   'translucent'   'opaque' Default: 'none'	Controls the appearance of the navigation panel in the canvas.

## Figure Preferences

The Simulink 3D Animation figure has a number of preferences:

- “Figure Rendering Preferences” on page 2-35
- “Figure 2-D Recording Preferences” on page 2-37

- “Figure Frame Capture Preferences” on page 2-38

## Figure Rendering Preferences

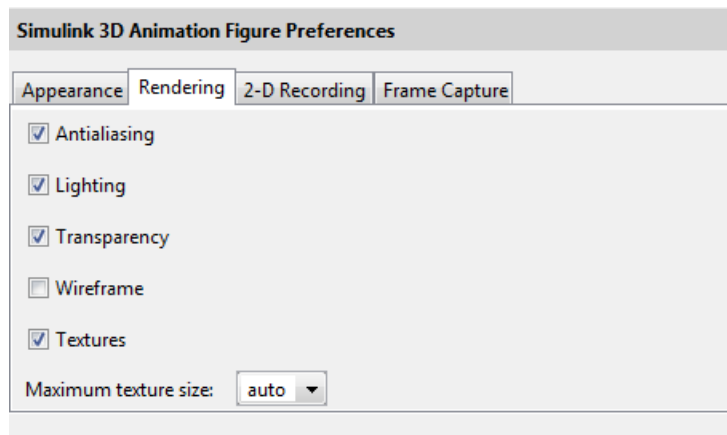
To access the virtual figure rendering preferences:

- 1 From the MATLAB Toolstrip, in the **Home** tab, in the **Environment** section, select **Preferences**.
- 2 In the left pane of the Preferences dialog box, select **Simulink 3D Animation**.
- 3 In the left pane under **Simulink 3D Animation**, select **Figure**.

The Simulink 3D Animation Figure Preferences dialog box opens in the right pane.

- 4 Select the **Rendering** tab.

The Simulink 3D Animation Figure Preferences dialog box opens in the right pane, with the **Rendering** tab selected.



- 5 Set the preferences as desired. See the following table for the rendering preferences you can change. Click **OK** to save the settings.

<b>Property</b>	<b>Value</b>	<b>Description</b>
Antialiasing	'on'   'off' Default: 'on'	Determines whether antialiasing is used when rendering scene. Antialiasing smooths textures by interpolating values between texture points.
Lighting	'off'   'on' Default: 'on'	Specifies whether the lighting is taken into account when rendering. If it is off, all the objects are drawn as if uniformly lit.
Maximum texture size	'auto'   $32 \leq x \leq$ 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 vrfigure 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 possible pixel size. If the value you enter is unsuitable, a warning might trigger. The software then automatically adjusts the property to the next smaller suitable value.
Textures	'off'   'on' Default: 'on'	Turns texture rendering on or off.
Transparency	'off'   'on' Default: 'on'	Specifies whether or not transparency information is taken into account when rendering.
Wireframe	'off'   'on' Default: 'off'	Specifies whether objects are drawn as solids or wireframes.

## Figure 2-D Recording Preferences

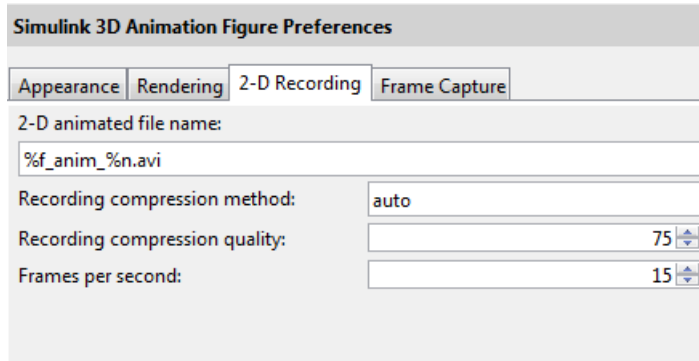
To access the virtual figure 2-D recording preferences:

- 1** From the MATLAB Toolstrip, in the **Home** tab, in the **Environment** section, select **Preferences**.
- 2** In the left pane of the Preferences dialog box, select **Simulink 3D Animation**.
- 3** In the left pane under **Simulink 3D Animation**, select **Figure**.

The Simulink 3D Animation Preferences dialog box opens in the right pane.

- 4** Select the **2-D Recording** tab.

The Simulink 3D Animation Figure Preferences dialog appears in the right pane, with the **2-D Recording** tab selected.



- 5** Set the preferences as desired. See the following table for the rendering preferences you can change. Click **OK** to save the settings.

<b>Property</b>	<b>Value</b>	<b>Description</b>
2-D animated file name	String. Default: '%f_anim_%n.avi'	Specifies the 2-D offline animation file name. The string can contain tokens that are replaced by the corresponding information when the animation recording takes place. For further details, see “Animation Recording File Tokens” on page 4-12.
Recording compression method	' '   '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 function documentation for <code>avifile</code> .
Recording compression quality	Integer 0 – 100. Default: 75	Specifies the default quality of 2-D animation file compression for new <code>vrfigure</code> objects.
Frames per second	Default: 15	Specifies the default frames per second playback speed.

## Figure Frame Capture Preferences

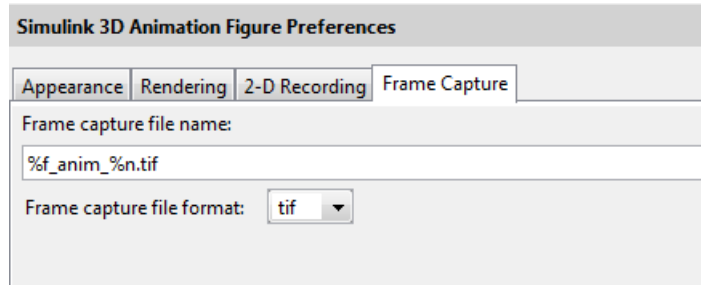
To access the virtual figure frame capture preferences:

- 1** From the MATLAB Toolstrip, in the **Home** tab, in the **Environment** section, select **Preferences**.
- 2** In the left pane of the Preferences dialog box, select **Simulink 3D Animation**.
- 3** In the left pane under **Simulink 3D Animation**, select **Figure**.

The Simulink 3D Animation Figure Preferences dialog box opens in the right pane.

#### 4 Select the **Frame Capture** tab.

The Simulink 3D Animation Figure Preferences dialog appears in the right pane, with the **Frame Capture** tab selected.



#### 5 Set the preferences that you want. See the following table for the rendering preferences that you can change. Click **OK** to save the settings.

Property	Value	Description
CaptureFileFormat	'tif'   'png' Default: 'tif'	Specifies file format for a captured frame file.
CaptureFileName	String. Default: '%f_anim_%n.ext'	Specifies the frame capture file name. The string 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 7-22.

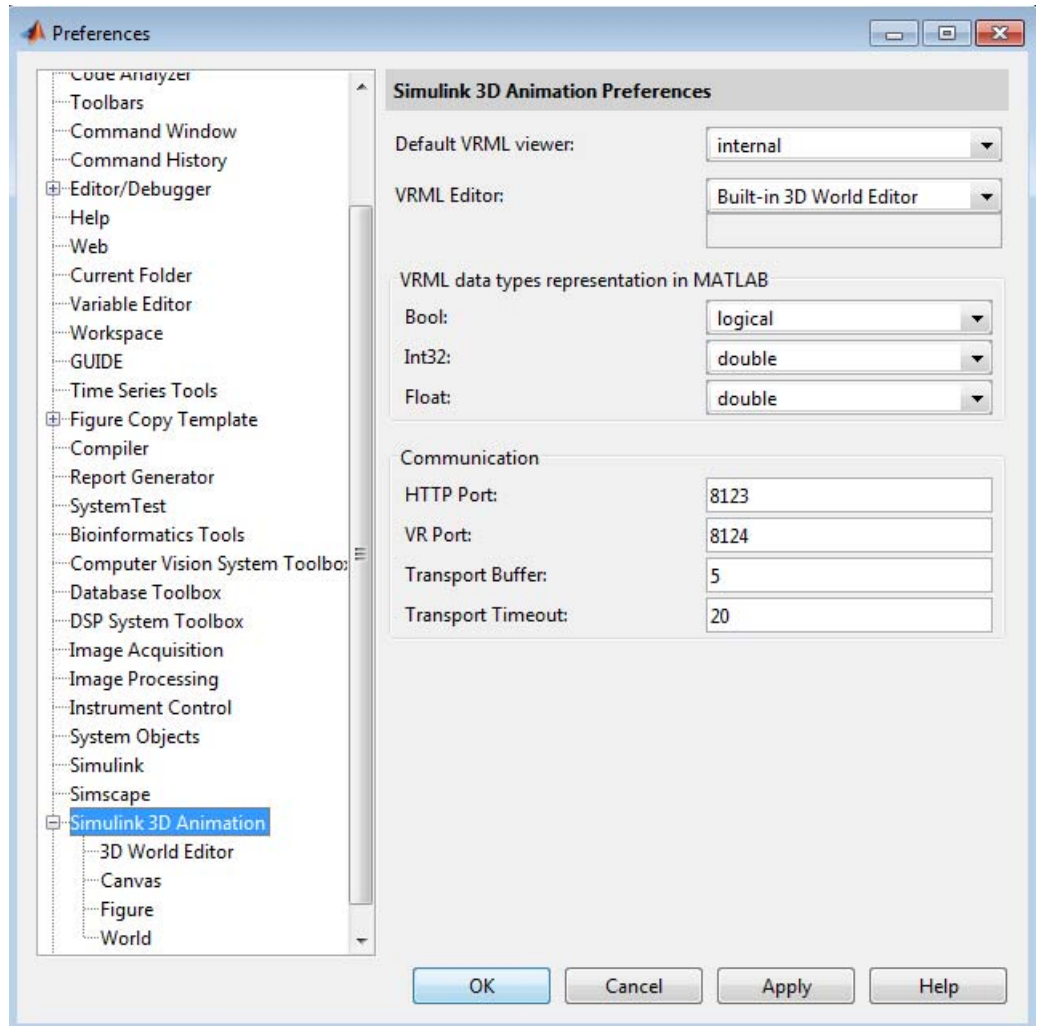
## Virtual World Preferences

To access the virtual world preferences:

- 1 From the MATLAB Toolstrip, in the **Home** tab, in the **Environment** section, select **Preferences**.
- 2 In the left pane of the Preferences dialog box, select **Simulink 3D Animation**.

3 In the left pane under **Simulink 3D Animation**, select **World**.

The Simulink 3D Animation World Preferences dialog box opens in the right pane.





- 4** Set the preferences as desired. See the following table for the rendering preferences you can change. Click **OK** to save the settings.

Property	Value	Description
Allowing viewing from the Internet	'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'.
3-D animated file name	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 “Animation Recording File Tokens” on page 4-12.
Recording mode	'manual'   'scheduled' Default: 'manual'	Animation recording mode.
Recording interval	Vector of two doubles Default: [0 0]	Start and stop times for scheduled animation recording. Corresponds to the virtual world object Time property.
Time source	'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 software (by setting the Time property) or the Simulink software (simulation time). If set to 'freerun', time in the scene advances independently based on the system timer.

## Removing Components (Windows)

In this section...
“Section Overview” on page 2-42
“Uninstall V-Realm Builder” on page 2-42
“Uninstall Blaxxun Contact from Host Computer” on page 2-43

### Section Overview

Normally, you should not have to uninstall the Simulink 3D Animation software, the Blaxxun Contact plug-in, or Ligos V-Realm Builder. If you need to do so, this section explains these procedures.

### 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.

---

**Note** The Blaxxun Contact plug-in is not uninstalled during the Simulink 3D Animation software removal.

---

## **Uninstall Blaxxun Contact from Host Computer**

To uninstall this VRML plug-in from the host computer:

- 1** From the Windows task bar, click **Start**, point to **Settings**, and click **Control Panel**.
- 2** In the **Control Panel** cascading menu, click **Add/Remove Programs**.
- 3** In the Add/Remove Programs dialog box, select blaxxun Contact, then click the **Change/Remove** button.

## Blaxxun Contact Client Computer Installation

In this section...
“Section Overview” on page 2-44
“Installing a VRML Plug-In (Windows)” on page 2-44

### Section Overview

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 very large models that consume considerable computational resources, you might want to use a client computer to run and view the virtual world.

The client computer must have a VRML97 plug-in with External Authoring Interface (EAI) support. This means that your client computer must be a PC platform with the Blaxxun Contact plug-in. Only the Blaxxun Contact software is supported.

“Installing a VRML Plug-In (Windows)” on page 2-16 describes how to install the Blaxxun Contact VRML plug-in on a computer running Windows.

### Installing a VRML Plug-In (Windows)

If you want to view a virtual world on a client computer, you need to use a Web browser with a VRML plug-in.

The Blaxxun Contact plug-in is provided with the Simulink 3D Animation software, but you cannot install the Blaxxun Contact plug-in Version 4.4 on a client computer with the MathWorks installer if you do not have this plug-in installed.

- Copy the file `blaxxuncontact44.exe` from your host computer to the client computer. This file is located at `C:\matlabroot\toolbox\s13d\blaxxun`.

## Testing the Viewer Installation

In this section...
“Section Overview” on page 2-45
“Simulink Testing” on page 2-45
“MATLAB Testing” on page 2-50

### 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 VRML viewer.

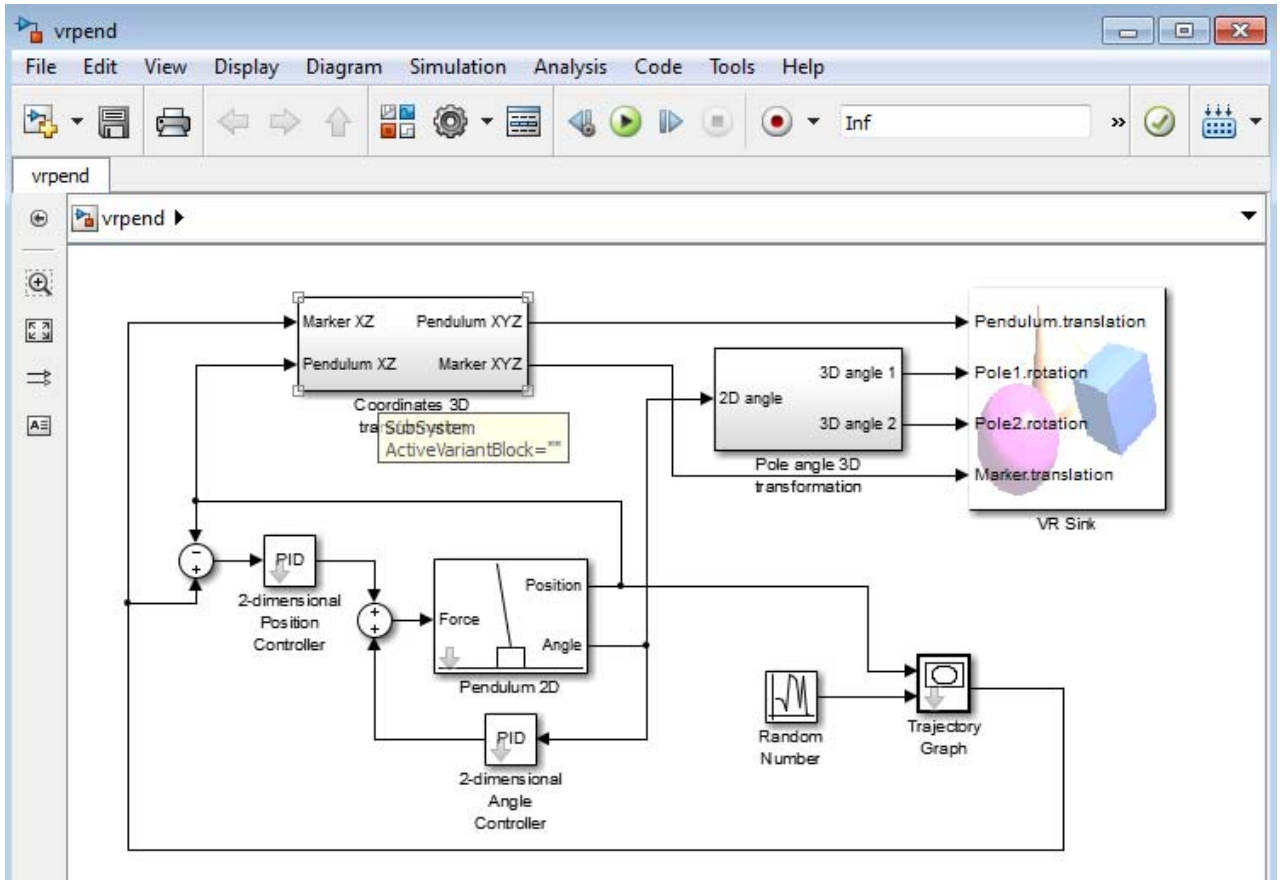
### Simulink Testing

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

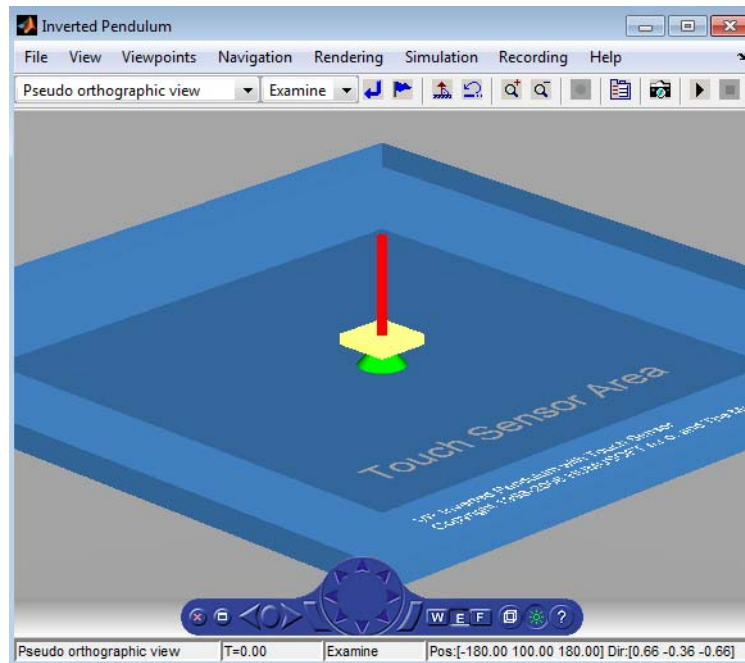
- 1 In the MATLAB Command Window, type

```
vrpend
```

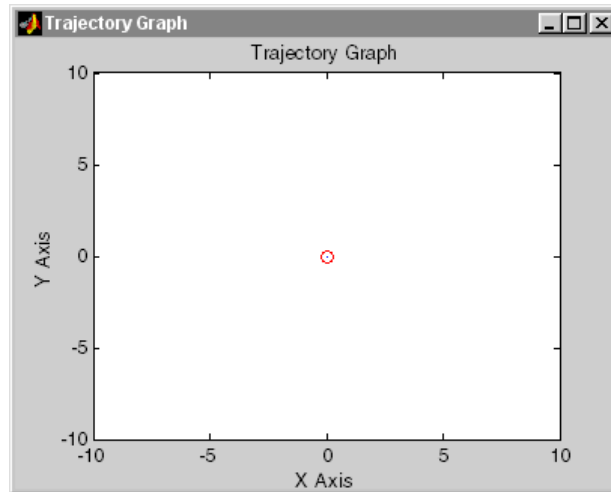
A Simulink window opens with the model for an inverted pendulum. This model, which you can view in three dimensions with the software, has an interactive set point and trajectory graph.



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.

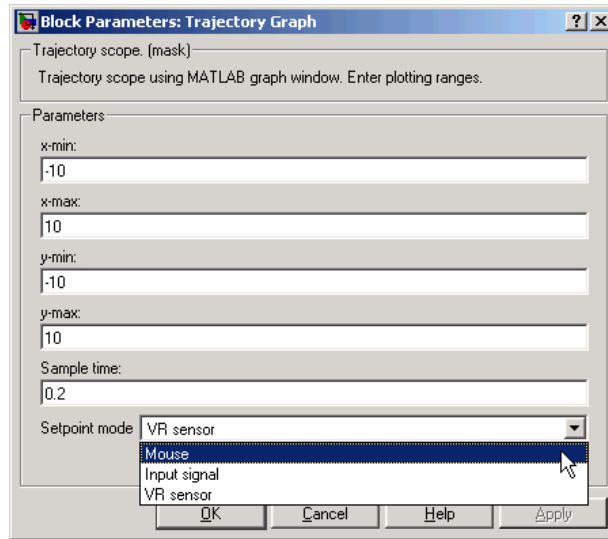
In the Simulink 3D Animation viewer, you see the animated movement of the pendulum. Use the viewer controls to navigate through the virtual world, change the viewpoints, and move the set point. For more information about using the Simulink 3D Animation viewer controls, see “Simulink® 3D Animation™ Viewer” 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 **Setpoint mode** list, choose **Mouse**, then click **OK**.





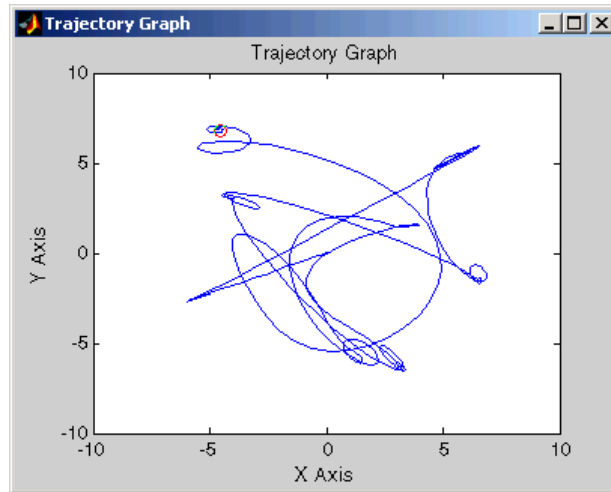
You can now use the trajectory graph as a 2-D input device to set the position of the pendulum.

- 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 1-18, 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 VRML viewer window.

Additional examples are listed in the table “MATLAB Interface Examples” on page 1-31.

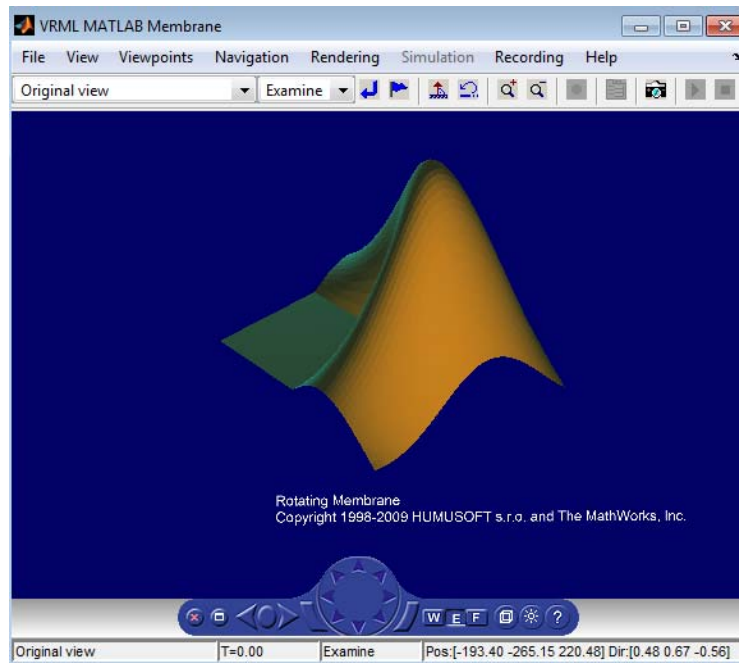
**1** In the MATLAB window, type

```
vrmemb
```

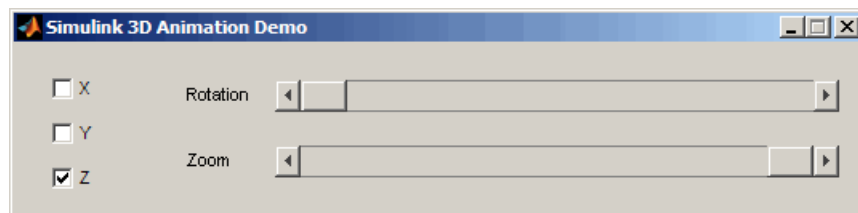
The MATLAB interface displays the following messages:

```
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. Note that sometimes the Simulink 3D Animation example dialog box is hidden behind the viewer window.





# 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 user interface (GUI).

- “Virtual World Connection to a Model” on page 3-2
- “Using the Simulink Interface” on page 3-11
- “Working with VRML Sensors” on page 3-25
- “VR Source Block Input to Simulink Models” on page 3-29
- “Interact with Generated Code” on page 3-30

## Virtual World Connection to a Model

In this section...
“Add a Simulink® 3D Animation™ Block” on page 3-2
“Changing the Virtual World Associated with a Simulink Block” on page 3-8

### Add a Simulink 3D Animation Block

To visualize a dynamic system simulation, connect a Simulink block diagram to a virtual world. The example in this section explains how to display a simulated virtual world on a host computer. This is the recommended way to view associated virtual worlds on the host computer.

Simulating a Simulink model generates signal data for a dynamic system. By connecting the Simulink model to a virtual world, you can use this data to control and animate the virtual world.

After you create a virtual world and a Simulink model, you can connect the two with Simulink 3D Animation blocks. The example in this procedure simulates a plane taking off and lets you view it in a virtual world.

---

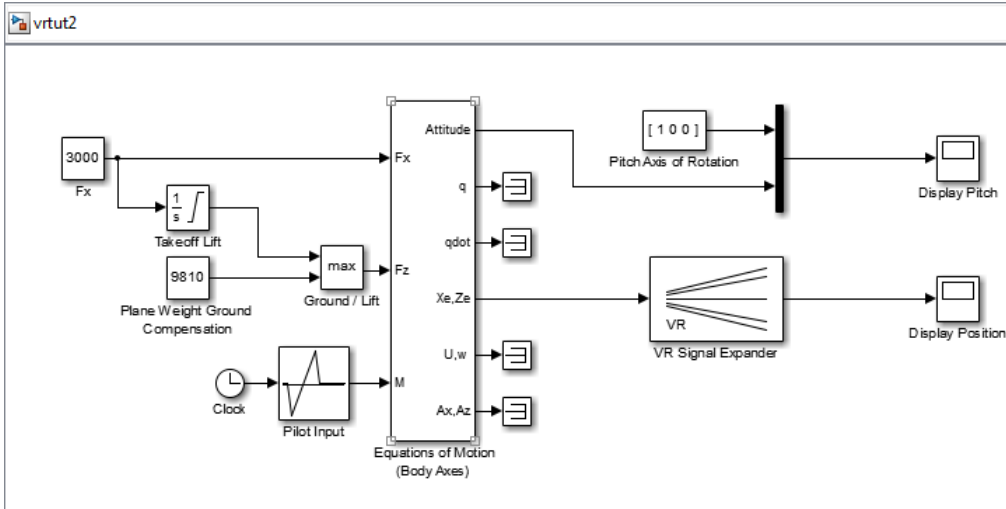
**Note** The examples in this topic are based on the Simulink 3D Animation default viewer. If you choose to use the Blaxxun Contact VRML plug-in to view virtual worlds, you must start and stop the model simulation from the Simulink window. You cannot start and stop the model simulation from the Blaxxun Contact VRML plug-in.

---

**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 From the **Simulation** menu, select **Mode > Normal**, then click **Simulation > Run**.

Observe the results of the simulation in the scope windows.

- 3 In the MATLAB Command Window, type

```
vrlib
```

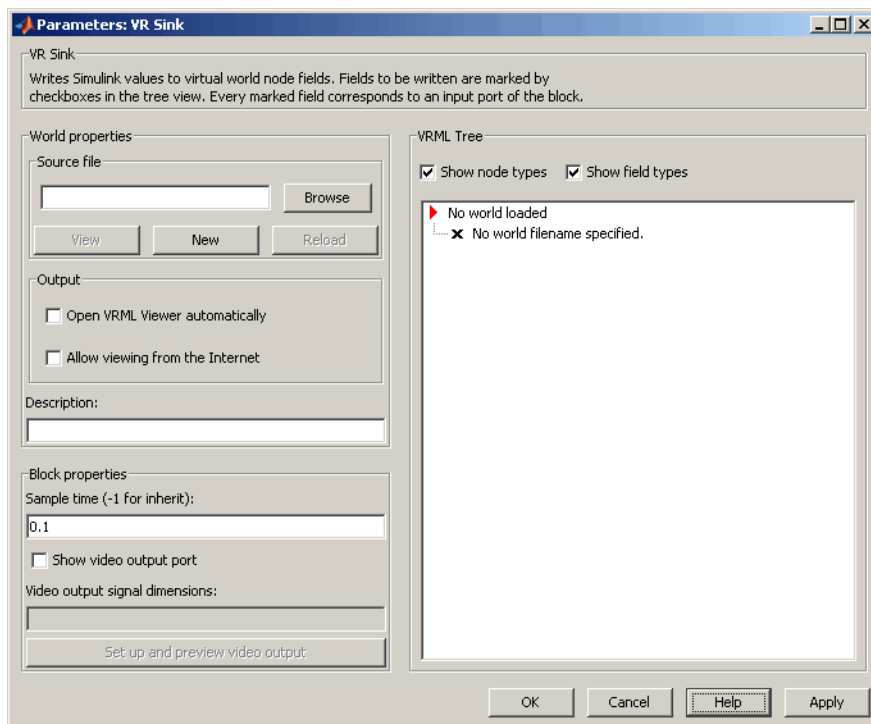
The Simulink 3D Animation library opens.

- 4 From the **Library** window, drag and drop the VR Sink block to the Simulink diagram. The VR Sink block writes data from the Simulink model to the virtual world. You can then close the **Library: vrlib** window.

Now you are ready to select a virtual world for the visualization of your simulation. A simple virtual world with a runway and a plane is in the VRML file `vrtkoff.wrl`, located in the `vrdemos` folder.

- 5 In the Simulink model, double-click the block labeled VR Sink.

The Parameters: VR Sink dialog box opens.



- 6 In the **Description** text box, enter a brief description of the model. This description appears on the list of available worlds served by the Simulink 3D Animation server. For example, type

VR Plane taking off

- 7 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 **Open**.
- 8 Select the **Open VRML Viewer automatically** parameter.
- 9 In the Parameters: VR Sink dialog box, click **Apply**.

A VRML tree appears on the right side, showing the structure of the associated virtual reality scene.

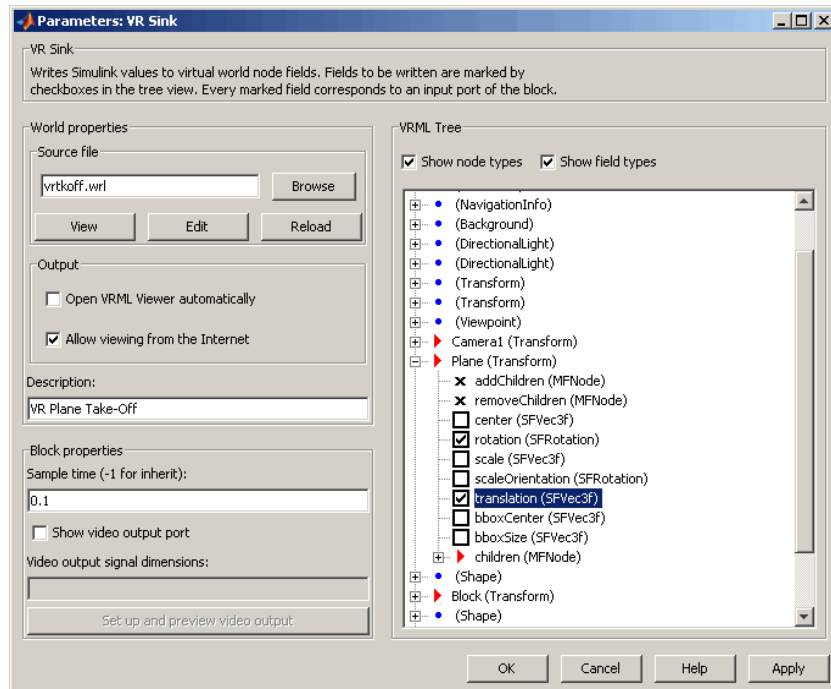


- 10** On the left of the Plane (Transform) node, click the + square.

The Plane Transform tree expands. Now you can see what characteristics of the plane can be driven from the Simulink interface. This model computes the position and the pitch of the plane.

- 11** In the Plane (Transform) tree, select the translation and rotation fields.

The selected fields are marked with checks. These fields represent the position (translation) and the pitch (rotation) of the plane.



- 12** Click **OK**.

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



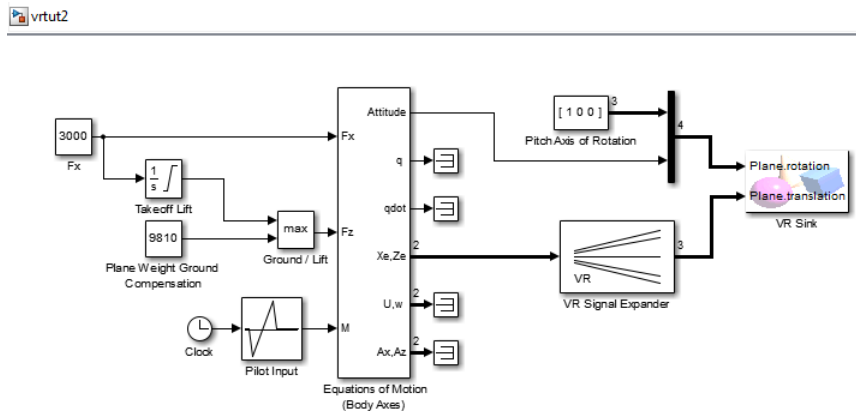
The first input is Plane rotation. The rotation is defined by a four-element vector. The first three numbers define the axis of rotation. In this example, it should be [1 0 0] for the  $x$ -axis (see the Pitch Axis of Rotation block in the model). The pitch of the plane is expressed by the rotation about the  $x$ -axis. The last number is the rotation angle around the  $x$ -axis, in radians.

- 13 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 plane's position 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 (see the VR Signal Expander block). The  $y$ -axis defines the altitude of the plane.

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

After you connect the signals and remove the Scope blocks, your model should look similar to the figure shown.

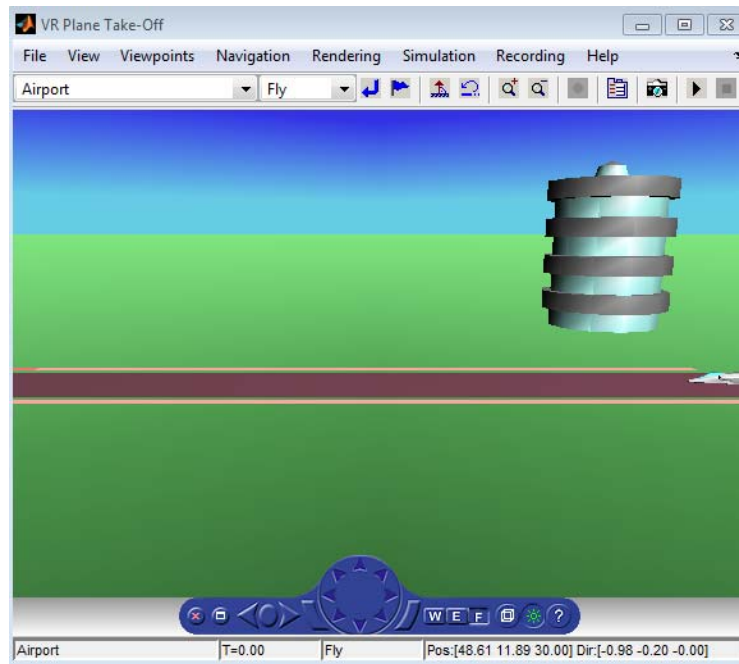


---

**Note** Virtual world degrees of freedom have different requested input vector sizes depending on the associated VRML field types. If the vector size of the connected signal does not match the associated VRML field size, an Incorrect input vector size error is reported when you start the simulation.

---

- 15** Double-click the VR Sink block in the Simulink model. A viewer window containing the plane's virtual world opens.



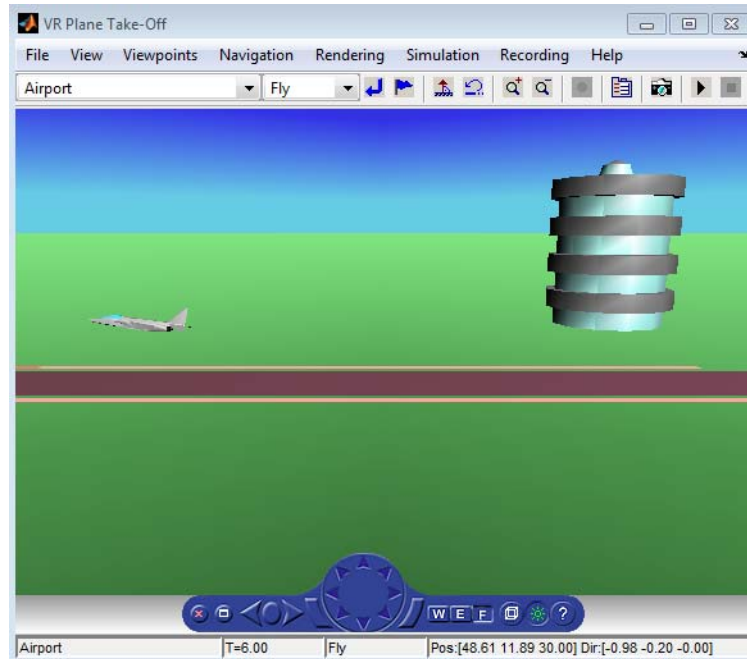
---

**Note** 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.

---

- 16 In the Simulink 3D Animation viewer, from the **Simulation** menu, click **Run** to run the simulation.

A plane, moving right to left, starts down the runway and takes off into the air.



## Changing the Virtual World Associated with a Simulink Block

On occasion, you might want to 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 procedure assumes that you have connected the `vrut2` Simulink model with a virtual world. See “Add a Simulink® 3D Animation™ Block” on page 3-2.

- 1 Double-click the VR Sink block in the model. The viewer opens.

**2** Select the **Simulation** menu **Block Parameters** option. The Parameters: VR Sink dialog box opens.

**3** 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 *vrtkoff2.wrl*, and click **Open**.

**4** In the Parameters: VR Sink dialog box, click **Apply**.

A VRML tree appears on the right side. The Simulink software associates a new virtual world with the model.

**5** On the left of the **Plane (Transform)** node, click the + square.

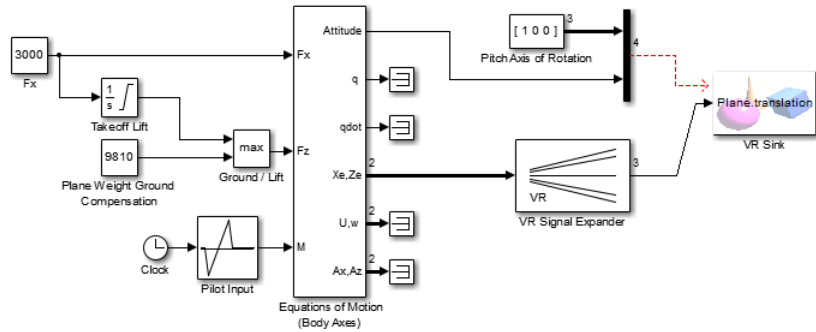
The **Plane Transform** tree expands. Now you can see what characteristics of the plane you can drive from the Simulink interface. This model computes the position.

**6** 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 **Virtual Reality** block is ready to use with the new parameters defined.

**7** Verify that the correct output is connected to your VR Sink block. The output from the **VR Signal Expander** should be connected to the single input.

vrutut2



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

## Using the Simulink Interface

In this section...
“Section Overview” on page 3-11
“Displaying a Virtual World and Starting Simulation” on page 3-11
“View Virtual World on Host Computer” on page 3-14
“View Virtual World Remotely” on page 3-18

### Section Overview

This section shows how to view a virtual world connected to a Simulink block diagram and make parameter changes from the Simulink block or the virtual world.

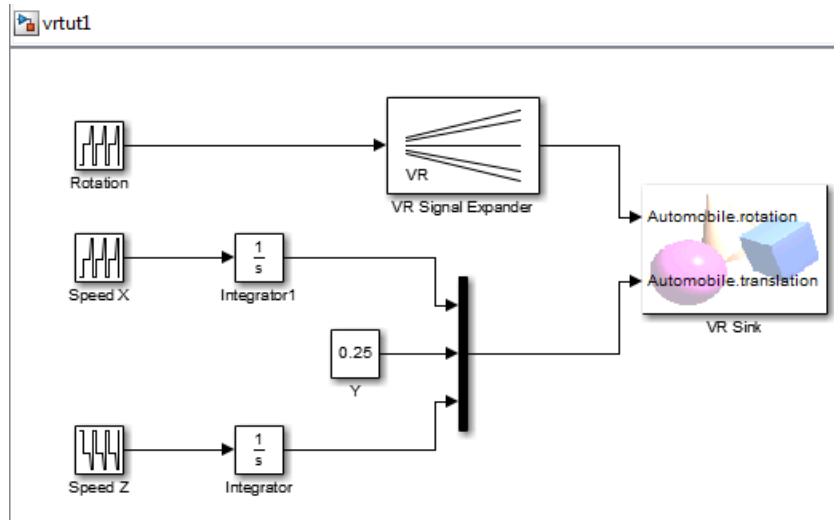
### Displaying a Virtual World and Starting Simulation

This example explains how to display a simulated virtual world using the Simulink 3D Animation viewer on your host computer. This 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

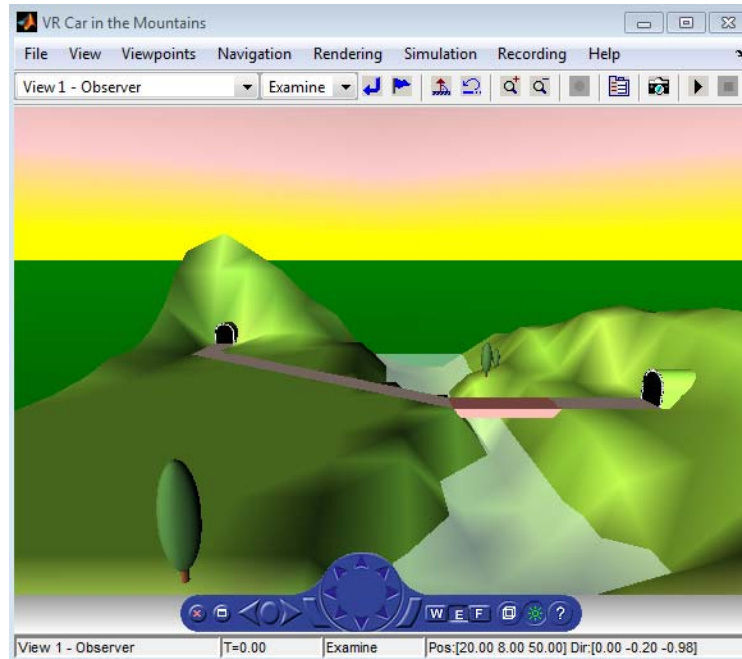
```
vrtut1
```

A Simulink window opens with the model of an automobile.



A VRML 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**.

### Opening a Viewer Window

If you close the viewer window, you might want to reopen it. In the Simulink model window, double-click the 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 2-20.

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 particularly useful if you want to view one scene from many different viewpoints at the same time.

## **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 VRML-enabled Web browser, depending on your `DefaultViewer` setting. For more information on setting your default viewer, see “Set the Default Viewer” on page 2-20.

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. This is the computer on which the Simulink 3D Animation software is currently running. You do not need a VRML-enabled Web browser to display this page.

Note that 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 VRML 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 VRML 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 VRML-enabled Web browser. In the address line of the browser, type

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

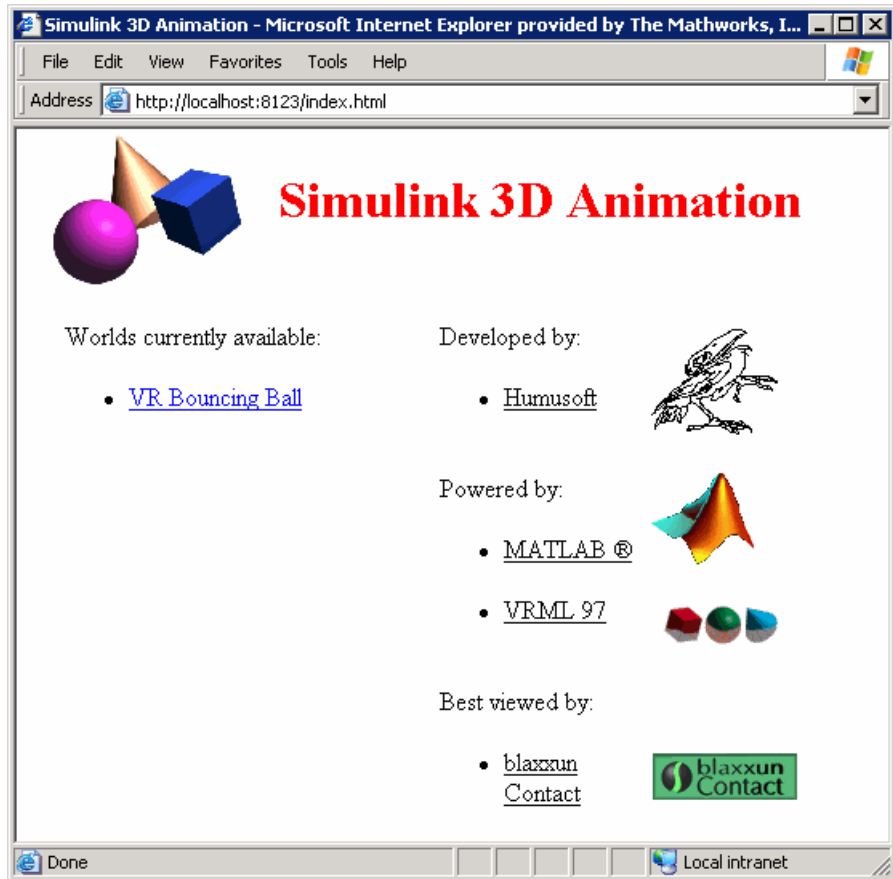
---

**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 currently running.

---

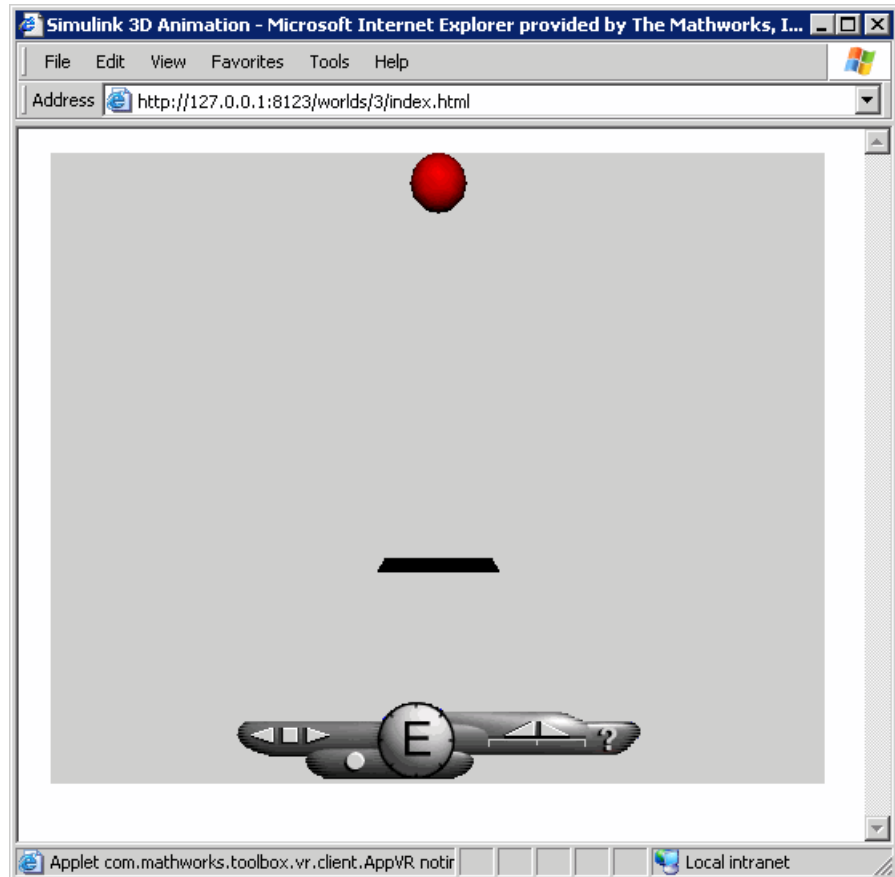
The following page is loaded and becomes active.

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, you can select one of the listed available 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 capabilities from the browser; it is a standard HTML page. Nevertheless, when you click one of the virtual world links in the list, the browser has to be VRML-enabled to display the virtual world correctly and to communicate with the Simulink 3D Animation product.

## **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. This means you need to know the name or IP address of the host computer you want to access from the client computer.

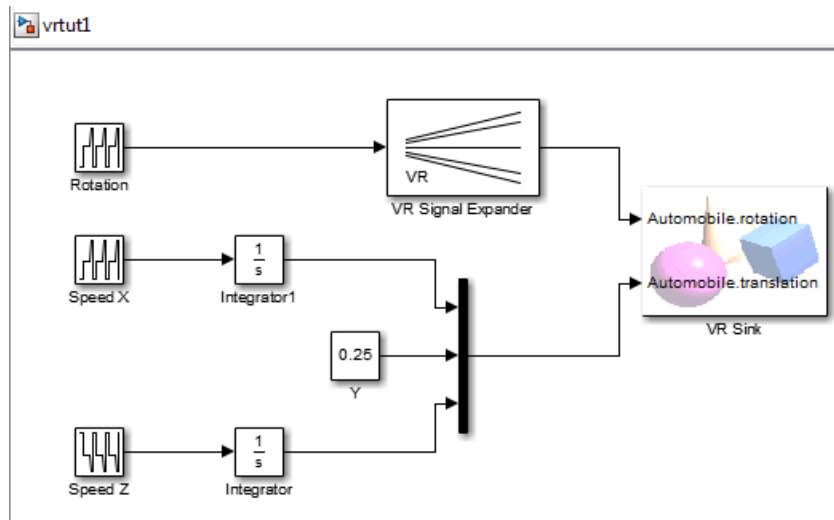
Viewing a virtual world on a client computer might be useful for remote computing, presentation of the results over the Web, or in situations where it is desirable to distribute computing and graphical power.

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 the Blaxxun Contact plug-in. You can also view a virtual world on a client computer by using the Orbisnap viewer (see “View Virtual Worlds Remotely” on page 8-7). In this 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.

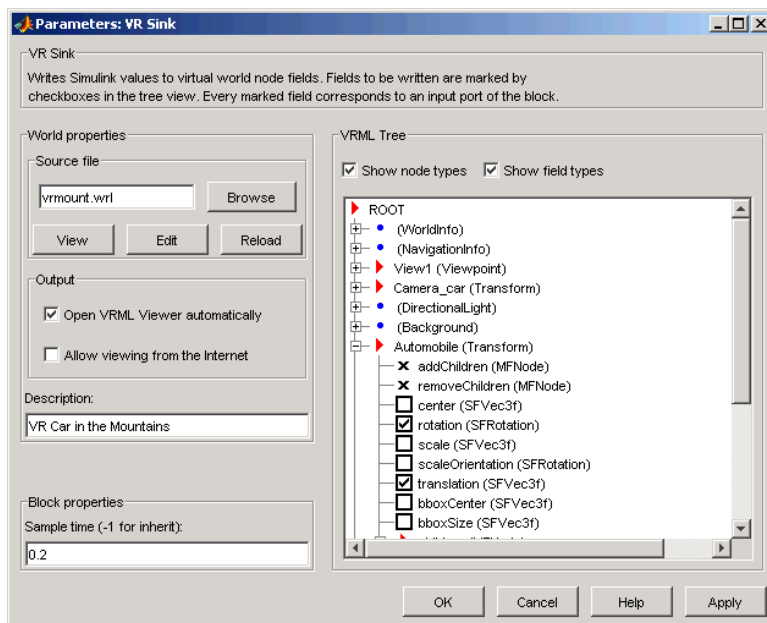


- 2** Double-click the VR Sink block. This block is in the right part of the model window.

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

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

A Parameters: VR Sink 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. You should never select this box when you want your model to be private or confidential.

---

- 5 Click **OK**.
- 6 On the client computer, open your VRML-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`.



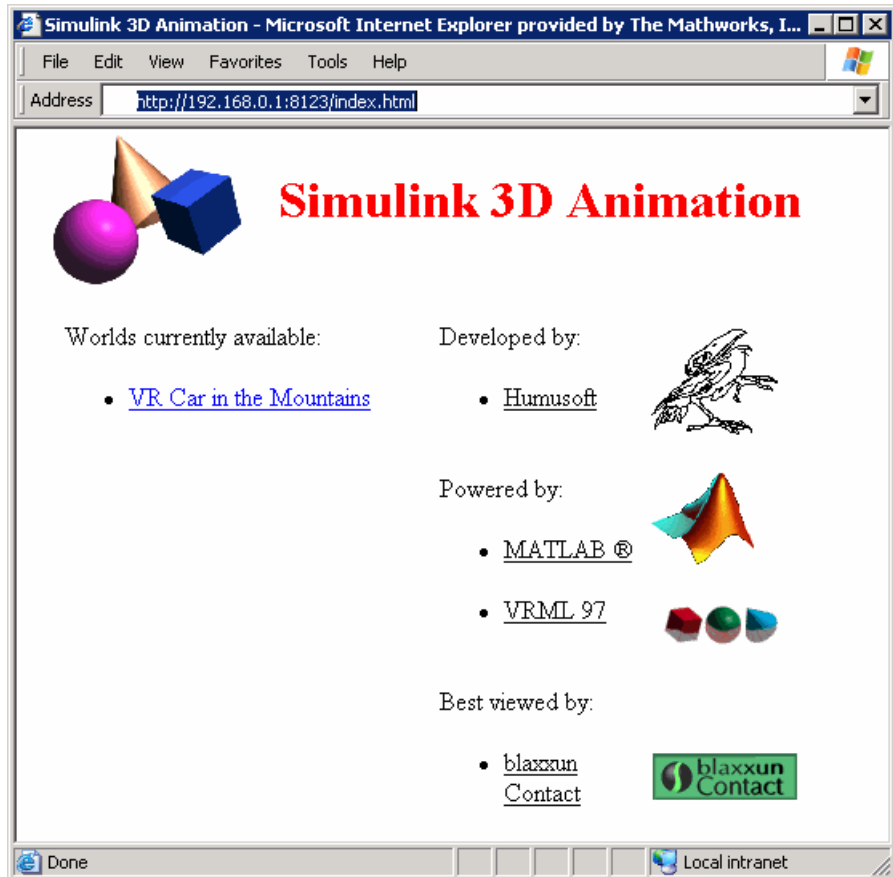
To determine your IP address on a UNIX system, type the command

```
ifconfig device_name
```

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

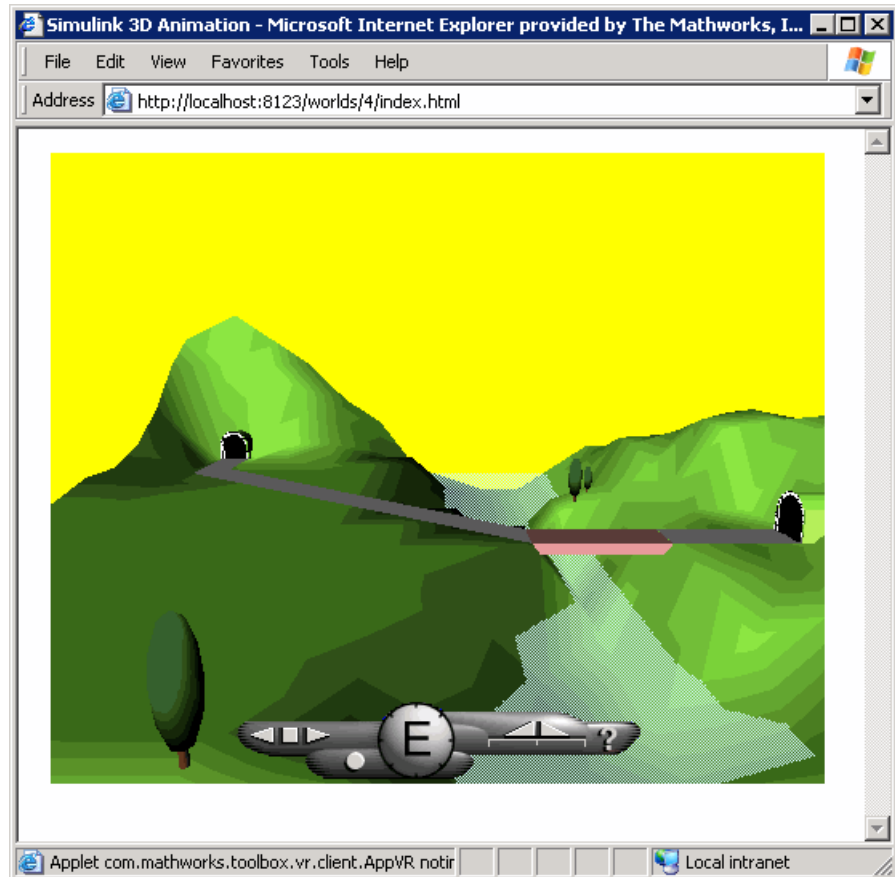
Alternatively, for Windows platforms, you can open a DOS shell and type `ipconfig`.

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.

## Working with VRML Sensors

### In this section...

“Add VRML Sensors to Virtual Worlds” on page 3-25

“Read VRML Sensor Values” on page 3-26

### Add VRML Sensors to Virtual Worlds

This section describes how to interface a Simulink block diagram to sensors in a virtual reality scene. It also describes how to programmatically input signals from the virtual world into a simulation model.

Virtual reality scenes can contain sensors, nodes able to generate events and output values depending on time, user navigation, and actions and distance changes in the scene. These nodes add interactivity to the virtual world. 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.

You can define the following VRML sensors in the scene:

Sensors	Description
CylinderSensor	Maps pointer motion (for example, a mouse or wand) 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.

Sensors	Description
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.

## Read VRML Sensor Values

To read a value of a readable VRML 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 the dot notation.

## Reading VRML Sensor Values Example

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` with the following:

```
% create the vrworld object and open the world
wh = vrworld('maglev.wrl');
open(wh);

% get the node handle
nh = vrnode(wh, 'Grab_Sensor');

% synchronize the translation field
sync(nh, 'translation', 'on');

% 3 alternative ways to read the synchronized field value
setpoint = getfield(nh, 'translation');
```

```
setpoint = nh.translation;
setpoint = wh.Grab_Sensor.translation;
```

To use the setpoint value in a Simulink model, you can write an S-function or an MATLAB Function block that reads the sensor output periodically. For an example of such 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 vrexin block, which is an S-function block. The vrexin S-function synchronizes the sensor field in the setup method and periodically reads its value in the mdlUpdate method.

- 2** To examine the S-function parameters, right-click vrexin and select **S-Function Parameters**.

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

Note the following about the vrexin S-function:

- Instead of setting its own block outputs, the vrexin 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 (grabbing the ball to a desired position using a mouse) closes through the associated Simulink model vrmaglev.mdl. As a result, grabbing the ball 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 and remove the comment symbol (#) to enable the last line of this file. This action activates direct routing of sensor output to a ball translation. You can then 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 method to input information from all VRML node fields of the type `exposedField` or `eventOut`, not only a `Sensor eventOut` field. See `VRML Data Class Types` for more information about VRML 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 above `Grab_Sensor` node.



## VR Source Block Input to Simulink Models

The VR Source reads values from virtual world fields specified in the Block Parameters dialog box and inputs their values.

Use the VR Source block to provide interactivity between a user navigating the virtual world and the Simulink model. The VR Source block registers user interactions with the virtual world and passes to the model values that can affect the simulation of the model. 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 object's changed location. 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 VRML file).

For an example of how to use the VR Sink block, see Magnetic Levitation Model.

### **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 Real-Time Windows Target environment. To do so, use the Simulink External mode.

For details about the Real-Time Windows Target environment, see the

# 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.

- “Using the MATLAB Interface” on page 4-2
- “Recording Offline Animations” on page 4-10

## Using the MATLAB Interface

In this section...
“Create a vrworld Object” on page 4-2
“Open a Virtual World” on page 4-3
“Interact with a Virtual World” on page 4-5
“Close and Delete a vrworld Object” on page 4-9

### Create a vrworld Object

To connect MATLAB to a virtual world and to interact with that virtual world through the MATLAB command-line interface, you need to create `vrworld` and `vrnode` objects. A virtual world is defined by a VRML file with the extension `.wrl`.

---

**Note** The Simulink interface and the MATLAB interface share the same virtual world objects. This 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” on page 4-3.

## Open a Virtual World

Opening a virtual world lets you view the virtual world in a VRML 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 is the description that is displayed 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. This is typically the Simulink 3D Animation viewer unless you have a different viewer set.

Alternatively, you can display the virtual world in a VRML-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`. For more information on the Simulink 3D Animation HTML page, see “View Virtual World on Host Computer” on page 3-14.

- 3** If the Web browser has the VRML plug-in installed, in the browser window, click **My first virtual world**.

- 4 Your default VRML-enabled Web browser displays the virtual world `vrmount.wrl`.

---

**Note** If your Web browser is not VRML-enabled, clicking on a virtual world link such as **My first virtual world** results in a broken link message. The browser cannot display the virtual world. If you get such a message, ensure that the Web browser is properly enabled for VRML with the Blaxxun Contact plug-in. For details, see “Blaxxun Contact Host Computer Installation” on page 2-15.

---

For more information on changing your default viewer, see “Set the Default Viewer” on page 2-20.

## Interact with a Virtual World

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 VRML 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:

- 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 like the following table.

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 VRML 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, and you can observe that the car is repositioned in the VRML 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.

---

## Close and Delete a vrworld Object

After you are finished with a session, you must 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.

---

## Recording Offline Animations

### In this section...

“Animation Recording” on page 4-10

“Manual and Scheduled Animation Recording” on page 4-11

“Animation Recording File Tokens” on page 4-12

“Manual 3-D VRML Animation Recording” on page 4-14

“Manual 2-D AVI Animation Recording” on page 4-16

“Scheduled 3-D VRML Animation Recording” on page 4-20

“Scheduled 2-D AVI Animation Recording” on page 4-22

“Viewing Animation Files” on page 4-25

“Animation Recording of Unconnected Virtual Worlds” on page 4-27

### Animation Recording

The Simulink 3D Animation software enables you to record animations of virtual scenes that are controlled by the Simulink or MATLAB product. 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 might want to 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. The Simulink 3D Animation viewer is the recommended interface to record animations. If you are working with virtual scenes controlled from MATLAB, you can still record virtual scenes through the MATLAB interface.

---

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

- 3-D VRML file — The Simulink 3D Animation software traces object movements and saves that data into a VRML 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 VRML animation files, be sure to also distribute all the inlined object and texture files referenced in the original VRML 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 might 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 VRML Animation Recording” on page 4-14
- “Manual 2-D AVI Animation Recording” on page 4-16
- “Scheduled 3-D VRML Animation Recording” on page 4-20
- “Scheduled 2-D AVI Animation Recording” on page 4-22

## Animation Recording File Tokens

By default, the Simulink 3D Animation software records animations in a file named according to the following format:

```
%f_anim_%n.<extension>
```

This format creates a unique filename each time you record the animation. The Simulink 3D Animation software places the file in the current folder. %f and %n are tokens, where %f is replaced with the name of the virtual world associated with the model and %n is a number that is incremented each time you record an animation for the same virtual world. If you do not change the default filename, for example, if the name of the virtual world file is vrplanets and you record the simulation for the first time, the animation file is:

```
vrplanets_anim_1.wrl
```

If you run and record the simulation a second time, the animation filename is vrplanets\_anim\_2.wrl.

Create multiple file names with time or date stamps, with a unique file created at each run.

You can use a number of tokens to customize the automated generation of animation files. This section describes how to use these tokens to create varying animation filenames. The following tokens are the same for .wrl and .avi files.

Token	Description
%d	The full path to the world VRML file replaces this token in the filename string and creates files in directories relative to the virtual world file location. For example, the format %d/animdir/animfile.avi saves the animation into the animdir subfolder of the folder containing the virtual world VRML file. This token is most helpful if you want to ensure that the virtual world file and animation file are in the same folder.
%D	The current day in the month replaces this token in the filename string. For example, the format %f_anim_%D.wrl saves the

Token	Description
	animation to vrplanets_anim_29.wrl for the 29th day of the month.
%f	The virtual world filename replaces this token in the filename string and creates files whose root names are the same as those of the virtual world. For example, the format %f_anim_%D.wrl saves the animation to vrplanets_anim_29.wrl. This token might be useful if you use different virtual worlds for one model.
%h	The current hour replaces this token in the filename string. For example, the format %f_anim_%h.wrl saves the animation to vrplanets_anim_14.wrl for any time between 14:00 and 15:00.
%m	The current minute replaces this token in the filename string. For example, the format %f_anim_%h%m.wrl saves the animation to vrplanets_anim_1434.wrl for a start record time of 14:34.
%M	The current month replaces this token in the filename string. For example, the format %f_anim_%M.wrl saves the animation to vrplanets_anim_4.wrl for a start record time in April.
%n	The current incremental number replaces this token in the filename string and creates rolling numbered filenames such that subsequent runs of the model simulation create incrementally numbered filenames. This feature allows you to run a Simulink model multiple times but create a unique file at each run. For example, the format %f_anim_%n.wrl saves the animation to vrplanets_anim_1.wrl on the first run, vrplanets_anim_2.wrl on the second run, and so forth. This token is useful if you expect to create files of different parts of the model simulation.
%s	The current second replaces this token in the filename 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.
%Y	The current four-digit year replaces this token in the filename string. For example, the format %f_anim_%Y.wrl saves the animation to vrplanets_anim_2005.wrl for the year 2005.

## Manual 3-D VRML Animation Recording

This topic describes how to manually record a 3-D animation 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 VRML animation filename 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, you must 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 VRML 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 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 manually stop the recording 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.

## Manual 2-D AVI Animation Recording

This topic describes how to manually record a 2-D animation 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 filename 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, you must 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 might want to 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 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 manually stop the simulation. 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.

## Scheduled 3-D VRML Animation Recording

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 VRML animation filename 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, you must 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 VRML 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. Note that 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 and 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.

## **Scheduled 2-D AVI Animation Recording**

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 filename 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, you must 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. Note that the recording can be slow.

- 7** Disable the Navigation Panel. The Navigation Panel appears at the bottom of the virtual scene view. You might want to 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 and 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.

## Viewing Animation Files

This topic assumes that you have a VRML or .avi animation file that you want to view. If you do not have an animation file, see “Manual 3-D VRML Animation Recording” on page 4-14 or “Scheduled 3-D VRML Animation Recording” on page 4-20 for descriptions of how to create one.

## Viewing VRML Files

You can view VRML animation files using one of these approaches:

- Open the Simulink 3D Animation player from the MATLAB Toolstrip.
- From the operating system, locate and double-click the VRML animation file
- At the MATLAB command line, use the `vrplay` command or `vrview` command.
- In the **Current Folder** pane of MATLAB, double-click the VRML animation file and from the context menu, select **Run**.

To open the Simulink 3D Animation player from the MATLAB Toolstrip, in the **Apps** tab, click the down arrow on the right side of the tab. Scroll down to the **Simulation Graphics and Reporting** area and click the **3D Animation Player** button. Select or specify an a VRML animation file.

A second option is to double-click the VRML file. A VRML-enabled Web browser opens with the animation running. To view the resulting animation file, you must have a VRML-enabled Web browser installed on your system. Also, ensure that the .wrl extension is associated with the Blaxxun Contact Web browser.

A third option is to use the `vrplay` command, where `filename` is the name of your VRML file. This opens the Simulink 3D Animation player and your

file. Using the Simulink 3D Animation player, you can control the playback of your file.

As an 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 VRML recording functionality.

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);
```

A fifth option is to right-click the animation file in the **Current Folder** pane of MATLAB. From the context menu, select **Run**.

### **View AVI 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.

## Animation Recording of Unconnected Virtual Worlds

This topic describes how to programmatically record animation files 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 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. You should be an advanced Simulink 3D Animation user.

- 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 also 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 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. Note that for a smoother animation, you should set the time at equal intervals, for example, every 5 seconds. Use a sequence like

```
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 VRML Animation Recording” 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. You should 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 VRML nodes 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.



# Build Virtual Reality Worlds

---

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

- “VRML Editors” on page 5-2
- “Build and Connect a Virtual World” on page 5-7
- “VRML Data Types” on page 5-24
- “Simulink® 3D Animation™ Textures” on page 5-29
- “Using CAD Models with the Simulink® 3D Animation™ Product” on page 5-30

## VRML Editors

### Editors for Virtual Worlds

There is more than one way to create a virtual world defined with VRML code. For example, you can use a text editor to write VRML code directly, or you can use a VRML editor to create a virtual world without knowing anything about the VRML language. However, you must understand the structure of a VRML scene to connect your virtual world to Simulink blocks and signals.

For a description of the tools to view virtual worlds, see “View Dynamic System Simulations”.

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.

Many people prefer to create simple virtual worlds using MATLAB Editor or other text editor. For more information, see “Text Editors” on page 5-3.

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 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.

The Simulink 3D Animation software includes the 3D World Editor, which you can use on all supported platforms for Simulink 3D Animation. The 3D World Editor is the default VRML editor for Simulink 3D Animation. For details about specifying a VRML editor, see “Set the Default Viewer” on page 2-20.

For Windows platforms, you can also use Ligos V-Realm Builder software to create and edit VRML code. For information on using V-Realm Builder software with the Simulink 3D Animation product, see “Ligos V-Realm Builder” on page 5-6.

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

- “Text Editors” on page 5-3

- “General 3-D Editors” on page 5-4
- “Native VRML Editors” on page 5-4
- “3D World Editor” on page 5-4

## Text Editors

A VRML file uses a standard text format that you can read with any text editor. Reading the VRML code in a text editor is useful for debugging and for directly changing VRML code, as well as for automated processing of the code. If you use the correct VRML 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 VRML virtual world when you want to:

- Create a very simple virtual world.
- Debug syntax and formatting errors in a VRML file. Corrupted VRML files do not open in most 3-D tools.
- Learn about VRML syntax by using VRML syntax highlighting in the MATLAB Editor. For details, see “VRML Syntax Highlighting in the MATLAB Editor” on page 5-3.
- Perform global search VRML editing operations across one or more VRML files.
- Combine several VRML models. Combining models can involve temporary model inconsistencies, which most 3-D tools cannot handle.

## VRML Syntax Highlighting in the MATLAB Editor

You can display VRML syntax highlighting in the MATLAB Editor.

To change MATLAB Editor properties for VRML 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.

3 Change the highlighting properties that you want.

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

General 3-D editors, such as 3D Studio, SolidWorks®, or Autodesk® Maya, do not use VRML as their native format. They export their formats to VRML. 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 very powerful, expensive, and complex to learn, but others are relatively inexpensive and might satisfy your specific needs.

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

### **Native VRML Editors**

Native VRML editors use VRML as their native format. All the features in the editor are compatible with VRML. Also, native VRML editors support features that are unique to the VRML format, such as interpolators and sensors.

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

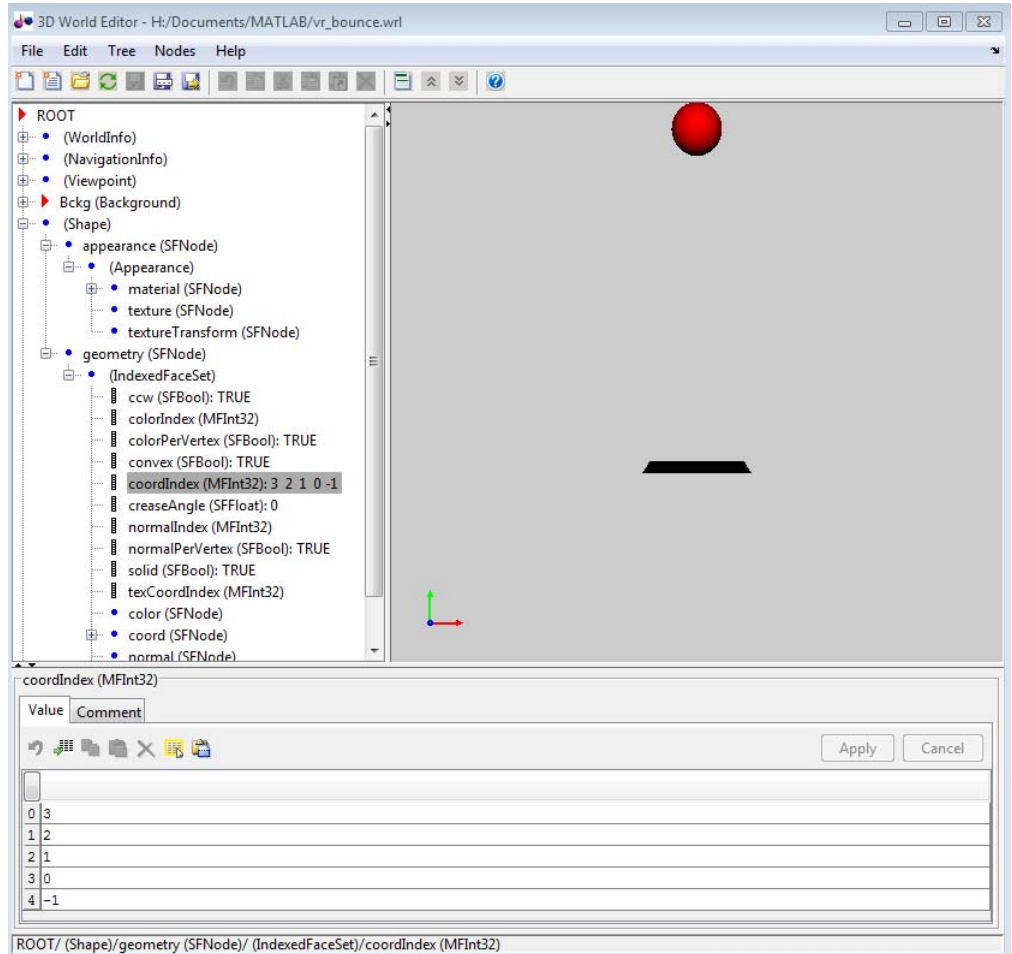
- “3D World Editor” on page 6-2, which works on all platforms supported for Simulink 3D Animation product
- The “Ligos V-Realm Builder” on page 5-6, which works on Windows platforms only

### **3D World Editor**

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

The 3D World Editor is a native VRML authoring tool that provides an interface to the VRML syntax. The editor supports all VRML97 types and language elements. Its primary file format is VRML97.

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 VRML 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 5-7 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:

- 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 2-26.

## Build and Connect a Virtual World

### In this section...

“Introduction” on page 5-7

“Define the Problem” on page 5-7

“Add a Simulink® 3D Animation™ Block” on page 5-9

“Open a New Virtual World” on page 5-11

“Add VRML Nodes” on page 5-11

“Link to a Simulink Model” on page 5-20

### Introduction

The example in this section 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, but it 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 Viewer” on page 2-20.

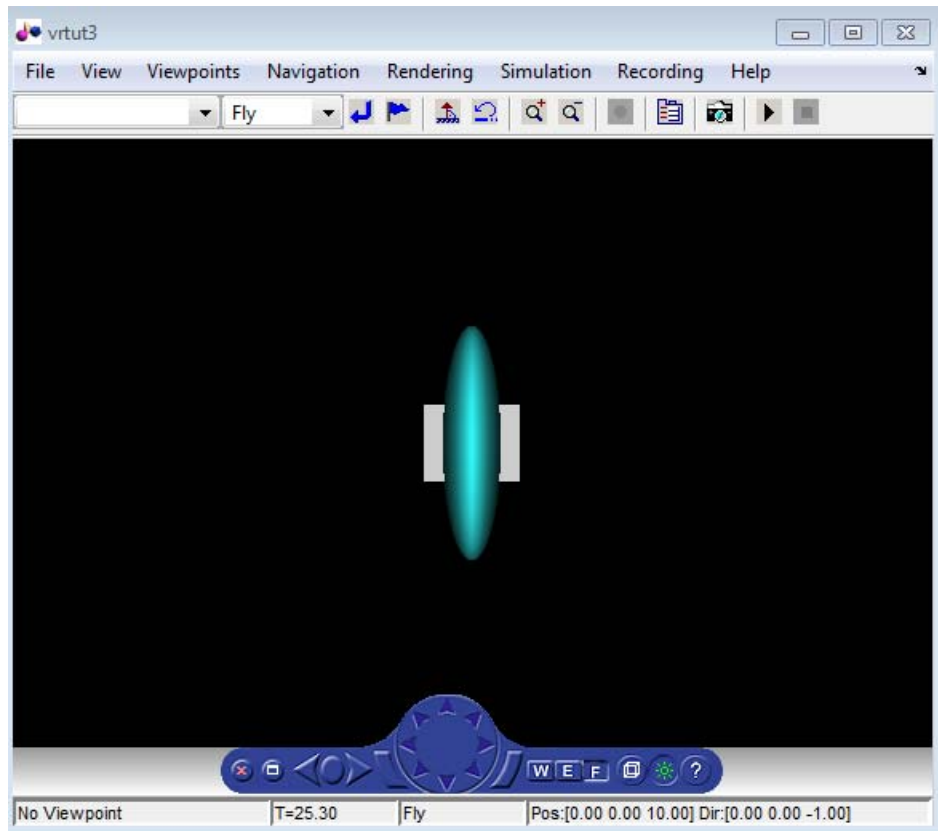
This example describes the steps to build a slightly simplified version of the virtual world that you see if you enter the following command in the MATLAB command window:

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

### Define the Problem

Suppose 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.slx`. This is a simplified model in which the deformation of an elastic sphere is simulated. After collision with the rigid blocks, the sphere's  $x$  dimension is



decreased by a factor from 1 to 0.4, and the  $y$  and  $z$  dimensions are expanded so that the volume of the deformed sphere-ellipsoid remains 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

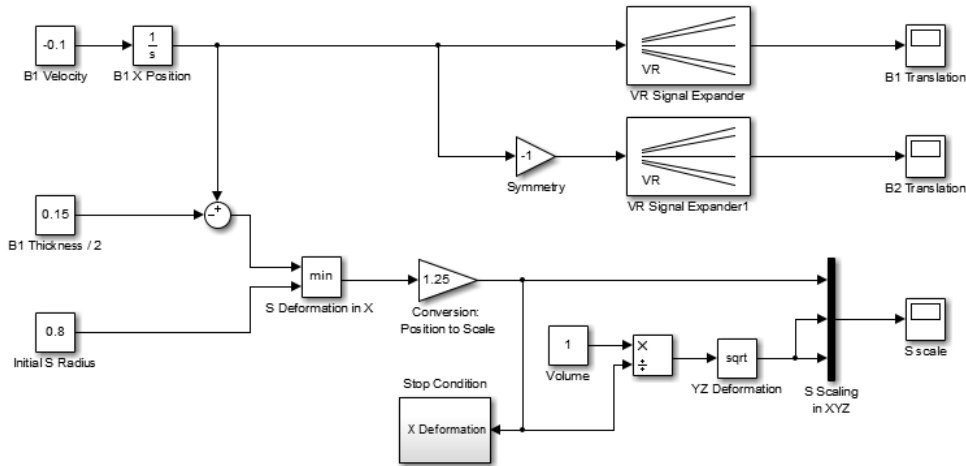
This procedure uses the Simulink model `vrtut3.slx` to show how to add a Simulink 3D Animation block to your model. The model generates the values for the position of B1, the position of B2, and the dimensions of S (as described in “Define the Problem” on page 5-7).

- 1** Open the Tutorial #3. example.
  - a** At the top of the page that opens, select **Open this model**.
  - b** Save the `vrtut3.slx` file to your MATLAB working folder.
- 2** Start MATLAB, and then 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 temporarily monitor the relevant signals.



**4** From the MATLAB Command Window, type

```
vrllib
```

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: vrllib 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 5-11.

## 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.slx`. 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 by entering the following at the MATLAB command line:

```
vredit
```

The 3D World Editor displays:

- In the left pane, a VRML 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 `vrtut3.wrl` in the same working folder where your `vrtut3.slx` file resides. Do not close the 3D World Editor.

Your next two tasks create a virtual world to use with the `vrtut3.slx` model:

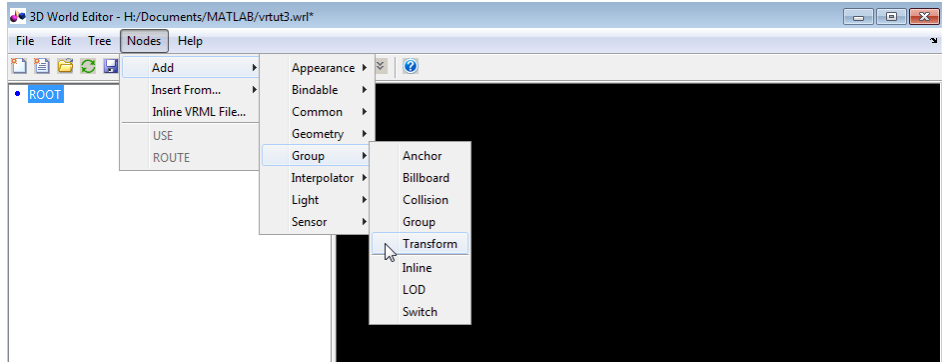
- “Add VRML Nodes” on page 5-11
- “Build and Connect a Virtual World” on page 5-7

## Add VRML Nodes

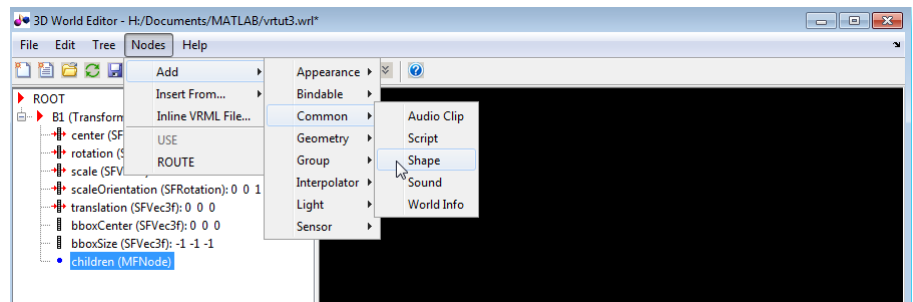
### Create Boxes

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.

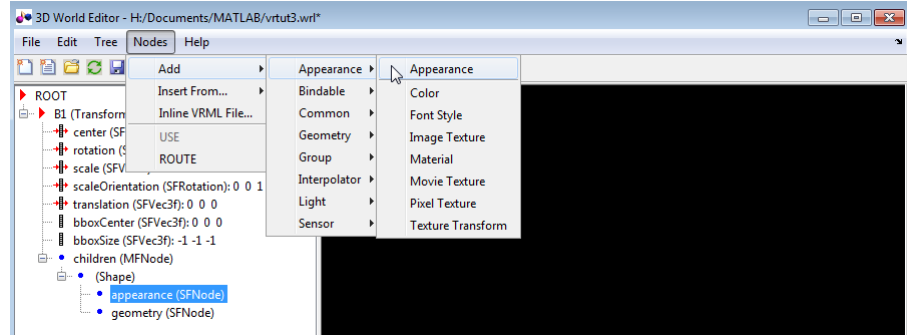


- 3 This Transform node is for the B1 box. To name the Transform node:
  - a Right-click the Transform node.
  - b Select the **Edit Name** menu item.
  - c In the edit box to the left of the Transform node, type B1 .
- 4 Add a Shape node:
  - a Expand the B1 Transform node.
  - b Select the children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



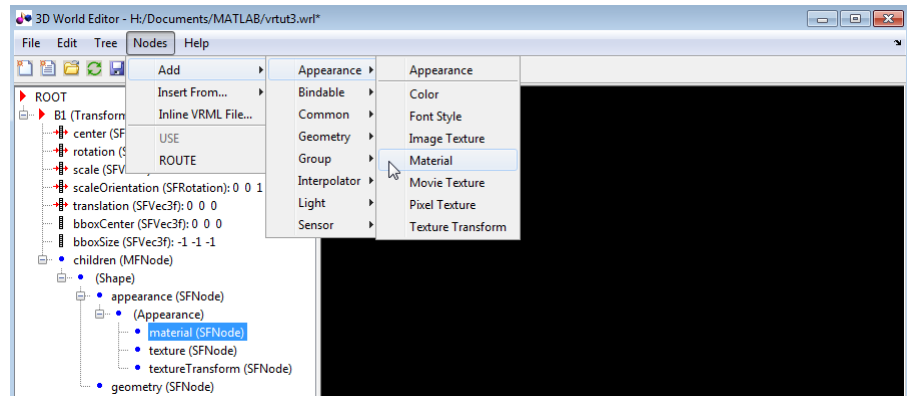
- 5 Add an Appearance node for the Shape node:

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



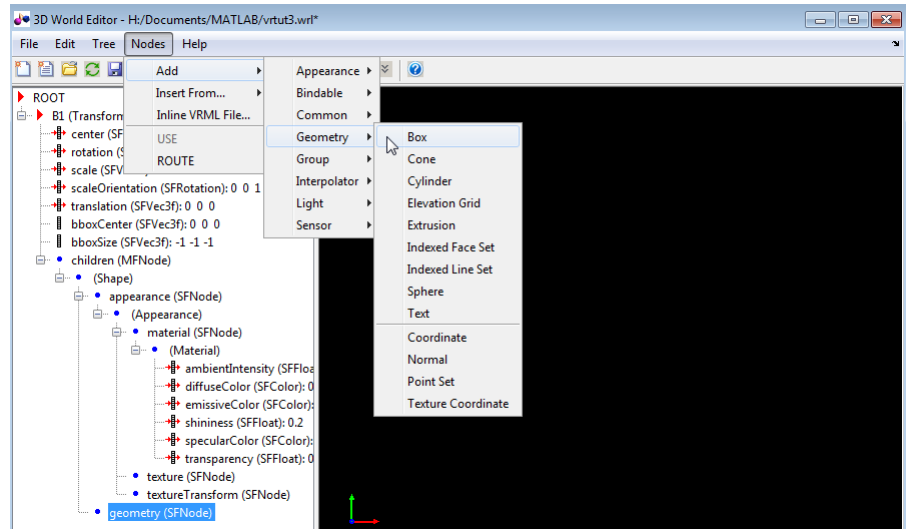
**6** Add a Material node to the Appearance node:

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

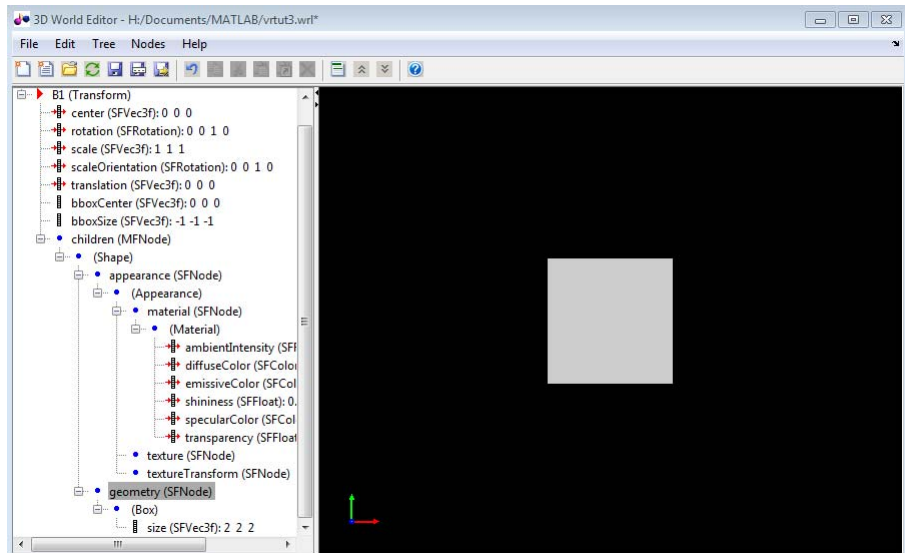


**7** Add a Box node to the geometry node:

- a Select the geometry (SFNode) node of the (Shape) node.
- b Add a Box node, using the following sequence of menu selections.



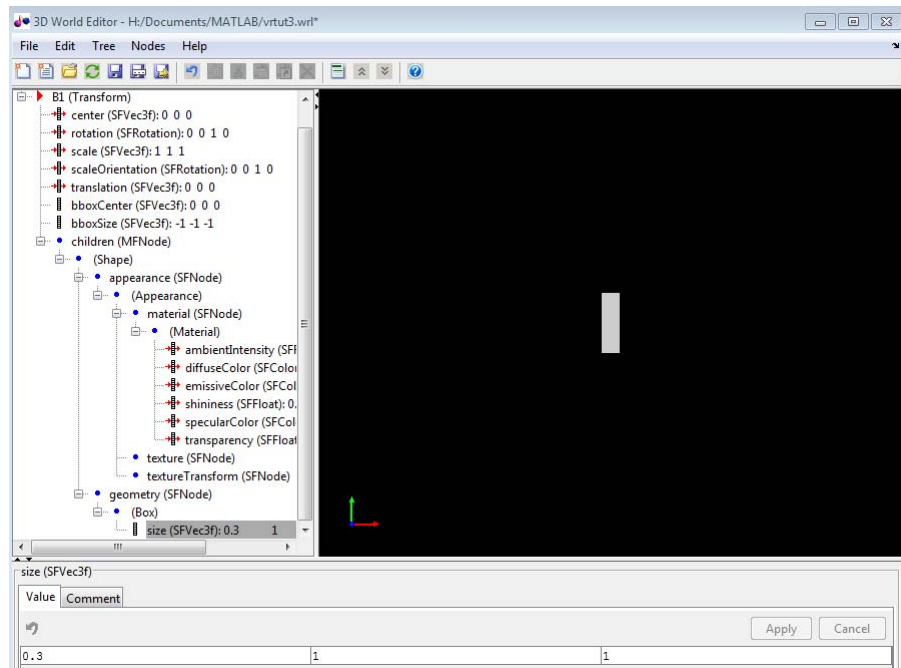
With all the nodes expanded, the 3D World Editor now displays a box in the **virtual world display** pane.



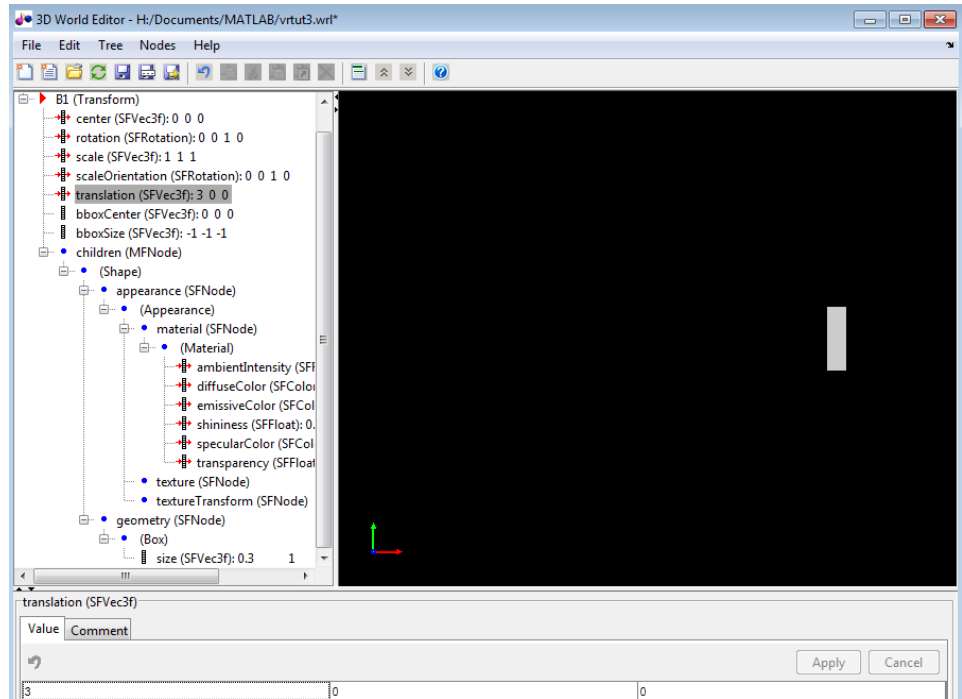
**8** Make the box smaller by editing its size property:

- a Select the size property of the Box node.
- b 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.
- c 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.

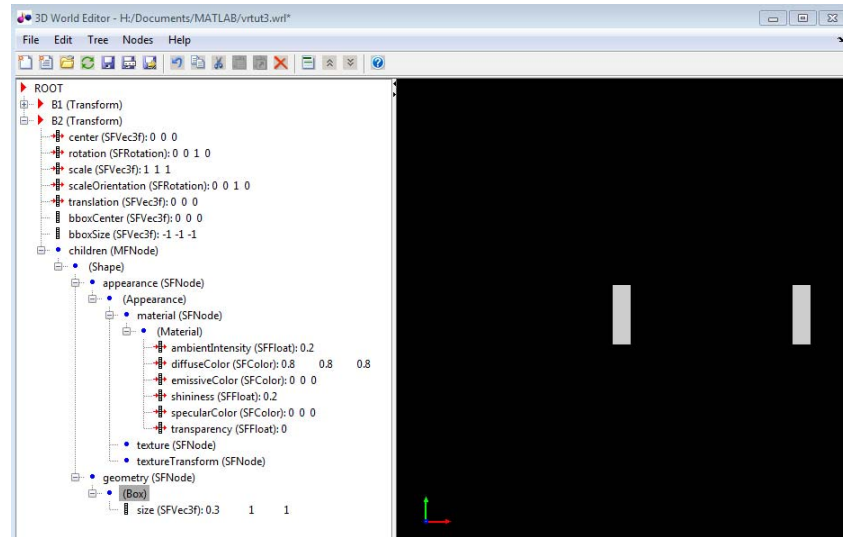


**10** Add a second box that is very similar to the first box.

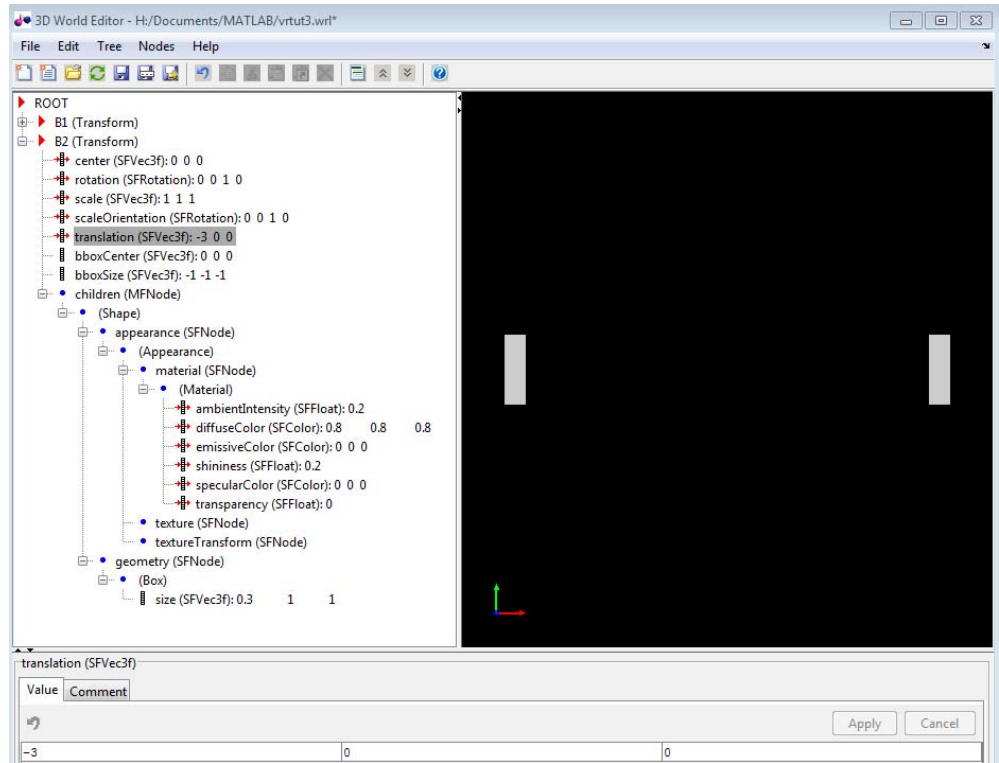
- a** Under the ROOT node, add a Transform node (see step 2) and name it B2 (see step 3).
- b** Copy the Shape node. Under the B1 Transform node, right-click the Shape node in the B1 Transform node and select the **Copy** menu item.
- c** Paste the copied Shape node into the B2 Transform node. Under 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 just 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.

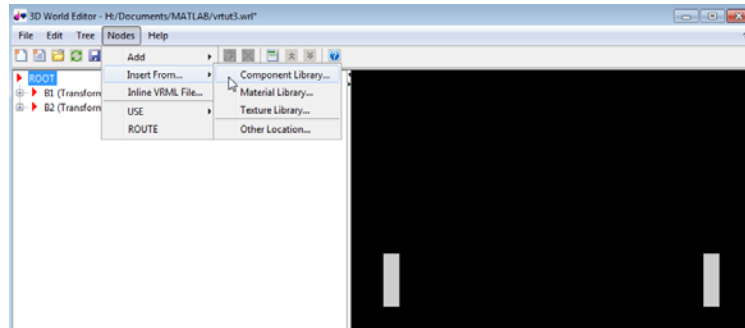


### Create a Sphere

Your next task is to add a sphere between the two boxes. This section assumes you have completed the tasks described in “Add VRML 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 VRML objects for building a virtual world, including a Sphere object.

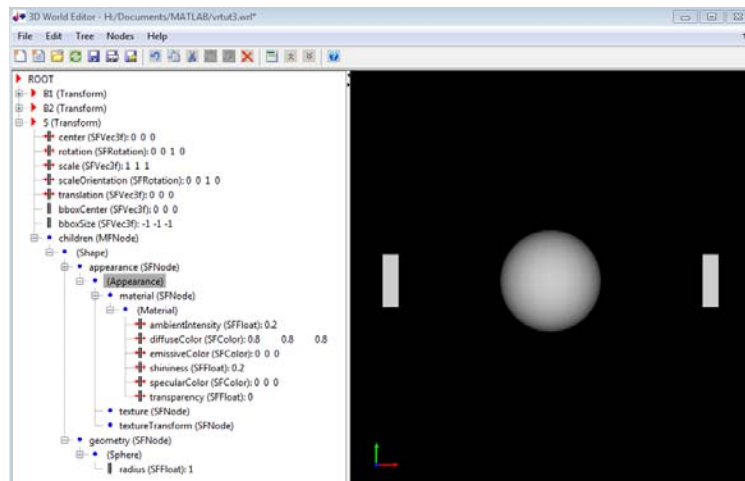
Add a Sphere library object using the following sequence of menu selections.



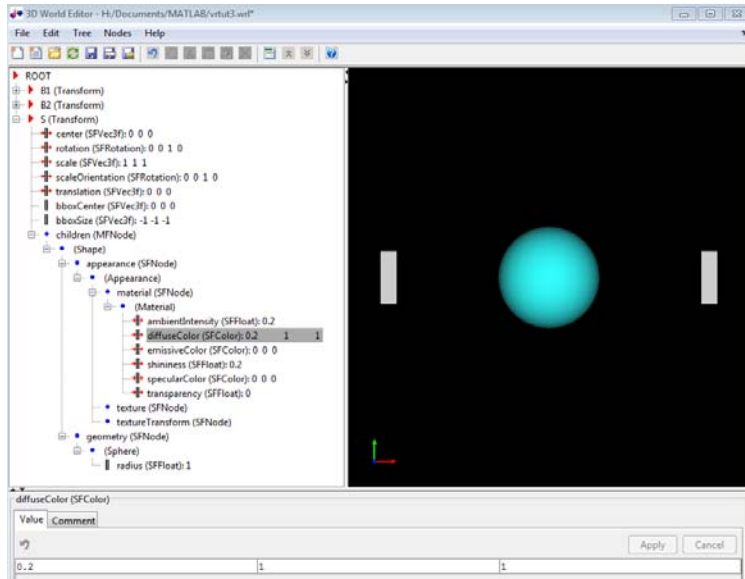
From the list of Component Library folders, select the Shapes folder, and then select the Sphere.wr1 file.

- 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 5-20.

### Link to a Simulink Model

After you create a virtual world and a Simulink model, and add a VR Sink block to your model, you can define the associations between the model signals and the virtual world. This procedure uses the model `virtut3.slx` 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 `virtut3.wrl`. See the tutorial starting with “Add a Simulink® 3D Animation™ Block” on page 5-9.

1 In the Simulink model window, double-click the VR Sink block.

The Parameters: VR Sink dialog box opens.

2 Next to the **Source file** edit box, click **Browse**.

The Open dialog box opens.

**3** Select `vrtut3.wr1`, and then click **Open**.

**4** In the **Output** pane, select the **Open VRML viewer automatically** check box.

This check box specifies that a viewer for the virtual world starts when you run the model.

**5** In the **Description** field, type `vrtut3`.

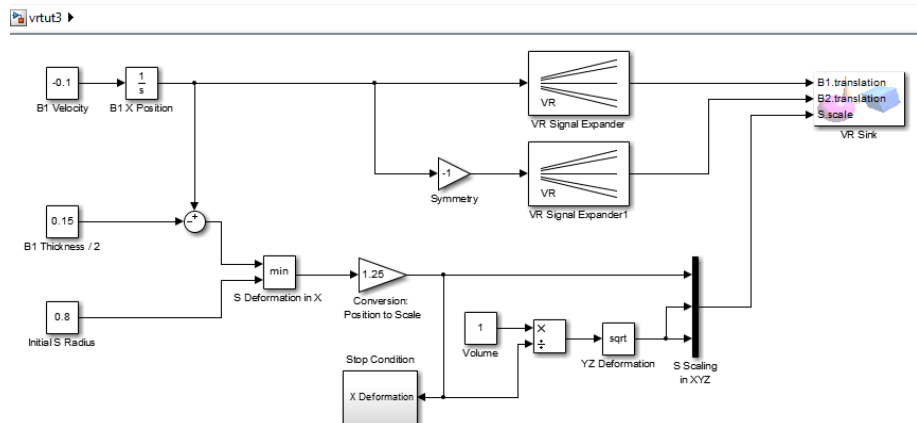
**6** Click **Apply** in the Parameters: VR Sink dialog box.

**7** In the **tree structure** pane of the dialog box, select the **B1 translation**, **B2 translation**, and **S scale** check boxes as the nodes you want to connect to your model signals. Click **OK** to close the dialog box.

The VR Sink block appears with corresponding inputs.

**8** Delete the three Scope blocks (B1 Translation, B2 Translation, and S scale) and their associated input signal lines.

**9** Connect the input lines from the VR Signal Expander, VR Signal Expander1, and S Scaling in XYZ blocks to the appropriate ports in the VR Sink block, as shown below.



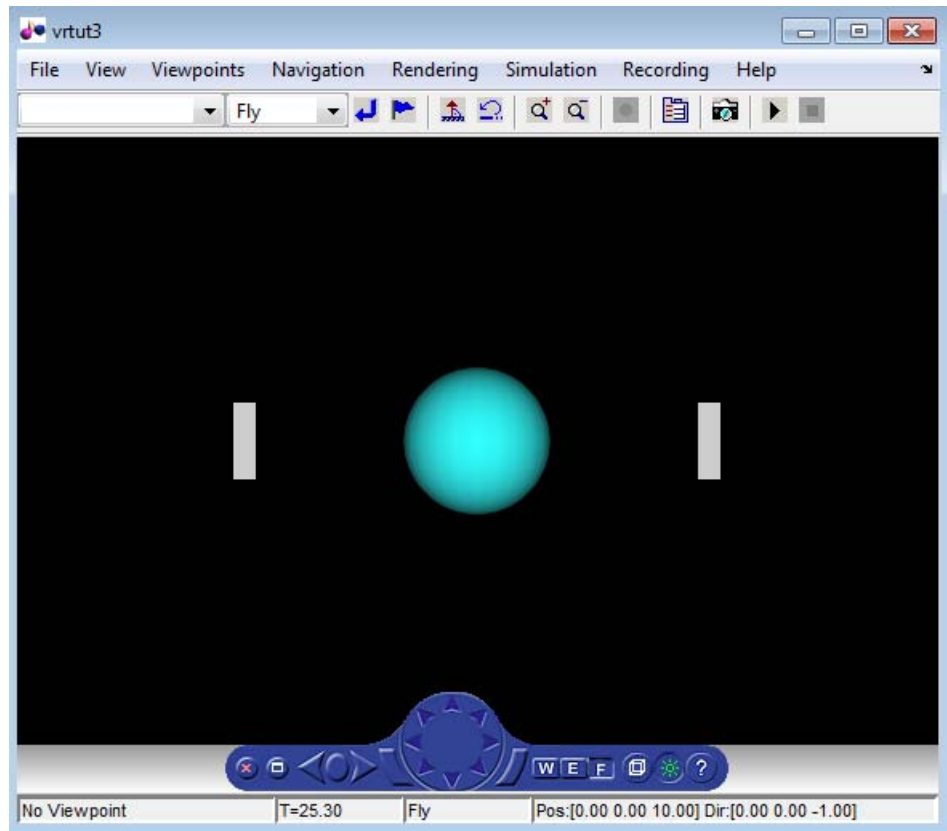
- 10 Double-click the VR Sink block.

The viewer appears.

- 11 Select the **Simulation** menu **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 2-20.

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

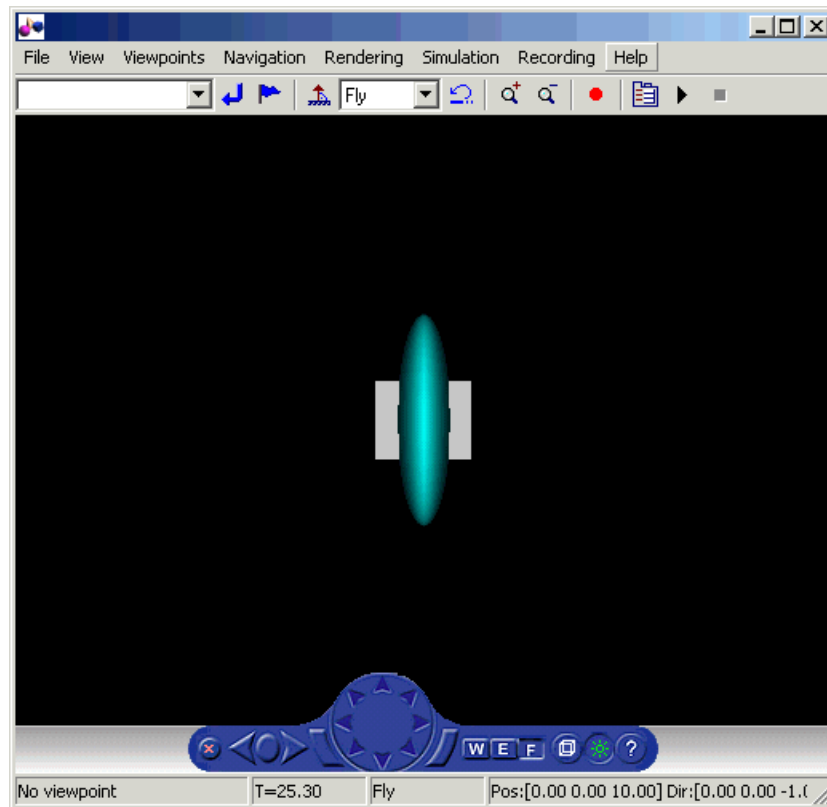
Your default viewer opens and displays the virtual world. For more information on changing your default viewer, see “Set the Default Viewer” on page 2-20.



**13** In the Simulink window, from the **Simulation** menu, click **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 very simple virtual reality model. Using the same method, you can create more complex models for solving the particular problems that you face.

## VRML Data Types

In this section...
“Section Overview” on page 5-24
“VRML Field Data Types” on page 5-24
“VRML Data Class Types” on page 5-27

### Section Overview

VRML data types are used by VRML nodes to define objects and types of data that can appear in the VRML node fields and events.

This section explains these VRML field data types and VRML data class types.

### VRML Field Data Types

The Simulink 3D Animation product provides an interface between the MATLAB and Simulink environment and VRML scenes. With this interface, you can set and get the VRML scene node field values. To work with these values, you must understand the relationship between VRML data types and the corresponding MATLAB data types. The following table illustrates the VRML 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.

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



<b>VRML Type</b>	<b>Description</b>	<b>Simulink 3D Animation Type</b>
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)
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 VRML node.	vrnode
MFFloat	Array of SFFloat values.	Single array (n-by-1)
MFInt32	Array of SFInt32 values.	int32 array (n-by-1)
MFVec2f	Array of SFVec2f values.	Single array (n-by-2)
MFVec3f	Array of SFvec3f values.	Single array (n-by-3)

<b>VRML Type</b>	<b>Description</b>	<b>Simulink 3D Animation Type</b>
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 VRML 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:

<b>Property</b>	<b>Description</b>
<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 VRML boolean data type is returned as a logical value. If set to 'char', the VRML 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 VRML int32 data type is returned as int32. If set to 'double', the VRML 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

Property	Description
	'double'. If set to 'single', the VRML float and color data types (the types of most VRML fields) are returned as 'single'. If set to 'double', the VRML float and color data types are returned as 'double'.

## VRML Data Class Types

A node can contain four classes of data: `field`, `exposedField`, `eventIn`, and `eventOut`. These classes define the behavior of the nodes, the way the nodes are stored in the computer memory, and how they can interact with other nodes and external objects.

VRML Data Class	Description
<code>eventIn</code>	An event that can be received by the node
<code>eventOut</code>	An event that can be sent by the node
<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 `addChildren` `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 VRML 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 the field itself cannot be changed on the fly by other nodes or via the external authoring interface.

You use this class type for fields that are not exposed and do not have the `eventOut` functionality.

### **exposedField**

This is a powerful VRML data class that 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. This allows the chaining of events through many nodes.

The `exposedField` class is accessible to scripts, whereas the `field` class is not.

## 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 perform this scaling, the Simulink 3D Animation viewer might 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. VRML also supports 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.

## Using CAD Models with the Simulink 3D Animation Product

In this section...
“Section Overview” on page 5-30
“Export VRML Models from CAD Tools” on page 5-30
“CAD Virtual World Modeling” on page 5-37
“Link to CAD Virtual Worlds” on page 5-40
“Export VRML Models from CATIA Software” on page 5-46

### Section Overview

When you work with models of dynamic systems, it is often necessary to visualize them in a three-dimensional virtual reality environment. As most of the 3D designs in companies are created using CAD tools, users need to be able to convert these designs into forms that can be used with Simulink or SimMechanics models and applications based on the MATLAB software.

This section describes how to adapt existing CAD designs for visualization using the Simulink 3D Animation software.

This section assumes that the reader has a moderate knowledge of the Simulink 3D Animation product. For VRML-specific information, such as the description of VRML nodes and their fields, refer to the VRML97 standard.

### Export VRML Models from CAD Tools

To export VRML models from CAD tools, you first convert your product assembly model into the VRML format used by the Simulink 3D Animation software. Most CAD tools have VRML export filters, but there are conversion utilities available from third parties if the export filter is not directly available in the CAD tool.

When exporting CAD models into the VRML format, several options can be set to customize the output. These include options 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). The most typical and useful properties are the following:

- “VRML Format Type” on page 5-31
- “Level of Detail Considerations” on page 5-32
- “Units Used in Exported Files” on page 5-32
- “Coordinate System Used” on page 5-33
- “Assembly Hierarchy” on page 5-33

### **VRML Format Type**

There are two major versions of the VRML format used by graphic tools: the older format, called VRML1, and the newer format, called VRML2 (or more often VRML97, according to the adoption year of the ISO standard). The Simulink 3D Animation software uses VRML97, so select VRML97 as the export format.

If your CAD tool allows only VRML1 export, you can use the Ligos V-Realm Builder application, a native VRML scene editor supplied with the Simulink 3D Animation software, to convert models from VRML1 to VRML97. Simply open a VRML1 file and resave it in V-Realm so that the file is automatically saved in the VRML97 format.

---

**Note** All references to the general abbreviation, VRML, refer to the VRML97 standard.

---

### **Level of Detail Considerations**

CAD models are usually parametric models that use proprietary object rendering methods for use in various contexts. During VRML model export, the internal parametric model of the assembly is tessellated. In this process, the model surface is divided into triangular meshes, represented in VRML by the IndexedFaceSet nodes. During tessellation, it is important to set the granularity of the mesh so that it is suitable for further use. Modifying the polygon count afterwards would not only be very difficult, but also not practical, as the resolution independent information of the object shape and structure is lost and cannot be reconstructed based on the tessellated model.

For the effective rendering of moving parts, VRML models should be as simple as possible. However, usually little, if any, visible model degradation is desired. It is often just an issue of finding the appropriate compromise between these two requirements.

As there are significant performance differences among various computers and graphic accelerators, there is no firm recommendation for the number of polygons or triangles suitable for use with the Simulink 3D Animation product. To assess the model's complexity, you can display the resulting VRML 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 usually suitable for further work. If you connect the virtual world to a Simulink model, you have access to more precise measures of suitability, such as the number of frames rendered per second during simulation.

### **Units Used in Exported Files**

VRML length units are meters. To scale exported parts correctly in the virtual world, export the parts using meters. If the exported objects are very small or very large, you may want to create your virtual world in some other scale. In this case, you should export the objects using units other than meters.

VRML viewers are made to measure using dimensions that are comparable to the dimensions of people, to achieve the immersion effect of virtual reality. Viewers assume that the author prepared the scene so that it can be walked through or examined by a virtual visitor to the scene (sometimes called the Avatar), whose physical dimensions are used in calculations for purposes like collision detection, near-object clipping, or terrain following. You can



customize avatar dimensions (and also other navigation-specific parameters such as default navigation speed) using the `NavigationInfo` VRML node. The Simulink 3D Animation viewer enables effective navigation in the virtual world, including scaled scenes (e.g., inspecting miniature objects or visualizing a large-scale aircraft operation in space). For such navigation to be successful, the scene's author must define the `NavigationInfo` parameters correctly.

## Coordinate System Used

VRML 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 VRML system later on, export CAD models using an identical coordinate system whenever possible. If your CAD tool uses a different coordinate system, and it does not allow you to change it for the exported objects, make sure to note the difference between the systems so that you can implement axes transformations in your model later.

Also, make a note of the orientation of the parts in the coordinate system. For instance, if a vehicle model is exported so that it points towards the  $+x$  axis on a road in the virtual world, then the road should also point towards the  $+x$  direction, and the model of vehicle dynamics should also use the  $x$  coordinate.

When the CAD tool allows you to animate parts and assemblies, reset their positions to the initial state before the export.

## Assembly Hierarchy

How assembly of parts are exported depends 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 VRML file has a flat structure, with all part coordinates defined in global coordinates.

- Parts follow some kind of hierarchy defined in the CAD tool. The exported VRML file will use this hierarchy via the VRML Transform-children mechanism, to create a nested structure. In this case, part coordinates are usually defined in the part's parent local coordinate system.

For example, a robot can be exported with the following object hierarchy, in which each part's coordinates are defined in the parent's local coordinate system:

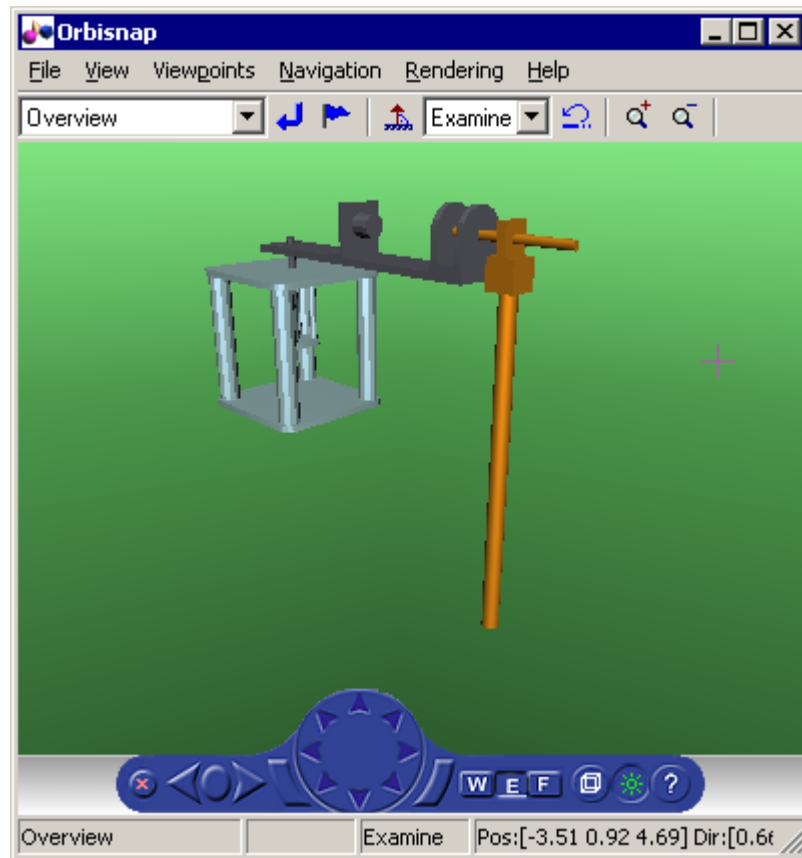
**rotating support — arm — wrist — hand — tool**

So when the rotating support moves, all other parts are usually designed to move with it.

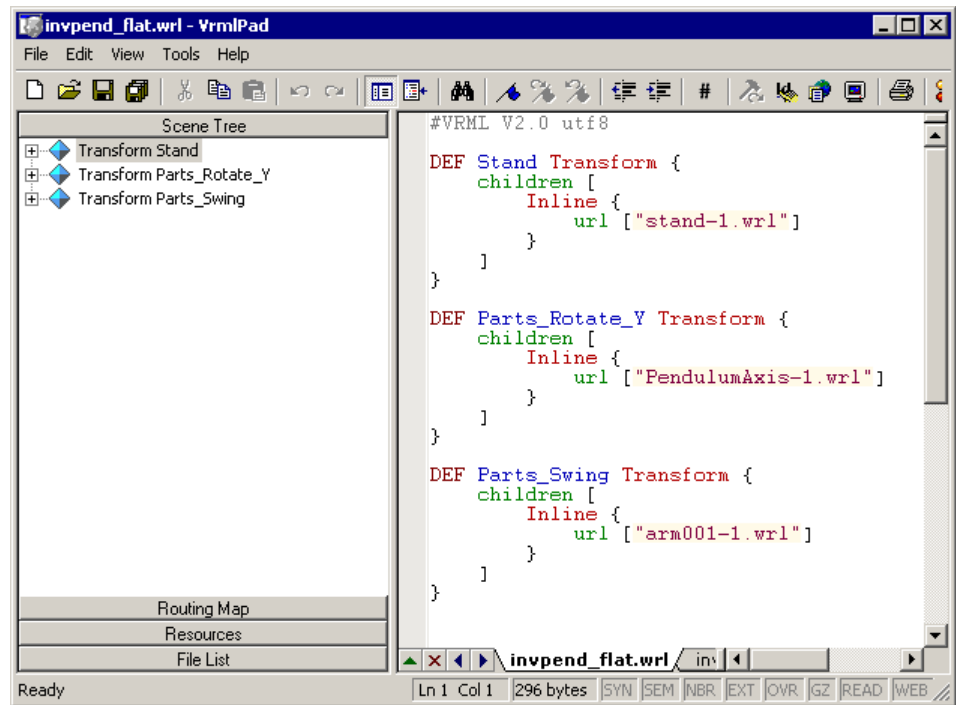
The hierarchy of the VRML file must correspond to the coordinates used in the dynamic model of the assembly as follows:

- If all parts in the Simulink or SimMechanics model are defined in global coordinates, use a flat virtual world structure.
- If all parts in the Simulink or SimMechanics model follow hierarchical relationships, use a nested virtual world structure.

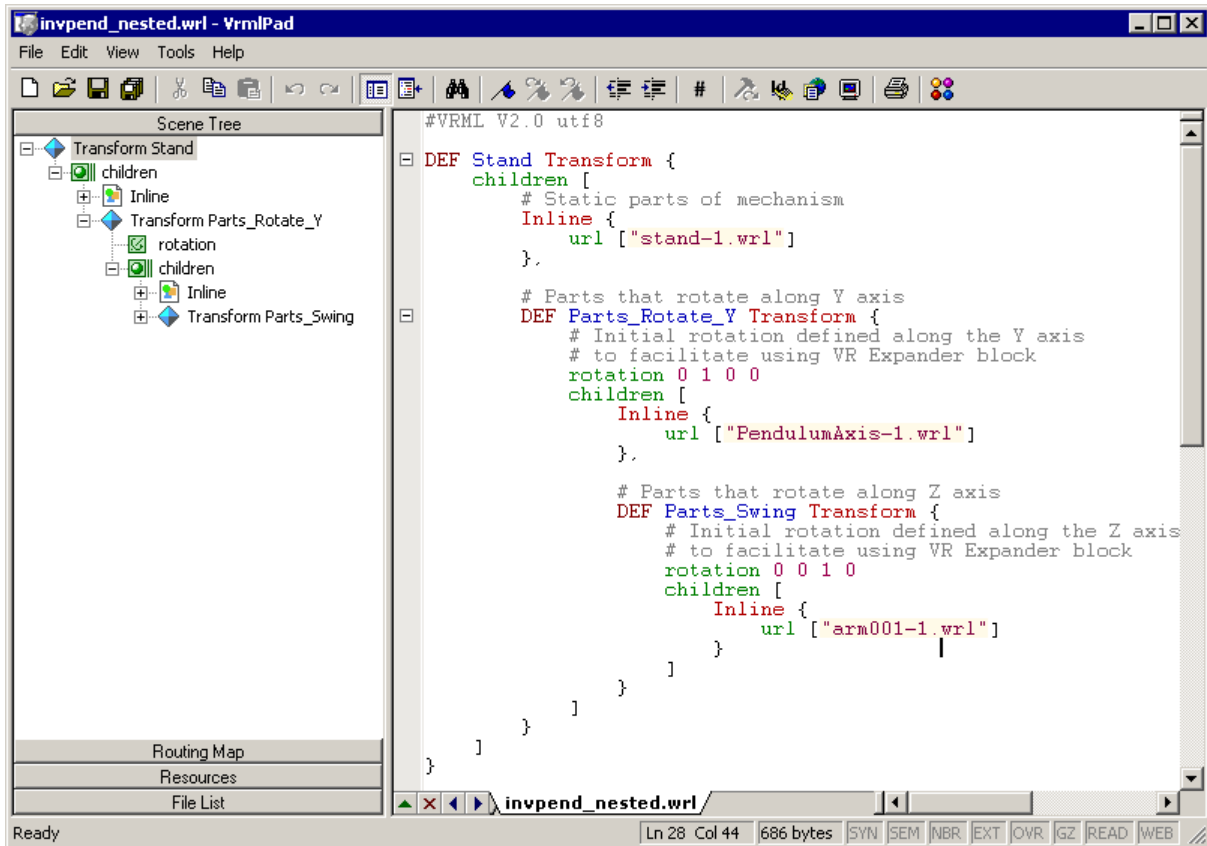
To illustrate these two cases, imagine a rotating pendulum according to the following figure. The gray arm rotates about the vertical axis, while the orange pendulum swings about the  $z$  axis in the rotating gray arm's local coordinates.



If the pendulum dynamics model uses global coordinates for all moving parts, the VRML model has a flat structure as shown in the following figure.



If the pendulum dynamics model uses local coordinates for moving parts, the corresponding VRML model has a nested structure, as shown in the following figure.



Some third-party tools allow you to export each part of the assembly into separate VRML files. All parts are then referenced in one main file using the VRML Inline mechanism. Referencing in this manner is the recommended way to work with assemblies, as the main file is small in size and easy to understand and modify.

## CAD Virtual World Modeling

You can use the 3D World Editor or other editor to manually modify the results of CAD tool export filters (for example, composing the converted model into an urban or manufacturing environment, or adding objects such as viewpoints, backgrounds, and lights) before using them in Simulink 3D

Animation virtual worlds. Typically, adjusting exported files manually in an editor requires the following changes:

- “Wrap Shape Objects with Transforms” on page 5-38
- “Add DEF Names” on page 5-38

### **Wrap Shape Objects with Transforms**

CAD tools export parts into VRML as individual shapes using various VRML object types (e.g., a VRML Shape node or the Inline mechanism). To control part positions and orientations, you need to 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, whose purpose is to transform the coordinates of its children. For instance, after wrapping with a Transform node, an Inline node may have the following syntax:

```
Transform {
  children [
    Inline {
      url ["robot_arm1.wrl"]
    }
  ]
}
```

To set the initial location of the entire assembly in the virtual world, it is a good practice to wrap 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 be accessible from MATLAB interface, each VRML object needs to be given a unique name in the VRML file. You 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 VRML file has the following syntax:

```
DEF Robot_Arm1 Transform {
  children [
    Inline {
      url ["robot_arm1.wrl"]
    }
  ]
}
```

These object names are used in the Simulink 3D Animation functions and in the user interface such as the descriptions of inputs to the VR Sink block. Therefore, it is good practice to give the parts descriptive names to help you manage the orientation in the object hierarchy.

---

**Note** Sometimes it is necessary to correct bugs introduced in the file by the CAD tool export filter. As the VRML 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 files in the Simulink 3D Animation software, consult technical support.

---

## Creating a Virtual World

The VRML file, adjusted manually in the previous steps, is now ready for association with Simulink or SimMechanics models. To work with the virtual world effectively, however, you may want to make additional modifications to the scene file. These changes can be added on an ongoing basis, in parallel with developing and using the dynamic model.

These modifications can be done using a text editor, V-Realm, or any other VRML editor:

- 1 Add the `WorldInfo` node with a scene title (used as the virtual world description in the Simulink 3D Animation software).
- 2 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.

- 3 Add the `Background` node to specify a color backdrop that simulates the ground and sky, as well as optional background textures, such as panoramas for the scene.
- 4 Add several viewpoints to be able to observe the object conveniently from different positions. The viewpoints can be static (defined as independent objects at the top level of the scene hierarchy) or attached to objects that move in the scene for observation 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 `vrtkoff.wrl` example.
- 5 Add lights to the scene in order to illuminate it. Although VRML viewers always have a “headlight” available, it is good practice to define lights in the scene so that it looks the same for every user, according to the scene author’s preferences. The most useful type of VRML light to illuminate a whole scene is the `DirectionalLight` node. It is often practical to use a combination of several such lights to illuminate objects from several directions.
- 6 Add scene surroundings. This step is not crucial for the visualization of interactions between parts in a machine assembly, but is very important for the visualization of simulations, such as those for aircraft and vehicle dynamics, where the position of one object relative to the scene in which it operates is important.

For example, if you want to visualize vehicle dynamics, you would place a virtual car on a virtual road. Both objects need to be to scale (the length units in the car and road models must match), and the car must be placed in an appropriate position relative to the road. You achieve proper car scaling, placement, and orientation in the scene by defining corresponding fields of the main object’s `Transform` node mentioned in “Wrap Shape Objects with Transforms” on page 5-38.

For an example of a complete scene definition, see the `octavia_scene.wrl` VRML file that belongs to the Simulink 3D Animation `vr_octavia` example.

### **Link to CAD Virtual Worlds**

The purpose of this step is to create associations between dynamic model object quantities and corresponding VRML object properties (positions,



rotations, etc.) to establish a live data connection between the model and the virtual world.

Mechanical systems are typically modeled using SimMechanics or Simulink, but the Simulink 3D Animation product allows for the visualization of models implemented in MATLAB. The following section describes the specifics of using the MATLAB interface.

### Linking 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 Double-click the VR Sink block to open the block parameters dialog, where you can define the virtual world. Enter the name of the scene's VRML file in **Source file**, or click **Browse** to select the file interactively. Click **Apply** to load the selected VRML scene.
- 3 For smooth visualization of the movement, it is sometimes necessary to change the block's **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 might result in nonequidistant (in simulation time) updating of the virtual world, giving the user a false impression of system dynamics.
- 4 In **VRML Tree** in the right side of the dialog box, expand the main object's Transform branch, and, 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 5-38. Each part is represented by named Transform nodes, and you select the check box next to its rotation and position fields. These selections tell the VR Sink block that you want to control the rotation and position of these parts. You can also 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 the number of input ports.

After the VR Sink block is associated with a virtual world, you can double-click it to open the Simulink 3D Animation viewer. Block parameters are available through the menu **Simulation > Block Properties** in the viewer.

VR Sink inputs take signals of the type corresponding to their VRML 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 5-33, where the angle value is in radians.

The user has to match the coordinate system used by the Simulink model to that of the virtual world. If the two systems are not identical, some kind of axes transformation is necessary.

While object positions are usually available in the form required by VRML (Cartesian coordinates), rotations usually have to be converted from some other representation. In many cases, object rotations are defined using the rotation matrix representation. For converting such rotations into the VRML format, use the Rotation Matrix to VRML Rotation block found in the Utilities sublibrary of `vrllib`.

The objects’ positions and rotations are treated differently depending on the virtual world hierarchy:

- When all parts in a Simulink model are defined in global coordinates, and the virtual world has a flat structure of independent objects, use the following 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 center of rotation defined as the coordinate system origin. (As the default center of rotation of VRML Transform objects is <code>[0 0 0]</code> , it is usually not necessary to define it for each part in the VRML file.

- When all parts in Simulink model follow hierarchical relations, and the virtual world has a nested structure, use the following 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's tool position relative to the robot's 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's tool rotation relative to the robot's hand.</p> <p>To visually match the positions of joints between objects, it is usually necessary to coincide the center of rotation defined in the virtual world with the center of rotation defined in the Simulink model, as joints between parts are usually positioned not in the origin, <math>[0\ 0\ 0]</math>, of the parent's coordinate system.</p> <p>To define a center of rotation different from the default value, <math>[0\ 0\ 0]</math>, define the <b>center</b> field of the child's Transform node in the VRML file. For example, define the robot's tool center of rotation to coincide with the joint connecting the hand and the tool in the hand's 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 VRML rotation axis, so constructing the  $[\text{axis}\ \text{angle}]$  four-element VRML rotation vector is trivial.

### Initial Conditions

A Simulink model's initial conditions must correspond to the initial object's positions and rotations defined in the virtual world. Otherwise, the object controlled from Simulink would "jump" from the position defined in the VRML file to the position dictated by the Simulink software at the start of the simulation. You can compensate for this offset either in the VRML file (by defining an another level of nested Transform around the controlled object)

or in the Simulink model by adding the object's initial position to the model calculations before sending to the VR Sink block.

You should align the Simulink model's initial conditions with the virtual world's object positions, while maintaining the correct position of the object relative to the surrounding scene. To do so, you may need to adjust the position of the object's surroundings (e.g., move the road position so that the car at position [0 0 0] stays on the road, with the wheels neither sinking nor floating above the road surface).

### **Use of VR Placeholder and VR Signal Expander**

The VR Sink block accepts only inputs that define fully qualified VRML 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, whether as a single value or as an element of a vector, 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 (as a part of the initial rotation of an object Transform node) is defined in the VRML file, it is possible to send to the VR Sink block a VRML rotation value consisting of three placeholder signals and the computed angle, forming a valid four-element [axis angle] vector.

## SimMechanics Models

You can use the Simulink 3D Animation product to view the behavior of a model created with the SimMechanics software. First, you build a model of a machine in the Simulink interface using SimMechanics blocks. Then, create a detailed picture of your machine in a virtual world, connect this world to the SimMechanics body sensor outputs, and view the behavior of the bodies in a VRML viewer.

The SimMechanics software is very well suited for 3D visualizations using the Simulink 3D Animation product. Apart from features that SimMechanics product offers for modeling mechanical assemblies, the following features simplify the visualization of SimMechanics models in virtual reality:

- SimMechanics and VRML coordinate systems are identical.
- In the SimMechanics software, you can work with both global and local object coordinates, so it is easy to adapt the model to the structure of the virtual world exported from the CAD tool.

The SimMechanics product also offers a convenient way of importing CAD assembly designs into SimMechanics machines through the SimMechanics Link interface. Alternatively, when you export a CAD assembly to the VRML format, the additional steps described in this section can add virtual reality visualization to such assemblies.

The Simulink 3D Animation software includes the following functions for working with SimMechanics files: `vrcadcleanup`, `vrphysmod`, and `st12vrml`.

## Link to a SimMechanics Model

Depending on the virtual world hierarchy, you can use one of two methods to help visualize SimMechanics 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 connected to appropriate coordinate systems attached to the bodies, with positions and rotations defined using global coordinates. In most cases, it is appropriate to connect the sensor to a body coordinate system with origin at  $[0 \ 0 \ 0]$  and with an initial rotation matrix defined as the identity matrix,  $[1 \ 0 \ 0; 0 \ 1 \ 0, 0 \ 0 \ 1]$ , in the global coordinate.

- When the virtual world has a hierarchical structure of nested objects, the body positions and rotations can be obtained using Body Sensor blocks with output set to use local body coordinates. In some special cases (e.g., when two bodies are connected through a revolving joint), it is possible to get the angle between the objects using a Joint Sensor block.

### Linking the Virtual World to a MATLAB Model

For interacting with virtual worlds, the Simulink 3D Animation product also 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 may 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, are taking place. In this case, you must 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 “MATLAB Interaction”.

### Export VRML Models from CATIA Software

This topic describes how to use CAD designs created in the CATIA® product to create Simulink 3D Animation VRML 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.

When exporting products, the CATIA software creates one compound VRML file that contains all the parts of the product.

Occasionally, you might need to export each part of the assembly hierarchy into a separate VRML file. You can do this in the CATIA environment as follows:

- 1 Save each part individually to a separate VRML file.
- 2 Create the main model VRML file manually, with `Inline` references to the part files.

### **CATIA Coordinate Systems**

The CATIA software also 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 (“VRML Coordinate System” on page 1-13). Account for the coordinate system when you export objects from the CATIA environment into VRML virtual worlds and/or manipulate them using the Simulink 3D Animation software.

You can also define a different coordinate system. To do this, create an axis system within the current geometrical set. 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 VRML coordinate system. This makes all the coordinates and orientations of objects compatible with other objects you combine into VRML virtual worlds.

### **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, select the 3D Accuracy options to control the visualization mesh detail.

You can achieve best results by using 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 VRML 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. You want to find the setting that gives you the smallest possible VRML model, but it must still be visually acceptable.

### **VRML Export Filter Settings**

The CATIA software enables you to tune some VRML export options. These are available in the **Tools > Options > General > Compatibility > VRML:**

- Select VRML97 as the export format  
The Simulink 3D Animation software uses VRML97 standard format.
- Select the **Save normals** check box  
This option affects whether or not to export explicit face normals definitions.
- Clear the **Save edges** check box  
Clear this check box for optimum performance. Selecting this check box directs the CATIA software to also export object edges (in the form of IndexedLineSets).
- Select 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.
- Select the VRML model background color  
This option applies only to exporting products.

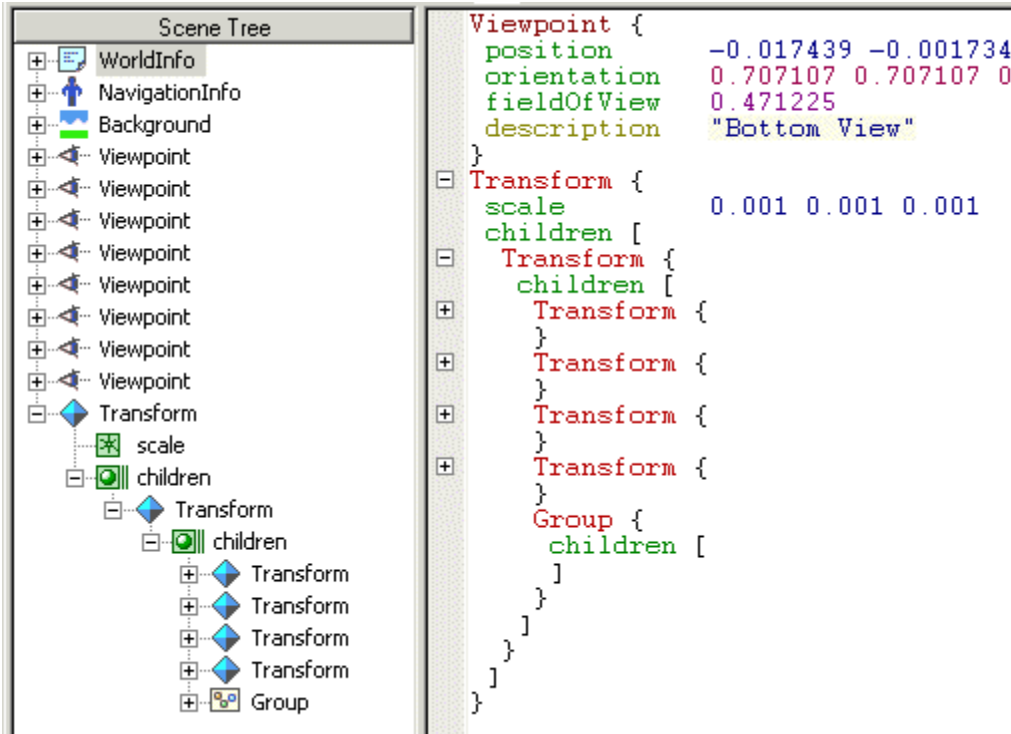


## **Structure of VRML Models Exported from the CATIA Environment**

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 to 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`). For the most part, you will only need to work with the main transforms that represent each part.

The following contains the VRML model of a cylinder assembly consisting of four parts:



The left tree view illustrates the overall structure of the model.

- 1 The CATIA software saves the general model information in the WorldInfo, NavigationInfo, and Background nodes.
- 2 The software exports the default CATIA viewpoints (it does not export user-defined viewpoints).

Following this section, common to all products exported to VRML, 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. In the figure, the contents of the part transforms are collapsed so that only the top-level objects are visible for clarity. After four transforms representing CATParts, the export adds an empty Group node at the place where CATIA Constraints are defined. You can delete such empty nodes from the VRML model.

The contents of the CATProduct are scaled down by a factor of 1000 (conversion of units from millimeters to meters).

When you have VRML files created with the CATIA software, take into account the following known features for further use with the Simulink 3D Animation software:

- Object - Exporting to VRML does not preserve CATProduct and CATPart names. The CATIA environment only creates synthetic VRML DEF names for sub-parts, materials, and object coordinate fields. These synthetic names change between two or more consecutive export operations.

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.

- 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. This field is referenced from several sub-parts throughout the file with the USE directive.

Preserve this reference. Do not delete or rename the original Coordinate field DEF name.

- The VRML file stores only one material per part. If the part consists of several sub-parts in VRML, their material also uses the USE reference to the material of the first sub-part.
- Textures are supported

- LOD (exporting parts in several levels of detail for more efficient visualization) is not supported
- The CATIA software exports models in millimeters, VRML units are meters.  
Scale resulting objects to visualize them effectively.
- The VRML file does not save user-defined CATIA viewpoints.
- The main Transform representing the CATIA product is always scaled by a factor of 0.001 (conversion from millimeters to meters), regardless of the units used in the CATIA document

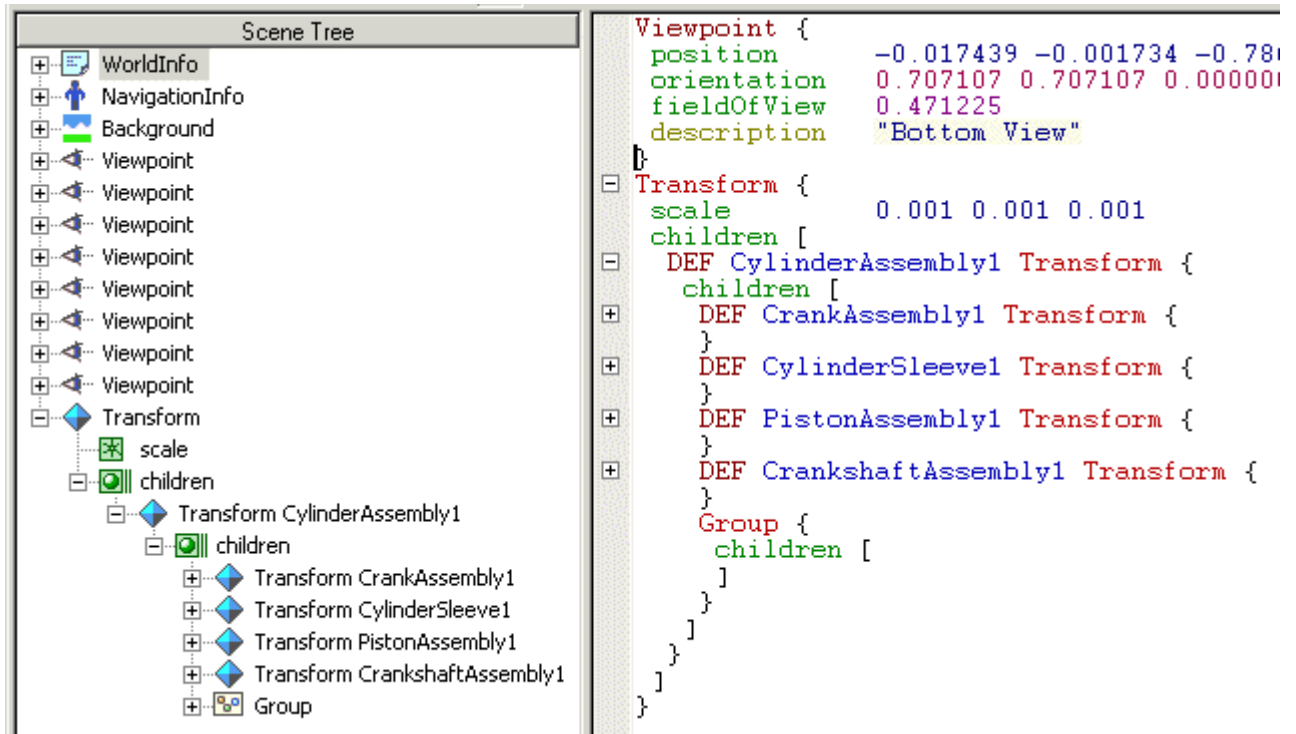
### **Adjusting Resulting VRML files**

Adjust exported VRML files to use the exported VRML models with the Simulink 3D Animation software. You can perform these adjustments manually, as described in this topic, or use the `vrcadcleanup` and `vrphysmod` functions to perform some of these tasks.

- Adding DEF Names to Part Transforms

In the VRML file, assign a unique name for each VRML object. To do this, add the `DEF Object_Name` statement to each part Transform line.

The following is an example of 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.

- Scaling of 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.

If you have a small object, or an object that you must 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. You can do this by deleting the scale field from the top level transform, and adding it to each individual part transform. For example,

```
Transform {
  children [
    DEF CylinderAssembly1 Transform {
      children [
        DEF CrankAssembly1 Transform {
          scale 0.001 0.001 0.001
          ..
        }
      ]
    }
  ]
}
```

# Using the 3D World Editor

---

- “3D World Editor” on page 6-2
- “Getting Started with the 3D World Editor” on page 6-7
- “Basic Editing” on page 6-12
- “3D World Editor Library” on page 6-22

## 3D World Editor

In this section...
“Supported Platforms” on page 6-2
“Use with Other Editors” on page 6-2
“VRML Support” on page 6-2
“VRML Nodes, Library Objects, and Templates” on page 6-3
“3D World Editor Interface” on page 6-4

The 3D World Editor is a native VRML authoring tool that provides a convenient graphical interface to the VRML syntax.

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 5-7.

### Supported Platforms

The 3D World Editor works on all supported platforms for the Simulink 3D Animation product. For details, see <http://www.mathworks.com/products/3d-animation/requirements.html>.

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

### Use with Other Editors

As you create a virtual world, you can use different editors for different 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 “VRML Editors” on page 5-2.

### VRML Support

The primary file format for the 3D World Editor is VRML.



The 3D World Editor supports all VRML97 types and language elements, except as noted in this section.

For general VRML limitations relating to the Simulink 3D Animation software as a whole, see “VRML Compatibility” on page 1-12.

### **PixelTexture Nodes**

You cannot create or edit `PixelTexture` node image contents. Existing `PixelTexture` node image contents are fully preserved

## **VRML Nodes, Library Objects, and Templates**

Use the 3D World Editor to specify VRML to create 3-D virtual worlds that you can connect to a Simulink model.

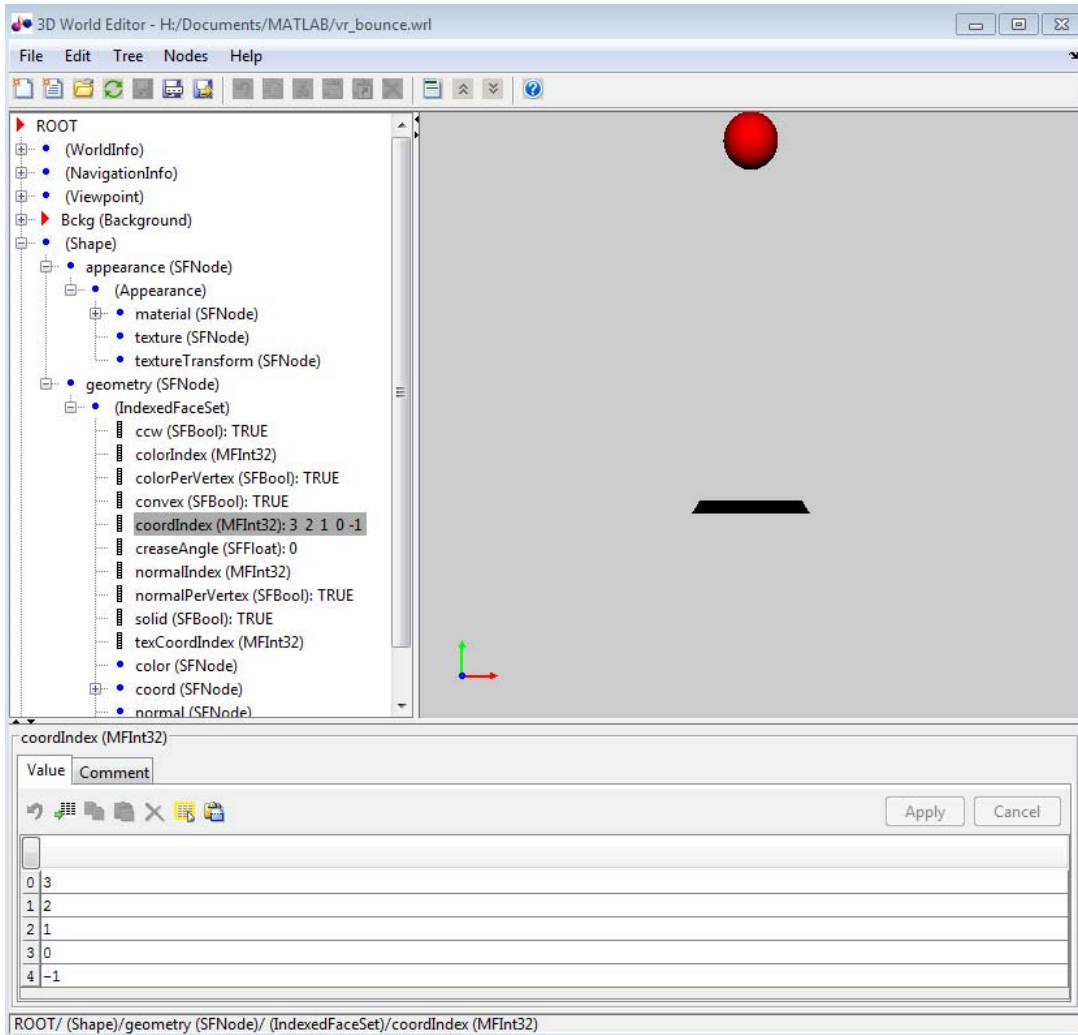
Use the 3D World Editor tools to:

- Assemble VRML nodes to create a virtual world. 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

### **3D World Editor Interface**

The 3D World Editor provides a simple graphical interface for 3-D editing, with 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.



The 3D World Editor offers not only the graphical representation of a 3-D scene and tools for interactive creation of graphical elements, but also a hierarchical tree style (in the **tree structure** pane) of all the 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 VRML node types. In the tree viewer, you give the nodes unique names.

In the **object properties edit** pane, you can edit a selected property or add a comment to a selected node or property.

## Getting Started with the 3D World Editor

### In this section...

“3D World Editor Is the Default Editor” on page 6-7

“Open the Editor” on page 6-7

“Create a Virtual World” on page 6-8

### 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 2-26).

---

**Note** You can also use the V-Realm Editor. For more information, see “V-Realm Builder Help” on page 2-26.

---

### Open the Editor

To open the 3D World Editor from the MATLAB Toolstrip, in the **Apps** tab, click the down arrow on the right side of the tab. Scroll down to the **Simulation Graphics and Reporting** area and click the **3D World Editor** button.

To open the 3D World Editor from the MATLAB command line, regardless of **Default Editor** preference setting, use the `vredit` command. To open an empty virtual world in the 3D World Editor, enter:

```
vredit
```

To open an existing virtual world in the 3D World Editor, enter:

```
vredit('membrane.wrl')
```

As an alternative, if the 3D World Editor is your default VRML editor, you can start it from the MATLAB command line using the `edit` command.

- To open an empty VRML file, enter:

```
edit(vrworld(''))
```

- To open an existing VRML file (`myVRMLfile.wrl`, in this example), enter:

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

You can also open an existing VRML file from within the 3D World Editor, using **File > Open**.

### Preference Options for 3D World Editor Startup Position

For specifying the startup position of the editor, the **Simulink 3D Animation Preferences > 3D World Editor** pane includes the following options:

- For the default location, select **Position**. Then specify the pixel location for the upper-left corner and the lower-right corner (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**.

### Create a Virtual World

There are a number of tasks involved in creating a virtual world. You can use the 3D World Editor throughout the process of building a virtual world, and you can perform activities 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 5-7.

In general, the following is 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 “Basic Editing” on page 6-12.

- 1 Open a new VRML 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** Insert 3D World Editor library objects to define aspects, such as textures, for virtual world objects.

- Give a VRML DEF name to each object that you create, so that you can access them using Simulink 3D Animation.
- You can use Orbisnap to view library objects to determine which objects you want to insert into the virtual world.







**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 VRML file.

### 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	

### Template VRML Files

The 3D World Editor includes template VRML 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::**

Edit the file to adapt the template world for your application. 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 that 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.

## Basic Editing

In this section...
“Add Objects” on page 6-12
“Copy and Paste a Node” on page 6-14
“Edit Object Properties” on page 6-15
“Document a Virtual World Using Comments” on page 6-15
“Expand and Collapse Nodes” on page 6-16
“Wrap Nodes as Children of Another Node” on page 6-16
“Remove Nodes” on page 6-17
“Save and Export Virtual World Files” on page 6-17
“Navigate a Virtual World” on page 6-18
“Specify Virtual World Rendering” on page 6-19
“Edit VRML Scripts” on page 6-20

These topics describes how to use the 3D World Editor for common tasks involved in creating a virtual world.

For information about opening a file in the editor, see “Getting Started with the 3D World Editor” on page 6-7.

For a step-by-step tutorial, see “Build and Connect a Virtual World” on page 5-7.

### Add Objects

Add virtual world objects 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> Select appropriate submenus to add the node that you want.</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> Select the <b>Add Node</b> menu, and then select appropriate submenus to add the node that you want.</li> </ol>
Insert an object from a library	<p>For <b>Material</b>, <b>Texture</b>, and <b>children</b> nodes, select the <b>Insert From</b> menu option (from either the <b>Nodes</b> menu or from the context menu for a node).</p> <p>For information about library objects, see “3D World Editor Library” on page 6-22.</p>
Add an inlined VRML file	<p>For a <b>ROOT</b> or <b>children</b> node, from the <b>Nodes</b> menu or the context menu for the node, select the <b>Inline VRML File</b> menu item.</p>

The node that you add gets added to different locations in the hierarchy, depending on the node that you select to begin the process of 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 VRML file to another virtual world in a different VRML file. After you copy the nodes from one virtual world, select **File > Open** to open the second VRML 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  $25 \cdot \pi / 180$ .

## Set Viewpoint Values in the 3D World Editor Based on Camera Position

You can use the current camera position to interactively specify 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**.

## 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 VRML file, next to the given node or property, on lines that begin with #.

### 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**.

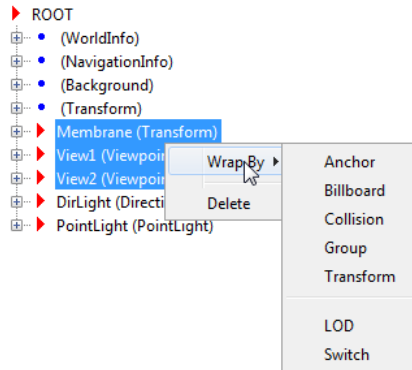
### Hide Default Values

To simplify the tree view, you can hide default VRML values. Select **Tree > Hide Default Values**. To display default values, clear the **Hide Default Values** option.

### 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 Files

You can save your virtual world files as VRML (.wrl) files 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` 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 is supported by the Simulink 3D Animation viewer and 3D World Editor (for example, the saved file preserves links to the library texture files).

Use the **File > Export** menu item to export a fully VRML97–compliant file for use:

- With other VRML tools
- On different computers
- In previous versions of the Simulink 3D Animation (previously the Virtual Reality Toolbox) product

For exported files, the 3D World Editor copies referenced inlined VRML 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.

### Navigate a Virtual World

You can 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 methods (for example, fly or walk) and speed.

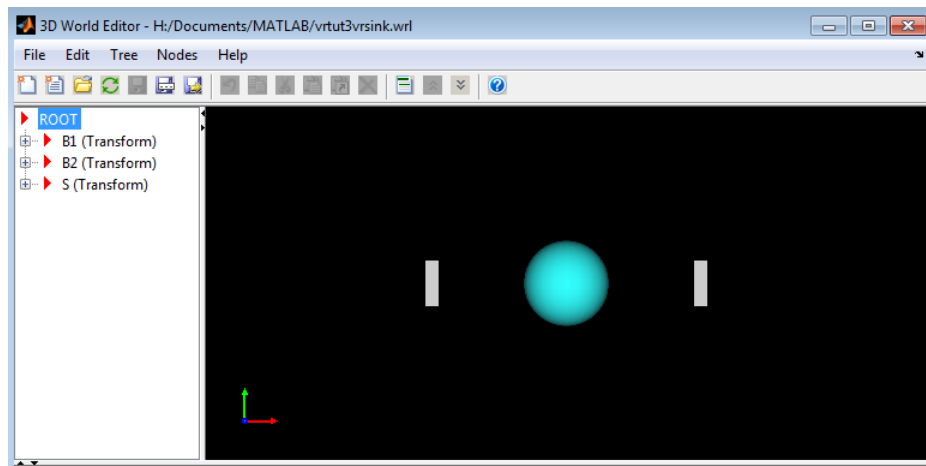
To save these navigation settings in a virtual world file, you must define the navigation properties in a `NavigationInfo` node.

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 3D World Editor **virtual world display** pane includes 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 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**.

## 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.

## Edit VRML Scripts

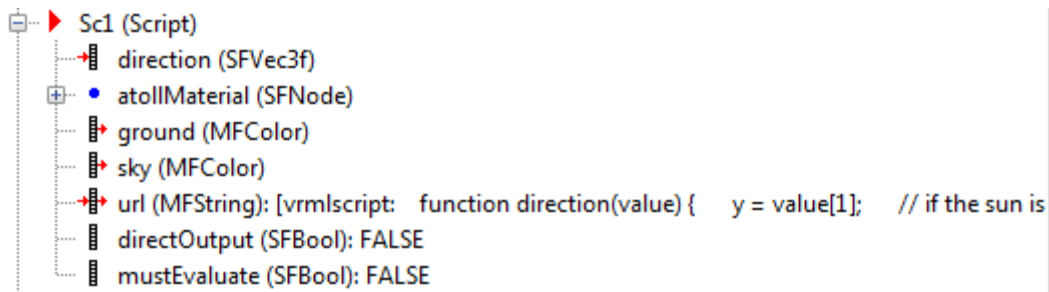
To add a VRML Script node:

- 1 In the **Tree structure** pane, select the ROOT node.
- 2 Select the appropriate kind of script, using the **Node > Add > Common > Script** menu.

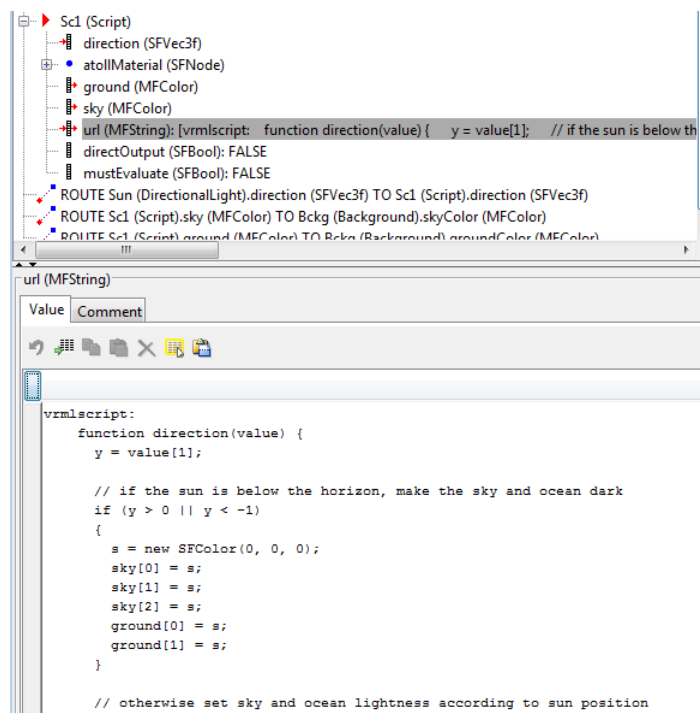
To add Script interface elements:

- 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.



To edit a url node, click the node and edit the URL code in the **Object property edit** pane.



## 3D World Editor Library

In this section...
“3D World Editor Library Objects” on page 6-22
“Add a Library Object” on page 6-22
“Guidelines for Using Custom Objects” on page 6-23

### 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. For example, from the **Component** sublibrary, you can choose from several vehicle objects, such as a van or pickup truck.

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.

- 3 Follow the folder paths to find the object that you want to insert.

You can also insert VRML 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 VRML file into the virtual world that you are editing, use the **Nodes > Inline VRML 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 6-23.

## Guidelines for Using Custom Objects

If you use a VRML object from a source other than from the 3D World Editor object library, the object must comply with the following guidelines:

- A component object must be in a VRML file that contains at least one **Transform** node.
- A material object must be a VRML 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 VRML file whose only content is a fully qualified **ImageTexture** node.

If you create VRML 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 to ensure 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.



# Viewing Virtual Worlds

---

- “VRML Viewers” on page 7-2
- “Simulink® 3D Animation™ Viewer” on page 7-4
- “Blaxxun Contact VRML Plug-In” on page 7-54
- “Legacy Simulink® 3D Animation™ Viewer” on page 7-61

## VRML Viewers

After you create a virtual world in VRML (as described in “Build Virtual Reality Worlds”), you can visualize that virtual world with the Simulink 3D Animation viewer or with a VRML-enabled Web browser. The product includes its own viewer as the default for all supported platforms. It is the preferred method of viewing virtual worlds.

If you are on a Windows platform, you can install a VRML plug-in and view a virtual world in your preferred Web browser. For Windows platforms, the software includes the VRML plug-in Blaxxun Contact. This plug-in is the only supported VRML plug-in. The Blaxxun Contact plug-in does not work reliably on Microsoft Windows 7 platforms.

---

**Note** The Blaxxun Contact VRML plug-in is required for sound. Other Web viewers may allow for sound playback, but are not officially supported.

---

The Simulink 3D Animation product also includes the Orbisnap viewer. Orbisnap is a free, optional, stand-alone VRML97 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 might want to 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, if you want to visualize a virtual world active in a Simulink 3D Animation session that is running on a computer in another part of the building, or across the network. This functionality allows you to remotely view a simulation, but not control it.
- View and navigate, but not simulate, a VRML world. You can navigate, render, and otherwise visualize a VRML world without simulating it.

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.



With Orbisnap or Blaxxun Contact (Windows only) viewers, you can display a simulated process remotely using the TCP/IP protocol. This allows you to watch a simulated virtual world not only on the computer where MATLAB and Simulink are running, but also on other computers connected through the Internet.

## Simulink 3D Animation Viewer

In this section...
“Interface” on page 7-4
“Menu Bar” on page 7-8
“Toolbar” on page 7-9
“Navigation Panel” on page 7-10
“Setting Viewer Appearance Preferences” on page 7-13
“Starting and Stopping Simulations” on page 7-13
“Navigate a Virtual World” on page 7-14
“Define File Name Tokens” on page 7-22
“File Name Tokens” on page 7-24
“Capture Frames” on page 7-25
“Animation Recording File Formats” on page 7-27
“Record Files in the VRML Format” on page 7-28
“Record Files in the Audio Video Interleave (AVI) Format” on page 7-29
“Schedule Files for Recording” on page 7-32
“Start and Stop Animation Recording” on page 7-35
“View Animation Files” on page 7-35
“Viewpoints” on page 7-37
“Navigate Through Viewpoints” on page 7-38
“Reset Viewpoints” on page 7-41
“Define Viewpoints” on page 7-41
“Specify Rendering Techniques” on page 7-45

### Interface

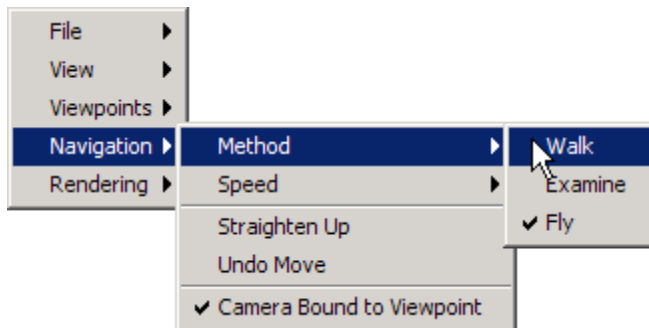
The Simulink 3D Animation software contains a viewer as the default method for viewing virtual worlds. You can use this viewer on any supported operating system. For a list of supported operating systems, see “System

Requirements” on page 2-6. This viewer uses MATLAB figures. It provides a number of 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:



- Right-click in the viewer window to display a context menu that contains the viewer commands. For example:



- Combine multiple virtual reality viewer figures in several tiles of a MATLAB figure.
- Minor visual differences, such as:
  - In the legacy viewer, checks denote selected menu command options. In the default viewer, circles denote selected menu command options.
  - In the legacy viewer, the status bar has a border. In the default viewer, the status bar has no border.

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:

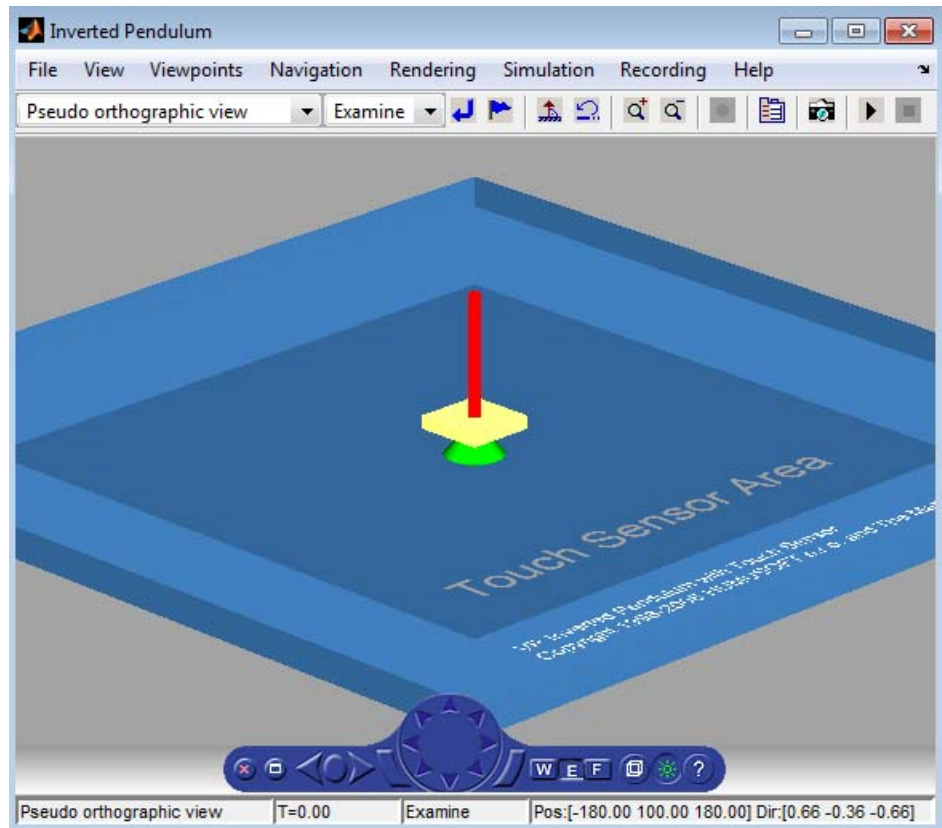
- 1 Select a Simulink 3D Animation example and type that example's name in the MATLAB Command Window. For example:

```
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.

## Menu Bar











The Simulink 3D Animation viewer menu bar has the following menus:

- **File** — General file operation options, including:
  - **New Window** — Opens another window for the virtual scene. Use this option if you want multiple views of the virtual scene open at the same time.
  - **Open in Editor** — Opens the default editor (V-Realm Builder) for the current virtual world. The editor opens with the virtual world already loaded into the editor.
  - **Reload** — Reloads the saved virtual world. If you have created any viewpoints in this session, they are not retained unless you have saved them with the **Save As** option.
  - **Save As** — Allows you to save the virtual world.
  - **Close** — Closes the viewer window.
- **View** — Enables you to customize the Simulink 3D Animation viewer, including:
  - **Toolbar** — Toggles the toolbar display.
  - **Status Bar** — Toggles the status bar display at the bottom of the viewer. This display includes the current viewpoint, simulation time, navigation method, and the camera position and direction.
  - **Navigation Zones** — Toggles the navigation zones on and off (see “Navigate a Virtual World” on page 7-14).
  - **Navigation Panel** — Controls the display of the navigation panel, including toggling it.
  - **Triad** — Displays changes in the orientation (coordinate axes) of nodes in a virtual world. The triad has 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.
  - **Zoom In/Out** — Zooms in or out of the viewer scene.
  - **Normal (100%)** — Returns the zoom to normal (initial viewpoint setting).

- **Viewpoints** — Manages the virtual world viewpoints.
- **Navigation** — Manages scene navigation.
- **Rendering** — Manages scene rendering (see “Specify Rendering Techniques” on page 7-45).
- **Simulation** — Manages model starting and stopping a simulation, as well block parameters for Simulink 3D Animation blocks.
- **Recording** — Manages frame capture and animation recording file parameters.
- **Help** — Displays the Help browser for the Simulink 3D Animation viewer.

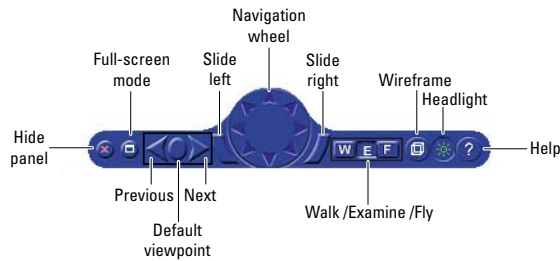
## Toolbar

You can use the Simulink 3D Animation viewer toolbar drop-down lists and buttons for several common operations that are also available from the menu bar:

- Drop-down list of all the viewpoints in the virtual world
- **Return to viewpoint** 
- **Create viewpoint** 
- **Straighten up** 
- Drop-down list that displays the navigation options Walk, Examine, and Fly
- **Undo move** 
- **Zoom in/out** 
- **Start/stop recording** 
- **Block parameters** 
- **Capture a frame screenshot** 
- **Start/pause/continue simulation** 
- **Stop simulation** 

## 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.



- **Hide panel** — Toggles the navigation panel.
- **Full-screen mode** — Toggles full-screen display of the virtual world.
- **Next/previous viewpoint** — Toggles through the list of viewpoints.
- **Return to default viewpoint** — Returns focus to original default viewpoint.
- **Slide left/right** — Slides the view left or right.
- **Navigation wheel** — Moves view in one of eight directions.
- **Navigation method** — Manages scene navigation.
- **Wireframe toggle** — Toggles scene wireframe rendering.
- **Headlight toggle** — Toggles camera headlight.
- **Help** — Invokes the viewer online help.

The following table summarizes the possible operations from the menu bar, toolbar, navigation panel, and keyboard.



<b>Operation</b>	<b>Menu Bar</b>	<b>Toolbar</b>	<b>Navigation Panel</b>	<b>Keyboard Shortcut</b>
New Window	X			
Open in Editor	X			
Reload	X			
Save As	X			
Close	X			
Toolbar	X			
Status Bar	X			
Navigation Zones	X			<b>F7</b>
Navigation Panel	X		X	
Full-screen mode	X		X	<b>Ctrl+F</b>
Zoom In	X	X		<b>+</b>
Zoom Out	X	X		<b>-</b>
Normal (100%)	X			
Previous Viewpoint	X		X	<b>Page Up</b>
Next Viewpoint	X		X	<b>Page Down</b>
Return to Viewpoint	X	X	X	<b>Home</b>
Go to Default Viewpoint	X			
Create Viewpoint	X	X		
Remove Current Viewpoint	X			
Pseudo orthographic view	X			
Close View	X			
View from top	X			

<b>Operation</b>	<b>Menu Bar</b>	<b>Toolbar</b>	<b>Navigation Panel</b>	<b>Keyboard Shortcut</b>
X axis	X			
Z axis	X			
Method	X	X	X	<b>Shift+W, Shift+E, Shift+F</b>
Speed	X			
Straighten Up	X	X		<b>F9</b>
Undo Move	X	X		<b>Backspace</b>
Camera Bound to Viewpoint	X			<b>Ctrl+F10</b>
Antialiasing	X			<b>F8</b>
Headlight	X		X	<b>F6</b>
Lighting	X			
Textures	X			
Maximum Texture Size	X			
Transparency	X			
Wireframe	X		X	<b>F5</b>
Start (Simulation)	X	X		<b>Ctrl+T</b>
Stop (Simulation)	X	X		<b>Ctrl+T</b>
Block Parameters	X	X		
Start Recording	X	X		<b>Ctrl+R</b>
Stop Recording	X	X		<b>Ctrl+R</b>
Capture Frame	X	X		<b>Ctrl+I</b>
Capture and Recording Parameters	X	X		

<b>Operation</b>	<b>Menu Bar</b>	<b>Toolbar</b>	<b>Navigation Panel</b>	<b>Keyboard Shortcut</b>
Slide Left			X	
Navigation Wheel			X	
Slide Right			X	
Help	X		X	
Pan Left/Right				<b>Left/Right Arrows, Shift Left/Right Arrows</b>
Pan Up/Down				<b>Up/Down Arrows</b>
Move Forward/Backward				<b>Shift+Up/Down Arrows</b>
Orbit Around Selected Object				<b>Ctrl+Left/Right/Up/Down Arrow</b>
Slide Left/Right/Up/Down				<b>Alt+arrows</b>
Tilt Left/Right				<b>Shift+Alt+Left/Right Arrow</b>

## Setting Viewer Appearance Preferences

You can configure the appearance of the viewer using Simulink 3D Animation preferences. For details, see “Figure Preferences” on page 2-34.

## Starting and Stopping Simulations

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.

---

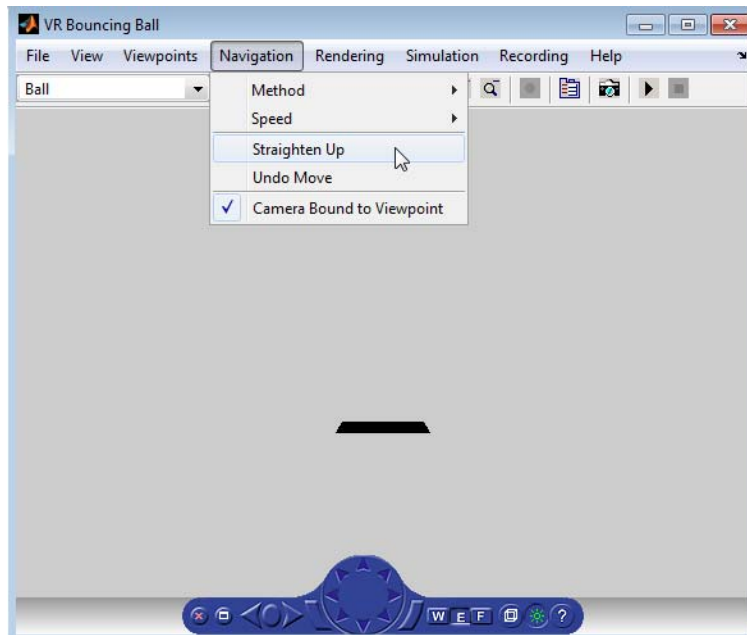
**Note** The **Ctrl+T** operation is available only if you started the viewer from a Simulink model. If you start the viewer through the MATLAB interface, no Simulink model is associated with the viewer. You cannot start and stop the simulation in this case.

---

### Navigate a Virtual World

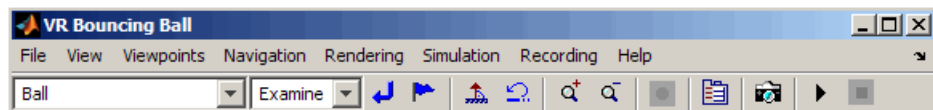
You can navigate around a virtual scene using the menu bar, toolbar, navigation panel, mouse, and keyboard. The `vrbounce` example illustrates the several key features of the viewer.

**Navigation view** — You can change the camera position. From the menu bar, select the **Navigation** menu **Straighten Up** option. Alternatively, from the toolbar, you can 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 the following:

- From the menu bar, select the **Navigation** menu **Method** option. This option provides three choices: Walk, Examine, or Fly. See the table Simulink® 3D Animation™ Viewer Mouse Navigation on page 7-17.
- From the toolbar, select the drop-down list that displays the navigation options Walk, Examine, and Fly.

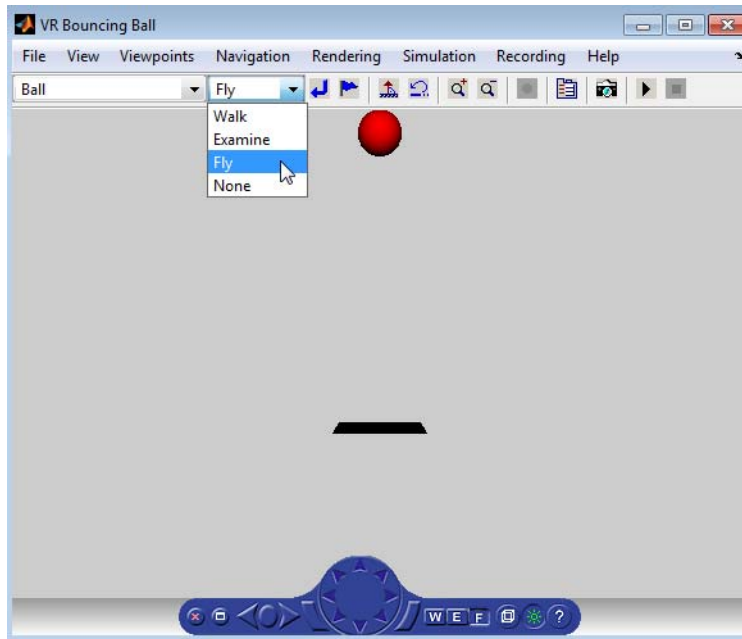


- From the navigation panel, click the **W**, **E**, or **F** buttons.
- On the keyboard, press **Shift+W**, **Shift+E**, **Shift+F**, or **Shift+N**.

**Navigation zones** — You can view the navigation zones for a scene through 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.



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 the navigation zones on 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's location. For more information, see “Example of How Sensors Affect Mouse Navigation” on page 7-18.

### Changing 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 VRML file. For further details, see the VRML97 standard.

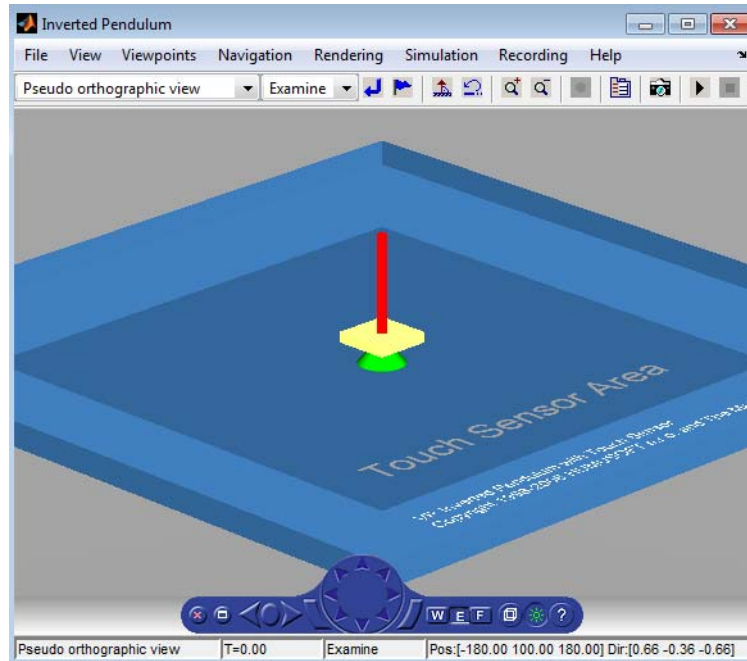
### **Example of How Sensors Affect Mouse Navigation**

1 At the MATLAB command prompt, type

```
vrpend
```

The Inverted Pendulum example starts, and the viewer displays the following 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 always 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.

**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.

**Simulink 3D Animation Viewer Keyboard Navigation**

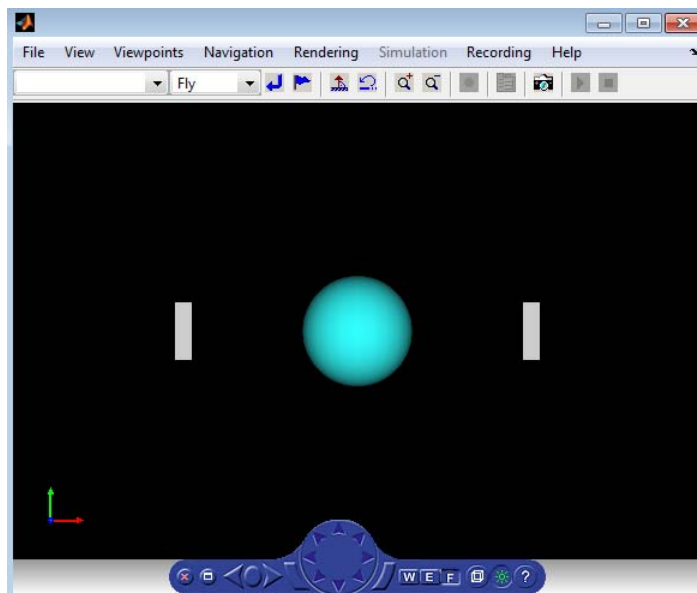
<b>Keyboard Command</b>	<b>Navigation Function</b>
<b>Backspace</b>	Undo move.
<b>Ctrl+r</b>	Start or stop recording.
<b>Ctrl+i</b>	Capture frame.
<b>Ctrl+t</b>	Start or stop simulation.
<b>F9</b>	Straighten up and make the camera stand on the horizontal plane of its local coordinates.
<b>+/-</b>	Zoom in and out.
<b>F6</b>	Toggle the headlight on and off.
<b>F7</b>	Toggle the navigation zones on and off.
<b>F5</b>	Toggle the wireframe option on and off.
<b>F8</b>	Toggle the antialiasing option on and 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>	Camera is bound/unbound 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.

## Simulink 3D Animation Viewer Keyboard Navigation (Continued)

Keyboard Command	Navigation Function
Up/Down Arrow	Pan the camera up and down.
Left/Right Arrow, Shift+Left/Right Arrow	Pan the camera right and left.
Alt+Up/Down Arrow	Slide up and down.
Alt+Left/Right Arrow	Slide left and right.
Ctrl+Left/Right/Up/Down Arrow	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.
Shift+Alt+Left/Right Arrow	Tilt the camera right and left.

## Display a Coordinate Axes Triad

To help you visualize changes in the orientation (coordinate axes) of nodes in a virtual world, you can 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 *one* of the following 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**.

### Define File Name Tokens

To control frame captures of a virtual scene, or to record animations into files, in the Simulink 3D Animation viewer, from the **Recording** menu, use the

Capture and Recording Parameters dialog box. The Simulink 3D Animation software supports a variety of file naming formats using file tokens. This topic describes the Simulink 3D Animation file tokens.

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

```
%f_anim_%n.<extension>
```

This format creates a unique file name each time you capture a frame or record the animation. %f and %n are tokens. %f is replaced with the name of the virtual world associated with the model and %n is a number that increments each time you record a simulation for the same virtual world. If you do not change the default file name, for example, if the name of the virtual world file is `vrplanets` and you record a simulation for the first time, the animation file is

```
vrplanets_anim_1.wr1
```

If you record the simulation a second time, the animation file name is `vrplanets_anim_2.wr1`. In the case of frame captures, capturing another frame of the scene increments the number.

You can use a number of 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 those 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 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 7-24 for a summary of the file name tokens.

Once you understand frame capture and recording file tokens, you can continue to the following topics:

- “Capture Frames” on page 7-25 — Describes how to create frame captures of a virtual scene
- “Animation Recording File Formats” on page 7-27— Describes how to configure animation recording parameters

## File Name Tokens

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

Token	Description
%d	The full path to the world VRML 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 %d/animdir/%f_anim_%n.avi saves the animation in the animdir subfolder of the folder containing the virtual world VRML file. It creates the animdir subfolder if one does not exist. This token is most helpful if you want to ensure that the virtual world file and animation file are in the same folder.
%D	The current day in the month replaces this token in the file name string. For example, the format %f_anim_%D.wrl saves the animation to vrplanets_anim_29.wrl for the 29th day of the month.
%f	The virtual world file name replaces this token in the file name string. For example, the format %f_anim_%D.wrl saves the animation to vrplanets_anim_29.wrl.
%h	The current hour replaces this token in the file name string. For example, the format %f_anim_%h.wrl saves the animation to vrplanets_anim_14.wrl for any time between 14:00 and 15:00.
%m	The current minute replaces this token in the file name string. For example, the format %f_anim_%h%m.wrl saves the animation to vrplanets_anim_1434.wrl for a start record time of 14:34.

Token	Description
%M	The current month replaces this token in the file name string. For example, the format %f_anim_%M.wrl saves the animation to vrplanets_anim_4.wrl for a start record time in April.
%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.
%n	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 %f_anim_%n.wrl saves the animation to vrplanets_anim_1.wrl on the first run, vrplanets_anim_2.wrl on the second run, and so forth.
%Y	The current four-digit year replaces this token in the file name string. For example, the format %f_anim_%Y.wrl saves the animation to vrplanets_anim_2005.wrl for the year 2005.

## 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, and so forth.

This topic describes how to configure and capture a frame, using the vrplanets example as the example. It assumes that you have read the topic “Define File Name Tokens” on page 7-22 about file names.

- “Configuring Frame Capture Parameters” on page 7-26 — Describes how to configure frame capture file formats
- “Capturing a Frame” on page 7-27 — Describes how to capture frames

### Configuring Frame Capture Parameters

- 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**.

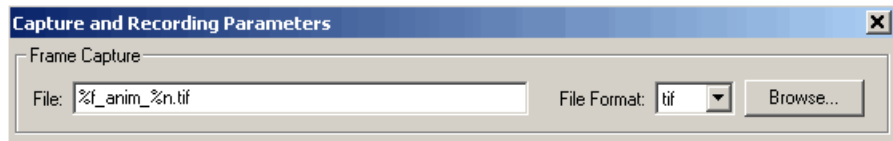
The Capture and Recording Parameters dialog box is displayed.

- 3 Find the **Frame Capture** section of the dialog. This is located at the top of the dialog.

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 might want to turn off this panel for a cleaner view of the virtual scene. Choose **View > Navigation Panel > None**.

You can reenble the Navigation Panel (for example, choose **View > Navigation Panel > Halfbar**) after you finish recording the `.tif` file.

- 7 Click **OK**.



You can now capture frames of a virtual scene. 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.

## Capturing a Frame

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. See “Configuring Frame Capture Parameters” on page 7-26 if you have not defined frame capture files.

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'))
```

## Animation Recording File Formats

The Simulink 3D Animation product allows you to record animations of virtual scenes controlled from Simulink or MATLAB. You can later play these animations offline (in other words, without the Simulink 3D Animation viewer). You might want to generate animation files for presentations, or to distribute or archive simulation results.

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

- 3-D VRML file — The software traces object movements and saves that data into a VRML file using VRML97 standard interpolators. These files allow you to observe animated virtual scenes in a virtual reality environment. 3-D VRML files typically use much less disk space than `.avi` files. The Simulink 3D Animation software does not save any navigation movements you make while recording the animation.

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

If you distribute the VRML animation files, also be sure to distribute all the inlined object and texture files referenced in the original VRML world file.

See the following topics:

- “Record Files in the VRML Format” on page 7-28 — Describes how to configure the record simulation parameters to create 3-D format animation files.
- “Record Files in the Audio Video Interleave (AVI) Format” on page 7-29 — Describes how to configure the record simulation parameters to create 2-D format animation files.
- “Schedule Files for Recording” on page 7-32 — Describes how to schedule record simulation operations to occur automatically.

### Record Files in the VRML Format

The following procedure describes how to set up recording parameters to create a 3-D VRML format file from a Simulink model execution. This procedure uses the `vrplanets` example. It describes how to create an animation file name with the default name format. See “Define File Name Tokens” on page 7-22 to save files to other file names. You can start the simulation before setting up the recording. This topic assumes that the model is not currently running.

- 1 Type the example’s name in the MATLAB Command Window. For example:

```
vrplanets
```

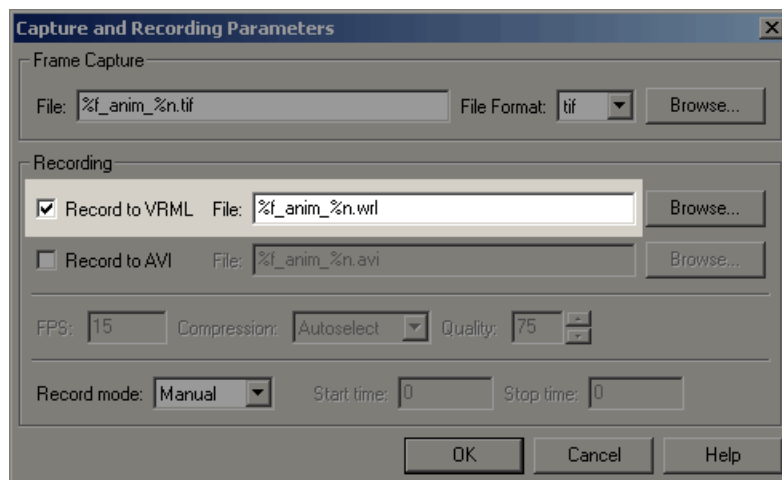
The Simulink model is displayed. Also, by default, the Simulink 3D Animation viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the VR Sink block in the Simulink model.

- 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 VRML** file check box.

The **File** text field becomes active and the default file name, `%f_anim_%n.wrl`, appears in the text field.



- 5 Click **OK**.

After you define an animation file, you can manually record simulations. See “Start and Stop Animation Recording” on page 7-35. If you want to record simulations on a schedule, see “Schedule Files for Recording” on page 7-32.

## Record Files in the Audio Video Interleave (AVI) Format

The following procedure describes how to set up recording parameters to create a 2-D AVI format file from a Simulink model execution. This procedure uses the `vrplanets` example. It describes how to create an animation file name with the default name format. See “Define File Name Tokens” on page 7-22 to save files to other file names. You can start the simulation before

setting up the recording. This topic assumes that the model is not currently running.

- 1 Type the example's 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.

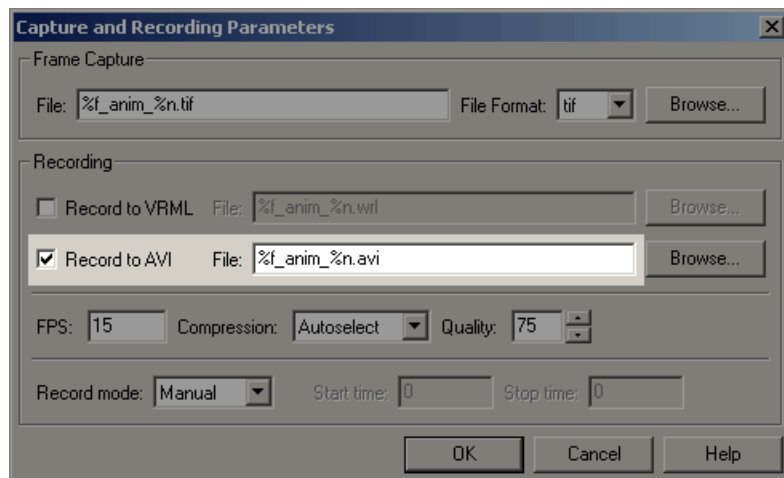
- 2 From the **Recording** menu, choose **Capture and Recording Parameters**.

The Capture and Recording Parameters dialog box is displayed.

- 3 Find the **Recording** section of the dialog box. This is located under the Frame Capture dialog box.

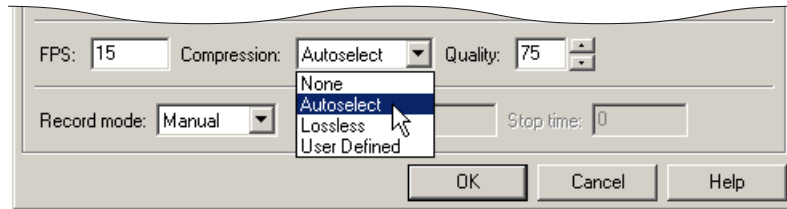
- 4 Select the **Record to AVI** file check box.

The **File** text field and Compression selection area become active, and the default file name, `%f_anim_%n.avi`, appears in the text field.



- 5 Set the **FPS** to an appropriate value.

- 6** From the **Compression** list, select a compression method for the .avi file. Because .avi files can become large, you might want to 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 between 0 and 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 between 0 and 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. You need to 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 might want to 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**.

---

**Note** The **FPS** parameter controls only the playback speed of the recorded AVI file and has nothing to do with the timing of the simulation. The sample time of the VR Sink block controls how often the simulation is recorded to a file.

For example, to record a Simulink simulation with 25 frames per second (of the simulation time), set **Sample time** in the VR Sink block to be 0.04. In that situation, if you want to create an AVI file where 1 second of simulation time corresponds to 1 second of AVI file playback time, set the **FPS** parameter to 25.

---

After you define an animation file, you can record animations. See “Start and Stop Animation Recording” on page 7-35. If you want to record animations on a schedule, see “Schedule Files for Recording” on page 7-32.

## 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. See “Record Files in the VRML Format” on page 7-28 or “Record Files in the Audio Video Interleave (AVI) Format” on page 7-29 for further details.

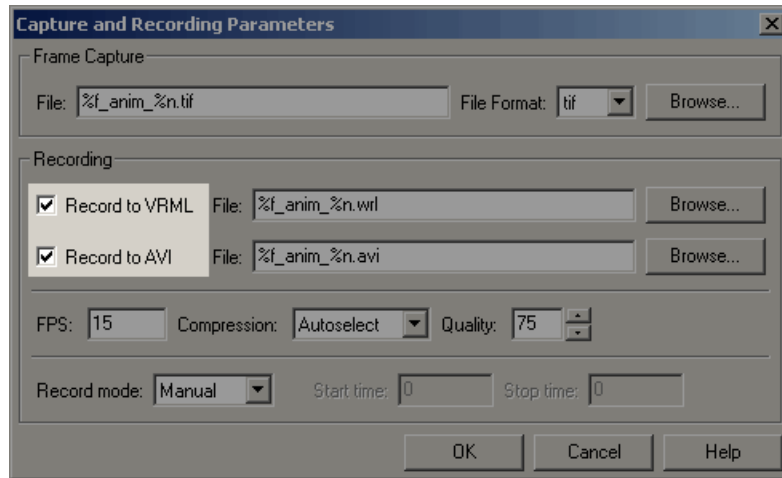
- 1 If the `vrplanets` example is not already open, type the example’s name in the MATLAB Command Window. For example:

```
vrplanets
```

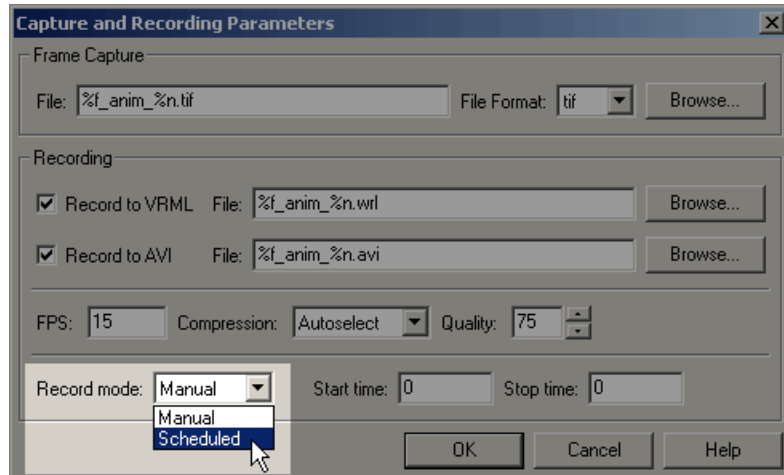
The Simulink model is displayed. Also, by default, the Simulink 3D Animation viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the VR Sink block in the Simulink model.

- 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. Note that the **Record mode** list is enabled only if you also select either or both of the **Record to VRML** 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. This allows 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 “Starting and Stopping Simulations” on page 7-13.



---

**Note** You can override the recording schedule by starting or stopping the recording interactively.

---

## 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 into which the animation is to be recorded. See “Animation Recording File Formats” on page 7-27 if you have not defined animation files.

- 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.

---

## View Animation Files

This topic assumes that you have a VRML or AVI animation file that you want to view. If you do not have an animation file, see “Record Files in the VRML Format” on page 7-28 or “Record Files in the Audio Video Interleave (AVI) Format” on page 7-29 for descriptions on how to create one.

### Viewing VRML Files

At the MATLAB window, type `vrplay(filename)`, where `filename` is the name of your VRML file. This opens the Simulink 3D Animation animation

player and your file. Using the Simulink 3D Animation animation player GUI, you can control the playback of your file.

As an example, play the animation file based on the `vrplanets_anim_1.wrl` example by running `vrplay('vrplanets_anim_1.wrl')`.

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

### To View VRML Files

**1** Change folder to the one that contains the VRML animation file.

**2** You can view the file in one of the following ways:

- Double-click on the VRML file. A VRML-enabled Web browser opens with the animation running. To view the resulting animation file, you must have a VRML-enabled Web browser installed on your system. Also, ensure that the `.wrl` extension is associated with the Blaxxun Contact Web browser.
- At the MATLAB window, type

```
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 to `'freerun'` directs the viewer to advance its time independent of the MATLAB software.

**3** If using `vrview`, to stop the animation, type

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

Alternatively, to close the viewer and delete the world, you can 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);
```

## To View AVI Files

- 1 Change folder to the one that contains the AVI animation file.
- 2 Double-click that file.

The program associated with .avi files in your system (for example, Windows 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.

## Viewpoints

You or visitors to a virtual world navigate through the virtual world environment using the Simulink 3D Animation viewer navigation methods Walk, Examine, Fly, and None. In addition to these navigation methods, a virtual world creator can set up points of interest, known as viewpoints, in the virtual world. You can use viewpoints to guide visitors through the virtual world and to emphasize important points.

When a visitor first enters a virtual world, he or she is defaulted to the default viewpoint. This is the first Viewpoint node in the virtual world file. Define the virtual world default viewpoint carefully; it should be the most interesting entry point to the virtual world.

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.

You can define a viewpoint to be either static or dynamic.

- Static -- Created typically at the top level of the virtual world object hierarchy. You can also create these viewpoints as children of static objects (Transforms).

- Dynamic -- Created as children of moving objects (objects controlled from MATLAB or Simulink) or linked to moving objects with the VRML ROUTE mechanism. Dynamic viewpoints allow you to create interesting views such as from the driver's seat at a racing course.

For more information about working with viewpoints, see:

- “Navigate Through Viewpoints” on page 7-38
- “Reset Viewpoints” on page 7-41
- “Define Viewpoints” on page 7-41

### Navigate Through Viewpoints

You can navigate through a virtual scene's 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 sequentially move up and down the list of existing viewpoints. 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 sequentially move up and down the list of existing viewpoints.
- 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 7-41. Resetting the viewpoint is useful when you have been moving about the scene and need to reorient yourself.

This topic illustrates viewpoints using the vrplanets example.

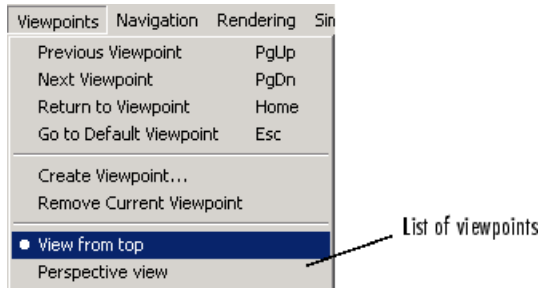
- 1 Select a Simulink 3D Animation example and type that example's name in the MATLAB command window. For example:

```
vrplanets
```

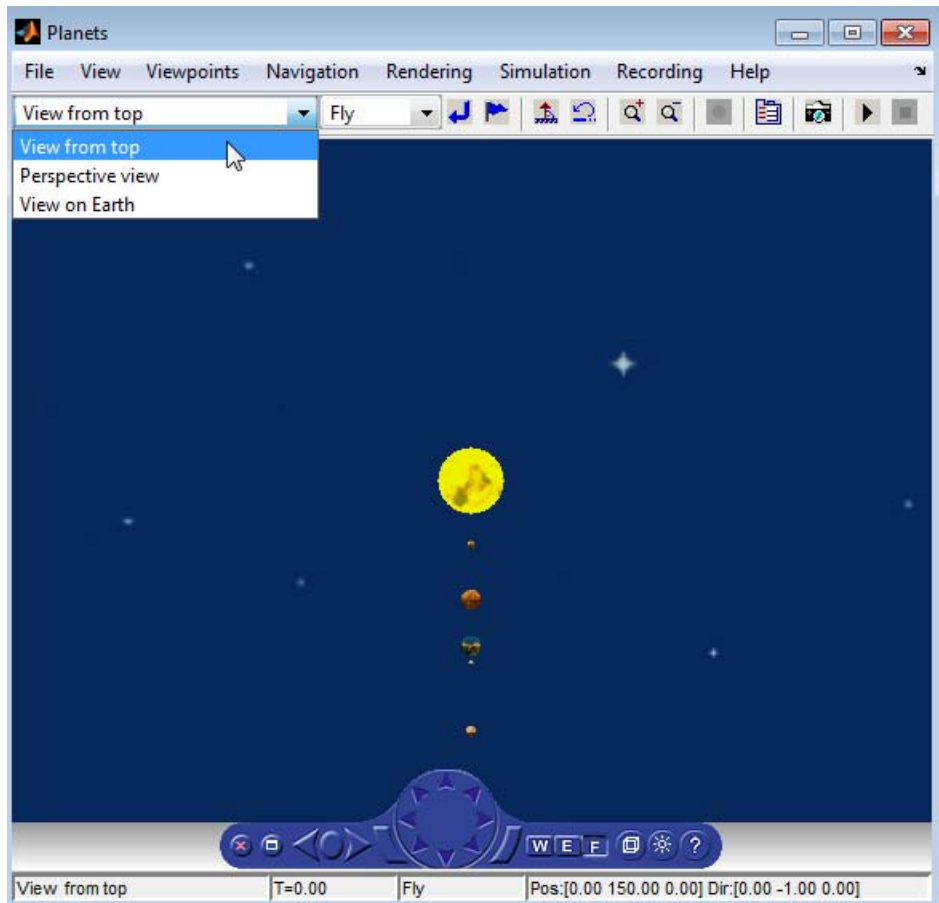
The Simulink model is displayed. By default, the Simulink 3D Animation viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the VR Sink block in the Simulink model.

- 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 Select the drop-down list on the leftmost side of the toolbar to see the list of existing viewpoints from 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 virtual world's default viewpoint
- Create and remove viewpoints
- Navigate to an existing viewpoint

## Reset Viewpoints

You can reset your position in a scene to 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:
  - Press the **Esc** key to return to the default viewpoint of the virtual world.
  - Press the **Home** key to return to the initial position of the current viewpoint.

## 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 topic 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:

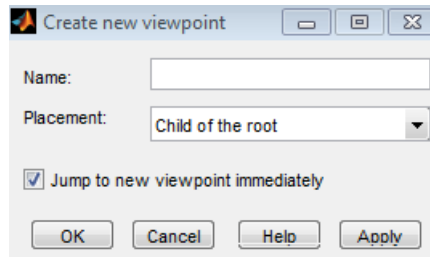
```
vrplanets
```

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** menu.
- 3 Choose **View on Earth**.
- 4 In the viewer window, navigate to a random position in the scene.
- 5 From the **Viewpoints** menu, choose **Create Viewpoint**. Alternatively, click **Create viewpoint** on the toolbar.

The Create Viewpoint dialog box is displayed.



- 6 In the **Name** box, enter a unique and descriptive name for the viewpoint.
- 7 The state of the **Placement** field 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 **Placement** field is grayed out. In this case, it is only meaningful to create the new viewpoint at the same top hierarchy level.

In this example, the **Placement** field is editable. Select **Child of the root** as the viewpoint type. This option makes the viewpoint a static one.

- 8 Select the **Jump to new viewpoint immediately** check box to make the new viewpoint become the current viewpoint for the viewer. If you do not select this check box, you still create a new viewpoint, but you remain bound to the current viewpoint, not to the new viewpoint.
- 9 Click **OK**.



- 10** From the **File** menu, click **Save As** to save the file with the new viewpoint. If you do not save the file, the new viewpoint will be lost during simulation.
- 11** From the **Simulation** menu, click **Start**. Observe the motion of the planets from the new, static viewpoint.
- 12** Stop the simulation.
- 13** Repeat steps 2 to 6.

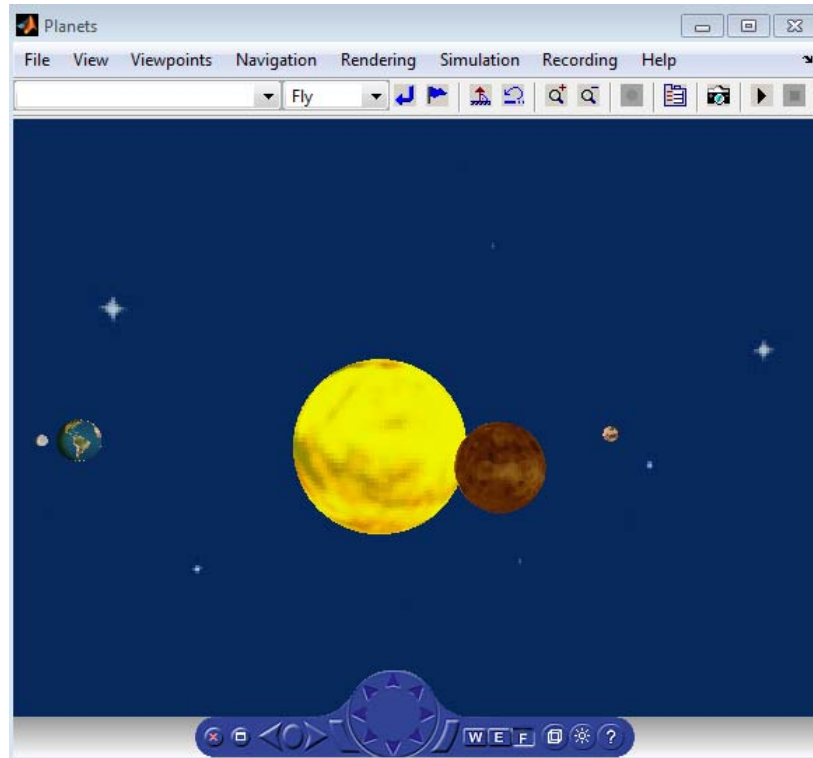
- 14** In the **Placement** field, select **Sibling** of the current viewpoint. This option creates a new viewpoint at the same level in the virtual world object hierarchy as the child of the parent transform of the current viewpoint. The local coordinate system of the parent transform defines the new 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) you first defined by navigating somewhere in the space from the current viewpoint (step 4).

---

**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 new viewpoint as a static one at the same top hierarchy level.

---

- 15** Select the **Jump to new viewpoint immediately** check box to make the new viewpoint become the current viewpoint for the viewer. If you do not select this check box, you still create a new viewpoint, but you remain bound to the current viewpoint, not to the new viewpoint.
- 16** Click **OK**.
- 17** From the **File** menu, choose **Save As** to save the file with the new viewpoint. If you do not save the file, the new viewpoint will be lost during simulation.
- 18** From the **Simulation** menu, choose **Start**. 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.

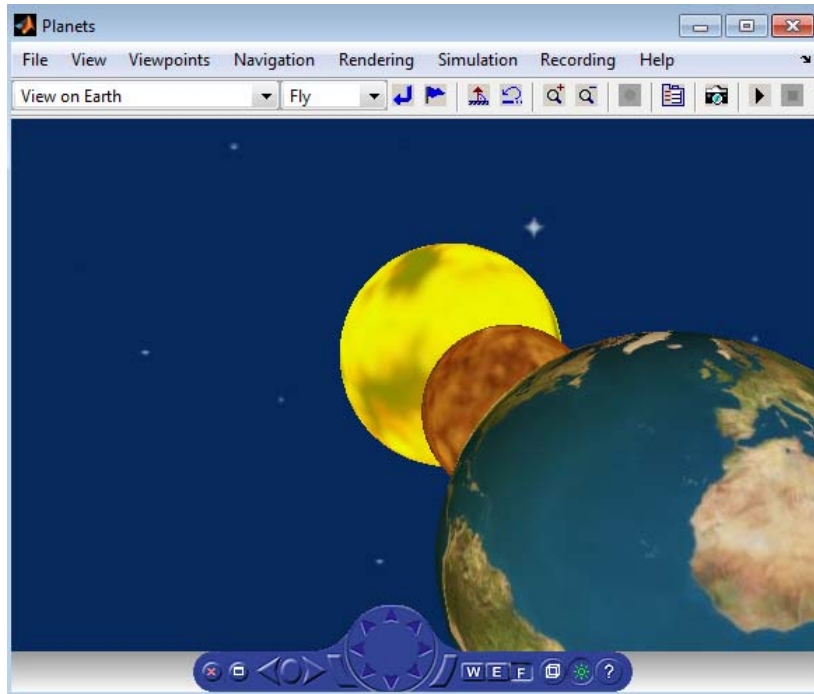


## Specify Rendering Techniques

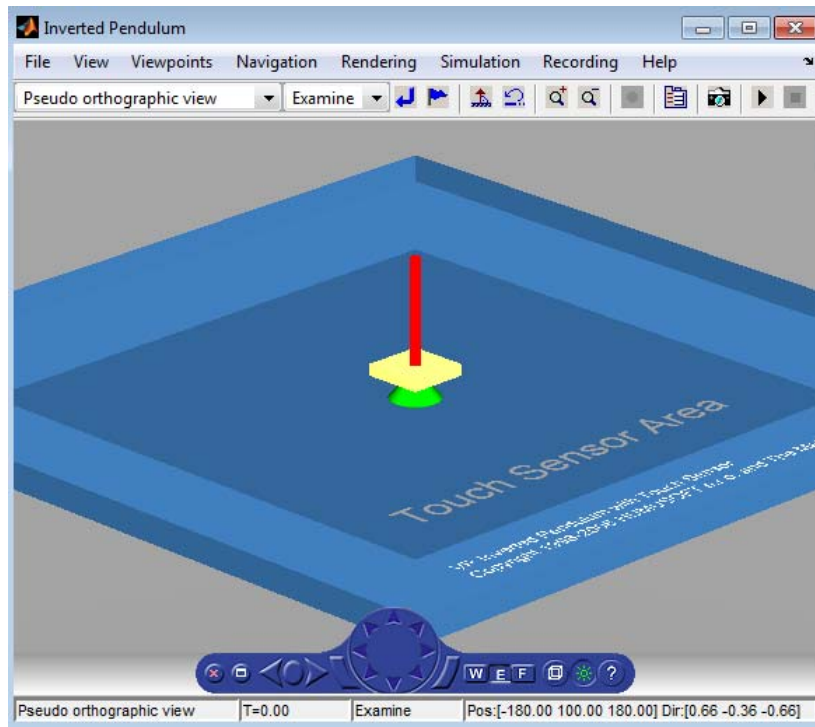
You can change the rendering of the scene through the controls on the navigation panel or options on the rendering menu. The `vrpend` and `vrplanets` examples are used to show the viewer's functionality.

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's 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.

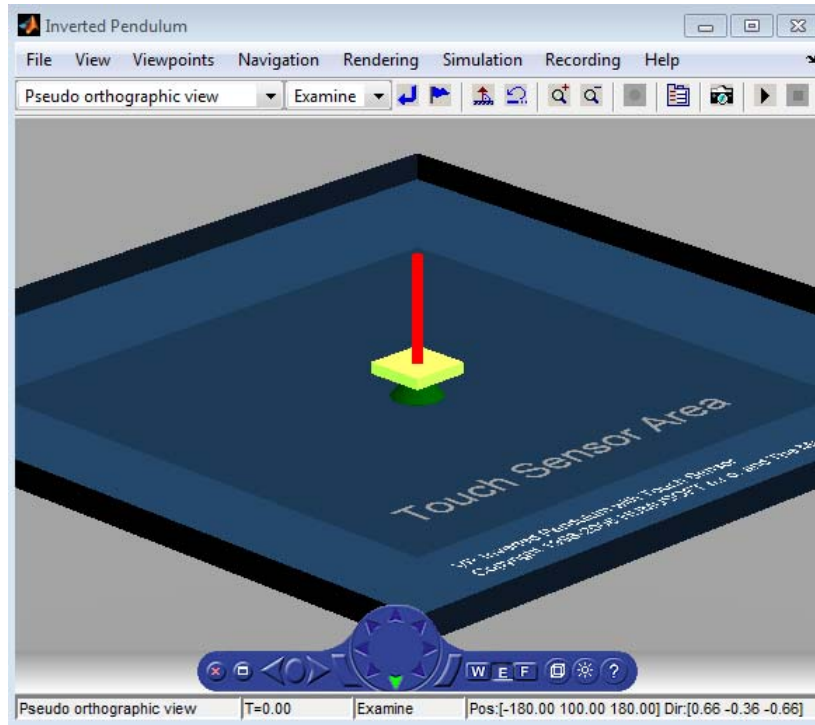
The following figure depicts the vrplanets example View on Earth viewpoint with **Antialiasing** on. To better display the effects of antialiasing, turn **Headlight** on. You can turn antialiasing on or off to observe the differences.



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. Consequently, 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.

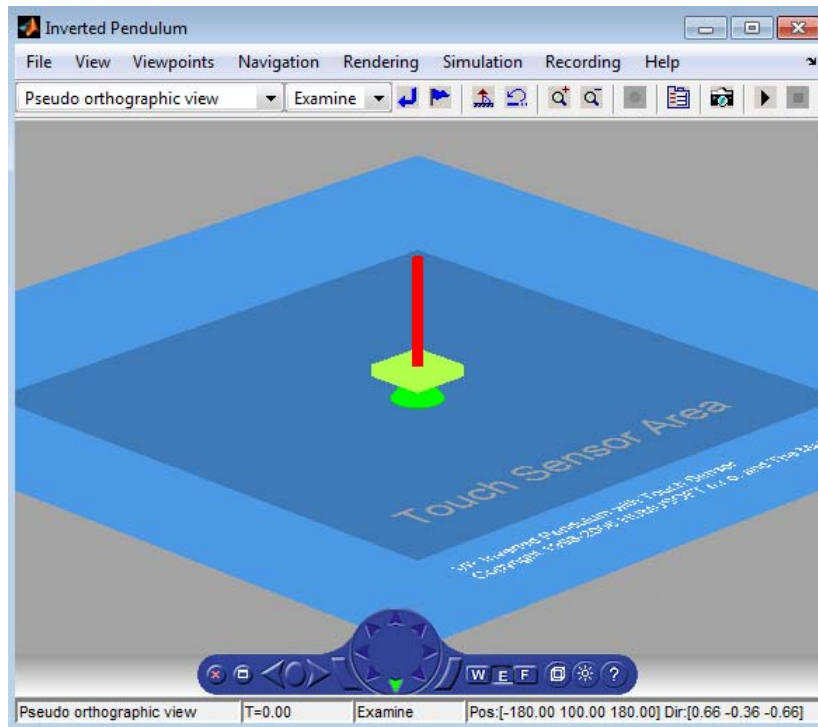


---

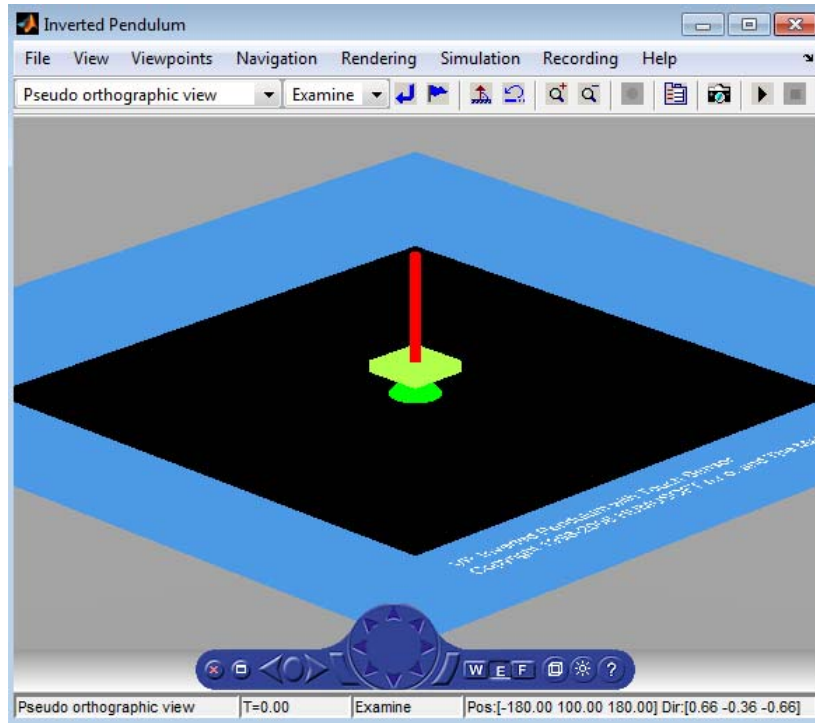
**Note** It is helpful to define enough lighting within the virtual scene so that it is lit regardless of the **Headlight** setting.

---

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 vrpnd example with **Lighting** off.

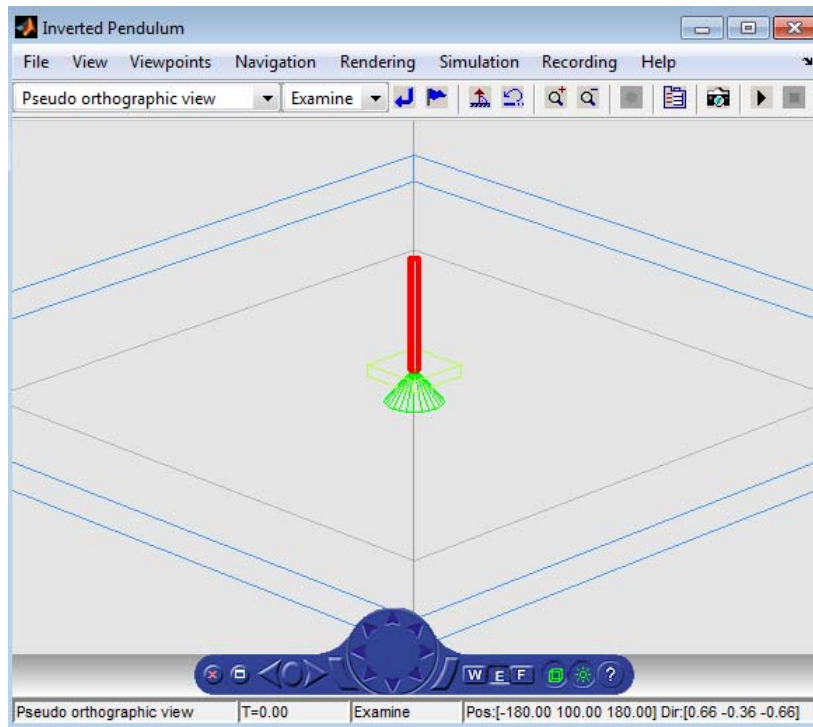


If **Transparency** is off, transparent objects are rendered as solid objects.

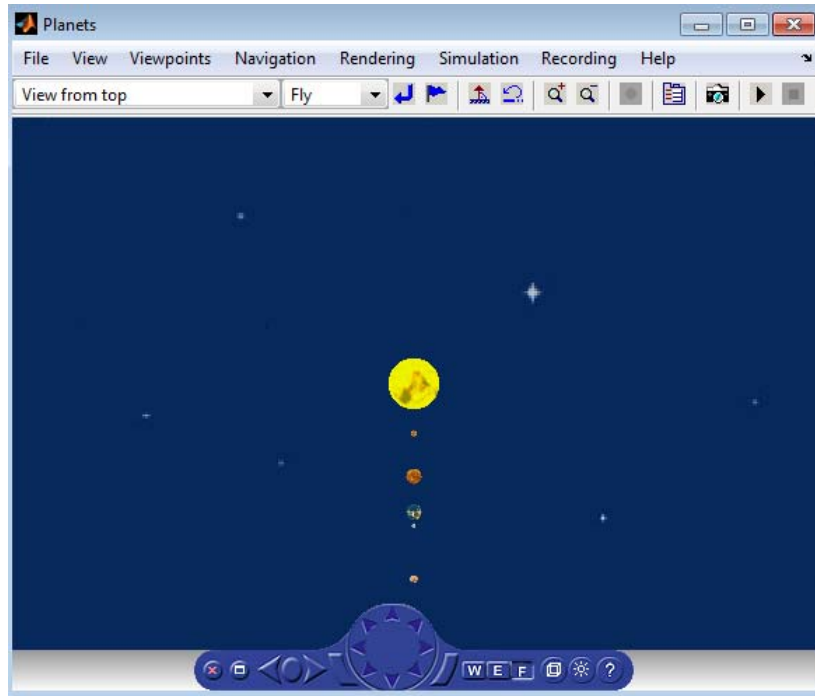


Turning **Wireframe** on changes the scene's objects from solid to wireframe rendering. The following is the vrend example with **Wireframe** on.

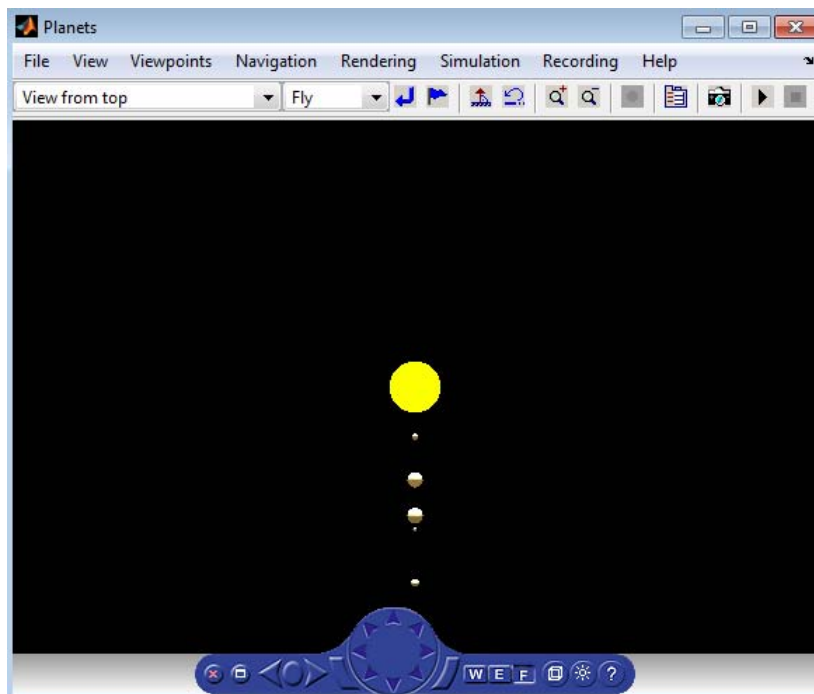




If **Textures** is on, objects have texture in the virtual scene. The following is the vrplanets example with **Textures** on.



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.

## Blaxxun Contact VRML Plug-In

### In this section...

“Blaxxun Contact Interface” on page 7-54

“Navigate Through Viewpoints” on page 7-55

“Control Menu” on page 7-55

“Navigate a Virtual World” on page 7-56

“Movement Modes” on page 7-57

“Blaxxun Contact Settings” on page 7-57

“Specify Rendering Techniques” on page 7-58

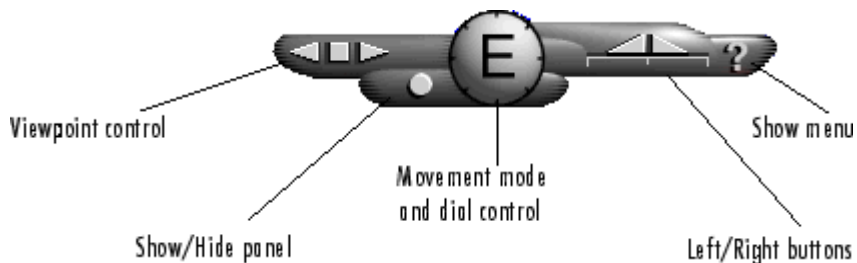
“Stereoscopic Vision” on page 7-58

“Using Blaxxun Contact on a Client Computer” on page 7-59

### Blaxxun Contact Interface

The Simulink 3D Animation product includes the Blaxxun Contact VRML plug-in. This is a VRML plug-in for either Internet Explorer or Netscape Navigator on a Windows platform. This section provides a quick overview of the functions and controls of the Blaxxun Contact VRML plug-in, and also describes full screen stereo support in Blaxxun.

When you open a VRML file with a Web browser, the Blaxxun Contact VRML plug-in is used to display a virtual scene. A control panel is located at the bottom of the scene.



## Navigate Through Viewpoints

Three buttons on the control panel control the viewpoint. The square button in the middle resets the current viewpoint to its initial position. This is the most useful viewpoint control button until you gain enough experience with the viewer to explore worlds using navigation. The keyboard shortcut for the square button is the **Esc** key.

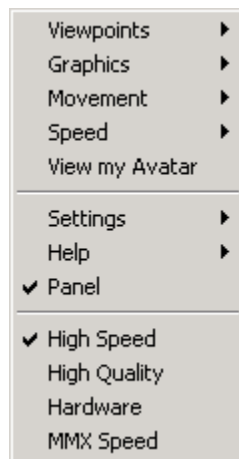
You use the other two triangular buttons to browse forward and backward through author-defined viewpoints of the virtual world. If the author does not define other viewpoints, these buttons are inactive. The keyboard shortcuts for the triangular buttons are the **Page Up** and **Page Down** keys.

## Control Menu

You use the control menu to review or select viewer settings and navigation methods. To access the control menu, use the following procedure:

- 1 On the Control Panel, click the question mark, or place your mouse pointer anywhere in the browser window, and then right-click.

If you selected Direct3D® with the Blaxxun Contact installation, a menu similar to the following appears:



- 2 From the menu, you can make changes to the navigational mode, graphic quality, and graphic speed.

Depending on the complexity of the virtual world and the required speed and rendering quality, you can choose the settings that best meet your needs.

Because the viewer's graphical performance strongly depends on several factors, you might want to experiment to find a reasonable compromise between the quality and speed for your system.

### Navigate a Virtual World

The dial control and left/right buttons give you direct access to the movement mode for walking through a virtual world. However, the movement behavior of your mouse pointer changes depending on the movement mode you select. When you select a different movement mode, clicking your left mouse button causes your viewpoint to move differently. Practice changing the movement mode and navigating through a virtual world until you get satisfactory results.

To select a movement mode, use the following procedure:

- 1 Place your mouse pointer over a virtual world, then right-click. A menu appears.
- 2 On the menu, point to **Movement**. A submenu appears.
- 3 Choose **Walk, Slide, Rotate, Examine, Fly, Pan, or Jump**.

A letter in the center of the dial indicates the current movement mode. For example, in the preceding illustration, the large E stands for Examine mode.

Initially, you should use Examine mode, which is for examining objects from various angles. You will find that the functions of the left/right button controls in Examine mode are the easiest for beginners to master.

## Movement Modes

The following table lists the movement modes.

Movement Mode	Description
Walk	Drag the mouse toward the top or the bottom of the screen to move forward or backward, and drag toward the left or right to turn left or right.
Slide	Drag the mouse to move up, down, left, or right within a plane that is perpendicular to your view.
Rotate	Press the left mouse button to select a rotation point within the scene. Then drag the mouse toward the top or bottom to move forward or back, or drag the mouse left or right to rotate around the fixed point.
Examine	Press the left mouse button to select a rotation point within the scene. Then drag the mouse up, down, left, or right to rotate the object.
Fly	Press the left mouse button to start flying. Drag the mouse toward the top or bottom to rise or sink, and drag left or right.
Pan	Drag the mouse toward the top or bottom of the scene to loop up and down, and drag left or right to turn left or right.
Jump	Place your mouse pointer over an object, then left-click. Your view moves to that point.

## Blaxxun Contact Settings

For PCs, the Simulink 3D Animation software includes the Blaxxun Contact VRML plug-in for Web browsers. The viewer allows you to select several working configurations, and its performance depends on several factors:

- The speed of your hardware
- System display driver settings

- Method of 3-D rendering
- Blaxxun Contact parameters
- The size of the window in which you display the 3-D visualization

You might want to test the various combinations possible on your system to find an optimal configuration for the best performance in 3-D visualization.

In Direct3D configuration, you can select the speed and quality on the fly from the top level of the menu. You can, depending on the system capabilities, select one of the options on the menu. For example, you can select High Speed, High Quality, Hardware Acceleration, and MMX Speed.

In the OpenGL configuration, you can set similar rendering properties. From the floating menu, choose **Settings**, and then choose **Preferences**.

### Specify Rendering Techniques

With respect to the 3-D rendering method, you can install Blaxxun Contact with two basic configurations using OpenGL and Direct3D drivers. You can tune the viewer performance by setting the parameters in the Settings-Preferences dialog box of the viewer floating menu, accessible by right-clicking when you are viewing a virtual scene.

### Stereoscopic Vision

Blaxxun Contact supports stereoscopic vision. If the graphic card and system driver enable full screen stereo mode, and if you have corresponding stereo vision hardware (such as stereoscopic shutter glasses), you can access this support. In full screen mode, no menus and other user interfaces are available to the user.

- To switch Blaxxun Contact to the full screen mode, press **F5**.
- To switch back to normal mode, press **Esc**.

If you have installed the appropriate stereo driver, Blaxxun Contact supports full screen stereo mode under Windows with most NVIDIA graphic cards. For details, refer to the card manufacturer documentation.



If you want to tune the full screen mode resolution or color depth.

- 1** In the Blaxxun Contact window, place your mouse pointer over a virtual world, then right-click.

A menu appears.

- 2** On the menu, point to **Settings**. A submenu appears.

- 3** Choose **Preferences**.

- 4** Tune the full screen mode resolution or color depth settings.

- 5** Click **OK** when done.

Note that your system configuration can switch to stereoscopic full screen mode only when using one of the Direct3D or OpenGL rendering engines. If you are unable to switch to full screen stereo mode, try to install Blaxxun Contact using another rendering engine. Typically, graphic card stereo drivers provide testing applications to confirm the functionality of stereoscopic modes.

## Using Blaxxun Contact on a Client Computer

The Simulink 3D Animation software uses a Simulink 3D Animation HTTP server for communication between a VRML-enabled Web browser and the MATLAB and Simulink environment. It generates the main Simulink 3D Animation HTML page with the list of currently available virtual worlds and sends VRML and other requested files and data to clients (VRML viewers).

The server is started when the software is loaded into the MATLAB interface. This happens whenever you use a Simulink 3D Animation block in a Simulink block diagram, or whenever you open a `vrworld` object in the MATLAB interface. The HTTP server is shut down when you close all Simulink models that contain Simulink 3D Animation blocks, or use the `vrclear` command.

When the HTTP server is running, your browser can see a list of available virtual worlds at the following address, where 8123 is the default port number:

`http://localhost:8123`

Remote users can connect to the following address, where 8123 is the default port number:

```
http://your_machine:8123
```

You can set the port number of the server in the Simulink 3D Animation Preferences dialog box from the Simulink interface, or use `vrsetpref` in the MATLAB Command Window.

Depending on the status of served `vrworld` objects, the list of available virtual worlds can be empty.

# Legacy Simulink 3D Animation Viewer

## In this section...

“Introduction” on page 7-61

“Starting the Legacy Viewer” on page 7-61

“Differences Between the Default and Legacy Viewer” on page 7-63

“Differences When Setting the DefaultViewer Property to 'internalv4'” on page 7-64

## Introduction

The Simulink 3D Animation viewer has a legacy viewer. This legacy viewer is the former default viewer. For details about the differences between the current viewer and the legacy viewer, see “Differences Between the Default and Legacy Viewer” on page 7-63.

## Starting the Legacy Viewer

By default, you use the default Simulink 3D Animation viewer to visualize virtual worlds. To configure your Simulink 3D Animation software to use the legacy viewer,

- 1 Determine your default viewer by typing

```
vrgetpref
```

The MATLAB Command Window displays

```
ans =
```

```

                                DataTypeBool: 'logical'
                                DataTypeInt32: 'double'
                                DataTypeFloat: 'double'
                                DefaultCanvasNavPanel: 'none'
                                DefaultCanvasUnits: 'normalized'
                                DefaultFigureAntialiasing: 'off'
                                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'
    DefaultFigureTransparency: 'on'
        DefaultFigureWireframe: 'off'
            DefaultViewer: 'internal'
    DefaultWorldRecord3DFileName: '%f_anim_%n.wrl'
        DefaultWorldRecordMode: 'manual'
    DefaultWorldRecordInterval: [0 0]
        DefaultWorldRemoteView: 'off'
        DefaultWorldTimeSource: 'external'
            Editor: [1x60 char]
            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 scenes. Any virtual scenes that you open are displayed in the viewer.

- 2** Change the default viewer to the legacy viewer by typing

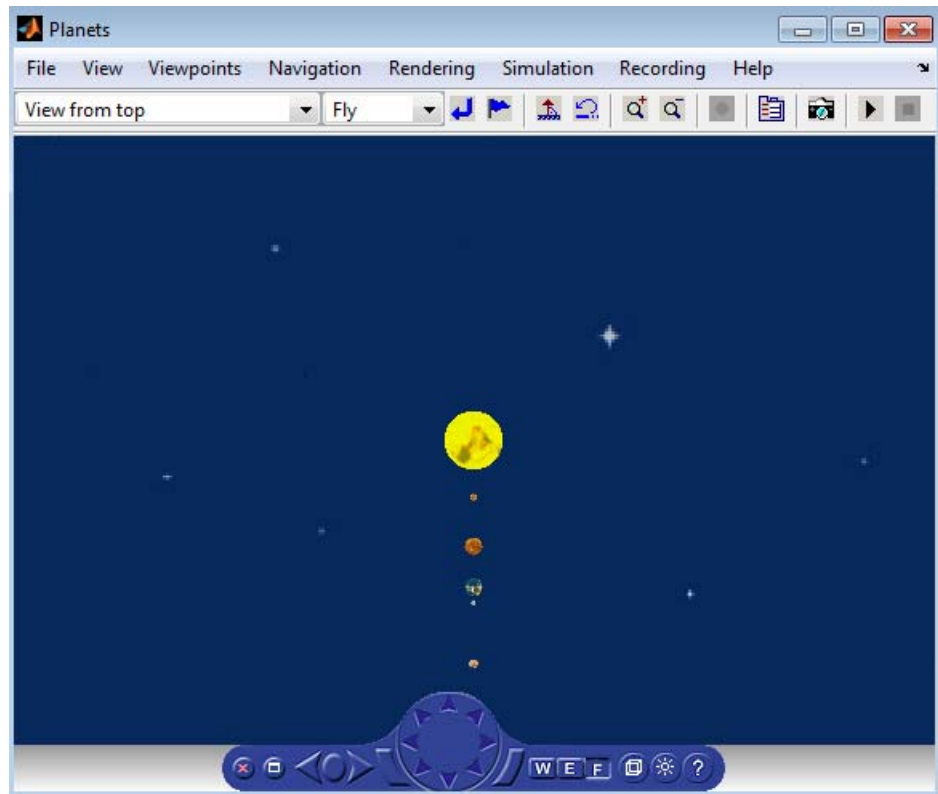
```
vrsetpref('DefaultViewer','internalv4')
```

The legacy viewer becomes the default viewer.

- 3** At the MATLAB command prompt, type

```
vrplanets
```

The Planets example is loaded and the virtual scene is displayed using the legacy viewer.



To revert to the original default viewer, type

```
vrsetpref('DefaultViewer','internal')
```

## Differences Between the Default and Legacy Viewer

The legacy viewer has the functionality and appearance of the default Simulink 3D Animation viewer:

- In the legacy viewer, you cannot dock the viewer window in the MATLAB window.

- In the legacy viewer, you cannot right-click in the viewer window to display a context menu that contains the viewer commands.
- In the legacy viewer, you cannot combine multiple figures in several tiles of a MATLAB figure.

The legacy and default viewer have minor visual differences, such as:

- In the legacy viewer, circles denote selected menu command options. In the default viewer, checks denote selected menu command options.
- In the legacy viewer, the status bar has no border. In the default viewer, the status bar has a border.

### **Differences When Setting the DefaultViewer Property to 'internalv4'**

Setting the DefaultViewer property to 'internalv4' also affects the behavior of the `vrplay` function. 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 as a MATLAB figure.

If you set the DefaultViewer property to 'internalv4', the **File > New Window** command creates the new player window as in previous releases.

# 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.

- “What Is Orbisnap?” on page 8-2
- “Orbisnap Installation” on page 8-3
- “Using Orbisnap” on page 8-5
- “Orbisnap Interface” on page 8-10
- “Start Orbisnap” on page 8-17

## What Is Orbisnap?

The Simulink 3D Animation product includes Orbisnap. Orbisnap is a free, optional, stand-alone VRML97 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 might want to 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, if you want to visualize a virtual world active in a Simulink 3D Animation session that is running on a computer in another part of the building, or across the network. This functionality allows you to remotely view a simulation, but not control it.
- View and navigate, but not simulate, a VRML world. You can navigate, render, and otherwise visualize a VRML world without simulating it.

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.



# Orbisnap Installation

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 8-4

## 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, but you might want to 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. It can also run on Solaris hardware running UNIX, Apple Power Macintosh hardware running Mac OS X, and Hewlett-Packard™ hardware running HP-UX. See the following page on the MathWorks Web site:

<http://www.mathworks.com/products/matlab/requirements.html>

## Copying Orbisnap to Another Location

Orbisnap runs independently of the MATLAB and Simulink 3D Animation products. This 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 GUI such as Windows Explorer, create a folder into which you can copy Orbisnap.

- 2 Copy all the files in the Orbisnap folder and its subdirectories. These files are likely located in the Simulink 3D Animation orbisnap folder, for example, *matlabroot\toolbox\s13d\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*.

## Using Orbisnap

### In this section...

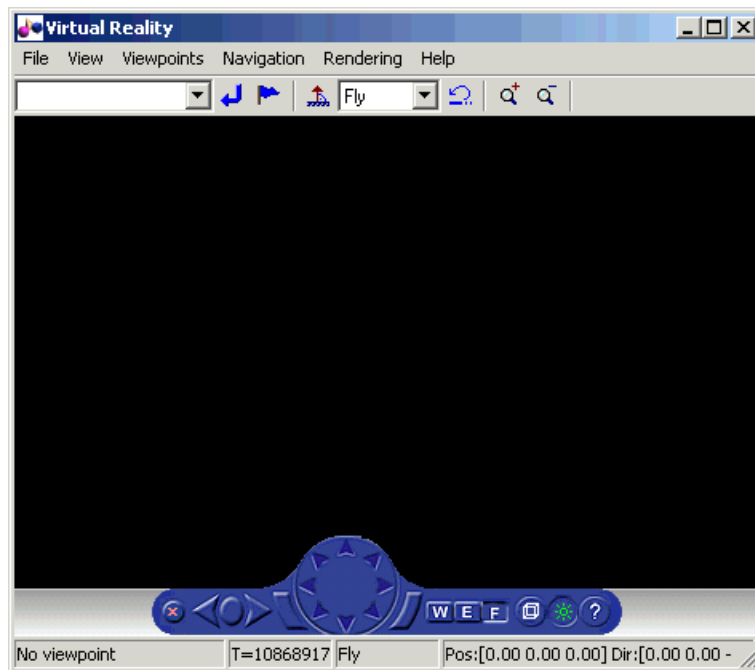
“Overview” on page 8-5

“View WRL Animations or Virtual Worlds” on page 8-6

“View Virtual Worlds Remotely” on page 8-7

### Overview

Orbisnap looks like the following:



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

## View WRL Animations or Virtual Worlds

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 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 8-10 for an overview of the Orbisnap interface. See “Start Orbisnap” on page 8-17 for a description of the Orbisnap command-line options.

## View Virtual Worlds Remotely

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, you must 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, and vice versa.
- 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 might slow when you connect Orbisnap remotely to the Simulink 3D Animation server.

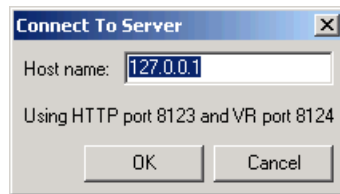
**1** Start Orbisnap. For example, in Windows, double-click orbisnap.bat in *matlabroot\toolbox\sl3d\orbisnap\bin*.

This 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 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.

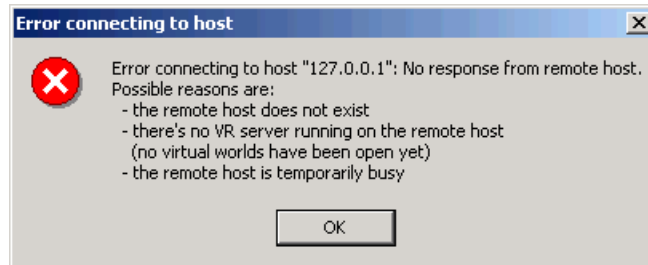


The Choose a world dialog is displayed. This dialog lists all the virtual worlds that are currently active on the Simulink 3D Animation server.



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 remotely access it.



**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 8-10 for a description of the Orbisnap interface. See “Start Orbisnap” on page 8-17 for a description of the Orbisnap command-line options.

## Orbisnap Interface

### In this section...

“Menu Bar” on page 8-10

“Toolbar” on page 8-11

“Navigation Panel” on page 8-11

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. For further details on using this interface, see “Orbisnap Standalone Viewer”.

### Menu Bar

The Orbisnap menu bar has the following menus:





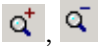
- **File** — General file operation options, including,
  - **Open** — Invokes a browser that you can use to browse to the virtual world you want to visualize.
  - **Connect to server** -- Allows you to connect to a Simulink 3D Animation server. Enter the IP address or hostname of the host computer running 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. Note that 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 Virtual World” on page 8-13 for a description of how to use navigation zones).
- **Navigation Panel** — Controls the display of the navigation panel, including toggling it.
- **Zoom In/Out** — Zooms in or out of the world view.
- **Normal (100%)** — Returns the zoom to normal (initial viewpoint setting).
- **Viewpoints** — Manages the virtual world viewpoints.
- **Navigation** — Manages scene navigation.
- **Rendering** — Manages scene rendering.
- **Help** — Displays the Help browser for Orbisnap.

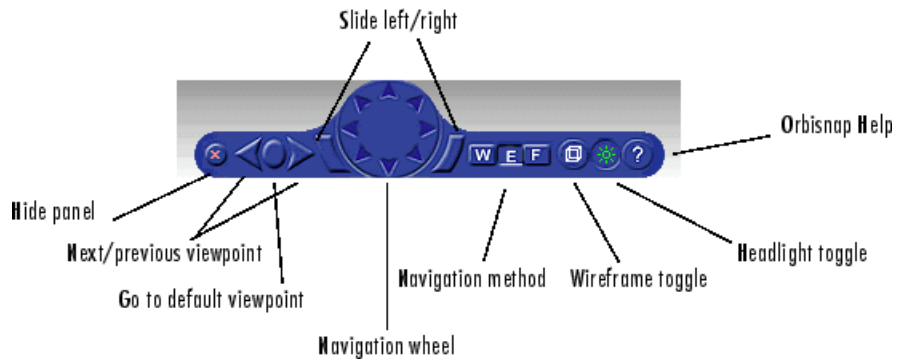
## Toolbar

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. These controls include



- **Hide panel** — Toggles the navigation panel.
- **Next/previous viewpoint** — Toggles through the list of viewpoints.
- **Return to default viewpoint** — Returns focus to original default viewpoint.
- **Slide left/right** — Slides the view left or right.
- **Navigation wheel** — Moves view in one of eight directions.
- **Navigation method** — Manages scene navigation.
- **Wireframe toggle** — Toggles scene wireframe rendering.
- **Headlight toggle** — Toggles camera headlight.
- **Help** — Invokes the Orbisnap online help.

## Navigate Virtual World

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. Alternatively, you can click the Straighten Up control from the toolbar or press **F9** on the keyboard. 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 on page 8-14.
- 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 the navigation zones on 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's 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. Note that the letters presented do not need to be capitalized to perform their intended function.

## Orbisnap Keyboard Navigation

<b>Keyboard Command</b>	<b>Navigation Function</b>
<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.

**Orbisnap Keyboard Navigation (Continued)**

<b>Keyboard Command</b>	<b>Navigation Function</b>
<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.

## 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 filename.

-c hostname — (Optional) Orbisnap starts and connects to the Simulink 3D Animation server located 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 filename (for example, vrbounce.wr1). If the string is not a valid vrworld filename, 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.



# Block Reference

---

Control Input Devices (p. 9-2)

Utilities (p. 9-3)

Virtual Worlds (p. 9-4)

VRML-Related Signals (p. 9-5)

Process input from devices

Vector and matrix calculations

Virtual World utilities

VRML signal utilities

## **Control Input Devices**

Joystick Input

Process input from asynchronous joystick device

Space Mouse Input

Process input from space mouse device

## Utilities

Cross Product	Cross product of two 3-D vectors
Normalize Vector	Unit vector parallel to input vector
Rotation Between 2 Vectors	VRML rotation between two 3-D vectors
Rotation Matrix to VRML Rotation	Convert rotation matrix into representation used in VRML
Viewpoint Direction to VRML Orientation	Convert viewpoint direction to VRML orientation

## Virtual Worlds

VR Sink	Write data from Simulink model to virtual world
VR Source	Read data from virtual world to Simulink model
VR To Video	Write data from Simulink model to virtual world (video output port enabled)
VR Tracer	Trace trajectory of object in associated virtual scene

## VRML-Related Signals

VR Placeholder	Send unspecified value to Simulink 3D Animation block
VR Signal Expander	Expand input vectors into fully qualified VRML field vectors
VR Text Output	Allows display of Simulink signal values as text in VRML scene



# Blocks — Alphabetical List

---

# Cross Product

---

**Purpose** Cross product of two 3-D vectors

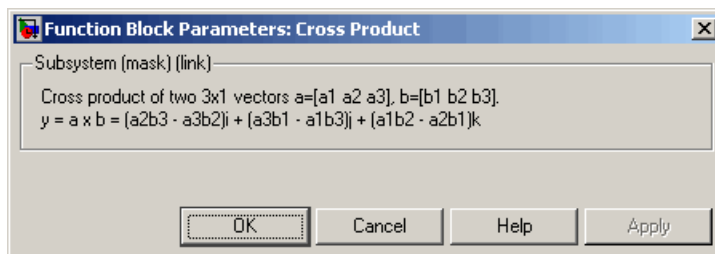
**Library** Simulink 3D Animation

**Description** Takes two 3-by-1 vectors as input and returns their cross product.



Cross Product

## Block Parameters Dialog Box





## Purpose

Process input from asynchronous joystick device

## Library

Simulink 3D Animation

## Description



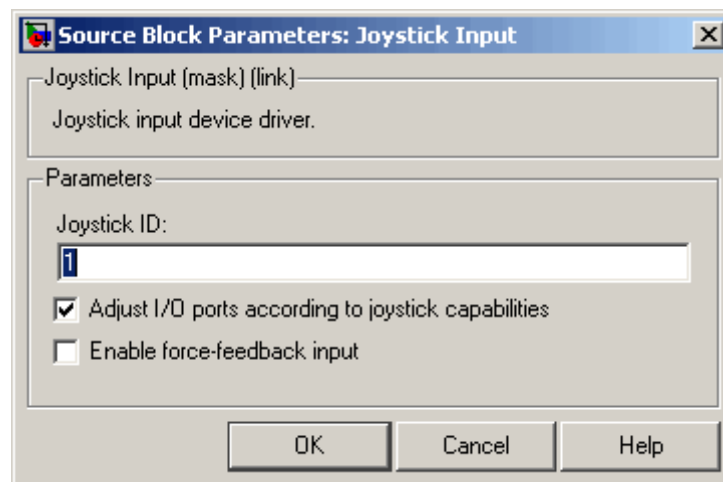
Joystick Input

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.

When building a model using Real-Time Windows Target, use the RTWin Joystick Input driver instead of the Joystick Input block.



## Block Parameters Dialog Box

**Joystick ID** — 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.

**Adjust I/O ports according to joystick capabilities** — If you select this check box, the Simulink 3D Animation software dynamically

# Joystick Input

---

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 select the **Enable force-feedback input** check box. The block ports do not have the full widths provided by the Windows Game Controllers interface.

**Enable force-feedback input** — 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** — Depending on the **Adjust I/O ports according to joystick capabilities** check box setting, the output ports change to correspond to the actual capabilities of the connected joystick. Or, on Windows platforms, the output ports have fixed maximum width provided by the system Game Controllers interface.

Output Port	Value	Description
Axes	Vector of doubles in the range < -1; 1 >	Outputs correspond to the current position of the joystick in the given axis. Values are normalized to range from -1 to 1.
Buttons	Vector of doubles 0 — Button released 1 — Button pressed	Outputs correspond to the current status of joystick buttons.
Point of view	-1 — Selector inactive <0; 360> — The angle of the POV selector, in degrees	Output corresponds to the current status of the joystick point-of-view selector.

Input Port	Value	Description
Force	Vector of doubles in the range < -1; 1 >	<p>Port active only for force-feedback devices. Inputs correspond to the force to be applied in the given axis.</p> <p>Usually, not all of the device axes have force-feedback. The size of the Force vector is then smaller than the Axes vector size.</p>

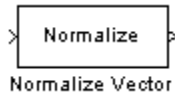
# Normalize Vector

---

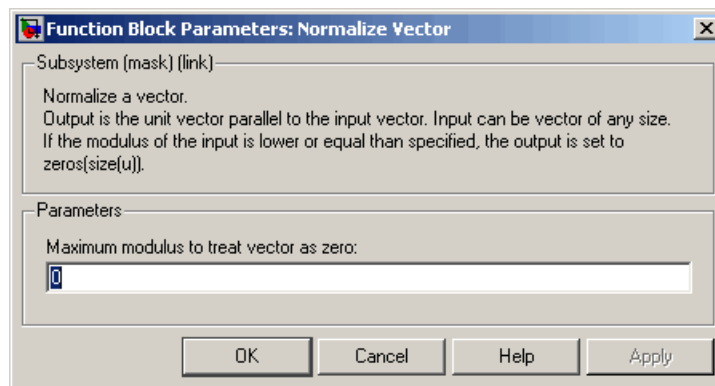
**Purpose** Unit vector parallel to input vector

**Library** Simulink 3D Animation

**Description** Takes an input vector of any size and outputs the unit vector parallel to it.



## Block Parameters Dialog Box



**Maximum modulus to treat vector as zero** — The output is set to zeroes if the modulus of the input is equal to or lower than this value.

**Purpose**

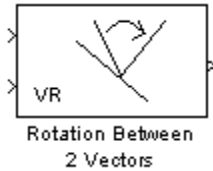
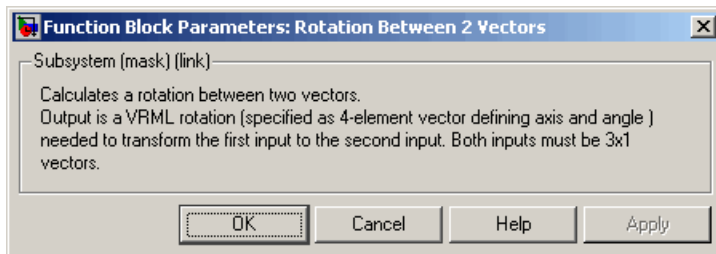
VRML rotation between two 3-D vectors

**Library**

Simulink 3D Animation

**Description**

Takes input of two 3-by-1 vectors and returns a VRML rotation (specified as a four-element vector defining axis and angle) that is needed to transform the first input vector to the second input vector.

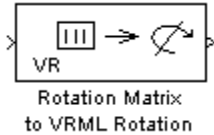
**Block Parameters Dialog Box**

# Rotation Matrix to VRML Rotation

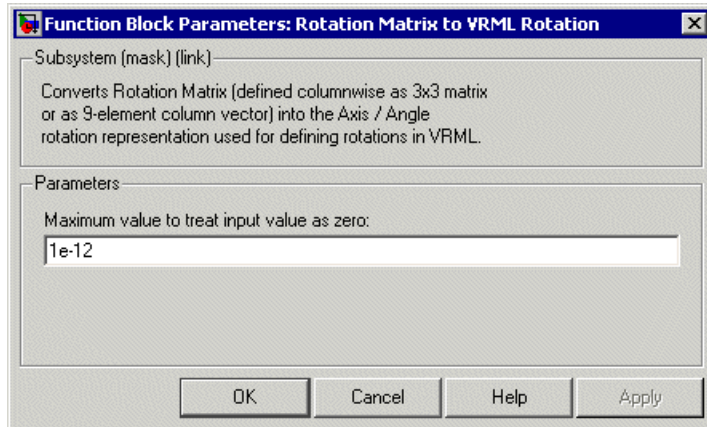
**Purpose** Convert rotation matrix into representation used in VRML

**Library** Simulink 3D Animation

**Description** Takes an input of a rotation matrix and outputs the axis/angle rotation representation used for defining rotations in VRML. The rotation matrix can be either a 9-element column vector or a 3-by-3 matrix defined columnwise.



## Block Parameters Dialog Box



**Maximum value to treat input value as zero** — The input is considered to be zero if it is equal to or lower than this value.

## Rotation 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  $R$ .

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

# Rotation Matrix to VRML Rotation

---

In general,  $R$  requires three independent angles to specify the rotation fully. There are many ways to represent the three independent angles. Here are two:

- You can form three independent rotation matrices  $R_1, R_2, R_3$ , each representing a single independent rotation. Then compose the full rotation matrix  $R$  with respect to fixed coordinate axes as a product of these three:  $R = R_3 * R_2 * R_1$ . The three angles are Euler angles.
- You can represent  $R$  in terms of an axis-angle rotation  $n = (n_x, n_y, n_z)$  and  $\theta$  with  $n * n = 1$ . The three independent angles are  $\theta$  and the two needed to orient  $n$ . Form the antisymmetric matrix:

$$\hat{J} = \begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}$$

Then Rodrigues' formula simplifies  $R$ :

$$R = \exp(\theta J) = I + J \sin \theta + J^2 (1 - \cos \theta)$$

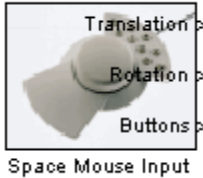
# Space Mouse Input

---

**Purpose** Process input from space mouse device

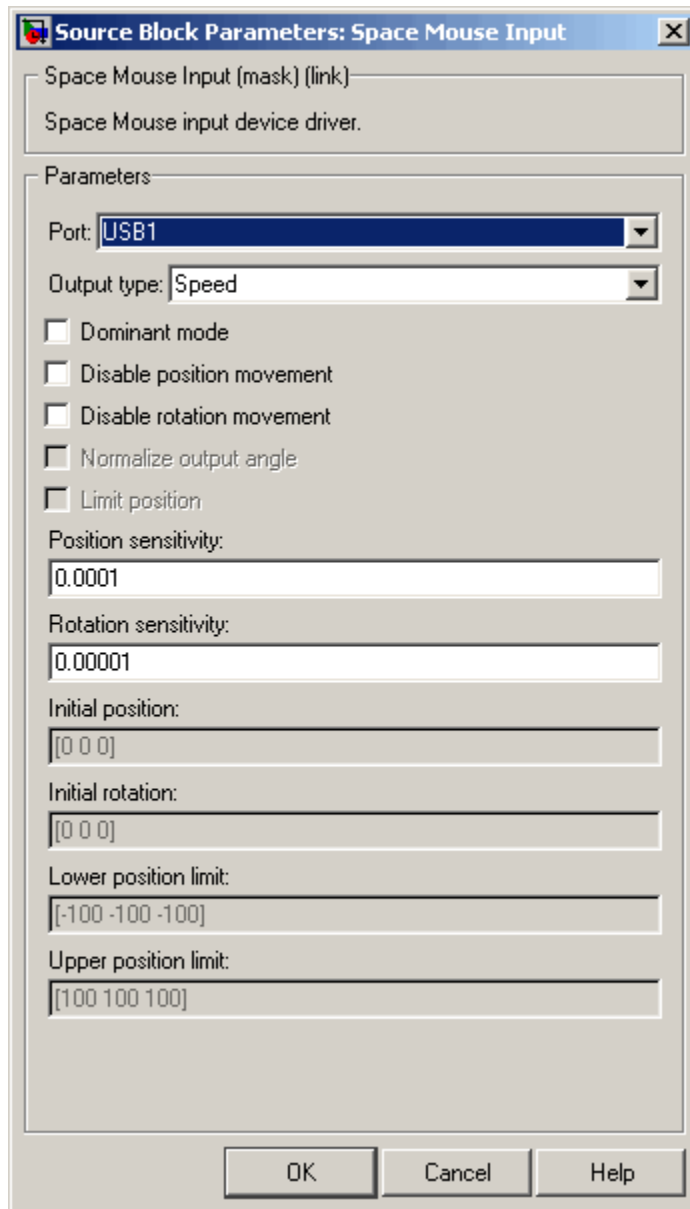
**Library** Simulink 3D Animation

**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 (<http://www.3dconnexion.com>). Contact MathWorks Technical Support (<http://www.mathworks.com/support>) for further information on the support of older 3Dconnexion devices.



**Data Type Support** The Space Mouse Input block outputs signals of type double.





## Block Parameters Dialog Box

**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:

# Space Mouse Input

---

- **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 VRML.

**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**

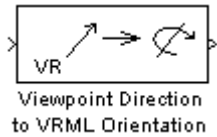
Manipulator with SpaceMouse

# Viewpoint Direction to VRML Orientation

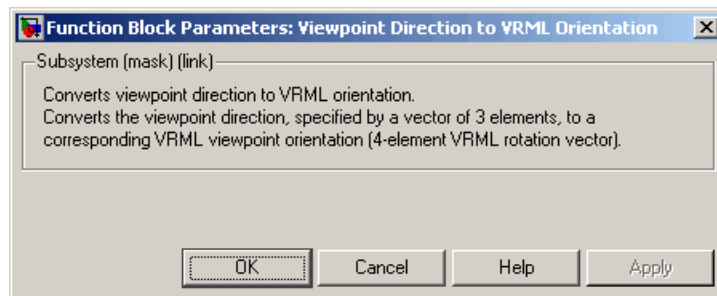
**Purpose** Convert viewpoint direction to VRML orientation

**Library** Simulink 3D Animation

**Description** Takes a viewpoint direction (3-by-1 vector) as input and outputs the corresponding VRML viewpoint orientation (four-element VRML rotation vector).



## Block Parameters Dialog Box



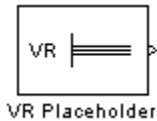
## Purpose

Send unspecified value to Simulink 3D Animation block

## Library

Simulink 3D Animation

## Description



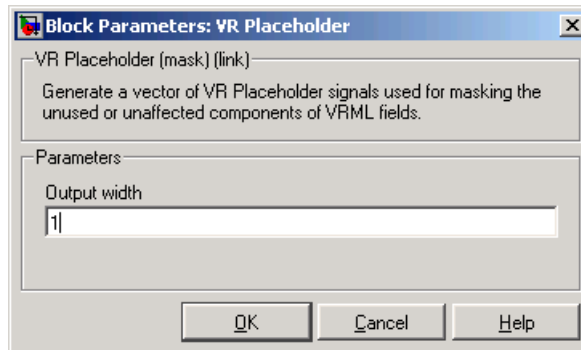
The VR Placeholder block sends out a special value that is interpreted as “unspecified” by the VR Sink block. When this value appears on the VR Sink input, whether as a single value or as an element of a vector, the appropriate value in the virtual world stays unchanged. Use this block to change only 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.

## Data Type Support

A VR Placeholder block outputs signals of type double.

## Block Parameters Dialog Box



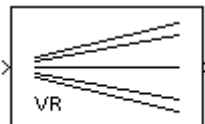
**Output Width** — Length of the vector containing placeholder signal values.

# VR Signal Expander

**Purpose** Expand input vectors into fully qualified VRML field vectors

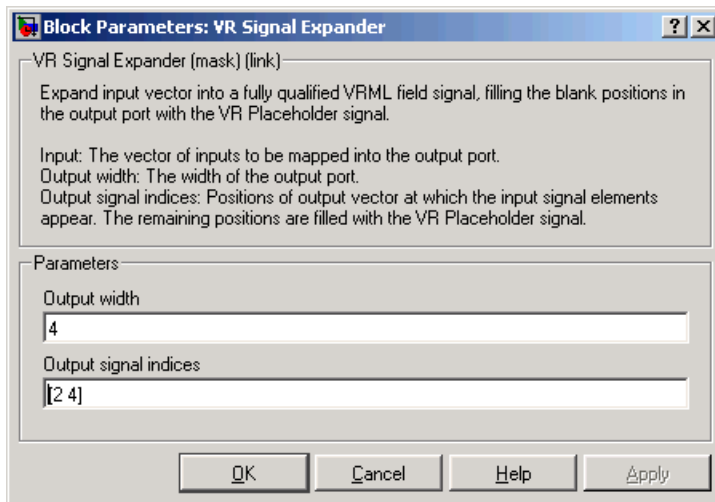
**Library** Simulink 3D Animation

**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.



VR Signal Expander

**Data Type Support** A VR Signal Expander block accepts and outputs signals of type double.



## Block Parameters Dialog Box

**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.

# VR Sink

---

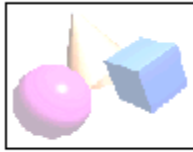
## Purpose

Write data from Simulink model to virtual world

## Library

Simulink 3D Animation

## Description



VR Sink

The VR Sink block writes values from its ports to virtual world fields specified in the Block Parameters dialog box.

For an example of how to use the VR Sink block, see the Foucault Pendulum Model with Virtual Reality Scene example.

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. The 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 closing the viewer window, the Simulink software does not alert you if these properties have changed.

---

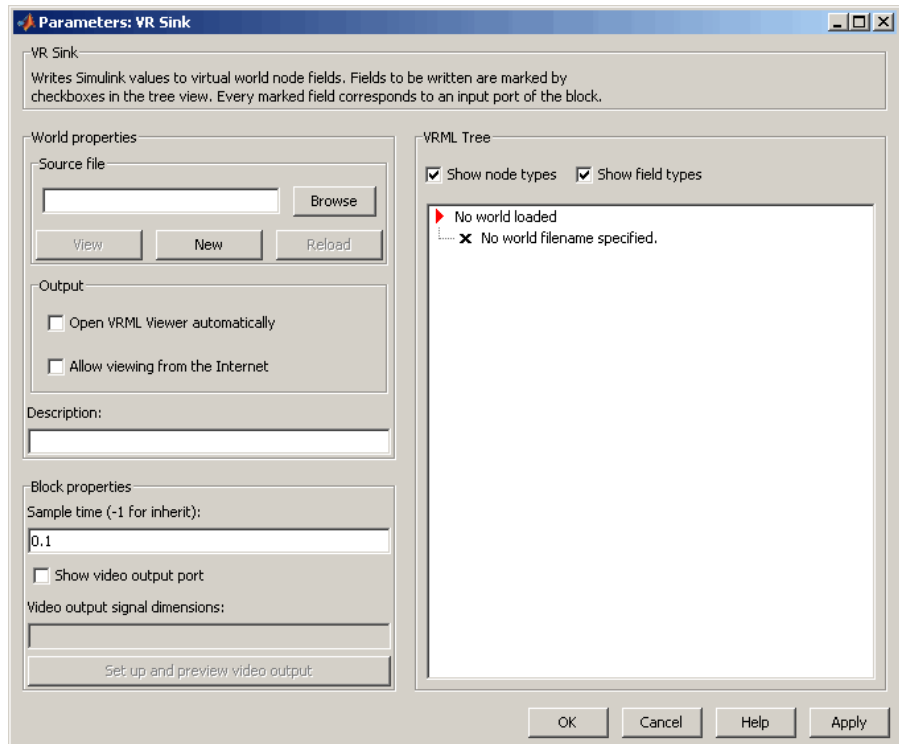
The VR Sink block is a Sim Viewing Device. You can include it in models that you compile with Simulink Coder software. If you use External mode to compile, build, and deploy the model on a target platform, such as xPC Target or Real-Time Windows Target, some sink blocks and Sim Viewing Device blocks stay in Normal mode during simulation, receive data from the target, and display that data. For more information, see “Sim Viewing Devices in External Mode” in the Simulink documentation.

## Data Type Support

A VR Sink block accepts all meaningful data types on input. The block converts these data types to natural VRML types, as necessary. These data types include logicals, many types of signed and unsigned integers,



singles, and doubles. The MATLAB and Simulink interfaces also accept matrices. For further details, see “VRML Data Types” on page 5-24.



## Block Parameters Dialog Box

**Source file** — VRML file name specifying the virtual world that connects to this block. By default, the full path to the associated `.wrl` file appears in this text box. If you enter only the file name in this box, the software assumes that the `.wrl` file resides in the same folder as the model file.

- Click the **New** to open an empty default VRML 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 VRML 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 VRML Viewer automatically** — If you select this check box, the default VRML 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, then the world is visible only on the host computer. This parameter is equivalent to the `RemoteView` property of a `vrworld` object. See “Using the MATLAB Interface” on page 4-2.

**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. See “Using the MATLAB Interface” on page 4-2.

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

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 that you want.

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

**Video output signal dimensions** — Dimensions ([vertical horizontal]) of the video output signal in pixels (default is [200 320]).

**Setup and preview video output** — Opens a figure window for navigation and viewing.

**VRML Tree** — This box shows the structure of the VRML 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 VRML file.

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

**Show node types** — If you select this check box, node types are shown in the VRML tree.

**Show field types** — If you select this check box, field types are shown in the VRML tree.

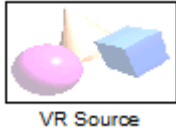
## Purpose

Read data from virtual world to Simulink model

## Library

Simulink 3D Animation

## Description



The VR Source provides access to virtual world fields, as chosen in the Block Parameters dialog box as input signals during simulation.

The VR Source block supports several activities. For example, use the VR Source block to:

- Provide interactivity between a user navigating the virtual world and the Simulink model. The VR Source block can register user interaction with the virtual world. The block can pass to the model those values which then can affect the simulation of the model.
- Read into the model events from the virtual world, such as time ticks or outputs from scripts.
- Read into the model static information about the virtual world (for example, the size of a box defined in the VRML file).
- Access values of 3D object nodes that are not driven by simulation, but whose monitoring is essential.

For an example of how to use the VR Source block, see the `vr crane_panel` example.

---

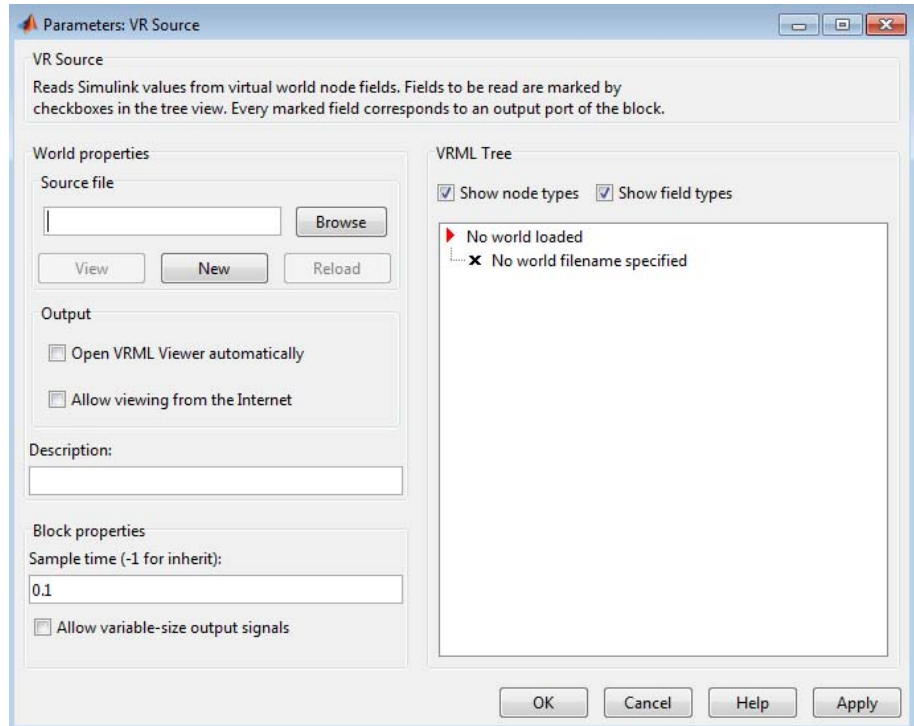
**Note** The current internal viewer window (`vrfigure`) properties are saved together with the Simulink model. The next time that you open the model, the internal viewer window opens with the same parameters that were saved, such as position, size, and navigation mode. When closing the viewer window,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

---

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.



## Block Parameters Dialog Box

**Source file** — VRML file name specifying the virtual world that connects to this block. By default, the full path to the associated `.wrl` file appears in this text box. If you enter only the file name in this box, the software assumes that the `.wrl` file resides in the same folder as the model file.

- Click the **New** to open an empty default VRML 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 VRML 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 VRML Viewer automatically** — If you select this check box, the default VRML 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. See “Using the MATLAB Interface” on page 4-2.

**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. See “Using the MATLAB Interface” on page 4-2.

**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 “VRML Field Data Types” on page 5-24.

---

**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.

---

**VRML Tree** — This box shows the structure of the VRML 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 **VRML 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 VRML file.

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 VRML tree.

**Show field types** — If you select this check box, field types are shown in the VRML tree.

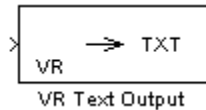
# VR Text Output

---

**Purpose** Allows display of Simulink signal values as text in VRML scene

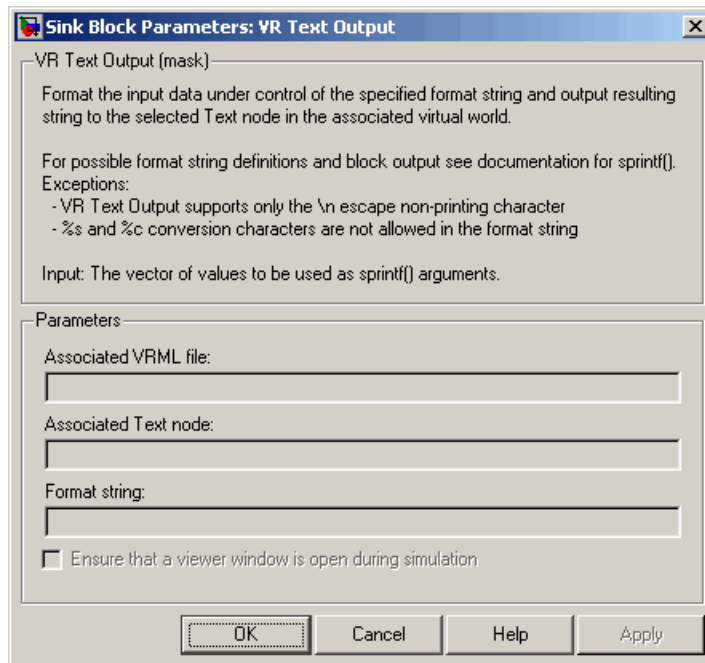
**Library** Simulink 3D Animation

**Description** The VR Text Output can display Simulink values of signal as text in a VRML scene.



Text rendering is a demanding task for VRML 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.





## Block Parameters Dialog Box

**Associated VRML file** — VRML 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.

# VR To Video

---

## Purpose

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

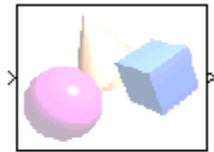
## Library

Simulink 3D Animation

## Description

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

See the VR Sink block for details.



VR To Video

**Purpose** Trace trajectory of object in associated virtual scene

**Library** Simulink 3D Animation

**Description** 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

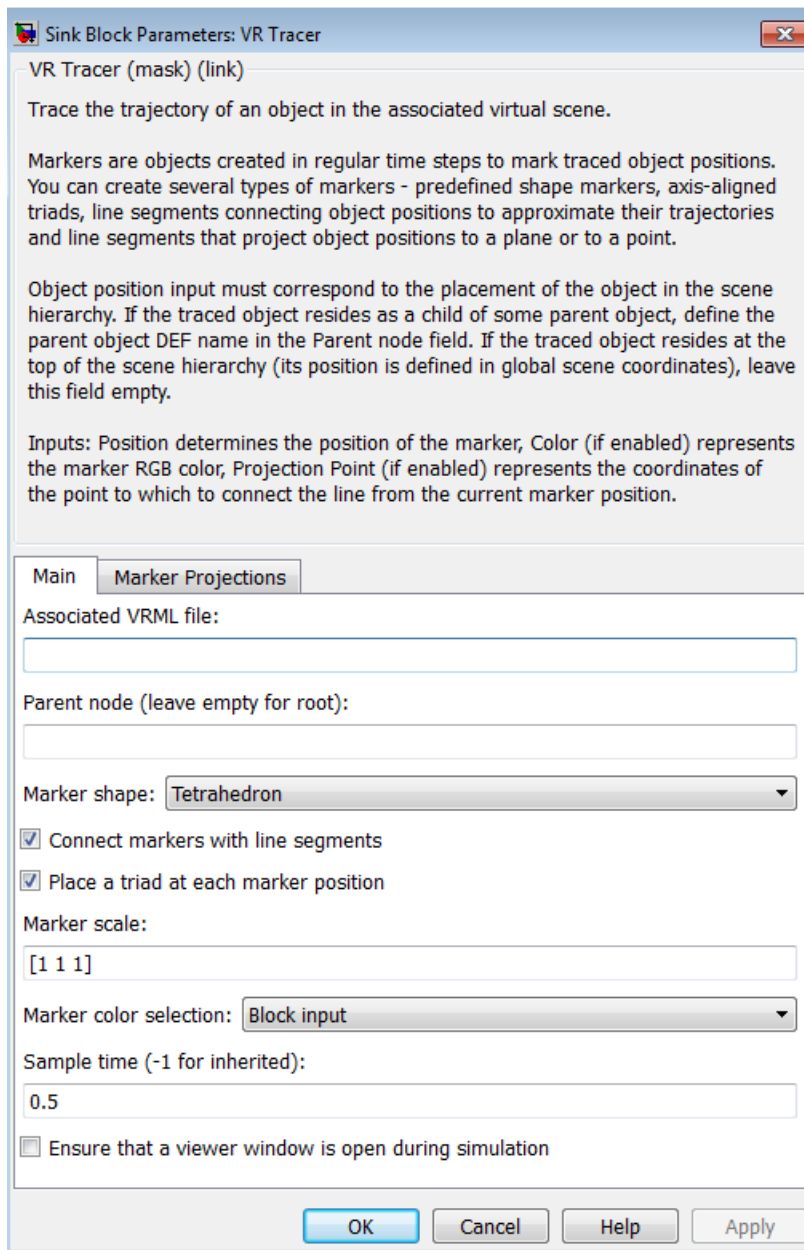
Also, you can project traced object positions to a plane or to a point.

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.

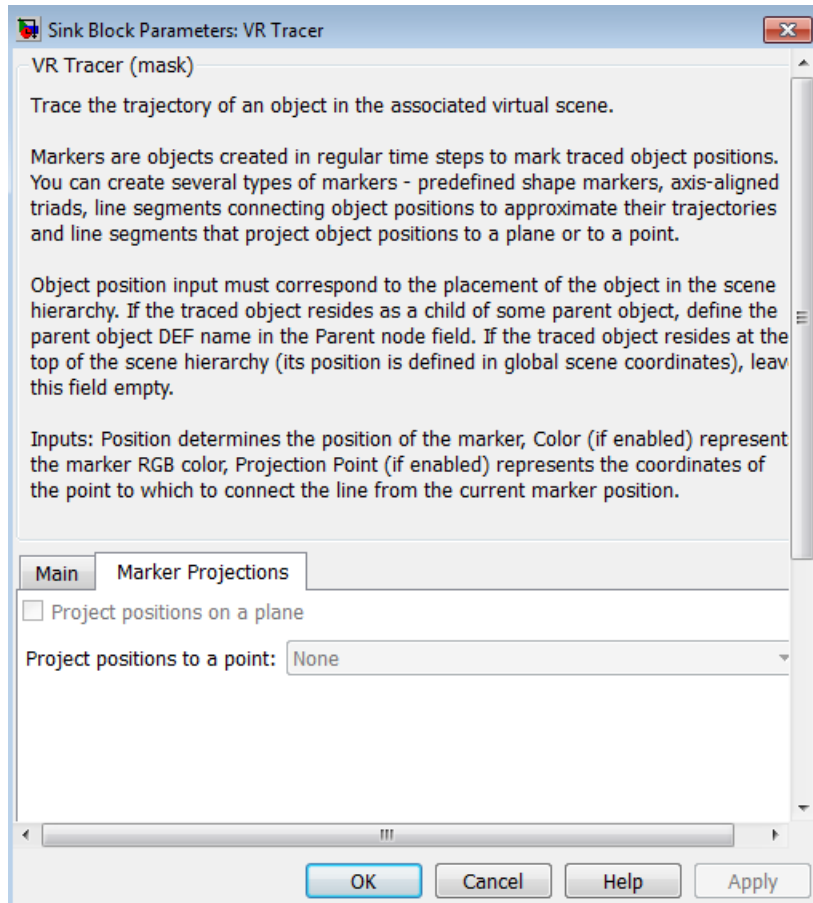
The first block input vector determines the position of the marker. The second block input (if enabled by the **Marker color selection** parameter) represents the marker color. The second or third block input vector (depending on whether the marker color input vector is enabled) specifies the project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

## Block Parameters Dialog Box

Following is the **Main** pane of the VR Tracer block dialog box.



Following is the **Marker Projections** pane of the VR Tracer block dialog box.



**Associated VRML file** — VRML file name specifying the associated virtual world.

**Parent node (leave empty for root)** — Specify the location of the traced object in the scene hierarchy.

**Marker shape** — From the list, select one shape:

- None
- Tetrahedron
- Pyramid
- Box
- Octahedron
- Sphere

**Connect markers with line segments** — Select this check box to connect the traced object path with lines.

**Place a triad at each marker position** — Select this check box 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 a three-component vector that defines the scaling of predefined marker shapes and triads. This parameter allows accommodation for scenes of various sizes.

**Marker color selection** — From the list, select:

- 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 in block mask from color list — Enables the **Marker color** parameter, for selecting one color from of a list of colors for the marker.
- Defined in block mask as RGB values — Enables **Marker color** parameter to accept RGB values for the marker color.

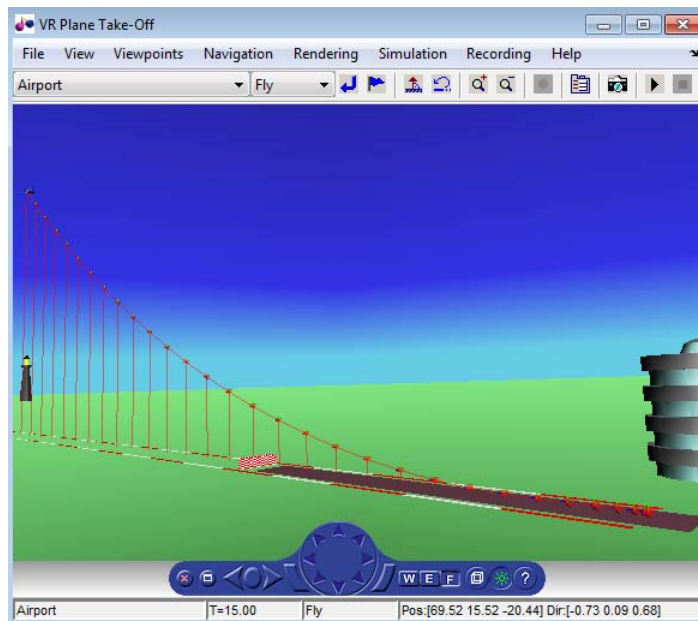
**Marker color** — If **Marker color selection** is Selected in block mask from color list, select the color from the list: yellow, magenta, cyan, red, green, blue, white, black

If **Marker color selection** is Defined in block mask as RGB values, enter RGB values for the marker color.

**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.

**Project positions on a plane** — Specify whether to display line segments from an object to a plane to approximate the trajectory of the object. If you enable this parameter, use the **Projection plane equation coefficients** edit box to specify the plane to 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.slx` model, then after you simulate the model, the object positions project to the  $y=0$  plane.



**Project positions to a point**— 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.



# Function Reference

---

MATLAB Interface (p. 11-2)	Interface with virtual worlds and miscellaneous features
vr.canvas Object Methods (p. 11-4)	Interact with virtual reality canvas
vrworld Object Methods (p. 11-5)	Interact with virtual scene
vrnode Object Methods (p. 11-6)	Get and set VRML node properties
vrfigure Object Methods (p. 11-7)	Get and set Simulink 3D Animation viewer properties
Input Device Objects (p. 11-8)	Create input device objects

## **MATLAB Interface**

<code>stl2vrm1</code>	Convert STL files to VRML format
<code>vrcadcleanup</code>	Clean up VRML file exported from CAD tools
<code>vrclear</code>	Remove all closed virtual worlds from memory
<code>vrclose</code>	Close virtual reality figure windows
<code>vrdir2ori</code>	Convert viewpoint direction to orientation
<code>vrdrawnow</code>	Update virtual world
<code>vrgcbf</code>	Current callback <code>vrfigure</code> object
<code>vrgcf</code>	Handle for active virtual reality figure
<code>vrgetpref</code>	Values of Simulink 3D Animation preferences
<code>vrinstall</code>	Install and check Simulink 3D Animation components
<code>vrlib</code>	Open Simulink block library for Simulink 3D Animation
<code>vrori2dir</code>	Convert viewpoint orientation to direction
<code>vrphysmod</code>	Add virtual reality visualization framework to block diagrams
<code>vrrotmat2vec</code>	Convert rotation from matrix to axis-angle representation
<code>vrrotvec2mat</code>	Convert rotation from axis-angle to matrix representation
<code>vrrotvec</code>	Calculate rotation between two vectors
<code>vrsetpref</code>	Change Simulink 3D Animation preferences

---

<code>vrview</code>	View virtual world using Simulink 3D Animation viewer or Web browser
<code>vrwho</code>	List virtual worlds in memory
<code>vrwhos</code>	List details about virtual worlds in memory

## **vr.canvas Object Methods**

`vr.canvas`

Create virtual reality canvas object

`vr.canvas/capture`

Capture virtual reality canvas to  
RGB image

## vrworld Object Methods

<code>vrworld</code>	Create new <code>vrworld</code> object associated with virtual world
<code>vrworld/addexternproto</code>	Add <code>externproto</code> declaration to virtual world
<code>vrworld/close</code>	Close virtual world
<code>vrworld/delete</code>	Remove virtual world from memory
<code>vrworld/edit</code>	Open virtual world file in external VRML editor
<code>vrworld/get</code>	Property value of <code>vrworld</code> object
<code>vrworld/isvalid</code>	1 if <code>vrworld</code> object is valid, 0 if not
<code>vrworld/nodes</code>	List nodes available in virtual world
<code>vrworld/open</code>	Open virtual world
<code>vrworld/reload</code>	Reload virtual world from VRML file
<code>vrworld/save</code>	Write virtual world to VRML file
<code>vrworld/set</code>	Change property values of <code>vrworld</code> object
<code>vrworld/view</code>	View virtual world

## **vrnode Object Methods**

<code>vrnode</code>	Create node or handle to existing node
<code>vrnode/delete</code>	Remove <code>vrnode</code> object
<code>vrnode/fields</code>	VRML field summary of node object
<code>vrnode/get</code>	Property value of <code>vrnode</code> object
<code>vrnode/getfield</code>	Field value of <code>vrnode</code> object
<code>vrnode/isvalid</code>	1 if <code>vrnode</code> object is valid, 0 if not
<code>vrnode/set</code>	Change property of virtual world node
<code>vrnode/setfield</code>	Change field value of <code>vrnode</code> object
<code>vrnode/sync</code>	Enable or disable synchronization of VRML fields with client

## vrfigure Object Methods

<code>figure</code>	Create new virtual reality figure
<code>vrfigure</code>	Create new virtual reality figure
<code>vrfigure/capture</code>	Create RGB image from virtual reality figure
<code>vrfigure/close</code>	Close virtual reality figure
<code>vrfigure/get</code>	Property value of <code>vrfigure</code> object
<code>vrfigure/isvalid</code>	1 if <code>vrfigure</code> object is valid, 0 if not
<code>vrfigure/set</code>	Change property value of <code>vrfigure</code> object

## **Input Device Objects**

`vrjoystick`

Create joystick object

`vrspacemouse`

Create space mouse object



# Functions — Alphabetical List

---

# figure

---

**Purpose** Create new virtual reality figure

**Syntax**

```
f = figure
f = figure([])
f = figure(world)
f = figure(world, 'PropertyName', propertyvalue, ...)
```

**Description** `f = figure` returns an empty figure object. This object does not have a visual representation. All properties are empty.

`f = figure([])` returns an empty vector of type figure.

`f = figure(world)` creates a new virtual reality figure. It shows the specified world(s) and returns the figure object, `f`.

`f = figure(world, 'PropertyName', propertyvalue, ...)` shows the specified world(s) with specified properties and returns the figure object, `f`.

Creating a virtual figure object with this method creates and displays the object in the figure window viewer (default viewer 'internalv5'). It ignores the value of the `vrsetpref DefaultViewer` property.

**Examples** Create a `vrfigure` object.

```
w=vrworld('membrane.wr1')
open(w);
f=vrfigure(w)
```

<b>Purpose</b>	Convert STL files to VRML format
<b>Syntax</b>	<code>stl2vrm1(source)</code> <code>stl2vrm1(source,destination)</code>
<b>Description</b>	<p><code>stl2vrm1(source)</code> Converts the STL file that you specify with <code>source</code> to a VRML file.</p> <p>Converts both ASCII and binary STL files. The resulting files are VRML97 compliant, UTF-8 encoded text files.</p> <p>VRML files have the same name as the source STL files, except that the extension is <code>.WRL</code> instead of <code>.STL</code>. The <code>stl2vrm1</code> function places the VRML files into the current folder.</p> <p><code>stl2vrm1(source,destination)</code> creates the converted VRML files in the <code>destination</code> folder.</p>
<b>Tips</b>	<ul style="list-style-type: none"><li>• You can use the created assembly VRML files as templates for creating virtual scenes in which you can work with objects of the converted assemblies. To work with the scene effectively, edit the scene as necessary. For example, consider whether you need to add lights, viewpoints, and surrounding objects, modify part materials, define navigation speeds, or make other additions and changes.</li><li>• The <code>stl2vrm1</code> function converts individual STL files according to the STL convention, which places parts in the global coordinate system. If you specify a Physical Modeling XML file as the <code>source</code>, the resulting VRML assembly file reflects the initial positions of the parts defined in the XML file.</li><li>• If you use SolidWorks, then do not use spaces when naming assemblies and components. Avoiding spaces in assembly and component names ensures that the assembly VRML file has the same tree structure as the related source in SolidWorks. You can then use the <code>vrphysmod</code> function to process the assembly VRML file to obtain a Simulink model with VRML visualization.</li></ul>

## Input Arguments

### source

The name of the source STL or Physical Modeling XML file. If `source` is a Physical Modeling XML file, `stl2vrm1` converts all STL files that the XML file references. The `stl2vrm1` function also creates a main assembly VRML file that contains inline references to all converted individual VRML files. All inlines are wrapped by transform nodes with DEF names corresponding to the part names defined in their respective STL source files.

**Default:** ''

### destination

(Optional) Folder in which to create converted files. If the destination folder does not exist, `stl2vrm1` attempts to create the destination folder.

**Default:** ''

## Examples

These examples use STL files that SimMechanics product includes. If you do not have the SimMechanics product installed, then substitute another STL file .

Convert the STL file `fourbar-Bar1-1.STL` (which is in `matlab/toolbox/phymod/mech/mechdemos`) to a VRML file and place the resulting file in the current folder. The resulting VRML file (`fourbar-Bar1-1.wrl`) has the same name as the source file, except that it has a `.wrl` extension instead of a `.stl` extension.

```
stl2vrm1('fourbar-Bar1-1.STL')
ls
.      ..   fourbar-Bar1-1.wrl
% Other files and folders in the current folder appear, as well
```

Convert the STL file `fourbar-Bar2-1.STL` to a VRML file and place the resulting file in a folder called `virtualworlds`. The resulting VRML file is in the destination folder that you specify.

```
mkdir('virtualworlds')
```

```
stl2vrm1('fourbar-Bar2-1.STL','virtualworlds')
cd virtualworlds
ls
.          ..    fourbar-Bar2-1.wrl
```

## See Also

[vrcadcleanup](#) | [vrphysmod](#)

# vrcadcleanup

---

**Purpose** Clean up VRML 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 VRML file exported from Pro/ENGINEER® or SolidWorks. This cleanup enables the Simulink 3D Animation software to use these files.

`vrcadcleanup` performs the following modifications:

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

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

Argument	Description
'solidworks'	Assumes that the software is exporting the original set of VRML 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 VRML 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

`stl2vrml` | `vrphysmod`

**Purpose** Create virtual reality canvas object

**Syntax**

```
c = vr.canvas(vrworld)
c = vr.canvas(vrworld, parent)
c = vr.canvas(vrworld, parent, position)
c = vr.canvas(world, 'propertyName', propertyValue,...)
```

**Description**

`c = vr.canvas(vrworld)` creates a MATLAB figure containing a virtual reality canvas that shows the specified virtual reality world. It returns a virtual reality canvas object.

`c = vr.canvas(vrworld, parent)` creates a virtual reality canvas that shows the specified virtual world in the figure specified in `parent`. It returns a virtual reality canvas object.

`c = vr.canvas(vrworld, parent, position)` creates a virtual reality canvas object that shows the specified virtual world in the figure specified in `parent` at `position`, specified in pixels. It returns a virtual reality canvas object.

`c = vr.canvas(world, 'propertyName', propertyValue,...)` creates a virtual reality canvas that shows the specified world with specified properties. See “Property Summary” on page 12-9 for a list of the properties and their values.

## Method Summary

Method	Description
<code>vr.canvas/capture</code>	Capture virtual reality canvas to RGB image



**Property  
Summary**

<b>Property</b>	<b>Values</b>	<b>Description</b>
Antialiasing	'off'   'on' Default: 'off'	Determines whether antialiasing is used when rendering the scene. Antialiasing smooths textures by interpolating values between texture points. This property causes a significant CPU load, but the resulting scene looks more natural. Read/write.
CameraBound	'off'   'on' Default: 'on'	Controls whether or not the camera moves with the current viewpoint. When the camera is bound, its position, direction, and up vector are relative to the viewpoint. For unbound camera, these values are absolute. Read/write.
CameraDirection	Vector of three doubles	Specifies the camera direction relative to the direction of the current viewpoint. Read/write.

<b>Property</b>	<b>Values</b>	<b>Description</b>
CameraDirectionAbs	Vector of three doubles	Specifies the camera direction in world coordinates. Read only.
CameraPosition	Vector of three doubles	Specifies the camera position relative to the position of the current viewpoint. Read/write.
CameraPositionAbs	Vector of three doubles	Specifies the camera position in world coordinates. Read only.
CameraUpVector	Vector of three doubles	Specifies the camera up vector relative to the up vector of the current viewpoint. Read/write.
CameraUpVectorAbs	Vector of three doubles	Specifies the camera up vector in world coordinates. Read only.
DeleteFcn	String	Specifies the callback invoked when closing the vr.canvas object. Read/write.

Property	Values	Description
Headlight	'off'   'on' Default: 'on'	Turns the headlight on or off. Read/write. Specifies whether headlight is enabled. The headlight is an additional white directional light that moves and rotates with the camera.
Lighting	'off'   'on' Default: 'on'	Specifies whether the lighting is taken into account when rendering. If it is off, all the objects are drawn as if uniformly lit. Read/write.
MaxTextureSize	'auto'   $32 \leq x \leq \text{video card limit}$ , 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 vr.canvas objects. The value must be a power of two and may be further adjusted to match specific hardware renderer limits. 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

Property	Values	Description
		maximum possible pixel size.
NavMode	'fly'   'examine'   'walk'   'none' Default: 'fly'	Specifies the current navigation mode.
NavPanel	<ul style="list-style-type: none"><li>'none' Panel is not visible.</li><li>'translucent' Panel floats half transparently above the scene.</li><li>'opaque' Panel floats above the scene.</li></ul> Default: 'none'	Controls the appearance of the control panel in the vr.canvas object.
NavSpeed	'veryslow'   'slow'   'normal'   'fast'   'veryfast' Default: 'normal'	Specifies navigation speed. Read/write.
NavZones	'off'   'on' Default: 'off'	Toggles navigation zones on/off. Read/write.
Parent	Double	Specifies handle of parent of this virtual reality canvas. Read-only.

Property	Values	Description
Position	Vector of four doubles	Specifies screen coordinates of this virtual reality canvas.
Textures	'off'   'on' Default: 'on'	Turns texture rendering on or off. Read/write.
Transparency	'off'   'on' Default: 'on'	Specifies whether or not transparency information is taken into account when rendering. If it is off, all objects are drawn opaque. Read/write.
Units	'pixels'   'normalized'	Specifies the units to interpret the Position property. The DefaultCanvasUnits preference controls the default for this property. Read/write
Viewpoint	String. If active viewpoint does not have a name, value is empty.	Specifies the vr.canvas object's active viewpoint. This value is an empty string if the active viewpoint has no description. Read/write.

Property	Values	Description
Wireframe	'off'   'on' Default: 'off'	Specifies whether objects are drawn as solids or wireframes. Read/write.
World	vrworld object	Specifies the world this vr.canvas object is displaying. Read only.
ZoomFactor	Double Default: 1	Specifies the camera zoom factor. Read/write.  A zoom factor of 2 makes the scene look twice as big. A zoom factor of 0.1 makes it look 10 times smaller, and so forth.

## See Also

vrworld

**Purpose** Capture virtual reality canvas to RGB image

**Syntax** `z = capture(vr.canvas object)`

**Description** `z = capture(vr.canvas object)` captures a virtual reality canvas object into a TrueColor RGB image. You can then display this image with the `image` function.

**Examples** Create a `vr.canvas` object to contain the virtual world, `vrlights`, capture that object to an RGB image, and display that image with the `image` function.

```
w = vrworld('vrlights');  
open(w);  
c = vr.canvas(w,(gcf,[30 30 300 200]));  
z=capture(c);  
image(z);
```

# vrclear

---

**Purpose** 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`



<b>Purpose</b>	Close virtual reality figure windows
<b>Syntax</b>	<code>vrclose</code> <code>vrclose all</code>
<b>Description</b>	<code>vrclose</code> and <code>vrclose all</code> close all the open virtual reality figures.
<b>Examples</b>	<p>Open a series of virtual reality figure windows by typing</p> <pre>vrpend vrbounce vrlights</pre> <p>Arrange the viewer windows so they are all visible. Type</p> <pre>vrclose</pre> <p>All the virtual reality figure windows disappear from the screen.</p>
<b>See Also</b>	<code>vrfigure/close</code>

# vrdir2ori

---

**Purpose** 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 (VRML 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` | `vrrotmat2vec` | `vrrotvec` | `vrrotvec2mat`

**Purpose** 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.

# vredit

---

**Purpose** 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.

**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`

**Purpose** Create new virtual reality figure

**Syntax**

```
f = vrfigure(world)
f = vrfigure(world,position)
f = vrfigure
f = vrfigure([])
```

**Description** `f = vrfigure(world)` creates a new virtual reality figure showing the specified world and returns an appropriate `vrfigure` object. The input argument `world` must be a `vrworld` object.

`f = vrfigure(world,position)` creates a new virtual reality figure at the specified position.

`f = vrfigure` returns an empty `vrfigure` object that does not have a visual representation.

`f = vrfigure([])` returns an empty vector of type `vrfigure`.

Creating a virtual figure with this method creates and displays the figure in the viewer specified in the `vrsetpref DefaultViewer` property.

## Method Summary

Method	Description
<code>capture</code>	Create RGB image from virtual reality figure
<code>close</code>	Close virtual reality figure
<code>get</code>	Property value of <code>vrfigure</code> object
<code>isvalid</code>	1 if <code>vrfigure</code> object is valid, 0 if not
<code>set</code>	Change property value of <code>vrfigure</code> object

**Examples** Create a `vrworld` object. At the MATLAB command prompt, type

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

The `vrworld` object `myworld` is associated with the virtual world `vrmount.wrl`.

## vrfigure

---

Next, open the virtual world using the `vrworld` object. You must open the virtual world before you can view it. At the MATLAB command prompt, type

```
open(myworld)
```

You can now view the virtual world in the Simulink 3D Animation viewer by typing

```
f = vrfigure(myworld)
```

Your viewer opens and displays the virtual scene.

### **See Also**

`vrworld` | `vrworld/open`

<b>Purpose</b>	Create RGB image from virtual reality figure
<b>Syntax</b>	<code>image_capture = capture(vrfigure_object)</code>
<b>Description</b>	<code>image_capture = capture(vrfigure_object)</code> captures a virtual reality figure into a TrueColor RGB image. This image can be displayed by the <code>image</code> command and subsequently printed.
<b>Examples</b>	<p>Create a <code>vrworld</code> object. At the MATLAB command prompt, type</p> <pre>myworld = vrworld('vrmount.wrl')</pre> <p>The <code>vrworld</code> object <code>myworld</code> is associated with the virtual world <code>vrmount.wrl</code>.</p> <p>Next, open the virtual world using the <code>vrworld</code> object. You must open the virtual world before you can view it. At the MATLAB command prompt, type</p> <pre>open(myworld)</pre> <p>You can now view the virtual world in the Simulink 3D Animation viewer by typing</p> <pre>f = vrfigure(myworld)</pre> <p>Your viewer opens and displays the virtual scene. Next, create an RGB image by typing</p> <pre>image_capture = capture(f);</pre> <p>Lastly, view the image</p> <pre>image(image_capture)</pre> <p>The scene from the viewer window is displayed in a MATLAB figure window.</p>

## **vrfigure/capture**

---

### **See Also**

`vrfigure`



<b>Purpose</b>	Close virtual reality figure
<b>Syntax</b>	<code>close(vrfigure_object)</code>
<b>Arguments</b>	<code>vrfigure_object</code> Name of a figure object.
<b>Description</b>	<code>close(vrfigure_object)</code> closes the virtual reality figure referenced by <code>vrfigure_object</code> . If <code>vrfigure_object</code> is a vector of <code>vrfigure</code> handles, then multiple figures are closed.
<b>Examples</b>	<pre>myworld = vrworld('vrpend.wrl') open(myworld) f = vrfigure(myworld) close(f)</pre>
<b>See Also</b>	<code>vrfigure</code>   <code>vrworld</code>   <code>vrworld/open</code>

# vrfigure/get

---

**Purpose** Property value of vrfigure object

**Syntax** `get(vrfigure_object)`  
`x = get(vrfigure_object, 'property_name')`

**Arguments** `vrfigure_object` Name of a vrfigure object.  
`property_name` Name of the property.

**Description** `get(vrfigure_object)` lists all the properties of the vrfigure object. This is useful when you want to determine the current values of these properties. Use a command like the following to return a value of the specified property of the vrfigure object.

`x = get(vrfigure_object, 'property_name')` returns a value of the specified property of the vrfigure object.

The following are properties of vrfigure objects.

Property	Value	Description
Antialiasing	'off'   'on' Default: 'off'	Determines whether antialiasing is used when rendering the scene. Antialiasing smooths textures by interpolating values between texture points. Read/write.
CameraBound	'off'   'on' Default: 'on'	Controls whether or not the camera moves with the current viewpoint. Read/write.
CameraDirection	Vector of three doubles	Specifies the camera direction relative to the direction of the current viewpoint. Read/write.
CameraDirectionAbs	Vector of three doubles	Specifies the camera direction in world coordinates. Read only.

Property	Value	Description
CameraPosition	Vector of three doubles	Specifies the camera position relative to the position of the current viewpoint. Read/write.
CameraPositionAbs	Vector of three doubles	Specifies the camera position in world coordinates. Read only.
CameraUpVector	Vector of three doubles	Specifies the camera up vector relative to the up vector of the current viewpoint. Read/write.
CameraUpVectorAbs	Vector of three doubles	Specifies the camera up vector in world coordinates. Read only.
CaptureFileFormat	'tif'   'png' Default: 'tif'	Specifies file format for a captured frame file. Read/write.
CaptureFileName	String. Default: '%f_anim_%n.ext'	Specifies the frame capture file name. The string can contain tokens that are replaced by the corresponding information when the frame capture takes place. For further details, see “Define File Name Tokens” on page 7-22. Read/write.
DeleteFcn	String	Specifies the callback invoked when closing the vrfigure object. Read/write.
Fullscreen	'off'   'on' Default: 'off'	Specifies whether the figure is in fullscreen mode. Read/write.
Headlight	'off'   'on' Default: 'on'	Turns the headlight on or off. Read/write.

## vrfigure/get

Property	Value	Description
Lighting	'off'   'on' Default: 'on'	Specifies whether the lighting is taken into account when rendering. If it is off, all the objects are drawn as if uniformly lit. Read/write.
MaxTextureSize	'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 vrfigure 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 possible pixel size. If the value you enter is unsuitable, a warning might trigger. The Simulink 3D Animation software then automatically adjusts the property to the next smaller suitable value.
Name	String	Specifies the name of this vrfigure object. Read/write.
NavMode	'fly'   'examine'   'walk'   'none' Default: 'examine'	Specifies navigation mode. Read/write.
NavPanel	'opaque'   'translucent'   'none'   'halfbar'   'bar' Default: 'halfbar'	Controls the appearance of the navigation panel in the Simulink 3D Animation viewer. Read/write.

Property	Value	Description
NavSpeed	'veryslow'   'slow'   'normal'   'fast'   'veryfast' Default: 'normal'	Specifies navigation speed. Read/write.
NavZones	'off'   'on' Default: 'off'	Toggles navigation zones on/off. Read/write.
Position	Vector of four doubles	Specifies the screen coordinates of this vrfigure object. Read/write.
Record2D	'off'   'on' Default: 'off'	Enables 2-D offline animation file recording. Read/write.
Record2DCompress Method	' '   'auto'   'lossless'   'none'   'compress_method' Default: 'auto'	Specifies the compression method for creating 2-D animation files. For valid compress_method settings, see profile in the MATLABVideoWriter documentation. Read/write.
Record2DCompress Quality	0-100 Default: '75'	Specifies the quality of 2-D animation file compression. For details, see the MATLABVideoWriter documentation. Read/write.
Record2DFileName	String. Default: '%f_anim_%n.ext'	Specifies the 2-D offline animation file name. The string can contain tokens that are replaced by the corresponding information when the animation recording takes place. For further details, see “Animation Recording File Tokens” on page 4-12. Read/write.

## vrfigure/get

Property	Value	Description
Record2DFPS	Scalar Default: 15	Specifies rate of playback for the 2-D offline animation video in frames per second (fps). For details, see the MATLABVideoWriter documentation.
StatusBar	'off'   'on' Default: 'on'	Toggles the status bar at the bottom of the Simulink 3D Animation viewer. Read/write.
Textures	'off'   'on' Default: 'on'	Turns texture rendering on or off. Read/write.
ToolBar	'off'   'on' Default: 'on'	Toggles toolbar on the Simulink 3D Animation viewer. Read/write.
Transparency	'off'   'on' Default: 'on'	Specifies whether or not transparency information is taken into account when rendering. Read/write.
Viewpoint	String. If active viewpoint does not have a name, value is empty.	Specifies the vrfigure object's active viewpoint. Read/write.
Wireframe	'off'   'on' Default: 'off'	Specifies whether objects are drawn as solids or wireframes. Read/write.
World	vrworld object	Specifies the world this vrfigure object is displaying. Read only.
ZoomFactor	Double	Specifies the camera zoom factor. Read/write.

## Examples

Create a vrworld object:

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

The vrworld object myworld is associated with the virtual world vrmount.wrl. Open the world:

```
open(myworld)
```

Create a vrfigure object:

```
f = vrfigure(myworld);
```

You can now get the object properties of the vrfigure object f:

```
get(f)
```

This returns the following object properties:

```
Antialiasing = 'off'  
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 = ''  
Fullscreen = 'off'  
Headlight = 'on'  
Lighting = 'on'  
MaxTextureSize = 4096  
Name = 'VR Car in the Mountains'  
NavMode = 'examine'  
NavPanel = 'halfbar'  
NavSpeed = 'normal'
```

## vrfigure/get

---

```
NavZones = 'off'  
Position = [5 92 576 380]  
Record2D = 'off'  
Record2DCompressMethod = 'auto'  
Record2DCompressQuality = 75  
Record2DFPS = 15  
Record2DFileName = '%f_anim_%n.avi'  
StatusBar = 'on'  
Textures = 'on'  
ToolBar = 'on'  
Transparency = 'on'  
Viewpoint = 'View 1 - Observer'  
Wireframe = 'off'  
World = vrworld object: 1-by-1  
ZoomFactor = 1
```

### See Also

vrfigure | vrfigure/set



<b>Purpose</b>	1 if vrfigure object is valid, 0 if not
<b>Syntax</b>	<code>x = isvalid(vrfigure_object_vector)</code>
<b>Arguments</b>	<code>vrfigure_object_vector</code> Name of an array of vrfigure objects.
<b>Description</b>	This method detects whether the vrfigure handles are valid and returns an array that contains a 1 where the vrfigure handles are valid and returns a 0 where they are not.
<b>See Also</b>	<code>vrnode/isvalid</code>   <code>vrworld/isvalid</code>

# vrfigure/set

**Purpose** Change property value of vrfigure object

**Syntax** `set(vrfigure_object, 'property_name', property_value)`

**Arguments**

- `vrfigure_object` Name of a vrfigure object.
- `property_name` Name of the property you want to set.
- `property_value` New value of the property.

**Description** The `set(vrfigure_object)` method allows you to set the property value of a vrfigure object. This method is useful when you want to change the value of a property.

The following are properties of vrfigure objects.

Property	Value	Description
Antialiasing	'off'   'on' Default: 'off'	Determines whether antialiasing is used when rendering the scene. Antialiasing smooths textures by interpolating values between texture points. Read/write.
CameraBound	'off'   'on' Default: 'on'	Controls whether or not the camera moves with the current viewpoint. Read/write.
CameraDirection	Vector of three doubles	Specifies the camera direction relative to the direction of the current viewpoint. Read/write.
CameraDirectionAbs	Vector of three doubles	Specifies the camera direction in world coordinates. Read only.
CameraPosition	Vector of three doubles	Specifies the camera position relative to the position of the current viewpoint. Read/write.

Property	Value	Description
CameraPositionAbs	Vector of three doubles	Specifies the camera position in world coordinates. Read only.
CameraUpVector	Vector of three doubles	Specifies the camera up vector relative to the up vector of the current viewpoint. Read/write.
CameraUpVectorAbs	Vector of three doubles	Specifies the camera up vector in world coordinates. Read only.
CaptureFileFormat	'tif'   'png' Default: 'tif'	Specifies file format for a captured frame file. Read/write.
CaptureFileName	String. Default: '%f_anim_%n.ext'	Specifies the frame capture file name. The string can contain tokens that are replaced by the corresponding information when the frame capture takes place. For further details, see “Define File Name Tokens” on page 7-22. Read/write.
DeleteFcn	String	Specifies the callback invoked when closing the vrfigure object. Read/write.
Headlight	'off'   'on' Default: 'on'	Turns the headlight on or off. Read/write.
Lighting	'off'   'on' Default: 'on'	Specifies whether the lighting is taken into account when rendering. If it is off, all the objects are drawn as if uniformly lit. Read/write.

## vrfigure/set

Property	Value	Description
MaxTextureSize	'auto'   $32 \leq x \leq$ 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 vrfigure 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 possible pixel size. If the value you enter is unsuitable, a warning might trigger. The Simulink 3D Animation software then automatically adjusts the property to the next smaller suitable value.
Name	String	Specifies the name of this vrfigure object. Read/write.
NavMode	'fly'   'examine'   'walk'   'none' Default: 'examine'	Specifies navigation mode. Read/write.
NavPanel	'opaque'   'translucent'   'none'   'halfbar'   'bar' Default: 'halfbar'	Controls the appearance of the navigation panel in the Simulink 3D Animation viewer. Read/write.
NavSpeed	'veryslow'   'slow'   'normal'   'fast'   'veryfast' Default: 'normal'	Specifies navigation speed. Read/write.
NavZones	'off'   'on' Default: 'off'	Toggles navigation zones on/off. Read/write.
Position	Vector of four doubles	Specifies the screen coordinates of this vrfigure object. Read/write.

Property	Value	Description
Record2D	'off'   'on' Default: 'off'	Enables 2-D offline animation file recording. Read/write.
Record2DCompress Method	' '   '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 function documentation for <code>avifile</code> . Read/write.
Record2DCompress Quality	0-100 Default: '75'	Specifies the quality of 2-D animation file compression. Read/write.
Record2DFileName	String. Default: '%f_anim_%n.ext'	Specifies the 2-D offline animation file name. The string can contain tokens that are replaced by the corresponding information when the animation recording takes place. For further details, see “Animation Recording File Tokens” on page 4-12. Read/write.
StatusBar	'off'   'on' Default: 'on'	Toggles the status bar at the bottom of the Simulink 3D Animation viewer. Read/write.
Textures	'off'   'on' Default: 'on'	Turns texture rendering on or off. Read/write.
ToolBar	'off'   'on' Default: 'on'	Toggles toolbar on the Simulink 3D Animation viewer. Read/write.
Transparency	'off'   'on' Default: 'on'	Specifies whether or not transparency information is taken into account when rendering. Read/write.

# vrfigure/set

Property	Value	Description
Viewpoint	String. If active viewpoint does not have a name, value is empty.	Specifies the vrfigure object's active viewpoint. Read/write.
Wireframe	'off'   'on' Default: 'off'	Specifies whether objects are drawn as solids or wireframes. Read/write.
World	vrworld object	Specifies the world this vrfigure object is displaying. Read only.
ZoomFactor	Double	Specifies the camera zoom factor. Read/write.

## Examples

Create a vrworld object.

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

The vrworld object myworld is associated with the virtual world vrmount.wrl. Open the world:

```
open(myworld)
```

Create a vrfigure object:

```
f = vrfigure(myworld);
```

The VR Car in the Mountains virtual world opens in the Simulink 3D Animation viewer. You can now set the object properties of the vrfigure object f:

```
set(f,'Name','Car on a Mountain Road')
```

You can see that the name of the virtual world has changed in the viewer.

**See Also**      vrfigure | vrfigure/get

# vrgcbf

---

**Purpose** 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.



**Purpose** 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` | `vrfigure/get` | `vrfigure/set`

# vrgetpref

---

**Purpose** 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>DataTypeBool</code>	Specifies the handling of the VRML Bool data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are 'logical' and 'char'. If set to 'logical', the VRML Bool data type is returned as a logical value. If set to 'char', the Bool data type

Preference	Description
	is returned 'on' or 'off'. Default is 'logical'.
DataTypeInt32	Specifies handling of the VRML Int32 data type for vrnnode/setfield and vrnnode/getfield. Valid values are 'int32' and 'double'. If set to 'int32', the VRML Int32 data type is returned as int32. If set to 'double', the Int32 data type is returned as 'double'. Default is 'double'.
DataTypeFloat	Specifies the handling of the VRML float data type for vrnnode/setfield and vrnnode/getfield. Valid values are 'single' and 'double'. If set to 'single', the VRML 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'.

Preference	Description
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>'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.canvas</code> objects. See <code>vr.canvas</code> for detailed description. Default is 'normalized'.
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>vrfigure/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.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.

Preference	Description
DefaultFigureRecord2D CompressMethod	Specifies the default compression method for creating 2-D animation files for new <code>vrfigure</code> objects. Valid values are <code>''</code> , <code>'auto'</code> , <code>'lossless'</code> , and <code>'codec_code'</code> .
DefaultFigureRecord2D CompressQuality	Specifies the default quality of 2-D animation file compression for new <code>vrfigure</code> objects. Valid values are 0-100.
DefaultFigureRecord2D FileName	Specifies the default 2-D offline animation file name for new <code>vrfigure</code> objects.
DefaultFigureRecord2DFPS	Specifies the default frames per second playback speed.
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 <code>'off'</code> and <code>'on'</code> .
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>vrfigure/get</code> for detailed description. Default is <code>'on'</code> .
DefaultFigureToolBar	Specifies whether the toolbar appears by default on the Simulink 3D Animation viewer for new <code>vrfigure</code> objects. Valid values are <code>'off'</code> and <code>'on'</code> .

Preference	Description
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	<p>Specifies which viewer is used to view a virtual scene.</p> <ul style="list-style-type: none"> <li>• 'internal'</li> </ul> <p>Default Simulink 3D Animation viewer. This is the same as setting the value to 'internalv5' for all platforms except the Linux platform.</p> <ul style="list-style-type: none"> <li>• 'internalv4'</li> </ul> <p>Legacy Simulink 3D Animation viewer.</p> <ul style="list-style-type: none"> <li>• 'internalv5'</li> </ul> <p>Viewer based on MATLAB figure windows.</p> <ul style="list-style-type: none"> <li>• 'web'</li> </ul>

Preference	Description
	<p>Web browser becomes viewer. This is the current Web browser VRML plug-in.</p> <hr/> <p><b>Note</b> For Linux platforms, 'internal' sets the default viewer to the legacy viewer, and 'internalv5' sets the default viewer to the viewer based on MATLAB figures.</p> <hr/>
DefaultWorldRecord3D FileName	Specifies the default 3-D animation file name for new vrworld objects.
DefaultWorldRecordMode	Specifies the default animation recording mode for new vrworld objects. Valid values are 'manual' and 'scheduled'.
DefaultWorldRecord Interval	Specifies the default start and stop times for scheduled animation recording for new vrworld objects. Valid value is a vector of two doubles.
DefaultWorldRemoteView	Specifies whether the virtual world is enabled by default for remote viewing for new vrworld objects. Valid values are 'off' and 'on'.
DefaultWorldTimeSource	Specifies the default source of the time for new vrworld objects. Valid values are 'external' and 'freerun'.



Preference	Description
Editor	Path to the VRML editor. If this path is empty, the MATLAB editor is used.
HttpPort	IP port number used to access the Simulink 3D Animation server over the Web via HTTP. If you change this preference, you must restart the MATLAB software before the change takes effect.
TransportBuffer	Length of the transport buffer (network packet overlay) for communication between the Simulink 3D Animation server and its clients.
TransportTimeout	Amount of time the Simulink 3D Animation server waits for a reply from the client. If there is no response from the client, the Simulink 3D Animation server disconnects from the client.
VrPort	IP port used for communication between the Simulink 3D Animation server and its clients. If you change this preference, you must restart the MATLAB software before the change takes effect.

The `HttpPort`, `VrPort`, and `TransportBuffer` preferences affect Web-based 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, based on MATLAB figure windows, the legacy Simulink 3D Animation viewer, or in your Web browser.

DefaultViewer Setting	Description
'internal'	Default Simulink 3D Animation viewer. This is the same as setting the value to 'internalv5' for all platforms except the Linux platform.
'internalv4'	Simulink 3D Animation viewer that is the previous default viewer for most platforms. This viewer continues to be the default viewer for Linux platforms.
'internalv5'	Viewer based on MATLAB figure windows.
'web'	Viewer is the default Web browser with the VRML plug-in.

---

**Note** If you are running the Simulink 3D Animation software on a Linux platform, the default viewer continues to be legacy viewer, which does not support MATLAB figures. If you have a Linux platform and want to access the MATLAB figure capability, you must install the hardware-accelerated OpenGL rendering engine. Otherwise, the Simulink 3D Animation software cannot properly display the viewer. You can then activate the viewer by using `vrsetpref` to set the `DefaultViewer` property to 'internalv5'.

---

**Editor** — Contains a path to the VRML editor executable file. When you use the `edit` command, Simulink 3D Animation runs the VRML editor executable with all parameters required to edit the VRML 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 VRML file name

For instance, a possible value for the `Editor` preference is

```
`%matlabroot\bin\win32\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

# vrgetpref

---

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

`vrsetpref`

**Purpose** Install and check Simulink 3D Animation components

**Syntax**

```
vrinstall('action')
vrinstall action
vrinstall('action','component')
vrinstall action component
x = vrinstall('action', 'component')
```

**Arguments**

*action* Type of action for this function. Values are -interactive, -selftest, -check, -install, and -uninstall.

*component* Name of the component for the action. Values are viewer and editor.

**Description** You use this function to manage the installation of optional software components related to the Simulink 3D Animation product. Currently there are two such components: VRML plug-in and VRML editor.

Action Value	Description
-selftest	Checks the integrity of the current installation. If this function reports an error, you should reinstall the Simulink 3D Animation software. The function <code>vrinstall</code> automatically does a self-test with any other actions.
-interactive	Checks for the installed components, and then displays a list of uninstalled components you can choose to install.
-check	Checks the installation of optional components. If the given component is installed, returns 1. If the given component is not installed, returns 0. If you do not specify a component, displays a list of components and their status.

# vrinstall

---

Action Value	Description
-install	Installs optional components. This action requires you to specify the component name. All components can be installed using this command, but some of them (currently only the plug-in) need to be uninstalled using the system standard uninstallation procedure.
-uninstall	Uninstalls optional components. This option is currently available for the editor only. Note that this action does not remove the files for the editor from the installation folder. It removes the editor registry information.  If you want to uninstall the VRML plug-in, exit the MATLAB software and, from the <b>Control Panel</b> window, select <b>Add/Remove Programs</b> .

## Examples

Install the VRML plug-in. This command starts the Blaxxun Contact install program and installs the plug-in to your default Web browser.

```
vrinstall -install viewer
```

Install the VRML editor. This command associates V-Realm Builder with the **Edit** button in the Block Parameters dialog boxes.

```
vrinstall -install editor
```

**Purpose** 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.

`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.

Method	Description
force	<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> .
pov	<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.
read	<code>[axes, buttons, povs] = read(joy)</code> reads the status of axes, buttons, and POVs of the specified joystick. <code>[axes, buttons, povs] = read(joy, forces)</code> applies feedback forces, in addition, to a force-feedback joystick.

where `joy` is the handle to the joystick object.



**Purpose**

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 **View** menu, click **Show Library Browser**.
- In the MATLAB Command Window, click the Simulink icon.

# vrnode

---

**Purpose** Create node or handle to existing node

**Syntax**

```
mynode = vrnode
mynode = vrnode([])
mynode = vrnode(vrworld_object, 'node_name')
mynode = vrnode(vrworld_object, 'node_name', 'node_type')
mynode = vrnode(vrworld_object, 'USE', othernode)
mynode = vrnode(parent_node, 'parent_field', 'node_name',
'node_type')
mynode = vrnode(parent_node, 'parent_field', 'USE',
'othernode')
```

**Arguments**

vrworld_object	Name of a vrworld object representing a virtual world.
node_name	Name of the node.
node_type	Type of the node.
parent_node	Name of the parent node that is a vrnode object.
parent_field	Name of the field of the parent node.
'USE'	Enables a USE reference to another node.
othernode	Name of another node for a USE reference.

**Description**

`mynode = vrnode` creates an empty vrnode handle that does not reference any node.

`mynode = vrnode([])` creates an empty array of vrnode handles.

`mynode = vrnode(vrworld_object, 'node_name')` creates a handle to an existing named node in the virtual world.

`mynode = vrnode(vrworld_object, 'node_name', 'node_type')` creates a new node called *node\_name* of type *node\_type* on the root of the virtual world. It returns the handle to the newly created node.

`mynode = vrnode(vrworld_object, 'USE', othernode)` creates a USE reference to the node `othernode` on the root of the world `vrworld_object`. It returns the handle to the virtual world to the original node.

`mynode = vrnode(parent_node, 'parent_field', 'node_name', 'node_type')` creates a new node called `node_name` of type `node_type` that is a child of the `parent_node` and resides in the field `parent_field`. It returns the handle to the newly created node.

`mynode = vrnode(parent_node, 'parent_field', 'USE', 'othernode')` creates a USE reference to the node `othernode` as a child of node `parentnode` and resides in the field `parentfield`. It returns the handle to the original node.

A `vrnode` object identifies a virtual world node in a way very similar to a handle. If you apply the `vrnode` method to a node that does not exist, the method creates a node, the `vrnode` object, and returns the handle to the `vrnode` object. If you apply the `vrnode` method to an existing node, the method returns the handle to the `vrnode` object associated with this node.

## Method Summary

Method	Description
<code>delete</code>	Remove <code>vrnode</code> object
<code>fields</code>	VRML 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 VRML fields with client

# vrnode

---

## **See Also**

`vrnode/delete` | `vrnode/get` | `vrnode/getfield` | `vrnode/set` |  
`vrnode/setfield` | `vrworld`

<b>Purpose</b>	Remove vrnode object
<b>Syntax</b>	<code>delete(vrnode_object)</code> <code>delete(n)</code>
<b>Arguments</b>	<code>vrnode_object</code> Name of a vrnode object.
<b>Description</b>	<code>delete(vrnode_object)</code> deletes the virtual world node. <code>delete(n)</code> 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.
<b>See Also</b>	<code>vrworld/delete</code>

# vrnode/fields

---

**Purpose** VRML 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 VRML fields of the node associated with the vrnode object in the MATLAB Command Window.

`x = fields(vrnode_object)` returns the VRML fields of the node associated with the vrnode object in a structure array. The resulting structure contains a field for every VRML field with the following subfields:

- `Type` is the name of the VRML field type, for example, 'MFString', 'SFColor'.
- `Access` is the accessibility description of the VRML data class, for example, 'eventIn', 'exposedField'.
- `Sync` is the synchronization status 'on' or 'off'. See also `vrnode/sync`.

**See Also** `vrnode/get` | `vrnode/set`

**Purpose** Property value of vrnode object

**Syntax**

```
get(vrnode_object)
x = get(vrnode_object)
x = get(vrnode_object, 'property_name')
```

**Arguments**

`vrnode_object` Name of a vrnode object representing the node to be queried.

`property_name` Name of the property to be read.

**Description** `get(vrnode_object)` lists all vrnode properties in the MATLAB Command Window.

`x = get(vrnode_object)`, where `vrnode_object` is a scalar, returns a structure where each field name is the name of a property and each field contains the value of that property.

`x = get(vrnode_object, 'property_name')` returns the value of given property.

If `vrnode_object` is a vector of vrnode handles, `get` returns an M-by-1 cell array of values, where M is equal to `length(vrnode_object)`.

The vrnode property values are case sensitive. Property names are not case sensitive.

The vrnode object properties allow you to control the behavior and appearance of objects. The vrnode objects have the following properties. All these properties are read only.

Property	Value	Description
Fields	Cell array	Valid field names for the VRML node.
Name	String	Name of the node.

## vrnode/get

---

Property	Value	Description
Type	String	VRML 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](#)



**Purpose**

Field value of vrnode object

**Syntax**

```
getfield(vrnode_object)
x = getfield(vrnode_object)
x = getfield(vrnode_object, 'fieldname')
```

**Arguments**

`vrnode_object` Name of a vrnode object representing the node to be queried.

`fieldname` Name of the vrnode object field whose values you want to query.

**Description**

`getfield(vrnode_object)` displays all the field names and their current values for the respective VRML 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.

---

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

---

**See Also**

[vrnode](#) | [vrnode/get](#) | [vrnode/set](#) | [vrnode/setfield](#)

# vrnode/isvalid

---

<b>Purpose</b>	1 if vrnode object is valid, 0 if not
<b>Syntax</b>	<code>x = isvalid(vrnode_object_vector)</code>
<b>Arguments</b>	<code>vrnode_object_vector</code> Name of an array of vrnode objects to be queried.
<b>Description</b>	<p>This method returns an array that contains 1 when the elements of <code>vrnode_object_vector</code> are valid vrnode objects, and 0 when they are not.</p> <p>The vrnode object is considered valid if the following conditions are met:</p> <ul style="list-style-type: none"><li>• The parent world of the node exists.</li><li>• The parent world of the node is open.</li><li>• The VRML node with the given vrnode handle exists in the parent world.</li></ul>
<b>See Also</b>	<code>vrfigure/isvalid</code>   <code>vrworld/isvalid</code>

**Purpose** Change property of virtual world node

**Syntax** `x = set(vrnode_object, 'property_name', 'property_value')`

**Arguments**

<code>vrnode_object</code>	Name of a vrnode 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 VRML node. Read only.
Name	String	Name of the node. Read only.
Type	String	VRML type of the node. The value is a string (for example, 'Transform', 'Shape'). Read only.
World	Handle	Handle of the parent vrworld object. This is a vrworld object that represents the node's parent world. Read only.

Currently, VRML nodes have no settable properties.

## vrnode/set

---

### **See Also**

`vrnode` | `vrnode/get` | `vrnode/getfield` | `vrnode/setfield`

**Purpose** 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 VRML 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 VRML 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`

# vrnode/sync

---

**Purpose** Enable or disable synchronization of VRML 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 VRML 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 VRML field is synchronized.

When the field is marked 'on', the field value is updated every time it is changed on the client computer. If the field is marked '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, you should 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.

Synchronization is meaningful only for readable fields. Readable fields are of VRML data class `eventOut` and `exposedField`. You cannot enable synchronization for `eventIn` or nonexposed fields.

**See Also** `vrnode` | `vrnode/get`

**Purpose** 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` | `vrrotmat2vec` | `vrrotvec` | `vrrotvec2mat`

# vrphysmod

---

**Purpose** Add virtual reality visualization framework to block diagrams

**Syntax**  
`vrphysmod(vrmlfile, model)`  
`vrphysmod(vrmlfile, subsystem)`

**Description** `vrphysmod(vrmlfile, model)` or `vrphysmod(vrmlfile, subsystem)` updates the model or subsystem that the SimMechanics `mech_import` function generates. As necessary, `vrphysmod` adds additional blocks to visualize the mechanical system in virtual reality. You can then save, rename, modify, and run the model. The `.wrl` extension for `vrmlfile` is optional. The association between mechanical system bodies and corresponding VRML nodes found in the VRML file is based on the name correspondence.

If your model contains several VR Sink blocks that refer to the same `vrmlfile`, this function attempts to consolidate the animation signals of that virtual scene into one VR Sink block.

---

**Note** The SolidWorks VRML export filter does not preserve part instance names and the part order in the resulting VRML 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 VRML file are identical to their corresponding bodies in the Simulink model before running this function.

If you receive this warning and the set of VRML 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 VRML 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 VRML file `four_link.wrl`:

```
vrphysmod('four_link.wrl', gcs);
```

## See Also

`stl2vrml` | `vrcadcleanup` | `mech_import`

# vrplay

**Purpose** Play VRML animation file

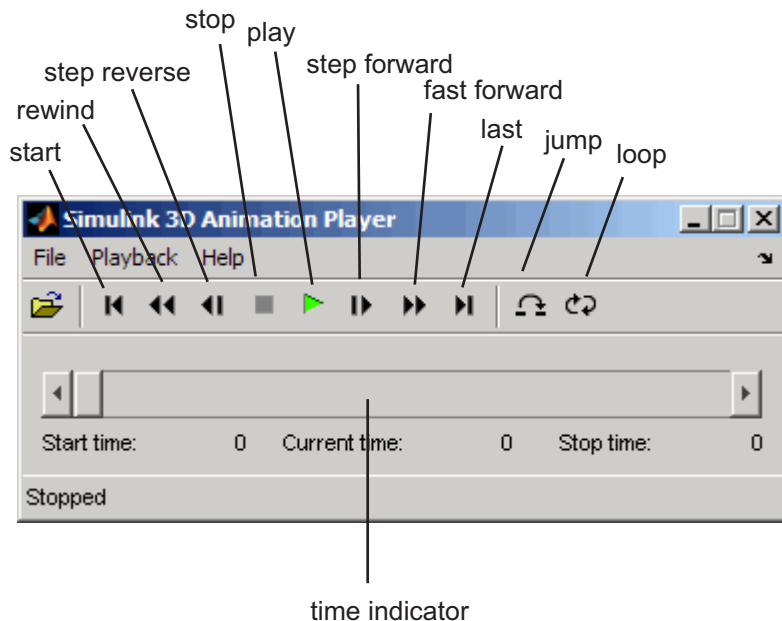
**Syntax**  
`vrplay`  
`vrplay(filename)`  
`x=vrplay(filename)`

**Description** `vrplay` opens the Virtual Reality animation player GUI that allows you to open and play VRML animation files.

`vrplay(filename)` opens the Virtual Reality animation player GUI and loads the virtual world `filename`.

`x=vrplay(filename)` also returns a Virtual Reality animation player GUI figure handle.

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



Setting the `DefaultViewer` property to `internalv4` affects the behavior of the `vrplay` function. 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.

If you set the `DefaultViewer` property to `internalv4`, the **File > New Window** command creates the new player window as in previous releases.

## Keyboard Support

The GUI's 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`

## How To

- “Recording Offline Animations” on page 4-10

# vrrotvec

---

**Purpose** 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` | `vrrotvec2mat`

<b>Purpose</b>	Convert rotation from matrix to axis-angle representation
<b>Syntax</b>	<pre>r = vrrotmat2vec(m) r = vrrotmat2vec(m,options)</pre>
<b>Description</b>	<p><code>r = vrrotmat2vec(m)</code> returns an axis-angle representation of rotation defined by the rotation matrix <code>m</code>.</p> <p><code>r = vrrotmat2vec(m,options)</code> converts the rotation with the default algorithm parameters replaced by values defined in <code>options</code>.</p> <p>The <code>options</code> structure contains the parameter <code>epsilon</code> that represents the value below which a number will be treated as zero (default value is <code>1e-12</code>).</p> <p>The result <code>r</code> 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.</p>
<b>See Also</b>	<code>vrrotvec</code>   <code>vrrotvec2mat</code>

# vrrotvec2mat

---

**Purpose** 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** `vrrotvec` | `vrrotmat2vec`

**Purpose** 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>DataTypeBool</code>	Specifies the handling of the VRML Bool data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are 'logical' and 'char'. If set to 'logical', the VRML Bool data type is returned as a logical value. If set to 'char', the Bool data type is returned 'on' or 'off'. Default is 'logical'.
<code>DataTypeInt32</code>	Specifies handling of the VRML Int32 data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are 'int32' and 'double'. If set to 'int32', the VRML Int32 data type is returned as int32. If set to 'double', the Int32 data type is returned as 'double'. Default is 'double'.

Preference	Description
<code>DataTypeFloat</code>	Specifies the handling of the VRML float data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are 'single' and 'double'. If set to 'single', the VRML 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'.
<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>• 'none' Panel is not visible.</li><li>• 'translucent' Panel floats half transparently above the scene.</li><li>• 'opaque' Panel floats above the scene.</li></ul> Default: 'none'
<code>DefaultCanvasUnits</code>	Specifies default units for new <code>vr.canvas</code> objects. See <code>vr.canvas</code> for detailed description. Default is 'normalized'.
<code>DefaultFigureAntiAliasing</code>	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'.



Preference	Description
DefaultFigureCapture FileName	Specifies default file name for capturing viewer figures. See <code>vrfigure/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'.
DefaultFigureMax TextureSize	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.

Preference	Description
DefaultFigureRecord2D CompressMethod	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'.
DefaultFigureRecord2D CompressQuality	Specifies the default quality of 2-D animation file compression for new <code>vrfigure</code> objects. Valid values are 0-100.
DefaultFigureRecord2D FileName	Specifies the default 2-D offline animation file name for new <code>vrfigure</code> objects.
DefaultFigureRecord2DFPS	Specifies the default frames per second playback speed.
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>vrfigure/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'.

Preference	Description
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	<p>Specifies which viewer is used to view a virtual scene.</p> <ul style="list-style-type: none"> <li>• 'internal'</li> </ul> <p>Default Simulink 3D Animation viewer. This is the same as setting the value to 'internalv5' for all platforms except the Linux platform.</p> <ul style="list-style-type: none"> <li>• 'internalv4'</li> </ul> <p>Legacy Simulink 3D Animation viewer.</p> <ul style="list-style-type: none"> <li>• 'internalv5'</li> </ul> <p>Viewer based on MATLAB figure windows.</p> <ul style="list-style-type: none"> <li>• 'web'</li> </ul> <p>Web browser becomes viewer. This is the current Web browser VRML plug-in.</p>

Preference	Description
	<p><b>Note</b> For Linux platforms, 'internal' sets the default viewer to the legacy viewer, and 'internalv5' sets the default viewer to the viewer integrated with MATLAB figures.</p>
DefaultWorldRecord3D FileName	Specifies the default 3-D animation file name for new vrworld objects.
DefaultWorldRecordMode	Specifies the default animation recording mode for new vrworld objects. Valid values are 'manual' and 'scheduled'.
DefaultWorldRecord Interval	Specifies the default start and stop times for scheduled animation recording for new vrworld objects. Valid value is a vector of two doubles.
DefaultWorldRemoteView	Specifies whether the virtual world is enabled by default for remote viewing for new vrworld objects. Valid values are 'off' and 'on'.
DefaultWorldTimeSource	Specifies the default source of the time for new vrworld objects. Valid values are 'external' and 'freerun'.
Editor	Path to the VRML editor. If this path is empty, the MATLAB editor is used.

Preference	Description
HttpPort	IP port number used to access the Simulink 3D Animation server over the Web via HTTP. If you change this preference, you must restart the MATLAB software before the change takes effect.
TransportBuffer	Length of the transport buffer (network packet overlay) for communication between the Simulink 3D Animation server and its clients.
TransportTimeout	Amount of time the Simulink 3D Animation server waits for a reply from the client. If there is no response from the client, the Simulink 3D Animation server disconnects from the client.
VrPort	IP port used for communication between the Simulink 3D Animation server and its clients. If you change this preference, you must restart the MATLAB software before the change takes effect.

When you use 'factory' as a single argument, all preferences are reset to their default values. If you use 'factory' for a preference value, that single preference is reset to its default.

The HttpPort, VrPort, and TransportBuffer preferences affect Web-based viewing of virtual worlds. DefaultFigurePosition and DefaultNavPanel affect the Simulink 3D Animation viewer. Changes to the HttpPort or VrPort preferences take effect only after you restart the MATLAB software.

DefaultFigureNavPanel — Controls the appearance of the navigation panel in the Simulink 3D Animation viewer. For example, setting

this value to 'translucent' causes the navigation panel to appear translucent.

**DefaultViewer** — Determines whether the virtual scene appears in the default Simulink 3D Animation viewer, based on MATLAB figure windows, legacy Simulink 3D Animation viewer, or in your Web browser.

DefaultViewer Setting	Description
'internal'	Default Simulink 3D Animation viewer. This is the same as setting the value to 'internalv5' for all platforms except the Linux platform.
'internalv4'	Simulink 3D Animation viewer that is the previous default viewer for most platforms. This viewer continues to be the default viewer for Linux platforms.
'internalv5'	Viewer based on MATLAB figure windows.
'web'	Viewer is the default Web browser with the VRML plug-in.

---

**Note** If you are running the Simulink 3D Animation software on a Linux platform, the default viewer continues to be legacy viewer, which does not support MATLAB figures. If you have a Linux platform and want to access the MATLAB figure capability, you must install the hardware-accelerated OpenGL rendering engine. Otherwise, the Simulink 3D Animation software cannot properly display the viewer. You can then activate the viewer by using `vrsetpref` to set the `DefaultViewer` property to 'internalv5'.

---

**Editor** — Contains a path to the VRML editor executable file. When you use the `edit` command, Simulink 3D Animation runs the VRML editor executable with all parameters required to edit the VRML 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 VRML file name

For instance, a possible value for the `Editor` preference is

```
`%matlabroot\bin\win32\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**

`vrgetpref`



**Purpose** 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.

Property	Description
LimitPosition	Enables mouse position limits. If false, the object ignores the UpperPositionLimit and LowerPositionLimit properties.
LowerPositionLimit	Position coordinates for the lower limit of the mouse.
NormalizeOutputAngle	Determines whether the integrated rotation angles should wrap on a full circle (360°). This is not used when you read the Output Type as Speed.
InitialPosition	Initial condition for integrated translations. This is not used when you set the Output Type to Speed.
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.

<b>Method</b>	<b>Description</b>
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 VRML 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 VRML.

# vrview

---

**Purpose** 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

**Purpose** 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

# vrwhos

---

**Purpose** 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

**Purpose**

Create new vrworld object associated with virtual world

**Syntax**

```
myworld = vrworld('filename')
myworld = vrworld('filename', 'new')
myworld = vrworld
myworld = vrworld([])
```

**Arguments**

filename	String containing the name of the VRML file from which the virtual world is loaded. 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 VRML file `filename` and returns its handle. If the virtual world already exists, a handle to the existing virtual world is returned.

`myworld = vrworld('filename', 'new')` creates a virtual world associated with the VRML file `filename` and returns its handle. It always creates a new virtual world object, even if another vrworld object associated with the same VRML file already exists.

`myworld = vrworld` creates an empty vrworld handle that does not refer to any virtual world.

`myworld = vrworld([])` 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 VRML file. The newly loaded virtual world is closed and must be opened before you can use it.

# vrworld

---

The vrworld object associated with a virtual world remains valid until you use either delete or vrclear.

## Examples

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

## Method Summary

Method	Description
addexternproto	Add externproto declaration to virtual world.
close	Close virtual world
delete	Remove virtual world from memory
edit	Open virtual world file in external VRML editor
get	Property value of vrworld object
isvalid	1 if vrworld object is valid, 0 if not
nodes	List nodes available in virtual world
open	Open virtual world
reload	Reload virtual world from VRML file
save	Write virtual world to VRML file
set	Change property values of vrworld object
view	View virtual world

## See Also

[vrworld/close](#) | [vrworld/delete](#) | [vrworld/open](#)



## Purpose

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 VRML 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.

# vrworld/close

---

**Purpose** 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`

**Purpose** 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** `vrclear` | `vrworld/close`

# vrworld/edit

---

**Purpose** Open virtual world file in external VRML editor

**Syntax** `edit(vrworld_object)`

**Arguments** `vrworld_object` A `vrworld` object representing a virtual world.

**Description** The `edit` method opens the VRML file associated with the `vrworld` object in a VRML editor. The `Editor` preference specifies the VRML editor to use. See `vrsetpref` for details on setting preferences.

The VRML editor saves any changes you make directly to a virtual world file. If the virtual world is open,

- Use the `save` command in the VRML 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 VRML file. Any changes you made in the editor are lost.

**See Also** `vrworld/reload` | `vrworld/save`

**Purpose** 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.

# vrworld/get

Property	Value	Description
Description	String. Default: automatically taken from the VRML file property title	Description of the virtual world as it appears on the main Web page. Read/write.
Figures	Vector of vrfigure objects	Vector of handles to Simulink 3D Animation viewer windows currently viewing the virtual world. Read only.
FileName	String	Name of the associated VRML 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 “Animation Recording File Tokens” on page 4-12. 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.

Property	Value	Description
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 MATLAB interface (simulation time).  If set to 'freerun', time in the scene advances independently based on the system timer. Read/write.
View	'off'   'on' Default: 'on'	Indicates an unviewable or viewable virtual world. Read/write.

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 VRML file containing a

## vrworld/get

---

**WorldInfo** node with a `title` property, the `Description` property is loaded from the VRML 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](#)



<b>Purpose</b>	1 if vrworld object is valid, 0 if not
<b>Syntax</b>	<code>x = isvalid(vrworld_object)</code>
<b>Arguments</b>	<code>vrworld_object</code> A vrworld object representing a virtual world.
<b>Description</b>	<p>A vrworld object is considered valid if its associated virtual world still exists.</p> <p><code>x = isvalid(vrworld_object)</code> returns an array that contains a 1 when the elements of <code>vrworld_object</code> are valid vrworld objects, and returns a 0 when they are not.</p> <p>You use this method to check whether the vrworld object is still valid. Using a <code>delete</code> or <code>vrclear</code> command can make a vrworld object invalid.</p>
<b>See Also</b>	<code>vrfigure/isvalid</code>   <code>vrnode/isvalid</code>

# vrworld/nodes

---

**Purpose** 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`

**Purpose** 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 VRML 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.wr1')
myworld2 = vrworld('vrpend.wr1')
```

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`

# vrworld/reload

---

**Purpose** Reload virtual world from VRML 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 VRML 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 VRML file.

**See Also** `vrworld/edit` | `vrworld/open` | `vrworld/save`

**Purpose** Write virtual world to VRML file

**Syntax** `save(vrworld_object, 'vrml_file')`

**Arguments**

<code>vrworld_object</code>	A <code>vrworld</code> object representing a virtual world.
<code>vrml_file</code>	Name of the VRML file to save the virtual world to.

**Description**

The `save` method saves the current virtual world to a VRML97 file. The virtual world must be open for you to use this method.

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.

**See Also** `vrworld/edit` | `vrworld/open` | `vrworld/reload`

# vrworld/set

---

**Purpose** Change property values of vrworld object

**Syntax** `set(vrworld_object, 'property_name', property_value)`

**Arguments**

- `vrworld_object` Name of a vrworld object representing a virtual world.
- `property_name` Name of the property.
- `property_value` New value of the property.

**Description** You can change the values of the read/write virtual world properties. The following are properties of vrworld objects. Names are not case sensitive.

Property	Value	Description
Clients	Scalar	Number of clients currently viewing the virtual world. Read only.
ClientUpdates	'off'   'on' Default: 'on'	Client cannot or can update the virtual scene. Read/write.
Description	String. Default: automatically taken from the VRML file property title	Description of the virtual world as it appears on the main Web page. Read/write.
Figures	Vector of vrfigure objects	Vector of handles to Simulink 3D Animation viewer windows currently viewing the virtual world. Read only.
FileName	String	Name of the associated VRML file. Read only.

Property	Value	Description
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 “Animation Recording File Tokens” on page 4-12. 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.

## vrworld/set

---

Property	Value	Description
TimeSource	'external'   'freerun' Default: 'external'	Source of the time for the virtual world. If set to 'external', time in the scene is controlled from the MATLAB interface (by setting the Time property) or the 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



---

<b>Purpose</b>	View virtual world
<b>Syntax</b>	<pre>view(vrworld_object) x = view(vrworld_object) x = view(vrworld_object, '-internal') x = view(vrworld_object, '-web')</pre>
<b>Arguments</b>	<code>vrworld_object</code> A vrworld object representing a virtual world.
<b>Description</b>	<p>The <code>view</code> method opens the default VRML viewer on the host computer and loads the virtual world associated with the <code>vrworld</code> object into the viewer window. You specify the default VRML viewer using the <code>DefaultViewer</code> preference. The virtual world must be open for you to use this method.</p> <p><code>x = view(vrworld_object)</code> opens the default VRML viewer on the host computer and loads the virtual world associated with the <code>vrworld</code> object into the viewer window. If the Simulink 3D Animation viewer is used, <code>view</code> also returns the <code>vrfigure</code> handle of the viewer window. If a Web browser is used, <code>view</code> returns an empty array of <code>vrfigure</code> handles.</p> <p><code>x = view(vrworld_object, '-internal')</code> opens the virtual world in the Simulink 3D Animation viewer.</p> <p><code>x = view(vrworld_object, '-web')</code> opens the virtual world in the Web browser.</p> <p>If the virtual world is disabled for viewing (that is, the <code>View</code> property for the associated <code>vrworld</code> object is set to <code>'off'</code>), the <code>view</code> method does nothing.</p>
<b>Examples</b>	<pre>myworld = vrworld('vrpend.wrl') open(myworld) view(myworld)</pre>
<b>See Also</b>	<code>vrview</code>   <code>vrworld</code>



**simulation**

The process of running a dynamic system in nonreal time to observe its behavior.

**virtual figure object**

A handle to a Simulink 3D Animation viewer window.

**virtual node object**

A handle to a node in a virtual world that allows access to the node's properties.

**Virtual Reality Modeling Language**

The specification for displaying three-dimensional objects using a VRML viewer.

**virtual world**

An imaginary world where you can navigate around objects in three dimensions.

**virtual world object**

A handle to a virtual world that allows you to interact with and control the world.

**VRML**

Virtual Reality Modeling Language. See "Virtual Reality Modeling Language (VRML)" on page 1-10.



## Symbols and Numerics

- 2-D AVI files
  - recording through MATLAB interface 4-11
  - recording through Simulink 3D Animation viewer 7-28
- 3-D VRML files
  - recording with MATLAB interface 4-11
  - recording with Simulink 3D Animation viewer 7-27
- 3D World Editor
  - VRML editor 5-4 6-2

## A

- adding Simulink 3D Animation blocks 3-2
- animation files
  - recording with MATLAB interface 4-10
  - recording with Simulink 3D Animation viewer 7-27
- associating virtual worlds with Simulink blocks 3-8

## B

- bitmap file formats 1-12
- Blaxxun Contact
  - creating virtual worlds 5-11
  - installing 2-16
  - known issue 2-18
  - VRML viewer 7-54
- bmp file formats 1-12
- bouncing ball
  - Simulink example 1-20

## C

- capturing frames 7-22
- car
  - MATLAB interface example 1-31
- CATIA

- exporting VRML models 5-46
- changing virtual world associated with Simulink block 3-8
- client computer
  - installation of VRML viewer (Windows) 2-44
  - system requirements 2-10
- closing virtual worlds 4-9
- components
  - client computer 2-44
  - host computer 2-12
- connecting Simulink model to a virtual world 5-20
- coordinate system
  - MATLAB 1-13
  - VRML 1-13
- creating vrworld object 4-2
- Cross Product
  - Simulink block 10-2

## D

- default editor
  - setting 2-26
- default viewer
  - setting 2-20
- deformation of a sphere example
  - adding Simulink 3D Animation blocks 5-9
  - connecting Simulink to a virtual world 5-20
  - creating a box in a virtual world 5-11
  - creating a sphere in a virtual world 5-11
  - defining the problem 5-7
- deleting virtual worlds 4-9
- displaying virtual worlds 3-11

## E

- editors
  - general 3-D 5-2
  - native VRML 5-2
  - uninstalling 2-42

## examples

- bouncing ball 1-20
- car 1-31
- deformation of a sphere 5-7
- geometry morphing 1-26
- heat transfer 1-31
- inverted pendulum 1-28
- lighting 1-22
- magnetic levitation 1-22
- magnetic levitation for Real-Time Windows Target 1-23
- manipulator with space mouse 1-23
- MATLAB interface 1-18
- plane taking off 1-29
- plane taking off with trajectory tracing 1-29
- rotating membrane 1-33
- Simulink interface 1-18
- solar system 1-28
- terrain visualization 1-34
- using MATLAB interface 1-31
- video output 1-27

**F**

## file format

- VRML 1-15

## files

- textures 1-12

## frame captures

- configuring 7-26
- creating 7-25
- introduction 7-22

## frames

- capture actions 7-27
- capturing 7-22
- configuring captures 7-26
- creating captures 7-25

## functions

- MATLAB interface 11-1
- vr cadcleanup 12-6

- vr close 12-17
- vr dir2ori 12-18
- vr gcbf 12-40
- vr gcf 12-41
- vr install 12-53
- vr joystick 12-55
- vr lib 12-57
- vr ori2dir 12-71
- vr physmod 12-72
- vr play 12-74
- vr rotmat2vec 12-77
- vr rotvec 12-76
- vr rotvec2mat 12-78
- vr setpref 12-79
- vr spacemouse 12-89
- vr view 12-92
- vr who 12-93
- vr whos 12-94

**G**

## geometry morphing

- Simulink example 1-26

## global coordinates

- Simulink example 1-25

**H**

## heat transfer

- MATLAB example 1-31

## history

- VRML 1-10

## host computer

- installing Simulink 3D Animation 2-12
- installing VRML editor (Windows) 2-25
- installing VRML viewer (UNIX) 2-19
- installing VRML viewer (Windows) 2-16
- required components 2-12
- Simulink 3D Animation viewer 2-15
- system requirements 2-8

**I**

installation  
    Blaxxun Contact 2-16  
    client computer 2-44  
    components  
        host computer 2-12  
    host computer 2-12  
    Simulink 3D Animation 2-12  
    supported platforms 2-6  
    system requirements 2-6  
    testing 2-45  
    viewer on host computer 2-15  
    VRML editor (Windows) 2-25  
    VRML viewer (UNIX) 2-19  
    VRML viewer (Windows) 2-16  
interacting with a virtual world 4-5  
interface overview 3-2  
inverted pendulum  
    Simulink example 1-28

**J**

Joystick Input  
    Simulink block 10-3

**L**

lighting  
    Simulink example 1-22

**M**

magnetic levitation  
    Simulink example 1-22  
    Simulink example for Real-Time Windows  
        Target 1-23  
manipulator with global coordinates  
    Simulink example 1-25  
manipulator with space mouse

    Simulink example 1-23  
MATLAB coordinate system 1-13  
MATLAB interface  
    creating a `vrworld` object 4-2  
    examples 1-31  
    interacting with a virtual world 4-5  
    table of general functions 11-1

**N**

native VRML 5-2  
navigation  
    about a virtual scene 7-14  
    example of navigation 7-18  
    keyboard 7-20  
    using the mouse 7-15  
navigation speed  
    changing 7-17  
network security setting 2-18  
    changing default 2-18  
    *See also* Blaxxun Contact  
Normalize Vector  
    Simulink block 10-6

**O**

opening a viewer window 3-13  
Orbisnap 8-1  
    installation 8-3  
    interface 8-10  
    remote viewing 8-7  
    usage 8-5  
    viewing virtual worlds 8-6  
overview  
    associating virtual worlds with Simulink 3-2  
    Simulink interface 3-2  
    virtual worlds 5-2  
    VRML 1-10  
    VRML editing tools 5-2

**P**

- plane taking off
  - Simulink example 1-29
- plane taking off with trajectory tracing
  - Simulink example 1-29
- platforms
  - supported 2-6
- preferences 2-29
  - See also* Simulink 3D Animation preferences

**R**

- rendering of a virtual scene 7-45
- rotating membrane
  - Simulink 3D Animation example 1-33
  - Simulink example 1-26
- Rotation Between 2 Vectors
  - Simulink block 10-7
- Rotation Matrix to VRML Rotation
  - Simulink block 10-8
- running Simulink example 2-45

**S**

- security settings
  - changing 2-18
- setting
  - default editor 2-26
  - default viewer 2-20
- simulation
  - displaying virtual worlds 3-11
  - starting 3-11
- Simulink 3-2
  - associating with virtual worlds 3-2
  - interface examples 1-18
  - See also* examples
- Simulink 3D Animation blocks
  - Cross Product 10-2
  - Joystick Input 10-3
  - Normalize Vector 10-6

- Rotation Between 2 Vectors 10-7
- Rotation Matrix to VRML Rotation 10-8
- Viewpoint Direction to VRML
  - Orientation 10-14
- VR Placeholder 10-15
- VR Signal Expander 10-16
- VR Sink 10-18
- VR Text Output 10-26
- VR to Video 10-28
- VR Tracer 10-29

- Simulink 3D Animation examples
  - car 1-31
  - heat transfer 1-31
  - rotating membrane 1-33
  - running and viewing 2-50
  - terrain visualization 1-34
- Simulink 3D Animation preferences
  - MATLAB GUI 2-29
  - vrsetpref* function 12-79
- Simulink 3D Animation stand-alone viewer 8-1
  - See also* Orbisnap
- Simulink 3D Animation viewer
  - changing navigation speed 7-17
  - introduction 7-4
  - navigation 7-14
  - rendering 7-45
  - viewpoint control 7-37
- Simulink blocks
  - adding Simulink 3D Animation blocks 3-2
  - changing virtual world association 3-8
  - Cross Product 10-2
  - Joystick Input 10-3
  - Normalize Vector 10-6
  - Rotation Between 2 Vectors 10-7
  - Rotation Matrix to VRML Rotation 10-8
  - Viewpoint Direction to VRML
    - Orientation 10-14
  - VR Placeholder 10-15
  - VR Signal Expander 10-16
  - VR Sink 10-18



- VR Text Output 10-26
- VR to Video 10-28
- VR Tracer 10-29
- Simulink interface examples
  - bouncing ball 1-20
  - deformation of a sphere 5-9
  - geometry morphing 1-26
  - global coordinates 1-25
  - inverted pendulum 1-28
  - lighting 1-22
  - magnetic levitation 1-22
  - magnetic levitation with Real-Time Windows Target 1-23
  - manipulator with global coordinates 1-25
  - manipulator with space mouse 1-23
  - plane taking off 1-29
  - plane taking off with trajectory tracing 1-29
  - rotating membrane 1-26
  - running and viewing 2-45
  - solar system 1-28
  - vehicle dynamics visualization 1-26
  - video output 1-27
- Simulink interface overview 3-2
- solar system
  - Simulink example 1-28
- space mouse
  - Simulink block 10-10
  - Simulink examples 1-23
- supported platforms 2-6
- system requirements
  - client computer 2-10
  - host computer 2-8
- T**
- terrain visualization
  - Simulink 3D Animation example 1-34
- testing
  - installation 2-45
  - MATLAB example 2-50
- Simulink example 2-45
- textures 1-12
- U**
- uninstalling
  - editor 2-42
  - Simulink 3D Animation 2-42
  - V-Realm Builder 2-42
  - VRML viewer (Windows) 2-42
- V**
- V-Realm Builder
  - installing 2-25
  - uninstalling 2-42
  - VRML editor 5-6
- vehicle visualization
  - Simulink example 1-26
- video output
  - Simulink example 1-27
- view a virtual world
  - using a Web browser on the client computer 3-18
  - using a Web browser on the host computer 3-14
- viewer
  - installation on client computer 2-44
  - installation on host computer 2-15
  - opening 3-13
- viewpoint control 7-37
- Viewpoint Direction to VRML Orientation
  - Simulink block 10-14
- virtual worlds
  - associating with Simulink 3-2
  - closing 4-9
  - deleting 4-9
  - displaying 3-11
  - interacting with 4-5
  - overview 5-2

- VR Placeholder
    - Simulink block 10-15
  - VR Signal Expander
    - Simulink block 10-16
  - VR Sink
    - Simulink block 10-18
  - VR Text Output
    - Simulink block 10-26
  - VR to Video
    - Simulink block 10-28
  - VR Tracer
    - Simulink block 10-29
  - vrcadcleanup
    - Simulink 3D Animation function 12-6
  - vrclose
    - Simulink 3D Animation function 12-17
  - vrdir2ori
    - Simulink 3D Animation function 12-18
  - vrgcbf
    - Simulink 3D Animation function 12-40
  - vrgcf
    - Simulink 3D Animation function 12-41
  - vrinstall
    - Simulink 3D Animation function 12-53
  - vrjoystick
    - Simulink 3D Animation function 12-55
  - vrlib
    - Simulink 3D Animation function 12-57
  - VRML
    - coordinate system 1-13
    - file format 1-15
    - history 1-10
    - overview 1-10
  - VRML editor
    - 3D World Editor 5-4 6-2
    - general 5-2
    - installing on host computer (Windows) 2-25
    - V-Realm Builder 5-6
  - VRML models
    - exporting from CATIA 5-46
  - VRML viewer
    - Blaxxun Contact 7-54
    - changing navigation speed 7-17
    - installing on client computer (Windows) 2-44
    - installing on host computer (UNIX) 2-19
    - installing on host computer (Windows) 2-16
    - known issue (Blaxxun Contact) 2-18
    - navigation 7-14
    - rendering 7-45
    - Simulink 3D Animation 7-4
    - uninstalling 2-42
    - viewpoint control 7-37
  - vrori2dir
    - Simulink 3D Animation function 12-71
  - vrphysmod
    - Simulink 3D Animation function 12-72
  - vrplay
    - Simulink 3D Animation function 12-74
  - vrrotmat2vec
    - Simulink 3D Animation function 12-77
  - vrrotvec
    - Simulink 3D Animation function 12-76
  - vrrotvec2mat
    - Simulink 3D Animation function 12-78
  - vrsetpref
    - Simulink 3D Animation function 12-79
  - vrspacemouse
    - Simulink 3D Animation function 12-89
  - vrview
    - Simulink 3D Animation function 12-92
  - vrwho
    - Simulink 3D Animation function 12-93
  - vrwhos
    - Simulink 3D Animation function 12-94
  - vrworld object
    - creation 4-2
- W**
- Web browser

viewing a virtual world on a client  
computer 3-18

viewing a virtual world on the host  
computer 3-14