Virtual Reality Toolbox 4
User's Guide

MATLAB®
&SIMULINK®

The MathWorks™
Accelerating the pace of engineering and science
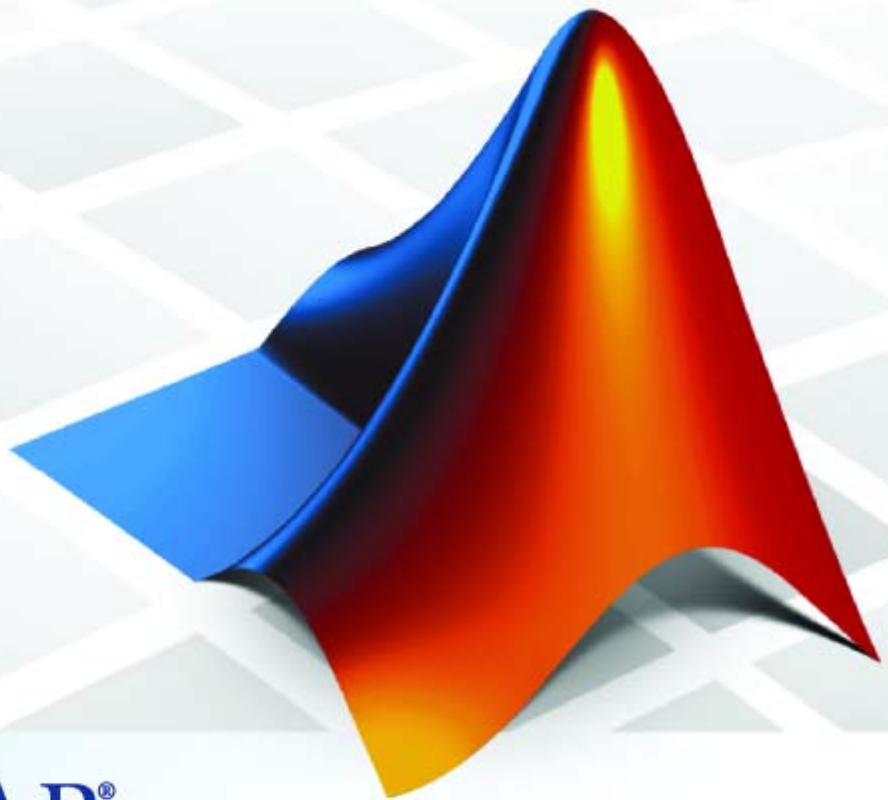
How to Contact The MathWorks

	www.mathworks.com	Web
	comp.soft-sys.matlab	Newsgroup
	www.mathworks.com/contact_TS.html	Technical Support
@	suggest@mathworks.com	Product enhancement suggestions
	bugs@mathworks.com	Bug reports
	doc@mathworks.com	Documentation error reports
	service@mathworks.com	Order status, license renewals, passcodes
	info@mathworks.com	Sales, pricing, and general information

☎ 508-647-7000 (Phone)

508-647-7001 (Fax)

✉ The MathWorks, Inc.
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

Trademarks

Patents

Contents

Getting Started

1

2

Simulink Interface

3

4

5

Viewing Virtual Worlds

6

7

Virtual Reality Toolbox Stand-Alone Viewer

8

Blocks — By Category

Blocks — Alphabetical List

9

Functions — By Category

10

Functions — Alphabetical List

11

Glossary

Index

Getting Started

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

What Is Virtual Reality Toolbox?

In this section...
"Overview" on page 1-2
"Expected Background" on page 1-2

Overview

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

- Virtual worlds — Create virtual worlds or three-dimensional scenes using standard Virtual Reality Modeling Language (VRML) technology.

- Dynamic systems — Create and define dynamic systems with MATLAB and Simulink.

- Animation — View moving three-dimensional scenes driven by signals from the Simulink environment.

- Manipulation — Change the positions and properties of objects in a virtual world while running a simulation.

To provide a complete working environment, Virtual Reality Toolbox 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 — For Windows platforms, use V-Realm Builder to create and edit VRML code. For UNIX or Linux platforms, use the MATLAB text editor to write VRML code to create virtual worlds.

Expected Background

To help you effectively read and use this guide, here is a brief description of the chapters and a suggested reading path. As a general rule, you can assume that Virtual Reality Toolbox on the Mac OS X platform works as described for the UNIX/Linux platforms.

This guide assumes that you are already familiar with

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

If you are a new Virtual Reality 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.
- Chapter 4, "MATLAB Interface" — Interact with a virtual world from MATLAB.

If you are an experienced Virtual Reality Toolbox user, you might want to review the following more advanced topics:

- Chapter 8, "Blocks — By Category" — Additional functionality has been added to the Virtual Reality Toolbox library.
- "vrworld Object Methods" on page 10-2 — Description of `vrworld` object properties and methods.
- "vrnode Object Methods" on page 10-3 — Description of `vrnode` object properties and methods.
- "vrfigure Object Methods" on page 10-3 — Description of `vrfigure` object properties and methods.

Features of Virtual Reality Toolbox

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.

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

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

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

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

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

Note Since some VRML worlds are automatically generated in VRML1.0, and Virtual Reality Toolbox does not support VRML1.0, you need to save these worlds in the current standard for VRML, VRML97.

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

MATLAB Interface

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

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

Virtual Reality Toolbox includes functions for retrieving and changing the virtual world properties and for saving the VRML files corresponding to the actual structure of a virtual world.

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

Simulink Interface

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

Virtual Reality Toolbox provides blocks to directly connect Simulink signals with virtual worlds. This connection lets you visualize your model as a three-dimensional animation.

You can implement most of the Virtual Reality Toolbox features with Simulink blocks. Once you include these blocks in a Simulink diagram, you can select a virtual world and connect Simulink signals to the virtual world. Virtual Reality Toolbox automatically scans a virtual world for available VRML nodes that Simulink can drive.

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

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

MATLAB Compiler Support

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

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

• No Simulink 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 Virtual Reality Toolbox (for example, creating, opening, and closing a `vrworld` object), then use the MATLAB Compiler.

VRML Viewers

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

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

If you install the VRML plug-in, Virtual Reality Toolbox connects MATLAB and Simulink with the VRML-enabled browser to display a simulated process using the TCP/IP protocol. This allows you to watch a simulated virtual world not only on the computer where MATLAB and Simulink are running, but also on other computers connected through the Internet

VRML Editor

For PC platforms, Virtual Reality Toolbox includes one of the classic VRML authoring tools, V-Realm Builder by Ligos Corp. With the addition of

this VRML authoring tool, Virtual Reality Toolbox provides a complete authoring, development, and working environment for carrying out 3-D visual simulations.

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

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

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

  For information on using V-Realm Builder with Virtual Reality Toolbox, see Chapter 5, "Virtual Worlds"

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

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

Real-Time Workshop Support

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

SimMechanics Support
You can use Virtual Reality Toolbox to view the behavior of a model created with SimMechanics. First, you build a model of a machine in Simulink using SimMechanics blocks. Then, create a detailed picture of your machine in a

virtual world, connect this world to the SimMechanics body sensor outputs, and view the behavior of the bodies in a VRML viewer.

Hardware Support

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

Client-Server Architecture

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

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

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

VRML Overview

In this section...
"VRML History" on page 1-10
"VRML Coordinate System" on page 1-11
"VRML File Format" on page 1-12

VRML History

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

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

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

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

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

browsers available on several platforms. Also, there are an increasing number of VRML authoring tools from which to choose. In addition, many traditional graphical software packages (CAD, visual art, and so on) offer VRML97 import/export features.

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

The VRML97 standard continues to be improved by the Web 3D Consortium. The newly released X3D (e**X**tensible **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

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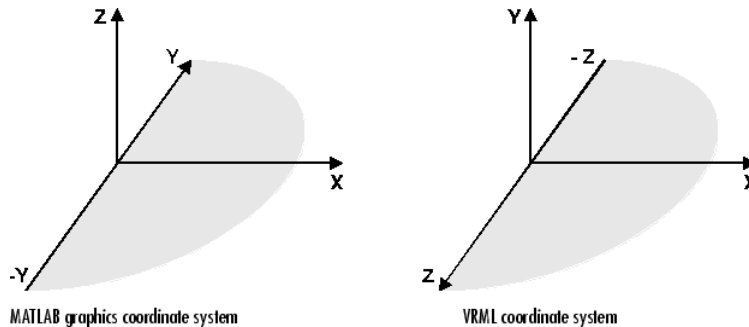
The VRML coordinate system is different from the MATLAB and Aerospace Blockset coordinate systems. VRML uses the *world coordinate system* in which the *y*-axis points upward and the *z*-axis places objects nearer or farther from the front of the screen. It is important to realize this fact in situations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

Rotation angles — In VRML, rotation angles are defined using the *right-hand rule*. Imagine your right hand holding an axis while your thumb points in the direction of the axis toward its positive end. Your four remaining fingers point in a counterclockwise direction. This counterclockwise direction is the positive rotation angle of an object moving around that axis.
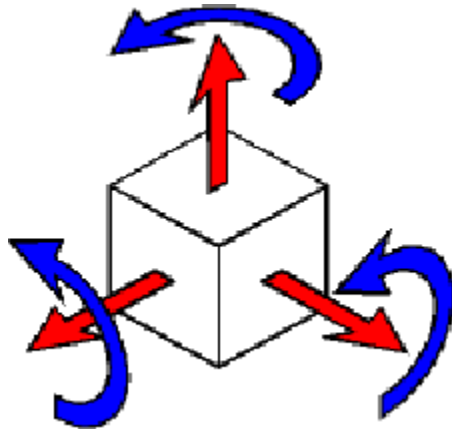
Child objects — In the hierarchical structure of a VRML file, the position and orientation of child objects are specified relative to the parent object. The parent object has its local coordinate space defined by its own position and orientation. Moving the parent object also moves the child objects relative to the parent object.

Measurement units — All lengths and distances are measured in *meters*, and all angles are measured in *radians*.

VRML File Format

You need not have any substantial knowledge of the VRML format to use the VRML authoring tools to create virtual worlds. However, it is useful to have a basic knowledge of VRML scene description. This helps you to create virtual worlds more effectively, and gives you a good understanding of how the virtual world elements can be controlled using Virtual Reality Toolbox.

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

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

the scene. There are 54 different types of nodes. Some of them are *shape nodes*(representing real 3-D objects), and some of them are *grouping nodes* used for holding child nodes. Here are some examples:

- Box node — Represents a box in a scene.

- Transform node — Defines position, scale, scale orientation, rotation, translation, and children of its subtree (grouping node).

- Material node — Corresponds to material in a scene.

- DirectionalLight node — Represents lighting in a scene.

- Fog node — Allows you to modify the environment optical properties.

- ProximitySensor node — Brings interactivity to VRML97. This node generates events when the user enters, exits, and moves within the defined region in space.

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

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

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

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

```
#VRML V2.0 utf8
# This is a comment line
WorldInfo {
  title "Bouncing Ball"
}
Viewpoint {
```

```
      position 0 5 30
      description  "Side View"
    }
    DEF Floor Box {
      size 6 0.2 6
    }
    DEF Ball Transform {
      translation   0 10 0
      children Shape {
        appearance Appearance {
          material Material {
            diffuseColor 1 0 0
          }
        }
        geometry Sphere {
        }
      }
    }
```

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

Most of the box properties are left at their default values – distance from the center of the coordinate system, material, color, and so on. Only the name Floor and the dimensions are assigned to the box. To be able to control the position and other properties of the ball, it is defined as a child node of a Transform type node. Here, the default unit sphere is assigned a red color and a position 10 m above the floor. In addition, the virtual world title is used by VRML viewers to distinguish between virtual worlds. A suitable initial viewpoint is defined in the virtual world VRML file.

When displayed in V-Realm Builder, the floor and red ball look like this:
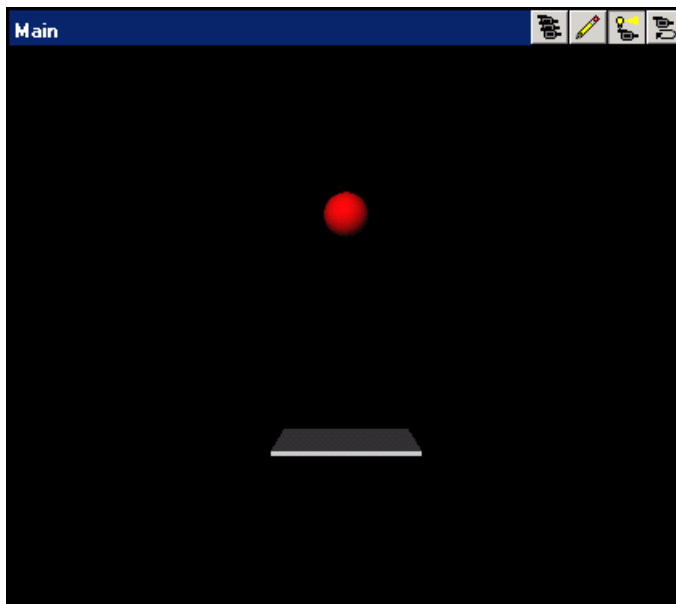
Examples Using Virtual Reality Toolbox

In this section...
"Simulink Interface Examples" on page 1-16
"MATLAB Interface Examples" on page 1-22

Simulink Interface Examples

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

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

    vrbounce

A Simulink window opens with the block diagram for the model. By default, a virtual world opens in the Virtual Reality Toolbox viewer or your VRML-enabled Web browser. If the viewer does not appear, double-click the VR Sink block in the Simulink model.

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

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

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

3 In the Simulink window, from the **Simulation** menu, click **Start**. (Alternatively, in the Virtual Reality Toolbox viewer, from the **Simulation** menu, click **Start**.)

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

The following table lists the Simulink examples provided with Virtual Reality Toolbox. Descriptions of the examples follow the table.

Example	RTW Ready	VR Sink	Joystick	Space Mouse
vrbounce	X	X		
vrcrane_joystick		X	X	
vrcrane_traj		X		
vrlights		X		
vrmaglev	X	X		
vrmaglev_rtwin	X	X		
vrmanipul		X		X
vrmemb1		X		
vr_octavia	X	X		
vrpend	X	X		
vrplanets	X	X		
vrtkoff		X		

Bouncing Ball Example (vrbounce)

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

Portal Crane with Joystick Control (vrcrane_joystick)

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

To minimize the number of signals transferred between the Simulink model and the virtual reality world, and to keep the model as simple and flexible as

possible, only the minimum set of moving objects properties are sent from the model to the VR Sink block. All other values that are necessary to describe the virtual reality objects movement are computed from this minimum set using VRMLScript in the associated VRML file.

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

Portal Crane with Predefined Trajectory Example (vrcrane_traj)

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

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

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

Lighting Example (vrlights)

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

Magnetic Levitation Model Example (vrmaglev)

The `vrmaglev` example shows the interaction between dynamic models in Simulink and virtual worlds. The Simulink model represents the

HUMUSOFT CE 152 Magnetic Levitation educational/presentation scale model. The plant model is controlled by a PID controller with feed-forward to cope with the nonlinearity of the magnetic levitation system.

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

Magnetic Levitation Model for Real-Time Windows Target Example (vrmaglev_rtwin)

In addition to the `vrmaglev` example, the `vrmaglev_rtwin` example works directly with the actual CE 152 scale model hardware in real time. The MathWorks created this model to work with Real-Time Workshop, Real-Time Windows Target, and the HUMUSOFT MF 614 data acquisition board. However, you can adapt this model for other targets and acquisition boards. A digital IIR filter, from the Signal Processing Blockset, filters the physical system output. You can bypass the physical system by using the built-in plant model. Running this model in real time is an example showing the capabilities of Simulink in control systems design and rapid prototyping.

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

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

Manipulator with Space Mouse Example (vrmanipul)

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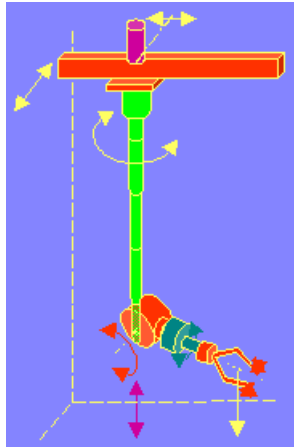
The VRML model represents a nuclear hot chamber manipulator. It is manipulated by a simple Simulink model containing the Space Mouse Input block. This model uses all six degrees of freedom of the space mouse for manipulating the mechanical arm, and this model uses mouse button 1 to close the grip of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

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.

Vehicle Dynamics Visualization (vr_octavia)

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

Inverted Pendulum Example (vrpend)

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

- Mouse — Click and drag a mouse pointer in the **Trajectory Graph** two-dimensional window

- Input Signal — External Trajectory Graph input in this model (driven by a random number generator)

- VR Sensor — Activates the input from a VRML TouchSensor

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

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

as the Examine mode. If you want to navigate around the virtual pendulum bench, you should use some other viewpoint.

Solar System Example (vrplanets)

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

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

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

Plane Takeoff Example (vrtkoff)

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

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

MATLAB Interface Examples

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

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

Car in the Mountains Example (vrcar)

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

1 In the MATLAB Command Window, type

    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 Virtual Reality Toolbox with the MATLAB interface for manipulating complex objects.

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

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

• Reshaping the object

- Applying the color palette to represent distributed parameters across an object shape

- Working with VRML text objects

- Animating a scene using the MATLAB interface

- Synchronization of multiple scene properties

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

Heat Transfer Visualization with 2-D Animation (vrheat_anim)

This demonstration illustrates the use of the Virtual Reality Toolbox MATLAB interface to create 2-D offline animation files.

You can control the offline animation recording mechanism by setting the relevant vrworld and vrfigure object properties. Note that you should use the Virtual Reality Toolbox viewer to record animations. However, direct control of the recording is also possible.

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

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

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

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

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

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

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

Rotating Membrane with MATLAB GUI Example (vrmemb)

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

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

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

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 Mapping Toolbox) 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 Mapping Toolbox.

Virtual Reality Toolbox Texture File

The following are texture file recommendations for Virtual Reality Toolbox:

- 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 1-28
"Virtual Reality Toolbox Server" on page 1-29

VRML Compatibility

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

- The Virtual Reality Toolbox server ignores the VRML Script node, but it passes the node to the VRML viewer. This allows you to run VRML scripts on the viewer side. You cannot run them on the Virtual Reality Toolbox server.

- The Virtual Reality Toolbox server ignores the Inline node, but it passes the node to the viewer. Therefore, the viewer sees the complete virtual world with all included substructures, but the included parts are not accessible from the toolbox. In some rare cases, this limitation can render the virtual world unusable with Virtual Reality Toolbox. This happens under either of the following conditions:

  - The virtual world contains a USE name reference to a node that is in the included part.

  - The virtual world contains an included part with a PROTO or EXTERNPROTO declaration that is referenced in the main virtual world file.

- In keeping with the VRML97 specification, the Virtual Reality Toolbox viewer ignores BMP files. As a result, VRML scene textures might not display properly in the Virtual Reality Toolbox viewer. To properly display scene textures, replace all BMP texture files in a VRML scene with PNG, JPG, or GIF equivalents. Note that blaxxun Contact supports BMP files in addition to the standard VRML texture file formats.

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

Virtual Reality Toolbox Server

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

Virtual Reality Toolbox uses a Virtual Reality Toolbox HTTP server for communication between a VRML-enabled Web browser and the MATLAB/Simulink environment. It generates the main Virtual Reality Toolbox HTML page with the list of currently available virtual worlds and sends VRML and other requested files and data to clients (VRML viewers).

The server is started when Virtual Reality Toolbox is loaded into MATLAB. This happens whenever you use a Virtual Reality Toolbox block in a Simulink block diagram, or whenever you open a vrworld object in the MATLAB interface. The HTTP server is shut down when you close all Simulink models that contain Virtual Reality Toolbox blocks, or use the vrclear command.

When the HTTP server is running, your browser can see a list of available virtual worlds at the following address, where 8123 is the default port number:

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

Remote users can connect to the following address, where 8123 is the default port number:

```
http://your_machine:8123
```

You can set the port number of the server in the Virtual Reality Toolbox Preferences dialog box from the Simulink interface, or use vrsetpref in the MATLAB Command Window.

Depending on the status of served vrworld objects, the list of available virtual worlds can be empty.

2

Installation

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

Required Products

In this section...
"Section Overview" on page 2-3
"MATLAB" on page 2-3
"VRML Viewer" on page 2-3

Section Overview

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

MATLAB

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

Note Version 4.2 of Virtual Reality Toolbox requires MATLAB Version 7.1.

MATLAB documentation — For information on using MATLAB, see the
MATLAB documentation. It explains how to work with data and how to
use the functions supplied with MATLAB. For a reference describing the
functions specific to Virtual Reality Toolbox, see Chapter 10, "Functions —
By Category".

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 Virtual
  Reality Toolbox and is the default viewer for virtual worlds. You can access
  this viewer from either a Virtual Reality Toolbox block in your Simulink
  model, or by using the `vrview` and `vrfigure` functions with MATLAB.

The Virtual Reality Toolbox viewer is a client to the Virtual Reality Toolbox server. It does not require a Web browser and it is available on more platforms than any other VRML97 viewer. It is supported on PC, Mac OS X, UNIX, and Linux platforms. The viewer is the recommended method for viewing virtual worlds on a host computer.

- blaxxun Contact Version 4.4 — VRML plug-in shipped with the PC version of Virtual Reality Toolbox. This VRML plug-in allows you to view virtual worlds in your Web browser. The blaxxun Contact plug-in is the only supported VRML plug-in.

You can view a virtual world in the Virtual Reality Toolbox viewer as soon as you install Virtual Reality Toolbox. If you want to view the virtual world in your Web browser, you need to use the `vrinstall` command to install the blaxxun Contact plug-in. See "Installing a VRML Plug-In (Windows)" on page 2-20.

For information on using a Web browser to view virtual worlds, see "Testing the Installation" on page 2-51. The blaxxun Contact installation executable files are located at `C:\`*matlabroot*`\toolbox\vr\blaxxun`.

Every VRML plug-in installs Java classes into the Web browser. Limit the number of plug-ins you use to avoid Java errors and conflicts. For this reason, use only the Virtual Reality Toolbox viewer or the blaxxun Contact VRML plug-in on PC platforms. On UNIX and Linux platforms, use only the Virtual Reality Toolbox viewer.

Recommended Product

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

Simulink

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

With Virtual Reality Toolbox, 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 4.2 of Virtual Reality Toolbox uses Simulink Version 6.2.

Simulink documentation — For information on using Simulink, see the Simulink documentation. It explains how to connect blocks, build models, and change block parameters. For a reference describing the Virtual Reality Toolbox blocks, see Chapter 8, "Blocks — By Category".

Related Products

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

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 2-7
"Supported Computer Platforms" on page 2-7
"Host Computer" on page 2-8
"Client Computer" on page 2-10

Section Overview

Virtual Reality Toolbox has the same hardware requirements as MATLAB. It is a multiplatform product that runs on PC-compatible computers with Windows or Linux. It runs on 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 Virtual Reality Toolbox that interfaces with your Simulink models. It stores information about the current state of virtual worlds and manages connections to VR clients. The VR client is a VRML viewer that displays a virtual world. The VR client can be either the Virtual Reality Toolbox viewer or a Web browser with a VRML plug-in.

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

Platform/Product	VR Server	Virtual Reality Toolbox Viewer	VRML Editor	VRML Browser Plug-In
Microsoft Windows XP or Windows 2000	Yes	Yes	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 Virtual Reality Toolbox.

Host Computer

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

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

Host Computer Hardware Requirements

Hardware	Description
CPU	Pentium, Athlon or higher (PC)
Graphics card	Graphics card with hardware 3-D acceleration
RAM	128 Mbytes or more
Peripherals	Hard disk drive with 45 Mbytes of free space 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 Virtual Reality Toolbox requires on your host computer. For a list of optional software products related to Virtual Reality Toolbox, see http://www.mathworks.com/products/virtualreality/related.jsp.

Host Computer Software Requirements

Software	Description
MATLAB	Version 7.0.4.

Host Computer Software Requirements (Continued)

Software	Description
Simulink	Version 6.2. Simulink is not required, but we highly recommend that you install it.
Virtual Reality Toolbox	Version 4.2.
VRML editor	For Windows platforms, you can install the VRML editor (V-Realm Builder 2.0) provided with Virtual Reality Toolbox. For UNIX/Linux, the default editor is the MATLAB editor. When you create VRML worlds on these operating systems, you can use any 3-D modeling tool with the VRML97 export capability.
Web browser	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.
	Use Microsoft Internet Explorer 4.0 or later, or Netscape Navigator 4.0 or later with Java enabled.
VRML plug-in	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.
	Windows platforms — You can install the blaxxun Contact 4.4 plug-in provided with Virtual Reality Toolbox.
	For information on how to install the blaxxun Contact plug-in, see "Installing a VRML Plug-In (Windows)" on page 2-20.

Client Computer

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

The client computer communicates with the host computer over TCP/IP, and it displays the virtual world using a VR client. In this case, the VR client is a VRML-enabled Web browser. You can verify the TCP/IP connection between the host and client computers by using the ping command from a command-line prompt. If there are problems, you must first fix the TCP/IP protocol settings according to the documentation for your operating system.

The following table lists the minimum hardware resources Virtual Reality Toolbox 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 Virtual Reality Toolbox requires on the client computer. You do not need to install Virtual Reality Toolbox on the client computer.

Because the only component required for the client computer is standard VRML97 viewing software, it is possible that different configurations will work. For example, you might be able to run an operating system not listed in the table "Supported Computer Platforms" on page 2-7. However, these configurations have not been tested and they are not supported.

Client Computer Software Requirements

Software	Description
Operating system	Microsoft Windows XP or Windows 2000 (the TCP/IP protocol must be installed).
Web browser	Use Microsoft Internet Explorer 4.0 or later, or Netscape Navigator 4.0 or later with Java enabled.
VRML plug-in	VRML97 plug-in with External Authoring Interface support. If you have blaxxun Contact (Windows) on your computer, you have already installed a VRML plug-in.
	Windows platforms -- You can install the blaxxun Contact 4.4 plug-in provided with Virtual Reality Toolbox.
	For information on how to install the blaxxun Contact plug-in, see "Installing a VRML Plug-In (Windows)" on page 2-20.

Installing Virtual Reality Toolbox on the Host Computer

Section Overview

You may install Virtual Reality Toolbox from a DVD or from the MathWorks Web site. For Web downloads, you need your online MathWorks Account. Before you install Virtual Reality Toolbox, you need to get a valid license file and/or personal license password. For detailed information about the installation process, see the installation documentation for your platform.

Components on a Host Computer

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

- Virtual Reality Toolbox — This component contains the core files that interconnect MATLAB and Simulink to VRML. This component is required for Virtual Reality Toolbox to operate, and you must install it on the host computer. This component is *not* used on a client computer.

- Virtual Reality Toolbox viewer — This is a multiplatform VRML viewer that is included with Virtual Reality Toolbox, and it is set as the default viewer for displaying virtual worlds.

- VRML plug-in — Optionally, you can use a VRML plug-in for a Web browser to view virtual reality worlds. The blaxxun Contact plug-in is included with Virtual Reality Toolbox for Windows platforms. However, you can also use the Virtual Reality Toolbox viewer. A VRML plug-in is the only component that you need to install on a client computer.

- VRML editor — If you are going to create and modify virtual worlds, you need a VRML97-compatible editor. V-Realm Builder is included with Virtual Reality Toolbox for Windows platforms. If you do not plan to edit virtual reality worlds or if you prefer to use a different VRML editor, you do not need to install it on your computer. For UNIX/Linux platforms, the MATLAB editor is the default VRML editor. This component is *not* used on a client computer.

- Example models — These are MATLAB and Simulink programs and models connected to prebuilt virtual reality worlds. You can use these models and virtual reality worlds both for discovering the capabilities of Virtual Reality Toolbox and as templates for building your own projects. This component is not used on the client computer.

- Online documentation — This component contains the guide you are reading now. You can access the online version through the MATLAB Help browser. An Adobe Acrobat PDF file is available on the MathWorks Web site at `http://www.mathworks.com`. Follow the links to product documentation. This documentation can be read using the Adobe Acrobat Reader. If you do not have this reader installed on your computer, you can download it from `http://www.adobe.com`.

Installing from a DVD (Windows)

To install Virtual Reality Toolbox 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.

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

2 Select the **Virtual Reality Toolbox** check box, then click **Next**.
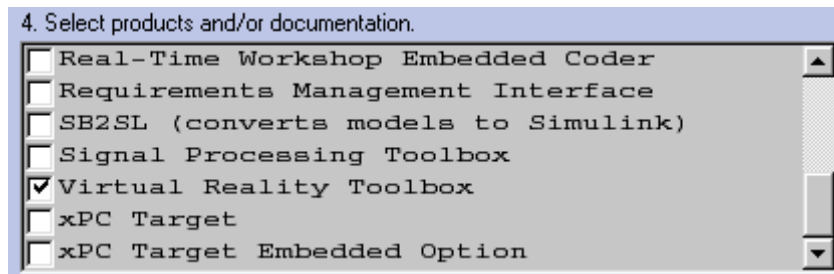
3 Follow the instructions on each of the remaining screens to complete the installation.

The Virtual Reality Toolbox viewer is installed with Virtual Reality Toolbox. For PC platforms, you have the option of installing a VRML plug-in for your browser as an alternative to the viewer. See "Installing a VRML Plug-In (Windows)" on page 2-20.

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

Installing from a DVD (UNIX/Linux)

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

1 Log in to your system.

2 Mount the DVD-ROM drive.

3 Create a directory to be the mount point for the DVD-ROM drive. For example:

```
mkdir
/cdrom
```

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

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

Subsequent instructions in this guide refer to this directory as *matlabroot*.

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

---

5 Move your license file, named `license.dat`, into the *matlabroot* directory.

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

6 Run the appropriate installation script for your platform.

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

7 During the installation process, a dialog box allows you to select the products to install.
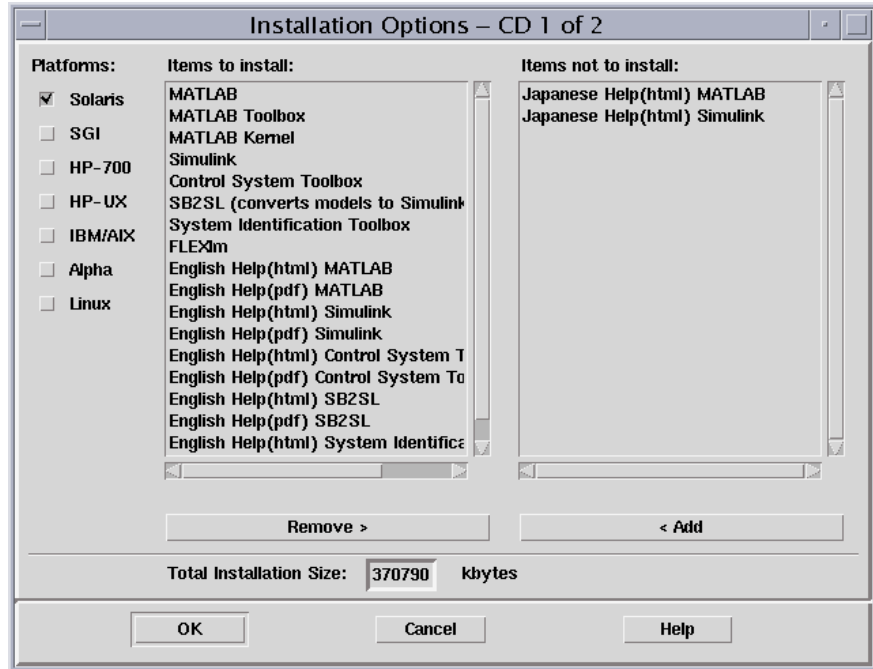
This dialog box lists all the products you are licensed to install in the **Items to Install** box. Make sure Virtual Reality Toolbox is listed in this box.

8 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 2-19.

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

LD_LIBRARY_PATH Environment Variable (UNIX)

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

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

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

If the `LD_LIBRARY_PATH` environment variable already exists, use a line like the following to add the new path to the existing one:

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

If the `LD_LIBRARY_PATH` environment variable does not already exist, use a line like the following:

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

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

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

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

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

Editing the Java Library Path

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

1 Run the `regedit` command.

2 Go to

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\JavaVM
```

A list of value names and their values appears.

3 Replace each instance of `%systemroot%` with the system root path. For example:

```
C:\WINNT
```

4 Restart the computer.

Installing the VRML Plug-In Viewer on the Host Computer

In this section...
"Section Overview" on page 2-19
"Virtual Reality Toolbox Viewer" on page 2-19
"Installing a VRML Plug-In (Windows)" on page 2-20
"Installing a VRML Plug-In (UNIX/Linux)" on page 2-23
"Setting the Default Viewer of Virtual Scenes" on page 2-24

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 Virtual Reality Toolbox. You can view virtual scenes as soon as Virtual Reality Toolbox is installed on your machine.

Note It is possible to view virtual scenes with a Web browser that contains a VRML plug-in. Every VRML plug-in installs Java classes into the Web browser. It is best to limit the number of plug-ins you install on your machine to avoid Java errors and conflicts. For this reason, use only the Virtual Reality Toolbox viewer and the blaxxun Contact VRML plug-in on PC platforms. On UNIX and Linux platforms, use only the Virtual Reality Toolbox viewer.

Installing a VRML Plug-In (Windows)

When you install Virtual Reality Toolbox, the Virtual Reality Toolbox viewer is set as the default viewer. If you want to use a Web browser as a VRML viewer, use the following procedure to install the blaxxun Contact plug-in. You can use this plug-in with either Microsoft Internet Explorer or Netscape Navigator. The blaxxun Contact plug-in is the only supported VRML plug-in.

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

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

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

1 Start MATLAB.

2 In the MATLAB Command Window, type

```
vrinstall -install viewer
```

MATLAB displays the message

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

3 Check the 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, MATLAB displays the following message:

```
VRML viewer:    installed
```

If the viewer installation was unsuccessful, MATLAB displays the message

```
VRML
viewer:    not installed
```

2-21

Known Issue with the blaxxun Contact Plug-In

The blaxxun Contact VRML plug-in can fail to update the virtual scene when used with Virtual Reality Toolbox and Microsoft Internet Explorer 5.5 and later. Netscape 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 blaxxun Contact does not resolve this problem.

Changing the Default Network Security Setting

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

- PC platform is Microsoft Windows 2000 or 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/Linux)

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

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

2-23

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 2-36.) The following procedure assumes that you are working on a PC platform:

1 At the MATLAB command prompt, type

```
vrinstall -check
```

to determine whether blaxxun Contact is installed.

MATLAB displays

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

The viewer and editor are installed. If the viewer is not installed, see "Installing a VRML Plug-In (Windows)" on page 2-20.

2 Determine your default viewer by typing

```
vrgetpref
```

MATLAB displays

```
ans =

                            DataTypeBool: 'logical'
                           DataTypeInt32: 'double'
                           DataTypeFloat: 'double'
                 DefaultFigureAntialiasing: 'off'
            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.
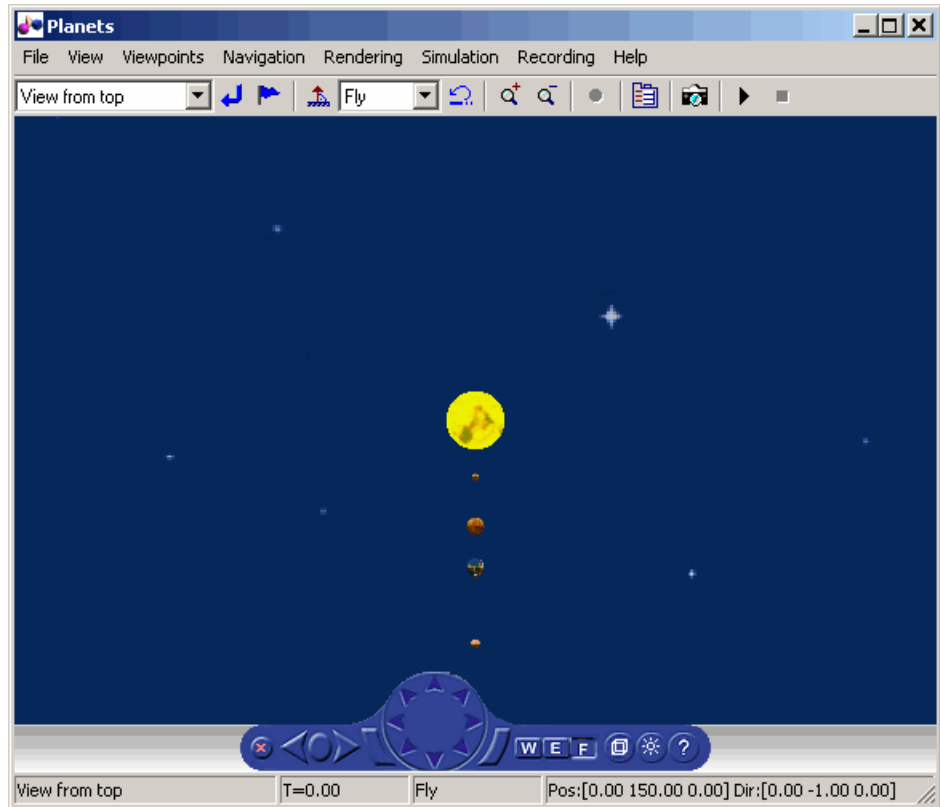
4 Change the default viewer to your Web browser by typing

```
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.
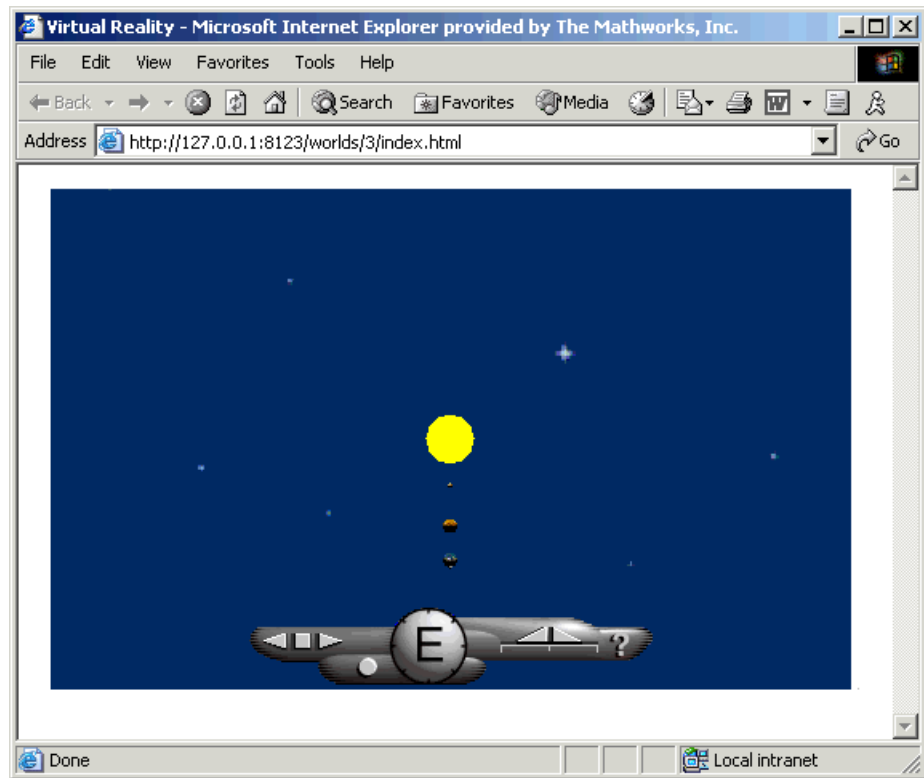
6 Reset the Virtual Reality Toolbox viewer as your default viewer by typing

```
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.
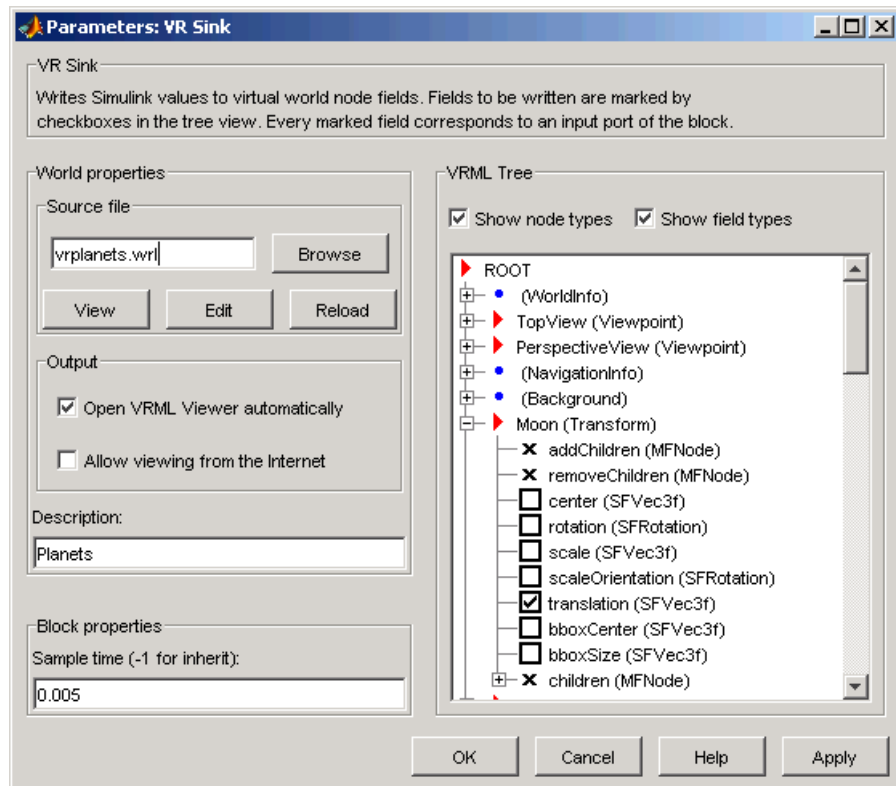
The target of the **View** button is determined by the DefaultViewer property. If the DefaultViewer property is set to 'internal', clicking the **View** button opens the virtual world in the Virtual Reality Toolbox viewer. If the DefaultViewer property is set to 'web', clicking the **View** button opens 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 2-29
"VRML Editor (UNIX/Linux)" on page 2-30
"Setting the Default Editor of Virtual Scenes" on page 2-30

Installing the VRML Editor (Windows)

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

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

1 Start MATLAB.

2 In the MATLAB Command Window, type

```
vrinstall
-install editor
```

or type

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

MATLAB displays the following messages:

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

3 Type

```
vrinstall
-check
```

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

```
VRML editor:    installed
```

VRML Editor (UNIX/Linux)

The MATLAB editor is the default VRML editor for UNIX platforms and no installation is required. To create your virtual worlds using the MATLAB editor, you need to understand the virtual reality modeling language and the VRML data types that are relevant to MATLAB. For information about the modeling language, refer to an appropriate third-party VRML book. Also, see "VRML Data Types" on page 5-21 for the data types to use with MATLAB.

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

Setting the Default Editor of Virtual Scenes

You can edit virtual scenes with a VRML authoring tool, such as V-Realm Builder, or with any text editor, as the VRML language is written in text files. You determine the editor that is used to edit your scene by using the vrsetpref and vrgetpref commands. (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 2-36.)

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

1 At the MATLAB command prompt, type

```
vrinstall
-check
```

to determine whether V-Realm Builder is installed.

MATLAB displays

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

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

2 Determine your default editor by typing

```
a = vrgetpref
```

MATLAB displays

```
a =

                            DataTypeBool: 'logical'
                           DataTypeInt32: 'double'
                           DataTypeFloat: 'double'
                 DefaultFigureAntialiasing: 'off'
                     DefaultFigureDeleteFcn: ''
                      DefaultFigureLighting: 'on'
               DefaultFigureMaxTextureSize: 'auto'
                     DefaultFigureNavPanel: 'halfbar'
                     DefaultFigureNavZones: 'off'
                    DefaultFigurePosition: [5 92 512 380]
      DefaultFigureRecord2DCompressMethod: 'auto'
     DefaultFigureRecord2DCompressQuality: 75
              DefaultFigureRecord2DFileName: '%f_anim_%n.avi'
                     DefaultFigureStatusBar: 'on'
                       DefaultFigureToolBar: 'on'
                 DefaultFigureTransparency: 'on'
                    DefaultFigureWireframe: 'off'
                            DefaultViewer: 'internal'
              DefaultWorldRecord3DFileName: '%f_anim_%n.wrl'
                   DefaultWorldRecordMode: 'manual'
                DefaultWorldRecordInterval: [0 0]
                   DefaultWorldRemoteView: 'off'
                   DefaultWorldTimeSource: 'external'
                                    Editor: [1x60 char]
                                  HttpPort: 8123
                           TransportBuffer: 5
                                    VrPort: 8123
```

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

3 To determine your default editor, type

```
a.Editor
```

MATLAB displays

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

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

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

```
vrpend
```

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

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

The Parameters: VR Sink dialog box opens.
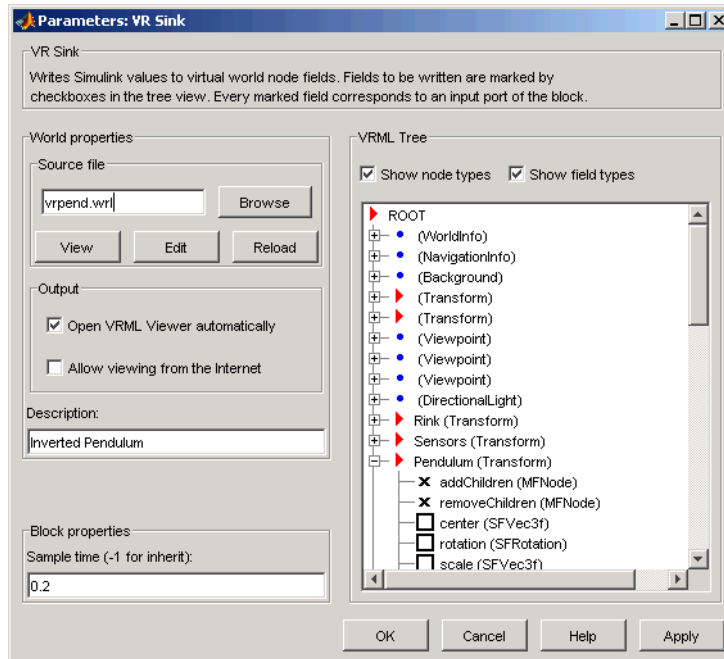
6 Click **Edit**.

The vrpend model opens in the V-Realm Builder authoring tool.
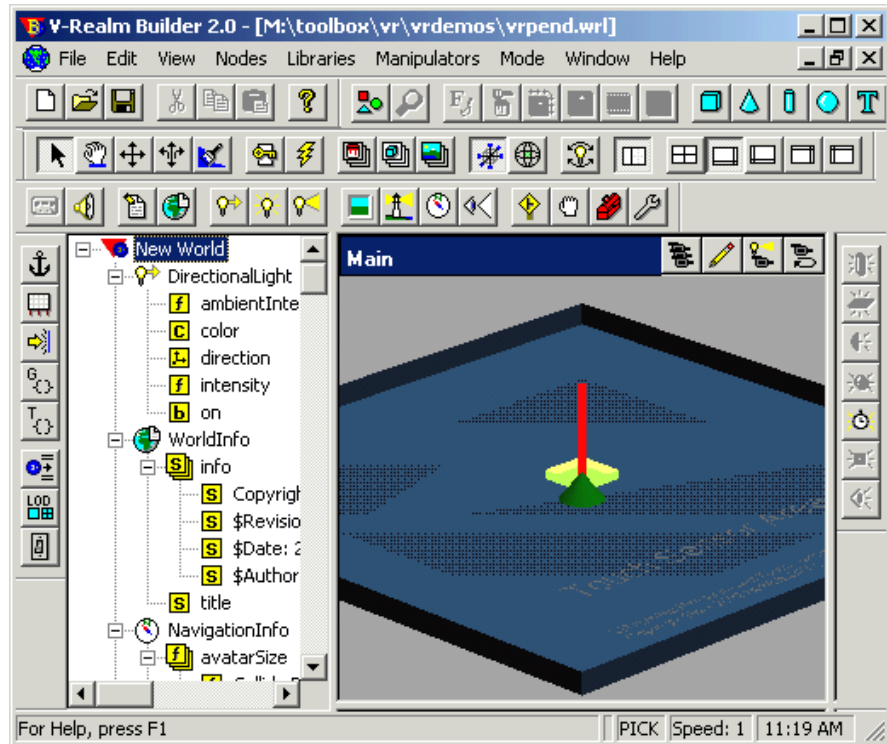
7 At the MATLAB window, change the default editor to the MATLAB editor by typing

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

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

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

The Parameters: VR Sink dialog box opens.

9 Click **Edit**.

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

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

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

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

Changing Virtual Reality Toolbox Preferences with the MATLAB Preferences Dialog

In this section...
"Section Overview" on page 2-36
"Virtual Reality Toolbox Preferences" on page 2-36
"Virtual Reality Toolbox Figure Preferences" on page 2-39
"Virtual Reality Toolbox World Preferences" on page 2-45

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.

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

- MATLAB **File > Preferences** dialog — This GUI has preference dialogs for MATLAB and its related products, including Virtual Reality Toolbox.
- 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.
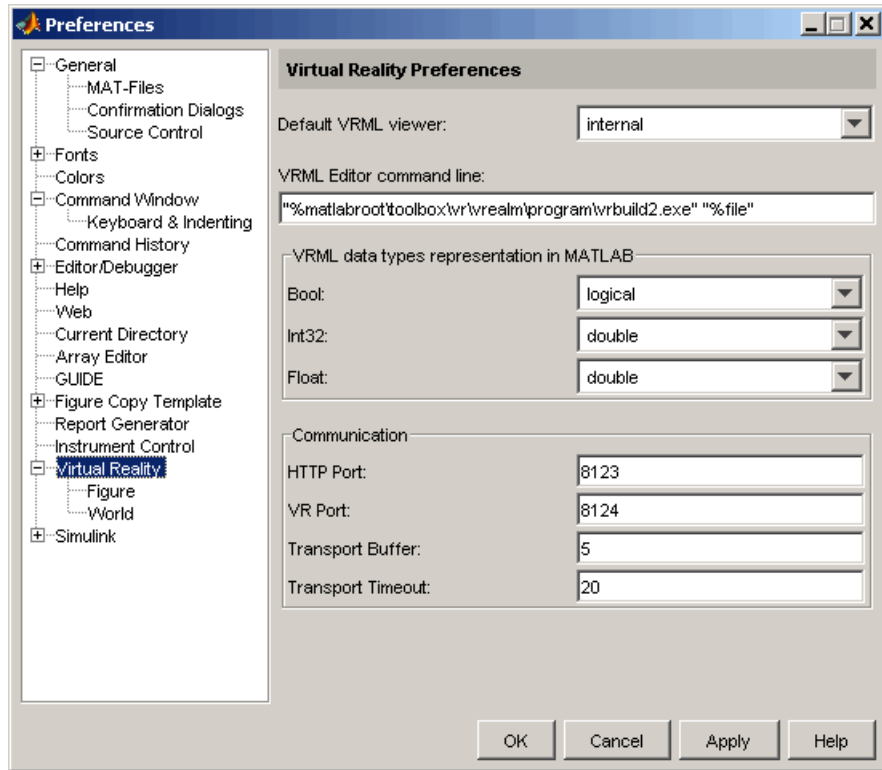
3 Set the preferences as desired. See the following table for the preferences you can change. Click **OK** to save the settings.

Preference	Value	Description
Bool	'logical' | 'char' Default: 'logical'	Specifies the handling of the VRML Bool data type for vrnode/setfield and vrnode/getfield. 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'.

Preference	Value	Description
`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 `vrnode/setfield` and `vrnode/getfield`. 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 `vrnode/setfield` and `vrnode/getfield`. 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 MATLAB 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.
`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 MATLAB 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 2-39
- "Virtual Reality Toolbox Figure Rendering Preferences" on page 2-40
- "Virtual Reality Toolbox Figure 2-D Recording Preferences" on page 2-42
- "Virtual Reality Toolbox Figure Frame Capture Preferences" on page 2-44

Virtual Reality Toolbox Figure Appearance Preferences

To access the virtual figure appearance 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, with the **Appearance** tab selected.
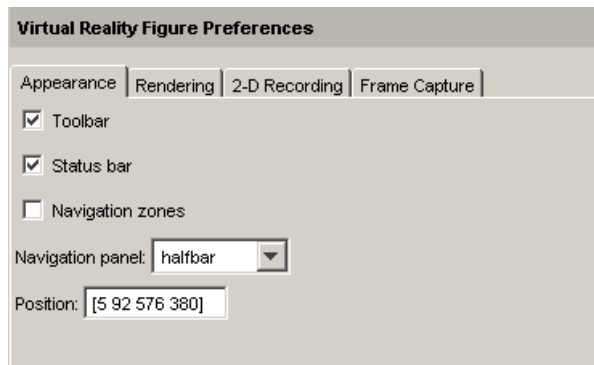
4 Set the preferences as desired. See the following table for the appearance preferences you can change. Click **OK** to save the settings.

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.

Property	Value	Description
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 `vrfigure` objects. The smaller the size, the faster the texture can render. Increasing this value improves image quality but decreases performance. A value of `'auto'` sets the maximum possible pixel size. If the value you enter is unsuitable, a warning might trigger. Virtual Reality Toolbox then automatically adjusts the property to the next smaller suitable value.
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
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 `avifile`. Read/write.
Recording compression quality	0-100 Default: `'75'`	Specifies the quality of 2-D animation file compression.

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 4-10.
`Frames per second`	0-100 Default: `'15'`	Specifies the frames per second during animation playback.

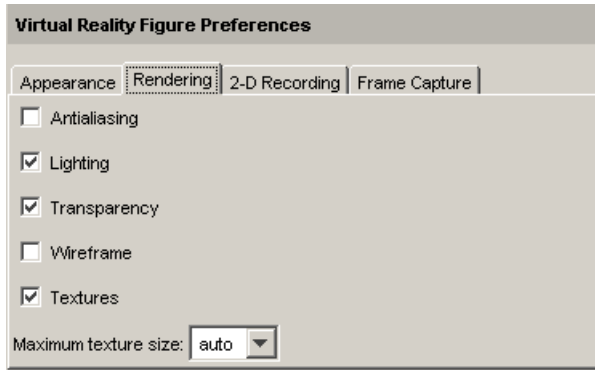### Virtual Reality Toolbox Figure Frame Capture Preferences

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.

Virtual Reality Figure Preferences

Appearance | Rendering | 2-D Recording | Frame Capture

Frame capture file name:

%f_anim_%n.tif

Frame capture file format: tif ▼

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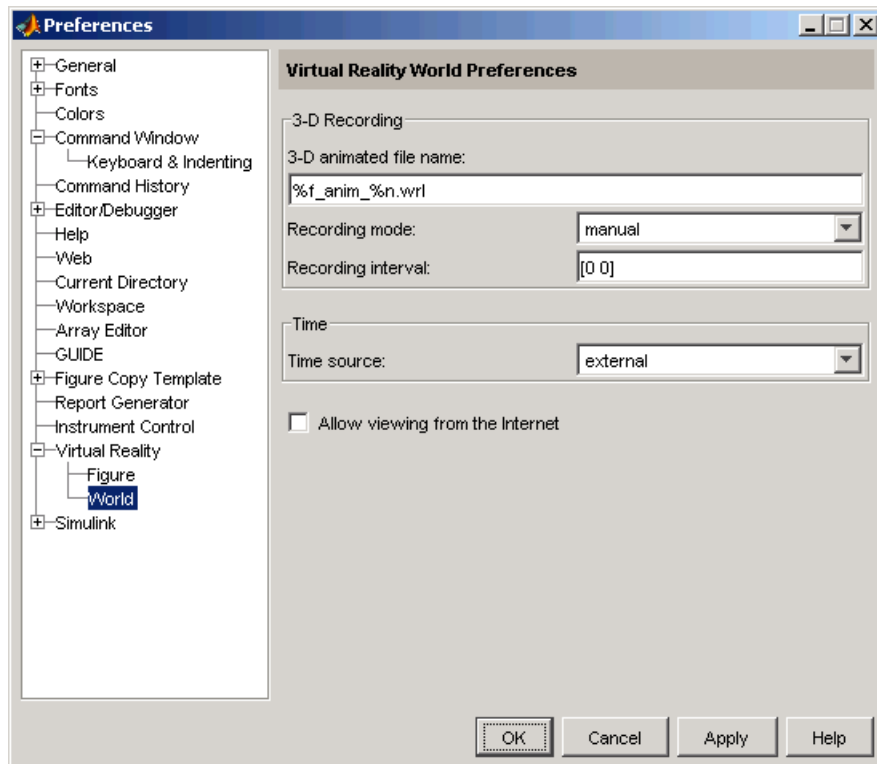
4 Set the preferences as desired. See the following table for the rendering preferences you can change. Click **OK** to save the settings.

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'.

Property	Value	Description
`3-D animated file name`	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 4-10.
`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 MATLAB (by setting the `Time` property) or Simulink (simulation time). If set to `'freerun'`, time in the scene advances independently based on the system timer.

Removing Components (Windows)

Section Overview

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

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

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

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

   The MathWorks uninstaller begins running.

2 Clear the **Virtual Reality Toolbox** check box.

3 Follow the remaining uninstall instructions.

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

Removing the blaxxun Contact Plug-In (Windows)

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

1 From the Microsoft Windows task bar, click **Start**, point to **Settings**, and click **Control Panel**.

2 In the **Control Panel** cascading menu, click **Add/Remove Programs**.

3 In the Add/Remove Programs dialog box, select `blaxxun Contact`, then click the **Change/Remove** button.

Installing on the Client Computer

In this section...
"Section Overview" on page 2-50
"Installing a VRML Plug-In (Windows)" on page 2-50

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 blaxxun Contact is supported.

"Installing a VRML Plug-In (Windows)" on page 2-50 describes how to install the blaxxun Contact VRML plug-in on a computer running Microsoft Windows.

Installing a VRML Plug-In (Windows)

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

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

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

Testing the Installation

In this section...
"Section Overview" on page 2-51
"Running a Simulink Interface Example" on page 2-51
"Running a MATLAB Interface Example" on page 2-56

Section Overview

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

Running a Simulink Interface Example

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

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

1 In the MATLAB Command Window, type

    vrpend

A Simulink window opens with the model for an inverted pendulum.
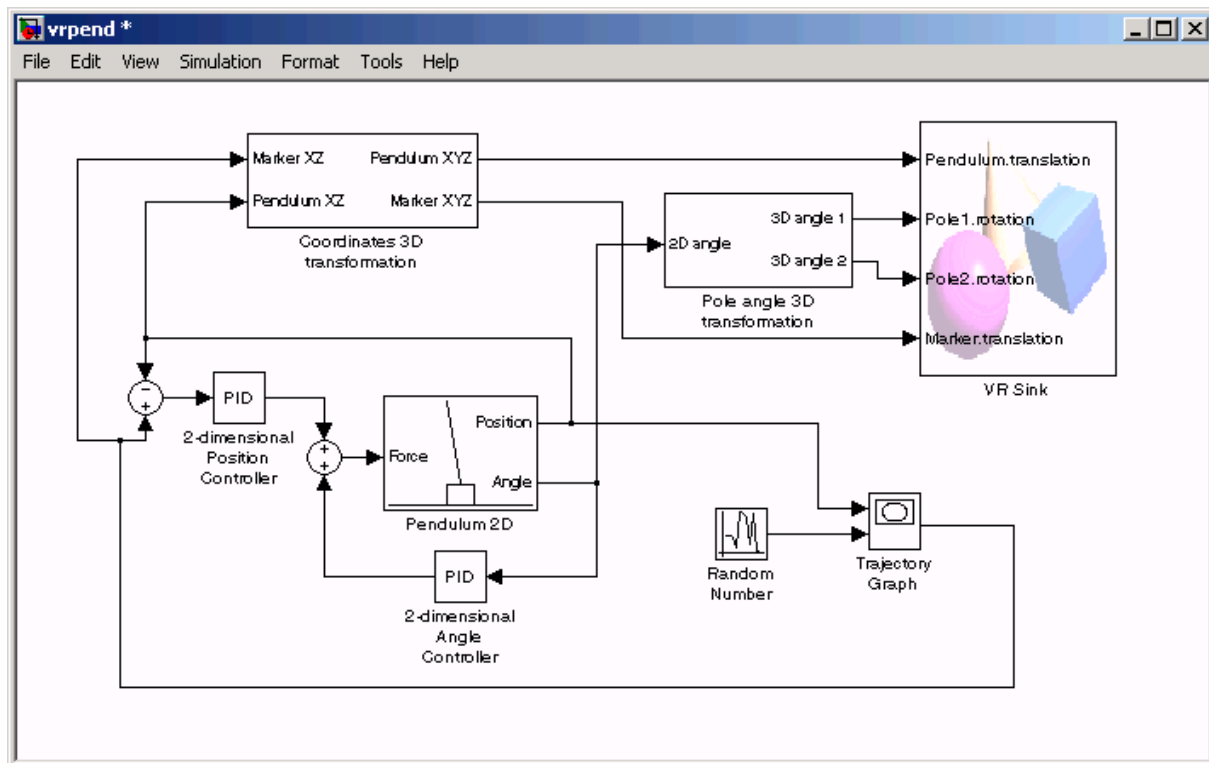
The Virtual Reality Toolbox viewer opens with a 3-D model of the pendulum.
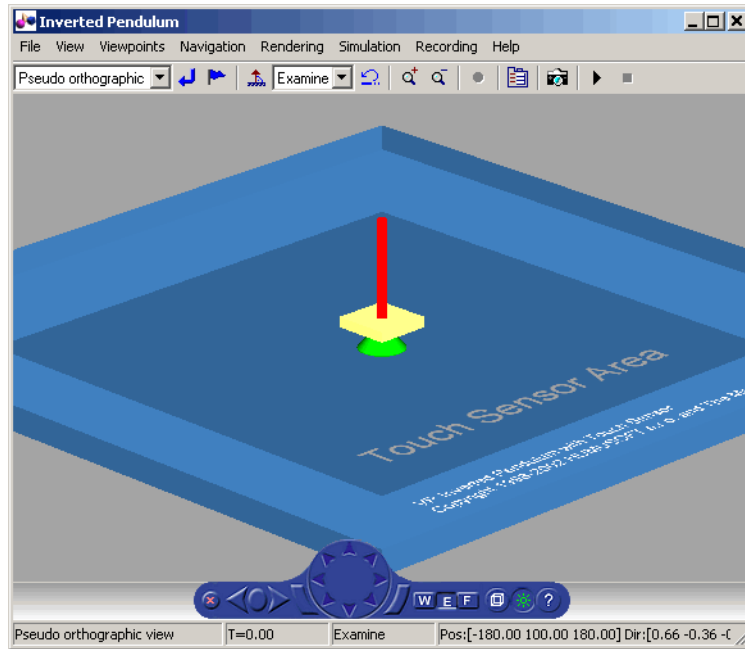
2 In the Virtual Reality Toolbox viewer, from the **Simulation** menu, click **Start**. A **Trajectory Graph** window opens, and a simulation starts running.
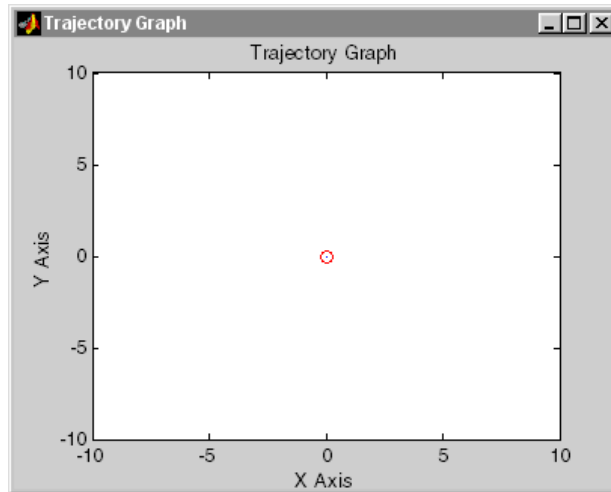
3 In the Virtual Reality Toolbox viewer, point to a position on the blue surface and left-click.

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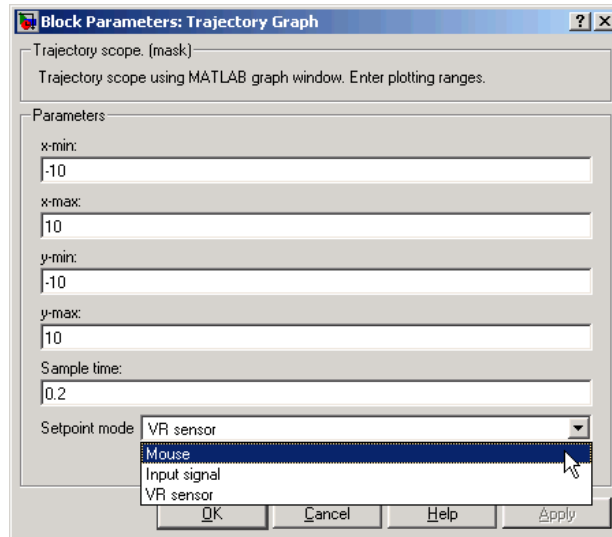
You can now use the trajectory graph as a 2-D input device to set the position of the pendulum.

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.
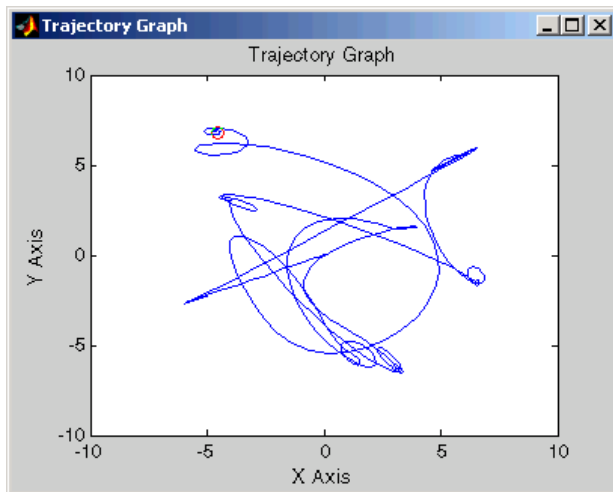
8 Close the Virtual Reality Toolbox viewer and close the Simulink window.

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

Running a MATLAB Interface Example

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

Additional examples are listed in the table "MATLAB Interface Examples" on page 1-22.

1 In the MATLAB window, type

```
vrmemb
```

MATLAB displays the following messages:

```
Loading... This example
shows you how to 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.
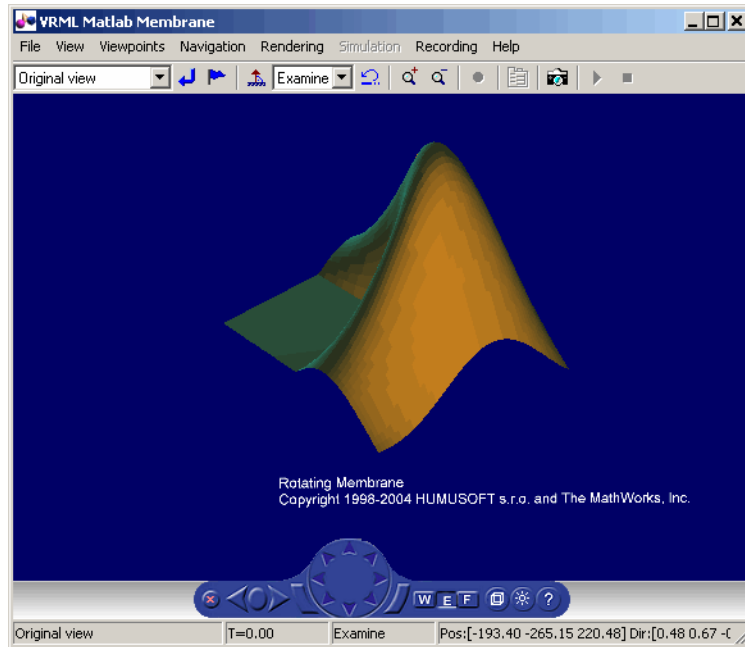
3 Use the viewer controls to move within the virtual world, or use the demo dialog box to rotate the membrane. Note that sometimes the Virtual Reality Toolbox Demo dialog box is hidden behind the viewer window.
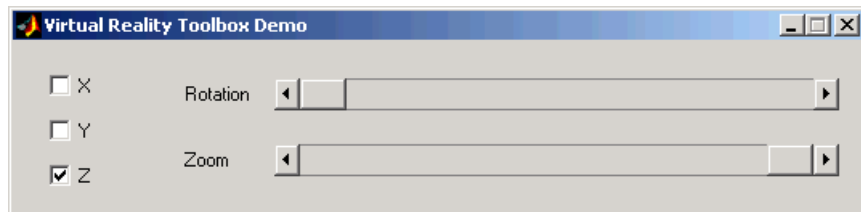
Simulink Interface

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

Associating a Virtual World with Simulink (p. 3-2)	Associate a Simulink model with a virtual world, and connect signals from the Simulink model to the virtual world
Using the Simulink Interface (p. 3-11)	Open a Simulink model, display the associated virtual world on a host computer or on a client computer, and observe the simulated process in the virtual world

Associating a Virtual World with Simulink

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 3-9

Section Overview

With Virtual Reality Toolbox you can interface a Simulink block diagram with a virtual world. The example in this section explains how to display a simulated virtual world on a host computer. This is the recommended way to view associated virtual worlds on the host computer.

Adding a Virtual Reality Toolbox Block

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

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

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

---

1 In the MATLAB Command Window, type

    vrtut2

A Simulink model opens without a Virtual Reality Toolbox block that connects the model to a virtual world.
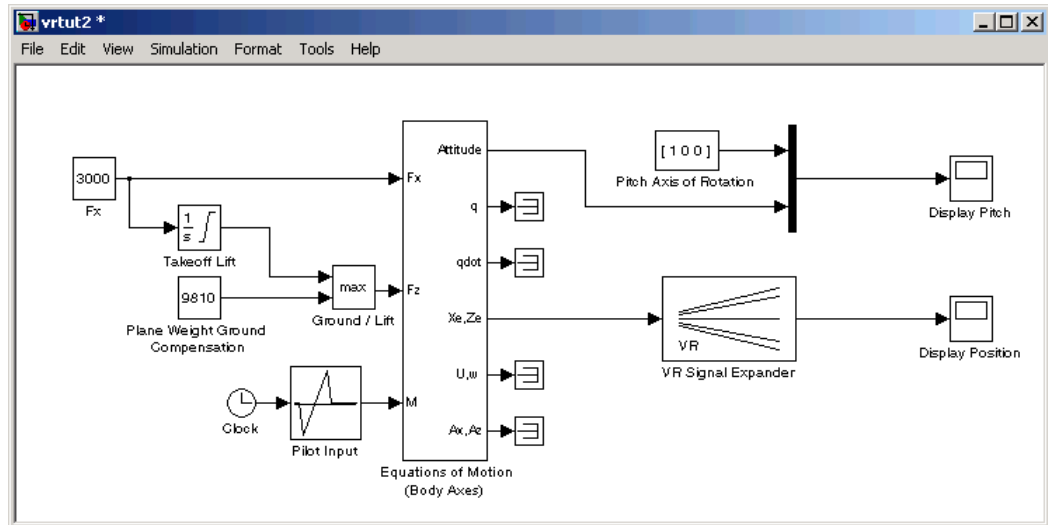
2 From the **Simulation** menu, select **Normal**, then click **Start**.

Observe the results of the simulation in the scope windows.

3 In the MATLAB Command Window, type

    vrlib

The Virtual Reality Toolbox library opens.
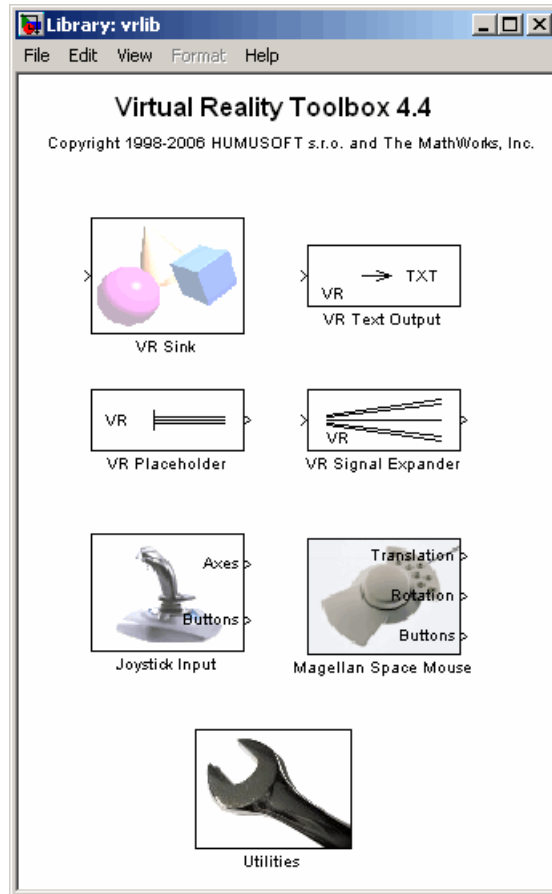
4 From the **Library** window, drag and drop the VR Sink block to the Simulink diagram. The VR Sink block writes data from the Simulink model to the virtual world. (For a description of all the Simulink blocks for Virtual Reality Toolbox, see Chapter 8, "Blocks — By Category") You can then close the **Library: vrlib** window.

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

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

The Parameters: VR Sink dialog box opens.
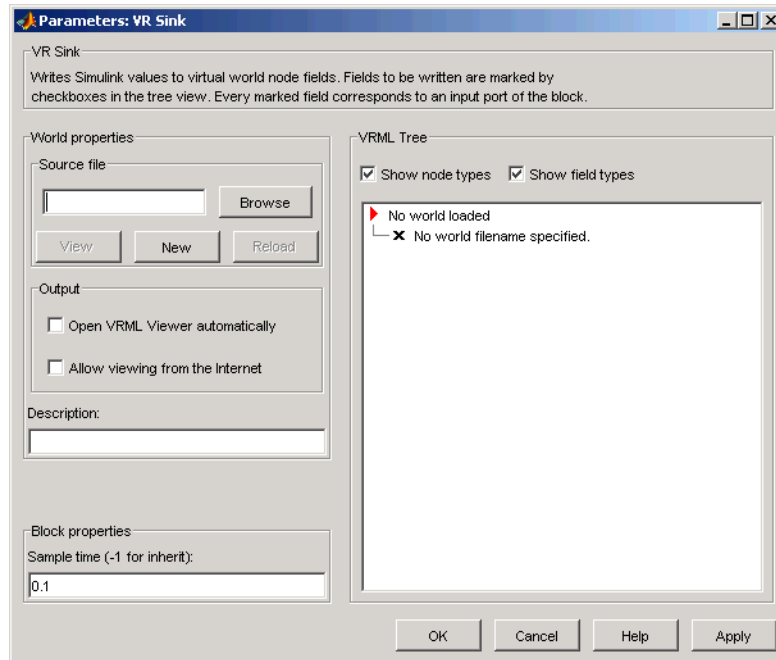
6 In the **Description** text box, enter a brief description of the model. This description appears on the list of available worlds served by the Virtual Reality Toolbox server. For example, type

```
VR Plane taking off
```

7 At the **Source File** text box, click the **Browse** button. The Select World dialog box opens. Find the directory <matlab root>\toolbox\vr\vrdemos. Select the file vrtkoff.wrl and click **Open**.

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

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

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

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

10 In the Plane (Transform) tree, select the translation and rotation fields.

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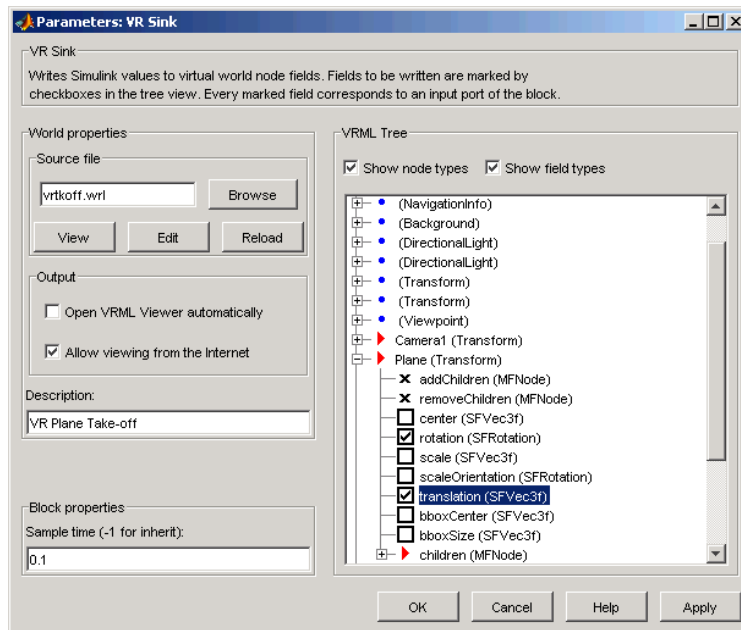
11 Click **OK**.

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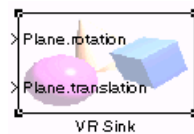
The first input is Plane rotation. The rotation is defined by a four-element vector. The first three numbers define the axis of rotation. In this example, it should be [1 0 0] for the *x*-axis (see the Pitch Axis of Rotation block in the model). The pitch of the plane is expressed by the rotation about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

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

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

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

After you connect the signals and remove the Scope blocks, your model should look similar to the figure shown.
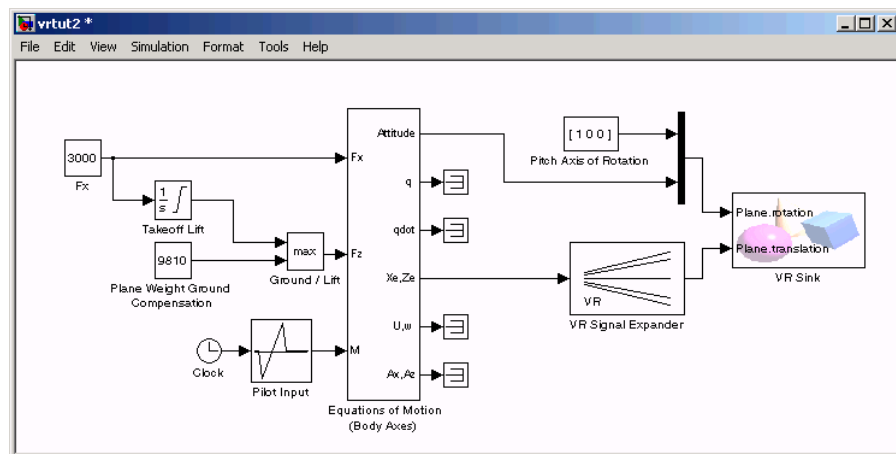
> **Note** Virtual world degrees of freedom have different requested input
> vector sizes depending on the associated VRML field types. If the vector
> size of the connected signal does not match the associated VRML field
> size, an `Incorrect input vector size` error is reported when you start
> the simulation.

14 Double-click the VR Sink block in the Simulink model. A viewer window
containing the plane's virtual world opens.
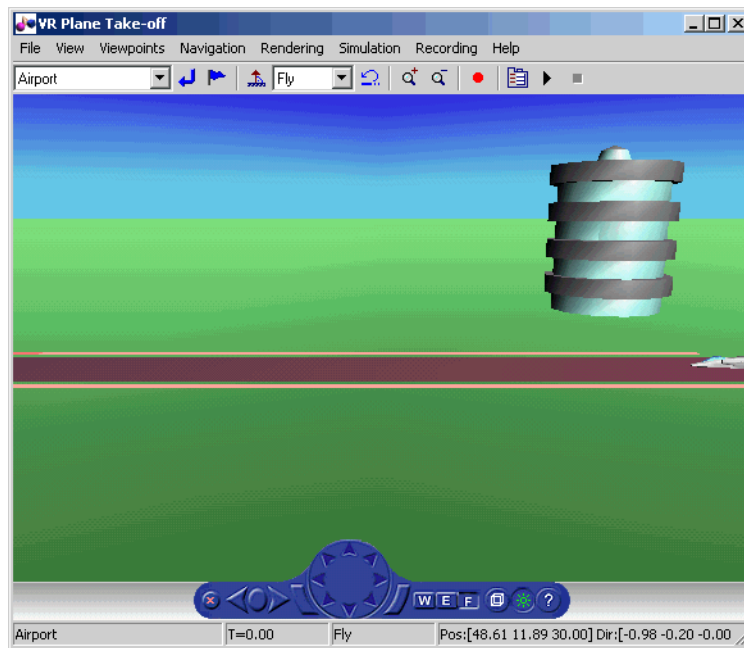
15 In the Virtual Reality Toolbox viewer, from the **Simulation** menu, click
Start to run the simulation.

A plane, moving right to left, starts down the runway and takes off into
the air.
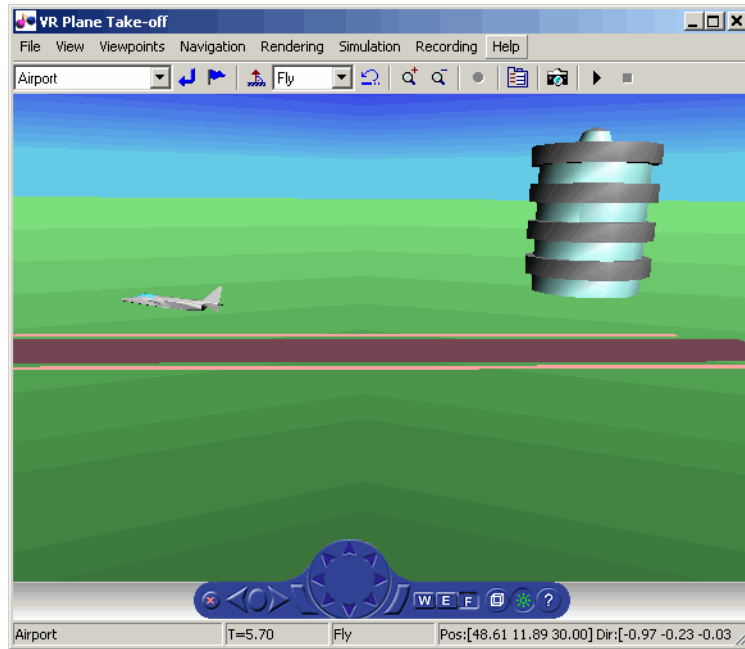
Changing the Virtual World Associated with a Simulink Block

On occasion, you might want to associate a different virtual world with a Simulink model or connect different signals.

After you associate a virtual world with a Simulink model, you can select another virtual world or change signals connected to the virtual world. This procedure assumes that you have connected the vrtut2 Simulink model with a virtual world. See "Adding a Virtual Reality Toolbox Block" on page 3-2.

1 Double-click the VR Sink block in the model. The viewer 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\vrdemos. 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. Simulink associates a new virtual world with the model.

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

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

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

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

7 Verify that the correct output is connected to your VR Sink block. The output from the VR Signal Expander should be connected to the single input.
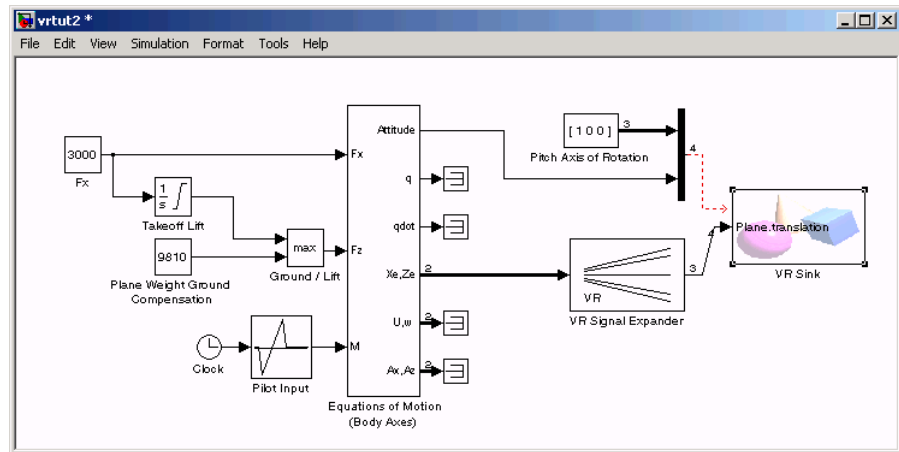
8 In the Virtual Reality Toolbox viewer, from the **Simulation** menu, run the simulation again and observe the simulation.

Using the Simulink Interface

In this section...

Section Overview

This section shows how to view a virtual world connected to a Simulink block diagram and make parameter changes from Simulink 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.
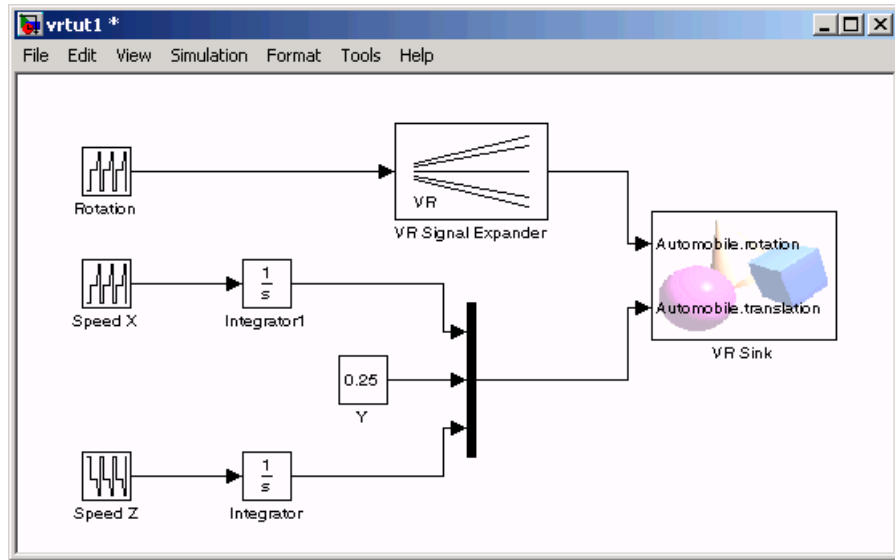
A VRML viewer also opens with a 3-D model of the virtual world associated with the model.
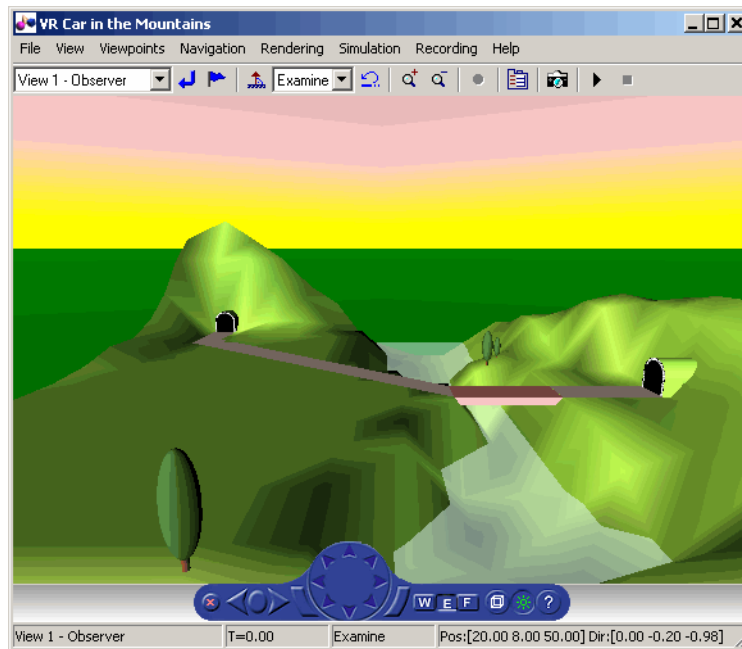
2 In the Virtual Reality Toolbox viewer, from the **Simulation** menu, click **Start**.

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 2-24.

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

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

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

Alternatively, you can view a virtual world in your Web browser by selecting an open virtual world from a list in your Web browser. You can display the HTML page that contains this list by connecting to the Virtual Reality Toolbox host. This is the computer on which the toolbox is currently running. You do not need a VRML-enabled Web browser to display this page.

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 Virtual Reality Toolbox lists the currently available (active) virtual worlds. In this example, the VR Bouncing Ball virtual world appears as a link.
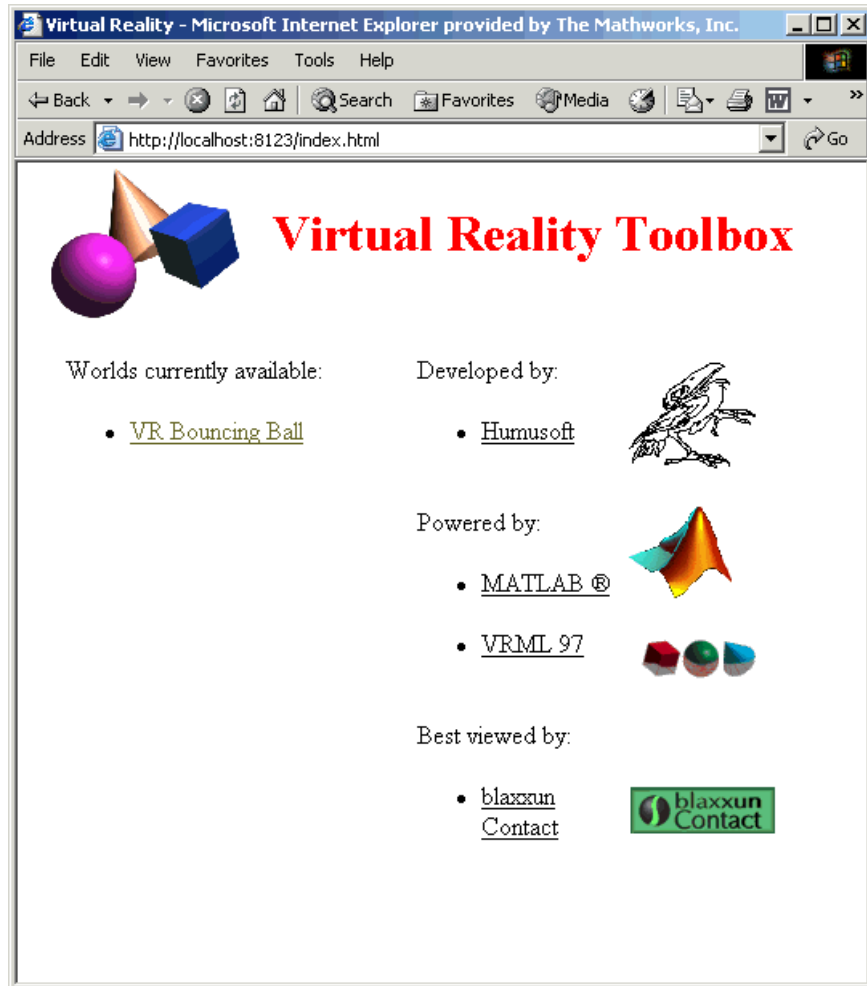
3 Click **VR Bouncing Ball**.

The VR Bouncing Ball virtual world appears in your Web browser.
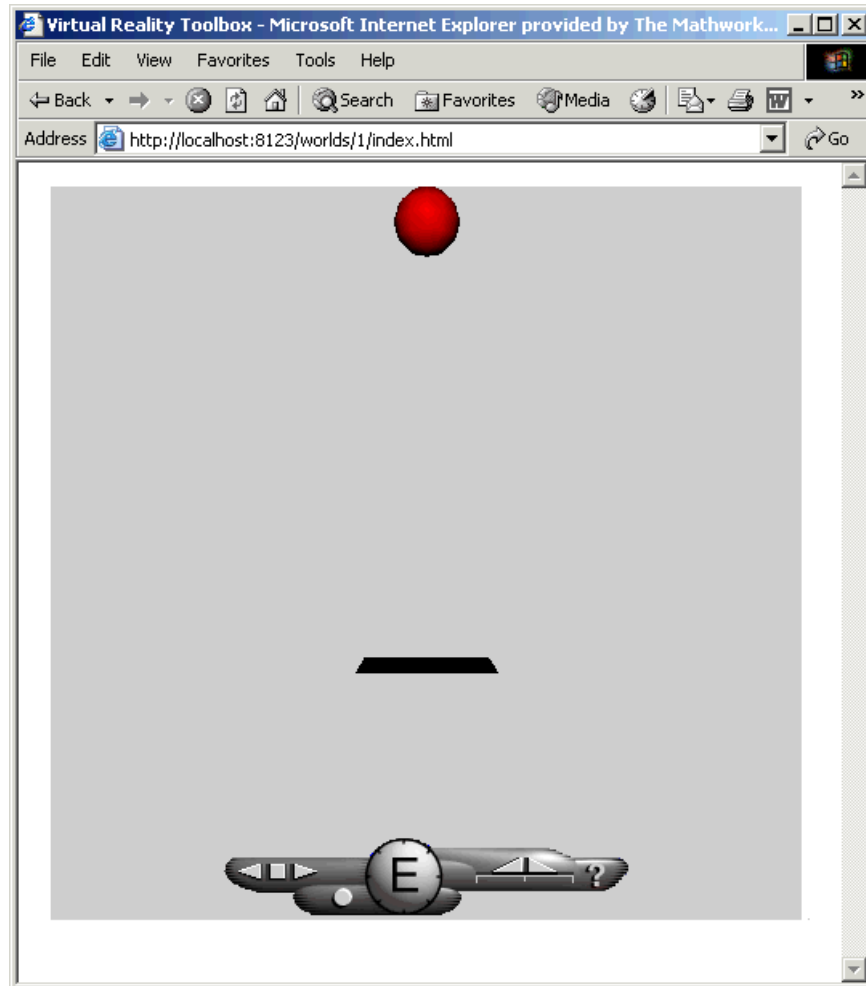
From the main HTML page, you can select one of the listed available worlds or click the **reload** link to update the status of the virtual worlds supported by the toolbox. This page does not require the VRML capabilities from the browser; it is a standard HTML page. Nevertheless, when you click one of the virtual world links in the list, the browser has to be VRML-enabled to display the virtual world correctly and to communicate with Virtual Reality Toolbox.

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

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

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

This example explains how to display a simulated virtual world on a client computer. In this case, the client computer is a PC platform with the blaxxun Contact plug-in. In this example, a Simulink window opens with the model of a simple automobile. The automobile trajectory (vehicle position and angle) is viewed in virtual reality:

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

```
vrtut1
```

A Simulink window opens with the model of an automobile.
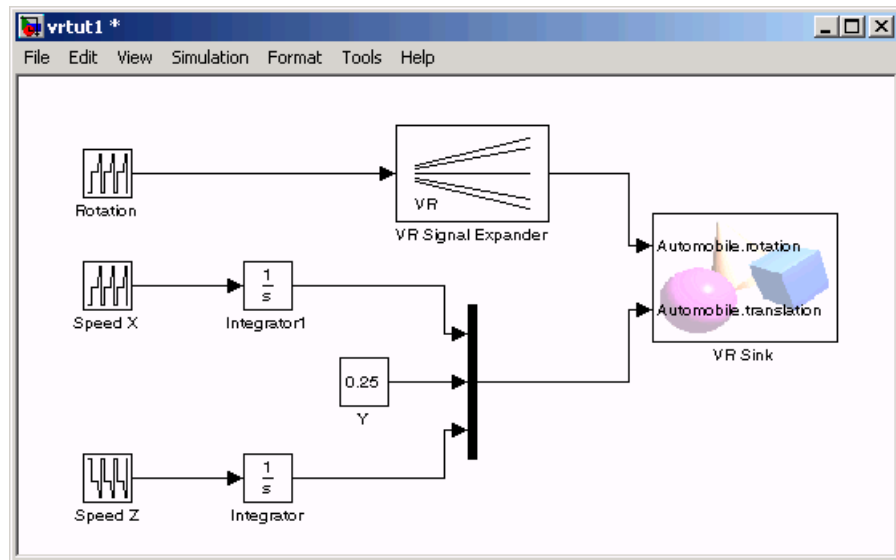
2 Double-click the VR Sink block. This block is in the right part of the model window.

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

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

A Parameters: VR Sink dialog box opens.
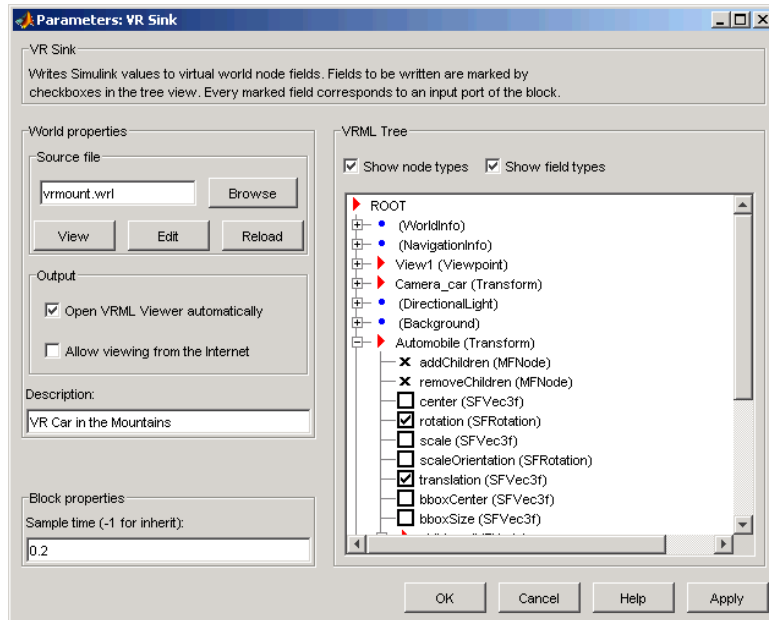
4 Select the **Allow viewing from the Internet** check box.

Note This option allows any computer connected to the network to view your model. You should never select this box when you want your model to be private or confidential.

---

5 Click **OK**.

6 On the client computer, open your VRML-enabled Web browser. In the **Address** line, enter the address and Virtual Reality Toolbox port number for the host computer running Simulink. For example, if the IP address of the host computer is 192.168.0.1, enter

```
http://192.168.0.1:8123
```

To determine your IP address on a Windows system,

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

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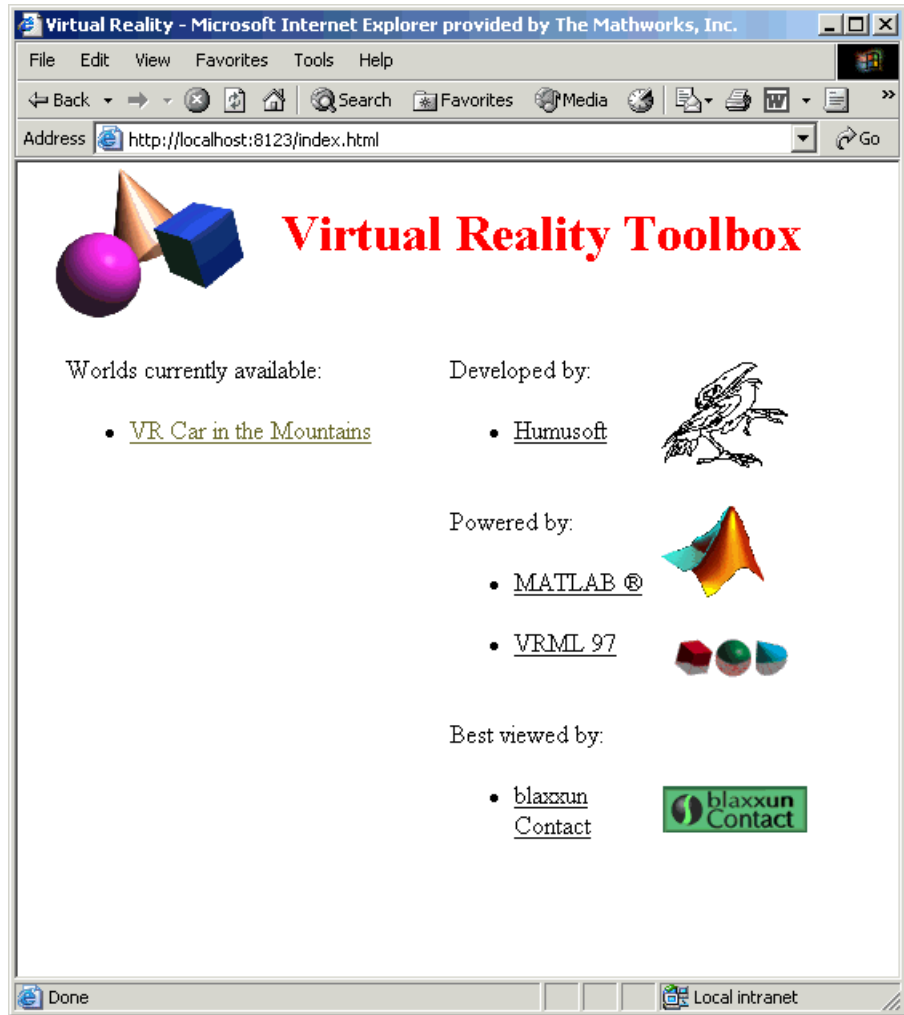
7 Click **VR Car in the Mountains**.

The Web browser displays a 3-D model of the virtual world associated with the model.
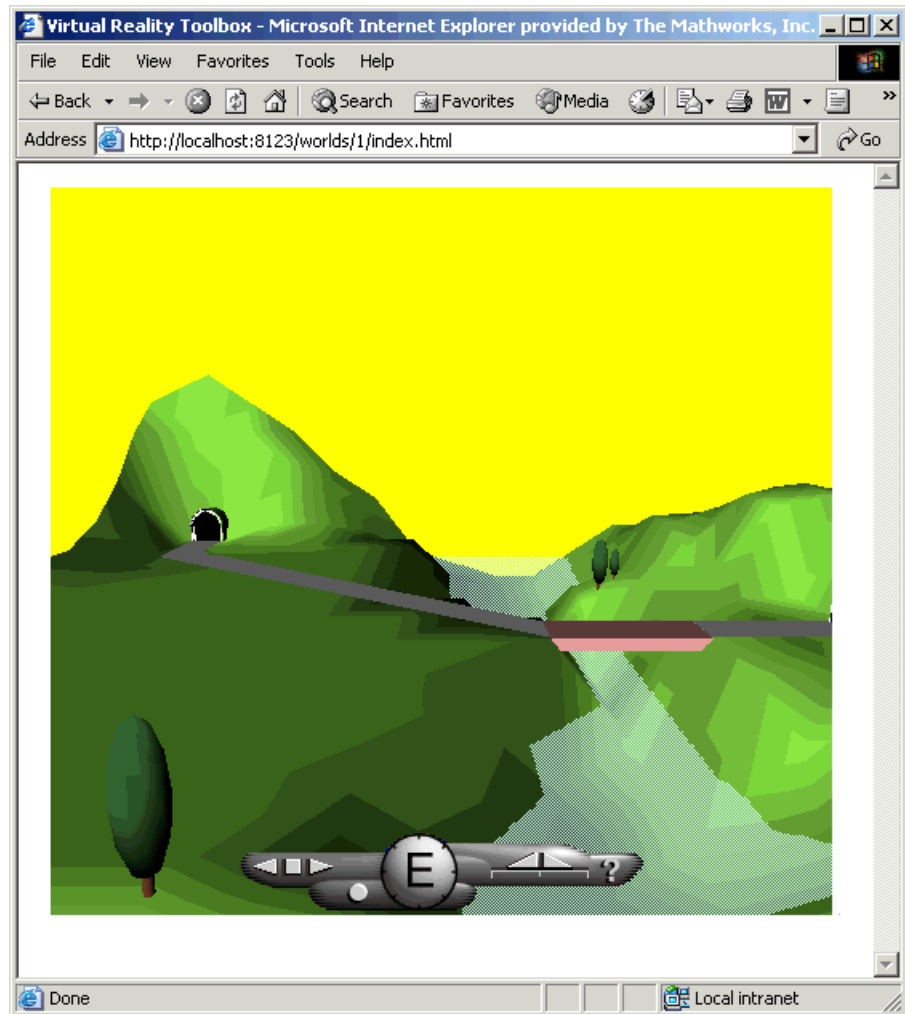
8 On the host computer, in the Simulink window, from the **Simulation** menu, click **Start**.

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

Using the MATLAB Interface

Creating a vrworld Object

To connect MATLAB to a virtual world and to interact with that virtual world through the MATLAB command-line interface, you need to create vrworld and vrnode objects. You cannot directly interact with a virtual world. A virtual world is defined by a VRML file with the extension .wrl. For a complete list of virtual world methods, see "vrworld Object Methods" on page 10-2, "vrnode Object Methods" on page 10-3, and "vrfigure Object Methods" on page 10-3.

Note The Simulink interface and the MATLAB interface share the same virtual world objects. This enables you to use the MATLAB interface to change the properties of vrworld objects originally created by Simulink with Virtual Reality Toolbox blocks.

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

1 Open MATLAB. In the MATLAB Command Window, type

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

MATLAB displays output like

```
myworld =
 vrworld object: 1-by-1

VR Car in the Mountains
(<matlab-root>/toolbox/vr/vrdemos/vrmount.wrl)
```

2 Type

```
vrwhos
```

MATLAB displays the messages

```
Closed, associated with
'C:<matlab root>\toolbox\vr\vrdemos\vrmount.wrl'.
Visible for local viewers.
No clients are logged on.
```

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

Your next step is to open a virtual world using the vrworld object. See "Opening a Virtual World" on page 4-3.

Opening a Virtual World

Opening a virtual world lets you view the virtual world in a VRML viewer, scan its structure, and change virtual world properties from the MATLAB Command Window.

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

1 In the MATLAB Command Window, type

```
open(myworld);
```

MATLAB opens the virtual world vrmount.wrl.

2 Type

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

The Description property is changed to My first virtual world. This is the description that is displayed in all Virtual Reality object listings, in the title bar of the Virtual Reality Toolbox viewer, and in the list of virtual worlds on the Virtual Reality Toolbox HTML page.

3 Display the virtual world vrmount.wrl. Type

```
view(myworld)
```

The viewer that is set as the default viewer displays the virtual scene. This is typically the Virtual Reality Toolbox viewer unless you have a different viewer set.

Alternatively, you can display the virtual world in a VRML-enabled Web browser.

1 Repeat steps 1 and 2 of the preceding procedure.

2 Open a Web browser. In the **Address** box, type

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

The browser displays the Virtual Reality Toolbox HTML page with a link to **My first virtual world**. The number 8123 is the default Virtual Reality Toolbox port number. If you set a different port number on your system, enter that number in place of 8123. For more information on the Virtual Reality Toolbox HTML page, see "Viewing a Virtual World with a Web Browser on the Host Computer" on page 3-14.

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

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

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

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

Interacting with a Virtual World

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

An object of type vrworld contains nodes named in the VRML file using the DEF statement. These nodes are of type vrnode. For more information, see "vrworld Object Methods" on page 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);
```

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

```
Tunnel (Transform) [My first virtual world]
Road (Shape) [My first virtual world]
Bridge (Shape) [My first virtual world]
River (Shape) [My first virtual world]
ElevApp (Appearance) [My first virtual world]
Canal (Shape) [My first virtual world]
```

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

2 Type

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

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

```
    mynodes =

    vrnode object: 13-by-1

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

3 Type

```
    whos
```

MATLAB displays the messages

```
    Name          Size                    Bytes  Class
```

```
ans            1x1                        132  vrfigure object
mynodes        13x1                      3564  vrnode object
myworld        1x1                        132  vrworld object
```

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

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

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

or

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

MATLAB displays information like the following table.

```
    Field                Access       Type         Sync


    --------------------------------------------------------
    addChildren          eventIn      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];
```

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

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

Note Optimally, use the Virtual Reality Toolbox viewer (Chapter 6, "Viewing Virtual Worlds") to record animations of virtual worlds associated with Simulink models. This method ensures that all necessary virtual world and `vrfigure` properties are properly set to record simulations. The Virtual Reality Toolbox viewer is the recommended interface to record animations. If you are working with virtual scenes controlled from MATLAB, you can still record virtual scenes through the MATLAB interface.

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

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

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

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

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

Animation Recording File Tokens

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

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

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

```
vrplanets_anim_1.wrl
```

If you run and record the simulation a second time, the animation filename is vrplanets_anim_2.wrl.

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

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

Token	Description
%d	The full path to the world VRML file replaces this token in the filename string and creates files in directories relative to the virtual world file location. For example, the format %d/animdir/animfile.avi saves the animation into the animdir subdirectory of the directory containing the virtual world VRML file. This token is most helpful if you want to ensure that the virtual world file and animation file are in the same directory.
%D	The current day in the month replaces this token in the filename string. For example, the format %f_anim_%D.wrl saves the animation to vrplanets_anim_29.wrl for the 29th day of the month.
%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.

Token	Description
%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 Virtual Reality Toolbox manually record the animation, set the
RecordMode property to manual. Type

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

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

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

5 Run the Simulink model. From the **Simulation** menu, select **Normal,**
then click **Start**. Alternatively, if you are using the Virtual Reality Toolbox
default viewer, you can run the Simulink model with one of the following
from the viewer.

- From the menu bar, select the **Simulation** menu **Start** option to start or
  stop the simulation.

- From the toolbar, click **Start/pause/continue simulation** to start the
  simulation.

- From the keyboard, press **Ctrl+T** to start the simulation.

6 As the simulation runs, start recording the animation by setting the virtual
world Recording property. Type

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

This turns on the recording state.

7 When you want to stop the recording operation, type

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

Virtual Reality Toolbox stops recording the animation. Virtual Reality
Toolbox creates the file vrplanets_anim_1.wrl in the current working

directory. If the simulation stops before you stop recording, the recording operation stops and creates the animation file.

8 Stop the simulation. You can use one of the following from the viewer.

- From the menu bar, select the **Simulation** menu **Stop** option to stop the simulation.

- From the toolbar, click **Stop simulation** to stop the simulation.

- From the keyboard, press **Ctrl+T** to stop the simulation.

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

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

Manual 2-D AVI Animation Recording

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

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

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

```
vrplanets
```

The Simulink model 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 Virtual Reality Toolbox manually record the animation, set the `RecordMode` property to `manual`. Type

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

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

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

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

```
set(f,'PanelMode','off');
```

8 Run the Simulink model. From the **Simulation** menu, select **Normal**, then click **Start**. Alternatively, if you are using the Virtual Reality Toolbox default viewer, you can run the Simulink model with one of the following from the viewer:

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

9 As the simulation runs, start recording the animation by setting the virtual world `Recording` property. Type

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

This turns on the recording state.

10 To stop the recording operation, type

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

Virtual Reality Toolbox stops recording the animation. Virtual Reality Toolbox creates the file vrplanets_anim_1.avi in the current working directory. If the simulation stops before you stop recording, the recording operation stops and creates the animation file.

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

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

4-17

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. Virtual Reality Toolbox records the animation during this interval in the simulation. In this example, the timing of the recorded animation file derives from the simulation time. One second of the recorded animation time corresponds to one second of Simulink time.

This procedure uses the vrplanets demo. It describes how to create a VRML animation filename with the default name format.

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

```
vrplanets
```

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

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

```
vrwhos
```

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

```
myworld = vrwho;
```

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

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

For the string comparison method, type

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

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

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

5 Direct Virtual Reality Toolbox to record the animation in a VRML format file.

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

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

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

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

7 Run the Simulink model. From the **Simulation** menu, select **Normal,** then click **Start**. Alternatively, if you are using the Virtual Reality Toolbox default viewer, you can run the Simulink model with one of the following from the viewer.

- From the menu bar, select the **Simulation** menu **Start** option to start the simulation.

- From the toolbar, click **Start/pause/continue simulation** to start the simulation.

- From the keyboard, press **Ctrl+T** to start the simulation.

The simulation runs. Virtual Reality Toolbox starts recording when the simulation time reaches the specified start time and creates the file vrplanets_anim_N.wrl in the current working directory when finished, where N is either 1 or more, depending on how many file iterations you have.

8 When you are done, stop the simulation. You can use one of the following from the viewer.

- From the menu bar, select the **Simulation** menu **Stop** option to stop the simulation.

- From the toolbar, click **Stop simulation** to stop the simulation.

- From the keyboard, press **Ctrl+T** to stop the simulation.

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

Scheduled 2-D AVI Animation Recording

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

This procedure uses the vrplanets demo. It describes how to create an .avi animation filename with the default name format.

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

    vrplanets

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

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

```
vrwhos
```

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

```
myworld = vrwho;
```

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

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

For the string comparison method, type

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

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

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

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

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

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

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

4-21

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

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

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

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

```
set(f,'PanelMode','off');
```

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

9 Run the Simulink model. From the **Simulation** menu, select **Normal**, then click **Start**. Alternatively, if you are using the Virtual Reality Toolbox default viewer, you can run the Simulink model with one of the following from the viewer:

- From the menu bar, select the **Simulation** menu **Start** option to start the simulation.

- From the toolbar, click **Start/pause/continue simulation** to start the simulation.

- From the keyboard, press **Ctrl+T** to start the simulation.

The simulation runs. Virtual Reality Toolbox starts recording when the simulation time reaches the specified start time and creates the file vrplanets_anim_N.avi in the current working directory when finished, where N is either 1 or more, depending on how many file iterations you have.

10 When you are done, stop the simulation. You can use one of the following from the viewer:

- From the menu bar, select the **Simulation** menu **Stop** option to stop the simulation.

- From the toolbar, click **Stop simulation** to stop the simulation.

- From the keyboard, press **Ctrl+T** to stop the simulation.

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

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

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

Viewing Animation Files

This topic assumes that you have a VRML or .avi animation file that you want to view. If you do not have an animation file, see "Manual 3-D VRML Animation Recording" on page 4-12 or "Scheduled 3-D VRML Animation Recording" on page 4-18 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 MATLAB.

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) 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,

  a Retrieve the corresponding `vrfigure` object. For 2-D animations, the Virtual Reality Toolbox records exactly what you see in the viewer window. Because 2-D animations record exactly what you see in the Virtual Reality Toolbox viewer window, the properties that control 2-D file recording belong to `vrfigure` objects.

  b Set the `Record2D` vrfigure property.

  c To override default filenames for animation files, set the `vrfigure Record2DFileName` property.

- To create 3-D animation files,

  a Retrieve the corresponding `vrworld` object.

b Set the `Record3D` vrworld property.

c To override default filenames for animation files, set the vrworld `Record3DFileName` property.

- Set the `RecordMode` vrworld object property to `manual` or `scheduled`. For optimal results, select `scheduled`.

- If you select `scheduled` for `RecordMode`, be sure to also set the vrworld `RecordInterval` property to a desired time interval.

- To specify that the virtual world time source is an external one, set the vrworld property `TimeSource` to `external`. This ensures that MATLAB controls the virtual world scene time. Type

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

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

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

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

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

If you select a start time of 60 and a stop time of 120 (as described in "Scheduled 3-D VRML Animation Recording" on page 4-18), Virtual Reality Toolbox starts recording at 60 and stops at 120.

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

- After you set the `vrworld` `Time` property, set the virtual scene object properties as necessary. You should set these properties to values that correspond to the given time frame to achieve the desired animation effect.

- In each time frame, issue the `vrdrawnow` command for scene changes. This command renders and updates the scene.

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

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

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

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

5

Virtual Worlds

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

VRML Editing Tools

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. V-Realm Builder by Ligos Corporation is one of the exceptions. It is one of the most advanced VRML editing tools currently available for personal computers. V-Realm Builder is available only for Windows operating systems. You can access V-Realm Builder documentation at `http://www.mathworks.com/support/product/VR/productnews/productnews.html`.

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

V-Realm Builder

V-Realm Builder is a flexible, graphically oriented tool for 3-D editing and is available for Windows operating systems only. It is a native VRML authoring tool that provides a convenient interface to the VRML syntax. Its primary file format is VRML. Its graphical user interface (GUI) offers not only the graphical representation of a 3-D scene and tools for interactive creation of graphical elements, but also a hierarchical tree style (tree viewer) of all the elements present in the virtual world.

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

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

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

Deformation of a Sphere Example

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 5-8
"Creating a Box in a Virtual World" on page 5-13
"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 V-Realm Builder. It does not describe everything you can do with V-Realm Builder, but it does describe the basics to get you started.

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

Defining the Problem

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

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

Object	Center Position	Dimensions
B1	[3 0 0]	[0.3 1 1]

Object	Center Position	Dimensions
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.

Virtual Reality Toolbox includes the tutorial model vrtut3.mdl. This is a simplified model in which the deformation of an elastic sphere is simulated. After collision with the rigid blocks, the sphere's *x* dimension is decreased by a factor from 1 to 0.4, and the *y* and *z* dimensions are expanded so that the volume of the deformed sphere-ellipsoid remains constant. Additional blocks in the model supply the correctly sized vectors to the Virtual Reality Toolbox block. The simulation stops when the sphere is deformed to 0.4 times its original size in the *x* direction.

Adding a Virtual Reality Toolbox Block

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

1 From the directory C:*matlabroot*\toolbox\vr\vrdemos\, copy the file vrtut3.mdl to your MATLAB working directory.

2 Start MATLAB, and then change the current directory to your MATLAB working directory.

3 In the MATLAB Command Window, type

   vrtut3

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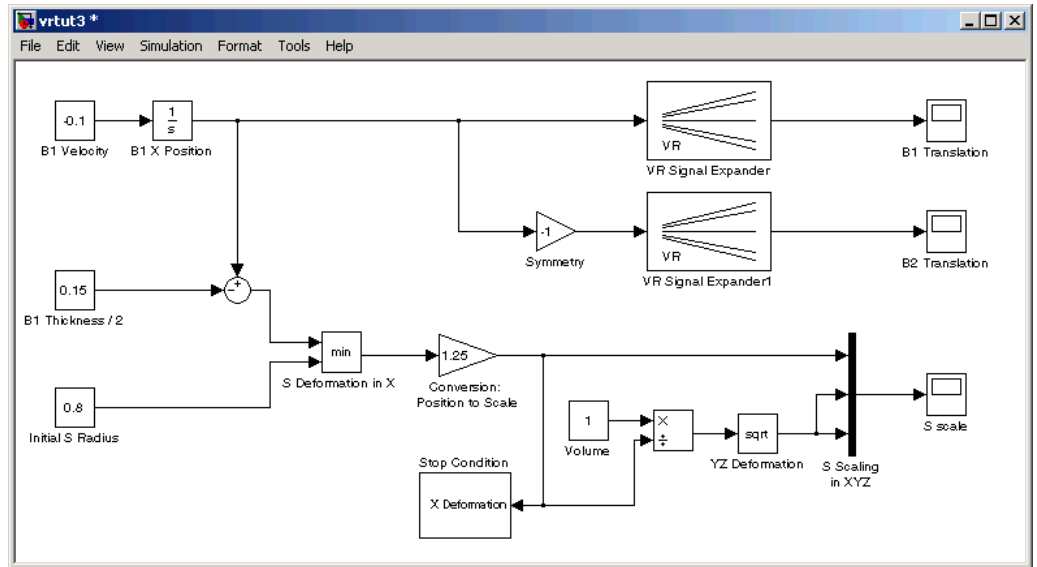
4 From the MATLAB Command Window, type

```
vrlib
```

The Virtual Reality Toolbox library opens.
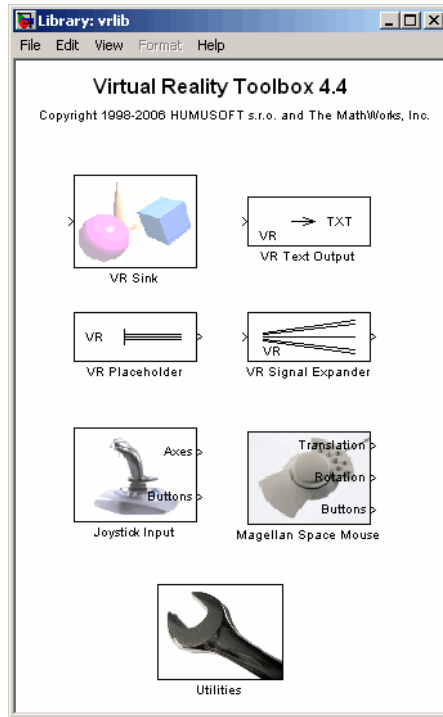
5 From the Library window, drag and drop the VR Sink block to the Simulink diagram. You can then close the **Library: vrlib** window.

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

Creating a Sphere in a Virtual World

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

After you add a VR Sink block to your Simulink model, you can create a virtual world using the V-Realm Builder. 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 Microsoft Windows task bar, click **Start**, and then click **Run**.

2 In the Run dialog box, enter

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

The V-Realm Builder application window opens.

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

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

4 On the toolbar, click the sphere icon .

In the left pane you can see the VRML tree for a sphere. This tree includes the nodes Transform, Shape, Appearance, Material, and Sphere. A yellow icon indicates the field of a node.
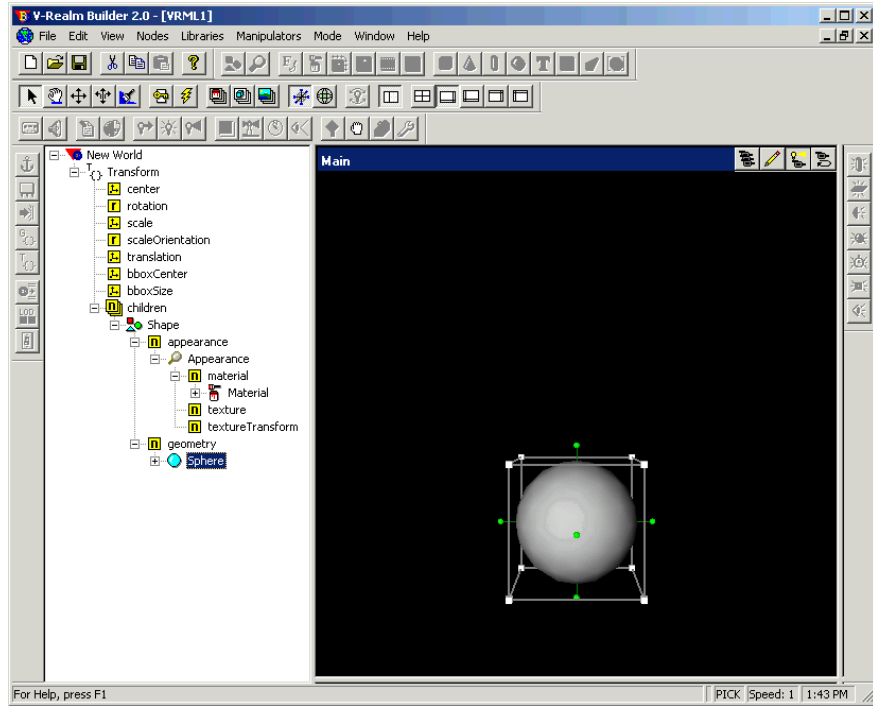
The top-level node is the `Transform` node. This grouping node allows you to change the position and scale of objects (children) that are part of this node. Its subtree consists of one object, which is described in the `Shape` node. The `Shape` node contains the `appearance` and `geometry` fields.

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.
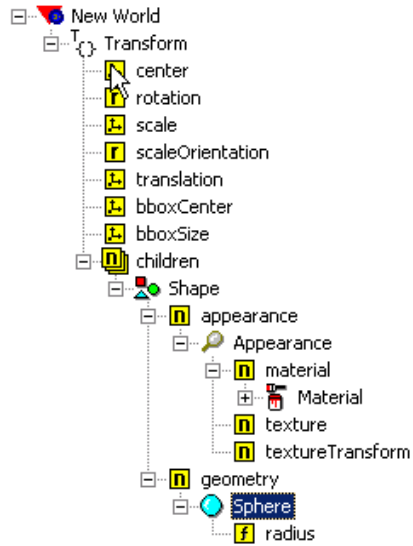
6 Double-click the radius field.

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.
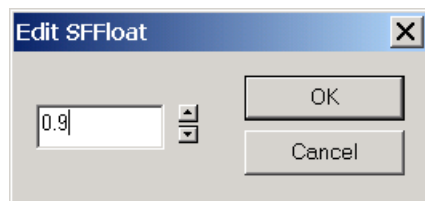
8 Under the Shape node, expand the appearance field. Under the appearance field, expand the Appearance node. Under the Appearance node, expand the material field. Under the material field, expand the Material node.
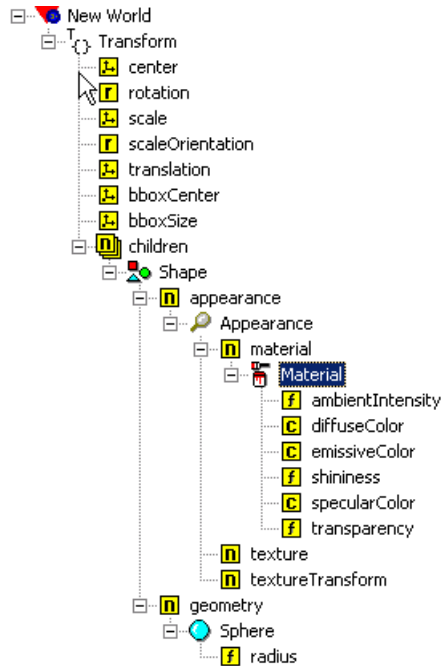
9 Under the `Material` node, double-click the `diffuseColor` field.

The Edit Color dialog box opens.

10 Set the color to blue or any other color you would like, and then click **OK**.
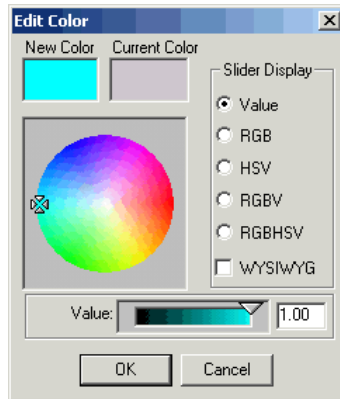
11 If you want to check or modify the position of the sphere, double-click the `translation` field under the `Transform` node. You do not need to change the default values from [0 0 0].

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

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

The text appears in edit mode.

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

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

Creating a Box in a Virtual World

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

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

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

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

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

3 Select the **X Axis** check box. In the text box below, enter **3**, then click **OK**.
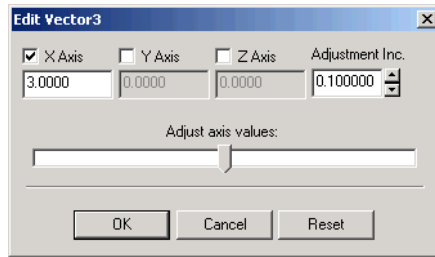
The position of the box is set to [3 0 0].

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

The Edit Vector 3 dialog box opens.
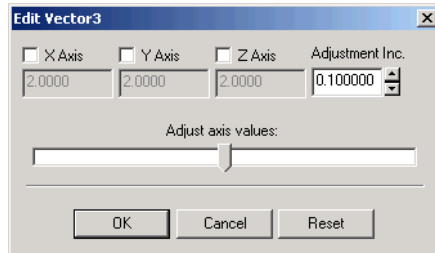
5 Set the values to [0.3 1 1], and then click **OK**.

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

The following figure shows the tree with the expanded branch of the box.
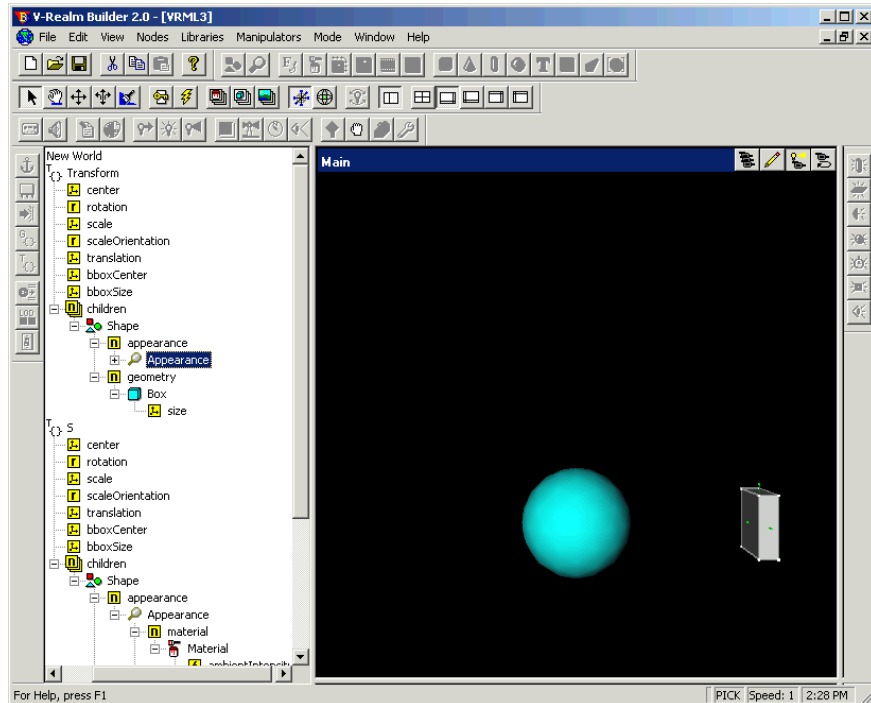
6 Create a second box the same way you created the first box.

7 To move the second box to its correct place, double-click the `translation` field of the second `Transform` node, and change its position to `[-3 0 0]`.

8 Double-click the `size` field under the `Box` node. Set the values to `[0.3 1 1]`, and then click **OK**.

The scene is now complete.

9 To access the positions of the boxes from a Virtual Reality Toolbox block, give each `Transform` node a name. For example, set the name of the first `Transform` node to B1 and the second `Transform` node to B2. Virtual Reality Toolbox allows you to access fields of only those nodes that are named in virtual worlds.
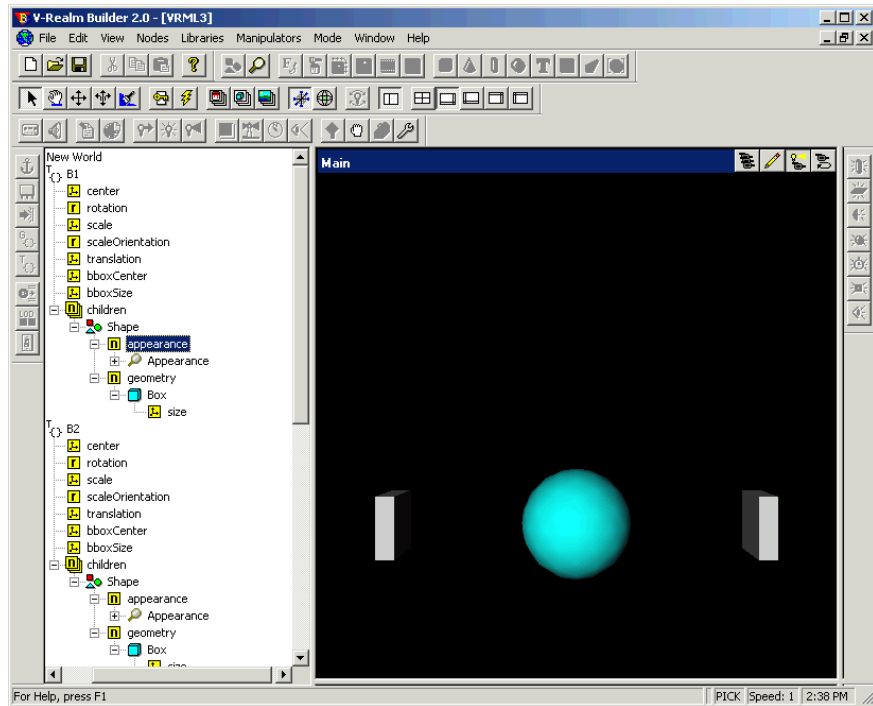
10 Save the virtual world as `vrtut3.wrl` in the same working directory where the file `vrtut3.mdl` resides, and then exit V-Realm Builder.

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

Your next task is to connect the model outputs to the Virtual Reality Toolbox block in your Simulink model. See "Connecting a Simulink Model to a Virtual World" on page 5-17.

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.wrl, 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.
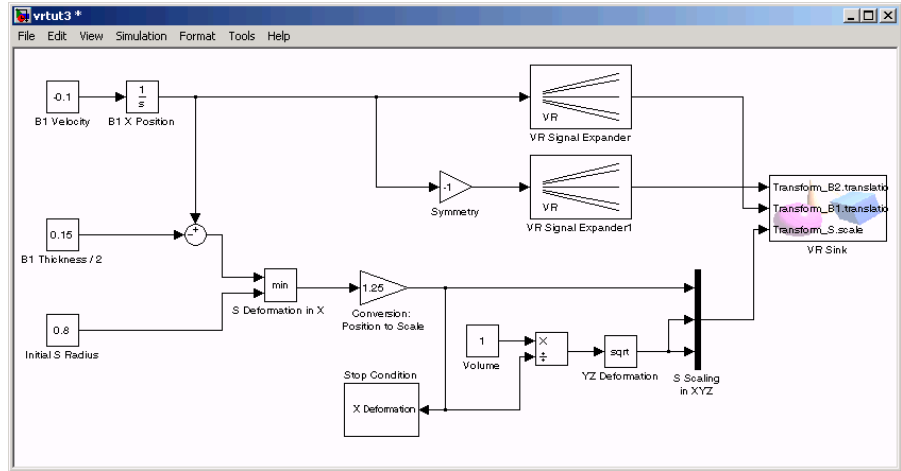
10 Double-click the VR Sink block.

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 2-24.
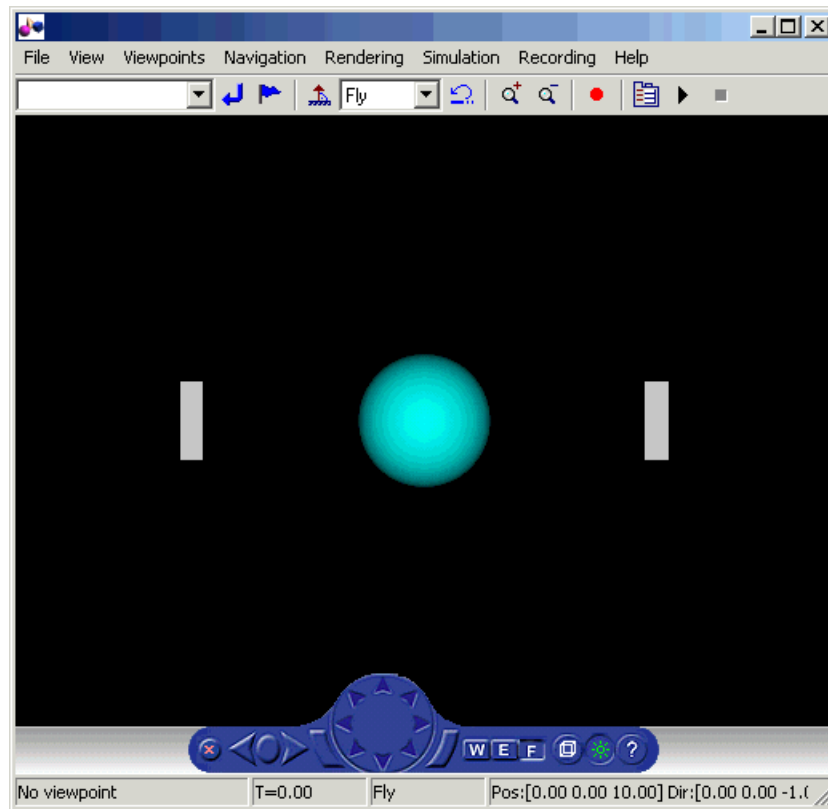
13 In the Simulink window, from the **Simulation** menu, click **Start**.

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

When the width of the sphere is reduced to 0.4 of its original size, the simulation stops running.
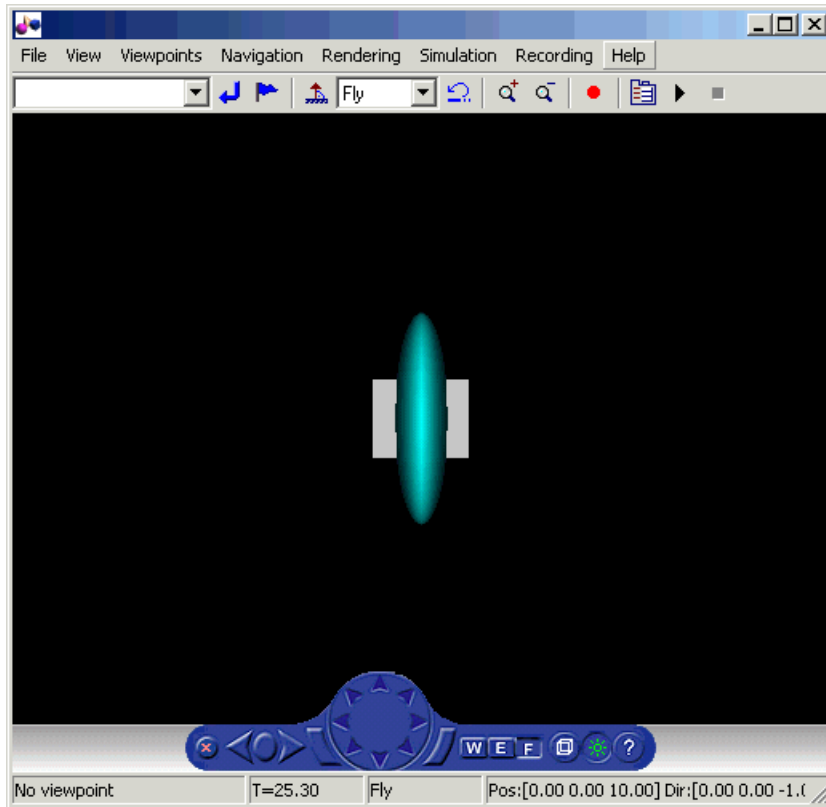
This example shows you how to create and use a very simple virtual reality model. Using the same method, you can create more complex models for solving the particular problems that you face.

VRML Data Types

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

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

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

VRML Type	Description	VR Toolbox Type
SFBool	Boolean value true or false.	logical
SFFloat	32–bit, floating-point value.	single
SFInt32	32–bit, signed-integer value.	int32
SFTime	Absolute or relative time value.	double

VRML Type	Description	VR Toolbox Type
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)

VRML Type	Description	VR Toolbox Type
MFRotation	Array of SFRotation values.	Single array (n-by-4)
MFString	Array of SFString values.	char array (n-by-1)
MFNode	Array of SFNode values.	N/A

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

- The setfield function (and its dot notation form) and **VR Sink** inputs accept all meaningful data types on input. Both convert the data types into natural VRML types as necessary. The data types include logicals, signed and unsigned integers, singles, and doubles.

- The getfield function (and its dot notation form) return their natural data types according to the table above.

To ensure backward compatibility with existing models and applications, use the Virtual Reality Toolbox vrsetpref function to define the data type support. Their names are as follows:

Property	Description
DataTypeBool	Specifies the boolean data type for vrnode/setfield and vrnode/getfield. 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
DataTypeInt32	Specifies the int32 data type for vrnode/setfield and vrnode/getfield. Valid values are 'int32' and 'double'. If set to 'int32', the VRML int32 data type is returned as int32. If set to 'double', the VRML int32 data type is returned as 'double'.
DataTypeFloat	Specifies the float data type for vrnode/setfield and vrnode/getfield. Valid values are 'single' and 'double'. If set to 'single', the VRML float and color data types (the types of most VRML fields) are returned as 'single'. If set to 'double', the VRML float and color data types are returned as 'double'.

VRML Data Class Types

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

VRML Data Class	Description
eventIn	An event that can be received by the node
eventOut	An event that can be sent by the node
field	A private node member, holding node data
exposedField	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 Virtual Reality Toolbox

In this section...
"Section Overview" on page 5-26
"Exporting VRML Models from CAD Tools" on page 5-26
"Virtual Scene Modeling" on page 5-32
"Linking Virtual Scene to a Simulink, SimMechanics, or MATLAB Model" on page 5-36

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 MATLAB-based applications.

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

This section assumes that the reader has a moderate knowledge of Virtual Reality Toolbox. 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 Virtual Reality Toolbox. 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 5-27
- "Units Used in Exported Files" on page 5-28
- "Coordinate System Used" on page 5-28
- "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). Virtual Reality Toolbox uses VRML97, so select VRML97 as the export format.

If your CAD tool allows only VRML1 export, you can use V-Realm Builder, a native VRML scene editor supplied with Virtual Reality Toolbox, 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 tesselation, 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 Virtual Reality Toolbox. 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.
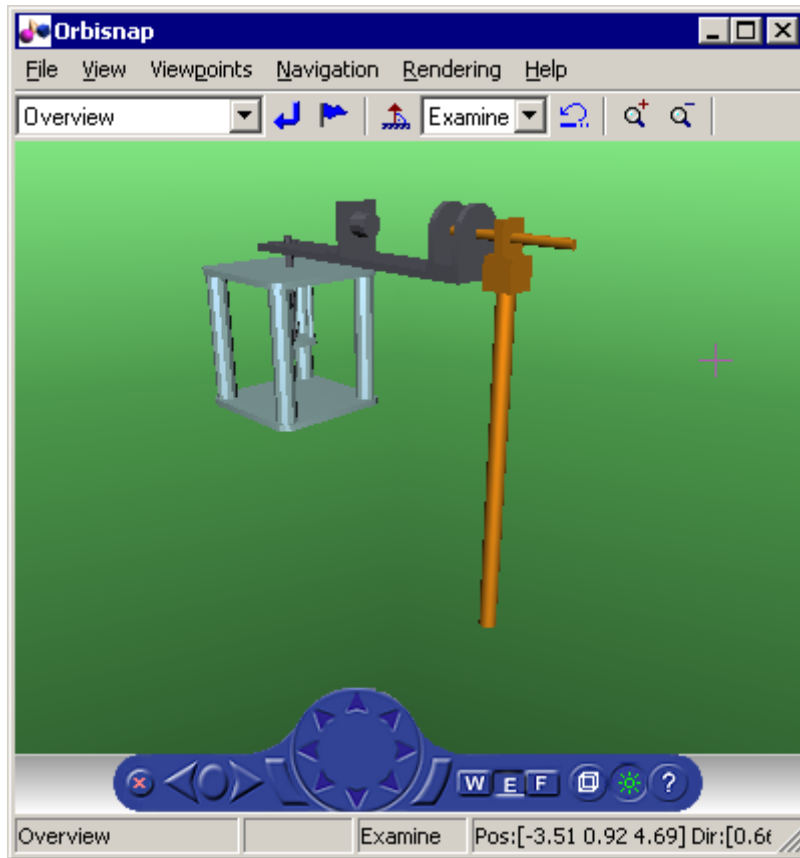
If the pendulum dynamics model uses global coordinates for all moving parts, the VRML model has a flat structure as shown in the following figure.
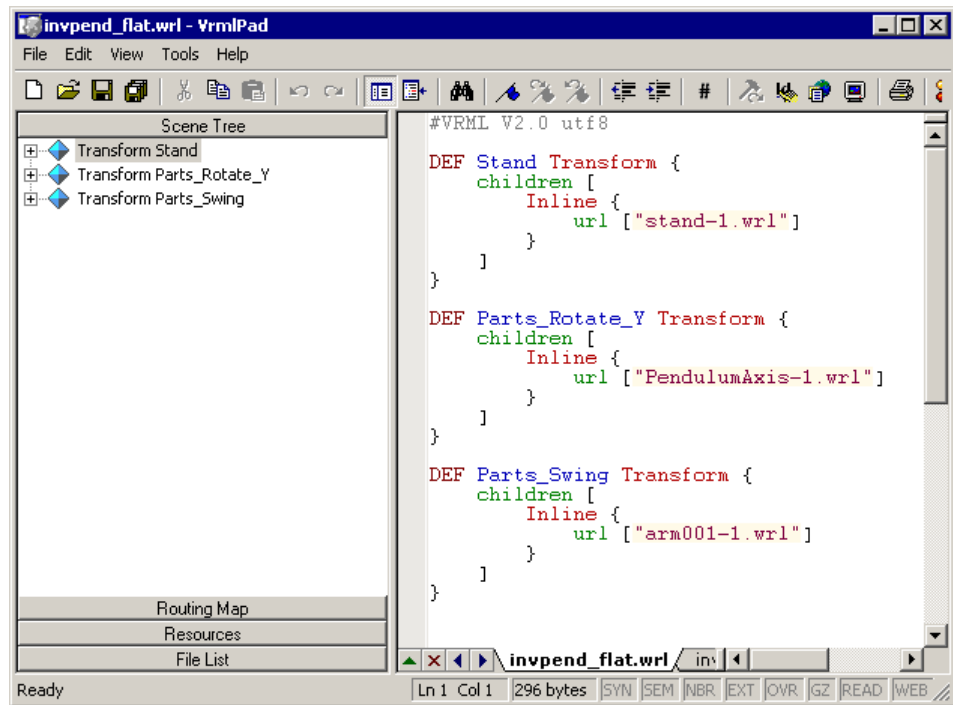
If the pendulum dynamics model uses local coordinates for moving parts, the corresponding VRML model has a nested structure, as shown in the following figure.
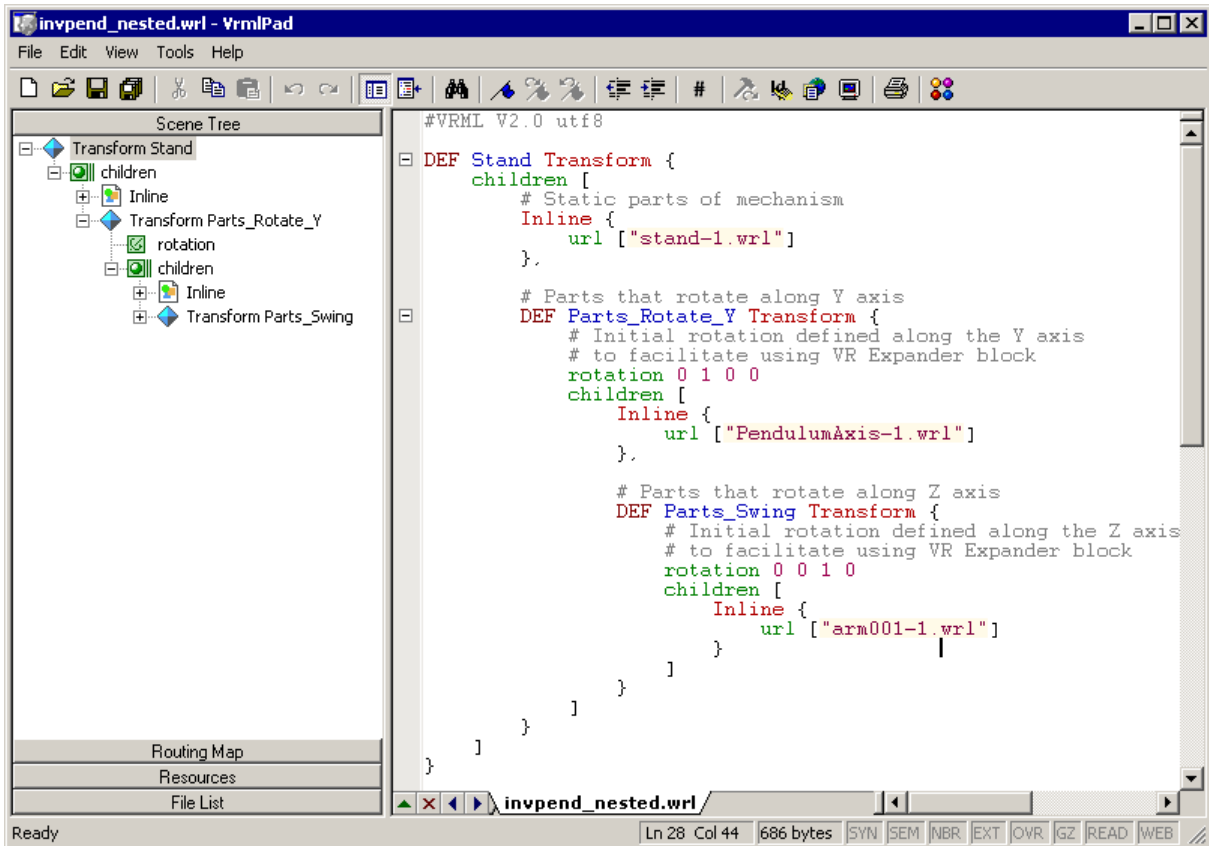
Some third-party tools allow you to export each part of the assembly into separate VRML files. All parts are then referenced in one main file using the VRML Inline mechanism. Referencing in this manner is the recommended way to work with assemblies, as the main file is small in size and easy to understand 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 Virtual Reality Toolbox (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 V-Realm Builder supplied with Virtual Reality Toolbox). Typically, adjusting exported files manually in a text editor requires the following changes:

- "Wrap shape objects with transforms" on page 5-33
- "Add DEF names" on page 5-33

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, 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 Virtual Reality Toolbox, 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 Virtual Reality Toolbox).

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 `vrtkoff.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 5-33.

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 Virtual Reality Toolbox allows for the visualization of models implemented in MATLAB. The following section disusses 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 5-33. 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 5-28, 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	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.
	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.
	To define a center of rotation different from the default value, `[0 0 0]`, define the **center** 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.

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 Simulink at the start of the simulation. You can compensate for this offset either in the VRML file (by defining an another level of nested `Transform` around the controlled object) or in the Simulink model by adding the object's initial position to the model calculations before sending to the VR Sink block.

You should align the Simulink model's initial conditions with the virtual 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 Virtual Reality Toolbox.

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

SimMechanics is very well suited for 3D visualizations using Virtual Reality Toolbox. Apart from features that SimMechanics 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 SimMechanics, 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.

SimMechanics also offers a convenient way of importing CAD assembly designs into SimMechnics 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, Virtual Reality Toolbox 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. Virtual Reality Toolbox includes its own viewer as the default for all supported platforms. It is the preferred method of viewing virtual worlds. For PC platforms, Virtual Reality Toolbox also includes a VRML plug-in (blaxxun Contact) that you can use as an alternative viewer for virtual worlds. A basic understanding of these tools and how to use them will help you to get started quickly.

Virtual Reality Toolbox Viewer

Section Overview

Virtual Reality Toolbox contains a viewer as the default method for viewing virtual worlds. You can use this viewer on any supported operating system. For a list of supported operating systems, see "System Requirements" on page 2-7. This section provides an overview of the features and controls of the viewer.

This section uses the `vrpend`, `vrplanets`, and `vrtut1` demos to illustrate the viewer features:

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

```
vrpend
```

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

2 Inspect the viewer window.

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

By default, the Virtual Reality Toolbox viewer displays the virtual scene with a navigation panel at the bottom.
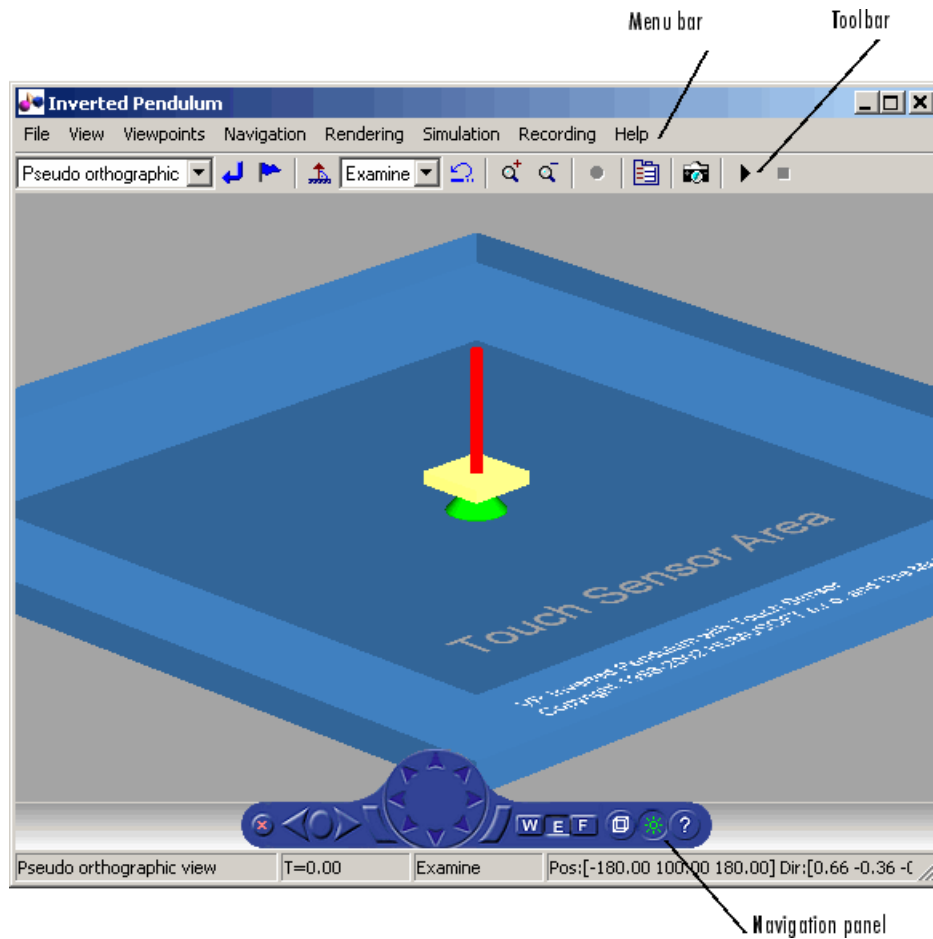
Note The Virtual Reality Toolbox viewer settings are saved when you save your model file.

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 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 q^+, q^-

- **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.

Operation	Menu Bar	Toolbar	Navigation Panel	Keyboard Shortcut
New Window	X			
Open in Editor	X			
Reload	X			
Save As	X			

Operation	Menu Bar	Toolbar	Navigation Panel	Keyboard Shortcut
Close	X			
Toolbar	X			
Status Bar	X			
Navigation Zones	X			**F7**
Navigation Panel	X		X	
Zoom In	X	X		**+**
Zoom Out	X	X		**-**
Normal (100%)	X			
Previous Viewpoint	X		X	**Page Up**
Next Viewpoint	X		X	**Page Down**
Return to Viewpoint	X	X	X	**Home**
Go to Default Viewpoint	X			
Create Viewpoint	X	X		
Remove Current Viewpoint	X			
Pseudo orthographic view	X			
Close View	X			
View from top	X			
X axis	X			
Z axis	X			
Method	X	X	X	**Shift+W**, **Shift+E**, **Shift+F**
Speed	X			

Operation	Menu Bar	Toolbar	Navigation Panel	Keyboard Shortcut
Straighten Up	X	X		**F9**
Undo Move	X	X		**Backspace**
Camera Bound to Viewpoint	X			**F10**
Antialiasing	X			**F8**
Headlight	X		X	**F6**
Lighting	X			
Textures	X			
Maximum Texture Size	X			
Transparency	X			
Wireframe	X		X	**F5**
Start (Simulation)	X	X		**Ctrl+T**
Stop (Simulation)	X	X		**Ctrl+T**
Block Parameters	X	X		
Start Recording	X	X		**Ctrl+R**
Stop Recording	X	X		**Ctrl+R**
Capture Frame	X	X		Ctrl+I
Capture and Recording Parameters	X	X		
Slide Left			X	
Navigation Wheel			X	
Slide Right			X	
Help	X		X	

Operation	Menu Bar	Toolbar	Navigation Panel	Keyboard Shortcut
Pan Left/Right				**Left/Right Arrows**, **Shift Left/Right Arrows**
Pan Up/Down				**Up/Down Arrows**
Move Forward/Backward				**Shift+Up/ Down Arrow**s
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.
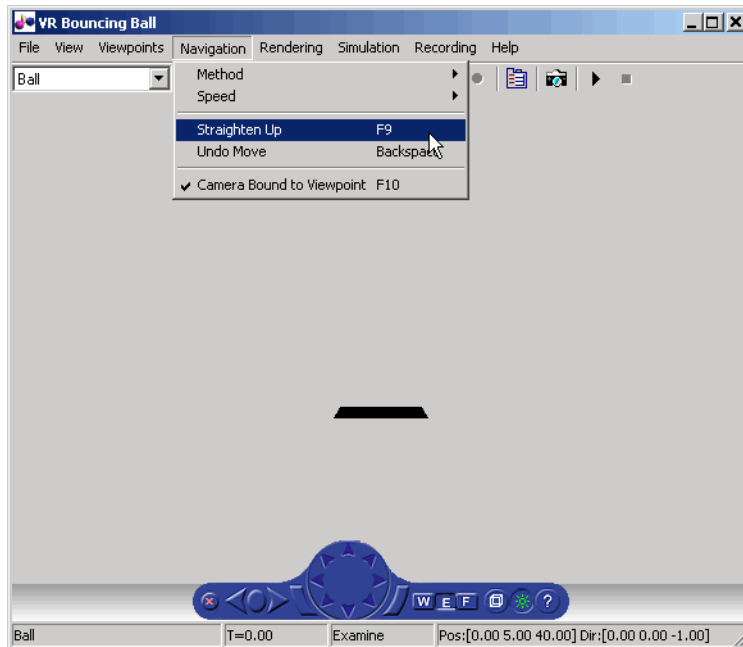
Navigation methods — Navigation with the mouse depends on the navigation method you select and the navigation zone you are in when you first click and hold down the mouse button. You can set the navigation method using one of the following:

- 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.
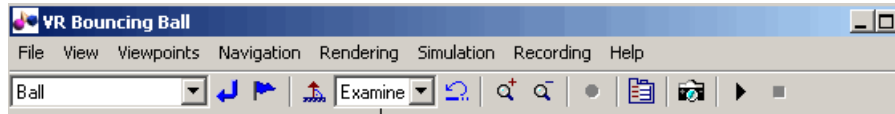
Navigation drop-down menu

- 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	**Outer** — Click and drag the mouse up, down, left, or right to slide the camera in any of these directions in a single plane.
	Inner — Click and drag the mouse up and down to move forward and backward. Drag the mouse left and right to turn left or right.

Virtual Reality Toolbox Viewer Mouse Navigation (Continued)

Movement Mode	Zone and Description
Examine	**Outer** — Click and drag the mouse up and down to move forward and backward. Drag the mouse left and right to slide left or right.
	Inner — Click and drag the mouse to rotate the viewpoint around the origin of the scene.
Fly	**Outer** — Click and drag the mouse to tilt the view either left or right.
	Inner — Click and drag the mouse to pan the camera up, down, left, or right within the scene.
	Center — 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.

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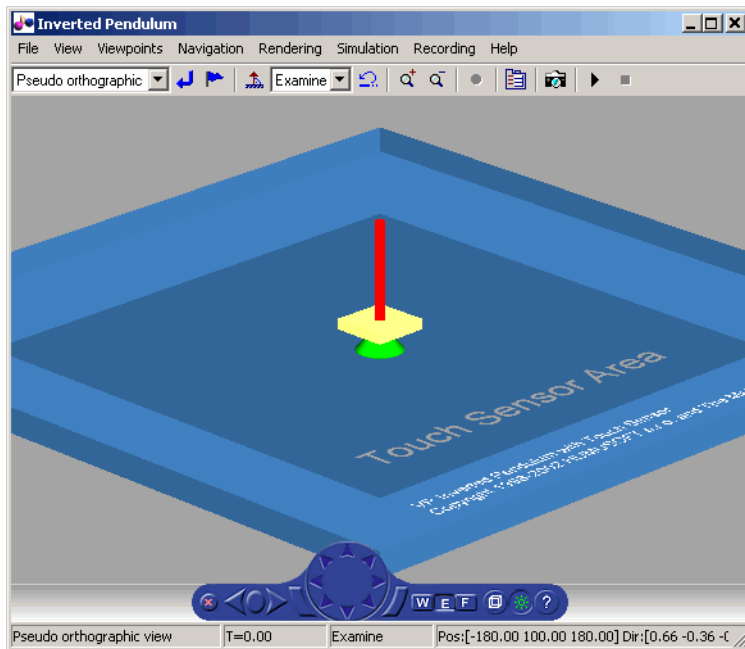
2 In the Simulink model window, from the **Simulation** menu, choose **Start**.

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

Keyboard Command	Navigation Function
Backspace	Undo move.
Ctrl+R	Start/stop recording.
Ctrl+I	Capture frame.
Ctrl+T	Start/stop simulation.
F9	Straighten up and make the camera stand on the horizontal plane of its local coordinates.
+/-	Zoom in/out.
F6	Toggle the headlight on/off.
F7	Toggle the navigation zones on/off.
F5	Toggle the wireframe option on/off.
F8	Toggle the antialiasing option on/off.
Esc	Go to default viewpoint.
Home	Return to current viewpoint.
Page Up, **Page Down**	Move between preset viewpoints.
F10	Camera is bound/unbound from the viewpoint.
Shift+W	Set the navigation method to Walk.
Shift+E	Set the navigation method to Examine.
Shift+F	Set the navigation method to Fly.
Shift Up/Down Arrow	Move the camera forward and backward.
Up/Down Arrow	Pan the camera up and down.

Virtual Reality Toolbox Viewer Keyboard Navigation (Continued)

Keyboard Command	Navigation Function
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 **Ctrl** 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 **Ctrl** 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. Virtual Reality Toolbox 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.wrl
```

If you record the simulation a second time, the animation filename is `vrplanets_anim_2.wrl`. 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 (`.wrl` and `.avi`) files.

Token	Description
%d	The full path to the world VRML file replaces this token in the filename string and creates files in directories relative to the virtual world file location. For example, the format `%d/animdir/%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.wrl` saves the animation to `vrplanets_anim_1.wrl` on the first run, `vrplanets_anim_2.wrl` on the second run, and so forth.
%Y	The current four-digit year replaces this token in the filename string. For example, the format `%f_anim_%Y.wrl` saves the animation to `vrplanets_anim_2005.wrl` 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

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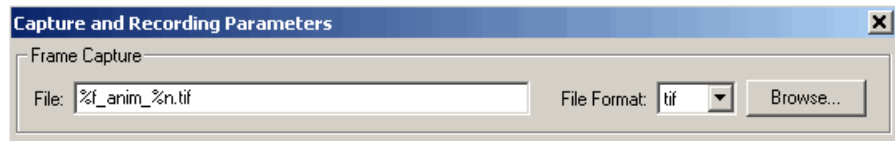
6 You can disable the navigation panel. The navigation panel appears at the bottom of the virtual scene view. You might want to turn off this panel for a cleaner view of the virtual scene. Choose **View > Navigation Panel > Off**.

You can reenable 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

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

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

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

- 2-D Audio Video Interleave (AVI) file — Virtual Reality Toolbox writes animation data in an `.avi` file. Virtual Reality Toolbox uses `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.
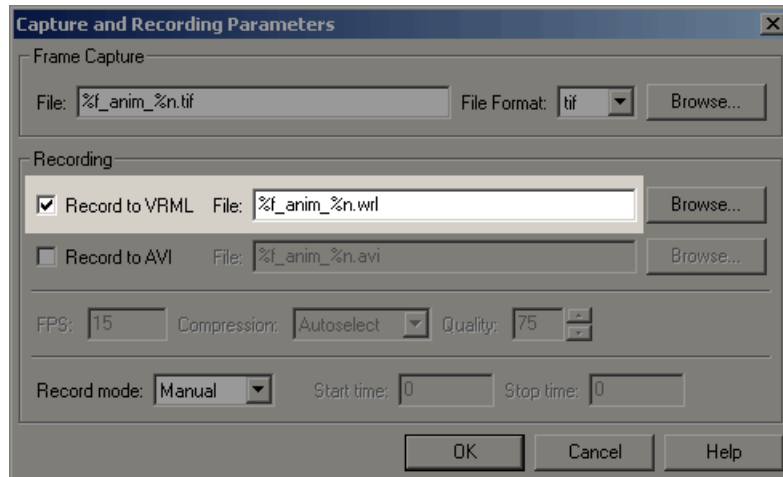
5 Click **OK**.

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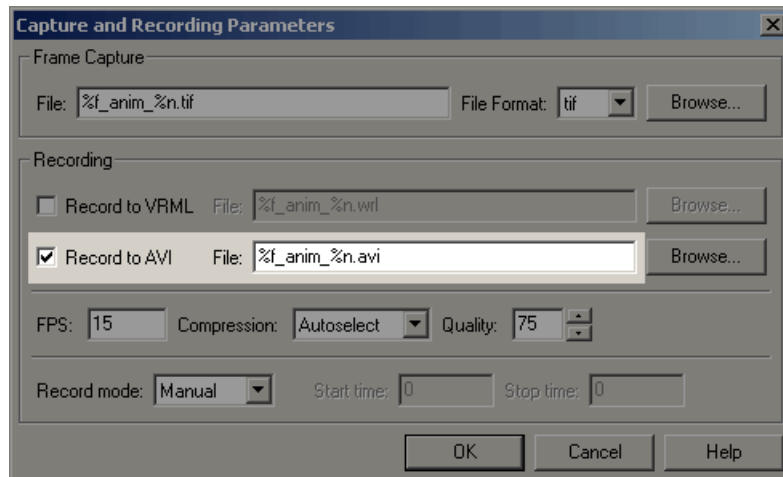
5 Set the **FPS** to an appropriate value.

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.
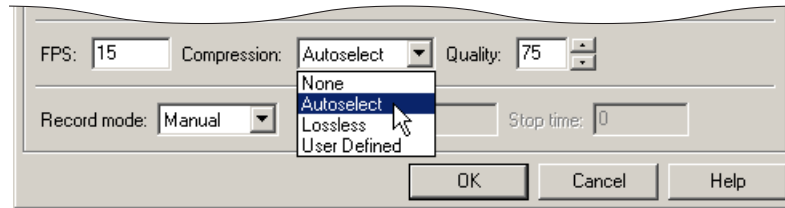
Choose from

- `Autoselect` — Allows Virtual Reality Toolbox 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 Virtual Reality Toolbox to compress the animation file without loss of data. (Typically, the compression of files sacrifices some data.)

- `User Defined` — Enables you to specify a particular compression codec. 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.
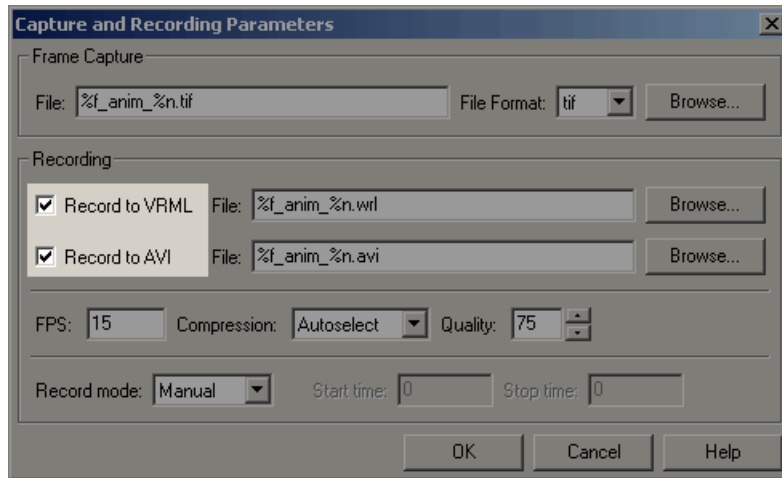
3 From the **Record mode** list, choose Scheduled.
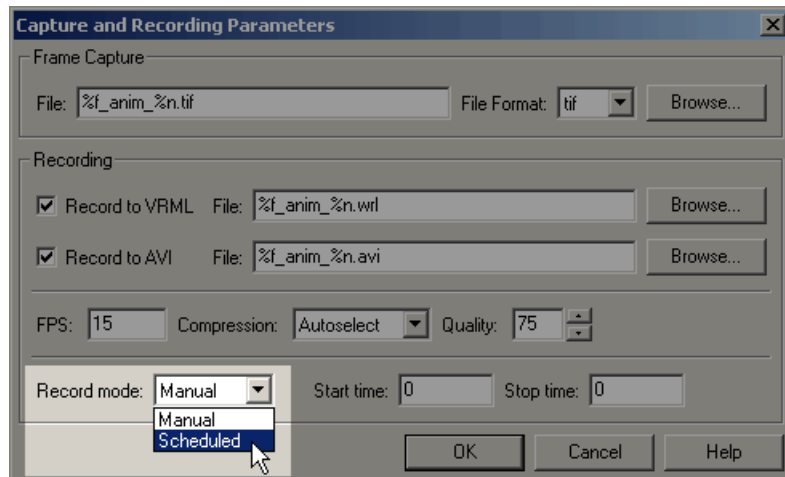
The **Start time** and **Stop time** text fields are enabled.

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.

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 MATLAB.

3 To stop the animation, type

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

Alternatively, to close the viewer and delete the world, you can get the handle of the `vrfigure` object and close it, as follows:

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

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

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

To View AVI Files

1 Change directory to the one that contains the AVI animation file.

2 Double-click that file.

The program associated with `.avi` files in your system (for example, Windows Media Player) opens for the `.avi` file. If your `.avi` file is not yet running, start it now from the application. The animation file runs.

Working with Viewpoints

You or visitors to a virtual world navigate through the virtual world environment using the Virtual Reality Toolbox viewer navigation methods Walk, Examine, and Fly. In addition to these navigation methods, a virtual world creator can set up points of interest, known as viewpoints, in the virtual world. You can use viewpoints to guide visitors through the virtual world and to emphasize important points.

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

Each virtual world has as many viewpoints as you define for it. You can define viewpoints in the virtual world through your chosen editor or through the Virtual Reality Toolbox viewer.

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

- Static -- Created typically at the top level of the virtual world object hierarchy. You can also create these viewpoints as children of static objects (`Transforms`).
- Dynamic -- Created as children of moving objects (objects controlled from MATLAB or Simulink) or linked to moving objects with the VRML `ROUTE` mechanism. Dynamic viewpoints allow you to create interesting views such as from the driver's seat at a racing course.

This topic illustrates viewpoints using the `vrplanets` demo.

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

```
vrplanets
```

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

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

A menu of the viewpoint options is displayed. Included is a list of the existing viewpoints.
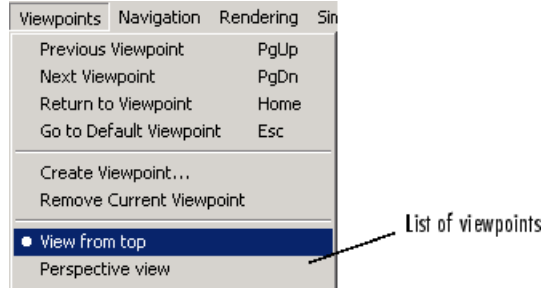
List of viewpoints

3 Select the drop-down list on the leftmost side of the toolbar to see the list of existing viewpoints from the toolbar.

List of viewpoints
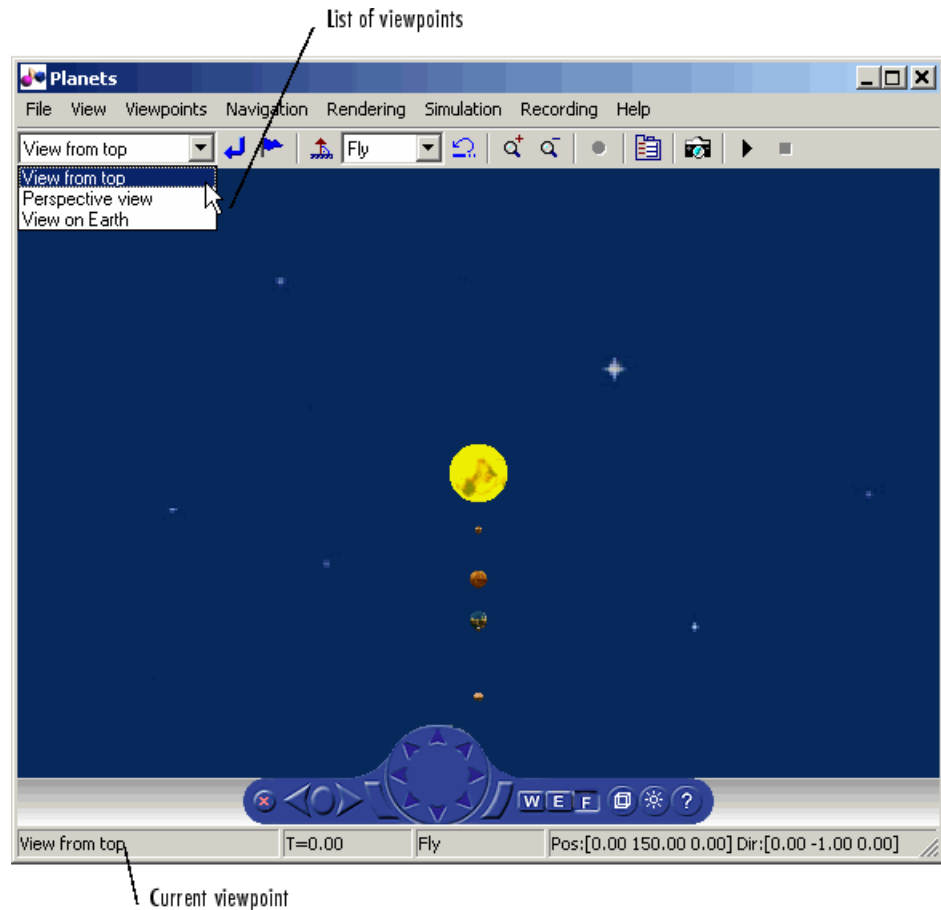
Current viewpoint

When you add new viewpoints to the world, these lists are updated to reflect the new viewpoints.

The current viewpoint is also displayed in the left pane of the status bar.

You manage and navigate through viewpoints from the menu bar, toolbar, navigation panel, and keyboard. In particular, you can

• Navigate to a previous or next viewpoint

• Return to the initial position of the current viewpoint

- Go to the virtual world's default viewpoint

- Create and remove viewpoints

- Navigate to an existing viewpoint

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.
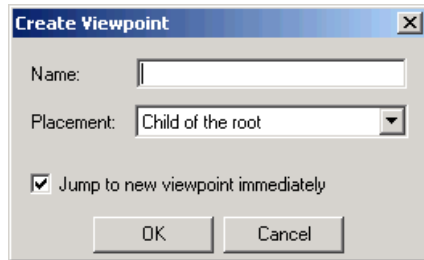
6 In the **Name** box, enter a unique and descriptive name for the viewpoint.

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.
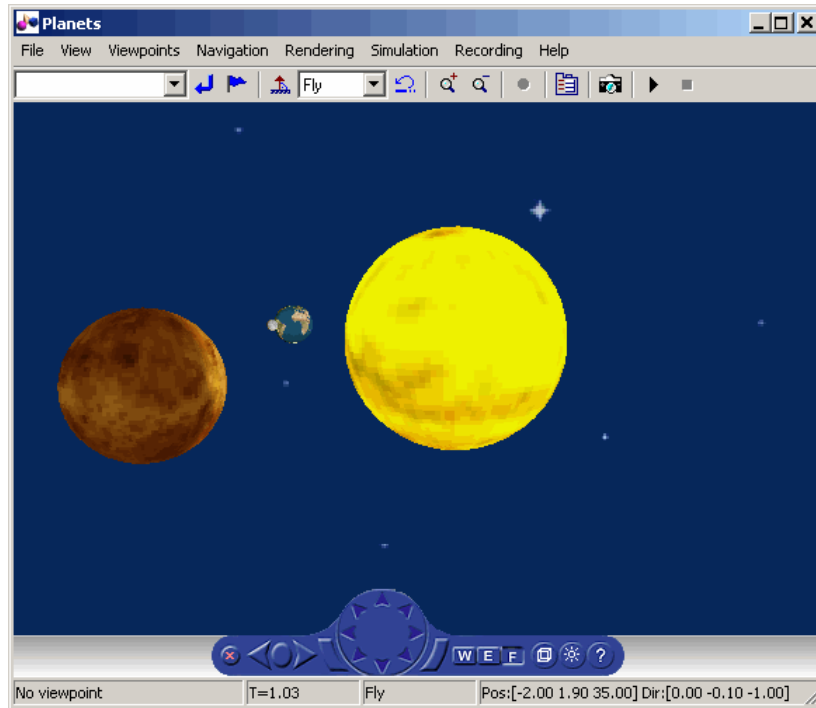
Rendering

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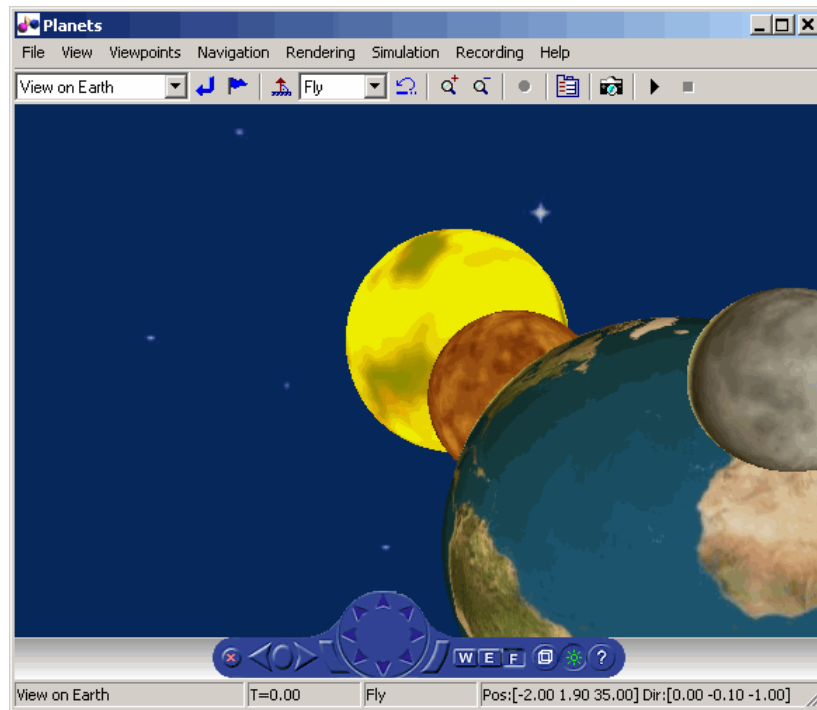
You can turn the camera headlight and the lighting of the scene on or off. When **Headlight** is off, the camera does not emit light. Consequently, the scene can appear dark. For example, the following figure depicts the vrpend demo with **Headlight** on.

Headlight toggle
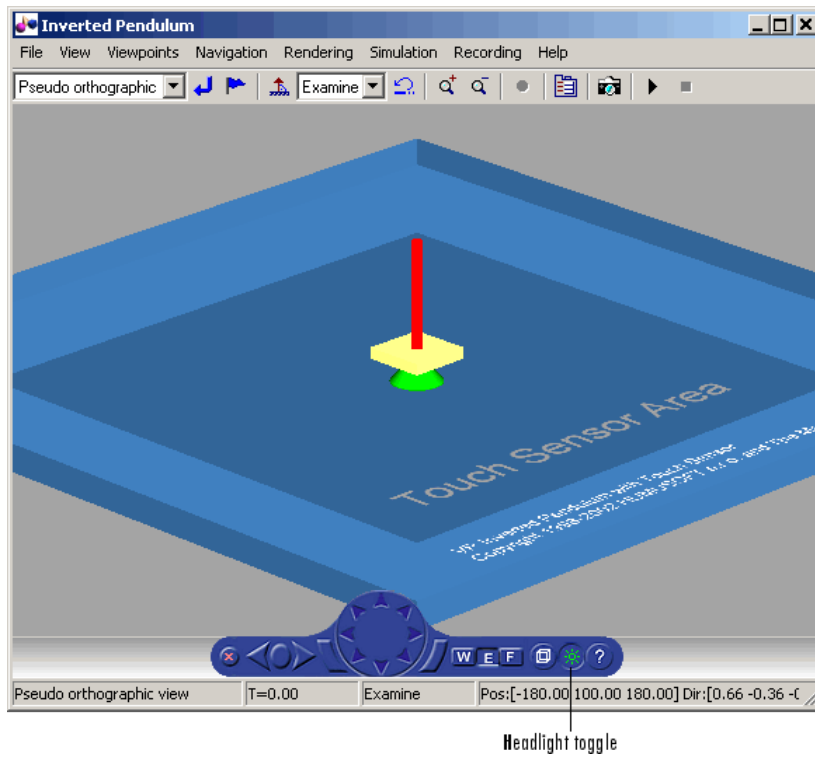
The scene looks darker when **Headlight** is set to off.
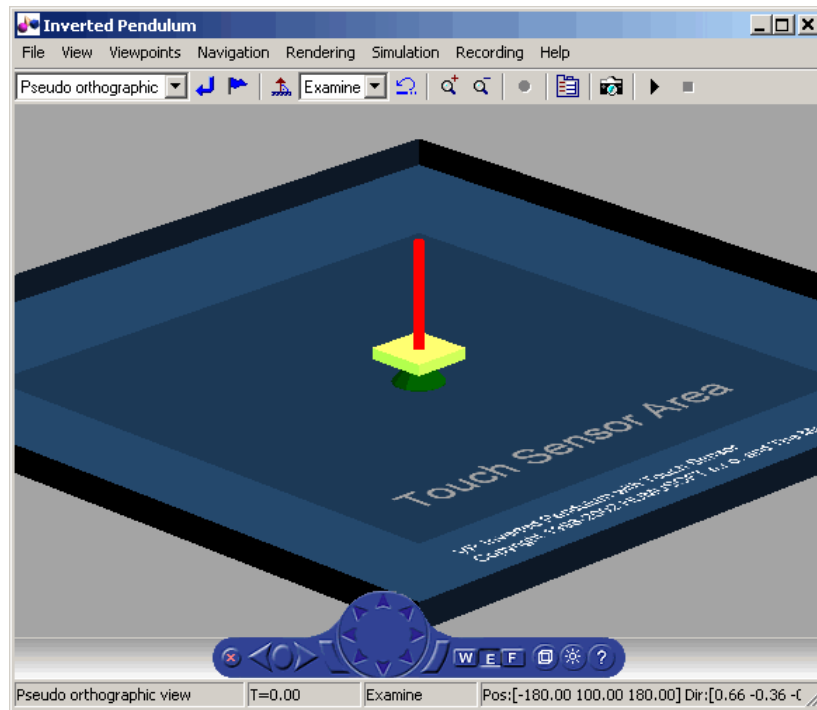
Note It is helpful to define enough lighting within the virtual scene so that it is lit regardless of the **Headlight** setting.

When **Lighting** is off, the virtual world appears as if lit in all directions. The Virtual Reality Toolbox viewer does not compute and render all the lighting effects at the surfaces of the objects. Shadows disappear and the scene loses some of its 3-D quality. The following is the vrpend demo with **Lighting** off.
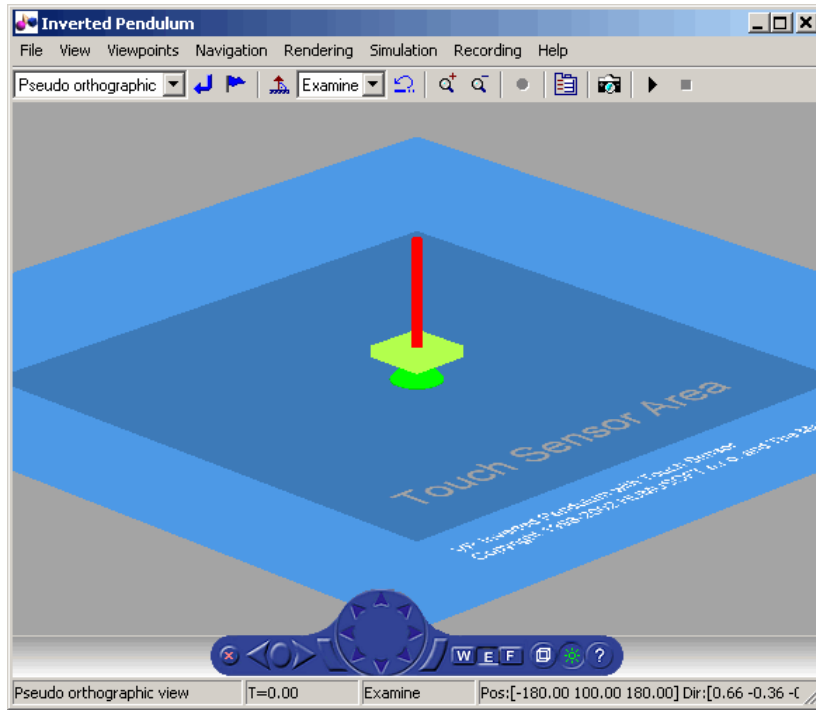
If **Transparency** is off, transparent objects are rendered as solid objects.
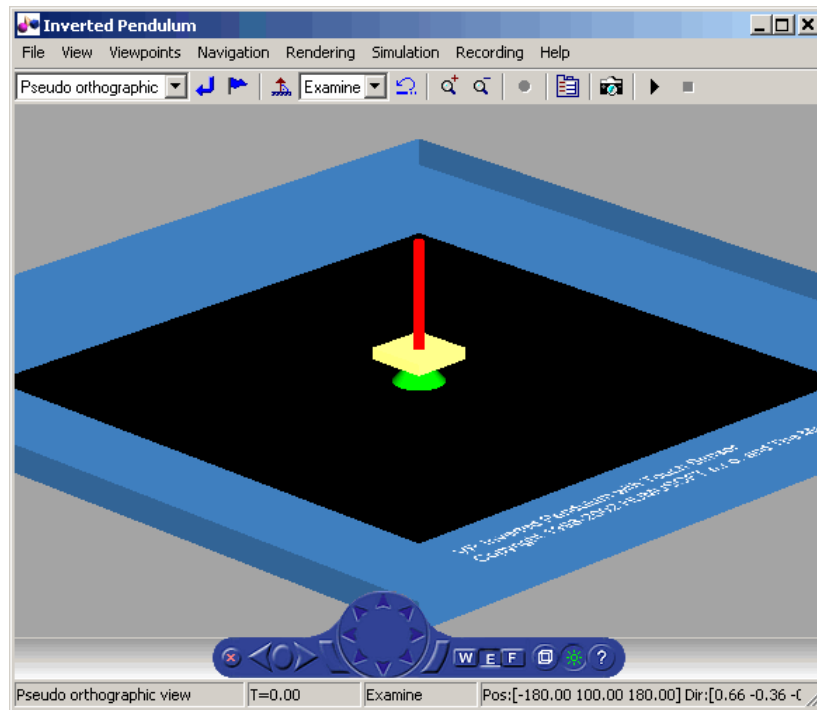
Turning **Wireframe** on changes the scene's objects from solid to wireframe
rendering. The following is the vrpend demo with **Wireframe** on.

Wireframe toggle
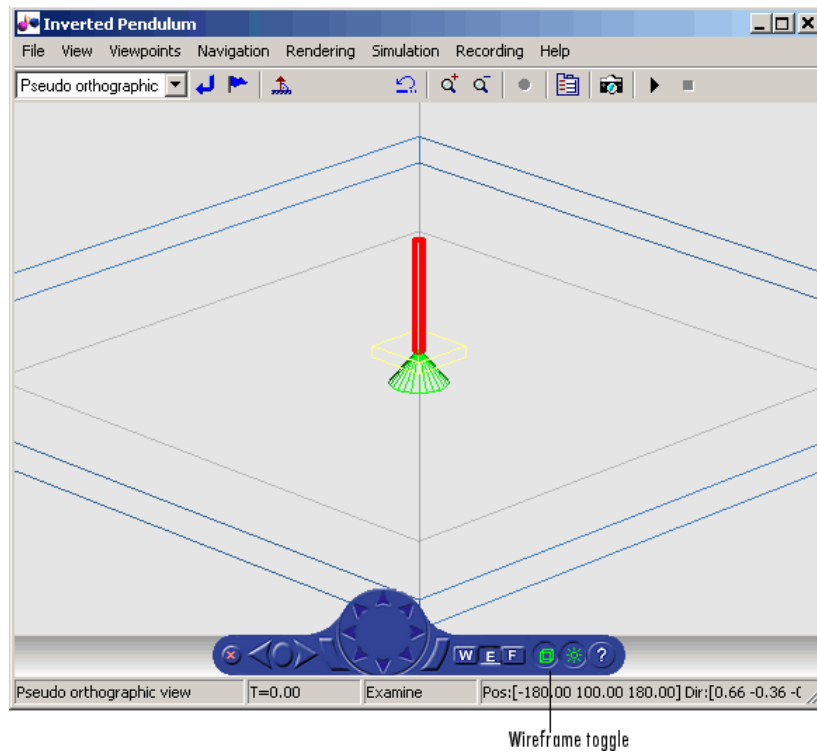
If **Textures** is on, objects have texture in the virtual scene. The following is the vrplanets demo with **Textures** on.
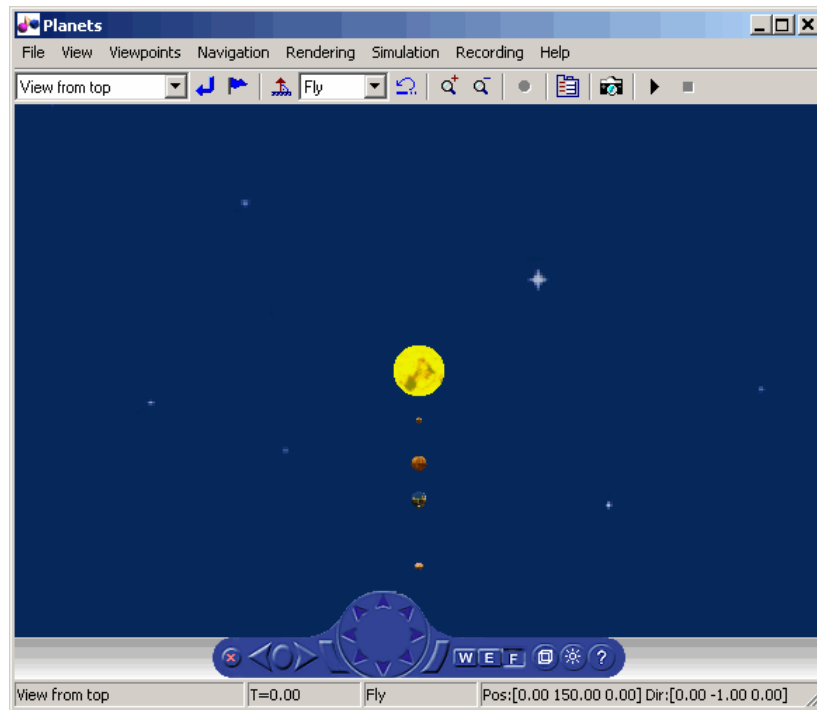
If **Textures** is off, objects do not have texture in the virtual scene. The following is the vrplanets demo with **Textures** off.
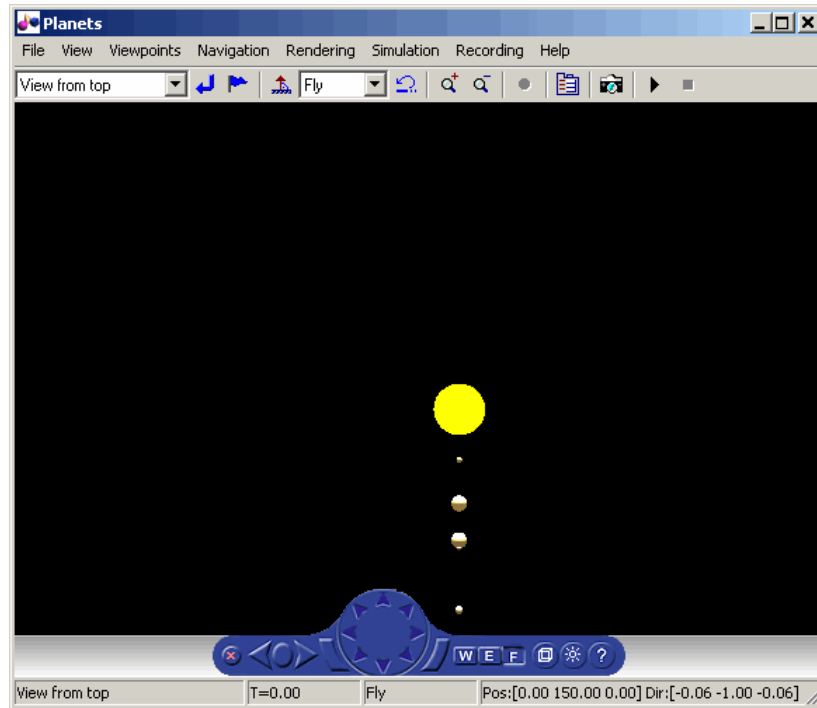
You can specify the maximum size of a texture used in rendering the vrfigure object. This option gives you a list of texture sizes to choose from. See the vrfigure MaxTextureSize property for further details.

blaxxun Contact VRML Plug-In

Section Overview

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

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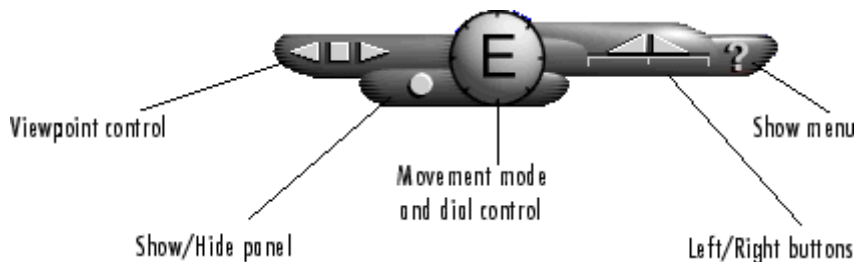
Viewpoint Control

Three buttons on the control panel control the viewpoint. The square button in the middle resets the current viewpoint to its initial position. This is the

most useful viewpoint control button until you gain enough experience with
the viewer to explore worlds using navigation. The keyboard shortcut for the
square button is the **Esc** key.

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

Control Menu

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

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

If you selected Direct3D with the blaxxun Contact installation, a menu
similar to the following appears:
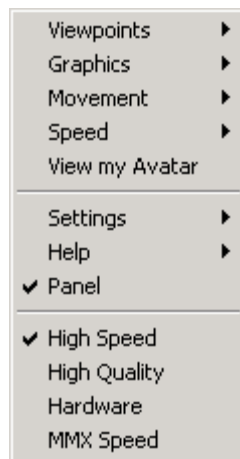
2 From the menu, you can make changes to the navigational mode, graphic
quality, and graphic speed.

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.

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

blaxxun Contact Settings

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

- The speed of your hardware

- System display driver settings

- Method of 3-D rendering

- blaxxun Contact parameters

• The size of the window in which you display the 3-D visualization

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

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

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

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

Stereoscopic Vision

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

• To switch blaxxun Contact to the full screen mode, press **F5**.

• To switch back to normal mode, press **Esc**.

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

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

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

A menu appears.

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

3 Choose **Preferences**.

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

5 Click **OK** when done.

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

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 MATLAB or Virtual Reality Toolbox.

What Is Orbisnap?

Virtual Reality Toolbox includes Orbisnap. Orbisnap is a free, optional, stand-alone VRML97 viewer that does not require you to have either MATLAB or Virtual Reality Toolbox 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 MATLAB or Virtual Reality Toolbox.

- Remotely view, from a client machine, a virtual world loaded in a current session of Virtual Reality Toolbox. 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 Virtual Reality Toolbox 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 MATLAB and Virtual Reality Toolbox. 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

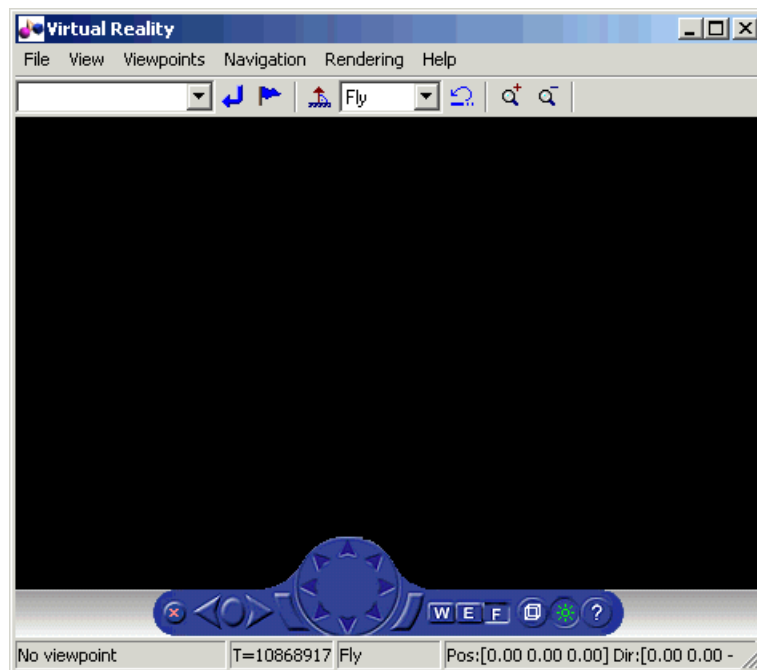| **In this section...** |
|---|
| "Overview" on page 7-5 |
| "Viewing Prerecorded WRL Animations or Virtual Worlds" on page 7-6 |
| "Viewing the Virtual Reality Toolbox Server Virtual Worlds Remotely" on page 7-7 |

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

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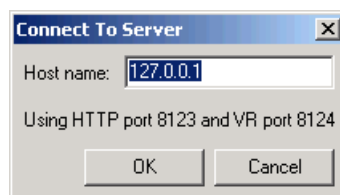
   The Choose a world dialog is displayed. This dialog lists all the virtual worlds that are currently active on the Virtual Reality Toolbox server.

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.
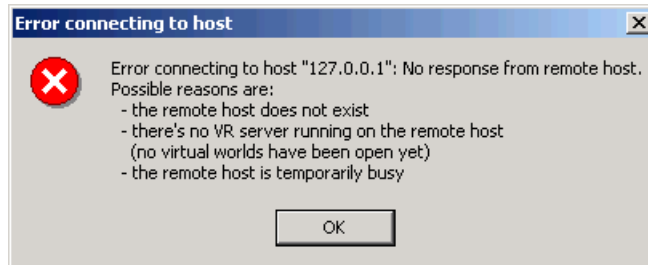
4 Select a virtual world.

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

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`

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 7-12 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	**Outer** -- Click and drag the mouse up, down, left, or right to slide the camera in any of these directions in a single plane.
	Inner -- Click and drag the mouse up and down to move forward and backward. Drag the mouse left and right to turn left or right.
Examine	**Outer** -- Click and drag the mouse up and down to move forward and backward. Drag the mouse left and right to slide left or right.
	Inner -- Click and drag the mouse to rotate the viewpoint around the origin of the scene.
Fly	**Outer** -- Click and drag the mouse to tilt the view either left or right.
	Inner -- Click and drag the mouse to pan the camera up, down, left, or right within the scene.
	Center -- 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.

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

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

Orbisnap Keyboard Navigation (Continued)

Keyboard Command	Navigation Function
Alt+Left/Right Arrow	Slide left and right.
Ctrl+Left/Right/Up/ Down Arrow	Pressing **Ctrl** 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 **Ctrl** rotates the viewpoint about the acquired center point.
Shift+Alt+Left/Right Arrow	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.wrl`). 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.

8

Blocks — By Category

Control Input Devices (p. 8-1)   Process input from devices

Utilities (p. 8-1)   Vector and matrix calculations

Virtual Worlds (p. 8-2)   Virtual World utilities

VRML-Related Signals (p. 8-2)   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)

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

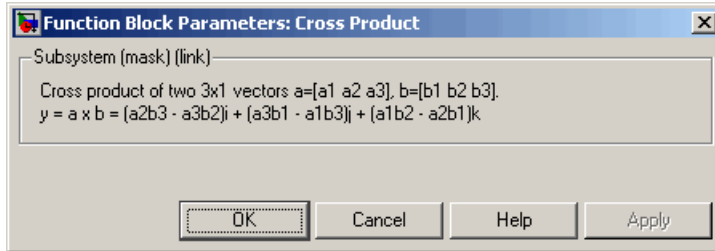**Library**   Virtual Reality Toolbox

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

Block Parameters Dialog Box

Cross Product

Function Block Parameters: Cross Product

Subsystem (mask) (link)

Cross product of two 3x1 vectors a=[a1 a2 a3], b=[b1 b2 b3].
$y = a \times b = (a2b3 - a3b2)i + (a3b1 - a1b3)j + (a1b2 - a2b1)k$

OK   Cancel   Help   Apply

Purpose   Process input from asynchronous joystick device

Library   Virtual Reality Toolbox

Description

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

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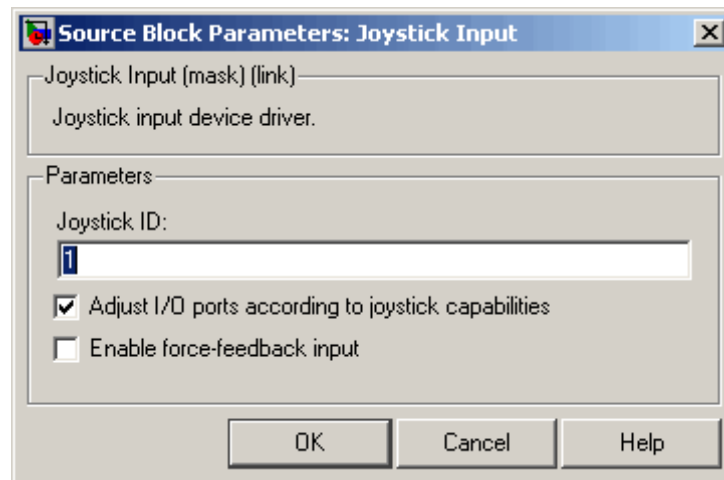
Joystick ID — The system ID assigned to the given joystick device. You can find the properties of the joystick connected to the system in the Game Controllers section of the system Control Panel.

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, Virtual Reality Toolbox dynamically adjusts the ports to correspond to the capabilities of the connected joystick each time the model is opened. If the connected device does not have force-feedback capability, selecting this check box causes the removal of the force-feedback input from the block, even if the **Enable force-feedback input** check box is selected.

Enable force-feedback input — If you select this check box, Virtual Reality Toolbox can support force-feedback joystick, steering wheel, and haptic (one that enables tactile feedback) devices. To use this feature, you must install Microsoft DirectX Version 8.0 or 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 >	Port active only for force-feedback devices. Inputs correspond to the desired force to be applied in the given axis. Usually not all of the device axes have force-feedback. The size of the Force vector is then smaller than the Axes vector size.

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.
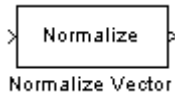
Block Parameters Dialog Box

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

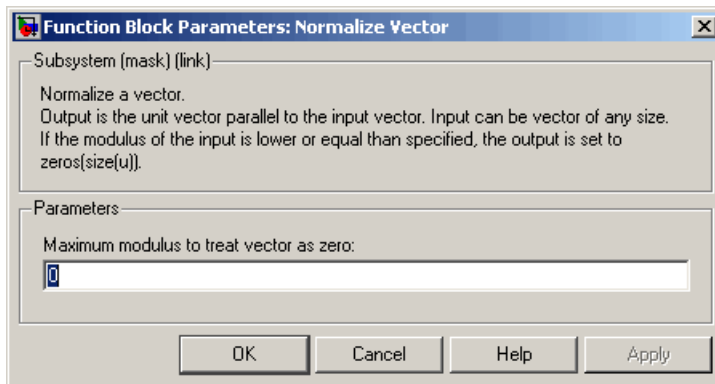**Description**   Takes input of two 3-by-1 vectors and returns a VRML rotation
(specified as a four-element vector defining axis and angle) that is
needed to transform the first input vector to the second input vector.
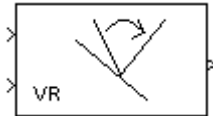
Rotation Between
2 Vectors

Block Parameters Dialog Box

Function Block Parameters: Rotation Between 2 Vectors

Subsystem (mask) (link)

Calculates a rotation between two vectors.
Output is a VRML rotation (specified as 4-element vector defining axis and angle)
needed to transform the first input to the second input. Both inputs must be 3x1
vectors.

OK   Cancel   Help   Apply
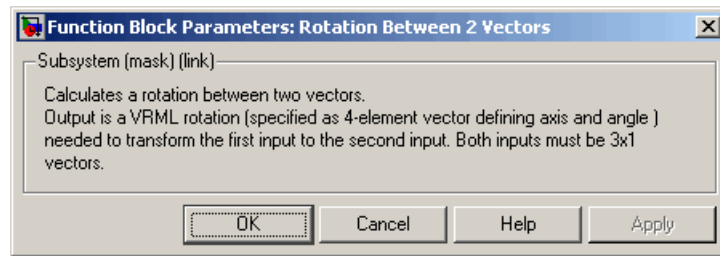
Rotation Matrix to VRML Rotation

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.

Rotation Matrix
to VRML Rotation

Block Parameters Dialog Box
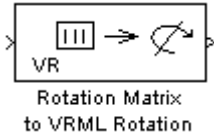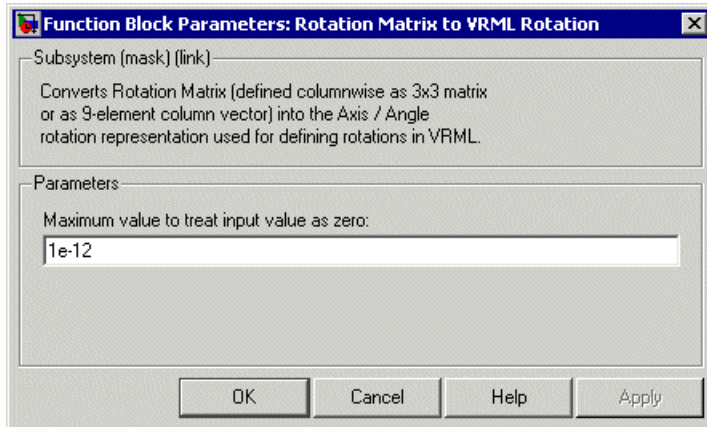

Maximum value to treat input value as zero — The input is considered to be zero if it is equal to or lower than this value.

Rotation Matrix

A representation of a three-dimensional spherical rotation as a 3-by-3 real, orthogonal matrix R: $R^{\mathrm{T}}R = RR^{\mathrm{T}} = I$, 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 θ with $n*n = 1$. The three independent angles are θ 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
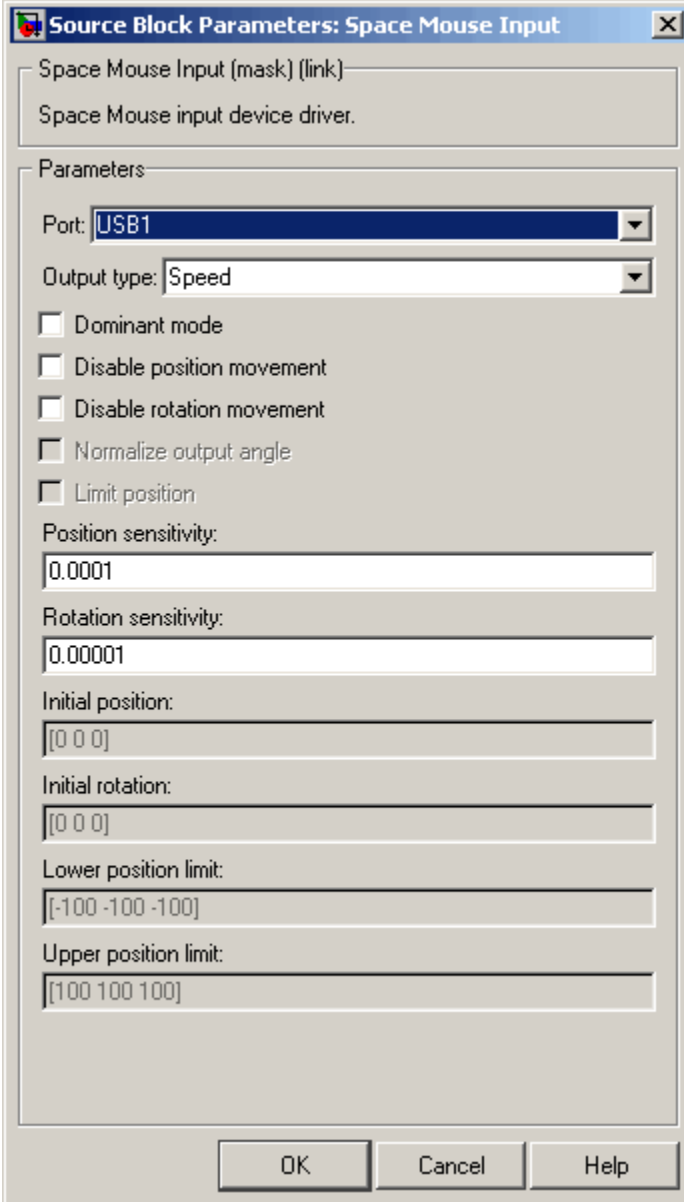
Library　　　　Virtual Reality Toolbox

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 all models of space mouse and PuckMan devices manufactured by 3Dconnexion (`http://www.3dconnexion.com/products.htm`). It also supports USB devices.
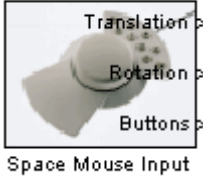
Space Mouse Input

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

**Block
Parameters
Dialog
Box**

Source Block Parameters: Space Mouse Input

Space Mouse Input (mask) (link)

Space Mouse input device driver.

Parameters

Port: USB1

Output type: Speed

☐ 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]

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.

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).
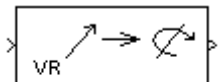
Block Parameters Dialog Box
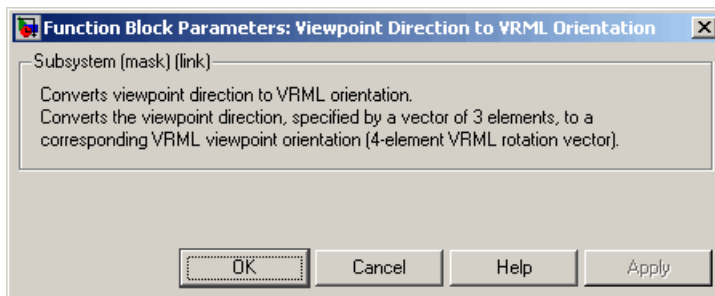
Purpose   Send unspecified value to Virtual Reality Toolbox block

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
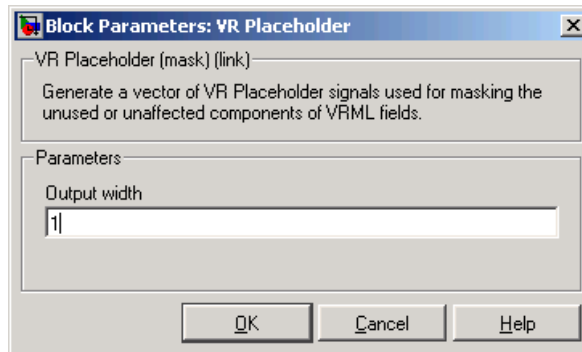
Output Width — Length of the vector containing placeholder signal values.

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
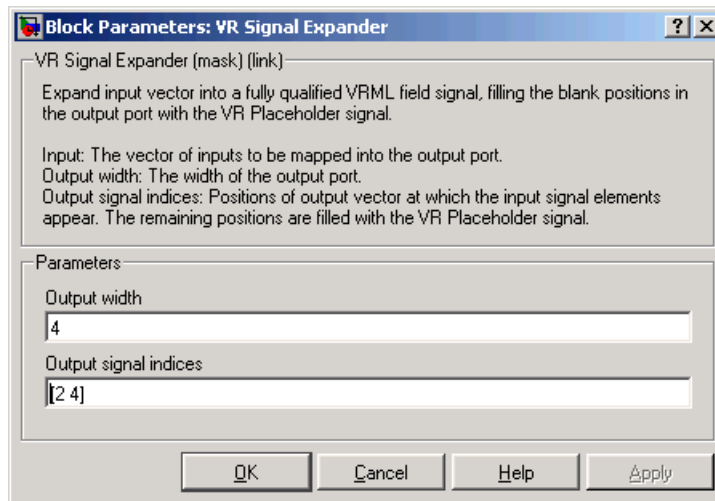
Output width — How long the output vector should be.

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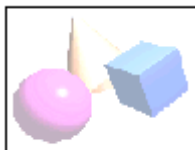
The VR Sink block cannot be compiled by Real-Time Workshop, but it can be used as a SimViewing device on the host computer.

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, Simulink 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. For further details, see "VRML Field Data Types" on page 5-21.

Block Parameters Dialog Box
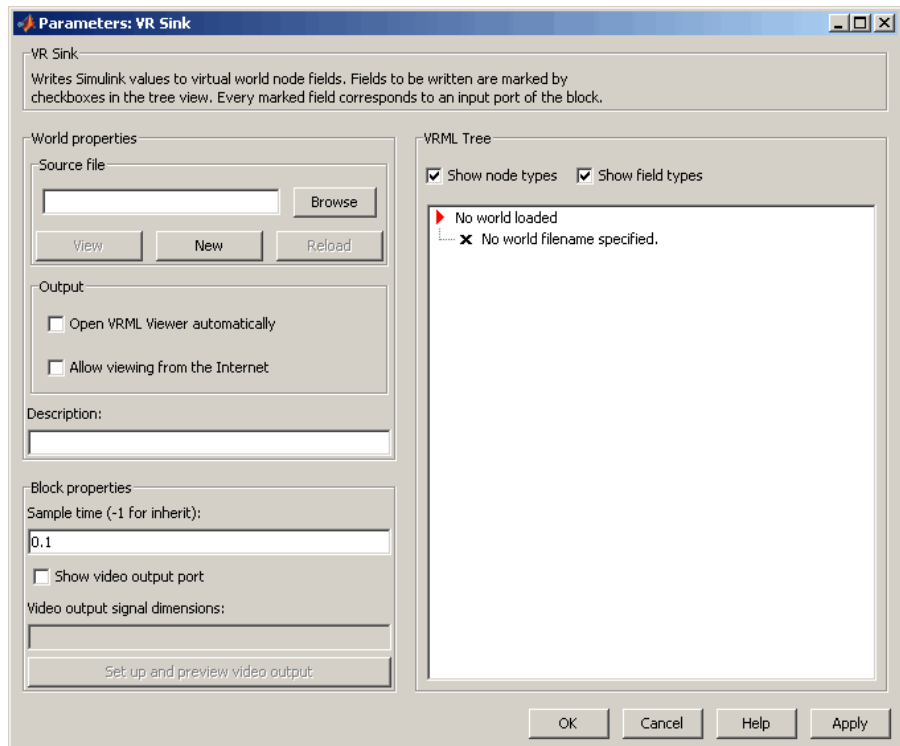
Source file — VRML file name specifying the virtual world this block is connected to. The **View** button allows you to view the world in the Virtual Reality Toolbox viewer or a Web browser. The **Edit** button launches an external VRML editor, and the **Reload** button reloads the world after you change 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, Virtual Reality Toolbox assumes that the .wrl file resides in the same directory as the model file.

Open VRML Viewer automatically — If you select this check box, the default VRML viewer displays the virtual world after loading the Simulink model.

Allow viewing from the Internet — If you select this check box, the virtual world is accessible for viewing on a client computer. If it is not selected, the world is visible only on the host computer. This is equivalent to the `RemoteView` property of a `vrworld` object. See Chapter 4, "MATLAB Interface".

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

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

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

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

Fields with settable values have check boxes. Use these check boxes to select the fields you want Simulink to output values to. For every selected field, an input port is created in the block. Input ports are

assigned to the selected nodes and fields in the order corresponding to the VRML file.

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

Show node types — If you select this check box, node types are shown in the VRML tree.

Show field types — If you select this check box, field types are shown in the VRML tree.

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

Property	Possible Values
FieldsWritten	String containing a list of `NodeName.FieldName` pairs, separated by # For example, `Membrane.translation#Membrane.rotation`
FigureProperties	Array of MATLAB figure properties associated with the VRML viewer.
SampleTime	String containing an expression that evaluates to a valid Simulink sample time value
WorldFileName	Associated VRML filename

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

VR 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.
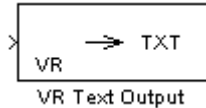
VR Text Output

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**
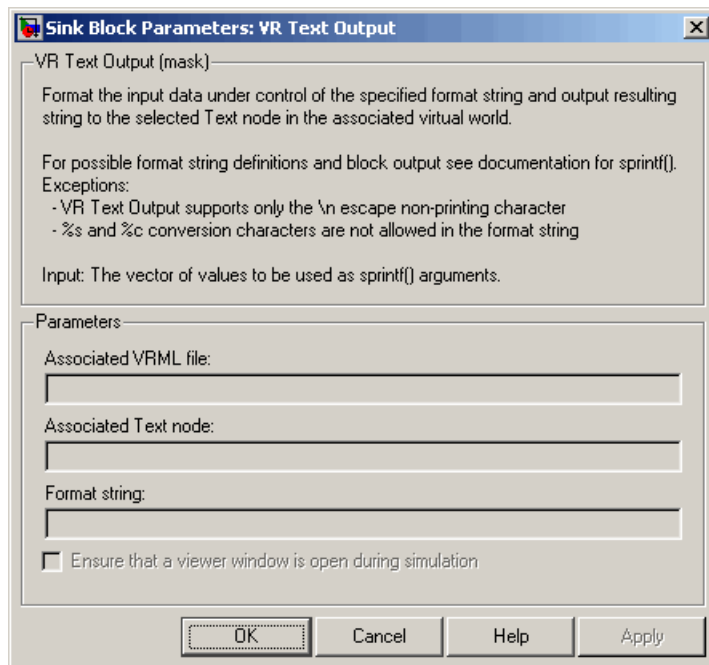
Associated VRML file — VRML file specifying the virtual world to which text is output.

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 Simulink 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.

VR To Video

See the VR Sink block for details.

10
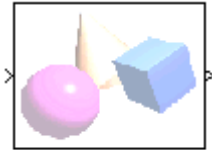
Functions — By Category

MATLAB Interface Functions

vrclear   Remove all closed virtual worlds
                                 from memory

vrclose   Close virtual reality figure windows

vrdir2ori   Convert viewpoint direction to
                                 orientation

vrdrawnow   Update virtual world

vrgetpref   Values of Virtual Reality Toolbox
                                 preferences

vrinstall   Install and check Virtual Reality
                                 Toolbox components

vrjoystick   Create joystick object

vrlib   Open Simulink block library for
                                 Virtual Reality Toolbox

vrori2dir	Convert viewpoint orientation to direction
vrplay	Play VRML animation file
vrrotmat2vec	Convert rotation from matrix to axis-angle representation
vrrotvec2mat	Convert rotation from axis-angle to matrix representation
vrrotvec	Calculate a rotation between two vectors
vrsetpref	Change Virtual Reality Toolbox preferences
vrspacemouse	Create space mouse object
vrview	View virtual world using Virtual Reality Toolbox viewer or Web browser
vrwho	List virtual worlds in memory
vrwhos	List details about virtual worlds in memory

vrworld Object Methods

vrworld	Create new vrworld object associated with virtual world
vrworld/close	Close virtual world
vrworld/delete	Remove virtual world from memory
vrworld/edit	Open virtual world file in external VRML editor
vrworld/get	Property value of vrworld object
vrworld/isvalid	1 if vrworld object is valid, 0 if not
vrworld/nodes	List nodes available in virtual world

`vrworld/open`	Open virtual world
`vrworld/reload`	Reload virtual world from VRML file
`vrworld/save`	Write virtual world to VRML file
`vrworld/set`	Change property values of `vrworld` object
`vrworld/view`	View virtual world

vrnode Object Methods

`vrnode`	Create node or handle to existing node
`vrnode/delete`	Remove `vrnode` object
`vrnode/fields`	VRML field summary of node object
`vrnode/get`	Property value of `vrnode` object
`vrnode/getfield`	Field value of `vrnode` object
`vrnode/isvalid`	1 if `vrnode` object is valid, 0 if not
`vrnode/set`	Change property of virtual world node
`vrnode/setfield`	Change field value of `vrnode` object
`vrnode/sync`	Enable or disable synchronization of VRML fields with client

vrfigure Object Methods

`vrfigure`	Create new virtual reality figure
`vrfigure/capture`	Create RGB image from virtual reality figure

vrfigure/close	Close virtual reality figure
vrfigure/get	Property value of vrfigure object
vrfigure/isvalid	1 if vrfigure object is valid, 0 if not
vrfigure/set	Change property value of vrfigure object
vrfigure/vrgcbf	Current callback vrfigure object
vrfigure/vrgcf	Handle for active virtual reality figure

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 Simulink. You use this command to

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

               • Perform a general cleanup of memory.

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

See Also   vrworld, vrworld/delete

Purpose   Close virtual reality figure windows

Syntax
```
vrclose
vrclose all
```

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

Examples   Open a series of virtual reality figure windows by typing

```
vrpend
vrbounce
vrlights
```

Arrange the viewer windows so they are all visible. Type

```
vrclose
```

All the virtual reality figure windows disappear from the screen.

See Also   `vrfigure/close`

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

- MATLAB 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`.

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.

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 <= x <= video card limit, where x is a power of 2 (video card limit is typically 1024 or 2048)	Sets the maximum pixel size of a texture used in rendering vrfigure objects. The smaller the size, the faster the texture can render. Increasing this value improves image quality but decreases performance. A value of `'auto'` sets the maximum possible pixel size. If the value you enter is unsuitable, a warning might trigger. Virtual Reality Toolbox then automatically adjusts the property to the next smaller suitable value.
Name	String	Specifies the name of this vrfigure object. Read/write.
NavMode	`'fly'` | `'examine'` | `'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.

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 4-10. 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.wrl');
```

The vrworld object myworld is associated with the virtual world vrmount.wrl. Open the world:

```
open(myworld)
```

Create a vrfigure object:

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

You can now get the object properties of the vrfigure object f:

```
get(f)
```

This returns the following object properties:

```
AntiAliasing = 'off'
CameraBound = 'on'
CameraDirection = [0 0 -1]
CameraDirectionAbs = [0 -0.198669 -0.980067]
CameraPosition = [0 0 0]
CameraPositionAbs = [20 8 50]
CameraUpVector = [0 1 0]
CameraUpVectorAbs = [0 0.980067 -0.198669]
CaptureFileFormat = 'tif'
CaptureFileName = '%f_anim_%n.tif'
DeleteFcn = ''
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

Purpose   1 if vrfigure object is valid, 0 if not

Syntax   x = isvalid(vrfigure_object_vector)

Arguments   vrfigure_object_vector Name of an array of vrfigure objects.

Description 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.

See Also   vrnode/isvalid, vrworld/isvalid

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.

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 <= x <= video card limit, where x is a power of 2 (video card limit is typically 1024 or 2048)	Sets the maximum pixel size of a texture used in rendering `vrfigure` objects. The smaller the size, the faster the texture can render. Increasing this value improves image quality but decreases performance. A value of `'auto'` sets the maximum possible pixel size. If the value you enter is unsuitable, a warning might trigger. Virtual Reality Toolbox then automatically adjusts the property to the next smaller suitable value.
Name	String	Specifies the name of this `vrfigure` object. Read/write.
NavMode	`'fly'` | `'examine'` | `'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.

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 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 4-10. 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.wrl');
```

The `vrworld` object `myworld` is associated with the virtual world vrmount.wrl. Open the world:

```
open(myworld)
```

Create a `vrfigure` object:

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

The VR Car in the Mountains virtual world opens in the Virtual Reality Toolbox viewer. You can now set the object properties of the `vrfigure` object `f`:

```
set(f,'Name','Car on a Mountain Road')
```

You can see that the name of the virtual world has changed in the viewer.

vrfigure/set

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, MATLAB creates one and returns its handle.

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

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

vrfigure/vrgcbf

Purpose   Current callback `vrfigure` object

Syntax   `f = vrgcbf`

Description   `f = vrgcbf` returns a `vrfigure` object representing the virtual reality figure that contains the callback currently being executed.

When no virtual reality figure callbacks are executing, `vrgcbf` returns an empty array of `vrfigure` objects.

Purpose   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
DataTypeBool	Specifies the handling of the VRML Bool data type for vrnode/setfield and vrnode/getfield. 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'.
DataTypeInt32	Specifies handling of the VRML Int32 data type for vrnode/setfield and vrnode/getfield. Valid values are 'int32' and 'double'. If set to 'int32', the VRML Int32 data type is returned as int32. If set to 'double', the Int32 data type is returned as 'double'. Default is 'double'.
DataTypeFloat	Specifies the handling of the VRML float data type for vrnode/setfield and vrnode/getfield. Valid values are 'single' and 'double'. If set to 'single', the VRML Float and Color data types are returned as 'single'. If set to 'double', the Float and Color data types are returned as 'double'. Default is 'double'.

Preference	Description
DefaultFigureAnti Aliasing	Determines whether antialiasing is used by default for new vrfigure objects. Valid values are 'off' and 'on'.
DefaultFigureDeleteFcn	Specifies the default callback invoked when closing a vrfigure object.
DefaultFigureLighting	Specifies whether the lights are rendered by default for new vrfigure objects. Valid values are 'off' and 'on'.
DefaultFigureMax TextureSize	Specifies the default maximum size of a texture used in rendering new vrfigure objects. Valid values are 'auto' and 32 <= x <= video card limit, where x is a power of 2.
DefaultFigureNavPanel	Specifies the default appearance of the control panel in the viewer. Valid values are 'opaque', 'translucent', 'none', 'halfbar', 'bar', and 'factory'. Default is 'halfbar'.
DefaultFigureNavZones	Specifies whether the navigation zone is on or off by default for new vrfigure objects. Valid values are 'off' and 'on'.
DefaultFigurePosition	Sets the default initial position and size of the Virtual Reality Toolbox viewer window. Valid value is a vector of four doubles.

Preference	Description
`DefaultFigureRecord2D CompressMethod`	Specifies the default compression method for creating 2-D animation files for new `vrfigure` objects. Valid values are `''`, `'auto'`, `'lossless'`, and `'codec_code'`.
`DefaultFigureRecord2D CompressQuality`	Specifies the default quality of 2-D animation file compression for new `vrfigure` objects. Valid values are 0-100.
`DefaultFigureRecord2D FileName`	Specifies the default 2-D offline animation filename for new `vrfigure` 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 `vrfigure` objects. Valid values are `'off'` and `'on'`.
`DefaultFigureToolBar`	Specifies whether the toolbar appears by default on the Virtual Reality Toolbox viewer for new `vrfigure` objects. Valid values are `'off'` and `'on'`.
`DefaultFigure Transparency`	Specifies whether or not transparency information is taken into account when rendering for new `vrfigure` objects. Valid values are `'off'` and `'on'`.

Preference	Description
DefaultFigureWireframe	Specifies whether objects are drawn as solids or wireframes by default for new `vrfigure` objects. Valid values are `'off'` and `'on'`.
DefaultViewer	Specifies which viewer is used to view a virtual scene. The Virtual Reality Toolbox viewer is used when the preference is set to `'internal'`. The Web browser is used when this preference is set to `'web'`. Default is `'internal'`.
DefaultWorldRecord3D FileName	Specifies the default 3-D animation filename for new `vrworld` objects.
DefaultWorldRecordMode	Specifies the default animation recording mode for new `vrworld` objects. Valid values are `'manual'` and `'scheduled'`.
DefaultWorldRecord Interval	Specifies the default start and stop times for scheduled animation recording for new `vrworld` objects. Valid value is a vector of two doubles.
DefaultWorldRemoteView	Specifies whether the virtual world is enabled by default for remote viewing for new `vrworld` objects. Valid values are `'off'` and `'on'`.
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.

vrgetpref

Preference	Description
HttpPort	IP port number used to access the VR server over the Web via HTTP. If you change this preference, you must restart MATLAB before the change takes effect.
TransportBuffer	Length of the transport buffer (network packet overlay) for communication between the VR server and its clients.
TransportTimeout	Amount of time the VR Toolbox server waits for a reply from the client. If there is no response from the client, the VR Toolbox server disconnects from the client.
VrPort	IP port used for communication between the VR server and its clients. If you change this preference, you must restart MATLAB before the change takes effect.

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:

%matlabroot	Refers to the MATLAB root directory
%file	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 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`

vrinstall

Purpose   Install and check Virtual Reality Toolbox components

Syntax
```
vrinstall('action')
vrinstall action
vrinstall('action','component')
vrinstall action component
x = vrinstall('action', 'component')
```

Arguments

action	Type of action for this function. Values are -interactive, -selftest, -check, -install, and -uninstall.
component	Name of the component for the action. Values are viewer and editor.

Description   You use this function to manage the installation of optional software components related to Virtual Reality Toolbox. Currently there are two such components: VRML plug-in and VRML editor.

Action Value	Description
-selftest	Checks the integrity of Virtual Reality Toolbox. If this function reports an error, you should reinstall Virtual Reality Toolbox. The function vrinstall automatically does a self-test with any other actions.
-interactive	Checks for the installed components, and then displays a list of uninstalled components you can choose to install.
-check	Checks the installation of optional components. If the given component is installed, returns 1. If the given component is not installed, returns 0. If you do not specify a component, displays a list of components and their status.

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 MATLAB and, from the **Control Panel** window, select **Add/Remove Programs**.

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
axis	a = axis(joy, n) reads the status of joystick with axis number n. Axis status is returned in the range of -1 to 1. The n parameter may be a vector to return multiple buttons.
button	b = button(joy, n) reads the status of joystick button number n. Button status is returned as logical 0 if not pressed and logical 1 if pressed. The n parameter may be a vector to return multiple buttons.
caps	c = caps(joy) returns joystick capabilities, such as the number of axes, buttons, POVs, and force-feedback axes. The return value is a structure with fields named Axes, Buttons, POVs, and Forces.
close	close(joy) closes and invalidates the joystick object. The object cannot be used once it is closed.

Method	Description
force	force(joy, n, f) applies force feedback to joystick axis n. The n parameter can be a vector to affect multiple axes. f values should be in range of -1 to 1, and the number of elements in f should either match the number of elements of n, or f can be a scalar to be applied to all the axes specified by n.
pov	p = pov(joy, n) reads the status of joystick POV (point of view) of control number n. pov is usually returned in degrees, with -1 meaning "not selected." n can be a vector to return multiple POVs.
read	[axes, buttons, povs] = read(joy) reads the status of axes, buttons, and POVs of the specified joystick. [axes, buttons, povs] = read(joy, forces) applies feedback forces, in addition, to a force-feedback joystick.

where joy is the handle to the joystick object.

Purpose   Open Simulink block library for Virtual Reality Toolbox

Syntax   `vrlib`

Description   The Simulink library for Virtual Reality Toolbox has five blocks: VR Sink, VR Placeholder, VR Signal Expander, Joystick Input, and Space Mouse Input.

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

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(parent_node,'parent_field', 'node_name',
'node_type')
```

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.

Description   mynode = vrnode creates an empty vrnode handle that does not reference any node.

mynode = vrnode([]) creates an empty array of vrnode handles.

mynode = vrnode(vrworld_object, 'node_name') creates a handle to an existing named node in the virtual world.

mynode = vrnode(vrworld_object, 'node_name','node_type') creates a new node called *node_name* of type *node_type* on the root of the virtual world. It returns the handle to the newly created node.

mynode = vrnode(parent_node,'parent_field', 'node_name','node_type') creates a new node called node_name of type node_type that is a child of the parent_node and resides in the field parent_field. It returns the handle to the newly created node.

A vrnode object identifies a virtual world node in a way very similar to a handle. If the vrnode method is applied to a node that does not

exist, the node is created, the `vrnode` object is created, and the handle to the `vrnode` object is returned. If the `vrnode` method is applied to an existing node, the handle to the `vrnode` object associated with this node is returned.

See Also   `vrnode/delete`, `vrnode/get`, `vrnode/getfield`, `vrnode/set`, `vrnode/setfield`, `vrworld`

vrnode/delete

Purpose　　Remove vrnode object

Syntax　　delete(vrnode_object)
delete(n)

Arguments　　vrnode_object　　Name of a vrnode object.

Description　　delete(vrnode_object) deletes the virtual world node.

delete(n) deletes the vrnode object referenced by the vrnode handle n. If n is a vector of vrnode handles, multiple nodes are deleted.

As soon as a node is deleted, it and all its child objects are removed from all clients connected to the virtual world.

See Also　　vrworld/delete

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`

vrnode/get

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.

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

vrnode/getfield

Purpose   Field value of vrnode object

Syntax   getfield(vrnode_object)
                 x = getfield(vrnode_object)
                 x = getfield(vrnode_object,'fieldname')

Arguments

vrnode_object	Name of a vrnode object representing the node to be queried.
fieldname	Name of the vrnode object field whose values you want to query.

Description getfield(vrnode_object) displays all the field names and their current values for the respective VRML node.

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

x = getfield(vrnode_object,'fieldname') returns the value of the specified field for the node referenced by the vrnode_object handle. If vrnode_object is a vector of vrnode handles, getfield returns an M-by-1 cell array of values, where M is equal to length(vrnode_object).

If 'fieldname' is a 1-by-N or N-by-1 cell array of strings containing field names, getfield returns an M-by-N cell array of values.

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

See Also   vrnode, vrnode/get, vrnode/set, vrnode/setfield

Purpose   1 if vrnode object is valid, 0 if not

Syntax   x = isvalid(vrnode_object_vector)

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

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

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

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

See Also   vrfigure/isvalid, vrworld/isvalid

vrnode/set

Purpose   Change property of virtual world node

Syntax   x = set(vrnode_object, '*property_name*','property_value')

Arguments

vrnode_object	Name of a vrnode object representing a node in the virtual world.
property_name	Name of a property.
property_value	Value of a property.

Description   x = set(vrnode_object, '*property_name*','property_value') changes the specified property of the vrnode object to the specified value.

The vrnode property values are case sensitive, while property names are not case sensitive.

The vrnode property values are case sensitive, while property names are not case sensitive.

The vrnode objects have the following properties. All these properties are read only.

Property	Value	Description
Fields	Cell array	Valid field names for the VRML node. Read only.
Name	String	Name of the node. Read only.
Type	String	VRML type of the node. The value is a string (for example, 'Transform', 'Shape'). Read only.
World	Handle	Handle of the parent vrworld object. This is a vrworld object that represents the node's parent world. Read only.

Currently, VRML nodes have no settable properties.

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

vrnode/setfield

Purpose	Change field value of vrnode object

Syntax

x = setfield(vrnode_object,'fieldname','fieldvalue')

Arguments

vrnode_object	Name of a vrnode object representing the node to be changed.
fieldname	Name of the vrnode object VRML field whose values you want to set.
fieldvalue	Value of fieldname.

Description

x = setfield(vrnode_object,'fieldname','fieldvalue') changes the specified field of the vrnode object to the specified value. You can specify multiple field names and field values in one line of code by grouping them in pairs. For example, x = setfield(vrnode_object, 'fieldname1', 'fieldvalue1', 'fieldname2', 'fieldvalue2',...).

Note that 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

Purpose   Enable or disable synchronization of VRML fields with client

Syntax   sync(vrnode_object, 'field_name', '*action*')

Arguments

vrnode_object	Name of a vrnode object representing the node.
field_name	Name of the VRML field to be synchronized.
action	The action parameter determines what should be done:

- 'on' enables synchronization of this field.

- 'off' disables synchronization of this field.

Description The sync method controls whether the value of a 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 Simulink.

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

vrori2dir

Purpose	Convert viewpoint orientation to direction
Syntax	`vrori2dir(r)` `vrori2dir(r,options)`
Description	`vrori2dir(r)` converts the viewpoint orientation, specified by a rotation vector, r, to a direction the viewpoint points to. `vrori2dir(r,options)` converts the viewpoint orientation with the default algorithm parameters replaced by values defined in `options`. The `options` structure contains the parameter `epsilon` that represents the value below which a number will be treated as zero (default value is 1e-12).
See Also	`vrdir2ori`, `vrrotmat2vec`, `vrrotvec`, and `vrrotvec2mat`

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.

time indicator

vrplay
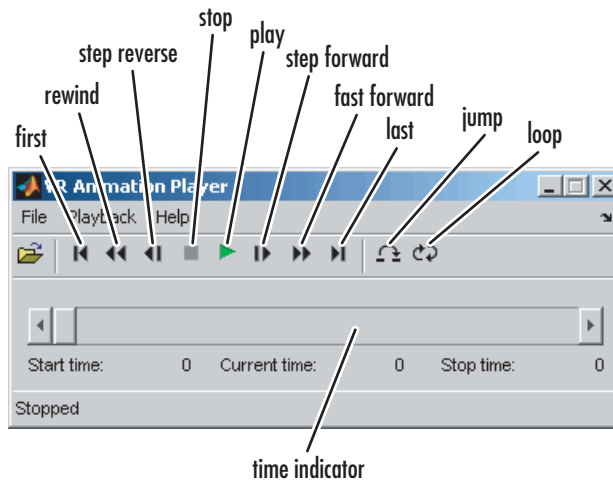
Keyboard Support

The GUI's playback controls can also be accessed from the keyboard.

Key	Function
f	fast forward
j	jump to time
l	loop
p	play
s	stop
r	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 4-9, `vrview`

Purpose   Calculate a 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

vrrotmat2vec

Purpose	Convert rotation from matrix to axis-angle representation
Syntax	`r = vrrotmat2vec(m)` `r = vrrotmat2vec(m,options)`
Description	`r = vrrotmat2vec(m)` returns an axis-angle representation of rotation defined by the rotation matrix `m`.

Description   `r = vrrotmat2vec(m,options)` converts the rotation with the default algorithm parameters replaced by values defined in `options`.

The `options` structure contains the parameter `epsilon` that represents the value below which a number will be treated as zero (default value is 1e-12).

The result `r` is a four-element axis-angle rotation row vector. The first three elements specify the rotation axis, and the last element defines the angle of rotation.

See Also   `vrrotvec` and `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`

# vrsetpref

**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 |
|---|---|
| DataTypeBool | Specifies the handling of the VRML `Bool` data type for `vrnode/setfield` and `vrnode/getfield`. 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'`. |
| DataTypeInt32 | Specifies handling of the VRML `Int32` data type for `vrnode/setfield` and `vrnode/getfield`. Valid values are `'int32'` and `'double'`. If set to `'int32'`, the VRML `Int32` data type is returned as `int32`. If set to `'double'`, the `Int32` data type is returned as `'double'`. Default is `'double'`. |

| Preference | Description |
|---|---|
| DataTypeFloat | Specifies the handling of the VRML float data type for vrnode/setfield and vrnode/getfield. Valid values are 'single' and 'double'. If set to 'single', the VRML Float and Color data types are returned as 'single'. If set to 'double', the Float and Color data types are returned as 'double'. Default is 'double'. |
| DefaultFigureAnti Aliasing | Determines whether antialiasing is used by default for new vrfigure objects. Valid values are 'off' and 'on'. |
| DefaultFigureDeleteFcn | Specifies the default callback invoked when closing a vrfigure object. |
| DefaultFigureLighting | Specifies whether the lights are rendered by default for new vrfigure objects. Valid values are 'off' and 'on'. |
| DefaultFigureMax TextureSize | Specifies the default maximum size of a texture used in rendering new vrfigure objects. Valid values are 'auto' and 32 <= x <= video card limit, where x is a power of 2. |
| DefaultFigureNavPanel | Specifies the default appearance of the control panel in the viewer. Valid values are 'opaque', 'translucent', 'none', 'halfbar', 'bar', and 'factory'. Default is 'halfbar'. |

| Preference | Description |
|---|---|
| `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. |
| `DefaultFigureRecord2D CompressMethod` | Specifies the default compression method for creating 2-D animation files for new `vrfigure` objects. Valid values are `''`, `'auto'`, `'lossless'`, and `'codec_code'`. |
| `DefaultFigureRecord2D CompressQuality` | Specifies the default quality of 2-D animation file compression for new `vrfigure` objects. Valid values are 0-100. |
| `DefaultFigureRecord2D FileName` | Specifies the default 2-D offline animation filename for new `vrfigure` 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 `vrfigure` objects. Valid values are `'off'` and `'on'`. |
| `DefaultFigureToolBar` | Specifies whether the toolbar appears by default on the Virtual Reality Toolbox viewer for new `vrfigure` objects. Valid values are `'off'` and `'on'`. |

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

# vrsetpref

| 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 MATLAB before the change takes effect. |
| TransportBuffer | Length of the transport buffer (network packet overlay) for communication between the VR server and its clients. |
| TransportTimeout | Amount of time the VR Toolbox server waits for a reply from the client. If there is no response from the client, the VR Toolbox server disconnects from the client. |
| VrPort | IP port used for communication between the VR server and its clients. If you change this preference, you must restart MATLAB before the change takes effect. |

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

When you use 'factory' as a single argument, all preferences are reset to their default values. If you use 'factory' for a preference value, that single preference is reset to its default.

**See Also**    vrgetpref

**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. |
| `LowerPositionLimit` | Position coordinates for the lower limit of the mouse. |

| Property | Description |
|----------|-------------|
| NormalizeOutputAngle | Determines whether the integrated rotation angles should wrap on a full circle (360°). This is not used when you read the Output Type as Speed. |
| InitialPosition | Initial condition for integrated translations. This is not used when you set the Output Type to Speed. |
| InitialRotation | Initial condition for integrated rotations. This is not used when you set the Output Type to Speed. |

**Methods**

| Method | Description |
|--------|-------------|
| button | b = button(mouse, n) reads the status of space mouse button number n. Button status is returned as logical 0 if not pressed and logical 1 if pressed. n can be a vector to return multiple buttons. |
| close | close(mouse) closes and invalidates the space mouse object. The object cannot be used once it is closed. |
| position | p = position(mouse, n) reads the position of space mouse axis number n. n can be a vector to return positions of multiple axes. Translations and rotations are integrated. Outputs are the position and orientation in the form of roll/pitch/yaw angles. |

| Method | Description |
|--------|-------------|
| speed | s = speed(mouse, n) reads the speed of space mouse axis number n. n can be a vector to return the speeds of multiple axes. No transformations are done. Outputs are the translation and rotation speeds. |
| viewpoint | p = viewpoint(mouse) reads the space mouse coordinates in VRML viewpoint format. Translations and rotations are integrated. Outputs are the position and orientation in the form of an axis and an angle. You can use these values as viewpoint coordinates in VRML. |

# vrview

**Purpose**     View virtual world using Virtual Reality Toolbox viewer or Web browser

**Syntax**
```
vrview
x = vrview('filename')
x = vrview('filename','-internal')
x = vrview('filename','-web')
```

**Description**     `vrview` opens the default Web browser and loads the Virtual Reality Toolbox Web page containing a list of virtual worlds available for viewing.

`x = vrview('filename')` creates a virtual world associated with the `.wrl` file, opens the virtual world, and displays it in the Virtual Reality Toolbox viewer or the Web browser depending on the value of the `DefaultViewer` preference. The handle to the virtual world is returned.

`x = vrview('filename','-internal')` creates a virtual world associated with the `.wrl` file, opens the virtual world, and displays it in the Virtual Reality Toolbox viewer.

`x = vrview('filename','-web')` creates a virtual world associated with the `.wrl` file, opens the virtual world, and displays it in your Web browser.

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

**Purpose**    List virtual worlds in memory

**Syntax**     vrwho
               x = vrwho

**Description**    If you do not specify an output parameter, `vrwho` displays a list of
                  virtual worlds in memory in the MATLAB Command Window.

                  If you specify an output parameter, `vrwho` returns a vector of handles to
                  existing `vrworld` objects, including those opened from Simulink.

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

# vrwhos

**Purpose**          List details about virtual worlds in memory

**Syntax**           vrwhos

**Description**      vrwhos displays a list of virtual worlds currently in memory, with a
                     description, in the MATLAB Command Window. The relation between
                     vrwho and vrwhos is similar to the relation between who and whos.

**See Also**         vrclear, vrwho

**Purpose**          Create new vrworld object associated with virtual world

**Syntax**           myworld = vrworld('filename')
                     myworld = vrworld
                     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 .wrl is assumed.

**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 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')

**See Also**         vrworld/close, vrworld/delete, vrworld/open

# vrworld/close

**Purpose**    Close virtual world

**Syntax**    `close(vrworld_object)`

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

**Description**    This method changes the virtual world from an opened to a closed state:

- If the world was opened more than once, you must use an appropriate number of `close` calls before the virtual world closes.
- If `vrworld_object` is a vector of `vrworld` objects, all associated virtual worlds close.
- If the virtual world is already closed, `close` does nothing.

Opening and closing virtual worlds is a mechanism of memory management. When the system needs more memory and the virtual world is closed, you can discard its contents at any time.

Generally, you should close a virtual world when you no longer need it. This allows you to reuse the memory it occupied. The `vrworld` objects associated with this virtual world stay valid after it is closed, so the virtual world can be opened again without creating a new `vrworld` object.

**Examples**
```
myworld = vrworld('vrpend.wrl')
open(myworld)
close(myworld)
```

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

**Purpose**      Remove virtual world from memory

**Syntax**       delete(vrworld_object)

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

**Description**  The delete method removes from memory the virtual world associated
                 with a vrworld object. The virtual world must be closed before you
                 can delete it.

                 Deleting a virtual world frees the virtual world from memory and
                 invalidates all existing vrworld objects associated with the virtual
                 world.

                 If vrworld_object is a vector of vrworld objects, all associated virtual
                 worlds are deleted.

                 You do not commonly use this method. One of the possible reasons to
                 use this method is to ensure that a large virtual world is removed from
                 memory before another memory-consuming operation starts.

**See Also**     vrclear, vrworld/close

# vrworld/edit

**Purpose**    Open virtual world file in external VRML editor

**Syntax**    edit(vrworld_object)

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

**Description**    The edit method opens the VRML file associated with the vrworld object in a VRML editor. The Editor preference specifies the VRML editor to use. See vrsetpref for details on setting preferences.

The VRML editor saves any changes you make directly to a virtual world file. If the virtual world is open,

- Use the save command in the VRML editor to save the changes to a virtual world file. In MATLAB, the changes appear after you reload the virtual world.

- Use the save method in MATLAB to replace the modified VRML file. Any changes you made in the editor are lost.

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

**Purpose**       Property value of vrworld object

**Syntax**        get(vrworld_object)
                  x = get(vrworld_object)
                  x = get(vrworld_object, '*property_name*')

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

                  *property_name*   Name of the property.

**Description**   get(vrworld_object) displays all the virtual world properties and
                  their values.

                  x = get(vrworld_object) returns an M-by-1 structure where the field
                  names are the names of the virtual world properties. Each field contains
                  the associated property value. M is equal to length(vrworld_object).

                  x = get(vrworld_object, '*property_name*') returns the value of
                  the specified property.

                  • If vrworld_object is a vector of vrworld handles, the get method
                    returns an M-by-1 cell array of values where M is equal to
                    length(vrworld_object).

                  • If *property_name* is a 1-by-N or N-by-1 cell array of strings containing
                    field names, the get method returns an M-by-N cell array of values.

                  The following are properties of vrworld objects. Names are not case
                  sensitive.

| Property | Value | Description |
|----------|-------|-------------|
| Clients | Scalar | Number of clients currently viewing the virtual world. Read only. |

| Property | Value | Description |
|---|---|---|
| ClientUpdates | `'off'` \| `'on'`<br>Default: `'on'` | Client cannot or can update the virtual scene. Read/write. |
| Description | String.<br>Default: automatically taken from the VRML file property `title` | Description of the virtual world as it appears on the main Web page. Read/write. |
| Figures | Vector of `vrfigure` objects | Vector of handles to Virtual Reality Toolbox viewer windows currently viewing the virtual world. Read only. |
| FileName | String | Name of the associated VRML file. Read only. |
| Nodes | Vector of `vrnode` objects | Vector of `vrnode` objects for all named nodes in the virtual world. Read only. |
| Open | `'off'` \| `'on'`<br>Default: `'off'` | Indicates a closed or open virtual world. Read only. |
| Record3D | `'off'` \| `'on'`<br>Default: `'off'` | Enables 3-D animation recording. Read/write. |
| Record3DFileName | String.<br>Default:<br>`'%f_anim_%n.wrl'` | 3-D animation filename. The string can contain tokens that are replaced by the corresponding information when the animation recording takes place. For details, see "Animation Recording File Tokens" on page 4-10. Read/write. |
| Recording | `'off'` \| `'on'`<br>Default: `'off'` | Animation recording toggle. This property acts as the master recording switch. Read/write. |
| RecordMode | `'manual'` \| `'scheduled'`<br>Default: `'manual'` | Animation recording mode. Read/write. |

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

The `ClientUpdates` property is set to `'on'` by default and can be set by the user. When it is set to `'off'`, the viewers looking at this virtual world should not update the view according to the virtual world changes. That is, the view is frozen until this property is changed to `'on'`. This is useful for preventing tearing effects with complex animations. Before every animation frame, set `ClientUpdates` to `'off'`, make the appropriate modifications to the object positions, and then switch `ClientUpdates` back to `'on'`.

The `Description` property defaults to `'(untitled)'` and can be set by the user. If the virtual world is loaded from a VRML file containing a

**WorldInfo** node with a `title` property, 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

# vrworld/nodes

**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

**Purpose**       Open virtual world

**Syntax**        open(vrworld_object)

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

**Description**   The open method opens the virtual world. When the virtual world is opened for the first time, the virtual world internal representation is created based on the associated VRML file.

If the input argument is an array of virtual world handles, all the virtual worlds associated with those handles are opened.

The virtual world must be open for you to use it. You can close the virtual world with the method close.

You can call the method open more than once, but you must use an appropriate number of close calls before the virtual world returns to a closed state.

**Examples**      Create two vrworld objects by typing

```
myworld1 = vrworld('vrmount.wrl')
myworld2 = vrworld('vrpend.wrl')
```

Next, create an array of virtual world handles by typing

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

open(myworlds) opens both of these virtual worlds.

See Also   vrworld, vrworld/close

vrworld/reload

Purpose   Reload virtual world from VRML file

Syntax   `reload(vrworld_object)`

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

Description   The `reload` method reloads the virtual world from the VRML file associated with the `vrworld` object. If the input argument is an array of virtual world handles, all the virtual worlds associated with those handles are reloaded. The virtual world must be open for you to use this method.

`reload` forces all the clients currently viewing the virtual world to reload it. This is useful when there are changes to the VRML file.

See Also   `vrworld/edit`, `vrworld/open`, `vrworld/save`

Purpose   Write virtual world to VRML file

Syntax   save(vrworld_object, 'vrml_file')

Arguments   vrworld_object A vrworld object representing a virtual world.

vrml_file   Name of the VRML file to save the virtual world to.

Description   The save method saves the current virtual world to a VRML97 file. The virtual world must be open for you to use this method.

The resulting file is a VRML97 compliant UTF-8 encoded text file. Lines are indented using spaces. Line ends are encoded as CR-LF or LF according to the local system default. Values are separated by spaces.

See Also   vrworld/edit, vrworld/open, vrworld/reload

vrworld/set

Purpose   Change property values of vrworld object

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

Arguments

vrworld_object	Name of a vrworld object representing a virtual world.
property_name	Name of the property.
property_value	New value of the property.

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

Property	Value	Description
Clients	Scalar	Number of clients currently viewing the virtual world. Read only.
ClientUpdates	'off' | 'on' Default: 'on'	Client cannot or can update the virtual scene. Read/write.
Description	String. Default: automatically taken from the VRML file property title	Description of the virtual world as it appears on the main Web page. Read/write.
Figures	Vector of vrfigure objects	Vector of handles to 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 4-10. Read/write.
Recording	'off' | 'on' Default: 'off'	Animation recording toggle. This property acts as the master recording switch. Read/write.
RecordMode	'manual' | 'scheduled' Default: 'manual'	Animation recording mode. Read/write.
RecordInterval	Vector of two doubles Default: [0 0]	Start and stop times for scheduled animation recording. Corresponds to the virtual world object Time property. Read/write.
RemoteView	'off' | 'on' Default: 'off'	Remote access flag. If the virtual world is enabled for remote viewing, it is set to 'on'; otherwise, it is set to 'off'. Read/write.
Time	Double	Current time in the virtual world. Read/write.

vrworld/set

Property	Value	Description
TimeSource	`'external'` | `'freerun'` Default: `'external'`	Source of the time for the virtual world. If set to `'external'`, time in the scene is controlled from MATLAB (by setting the `Time` property) or Simulink (simulation time). If set to `'freerun'`, time in the scene advances independently based on the system timer. Read/write.
View	`'off'` | `'on'` Default: `'on'`	Indicates an unviewable or viewable virtual world. Read/write.

See Also   `vrworld`, `vrworld/get`

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

vrworld/view

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 1-10.

Glossary-2

W

Index-6