

# Virtual Reality Toolbox

For Use with MATLAB® and Simulink®

- Computation
- Visualization
- Programming
- Simulation

## How to Contact The MathWorks:



www.mathworks.com      Web  
comp.soft-sys.matlab      Newsgroup



support@mathworks.com      Technical support  
suggest@mathworks.com      Product enhancement suggestions  
bugs@mathworks.com      Bug reports  
doc@mathworks.com      Documentation error reports  
service@mathworks.com      Order status, license renewals, passcodes  
info@mathworks.com      Sales, pricing, and general information



508-647-7000      Phone



508-647-7001      Fax



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

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

### *Virtual Reality Toolbox User's Guide*

© COPYRIGHT 2001-2004 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.

MATLAB, Simulink, Stateflow, Handle Graphics, and Real-Time Workshop are registered trademarks, and TargetBox is a trademark of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

|                   |              |                 |   |
|-------------------|--------------|-----------------|---|
| Printing 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) |

## Getting Started

### 1

|   |             |
|---|-------------|
| <b>What Is the Virtual Reality Toolbox?</b> .....       | <b>1-2</b>  |
| Expected Background .....                               | 1-3         |
| <b>Features of the Virtual Reality Toolbox</b> .....    | <b>1-4</b>  |
| VRML Support .....                                      | 1-4         |
| MATLAB Interface .....                                  | 1-5         |
| Simulink Interface .....                                | 1-6         |
| VRML Viewers .....                                      | 1-6         |
| VRML Editor .....                                       | 1-7         |
| Real-Time Workshop Support .....                        | 1-8         |
| SimMechanics Support .....                              | 1-8         |
| Hardware Support .....                                  | 1-8         |
| Client-Server Architecture .....                        | 1-8         |
| <b>VRML Overview</b> .....                              | <b>1-10</b> |
| VRML History .....                                      | 1-10        |
| VRML Coordinate System .....                            | 1-11        |
| VRML File Format .....                                  | 1-13        |
| <b>Examples Using the Virtual Reality Toolbox</b> ..... | <b>1-16</b> |
| Simulink Interface Examples .....                       | 1-16        |
| MATLAB Interface Examples .....                         | 1-24        |
| <b>Virtual Reality Toolbox Texture File</b> .....       | <b>1-27</b> |
| <b>Implementation Notes</b> .....                       | <b>1-28</b> |
| VRML Compatibility .....                                | 1-28        |
| Virtual Reality Toolbox Server .....                    | 1-29        |

|  |                 |
|--|-----------------|
| <b>Required Products</b> .....                                   | <b>2-2</b>      |
| MATLAB .....   | 2-2             |
| VRML Viewer .....  | 2-3             |
| <br><b>Recommended Product</b> .....                             | <br><b>2-4</b>  |
| Simulink .....   | 2-4             |
| <br><b>Related Products</b> .....                                | <br><b>2-5</b>  |
| <br><b>System Requirements</b> .....                             | <br><b>2-6</b>  |
| Supported Computer Platforms .....                               | 2-6             |
| Host Computer .....  | 2-7             |
| Client Computer .....  | 2-10            |
| <br><b>Installing the Virtual Reality Toolbox on the Host</b>    |                 |
| <b>Computer</b> .....  | <b>2-12</b>     |
| Getting or Updating Your License .....                           | 2-12            |
| Components on a Host Computer .....                              | 2-13            |
| Installing from CD (Windows) .....                               | 2-14            |
| Installing from CD (UNIX/Linux) .....                            | 2-15            |
| Downloading from the Web .....                                   | 2-16            |
| LD_LIBRARY_PATH Environment Variable (UNIX) .....                | 2-17            |
| Known Issue with the Virtual Reality Toolbox and Microsoft       |                 |
| Internet Explorer 6.0 (Windows) .....                            | 2-18            |
| <br><b>Installing the VRML Viewer on the Host Computer</b> ..... | <br><b>2-19</b> |
| Virtual Reality Toolbox Viewer .....                             | 2-19            |
| Installing a VRML Plug-In (Windows) .....                        | 2-20            |
| Installing a VRML Plug-In (UNIX/Linux) .....                     | 2-23            |
| Setting the Default Viewer of Virtual Scenes .....               | 2-24            |
| <br><b>Installing the VRML Editor on the Host Computer</b> ..... | <br><b>2-29</b> |
| Installing the VRML Editor (Windows) .....                       | 2-29            |
| VRML Editor (UNIX/Linux) .....                                   | 2-30            |
| Setting the Default Editor of Virtual Scenes .....               | 2-30            |

|   |             |
|---|-------------|
| <b>Removing Components (Windows)</b> .....                                  | <b>2-36</b> |
| Removing the Virtual Reality Toolbox and V-Realm<br>Builder (Windows) ..... | <b>2-36</b> |
| Removing the blaxxun Contact Plug-In (Windows) .....                        | <b>2-37</b> |
| <br><b>Installing on the Client Computer</b> .....                          | <b>2-38</b> |
| Installing a VRML Plug-In (Windows) .....                                   | <b>2-38</b> |
| <br><b>Testing the Installation</b> .....                                   | <b>2-39</b> |
| Running a Simulink Interface Example .....                                  | <b>2-39</b> |
| Running a MATLAB Interface Example .....                                    | <b>2-44</b> |

## Simulink Interface

# 3

|  |             |
|--|-------------|
| <b>Associating a Virtual World with Simulink</b> .....                     | <b>3-2</b>  |
| Adding a Virtual Reality Toolbox Block .....                               | <b>3-2</b>  |
| Changing the Virtual World Associated with a<br>Simulink Block .....       | <b>3-10</b> |
| <br><b>Using the Simulink Interface</b> .....                              | <b>3-12</b> |
| Displaying a Virtual World and Starting Simulation .....                   | <b>3-12</b> |
| Viewing a Virtual World with a Web Browser on the<br>Host Computer .....   | <b>3-15</b> |
| Viewing a Virtual World with a Web Browser on the<br>Client Computer ..... | <b>3-19</b> |

**4**

|   |             |
|---|-------------|
| <b>Using the MATLAB Interface</b> .....   | <b>4-2</b>  |
| Creating a vrworld Object .....   | 4-2         |
| Opening a Virtual World .....   | 4-3         |
| Interacting with a Virtual World .....  | 4-5         |
| Closing and Deleting a vrworld Object .....   | 4-8         |
| <br>  |             |
| <b>Recording Offline Animations</b> .....   | <b>4-10</b> |
| Animation Recording File Tokens .....   | 4-12        |
| Manual 3-D VRML Animation Recording .....   | 4-14        |
| Manual 2-D AVI Animation Recording .....  | 4-17        |
| Scheduled 3-D VRML Animation Recording .....  | 4-19        |
| Scheduled 2-D AVI Animation Recording .....   | 4-22        |
| Viewing Animation Files .....   | 4-24        |
| MATLAB Animation Recording of Virtual Worlds Not<br>Associated with Simulink Models ..... | 4-26        |

**Virtual Worlds**

**5**

|  |             |
|--|-------------|
| <b>VRML Editing Tools</b> .....                      | <b>5-2</b>  |
| Editors for Virtual Worlds .....                     | 5-2         |
| V-Realm Builder .....                                | 5-4         |
| <br>   |             |
| <b>Deformation of a Sphere Example</b> .....         | <b>5-5</b>  |
| Defining the Problem .....                           | 5-5         |
| Adding a Virtual Reality Toolbox Block .....         | 5-6         |
| Creating a Sphere in a Virtual World .....           | 5-8         |
| Creating a Box in a Virtual World .....              | 5-13        |
| Connecting a Simulink Model to a Virtual World ..... | 5-16        |
| <br>   |             |
| <b>VRML Data Types</b> .....                         | <b>5-20</b> |
| VRML Field Data Types .....                          | 5-20        |
| VRML Data Class Types .....                          | 5-24        |

|  |             |
|--|-------------|
| <b>Virtual Reality Toolbox Viewer</b> .....                  | <b>6-2</b>  |
| Menu Bar .....   | <b>6-4</b>  |
| Toolbar .....  | <b>6-5</b>  |
| Navigation Panel .....                                       | <b>6-5</b>  |
| Starting and Stopping Simulations .....                      | <b>6-9</b>  |
| Navigation .....   | <b>6-10</b> |
| Configuring Animation Recording Parameters .....             | <b>6-17</b> |
| Recording Files in the VRML Format .....                     | <b>6-21</b> |
| Recording Files in the Audio Video Interleave (AVI) Format . | <b>6-22</b> |
| Scheduling Files for Recording .....                         | <b>6-24</b> |
| Interactively Starting and Stopping Animation Recording ...  | <b>6-26</b> |
| Viewing the Animation File .....                             | <b>6-27</b> |
| Working with Viewpoints .....                                | <b>6-28</b> |
| Rendering .....  | <b>6-35</b> |
| <br>   |             |
| <b>blaxxun Contact VRML Plug-In</b> .....                    | <b>6-44</b> |
| Viewpoint Control .....                                      | <b>6-44</b> |
| Control Menu .....   | <b>6-45</b> |
| Navigation .....   | <b>6-45</b> |
| Movement Modes .....   | <b>6-46</b> |
| blaxxun Contact Settings .....                               | <b>6-47</b> |
| Stereoscopic Vision .....                                    | <b>6-48</b> |

**Block Reference**

**7**

---

|   |                |
|---|----------------|
| <b>Blocks — Categorical List</b> .....      | <b>7-2</b>     |
| Control Input Devices .....                 | 7-2            |
| Virtual Worlds .....                        | 7-2            |
| VRML Related Signals .....                  | 7-2            |
| <br><b>Blocks — Alphabetical List</b> ..... | <br><b>7-3</b> |

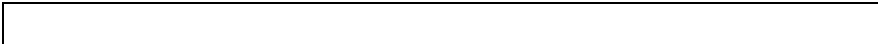
**Function Reference**

**8**

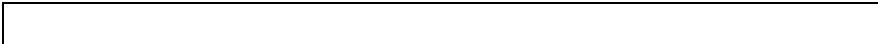
---

|  |                |
|--|----------------|
| <b>Functions — Categorical List</b> .....      | <b>8-2</b>     |
| MATLAB Interface Functions .....               | 8-3            |
| vrworld Object Methods .....                   | 8-3            |
| vrnode Object Methods .....                    | 8-4            |
| vrfigure Object Methods .....                  | 8-4            |
| <br><b>Functions — Alphabetical List</b> ..... | <br><b>8-5</b> |

**Glossary**



**Index**





# Getting Started

---

The Virtual Reality Toolbox allows you to connect an existing virtual world, defined with VRML, to Simulink® and MATLAB®. Understanding the features of the Virtual Reality Toolbox and some basic VRML concepts will help you to use this product more effectively.

|   |   |
|---|---|
| What Is the Virtual Reality Toolbox?<br>(p. 1-2)        | Solution for virtual interaction with models of dynamic systems over time                                       |
| Features of the Virtual Reality Toolbox<br>(p. 1-4)     | Description of the many features available to create and view dynamic systems                                   |
| VRML Overview (p. 1-10)                                 | Brief history of VRML, differences between the VRML and MATLAB coordinate systems, and the format of VRML files |
| Examples Using the Virtual Reality Toolbox<br>(p. 1-16) | VRML worlds with an interface to Simulink block diagrams and an interface to MATLAB objects and functions       |
| Virtual Reality Toolbox Texture File<br>(p. 1-27)       | Virtual Reality Toolbox texture file usage recommendations  |
| Implementation Notes (p. 1-28)                          | Outline of the Virtual Reality Toolbox server and VRML compatibility  |

## What Is the Virtual Reality Toolbox?

The Virtual Reality Toolbox is a solution for interacting with virtual reality models of dynamic systems over time. It extends the capabilities of MATLAB and Simulink into the world of virtual reality graphics.

- **Virtual worlds** — Create virtual worlds or three-dimensional scenes using standard Virtual Reality Modeling Language (VRML) technology.
- **Dynamic systems** — Create and define dynamic systems with MATLAB and Simulink.
- **Animation** — View moving three-dimensional scenes driven by signals from the Simulink environment.
- **Manipulation** — Change the positions and properties of objects in a virtual world while running a simulation.

To provide a complete working environment, the Virtual Reality Toolbox includes additional components:

- **VRML viewer** — Use either the Virtual Reality Toolbox viewer or, for PC platforms, the blaxxun Contact plug-in for Web browsers to display your virtual worlds.
- **VRML editor** — For PC platforms, use V-Realm Builder to create and edit VRML code. For UNIX or Linux platforms, use the MATLAB text editor to write VRML code to create virtual worlds.

## 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 the Virtual Reality Toolbox on the Mac OS X platform works as described for the UNIX/Linux platforms.

This guide assumes that you are already familiar with

- MATLAB, to write scripts and functions with M-code, and to use functions with the command-line interface
- Simulink and Stateflow<sup>®</sup> 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

**If You Are a New User** — You might want to review

- Chapter 1, “Getting Started” — This chapter gives you an overview of the Virtual Reality Toolbox features.
- Chapter 3, “Simulink Interface” — Interact with a virtual world from Simulink.
- Chapter 4, “MATLAB Interface” — Interact with a virtual world from MATLAB.

**If You Are an Experienced Virtual Reality Toolbox User** — You might want to review

- Chapter 7, “Block Reference” — Additional functionality has been added to the Virtual Reality Toolbox library.
- “vrworld Object Methods” in Chapter 8 — Description of vrworld object properties and methods.
- “vrnode Object Methods” in Chapter 8 — Description of vrnode object properties and methods.
- “vrfigure Object Methods” in Chapter 8 — Description of vrfigure object properties and methods.

## Features of the Virtual Reality Toolbox

The Virtual Reality Toolbox includes many features for you to create and visualize virtual reality models of dynamic systems. It also provides real-time virtual interaction with dynamic models.

This section includes the following topics that describe these features:

- “VRML Support” on page 1-4 — Use VRML to define a virtual world
- “MATLAB Interface” on page 1-5 — Control the virtual world from the MATLAB interface
- “Simulink Interface” on page 1-6 — Use Virtual Reality Toolbox blocks to connect your Simulink model to a virtual world
- “VRML Viewers” on page 1-6 — View your virtual world with the Virtual Reality Toolbox viewer or your Web browser
- “VRML Editor” on page 1-7 — Create virtual worlds using a VRML authoring tool or text editor
- “Real-Time Workshop Support” on page 1-8 — Support for simulations that use code generated by Real-Time Workshop®
- “SimMechanics Support” on page 1-8 — View the behavior of your SimMechanics model in a virtual world
- “Hardware Support” on page 1-8 — Functions for using special hardware devices
- “Client-Server Architecture” on page 1-8 — Provide client-server architecture for a single computer or network operation

### 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 Virtual Reality Toolbox 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>. This format includes a description of 3-D scenes, sounds, internal actions, and WWW anchors.

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

The Virtual Reality Toolbox viewer supports the majority of VRML97 standard nodes, allowing you almost complete control over associated virtual worlds. The blaxxun Contact plug-in supports all VRML97 standard nodes.

The Virtual Reality Toolbox makes sure that the changes made to a virtual world are reflected in MATLAB and Simulink. If you change the viewpoint in your virtual world, this change occurs in the `vrworld` object properties in MATLAB and Simulink.

The Virtual Reality Toolbox includes functions for retrieving and changing virtual world properties.

---

**Note** Since some VRML worlds are automatically generated in VRML1.0, and the Virtual Reality Toolbox 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 V-Realm Builder and saving them. V-Realm Builder is shipped with the PC version of the Virtual Reality Toolbox. Other commercially available software programs can also perform the VRML1.0 to VRML97 conversion.

---

## **MATLAB Interface**

The Virtual Reality Toolbox 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 MATLAB, 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 Virtual Reality Toolbox 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.

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

## Simulink Interface

With a Simulink model, you can observe a simulation of your dynamic system over time in a visually realistic 3-D model.

The Virtual Reality Toolbox 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 Virtual Reality Toolbox 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 Virtual Reality Toolbox automatically scans a virtual world for available VRML nodes that Simulink 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 Simulink. After you close a **Block Parameters** dialog box, Simulink 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.

Simulink provides communication for control and manipulation of virtual reality objects, using Virtual Reality Toolbox blocks.

## VRML Viewers

The Virtual Reality Toolbox contains a viewer that is the default viewing method for virtual worlds. This Virtual Reality Toolbox viewer is supported on PC, UNIX, Mac OS X, and Linux platforms.

If you are on a PC platform, you can install a VRML plug-in and view a virtual world in your preferred Web browser. For PC platforms, the Virtual Reality Toolbox includes the VRML plug-in blaxxun Contact. This is the only supported VRML plug-in.

If you install the VRML plug-in, the Virtual Reality Toolbox connects MATLAB and Simulink with the VRML-enabled browser to display a simulated process 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.

## VRML Editor

For PC platforms, the Virtual Reality Toolbox includes one of the classic VRML authoring tools, V-Realm Builder by Ligos Corp. With the addition of this VRML authoring tool, the Virtual Reality Toolbox provides a complete authoring, development, and working environment for carrying out 3-D visual simulations.

You use a VRML editor to create the virtual worlds you connect to Simulink block diagrams:

- **PC platforms** — V-Realm Builder Version 2.0 is included with the Virtual Reality Toolbox. If you do not want to use V-Realm Builder, you can use your favorite VRML editor.

Use the command `vrinstall` to install the editor before editing a virtual world. See “Installing the VRML Editor (Windows)” on page 2-29.

For information on using V-Realm Builder with the Virtual Reality Toolbox, see Chapter 5, “Virtual Worlds.”

- **UNIX/Linux platforms** — The default VRML editor for UNIX/Linux platforms is the MATLAB editor. If you do not want to use the MATLAB editor, you can set the Editor preference to your favorite text editor.

V-Realm Builder is the only supported VRML editor. It is provided with the PC version of the Virtual Reality Toolbox.

## Real-Time Workshop Support

The Virtual Reality Toolbox seamlessly integrates with Real-Time Workshop targets. It supports simulations that use code generated by Real-Time Workshop and a third-party compiler on your desktop computer. The Virtual Reality Toolbox also supports code executed in real time on external target computers. It enables interaction with real-time code generated by Real-Time Workshop and compiled with a third-party C/C++ compiler.

## Real-Time Windows Target

The Simulink interface in the Virtual Reality Toolbox supports the Real-Time Windows Target. Using the Simulink external mode, you can interact with real-time code generated by Real-Time Workshop and compiled with a third-party C/C++ compiler in the Real-Time Windows Target environment. See the Real-Time Windows Target User's Guide documentation for further details.

## SimMechanics Support

You can use the Virtual Reality Toolbox to view the behavior of a model created with SimMechanics. First, you build a model of a machine in Simulink 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.

## Hardware Support

The Virtual Reality Toolbox contains functions for using special hardware devices, including Joystick and SpaceMouse. It can also connect to common hardware devices, including joysticks and Magellan SpaceMouse, using Simulink blocks.

## Client-Server Architecture

The Virtual Reality Toolbox connects MATLAB and Simulink to a VRML-enabled Web browser using the TCP/IP protocol. The toolbox can be used in two configurations:

- **Single computer** — MATLAB, Simulink, and the virtual reality representations run on the same host computer.



- **Network computer** — You can view an animated virtual world on a computer separate from the computer with the Virtual Reality Toolbox server. Multiple clients can be connected to one server. You can adjust parameters to tune network performance.

## VRML Overview

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

This section includes the following topics:

- “VRML History” on page 1-10 — Events leading up to the creation of the VRML97 standard.
- “VRML Coordinate System” on page 1-11 — The VRML coordinate system is different from the MATLAB coordinate system.
- “VRML File Format” on page 1-13 — VRML files use a hierarchical structure to describe three-dimensional objects and their movements.

## VRML History

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.

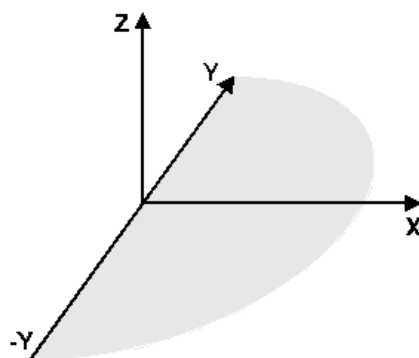
VRML97 represents an open and flexible platform for creating interactive three-dimensional scenes (virtual worlds). As computers improve in computational power and graphic capability, and communication lines become faster, the use of 3-D graphics becomes more popular outside the traditional domain of 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 Virtual Reality Toolbox 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.

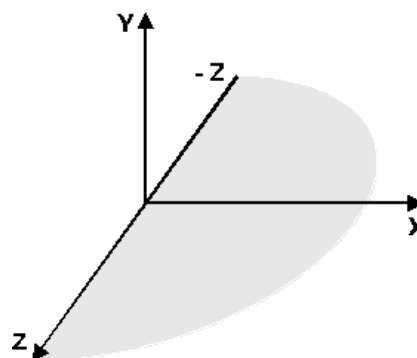
The VRML97 standard continues to be improved by the Web 3D Consortium. The newly released X3D (eXtensible 3D) standard is the successor to VRML97. X3D is an extensible standard that provides compatibility with existing VRML content and browsers. For more information, see <http://www.web3d.org>.

## VRML Coordinate System

VRML uses the right-handed *Cartesian coordinate system*. If your thumb, index finger, and middle finger of the right hand are held so that they form three right angles, then your thumb symbolizes the  $x$ -axis, your index finger the  $y$ -axis (pointing up), and your middle finger the  $z$ -axis.



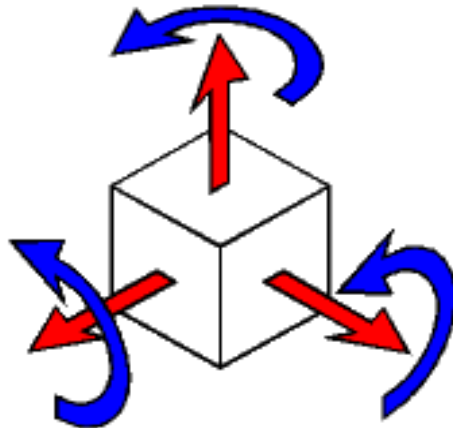
MATLAB graphics coordinate system



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 VRML coordinate system.

**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 to create virtual worlds more effectively, and gives you a good understanding of how the virtual world elements can be controlled using the Virtual Reality Toolbox.

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.

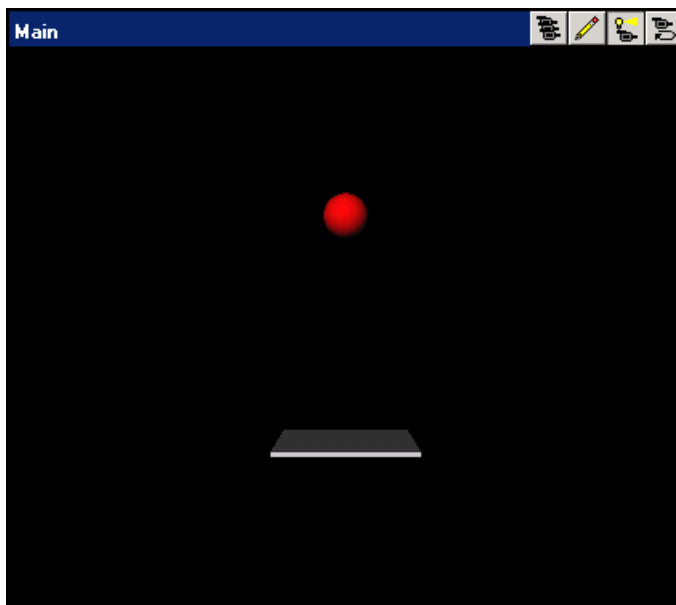
In the following example of a simple VRML file, two graphical objects are modeled in a 3-D scene: A floor is represented by a flat box with a red ball above it. Note that the VRML file is a readable text file that you can write in any text editor.

```
#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 V-Realm Builder, the floor and red ball look like



## Examples Using the Virtual Reality Toolbox

The Virtual Reality Toolbox includes examples using both the Simulink and MATLAB interfaces. You can use these examples to learn what you can do with the Virtual Reality Toolbox.

This section includes the following topics:

- “Simulink Interface Examples” on page 1-16 — Examples that use the VR Sink block in Simulink block diagrams
- “MATLAB Interface Examples” on page 1-24 — Examples that use MATLAB objects to interact with a virtual world

### Simulink Interface Examples

For all the examples that have a Simulink model, use the following procedure to view a virtual world:

- 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 opens in the Virtual Reality Toolbox viewer or your VRML-enabled Web browser. If the viewer does not appear, double-click the VR Sink block in the Simulink model.

- 2** In the Virtual Reality Toolbox viewer, from the **Simulation** menu, click **Block Parameters**.

A **Block Parameters** dialog box opens. Note that the **Open VRML viewer automatically** check box is selected by default for all Virtual Reality Toolbox demos.

If you close the virtual world window, you can display it again by double-clicking on the VR Sink block.



- 3** In the Simulink window, from the **Simulation** menu, click **Start**.  
(Alternatively, in the Virtual Reality Toolbox 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 Virtual Reality Toolbox. Descriptions of the examples follow the table.

| <b>Example</b>   | <b>RTW Ready</b> | <b>VR Sink</b> | <b>Joystick</b> | <b>SpaceMouse</b> |
|------------------|------------------|----------------|-----------------|-------------------|
| vrbounce         | X                | X              |                 |                   |
| vrcrane_joystick |                  | X              | X               |                   |
| vrcrane_traj     |                  | X              |                 |                   |
| vrlights         |                  | X              |                 |                   |
| vrmaglev         | X                | X              |                 |                   |
| vrmaglev_rtwin   | X                | X              |                 |                   |
| vrmanipul        |                  | X              |                 | X                 |
| vrmemb1          |                  | X              |                 |                   |
| vr_octavia       | X                | X              |                 |                   |
| vrpend           | X                | X              |                 |                   |
| vrplanets        | X                | X              |                 |                   |
| vrtkoff          |                  | 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 Simulink 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.wrl`.

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

The `vrcrane_traj` example is based on the `vrcrane_joystick` demo, but instead of interactive control, it has a predefined load trajectory. The `vrcrane_traj` model illustrates a technique to create the visual impression of joining and splitting moving objects in the 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 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 demonstrates light sources. In the scene, you can move Sun (modeled as `DirectionalLight`) and Lamp (modeled as `PointLight`) objects around the Simulink model. This 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 Simulink 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.

The position of the ball responds to the changing value of the set point. You can observe this change not only in the Scope window, but also with a VRML viewer displaying the virtual world. To display the virtual world, double-click the VR Sink block, then click the **View** button in the dialog box.

### **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. The MathWorks created this model to work with Real-Time Workshop, Real-Time Windows Target, and the HUMUSOFT MF 614 data acquisition board. However, you can adapt this model for other targets and acquisition boards. A digital IIR filter, from the Signal Processing Blockset, filters the physical system output. You can bypass the physical system by using the built-in plant model.

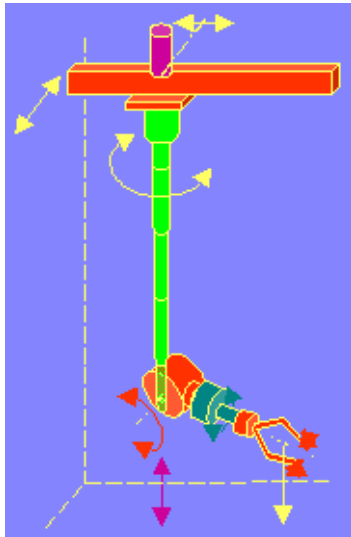
Running this model in real time is an example showing the capabilities of Simulink in control systems design and rapid prototyping.

Note that 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 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/rendering over two machines.

### **Manipulator with SpaceMouse Example (vrmanipul)**

The vrmanipul example illustrates the use of the Virtual Reality Toolbox for virtual reality prototyping and testing the viability of designs before the implementation phase. Also, this example illustrates the use of the Magellan SpaceMouse for manipulating objects in a virtual world. Note that you must have the Magellan SpaceMouse to run this demo.



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

Magellan SpaceMouse is an input device with six degrees of freedom. It is useful for navigating and manipulating objects in a virtual world. SpaceMouse is also suitable as a general input device for Simulink models. This professional device greatly facilitates all the previously mentioned tasks. You can use the SpaceMouse for higher performance applications and user comfort. SpaceMouse is supported through the Magellan Space Mouse input block, which is included in the Virtual Reality Toolbox block library for Simulink.

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

- Speeds
- Positions
- Viewpoint coordinates

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

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

### **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 demonstrates the Virtual Reality Toolbox 3-D off-line animation recording functionality.

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

The vrpend example illustrates the various ways a dynamic model in Simulink 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 scene, 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 demonstrates 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 V-Realm Builder Object Library is incorporated into the scene. The file `vrtkoff2.wrl` uses the same scene with a different type of aircraft.

## MATLAB Interface Examples

The following table is a list of the MATLAB interface examples provided with the Virtual Reality Toolbox. Descriptions of the examples follow the table.

| Example     | Moving Objects | Morphing Objects | Text | Recording | vrml() Function Use |
|-------------|----------------|------------------|------|-----------|---------------------|
| vracar      | X              |                  |      |           |                     |
| vrheat      |                | X                | X    |           |                     |
| vrheat_anim |                | X                | X    | X         |                     |
| vrmemb      | X              |                  | X    |           | X                   |

### Car in the Mountains Example (vracar)

This demonstration illustrates the use of the Virtual Reality Toolbox 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

```
vracar
```

A tutorial script starts running. Follow the instructions in the MATLAB Command Window.

### Heat Transfer Example (vrheat)

This demonstration illustrates the use of the Virtual Reality Toolbox with the MATLAB interface for manipulating complex objects.

In this demonstration, matrix-type data is transferred between MATLAB 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 demonstration. 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 demonstration illustrates the use of the Virtual Reality Toolbox 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. Note that you should use the Virtual Reality Toolbox 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 Virtual Reality Toolbox 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 Virtual Reality Toolbox 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.

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

This is a step-by-step demonstration. 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 directory for later use.

### **Rotating Membrane with MATLAB GUI Example (vrmemb)**

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

After starting the demo, 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).

## Virtual Reality Toolbox Texture File

The following are texture file recommendations for the Virtual Reality Toolbox:

- Where possible, scale source texture files to a size equal to the power of 2 in both dimensions. Doing so ensures optimal performance for the Virtual Reality Toolbox viewer. If you do not perform this scaling, the Virtual Reality Toolbox 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.

## Implementation Notes

This section includes the following topics:

- “VRML Compatibility” on page 1-28 — Limitations on support for VRML97 features
- “Virtual Reality Toolbox Server” on page 1-29 — Accesses information about VRML scenes, provides an interface between MATLAB and Simulink, and communicates with clients

### VRML Compatibility

The Virtual Reality Toolbox currently supports most features of VRML97, with the following limitations:

- The Virtual Reality Toolbox 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 Virtual Reality Toolbox server.
- The Virtual Reality Toolbox server ignores the Inline node, but it passes the node to the viewer. Therefore, the viewer sees the complete virtual world with all included substructures, but the included parts are not accessible from the toolbox. In some rare cases, this limitation can render the virtual world unusable with the Virtual Reality Toolbox. This happens under either of the following conditions:
  - The virtual world contains a USE name reference to a node that is in the included part.
  - The virtual world contains an included part with a PROTO or EXTERNPROTO declaration that is referenced in the main virtual world file.
- In keeping with the VRML97 specification, the Virtual Reality Toolbox Viewer ignores BMP files. As a result, VRML scene textures might not display properly in the Virtual Reality Toolbox 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.

## Virtual Reality Toolbox Server

This note is applicable only if you are using blaxxun Contact as your VRML viewer.

The Virtual Reality Toolbox uses a Virtual Reality Toolbox HTTP server for communication between a VRML-enabled Web browser and the MATLAB/Simulink environment. It generates the main Virtual Reality Toolbox 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 Virtual Reality Toolbox is loaded into MATLAB. This happens whenever you use a Virtual Reality Toolbox 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 Virtual Reality Toolbox 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 **Virtual Reality Toolbox 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.



# Installation

---

The Virtual Reality Toolbox provides the files you need for installation on both your host computer and client computer.

|   |  |
|---|--|
| Required Products (p. 2-2)  | MATLAB, Web browser with VRML plug-in (optional)   |
| Recommended Product (p. 2-4)  | Simulink (optional) to use the Virtual Reality Toolbox   |
| Related Products (p. 2-5)   | Where to find information about other MathWorks products for use with the Virtual Reality Toolbox      |
| System Requirements (p. 2-6)  | Minimum hardware and software requirements to run the Virtual Reality Toolbox with MATLAB and Simulink |
| Installing the Virtual Reality Toolbox on the Host Computer (p. 2-12) | Install the Virtual Reality Toolbox on your desktop computer   |
| Installing the VRML Viewer on the Host Computer (p. 2-19)             | Install a viewer to view virtual worlds  |
| Installing the VRML Editor on the Host Computer (p. 2-29)             | Install VRML authoring tools to create virtual worlds  |
| Removing Components (Windows) (p. 2-36)                               | Uninstall the Virtual Reality Toolbox and its components   |
| Installing on the Client Computer (p. 2-38)                           | Install a viewer on another computer to view virtual worlds remotely                                   |
| Testing the Installation (p. 2-39)                                    | Open a Simulink model, display a virtual world, and run a simulation                                   |

## Required Products

The Virtual Reality Toolbox is part of a family of products from The MathWorks. You need to install some of these products and other third-party products to use the Virtual Reality Toolbox.

This section includes the following topics:

- **MATLAB** — Create objects in the MATLAB workspace, connect these objects to a virtual world, and then use a command-line interface to control and make changes to the virtual world.
- **VRML Viewer** — View virtual worlds described with VRML.

### MATLAB

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

---

**Note** Version 4.0.1 of the Virtual Reality Toolbox requires MATLAB Version 7.0.1. The product is also available for Web download.

---

**MATLAB documentation** — For information on using MATLAB, see the MATLAB documentation. It explains how to work with data and how to use the functions supplied with MATLAB. For a reference describing the functions specific to the Virtual Reality Toolbox, see “Functions — Categorical List” in Chapter 8 of this guide.



---

## VRML Viewer

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

- **Virtual Reality Toolbox viewer** — This viewer is installed with the Virtual Reality Toolbox and is the default viewer for virtual worlds. You can access this viewer from either a Virtual Reality Toolbox block in your Simulink model, or by using the `vrview` and `vrfigure` functions with MATLAB.  
The Virtual Reality Toolbox viewer is a client to the Virtual Reality Toolbox server. It does not require a Web browser and it is available on more platforms than any other VRML97 viewer. It is supported on PC, 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 Virtual Reality Toolbox. This VRML plug-in allows you to view virtual worlds in your Web browser. The blaxxun Contact plug-in is the only supported VRML plug-in.

You can view a virtual world in the Virtual Reality Toolbox viewer as soon as you install the Virtual Reality Toolbox. 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 2-20.

For information on using a Web browser to view virtual worlds, see “Testing the Installation” on page 2-39. The blaxxun Contact installation executable files are located at `C:\<MATLAB root>\toolbox\vr\blaxxun`.

---

**Note** Every VRML plug-in installs 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 Virtual Reality Toolbox viewer or the blaxxun Contact VRML plug-in on PC platforms. On UNIX and Linux platforms, use only the Virtual Reality Toolbox viewer.

---

## Recommended Product

Optionally, you can install Simulink to use the Virtual Reality Toolbox.

This section includes the following topic:

**Simulink** — Create a model of your physical system and controller using a block diagram, connect your block diagram to a virtual world, and then use the block diagram to make changes to your model and view those changes in the virtual world.

### Simulink

Simulink provides an environment where you model your physical system and controller as a block diagram. You create the block diagram by using a mouse to connect blocks and a keyboard to edit block parameters.

With the Virtual Reality Toolbox, you can interact with the VR representation of the model you created with Simulink blocks. You can visualize the simulation of your dynamic system over time.

---

**Note** Version 4.0.1 of the Virtual Reality Toolbox requires Simulink Version 6.1.

---

**Simulink documentation** — For information on using Simulink, see the Simulink documentation. It explains how to connect blocks, build models, and change block parameters. For a reference describing the Virtual Reality Toolbox blocks, see Chapter 7, “Block Reference,” in this guide.

## Related Products

The MathWorks provides several products that are especially relevant to the kinds of tasks you can perform with the Virtual Reality Toolbox.

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

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

## System Requirements

The Virtual Reality Toolbox has the same hardware requirements as MATLAB. It is a multiplatform product that runs on PC-compatible computers with Windows or Linux. It runs on Solaris hardware running UNIX, and also on Apple Power Macintosh hardware running Mac OS X. For a list of supported operating systems, see “Supported Computer Platforms” on page 2-6.

This section includes the following topics:

- “Supported Computer Platforms” on page 2-6 — Summary of the supported computer platforms and the viewer and editor that are provided for each of them.
- “Host Computer” on page 2-7 — Run MATLAB, Simulink, the Virtual Reality Toolbox, VRML editor, and VRML viewer (the Virtual Reality Toolbox viewer or Web browser with VRML plug-in).
- “Client Computer” on page 2-10 — Run a Web browser with a VRML plug-in.

### Supported Computer Platforms

The VR server is the part of the Virtual Reality Toolbox 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 Virtual Reality Toolbox 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>Virtual Reality Toolbox Viewer</b> | <b>VRML Editor</b> | <b>VRML Browser Plug-In</b> |
|---|------------------|---------------------------------------|--------------------|-----------------------------|
| Microsoft Windows NT 4.0, Windows XP, or Windows 2000 | Yes              | Yes                                   | V-Realm Builder*   | blaxxun Contact*            |
| Linux 2.4.x kernels                                   | Yes              | Yes                                   | MATLAB editor*     | No                          |
| Sun Solaris 2.8, 2.9                                  | Yes              | Yes                                   | MATLAB editor*     | No                          |
| HP-UX 11.00   | Yes              | Yes                                   | MATLAB editor*     | No                          |
| Power Macintosh G3 or G4 running OS X (10.2 or later) | Yes              | Yes                                   | MATLAB editor*     | No                          |

\* Distributed with the Virtual Reality Toolbox product.

## Host Computer

The host computer is a desktop computer where you install MATLAB, Simulink, the Virtual Reality Toolbox, a VRML editor and, optionally, a Web browser with a VRML plug-in. You can also install Real-Time Workshop with Real-Time Windows Target or xPC Target to run and view a real-time application.

The following table lists the minimum resources the Virtual Reality Toolbox requires on the host computer.

**Hardware Requirements**

| <b>Hardware</b>      | <b>Description</b>   |
|----------------------|--|
| CPU                  | 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<br>CD-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 Virtual Reality Toolbox requires on your host computer. For a list of optional software products related to the Virtual Reality Toolbox, see <http://www.mathworks.com/products/virtualreality/related.jsp>.

**Software Requirements**

| <b>Software</b>  | <b>Description</b>  |
|------------------|---|
| Operating system | Microsoft Windows NT 4.0, Windows XP, or Windows 2000<br>Sun Solaris 2.6, 2.7, 2.8<br>Linux 2.2.x or 2.4.x kernel<br>Mac OS X 10.2 or later<br>The TCP/IP protocol must be installed. |
| MATLAB           | Version 7.0.1.  |
| Simulink         | Version 6.1. Simulink is not required, but we highly recommend that you install it.   |

**Software Requirements (Continued)**

| <b>Software</b>         | <b>Description</b>   |
|-------------------------|--|
| Virtual Reality Toolbox | Version 4.0.1.   |
| VRML editor             | For Windows platforms, you can install the VRML editor (V-Realm Builder 2.0) provided with the Virtual Reality Toolbox. For UNIX/Linux, the default editor is the MATLAB editor. When you create VRML worlds on these operating systems, you can use any 3-D modeling tool with the VRML97 export capability.  |
| 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 Virtual Reality Toolbox viewer.</p> <p>Use Microsoft Internet Explorer 4.0 or higher, or Netscape Navigator 4.0 or higher with Java enabled.</p>   |
| VRML plug-in            | <p>If you are using a Web browser instead of the Virtual Reality Toolbox 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 Virtual Reality Toolbox.</p> <p>For information on how to install the blaxxun Contact plug-in, see “Installing a VRML Plug-In (Windows)” on page 2-20.</p> |

## Client Computer

You can use a client computer to view and control a virtual world. Because MATLAB or Simulink does not run on this computer, you need to connect to a host computer running a simulation or executable code. The host computer, through the VR 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 Virtual Reality Toolbox needs on the 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 Virtual Reality Toolbox requires on the client computer. You do not need to install the Virtual Reality Toolbox 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.

### Software Requirements

| Software         | Description  |
|------------------|--|
| Operating system | Microsoft Windows NT 4.0, Windows XP, or Windows 2000: the TCP/IP protocol must be installed.  |
| Web browser      | Use Microsoft Internet Explorer 4.0 or higher, or Netscape Navigator 4.0 or higher with Java enabled.  |
| VRML plug-in     | <p>VRML97 plug-in with External Authoring Interface 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 Virtual Reality Toolbox.</p> <p>For information on how to install the blaxxun Contact plug-in, see “Installing a VRML Plug-In (Windows)” on page 2-20.</p> |

## Installing the Virtual Reality Toolbox on the Host Computer

You might want to install the Virtual Reality Toolbox Version from a CD or from the MathWorks Web site. For Web downloads, you need your Access Login account. Before you install the Virtual Reality Toolbox, you need to get a valid license file and/or personal license password. For detailed information about the installation process, see the installation documentation for your platform.

This section contains the following topics:

- “Getting or Updating Your License” on page 2-12 — Valid license file and personal license password (PLP)
- “Components on a Host Computer” on page 2-13 — Description of the individual components used with the Virtual Reality Toolbox
- “Installing from CD (Windows)” on page 2-14 — PC installation procedure
- “Installing from CD (UNIX/Linux)” on page 2-15 — UNIX/Linux installation procedure
- “Downloading from the Web” on page 2-16— Downloading the product from the Web
- “LD\_LIBRARY\_PATH Environment Variable (UNIX)” on page 2-17 — Setting the library path environment variable
- “Known Issue with the Virtual Reality Toolbox and Microsoft Internet Explorer 6.0 (Windows)” on page 2-18 — Running the Virtual Reality Toolbox viewer with Microsoft Internet Explorer 6.0

### Getting or Updating Your License

Before you install the Virtual Reality Toolbox, you must have a valid license file and/or personal license password (PLP). The license file and/or personal license password identify the products you purchased from The MathWorks. These are the products you are permitted to install and use.

When you purchase a product, The MathWorks sends you a license file and/or personal license password (PLP) in an e-mail message. If you have not received a PLP number, contact The MathWorks.

|                  |   |
|------------------|---|
| <b>Internet</b>  | <a href="http://www.mathworks.com/accesslogin">http://www.mathworks.com/accesslogin</a><br><br>Log in to <b>Access Login</b> using your e-mail address and password. Go to the <b>My Licenses</b> panel to determine your PLP number. |
| <b>E-mail</b>    | <a href="mailto:service@mathworks.com">mailto:service@mathworks.com</a> . Include your license number.  |
| <b>Telephone</b> | 508-647-7000. Ask for Customer Service.   |
| <b>Fax</b>       | 508-647-7001. Include your license number.  |

## Components on a Host Computer

This section introduces you to the individual components of the Virtual Reality Toolbox: 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:

- **Virtual Reality Toolbox** — This component contains the core files that interconnect MATLAB and Simulink to VRML. This component is required for the Virtual Reality Toolbox to operate, and you must install it on the host computer. This component is *not* used on a client computer.
- **Virtual Reality Toolbox viewer** — This is a multiplatform VRML viewer that is included with the Virtual Reality Toolbox, 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 Virtual Reality Toolbox for Windows platforms. However, you can also use the Virtual Reality Toolbox viewer. 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. V-Realm Builder is included with the Virtual Reality Toolbox for Windows platforms. If you do not plan to edit virtual reality worlds or if you prefer to use a different VRML editor, you do not need to install it on your computer. For UNIX/Linux platforms, the MATLAB editor is the default VRML editor. This component is *not* used on a client computer.
- **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 Virtual Reality Toolbox 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 CD (Windows)

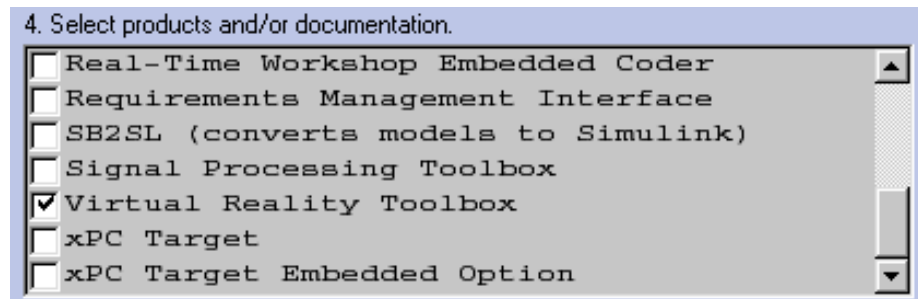
To install the Virtual Reality Toolbox from a CD on a Windows platform:

- 1 Insert the CD into your host CD-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 CD.

During the installation process, a screen similar to the following allows you to select the products to install.

- 2 Select the **Virtual Reality Toolbox** check box, then click **Next**.



- 3 Follow the instructions on each of the remaining screens.

Installation for the Virtual Reality Toolbox is complete.

The Virtual Reality Toolbox viewer is installed with the Virtual Reality Toolbox. 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 2-20.

If you are on a PC platform, you need to complete additional steps for installing the VRML editor. See “Installing the VRML Editor (Windows)” on page 2-29.

## Installing from CD (UNIX/Linux)

The following is an overview of how to install the Virtual Reality Toolbox on a UNIX/Linux platform from the CD. If you have not installed any MathWorks products before, consult the installation guide for your platform for a more comprehensive explanation of the installation process:

- 1 Log in to your system.
- 2 Mount the CD-ROM drive.
- 3 Create a directory to be the mount point for the CD-ROM drive. For example:
 

```
mkdir /cdrom
```
- 4 Create the installation directory and move into it using the `cd` command. For example, to install into the location `/usr/local/matlab7`, use these commands:

```
cd /usr/local
mkdir matlab7
cd matlab7
```

Subsequent instructions in this guide refer to this directory as \$MATLAB.

**Note** This installation directory might already exist if you have installed MATLAB on your system. In this case, move into the existing directory using the `cd` command.

- 5 Move your license file, named `license.dat`, into the \$MATLAB directory.

If you are upgrading an existing MATLAB installation, rename the license file in \$MATLAB/etc directory. The installer does not process the new license file if it finds an existing license file in \$MATLAB/etc.

- 6 Run the appropriate installation script for your platform.

`/cdrom/install* &` (Sun and Linux platforms)

- 7 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 Virtual Reality Toolbox is listed in this box.

- 8 Follow the instructions on each of the remaining screens.

Installation for the Virtual Reality Toolbox is complete.

The Virtual Reality Toolbox viewer is the default viewer for UNIX platforms. For more information, see “Virtual Reality Toolbox Viewer” on page 2-19.

If you are on a UNIX platform, the MATLAB editor is your default VRML editor. For more information, see “VRML Editor (UNIX/Linux)” on page 2-30.

## Downloading from the Web

The Virtual Reality Toolbox is available for Web download. You download products from the Web when you want to obtain a demo, product update, or any product available on a MATLAB installation CD:

- 1 Open your Web browser and navigate to <http://www.mathworks.com>.

- 2** From the list on the right side of the page, select **Downloads**.
- 3** Under **Access Login Users**, select **download products**.  
The **Access Login** page appears.
- 4** Enter your **E-mail Address** and **Password**.
- 5** Click **Log In**.  
The **downloads** page appears.
- 6** Select your platform and click **Continue**.
- 7** Select the **Virtual Reality Toolbox** and click **Continue**.
- 8** Follow the instructions on the **Download and Install** page to download and install the Virtual Reality Toolbox successfully. For more specific information relating to the installation of the Virtual Reality Toolbox, see the installation guide for your platform.

---

**Note** To get the latest PDF file for a product, go to <http://www.mathworks.com> and browse to the product's name. The Roadmap page for the selected product appears. This Roadmap page contains a link to the latest version of the PDF documentation.

---

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

If your system does not have OpenGL properly installed when you run the Virtual Reality Toolbox viewer, you might see an error message like the following in the MATLAB window:

```
Invalid MEX-file 'matlab/toolbox/vr/vr/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.

### **Known Issue with the Virtual Reality Toolbox and Microsoft Internet Explorer 6.0 (Windows)**

Microsoft Internet Explorer 6.0 might incorrectly interpret system Java library paths, preventing Virtual Reality Toolbox components (such as those for the Virtual Reality Toolbox viewer) from running properly. Netscape users do not experience this problem.

If you are using Internet Explorer 6.0, you should manually edit the Java library path for Microsoft Internet Explorer 6.0. Alternatively, you can also use Microsoft Internet Explorer 5.5 with the Virtual Reality Toolbox.

#### **Editing the Java Library Path**

To manually edit the Java library path for Microsoft Internet Explorer 6.0,

- 1 Run the `regedit` command.
- 2 Go to  
`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\JavaVM`  
  
A list of value names and their values appears.
- 3 Replace each instance of `%systemroot%` with the system root path. For example,  
`C:\WINNT`
- 4 Restart the computer.



## Installing the VRML Viewer on the Host Computer

You can use the Virtual Reality Toolbox viewer or VRML-enabled Web browser to view virtual worlds. The Virtual Reality Toolbox viewer is the only viewer that can be used on all supported platforms. The blaxxun Contact plug-in is available for PC platforms only.

This section includes the following topics:

- “Virtual Reality Toolbox Viewer” on page 2-19 — Preferred method of viewing virtual scenes.
- “Installing a VRML Plug-In (Windows)” on page 2-20 — Install the blaxxun Contact plug-in.
- “Installing a VRML Plug-In (UNIX/Linux)” on page 2-23 — Install a VRML97 plug-in with External Authoring Interface support.
- “Setting the Default Viewer of Virtual Scenes” on page 2-24 — View virtual scenes with the Virtual Reality Toolbox viewer or your VRML-enabled Web browser.

### Virtual Reality Toolbox Viewer

The Virtual Reality Toolbox viewer is the preferred method of viewing a virtual scene. The viewer can be used on any supported operating system. It is installed and set as the default viewer when you install the Virtual Reality Toolbox. You can view virtual scenes as soon as the Virtual Reality Toolbox is installed on your machine.

---

**Note** It is possible to view virtual scenes 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 in order to avoid Java errors and conflicts. For this reason, use only the Virtual Reality Toolbox viewer and the blaxxun Contact VRML plug-in on PC platforms. On UNIX and Linux platforms, use only the Virtual Reality Toolbox viewer.

---

## Installing a VRML Plug-In (Windows)

When you install the Virtual Reality Toolbox, the Virtual Reality Toolbox 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 Microsoft Internet Explorer or Netscape Navigator. 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:\<MATLAB root>\toolbox\vr\blaxxun.

---

You must use blaxxun Contact 4.4 with the Virtual Reality Toolbox. This version of the blaxxun Contact VRML plug-in is distributed with the Virtual Reality Toolbox. The following procedure describes how to install the blaxxun Contact VRML 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 MATLAB.

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

```
vrinstall -install viewer
```

MATLAB displays the message

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

**3** Check the graphic 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, MATLAB displays the following message:

```
VRML viewer:    installed
```

If the viewer installation was unsuccessful, MATLAB 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 scene when used with the Virtual Reality Toolbox and Microsoft Internet Explorer 5.5 and above. Netscape users do not experience this problem.

If you are using Internet Explorer 5.5 or above, you must manually change a network security setting before you can use blaxxun Contact 4.4 with the Virtual Reality Toolbox Version 3.0 or later. Upgrading your version of blaxxun Contact does not resolve this problem.

### **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 above to ensure that the virtual scene is updated appropriately:

- 1** Open Internet Explorer.
- 2** From the **Tools** menu, choose **Internet Options**.

The **Internet Options** dialog box opens.

- 3** Click the **Security** tab.
- 4** Click the **Custom Level** button.

The **Security Settings** dialog box opens.

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

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

- 7** Click **Java Custom Settings**.

The **Local intranet** dialog box opens.

- 8** Click the **Edit Permissions** tab.

- 9** Scan the main headings and subheadings (marked with a lock icon) until you see **Run Unsigned Content**.
- 10** Under **Run Unsigned Content**, find **Access to all Network Addresses**.
- 11** Under **Access to all Network Addresses**, select **Enable**.
- 12** Click **OK**.

The **Local intranet** dialog box closes.

- 13** In the **Security Settings** dialog box, click **OK**.

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

- 14** Select **Yes**.

- 15** In the **Internet Options** dialog box, click **OK**.

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

If you want to use a Web browser instead of the Virtual Reality Toolbox viewer to view virtual scenes, you need to install a VRML97 plug-in with External Authoring Interface (EAI) support. This requirement is met by blaxxun Contact for Windows platforms. If you are using any other operating system, you need to use the Virtual Reality Toolbox viewer to view virtual worlds.

---

**Note** blaxxun Contact is the only supported VRML plug-in.

---

## Setting the Default Viewer of Virtual Scenes

If you install a VRML plug-in in your Web browser, it is possible to view virtual scenes with either the Virtual Reality Toolbox viewer or your Web browser. You determine the viewer used to display your scene using the `vrsetpref` and `vrgetpref` commands. The following procedure assumes that you are working on a PC platform:

- 1 At the MATLAB command prompt, type

```
vrinstall -check
```

to determine whether blaxxun Contact is installed.

MATLAB 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 2-20.

- 2 Determine your default viewer by typing

```
vrgetpref
```

MATLAB displays

```

ans =

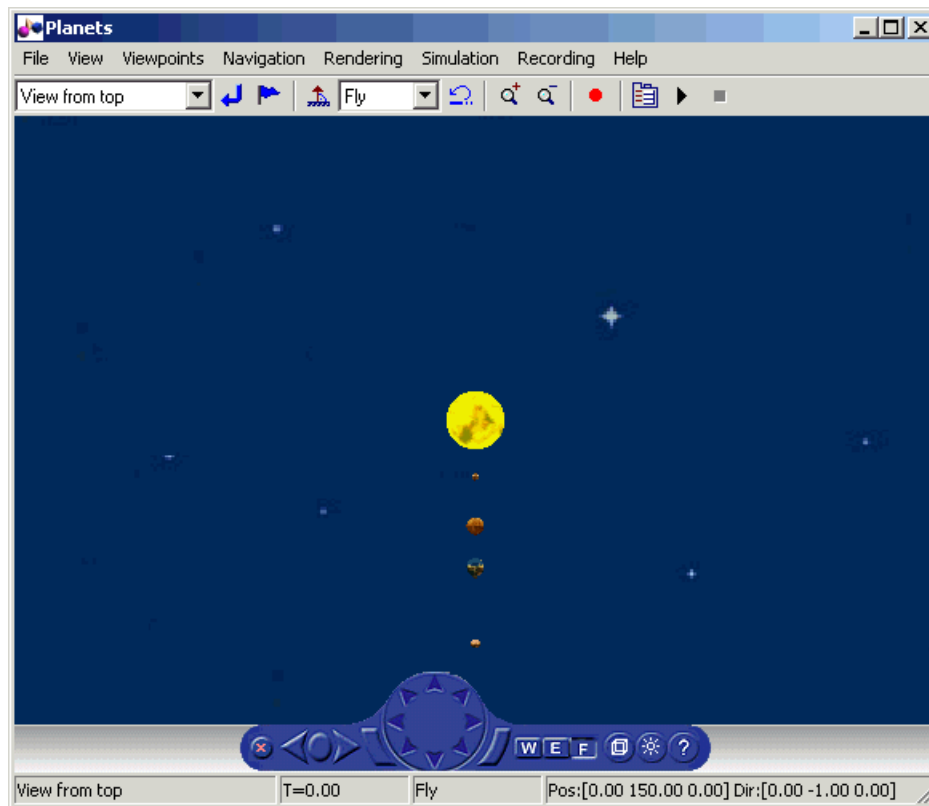
        DataTypeBool: 'logical'
        DataTypeInt32: 'double'
        DataTypeFloat: 'double'
    DefaultFigureAntialiasing: 'off'
        DefaultFigureDeleteFcn: ''
        DefaultFigureLighting: 'on'
    DefaultFigureMaxTextureSize: 'auto'
        DefaultFigureNavPanel: 'halfbar'
        DefaultFigureNavZones: 'off'
        DefaultFigurePosition: [5 92 576 380]
    DefaultFigureRecord2DCompressMethod: 'auto'
    DefaultFigureRecord2DCompressQuality: 75
    DefaultFigureRecord2DFileName: '%f_anim_%n.avi'
        DefaultFigureStatusBar: '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: 8123

```

The `DefaultViewer` property is set to `'internal'`. The Virtual Reality Toolbox viewer is the default viewer for viewing virtual scenes. Any virtual scenes that you open are displayed in the viewer.

- 3** For example, at the MATLAB command prompt, type  
`vrplanets`

The Planets demo is loaded and the virtual scene is displayed in the Virtual Reality Toolbox viewer.



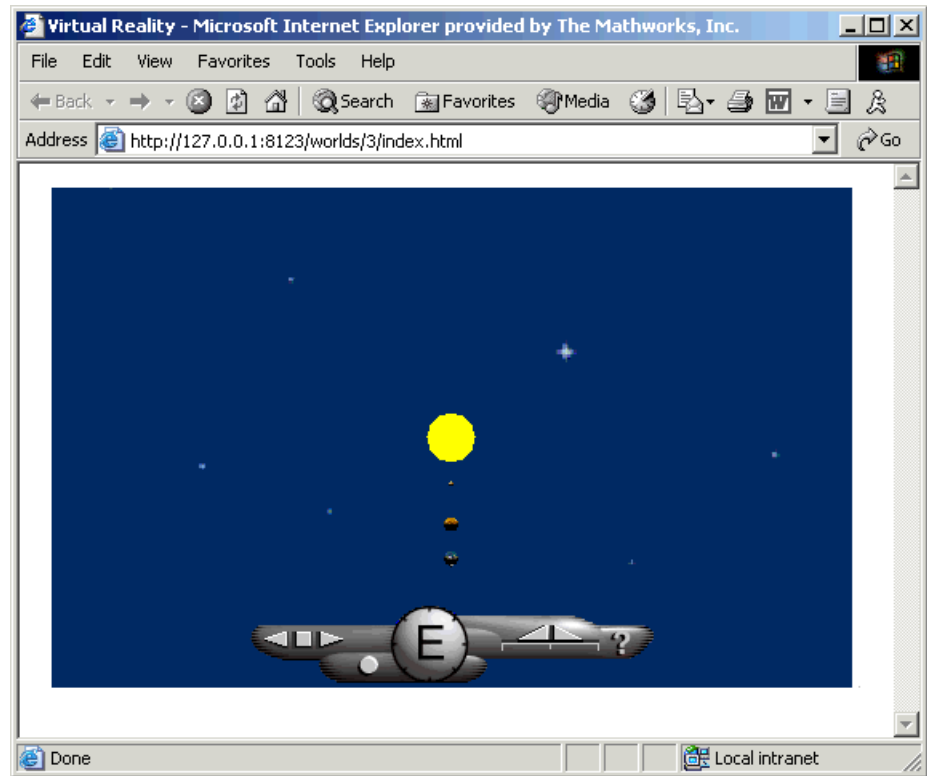
- 4 Change the default viewer to your Web browser by typing  
`vrsetpref('DefaultViewer','web')`

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`

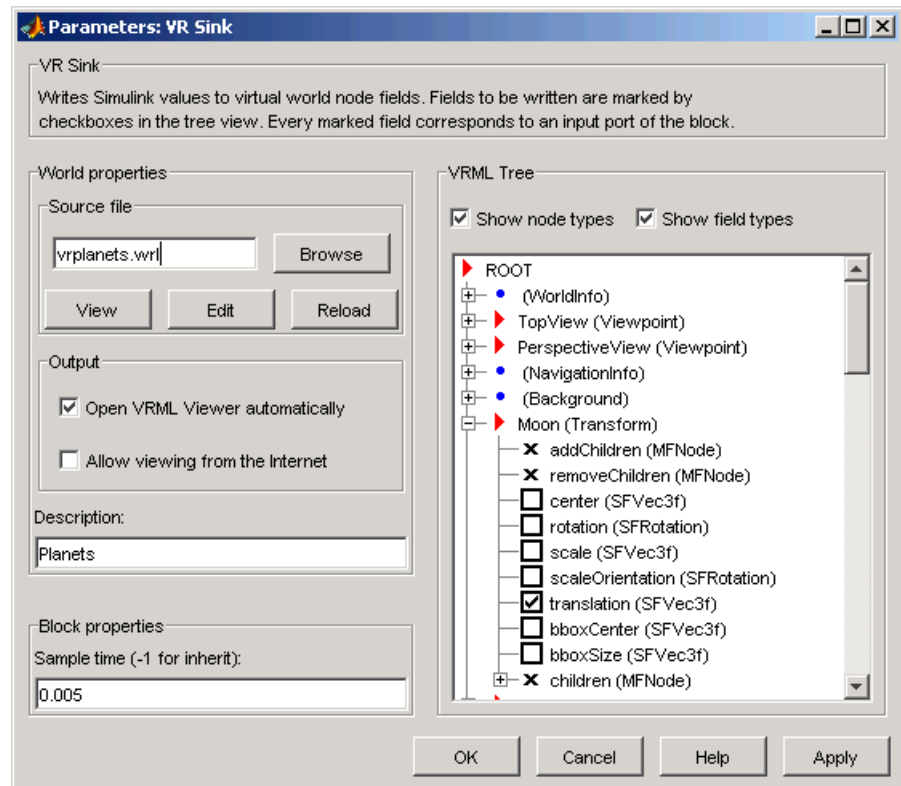
The Planets demo is loaded and the virtual scene is displayed in your Web browser.





- 6 Reset the Virtual Reality Toolbox viewer as your default viewer by typing `vrsetpref('DefaultViewer','factory')`
- 7 In the Virtual Reality Toolbox 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 Virtual Reality Toolbox 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

You can create virtual worlds with a VRML authoring tool or by writing VRML code in a text editor.

This section contains the following topics:

- “Installing the VRML Editor (Windows)” on page 2-29 — Install V-Realm Builder on your PC.
- “VRML Editor (UNIX/Linux)” on page 2-30 — The MATLAB editor is the default VRML editor for UNIX platforms.
- “Setting the Default Editor of Virtual Scenes” on page 2-30 — Edit virtual scenes with a VRML authoring tool or a text editor.

### Installing the VRML Editor (Windows)

When you install the Virtual Reality Toolbox, files are copied to your hard drive for V-Realm Builder, but the installation is not complete.

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

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

```
vrinstall -install editor
```

or type

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

MATLAB displays the following messages:

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

### 3 Type

```
vrinstall -check
```

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

```
VRML editor:      installed
```

### **VRML Editor (UNIX/Linux)**

The MATLAB editor is the default VRML editor for UNIX platforms and no installation is required. To create your virtual worlds using the MATLAB editor, you need to understand the virtual reality modeling language and the VRML data types that are relevant to MATLAB. For information about the modeling language, refer to an appropriate third-party VRML book. Also, see “VRML Data Types” on page 5-20 for the data types to use with MATLAB.

Alternatively, you can use a general 3-D modeling tool with VRML97 export capabilities. Currently, no VRML editor with the functionality of those available for Window platforms is commercially available for UNIX platforms. However, an open source VRML editor, `white_dune`, is under development for UNIX systems. See <http://www.csv.ica.uni-stuttgart.de/vrml/dune> for more information.

### **Setting the Default Editor of Virtual Scenes**

You can edit virtual scenes with a VRML authoring tool, such as V-Realm Builder, or with any text editor, as the VRML language is written in text files. You determine the editor that is used to edit your scene by using the `vrsetpref` and `vrgetpref` commands.

The following procedure demonstrates how to change your editor from V-Realm Builder to a text editor. It assumes that you are working on a PC platform:

**1** At the MATLAB command prompt, type

```
vrinstall -check
```

to determine whether V-Realm Builder is installed.

MATLAB displays

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

The viewer and editor are installed. If the editor is not installed, see “Installing the VRML Editor (Windows)” on page 2-29.

**2** Determine your default editor by typing

```
a = vrgetpref
```

MATLAB displays

```
a =
                                     DataTypeBool: 'logical'
                                     DataTypeInt32: 'double'
                                     DataTypeFloat: 'double'
DefaultFigureAntialiasing: 'off'
DefaultFigureDeleteFcn: ' '
DefaultFigureLighting: 'on'
DefaultFigureMaxTextureSize: 'auto'
DefaultFigureNavPanel: 'halfbar'
DefaultFigureNavZones: 'off'
DefaultFigurePosition: [5 92 512 380]
DefaultFigureRecord2DCompressMethod: 'auto'
DefaultFigureRecord2DCompressQuality: 75
DefaultFigureRecord2DFileName: '%f_anim_%n.avi'
DefaultFigureStatusBar: '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  
      VrPort: 8123
```

The variable `a` is a structure array. You need to index into it to determine the `Editor` property.

- 3 To determine your default editor, type

```
a.Editor
```

MATLAB displays

```
ans =  
"%matlabroot\toolbox\vr\vrealm\program\vrbuild2.exe" "%file"
```

This is the path to the V-Realm Builder executable file. V-Realm Builder is the current VRML editor.

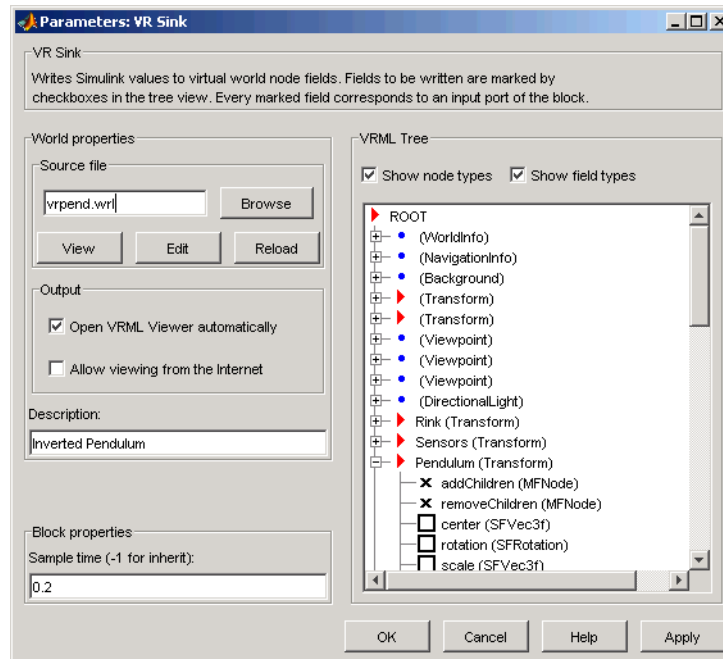
- 4 Verify that V-Realm Builder is your default editor. At the MATLAB command prompt, type

```
vrpend
```

The Inverted Pendulum demo loads and the pendulum is visible in the viewer.

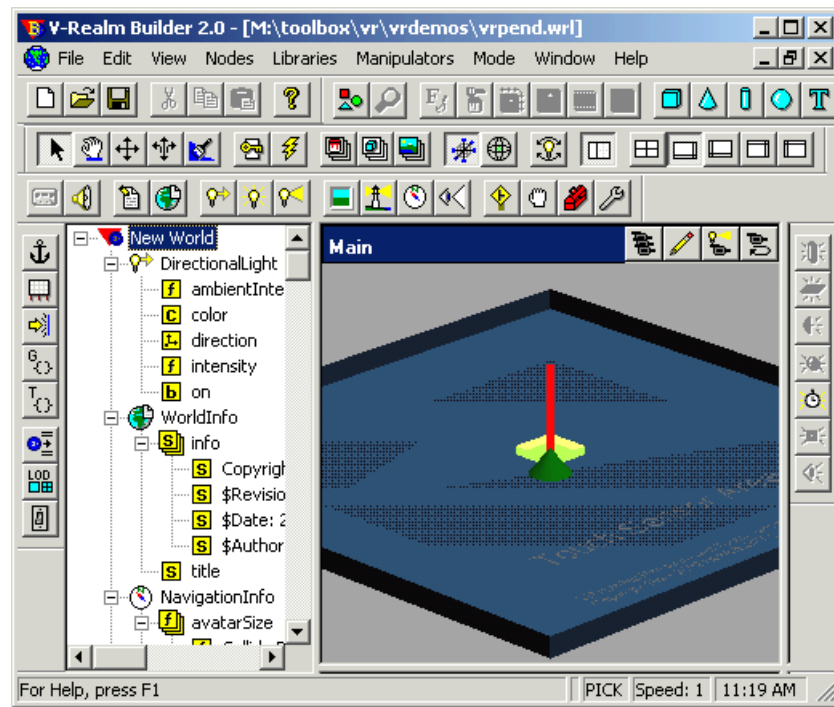
- 5 In the Virtual Reality Toolbox viewer for `vrpend`, from the **Simulation** menu, select **Block Parameters**.

The **Parameters: VR Sink** dialog box opens.



## 6 Click **Edit**.

The vrpend model opens in the V-Realm Builder authoring tool.



- 7 At the MATLAB window, change the default editor to the MATLAB editor by typing

```
vrsetpref('Editor', '%matlabroot\bin\win32\meditor.exe %file')
```

You can set your editor to any text editor you want to use by specifying the path to the executable of the text editor.

- 8 In the Virtual Reality Toolbox viewer for vrpend, from the **Simulation** menu, select **Block Parameters**.

The **Parameters: VR Sink** dialog box opens.

- 9 Click **Edit**.

The MATLAB editor opens and is now set as your default VRML editor.



- 10** To reset the V-Realm Builder authoring tool as your default VRML editor, type

```
vrsetpref('Editor','factory')
```

Clicking the **Edit** button now launches V-Realm Builder.

## Removing Components (Windows)

Normally, you should not have to uninstall the Virtual Reality Toolbox, the blaxxun Contact plug-in, or V-Realm Builder. If you do, see the following topics for the appropriate procedures:

- “Removing the Virtual Reality Toolbox and V-Realm Builder (Windows)” on page 2-36 — Uninstalling the Virtual Reality Toolbox and V-Realm Builder
- “Removing the blaxxun Contact Plug-In (Windows)” in Chapter 2 — Uninstalling the blaxxun Contact plug-in

### Removing the Virtual Reality Toolbox and V-Realm Builder (Windows)

Use the MathWorks uninstaller. Running this utility removes the Virtual Reality Toolbox and V-Realm Builder from your system. It also restores your previous system configuration:

- 1** On the Microsoft Windows task bar, click **Start**, point to **MATLAB**, and then click **R14 uninstaller**.

The MathWorks uninstaller begins running.

- 2** Clear the **Virtual Reality Toolbox** check box.
- 3** Follow the remaining uninstall instructions.

---

**Note** The blaxxun Contact plug-in is not uninstalled during the Virtual Reality Toolbox removal.

---

## **Removing the blaxxun Contact Plug-In (Windows)**

You can uninstall this VRML plug-in from the host computer by using the following procedure:

- 1** From the Microsoft 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.

## Installing on the Client Computer

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 take 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 blaxxun Contact is supported.

The section “Installing a VRML Plug-In (Windows)” on page 2-38 describes how to install the blaxxun Contact VRML plug-in on a computer running Microsoft 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 Virtual Reality Toolbox, 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:\<MATLAB root>\toolbox\vr\blaxxun`.

## Testing the Installation

The Virtual Reality Toolbox includes several Simulink models with the associated virtual worlds. These models are examples of what you can do with this toolbox. You can use one of these examples to test the installation of the Virtual Reality Toolbox, the VRML viewer, and the VRML editor.

This section contains the following topics:

- “Running a Simulink Interface Example” on page 2-39 — Open a Simulink model for an inverted pendulum, start a simulation, and view the pendulum in a virtual world.
- “Running a MATLAB Interface Example” on page 2-44 — View a virtual world of the MathWorks membrane.

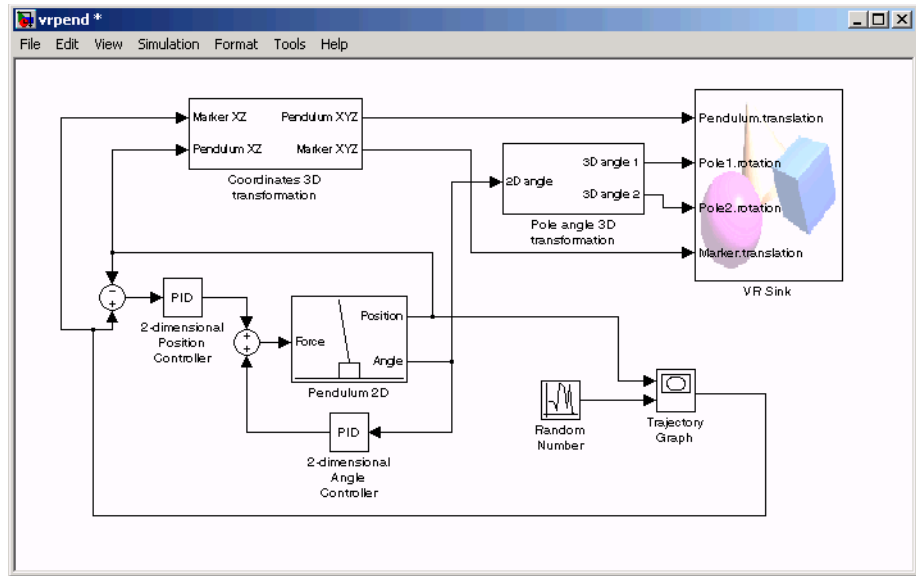
### Running a Simulink Interface Example

In the demo directory for the Virtual Reality Toolbox, there is a Simulink model for a two-dimensional inverted pendulum. This model, which you can view in three dimensions with the toolbox, has an interactive set point and trajectory graph.

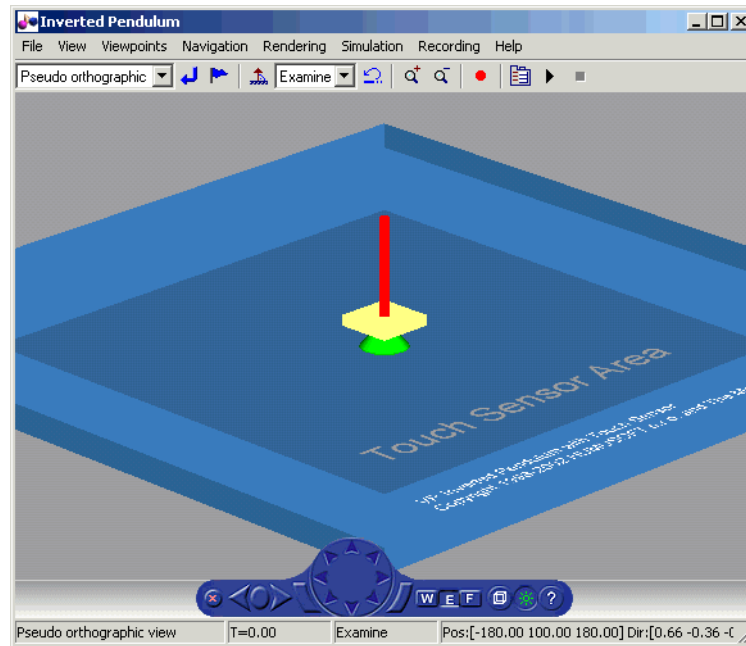
Before you can run this demo, you have to install MATLAB, Simulink, and the Virtual Reality Toolbox:

- 1 In the MATLAB Command Window, type  
`vrpend`

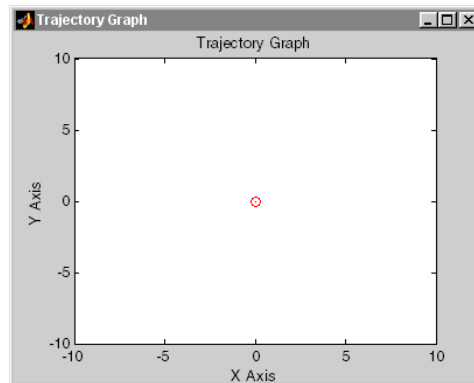
A Simulink window opens with the model for an inverted pendulum.



The Virtual Reality Toolbox viewer opens with a 3-D model of the pendulum.



- 2 In the Virtual Reality Toolbox viewer, from the **Simulation** menu, click **Start**. A **Trajectory Graph** window opens, and a simulation starts running.



- 3 In the Virtual Reality Toolbox viewer, point to a position on the blue surface and left-click.

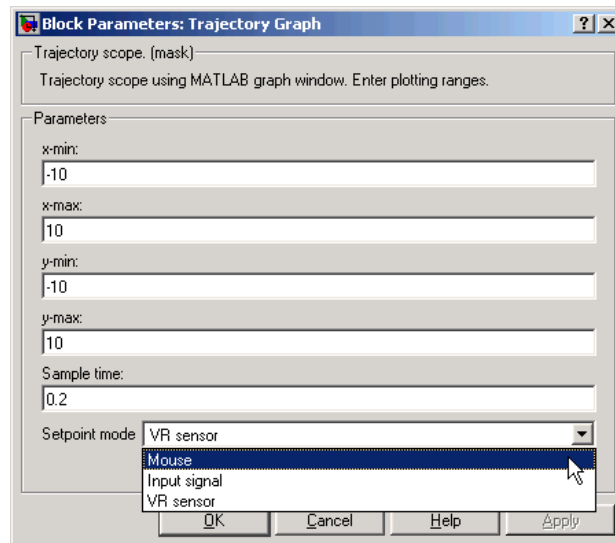
The pendulum set point, represented by the green cone, moves to a new location. Next, the path is drawn on the trajectory graph, and then the pendulum itself moves to the new location.

In the Virtual Reality Toolbox 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 Virtual Reality Toolbox viewer controls, see “Virtual Reality Toolbox Viewer” on page 6-2.

- 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.

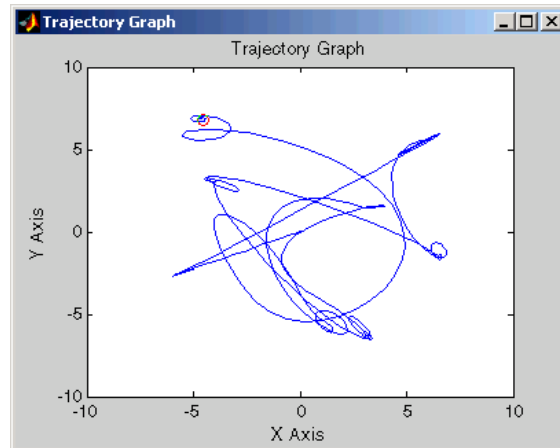


- 6 Move the mouse pointer into the graph area and click.

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 Virtual Reality Toolbox viewer and close the Simulink window.

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

### Running a MATLAB Interface Example

This model, which can be viewed in three dimensions with the toolbox, 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 1-24:

- 1 In the MATLAB window, type

```
vrmemb
```

MATLAB displays the following messages:

```
Loading...
```

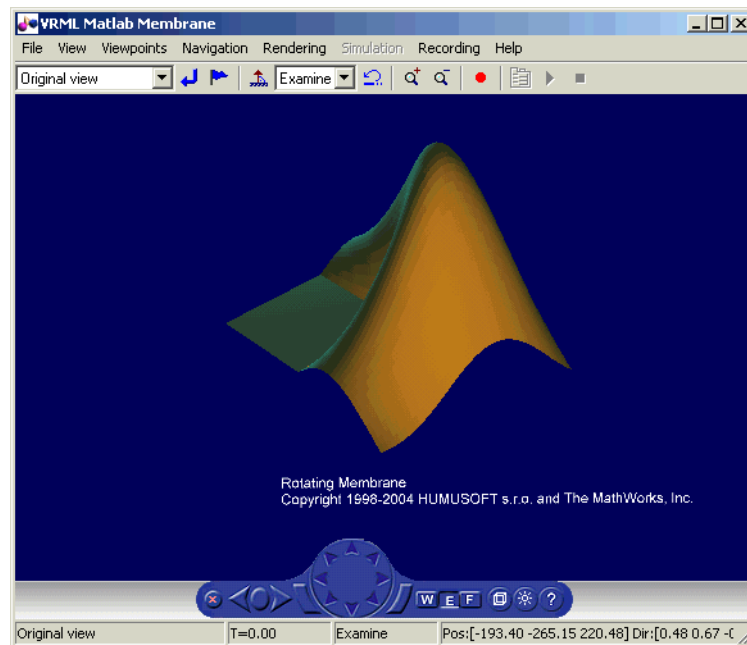
```
This example shows you how to use a MATLAB generated 3-D graphic  
object in the Virtual Reality Toolbox.
```

```
. . .
```

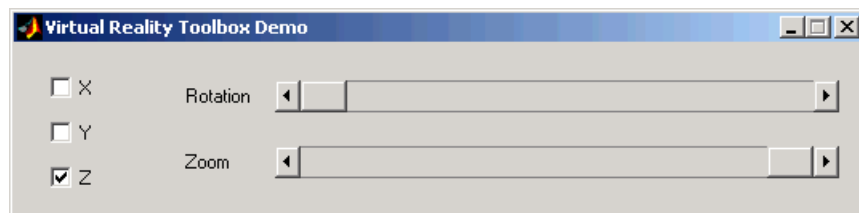
```
Press Enter to start the demonstration.
```

- 2 Press the **Enter** key.

The Virtual Reality Toolbox viewer opens with a 3-D model.



- 3 Use the viewer controls to move within the virtual world, or use the demo dialog box to rotate the membrane. Note that sometimes the **Virtual Reality Toolbox Demo** dialog box is hidden behind the viewer window.





# Simulink Interface

---

The Virtual Reality Toolbox works with both MATLAB and Simulink. However, the Simulink interface is the preferred way of working with the toolbox. It is more straightforward to use and all the toolbox features are easily accessible through a graphical user interface (GUI).

Associating a Virtual World with Simulink (p. 3-2)

Associate a Simulink model with a virtual world, and connect signals from the Simulink model to the virtual world

Using the Simulink Interface (p. 3-12)

Open a Simulink model, display the associated virtual world on a host computer or on a client computer, and observe the simulated process in the virtual world

## Associating a Virtual World with Simulink

With the Virtual Reality Toolbox you can interface a Simulink block diagram with 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.

This section includes the following topics:

- “Adding a Virtual Reality Toolbox Block” on page 3-2 — Connect a Simulink model to a virtual world
- “Changing the Virtual World Associated with a Simulink Block” on page 3-10 — Change the virtual world associated with a Simulink model, and change the signals passed between Simulink and the virtual world

### Adding a Virtual Reality Toolbox Block

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 Virtual Reality Toolbox blocks. The example in this procedure simulates a plane taking off and lets you view it in a virtual world.

---

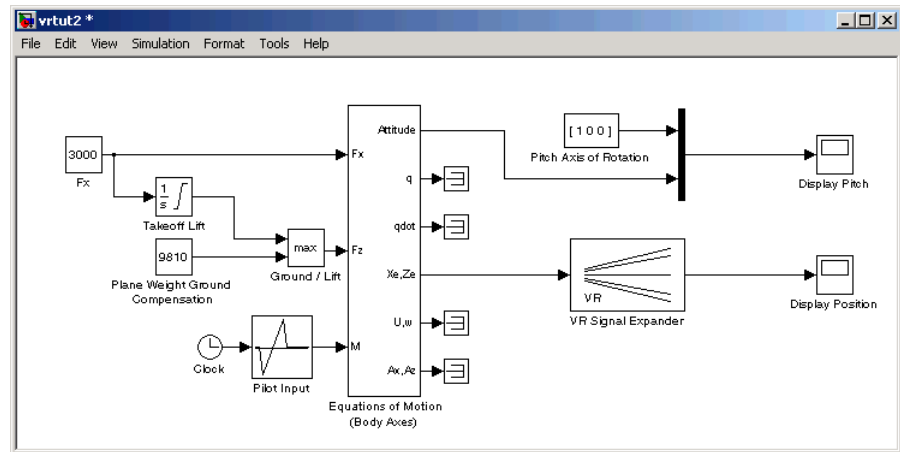
**Note** The examples in this topic are based on the Virtual Reality Toolbox default viewer. If you choose to use the blaxxun Contact VRML plug-in to view virtual worlds, you must start and stop the model simulation from the Simulink window. You cannot start and stop the model simulation from the blaxxun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

---

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

```
vrtut2
```

A Simulink model opens without a Virtual Reality Toolbox block that connects the model to a virtual world.

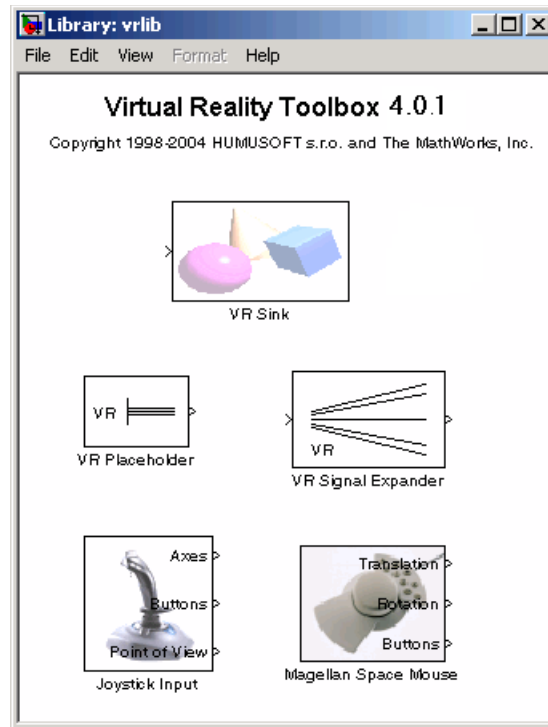


- 2 From the **Simulation** menu, select **Normal**, then click **Start**.

Observe the results of the simulation in the scope windows.

- 3 In the MATLAB Command Window, type  
`vrlib`

The Virtual Reality Toolbox library opens.



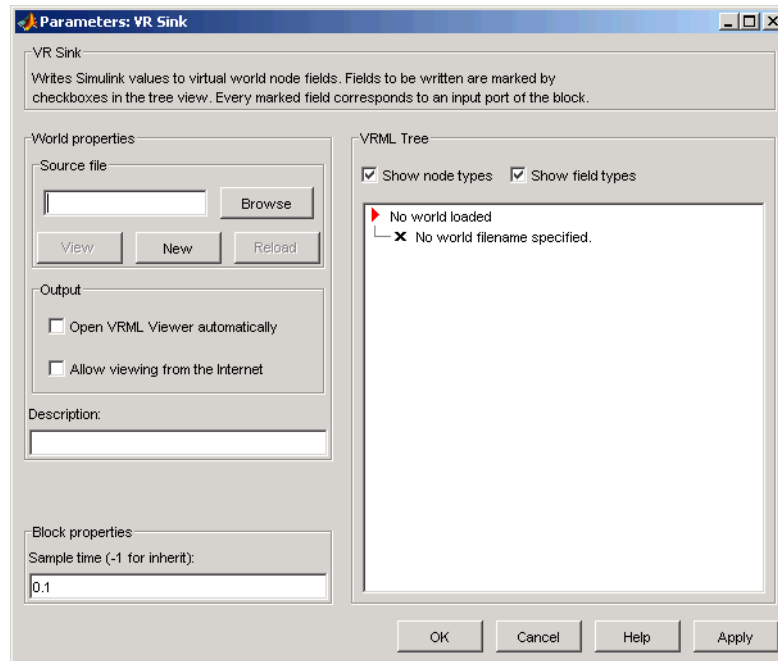
- 4 From the **Library** window, drag and drop the VR Sink block to the Simulink diagram. The VR Sink block writes data from the Simulink model to the virtual world. (For a description of all the Simulink blocks for the Virtual Reality Toolbox, see Chapter 7, “Block Reference.”) You can then close the **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` directory.

- 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 Virtual Reality Toolbox server. For example, type  
     VR Plane taking off
- 7 Click the **Browse** button. The **Select World** dialog box opens. Find the directory `<matlab root>\toolbox\vr\vr demos`. Select the file `vrtkoff.wrl` and click **Open**.
- 8 In the **Parameters: VR Sink** dialog box, click **Apply**.

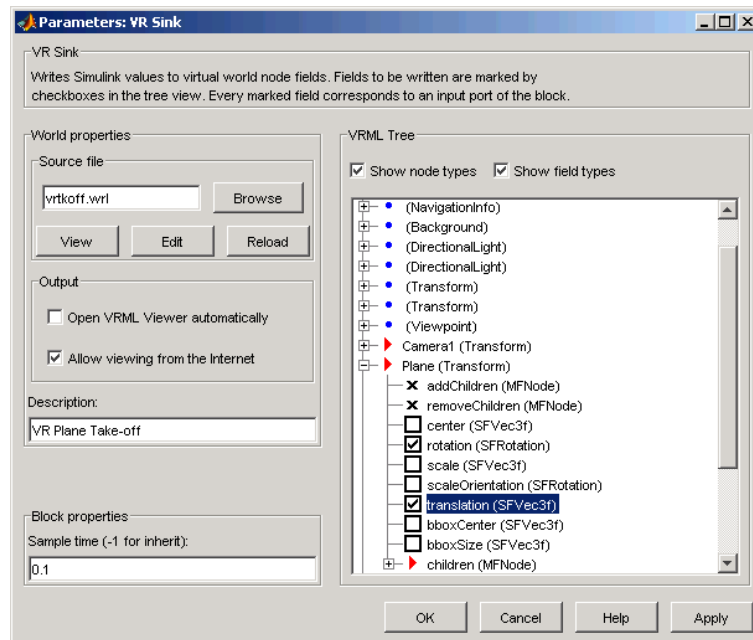
A VRML tree appears on the right side, showing the structure of the associated virtual reality scene.

- 9** 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 Simulink. This model computes the position and the pitch of the plane.

- 10** 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.



- 11** Click OK.

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



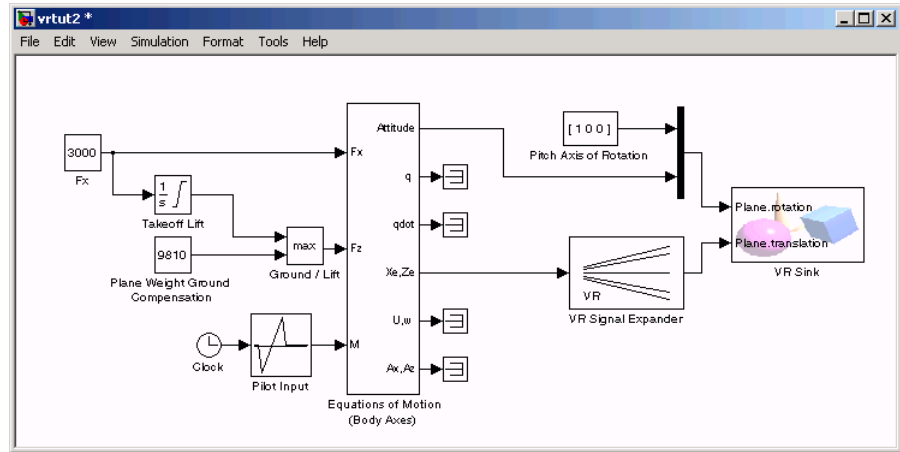
The first input is Plane rotation. The rotation is defined by a four-element vector. The first three numbers define the axis of rotation. In this example, it should be  $[1\ 0\ 0]$  for the  $x$ -axis (see the Pitch Axis of Rotation block in the model). The pitch of the plane is expressed by the rotation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

- 12** 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.

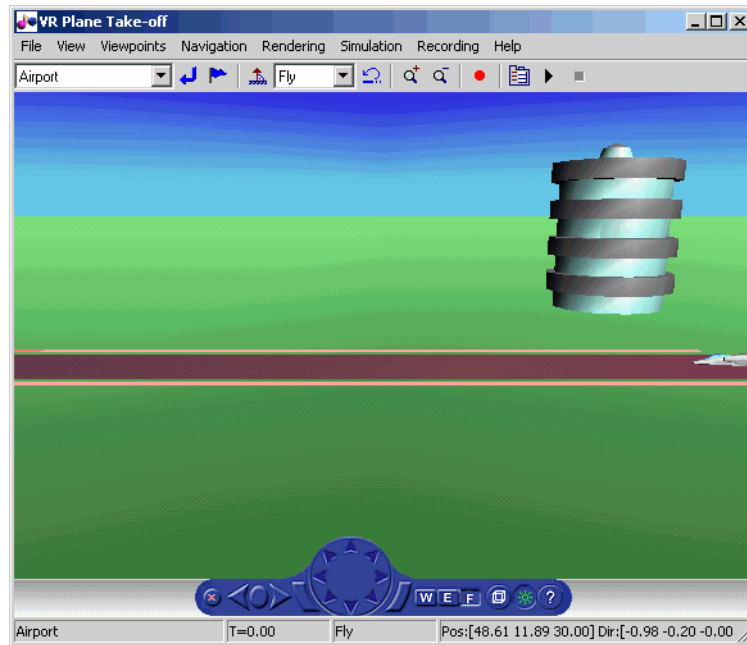
- 13** In the Simulink model, connect the line going to the Scope block labeled Display Position to the Plane translation input.

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



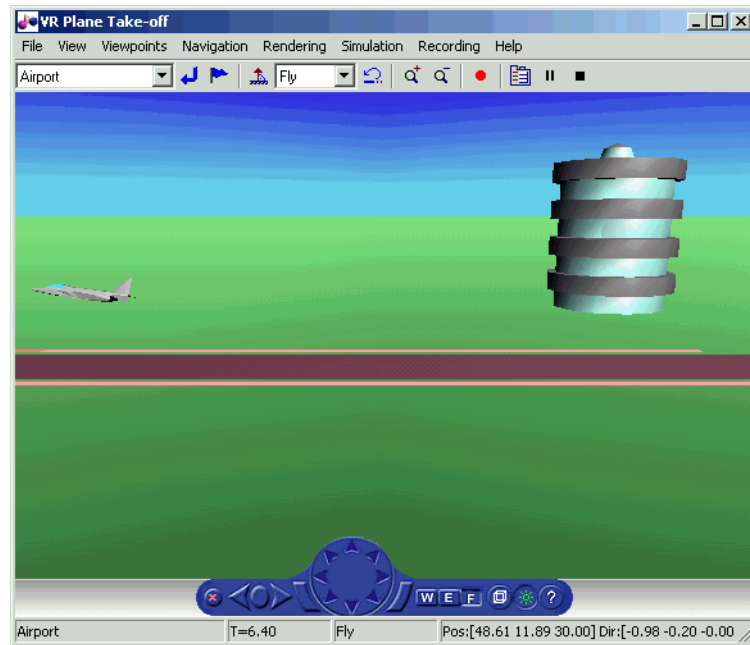
**Note** Virtual world degrees of freedom have different requested input vector sizes depending on the associated VRML field types. If the vector size of the connected signal does not match the associated VRML field size, an Incorrect input vector size error is reported when you start the simulation.

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



- 15 In the Virtual Reality Toolbox viewer, from the **Simulation** menu, click **Start** 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 `vrtut2` Simulink model with a virtual world. See “Adding a Virtual Reality Toolbox Block” on page 3-2.

- 1 Double-click the VR Sink block in the model.

The viewer displays.

- 2 Select the **Simulation** menu **Block Parameters** option.

The **Parameters: VR Sink** dialog box opens.

**3** Click the **Browse** button. The **Select World** dialog box opens. Find the directory `<matlab root>\toolbox\vr\vr demos`. Select the file `vrtkoff2.wr1`, and click **Open**.

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

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

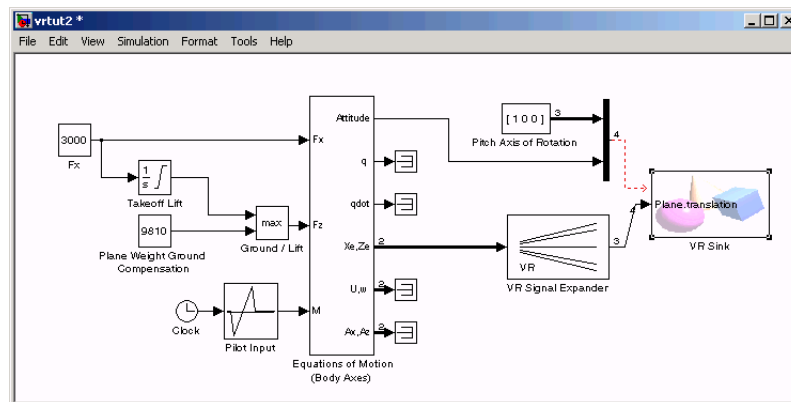
**5** On the left of the **Plane (Transform)** node, click the + square.

The **Plane Transform** tree expands. Now you can see what characteristics of the plane you can drive from Simulink. 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.



**8** In the Virtual Reality Toolbox viewer, from the **Simulation** menu, run the simulation again and observe the simulation.

## Using the Simulink Interface

You can view a virtual world connected to a Simulink block diagram and make parameter changes from Simulink or the virtual world.

This section includes the following topics:

- “Displaying a Virtual World and Starting Simulation” on page 3-12 — Display and interact with a virtual world on your host computer using the Virtual Reality Toolbox viewer.
- “Viewing a Virtual World with a Web Browser on the Host Computer” on page 3-15 — Connect to the Virtual Reality Toolbox host to access and view virtual worlds.
- “Viewing a Virtual World with a Web Browser on the Client Computer” on page 3-19 — Display and interact with a virtual world on a client computer.

### Displaying a Virtual World and Starting Simulation

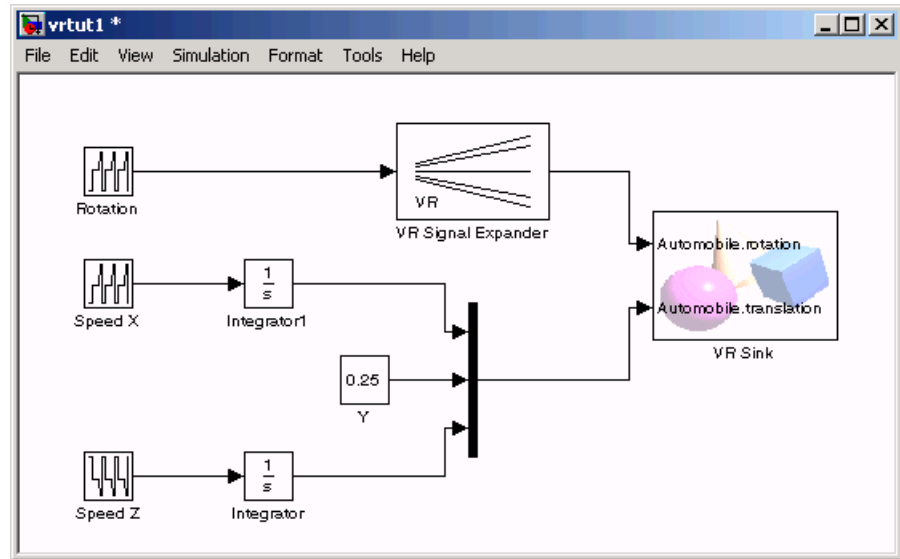
This example explains how to display a simulated virtual world using the Virtual Reality Toolbox viewer on your host computer. This is the default and recommended method for viewing virtual worlds. A Simulink window opens with the model of a simple automobile. Automobile trajectory (vehicle position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

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

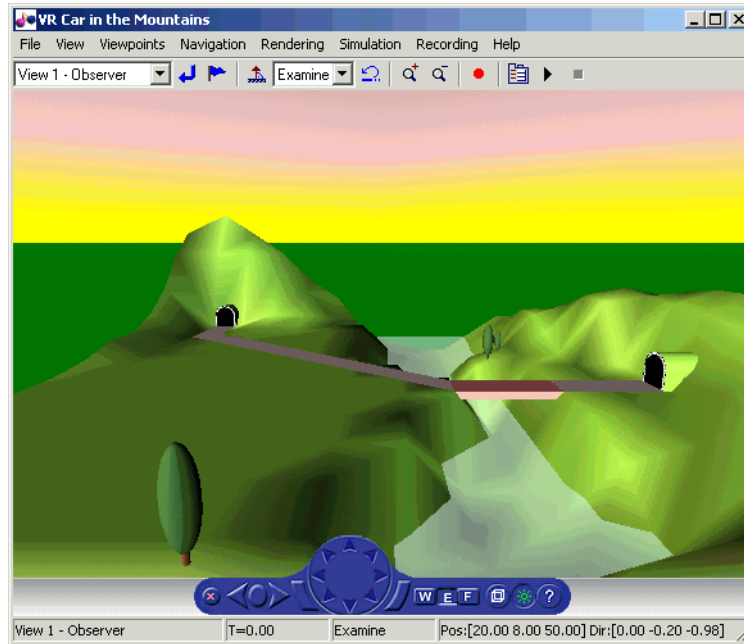
```
vrtut1
```



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 Virtual Reality Toolbox viewer, from the **Simulation** menu, click **Start**.  
  
The simulation starts. In the Virtual Reality Toolbox viewer, a car moves along the mountain road.
- 3 Use the Virtual Reality Toolbox viewer controls to move the camera within this virtual world while the simulation is running. For more information on the Virtual Reality Toolbox viewer controls, see “Virtual Reality Toolbox Viewer” on page 6-2.
- 4 In the Virtual Toolbox 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 “Setting the Default Viewer of Virtual Scenes” on page 2-24.

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 Virtual Reality Toolbox viewer. This feature is particularly useful if you want to view one scene from many different viewpoints at the same time.

## Viewing a Virtual World with a Web Browser on the Host Computer

Normally, you view a virtual world by double-clicking the VR Sink in the Simulink model. The virtual world opens in the Virtual Reality Toolbox viewer or your VRML-enabled Web browser, depending on your `DefaultViewer` setting. For more information on setting your default viewer, see “Setting the Default Viewer of Virtual Scenes” on page 2-24.

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 Virtual Reality Toolbox host. This is the computer on which the toolbox is currently running. You do not need a VRML-enabled Web browser to display this page.

The following procedure describes how to connect to the Virtual Reality Toolbox host:

- 1 At the MATLAB command prompt, type

```
vrbounce
```

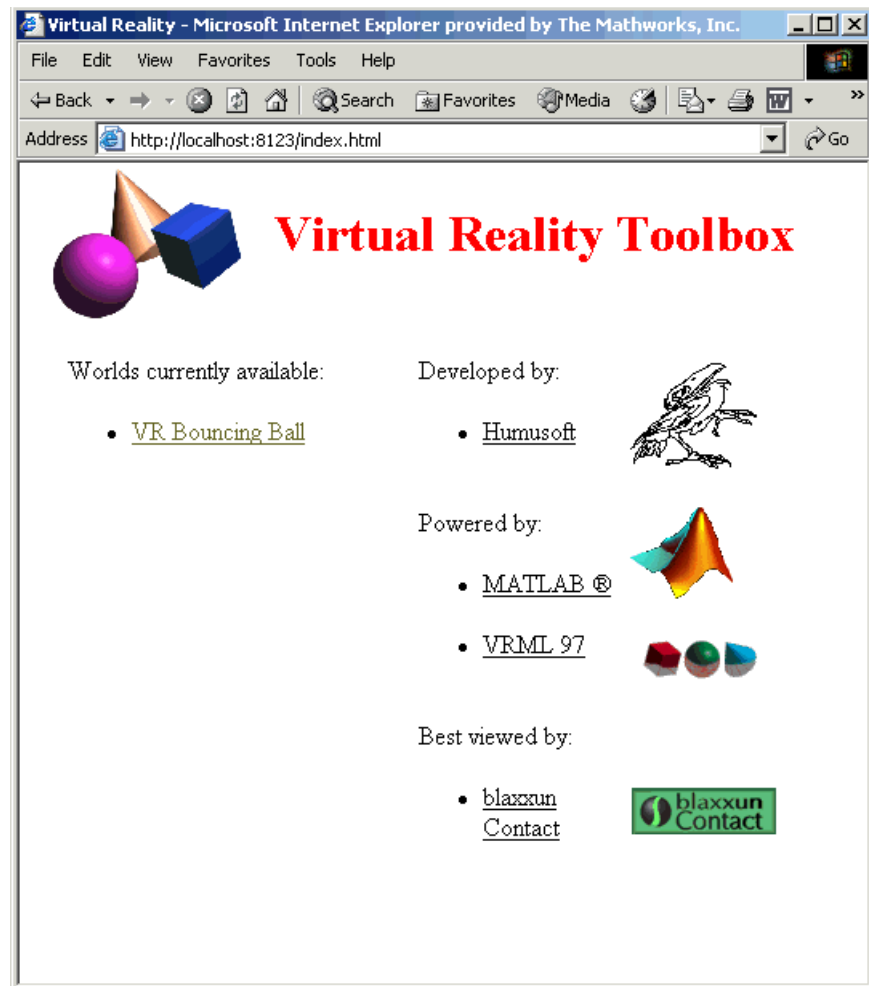
The VR Bouncing Ball demo is loaded and becomes active.

- 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 toolbox is currently running.

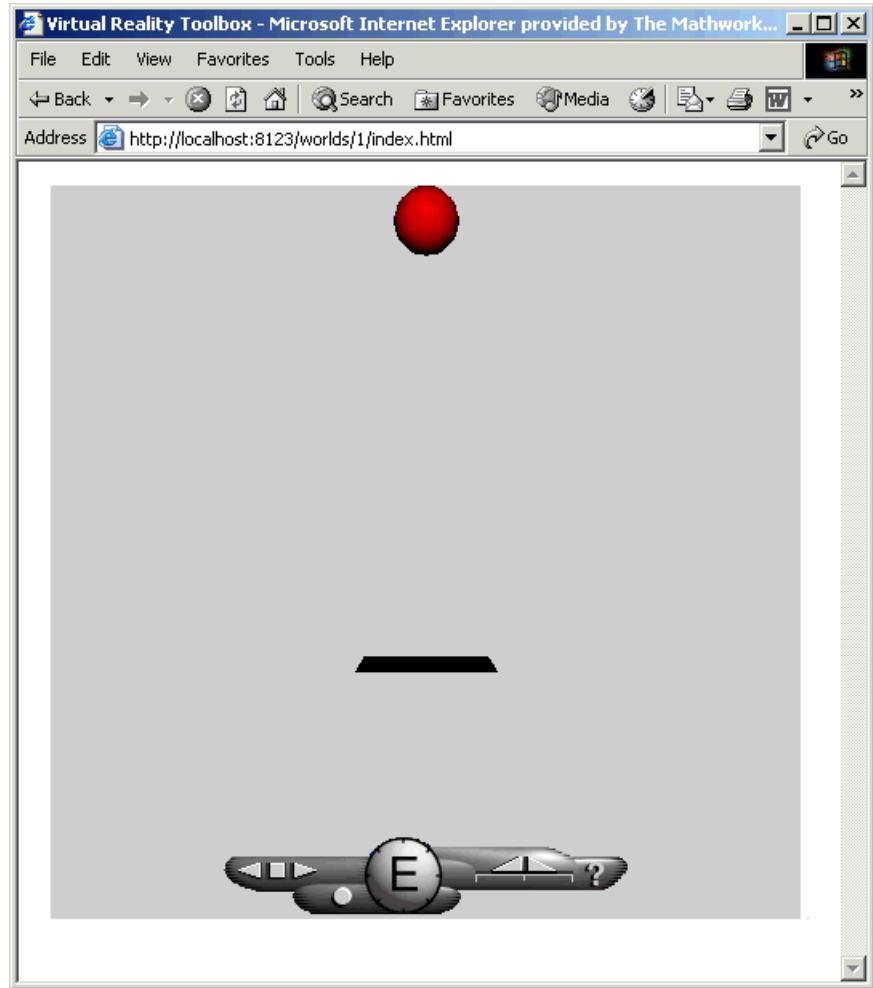
The following page is loaded and becomes active.



The main HTML page for the Virtual Reality Toolbox 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 toolbox. 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 Virtual Reality Toolbox.

## Viewing a Virtual World with a Web Browser on the Client Computer

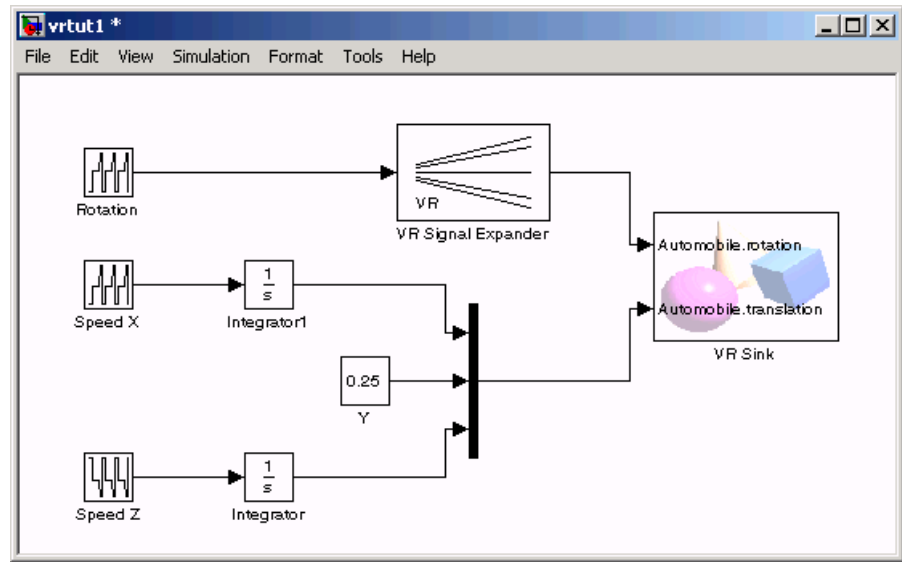
The Virtual Reality Toolbox allows you to simulate a process on a host computer while running the visualization of the process on a client computer. You view the virtual world on the client computer using a Web browser. This client computer is connected to the host computer through a network using the 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. 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.



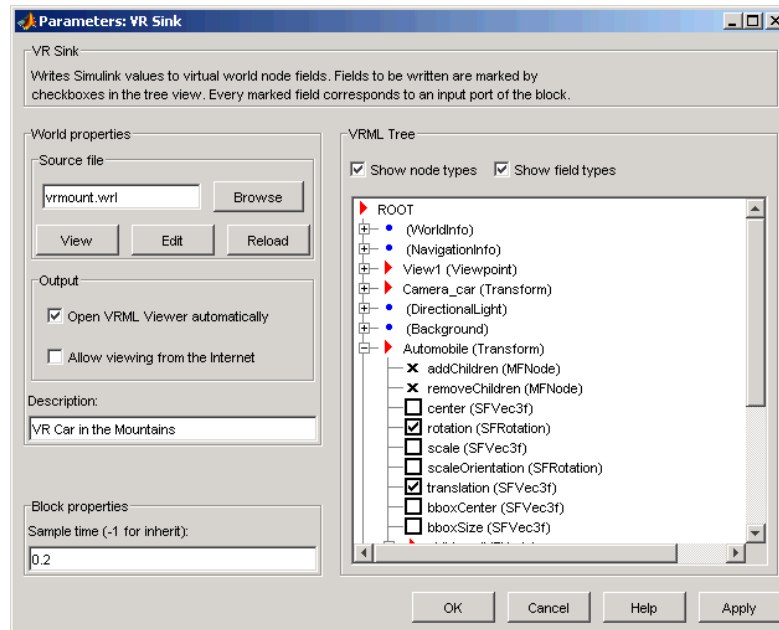
- 2 Double-click the VR Sink block. This block is in the right part of the model window.

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 Virtual Reality Toolbox port number for the host computer running Simulink. 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,

- Click **Start**, click **Run**, type `cmd`, and enter `ipconfig` (Windows 2000).
- Click **Start**, click **Run**, in the Open box enter `wntipcfg` (Windows NT).

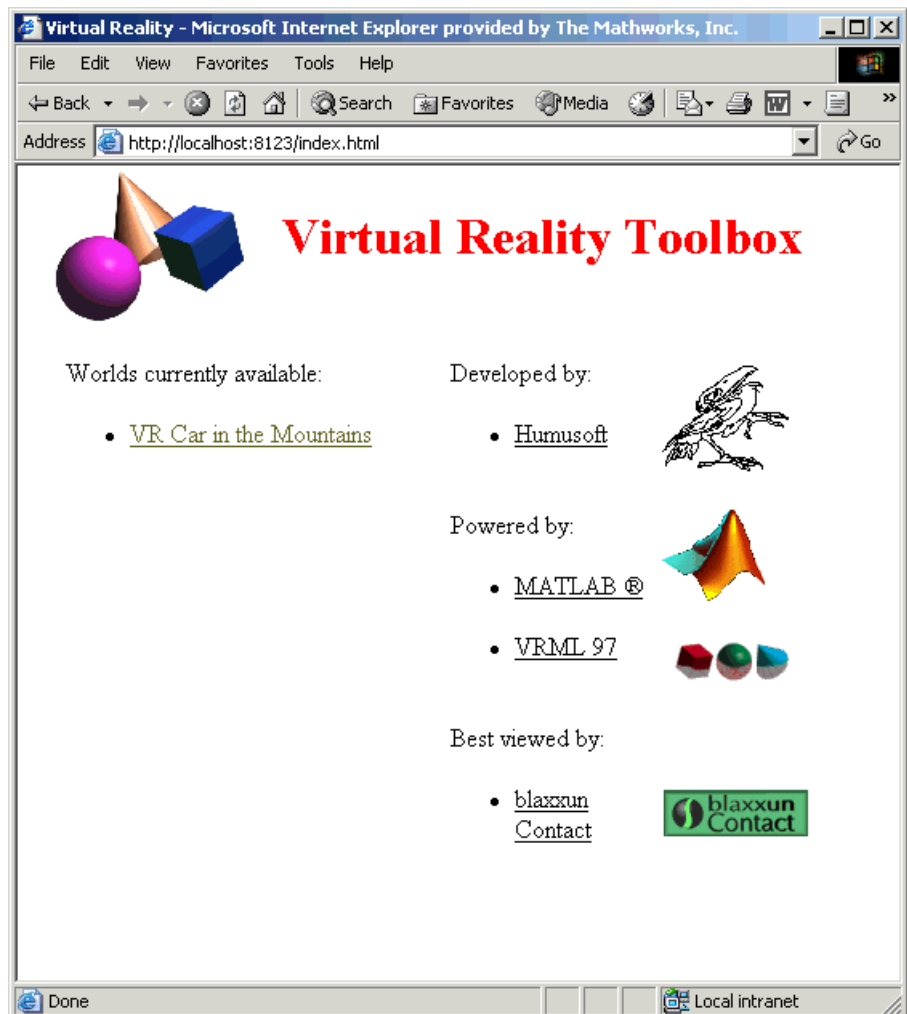
To determine your IP address on a UNIX system, type the command

```
ifconfig device_name
```

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

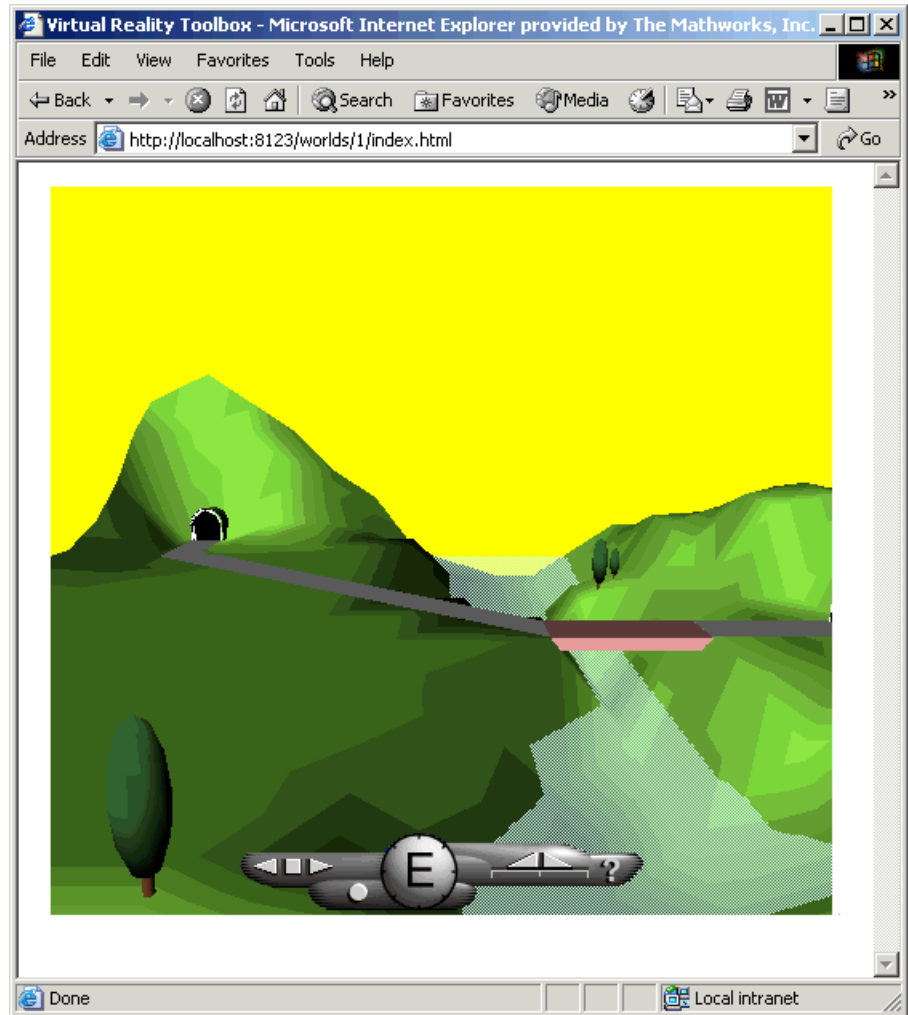
Alternatively, for Windows platforms, you can open a DOS shell and type `ipconfig`.

The Web browser displays the main Virtual Reality Toolbox HTML page. Only one virtual world is in the list because you have only one Simulink model open.



**7 Click 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 **Start**.

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.



# MATLAB Interface

---

Although using the Virtual Reality Toolbox with the Simulink interface is the preferred way of working with the toolbox, you can also use the MATLAB interface. Enter commands directly in the MATLAB Command Window or use M-files to control virtual worlds.

- Using the MATLAB Interface (p. 4-2)    Control virtual worlds by entering commands directly in the MATLAB Command Window or by using M-files
- Recording Offline Animations (p. 4-10)    Record simulations and object movement into animation files for later offline viewing.

## Using the MATLAB Interface

This section includes the following topics:

- “Creating a vrworld Object” on page 4-2 — Create a `vrworld` object to connect MATLAB with a virtual world
- “Opening a Virtual World” in Chapter 4 — Open a virtual world and scan its structure
- “Interacting with a Virtual World” on page 4-5 — Set new values for the available virtual world nodes and their fields
- “Closing and Deleting a vrworld Object” on page 4-8 — Close open virtual worlds and remove them from memory

### Creating 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. You cannot directly interact with a virtual world. A virtual world is defined by a VRML file with the extension `.wrl`. For a complete list of virtual world methods, see “`vrworld` Object Methods” on page 8-3, “`vrnode` Object Methods” on page 8-4, and “`vrfigure` Object Methods” on page 8-4.

---

**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 Virtual Reality Toolbox 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')
```



MATLAB displays output like

```
myworld =  
vrworld object: 1-by-1  
  
VR Car in the Mountains  
(<matlab-root>/toolbox/vr/vrdemos/vrmount.wrl)
```

## 2 Type

```
vrwhos
```

MATLAB displays the messages

```
Closed, associated with  
'C:<matlab root>\toolbox\vr\vrdemos\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 “Opening a Virtual World” on page 4-3.

## Opening 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);
```

MATLAB 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 Virtual Reality object listings, in the title bar of the Virtual Reality Toolbox viewer, and in the list of virtual worlds on the Virtual Reality Toolbox 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 Virtual Reality Toolbox 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 Virtual Reality Toolbox HTML page with a link to **My first virtual world**. The number 8123 is the default Virtual Reality Toolbox port number. If you set a different port number on your system, enter that number in place of 8123. For more information on the Virtual Reality Toolbox HTML page, see “Viewing a Virtual World with a Web Browser on the Host Computer” on page 3-15.

### 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 Chapter 2, “Installation.”

---

For more information on changing your default viewer, see “Setting the Default Viewer of Virtual Scenes” on page 2-24.

## Interacting 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` Object Methods” on page 8-3 and “`vrnode` Object Methods” on page 8-4 for a full description of these objects.

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.wr1` as an example:

1 In the MATLAB Command Window, type

```
nodes(myworld);
```

MATLAB displays a list of the `vrnode` objects and their fields that are accessible from the Virtual Reality Toolbox.

```
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')
```

MATLAB 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
```

MATLAB 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));
```

MATLAB displays information like the following table.

| Field            | Access       | Type       | Sync |
|------------------|--------------|------------|------|
| addChildren      | eventIn      | MNode      | off  |
| removeChildren   | eventIn      | MNode      | off  |
| children         | exposedField | MNode      | 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];
```

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.

---

## Closing and Deleting 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

The Virtual Reality Toolbox enables you to record animations of virtual scenes that are controlled by Simulink or MATLAB. You can record simulations through either the Virtual Reality Toolbox viewer (described in Chapter 6, “Viewing Virtual Worlds”) or the MATLAB interface (described in this section). You can then play back these animations offline, in other words, independent of MATLAB, Simulink, or the Virtual Reality Toolbox. You might want to generate such files for presentations, to distribute simulation results, or to generate archives.

---

**Note** Optimally, use the Virtual Reality Toolbox viewer (Chapter 6, “Viewing Virtual Worlds”) 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 Virtual Reality Toolbox 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 Virtual Reality Toolbox traces object movements and saves that data into a VRML file using VRML97 standard interpolators. You can then view these files with the Virtual Reality Toolbox 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 Virtual Reality Toolbox viewer while recording the animation, the Virtual Reality Toolbox 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 Virtual Reality Toolbox writes animation data into an .avi file. The Virtual Reality Toolbox 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 Virtual Reality Toolbox viewer is the topmost window and fully visible. Graphics acceleration limitations might prevent the proper recording of 2-D animation otherwise.

---

This section contains the following topics. These topics use the `vrplanet`s demo as the example.

- “Animation Recording File Tokens” on page 4-12 — Describes the filename tokens you can use to direct the Virtual Reality Toolbox viewer to record an animation.
- “Manual 3-D VRML Animation Recording” on page 4-14 — Describes how to manually record animation files.
- “Scheduled 3-D VRML Animation Recording” on page 4-19 — Describes how to perform scheduled animation recording.
- “Viewing Animation Files” on page 4-24 — Describes how to view recorded animations.
- “MATLAB Animation Recording of Virtual Worlds Not Associated with Simulink Models” on page 4-26 — Describes how you can record offline animations for virtual worlds that are not associated with Simulink models.

## Animation Recording File Tokens

By default, the Virtual Reality Toolbox 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 Virtual Reality Toolbox places the file in the current directory. %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.

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 subdirectory of the directory 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 directory. |
| %D    | The current day in the month replaces this token in the filename string. For example, the format %f_anim_%D.wrl saves the animation to vrplanets_anim_29.wrl for the 29th day of the month.   |

| Token | Description  |
|-------|--|
| %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. |

| <b>Token</b>    | <b>Description</b>  |
|-----------------|---|
| <code>%s</code> | The current second replaces this token in the filename string. For example, the format <code>%f_anim_%h%m%s.wrl</code> saves the animation to <code>vrplanets_anim_150430.wrl</code> for a start record time of 15:04:30. |
| <code>%Y</code> | The current four digit year replaces this token in the filename string. For example, the format <code>%f_anim_%Y.wrl</code> saves the animation to <code>vrplanets_anim_2003.wrl</code> for the year 2003.                |

### **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` demo. 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 displays. Also by default, the Virtual Reality Toolbox viewer for that model is loaded and becomes active. If the viewer does not display, double-click the Virtual Reality 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(strmatch('Planets',get(worlds,'Description')));
```

- 3** To have the Virtual Reality Toolbox manually record the animation, set the `RecordMode` property to `manual`. Type

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

- 4** Direct the Virtual Reality Toolbox to record the animation to a VRML format file. Type

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

- 5 Run the Simulink model. From the **Simulation** menu, select **Normal**, then click **Start**. Alternatively, if you are using the Virtual Reality Toolbox 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 Virtual Reality Toolbox stops recording the animation. The Virtual Reality Toolbox creates the file `vrplanets_anim_1.wrl` in the current working directory. 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` demo. 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 displays. Also by default, the Virtual Reality Toolbox viewer for that model is loaded and becomes active. If the viewer does not display, double-click the Virtual Reality 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(strmatch('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 Virtual Reality Toolbox viewer figure, type

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

- 5** To have the Virtual Reality Toolbox manually record the animation, set the `RecordMode` property to `manual`. Type

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

- 6** Direct the Virtual Reality Toolbox 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,'PanelMode','off');
```

- 8** Run the Simulink model. From the **Simulation** menu, select **Normal**, then click **Start**. Alternatively, if you are using the Virtual Reality Toolbox 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 Virtual Reality Toolbox stops recording the animation. The Virtual Reality Toolbox creates the file `vrplanets_anim_1.avi` in the current working directory. 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, 'PanelMode', 'on');
```

**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 Virtual Reality Toolbox 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` demo. 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 Virtual Reality Toolbox viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the Virtual Reality 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(strmatch('Planets',get(worlds,'Description')));
```

- 4** Set the Virtual Reality Toolbox to record the animation on a schedule by setting the `RecordMode` property to `scheduled`. Type

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

- 5** Direct the Virtual Reality Toolbox 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 **Normal**, then click **Start**. Alternatively, if you are using the Virtual Reality Toolbox 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 Virtual Reality Toolbox starts recording when the simulation time reaches the specified start time and creates the file `vrplanets_anim_N.wrl` in the current working directory 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 Virtual Reality Toolbox 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` demo. 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 Virtual Reality Toolbox viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the Virtual Reality 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(strmatch('Planets',get(worlds,'Description')));
```

- 3 To retrieve the handle to the currently displayed Virtual Reality Toolbox viewer figure, type

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

- 4 To have Virtual Reality Toolbox manually record the animation, set the RecordMode property to manual. Type

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

- 5 Direct Virtual Reality Toolbox 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,'PanelMode','off');
```

- 8 Ensure that the virtual reality figure window is the topmost window.

- 9 Run the Simulink model. From the **Simulation** menu, select **Normal**, then click **Start**. Alternatively, if you are using the Virtual Reality Toolbox 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. Virtual Reality Toolbox starts recording when the simulation time reaches the specified start time and creates the file `vrplanets_anim_N.avi` in the current working directory 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, 'PanelMode', 'on');
```

**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 4-19 for descriptions of how to create one.

## To View VRML Files

- 1 Change directory to one that contains the VRML animation file.
- 2 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 Virtual Reality Toolbox viewer for the animation file. Setting the `TimeSource` property of the `set` method to `'freerun'` directs the viewer to advance its time independent of MATLAB.

- 3 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 directory 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.

### **MATLAB Animation Recording of Virtual Worlds Not Associated with Simulink Models**

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 Virtual Reality Toolbox demo `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 Virtual Reality Toolbox user.



- Retrieve the `vrworld` object handle of the virtual scene that you want to record.
- To record 2-D animations,
  - Retrieve the corresponding `vrfigure` object. For 2-D animations, Virtual Reality Toolbox records exactly what you see in the viewer window. Because 2-D animations record exactly what you see in the Virtual Reality Toolbox viewer window, the properties that control 2-D file recording belong to `vrfigure` objects.
  - Set the `Record2D` `vrfigure` property.
  - To override default filenames for animation files, set the `vrfigure` `Record2DFileName` property.
- To create 3-D animation files,
  - Retrieve the corresponding `vrworld` object.
  - Set the `Record3D` `vrworld` property.
  - 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 MATLAB 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 4-19), Virtual Reality Toolbox 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`, Virtual Reality Toolbox 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 Virtual Reality Toolbox `vrheat_anim` demo.



# Virtual Worlds

---

Virtual Reality Toolbox includes tools that you can use to edit and create VRML virtual worlds. For Microsoft Windows platforms, the Virtual Reality Toolbox includes a VRML editor (V-Realm Builder). For UNIX/Linux platforms, the default VRML editor is the MATLAB editor. A basic understanding of these tools and how to use them will help you to get started quickly.

VRML Editing Tools (p. 5-2)

Description of the differences between general and native editors

Deformation of a Sphere Example  
(p. 5-5)

Tutorial for creating a simple virtual world with V-Realm Builder and associating this virtual world with Simulink blocks from the Virtual Reality Toolbox

VRML Data Types (p. 5-20)

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

## VRML Editing Tools

There is more than one way to create a virtual world described with the 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 having to know anything about the VRML language. However, you need to understand the structure of a VRML tree to connect your virtual world to Simulink blocks and signals.

This section includes the following topics:

- “Editors for Virtual Worlds” on page 5-2 — General 3-D and native VRML editors
- “V-Realm Builder” on page 5-4 — Native VRML editor shipped with the PC version of the Virtual Reality Toolbox

For a description of the tools to view virtual worlds, see Chapter 6, “Viewing Virtual Worlds.”

### Editors for Virtual Worlds

A VRML file uses a standard text format that you can read with any text editor. Reading the text is useful for debugging, automated processing, and directly changing VRML code. Also, if you use the correct VRML syntax, you can use any common text editor to create virtual scenes in the same way you create HTML pages.

Many people prefer to create simple virtual worlds using their favorite text editor. However, the primary way for you to create a virtual world is with a 3-D editing tool. These tools allow you to create complex virtual scenes without a deep understanding of the VRML language.

These 3-D editing tools offer the power and versatility necessary 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. For VRML authoring, there are basically two types of 3-D editing tools:

- General 3-D authoring packages that can export into VRML format
- Native VRML authoring tools

**General 3-D Editors** — General 3-D editors do not use VRML as their native format. They export their formats to VRML. There are many commercial packages, such as 3D Studio, SolidWorks, or mantra4D, that can do this. 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 can be very powerful, expensive, and complex to learn, but others are relatively inexpensive and might satisfy your specific needs.

It is interesting to note that the graphical user interfaces for many of the general commercial 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 a good overview of the model structure and a convenient shortcut to 3-D element definitions.

**Native VRML Editors** — Native VRML editors use VRML as their native format. This guarantees that all the features in the editor are compatible with VRML. Also, native VRML editors can use features that are unique for the VRML format, like interpolators and sensors.

Unfortunately, there are currently few advanced VRML editors of commercial quality. Most native VRML editors are in the development stage and are harder to use than a general 3-D editor. V-Realm Builder by Ligos Corporation is one of the exceptions. It is one of the most advanced VRML editing tools currently available for personal computers. V-Realm Builder is available only for Windows operating systems. You can access V-Realm Builder documentation at

<http://www.mathworks.com/support/product/VR/productnews/productnews.html>.

For PCs, the Virtual Reality Toolbox includes V-Realm Builder as a native 3-D editor. For more information, see “V-Realm Builder” on page 5-4 and “Deformation of a Sphere Example” on page 5-5.

## **V-Realm Builder**

V-Realm Builder is a flexible, graphically oriented tool for 3-D editing and is available for Windows operating systems only. It is a native VRML authoring tool that provides a convenient interface to the VRML syntax. Its primary file format is VRML. Its graphical user interface (GUI) 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 (tree viewer) of all the elements present in the virtual world.

These structure elements are called nodes. V-Realm Builder lists the nodes and their properties according to their respective VRML node types, and it supports all 54 VRML97 types. For each type of node, there is a specific tool for convenient modification of the node parameters. You can access node properties in two ways:

- Using dialog boxes accessible from the tree viewer
- Directly, using a pointing device

In many cases, it is easier to use the tree viewer to access nodes because it can be difficult to select a specific object in a 3-D scene. The tree also lets you easily change the nesting levels of certain nodes to modify the virtual world according to your ideas. In the tree viewer, you can give the nodes unique names — a feature necessary for working with Virtual Reality Toolbox.



## Deformation of a Sphere Example

The example in this section shows you how to create a simple virtual world using V-Realm Builder. It does not describe everything you can do with V-Realm Builder, but it does describe the basics to get you started.

This example assumes you finished the installation of V-Realm Builder using the function `vrinstall`. See “Installing the VRML Editor (Windows)” on page 2-29.

This section includes the following topics:

- “Defining the Problem” on page 5-5
- “Adding a Virtual Reality Toolbox Block” on page 5-6
- “Creating a Sphere in a Virtual World” on page 5-8
- “Creating a Box in a Virtual World” on page 5-13
- “Connecting a Simulink Model to a Virtual World” on page 5-16

### Defining 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 it in both its  $y$  and  $z$  dimensions.

Positions and dimensions of the objects are listed in the following table.

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

Your first task is to open a Simulink model and add a Virtual Reality Toolbox block to your model. See “Adding a Virtual Reality Toolbox Block” on page 5-6.

**Simulink model** — The Virtual Reality Toolbox includes the tutorial model `vrtut3.mdl`. 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 Virtual Reality Toolbox block. The simulation stops when the sphere is deformed to 0.4 times its original size in the  $x$  direction.

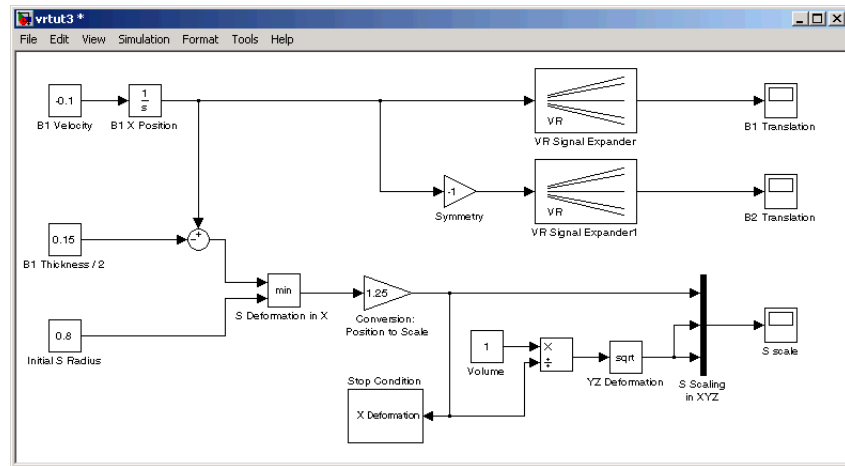
## Adding a Virtual Reality Toolbox Block

This procedure uses the Simulink model `vrtut3.mdl` as an example to explain how to add a Virtual Reality Toolbox block to your model. The model generates the values for the position of B1, the position of B2, and the dimensions of S for the problem previously defined. See “Defining the Problem” on page 5-5.

- 1 From the directory `C:\<matlab root>\toolbox\vr\vr demos\`, copy the file `vrtut3.mdl` to your MATLAB working directory.
- 2 Start MATLAB, and then change the current directory to your MATLAB working directory.
- 3 In the MATLAB Command Window, type

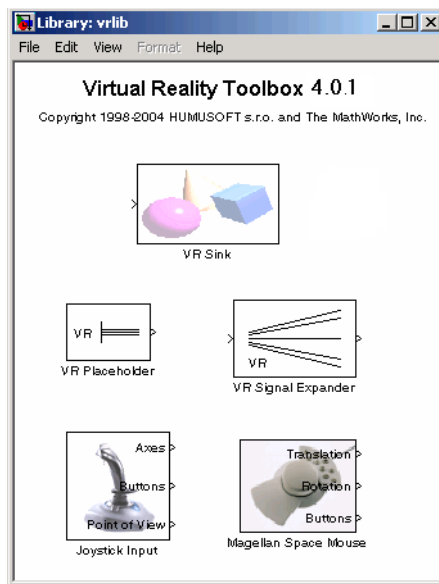
```
vrtut3
```

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



4 From the MATLAB Command Window, type  
`vrlib`

The Virtual Reality Toolbox library opens.



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

Your next task is to create a virtual world that you will associate with the VR Sink block. See “Creating a Sphere in a Virtual World” on page 5-8.

### Creating a Sphere in a Virtual World


You need to create a virtual world before you can connect it to a Simulink model and visualize signals.

After you add a VR Sink block to your Simulink model, you can create a virtual world using V-Realm Builder. This procedure uses the model `vrutut3.mdl` as an example and assumes that you have opened the model and that you have added a VR Sink block. See “Adding a Virtual Reality Toolbox Block” on page 5-6.

- 1 From the Microsoft Windows task bar, click **Start**, and then click **Run**.
- 2 In the **Run** dialog box, enter

```
<matlab root>\toolbox\vr\vrealm\program\vrbuild2.exe
```

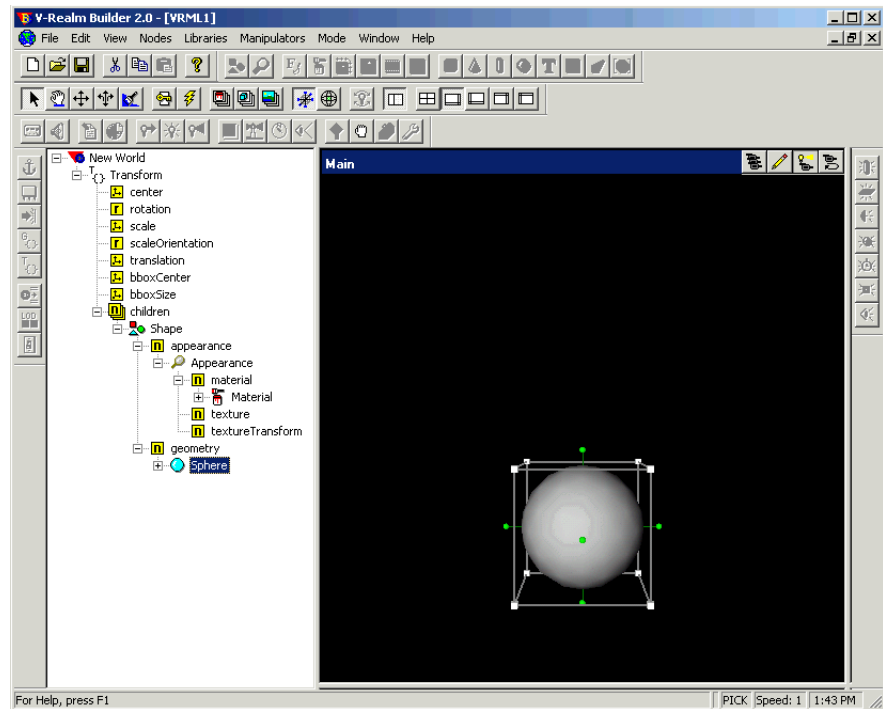
The V-Realm Builder application window opens.

- 3 From the **File** menu, click **New** or click the blank page icon .

In the left pane, V-Realm Builder displays an empty VRML tree, and in the right pane it displays an empty virtual world.

- 4 On the toolbar, click the sphere icon .

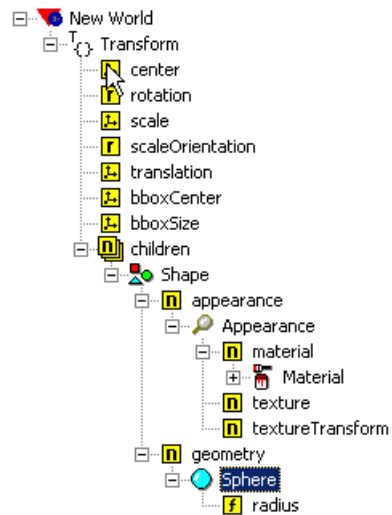
In the left pane you can see the VRML tree for a sphere. This tree includes the following nodes: Transform, Shape, Appearance, Material, and Sphere. A yellow icon indicates the field of a node.



The top-level node is the Transform node. This grouping node allows you to change the position and scale of objects (children) that are part of this node. Its subtree consists of one object, which is described in the Shape node. The Shape node contains the appearance and geometry fields.

##### 5 Expand the Sphere node.

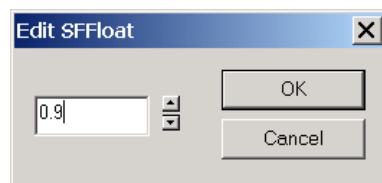
The radius field appears. The yellow icon indicates the type of value. In this case, **f** indicates a value with the type SFFloat. SFFloat is a 32-bit floating-point value.



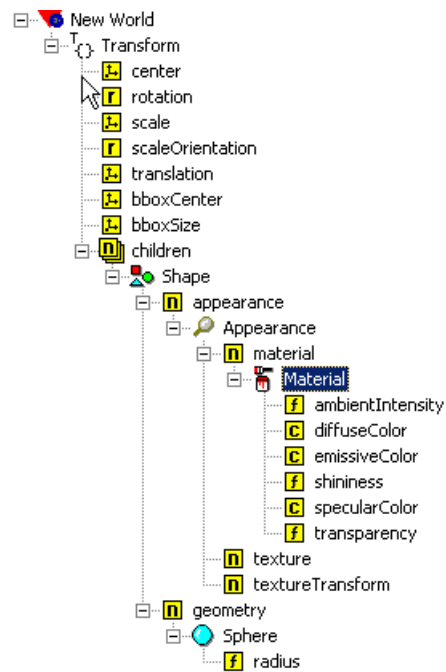
- 6 Double-click the radius field.

The **Edit SFFloat** dialog box opens.

- 7 In the text box, enter 0.9, and then click **OK**. In the right pane, the sphere appears smaller.



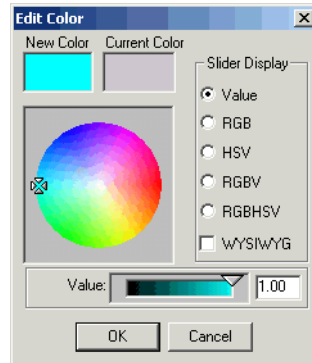
- 8 Under the Shape node, expand the appearance field. Under the appearance field, expand the Appearance node. Under the Appearance node, expand the material field. Under the material field, expand the Material node.



9 Under the Material node, double-click the diffuseColor field.

The **Edit Color** dialog box opens.

- 10** Set the color to blue or any other color you would like, and then click **OK**.



- 11** If you want to check or modify the position of the sphere, double-click the translation field under the Transform node. You do not need to change the default values from [0 0 0].

In this exercise, you want to deform the sphere. You can apply deformation by changing the scale field of the sphere's Transform node. The Virtual Reality Toolbox requires you to assign a name to this node so that it can access it. In VRML syntax, the named nodes are indicated by the "DEF Name Node" statement. V-Realm Builder lists node names next to their icons in the tree viewer.

- 12** Click the Transform node, and then click the node a second time.

The text appears in edit mode.


- 13** Enter a name for the node. For example, enter the letter S, and then click anywhere to exit the text mode.

Your next task is to create two boxes in the virtual world. See "Creating a Box in a Virtual World" on page 5-13.



## Creating a Box in a Virtual World

This topic describes how to create two boxes in the virtual world.

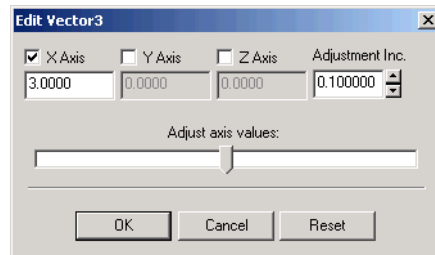
- 1 In the tree, click New World (the topmost item). On the toolbar, click the box icon .

A new box object appears at the same position as the sphere centered in the origin of the coordinate system. Note that the sphere is hidden behind the box and currently is not visible.

- 2 Double-click the translation field under the Transform node.

The **Edit Vector 3** dialog box opens. Notice that there are two Transform nodes. Use the one with the Box node in its subtree.

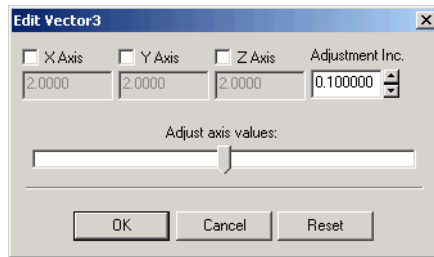
- 3 Select the **X Axis** check box. In the text box below, enter **3**, then click **OK**.



The position of the box is set to [3 0 0].

- 4 Expand the Box node. Double-click the size field under the Box node.

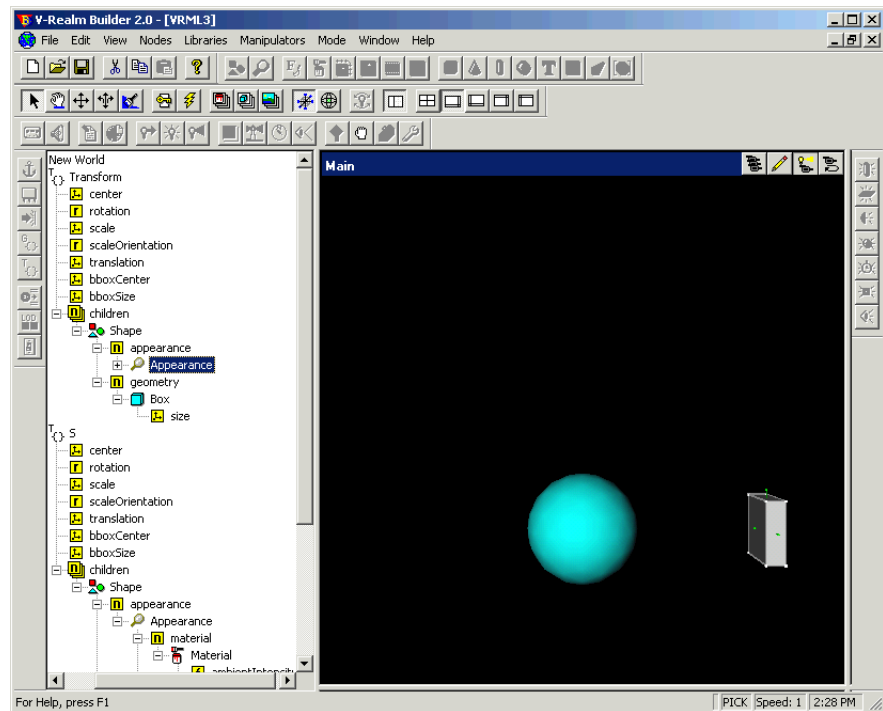
The **Edit Vector 3** dialog box opens.



5 Set the values to [0.3 1 1], and then click **OK**.

**Note** You might need to select the `diffuseColor` node to adjust the box.

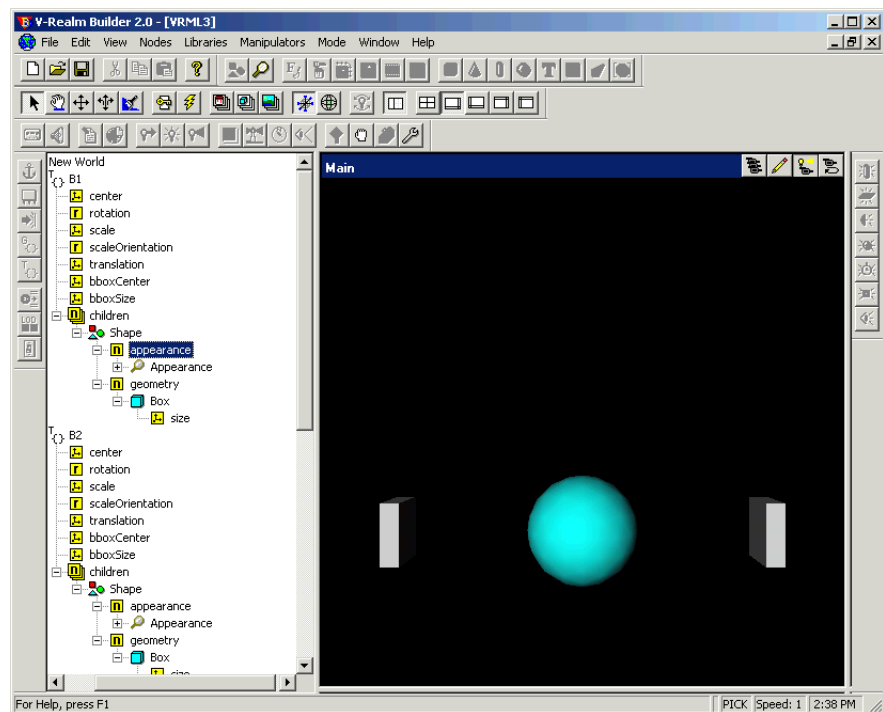
The following figure shows the tree with the expanded branch of the box.



- 6 Create a second box the same way you created the first box.
- 7 To move the second box to its correct place, double-click the translation field of the second Transform node, and change its position to  $[-3 \ 0 \ 0]$ .
- 8 Double-click the size field under the Box node. Set the values to  $[0.3 \ 1 \ 1]$ , and then click **OK**.

The scene is now complete.

- 9 To access the positions of the boxes from a Virtual Reality Toolbox block, give each Transform node a name. For example, set the name of the first Transform node to B1 and the second Transform node to B2. The Virtual Reality Toolbox allows you to access fields of only those nodes that are named in virtual worlds.



- 10 Save the virtual world as `vrtut3.wr1` in the same working directory where the file `vrtut3.mdl` resides, and then exit V-Realm Builder.

---

**Caution** If you want to use your virtual worlds with the Virtual Reality Toolbox, do not save them in a compressed Gzip format.

---

Your next task is to connect the model outputs to the Virtual Reality Toolbox block in your Simulink model. See “Connecting a Simulink Model to a Virtual World” on page 5-16.

### Connecting a Simulink Model to a Virtual World

After you create a virtual world and a Simulink model, and add a Virtual Reality Toolbox block to your model, you can define the associations between the model signals and the virtual world. This procedure uses the model `vrtut3.mdl` as an example. It assumes that you have opened the model and that you have added a VR Sink block. See “Adding a Virtual Reality Toolbox Block” on page 5-6.

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

The viewer displays.

- 2 Select the **Simulation** menu **Block Parameters** option

The **Parameters: VR Sink** dialog box opens again.

- 3 Click **Browse**.

The **Select World** dialog box opens.

- 4 Select `vrtut3.wr1`, and then click **Open**.

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

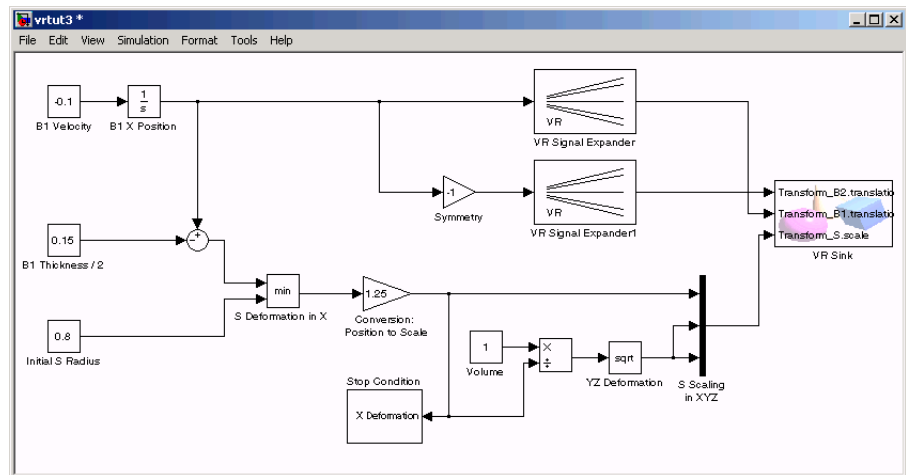
This check box specifies that a viewer for the virtual world starts when you run the model.

- 6 In the **Description** field, type `vrtut3`.

- 7 Click **Apply** in the **Parameters: VR Sink** dialog box.
- 8 In the tree viewer, select the **S** scale, **B1** translation, and **B2** translation check boxes as the nodes you want to connect to your model signals. Click **OK** to close the dialog box.

The Virtual Reality Toolbox block appears with corresponding inputs.

- 9 Connect these input lines to the matching signals in the model. These signals were originally connected to Scope blocks.

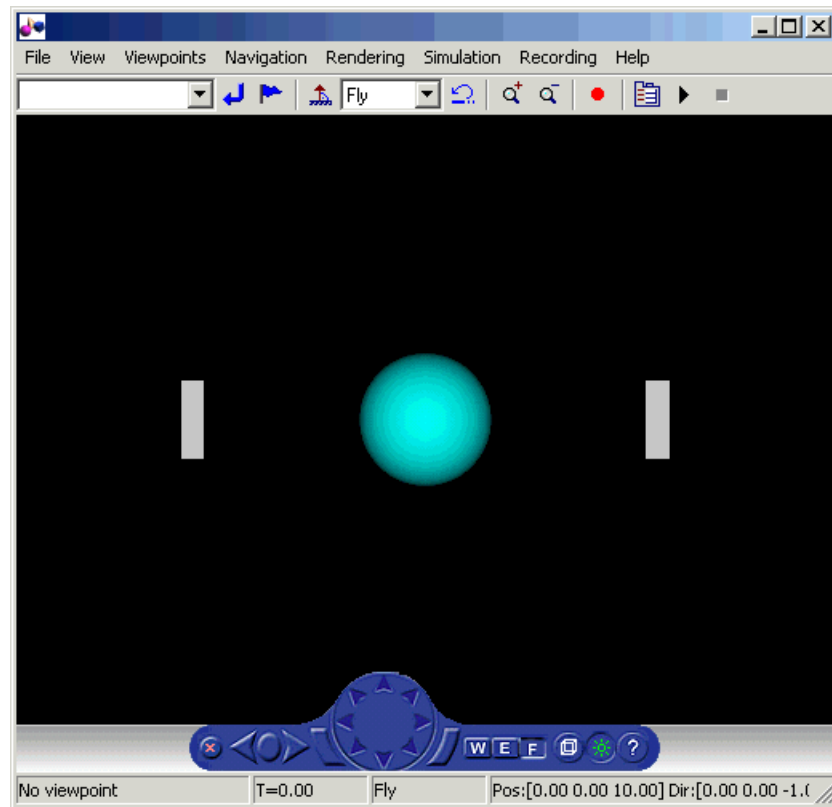


- 10 Double-click the VR Sink block.

The viewer displays.

- 11 Select the **Simulation** menu **Block Parameters** option.
- 12 In the **Parameters: VR Sink** dialog box, click the **View** button.

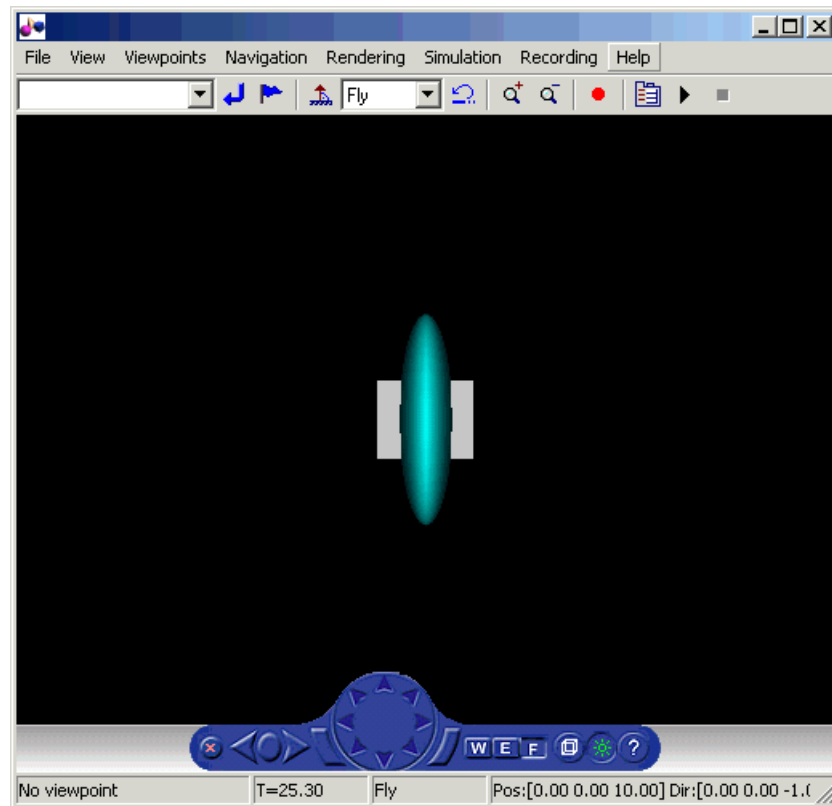
Your default viewer opens and displays the virtual world. For more information on changing your default viewer, see “Setting the Default Viewer of Virtual Scenes” on page 2-24.



**13** In the Simulink window, from the **Simulation** menu, click **Start**.

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

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 includes the following topics:

- “VRML Field Data Types” on page 5-20
- “VRML Data Class Types” on page 5-24

### VRML Field Data Types

Virtual Reality Toolbox 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   | VR Toolbox Type       |
|-----------|---|-----------------------|
| SFBool    | Boolean value true or false.  | logical               |
| SFFloat   | 32 bit floating-point value.  | single                |
| SFInt32   | 32 bit signed-integer value.  | int32                 |
| SFTime    | Absolute or relative time value.  | double                |
| SFVec2f   | Vector of two floating-point values that you usually use for 2-D coordinates. For example, texture coordinates. | Single array (1-by-2) |



| <b>VRML Type</b> | <b>Description</b>  | <b>VR Toolbox Type</b> |
|------------------|---|------------------------|
| 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.  | N/A                    |
| SFString         | String in UTF-8 encoding. Compatible with ASCII, allowing you to use Unicode characters.  | char                   |
| SFNode           | Container for a VRML node.  | N/A                    |
| MFFloat          | Array of SFFloat values.  | Single array (n-by-1)  |
| MFInt32          | Array of SFInt32 values.  | int32 array (n-by-1)   |

| <b>VRML Type</b> | <b>Description</b>          | <b>VR Toolbox Type</b> |
|------------------|-----------------------------|------------------------|
| MFVec2f          | Array of SFVec2f values.    | Single array (n-by-2)  |
| MFVec3f          | Array of SFVec3f values.    | Single array (n-by-3)  |
| MFCOLOR          | Array of SFCOLOR values.    | Single array (n-by-3)  |
| MFRotation       | Array of SFRotation values. | Single array (n-by-4)  |
| MFString         | Array of SFString values.   | char array (n-by-1)    |
| MFNode           | Array of SFNode values.     | N/A                    |

Virtual Reality Toolbox can work with various MATLAB data types, converting them if necessary:

- The `setfield` function (and its dot notation form) and **VR Sink** inputs 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 Virtual Reality Toolbox `vrsetpref` function to define the data type support. Their names are

| <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 '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.



# Viewing Virtual Worlds

---

After you create a virtual world in VRML (as described in Chapter 5, “Virtual Worlds”), you can visualize that virtual world with the Virtual Reality Toolbox viewer or with a VRML-enabled Web browser. The Virtual Reality Toolbox includes its own viewer as the default for all supported platforms. It is the preferred method of viewing virtual worlds. For PC platforms, Virtual Reality Toolbox also includes a VRML plug-in (blaxxun Contact) that you can use as an alternative viewer for virtual worlds. A basic understanding of these tools and how to use them will help you to get started quickly.

|   |   |
|---|---|
| Virtual Reality Toolbox Viewer (p. 6-2) | Description of the Virtual Reality Toolbox viewer                                       |
| blaxxun Contact VRML Plug-In (p. 6-44)  | Description of the blaxxun Contact VRML plug-in that you can use to view virtual worlds |

## Virtual Reality Toolbox Viewer

The Virtual Reality Toolbox 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. The following topics provide an overview of the features and controls of the viewer. This section uses the `vrpend`, `vrplanets`, and `vrtut1` demos to illustrate the viewer features.

- 1** Select a Virtual Reality demo and type that demo’s name in the MATLAB Command Window. For example:

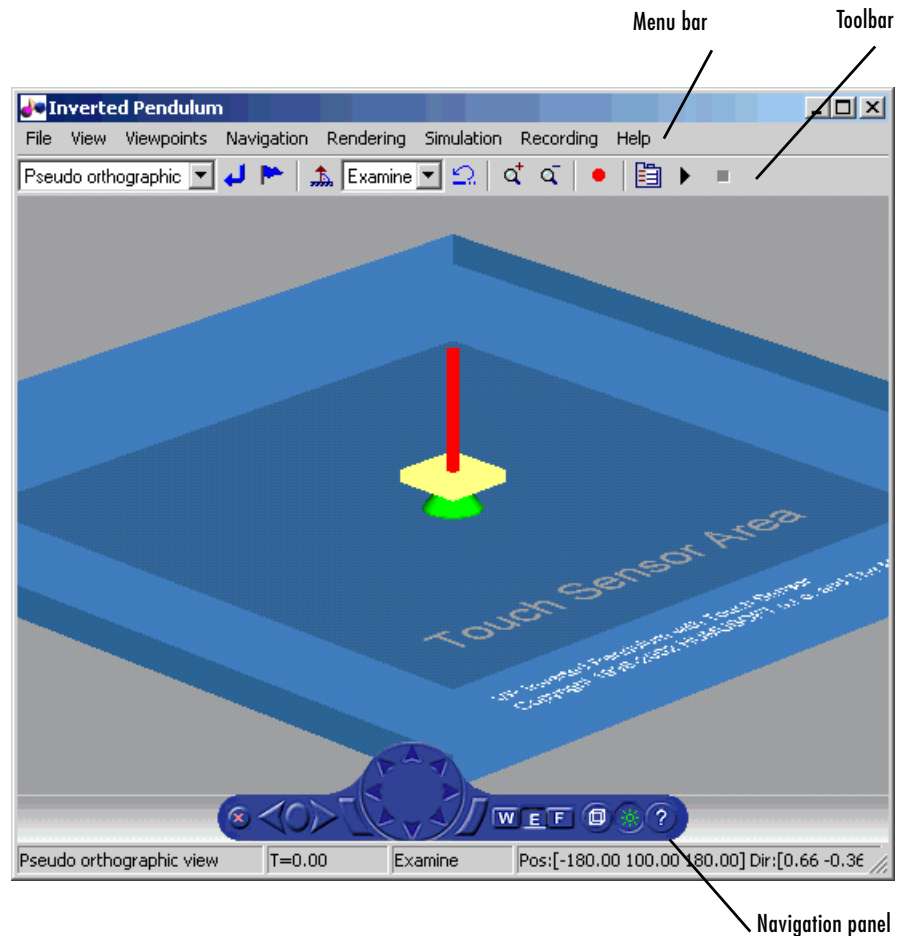
```
vrpend
```

The Simulink model is displayed. By default, the Virtual Reality Toolbox 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 Virtual Reality Toolbox viewer displays the virtual scene. The top of the viewer contains a menu bar and toolbar. The bottom of the viewer contains a navigation panel. These three areas give you alternate ways to work with the virtual scene.





**Note** The Virtual Reality Toolbox viewer settings are saved when you save your model file.

By default, the Virtual Reality Toolbox viewer displays the virtual scene with a navigation panel at the bottom.










### Menu Bar

The Virtual Reality Toolbox viewer menu bar has the following menus:

- **File** — General file operation options, including:
  - **New Window** — Opens another window for the virtual scene. You might want to use this option if you want to have multiple views of the virtual scene open.
  - **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. 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 viewer window.
- **View** — Enables you to customize the Virtual Reality 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/off (see “Navigation” on page 6-10 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 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 “Rendering” on page 6-35).
- **Simulation** — Manages model starts/stops and VR block parameters.
- **Recording** — Manages animation file recording parameters.
- **Help** — Displays the Help browser for the Virtual Reality Toolbox viewer.

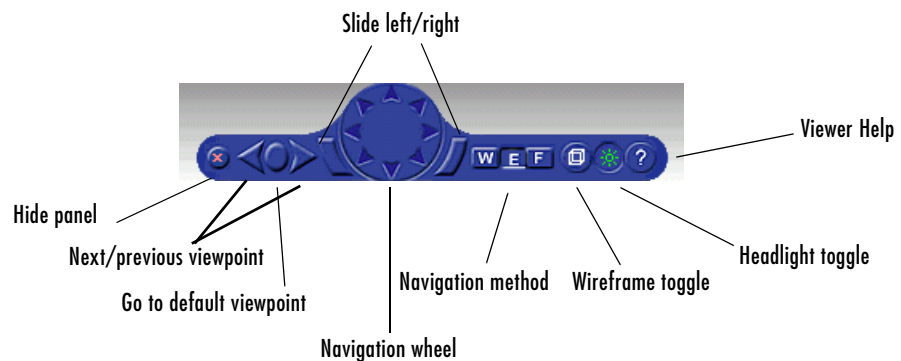
## Toolbar

The Virtual Reality Toolbox viewer 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 
- **Start/stop recording** button 
- **Block parameters** button 
- **Start/pause/continue simulation** button 
- **Stop simulation** button 

## Navigation Panel

The Virtual Reality Toolbox viewer 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 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                       |                          |
| Zoom In          | X               | X              |                         | <b>+</b>                 |
| Zoom Out         | X               | X              |                         | <b>-</b>                 |
| Normal (100%)    | X               |                |                         |                          |

| <b>Operation</b>          | <b>Menu Bar</b> | <b>Toolbar</b> | <b>Navigation Panel</b> | <b>Keyboard Shortcut</b>                 |
|---------------------------|-----------------|----------------|-------------------------|--|
| 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               |                |                         |  |
| X axis                    | X               |                |                         |  |
| Z axis                    | X               |                |                         |  |
| Method                    | X               | X              | X                       | <b>Shift+W,<br/>Shift+E,<br/>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>F10</b>                               |
| Antialiasing              | X               |                |                         | <b>F8</b>                                |
| Headlight                 | X               |                | X                       | <b>F6</b>                                |

| <b>Operation</b>     | <b>Menu Bar</b> | <b>Toolbar</b> | <b>Navigation Panel</b> | <b>Keyboard Shortcut</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>                                     |
| Recording Parameters | X               | X              |                         |   |
| 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>                             |

| <b>Operation</b>             | <b>Menu Bar</b> | <b>Toolbar</b> | <b>Navigation Panel</b> | <b>Keyboard Shortcut</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>    |

## Starting and Stopping Simulations

You can start and stop simulations of the virtual world from the Virtual Reality Toolbox viewer through the menu bar, toolbar, or keyboard.

- From the menu bar, select the **Simulation** menu **Start** or **Stop** option to start or stop the simulation.
- From the toolbar, click **Start/pause/continue simulation** or **Stop simulation** to start or stop the simulation.
- From the keyboard, press **Ctrl+T** to toggle between starting or stopping the simulation.

---

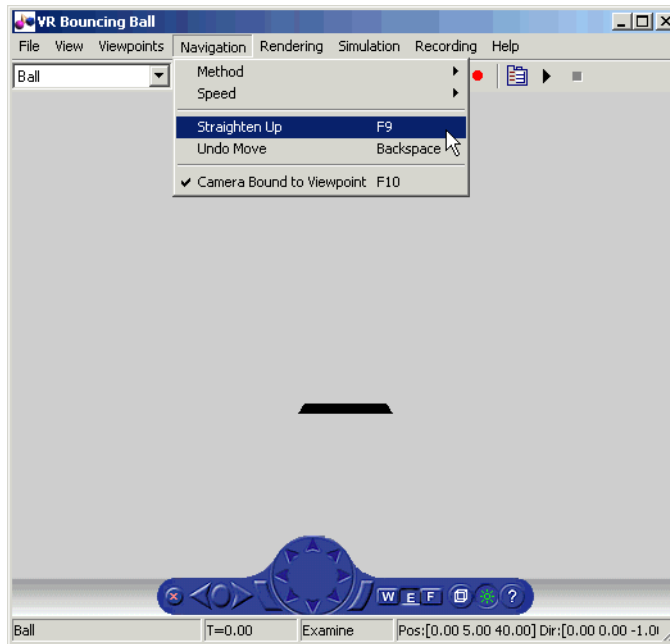
**Note** The **Ctrl+T** operation is available only if you started the viewer from a Simulink model. If you start the viewer through the MATLAB interface, no Simulink model is associated. You cannot start and stop the simulation in this case.

---

## Navigation

You can navigate around a virtual scene using the menu bar, toolbar, navigation panel, mouse, and keyboard. The vrbounce demo shows the viewer's functionality.

**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 “Mouse Navigation” on page 6-13.



- From the toolbar, select the drop-down menu that displays the navigation options Walk, Examine, and Fly.



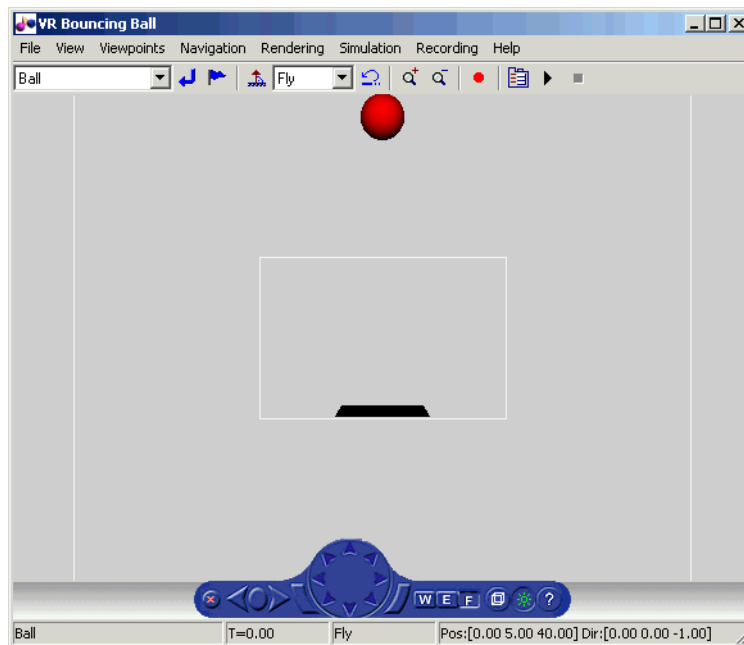
Navigation drop-down menu

- 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 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, from the keyboard, press the **F7** key.

The vrbounce demo 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.

### Mouse Navigation

| <b>Movement Mode</b> | <b>Zone and Description</b>   |
|----------------------|---|
| 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. See "Example of How Sensors Affect Mouse Navigation" on page 6-14 for a description of how sensors affect this navigation.

### 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 **Very Slow**.
- 4 Navigate the virtual world.

Your navigation speed within the virtual world is much slower than before.

---

**Note** Your navigation speed controls the distance you move with each keystroke. It does not affect rendering speed.

---

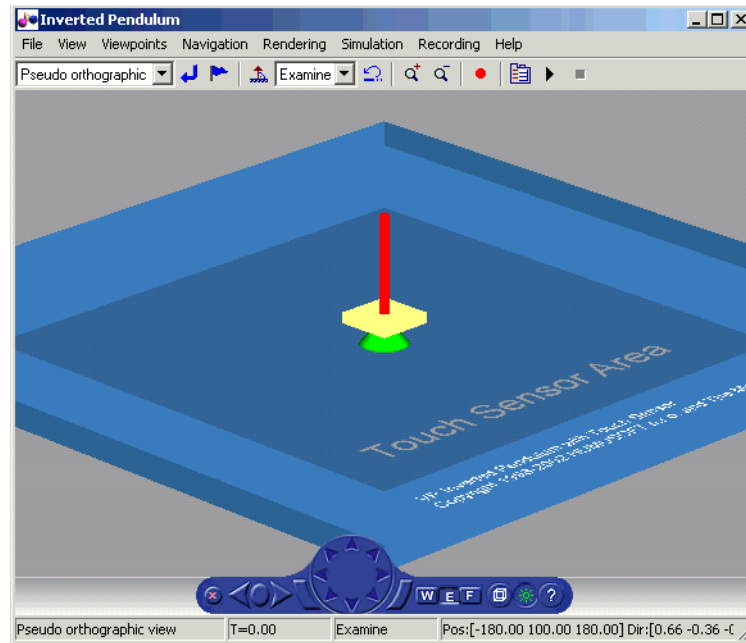
Consider setting a higher speed for large scenes and a slower speed for more controlled navigation in smaller scenes.

### Example of How Sensors Affect Mouse Navigation

- 1 In the MATLAB Command Window, type

```
vrpend
```

at the MATLAB command prompt. The Inverted Pendulum demo starts, and the viewer displays the following scene.



- 2 In the Simulink model window, from the **Simulation** menu, choose **Start**.

The demo starts running.

- 3 Click inside and outside the sensor area of the viewer window. Note that 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. Note that the letters presented do not need to be capitalized in order to perform their intended function.

#### Keyboard Navigation

| Keyboard Command          | Navigation Function   |
|---------------------------|---|
| <b>Backspace</b>          | Undo move.  |
| <b>Ctrl+R</b>             | Start/stop recording.   |
| <b>Ctrl+T</b>             | Start/stop simulation.  |
| <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>                | 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.   |

**Keyboard Navigation (Continued)**

| <b>Keyboard Command</b>                         | <b>Navigation Function</b>  |
|---|---|
| <b>Shift Up/Down Arrow</b>                      | Move the camera forward and backward.   |
| <b>Up/Down Arrow</b>                            | Pan the camera up and down.   |
| <b>Left/Right Arrow, Shift+Left/Right Arrow</b> | Pan the camera right and left.  |
| <b>Alt+Up/Down Arrow</b>                        | Slide up and down.  |
| <b>Alt+Left/Right Arrow</b>                     | Slide left and right.   |
| <b>Ctrl+Left/Right/Up/Down Arrow</b>            | Pressing <b>Ctrl</b> alone acquires the examine lock at the point of intersection between the line perpendicular to the screen, coming through the center of the viewer window, and the closest visible surface to the camera. Pressing the arrow keys without releasing <b>Ctrl</b> rotates the viewpoint about the acquired center point. |
| <b>Shift+Alt+Left/Right Arrow</b>               | Tilt the camera right and left.   |

**Configuring Animation Recording Parameters**

Virtual Reality Toolbox 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 Virtual Reality Toolbox 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 — Virtual Reality Toolbox 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. Virtual Reality Toolbox does not save any navigation movements you make while recording the animation.
- 2-D Audio Video Interleave (AVI) file — Virtual Reality Toolbox writes animation data in an .avi file. Virtual Reality Toolbox uses `vrf` figure objects to record 2-D animation files. The recorded animation reflects exactly what you see in the Virtual Reality Toolbox viewer window, including navigation movements, during the recording.

---

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

---

This section contains the following topic. This topic uses the `vrplanets` demo as the example.

- “Animation Recording File Tokens” on page 6-19 — Describes the filename tokens you can use to direct the Virtual Reality Toolbox viewer to record an animation.

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

- “Recording Files in the VRML Format” on page 6-21 — Describes how to configure the record simulation parameters to create 3-D format animation files.
- “Recording Files in the Audio Video Interleave (AVI) Format” on page 6-22 — Describes how to configure the record simulation parameters to create 2-D format animation files.
- “Scheduling Files for Recording” on page 6-24 — Describes how to schedule record simulation operations to occur automatically.



## Animation Recording File Tokens

By default, the Virtual Reality Toolbox viewer records simulations in a file named with the following format:

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

This format creates a unique filename each time you record the animation. %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 a simulation 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`.

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.

You can

- Create files whose root names are the same as those of the virtual world. This might be useful if you use different virtual worlds for one model.
- Create files in directories relative to the virtual world location. (This is most helpful if you want to ensure that the virtual world file and animation file are in the same directory.)
- Create rolling numbered filenames such that subsequent runs of the model simulation create incrementally numbered filenames. This is useful if you expect to create files of different parts of the model simulation. This feature allows you to run a Simulink model multiple times, but create a unique file at each run.
- Create multiple filenames with time or date stamps, with a unique file created at each run.

The following tokens are the same for .wrl and .avi files.

| <b>Token</b> | <b>Description</b>  |
|--------------|---|
| %d           | The full path to the world VRML file replaces this token in the filename string. For example, the format %d/animdir/animfile.avi saves the animation into the animdir subdirectory of the directory containing the virtual world VRML file. |
| %D           | The current day in the month replaces this token in the filename 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 filename replaces this token in the filename 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 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.  |
| %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.   |

| <b>Token</b>    | <b>Description</b>  |
|-----------------|---|
| <code>%n</code> | The current incremental number replaces this token in the filename string. Each subsequent run of the simulation increments the number. For example, the format <code>%f_anim_%n.wrl</code> saves the animation to <code>vrplanets_anim_1.wrl</code> on the first run, <code>vrplanets_anim_2.wrl</code> on the second run, and so forth. |
| <code>%Y</code> | The current four digit year replaces this token in the filename string. For example, the format <code>%f_anim_%Y.wrl</code> saves the animation to <code>vrplanets_anim_2003.wrl</code> for the year 2003.  |

## Recording 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` demo. It describes how to create an animation filename with the default name format. See “Animation Recording File Tokens” on page 6-19 to save files to other filenames.

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

```
vrplanets
```

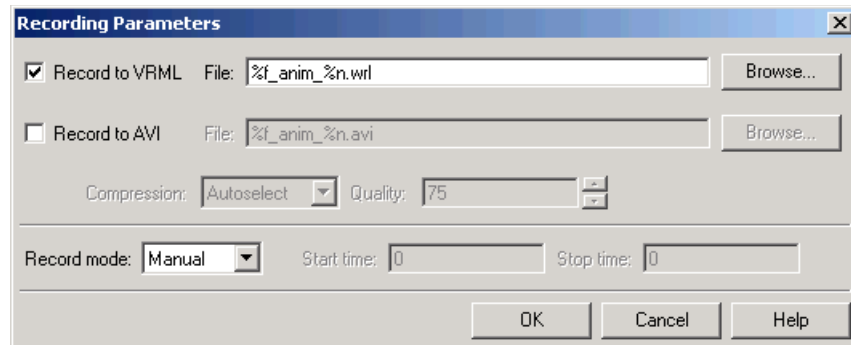
The Simulink model is displayed. Also, by default, the Virtual Reality Toolbox viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the VR Sink block in the Simulink model.

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

The **Recording Parameters** dialog box is displayed.

- 3 Select the **Record to VRML** file check box.

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



**4** Click **OK**.

After you define an animation file, you can manually record simulations. See “Interactively Starting and Stopping Animation Recording” on page 6-26. If you want to record simulations on a schedule, see “Scheduling Files for Recording” on page 6-24.

## Recording 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` demo. It describes how to create an animation filename with the default name format. See “Animation Recording File Tokens” on page 6-19 to save files to other filenames.

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

```
vrplanets
```

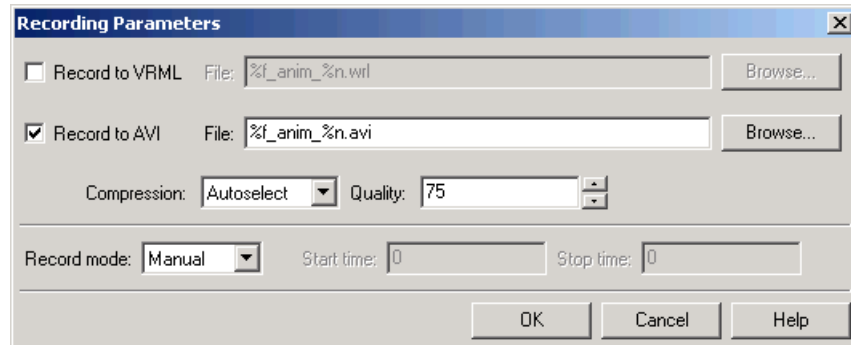
The Simulink model is displayed. Also, by default, the Virtual Reality Toolbox viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the VR Sink block in the Simulink model.

**2** From the **Recording** menu, choose **Recording Parameters**.

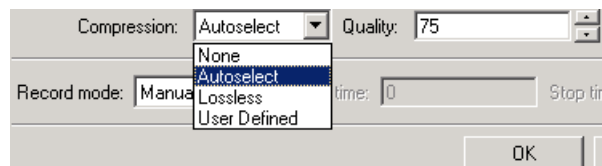
The **Recording Parameters** dialog box is displayed.

**3** Select the **Record to AVI** file check box.

The **File** text field and Compression selection area become active and the default filename, %f\_anim\_%n.avi, appears in the text field.



**4** 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 Virtual Reality Toolbox to select the most appropriate compression codec. Associated with this option is 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 Virtual Reality Toolbox 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. Associated with this option is 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.
- 5** 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 -> Off**.

You can reenable the Navigation Panel (for example, choose **View -> Navigation Panel -> Halfbar**) after you are finished recording the .avi file.

- 6** Click **OK**.

After you define an animation file, you can record animations. See “Interactively Starting and Stopping Animation Recording” on page 6-26. If you want to record animations on a schedule, see “Scheduling Files for Recording” on page 6-24.

### Scheduling 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 demo. It assumes that you have already configured the recording parameters for an animation file. See “Recording Files in the VRML Format” on page 6-21 or “Recording Files in the Audio Video Interleave (AVI) Format” on page 6-22 for further details.

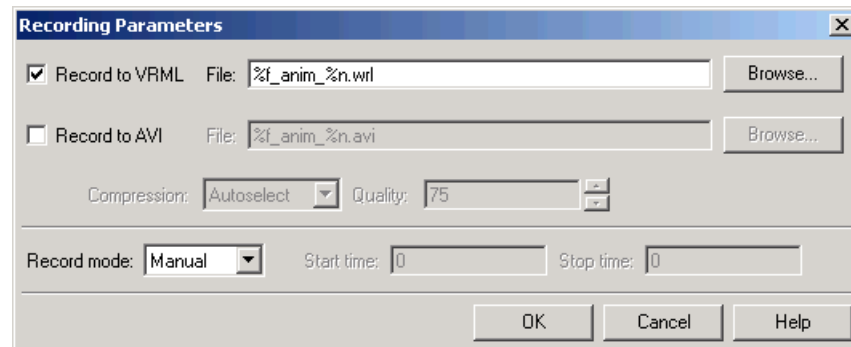
- 1 If the `vrplanets` demo is not already open, type the demo's name in the MATLAB Command Window. For example:

```
vrplanets
```

The Simulink model is displayed. Also, by default, the Virtual Reality Toolbox 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 **Recording Parameters**.

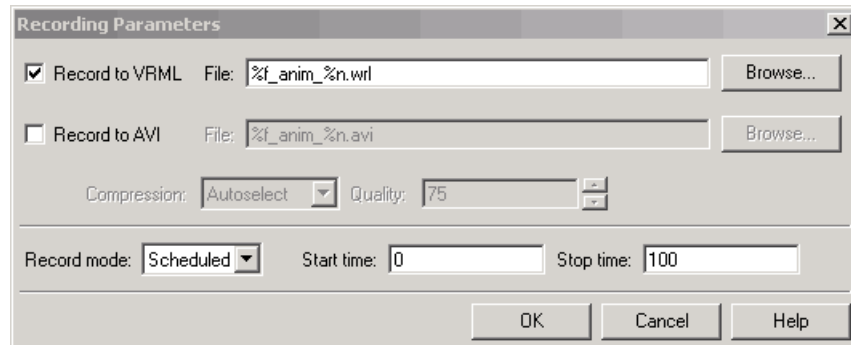
The **Recording Parameters** dialog box is displayed. This dialog box contains the **Record mode** list. Note that the **Record mode** list is enabled only if you also select either or both the **Record to VRML** or **Record to AVI** check box.



- 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.

**5** Click **OK**.

After you define the schedule, you can record simulations. See “Starting and Stopping Simulations” on page 6-9.

---

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

---

### Interactively Starting and Stopping Animation Recording

You can start or stop recording animations of the virtual world from the Virtual Reality Toolbox 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 “Configuring Animation Recording Parameters” on page 6-17 if you have not defined animation files.

- From the menu bar, choose the **Simulation** menu, **Start** 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.

---

## Viewing the Animation File

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 “Recording Files in the VRML Format” on page 6-21 or “Recording Files in the Audio Video Interleave (AVI) Format” on page 6-22 for descriptions on how to create one.

### To View VRML Files

- 1 Change directory 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 Virtual Reality Toolbox viewer for the animation file. Setting the `TimeSource` property to `'freerun'` directs the viewer to advance its time independent of MATLAB.

- 3 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 directory 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.

## Working with Viewpoints

You or visitors to a virtual world navigate through the virtual world environment using the Virtual Reality Toolbox viewer navigation methods Walk, Examine, and Fly. 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 Virtual Reality Toolbox 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.

This topic illustrates viewpoints using the `vrplanets` demo.

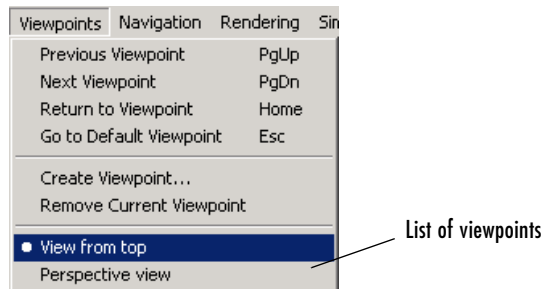
- 1 Select a Virtual Reality Toolbox demo and type that demo's name in the MATLAB command window. For example:

```
vrplanets
```

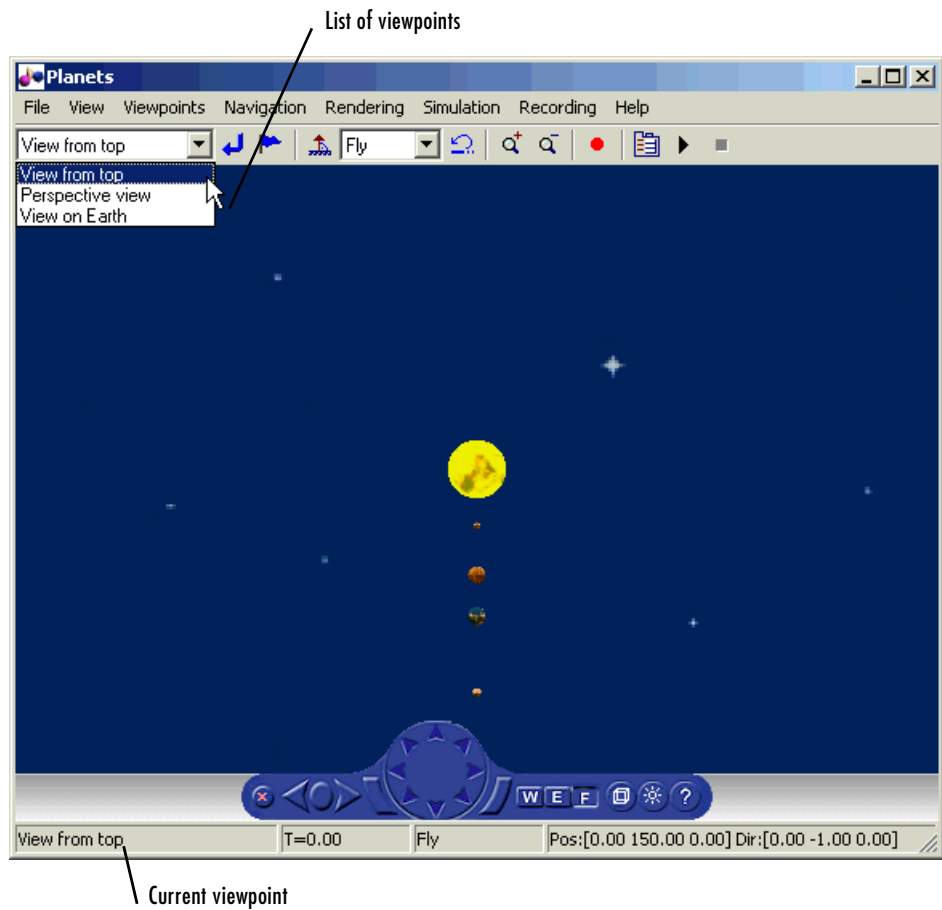
The Simulink model is displayed. By default, the Virtual Reality Toolbox viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the VR Sink block in the Simulink model.

- 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.



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

### Navigating 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 “Resetting Viewpoints” on page 6-32. Resetting the viewpoint is useful when you have been moving about the scene and need to reorient yourself.

### Resetting 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.

### Creating Viewpoints

You can add new viewpoints to the virtual world through the menu bar or toolbar.

- 1 Select a Virtual Reality Toolbox demo and type that demo's name in the MATLAB Command Window. For example:

```
vrplanets
```

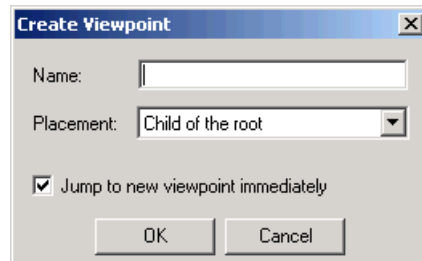
The Simulink model is displayed. Also, by default, the Virtual Reality Toolbox viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the VR Sink block in the Simulink model.

In the Virtual Reality Toolbox 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 as a static one 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.

---

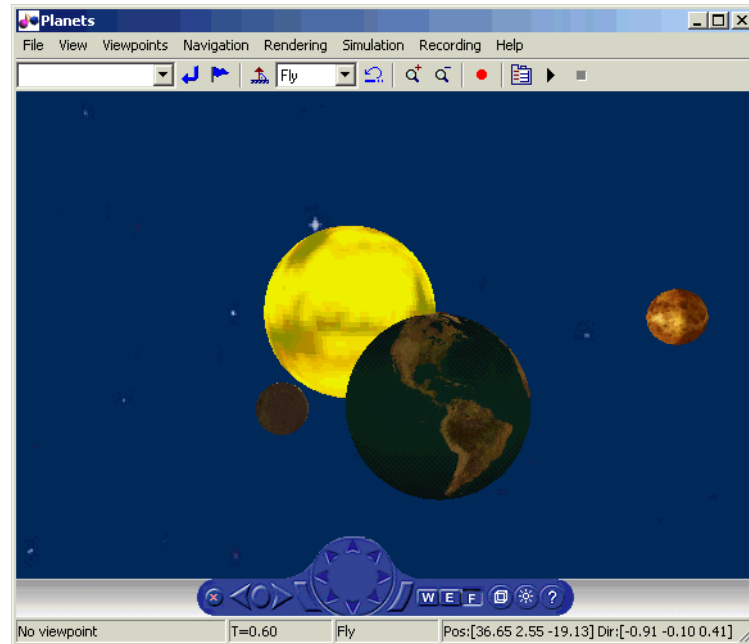
**15** Select the **Jump to new viewpoint immediately** check box to make the new viewpoint become the current viewpoint for the viewer. If you do not select this check box, you still create a new viewpoint, but you remain bound to the current viewpoint, not to the new viewpoint.

**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.

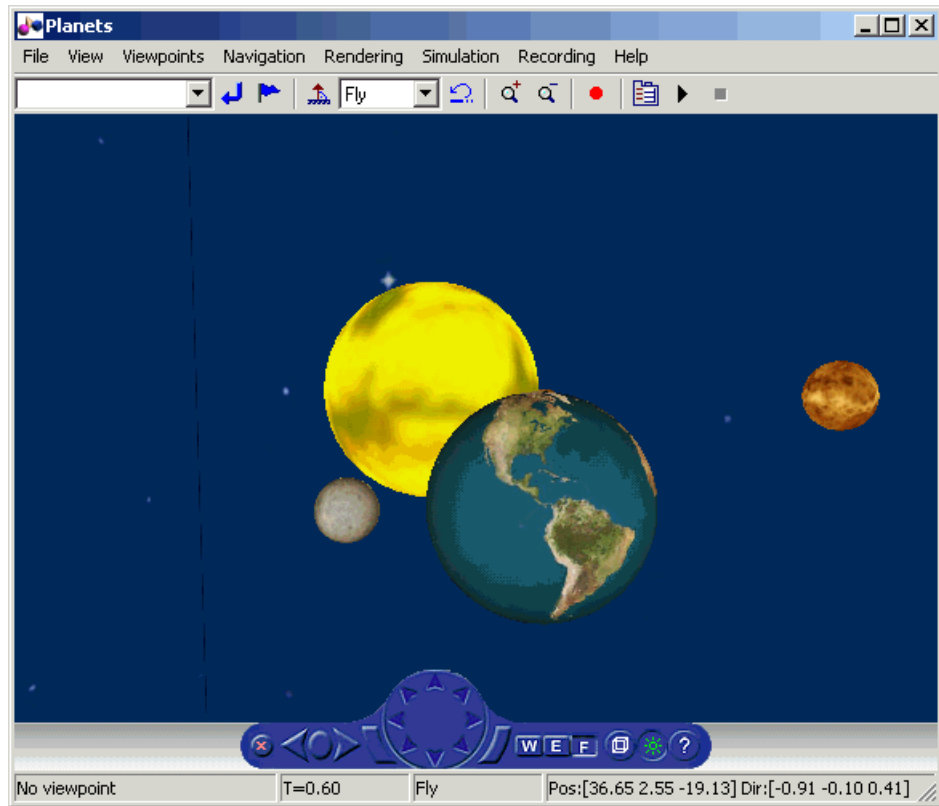




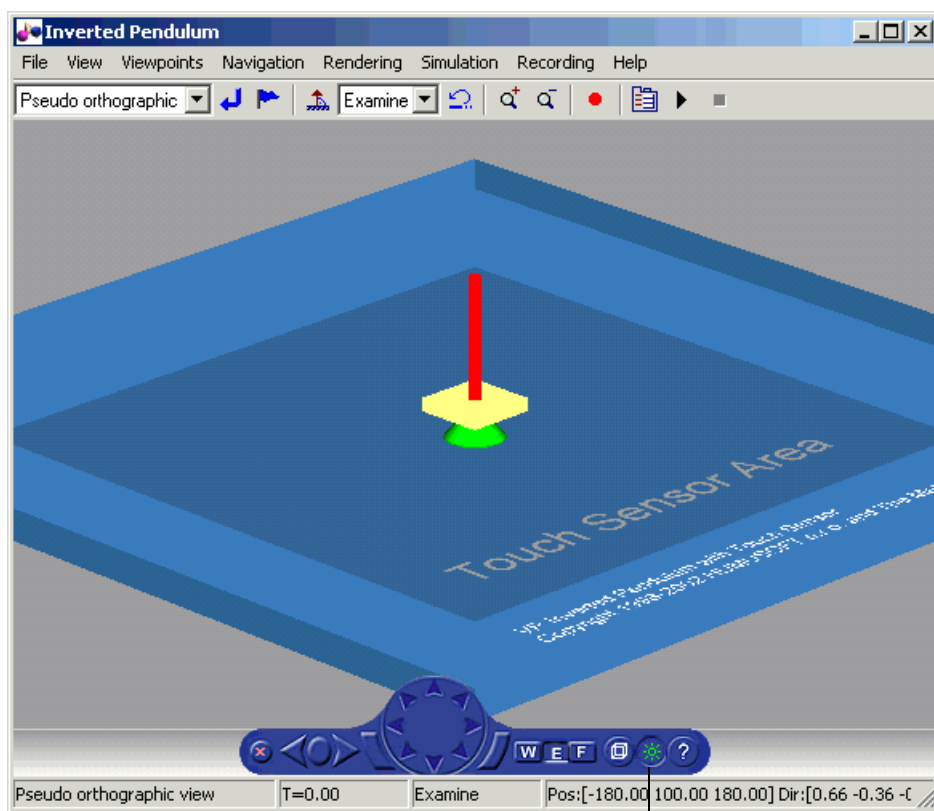
## Rendering

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` demos are used to demonstrate 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. For example, the following figure depicts the `vrplanets` demo View on Earth viewpoint with **Antialiasing** on. To better display the affects of antialiasing, turn **Headlight** on. You can turn antialiasing on or off to observe the differences.

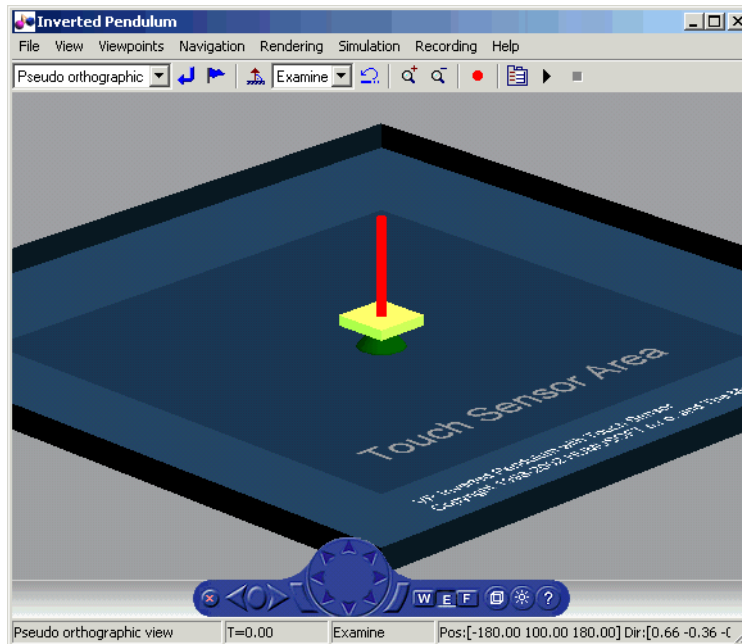


You can turn the camera headlight and the lighting of the scene on or off. When **Headlight** is off, the camera does not emit light. Consequently, the scene can appear dark. For example, the following figure depicts the vrend demo with **Headlight** on.



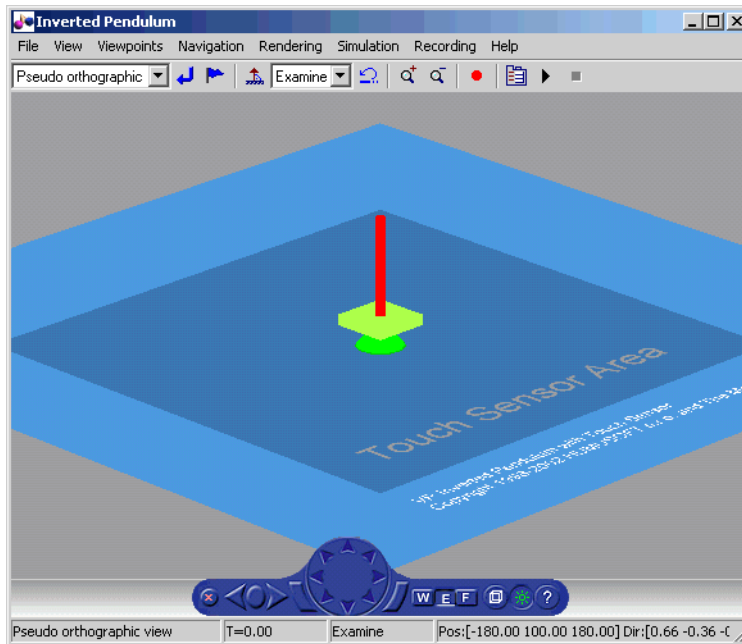
Headlight toggle

The scene looks darker when **Headlight** is set to off.

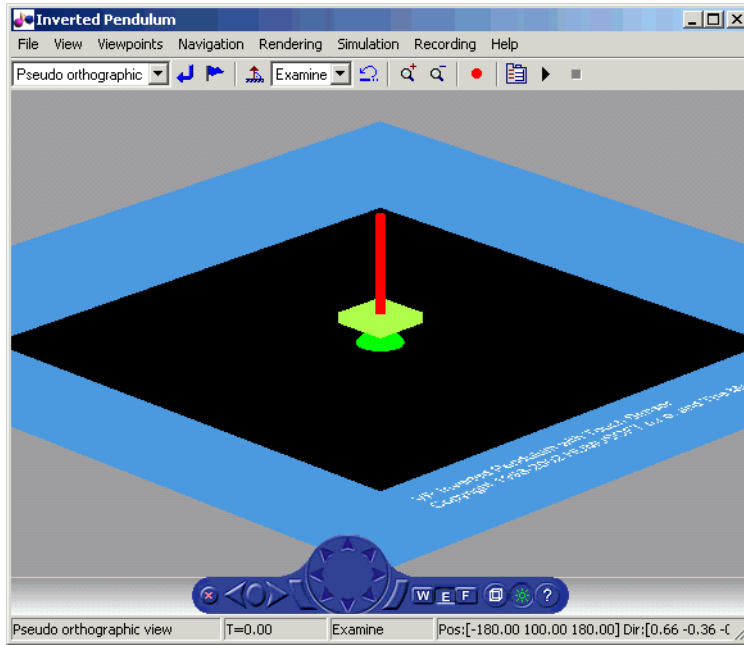


**Note** It is helpful to define enough lighting within the virtual scene so that it is lit regardless of the **Headlight** setting.

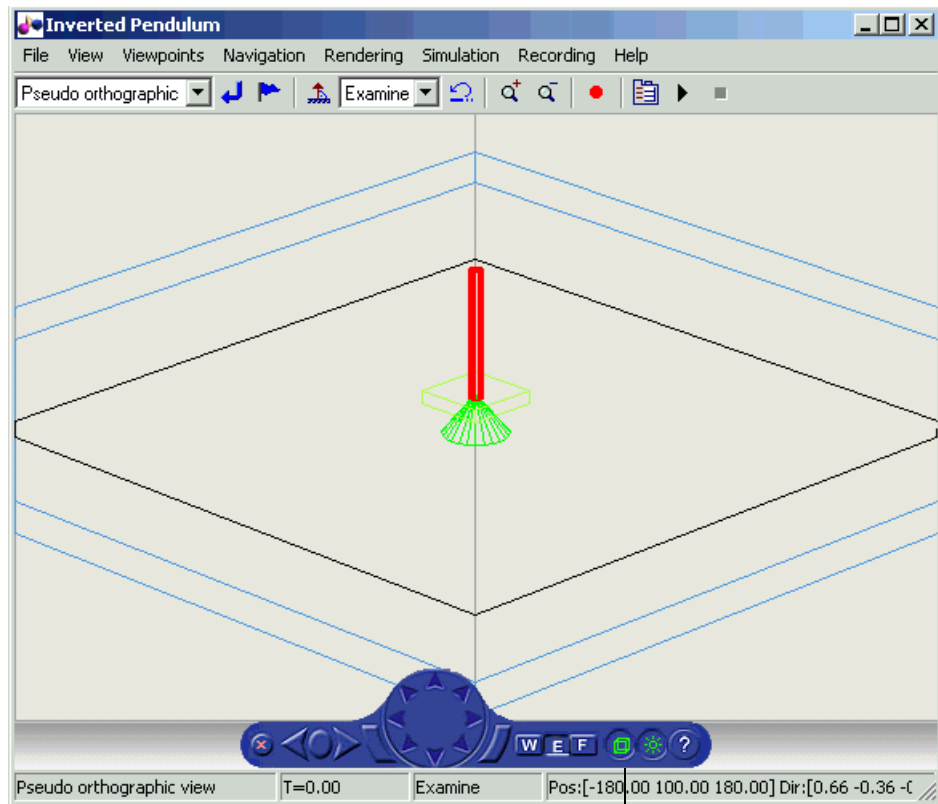
When **Lighting** is off, the virtual world appears as if lit in all directions. The Virtual Reality Toolbox viewer does not compute and render all the lighting effects at the surfaces of the objects. Shadows disappear and the scene loses some of its 3-D quality. The following is the vrpend demo with **Lighting** off.



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

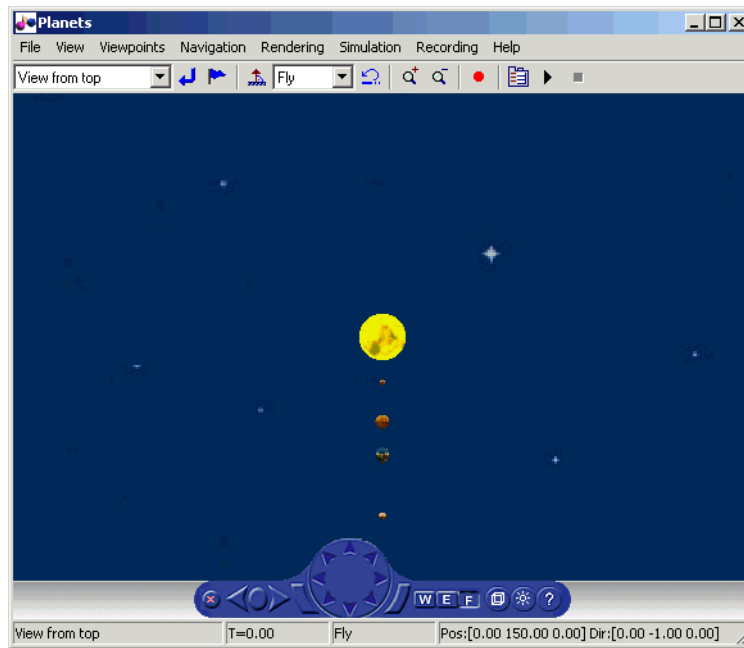


Turning **Wireframe** on changes the scene's objects from solid to wireframe rendering. The following is the vrpend demo with **Wireframe** on.



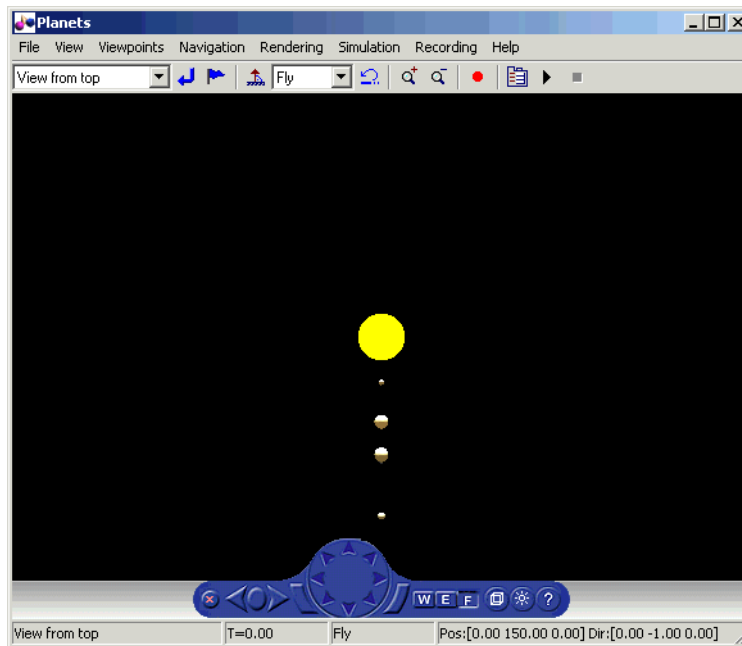
Wireframe toggle

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



If **Textures** is off, objects do not have texture in the virtual scene. The following is the vrplanets demo with **Textures** off.

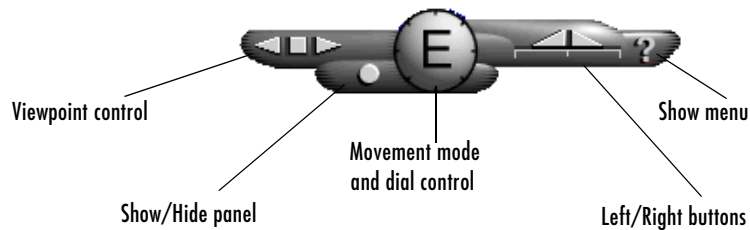




## blaxxun Contact VRML Plug-In

The Virtual Reality Toolbox includes the blaxxun Contact VRML plug-in. This is a VRML plug-in for either Microsoft 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.



### Viewpoint Control

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.

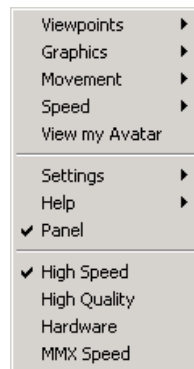
You use the other two triangular buttons to browse forward and backward through author-defined viewpoints of the virtual world. If the author does not define other viewpoints, these buttons are inactive. The keyboard shortcuts for the triangular buttons are the **Page Up** and **Page Down** keys.

## 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 one shown 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.

## Navigation

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.

| <b>Movement Mode</b> | <b>Description</b>   |
|----------------------|--|
| 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.  |

| <b>Movement Mode</b> | <b>Description</b>  |
|----------------------|---|
| 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 Virtual Reality Toolbox 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.

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.

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**.

### 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 Microsoft 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.

## Block Reference

---

## Blocks – Categorical List

### Control Input Devices

|                      |   |
|----------------------|---|
| Joystick Input       | Process input from asynchronous joystick device |
| Magellan Space Mouse | Process input from Magellan Space Mouse device  |

### Virtual Worlds

|         |   |
|---------|---|
| VR Sink | Write data from Simulink model to virtual world |
|---------|---|

### VRML Related Signals

|                    |  |
|--------------------|--|
| VR Placeholder     | Send unspecified value to Virtual Reality Toolbox block      |
| VR Signal Expander | Expand input vectors into fully qualified VRML field vectors |



## **Blocks — Alphabetical List**

This section contains block reference pages listed alphabetically.

# Joystick Input

**Purpose** Process input from asynchronous joystick device

**Library** Virtual Reality Toolbox

## Description



The Joystick Input block provides a convenient interaction between a Simulink model and the virtual world associated with a Virtual Reality Toolbox block. It works only on Windows operating systems.

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.

## Block Parameters Dialog Box

**Joystick ID** — The system ID assigned to the given joystick device. You can find the properties of the joystick 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 block ports do not have the full width provided by the Windows Game Controllers interface. Instead, the Virtual Reality Toolbox dynamically adjusts the ports to correspond to the capabilities of the connected joystick each time the model is opened. 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 the **Enable force-feedback input** check box is selected.

**Enable force-feedback input** — If you select this check box, Virtual Reality Toolbox can support force-feedback joystick, steering wheel, and haptic (one that enables tactile feedback) devices. To use this feature, you must install Microsoft DirectX Version 8.0 or higher.

**Output Ports** — Depending on the **Adjust I/O ports according to joystick capabilities** check box setting previously described, output ports either have fixed maximum width provided by the system Game Controllers interface or the output ports change to correspond to the actual capabilities of the connected joystick.

| <b>Output Port</b> | <b>Value</b>   | <b>Description</b>   |
|--------------------|--|--|
| 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 the range from -1 to 1. |
| Buttons            | Vector of doubles<br>0 — Button released<br>1 — Button pressed                 | Outputs correspond to the current status of joystick buttons.  |
| Point of view      | -1 — Selector inactive<br><0; 360> — The angle of the POV selector, in degrees | Output corresponds to the current status of the joystick Point of View selector.   |

| <b>Input Port</b> | <b>Value</b>                             | <b>Description</b>   |
|-------------------|--|--|
| Force             | Vector of doubles in the range < -1; 1 > | Port active only for force-feedback devices. Inputs correspond to the desired force to be applied in the given axis.<br><br>Please note that usually not all of the device axes have force-feedback. The size of the Force vector is then smaller than the Axes vector size. |

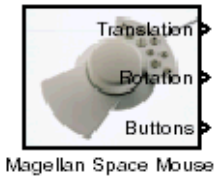
# Magellan Space Mouse

---

**Purpose** Process input from Magellan Space Mouse device

**Library** Virtual Reality Toolbox

## Description



The Magellan 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 block supports all models of Space Mouse and PuckMan devices manufactured by 3Dconnexion. It also supports USB devices. Space Mouse devices are supported only on Windows operating systems.

**Data Type Support** A Magellan Space Mouse block outputs signals of type double.

## Block Parameters Dialog Box

**Port** — Serial port to which the Magellan Space Mouse is connected. Possible values are COM1...COM4 and USB.

**Output Type** — This field specifies how the inputs from the device are transformed:

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

**Disable position movement** — Fixes the positions at the initial values, allowing you to change rotations only.

**Disable rotation 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 mode to **Speed**.

**Position sensitivity** — Mouse sensitivity for translations. Lower values correspond to higher sensitivity.

**Rotation sensitivity** — Mouse sensitivity for rotations. Lower values correspond to higher sensitivity.

**Initial position** — Initial condition for integrated translations. This is not used when you set the output mode to **Speed**.

**Initial rotation** — Initial condition for integrated rotations. This is not used when you set the output mode to **Speed**.

# VR Placeholder

**Purpose** Send unspecified value to Virtual Reality Toolbox block

**Library** Virtual Reality Toolbox

## Description



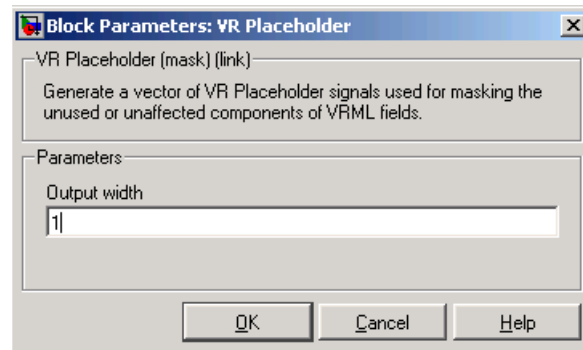
The VR Placeholder block sends out a special value that is interpreted as “unspecified” by the VR Sink block. When this value appears on the VR Sink input, whether as a single value or as an element of a vector, the appropriate value in the virtual world stays unchanged. Use this block to change only 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.

**Purpose** Expand input vectors into fully qualified VRML field vectors

**Library** Virtual Reality Toolbox

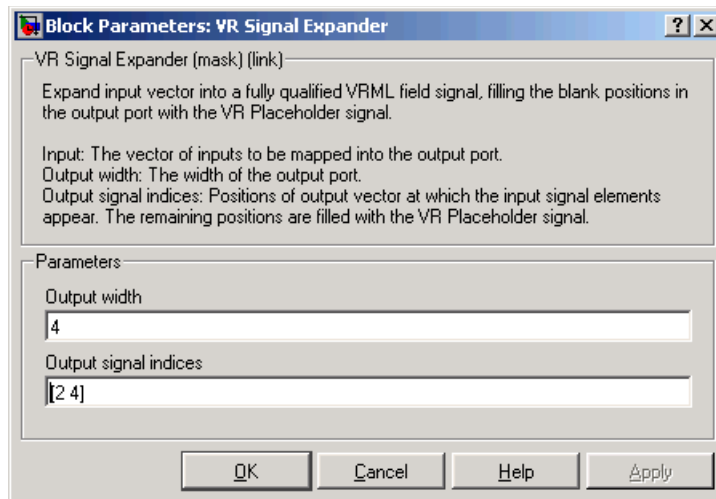
**Description** The VR Signal Expander block creates a vector of predefined length, using some values from the input ports and filling the rest with placeholder signal values.



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.

## VR Signal Expander

---

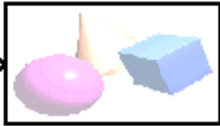
For example, suppose you want an input vector with two signals, and you want an output vector with four signals, and you want the first input signal in position 2 and the second input signal in position 4. In the **Output width** box, enter 4 and in the **Output signal indices** box, enter [2, 4]. The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



**Purpose** Write data from Simulink model to virtual world

**Library** Virtual Reality Toolbox

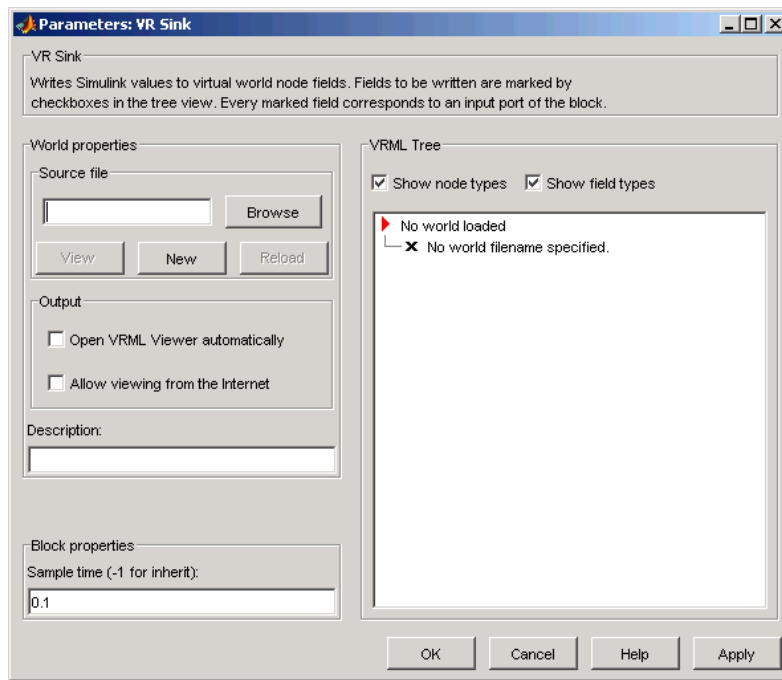
**Description** The VR Sink block writes values from its ports to virtual world fields specified in the **Block Parameters** dialog box.



VR Sink

**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. For further details, see “VRML Field Data Types” on page 5-20.

## Block Parameters Dialog Box



**Source file** — VRML filename specifying the virtual world this block is connected to. The **View** button allows you to view the world in the Virtual Reality Toolbox viewer or a Web browser. The **Edit** button launches an external VRML editor, and the **Reload** button reloads the world after you change it. By default, the full path to the associated .wrl file appears in this text box. If you enter only the filename in this box, the Virtual Reality Toolbox assumes that the .wrl file resides in the same directory as the model file.

**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 it is not selected, the world is visible only on the host computer. This is equivalent to the RemoteView property of a vrworld object. See Chapter 4, “MATLAB Interface.”

**Description** — Description that is displayed in all virtual reality object listings, in the title bar of the Virtual Reality Toolbox viewer, and in the list of virtual worlds on the Virtual Reality Toolbox HTML page. This is equivalent to the Description property of a vrworld object. See Chapter 4, “MATLAB Interface.”

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

---

**Note** To better record the animation, you might want to experimentally change the value of this property.

---

**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 and can be accessed from MATLAB. 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 settable values have check boxes. Use these check boxes to select the fields you want Simulink to output values to. For every selected field, an input port is created in the block. Input ports are assigned to the selected nodes and fields in the order corresponding 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.

You can use the Simulink `get_param` and `set_param` functions to access the following VR Sink block dialog parameters:

| Property      | Possible Values   |
|---------------|---|
| FieldsWritten | String containing list of <code>NodeName.FieldName</code> pairs, separated by #<br>For example<br><code>Membrane.translation#Membrane.rotation</code> |
| SampleTime    | String containing an expression that evaluates to a valid Simulink sample time value  |
| WorldFileName | Associated VRML filename  |

**Note** Use these parameters with care. It is your responsibility to maintain consistency in the block parameters. For example, if you change the `WorldFileName` property, also change the `FieldsWritten` property to reflect the actual nodes and fields accessible in the newly associated VRML file.

# VR Sink

---

# Function Reference

---

## Functions – Categorical List

This topic contains reference pages for MATLAB interface and Virtual Reality Toolbox object functions.

- “MATLAB Interface Functions” on page 8-3 — Interfaces with virtual worlds and miscellaneous features, such as opening the Virtual Reality Toolbox library, closing virtual reality figure windows, and setting and getting Virtual Reality Toolbox preferences.
- “vrworld Object Methods” on page 8-3 — Handle of a virtual scene. It allows you to interact with and control the scene.
- “vrnode Object Methods” on page 8-4 — Handle of a VRML node. It allows you to get and set the node properties. A vrnode object is a child object of a vrworld object.
- “vrfigure Object Methods” on page 8-4 — Handle to the Virtual Reality Toolbox viewer window that allows you to get and set the viewer properties. A vrfigure object is a child object of a vrworld object.

---

**About Virtual Reality Toolbox objects** While the Simulink interface is the preferred method for using the Virtual Reality Toolbox, you can access virtual worlds through the MATLAB interface. To use this interface, you create objects in the MATLAB workspace and associate those objects with your virtual worlds. MATLAB functions and the Simulink interface share the same Virtual Reality Toolbox objects. These objects are accessible from both the MATLAB and Simulink interfaces simultaneously.

---

## **MATLAB Interface Functions**

|                        |  |
|------------------------|--|
| <code>vrclear</code>   | Delete all closed virtual worlds from memory                           |
| <code>vrclose</code>   | Close virtual reality figure windows                                   |
| <code>vrdrawnow</code> | Update virtual world   |
| <code>vrgetpref</code> | Read values of Virtual Reality Toolbox preferences                     |
| <code>vrinstall</code> | Install and check Virtual Reality Toolbox components                   |
| <code>vrlib</code>     | Open Simulink block library for the Virtual Reality Toolbox            |
| <code>vrsetpref</code> | Change Virtual Reality Toolbox preferences                             |
| <code>vrview</code>    | View virtual world using Virtual Reality Toolbox viewer or Web browser |
| <code>vrwho</code>     | List virtual worlds in memory  |
| <code>vrwhos</code>    | List details about virtual worlds in memory                            |

## **vrworld Object Methods**

|                              |  |
|------------------------------|--|
| <code>vrworld</code>         | Create new <code>vrworld</code> object associated with virtual world |
| <code>vrworld/close</code>   | Close virtual world  |
| <code>vrworld/delete</code>  | Delete virtual world from memory                                     |
| <code>vrworld/edit</code>    | Open virtual world file in external VRML editor                      |
| <code>vrworld/get</code>     | Read property value of <code>vrworld</code> object                   |
| <code>vrworld/isvalid</code> | Return 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>   | Delete <code>vrnode</code> object                             |
| <code>vrnode/fields</code>   | Return VRML field summary of node object                      |
| <code>vrnode/get</code>      | Read property value of <code>vrnode</code> object             |
| <code>vrnode/getfield</code> | Get field value of <code>vrnode</code> object                 |
| <code>vrnode/isvalid</code>  | Return 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 clients |

### **vrfigure Object Methods**

|                               |   |
|-------------------------------|---|
| <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>     | Read property value of <code>vrfigure</code> object         |
| <code>vrfigure/isvalid</code> | Return 1 if <code>vrfigure</code> object is valid, 0 if not |
| <code>vrfigure/set</code>     | Change property value of <code>vrfigure</code> object       |
| <code>vrfigure/vrgcf</code>   | Get handle for currently active virtual reality figure      |
| <code>vrfigure/vrgcbf</code>  | Get current callback <code>vrfigure</code> object           |



## **Functions — Alphabetical List**

This section contains function reference pages listed alphabetically.

# vrclear

---

**Purpose** Delete 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 Simulink. 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 Simulink.

**See Also** `vrworld/delete`, `vrworld`

|                    |  |
|--------------------|--|
| <b>Purpose</b>     | Close virtual reality figure windows   |
| <b>Syntax</b>      | <code>vrclose</code><br><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>  |

# vrdrawnow

---

**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

- MATLAB is idle for some time (no Simulink model is running and no M-file is being executed).
- A Simulink step is finished.

**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`.

**Examples** Create a `vrworld` object. At the MATLAB command prompt, type

```
myworld = vrworld('vrmount.wr1')
```

The `vrworld` object `myworld` is associated with the virtual world `vrmount.wr1`.

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 Virtual Reality Toolbox viewer by typing

```
f = vrfigure(myworld)
```

Your viewer opens and displays the virtual scene.

**See Also** `vrworld`, `vrworld/open`

# vrfigure/capture

---

**Purpose** Create RGB image from virtual reality figure

**Syntax** `image_capture = capture(vrfigure_object)`

**Description** `image_capture = capture(vrfigure_object)` captures a virtual reality figure into a TrueColor RGB image that can be displayed by the `image` command.

**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`. 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 Virtual Reality Toolbox viewer by typing

```
f = vrfigure(myworld)
```

Your viewer opens and displays the virtual scene. Next, create an RGB image by typing

```
image_capture = capture(f);
```

Lastly, view the image

```
image(image_capture)
```

The scene from the viewer window is displayed in a MATLAB figure window.

**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>vrworld</code> , <code>vrworld/open</code> , <code>vrfigure</code>   |

# vrfigure/get

**Purpose** Read property value of vrfigure object

**Syntax**  
`get(vrfigure_object)`  
`x = get(vrfigure_object, 'property_name')`

**Arguments**

|                              |                            |
|------------------------------|----------------------------|
| <code>vrfigure_object</code> | Name of a vrfigure object. |
| <code>property_name</code>   | 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'<br>Default: 'off' | Determines whether antialiasing is used when rendering scene. Antialiasing smooths textures by interpolating values between texture points. Read/write. |
| CameraBound     | 'off'   'on'<br>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.  |



| <b>Property</b>    | <b>Value</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 vrfigure object. Read/write.                   |
| Headlight          | 'off'   'on'<br>Default: 'on' | Turns the headlight on or off. Read/write.   |

## vrfigure/get

| Property       | Value  | Description  |
|----------------|--|--|
| Lighting       | 'off'   'on'<br>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'   $32 \leq x \leq$<br>video card limit, where<br>$x$ is a power of 2 (video<br>card limit is typically<br>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. Virtual Reality Toolbox 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'  <br>'walk'<br>Default: 'examine'  | Specifies navigation mode. Read/write.   |

| Property                | Value  | Description  |
|-------------------------|--|--|
| NavPanel                | 'opaque'  <br>'translucent'  <br>'none'   'halfbar'  <br>'bar'<br>Default: 'halfbar' | Controls the appearance of the navigation panel in the Virtual Reality Toolbox viewer. Read/write.   |
| NavSpeed                | 'very slow'   'slow'<br>  'normal'   'fast'<br>  'very fast'<br>Default: 'normal'    | Specifies navigation speed. Read/write.  |
| NavZones                | 'off'   'on'<br>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'<br>Default: 'off'   | Enables 2-D offline animation file recording. Read/write.  |
| Record2DCompress Method | ' '   'auto'  <br>'lossless'  <br>'codec_code'<br>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 avifile. Read/write. |

## vrfigure/get

| <b>Property</b>             | <b>Value</b>                           | <b>Description</b>  |
|-----------------------------|--|---|
| Record2DCompress<br>Quality | 0–100<br>Default: '75'                 | Specifies the quality of 2-D animation file compression. Read/write.  |
| Record2DFileName            | String<br>Default:<br>'%f_anim_%n.ext' | Specifies the 2-D offline animation filename. 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'<br>Default: 'on'          | Toggles the status bar at the bottom of the Virtual Reality Toolbox viewer. Read/write.   |
| Textures                    | 'off'   'on'<br>Default: 'on'          | Turns texture rendering on or off. Read/write.  |
| Toolbar                     | 'off'   'on'<br>Default: 'on'          | Toggles toolbar on the Virtual Reality Toolbox viewer. Read/write.  |

| <b>Property</b> | <b>Value</b>  | <b>Description</b>  |
|-----------------|---|---|
| Transparency    | 'off'   'on'<br>Default: 'on'                                       | Specifies whether or not transparency information is taken into account when rendering. Read/write. |
| Viewpoint       | String<br>If active viewpoint does not have a name, value is empty. | Specifies the vrfigure object's active viewpoint. Read/write.                                       |
| Wireframe       | 'off'   'on'<br>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.   |

## Example

Create a vrworld object:

```
myworld = vrworld('vrmount.wr1');
```

The vrworld object myworld is associated with the virtual world vrmount.wr1.  
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 -1 3.61999e-006]  
CameraPosition = [0 0 0]  
CameraPositionAbs = [0 150 0]  
CameraUpVector = [0 1 0]  
CameraUpVectorAbs = [0 -3.61999e-006 -1]  
DeleteFcn = ''  
Headlight = 'off'  
Lighting = 'on'  
Name = 'Planets'  
NavMode = 'fly'  
NavPanel = 'halfbar'  
NavSpeed = 'normal'  
NavZones = 'off'  
Position = [9 91 512 391]  
Record2D = 'off'  
Record2DCompressMethod = 'auto'  
Record2DCompressQuality = 75  
Record2DFileName = '%f_anim_%n.avi'  
StatusBar = 'on'  
Textures = 'on'  
Toolbar = 'on'
```

```
Transparency = 'on'  
Viewpoint = 'View from top'  
Wireframe = 'off'  
World = vrworld object: 1-by-1  
ZoomFactor = 1
```

**See Also**

vrfigure, vrfigure/set

# vrfigure/isvalid

---

|                    |  |
|--------------------|--|
| <b>Purpose</b>     | Return 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>vrworld/isvalid</code> , <code>vrnode/isvalid</code>   |



**Purpose** Change property value of vrfigure object

**Syntax** `set(vrfigure_object, 'property_name', property_value)`

**Arguments**

|                              |                                       |
|------------------------------|---------------------------------------|
| <code>vrfigure_object</code> | Name of a vrfigure object.            |
| <code>property_name</code>   | Name of the property you want to set. |
| <code>property_value</code>  | 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'<br>Default: 'off' | Determines whether antialiasing is used when rendering scene. Antialiasing smooths textures by interpolating values between texture points. Read/write. |
| CameraBound     | 'off'   'on'<br>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.  |

## vrfigure/set

| <b>Property</b>    | <b>Value</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 vrfigure object. Read/write.                   |
| Headlight          | 'off'   'on'<br>Default: 'on' | Turns the headlight on or off. Read/write.   |

| Property       | Value  | Description  |
|----------------|--|--|
| Lighting       | 'off'   'on'<br>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'   $32 \leq x \leq$<br>video card limit, where<br>x is a power of 2 (video<br>card limit is typically<br>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. Virtual Reality Toolbox 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'  <br>'walk'<br>Default: 'examine'  | Specifies navigation mode. Read/write.   |

## vrfigure/set

| Property                | Value  | Description  |
|-------------------------|--|--|
| NavPanel                | 'opaque'  <br>'translucent'  <br>'none'   'halfbar'  <br>'bar'<br>Default: 'halfbar' | Controls the appearance of the navigation panel in the Virtual Reality Toolbox viewer. Read/write.   |
| NavSpeed                | 'very slow'   'slow'<br>  'normal'   'fast'<br>  'very fast'<br>Default: 'normal'    | Specifies navigation speed. Read/write.  |
| NavZones                | 'off'   'on'<br>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'<br>Default: 'off'   | Enables 2-D offline animation file recording. Read/write.  |
| Record2DCompress Method | ' '   'auto'  <br>'lossless'  <br>'codec_code'<br>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 avifile. Read/write. |

| <b>Property</b>             | <b>Value</b>                           | <b>Description</b>  |
|-----------------------------|--|---|
| Record2DCompress<br>Quality | 0–100<br>Default: '75'                 | Specifies the quality of 2-D animation file compression. Read/write.  |
| Record2DFileName            | String<br>Default:<br>'%f_anim_%n.ext' | Specifies the 2-D offline animation filename. 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'<br>Default: 'on'          | Toggles the status bar at the bottom of the Virtual Reality Toolbox viewer. Read/write.   |
| Textures                    | 'off'   'on'<br>Default: 'on'          | Turns texture rendering on or off. Read/write.  |
| Toolbar                     | 'off'   'on'<br>Default: 'on'          | Toggles toolbar on the Virtual Reality Toolbox viewer. Read/write.  |

## vrfigure/set

| <b>Property</b> | <b>Value</b>  | <b>Description</b>  |
|-----------------|---|---|
| Transparency    | 'off'   'on'<br>Default: 'on'                                       | Specifies whether or not transparency information is taken into account when rendering. Read/write. |
| Viewpoint       | String<br>If active viewpoint does not have a name, value is empty. | Specifies the vrfigure object's active viewpoint. Read/write.                                       |
| Wireframe       | 'off'   'on'<br>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 Virtual Reality Toolbox 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`

## vrfigure/vrgcf

---

**Purpose** Get handle for currently 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, MATLAB creates one and returns its handle.

This method is most useful to query and set virtual reality figure properties.

**See Also** `vrfigure`, `vrfigure/get`, `vrfigure/set`



**Purpose** Get current callback vrfigure object

**Syntax** `f = vrgcbf`

`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.

# vrgetpref

---

**Purpose** Read values of Virtual Reality Toolbox 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 Virtual Reality Toolbox 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 <code>Bool</code> data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are <code>'logical'</code> and <code>'char'</code> . If set to <code>'logical'</code> , the VRML <code>Bool</code> data type is returned as a logical value. If set to <code>'char'</code> , the <code>Bool</code> data type is returned <code>'on'</code> or <code>'off'</code> . Default is <code>'logical'</code> .   |
| <code>DataTypeInt32</code>             | Specifies handling of the VRML <code>Int32</code> data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are <code>'int32'</code> and <code>'double'</code> . If set to <code>'int32'</code> , the VRML <code>Int32</code> data type is returned as <code>int32</code> . If set to <code>'double'</code> , the <code>Int32</code> data type is returned as <code>'double'</code> . Default is <code>'double'</code> .  |
| <code>DataTypeFloat</code>             | Specifies the handling of the VRML <code>float</code> data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are <code>'single'</code> and <code>'double'</code> . If set to <code>'single'</code> , the VRML <code>Float</code> and <code>Color</code> data types are returned as <code>'single'</code> . If set to <code>'double'</code> , the <code>Float</code> and <code>Color</code> data types are returned as <code>'double'</code> . Default is <code>'double'</code> . |
| <code>DefaultFigureAntiAliasing</code> | Determines whether antialiasing is used by default for new <code>vrfigure</code> objects. Valid values are <code>'off'</code> and <code>'on'</code> .  |
| <code>DefaultFigureDeleteFcn</code>    | 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 vrfigure objects. Valid values are 'off' and 'on'.   |
| DefaultFigureMaxTextureSize          | Specifies the default maximum size of a texture used in rendering new vrfigure objects. Valid values are 'auto' and $32 \leq x \leq$ video card limit, where $x$ is a power of 2. |
| DefaultFigureNavPanel                | Specifies the default appearance of the control panel in the viewer. Valid values are 'opaque', 'translucent', 'none', 'halfbar', 'bar', and 'factory'. Default is 'halfbar'.     |
| DefaultFigureNavZones                | Specifies whether the navigation zone is on or off by default for new vrfigure objects. Valid values are 'off' and 'on'.  |
| DefaultFigurePosition                | Sets the default initial position and size of the Virtual Reality Toolbox viewer window. Valid value is a vector of four doubles.   |
| DefaultFigureRecord2DCompressMethod  | Specifies the default compression method for creating 2-D animation files for new vrfigure objects. Valid values are '', 'auto', 'lossless', and 'codec_code'.                    |
| DefaultFigureRecord2DCompressQuality | Specifies the default quality of 2-D animation file compression for new vrfigure objects. Valid values are 0–100.   |
| DefaultFigureRecord2DFileName        | Specifies the default 2-D offline animation filename for new vrfigure objects.  |
| DefaultFigureStatusBar               | Specifies whether the status bar appears by default at the bottom of the Virtual Reality Toolbox viewer for new vrfigure objects. Valid values are 'off' and 'on'.                |

| Preference                    | Description   |
|-------------------------------|---|
| DefaultFigureToolBar          | Specifies whether the toolbar appears by default on the Virtual Reality Toolbox viewer for new vrfigure objects. Valid values are 'off' and 'on'.   |
| DefaultFigure Transparency    | Specifies whether or not transparency information is taken into account when rendering for new vrfigure objects. Valid values are 'off' and 'on'.   |
| DefaultFigureWireframe        | Specifies whether objects are drawn as solids or wireframes by default for new vrfigure objects. Valid values are 'off' and 'on'.   |
| DefaultViewer                 | Specifies which viewer is used to view a virtual scene. The Virtual Reality Toolbox viewer is used when the preference is set to 'internal'. The Web browser is used when this preference is set to 'web'. Default is 'internal'. |
| DefaultWorldRecord3D FileName | Specifies the default 3-D animation filename for new vrworld objects.   |
| DefaultWorldRecordMode        | Specifies the default animation recording mode for new vrworld objects. Valid values are 'manual' and 'scheduled'.  |
| DefaultWorldRecord Interval   | Specifies the default start and stop times for scheduled animation recording for new vrworld objects. Valid value is a vector of two doubles.   |
| DefaultWorldRemoteView        | Specifies whether the virtual world is enabled by default for remote viewing for new vrworld objects. Valid values are 'off' and 'on'.  |

| Preference             | Description   |
|------------------------|---|
| DefaultWorldTimeSource | Specifies the default source of the time for new vrworld objects. Valid values are 'external' and 'freerun'.  |
| Editor                 | Path to the VRML editor. If this path is empty, the MATLAB editor is used.  |
| HttpPort               | IP port number used to access the VR server over the Web via HTTP. If you change this preference, you must restart MATLAB before the change takes effect.           |
| TransportBuffer        | Length of the transport buffer (network packet overlay) for communication between the VR server and its clients.  |
| TransportTimeout       | Amount of time the VR Toolbox server waits for a reply from the client. If there is no response from the client, the VR Toolbox server disconnects from the client. |
| VrPort                 | IP port used for communication between the VR server and its clients. If you change this preference, you must restart MATLAB before the change takes effect.        |

Note that the `HttpPort`, `VrPort`, and `TransportBuffer` preferences affect Web-based viewing of virtual worlds. `DefaultFigurePosition` and `DefaultNavPanel` affect the Virtual Reality Toolbox viewer.

`DefaultFigureNavPanel` — Controls the appearance of the navigation panel in the Virtual Reality Toolbox viewer. For example, setting this value to 'translucent' causes the navigation panel to appear translucent.

`DefaultViewer` — Determines whether the virtual scene appears in the Virtual Reality Toolbox viewer or in your Web browser. If the preference is set to 'internal', the Virtual Reality Toolbox viewer is the default viewer. If it is set to 'web', the default Web browser with the VRML plug-in is the default viewer.

**Editor** — Contains a path to the VRML editor executable file. When you use the `edit` command, the Virtual Reality Toolbox runs the VRML editor executable with all parameters required to edit the VRML file.

When you run the editor, the Virtual Reality Toolbox 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 directory |
| <code>%file</code>       | Refers to the VRML filename         |

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.

## vrgetpref

---

VrPort — Specifies the network port to use for communication between the Virtual Reality Toolbox 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** 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(parent_node, 'parent_field', 'node_name',
'node_type')
```

**Arguments**

|                             |  |
|-----------------------------|--|
| <code>vrworld_object</code> | Name of a vrworld object representing a virtual world. |
| <code>node_name</code>      | Name of the node.                                      |
| <code>node_type</code>      | Type of the node.                                      |
| <code>parent_node</code>    | Name of the parent node that is a vrnode object.       |
| <code>parent_field</code>   | Name of the field of the parent node.                  |

**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(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.

A vrnode object identifies a virtual world node in a way very similar to a handle. If the vrnode method is applied to a node that does not exist, the node is created, the vrnode object is created, and the handle to the vrnode object is returned. If the vrnode method is applied to an existing node, the handle to the vrnode object associated with this node is returned.

# vrnode

---

## **See Also**

vrworld, vrnode/get, vrnode/set, vrnode/getfield, vrnode/setfield,  
vrnode/delete

**Purpose** Delete vrnode object

**Syntax** `delete(vrnode_object)`  
`delete(n)`

**Arguments** `vrnode_object` Name of a vrnode object.

**Description** `delete(vrnode_object)` deletes the virtual world node.  
`delete(n)` deletes the vrnode object referenced by the vrnode handle n. If n is a vector of vrnode handles, multiple nodes are deleted.  
As soon as a node is deleted, it and all its child objects are removed from all clients connected to the virtual world.

**See Also** `vrworld/delete`

# vrnode/fields

---

**Purpose** Return 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` on page 8-47.

**See Also** `vrnode/get`, `vrnode/set`

**Purpose** Read property value of vrnode object

**Syntax**

```
get(vrnode_object)
x = get(vrnode_object)
x = get(vrnode_object, 'property_name')
```

**Arguments**

|                            |  |
|----------------------------|--|
| <code>vrnode_object</code> | Name of a vrnode object representing the node to be queried. |
| <code>property_name</code> | Name of the property to be read.                             |

**Description** `get(vrnode_object)` lists all vrnode properties in the MATLAB Command Window.

`x = get(vrnode_object)`, where `vrnode_object` is a scalar, returns a structure where each field name is the name of a property and each field contains the value of that property.

`x = get(vrnode_object, 'property_name')` returns the value of given property.

If `vrnode_object` is a vector of vrnode handles, `get` returns an M-by-1 cell array of values, where M is equal to `length(vrnode_object)`.

The vrnode property values are case sensitive. Property names are not case sensitive.

The vrnode object properties allow you to control the behavior 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.  |
| 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. |

## vrnode/get

---

### **See Also**

`vrnode/set`, `vrnode/getfield`, `vrnode/setfield`, `vrnode`

**Purpose** Get 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/get`, `vrnode/set`, `vrnode/setfield`, `vrnode`

# vrnode/isvalid

---

**Purpose** Return 1 if vrnode object is valid, 0 if not

**Syntax** `x = isvalid(vrnode_object_vector)`

**Arguments** `vrnode_object_vector` Name of an array of vrnode objects to be queried.

**Description** This method returns an array that contains 1 when the elements of `vrnode_object_vector` are valid vrnode objects, and 0 when they are not.

The vrnode object is considered valid if the following conditions are met:

- The parent world of the node exists.
- The parent world of the node is open.
- The VRML node with the given vrnode handle exists in the parent world.

**See Also** `vrworld/isvalid`, `vrfigure/isvalid`



**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.

**See Also** `vrnode/get`, `vrnode/getfield`, `vrnode/setfield`, `vrnode`

# 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.

---

**See Also** `vrnode/get`, `vrnode/set`, `vrnode/getfield`, `vrnode`

|                            |   |                            |  |                         |  |                     |  |
|----------------------------|---|----------------------------|--|-------------------------|--|---------------------|--|
| <b>Purpose</b>             | Enable or disable synchronization of VRML fields with clients   |                            |  |                         |  |                     |  |
| <b>Syntax</b>              | <code>sync(vrnode_object, 'field_name', 'action')</code>  |                            |  |                         |  |                     |  |
| <b>Arguments</b>           | <table><tr><td><code>vrnode_object</code></td><td>Name of a vrnode object representing the node.</td></tr><tr><td><code>field_name</code></td><td>Name of the VRML field to be synchronized.</td></tr><tr><td><code>action</code></td><td>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></td></tr></table>   | <code>vrnode_object</code> | Name of a vrnode 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> |
| <code>vrnode_object</code> | Name of a vrnode 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>  |                            |  |                         |  |                     |  |
| <b>Description</b>         | <p>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.</p> <p>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 Simulink.</p> <p>Synchronization is meaningful only for readable fields. Readable fields are of VRML data class <code>eventOut</code> and <code>exposedField</code>. You cannot enable synchronization for <code>eventIn</code> or <code>nonexposed</code> fields.</p> |                            |  |                         |  |                     |  |
| <b>See Also</b>            | <code>vrnode/get</code> , <code>vrnode</code>   |                            |  |                         |  |                     |  |

# vrinstall

---

**Purpose** Install and check Virtual Reality Toolbox 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 Virtual Reality Toolbox. Currently there are two such components: VRML plug-in and VRML editor.

| Action Value | Description  |
|--------------|--|
| -selftest    | Checks the integrity of the Virtual Reality Toolbox. If this function reports an error, you should reinstall the Virtual Reality Toolbox. 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. |

| 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   | <p>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 directory. It removes the editor registry information.</p> <p>If you want to uninstall the VRML plug-in, exit MATLAB and, from the <b>Control Panel</b> window, select <b>Add/Remove Programs</b>.</p> |

## 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
```

# vrlib

---

**Purpose** Open Simulink block library for the Virtual Reality Toolbox

**Syntax** `vrlib`

**Description** The Simulink library for the Virtual Reality Toolbox has six blocks: VR Sink, VR Placeholder, VR Signal Expander, Joystick Input, and Magellan SpaceMouse.

Alternatively, you can access these blocks from a Simulink block diagram. In the Simulink window, from the **View** menu, click **Show Library Browser**.

**Purpose** Change Virtual Reality Toolbox preferences

**Syntax** `vrsetpref('preference_name', 'preference_value')`  
`vrsetpref('factory')`

**Arguments**

|                         |                              |
|-------------------------|------------------------------|
| <i>preference_name</i>  | Name of the preference.      |
| <i>preference_value</i> | New value of the preference. |

**Description** This function sets the given Virtual Reality Toolbox 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  |
|-----------------------------|--|
| DataTypeFloat               | 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'. |
| DefaultFigureAntiAliasing   | Determines whether antialiasing is used by default for new <code>vrfigure</code> objects. Valid values are 'off' and 'on'.   |
| 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. Valid values are 'off' and 'on'.   |
| DefaultFigureMaxTextureSize | Specifies the default maximum size of a texture used in rendering new <code>vrfigure</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. Valid values are 'off' and 'on'.  |
| DefaultFigurePosition       | Sets the default initial position and size of the Virtual Reality Toolbox viewer window. Valid value is a vector of four doubles.  |



| Preference                               | Description   |
|--|---|
| DefaultFigureRecord2D<br>CompressMethod  | Specifies the default compression method for creating 2-D animation files for new vrfigure objects. Valid values are '', 'auto', 'lossless', and 'codec_code'.  |
| DefaultFigureRecord2D<br>CompressQuality | Specifies the default quality of 2-D animation file compression for new vrfigure objects. Valid values are 0–100.   |
| DefaultFigureRecord2D<br>FileName        | Specifies the default 2-D offline animation filename for new vrfigure objects.  |
| DefaultFigureStatusBar                   | Specifies whether the status bar appears by default at the bottom of the Virtual Reality Toolbox viewer for new vrfigure objects. Valid values are 'off' and 'on'.  |
| DefaultFigureToolBar                     | Specifies whether the toolbar appears by default on the Virtual Reality Toolbox viewer for new vrfigure objects. Valid values are 'off' and 'on'.   |
| DefaultFigure<br>Transparency            | Specifies whether or not transparency information is taken into account when rendering for new vrfigure objects. Valid values are 'off' and 'on'.   |
| DefaultFigureWireframe                   | Specifies whether objects are drawn as solids or wireframes by default for new vrfigure objects. Valid values are 'off' and 'on'.   |
| DefaultViewer                            | Specifies which viewer is used to view a virtual scene. The Virtual Reality Toolbox viewer is used when the preference is set to 'internal'. The Web browser is used when this preference is set to 'web'. Default is 'internal'. |

## vrsetpref

| <b>Preference</b>                | <b>Description</b>  |
|----------------------------------|---|
| DefaultWorldRecord3D<br>FileName | Specifies the default 3-D animation filename for new vrworld objects.   |
| DefaultWorldRecordMode           | Specifies the default animation recording mode for new vrworld objects. Valid values are 'manual' and 'scheduled'.  |
| DefaultWorldRecord<br>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.  |
| HttpPort                         | IP port number used to access the VR server over the Web via HTTP. If you change this preference, you must restart MATLAB before the change takes effect. |
| TransportBuffer                  | Length of the transport buffer (network packet overlay) for communication between the VR server and its clients.  |

| <b>Preference</b> | <b>Description</b>  |
|-------------------|---|
| TransportTimeout  | Amount of time the VR Toolbox server waits for a reply from the client. If there is no response from the client, the VR Toolbox server disconnects from the client. |
| VrPort            | IP port used for communication between the VR server and its clients. If you change this preference, you must restart MATLAB before the change takes effect.        |

Changes to the HttpPort or VrPort preferences take effect only after you restart MATLAB.

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.

**See Also**

vrgetpref

# vrview

---

**Purpose** View virtual world using Virtual Reality Toolbox 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 Virtual Reality Toolbox 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 Virtual Reality Toolbox 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 Virtual Reality Toolbox 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.

**See Also** `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 Simulink.

**See Also** `vrwhos`, `vrworld`, `vrclear`

# vrwhos

---

|                    |  |
|--------------------|--|
| <b>Purpose</b>     | List details about virtual worlds in memory  |
| <b>Syntax</b>      | <code>vrwhos</code>  |
| <b>Description</b> | <code>vrwhos</code> displays a list of virtual worlds currently in memory, with a description, in the MATLAB Command Window. The relation between <code>vrwho</code> and <code>vrwhos</code> is similar to the relation between <code>who</code> and <code>whos</code> . |
| <b>See Also</b>    | <code>vrwho</code> , <code>vrclear</code>  |

---

|                    |  |          |   |
|--------------------|--|----------|---|
| <b>Purpose</b>     | Create new vrworld object associated with virtual world  |          |   |
| <b>Syntax</b>      | <pre>myworld = vrworld('filename') myworld = vrworld myworld = vrworld([])</pre>   |          |   |
| <b>Arguments</b>   | <table><tr><td>filename</td><td>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.</td></tr></table>  | 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. |
| 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.  |          |   |
| <b>Description</b> | <p><code>myworld = vrworld('filename')</code> creates a virtual world associated with the VRML file <code>filename</code> and returns its handle. If the virtual world already exists, a handle to the existing virtual world is returned.</p> <p><code>myworld = vrworld</code> creates an empty vrworld handle that does not refer to any virtual world.</p> <p><code>myworld = vrworld([])</code> returns an empty array of vrworld handles.</p> <p>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.</p> <p>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.</p> <p>The vrworld object associated with a virtual world remains valid until you use either <code>delete</code> or <code>vrclear</code>.</p> |          |   |
| <b>Examples</b>    | <pre>myworld = vrworld('vrpend.wrl')</pre>   |          |   |
| <b>See Also</b>    | <code>vrworld/open</code> , <code>vrworld/delete</code> , <code>vrworld/close</code>   |          |   |

# vrworld/close

---

|                  |   |
|------------------|---|
| <b>Purpose</b>   | Close virtual world   |
| <b>Syntax</b>    | <code>close(vrworld_object)</code>  |
| <b>Arguments</b> | <code>vrworld_object</code> A <code>vrworld</code> 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/open`, `vrworld/delete`



|                    |  |
|--------------------|--|
| <b>Purpose</b>     | Delete virtual world from memory   |
| <b>Syntax</b>      | <code>delete(vrworld_object)</code>  |
| <b>Arguments</b>   | <code>vrworld_object</code> A <code>vrworld</code> object representing a virtual world.  |
| <b>Description</b> | <p>The <code>delete</code> method removes from memory the virtual world associated with a <code>vrworld</code> object. The virtual world must be closed before you can delete it.</p> <p>Deleting a virtual world frees the virtual world from memory and invalidates all existing <code>vrworld</code> objects associated with the virtual world.</p> <p>If <code>vrworld_object</code> is a vector of <code>vrworld</code> objects, all associated virtual worlds are deleted.</p> <p>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.</p> |
| <b>See Also</b>    | <code>vrworld/close</code> , <code>vrclear</code>  |

# 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 MATLAB, the changes appear after you reload the virtual world.
- Use the `save` method in MATLAB to replace the modified VRML file. Any changes you made in the editor are lost.

**See Also** `vrworld/reload`, `vrworld/save`

**Purpose** Read property value of vrworld object

**Syntax**

```
get(vrworld_object)
x = get(vrworld_object)
x = get(vrworld_object, 'property_name')
```

**Arguments**

|                             |  |
|-----------------------------|--|
| <code>vrworld_object</code> | A vrworld object representing a virtual world. |
| <code>property_name</code>  | 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'<br>Default: 'on' | Client cannot or can update the virtual scene. Read/write.        |

| Property         | Value  | Description   |
|------------------|--|---|
| Description      | String<br>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 Virtual Reality Toolbox 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'<br>Default: 'off'   | Indicates a closed or open virtual world. Read only.  |
| Record3D         | 'off'   'on'<br>Default: 'off'   | Enables 3-D animation recording. Read/write.  |
| Record3DFileName | String<br>Default:<br>'%f_anim_%n.wrl'                                   | 3-D animation filename. 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. |

| Property       | Value                                       | Description   |
|----------------|---|---|
| Recording      | 'off'   'on'<br>Default: 'off'              | Animation recording toggle. This property acts as the master recording switch. Read/write.  |
| RecordMode     | 'manual'   'scheduled'<br>Default: 'manual' | Animation recording mode. Read/write.   |
| RecordInterval | Vector of two doubles<br>Default: [0 0]     | Start and stop times for scheduled animation recording. Corresponds to the virtual world object Time property. Read/write.            |
| RemoteView     | 'off'   'on'<br>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.  |

| Property   | Value  | Description  |
|------------|--|--|
| TimeSource | 'external'  <br>'freerun'<br><br>Default: 'external' | Source of the time for the virtual world. If set to 'external', time in the scene is controlled from MATLAB (by setting the Time property) or Simulink (simulation time). If set to 'freerun', time in the scene advances independently based on the system timer. Read/write. |
| View       | 'off'   'on'<br>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 **WorldInfo** node with a title property (see the VRML reference), 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/set](#), [vrworld](#)

# vrworld/invalid

---

**Purpose** Return 1 if vrworld object is valid, 0 if not

**Syntax** `x = invalid(vrworld_object)`

**Arguments** `vrworld_object` A vrworld object representing a virtual world.

**Description** A vrworld object is considered valid if its associated virtual world still exists.

`x = invalid(vrworld_object)` returns an array that contains a 1 when the elements of `vrworld_object` are valid vrworld objects, and returns a 0 when they are not.

You use this method to check whether the vrworld object is still valid. Using a `delete` or `vrclear` command can make a vrworld object invalid.

**See Also** `vrfigure/invalid`, `vrnode/invalid`



**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>        | 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`

# 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.wrl')
myworld2 = vrworld('vrpend.wrl')
```

Next, create an array of virtual world handles by typing

```
myworlds = [myworld1 myworld2];
```

`open(myworlds)` opens both of these virtual worlds.

**See Also** `vrworld`, `vrworld/close`

**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/save`, `vrworld/open`

# vrworld/save

---

**Purpose** Write virtual world to VRML file

**Syntax** `save(vrworld_object, 'vrml_file')`

**Arguments**

|                             |   |
|-----------------------------|---|
| <code>vrworld_object</code> | A vrworld 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/reload`, `vrworld/open`

**Purpose** Change property values of vrworld object

**Syntax** `set(vrworld_object, 'property_name', property_value)`

**Arguments**

|                             |  |
|-----------------------------|--|
| <code>vrworld_object</code> | Name of a vrworld object representing a virtual world. |
| <code>property_name</code>  | Name of the property.                                  |
| <code>property_value</code> | New value of the property.                             |

**Description** You can change the values of the read/write virtual world properties. The following are properties of vrworld objects. Names are not case sensitive.

| Property      | Value  | Description   |
|---------------|--|---|
| Clients       | Scalar   | Number of clients currently viewing the virtual world. Read only.   |
| ClientUpdates | 'off'   'on'<br>Default: 'on'  | Client cannot or can update the virtual scene. Read/write.  |
| Description   | String<br>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 Virtual Reality Toolbox viewer windows currently viewing the virtual world. Read only. |
| FileName      | String   | Name of the associated VRML file. Read only.  |

| <b>Property</b>  | <b>Value</b>                                | <b>Description</b>  |
|------------------|---|---|
| Nodes            | Vector of vrnode objects                    | Vector of vrnode objects for all named nodes in the virtual world. Read only.   |
| Open             | 'off'   'on'<br>Default: 'off'              | Indicates a closed or open virtual world. Read only.  |
| Record3D         | 'off'   'on'<br>Default: 'off'              | Enables 3-D animation recording. Read/write.  |
| Record3DFileName | String<br>Default:<br>'%f_anim_%n.wrl'      | 3-D animation filename. 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'<br>Default: 'off'              | Animation recording toggle. This property acts as the master recording switch. Read/write.  |
| RecordMode       | 'manual'   'scheduled'<br>Default: 'manual' | Animation recording mode. Read/write.   |

| Property       | Value   | Description  |
|----------------|---|--|
| RecordInterval | Vector of two doubles<br>Default: [0 0]       | Start and stop times for scheduled animation recording. Corresponds to the virtual world object Time property. Read/write.   |
| RemoteView     | 'off'   'on'<br>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'<br>Default: 'external' | Source of the time for the virtual world. If set to 'external', time in the scene is controlled from MATLAB (by setting the Time property) or Simulink (simulation time). If set to 'freerun', time in the scene advances independently based on the system timer. Read/write. |
| View           | 'off'   'on'<br>Default: 'on'                 | Indicates an unviewable or viewable virtual world. Read/write.   |

**See Also**

vrworld/get, vrworld

# vrworld/view

---

**Purpose** View virtual world

**Syntax**

```
view(vrworld_object)
x = view(vrworld_object)
x = view(vrworld_object, '-internal')
x = view(vrworld_object, '-web')
```

**Arguments** vrworld\_object            A vrworld object representing a virtual world.

**Description** The view method opens the default VRML viewer on the host computer and loads the virtual world associated with the vrworld object into the viewer window. You specify the default VRML viewer using the DefaultViewer preference. The virtual world must be open for you to use this method.

x = view(vrworld\_object) opens the default VRML viewer on the host computer and loads the virtual world associated with the vrworld object into the viewer window. If the Virtual Reality Toolbox viewer is used, view also returns the vrfigure handle of the viewer window. If a Web browser is used, view returns an empty array of vrfigure handles.

x = view(vrworld\_object, '-internal') opens the virtual world in the Virtual Reality Toolbox viewer.

x = view(vrworld\_object, '-web') opens the virtual world in the Web browser.

If the virtual world is disabled for viewing (that is, the View property for the associated vrworld object is set to 'off'), the view method does nothing.

**Examples**

```
myworld = vrworld('vrpend.wrl')
open(myworld)
view(myworld)
```

**See Also** vrworld, vrview



|  |   |
|--|---|
| <b>simulation</b>                        | The process of running a dynamic system in nonreal time to observe its behavior.    |
| <b>Virtual Reality Modeling Language</b> | The specification for displaying three-dimensional objects using a VRML viewer.     |
| <b>virtual figure object</b>             | A handle to a Virtual Reality Toolbox viewer window.                                |
| <b>virtual node object</b>               | A handle to a node in a virtual world that allows access to the node's properties.  |
| <b>virtual world</b>                     | An imaginary world where you can navigate around objects in three dimensions.       |
| <b>virtual world object</b>              | A handle to a virtual world that allows you to interact with and control the world. |
| <b>VRML</b>                              | Virtual Reality Modeling Language. See "VRML Overview" on page 1-10 of this guide.  |



## Numerics

### 2-D AVI files

- recording through MATLAB interface 4-11
- recording through Virtual Reality Toolbox viewer 6-18

### 3-D VRML files

- recording with MATLAB interface 4-10
- recording with Virtual Reality Toolbox viewer 6-18

## A

### adding

- Virtual Reality Toolbox blocks 3-2

### animation files

- recording with MATLAB interface 4-10
- recording with Virtual Reality Toolbox viewer 6-17

### associating virtual worlds with Simulink blocks 3-10

## B

### bitmap file formats 1-28

### blaxxun Contact

- creating virtual worlds 5-8
- installing 2-20
- known issue 2-22
- VRML viewer 6-44

### bmp file formats 1-28

### bouncing ball

- Simulink example 1-17

## C

### car

- MATLAB interface example 1-24

### changing virtual world associated with Simulink block 3-10

### client computer

- installation of VRML viewer (Windows) 2-38
- system requirements 2-10

### closing virtual worlds 4-8

### components

- client computer 2-38
- host computer 2-13

### connecting

- Simulink model to a virtual world 5-16

### coordinate system

- MATLAB 1-11
- VRML 1-11

### creating vrworld object 4-2

## D

### default editor

- setting 2-30

### default viewer

- setting 2-24

### deformation of a sphere example

- adding Virtual Reality Toolbox blocks 5-6
- connecting Simulink to a virtual world 5-16
- creating a box in a virtual world 5-13
- creating a sphere in a virtual world 5-8
- defining the problem 5-5

### deleting virtual worlds 4-8

### displaying virtual worlds 3-12

**E**

## editors

- general 3-D 5-2
- native VRML 5-2
- uninstalling 2-36

## examples

- bouncing ball 1-17
- car 1-24
- deformation of a sphere 5-5
- heat transfer 1-24
- inverted pendulum 1-22
- lighting 1-19
- magnetic levitation 1-19
- magnetic levitation for Real-Time Windows Target 1-20
- manipulator with SpaceMouse 1-20
- MATLAB interface 1-16
- plane taking off 1-23
- rotating membrane 1-26
- Simulink interface 1-16
- solar system 1-23
- using MATLAB interface 1-24

**F**

## file format

- VRML 1-13

## files

- textures 1-28

## functions

- MATLAB interface 8-1
- vrclear 8-6
- vrgetpref 8-30
- vrinstall 8-48
- vrlib 8-50
- vrsetpref 8-51
- vrview 8-56

vrwho 8-57

vrwhos 8-58

**H**

## heat transfer

- MATLAB example 1-24

## history

- VRML 1-10

## host computer

- installing Virtual Reality Toolbox 2-12
- installing VRML editor (Windows) 2-29
- installing VRML viewer (UNIX) 2-23
- installing VRML viewer (Windows) 2-20
- required components 2-13
- system requirements 2-7
- Virtual Reality Toolbox viewer 2-19
- VRML editor (UNIX) 2-30

**I**

## installation

- blaxxun Contact 2-20
- client computer 2-38
- components
  - host computer 2-13
- host computer 2-12
- supported platforms 2-6
- system requirements 2-6
- testing 2-39
- viewer on host computer 2-19
- Virtual Reality Toolbox 2-12
- VRML editor (UNIX) 2-30
- VRML editor (Windows) 2-29
- VRML viewer (UNIX) 2-23
- VRML viewer (Windows) 2-20
- interacting with a virtual world 4-5

interface overview 3-2

inverted pendulum

    Simulink example 1-22

## J

Joystick Input

    Simulink block 7-4

## L

license

    getting or updating 2-12

lighting

    Simulink example 1-19

## M

Magellan SpaceMouse

    Simulink block 7-6

SpaceMouse

*See also* Magellan SpaceMouse

magnetic levitation

    Simulink example 1-19

    Simulink example for Real-Time Windows

        Target 1-20

manipulator with Space Mouse

    Simulink example 1-20

MATLAB

    coordinate system 1-11

    interface examples 1-24

MATLAB interface

    creating a vrworld object 4-2

    interacting with a virtual world 4-5

    opening a virtual world 4-3

    table of general functions 8-1

## N

native VRML 5-2

navigation

    about a virtual scene 6-10

    example of navigation 6-14

    keyboard 6-16

    using the mouse 6-10

navigation speed

    changing 6-14

network security setting

    changing default 2-22

*See also* blaxxun Contact

## O

opening a viewer window 3-15

opening virtual worlds 4-3

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 1-23

platforms

    supported 2-6

**R**

- rendering of a virtual scene 6-35
- rotating membrane
  - Simulink example 1-22
  - Virtual Reality Toolbox example 1-26
- running Simulink example 2-39

**S**

- security settings
  - changing 2-22
- server
  - Virtual Reality Toolbox 1-29
- setting
  - default editor 2-30
  - default viewer 2-24
- simulation
  - displaying virtual worlds 3-12
  - starting 3-12
- Simulink
  - associating with virtual worlds 3-2
  - interface examples 1-16
  - See also* examples
- Simulink blocks
  - adding Virtual Reality Toolbox blocks 3-2
  - changing virtual world association 3-10
  - VR Placeholder 7-8
  - VR Signal Expander 7-9
  - VR Sink 7-4

## Simulink interface examples

- bouncing ball 1-17
- deformation of a sphere 5-6
- inverted pendulum 1-22
- lighting 1-19
- magnetic levitation 1-19
- magnetic levitation with Real-Time Windows Target 1-20
- manipulator with SpaceMouse 1-20
- plane taking off 1-23
- rotating membrane 1-22
- running and viewing 2-39
- solar system 1-23
- vehicle dynamics visualization 1-22

## Simulink interface overview 3-2

- solar system
  - Simulink example 1-23
- SpaceMouse
  - Simulink examples 1-20
- supported platforms 2-6
- system requirements
  - client computer 2-10
  - host computer 2-7

**T**

- testing
  - installation 2-39
  - MATLAB example 2-44
  - Simulink example 2-39
- textures 1-28

**U**

- uninstalling
  - editor 2-36
  - Virtual Reality Toolbox 2-36
  - V-Realm Builder 2-36
  - VRML viewer (Windows) 2-36

**V**

- vehicle visualization
  - Simulink example 1-22
- view a virtual world
  - using a Web browser on the client computer 3-19
  - using a Web browser on the host computer 3-15
- viewer
  - installation on client computer 2-38
  - installation on host computer 2-19
  - opening 3-15
- viewpoint control 6-28
- Virtual Reality Toolbox
  - description 1-2
  - features 1-4
- Virtual Reality Toolbox blocks
  - VR Placeholder 7-8
  - VR Signal Expander 7-9
  - VR Sink 7-4
- Virtual Reality Toolbox examples
  - car 1-24
  - heat transfer 1-24
  - rotating membrane 1-26
  - running and viewing 2-44
- Virtual Reality Toolbox preferences
  - vrsetpref function 8-51

- Virtual Reality Toolbox viewer
  - changing navigation speed 6-14
  - introduction 6-2
  - navigation 6-10
  - rendering 6-35
  - viewpoint control 6-28
- virtual worlds
  - associating with Simulink 3-2
  - closing 4-8
  - deleting 4-8
  - displaying 3-12
  - interacting with 4-5
  - opening 4-3
  - overview 5-2
- VR Placeholder
  - Simulink block 7-8
- VR Signal Expander
  - Simulink block 7-9
- VR Sink
  - Simulink block 7-4
- vrclear
  - Virtual Reality Toolbox function 8-6
- V-Realm Builder
  - installing 2-29
  - uninstalling 2-36
  - VRML editor 5-4
- vrgetpref
  - Virtual Reality Toolbox function 8-30
- vrinstall
  - Virtual Reality Toolbox function 8-48
- vrlib
  - Virtual Reality Toolbox function 8-50
- VRML
  - coordinate system 1-11
  - file format 1-13
  - history 1-10
  - overview 1-10

VRML editor

- general 5-2
- installing on host computer (Windows) 2-29
- on UNIX platforms 2-30
- V-Realm Builder 5-4

VRML viewer

- blaxxun Contact 6-44
- changing navigation speed 6-14
- installing on client computer (Windows) 2-38
- installing on host computer (UNIX) 2-23
- installing on host computer (Windows) 2-20
- known issue (blaxxun Contact) 2-22
- navigation 6-10
- rendering 6-35
- uninstalling 2-36
- viewpoint control 6-28
- Virtual Reality Toolbox 6-2

vrsetpref

- Virtual Reality Toolbox function 8-51

vrview

- Virtual Reality Toolbox function 8-56

vrwho

- Virtual Reality Toolbox function 8-57

vrwhos

- Virtual Reality Toolbox function 8-58

vrworld object

- creation 4-2

**W**

Web browser

- viewing a virtual world on a client computer  
3-19
- viewing a virtual world on the host computer  
3-15