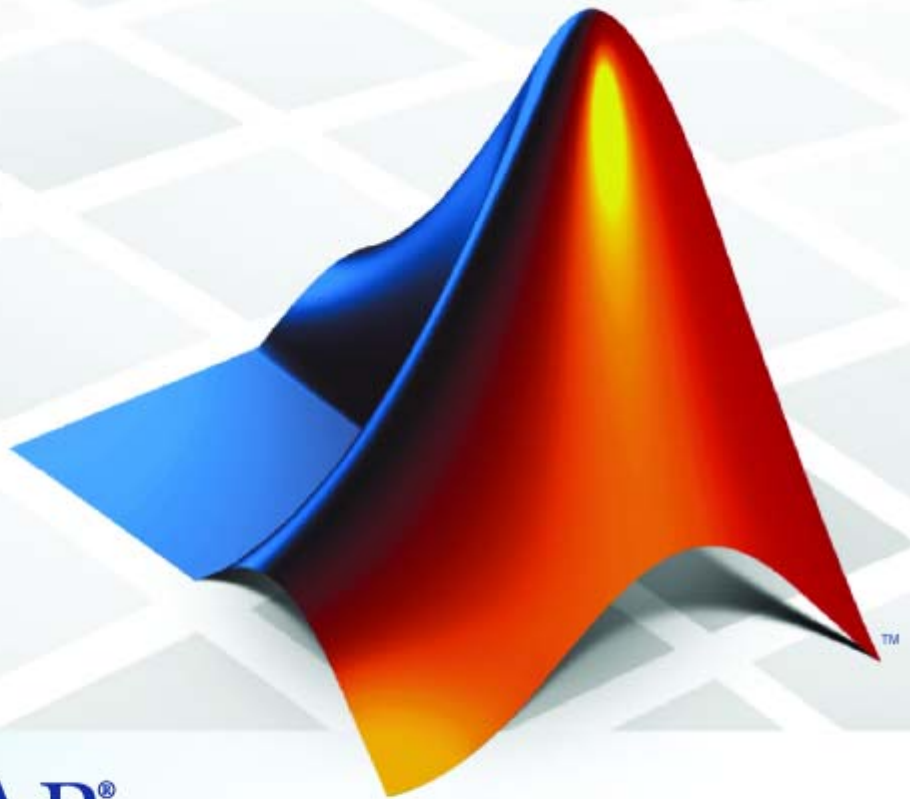


# Virtual Reality Toolbox™ 4

## User's Guide



**MATLAB®**  
& **SIMULINK®**

## How to Contact The MathWorks



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



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



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

*Virtual Reality Toolbox™ User's Guide*

© COPYRIGHT 2001–2008 by HUMUSOFT s.r.o. and The MathWorks, Inc.

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

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

### Trademarks

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

### Patents

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

## Revision History

August 2001	First printing	New for Version 2.0 (Release 12.1)
July 2002	Second printing	Revised for Version 3.0 (Release 13)
October 2002	Online only	Revised for Version 3.1 (Release 13)
June 2004	Third printing	Revised for Version 4.0 (Release 14)
October 2004	Fourth printing	Revised for Version 4.0.1 (Release 14SP1)
March 2005	Online only	Revised for Version 4.1 (Release 14SP2)
April 2005	Online only	Revised for Version 4.2 (Release 14SP2+)
September 2005	Online only	Minor revision for Version 4.2.1 (Release 14SP3)
March 2006	Online only	Revised for Version 4.3 (Release 2006a)
September 2006	Online only	Revised for Version 4.4 (Release 2006b)
March 2007	Online only	Revised for Version 4.5 (Release 2007a)
September 2007	Online only	Revised for Version 4.6 (Release 2007b)
March 2008	Online only	Revised for Version 4.7 (Release 2008a)
October 2008	Online only	Revised for Version 4.8 (Release 2008b)



## Getting Started

# 1

<b>Product Overview</b> .....	<b>1-2</b>
Overview .....	1-2
Expected Background .....	1-3
<b>Virtual Reality Toolbox Software Features</b> .....	<b>1-4</b>
VRML Support .....	1-4
MATLAB Interface .....	1-6
Simulink Interface .....	1-6
MATLAB® Compiler Support .....	1-7
VRML Viewers .....	1-8
VRML Editor .....	1-9
Real-Time Workshop Software Support .....	1-9
SimMechanics Support .....	1-10
Hardware Support .....	1-10
Client-Server Architecture .....	1-10
<b>VRML Overview</b> .....	<b>1-11</b>
VRML History .....	1-11
VRML Coordinate System .....	1-12
VRML File Format .....	1-14
<b>Examples Using the Virtual Reality Toolbox Product</b> ..	<b>1-17</b>
Simulink Interface Examples .....	1-17
MATLAB Interface Examples .....	1-25
<b>Virtual Reality Toolbox Texture File</b> .....	<b>1-29</b>
<b>Implementation Notes</b> .....	<b>1-30</b>
VRML Compatibility .....	1-30
Virtual Reality Toolbox Server .....	1-31

<b>Required Products</b> .....	2-2
Section Overview .....	2-2
MATLAB Product .....	2-2
VRML Viewer .....	2-3
<b>Recommended Product</b> .....	2-4
Simulink Product .....	2-4
<b>Related Products</b> .....	2-5
<b>System Requirements</b> .....	2-6
Section Overview .....	2-6
Supported Computer Platforms .....	2-6
Host Computer .....	2-8
Client Computer .....	2-9
<b>Installing Virtual Reality Toolbox Software on the Host Computer</b> .....	2-12
Section Overview .....	2-12
Components on a Host Computer .....	2-12
Installing from a DVD (Windows) .....	2-13
Installing from a DVD (UNIX/Linux) .....	2-14
LD_LIBRARY_PATH Environment Variable (UNIX) ....	2-15
Known Issue with the Virtual Reality Toolbox and Internet Explorer 6.0 (Windows) Products .....	2-15
<b>Installing the VRML Plug-In Viewer on the Host Computer</b> .....	2-17
Section Overview .....	2-17
Virtual Reality Toolbox Viewer .....	2-17
Installing a VRML Plug-In (Windows) .....	2-18
Installing a VRML Plug-In (UNIX and Linux) .....	2-21
Setting the Default Viewer of Virtual Scenes .....	2-22
<b>Installing the VRML Editor on the Host Computer</b> ....	2-27
Installing the VRML Editor (Windows) .....	2-27
VRML Editor (UNIX/Linux) .....	2-28
Setting the Default Editor of Virtual Scenes .....	2-28

<b>Changing Virtual Reality Toolbox Preferences with the MATLAB Preferences Dialog</b> .....	<b>2-34</b>
Section Overview .....	<b>2-34</b>
Virtual Reality Toolbox Preferences .....	<b>2-35</b>
Virtual Reality Toolbox Figure Preferences .....	<b>2-37</b>
Virtual Reality Toolbox World Preferences .....	<b>2-44</b>
<b>Removing Components (Windows)</b> .....	<b>2-46</b>
Section Overview .....	<b>2-46</b>
Removing Virtual Reality Toolbox and Ligos V-Realm Builder (Microsoft Windows) Software .....	<b>2-46</b>
Removing the Blaxxun Contact Plug-In (Windows) .....	<b>2-47</b>
<b>Installing on the Client Computer</b> .....	<b>2-48</b>
Section Overview .....	<b>2-48</b>
Installing a VRML Plug-In (Windows) .....	<b>2-48</b>
<b>Testing the Installation</b> .....	<b>2-49</b>
Section Overview .....	<b>2-49</b>
Running a Simulink Interface Example .....	<b>2-49</b>
Running a MATLAB Interface Example .....	<b>2-54</b>

## Simulink Interface

# 3

<b>Associating a Virtual World with a Simulink Block</b> ...	<b>3-2</b>
Section Overview .....	<b>3-2</b>
Adding a Virtual Reality Toolbox Block .....	<b>3-2</b>
Changing the Virtual World Associated with a Simulink Block .....	<b>3-10</b>
<b>Using the Simulink Interface</b> .....	<b>3-12</b>
Section Overview .....	<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 Host Computer .....	<b>3-15</b>
Viewing a Virtual World with a Web Browser on the 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 .....	4-9
<b>Recording Offline Animations</b> .....	<b>4-10</b>
Overview .....	4-10
Animation Recording File Tokens .....	4-12
Manual 3-D VRML Animation Recording .....	4-14
Manual 2-D AVI Animation Recording .....	4-16
Scheduled 3-D VRML Animation Recording .....	4-20
Scheduled 2-D AVI Animation Recording .....	4-22
Viewing Animation Files .....	4-25
MATLAB Animation Recording of Virtual Worlds Not Associated with Simulink Models .....	4-27

**Virtual Worlds**

**5**

<b>VRML Editing Tools</b> .....	<b>5-2</b>
Section Overview .....	5-2
Editors for Virtual Worlds .....	5-2
Ligos V-Realm Builder .....	5-4
<b>Deformation of a Sphere Example</b> .....	<b>5-5</b>
Section Overview .....	5-5
Defining the Problem .....	5-5
Adding a Virtual Reality Toolbox Block .....	5-6
Creating a Sphere in a Virtual World .....	5-9
Creating a Box in a Virtual World .....	5-14
Connecting a Simulink Model to a Virtual World .....	5-17
<b>VRML Data Types</b> .....	<b>5-21</b>
Section Overview .....	5-21
VRML Field Data Types .....	5-21



VRML Data Class Types .....	5-24
-----------------------------	------

## Using CAD Models with the Virtual Reality Toolbox

<b>Product</b> .....	5-26
Section Overview .....	5-26
Exporting VRML Models from CAD Tools .....	5-26
Virtual Scene Modeling .....	5-33
Linking Virtual Scene to a Simulink, SimMechanics, or MATLAB Model .....	5-37

## Viewing Virtual Worlds

# 6

<b>Virtual Reality Toolbox Viewer</b> .....	6-2
Section Overview .....	6-2
Menu Bar .....	6-5
Toolbar .....	6-6
Navigation Panel .....	6-7
Starting and Stopping Simulations .....	6-10
Navigation .....	6-11
Frame Capture and Animation Recording File Tokens ...	6-18
Creating Frame Captures .....	6-21
Configuring Animation Recording Parameters .....	6-23
Recording Files in the VRML Format .....	6-24
Recording Files in the Audio Video Interleave (AVI) Format .....	6-25
Scheduling Files for Recording .....	6-28
Interactively Starting and Stopping Animation Recording .....	6-31
Viewing the Animation File .....	6-31
Working with Viewpoints .....	6-33
Rendering .....	6-40
 <b>blaxxun Contact VRML Plug-In</b> .....	 6-49
Section Overview .....	6-49
Viewpoint Control .....	6-50
Control Menu .....	6-50
Navigation .....	6-51
Movement Modes .....	6-52
blaxxun Contact Settings .....	6-52

Stereoscopic Vision .....	6-53
---------------------------	------

## Virtual Reality Toolbox Stand-Alone Viewer

# 7

<b>What Is Orbisnap?</b> .....	7-2
<b>Installing Orbisnap</b> .....	7-3
Section Overview .....	7-3
System Requirements .....	7-3
Copying Orbisnap to Another Location .....	7-3
Adding Shortcuts or Symbolic Links .....	7-4
<b>Using Orbisnap</b> .....	7-5
Overview .....	7-5
Viewing Prerecorded WRL Animations or Virtual Worlds .....	7-6
Viewing the Virtual Reality Toolbox Server Virtual Worlds Remotely .....	7-7
<b>Orbisnap Interface</b> .....	7-10
Menu Bar .....	7-10
Toolbar .....	7-11
Navigation Panel .....	7-12
<b>Orbisnap Command Line</b> .....	7-17

## Block Reference

# 8

<b>Control Input Devices</b> .....	8-1
<b>Utilities</b> .....	8-1
<b>Virtual Worlds</b> .....	8-2

VRML-Related Signals ..... 8-2

**Blocks — Alphabetical List**

**9**

**Function Reference**

**10**

MATLAB Interface ..... 10-1

vrworld Object Methods ..... 10-2

vrnode Object Methods ..... 10-3

vrfigure Object Methods ..... 10-4

**Functions — Alphabetical List**

**11**

**Glossary**

**Index**



# Getting Started

---

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

- “Product Overview” on page 1-2
- “Virtual Reality Toolbox Software Features” on page 1-4
- “VRML Overview” on page 1-11
- “Examples Using the Virtual Reality Toolbox Product” on page 1-17
- “Virtual Reality Toolbox Texture File” on page 1-29
- “Implementation Notes” on page 1-30

## Product Overview

In this section...
“Overview” on page 1-2
“Expected Background” on page 1-3

### Overview

The Virtual Reality Toolbox product is a solution for interacting with virtual reality models of dynamic systems over time. It extends the capabilities of your MATLAB and Simulink software 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 the MATLAB and Simulink products.
- 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 product includes additional components:

- VRML viewer — Use either the Virtual Reality Toolbox viewer or, for Windows® platforms, the Blaxxun Contact® plug-in for Web browsers to display your virtual worlds.
- VRML editor — You can use any VRML editor to build your VRML environment or world. For your convenience, for Windows platforms, use Ligos® V-Realm Builder software 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 Virtual Reality Toolbox software on the Apple® Mac OS® X platform works as described for the UNIX/Linux platforms.

This guide assumes that you are already familiar with

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

If you are a new toolbox user, you might want to review the following topics:

- 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 environment.
- Chapter 4, “MATLAB Interface” — Interact with a virtual world from MATLAB environment.

## Virtual Reality Toolbox Software Features

### In this section...

“VRML Support” on page 1-4

“MATLAB Interface” on page 1-6

“Simulink Interface” on page 1-6

“MATLAB® Compiler Support” on page 1-7

“VRML Viewers” on page 1-8

“VRML Editor” on page 1-9

“Real-Time Workshop Software Support” on page 1-9

“SimMechanics Support” on page 1-10

“Hardware Support” on page 1-10

“Client-Server Architecture” on page 1-10

### 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 software uses many of the advanced features defined in the current VRML97 specification. The term VRML, in this guide, always refers to VRML as defined in the VRML97 standard ISO/IEC 14772-1:1997, available from <http://www.web3d.org>. This format includes a description of 3-D scenes, sounds, internal actions, and WWW anchors.



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

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.

---

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

---

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

The 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 product does not support VRML1.0, you need to save these worlds in the current standard for VRML, VRML97.

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

---

## **MATLAB Interface**

Virtual Reality Toolbox software provides a flexible MATLAB interface to virtual reality worlds. After creating MATLAB objects and associating them with a virtual world, you can control the virtual world by using functions and methods.

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

The 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.

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

You can implement most of the 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 toolbox automatically scans a virtual world for available VRML nodes that the Simulink software can drive.

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

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

## **MATLAB Compiler Support**

The Virtual Reality Toolbox product supports the MATLAB® Compiler™ product. With this capability, you can use the compiler to take M-files as input and generate redistributable, stand-alone applications that include toolbox functionality, including the Virtual Reality Toolbox viewer.

Stand-alone applications that include Virtual Reality Toolbox functionality have the following limitations:

- No Simulink software support, which results in no access to the Virtual Reality Toolbox Simulink library (`vrlib`).
- No Virtual Reality Toolbox server, which results in no remote connection for the Orbisnap or Blaxxun viewers
- No animation recording ability
- No editing world ability
- The following Virtual Reality Toolbox viewer features cannot be used in stand-alone applications:
  - **File > Open in Editor**
  - **Recording** menu
  - **Simulation** menu
  - **Help** access

To use these features, write an M-file that uses the MATLAB interface for the Virtual Reality Toolbox product (for example, creating, opening, and closing a `vrworld` object), then use the MATLAB Compiler product.

## **VRML Viewers**

The Virtual Reality Toolbox product contains a viewer that is the default viewing method for virtual worlds. This Virtual Reality Toolbox viewer is supported on PC, UNIX, Apple 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 toolbox includes the VRML plug-in blaxxun Contact. This is the only supported VRML plug-in.

If you install the VRML plug-in, the 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 these products are running, but also on other computers connected through the Internet

## VRML Editor

For PC platforms, the Virtual Reality Toolbox product includes one of the classic VRML authoring tools, V-Realm Builder by Ligos. With the addition of this VRML authoring tool, the product 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 — Ligos V-Realm Builder software Version 2.0 is included with the Virtual Reality Toolbox software. If you do not want to use V-Realm Builder software, 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 2-27.

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

- UNIX and Linux platforms — The default VRML editor for UNIX and 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 product.

## Real-Time Workshop Software Support

The following products support interaction with Real-Time Workshop® code.

The Simulink interface in the Virtual Reality Toolbox product supports the Real-Time Windows Target™ product. Using the Simulink external mode, you can interact with real-time code generated by the Real-Time Workshop product 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 product to view the behavior of a model created with the SimMechanics™ software. First, you build a model of a machine in the Simulink interface using SimMechanics blocks. Then, create a detailed picture of your machine in a virtual world, connect this world to the SimMechanics body sensor outputs, and view the behavior of the bodies in a VRML viewer.

## **Hardware Support**

The Virtual Reality Toolbox product contains functions for connecting to hardware devices, including joysticks and space mice, using Simulink blocks.

## **Client-Server Architecture**

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

- Single computer — MATLAB and 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.

## VRML Overview

In this section...
“VRML History” on page 1-11
“VRML Coordinate System” on page 1-12
“VRML File Format” on page 1-14

### VRML History

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

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

The term Virtual Reality Markup Language (VRML) was first used by Tim Berners-Lee at a European Web conference in 1994 when he talked about a need for a 3-D Web standard. Soon afterward, an active group of artists and engineers formed around a mailing list called `www-vmr1`. 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.

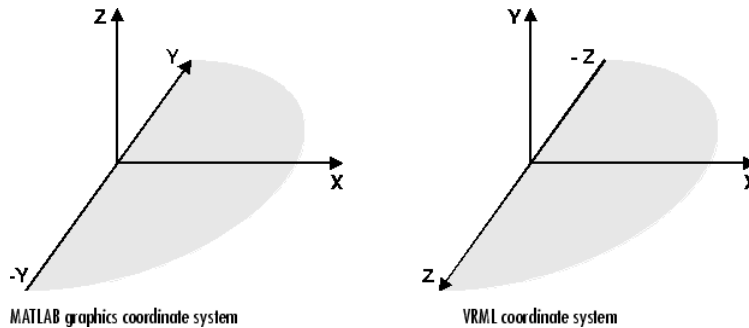
The VRML 1 standard was implemented in several VRML browsers, but it allowed you to create only static virtual worlds. This limitation reduced the possibility of its widespread use. Quickly it became clear that the language needed a robust extension to add animation and interactivity, and bring life 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 product uses VRML97 technology to deliver a unique, open 3-D visualization solution for MATLAB users. It is a useful contribution to a wide use of VRML97 in the field of technical and scientific computation and interactive 3-D animation.

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

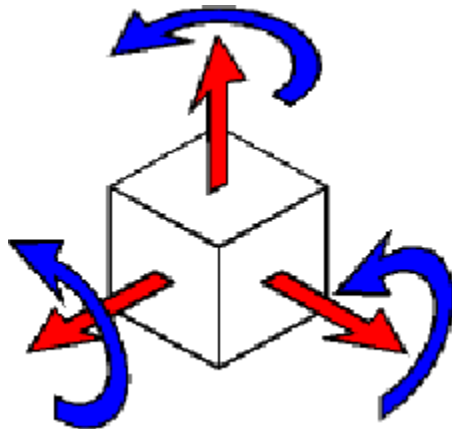
## VRML Coordinate System





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

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

In VRML, a 3-D scene is described by a hierarchical tree structure of objects (nodes). Every node in the tree represents some functionality of the scene. There are 54 different types of nodes. Some of them are *shape nodes* (representing real 3-D objects), and some of them are *grouping nodes* used for holding child nodes. Here are some examples:

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

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

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

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

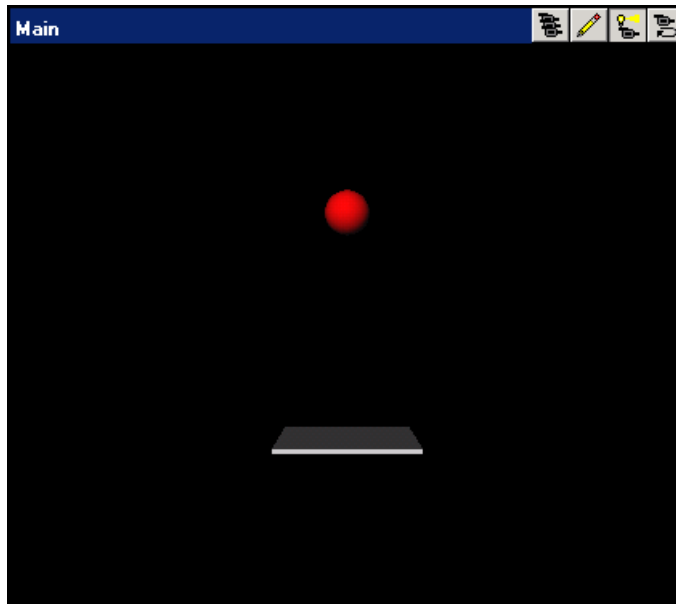
In the following example of a simple VRML file, two graphical objects are modeled in a 3-D scene: A floor is represented by a flat box with a red ball above it. Note that the VRML file is a readable text file that you can write in any text editor.

```
#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 this:



## Examples Using the Virtual Reality Toolbox Product

In this section...
“Simulink Interface Examples” on page 1-17
“MATLAB Interface Examples” on page 1-25

### Simulink Interface Examples

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

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

```
vrbounce
```

A Simulink window opens with the block diagram for the model. By default, a virtual world also opens in the Virtual Reality Toolbox viewer or your VRML-enabled Web browser. If you close the virtual world window, double-click the VR Sink block to display it again.

---

**Note** If the viewer does not appear, double-click the VR Sink block in the Simulink model. In the Virtual Reality Toolbox viewer, from the **Simulation** menu, click **Block Parameters**. A Block Parameters dialog box opens. The **Open VRML viewer automatically** check box should be selected by default. This selection enables the virtual world window to be displayed when you double-click the VR Sink block.

---

- 2 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 product. Descriptions of the examples follow the table.

Demo	Real-Time Workshop Ready	VR Sink	Joystick	Space Mouse
vrbounce	X	X		
vrcrane_joystick		X	X	
vrcrane_traj	X	X		
vrlights	X	X		
vrmaglev	X	X		
vrmaglev_rtwin	X	X		
vrmanipul		X		X
vrmemb1	X	X		
vrmorph	X	X		
vr_octavia	X	X		
vr_octavia_video		X		
vrpend	X	X		
vrplanets	X	X		
vrtkoff	X	X		
vrtkoff_trace		X		

### Bouncing Ball Example (vrbounce)

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

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

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

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

For details on how the crane model hierarchy and scripting logic is implemented, see the associated commented VRML file `portal_crane.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 the Simulink environment and virtual worlds. The Simulink model represents the HUMUSOFT® CE 152 Magnetic Levitation educational/presentation scale model. The plant model is controlled by a PID controller with feed-forward to cope with the nonlinearity of the magnetic levitation system.

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 the HUMUSOFT MF 624 data acquisition board, and Real-Time Workshop and Real-Time Windows Target software. However, you can adapt this model for other targets and acquisition boards. A digital IIR filter, from the Signal Processing Blockset™ library, filters the physical system output. You can bypass the physical system by using the built-in plant model. Running this model in real time is an example showing the capabilities of the Simulink product in control systems design and rapid prototyping.

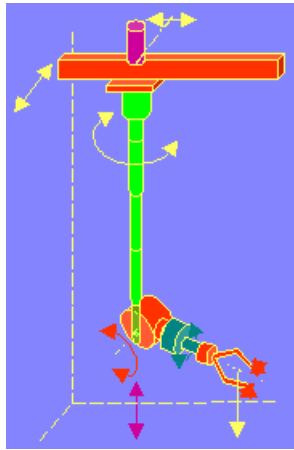
Note that after enabling the remote view in the VR Sink block dialog box, you can control the Simulink model even from another (remote) client computer. This can be useful for distributing the computing power between a real-time Simulink model running on one machine and the rendering of a virtual reality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

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



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

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

The 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.

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

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

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

The vr\_octavia example illustrates the benefits of the visualization of complex dynamic model in the virtual reality environment. It also demonstrates the Virtual Reality Toolbox 3-D off-line animation recording functionality.

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

The vr\_octavia\_video example illustrates how to use video output from the VR To Video block. This model performs simple operations on the video output. It requires the Video and Image Processing Blockset™ product.

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

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

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

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

Notice the pseudoorthographic view defined in the associated VRML file. You achieve this effect by creating a viewpoint that is located far from the object of interest with a very narrow view defined by the VRML **FieldOfView** parameter. An orthographic view is useful for eliminating the panoramic distortion that occurs when you are using a wide-angle lens. The disadvantage of this technique is that locating the viewpoint at a distance makes the standard viewer navigation tricky or difficult in some navigation modes, such as the Examine mode. If you want to navigate around the virtual pendulum bench, you should use some other viewpoint.

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

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

Several viewpoints are defined in the virtual 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 Ligos V-Realm Builder Object Library is incorporated into the scene. The file vrtkoff2.wrl uses the same scene with a different type of aircraft.

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

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

## MATLAB Interface Examples

The following table lists the MATLAB interface examples provided with the toolbox. Descriptions of the examples follow the table.

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

### Car in the Mountains Example (vrcar)

This demonstration illustrates the use of the Virtual Reality Toolbox product with the MATLAB interface. In a step-by-step tutorial, it shows commands for navigating a virtual car along a path through the mountains.

- 1 In the MATLAB Command Window, type

```
vrcar
```

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

### Heat Transfer Example (vrheat)

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

In this demonstration, matrix-type data is transferred between the MATLAB software and a virtual reality world. Using this feature, you can achieve massive color changes or morphing. This is useful for representing various physical processes. Precalculated data of time-based temperature distribution

in an L-shaped metal block is used. The data is then sent to the virtual world. This forms an animation with relatively large changes.

This is a step-by-step 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 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 3-D graphic object generated from the MATLAB environment with the Virtual Reality Toolbox product. The membrane was generated by the `logo` function and saved in the VRML format using the standard `vrml` function. You can save all Handle Graphics® objects this way and use them with the Virtual Reality Toolbox software 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).

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

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

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

This demo requires the Mapping Toolbox software from The MathWorks.

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

This demonstration illustrates how to use a space mouse using the MATLAB interface. After you start this demo, a virtual scene with an aircraft is displayed in the Virtual Reality Toolbox viewer. You can navigate the plane in the scene using a space mouse input device. Press button 1 to place a marker at the current plane position.

This demo requires a space mouse or compatible device.



## Virtual Reality Toolbox Texture File

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

- Where possible, scale source texture files to a size equal to a power of 2 in both dimensions. Doing so ensures optimal performance for the 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

In this section...
“VRML Compatibility” on page 1-30
“Virtual Reality Toolbox Server” on page 1-31

### VRML Compatibility

The Virtual Reality Toolbox product 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 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 software uses a Virtual Reality Toolbox HTTP server for communication between a VRML-enabled Web browser and the MATLAB and 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 toolbox is loaded into the MATLAB interface. 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 product provides the files you need for installation on both your host computer and client computer.

- “Required Products” on page 2-2
- “Recommended Product” on page 2-4
- “Related Products” on page 2-5
- “System Requirements” on page 2-6
- “Installing Virtual Reality Toolbox Software on the Host Computer” on page 2-12
- “Installing the VRML Plug-In Viewer on the Host Computer” on page 2-17
- “Installing the VRML Editor on the Host Computer” on page 2-27
- “Changing Virtual Reality Toolbox Preferences with the MATLAB Preferences Dialog” on page 2-34
- “Removing Components (Windows)” on page 2-46
- “Installing on the Client Computer” on page 2-48
- “Testing the Installation” on page 2-49

## Required Products

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

### Section Overview

The Virtual Reality Toolbox product 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 product.

### MATLAB Product

the MATLAB software 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 4.8 of Virtual Reality Toolbox software requires MATLAB software Version 7.7.

---

MATLAB documentation — For information on using the MATLAB software, see the MATLAB documentation. It explains how to work with data and how to use the functions supplied with the MATLAB software. For a reference describing the functions specific to the Virtual Reality Toolbox software, see the Virtual Reality Toolbox online documentation.

## 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 product 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 the MATLAB software.

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, Apple 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 product. 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 product. 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 2-18.

For information on using a Web browser to view virtual worlds, see “Testing the Installation” on page 2-49. The Blaxxun Contact installation executable files are located at `C:\matlabroot\toolbox\vr\blaxxun`.

Every VRML plug-in installs Sun™ Java™ classes into the Web browser. Limit the number of plug-ins you use to avoid Java errors and conflicts. For this reason, use only the 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 the Simulink product to use the Virtual Reality Toolbox product.

### Simulink Product

The Simulink product 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 product, you can interact with the VR representation of the model you created with Simulink blocks, and visualize the simulation of your dynamic system over time.

---

**Note** Version 4.8 of Virtual Reality Toolbox software uses Simulink software Version 7.2.

---

Simulink documentation — For information on using the Simulink software, 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 8, “Block Reference”.



## Related Products

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

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

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

## System Requirements

In this section...
“Section Overview” on page 2-6
“Supported Computer Platforms” on page 2-6
“Host Computer” on page 2-8
“Client Computer” on page 2-9

### Section Overview

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

### Supported Computer Platforms

The VR server is the part of the Virtual Reality Toolbox software that interfaces with your Simulink models. It stores information about the current state of virtual worlds and manages connections to VR clients. The VR client is a VRML viewer that displays a virtual world. The VR client can be either the 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>
Windows XP	Yes	Yes	Ligos V-Realm Builder*	blaxxun Contact*
Linux 2.4.x kernels	Yes	Yes	MATLAB editor*	No
Sun Solaris 10	Yes	Yes	MATLAB editor*	No
Apple Power Macintosh running OS X (10.4.9 or later)	Yes	Yes	MATLAB editor*	No

\* Distributed with the Virtual Reality Toolbox software.

## Host Computer

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

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

### Host Computer Hardware Requirements

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

The following table lists the minimum software the toolbox requires on your host computer. For a list of optional software products, see <http://www.mathworks.com/products/virtualreality/related.jsp>.

### Host Computer Software Requirements

Software	Description
MATLAB product	Version 7.7.
Simulink product	Version 7.2. The Simulink product is not required, but highly recommend.
Virtual Reality Toolbox product	Version 4.8.

## Host Computer Software Requirements (Continued)

Software	Description
VRML editor	<p>For Microsoft Windows platforms, you can install the Ligos VRML editor (V-Realm Builder 2.0) provided with the Virtual Reality Toolbox product. For UNIX and 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.</p>
Web browser	<p>On PC platforms, you can use a Web browser and the Blaxxun Contact plug-in to view virtual worlds. This is an alternative to using the Virtual Reality Toolbox viewer.</p> <p>Use Internet Explorer® software Version 4.0 or later, or Netscape Navigator® 4.0 or later with Sun Java enabled.</p>
VRML plug-in	<p>If you are using a Web browser instead of the 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 product.</p> <p>For information on how to install the Blaxxun Contact plug-in, see “Installing a VRML Plug-In (Windows)” on page 2-18.</p>

## Client Computer

You can use a client computer to view and control a virtual world. Because the MATLAB or Simulink product does not run on this computer, you must connect to a host computer running a simulation or executable code. The host

computer, through the 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 product needs on the client computer.

### Client Computer Hardware Requirements

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

The following table lists the software the Virtual Reality Toolbox installation requires on the client computer. You do not need to install the 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.

### Client Computer Software Requirements

Software	Description
Operating system	Windows XP (the TCP/IP protocol must be installed).

**Client Computer Software Requirements (Continued)**

<b>Software</b>	<b>Description</b>
Web browser	Use Internet Explorer 4.0 or later, or Netscape Navigator 4.0 or later with SunJava enabled.
VRML plug-in	<p>VRML97 plug-in with External Authoring Interface support. If you have the Blaxxun Contact software (Windows) on your computer, you have already installed a VRML plug-in.</p> <p><b>Windows platforms</b> – You can install the Blaxxun Contact 4.4 plug-in provided with the Virtual Reality Toolbox product.</p> <p>For information on how to install the Blaxxun Contact plug-in, see “Installing a VRML Plug-In (Windows)” on page 2-18.</p>

# Installing Virtual Reality Toolbox Software on the Host Computer

## In this section...

“Section Overview” on page 2-12

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

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

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

“LD\_LIBRARY\_PATH Environment Variable (UNIX)” on page 2-15

“Known Issue with the Virtual Reality Toolbox and Internet Explorer 6.0 (Windows) Products” on page 2-15

## Section Overview

You may install the Virtual Reality Toolbox software from a DVD or from the MathWorks Web site. Before you install the software for a standard installation, you need your online MathWorks Account. For detailed information about the installation process, see the MathWorks installation documentation for your platform.

## Components on a Host Computer

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

- Virtual Reality Toolbox software — This component contains the core files that interconnect the MATLAB and Simulink interfaces to VRML. This component is required for the 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 software, and it is set as the default viewer for displaying virtual worlds.



- VRML plug-in — Optionally, you can use a VRML plug-in for a Web browser to view virtual reality worlds. The Blaxxun Contact plug-in is included with the Virtual Reality Toolbox product for Microsoft 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. Ligos V-Realm Builder software is included with the Virtual Reality Toolbox product 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 and 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 product and as templates for building your own projects. This component is not used on the client computer.
- Online documentation — This component contains the guide you are reading now. You can access the online version through the MATLAB Help browser. An Adobe® Acrobat® PDF file is available on the MathWorks Web site at <http://www.mathworks.com>. Follow the links to product documentation. This documentation can be read using the Adobe Acrobat Reader. If you do not have this reader installed on your computer, you can download it from <http://www.adobe.com>.

## Installing from a DVD (Windows)

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

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

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

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

The Virtual Reality Toolbox viewer is installed with the 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 2-18.

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 2-27.

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

The following is an overview of how to install the Virtual Reality Toolbox software on a UNIX or Linux platform from the DVD. Consult the installation guide for your platform for a comprehensive explanation of the installation process.

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

This dialog box lists all the products you are licensed to install in the **Items to Install** box. Make sure the Virtual Reality Toolbox product is listed in this box.

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

The Virtual Reality Toolbox viewer is the default viewer for UNIX platforms. For more information, see “Virtual Reality Toolbox Viewer” on page 2-17.

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 2-28.

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

If your system does not have the OpenGL® software 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 Internet Explorer 6.0 (Windows) Products**

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

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

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

To manually edit the Java library path for 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 Plug-In Viewer on the Host Computer

In this section...
“Section Overview” on page 2-17
“Virtual Reality Toolbox Viewer” on page 2-17
“Installing a VRML Plug-In (Windows)” on page 2-18
“Installing a VRML Plug-In (UNIX and Linux)” on page 2-21
“Setting the Default Viewer of Virtual Scenes” on page 2-22

### Section Overview

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.

### 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 toolbox. You can view virtual scenes as soon as the toolbox software 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 to avoid Sun 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 software, 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 the Internet Explorer or Netscape Navigator browser. The Blaxxun Contact plug-in is the only supported VRML plug-in.

---

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

---

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

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

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

```
vrinstall -install viewer
```

The MATLAB interface displays the message

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

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

```
d
```

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



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

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

```
vrinstall
-check
```

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

```
External VRML viewer:    installed
```

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

```
VRML
viewer:    not installed
```

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

The Blaxxun Contact VRML plug-in can fail to update the virtual scene when used with the Virtual Reality Toolbox and Internet Explorer 5.5 and later products. Netscape Navigator users do not experience this problem.

If you are using Internet Explorer 5.5 or later, you must manually change a network security setting before you can use Blaxxun Contact 4.4 with Virtual Reality Toolbox Version 3.0 or later. Upgrading your version of the Blaxxun Contact plug-in 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 later to ensure that the virtual scene is updated appropriately. You can use this workaround for the following:

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

**1** Open Internet Explorer.

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

The Internet Options dialog box opens.

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

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

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

The Security Settings dialog box opens.

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



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

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

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

The Local intranet dialog box opens.

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

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

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

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

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

The Local intranet dialog box closes.

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

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

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

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

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

If you want to use a Web browser instead of the 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 the Blaxxun Contact plug-in for Microsoft Microsoft platforms. If you are using any other operating system, you need to use the Virtual Reality Toolbox viewer to view virtual worlds.

---

**Note** The Blaxxun Contact interface 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. (Alternatively, if you want to use the MATLAB File menu Preferences dialog, see “Changing Virtual Reality Toolbox Preferences with the MATLAB Preferences Dialog” on page 2-34.) The following procedure assumes that you are working on a PC platform:

- 1 At the MATLAB command prompt, type

```
vrinstall -check
```

to determine whether the Blaxxun Contact software is installed.

The MATLAB Command Window displays

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

The viewer and editor are installed. If the viewer is not installed, see “Installing a VRML Plug-In (Windows)” on page 2-18.

- 2 Determine your default viewer by typing

```
vrgetpref
```

The MATLAB Command Window displays

```
ans =

                                         DataTypeBool: 'logical'
                                         DataTypeInt32: 'double'
                                         DataTypeFloat: 'double'
DefaultFigureAntialiasing: 'off'
DefaultFigureCaptureFileFormat: 'tif'
DefaultFigureCaptureFileName: '%f_anim_%n.tif'
DefaultFigureDeleteFcn: ''
DefaultFigureLighting: 'on'
DefaultFigureMaxTextureSize: 'auto'
DefaultFigureNavPanel: 'halfbar'
```

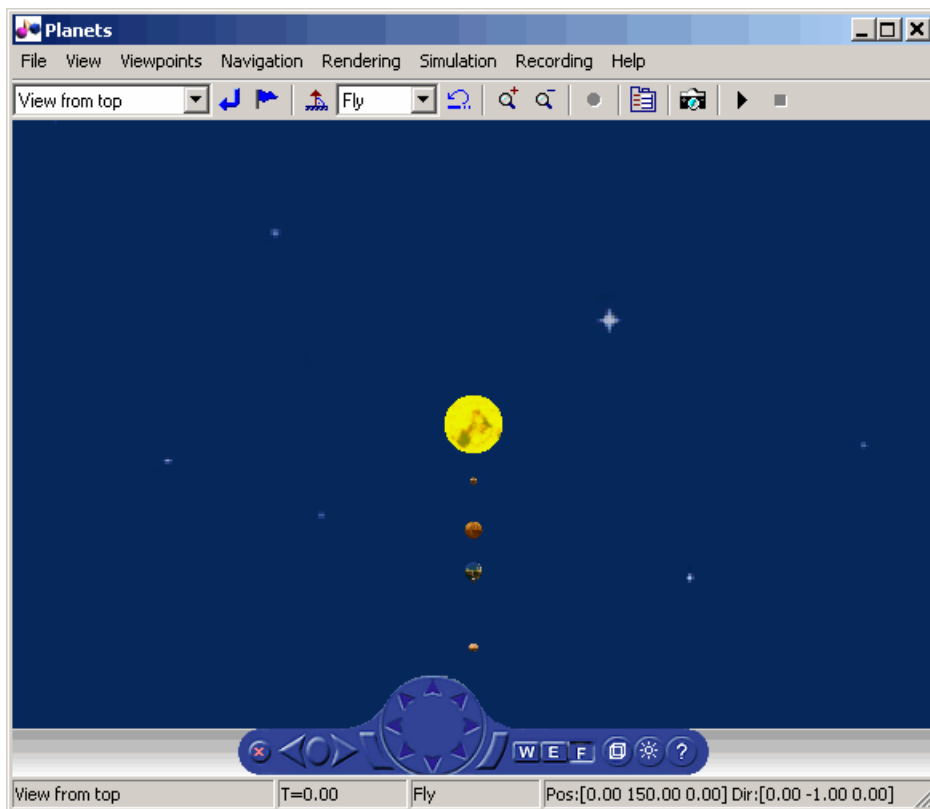
```
DefaultFigureNavZones: 'off'  
DefaultFigurePosition: [5 92 576 380]  
DefaultFigureRecord2DCompressMethod: 'auto'  
DefaultFigureRecord2DCompressQuality: 75  
DefaultFigureRecord2DFileName: '%f_anim_%n.avi'  
DefaultFigureStatusBar: 'on'  
DefaultFigureTextures: 'on'  
DefaultFigureToolBar: 'on'  
DefaultFigureTransparency: 'on'  
DefaultFigureWireframe: 'off'  
DefaultViewer: 'internal'  
DefaultWorldRecord3DFileName: '%f_anim_%n.wrl'  
DefaultWorldRecordMode: 'manual'  
DefaultWorldRecordInterval: [0 0]  
DefaultWorldRemoteView: 'off'  
DefaultWorldTimeSource: 'external'  
Editor: [1x60 char]  
HttpPort: 8123  
TransportBuffer: 5  
TransportTimeout: 20  
VrPort: 8124
```

The `DefaultViewer` property is set to `'internal'`. The 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

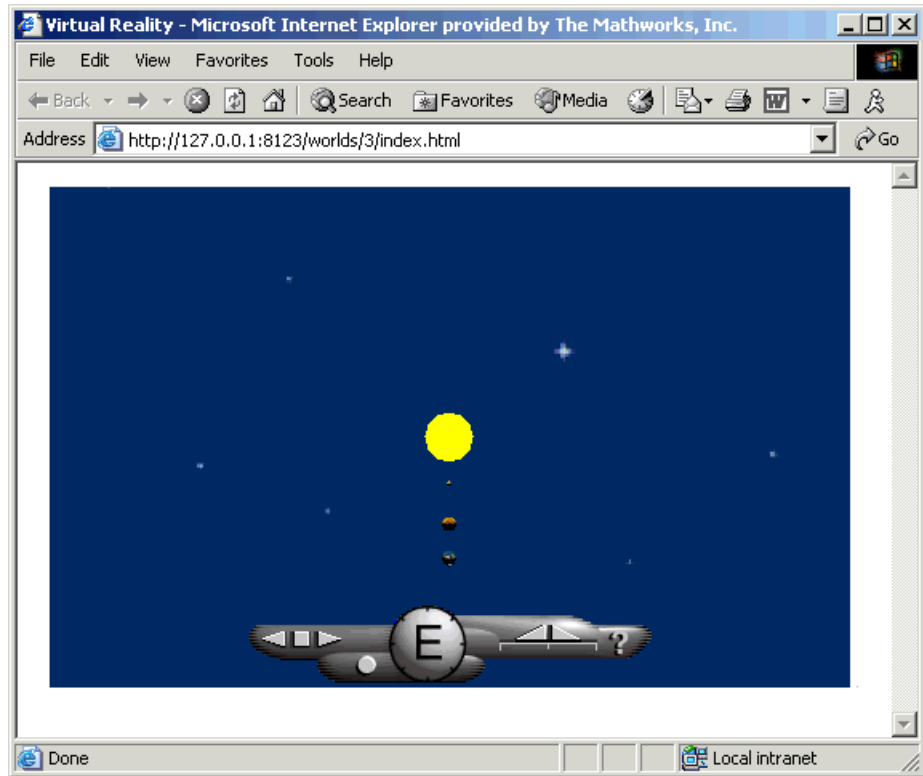
```
vrsetpref('DefaultViewer','web')
```

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

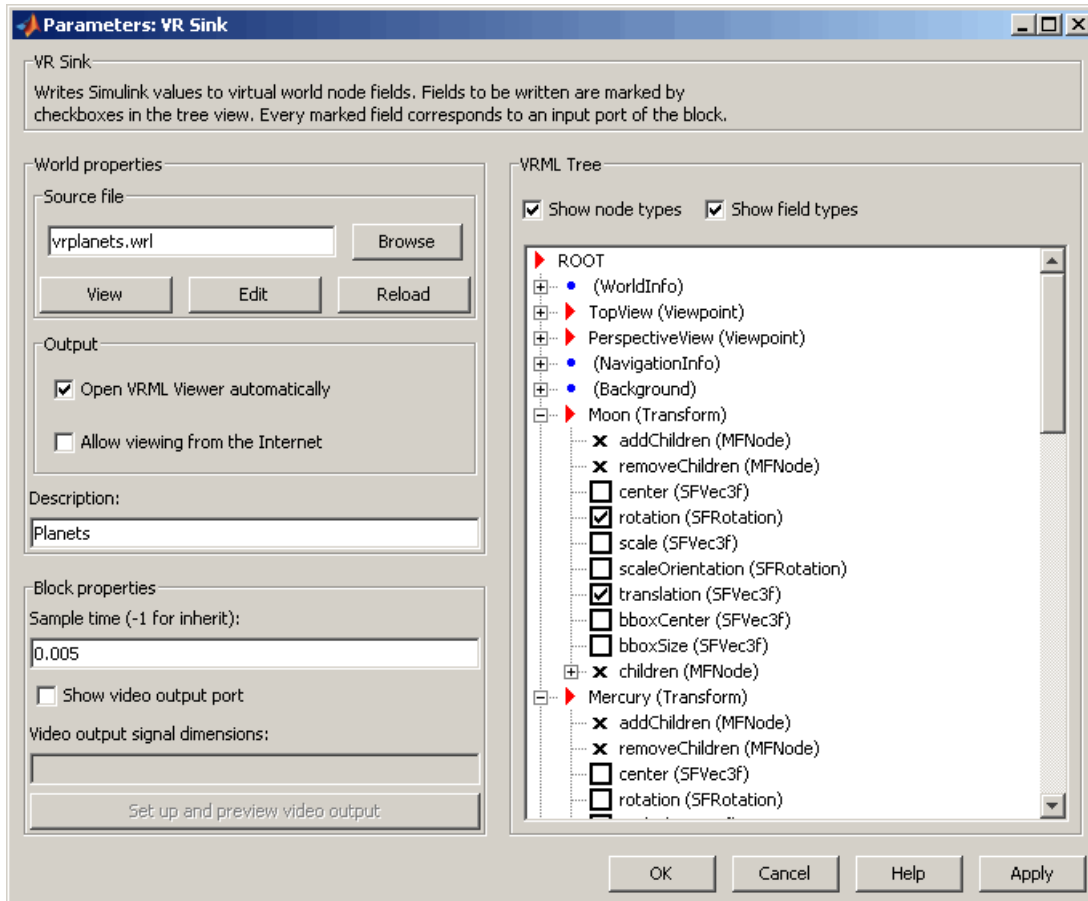
```
vrplanets
```

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

### In this section...

“Installing the VRML Editor (Windows)” on page 2-27

“VRML Editor (UNIX/Linux)” on page 2-28

“Setting the Default Editor of Virtual Scenes” on page 2-28

### Installing the VRML Editor (Windows)

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

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

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

```
vrinstall  
-install editor
```

or type

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

The MATLAB Command Window displays the following messages:

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

### 3 Type

```
vrinstall  
-check
```

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

```
VRML editor:    installed
```

### **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 5-21 for the data types to use with the MATLAB software.

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 Windows 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. (Alternatively, if you want to use the MATLAB **File** menu Preferences dialog, see “Changing Virtual Reality Toolbox Preferences with the MATLAB Preferences Dialog” on page 2-34.)

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.

The MATLAB Command Window 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 2-27.

- 2 Determine your default editor by typing

```
a = vrgetpref
```

The MATLAB Command Window 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
```

The MATLAB Command Window 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.

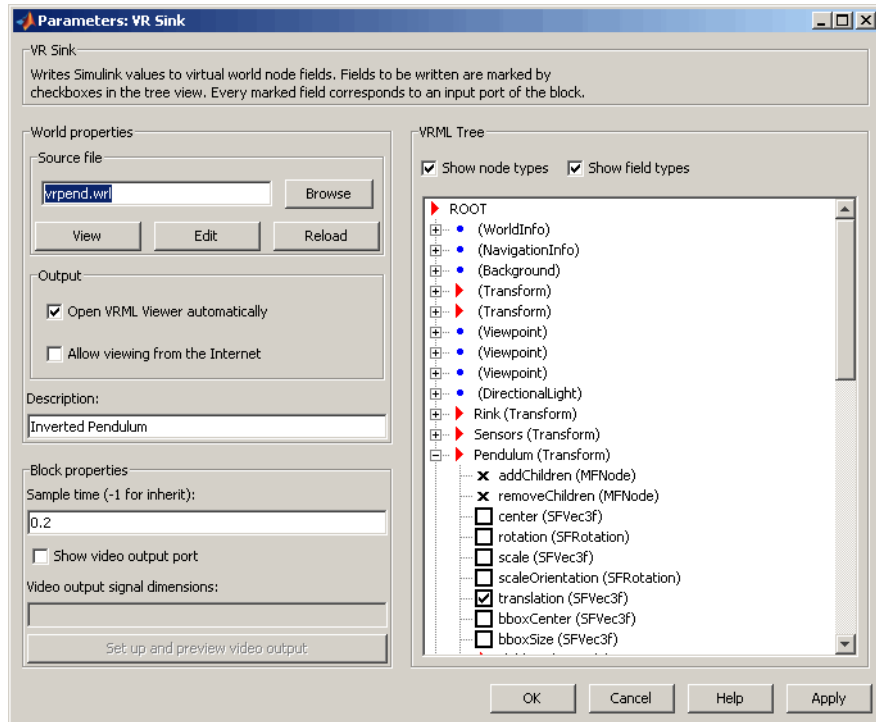
- 4 Verify that V-Realm Builder is your default editor. At the MATLAB command prompt, type

```
vrpend
```

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

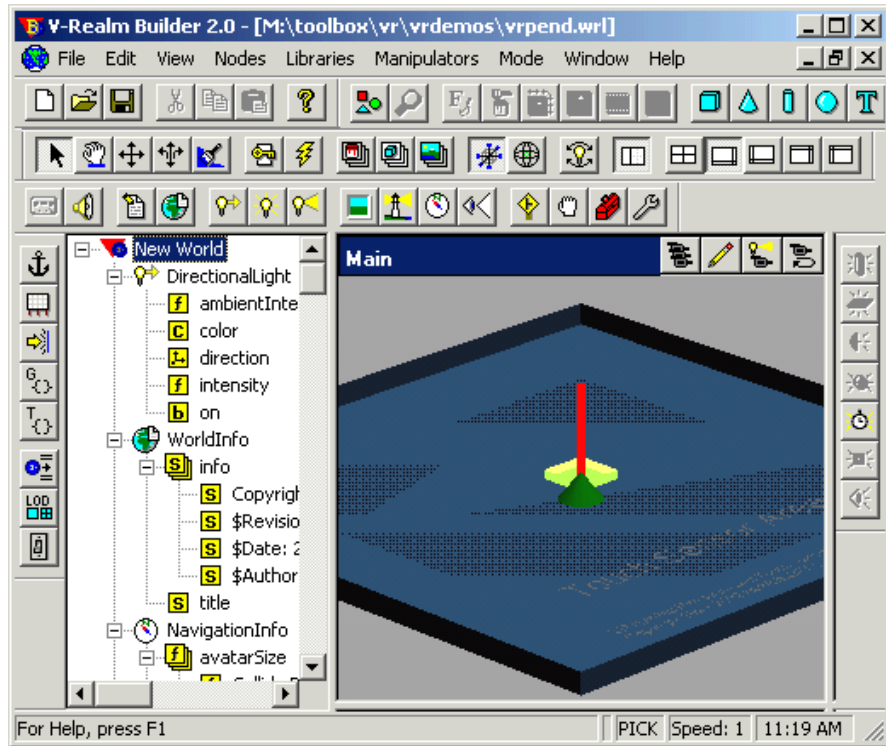
- 5 In the Virtual Reality Toolbox viewer for `vrpend`, from the **Simulation** menu, select **Block Parameters**

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.

## Changing Virtual Reality Toolbox Preferences with the MATLAB Preferences Dialog

### In this section...

“Section Overview” on page 2-34

“Virtual Reality Toolbox Preferences” on page 2-35

“Virtual Reality Toolbox Figure Preferences” on page 2-37

“Virtual Reality Toolbox World Preferences” on page 2-44

### Section Overview

The topics in this section describe how to set the Virtual Reality Toolbox preferences using the MATLAB **File > Preferences** dialog. The list of settable preferences is a subset of those available through the MATLAB interface functions.

The Virtual Reality Toolbox software installs with default preference settings. You can change these settings with

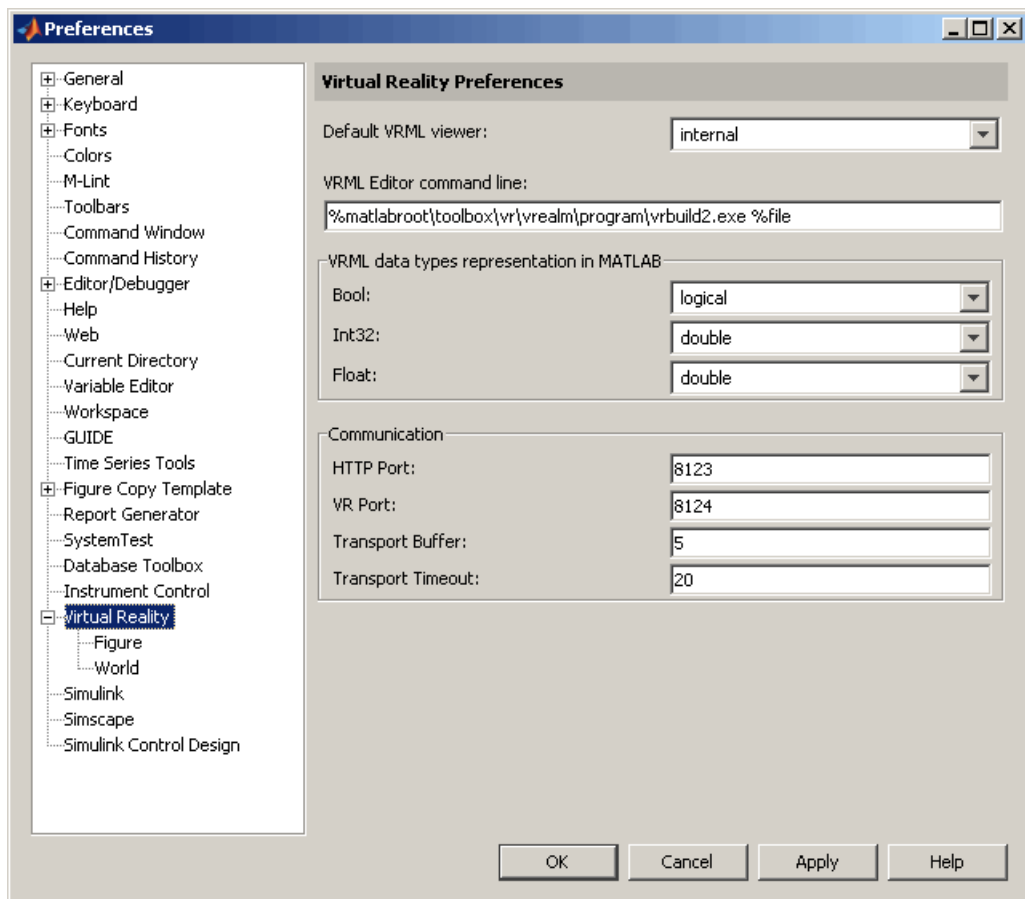
- MATLAB **File > Preferences** dialog — This GUI has preference dialogs for the MATLAB product and its related products, including the Virtual Reality Toolbox product.
- Virtual Reality Toolbox MATLAB interface functions
  - Virtual Reality Toolbox — `vrsetpref` and `vrgetpref` functions
  - Virtual figures — `vrfigure/set` and `vrfigure/get` functions
  - Virtual worlds — `vrworld/set` and `vrworld/get` functions

## Virtual Reality Toolbox Preferences

To access the Virtual Reality Toolbox preferences GUI:

- 1 From the MATLAB desktop, select **File > Preferences**.
- 2 In the left pane of the Preferences dialog box, select **Virtual Reality**.

The Virtual Reality Preferences dialog appears in the right pane.



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

<b>Preference</b>	<b>Value</b>	<b>Description</b>
Bool	'logical'   'char' Default: 'logical'	Specifies the handling of the VRML Bool data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . If set to 'logical', the VRML Bool data type is returned as a logical value. If set to 'char', the Bool data type is returned 'on' or 'off'.
Default VRML Viewer	'internal'   'web' Default: 'internal'	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'.
Float	'single'   'double' Default: 'double'	Specifies the handling of the VRML float data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . If set to 'single', the VRML Float and Color data types are returned as 'single'. If set to 'double', the Float and Color data types are returned as 'double'.
Int32	'int32'   'double' Default: 'double'	Specifies handling of the VRML Int32 data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . If set to 'int32', the VRML Int32 data type is returned as int32. If set to 'double', the Int32 data type is returned as 'double'.
HTTP Port	Numeric Default: 8123	IP port number used to access the VR server over the Web via HTTP. If you change this preference, you must restart the MATLAB software before the change takes effect.
Transport Buffer	Numeric Default: 5	Length of the transport buffer (network packet overlay) for communication between the VR server and its clients.
Transport Timeout	Numeric Default: 20	Amount of time, in seconds, 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.



Preference	Value	Description
VRML Editor command line	String	Path to the VRML editor. If this path is empty, the MATLAB editor is used.
VR Port	Numeric Default: 8124	IP port used for communication between the VR server and its clients. If you change this preference, you must restart the MATLAB software before the change takes effect.

## Virtual Reality Toolbox Figure Preferences

The Virtual Reality Toolbox figure has a number of preferences, presented in the following categories:

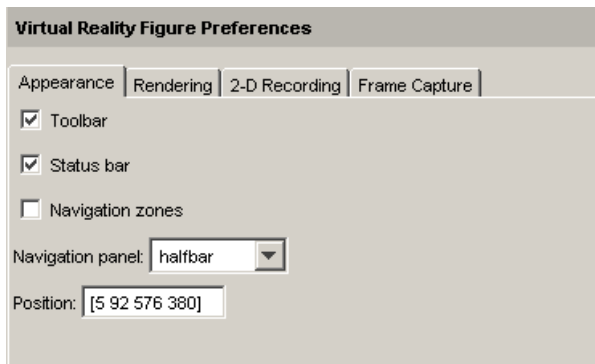
- “Virtual Reality Toolbox Figure Appearance Preferences” on page 2-37
- “Virtual Reality Toolbox Figure Rendering Preferences” on page 2-38
- “Virtual Reality Toolbox Figure 2-D Recording Preferences” on page 2-40
- “Virtual Reality Toolbox Figure Frame Capture Preferences” on page 2-42

### Virtual Reality Toolbox Figure Appearance Preferences

To access the virtual figure appearance preferences:

- 1** From the MATLAB software desktop, select **File > Preferences**.
- 2** In the left pane of the Preferences dialog box, select **Virtual Reality**.
- 3** In the left pane under **Virtual Reality**, select **Figure**.

The Virtual Reality Figure Preferences dialog appears in the right pane, with the **Appearance** tab selected.



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

Property	Value	Description
Navigation panel	'opaque'   'translucent'   'none'   'halfbar'   'bar' Default: 'halfbar'	Controls the appearance of the navigation panel in the Virtual Reality Toolbox viewer.
Navigation zones	'off'   'on' Default: 'off'	Toggles navigation zones on/off.
Position	Vector of four doubles	Specifies the screen coordinates of this vrfigure object.
Status bar	'off'   'on' Default: 'on'	Toggles the status bar at the bottom of the Virtual Reality Toolbox viewer.
Toolbar	'off'   'on' Default: 'on'	Toggles the toolbar on the Virtual Reality Toolbox viewer.

### Virtual Reality Toolbox Figure Rendering Preferences

To access the virtual figure rendering preferences:

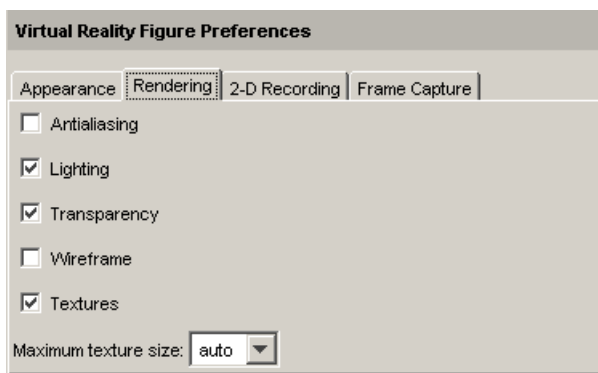
- 1** From the MATLAB desktop, select **File > Preferences**.
- 2** In the left pane of the Preferences dialog box, select **Virtual Reality**.

**3** In the left pane under **Virtual Reality**, select **Figure**.

The Virtual Reality Figure Preferences dialog appears in the right pane.

**4** Select the **Rendering** tab.

The Virtual Reality Figure Preferences dialog appears in the right pane, with the **Rendering** tab selected.



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

Property	Value	Description
Antialiasing	'off'   'on' Default: 'off'	Determines whether antialiasing is used when rendering scene. Antialiasing smooths textures by interpolating values between texture points.
Lighting	'off'   'on' Default: 'on'	Specifies whether the lighting is taken into account when rendering. If it is off, all the objects are drawn as if uniformly lit.

<b>Property</b>	<b>Value</b>	<b>Description</b>
Maximum texture size	'auto'   32 <= x <= video card limit, where x is a power of 2 (video card limit is typically 1024 or 2048)	Sets the maximum pixel size of a texture used in rendering <code>vrfigure</code> objects. The smaller the size, the faster the texture can render. Increasing this value improves image quality but decreases performance. A value of 'auto' sets the maximum possible pixel size. If the value you enter is unsuitable, a warning might trigger. The toolbox then automatically adjusts the property to the next smaller suitable value.
Textures	'off'   'on' Default: 'on'	Turns texture rendering on or off.
Transparency	'off'   'on' Default: 'on'	Specifies whether or not transparency information is taken into account when rendering.
Wireframe	'off'   'on' Default: 'off'	Specifies whether objects are drawn as solids or wireframes.

### **Virtual Reality Toolbox Figure 2-D Recording Preferences**

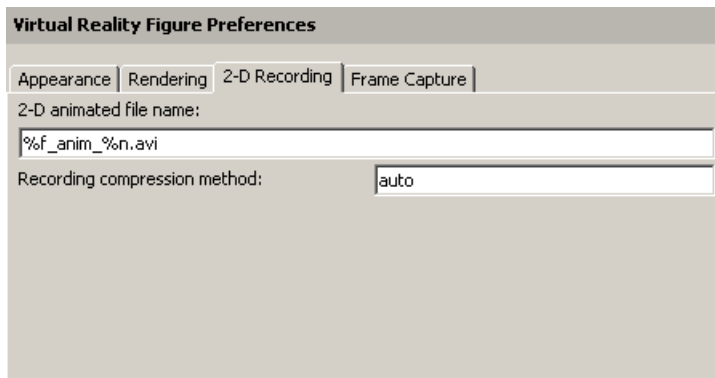
To access the virtual figure 2-D recording preferences:

- 1** From the MATLAB desktop, select **File > Preferences**.
- 2** In the left pane of the Preferences dialog box, select **Virtual Reality**.
- 3** In the left pane under **Virtual Reality**, select **Figure**.

The Virtual Reality Figure Preferences dialog appears in the right pane.

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

The Virtual Reality Figure Preferences dialog appears in the right pane, with the **2-D Recording** tab selected.



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

Property	Value	Description
2-D animated file name	String. Default: '%f_anim_%n.avi'	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.
Recording compression method	' '   'auto'   'lossless'   'codec_code' Default: 'auto'	Specifies the compression method for creating 2-D animation files. The codec code must be registered in the system. See the MATLAB function documentation for <code>avifile</code> . Read/write.

### **Virtual Reality Toolbox Figure Frame Capture Preferences**

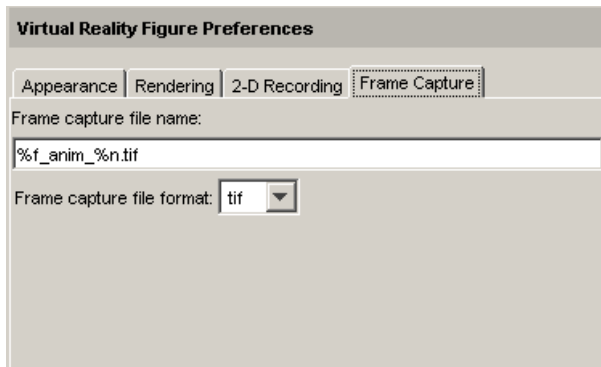
To access the virtual figure frame capture preferences:

- 1** From the MATLAB desktop, select **File > Preferences**.
- 2** In the left pane of the Preferences dialog box, select **Virtual Reality**.
- 3** In the left pane under **Virtual Reality**, select **Figure**.

The Virtual Reality Figure Preferences dialog appears in the right pane.

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

The Virtual Reality Figure Preferences dialog appears in the right pane, with the **Frame Capture** tab selected.



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

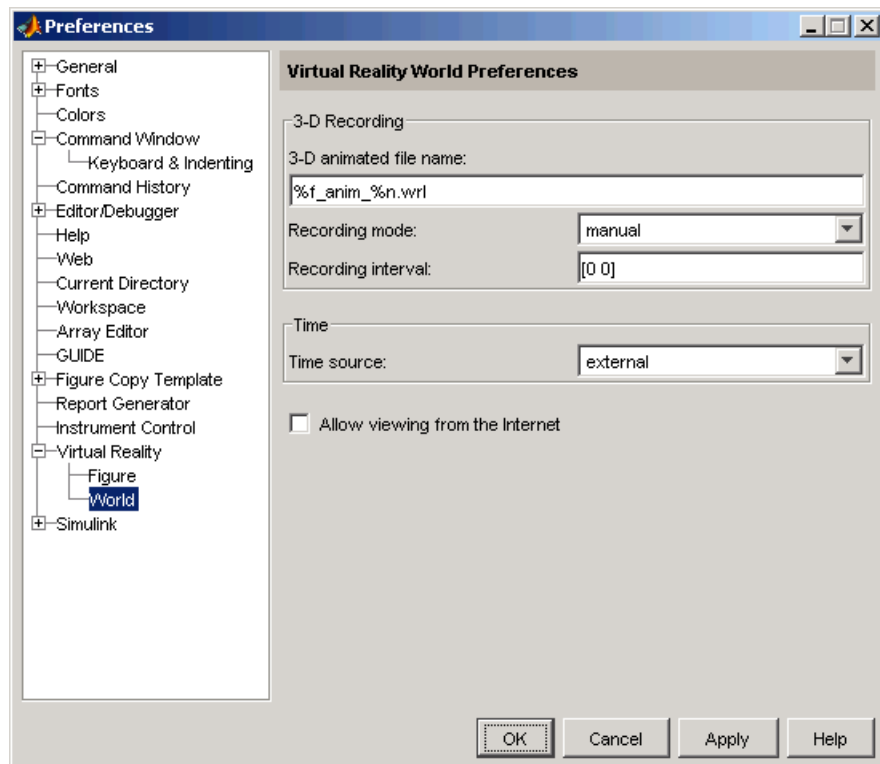
Property	Value	Description
CaptureFileFormat	'tif'   'png' Default: 'tif'	Specifies file format for a captured frame file. Read/write.
CaptureFileName	String. Default: '%f_anim_%n.ext'	Specifies the frame capture filename. The string can contain tokens that are replaced by the corresponding information when the animation recording takes place. For further details, see “Frame Capture and Animation Recording File Tokens” on page 6-18. Read/write.

## Virtual Reality Toolbox World Preferences

To access the virtual world preferences:

- 1 From the MATLAB desktop, select **File > Preferences**.
- 2 In the left pane of the Preferences dialog box, select **Virtual Reality**.
- 3 In the left pane under **Virtual Reality**, select **World**.

The Virtual Reality World Preferences dialog appears in the right pane.





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

Property	Value	Description
Allowing viewing from the Internet	'off'   'on' Default: 'off'	Remote access flag. If the virtual world is enabled for remote viewing, it is set to 'on'; otherwise, it is set to 'off'.
3-D animated file name	String. Default: '%f_anim_%n.wr1'	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.
Recording mode	'manual'   'scheduled' Default: 'manual'	Animation recording mode.
Recording interval	Vector of two doubles Default: [0 0]	Start and stop times for scheduled animation recording. Corresponds to the virtual world object Time property.
Time source	'external'   'freerun' Default: 'external'	Source of the time for the virtual world. If set to 'external', time in the scene is controlled from the MATLAB software (by setting the Time property) or the Simulink software (simulation time). If set to 'freerun', time in the scene advances independently based on the system timer.

## Removing Components (Windows)

In this section...
“Section Overview” on page 2-46
“Removing Virtual Reality Toolbox and Ligos V-Realm Builder (Microsoft Windows) Software” on page 2-46
“Removing the Blaxxun Contact Plug-In (Windows)” on page 2-47

### Section Overview

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

### Removing Virtual Reality Toolbox and Ligos V-Realm Builder (Microsoft Windows) Software

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

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

The MathWorks uninstaller begins running.

- 2 Select the **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 software removal.

---

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

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

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

## Installing on the Client Computer

In this section...
“Section Overview” on page 2-48
“Installing a VRML Plug-In (Windows)” on page 2-48

### Section Overview

In most configurations, you do not need to install a viewer on a client computer because you can perform all the tasks on a host computer. However, if you have very large models that consume considerable computational resources, you might want to use a client computer to run and view the virtual world.

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

“Installing a VRML Plug-In (Windows)” on page 2-48 describes how to install the Blaxxun Contact VRML plug-in on a computer running Windows.

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

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

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

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

## Testing the Installation

In this section...
“Section Overview” on page 2-49
“Running a Simulink Interface Example” on page 2-49
“Running a MATLAB Interface Example” on page 2-54

### Section Overview

The Virtual Reality Toolbox product 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 software, the VRML viewer, and the VRML editor.

### Running a Simulink Interface Example

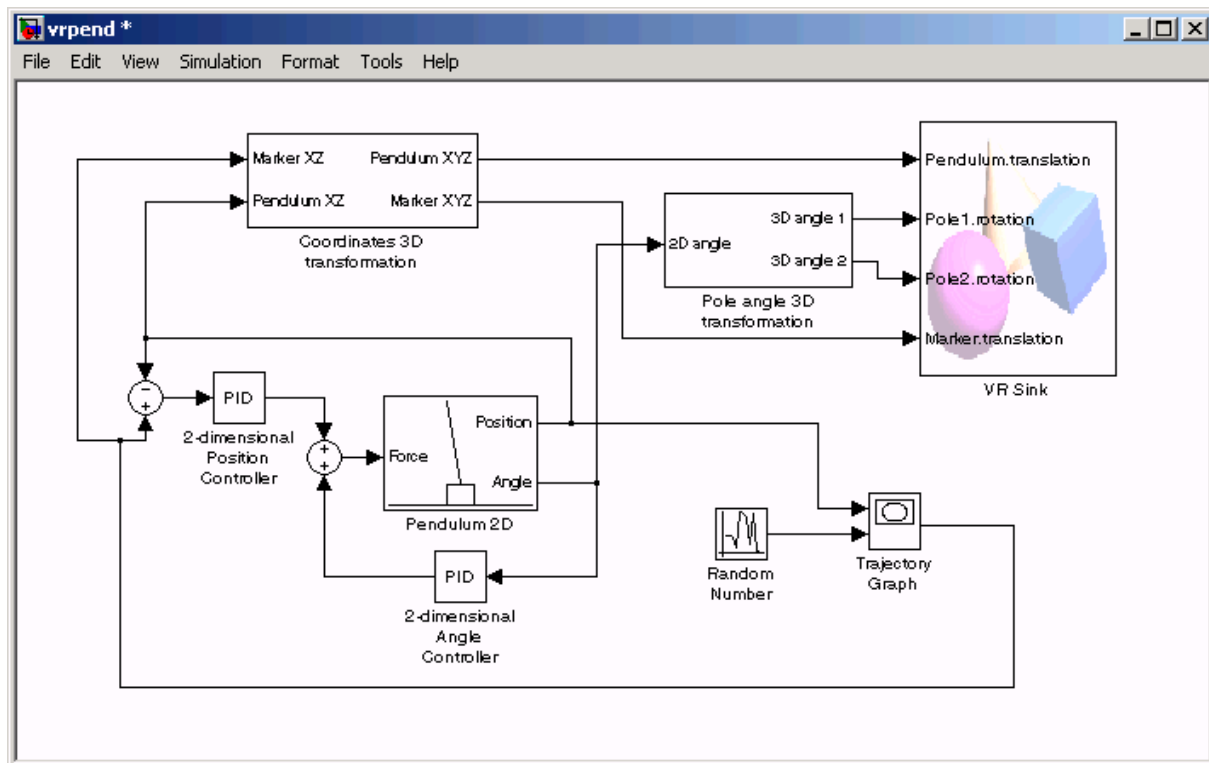
In the demo directory for the toolbox, there is a Simulink model for a two-dimensional inverted pendulum. This model, which you can view in three dimensions with the toolbox, has an interactive set point and trajectory graph.

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

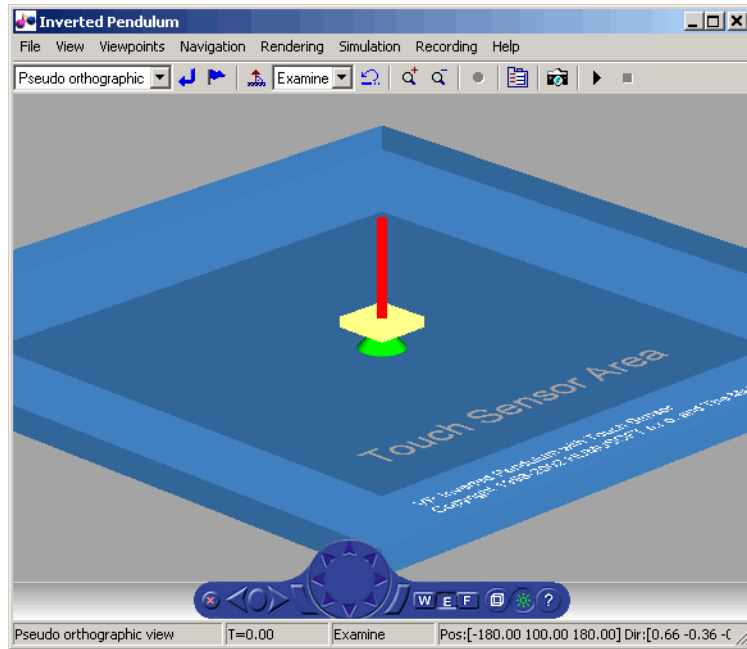
- 1 In the MATLAB Command Window, type

```
vrpend
```

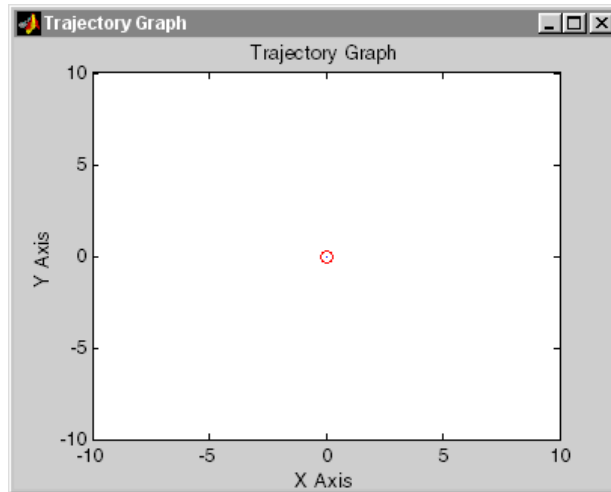
A Simulink window opens with the model for an inverted pendulum.



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.

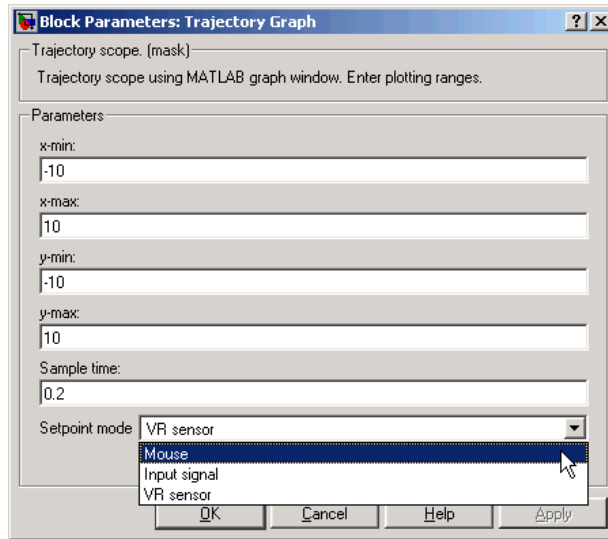
In the Virtual Reality Toolbox viewer, you see the animated movement of the pendulum. Use the viewer controls to navigate through the virtual world, change the viewpoints, and move the set point. For more information about using the Virtual Reality Toolbox viewer controls, see “Virtual Reality Toolbox 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**.





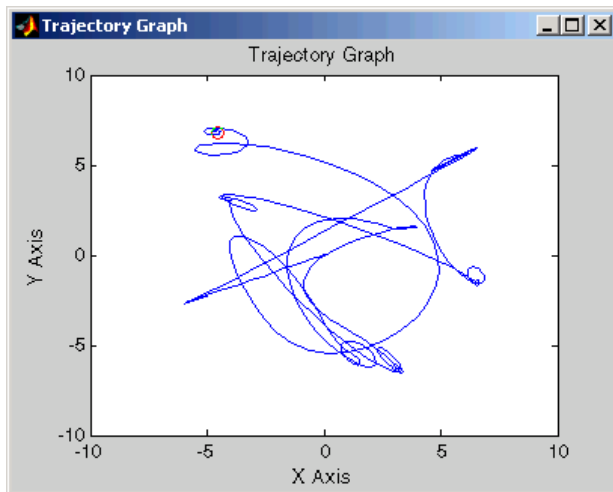
You can now use the trajectory graph as a 2-D input device to set the position of the pendulum.

- 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 1-17, 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 1-25.

**1** In the MATLAB window, type

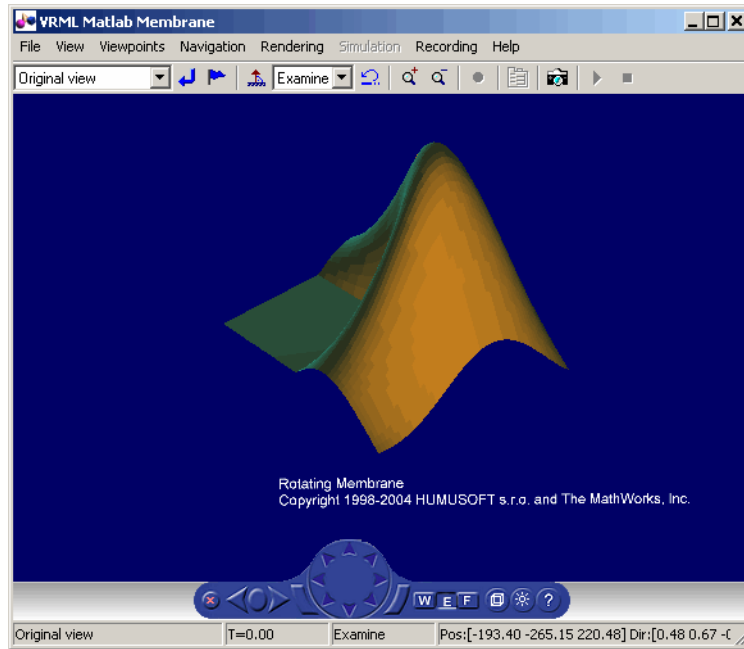
```
vrmemb
```

The MATLAB interface displays the following messages:

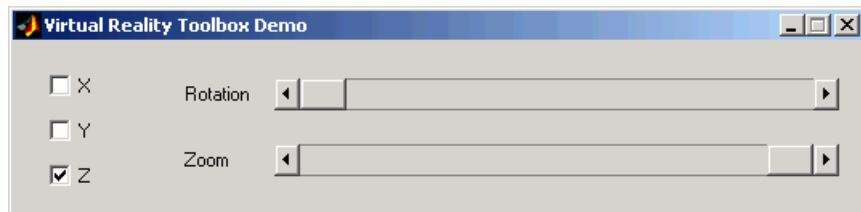
```
Loading... This example
shows you how to use a MATLAB generated 3-D graphic object
in the Virtual Reality Toolbox.
. . . Press Enter to start the demonstration.
```

- 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 product works with both the MATLAB and the Simulink products. 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 a Simulink Block” on page 3-2
- “Using the Simulink Interface” on page 3-12

## Associating a Virtual World with a Simulink Block

### In this section...

“Section Overview” on page 3-2

“Adding a Virtual Reality Toolbox Block” on page 3-2

“Changing the Virtual World Associated with a Simulink Block” on page 3-10

### Section Overview

With this 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.

### 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.

---

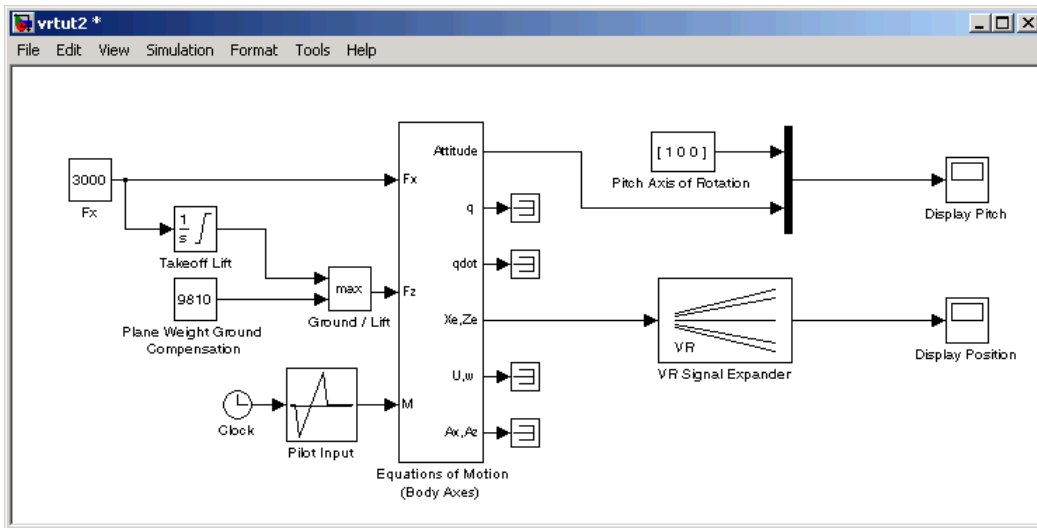
**Note** The examples in this topic are based on the Virtual Reality Toolbox default viewer. If you choose to use the blaxxun Contact VRML plug-in to view virtual worlds, you must start and stop the model simulation from the Simulink window. You cannot start and stop the model simulation from the blaxxun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



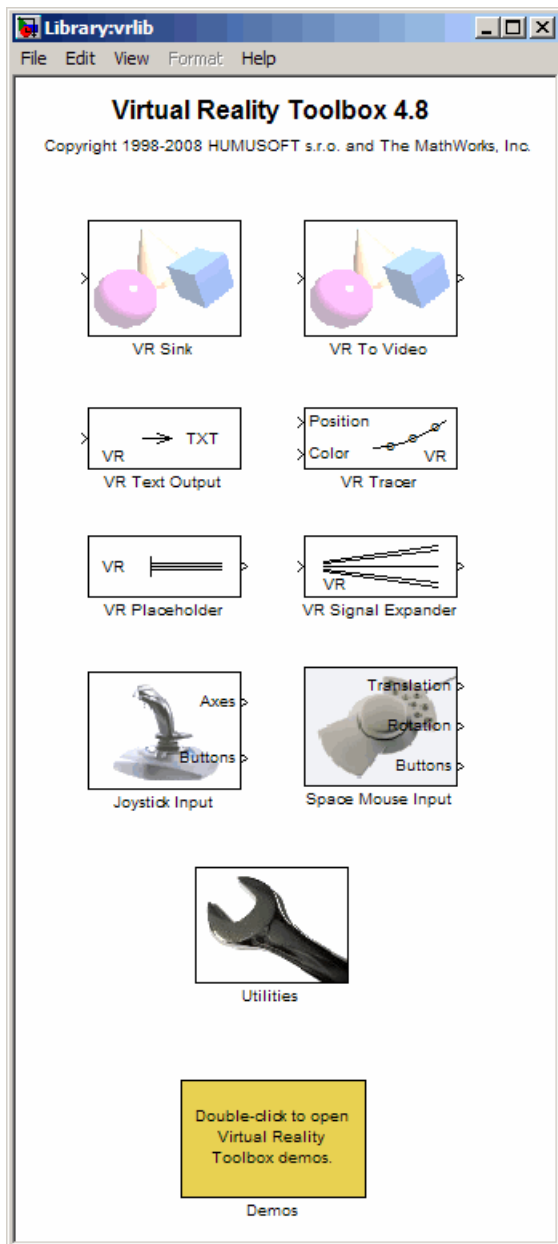
**2** From the **Simulation** menu, select **Normal**, then click **Start**.

Observe the results of the simulation in the scope windows.

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

```
vrlib
```

The Virtual Reality Toolbox library opens.



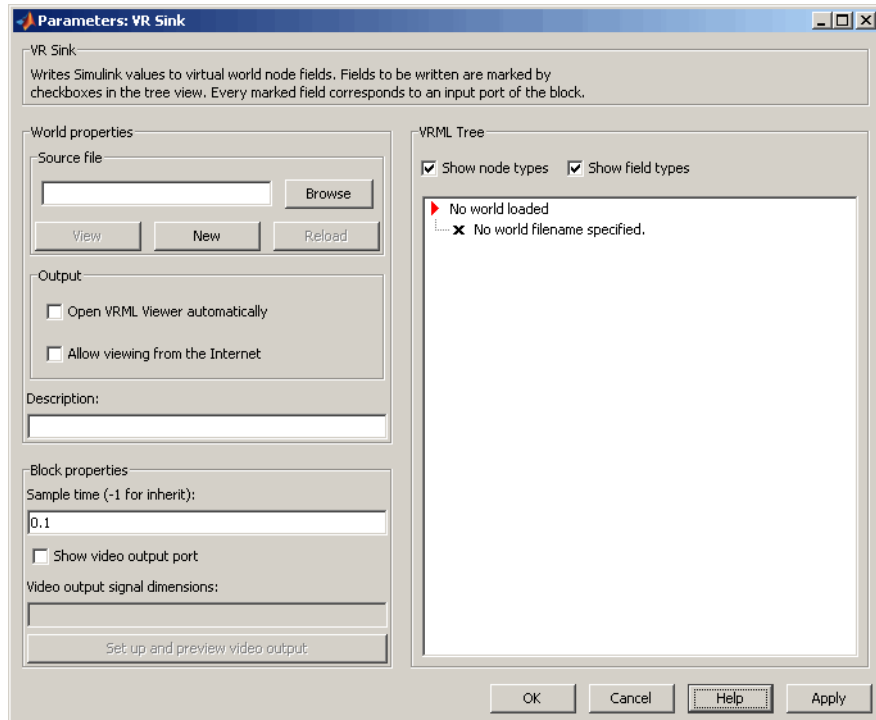


- 4** From the **Library** window, drag and drop the VR Sink block to the Simulink diagram. The VR Sink block writes data from the Simulink model to the virtual world. (For a description of all the Simulink blocks in the Virtual Reality Toolbox library, see Chapter 8, “Block Reference”) You can then 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 `vrkoff.wrl`, located in the `vr demos` 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** At the **Source File** text box, click the **Browse** button. The Select World dialog box opens. Find the directory <matlab root>\toolbox\vr\vr demos. Select the file vr tkoff.wrl and click **Open**.

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

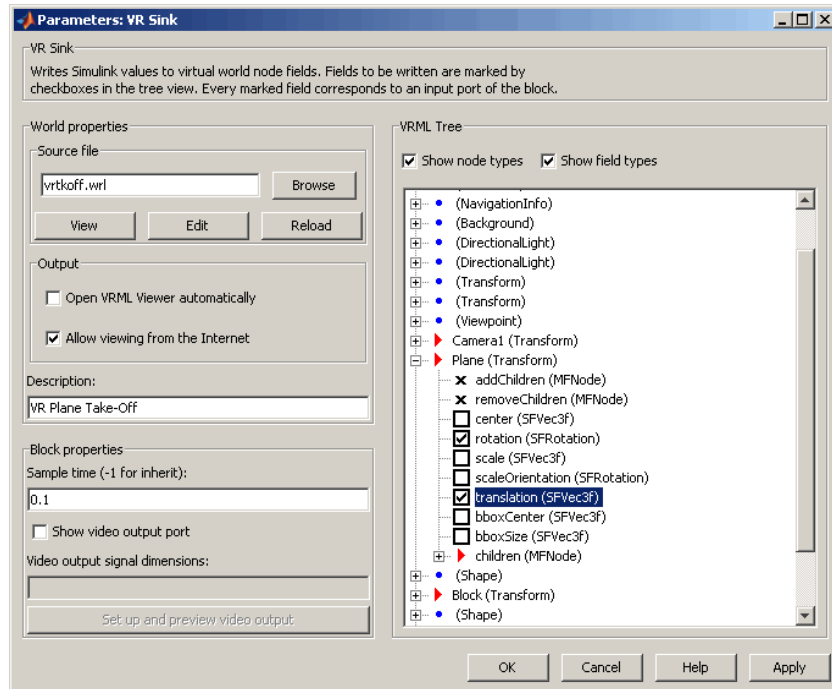
A VRML tree appears on the right side, showing the structure of the associated virtual reality scene.

**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 the Simulink interface. 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.



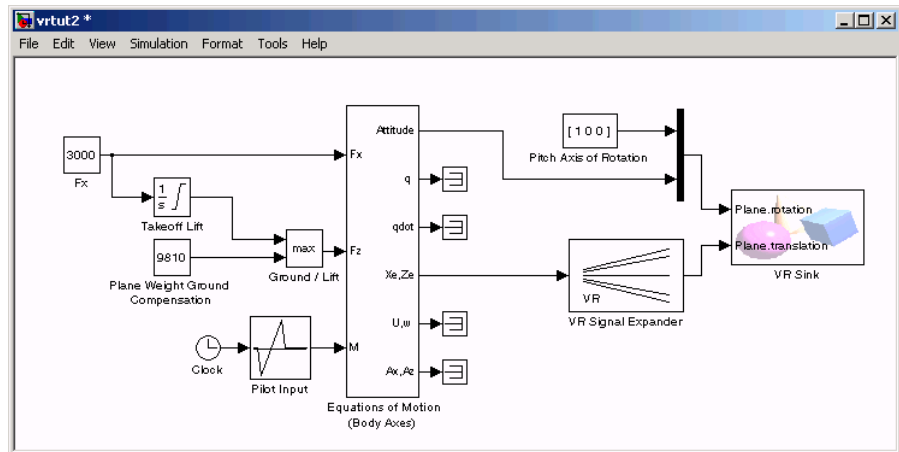
The first input is Plane rotation. The rotation is defined by a four-element vector. The first three numbers define the axis of rotation. In this example, it should be  $[1 \ 0 \ 0]$  for the  $x$ -axis (see the Pitch Axis of Rotation block in the model). The pitch of the plane is expressed by the rotation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

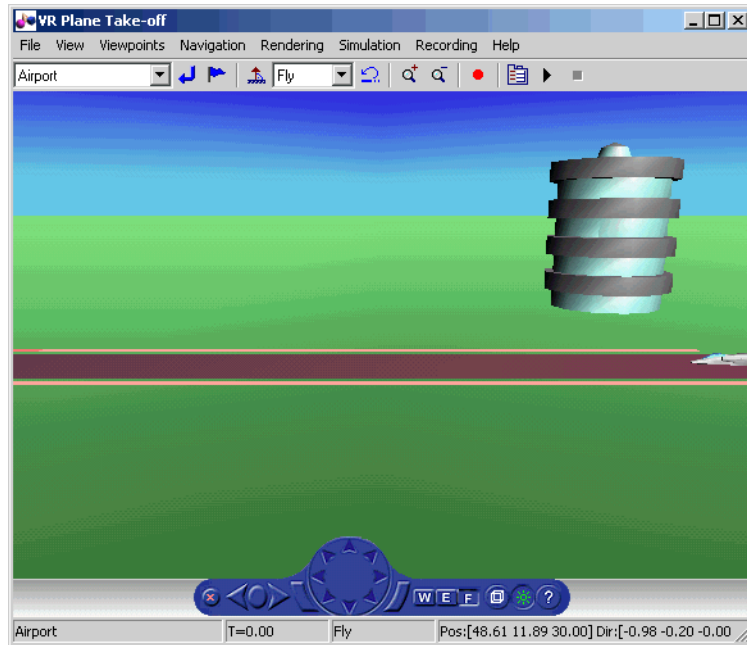
- 13 In the Simulink model, connect the line going to the Scope block labeled Display Position to the Plane translation input.

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



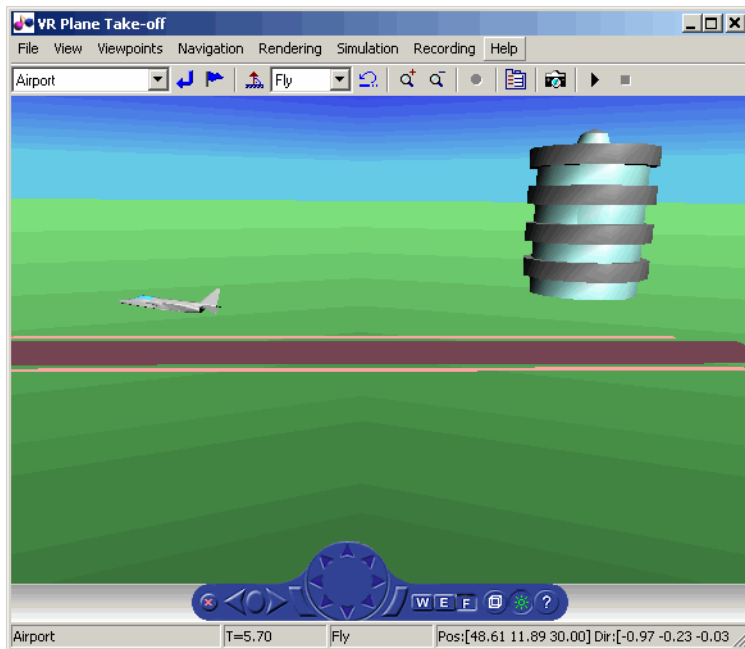
**Note** Virtual world degrees of freedom have different requested input vector sizes depending on the associated VRML field types. If the vector size of the connected signal does not match the associated VRML field size, an Incorrect input vector size error is reported when you start the simulation.

- 14 Double-click the VR Sink block in the Simulink model. 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 `vrut2` 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 opens.
- 2 Select the **Simulation** menu **Block Parameters** option. The Parameters: VR Sink dialog box opens.
- 3 At the **Source File** text box, click the **Browse** button. The Select World dialog box opens. Find the directory `<matlab root>\toolbox\vr\vr demos`. Select the file `vrtkoff2.wrl`, and click **Open**.

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

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

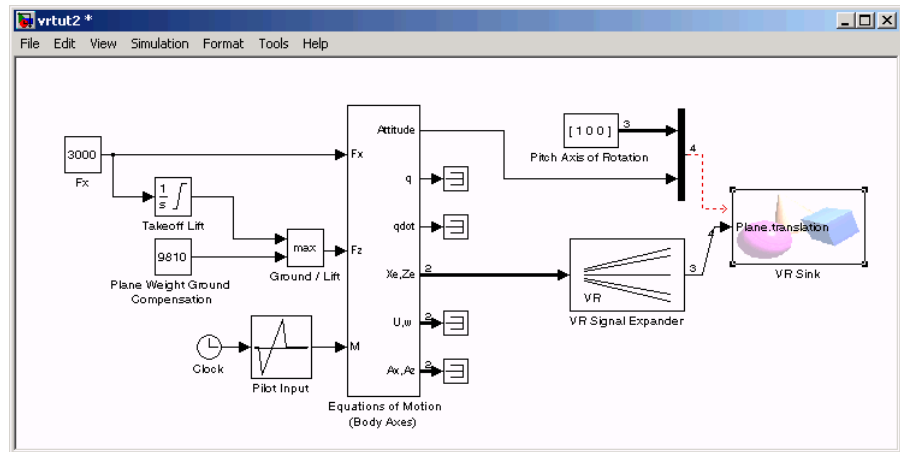
- 5** On the left of the Plane (Transform) node, click the + square.

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

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

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

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



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

## Using the Simulink Interface

In this section...
“Section Overview” on page 3-12
“Displaying a Virtual World and Starting Simulation” on page 3-12
“Viewing a Virtual World with a Web Browser on the Host Computer” on page 3-15
“Viewing a Virtual World with a Web Browser on the Client Computer” on page 3-19

### Section Overview

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

### Displaying a Virtual World and Starting Simulation

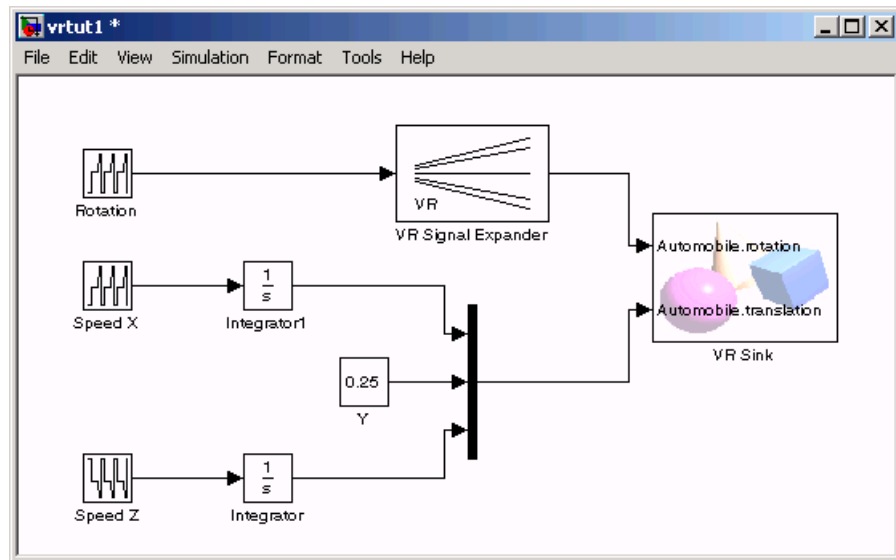
This example explains how to display a simulated virtual world using the Virtual Reality Toolbox viewer on your host computer. This is the default and recommended method for viewing virtual worlds. A Simulink window opens with the model of a simple automobile. Automobile trajectory (vehicle position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

- 1 In the MATLAB Command Window, type

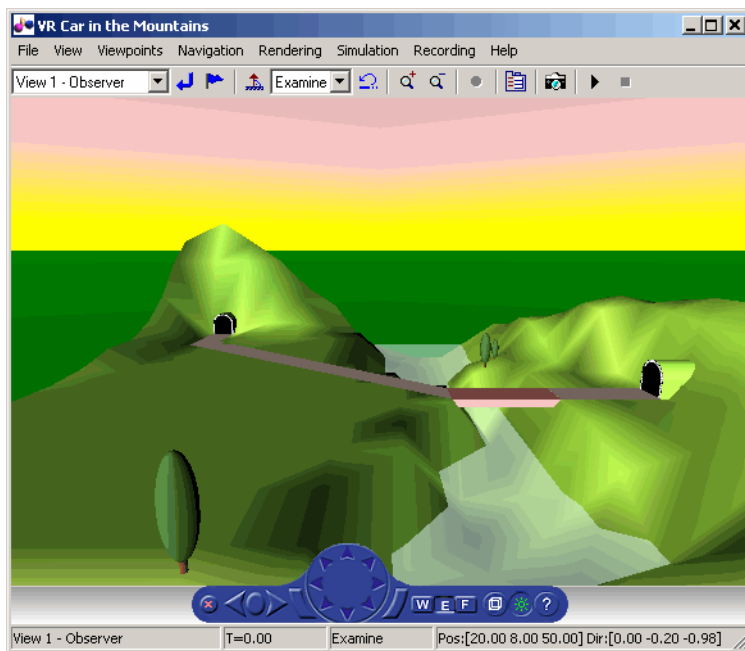
```
vrtut1
```

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 2-22.

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 2-22.

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.

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

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

The `vrworld` object uses the `title` string in the VRML file for the `Description` property of the `vrworld` object. You can change this property with the Virtual Reality Toolbox MATLAB interface (`vrworld/set`).

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
```

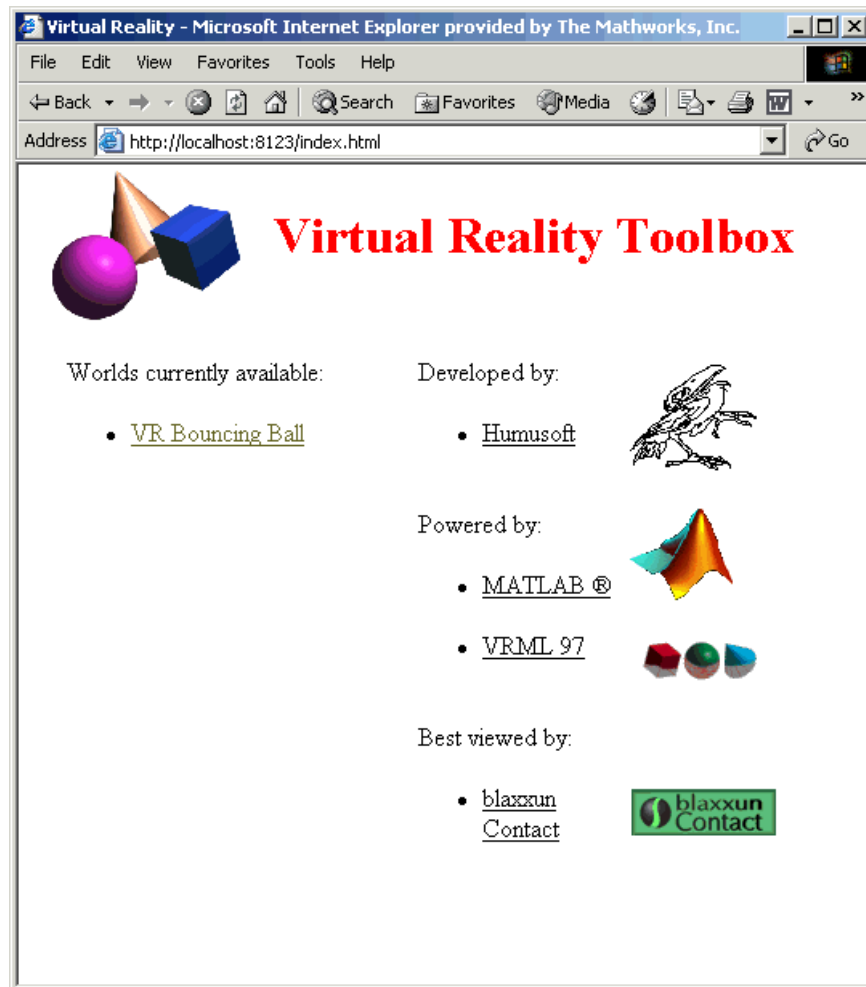
---

**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.

---

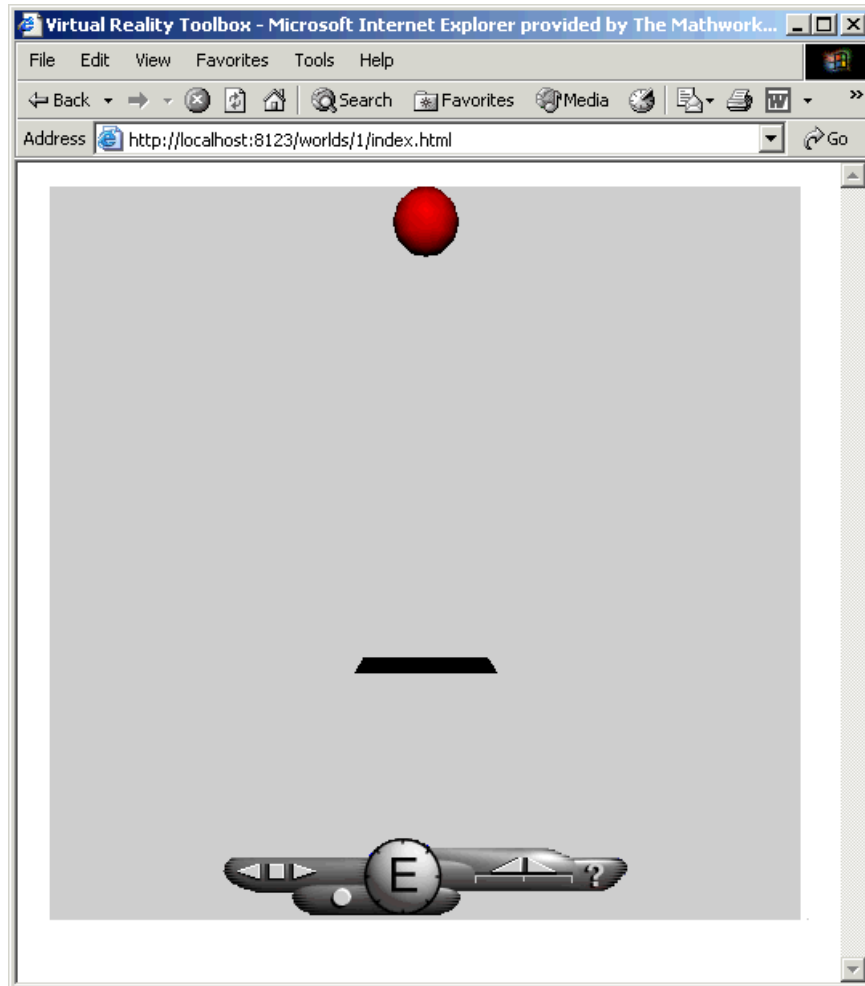
The following page is loaded and becomes active.

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



### 3 Click VR Bouncing Ball.

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



From the main HTML page, you can select one of the listed available worlds or click the **reload** link to update the status of the virtual worlds supported by the 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 product.

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

The Virtual Reality Toolbox software allows you to simulate a process on a host computer while running the visualization of the process on a client computer. You view the virtual world on the client computer using a Web browser. This client computer is connected to the host computer through a network using the TCP/IP protocol. This means you need to know the name or IP address of the host computer you want to access from the client computer.

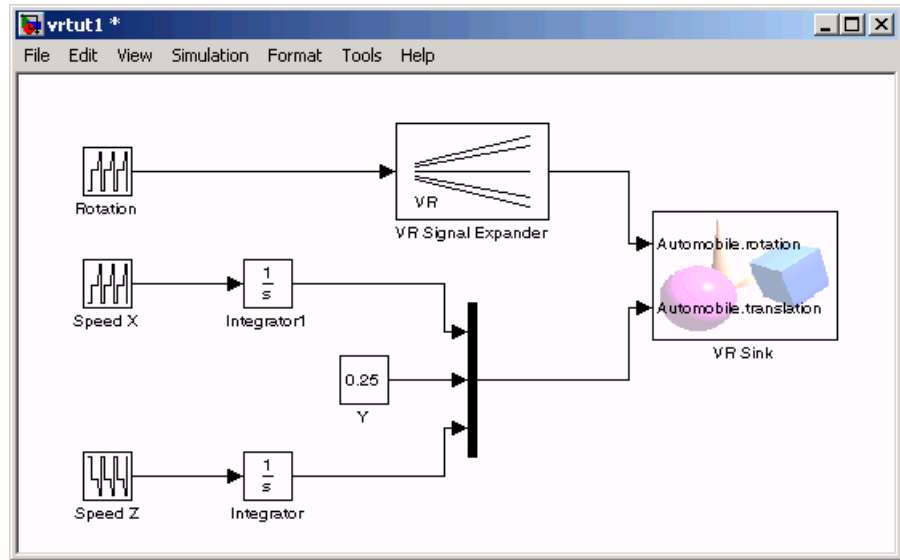
Viewing a virtual world on a client computer might be useful for remote computing, presentation of the results over the Web, or in situations where it is desirable to distribute computing and graphical power.

This example explains how to display a simulated virtual world on a client computer. In this case, the client computer is a PC platform with the blaxxun Contact plug-in. 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.



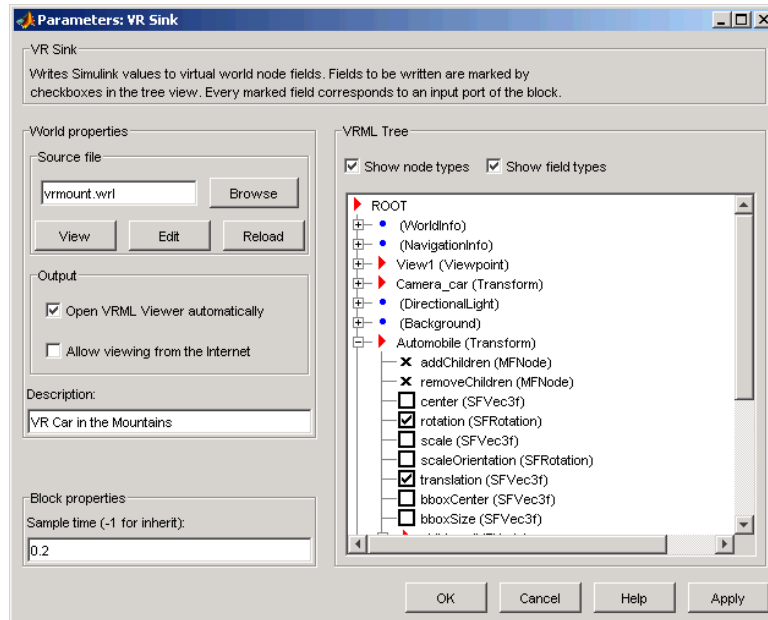
- 2 Double-click the VR Sink block. This block is in the right part of the model window.

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 the Simulink software. For example, if the IP address of the host computer is 192.168.0.1, enter

`http://192.168.0.1:8123`

To determine your IP address on a Windows system,

Click **Start**. Click **Run**. Type `cmd`, and enter `ipconfig`.

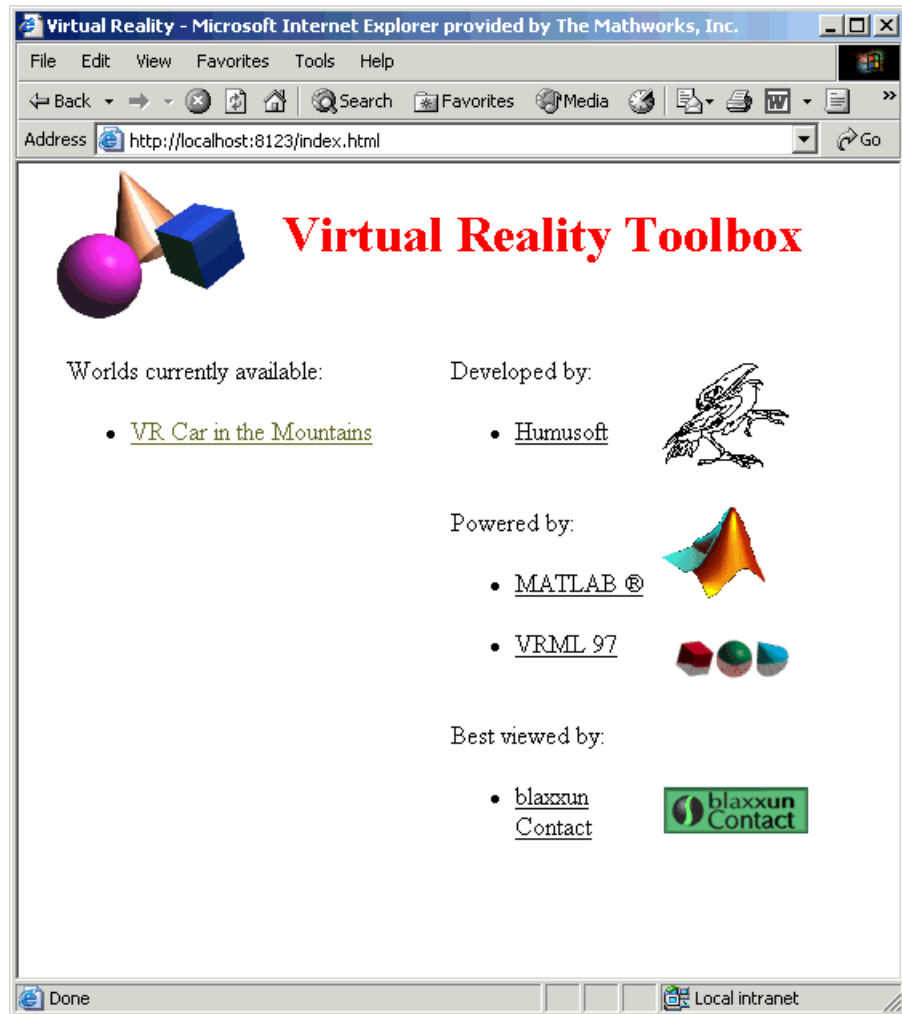
To determine your IP address on a UNIX system, type the command

```
ifconfig device_name
```

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

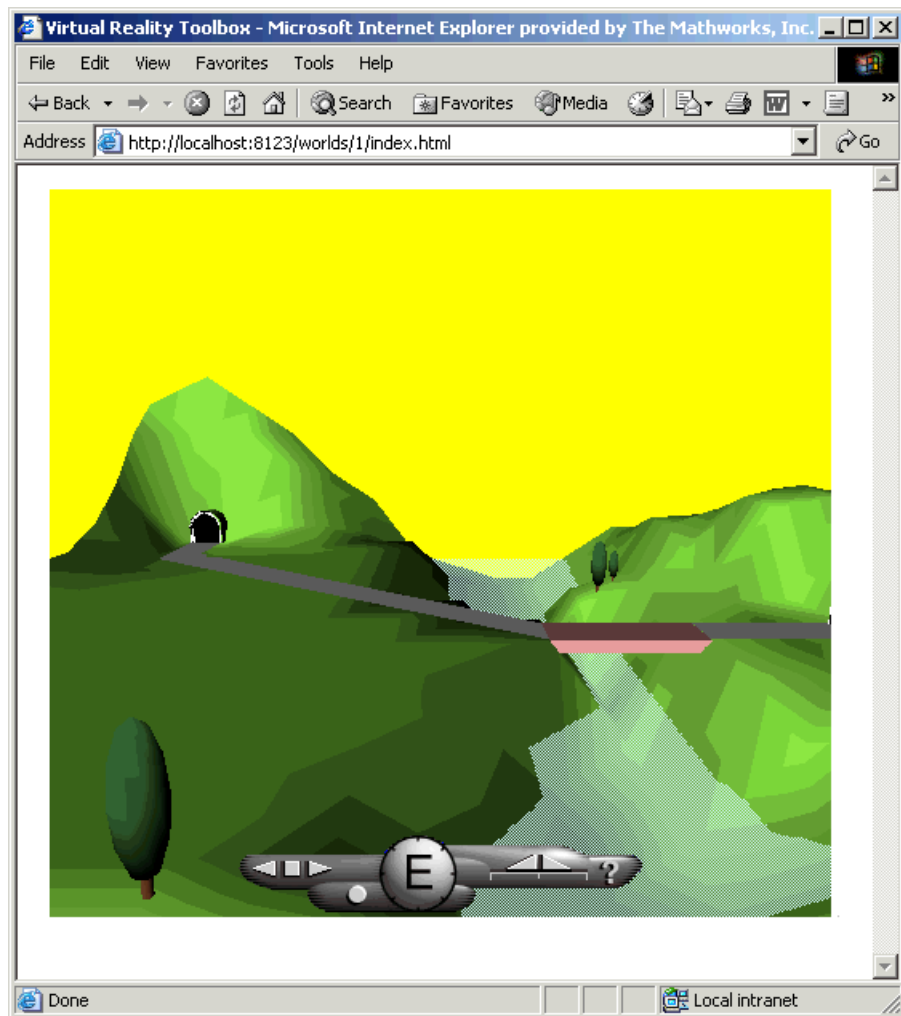
Alternatively, for Windows platforms, you can open a DOS shell and type `ipconfig`.

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



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

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



- 8 On the host computer, in the Simulink window, from the **Simulation** menu, click **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 software 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” on page 4-2
- “Recording Offline Animations” on page 4-10

## Using the MATLAB Interface

In this section...
“Creating a vrworld Object” on page 4-2
“Opening a Virtual World” on page 4-3
“Interacting with a Virtual World” on page 4-5
“Closing and Deleting a vrworld Object” on page 4-9

### 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 `.wr1`.

---

**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.wr1` as an example.



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

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

The MATLAB Command Window displays output like

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

**2** Type

```
vrwhos
```

The MATLAB Command Window 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);
```

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

- 2** Type

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

The `Description` property is changed to `My first virtual world`. This is the description that is displayed in all 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 2-22.

## 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 10-2 and “`vrnode` Object Methods” on page 10-3.

After you open a `vrworld` object, you can get a list of available nodes in the virtual world. This procedure uses the `vrworld` object `myworld` and the virtual world `vrmount.wrl` as an example:

- 1** In the MATLAB Command Window, type

```
nodes(myworld);
```

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

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

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

## 2 Type

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

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

```
mynodes =
```

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

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

**3** Type

```
whos
```

The MATLAB Command Window displays the messages

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

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

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

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

or

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

The MATLAB Command Window displays information like the following table.

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

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

**5** Type

```
view(myworld)
```

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

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

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

The MATLAB sets a new position for the `Automobile` node, and you can observe that the car is repositioned in the VRML browser window.

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

---

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

---

## 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

### In this section...

“Overview” on page 4-10

“Animation Recording File Tokens” on page 4-12

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

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

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

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

“Viewing Animation Files” on page 4-25

“MATLAB Animation Recording of Virtual Worlds Not Associated with Simulink Models” on page 4-27

### Overview

The toolbox enables you to record animations of virtual scenes that are controlled by the Simulink or MATLAB product. 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 the MATLAB, Simulink, or Virtual Reality Toolbox products. 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 software 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 software does not save any of these movements.

---

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

---

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

---

**Note** While recording 2-D `.avi` animation data, always ensure that the Virtual Reality Toolbox viewer is the topmost window and fully visible. Graphics acceleration limitations might prevent the proper recording of 2-D animation otherwise.

---

## Animation Recording File Tokens

By default, the Virtual Reality Toolbox software records animations in a file named according to the following format:

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

This format creates a unique filename each time you record the animation. The Virtual Reality Toolbox software 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`.

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

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

Token	Description
%d	The full path to the world VRML file replaces this token in the filename string and creates files in directories relative to the virtual world file location. For example, the format <code>%d/animdir/animfile.avi</code> saves the animation into the <code>animdir</code> 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 <code>%f_anim_%D.wrl</code> saves the animation to <code>vrplanets_anim_29.wrl</code> 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.
%S	The current second replaces this token in the filename string. For example, the format %f_anim_%h%m%S.wrl saves the animation to vrplanets_anim_150430.wrl for a start record time of 15:04:30.
%Y	The current four-digit year replaces this token in the filename string. For example, the format %f_anim_%Y.wrl saves the animation to vrplanets_anim_2005.wrl for the year 2005.

## Manual 3-D VRML Animation Recording

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

This procedure uses the `vrplanets` 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 appears. Also by default, the Virtual Reality Toolbox viewer for that model is loaded and becomes active. If the viewer does not appear, 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 software manually record the animation, set the `RecordMode` property to `manual`. Type

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

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

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

- 5** Run the Simulink model. From the **Simulation** menu, select **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 software stops recording the animation. The Virtual Reality Toolbox software 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 appears. Also by default, the Virtual Reality Toolbox viewer for that model is loaded and becomes active. If the viewer does not appear, 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 toolbox manually record the animation, set the `RecordMode` property to `manual`. Type

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

- 6** Direct the Virtual Reality Toolbox software to record the animation as a .avi format file. Type

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

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

```
set(f, '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 software stops recording the animation. The Virtual Reality Toolbox software 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 software records the animation during this interval in the simulation. In this example, the timing of the recorded animation file derives from the simulation time. One second of the recorded animation time corresponds to one second of Simulink time.

This procedure uses the `vrplanets` 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 Direct the Virtual Reality Toolbox software 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 software to record the animation in a VRML format file.

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

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

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

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

- 7 Run the Simulink model. From the **Simulation** menu, select **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 software 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 software records the animation during this interval in the simulation. In this example, the timing of the recorded animation file derives from the simulation time. One second of the recorded animation time corresponds to one second of Simulink time.

This procedure uses the `vrplanets` 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 the Virtual Reality Toolbox software manually record the animation, set the `RecordMode` property to `manual`. Type

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

- 5** Direct the Virtual Reality Toolbox software to record the animation as an `.avi` format file. Type

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

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

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

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

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

```
set(f, '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. The Virtual Reality Toolbox software 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 4-20 for descriptions of how to create one.

### Viewing VRML Files

At the MATLAB window, type `vrplay(filename)`, where `filename` is the name of your VRML file. This opens the Virtual Reality animation player and your file. Using the Virtual Reality animation player GUI, you can control the playback of your file.

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

`vrplay` works only with VRML animation files created using the Virtual Reality Toolbox VRML recording functionality.

## Other Methods for Viewing VRML Files

**1** Alternatively, you can view the VRML file in one of the following ways:

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

**2** To stop the animation, type

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

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

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

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

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



## Viewing 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 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,
  - 1** Retrieve the corresponding `vrfigure` object. For 2-D animations, the Virtual Reality Toolbox software 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.
  - 2** Set the `Record2D` `vrfigure` property.
  - 3** To override default filenames for animation files, set the `vrfigure` `Record2DFilename` property.
- To create 3-D animation files,
  - 1** Retrieve the corresponding `vrworld` object.
  - 2** Set the `Record3D` `vrworld` property.
  - 3** To override default filenames for animation files, set the `vrworld` `Record3DFilename` property.
- Set the `RecordMode` `vrworld` object property to `manual` or `scheduled`. For optimal results, select `scheduled`.
- If you select `scheduled` for `RecordMode`, be sure to also set the `vrworld` `RecordInterval` property to a desired time interval.
- To specify that the virtual world time source is an external one, set the `vrworld` property `TimeSource` to `external`. This ensures that the MATLAB software controls the virtual world scene time. Type

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

- To specify time values at which you want to save animation frames, iteratively set the `vrworld` `Time` property. Note that for a smoother animation, you should set the time at equal intervals, for example, every 5 seconds. Use a sequence like

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

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

```

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

```

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

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

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

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

```

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

```

If you set the `Time` property at or outside the end boundary of `RecordInterval`, the Virtual Reality Toolbox software stops recording. You can then view the resulting animation file.

For a complete example of how to perform this kind of animation recording, refer to the Virtual Reality Toolbox `vrheat_anim` demo.

# Virtual Worlds

---

The Virtual Reality Toolbox product includes tools that you can use to edit and create VRML virtual worlds. For Windows platforms, the toolbox includes a VRML editor (Ligos 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” on page 5-2
- “Deformation of a Sphere Example” on page 5-5
- “VRML Data Types” on page 5-21
- “Using CAD Models with the Virtual Reality Toolbox Product” on page 5-26

## VRML Editing Tools

In this section...
“Section Overview” on page 5-2
“Editors for Virtual Worlds” on page 5-2
“Ligos V-Realm Builder” on page 5-4

### Section Overview

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.

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. The Ligos V-Realm Builder interface is one of the exceptions. It is one of the most advanced VRML editing tools currently available for personal computers. The Ligos V-Realm Builder application is available only for Windows operating systems. You can access Ligos V-Realm Builder documentation at:

[http://www.mathworks.com/support/solutions/files/s35543/-VRealm\\_Man.zip](http://www.mathworks.com/support/solutions/files/s35543/-VRealm_Man.zip)

For PCs, the Virtual Reality Toolbox software includes the Ligos V-Realm Builder application as a native 3-D editor. For more information, see “Ligos

V-Realm Builder” on page 5-4 and “Deformation of a Sphere Example” on page 5-5.

## **Ligos V-Realm Builder**

The Ligos V-Realm Builder application 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. The Ligos V-Realm Builder interface 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 the Virtual Reality Toolbox product.



## Deformation of a Sphere Example

### In this section...

“Section Overview” on page 5-5

“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 5-9

“Creating a Box in a Virtual World” on page 5-14

“Connecting a Simulink Model to a Virtual World” on page 5-17

### Section Overview

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

This example assumes you finished the installation of the Ligos V-Realm Builder software using the function `vrinstall`. See “Installing the VRML Editor (Windows)” on page 2-27.

### 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.

<b>Object</b>	<b>Center Position</b>	<b>Dimensions</b>
B1	[3 0 0]	[0.3 1 1]
B2	[-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.

The Virtual Reality Toolbox product 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**

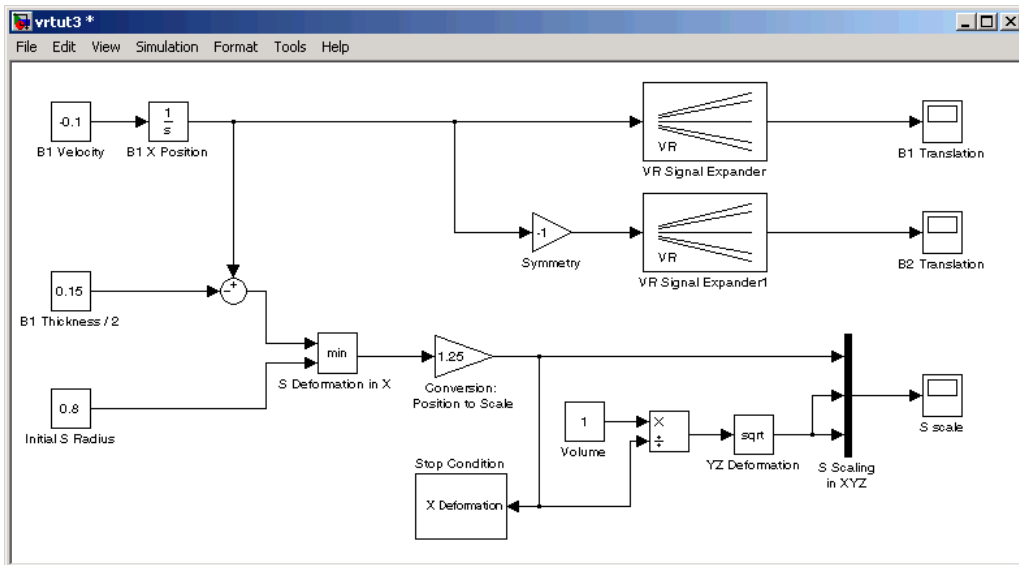
This procedure uses the Simulink model `vrtut3.mdl` as an example to explain how to add a Virtual Reality Toolbox block to your model. The model generates the values for the position of B1, the position of B2, and the dimensions of S for the problem previously defined. See “Defining the Problem” on page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

- 1** From the directory `C:\matlabroot\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
```

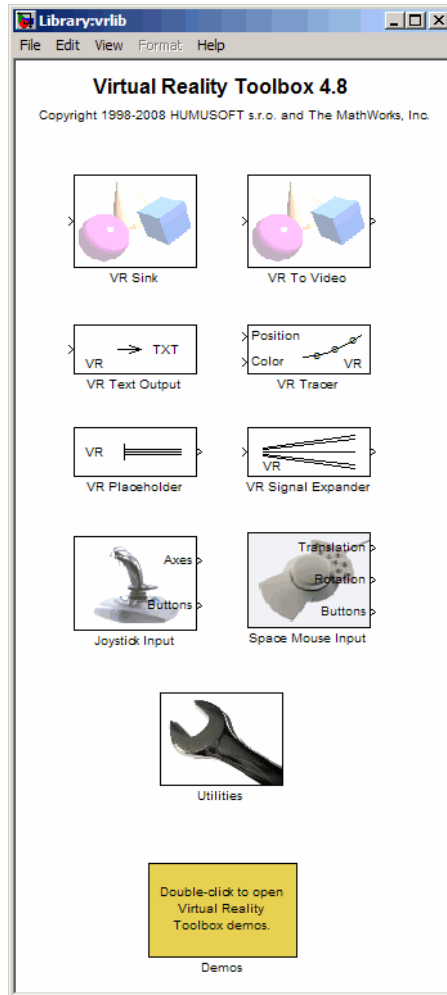
A Simulink window opens with a model that contains Virtual Reality Toolbox VR Signal Expander blocks, but no VR Sink block to write data from the model to Virtual Reality Toolbox. Instead, this model uses Scope blocks to temporarily monitor the relevant signals.



**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 5-9.

## 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 the Ligos V-Realm Builder software. This procedure uses the model `vrtut3.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 Windows task bar, click **Start**, and then click **Run**.


**2** In the Run dialog box, enter

```
matlabroot\toolbox\vr\vrealm\program\vrbuild2.exe
```

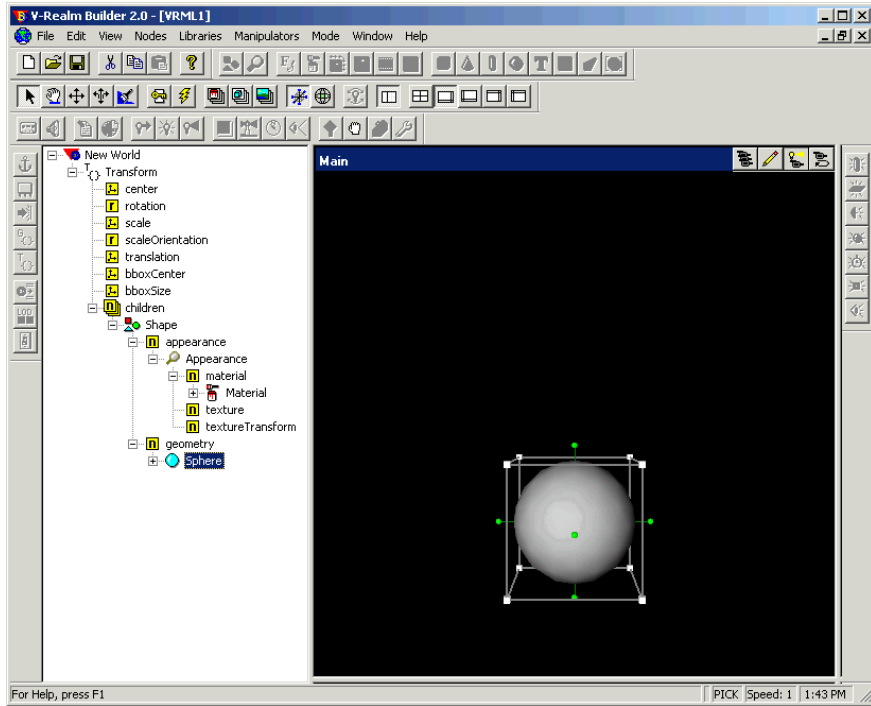
The Ligos V-Realm Builder application window opens.

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

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

**4** On the toolbar, click the sphere icon .

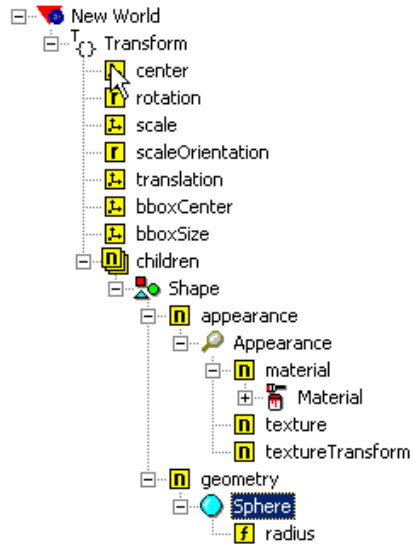
In the left pane you can see the VRML tree for a sphere. This tree includes the nodes **Transform**, **Shape**, **Appearance**, **Material**, and **Sphere**. A yellow icon indicates the field of a node.



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.

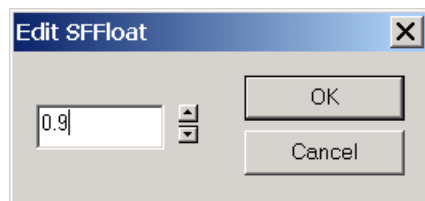
The radius field appears. The yellow icon indicates the type of value. In this case, **f** indicates a value with the type SFFloat. SFFloat is a 32-bit floating-point value.



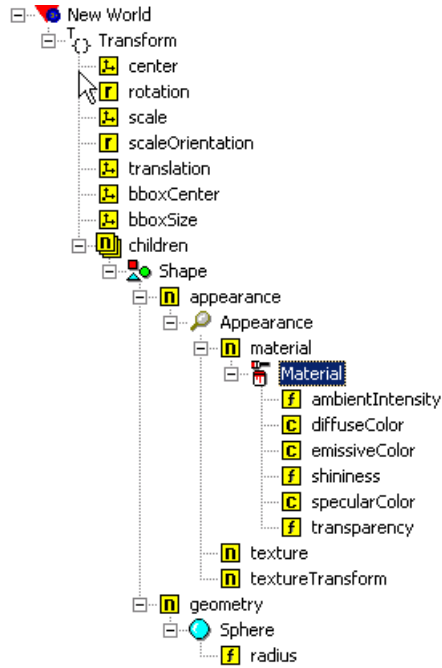
- 6 Double-click the `radius` field.

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.

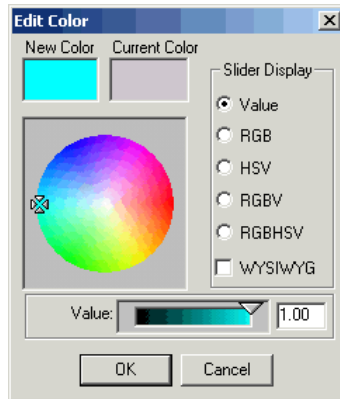


- 9 Under the Material node, double-click the diffuseColor field.

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 software requires you to assign a name to this node so that it can control access to and its properties. In VRML syntax, the named nodes are indicated by the "DEF Name Node" statement. The Ligos V-Realm Builder interface 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 5-14.

## Creating a Box in a Virtual World

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

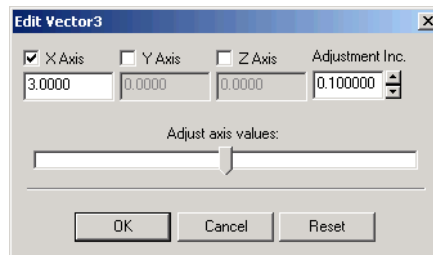
- 1 In the tree, click New World (the topmost item). On the toolbar, click the box icon .

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.

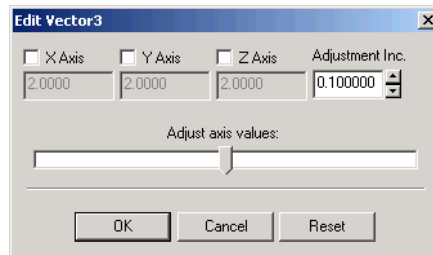
- 3 Select the **X Axis** check box. In the text box below, enter **3**, then click **OK**.



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.

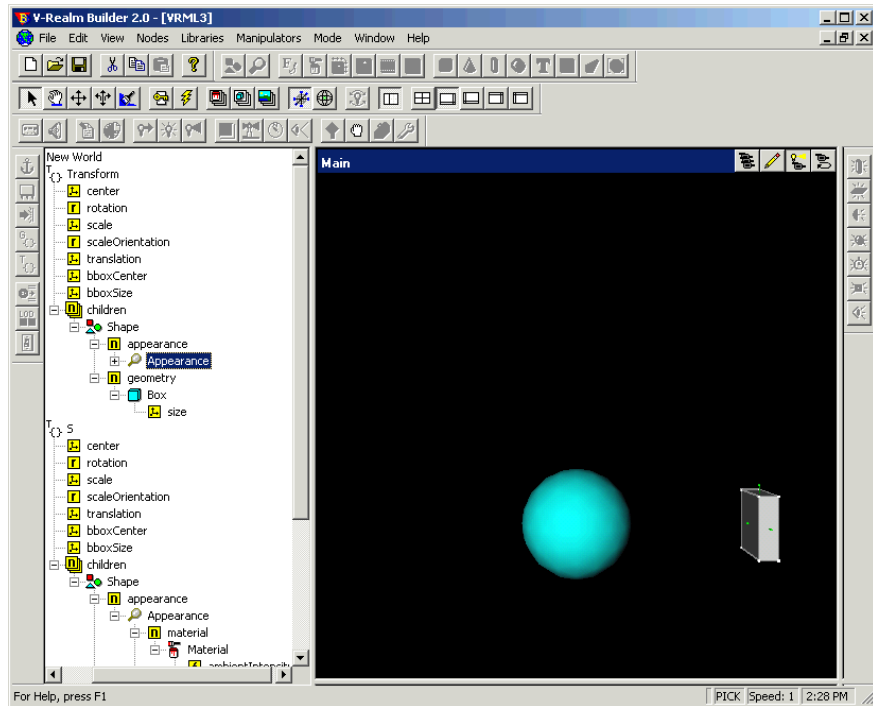
The Edit Vector 3 dialog box opens.



- 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.

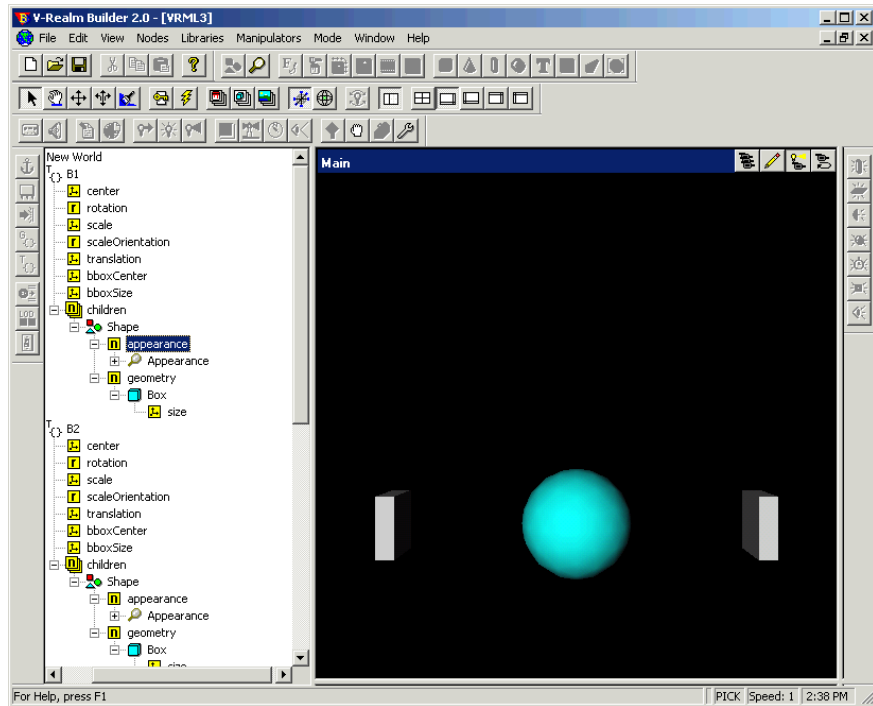
The following figure shows the tree with the expanded branch of the box.



- 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 software 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 the Ligos V-Realm Builder application.

---

**Caution** If you want to use your virtual worlds with the Virtual Reality Toolbox software, 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 5-17.

## 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 appears.

- 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.

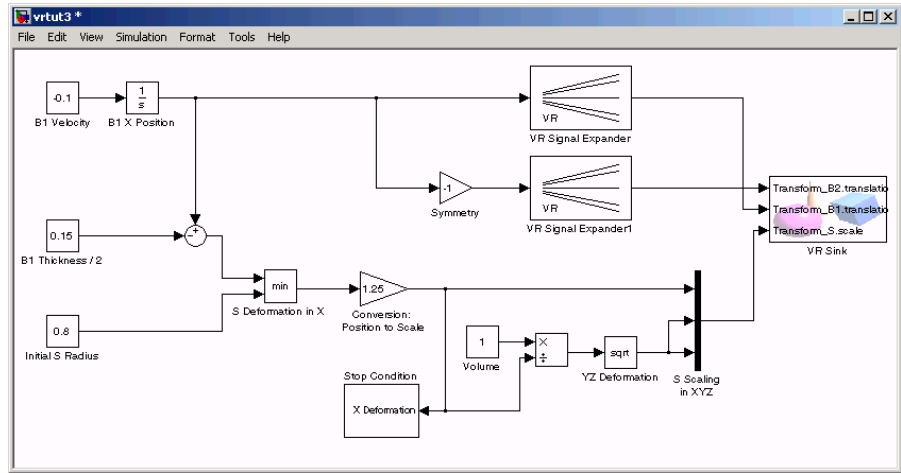
- 6** In the **Description** field, type `vrtut3`.

- 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.



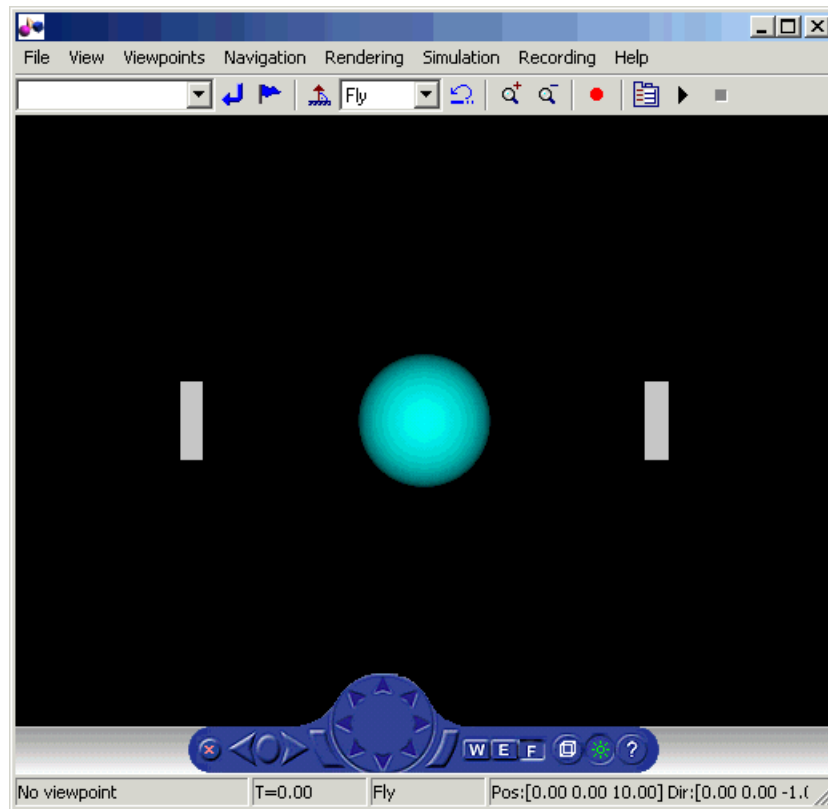
- 10 Double-click the VR Sink block.

The viewer appears.

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

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

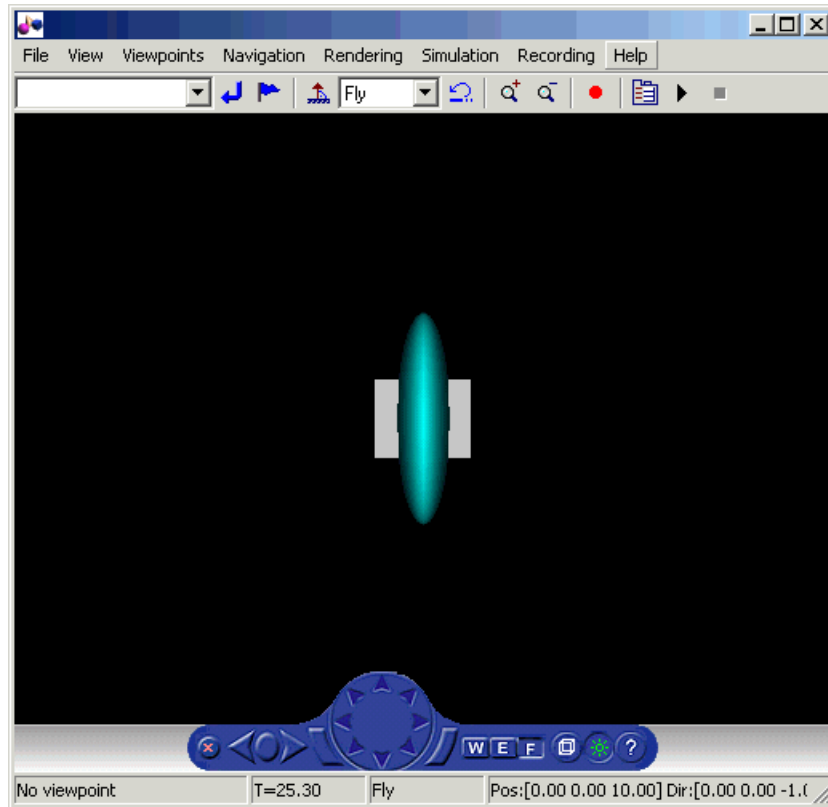
Your default viewer opens and displays the virtual world. For more information on changing your default viewer, see “Setting the Default Viewer of Virtual Scenes” on page 2-22.



**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

### In this section...

“Section Overview” on page 5-21

“VRML Field Data Types” on page 5-21

“VRML Data Class Types” on page 5-24

### Section Overview

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

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

### VRML Field Data Types

The Virtual Reality Toolbox product provides an interface between the MATLAB and Simulink environment and VRML scenes. With this interface, you can set and get the VRML scene node field values. To work with these values, you must understand the relationship between VRML data types and the corresponding MATLAB data types. The following table illustrates the VRML data types and how they are converted to and from MATLAB types.

For a detailed description of the VRML fields, refer to the VRML97 Standard.

VRML Type	Description	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

<b>VRML Type</b>	<b>Description</b>	<b>VR Toolbox Type</b>
SFVec2f	Vector of two floating-point values that you usually use for 2-D coordinates. For example, texture coordinates.	Single array (1-by-2)
SFVec3f	Vector of three floating-point values that you usually use for 3-D coordinates.	Single array (1-by-3)
SFColor	Vector of three floating-point values you use for RGB color specification.	Single array (1-by-3)
SFRotation	Vector of four floating-point values you use for specifying rotation coordinates ( $x, y, z$ ) of an axis plus rotation angle around that axis.	Single array (1-by-4)
SFImage	Two-dimensional array represented by a sequence of floating-point numbers.	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)
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)

<b>VRML Type</b>	<b>Description</b>	<b>VR Toolbox Type</b>
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

The Virtual Reality Toolbox software 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 as follows:

<b>Property</b>	<b>Description</b>
<code>DataTypeBool</code>	Specifies the boolean data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are 'logical' and 'char'. If set to 'logical', the VRML boolean data type is returned as a logical value. If set to 'char', the VRML boolean data type is returned 'on' or 'off'.

Property	Description
<code>DataTypeInt32</code>	Specifies the <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 VRML <code>int32</code> data type is returned as <code>'double'</code> .
<code>DataTypeFloat</code>	Specifies the <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 (the types of most VRML fields) are returned as <code>'single'</code> . If set to <code>'double'</code> , the VRML <code>float</code> and <code>color</code> data types are returned as <code>'double'</code> .

## 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.

## Using CAD Models with the Virtual Reality Toolbox Product

In this section...
“Section Overview” on page 5-26
“Exporting VRML Models from CAD Tools” on page 5-26
“Virtual Scene Modeling” on page 5-33
“Linking Virtual Scene to a Simulink, SimMechanics, or MATLAB Model” on page 5-37

### Section Overview

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

This section describes how to adapt existing CAD designs for visualization using the Virtual Reality Toolbox software.

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

### Exporting VRML Models from CAD Tools

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

When exporting CAD models into the VRML format, several options can be set to customize the output. These include options specific to the export filters or are general CAD file properties (consult your CAD system documentation for specific details on how to set these properties). The most typical and useful properties are the following:

- “VRML Format Type” on page 5-27
- “Level of Detail Considerations” on page 5-28
- “Units Used in Exported Files” on page 5-28
- “Coordinate System Used” on page 5-29
- “Assembly Hierarchy” on page 5-29

### **VRML Format Type**

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

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

---

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

---

### **Level of Detail Considerations**

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

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

As there are significant performance differences among various computers and graphic accelerators, there is no firm recommendation for the number of polygons or triangles suitable for use with the Virtual Reality Toolbox product. To assess the model's complexity, you can display the resulting VRML file in the Virtual Reality Toolbox viewer and observe the viewer response to navigation. If you can navigate the virtual scene without any significant delays, the model is usually suitable for further work. If you connect the virtual scene to a Simulink model, you have access to more precise measures of suitability, such as the number of frames rendered per second during simulation.

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

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

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



like collision detection, near-object clipping, or terrain following. You can customize avatar dimensions (and also other navigation-specific parameters such as default navigation speed) using the `NavigationInfo` VRML node. The Virtual Reality Toolbox viewer enables effective navigation in the virtual scene, including scaled scenes (e.g., inspecting miniature objects or visualizing a large-scale aircraft operation in space). For such navigation to be successful, the scene's author must define the `NavigationInfo` parameters correctly.

## Coordinate System Used

VRML uses a Cartesian coordinate system with axes defined so that:

- $+x$  points right
- $+y$  points up
- $+z$  points out of the screen

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

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

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

## Assembly Hierarchy

How assembly of parts are exported depends on the structure of the model, which usually comes in two forms:

- All parts are independent from each other, or objects in the scene are independent from each other at the same level of the scene hierarchy. The exported VRML file has a flat structure, with all part coordinates defined in global coordinates.

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

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

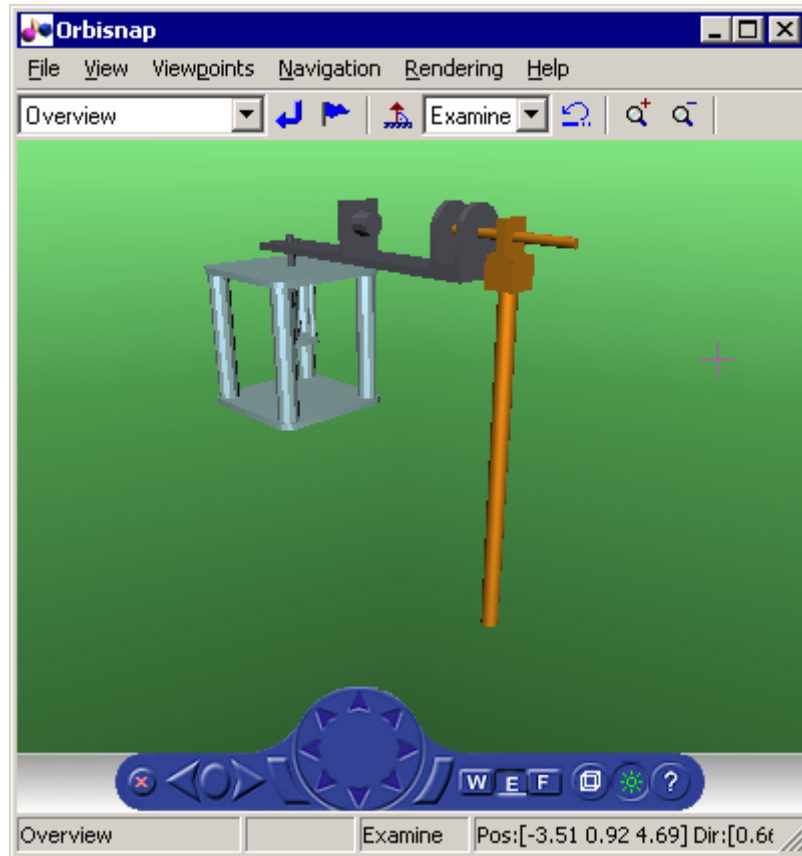
**rotating support — arm — wrist — hand — tool**

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

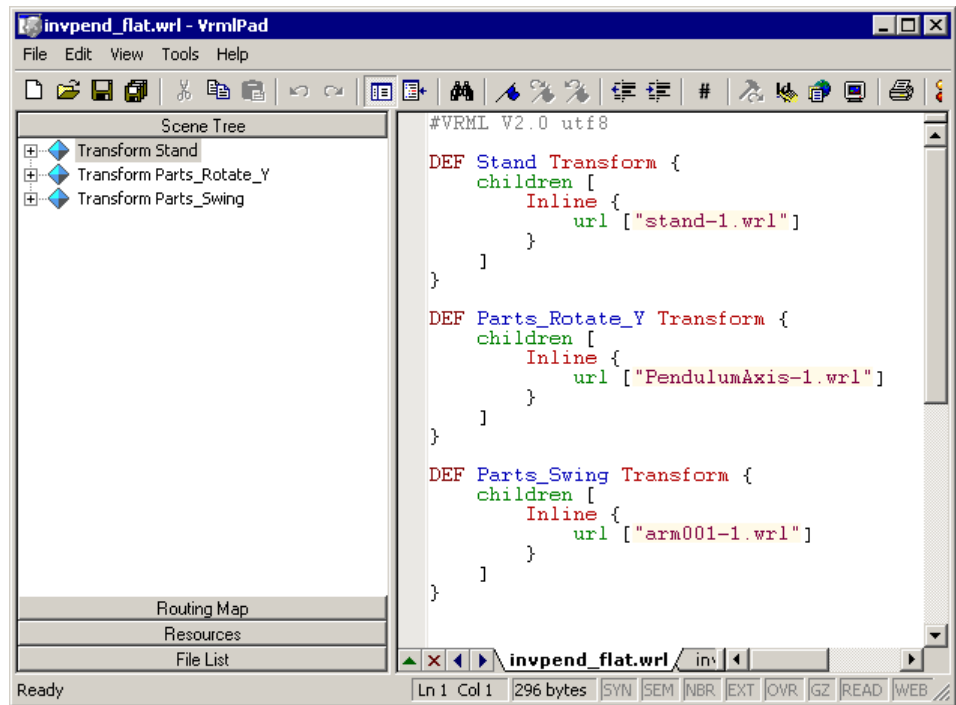
The hierarchy of the VRML file must correspond to the coordinates used in the dynamic model of the assembly as follows:

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

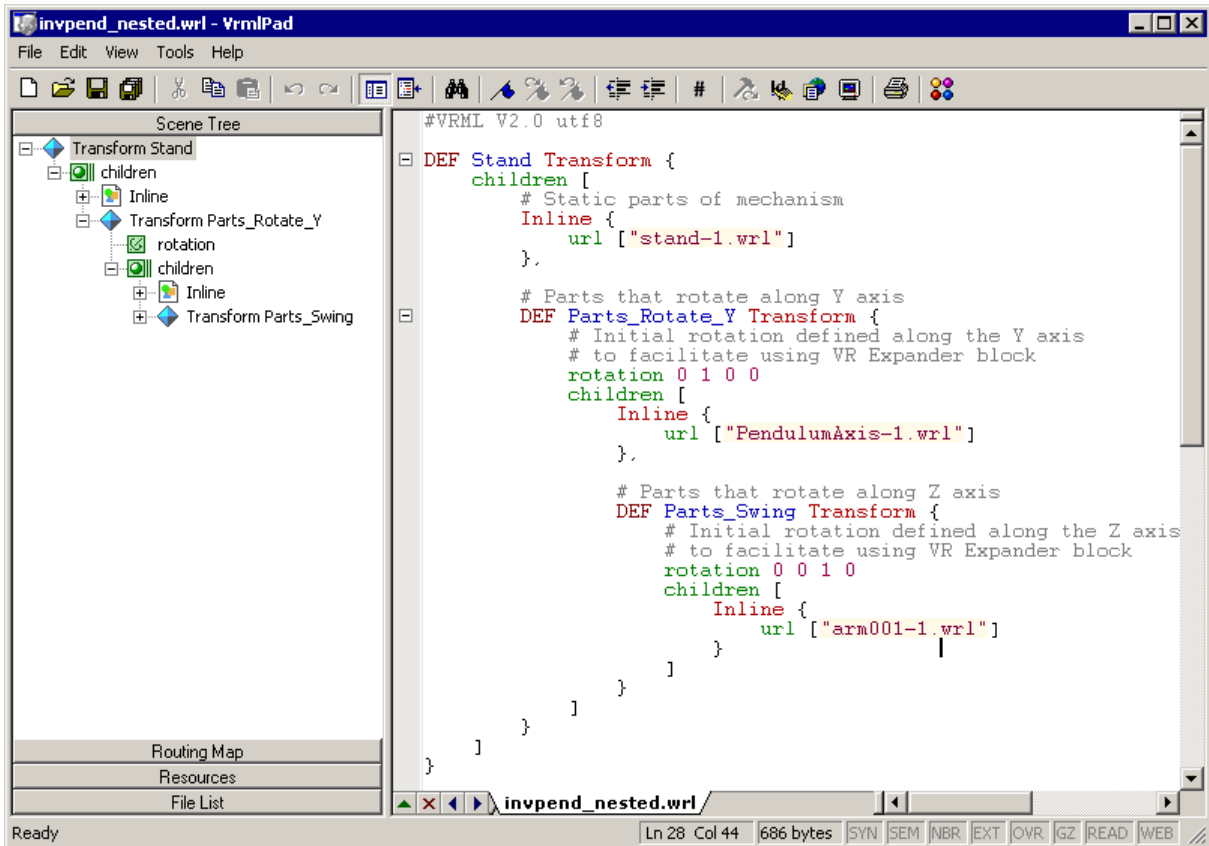
To illustrate these two cases, imagine a rotating pendulum according to the following figure. The gray arm rotates about the vertical axis, while the orange pendulum swings about the  $z$  axis in the rotating gray arm's local coordinates.



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



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



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

## Virtual Scene Modeling

The results of CAD tool export filters often need a few manual changes before you can use them in scenes in the Virtual Reality Toolbox software (e.g., composing the converted model into an urban or manufacturing environment, or adding objects such as viewpoints, backgrounds, lights, etc.).

## Manual Modifications of Exported VRML Files

Manual adjustment of the exported VRML files can be done easily using a plain text editor (alternatively, you can use the Ligos V-Realm Builder application supplied with the toolbox). Typically, adjusting exported files manually in a text editor requires the following changes:

- “Wrap shape objects with transforms” on page 5-34
- “Add DEF names” on page 5-34

**Wrap shape objects with transforms.** CAD tools export parts into VRML as individual shapes using various VRML object types (e.g., a VRML Shape node or the Inline mechanism). To control part positions and orientations, you need to wrap each such Shape or Inline node with a node that allows for the changing of these properties. This wrapping node is the Transform node, whose purpose is to transform the coordinates of its children. For instance, after wrapping with a Transform node, an Inline node may have the following syntax:

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

To set the initial location of the entire assembly in the virtual scene, it is a good practice to wrap all parts of the assembly with an additional Transform node.

**Add DEF names.** CAD export filters often export objects with no names or with synthetic nondescriptive names. To be accessible from MATLAB interface, each VRML object needs to be given a unique name in the VRML file. You name the object by adding a DEF Object\_Name statement to the Transform line. After adding the DEF Object\_Name, the Robot\_Arm1 definition in the main VRML file has the following syntax:

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

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

---

**Note** Sometimes it is necessary to correct bugs introduced in the file by the CAD tool export filter. As the VRML format is a text-based format codified by an ISO standard, these bugs are relatively easy to identify and correct. If problems occur when you are using exported VRML files in the Virtual Reality Toolbox software, consult technical support.

---

## Creating a Virtual Scene

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

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

- 1 Add the `WorldInfo` node with a scene title (used as the virtual world description in the Virtual Reality Toolbox software).
- 2 Add the `NavigationInfo` node defining the scene default navigation speed and avatar size that ensures correct display of the object from near and far distances.

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

For example, if you want to visualize vehicle dynamics, you would place a virtual car on a virtual road. Both objects need to be to scale (the length units in the car and road models must match), and the car must be placed in an appropriate position relative to the road. You achieve proper car scaling, placement, and orientation in the scene by defining corresponding fields of the main object’s `Transform` node mentioned in “Wrap shape objects with transforms” on page 5-34.

For an example of a complete scene definition, see the `octavia_scene.wrl` VRML file that belongs to the Virtual Reality Toolbox demo `vr_octavia`.



## Linking Virtual Scene to a Simulink, SimMechanics, or MATLAB Model

The purpose of this step is to create associations between dynamic model object quantities and corresponding VRML object properties (positions, rotations, etc.) to establish a live data connection between the model and the virtual scene.

Mechanical systems are typically modeled using SimMechanics or Simulink, but the Virtual Reality Toolbox product allows for the visualization of models implemented in MATLAB. The following section discusses specifics on using the MATLAB interface.

### Linking the Virtual Scene to a Simulink Model

You associate Simulink model signals to virtual scene object properties through the VR Sink block from the Virtual Reality Toolbox block library, `vrlib`.

To associate a Simulink signal to a virtual object property:

- 1** From the `vrlib` library, insert a VR Sink block into your Simulink model.
- 2** Double-click the VR Sink block to open the block parameters dialog, where you can define the virtual scene. Enter the name of the scene's VRML file in **Source file**, or click **Browse** to select the file interactively. Click **Apply** to load the selected VRML scene.
- 3** For smooth visualization of the movement, it is sometimes necessary to change the block's **Sample time**. For example, to update the virtual scene 25 times per simulation second, set the **Sample time** to **0.04**. Be careful when using the inherited sample time for the VR Sink block. Depending on the solver used, using inherited sample time might result in nonequidistant (in simulation time) updating of the virtual scene, giving the user a false impression of system dynamics.
- 4** In **VRML Tree** in the right side of the dialog box, expand the main object's Transform branch, and, in the scene object hierarchy, locate all parts you want to control from Simulink according to their names as given in "Add DEF names" on page 5-34. Each part is represented by named Transform nodes, and you select the check box next to its rotation and position fields.

These selections tell the VR Sink block that you want to control the rotation and position of these parts. You can also select other properties of virtual scene objects, such as color, but rotations and positions are the ones most frequently controlled.

- 5** Click **OK**. For each selected field, the VR Sink block creates an input port. Increase the VR Sink block size as appropriate to the number of input ports.

After the VR Sink block is associated with a virtual scene, you can double-click it to open the Virtual Reality Toolbox viewer. Block parameters are available through the menu **Simulation > Block Properties** in the viewer.

VR Sink inputs take signals of the type corresponding to their VRML representation. Position inputs are of type `SFVec3f`, which is the position represented in `[x y z]` coordinates. Rotation inputs are of type `SFRotation`, the four-element vector defining rotation as `[axis angle]`, using the coordinate system described in “Coordinate System Used” on page 5-29, where the angle value is in radians.

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

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

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

- When all parts in a Simulink model are defined in global coordinates, and the virtual scene has a flat structure of independent objects, use the following positions and rotations.

Object positions	Send to VR Sink all positions in global coordinates.
Object rotations	Send to VR Sink all rotations in global coordinates, with center of rotation defined as the coordinate system origin. (As the default center of rotation of VRML Transform objects is [0 0 0], it is usually not necessary to define it for each part in the VRML file.

- When all parts in Simulink model follow hierarchical relations, and the virtual scene has a nested structure, use the following positions and rotations.

Object positions	Send to VR Sink all positions in local coordinates (relative to their parents or predecessors in the object hierarchy). For example, send the robot's tool position relative to the robot's hand.
Object rotations	<p>Send to VR Sink all rotations in local coordinates (relative to their parents or predecessors in the object hierarchy). For example, send the robot's tool rotation relative to the robot's hand.</p> <p>To visually match the positions of joints between objects, it is usually necessary to coincide the center of rotation defined in the virtual scene with the center of rotation defined in the Simulink model, as joints between parts are usually positioned not in the origin, [0 0 0], of the parent's coordinate system.</p> <p>To define a center of rotation different from the default value, [0 0 0], define the <b>center</b> field of the child's Transform node in the VRML file. For example, define the robot's tool center of rotation to coincide with the joint connecting the hand and the tool in the hand's local coordinates.</p>

In a hierarchical scene structure, when the parts are connected by revolving joints, define state by using the relative rotations between parts. The joint axis directly defines the VRML rotation axis, so constructing the [axis angle] four-element VRML rotation vector is trivial.

## **Initial Conditions**

A Simulink model's initial conditions must correspond to the initial object's positions and rotations defined in the virtual scene. Otherwise, the object controlled from Simulink would “jump” from the position defined in the VRML file to the position dictated by the Simulink software at the start of the simulation. You can compensate for this offset either in the VRML file (by defining another level of nested `Transform` around the controlled object) or in the Simulink model by adding the object's initial position to the model calculations before sending to the VR Sink block.

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

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

The VR Sink block accepts only inputs that define fully qualified VRML field values. Dynamic models that describe the system behavior in only one dimension still require full 3D positions for all controlled objects for their virtual reality visualization.

To simplify the modeling in such cases, you can use the VR Placeholder and VR Expander blocks of the Virtual Reality Toolbox library.

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

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

To control the position of a virtual object in a one-dimensional dynamic model, use the VR Signal Expander block with the controlled dimension as its input. For its output use a three-component vector in the VR Sink block. The remaining vector elements are filled with placeholder signals.

Use of the VR Signal Expander block is also a possibility when defining rotations. When the axis of rotation (as a part of the initial rotation of an object Transform node) is defined in the VRML file, it is possible to send to the VR Sink block a VRML rotation value consisting of three placeholder signals and the computed angle, forming a valid four-element [axis angle] vector.

### **Linking the Virtual Scene to a SimMechanics Model**

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

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

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

Depending on the virtual scene hierarchy, you can use one of two methods to help visualize SimMechanics machines:

- When the virtual scene has a flat structure of independent objects, you can obtain the positions and rotations of machine parts using Body Sensor blocks connected to appropriate coordinate systems attached to the bodies, with positions and rotations defined using global coordinates. In most cases, it is appropriate to connect the sensor to a body coordinate system with origin at [0 0 0] and with an initial rotation matrix defined as the identity matrix, [1 0 0; 0 1 0, 0 0 1], in the global coordinate.
- When the virtual scene has a hierarchical structure of nested objects, the body positions and rotations can be obtained using Body Sensor blocks with output set to use local body coordinates. In some special cases (e.g., when two bodies are connected through a revolving joint), it is possible to get the angle between the objects using a Joint Sensor block.

### **Linking the Virtual Scene to a MATLAB Model**

For interacting with virtual scenes, the Virtual Reality Toolbox product also offers a set of MATLAB functions and constructs referred to collectively as its “MATLAB interface.” Circumstances when this MATLAB functionality is appropriate for use with CAD-based designs may include:

- Using customized GUIs to visualize static objects and their relations in a virtual environment, such as in interactive machine assembly instructions.
- Visualizing 3D information based on an independent quantity (not necessarily time).
- Using MATLAB interface functions in Simulink model callbacks.
- Visualizing systems whose dynamic models are available as MATLAB code.
- Visualizing systems where massive object changes, such as deformations, are taking place. In this case, you must send dynamically-sized matrix-type data from the dynamic models to virtual scenes, which is not possible using just Simulink signals.

For preparing virtual scenes to work with the Virtual Reality Toolbox MATLAB interface, most of the considerations mentioned throughout this chapter apply. For information on setting object properties using the MATLAB interface, see “Using the MATLAB Interface” on page 4-2.

# 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 product includes its own viewer as the default for all supported platforms. It is the preferred method of viewing virtual worlds. For PC platforms, the product 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” on page 6-2
- “blaxxun Contact VRML Plug-In” on page 6-49

## Virtual Reality Toolbox Viewer

In this section...
“Section Overview” on page 6-2
“Menu Bar” on page 6-5
“Toolbar” on page 6-6
“Navigation Panel” on page 6-7
“Starting and Stopping Simulations” on page 6-10
“Navigation” on page 6-11
“Frame Capture and Animation Recording File Tokens” on page 6-18
“Creating Frame Captures” on page 6-21
“Configuring Animation Recording Parameters” on page 6-23
“Recording Files in the VRML Format” on page 6-24
“Recording Files in the Audio Video Interleave (AVI) Format” on page 6-25
“Scheduling Files for Recording” on page 6-28
“Interactively Starting and Stopping Animation Recording” on page 6-31
“Viewing the Animation File” on page 6-31
“Working with Viewpoints” on page 6-33
“Rendering” on page 6-40

### Section Overview

The Virtual Reality Toolbox software contains a viewer as the default method for viewing virtual worlds. You can use this viewer on any supported operating system. For a list of supported operating systems, see “System Requirements” on page 2-6. This section provides an overview of the features and controls of the viewer.

This section uses the `vrpend`, `vrplanets`, and `virtut1` demos to illustrate the viewer features:



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

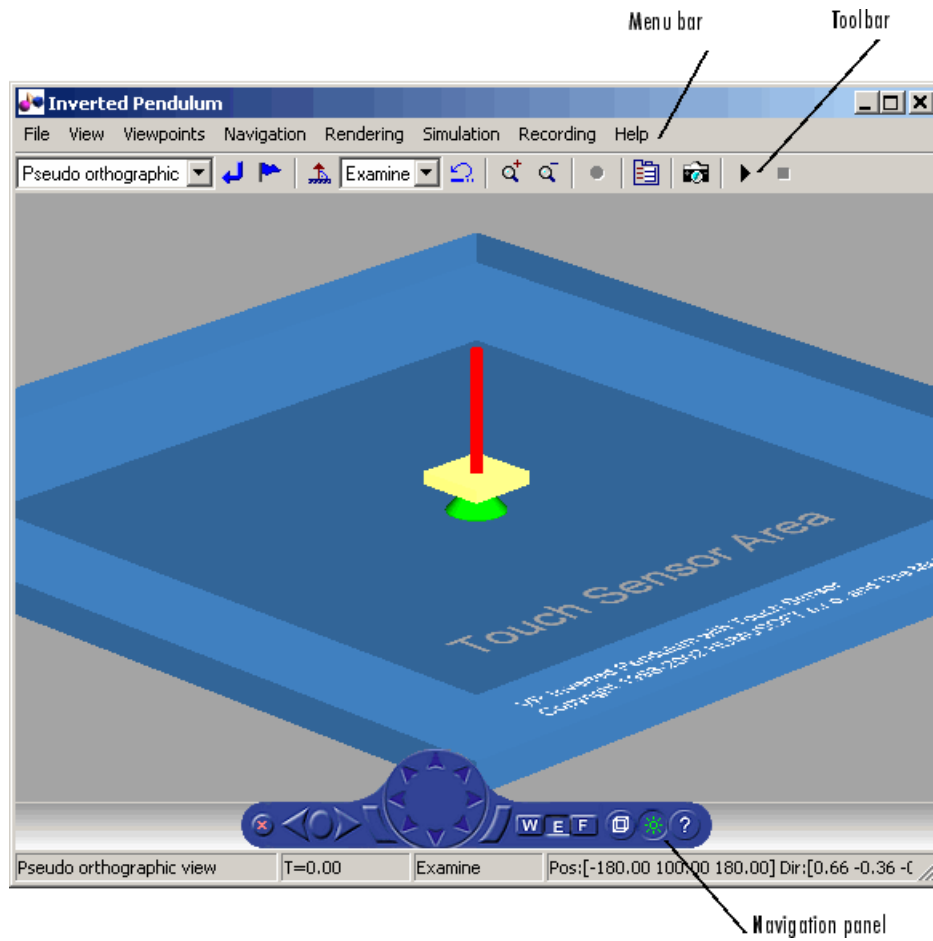
```
vrpend
```

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.

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



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

## 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 Toolbox 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 6-11 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 6-40).

- **Simulation** — Manages model starts/stops and VR block parameters.
- **Recording** — Manages frame capture and animation recording file parameters.
- **Help** — Displays the Help browser for the Virtual Reality Toolbox viewer.

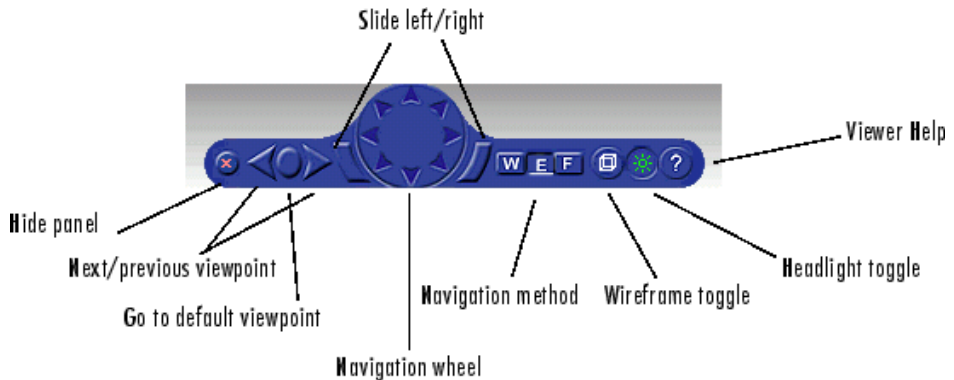
### Toolbar

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 
- **Capture a frame screenshot** 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			
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			

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

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

## 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.



---

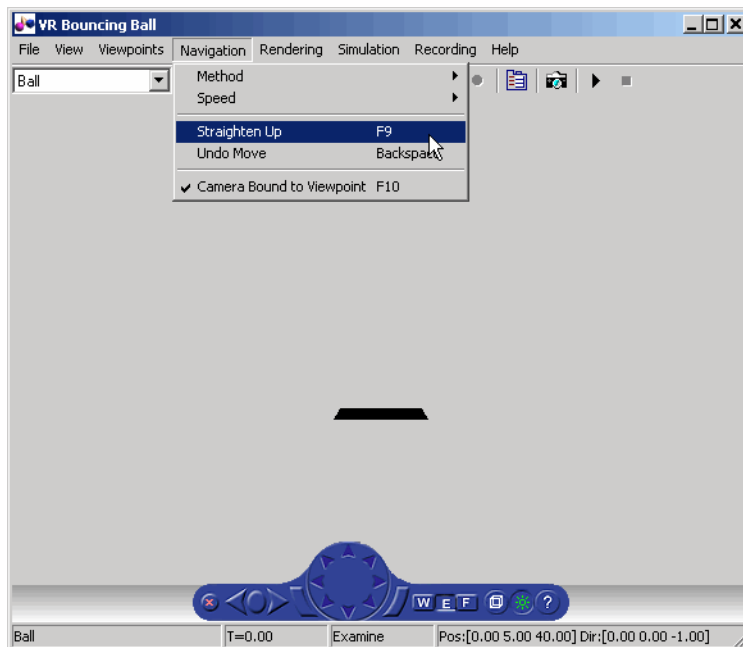
**Note** The **Ctrl+T** operation is available only if you started the viewer from a Simulink model. If you start the viewer through the MATLAB interface, no Simulink model is associated. You cannot start and stop the simulation in this case.

---

## Navigation

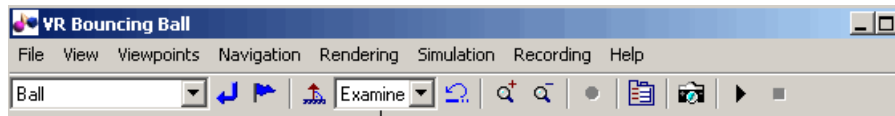
You can navigate around a virtual scene using the menu bar, toolbar, navigation panel, mouse, and keyboard. The `vrbounce` demo shows the viewer's functionality.

**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 Virtual Reality Toolbox™ Viewer Mouse Navigation on page 6-13.
- From the toolbar, select the drop-down menu that displays the navigation options **Walk**, **Examine**, and **Fly**.



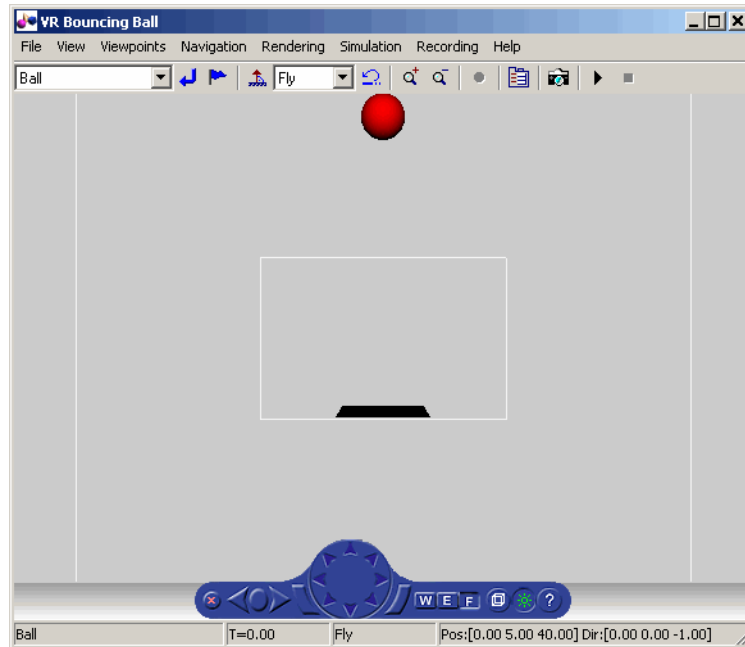
Navigation drop-down menu

- 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.

### Virtual Reality Toolbox Viewer Mouse Navigation

Movement Mode	Zone and Description
Walk	<p><b>Outer</b> — Click and drag the mouse up, down, left, or right to slide the camera in any of these directions in a single plane.</p> <p><b>Inner</b> — Click and drag the mouse up and down to move forward and backward. Drag the mouse left and right to turn left or right.</p>

### Virtual Reality Toolbox Viewer Mouse Navigation (Continued)

Movement Mode	Zone and Description
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 6-15 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.

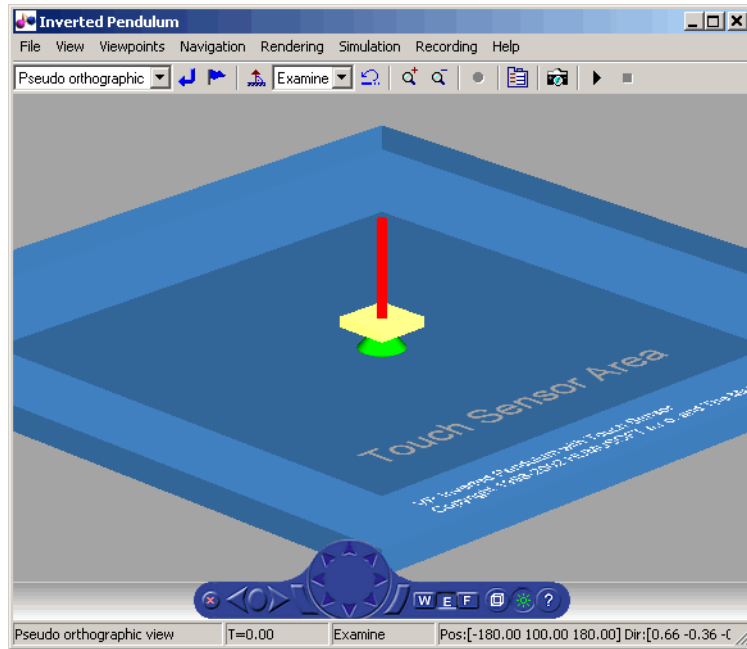
To change the default navigation speed for a virtual scene, modify the **speed** field of the `NavigationInfo` node in the scene VRML file. For further details, see the VRML97 standard.

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

1 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.

### Virtual Reality Toolbox Viewer Keyboard Navigation

<b>Keyboard Command</b>	<b>Navigation Function</b>
<b>Backspace</b>	Undo move.
<b>Ctrl+R</b>	Start/stop recording.
<b>Ctrl+I</b>	Capture frame.
<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.

### Virtual Reality Toolbox Viewer Keyboard Navigation (Continued)

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

### Frame Capture and Animation Recording File Tokens

You can control frame captures of a virtual scene, or record animations into files, using the Virtual Reality Toolbox viewer Capture and Recording Parameters dialog of the **Recording** menu. The Virtual Reality Toolbox software supports a variety of file naming formats using file tokens. This topic describes the Virtual Reality Toolbox file tokens.

By default, the Virtual Reality Toolbox viewer captures virtual scene frames or records simulations in a file named with the following format:



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

This format creates a unique filename each time you capture a frame or 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 a simulation for the first time, the animation file is

```
vrplanets_anim_1.wr1
```

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

You can use a number of tokens to customize the automated generation of frame capture or animation files. This section describes how to use these tokens to create varying frame capture or 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 frame capture or animation file are in the same directory.)
- Create rolling numbered filenames such that subsequent frame captures or 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 capture a frame or run a Simulink model multiple times, but create a unique file each time.
- Create multiple filenames with time or date stamps, with a unique file created each time.

The following tokens are the same for frame capture (`.tif` or `.png`) or animation (`.wr1` 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/%f_anim_%n.avi saves the animation in 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. Note that in this case you must create the animdir subdirectory first.
%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.

Token	Description
%n	The current incremental number replaces this token in the filename string. Each subsequent frame capture or run of the simulation increments the number. For example, the format %f_anim_%n.wr1 saves the animation to vrplanets_anim_1.wr1 on the first run, vrplanets_anim_2.wr1 on the second run, and so forth.
%Y	The current four-digit year replaces this token in the filename string. For example, the format %f_anim_%Y.wr1 saves the animation to vrplanets_anim_2005.wr1 for the year 2005.

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

- “Creating Frame Captures” on page 6-21 — Describes how to create frame captures of a virtual scene
- “Configuring Animation Recording Parameters” on page 6-23 — Describes how to configure animation recording parameters

## Creating Frame Captures

The Virtual Reality Toolbox product allows you to save a frame snapshot (capture) of the current Virtual Reality Toolbox viewer scene. You can save this frame as either a TIF or PNG format file. You can later view this scene offline (in other words, without the Virtual Reality Toolbox viewer). You can treat this file like any other TIF or PNG file, such as print it, include it in other files, and so forth.

This topic describes how to configure and capture a frame, using the vrplanets demo as the example. It assumes that you have read the topic “Frame Capture and Animation Recording File Tokens” on page 6-18 about file names.

- “Configuring Frame Capture Parameters” on page 6-22 — Describes how to configure frame capture file formats
- “Capturing a Frame” on page 6-23 — Describes how to capture frames

### Configuring Frame Capture Parameters

- 1 In the MATLAB Command Window, type

```
vrplanets
```

at the MATLAB command prompt. The Planets demo starts.

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

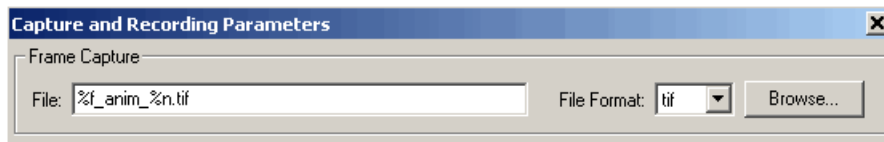
The Capture and Recording Parameters dialog box is displayed.

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

The filename `%f_anim_%n.wrl` appears in the first text field, **File**.

- 4 Leave this filename as is.

- 5 In the **File Format** list, `tif` or `png` specify the graphic file format for the captured frame. The default is `tif`. For this procedure, leave this format setting at `tif`.



- 6 You can disable the navigation panel. The navigation panel appears at the bottom of the virtual scene view. You might want to turn off this panel for a cleaner view of the virtual scene. Choose **View > Navigation Panel > Off**.

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

- 7 Click **OK**.

You can now capture frames of a virtual scene. With this configuration, each subsequent capture of a scene in the same world increments the file name number (`%n`) and saves it in a `tif` file.

## Capturing a Frame

You can capture frames of the virtual world from the Virtual Reality Toolbox viewer through the menu bar, toolbar, or keyboard. This section assumes that you have specified frame capture file formats. See “Configuring Frame Capture Parameters” on page 6-22 if you have not defined frame capture files.

These actions save the captures in files in the current directory.

- From the menu bar, choose **Recording > Capture Frame** to capture a frame.
- From the toolbar, click the **Capture a frame screenshot** button to capture a frame.
- From the keyboard, press **Ctrl+I** to capture a frame.

You can view the frame capture files using any tool that reads `tif` or `png` files, including the MATLAB `imread` function. For example,

```
image(imread('vrplanets_anim_1.tif'))
```

## Configuring Animation Recording Parameters

The Virtual Reality Toolbox product allows you to record animations of virtual scenes controlled from Simulink or MATLAB. You can later play these animations offline (in other words, without the 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 — The 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. The Virtual Reality Toolbox software does not save any navigation movements you make while recording the animation.

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

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

See the following topics:

- “Recording Files in the VRML Format” on page 6-24 — 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 6-25 — Describes how to configure the record simulation parameters to create 2-D format animation files.
- “Scheduling Files for Recording” on page 6-28 — Describes how to schedule record simulation operations to occur automatically.

### 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 “Frame Capture and Animation Recording File Tokens” on page 6-18 to save files to other filenames.

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

```
vrplanets
```

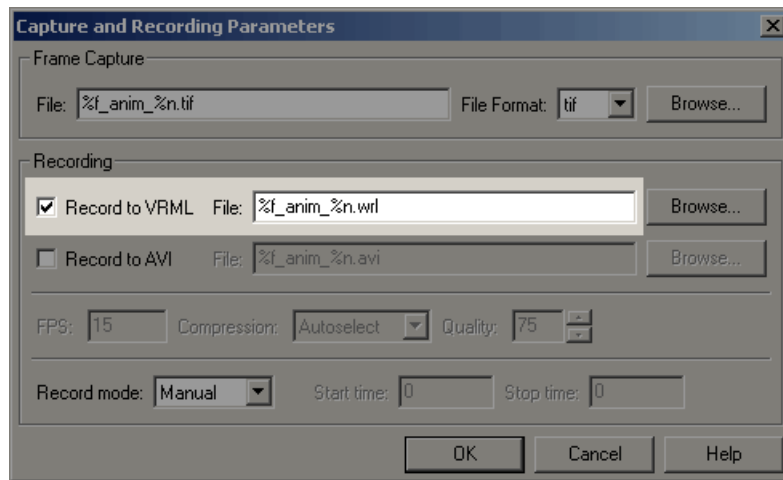
The Simulink model is displayed. Also, by default, the Virtual Reality Toolbox viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the VR Sink block in the Simulink model.

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

The Capture and Recording Parameters dialog box is displayed.

- 3 Find the **Recording** section of the dialog. This is located under the Frame Capture dialog.
- 4 Select the **Record to VRML** file check box.

The **File** text field becomes active and the default filename, %f\_anim\_%n.wrl, appears in the text field.



- 5 Click **OK**.

After you define an animation file, you can manually record simulations. See “Interactively Starting and Stopping Animation Recording” on page 6-31. If you want to record simulations on a schedule, see “Scheduling Files for Recording” on page 6-28.

## 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 “Frame Capture and Animation Recording File Tokens” on page 6-18 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.

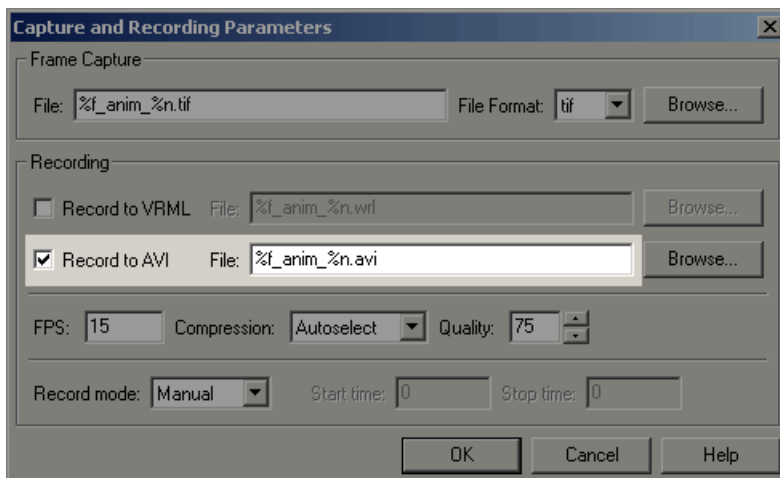
- 2 From the **Recording** menu, choose **Capture and Recording Parameters**.

The Capture and Recording Parameters dialog box is displayed.

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

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

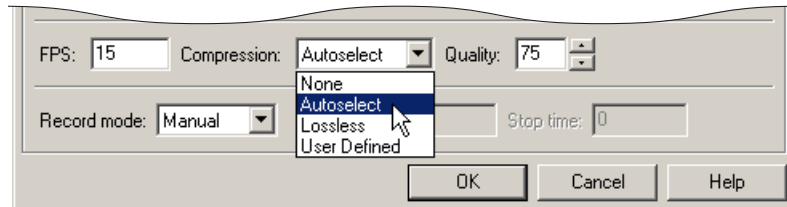
The **File** text field and Compression selection area become active, and the default filename, `%f_anim_%n.avi`, appears in the text field.



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



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



Choose from

- **Autoselect** — Allows the Virtual Reality Toolbox software to select the most appropriate compression codec. This option allows you to specify a quality setting that is a number between 0 and 100. Higher quality numbers result in higher video quality and larger file sizes. Lower quality numbers result in lower video quality and smaller file sizes.
  - **Lossless** — Forces the Virtual Reality Toolbox software to compress the animation file without loss of data. (Typically, the compression of files sacrifices some data.)
  - **User Defined** — Enables you to specify a particular compression codec. This option allows you to specify a quality setting that is a number between 0 and 100. Higher quality numbers result in higher video quality and larger file sizes. Lower quality numbers result in lower video quality and smaller file sizes. You need to specify an identification string of a codec that your system supports.
  - **None** — Prevents any compression for the animation file.
- 7** Disable the navigation panel. The navigation panel appears at the bottom of the virtual scene view. You might want to turn off this panel for a cleaner view of the virtual scene. Choose **View > Navigation Panel > Off**.

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

- 8** Click **OK**.

---

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

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

---

After you define an animation file, you can record animations. See “Interactively Starting and Stopping Animation Recording” on page 6-31. If you want to record animations on a schedule, see “Scheduling Files for Recording” on page 6-28.

## 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 6-24 or “Recording Files in the Audio Video Interleave (AVI) Format” on page 6-25 for further details.

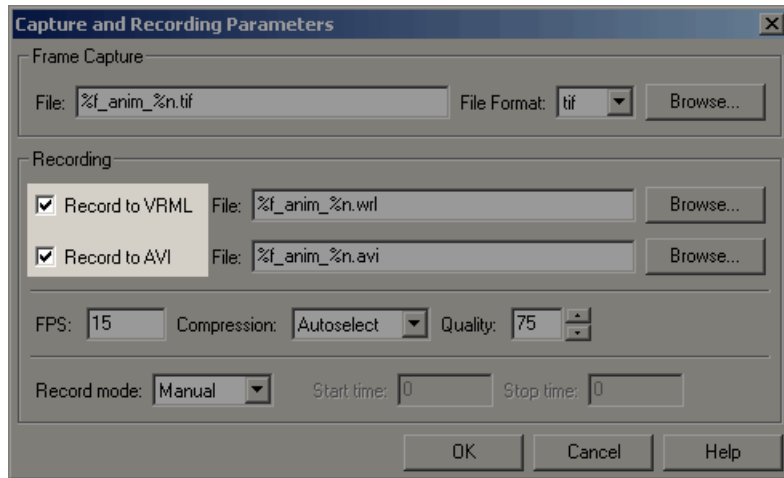
- 1 If the `vrplanets` demo is not already open, type the demo’s name in the MATLAB Command Window. For example:

```
vrplanets
```

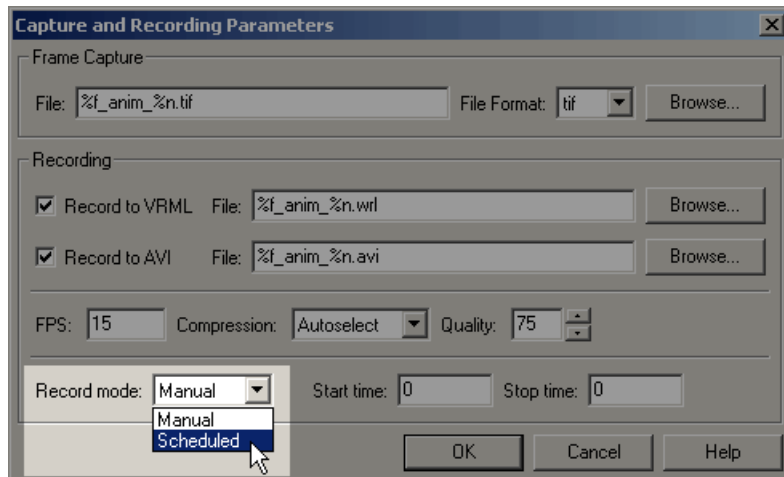
The Simulink model is displayed. Also, by default, the Virtual Reality Toolbox viewer for that model is loaded and becomes active. If the viewer is not displayed, double-click the VR Sink block in the Simulink model.

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

The Capture and Recording Parameters dialog box is displayed. In the **Recording** section, this dialog box contains the **Record mode** list. Note that the **Record mode** list is enabled only if you also select either or both of the **Record to VRML** and **Record to AVI** check boxes.



- 3 From the **Record mode** list, choose Scheduled.



The **Start time** and **Stop time** text fields are enabled.

- 4 Enter in **Start time** and **Stop time** the start and stop times during which you want to record the animation. For example, enter 0 as the start time and 100 as the stop time.

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

---

**Note** You can also set the stop time before the start time. This allows for a scenario where the simulation starts first and you manually start recording. The recording then automatically stops at stop time and automatically restarts at start time.

---

- 5 Click **OK**.

After you define the schedule, you can record simulations. See “Starting and Stopping Simulations” on page 6-10.

---

**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 6-23 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 6-24 or “Recording Files in the Audio Video Interleave (AVI) Format” on page 6-25 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 the MATLAB software.

- 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.

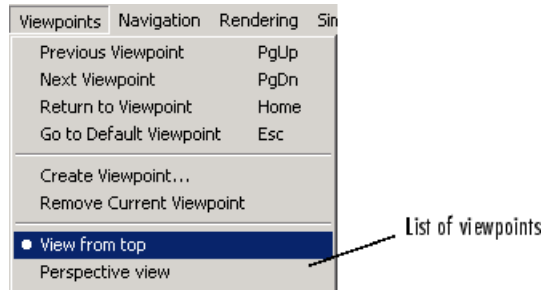
- 1 Select a Virtual Reality Toolbox demo and type that demo's name in the MATLAB command window. For example:

```
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.

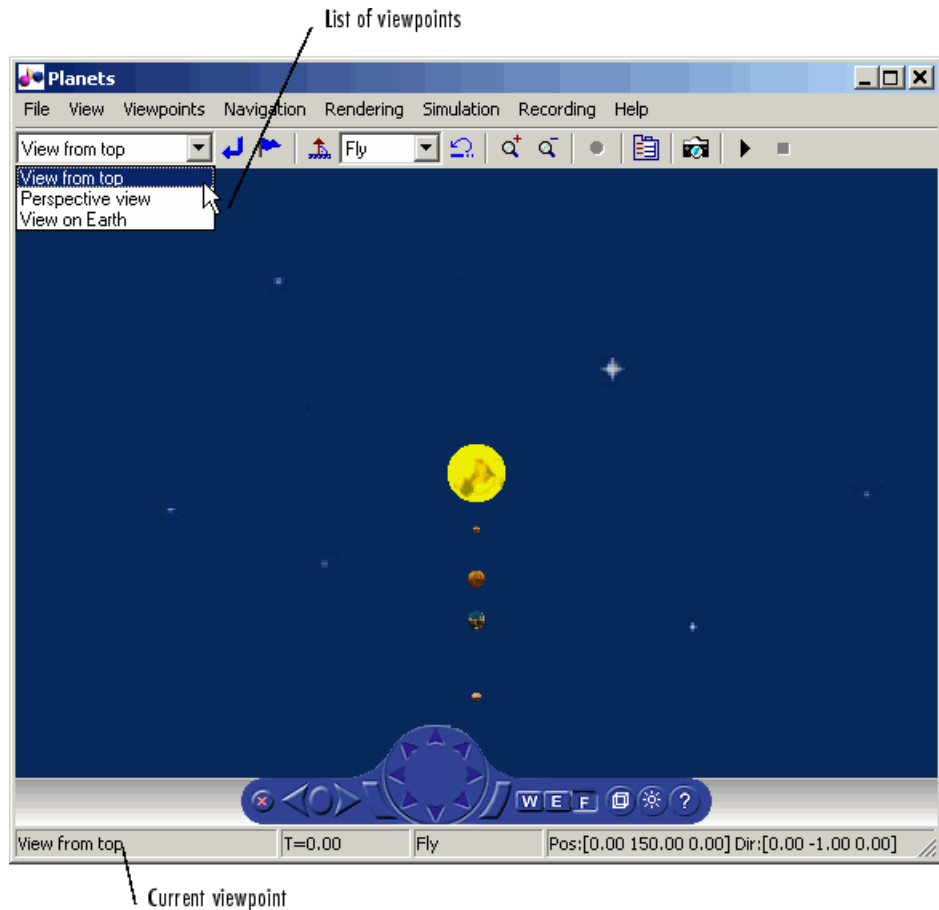
- 2 From the menu bar, select the **Viewpoints** menu.

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 6-36. 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:

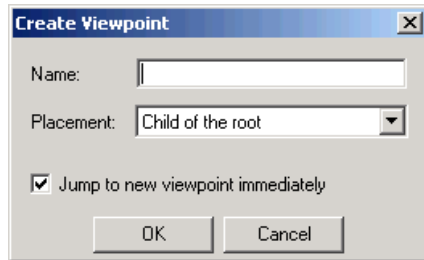
```
vrplanets
```

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.

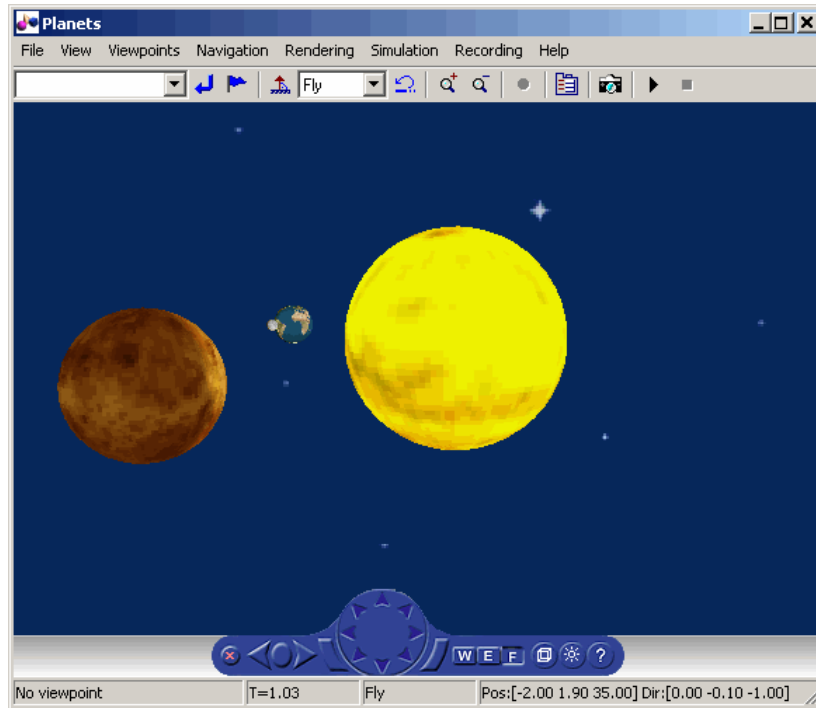
- 14** In the **Placement** field, select **Sibling** of the current viewpoint. This option creates a new viewpoint at the same level in the virtual world object hierarchy as the child of the parent transform of the current viewpoint. The local coordinate system of the parent transform defines the new 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.

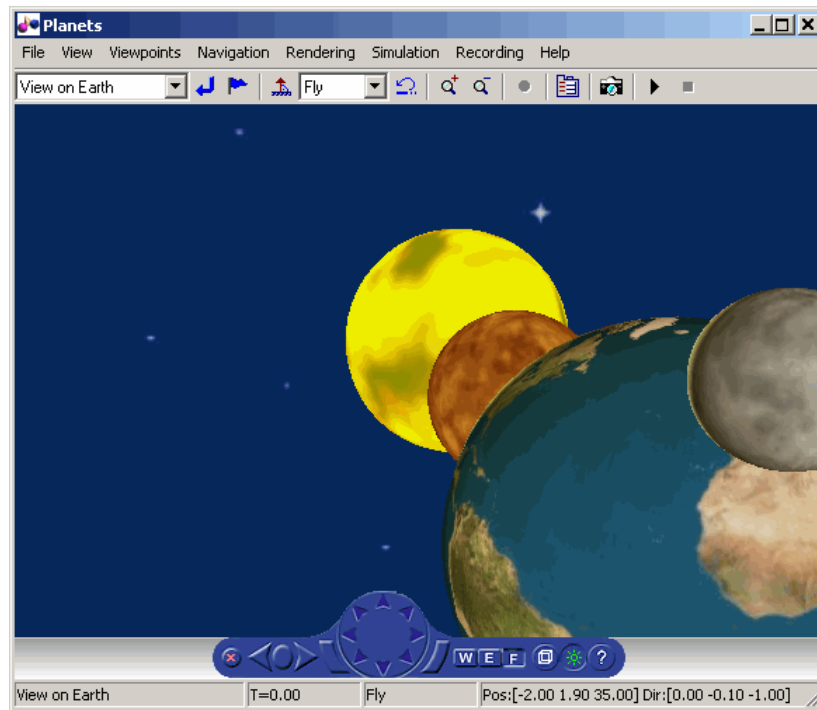


### Rendering

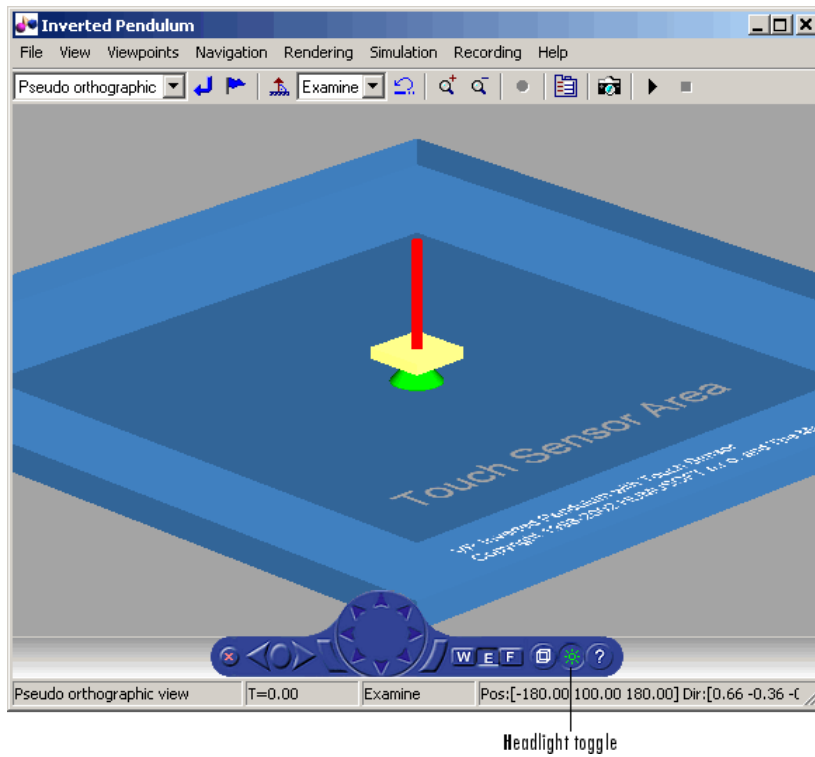
You can change the rendering of the scene through the controls on the navigation panel or options on the rendering menu. The `vrpend` and `vrplanets` demos are used to demonstrate the viewer's functionality.

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.

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

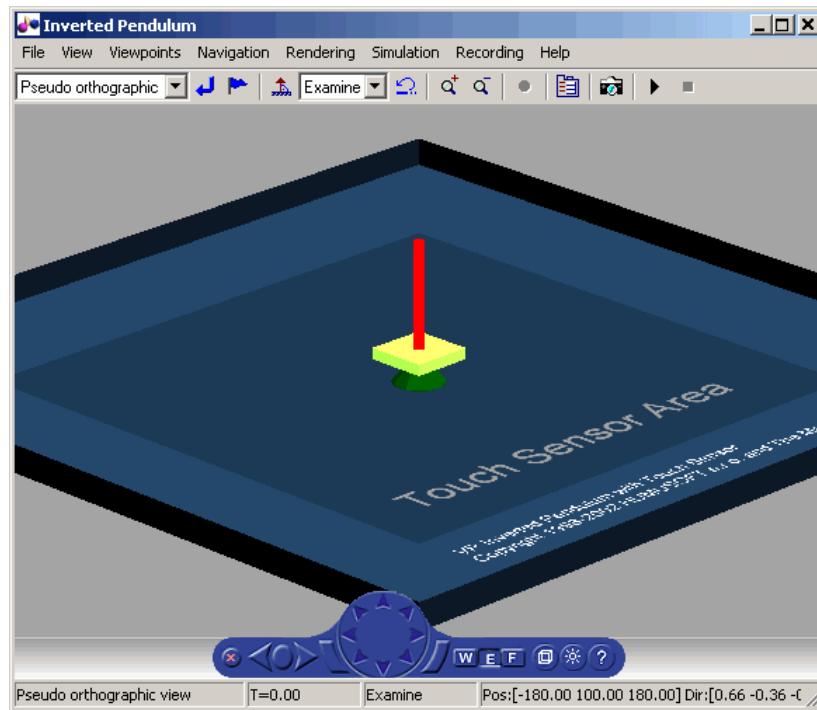


You can turn the camera headlight and the lighting of the scene on or off. When **Headlight** is off, the camera does not emit light. Consequently, the scene can appear dark. For example, the following figure depicts the vrpend demo with **Headlight** on.



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



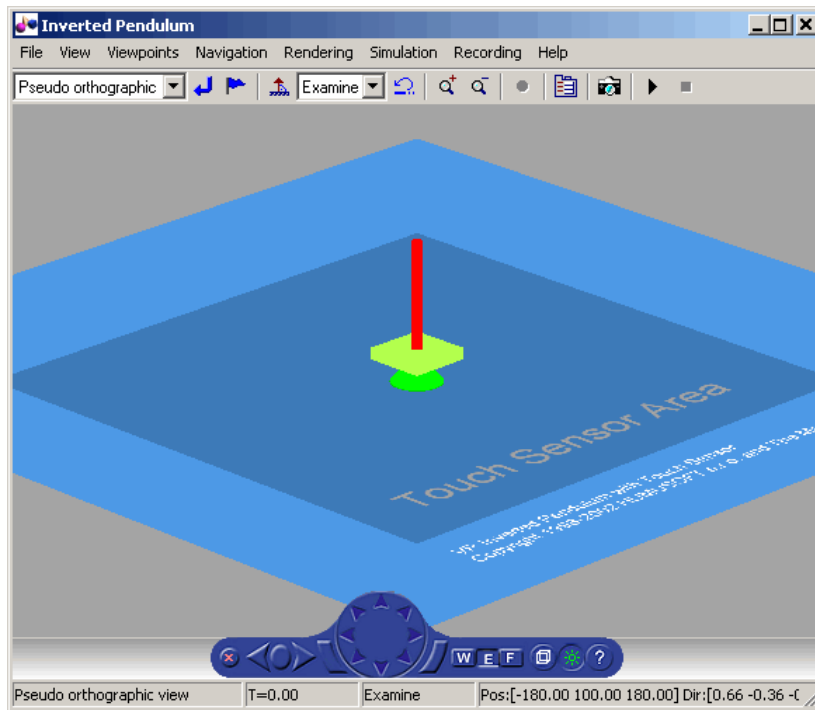



---

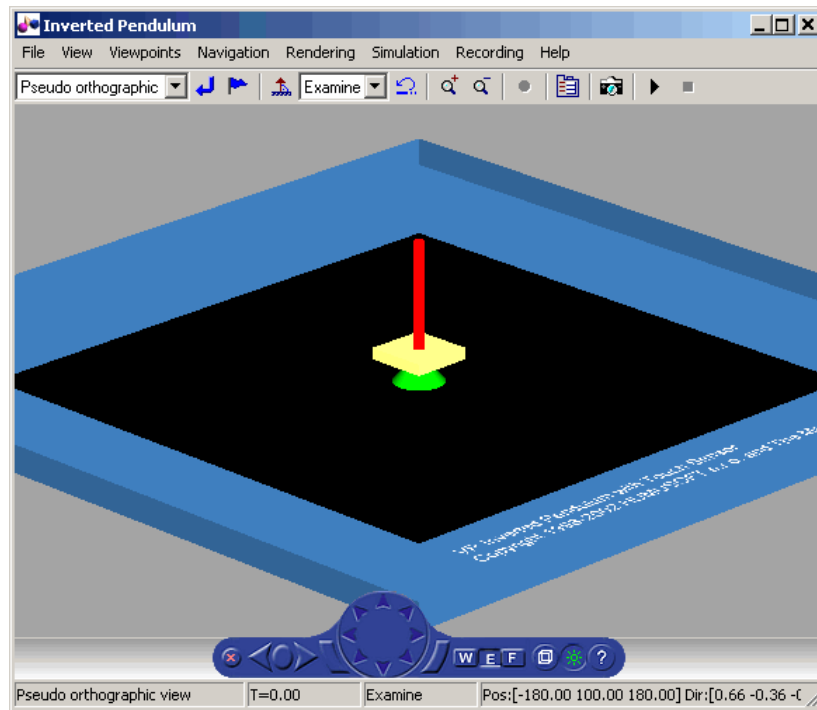
**Note** It is helpful to define enough lighting within the virtual scene so that it is lit regardless of the **Headlight** setting.

---

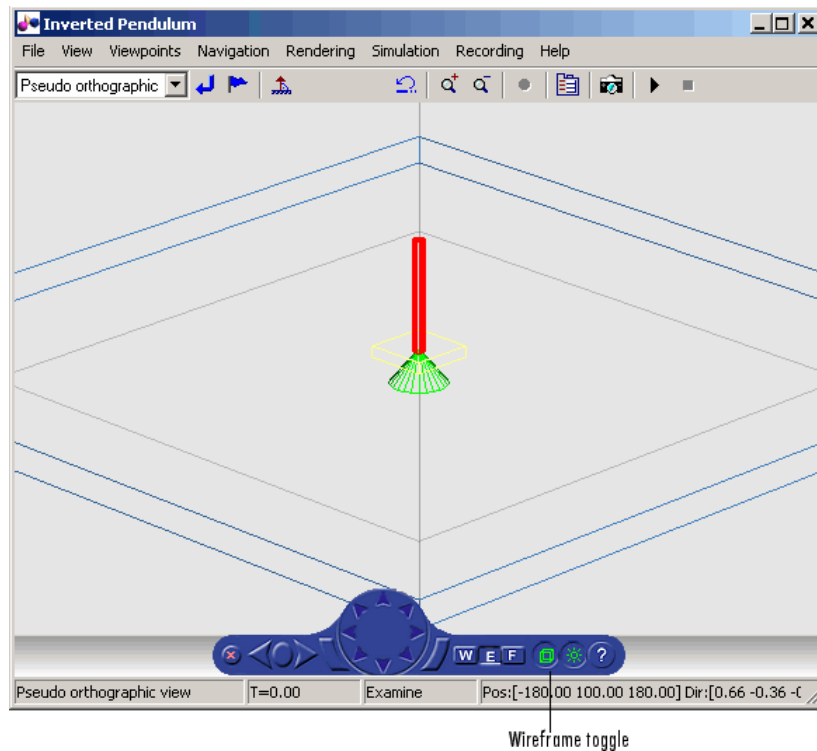
When **Lighting** is off, the virtual world appears as if lit in all directions. The Virtual Reality Toolbox viewer does not compute and render all the lighting effects at the surfaces of the objects. Shadows disappear and the scene loses some of its 3-D quality. The following is the vrpend demo with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



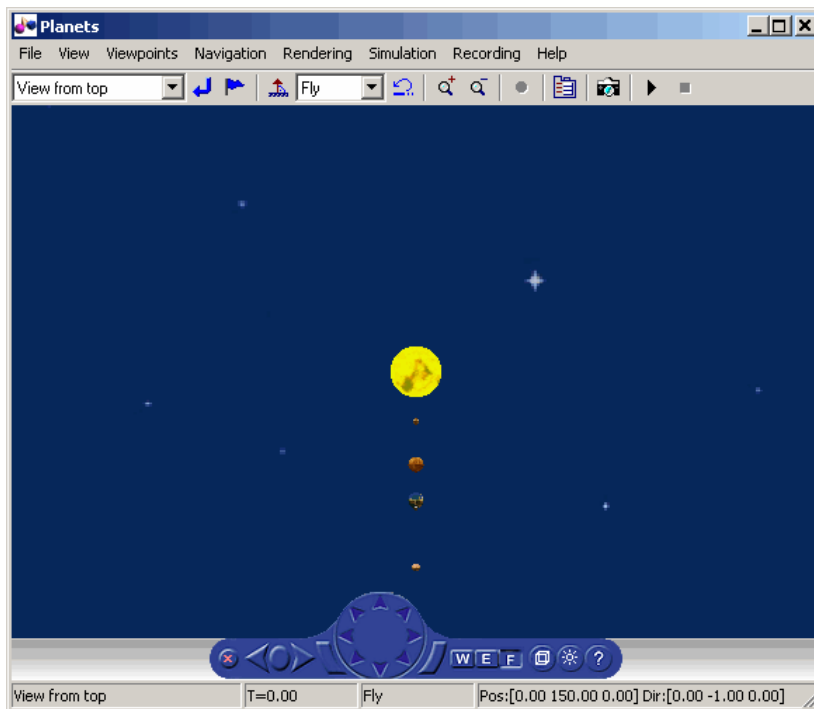
If **Transparency** is off, transparent objects are rendered as solid objects.



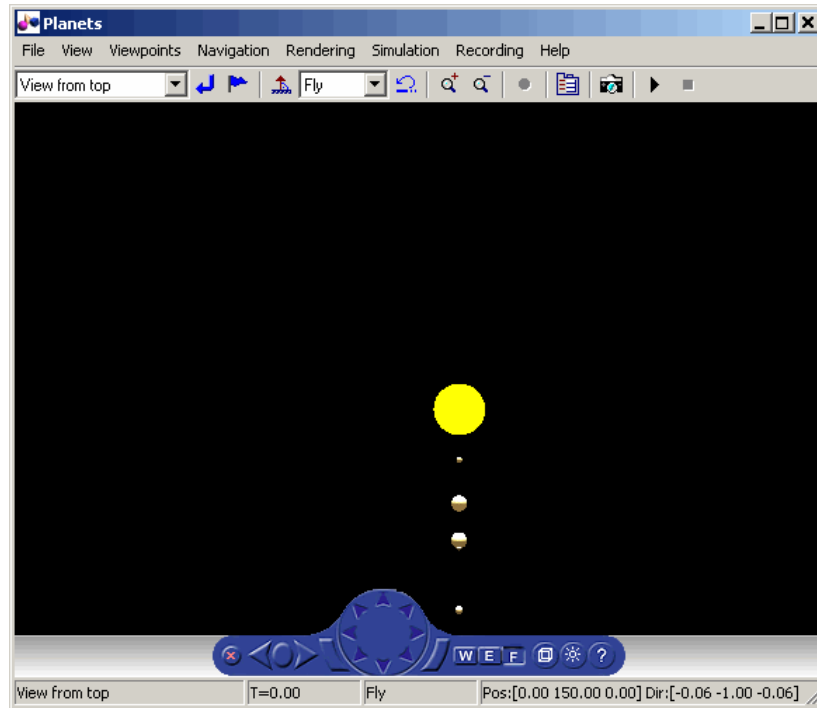
Turning **Wireframe** on changes the scene's objects from solid to wireframe rendering. The following is the vrpend demo with **Wireframe** on.



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.



You can specify the maximum size of a texture used in rendering the `vrfigure` object. This option gives you a list of texture sizes to choose from. See the `vrfigure` `MaxTextureSize` property for further details.

## blaxxun Contact VRML Plug-In

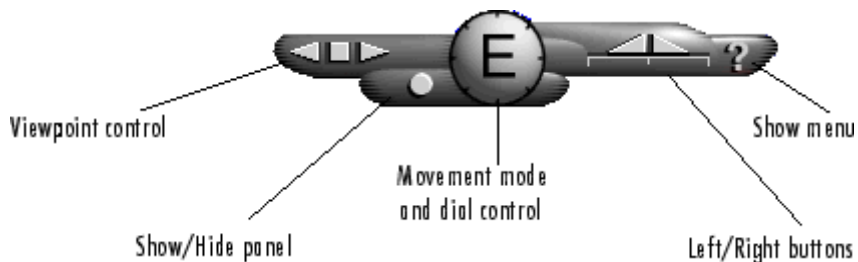
### In this section...

“Section Overview” on page 6-49  
 “Viewpoint Control” on page 6-50  
 “Control Menu” on page 6-50  
 “Navigation” on page 6-51  
 “Movement Modes” on page 6-52  
 “blaxxun Contact Settings” on page 6-52  
 “Stereoscopic Vision” on page 6-53

### Section Overview

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

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



### 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.

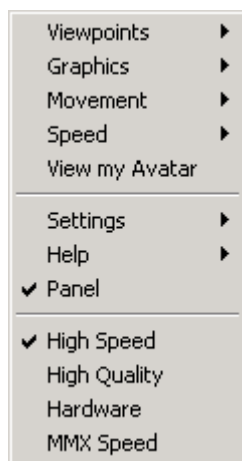
You use the other two triangular buttons to browse forward and backward through author-defined viewpoints of the virtual world. If the author does not define other viewpoints, these buttons are inactive. The keyboard shortcuts for the triangular buttons are the **Page Up** and **Page Down** keys.

### 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 following appears:





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

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

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

## 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.
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 software includes the blaxxun Contact VRML plug-in for Web browsers. The viewer allows you to select several working configurations, and its performance depends on several factors:

- The speed of your hardware
- System display driver settings

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

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

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 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.

# Virtual Reality Toolbox Stand-Alone Viewer

---

The Virtual Reality Toolbox stand-alone viewer, Orbisnap, allows you to visualize virtual worlds or prerecorded animation files without running the MATLAB or Virtual Reality Toolbox products.

- “What Is Orbisnap?” on page 7-2
- “Installing Orbisnap” on page 7-3
- “Using Orbisnap” on page 7-5
- “Orbisnap Interface” on page 7-10
- “Orbisnap Command Line” on page 7-17

## What Is Orbisnap?

The Virtual Reality Toolbox product includes Orbisnap. Orbisnap is a free, optional, stand-alone VRML97 viewer that does not require you to have either the MATLAB or Virtual Reality Toolbox products running. You can use Orbisnap to

- View prerecorded WRL animation files. For example, you might want to show prerecorded animation files in a meeting at which you do not have access to the MATLAB or Virtual Reality Toolbox products.
- Remotely view, from a client machine, a virtual world loaded in a current session of the Virtual Reality Toolbox product. For example, if you want to visualize a virtual world active in a Virtual Reality Toolbox session that is running on a computer in another part of the building, or across the network. This functionality allows you to remotely view a simulation, but not control it.
- View and navigate, but not simulate, a VRML world. You can navigate, render, and otherwise visualize a VRML world without simulating it.

Orbisnap is multiplatform. You can run Orbisnap on any of the platforms that the Virtual Reality Toolbox product supports. You do not need a MathWorks license to run Orbisnap.

# Installing Orbisnap

**In this section...**

“Section Overview” on page 7-3

“System Requirements” on page 7-3

“Copying Orbisnap to Another Location” on page 7-3

“Adding Shortcuts or Symbolic Links” on page 7-4

## Section Overview

The collection of Orbisnap files includes the Orbisnap starter file, Orbisnap executable file, and supporting files. These files are located under the Virtual Reality Toolbox orbisnap directory (for example, *matlabroot\toolbox\vr\orbisnap\bin* for the Windows platform). No further installation is necessary, but you might want to copy the Orbisnap files to another location or create shortcuts or symbolic links to the Orbisnap starter file for convenience.

## System Requirements

Orbisnap has the same hardware and software requirements as MATLAB. It is a multiplatform product that can run on PC-compatible computers with Windows or Linux. It can also run on Solaris hardware running UNIX, Apple Power Macintosh hardware running Mac OS X, and Hewlett-Packard™ hardware running HP-UX. See the following page on the MathWorks Web site:

<http://www.mathworks.com/products/matlab/requirements.html>

## Copying Orbisnap to Another Location

Orbisnap runs independently of the MATLAB and Virtual Reality Toolbox products. This means that you can copy Orbisnap to another location or even another machine. The following is a general procedure on how to copy Orbisnap to another location:

- 1 From a command line or GUI such as Windows Explorer, create a directory into which you can copy Orbisnap.

- 2 Copy all the files in the Orbisnap directory and its subdirectories. These files are likely located in the Virtual Reality Toolbox orbisnap directory, for example, `matlabroot\toolbox\vr\orbisnap` for the Windows platform.
- 3 Paste the files into the directory you created in step 1.

### **Adding Shortcuts or Symbolic Links**

For convenience, you can create a shortcut (Windows) or symbolic link (UNIX) to the Orbisnap starter file.

- In Windows Explorer, right-click `orbisnap.bat` and select **Properties**. You can start Orbisnap from either the shortcut or the original starter file.
- In UNIX, use the `ln -s` command to create a symbolic link to `orbisnap`.



## Using Orbisnap

### In this section...

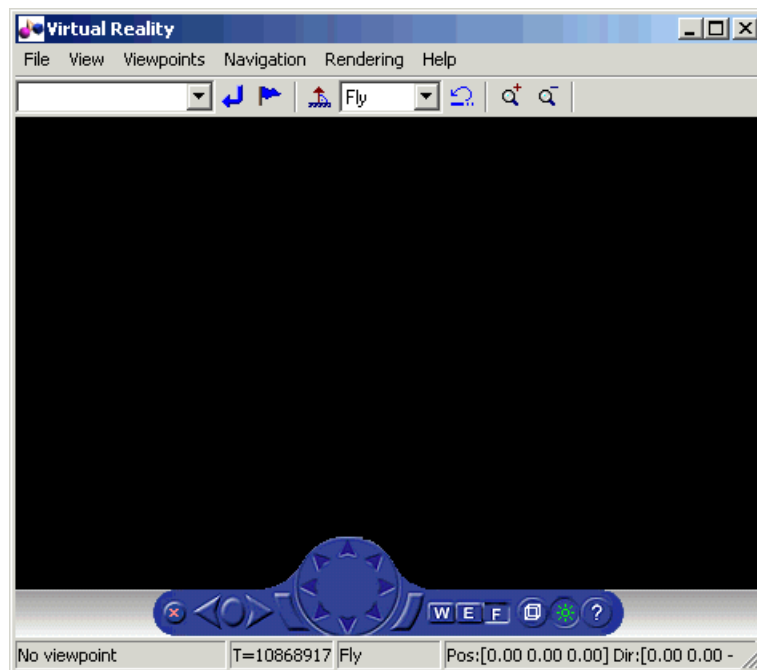
“Overview” on page 7-5

“Viewing Prerecorded WRL Animations or Virtual Worlds” on page 7-6

“Viewing the Virtual Reality Toolbox Server Virtual Worlds Remotely” on page 7-7

### Overview

Orbisnap looks like the following:



Orbisnap provides much of the functionality of the Virtual Reality Toolbox viewer. Using the menu bar, toolbar, and navigation panel, you can

- Customize the Orbisnap window
- Manage virtual world viewpoints
- Manage scene rendering

You cannot

- Open an editor for the virtual world
- Open another window for the virtual world
- Simulate the world (start/stop the model)
- Record or manage animation files

### Viewing Prerecorded WRL Animations or Virtual Worlds

This topic assumes that you have a prerecorded WRL animation file or an existing virtual world file. This procedure uses a file named `vr_bounce_anim.wrl`.

- 1** Start Orbisnap. For example, in Windows double-click `orbisnap.bat` in `matlabroot\toolbox\vr\orbisnap\bin`.

This is an Orbisnap starter file that calls the Orbisnap executable. Orbisnap is displayed.

- 2** In Orbisnap, select **File > Open**.

A file browser is displayed.

- 3** Browse to the directory that contains the prerecorded WRL animation file or virtual world you want to view.

- 4** Select the virtual world or prerecorded WRL file you want to view.

- 5** Click **Open**.

The file is displayed. If the file is an animation file, the simulation begins.

- 6** To close Orbisnap, select **File > Close**.

Using the menus, toolbar, and navigation panel, you can perform many of the same operations on the virtual world that you can with the Virtual Reality Toolbox viewer. See “Orbisnap Interface” on page 7-10 for an overview of the Orbisnap interface. See “Orbisnap Command Line” on page 7-17 for a description of the Orbisnap command-line options.

## Viewing the Virtual Reality Toolbox Server Virtual Worlds Remotely

To view virtual worlds from the Virtual Reality Toolbox server in Orbisnap, you must have

- The MATLAB software running a Virtual Reality Toolbox server session
- The version of the Virtual Reality Toolbox server to which you want to connect must be compatible with the Orbisnap version you are running. For example, you cannot connect Orbisnap to Virtual Reality Toolbox software Version 3.1.
- Network access between the client computer (running Orbisnap) and host computer (running MATLAB and Virtual Reality Toolbox server)

---

**Note** If you expect Orbisnap to access a virtual world on the Virtual Reality Toolbox server from a remote computer, you must make that virtual world available for Internet viewing. In the Virtual Reality Toolbox viewer for the virtual world you want to make available, select **Simulation > Block Parameters**, select the **Allow viewing from the Internet** check box, then click **OK**.

---

Note the following when using Orbisnap remotely:

- Although you can visualize a virtual world from the Virtual Reality Toolbox server in Orbisnap, any navigation or rendering in one viewer is not reflected in the other. For example, any navigation you do on the virtual world in Orbisnap is not reflected in the virtual world in the Virtual Reality Toolbox viewer, and vice versa.

- You cannot start or stop a simulation of the virtual world in Orbisnap. You can see a simulation on Orbisnap only if the virtual world is simulated in the Virtual Reality Toolbox server.
- The simulation might slow when you connect Orbisnap remotely to the Virtual Reality Toolbox server.

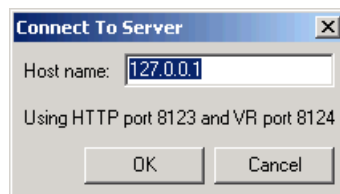
**1** Start Orbisnap. For example, in Windows, double-click `orbisnap.bat` in `matlabroot\toolbox\vr\orbisnap\bin`.

This is an Orbisnap starter file that calls the Orbisnap executable. Orbisnap is displayed.

**2** In Orbisnap, select **File > Connect to Server**.

The Connect to Server dialog is displayed.

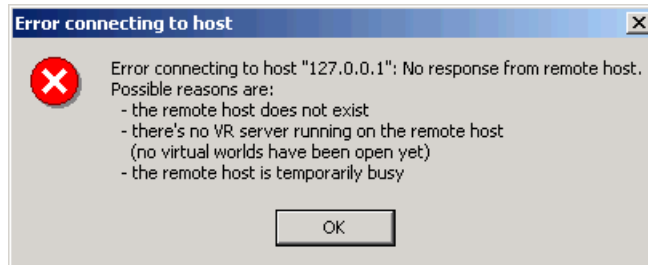
**3** Enter the IP address or hostname of the host computer running the Virtual Reality Toolbox server (127.0.0.1 by default). The HTTP port number is 8123 by default and the port number at which the Virtual Reality Toolbox server is listening is 8124 by default.



The Choose a world dialog is displayed. This dialog lists all the virtual worlds that are currently active on the Virtual Reality Toolbox server.



If no virtual world has ever been opened in this session of the Virtual Reality Toolbox server, Orbisnap displays a message. If you see this message, contact your counterpart running the Virtual Reality Toolbox server to better synchronize your activities. A virtual world must be fully active on the Virtual Reality Toolbox server for Orbisnap to remotely access it.



**4** Select a virtual world.

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

Orbisnap displays the selected virtual world of the remote Virtual Reality Toolbox server.

**6** Navigate and render the virtual world as you want.

**7** To close Orbisnap, select **File > Close**.

Using the menus, toolbar, and navigation panel, you can perform many of the same operations on the virtual world that you can with the Virtual Reality Toolbox viewer. See “Orbisnap Interface” on page 7-10 for a description of the Orbisnap interface. See “Orbisnap Command Line” on page 7-17 for a description of the Orbisnap command-line options.

## Orbisnap Interface

In this section...
“Menu Bar” on page 7-10
“Toolbar” on page 7-11
“Navigation Panel” on page 7-12

Using the menus, toolbar, and navigation panel, you can perform many of the same operations on the virtual world that you can with the Virtual Reality Toolbox viewer. For further details on using this interface, see Chapter 6, “Viewing Virtual Worlds”.

You can also download the PDF version of the Virtual Reality Toolbox user documentation from the MathWorks Web site:

[http://www.mathworks.com/access/helpdesk/help/pdf\\_doc/vr/vr.pdf](http://www.mathworks.com/access/helpdesk/help/pdf_doc/vr/vr.pdf)

### Menu Bar






The Orbisnap menu bar has the following menus:

- **File** — General file operation options, including,
  - **Open** — Invokes a browser that you can use to browse to the virtual world you want to visualize.
  - **Connect to server** — Allows you to connect to a Virtual Reality Toolbox server. Enter the IP address or hostname of the host computer running the Virtual Reality Toolbox server (127.0.0.1 by default) and the port number at which the Virtual Reality Toolbox server is listening (8124 by default).
  - **Reload** — Reloads the saved virtual world. Note that if you have created any viewpoints in this session, they are not retained unless you have saved those viewpoints with the **Save As** option.
  - **Save As** — Allows you to save the virtual world.
  - **Close** — Closes the Orbisnap window.
- **View** — Enables you to customize Orbisnap, including,

- **Toolbar** — Toggles the toolbar display.
- **Status Bar** — Toggles the status bar display at the bottom of Orbisnap. This display includes the current viewpoint, simulation time, navigation method, and the camera position and direction.
- **Navigation Zones** — Toggles the navigation zones on/off (see “Navigation” on page 7-13 for a description of how to use navigation zones).
- **Navigation Panel** — Controls the display of the navigation panel, including toggling it.
- **Zoom In/Out** — Zooms in or out of the world view.
- **Normal (100%)** — Returns the zoom to normal (initial viewpoint setting).
- **Viewpoints** — Manages the virtual world viewpoints.
- **Navigation** — Manages scene navigation.
- **Rendering** — Manages scene rendering.
- **Help** — Displays the Help browser for Orbisnap.

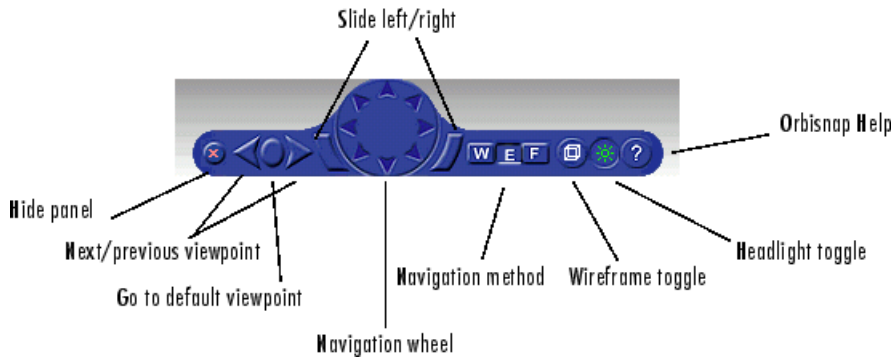
## Toolbar

The Orbisnap toolbar has buttons for some of the more commonly used operations available from the menu bar. These buttons include:

- Drop-down list that displays all the viewpoints in the virtual world
- Return to viewpoint button 
- Create viewpoint button 
- Straighten up button 
- Drop-down list that displays the navigation options Walk, Examine, and Fly
- Undo move button 
- Zoom in/out buttons 

## Navigation Panel

The Orbisnap navigation panel has navigation controls for some of the more commonly used navigation operations available from the menu bar. These controls include



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



## Navigation

You can navigate around a virtual world using the menu bar, toolbar, navigation panel, mouse, and keyboard.

**Navigation view** — You can change the camera position. From the menu bar, select the **Navigation** menu **Straighten Up** option. Alternatively, you can click the Straighten Up control from the toolbar or press **F9** on the keyboard. This option resets the camera so that it points straight ahead.

**Navigation methods** — Navigation with the mouse depends on the navigation method you select and the navigation zone you are in when you first click and hold down the mouse button. You can set the navigation method using one of the following:

- From the menu bar, select the **Navigation** menu **Method** option. This option provides three choices, Walk, Examine, or Fly. See the table Orbisnap Mouse Navigation on page 7-14.
- From the toolbar, select the drop-down menu that displays the navigation options Walk, Examine, and Fly.
- From the navigation panel, click the W, E, or F buttons.
- From the keyboard, press **Shift+W**, **Shift+E**, or **Shift+F**.

**Navigation zones** — You can view the navigation zones for a virtual world through the menu bar or keyboard.

From the menu bar, select the **View** menu **Navigation Zones** option. The virtual world changes as the navigation zones are toggled on and appear in the virtual world. Alternatively, from the keyboard, press the **F7** key.

The following table summarizes the behavior associated with the movement modes and navigation zones when you use your mouse to navigate through a virtual world. Turn the navigation zones on and experiment by clicking and dragging your mouse in the different zones of a virtual world.

### Orbisnap Mouse Navigation

Movement Mode	Zone and Description
Walk	<p><b>Outer</b> – Click and drag the mouse up, down, left, or right to slide the camera in any of these directions in a single plane.</p> <p><b>Inner</b> – Click and drag the mouse up and down to move forward and backward. Drag the mouse left and right to turn left or right.</p>
Examine	<p><b>Outer</b> – Click and drag the mouse up and down to move forward and backward. Drag the mouse left and right to slide left or right.</p> <p><b>Inner</b> – Click and drag the mouse to rotate the viewpoint around the origin of the scene.</p>
Fly	<p><b>Outer</b> – Click and drag the mouse to tilt the view either left or right.</p> <p><b>Inner</b> – Click and drag the mouse to pan the camera up, down, left, or right within the scene.</p> <p><b>Center</b> – Click and drag the mouse up and down to move forward and backward. Move the mouse left or right to turn in either of these directions.</p>

If your virtual world contains sensors, these sensors take precedence over mouse navigation at the sensor’s location. In this case, mouse navigation is still possible through the right or middle mouse buttons.

**Keyboard** — You can also use the keyboard to navigate through a virtual world. It can be faster and easier to issue a keyboard command, especially if you want to move the camera repeatedly in a single direction. The following table summarizes the keyboard commands and their associated navigation functions. Note that the letters presented do not need to be capitalized to perform their intended function.

## Orbisnap Keyboard Navigation

<b>Keyboard Command</b>	<b>Navigation Function</b>
<b>Backspace</b>	Undo move.
<b>F9</b>	Straighten up and make the camera stand on the horizontal plane of its local coordinates.
<b>+/-</b>	Zoom in/out.
<b>F6</b>	Toggle the headlight on/off.
<b>F7</b>	Toggle the navigation zones on/off.
<b>F5</b>	Toggle the wireframe option on/off.
<b>F8</b>	Toggle the antialiasing option on/off.
<b>Esc</b>	Go to default viewpoint.
<b>Home</b>	Return to current viewpoint.
<b>Page Up, Page Down</b>	Move between preset viewpoints.
<b>F10</b>	Toggle camera binding from the viewpoint.
<b>Shift+W</b>	Set the navigation method to Walk.
<b>Shift+E</b>	Set the navigation method to Examine.
<b>Shift+F</b>	Set the navigation method to Fly.
<b>Shift Up/Down Arrow</b>	Move the camera forward and backward.
<b>Up/Down Arrow</b>	Pan the camera up and down.
<b>Left/Right Arrow, Shift+Left/Right Arrow</b>	Pan the camera right and left.
<b>Alt+Up/Down Arrow</b>	Slide up and down.

**Orbisnap Keyboard Navigation (Continued)**

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

## Orbisnap Command Line

You can start Orbisnap from any command line with the following:

```
orbisnap
orbisnap -f vr_filename
orbisnap -c hostname -w "vrworld" -t http -v vrport -q=end_time
orbisnap -t http -v vrport vr_filename_or_hostname -q=end_time
orbisnap -h
```

No arguments – Starts the default Orbisnap. There is no loaded vrworld file and no connection to a Virtual Reality Toolbox server.

-f vr\_filename — (Optional) Orbisnap starts and loads the vrworld contained in vr\_filename. If you do not provide vr\_filename, Orbisnap prompts you for the filename.

-c hostname — (Optional) Orbisnap starts and connects to the Virtual Reality Toolbox server located at hostname. hostname can be a hostname or IP address. If you do not provide hostname, Orbisnap prompts you for the hostname.

-w vrworld — (Optional) Orbisnap starts, connects to the Virtual Reality Toolbox server, and loads the virtual world associated with the title "vrworld". If "vrworld" is not currently active in the Virtual Reality Toolbox server, the connection to the server does not succeed and the default Orbisnap starts.

-t http — (Optional) Orbisnap starts and connects to the Virtual Reality Toolbox server at this HTTP port (default 8123).

-t vrport — (Optional) Orbisnap starts and connects to the Virtual Reality Toolbox server listening at this port (default 8124).

vr\_filename\_or\_hostname — (Optional) Orbisnap starts and interprets this string first as a vrworld filename (for example, vrbounce.wr1). If the string is not a valid vrworld filename, Orbisnap tries to interpret the string as the name of the host that is running the Virtual Reality Toolbox server.

-q=end\_time — (Optional) Orbisnap ends when virtual scene time equals end\_time.

-h — (Optional) Orbisnap displays the Orbisnap command-line help.

# Block Reference

---

Control Input Devices (p. 8-1)

Utilities (p. 8-1)

Virtual Worlds (p. 8-2)

VRML-Related Signals (p. 8-2)

Process input from devices

Vector and matrix calculations

Virtual World utilities

VRML signal utilities

## Control Input Devices

Joystick Input

Process input from asynchronous joystick device

Space Mouse Input

Process input from space mouse device

## Utilities

Cross Product

Cross product of two 3-D vectors

Normalize Vector

Unit vector parallel to input vector

Rotation Between 2 Vectors

VRML rotation between two 3-D vectors

Rotation Matrix to VRML Rotation	Convert rotation matrix into representation used in VRML
Viewpoint Direction to VRML Orientation	Convert viewpoint direction to VRML orientation

## **Virtual Worlds**

VR Sink	Write data from Simulink model to virtual world
VR To Video	Write data from Simulink model to virtual world (video output port enabled)
VR Tracer	Trace trajectory of object in associated virtual scene

## **VRML-Related Signals**

VR Placeholder	Send unspecified value to Virtual Reality Toolbox block
VR Signal Expander	Expand input vectors into fully qualified VRML field vectors
VR Text Output	Allows display of Simulink signal values as text in VRML scene



# Blocks — Alphabetical List

---

# Cross Product

---

**Purpose** Cross product of two 3-D vectors

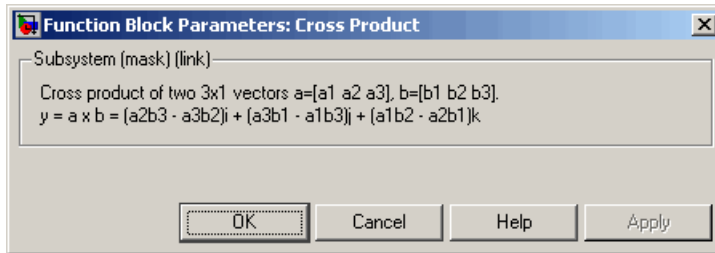
**Library** Virtual Reality Toolbox

**Description** Takes two 3-by-1 vectors as input and returns their cross product.



Cross Product

## Block Parameters Dialog Box



## Purpose

Process input from asynchronous joystick device

## Library

Virtual Reality Toolbox

## Description



Joystick Input

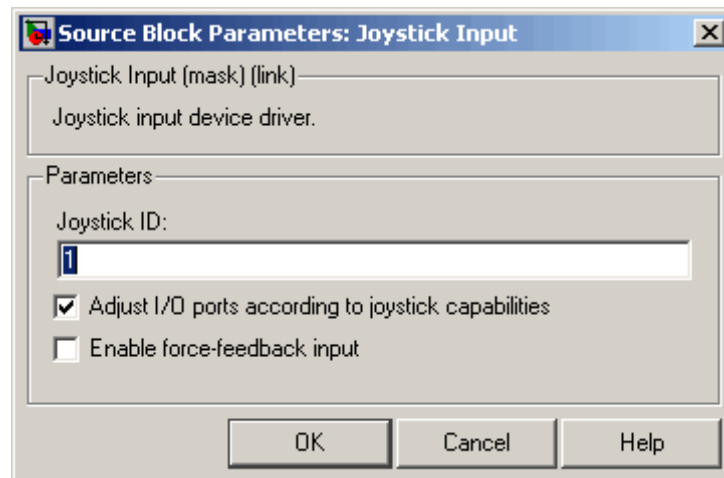
The Joystick Input block provides a convenient interaction between a Simulink model and the virtual world associated with a Virtual Reality Toolbox block.

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

The Joystick Input block also supports force-feedback devices.

When building a model using Real-Time Windows Target, the RTWin Joystick Input driver should be used instead of the Joystick Input block.

## Block Parameters Dialog Box



**Joystick ID** — The system ID assigned to the given joystick device. You can find the properties of the joystick connected to the system in the Game Controllers section of the system Control Panel.

# Joystick Input

---

**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 software 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, the Virtual Reality Toolbox software can support force-feedback joystick, steering wheel, and haptic (one that enables tactile feedback) devices. To use this feature, you must install DirectX® Version 8.0 or later.

**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.

Output Port	Value	Description
Axes	Vector of doubles in the range < -1; 1 >	Outputs correspond to the current position of the joystick in the given axis. Values are normalized to the range from -1 to 1.
Buttons	Vector of doubles 0 — Button released 1 — Button pressed	Outputs correspond to the current status of joystick buttons.
Point of view	-1 — Selector inactive <0; 360> — The angle of the POV selector, in degrees	Output corresponds to the current status of the joystick point-of-view selector.

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

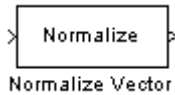
# Normalize Vector

---

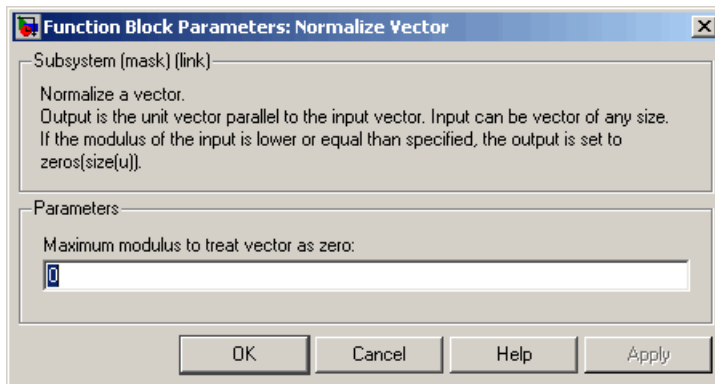
**Purpose** Unit vector parallel to input vector

**Library** Virtual Reality Toolbox

**Description** Takes an input vector of any size and outputs the unit vector parallel to it.



## Block Parameters Dialog Box



**Maximum modulus to treat vector as zero** — The output is set to zeroes if the modulus of the input is equal to or lower than this value.

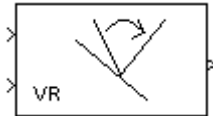
## Purpose

VRML rotation between two 3-D vectors

## Library

Virtual Reality Toolbox

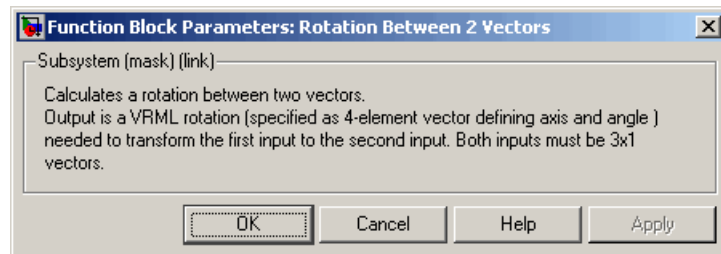
## Description



Rotation Between  
2 Vectors

Takes input of two 3-by-1 vectors and returns a VRML rotation (specified as a four-element vector defining axis and angle) that is needed to transform the first input vector to the second input vector.

## Block Parameters Dialog Box

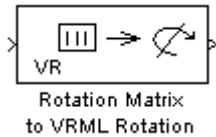


# Rotation Matrix to VRML Rotation

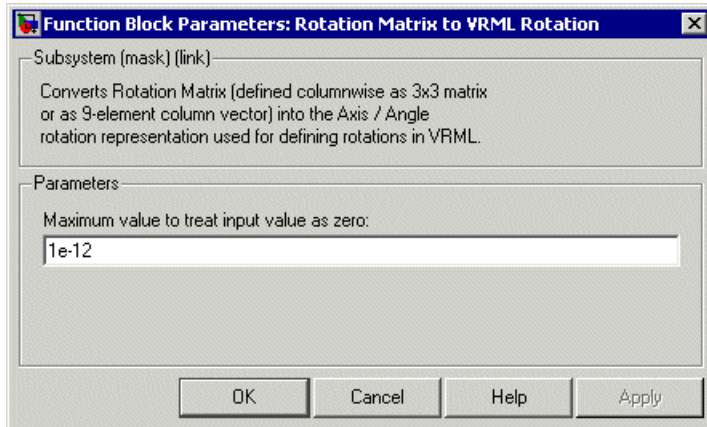
**Purpose** Convert rotation matrix into representation used in VRML

**Library** Virtual Reality Toolbox

**Description** Takes an input of a rotation matrix and outputs the axis/angle rotation representation used for defining rotations in VRML. The rotation matrix can be either a 9-element column vector or a 3-by-3 matrix defined columnwise.



## Block Parameters Dialog Box



**Maximum value to treat input value as zero** — The input is considered to be zero if it is equal to or lower than this value.

## Rotation Matrix

A representation of a three-dimensional spherical rotation as a 3-by-3 real, orthogonal matrix  $R$ :  $R^T R = R R^T = I$ , where  $I$  is the 3-by-3 identity and  $R^T$  is the transpose of  $R$ .

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



In general,  $R$  requires three independent angles to specify the rotation fully. There are many ways to represent the three independent angles. Here are two:

- You can form three independent rotation matrices  $R_1, R_2, R_3$ , each representing a single independent rotation. Then compose the full rotation matrix  $R$  with respect to fixed coordinate axes as a product of these three:  $R = R_3 * R_2 * R_1$ . The three angles are Euler angles.
- You can represent  $R$  in terms of an axis-angle rotation  $n = (n_x, n_y, n_z)$  and  $\theta$  with  $n * n = 1$ . The three independent angles are  $\theta$  and the two needed to orient  $n$ . Form the antisymmetric matrix:

$$\hat{J} = \begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}$$

Then Rodrigues' formula simplifies  $R$ :

$$R = \exp(\theta J) = I + J \sin \theta + J^2 (1 - \cos \theta)$$

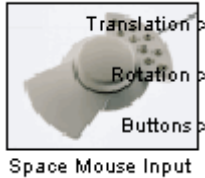
# Space Mouse Input

---

**Purpose** Process input from space mouse device

**Library** Virtual Reality Toolbox

## Description

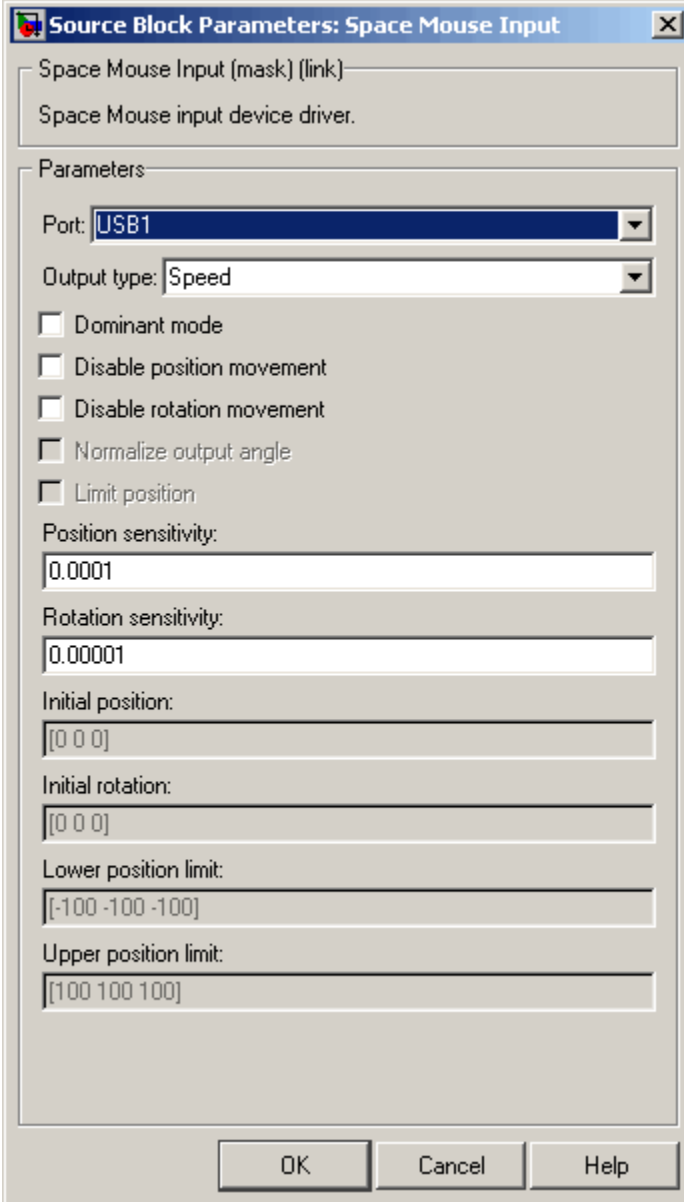


A space mouse is a device similar to a joystick in purpose, but it also provides movement control with six degrees of freedom. This block reads the status of the space mouse and provides some commonly used transformations of the input. The Space Mouse Input block supports current models of 3-D navigation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

## Data Type Support

The Space Mouse Input block outputs signals of type `double`.

## Block Parameters Dialog Box



The dialog box is titled "Source Block Parameters: Space Mouse Input". It contains the following fields and controls:

- Space Mouse Input (mask) (link): Space Mouse input device driver.
- Parameters section:
  - Port: USB1 (dropdown menu)
  - Output type: Speed (dropdown menu)
  - Dominant mode
  - Disable position movement
  - Disable rotation movement
  - Normalize output angle
  - Limit position
  - Position sensitivity: 0.0001
  - Rotation sensitivity: 0.00001
  - Initial position: [0 0 0]
  - Initial rotation: [0 0 0]
  - Lower position limit: [-100 -100 -100]
  - Upper position limit: [100 100 100]
- Buttons: OK, Cancel, Help

# Space Mouse Input

---

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

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

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

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

**Disable 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 Type** to **Speed**.

**Limit position** — Determines whether you can limit the upper and lower positions of the mouse.

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

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

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

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

**Lower position limit** — Position coordinates for the lower limit of the mouse.

**Upper position limit** — Position coordinates for the upper limit of the mouse.

### **See Also**

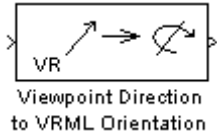
Manipulator with SpaceMouse Demo

# Viewpoint Direction to VRML Orientation

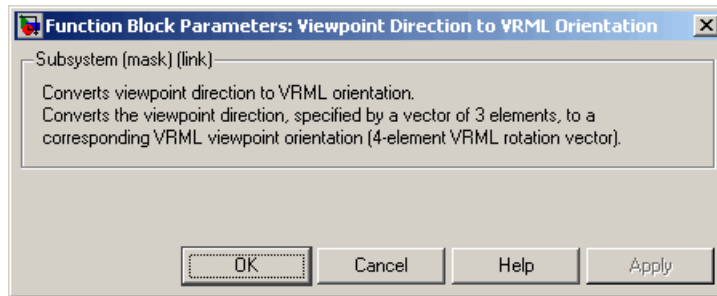
**Purpose** Convert viewpoint direction to VRML orientation

**Library** Virtual Reality Toolbox

**Description** Takes a viewpoint direction (3-by-1 vector) as input and outputs the corresponding VRML viewpoint orientation (four-element VRML rotation vector).



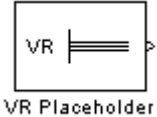
## Block Parameters Dialog Box



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

**Library** Virtual Reality Toolbox

**Description**

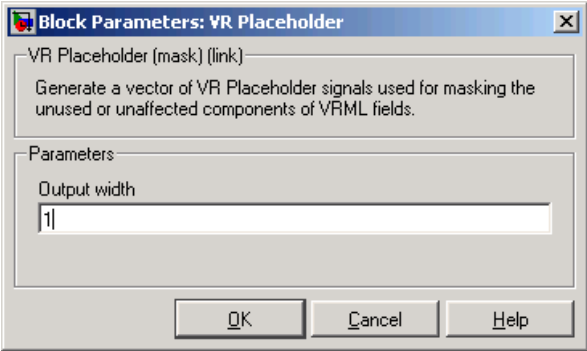


The VR Placeholder block sends out a special value that is interpreted as “unspecified” by the VR Sink block. When this value appears on the VR Sink input, whether as a single value or as an element of a vector, the appropriate value in the virtual world stays unchanged. Use this block to change only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

# VR Signal Expander

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

**Library** Virtual Reality Toolbox

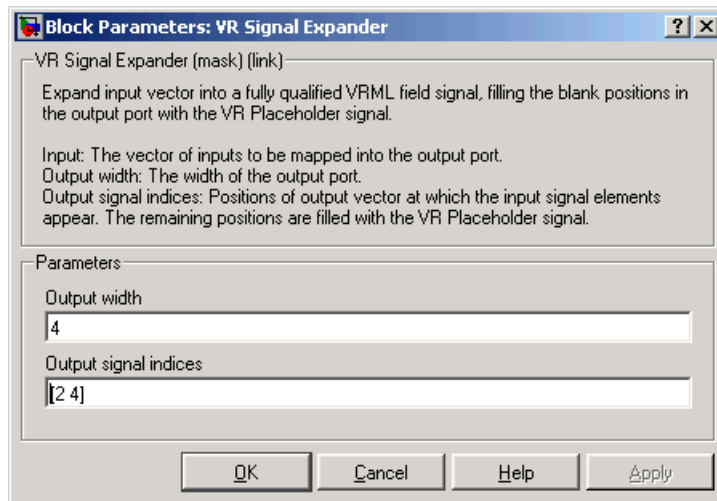
**Description** The VR Signal Expander block creates a vector of predefined length, using some values from the input ports and filling the rest with placeholder signal values.



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.



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

# VR Sink

---

## Purpose

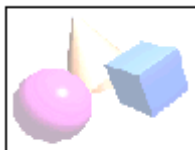
Write data from Simulink model to virtual world

## Library

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

The VR Sink block cannot be compiled by the Real-Time Workshop software, but it can be used as a SimViewing device on the host computer.

This block is equivalent to the VR To Video block, except that its **Show video output port** is unchecked by default.

---

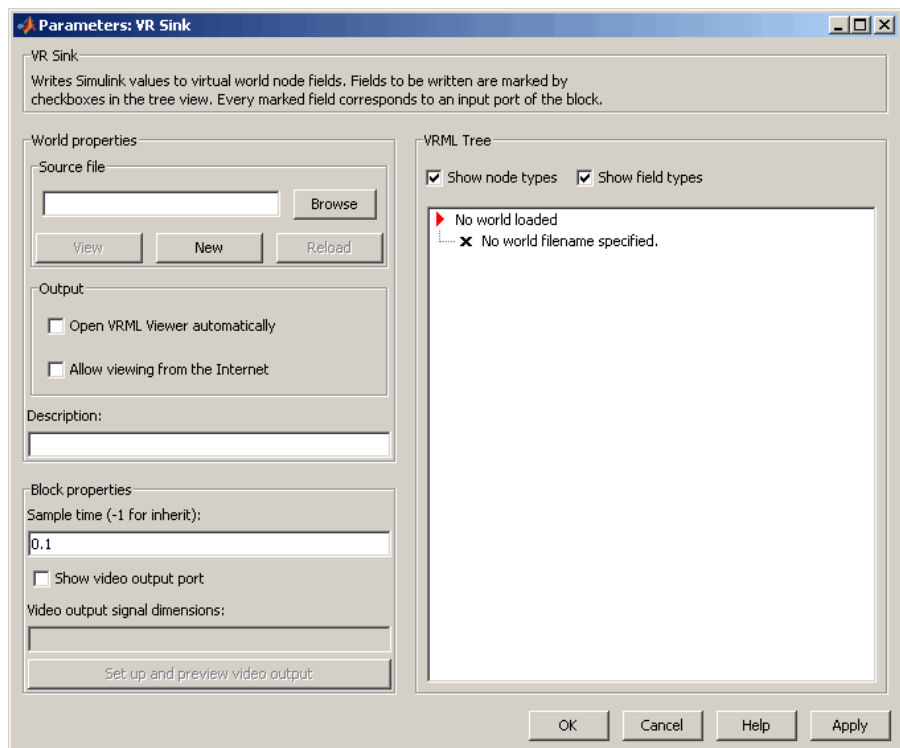
**Note** The current internal viewer window (`vrfigure`) properties are saved together with the Simulink model, so that the subsequent time you open the model, the internal viewer window opens with the same parameters at the time of saving, such as position, size, and navigation mode. When closing the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

---

## Data Type Support

A VR Sink block accepts all meaningful data types on input. The block converts these data types to natural VRML types as necessary. These data types include logicals, many types of signed and unsigned integers, singles, and doubles. The MATLAB and Simulink interfaces also accept matrices. For further details, see “VRML Field Data Types” on page 5-21.

## Block Parameters Dialog Box



**Source file** — VRML file name 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 software 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.

---

**Show video output port** — This enables a port to output an RGB video stream for further 2D video processing.

**Video output signal dimensions** — Dimensions ([vertical horizontal]) of the video output signal in pixels (default is [200 320])

**Setup and preview video output** — Opens a figure window for navigation and viewing.

**VRML tree** — This box shows the structure of the VRML file and the virtual world itself.

Nodes that have names are marked with red arrows and can be accessed from the MATLAB interface. Nodes without names, but whose children are named, are also marked with red arrows. This marking scheme makes it possible for you to find all accessible nodes by traversing the tree using arrows. Other nodes have a blue dot before their names.

Fields with settable values have check boxes. Use these check boxes to select the fields you want the Simulink software 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.

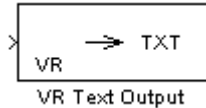
For an example of how to use the VR Sink block, see Foucault Pendulum Model with Virtual Reality Scene.

# VR Text Output

**Purpose** Allows display of Simulink signal values as text in VRML scene

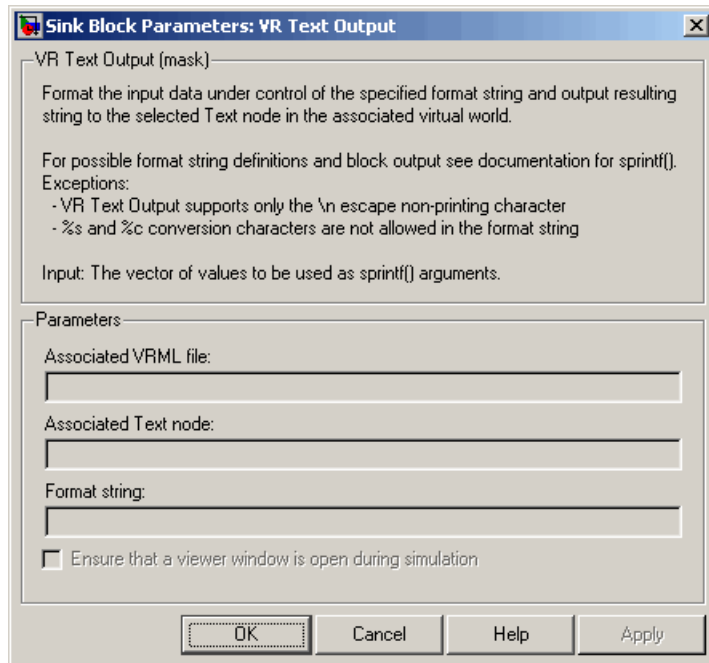
**Library** Virtual Reality Toolbox

**Description** The VR Text Output can display Simulink values of signal as text in a VRML scene.



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

## Block Parameters Dialog Box



**Associated VRML file** — VRML file specifying the virtual world to which text is output.

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

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

# VR To Video

---

## Purpose

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

## Library

Virtual Reality Toolbox

## Description

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

See the VR Sink block for details.



VR To Video



**Purpose** Trace trajectory of object in associated virtual scene

**Library** Virtual Reality Toolbox

**Description** Trace the trajectory of an object in the associated virtual scene.

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

You can specify one of three types of markers:

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

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

The first block input vector determines the position of the marker. The second block input (if enabled by the **Marker color selection** parameter) represents the marker color.

## Block Parameters Dialog Box

**Sink Block Parameters: VR Tracer**

VR Tracer (mask) (link)

Trace the trajectory of an object in the associated virtual scene.

Marker nodes are created in regular time steps as children of specified parent node, or at the top level of scene hierarchy. You can create three types of markers - pre-defined shape markers, line segments connecting object positions in every time step and axis-aligned triads useful for orienting the trajectory in the 3D space.

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

Input: First input vector determines the position of the marker, second input (if enabled) represents the marker color.

Parameters

Associated VRML file:

Parent node (leave empty for root):

Marker shape:

Connect markers with line segments

Place a triad at each marker position

Marker scale:

Marker color selection:

Sample time (-1 for inherited):

Ensure that a viewer window is open during simulation

OK Cancel Help Apply

**Associated VRML file** — VRML file name specifying the associated virtual world.

**Parent node (leave empty for root)** — Specify the location of the traced object in the scene hierarchy.

**Marker shape** — From the list, select one of:

- None
- Tetrahedron
- Pyramid
- Box
- Octahedron
- Sphere

**Connect markers with line segments** — Select this check box to connect the traced object path with lines.

**Place a triad at each marker position** — Select this check box to place a triad at each marker position. A triad helps you orient the object trajectory in the  $x$ - $y$ - $z$  plane.

**Marker scale** — Specify a three-component vector that defines the scaling of predefined marker shapes and triads. This parameter allows to accommodate for scenes of various sizes.

**Marker color selection** — From the list, select:

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

**Marker color** — If **Marker color selection** is Selected in block mask from color list, select the color from the list: yellow, magenta, cyan, red, green, blue, white, black

If **Marker color selection** is Defined in block mask as RGB values, enter RGB values for the marker color.

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

**Ensure that a viewer window is open during simulation** — Select this check box to ensure that the Virtual Reality Toolbox viewer is open during simulation.

# Function Reference

---

MATLAB Interface (p. 10-1)	Interface with virtual worlds and miscellaneous features
vrworld Object Methods (p. 10-2)	Interact with the virtual scene
vrnode Object Methods (p. 10-3)	Get and set the VRML node properties
vrfigure Object Methods (p. 10-4)	Get and set the Virtual Reality Toolbox viewer properties

## MATLAB Interface

<code>vrclear</code>	Remove all closed virtual worlds from memory
<code>vrclose</code>	Close virtual reality figure windows
<code>vrdir2ori</code>	Convert viewpoint direction to orientation
<code>vrdrawnow</code>	Update virtual world
<code>vrgetcbf</code>	Current callback <code>vrfigure</code> object
<code>vrgetcf</code>	Handle for active virtual reality figure
<code>vrgetpref</code>	Values of Virtual Reality Toolbox preferences
<code>vrinstall</code>	Install and check Virtual Reality Toolbox components

<code>vrjoystick</code>	Create joystick object
<code>vrlib</code>	Open Simulink block library for the Virtual Reality Toolbox product
<code>vrori2dir</code>	Convert viewpoint orientation to direction
<code>vrplay</code>	Play VRML animation file
<code>vrrotmat2vec</code>	Convert rotation from matrix to axis-angle representation
<code>vrrotvec2mat</code>	Convert rotation from axis-angle to matrix representation
<code>vrrotvec</code>	Calculate rotation between two vectors
<code>vrsetpref</code>	Change Virtual Reality Toolbox preferences
<code>vrspacemouse</code>	Create space mouse object
<code>vrview</code>	View virtual world using the Virtual Reality Toolbox software 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/addexternproto</code>	Add <code>externproto</code> declaration to virtual world.
<code>vrworld/close</code>	Close virtual world
<code>vrworld/delete</code>	Remove virtual world from memory

<code>vrworld/edit</code>	Open virtual world file in external VRML editor
<code>vrworld/get</code>	Property value of <code>vrworld</code> object
<code>vrworld/isvalid</code>	1 if <code>vrworld</code> object is valid, 0 if not
<code>vrworld/nodes</code>	List nodes available in virtual world
<code>vrworld/open</code>	Open virtual world
<code>vrworld/reload</code>	Reload virtual world from VRML file
<code>vrworld/save</code>	Write virtual world to VRML file
<code>vrworld/set</code>	Change property values of <code>vrworld</code> object
<code>vrworld/view</code>	View virtual world

## **vrnode Object Methods**

<code>vrnode</code>	Create node or handle to existing node
<code>vrnode/delete</code>	Remove <code>vrnode</code> object
<code>vrnode/fields</code>	VRML field summary of node object
<code>vrnode/get</code>	Property value of <code>vrnode</code> object
<code>vrnode/getfield</code>	Field value of <code>vrnode</code> object
<code>vrnode/isvalid</code>	1 if <code>vrnode</code> object is valid, 0 if not
<code>vrnode/set</code>	Change property of virtual world node
<code>vrnode/setfield</code>	Change field value of <code>vrnode</code> object
<code>vrnode/sync</code>	Enable or disable synchronization of VRML fields with client

## **vrfigure Object Methods**

<code>vrfigure</code>	Create new virtual reality figure
<code>vrfigure/capture</code>	Create RGB image from virtual reality figure
<code>vrfigure/close</code>	Close virtual reality figure
<code>vrfigure/get</code>	Property value of <code>vrfigure</code> object
<code>vrfigure/isvalid</code>	1 if <code>vrfigure</code> object is valid, 0 if not
<code>vrfigure/set</code>	Change property value of <code>vrfigure</code> object



# Functions — Alphabetical List

---

# vrclear

---

**Purpose** Remove all closed virtual worlds from memory

**Syntax** `vrclear`  
`vrclear(' -force')`

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

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

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

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

<b>Purpose</b>	Close virtual reality figure windows
<b>Syntax</b>	<code>vrclose</code> <code>vrclose all</code>
<b>Description</b>	<code>vrclose</code> and <code>vrclose all</code> close all the open virtual reality figures.
<b>Examples</b>	<p>Open a series of virtual reality figure windows by typing</p> <pre>vrpend vrbounce vrlights</pre> <p>Arrange the viewer windows so they are all visible. Type</p> <pre>vrclose</pre> <p>All the virtual reality figure windows disappear from the screen.</p>
<b>See Also</b>	<code>vrfigure/close</code>

# vrdir2ori

---

**Purpose** Convert viewpoint direction to orientation

**Syntax** `vrdir2ori(d)`  
`vrdir2ori(d,options)`

**Description** `vrdir2ori(d)` converts the viewpoint direction, specified by a vector of three elements, to an appropriate orientation (VRML rotation vector).  
`vrdir2ori(d,options)` converts the viewpoint direction with the default algorithm parameters replaced by values defined in `options`.  
The `options` structure contains the parameter `epsilon` that represents the value below which a number will be treated as zero (default value is `1e-12`).

**See Also** `vrori2dir`, `vrrotmat2vec`, `vrrotvec`, and `vrrotvec2mat`

**Purpose** Update virtual world

**Syntax** vrdrawnow

**Description** vrdrawnow removes from the queue pending changes to the virtual world and makes these changes to the scene in the viewer.

Changes to the scene are normally queued and the views are updated when

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

# vrfigure

---

**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`.

## Method Summary

Method	Description
capture	Create RGB image from virtual reality figure
close	Close virtual reality figure
get	Property value of <code>vrfigure</code> object
isvalid	1 if <code>vrfigure</code> object is valid, 0 if not
set	Change property value of <code>vrfigure</code> object

## Examples

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

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

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

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. This image can be displayed by the `image` command and subsequently printed.

**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

## vrfigure/close

---

**Purpose** Close virtual reality figure

**Syntax** `close(vrfigure_object)`

**Arguments** `vrfigure_object` Name of a figure object.

**Description** `close(vrfigure_object)` closes the virtual reality figure referenced by `vrfigure_object`. If `vrfigure_object` is a vector of `vrfigure` handles, then multiple figures are closed.

**Examples**

```
myworld = vrworld('vrpend.wrl')
open(myworld)
f = vrfigure(myworld)
close(f)
```

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

**Purpose** Property value of vrfigure object

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

**Arguments**

`vrfigure_object` Name of a vrfigure object.

`property_name` Name of the property.

**Description** `get(vrfigure_object)` lists all the properties of the vrfigure object. This is useful when you want to determine the current values of these properties. Use a command like the following to return a value of the specified property of the vrfigure object.

`x = get(vrfigure_object, 'property_name')` returns a value of the specified property of the vrfigure object.

The following are properties of vrfigure objects.

Property	Value	Description
Antialiasing	'off'   'on' Default: 'off'	Determines whether antialiasing is used when rendering scene. Antialiasing smooths textures by interpolating values between texture points. Read/write.
CameraBound	'off'   'on' Default: 'on'	Controls whether or not the camera moves with the current viewpoint. Read/write.
CameraDirection	Vector of three doubles	Specifies the camera direction relative to the direction of the current viewpoint. Read/write.
CameraDirectionAbs	Vector of three doubles	Specifies the camera direction in world coordinates. Read only.

## vrfigure/get

Property	Value	Description
CameraPosition	Vector of three doubles	Specifies the camera position relative to the position of the current viewpoint. Read/write.
CameraPositionAbs	Vector of three doubles	Specifies the camera position in world coordinates. Read only.
CameraUpVector	Vector of three doubles	Specifies the camera up vector relative to the up vector of the current viewpoint. Read/write.
CameraUpVectorAbs	Vector of three doubles	Specifies the camera up vector in world coordinates. Read only.
CaptureFileFormat	'tif'   'png' Default: 'tif'	Specifies file format for a captured frame file. Read/write.
CaptureFileName	String. Default: '%f_anim_%n.ext'	Specifies the frame capture filename. The string can contain tokens that are replaced by the corresponding information when the frame capture takes place. For further details, see “Frame Capture and Animation Recording File Tokens” on page 6-18. Read/write.
DeleteFcn	String	Specifies the callback invoked when closing the vrfigure object. Read/write.
Headlight	'off'   'on' Default: 'on'	Turns the headlight on or off. Read/write.

Property	Value	Description
Lighting	'off'   'on' Default: 'on'	Specifies whether the lighting is taken into account when rendering. If it is off, all the objects are drawn as if uniformly lit. Read/write.
MaxTextureSize	'auto'   $32 \leq x \leq$ video card limit, where x is a power of 2 (video card limit is typically 1024 or 2048)	Sets the maximum pixel size of a texture used in rendering vrfigure objects. The smaller the size, the faster the texture can render. Increasing this value improves image quality but decreases performance. A value of 'auto' sets the maximum possible pixel size. If the value you enter is unsuitable, a warning might trigger. The Virtual Reality Toolbox software then automatically adjusts the property to the next smaller suitable value.
Name	String	Specifies the name of this vrfigure object. Read/write.
NavMode	'fly'   'examine'   'walk' Default: 'examine'	Specifies navigation mode. Read/write.
NavPanel	'opaque'   'translucent'   'none'   'halfbar'   'bar' Default: 'halfbar'	Controls the appearance of the navigation panel in the Virtual Reality Toolbox viewer. Read/write.

## vrfigure/get

Property	Value	Description
NavSpeed	'very slow'   'slow'   'normal'   'fast'   'very fast' Default: 'normal'	Specifies navigation speed. Read/write.
NavZones	'off'   'on' Default: 'off'	Toggles navigation zones on/off. Read/write.
Position	Vector of four doubles	Specifies the screen coordinates of this vrfigure object. Read/write.
Record2D	'off'   'on' Default: 'off'	Enables 2-D offline animation file recording. Read/write.
Record2DCompress Method	' '   'auto'   'lossless'   'codec_code' Default: 'auto'	Specifies the compression method for creating 2-D animation files. The codec code must be registered in the system. See the MATLAB function documentation for avifile. Read/write.
Record2DCompress Quality	0-100 Default: '75'	Specifies the quality of 2-D animation file compression. Read/write.
Record2DFileName	String. Default: '%f_anim_%n.ext'	Specifies the 2-D offline animation 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' Default: 'on'	Toggles the status bar at the bottom of the Virtual Reality Toolbox viewer. Read/write.

Property	Value	Description
Textures	'off'   'on' Default: 'on'	Turns texture rendering on or off. Read/write.
Toolbar	'off'   'on' Default: 'on'	Toggles toolbar on the Virtual Reality Toolbox viewer. Read/write.
Transparency	'off'   'on' Default: 'on'	Specifies whether or not transparency information is taken into account when rendering. Read/write.
Viewpoint	String. If active viewpoint does not have a name, value is empty.	Specifies the vrfigure object's active viewpoint. Read/write.
Wireframe	'off'   'on' Default: 'off'	Specifies whether objects are drawn as solids or wireframes. Read/write.
World	vrworld object	Specifies the world this vrfigure object is displaying. Read only.
ZoomFactor	Double	Specifies the camera zoom factor. Read/write.

## Examples

Create a vrworld object:

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

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

```
open(myworld)
```

Create a vrfigure object:

## vrfigure/get

---

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

You can now get the object properties of the `vrfigure` object `f`:

```
get(f)
```

This returns the following object properties:

```
AntiAliasing = 'off'  
CameraBound = 'on'  
CameraDirection = [0 0 -1]  
CameraDirectionAbs = [0 -0.198669 -0.980067]  
CameraPosition = [0 0 0]  
CameraPositionAbs = [20 8 50]  
CameraUpVector = [0 1 0]  
CameraUpVectorAbs = [0 0.980067 -0.198669]  
CaptureFileFormat = 'tif'  
CaptureFileName = '%f_anim_%n.tif'  
DeleteFcn = ''  
Headlight = 'on'  
Lighting = 'on'  
MaxTextureSize = 2048  
Name = 'VR Car in the Mountains'  
NavMode = 'examine'  
NavPanel = 'halfbar'  
NavSpeed = 'normal'  
NavZones = 'off'  
Position = [5 92 576 380]  
Record2D = 'off'  
Record2DCompressMethod = 'auto'  
Record2DCompressQuality = 75  
Record2DFileName = '%f_anim_%n.avi'  
StatusBar = 'on'  
Textures = 'on'  
ToolBar = 'on'  
Transparency = 'on'  
Viewpoint = 'View 1 - Observer'
```



```
Wireframe = 'off'  
World = vrworld object: 1-by-1  
ZoomFactor = 1  
  Antialiasing = 'off'  
  CameraBound = 'on'  
  CameraDirection = [0 0 -1]  
  CameraDirectionAbs = [0 -0.198669 -0.980067]  
  CameraPosition = [0 0 0]  
  CameraPositionAbs = [20 4 50]  
  CameraUpVector = [0 1 0]  
  CameraUpVectorAbs = [0 0.980067 -0.198669]  
  Headlight = 'on'  
  Lighting = 'on'  
  Name = 'VR Car in the Mountains'  
  PanelMode = 'opaque'  
  Textures = 'on'  
  Transparency = 'on'  
  Viewpoint = 'View1'  
  Wireframe = 'off'  
  ZoomFactor = 1
```

**See Also** [vrfigure](#), [vrfigure/set](#)

# vrfigure/isvalid

---

<b>Purpose</b>	1 if vrfigure object is valid, 0 if not
<b>Syntax</b>	<code>x = isvalid(vrfigure_object_vector)</code>
<b>Arguments</b>	<code>vrfigure_object_vector</code> Name of an array of vrfigure objects.
<b>Description</b>	This method detects whether the vrfigure handles are valid and returns an array that contains a 1 where the vrfigure handles are valid and returns a 0 where they are not.
<b>See Also</b>	<code>vrnode/isvalid</code> , <code>vrworld/isvalid</code>

**Purpose** Change property value of vrfigure object

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

**Arguments**

- `vrfigure_object` Name of a vrfigure object.
- `property_name` Name of the property you want to set.
- `property_value` New value of the property.

**Description** The `set(vrfigure_object)` method allows you to set the property value of a vrfigure object. This method is useful when you want to change the value of a property.

The following are properties of vrfigure objects.

Property	Value	Description
Antialiasing	'off'   'on' Default: 'off'	Determines whether antialiasing is used when rendering scene. Antialiasing smooths textures by interpolating values between texture points. Read/write.
CameraBound	'off'   'on' Default: 'on'	Controls whether or not the camera moves with the current viewpoint. Read/write.
CameraDirection	Vector of three doubles	Specifies the camera direction relative to the direction of the current viewpoint. Read/write.
CameraDirectionAbs	Vector of three doubles	Specifies the camera direction in world coordinates. Read only.
CameraPosition	Vector of three doubles	Specifies the camera position relative to the position of the current viewpoint. Read/write.

## vrfigure/set

Property	Value	Description
CameraPositionAbs	Vector of three doubles	Specifies the camera position in world coordinates. Read only.
CameraUpVector	Vector of three doubles	Specifies the camera up vector relative to the up vector of the current viewpoint. Read/write.
CameraUpVectorAbs	Vector of three doubles	Specifies the camera up vector in world coordinates. Read only.
CaptureFileFormat	'tif'   'png' Default: 'tif'	Specifies file format for a captured frame file. Read/write.
CaptureFileName	String. Default: '%f_anim_%n.ext'	Specifies the frame capture filename. The string can contain tokens that are replaced by the corresponding information when the frame capture takes place. For further details, see “Frame Capture and Animation Recording File Tokens” on page 6-18. Read/write.
DeleteFcn	String	Specifies the callback invoked when closing the vrfigure object. Read/write.
Headlight	'off'   'on' Default: 'on'	Turns the headlight on or off. Read/write.
Lighting	'off'   'on' Default: 'on'	Specifies whether the lighting is taken into account when rendering. If it is off, all the objects are drawn as if uniformly lit. Read/write.

Property	Value	Description
MaxTextureSize	'auto'   $32 \leq x \leq$ video card limit, where x is a power of 2 (video card limit is typically 1024 or 2048)	Sets the maximum pixel size of a texture used in rendering vrfigure objects. The smaller the size, the faster the texture can render. Increasing this value improves image quality but decreases performance. A value of 'auto' sets the maximum possible pixel size. If the value you enter is unsuitable, a warning might trigger. The Virtual Reality Toolbox software then automatically adjusts the property to the next smaller suitable value.
Name	String	Specifies the name of this vrfigure object. Read/write.
NavMode	'fly'   'examine'   'walk' Default: 'examine'	Specifies navigation mode. Read/write.
NavPanel	'opaque'   'translucent'   'none'   'halfbar'   'bar' Default: 'halfbar'	Controls the appearance of the navigation panel in the Virtual Reality Toolbox viewer. Read/write.
NavSpeed	'very slow'   'slow'   'normal'   'fast'   'very fast' Default: 'normal'	Specifies navigation speed. Read/write.
NavZones	'off'   'on' Default: 'off'	Toggles navigation zones on/off. Read/write.
Position	Vector of four doubles	Specifies the screen coordinates of this vrfigure object. Read/write.

## vrfigure/set

Property	Value	Description
Record2D	'off'   'on' Default: 'off'	Enables 2-D offline animation file recording. Read/write.
Record2DCompress Method	' '   'auto'   'lossless'   'codec_code' Default: 'auto'	Specifies the compression method for creating 2-D animation files. The codec code must be registered in the system. See the MATLAB function documentation for <code>avifile</code> . Read/write.
Record2DCompress Quality	0-100 Default: '75'	Specifies the quality of 2-D animation file compression. Read/write.
Record2DFileName	String. Default: '%f_anim_%n.ext'	Specifies the 2-D offline animation 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' Default: 'on'	Toggles the status bar at the bottom of the Virtual Reality Toolbox viewer. Read/write.
Textures	'off'   'on' Default: 'on'	Turns texture rendering on or off. Read/write.
Toolbar	'off'   'on' Default: 'on'	Toggles toolbar on the Virtual Reality Toolbox viewer. Read/write.
Transparency	'off'   'on' Default: 'on'	Specifies whether or not transparency information is taken into account when rendering. Read/write.

Property	Value	Description
Viewpoint	String. If active viewpoint does not have a name, value is empty.	Specifies the vrfigure object's active viewpoint. Read/write.
Wireframe	'off'   'on' Default: 'off'	Specifies whether objects are drawn as solids or wireframes. Read/write.
World	vrworld object	Specifies the world this vrfigure object is displaying. Read only.
ZoomFactor	Double	Specifies the camera zoom factor. Read/write.

## Examples

Create a vrworld object.

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

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.

## **vrfigure/set**

---

### **See Also**

vrfigure, vrfigure/get



**Purpose** Current callback `vrfigure` object

**Syntax** `f = vrgcbf`

**Description** `f = vrgcbf` returns a `vrfigure` object representing the virtual reality figure that contains the callback currently being executed.

When no virtual reality figure callbacks are executing, `vrgcbf` returns an empty array of `vrfigure` objects.

# vrgcf

---

**Purpose** Handle for active virtual reality figure

**Syntax** `h = vrgcf`

**Description** `h = vrgcf` returns the handle of the current virtual reality figure. The current virtual reality figure is the currently active virtual reality figure window in which you can get and set the viewer properties. If no virtual reality figure exists, the MATLAB software 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** 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 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'.

Preference	Description
<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'.
<code>DataTypeFloat</code>	Specifies the handling of the VRML float data type for <code>vrnode/setfield</code> and <code>vrnode/getfield</code> . Valid values are 'single' and 'double'. If set to 'single', the VRML Float and Color data types are returned as 'single'. If set to 'double', the Float and Color data types are returned as 'double'. Default is 'double'.
<code>DefaultFigureAntiAliasing</code>	Determines whether antialiasing is used by default for new <code>vrfigure</code> objects. Valid values are 'off' and 'on'.
<code>DefaultFigureDeleteFcn</code>	Specifies the default callback invoked when closing a <code>vrfigure</code> object.
<code>DefaultFigureLighting</code>	Specifies whether the lights are rendered by default for new <code>vrfigure</code> objects. Valid values are 'off' and 'on'.

Preference	Description
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.

Preference	Description
DefaultFigureRecord2DFPS	Specifies the default frames per second playback speed.
DefaultFigureStatusBar	Specifies whether the status bar appears by default at the bottom of the Virtual Reality Toolbox viewer for new <code>vrfigure</code> objects. Valid values are 'off' and 'on'.
DefaultFigureToolBar	Specifies whether the toolbar appears by default on the Virtual Reality Toolbox viewer for new <code>vrfigure</code> objects. Valid values are 'off' and 'on'.
DefaultFigureTransparency	Specifies whether or not transparency information is taken into account when rendering for new <code>vrfigure</code> objects. Valid values are 'off' and 'on'.
DefaultFigureWireframe	Specifies whether objects are drawn as solids or wireframes by default for new <code>vrfigure</code> 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'.
DefaultWorldRecord3DFileName	Specifies the default 3-D animation filename for new <code>vrworld</code> objects.

Preference	Description
DefaultWorldRecordMode	Specifies the default animation recording mode for new vrworld objects. Valid values are 'manual' and 'scheduled'.
DefaultWorldRecordInterval	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 the MATLAB software before the change takes effect.
TransportBuffer	Length of the transport buffer (network packet overlay) for communication between the VR server and its clients.

Preference	Description
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 the MATLAB software before the change takes effect.

The `HttpPort`, `VrPort`, and `TransportBuffer` preferences affect Web-based viewing of virtual worlds. `DefaultFigurePosition` and `DefaultNavPanel` affect the 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, Virtual Reality Toolbox runs the VRML editor executable with all parameters required to edit the VRML file.

When you run the editor, 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.

**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

# vrgetpref

---

might need to configure the security network system (firewall) so that it allows connections on this port. The default value of this preference is 8124.

## **See Also**

vrsetpref

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

Action Value	Description
-selftest	Checks the integrity of the current installation. If this function reports an error, you should reinstall the Virtual Reality Toolbox software. The function <code>vrinstall</code> automatically does a self-test with any other actions.
-interactive	Checks for the installed components, and then displays a list of uninstalled components you can choose to install.
-check	Checks the installation of optional components. If the given component is installed, returns 1. If the given component is not installed, returns 0. If you do not specify a component, displays a list of components and their status.

# vrinstall

---

Action Value	Description
-install	Installs optional components. This action requires you to specify the component name. All components can be installed using this command, but some of them (currently only the plug-in) need to be uninstalled using the system standard uninstallation procedure.
-uninstall	Uninstalls optional components. This option is currently available for the editor only. Note that this action does not remove the files for the editor from the installation directory. It removes the editor registry information.  If you want to uninstall the VRML plug-in, exit the MATLAB software and, from the <b>Control Panel</b> window, select <b>Add/Remove Programs</b> .

## Examples

Install the VRML plug-in. This command starts the blaxxun Contact install program and installs the plug-in to your default Web browser.

```
vrinstall -install viewer
```

Install the VRML editor. This command associates V-Realm Builder with the **Edit** button in the Block Parameters dialog boxes.

```
vrinstall -install editor
```

**Purpose** Create joystick object

**Syntax**  
`joy = vrjoystick(id)`  
`joy = vrjoystick(id, 'forcefeedback')`

**Description** `joy = vrjoystick(id)` creates a joystick object capable of interfacing with a joystick device. The `id` parameter is a one-based joystick ID.

`joy = vrjoystick(id, 'forcefeedback')` enables force feedback if the joystick supports this capability.

## Methods

Method	Description
<code>axis</code>	<code>a = axis(joy, n)</code> reads the status of joystick with axis number <code>n</code> . Axis status is returned in the range of -1 to 1. The <code>n</code> parameter may be a vector to return multiple buttons.
<code>button</code>	<code>b = button(joy, n)</code> reads the status of joystick button number <code>n</code> . Button status is returned as logical 0 if not pressed and logical 1 if pressed. The <code>n</code> parameter may be a vector to return multiple buttons.
<code>caps</code>	<code>c = caps(joy)</code> returns joystick capabilities, such as the number of axes, buttons, POVs, and force-feedback axes. The return value is a structure with fields named <code>Axes</code> , <code>Buttons</code> , <code>POVs</code> , and <code>Forces</code> .
<code>close</code>	<code>close(joy)</code> closes and invalidates the joystick object. The object cannot be used once it is closed.

Method	Description
force	<code>force(joy, n, f)</code> applies force feedback to joystick axis <code>n</code> . The <code>n</code> parameter can be a vector to affect multiple axes. <code>f</code> values should be in range of -1 to 1, and the number of elements in <code>f</code> should either match the number of elements of <code>n</code> , or <code>f</code> can be a scalar to be applied to all the axes specified by <code>n</code> .
pov	<code>p = pov(joy, n)</code> reads the status of joystick POV (point of view) of control number <code>n</code> . <code>pov</code> is usually returned in degrees, with -1 meaning "not selected." <code>n</code> can be a vector to return multiple POVs.
read	<code>[axes, buttons, povs] = read(joy)</code> reads the status of axes, buttons, and POVs of the specified joystick. <code>[axes, buttons, povs] = read(joy, forces)</code> applies feedback forces, in addition, to a force-feedback joystick.

where `joy` is the handle to the joystick object.

<b>Purpose</b>	Open Simulink block library for the Virtual Reality Toolbox product
<b>Syntax</b>	<code>vrlib</code>
<b>Description</b>	<p>The Simulink library for the Virtual Reality Toolbox product has five blocks: VR Sink, VR Placeholder, VR Signal Expander, Joystick Input, and Space Mouse Input.</p> <p>Alternatively, you can access these blocks from a Simulink block diagram. In the Simulink window, from the <b>View</b> menu, click <b>Show Library Browser</b>.</p>

# vrnode

---

**Purpose** Create node or handle to existing node

**Syntax**

```
mynode = vrnode
mynode = vrnode([])
mynode = vrnode(vrworld_object, 'node_name')
mynode = vrnode(vrworld_object, 'node_name', 'node_type')
mynode = vrnode(vrworld_object, 'USE', othernode)
mynode = vrnode(parent_node, 'parent_field', 'node_name',
'node_type')
mynode = vrnode(parent_node, 'parent_field', 'USE',
'othernode')
```

**Arguments**

vrworld_object	Name of a vrworld object representing a virtual world.
node_name	Name of the node.
node_type	Type of the node.
parent_node	Name of the parent node that is a vrnode object.
parent_field	Name of the field of the parent node.
'USE'	Enables a USE reference to another node.
othernode	Name of another node for a USE reference.

**Description**

`mynode = vrnode` creates an empty vrnode handle that does not reference any node.

`mynode = vrnode([])` creates an empty array of vrnode handles.

`mynode = vrnode(vrworld_object, 'node_name')` creates a handle to an existing named node in the virtual world.

`mynode = vrnode(vrworld_object, 'node_name', 'node_type')` creates a new node called *node\_name* of type *node\_type* on the root of the virtual world. It returns the handle to the newly created node.



`mynode = vrnode(vrworld_object, 'USE', othernode)` creates a USE reference to the node `othernode` on the root of the world `vrworld_object`. It returns the handle to the virtual world to the original node.

`mynode = vrnode(parent_node, 'parent_field', 'node_name', 'node_type')` creates a new node called `node_name` of type `node_type` that is a child of the `parent_node` and resides in the field `parent_field`. It returns the handle to the newly created node.

`mynode = vrnode(parent_node, 'parent_field', 'USE', 'othernode')` creates a USE reference to the node `othernode` as a child of node `parentnode` and resides in the field `parentfield`. It returns the handle to the original node.

A `vrnode` object identifies a virtual world node in a way very similar to a handle. If you apply the `vrnode` method to a node that does not exist, the method creates a node, the `vrnode` object, and returns the handle to the `vrnode` object. If you apply the `vrnode` method to an existing node, the method returns the handle to the `vrnode` object associated with this node.

## Method Summary

Method	Description
<code>delete</code>	Remove <code>vrnode</code> object
<code>fields</code>	VRML field summary of node object
<code>get</code>	Property value of <code>vrnode</code> object
<code>getfield</code>	Field value of <code>vrnode</code> object
<code>isvalid</code>	1 if <code>vrnode</code> object is valid, 0 if not
<code>set</code>	Change property of virtual world node
<code>setfield</code>	Change field value of <code>vrnode</code> object
<code>sync</code>	Enable or disable synchronization of VRML fields with client

# vrnode

---

## **See Also**

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

<b>Purpose</b>	Remove vrnode object
<b>Syntax</b>	<code>delete(vrnode_object)</code> <code>delete(n)</code>
<b>Arguments</b>	<code>vrnode_object</code> Name of a vrnode object.
<b>Description</b>	<code>delete(vrnode_object)</code> deletes the virtual world node. <code>delete(n)</code> deletes the vrnode object referenced by the vrnode handle n. If n is a vector of vrnode handles, multiple nodes are deleted. As soon as a node is deleted, it and all its child objects are removed from all clients connected to the virtual world.
<b>See Also</b>	<code>vrworld/delete</code>

# vrnode/fields

---

**Purpose** VRML field summary of node object

**Syntax** `fields(vrnode_object)`  
`x = fields(vrnode_object)`

**Arguments** `vrnode_object` Name of a vrnode object representing the node to be queried.

**Description** `fields(vrnode_object)` displays a list of VRML fields of the node associated with the vrnode object in the MATLAB Command Window.  
`x = fields(vrnode_object)` returns the VRML fields of the node associated with the vrnode object in a structure array. The resulting structure contains a field for every VRML field with the following subfields:

- `Type` is the name of the VRML field type, for example, 'MFString', 'SFColor'.
- `Access` is the accessibility description of the VRML data class, for example, 'eventIn', 'exposedField'.
- `Sync` is the synchronization status 'on' or 'off'. See also `vrnode/sync`.

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

**Purpose** Property value of vrnode object

**Syntax**

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

**Arguments**

`vrnode_object` Name of a vrnode object representing the node to be queried.

`property_name` Name of the property to be read.

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

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

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

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

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

The vrnode object properties allow you to control the behavior and appearance of objects. The vrnode objects have the following properties. All these properties are read only.

Property	Value	Description
Fields	Cell array	Valid field names for the VRML node.
Name	String	Name of the node.

## vrnode/get

---

Property	Value	Description
Type	String	VRML type of the node. The value is a string (for example, 'Transform', 'Shape').
World	Handle	Handle of the parent vrworld object. This is a vrworld object that represents the node's parent world.

### See Also

[vrnode](#), [vrnode/getfield](#), [vrnode/set](#), [vrnode/setfield](#)

**Purpose**

Field value of vrnode object

**Syntax**

```
getfield(vrnode_object)
x = getfield(vrnode_object)
x = getfield(vrnode_object, 'fieldname')
```

**Arguments**

<code>vrnode_object</code>	Name of a vrnode object representing the node to be queried.
<code>fieldname</code>	Name of the vrnode object field whose values you want to query.

**Description**

`getfield(vrnode_object)` displays all the field names and their current values for the respective VRML node.

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

`x = getfield(vrnode_object, 'fieldname')` returns the value of the specified field for the node referenced by the `vrnode_object` handle. If `vrnode_object` is a vector of vrnode handles, `getfield` returns an M-by-1 cell array of values, where M is equal to `length(vrnode_object)`.

If `'fieldname'` is a 1-by-N or N-by-1 cell array of strings containing field names, `getfield` returns an M-by-N cell array of values.

---

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

---

**See Also**

`vrnode`, `vrnode/get`, `vrnode/set`, `vrnode/setfield`

# vrnode/isvalid

---

<b>Purpose</b>	1 if vrnode object is valid, 0 if not
<b>Syntax</b>	<code>x = isvalid(vrnode_object_vector)</code>
<b>Arguments</b>	<code>vrnode_object_vector</code> Name of an array of vrnode objects to be queried.
<b>Description</b>	<p>This method returns an array that contains 1 when the elements of <code>vrnode_object_vector</code> are valid vrnode objects, and 0 when they are not.</p> <p>The vrnode object is considered valid if the following conditions are met:</p> <ul style="list-style-type: none"><li>• The parent world of the node exists.</li><li>• The parent world of the node is open.</li><li>• The VRML node with the given vrnode handle exists in the parent world.</li></ul>
<b>See Also</b>	<code>vrfigure/isvalid</code> , <code>vrworld/isvalid</code>



**Purpose** Change property of virtual world node

**Syntax** `x = set(vrnode_object, 'property_name', 'property_value')`

**Arguments**

<code>vrnode_object</code>	Name of a vrnode object representing a node in the virtual world.
<code>property_name</code>	Name of a property.
<code>property_value</code>	Value of a property.

**Description** `x = set(vrnode_object, 'property_name', 'property_value')` changes the specified property of the vrnode object to the specified value.

The vrnode property values are case sensitive, while property names are not case sensitive.

The vrnode property values are case sensitive, while property names are not case sensitive.

The vrnode objects have the following properties. All these properties are read only.

Property	Value	Description
Fields	Cell array	Valid field names for the VRML node. Read only.
Name	String	Name of the node. Read only.
Type	String	VRML type of the node. The value is a string (for example, 'Transform', 'Shape'). Read only.
World	Handle	Handle of the parent vrworld object. This is a vrworld object that represents the node's parent world. Read only.

Currently, VRML nodes have no settable properties.

## **vrnode/set**

---

### **See Also**

`vrnode`, `vrnode/get`, `vrnode/getfield`, `vrnode/setfield`

**Purpose** Change field value of vrnode object

**Syntax** `x = setfield(vrnode_object, 'fieldname', 'fieldvalue')`

**Arguments**

<code>vrnode_object</code>	Name of a vrnode object representing the node to be changed.
<code>fieldname</code>	Name of the vrnode object VRML field whose values you want to set.
<code>fieldvalue</code>	Value of <code>fieldname</code> .

**Description** `x = setfield(vrnode_object, 'fieldname', 'fieldvalue')` changes the specified field of the vrnode object to the specified value. You can specify multiple field names and field values in one line of code by grouping them in pairs. For example, `x = setfield(vrnode_object, 'fieldname1', 'fieldvalue1', 'fieldname2', 'fieldvalue2', ...)`.  
Note that VRML field names are case sensitive, while property names are not.

---

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

---

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

# vrnode/sync

---

**Purpose** Enable or disable synchronization of VRML fields with client

**Syntax** `sync(vrnode_object, 'field_name', 'action')`

**Arguments**

<code>vrnode_object</code>	Name of a <code>vrnode</code> object representing the node.
<code>field_name</code>	Name of the VRML field to be synchronized.
<code>action</code>	The action parameter determines what should be done: <ul style="list-style-type: none"><li>• 'on' enables synchronization of this field.</li><li>• 'off' disables synchronization of this field.</li></ul>

**Description** The `sync` method controls whether the value of a VRML field is synchronized.

When the field is marked 'on', the field value is updated every time it is changed on the client computer. If the field is marked 'off', the host computer ignores the changes on the client computer.

Synchronized fields add more traffic to the network line because the value of the field must be resent by the client any time it is changed. Because of this, you should mark for synchronization only the fields you need to scan for changes made on clients (typically sensors). By default, fields are not synchronized and their values reflect only settings from MATLAB or the Simulink software.

Synchronization is meaningful only for readable fields. Readable fields are of VRML data class `eventOut` and `exposedField`. You cannot enable synchronization for `eventIn` or `nonexposed` fields.

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

<b>Purpose</b>	Convert viewpoint orientation to direction
<b>Syntax</b>	<code>vrori2dir(r)</code> <code>vrori2dir(r,options)</code>
<b>Description</b>	<p><code>vrori2dir(r)</code> converts the viewpoint orientation, specified by a rotation vector, <code>r</code>, to a direction the viewpoint points to.</p> <p><code>vrori2dir(r,options)</code> converts the viewpoint orientation with the default algorithm parameters replaced by values defined in <code>options</code>.</p> <p>The <code>options</code> structure contains the parameter <code>epsilon</code> that represents the value below which a number will be treated as zero (default value is <code>1e-12</code>).</p>
<b>See Also</b>	<code>vrdir2ori</code> , <code>vrrotmat2vec</code> , <code>vrrotvec</code> , and <code>vrrotvec2mat</code>

# vrplay

**Purpose** Play VRML animation file

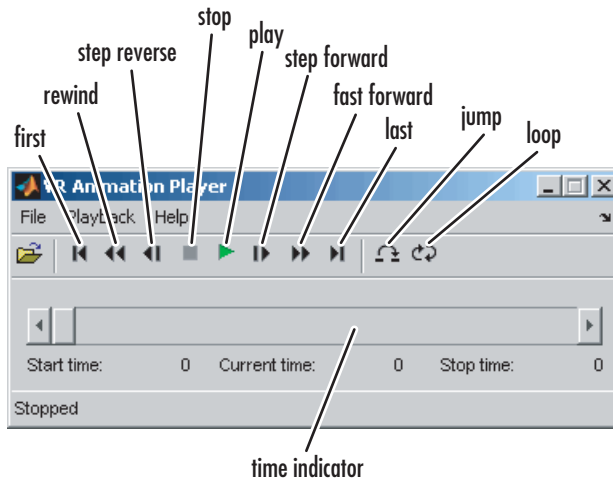
**Syntax**  
vrplay  
vrplay(filename)  
x=vrplay(filename)

**Description** vrplay opens the Virtual Reality animation player GUI that allows you to open and play VRML animation files.

vrplay(filename) opens the Virtual Reality animation player GUI and loads the virtual world filename.

x=vrplay(filename) also returns a Virtual Reality animation player GUI figure handle.

vrplay works only with VRML animation files created using the Virtual Reality Toolbox VRML recording functionality.



## Keyboard Support

The GUI's playback controls can also be accessed from the keyboard.

Key	Function
F, Page Down	Fast forward
J	Jump to time
L	Loop
P	Play/pause toggle
S	Stop
R, Page Up	Rewind
Right arrow key	Step forward
Left arrow key	Step reverse
Up arrow key	First
Down arrow key	Last

## Example

To play the animation file based on the `vr_octavia` demo, run `vrplay('octavia_scene_anim.wrl')`.

## See Also

“Recording Offline Animations” on page 4-10, `vrview`

# vrrotvec

---

**Purpose** Calculate rotation between two vectors

**Syntax** `r = vrrotvec(a,b)`  
`r = vrrotvec(a,b,options)`

**Description** `r = vrrotvec(a,b)` calculates a rotation needed to transform the 3D vector `a` to the 3D vector `b`.

`r = vrrotvec(a,b,options)` calculates the rotation with the default algorithm parameters replaced by values defined in `options`.

The `options` structure contains the parameter `epsilon` that represents the value below which a number will be treated as zero (default value is  $1e-12$ ).

The result, `r`, is a four-element axis-angle rotation row vector. The first three elements specify the rotation axis, and the last element defines the angle of rotation.

**See Also** `vrrotmat2vec` and `vrrotvec2mat`



<b>Purpose</b>	Convert rotation from matrix to axis-angle representation
<b>Syntax</b>	<code>r = vrrotmat2vec(m)</code> <code>r = vrrotmat2vec(m,options)</code>
<b>Description</b>	<p><code>r = vrrotmat2vec(m)</code> returns an axis-angle representation of rotation defined by the rotation matrix <code>m</code>.</p> <p><code>r = vrrotmat2vec(m,options)</code> converts the rotation with the default algorithm parameters replaced by values defined in <code>options</code>.</p> <p>The <code>options</code> structure contains the parameter <code>epsilon</code> that represents the value below which a number will be treated as zero (default value is <code>1e-12</code>).</p> <p>The result <code>r</code> is a four-element axis-angle rotation row vector. The first three elements specify the rotation axis, and the last element defines the angle of rotation.</p>
<b>See Also</b>	<code>vrrotvec</code> and <code>vrrotvec2mat</code>

# vrrotvec2mat

---

**Purpose** Convert rotation from axis-angle to matrix representation

**Syntax**  
`m = vrrotvec2mat(r)`  
`m = vrrotvec2mat(r,options)`

**Description** `m = vrrotvec2mat(r)` returns a matrix representation of the rotation defined by the axis-angle rotation vector, `r`.

`m = vrrotvec2mat(r,options)` returns a matrix representation of rotation defined by the axis-angle rotation vector `r`, with the default algorithm parameters replaced by values defined in `options`.

The `options` structure contains the parameter `epsilon` that represents the value below which a number will be treated as zero (default value is `1e-12`).

The rotation vector, `r`, is a row vector of four elements, where the first three elements specify the rotation axis, and the last element defines the angle.

To rotate a column vector of three elements, multiply it by the rotation matrix. To rotate a row vector of three elements, multiply it by the transposed rotation matrix.

**See Also** `vrrotvec` and `vrrotmat2vec`

**Purpose** Change Virtual Reality Toolbox preferences

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

**Arguments**

*preference\_name* Name of the preference.

*preference\_value* New value of the preference.

**Description** This function sets the given 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'.

Preference	Description
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.
DefaultFigureRecord2D CompressMethod	Specifies the default compression method for creating 2-D animation files for new <code>vrfigure</code> objects. Valid values are '', 'auto', 'lossless', and 'codec_code'.
DefaultFigureRecord2D CompressQuality	Specifies the default quality of 2-D animation file compression for new <code>vrfigure</code> objects. Valid values are 0-100.
DefaultFigureRecord2D FileName	Specifies the default 2-D offline animation filename for new <code>vrfigure</code> objects.
DefaultFigureRecord2DFPS	Specifies the default frames per second playback speed.
DefaultFigureStatusBar	Specifies whether the status bar appears by default at the bottom of the Virtual Reality Toolbox viewer for new <code>vrfigure</code> objects. Valid values are 'off' and 'on'.
DefaultFigureToolBar	Specifies whether the toolbar appears by default on the Virtual Reality Toolbox viewer for new <code>vrfigure</code> objects. Valid values are 'off' and 'on'.

Preference	Description
DefaultFigureTransparency	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'.
DefaultWorldRecord3DFileName	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'.
DefaultWorldRecordInterval	Specifies the default start and stop times for scheduled animation recording for new vrworld objects. Valid value is a vector of two doubles.
DefaultWorldRemoteView	Specifies whether the virtual world is enabled by default for remote viewing for new vrworld objects. Valid values are 'off' and 'on'.
DefaultWorldTimeSource	Specifies the default source of the time for new vrworld objects. Valid values are 'external' and 'freerun'.

Preference	Description
Editor	Path to the VRML editor. If this path is empty, the MATLAB editor is used.
HttpPort	IP port number used to access the VR server over the Web via HTTP. If you change this preference, you must restart the MATLAB software before the change takes effect.
TransportBuffer	Length of the transport buffer (network packet overlay) for communication between the 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 the MATLAB software before the change takes effect.

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

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`

# vrspacemouse

---

**Purpose** Create space mouse object

**Syntax** `mouse = vrspacemouse(id)`

**Description** `mouse = vrspacemouse(id)` creates a space mouse object capable of interfacing with a space mouse input device. The `id` parameter is a string that specifies the space mouse connection: COM1, COM2, COM3, COM4, USB1, USB2, USB3, or USB4.

The `vrspacemouse` object has several properties that influence the behavior of the space mouse input device. The properties can be read or modified using dot notation (e.g., `mouse.DominantMode = true;`).

**Properties** Valid properties are (property names are case-sensitive):

Property	Description
PositionSensitivity	Mouse sensitivity for translations. Higher values correspond to higher sensitivity.
RotationSensitivity	Mouse sensitivity for rotations. Higher values correspond to higher sensitivity.
DisableRotation	Fixes the rotations at initial values, allowing you to change positions only.
DisableTranslation	Fixes the positions at the initial values, allowing you to change rotations only.
DominantMode	If this property is true, the mouse accepts only the prevailing movement and rotation and ignores the others. This mode is very useful for beginners using a space mouse.
UpperPositionLimit	Position coordinates for the upper limit of the mouse.



<b>Property</b>	<b>Description</b>
LimitPosition	Enables mouse position limits. If false, the object ignores the UpperPositionLimit and LowerPositionLimit properties.
LowerPositionLimit	Position coordinates for the lower limit of the mouse.
NormalizeOutputAngle	Determines whether the integrated rotation angles should wrap on a full circle (360°). This is not used when you read the Output Type as Speed.
InitialPosition	Initial condition for integrated translations. This is not used when you set the Output Type to Speed.
InitialRotation	Initial condition for integrated rotations. This is not used when you set the Output Type to Speed.

**Methods**

<b>Method</b>	<b>Description</b>
button	<code>b = button(mouse, n)</code> reads the status of space mouse button number <code>n</code> . Button status is returned as logical 0 if not pressed and logical 1 if pressed. <code>n</code> can be a vector to return multiple buttons.
close	<code>close(mouse)</code> closes and invalidates the space mouse object. The object cannot be used once it is closed.

Method	Description
position	<code>p = position(mouse, n)</code> reads the position of space mouse axis number <code>n</code> . <code>n</code> can be a vector to return positions of multiple axes. Translations and rotations are integrated. Outputs are the position and orientation in the form of roll/pitch/yaw angles.
speed	<code>s = speed(mouse, n)</code> reads the speed of space mouse axis number <code>n</code> . <code>n</code> can be a vector to return the speeds of multiple axes. No transformations are done. Outputs are the translation and rotation speeds.
viewpoint	<code>p = viewpoint(mouse)</code> reads the space mouse coordinates in VRML viewpoint format. Translations and rotations are integrated. Outputs are the position and orientation in the form of an axis and an angle. You can use these values as viewpoint coordinates in VRML.

---

<b>Purpose</b>	View virtual world using the Virtual Reality Toolbox software viewer or Web browser
<b>Syntax</b>	<pre>vrview x = vrview('filename') x = vrview('filename', '-internal') x = vrview('filename', '-web')</pre>
<b>Description</b>	<p>vrview opens the default Web browser and loads the Virtual Reality Toolbox software Web page containing a list of virtual worlds available for viewing.</p> <p>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.</p> <p>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.</p> <p>x = vrview('filename', '-web') creates a virtual world associated with the .wrl file, opens the virtual world, and displays it in your Web browser.</p>
<b>See Also</b>	vrplay, vrworld, vrworld/open, vrworld/view

# vrwho

---

**Purpose** List virtual worlds in memory

**Syntax** vrwho  
x = vrwho

**Description** If you do not specify an output parameter, vrwho displays a list of virtual worlds in memory in the MATLAB Command Window.

If you specify an output parameter, vrwho returns a vector of handles to existing vrworld objects, including those opened from the Simulink interface.

**See Also** vrclear, vrwhos, vrworld

**Purpose** List details about virtual worlds in memory

**Syntax** vrwhos

**Description** vrwhos displays a list of virtual worlds currently in memory, with a description, in the MATLAB Command Window. The relation between vrwho and vrwhos is similar to the relation between who and whos.

**See Also** vrclear, vrwho

# vrworld

---

**Purpose** Create new vrworld object associated with virtual world

**Syntax**

```
myworld = vrworld('filename')
myworld = vrworld('filename', 'new')
myworld = vrworld
myworld = vrworld([])
```

**Arguments**

filename	String containing the name of the VRML file from which the virtual world is loaded. If no file extension is specified, the file extension <code>.wrl</code> is assumed.
'new'	Argument to create a virtual world associated with filename.

**Description**

`myworld = vrworld('filename')` creates a virtual world associated with the VRML file `filename` and returns its handle. If the virtual world already exists, a handle to the existing virtual world is returned.

`myworld = vrworld('filename', 'new')` creates a virtual world associated with the VRML file `filename` and returns its handle. It always creates a new virtual world object, even if another `vrworld` object associated with the same VRML file already exists.

`myworld = vrworld` creates an empty `vrworld` handle that does not refer to any virtual world.

`myworld = vrworld([])` returns an empty array of `vrworld` handles.

A `vrworld` object identifies a virtual world in a way very similar to a handle. All functions that affect virtual worlds accept a `vrworld` object as an argument to identify the virtual world.

If the given virtual world already exists in memory, the handle to the existing virtual world is returned. A second virtual world is not loaded into memory. If the virtual world does not exist in memory, it is loaded from the associated VRML file. The newly loaded virtual world is closed and must be opened before you can use it.

The vrworld object associated with a virtual world remains valid until you use either `delete` or `vrclear`.

## Examples

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

## Method Summary

Method	Description
<code>addexternproto</code>	Add externproto declaration to virtual world.
<code>close</code>	Close virtual world
<code>delete</code>	Remove virtual world from memory
<code>edit</code>	Open virtual world file in external VRML editor
<code>get</code>	Property value of vrworld object
<code>isvalid</code>	1 if vrworld object is valid, 0 if not
<code>nodes</code>	List nodes available in virtual world
<code>open</code>	Open virtual world
<code>reload</code>	Reload virtual world from VRML file
<code>save</code>	Write virtual world to VRML file
<code>set</code>	Change property values of vrworld object
<code>view</code>	View virtual world

## See Also

`vrworld/close`, `vrworld/delete`, `vrworld/open`

# vrworld/addexternproto

---

**Purpose** Add externproto declaration to virtual world.

**Syntax** `addexternproto(vrworld_object, protofile, protoname)`  
`addexternproto(vrworld_object, protofile, protoname, protodef)`

**Arguments**

<code>vrworld_object</code>	A <code>vrworld</code> object representing the virtual world.
<code>protofile</code>	String containing the name of the prototype file from which the EXTERNPROTO declaration is added.
<code>protoname</code>	String containing the name of the EXTERNPROTO declaration.
<code>protodef</code>	String containing a new name for the EXTERNPROTO declaration.

**Description** `addexternproto(vrworld_object, protofile, protoname)` adds an EXTERNPROTO declaration from file `protofile` to the virtual world. The handle `vrworld_object` refers to the virtual world. The EXTERNPROTO declaration is identified as `protoname`. If `protoname` is a cell array of identifiers, the function adds multiple EXTERNPROTOs from one file to the virtual world.

`addexternproto(vrworld_object, protofile, protoname, protodef)` adds an EXTERNPROTO declaration from file `protofile` to the virtual world. The handle `vrworld_object` refers to the virtual world. The EXTERNPROTO declaration is identified as `protoname`. If `protoname` is a cell array of identifiers, the function adds multiple EXTERNPROTOs from one file to the virtual world. This command then renames the new EXTERNPROTO declaration to `protodef`.

In both cases, the EXTERNPROTO declaration becomes equivalent to the VRML PROTO declaration. In other words, `protoname` or `protodef` becomes an internal PROTO type in the virtual scene associated with `vrworld_object`. After you save the virtual world, these PROTO declarations no longer require a reference to the original file, `protofile`, that contains the EXTERNPROTO declarations.



**Purpose** Close virtual world

**Syntax** `close(vrworld_object)`

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

**Description** This method changes the virtual world from an opened to a closed state:

- If the world was opened more than once, you must use an appropriate number of `close` calls before the virtual world closes.
- If `vrworld_object` is a vector of `vrworld` objects, all associated virtual worlds close.
- If the virtual world is already closed, `close` does nothing.

Opening and closing virtual worlds is a mechanism of memory management. When the system needs more memory and the virtual world is closed, you can discard its contents at any time.

Generally, you should close a virtual world when you no longer need it. This allows you to reuse the memory it occupied. The `vrworld` objects associated with this virtual world stay valid after it is closed, so the virtual world can be opened again without creating a new `vrworld` object.

**Examples**

```
myworld = vrworld('vrpend.wr1')
open(myworld)
close(myworld)
```

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

# vrworld/delete

---

**Purpose** Remove virtual world from memory

**Syntax** `delete(vrworld_object)`

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

**Description** The `delete` method removes from memory the virtual world associated with a `vrworld` object. The virtual world must be closed before you can delete it.

Deleting a virtual world frees the virtual world from memory and invalidates all existing `vrworld` objects associated with the virtual world.

If `vrworld_object` is a vector of `vrworld` objects, all associated virtual worlds are deleted.

You do not commonly use this method. One of the possible reasons to use this method is to ensure that a large virtual world is removed from memory before another memory-consuming operation starts.

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

<b>Purpose</b>	Open virtual world file in external VRML editor
<b>Syntax</b>	<code>edit(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>edit</code> method opens the VRML file associated with the <code>vrworld</code> object in a VRML editor. The <code>Editor</code> preference specifies the VRML editor to use. See <code>vrsetpref</code> for details on setting preferences.</p> <p>The VRML editor saves any changes you make directly to a virtual world file. If the virtual world is open,</p> <ul style="list-style-type: none"><li>• Use the <code>save</code> command in the VRML editor to save the changes to a virtual world file. In the MATLAB interface, the changes appear after you reload the virtual world.</li><li>• Use the <code>save</code> method in the MATLAB software to replace the modified VRML file. Any changes you made in the editor are lost.</li></ul>
<b>See Also</b>	<code>vrworld/reload</code> , <code>vrworld/save</code>

# vrworld/get

---

**Purpose** 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' Default: 'on'	Client cannot or can update the virtual scene. Read/write.

Property	Value	Description
Description	String. Default: automatically taken from the VRML file property title	Description of the virtual world as it appears on the main Web page. Read/write.
Figures	Vector of vrfigure objects	Vector of handles to 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' Default: 'off'	Indicates a closed or open virtual world. Read only.
Record3D	'off'   'on' Default: 'off'	Enables 3-D animation recording. Read/write.
Record3DFileName	String. Default: '%f_anim_%n.wrl'	3-D animation 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' Default: 'off'	Animation recording toggle. This property acts as the master recording switch. Read/write.
RecordMode	'manual'   'scheduled' Default: 'manual'	Animation recording mode. Read/write.

Property	Value	Description
RecordInterval	Vector of two doubles Default: [0 0]	Start and stop times for scheduled animation recording. Corresponds to the virtual world object Time property. Read/write.
RemoteView	'off'   'on' Default: 'off'	Remote access flag. If the virtual world is enabled for remote viewing, it is set to 'on'; otherwise, it is set to 'off'. Read/write.
Time	Double	Current time in the virtual world. Read/write.
TimeSource	'external'   'freerun' Default: 'external'	Source of the time for the virtual world. If set to 'external', time in the scene is controlled from the MATLAB interface (by setting the Time property) or the MATLAB interface (simulation time).  If set to 'freerun', time in the scene advances independently based on the system timer. Read/write.
View	'off'   'on' Default: 'on'	Indicates an unviewable or viewable virtual world. Read/write.

The `ClientUpdates` property is set to 'on' by default and can be set by the user. When it is set to 'off', the viewers looking at this virtual world should not update the view according to the virtual world changes. That is, the view is frozen until this property is changed to 'on'. This is useful for preventing tearing effects with complex animations. Before every animation frame, set `ClientUpdates` to 'off', make the appropriate modifications to the object positions, and then switch `ClientUpdates` back to 'on'.

The `Description` property defaults to '(untitled)' and can be set by the user. If the virtual world is loaded from a VRML file containing a

**WorldInfo** node with a `title` property, the `Description` property is loaded from the VRML file instead.

The `Nodes` property is valid only when the virtual world is open. If the virtual world is closed, `Nodes` always contains an empty vector.

The `RemoteView` property is set to 'off' by default and can be set by the user. If it is set to 'on', all viewers can access the virtual world through the Web interface. If it is set to 'off', only host viewers can access it.

The `View` property is set to 'on' by default and can be set by the user. When it is set to 'off', the virtual world is not accessible by the viewer. You rarely use this property.

## See Also

`vrworld`, `vrworld/set`

# vrworld/isvalid

---

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

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

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

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

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

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

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



**Purpose**

List nodes available in virtual world

**Syntax**

```
nodes(vrworld_object, '-full')  
x = nodes(vrworld_object, '-full')
```

**Arguments**

`vrworld_object` A `vrworld` object representing a virtual world.  
`'-full'` 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.wr1')
myworld2 = vrworld('vrpend.wr1')
```

Next, create an array of virtual world handles by typing

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

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

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

<b>Purpose</b>	Reload virtual world from VRML file
<b>Syntax</b>	<code>reload(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>reload</code> method reloads the virtual world from the VRML file associated with the <code>vrworld</code> 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.</p> <p><code>reload</code> forces all the clients currently viewing the virtual world to reload it. This is useful when there are changes to the VRML file.</p>
<b>See Also</b>	<code>vrworld/edit</code> , <code>vrworld/open</code> , <code>vrworld/save</code>

# 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/open`, `vrworld/reload`

**Purpose** Change property values of vrworld object

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

**Arguments**

`vrworld_object` Name of a vrworld object representing a virtual world.

`property_name` Name of the property.

`property_value` New value of the property.

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

Property	Value	Description
Clients	Scalar	Number of clients currently viewing the virtual world. Read only.
ClientUpdates	'off'   'on' Default: 'on'	Client cannot or can update the virtual scene. Read/write.
Description	String. Default: automatically taken from the VRML file property title	Description of the virtual world as it appears on the main Web page. Read/write.
Figures	Vector of vrfigure objects	Vector of handles to Virtual Reality Toolbox viewer windows currently viewing the virtual world. Read only.
FileName	String	Name of the associated VRML file. Read only.

Property	Value	Description
Nodes	Vector of vrnode objects	Vector of vrnode objects for all named nodes in the virtual world. Read only.
Open	'off'   'on' Default: 'off'	Indicates a closed or open virtual world. Read only.
Record3D	'off'   'on' Default: 'off'	Enables 3-D animation recording. Read/write.
Record3DFileName	String. Default: '%f_anim_%n.wrl'	3-D animation 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' Default: 'off'	Animation recording toggle. This property acts as the master recording switch. Read/write.
RecordMode	'manual'   'scheduled' Default: 'manual'	Animation recording mode. Read/write.
RecordInterval	Vector of two doubles Default: [0 0]	Start and stop times for scheduled animation recording. Corresponds to the virtual world object Time property. Read/write.
RemoteView	'off'   'on' Default: 'off'	Remote access flag. If the virtual world is enabled for remote viewing, it is set to 'on'; otherwise, it is set to 'off'. Read/write.
Time	Double	Current time in the virtual world. Read/write.

Property	Value	Description
TimeSource	'external'   'freerun' Default: 'external'	Source of the time for the virtual world. If set to 'external', time in the scene is controlled from the MATLAB interface (by setting the Time property) or the Simulink interface (simulation time).  If set to 'freerun', time in the scene advances independently based on the system timer. Read/write.
View	'off'   'on' Default: 'on'	Indicates an unviewable or viewable virtual world. Read/write.

**See Also**

vrworld, vrworld/get

# 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** `vrview`, `vrworld`



**simulation**

The process of running a dynamic system in nonreal time to observe its behavior.

**virtual figure object**

A handle to a Virtual Reality Toolbox viewer window.

**virtual node object**

A handle to a node in a virtual world that allows access to the node's properties.

**Virtual Reality Modeling Language**

The specification for displaying three-dimensional objects using a VRML viewer.

**virtual world**

An imaginary world where you can navigate around objects in three dimensions.

**virtual world object**

A handle to a virtual world that allows you to interact with and control the world.

**VRML**

Virtual Reality Modeling Language. See "VRML Overview" on page 1-11.



## Symbols and Numerics

- 2-D AVI files
  - recording through MATLAB interface 4-11
  - recording through Virtual Reality Toolbox viewer 6-24
- 3-D VRML files
  - recording with MATLAB interface 4-11
  - recording with Virtual Reality Toolbox viewer 6-23

## A

- adding Virtual Reality Toolbox blocks 3-2
- animation files
  - recording with MATLAB interface 4-10
  - recording with Virtual Reality Toolbox viewer 6-23
- associating virtual worlds with Simulink blocks 3-10

## B

- bitmap file formats 1-30
- blaxxun Contact
  - creating virtual worlds 5-9
  - installing 2-18
  - known issue 2-20
  - VRML viewer 6-49
- bmp file formats 1-30
- bouncing ball
  - Simulink example 1-18

## C

- capturing frames 6-18
- car
  - MATLAB interface example 1-25
- changing virtual world associated with Simulink block 3-10

- client computer
  - installation of VRML viewer (Windows) 2-48
  - system requirements 2-9
- closing virtual worlds 4-9
- components
  - client computer 2-48
  - host computer 2-12
- connecting Simulink model to a virtual world 5-17
- coordinate system
  - MATLAB 1-12
  - VRML 1-12
- creating vrworld object 4-2
- Cross Product
  - Simulink block 9-2

## D

- default editor
  - setting 2-28
- default viewer
  - setting 2-22
- deformation of a sphere example
  - adding Virtual Reality Toolbox blocks 5-6
  - connecting Simulink to a virtual world 5-17
  - creating a box in a virtual world 5-14
  - creating a sphere in a virtual world 5-9
  - defining the problem 5-5
- deleting virtual worlds 4-9
- displaying virtual worlds 3-12

## E

- editors
  - general 3-D 5-2
  - native VRML 5-2
  - uninstalling 2-46

## examples

- bouncing ball 1-18
- car 1-25
- deformation of a sphere 5-5
- geometry morphing 1-22
- heat transfer 1-25
- inverted pendulum 1-23
- lighting 1-20
- magnetic levitation 1-20
- magnetic levitation for Real-Time Windows Target 1-20
- manipulator with space mouse 1-21
- MATLAB interface 1-17
- plane taking off 1-24
- plane taking off with trajectory tracing 1-24
- rotating membrane 1-27
- Simulink interface 1-17
- solar system 1-23
- terrain visualization 1-28
- using MATLAB interface 1-25
- video output 1-22

**F**

## file format

- VRML 1-14

## files

- textures 1-30

## frame captures

- configuring 6-22
- creating 6-21
- introduction 6-18

## frames

- capture actions 6-23
- capturing 6-18
- configuring captures 6-22
- creating captures 6-21

## functions

- MATLAB interface 10-1
- vrclear 11-2
- vrclose 11-3
- vrdir2ori 11-4
- vrgcbf 11-25
- vrgcf 11-26
- vrinstall 11-35
- vrjoystick 11-37
- vrlib 11-39
- vrori2dir 11-53
- vrplay 11-54
- vrrotmat2vec 11-57
- vrrotvec 11-56
- vrrotvec2mat 11-58
- vrsetpref 11-59
- vrspacemouse 11-64
- vrview 11-67
- vrwho 11-68
- vrwhos 11-69

**G**

## geometry morphing

- Simulink example 1-22

**H**

## heat transfer

- MATLAB example 1-25

## history

- VRML 1-11

## host computer

- installing Virtual Reality Toolbox 2-12
- installing VRML editor (Windows) 2-27
- installing VRML viewer (UNIX) 2-21
- installing VRML viewer (Windows) 2-18
- required components 2-12
- system requirements 2-8
- Virtual Reality Toolbox viewer 2-17
- VRML editor (UNIX) 2-28

## I

### installation

- blaxxun Contact 2-18
  - client computer 2-48
  - components
    - host computer 2-12
  - host computer 2-12
  - supported platforms 2-6
  - system requirements 2-6
  - testing 2-49
  - viewer on host computer 2-17
  - Virtual Reality Toolbox 2-12
  - VRML editor (UNIX) 2-28
  - VRML editor (Windows) 2-27
  - VRML viewer (UNIX) 2-21
  - VRML viewer (Windows) 2-18
- interacting with a virtual world 4-5
- interface overview 3-2
- inverted pendulum
  - Simulink example 1-23

## J

### Joystick Input

- Simulink block 9-3

## L

### lighting

- Simulink example 1-20

## M

### magnetic levitation

- Simulink example 1-20
- Simulink example for Real-Time Windows Target 1-20

### manipulator with space mouse

- Simulink example 1-21

### MATLAB coordinate system 1-12

### MATLAB interface

- creating a vrworld object 4-2
- examples 1-25
- interacting with a virtual world 4-5
- table of general functions 10-1

## N

### native VRML 5-2

### navigation

- about a virtual scene 6-11
- example of navigation 6-15
- keyboard 6-17
- using the mouse 6-12

### navigation speed

- changing 6-14

### network security setting 2-20

- changing default 2-20

*See also* blaxxun Contact

### Normalize Vector

- Simulink block 9-6

## O

### opening a viewer window 3-14

### Orbisnap 7-1

- installation 7-3
- interface 7-10
- remote viewing 7-7
- usage 7-5
- viewing virtual worlds 7-6

## overview

- associating virtual worlds with Simulink 3-2
- Simulink interface 3-2
- virtual worlds 5-2
- VRML 1-11
- VRML editing tools 5-2

**P**

## plane taking off

- Simulink example 1-24

## plane taking off with trajectory tracing

- Simulink example 1-24

## platforms

- supported 2-6

## preferences 2-34

- See also* Virtual Reality Toolbox preferences

**R**

## rendering of a virtual scene 6-40

## rotating membrane

- Simulink example 1-22
- Virtual Reality Toolbox example 1-27

## Rotation Between 2 Vectors

- Simulink block 9-7

## Rotation Matrix to VRML Rotation

- Simulink block 9-8

## running Simulink example 2-49

**S**

## security settings

- changing 2-20

## server

- Virtual Reality Toolbox 1-31

## setting

- default editor 2-28
- default viewer 2-22

## simulation

- displaying virtual worlds 3-12
- starting 3-12

## Simulink 3-2

- associating with virtual worlds 3-2
- interface examples 1-17
- See also* examples

## Simulink blocks

- adding Virtual Reality Toolbox blocks 3-2
- changing virtual world association 3-10
- Cross Product 9-2
- Joystick Input 9-3
- Normalize Vector 9-6
- Rotation Between 2 Vectors 9-7
- Rotation Matrix to VRML Rotation 9-8
- Viewpoint Direction to VRML Orientation 9-14
- VR Placeholder 9-15
- VR Signal Expander 9-16
- VR Sink 9-18
- VR Text Output 9-22
- VR to Video 9-24
- VR Tracer 9-25

## Simulink interface examples

- bouncing ball 1-18
- deformation of a sphere 5-6
- geometry morphing 1-22
- inverted pendulum 1-23
- lighting 1-20
- magnetic levitation 1-20
- magnetic levitation with Real-Time Windows Target 1-20
- manipulator with space mouse 1-21
- plane taking off 1-24
- plane taking off with trajectory tracing 1-24
- rotating membrane 1-22
- running and viewing 2-49
- solar system 1-23
- vehicle dynamics visualization 1-22
- video output 1-22

- Simulink interface overview 3-2
- solar system
  - Simulink example 1-23
- space mouse
  - Simulink block 9-10
  - Simulink examples 1-21
- supported platforms 2-6
- system requirements
  - client computer 2-9
  - host computer 2-8

## T

- terrain visualization
  - Virtual Reality Toolbox example 1-28
- testing
  - installation 2-49
  - MATLAB example 2-54
  - Simulink example 2-49
- textures 1-30

## U

- uninstalling
  - editor 2-46
  - V-Realm Builder 2-46
  - Virtual Reality Toolbox 2-46
  - VRML viewer (Windows) 2-46

## V

- V-Realm Builder
  - installing 2-27
  - uninstalling 2-46
  - VRML editor 5-4
- vehicle visualization
  - Simulink example 1-22
- video output
  - 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 2-48
  - installation on host computer 2-17
  - opening 3-14
- viewpoint control 6-33
- Viewpoint Direction to VRML Orientation
  - Simulink block 9-14
- Virtual Reality Toolbox
  - description 1-2
  - features 1-4
- Virtual Reality Toolbox blocks
  - Cross Product 9-2
  - Joystick Input 9-3
  - Normalize Vector 9-6
  - Rotation Between 2 Vectors 9-7
  - Rotation Matrix to VRML Rotation 9-8
  - Viewpoint Direction to VRML
    - Orientation 9-14
  - VR Placeholder 9-15
  - VR Signal Expander 9-16
  - VR Sink 9-18
  - VR Text Output 9-22
  - VR to Video 9-24
  - VR Tracer 9-25
- Virtual Reality Toolbox examples
  - car 1-25
  - heat transfer 1-25
  - rotating membrane 1-27
  - running and viewing 2-54
  - terrain visualization 1-28
- Virtual Reality Toolbox preferences
  - MATLAB GUI 2-34
  - vrsetpref function 11-59
- Virtual Reality Toolbox stand-alone viewer 7-1
  - See also* Orbisnap

- Virtual Reality Toolbox viewer
  - changing navigation speed 6-14
  - introduction 6-2
  - navigation 6-11
  - rendering 6-40
  - viewpoint control 6-33
- virtual worlds
  - associating with Simulink 3-2
  - closing 4-9
  - deleting 4-9
  - displaying 3-12
  - interacting with 4-5
  - overview 5-2
- VR Placeholder
  - Simulink block 9-15
- VR Signal Expander
  - Simulink block 9-16
- VR Sink
  - Simulink block 9-18
- VR Text Output
  - Simulink block 9-22
- VR to Video
  - Simulink block 9-24
- VR Tracer
  - Simulink block 9-25
- vrclear
  - Virtual Reality Toolbox function 11-2
- vrclose
  - Virtual Reality Toolbox function 11-3
- vrdir2ori
  - Virtual Reality Toolbox function 11-4
- vrgcbf
  - Virtual Reality Toolbox function 11-25
- vrgcf
  - Virtual Reality Toolbox function 11-26
- vrinstall
  - Virtual Reality Toolbox function 11-35
- vrjoystick
  - Virtual Reality Toolbox function 11-37
- vrllib
  - Virtual Reality Toolbox function 11-39
- VRML
  - coordinate system 1-12
  - file format 1-14
  - history 1-11
  - overview 1-11
- VRML editor
  - general 5-2
  - installing on host computer (Windows) 2-27
  - on UNIX platforms 2-28
  - V-Realm Builder 5-4
- VRML viewer
  - blaxxun Contact 6-49
  - changing navigation speed 6-14
  - installing on client computer (Windows) 2-48
  - installing on host computer (UNIX) 2-21
  - installing on host computer (Windows) 2-18
  - known issue (blaxxun Contact) 2-20
  - navigation 6-11
  - rendering 6-40
  - uninstalling 2-46
  - viewpoint control 6-33
  - Virtual Reality Toolbox 6-2
- vrori2dir
  - Virtual Reality Toolbox function 11-53
- vrplay
  - Virtual Reality Toolbox function 11-54
- vrrotmat2vec
  - Virtual Reality Toolbox function 11-57
- vrrotvec
  - Virtual Reality Toolbox function 11-56
- vrrotvec2mat
  - Virtual Reality Toolbox function 11-58
- vrsetpref
  - Virtual Reality Toolbox function 11-59
- vrspacemouse
  - Virtual Reality Toolbox function 11-64
- vrview
  - Virtual Reality Toolbox function 11-67



vrwho

Virtual Reality Toolbox function 11-68

vrwhos

Virtual Reality Toolbox function 11-69

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