

# **VR Juggler**

## **Getting Started Guide**

---

# VR Juggler: Getting Started Guide

Published \$Date: 2002/04/05 21:37:48 \$

---

---

---

# Table of Contents

Preface .....	vii
1. Installing VR Juggler .....	1
Installing from a Compressed TAR File .....	1
Installing from a ZIP File (Win32 only) .....	1
2. Environment Variables .....	3
How to Set Environment Variables .....	3
Common Conventions and Background .....	3
C-Style Shells (csh, tcsh) .....	3
sh-Derived Shells (sh, ksh, bash, zsh, etc.) .....	4
DOS Shell .....	4
Win 32 GUI .....	5
Syntax Used in this Document .....	5
Required Environment Variables .....	6
Optional Related Environment Variables .....	7
3. VR Juggler Sample Applications .....	8
OpenGL Applications .....	8
OpenGL Performer Applications .....	9
4. Compiling a VR Juggler Sample Program .....	10
Required Reading .....	10
Compiling an Application .....	10
Compiling from the Command Line .....	10
Compiling Using Microsoft Visual Studio .....	10
Additional Notes .....	11
5. Running a VR Juggler Sample Program .....	12
Required Reading .....	12
Running an Application in Simulator Mode .....	12
Using a Simulated Glove .....	15
Using a Simulated Analog Device .....	15
Running an Application in a VR System .....	16

---

# List of Figures

- 1.1. Windows Folder View of ZIP File .....
- 1.2. Open WinZip Window .....
- 1.3. WinZip Extract Dialog .....
- 4.1. Selecting the Workspace File .....
- 4.2. torus Project .....
- 4.3. Project Menu .....
- 4.4. Project Settings Dialog .....
- 4.5. Build torus.exe .....
- 4.6. Execute torus.exe .....
- 5.1. torus running on an IRIX desktop .....

---

## List of Tables

5.1. Moving the simulated head .....	14
5.2. Moving the simulated wand .....	14
5.3. Moving the camera .....	15
5.4. Analog Device Simulator Keys .....	15

---

# Preface

This book is for people who are just getting started with VR Juggler. It guides new users through getting and installing VR Juggler, configuring users' environment to use it, and compiling and running a sample application.

The prerequisites for reading this book are minimal. They are:

- Some experience with a command-line interface (i.e., a shell such as tcsh or the DOS shell)
- Creating and browsing directories

Those users who want to get more involved with VR Juggler to do more than just run applications should be aware right away of the following prerequisites:

- Knowledge of C++ and object-oriented design
- Knowledge of one of VR Juggler's currently supported graphics APIs (OpenGL [<http://www.opengl.org/>], OpenGL Performer [<http://www.sgi.com/software/performer/>], or Open Scene Graph [<http://www.openscenegraph.org/>])

---

---

# Chapter 1. Installing VR Juggler

As with most Open Source projects, VR Juggler is distributed as compressed archive files using popular formats. Installing a distribution requires very little effort, but you do need to know how to use archiving utilities to extract the installation tree. Automation of the installation is a goal of the VR Juggler team, but we are still finalizing the details of cross-platform installation management. Before reading further, you should know where you want to install VR Juggler, and you should make sure that you have access to write to that directory.

## Installing from a Compressed TAR File

The TAR (Tape ARchive) format has been around for a long, long time in the UNIX world. It is simply a collection of files in a directory tree that are lumped into a single file suitable for writing to a tape or for downloading. The format is a standard, and the **tar**(1) utility is available on every UNIX-based platform and on Win32. A free version can be downloaded from the GNU Project [<http://www.gnu.org/>]. A compressed TAR file is made for each VR Juggler distribution, and some distributions come in other formats as well. You can always count on the availability of a TAR file, though. The TAR files are compressed using GZIP which is a standard compression format. The **gzip**(1) utility is freely available from the GNU Project (the GNU version of TAR has the GZIP algorithm built in) if your platform does not have it.

Once you have downloaded a VR Juggler TAR distribution, you can unpack it one of two ways depending on what your platform's version of TAR supports. Before extracting the installation tree, make sure that your current directory is the one where you want to install VR Juggler. If your version of TAR does not have GZIP built in (it does not support the `-z` option), the following command will do the decompression and extraction:

```
% gzip -cd vrjuggler-distribution.tar.gz | tar -xvf -vrjuggler-distribution.tar.gz
```

Here, you should fill in `vrjuggler-distribution.tar.gz` with the name of the VR Juggler distribution file you downloaded. The above command will work with any shell that supports redirection of standard output to a pipe. If that looks too scary, you can separate the decompression and extraction into two commands:

```
% gunzip vrjuggler-distribution.tar.gz
% tar -xvf vrjuggler-distribution.tar
```

Note that the distribution file in the second command does not have the `.gz` extension after **gzip**(1) is run. These steps also work if your version of **tar**(1) supports the `-z` option, but you can simplify your work if that option is supported. The following illustrates how to uncompress and extract all in one step:

```
% tar -xzvf vrjuggler-distribution.tar.gz
```

In either case, while the command runs, you will see the name of each file as it is written to disk. This is because of the `-v` option to **tar**(1) that tells it to be verbose in its efforts. **tar**(1) takes care of creating all the directories in the installation tree, so you only need to have the base directory (for example, `/usr/local`) when you start. For more information about these utilities, please refer to the **tar**(1) and **gzip**(1) manual pages.

## Installing from a ZIP File (Win32 only)

On the Win32 family of platforms, the ZIP format rules. In the old days, you would use the PKZIP utility to uncompress and extract a ZIP file. Nowadays, most people use WinZip [<http://www.winzip.com/>] or some other comparable graphical interface. This documentation covers only the use of WinZip when extracting a ZIP file.

Once you have downloaded the VR Juggler ZIP file, the easiest way to extract it is to double-click on its icon in the open folder window as shown in Figure 1.1.

Double-clicking opens the main WinZip window, as shown in Figure 1.2.

Note that in this screen shot, the Extract button is highlighted. Click this button to open the following window: Note that in this screen shot, the Extract button is highlighted. Click this button to open the dialog box shown in Figure 1.3.

In this window, choose the directory where VR Juggler will be installed and click Extract. WinZip will then proceed to extract the ZIP file into the directory you named. That is all there is to it.

---

# Chapter 2. Environment Variables

There are several *environment variables* that affect the way VR Juggler works. Some of these are required to compile and run applications while others are optional. This chapter lists all such variables and explains their meanings and uses.

## How to Set Environment Variables

The syntax for setting or changing an environment variable varies with operating systems and shell interpreters. Instead of choosing one style of syntax that is specific to a particular shell type, we define our own syntax which you must then translate to your shell's specific syntax. Before defining this syntax, we present the method used to set environment variables in the three most common types of shells. We also provide a quick overview of how to set environment variables using Win32-based GUIs.

## Common Conventions and Background

A convention used throughout this book is to name the variables using all capital letters. In almost all cases, regardless of the shell, this is the naming convention used for environment variables.

Setting a path with an environment variable can require special syntax. Because of this, the method for doing so may vary from shell to shell. Paths are important with VR Juggler when looking up the path to a shared library (dynamically linked library). For each shell, the syntax for setting a path is given.

Referring to environment variables can also vary from shell to shell. An example of how to print the value of an environment variable will be given for each shell. An example of how to refer to an environment variable is also provided as these two operations may vary even within one kind of shell!

In all shells, an environment variable is only available within that single shell instance. That is, setting an environment variable at a command prompt only affects that specific shell and will not be available from other concurrent or future shells. To make a setting “permanent”, it should be done in a file read by all shell instances when they are started. This is addressed briefly as appropriate for each shell type.

## C-Style Shells (csh, tcsh)

In a C-style shell (i.e., one whose interface is based on the C programming language), setting environment variables is done using the built-in command **setenv**. It is used as follows:

```
% setenv <VARIABLE_NAME> <value>
```

where the string `<VARIABLE_NAME>` represents the name of the variable you are going to set and `<value>` represents the value assigned to that variable. Both are required. If the named variable did not exist before, it will pop into existence. Otherwise, you overwrite the old setting with the new one.

To print the value of an environment variable, use the following command:

```
% printenv <VARIABLE_NAME>
```

Referring to a variable, however, is done using the following syntax:

```
% cd $VARIABLE_NAME/bin
```

Paths are specified as a colon-separated list. An example of this is:

```
% printenv PATH
/bin:/sbin:/usr/bin:/usr/sbin
```

For these types of shells, a “permanent” setting for a given variable should usually be done in your `.cshrc` file or in your `.login` file, both of which should be in your home directory. In most cases, it is better to use `.cshrc` because it is evaluated for every shell instance.

## sh-Derived Shells (sh, ksh, bash, zsh, etc.)

In a shell based on sh, setting environment variables is done using the built-in command **export**. It is used as follows:

```
% export <VARIABLE_NAME>=<value>
```

or

```
% <VARIABLE_NAME>=<value>
% export <VARIABLE_NAME>
```

Here, the string `<VARIABLE_NAME>` represents the name of the variable you are going to set and `<value>` represents the value assigned to that variable. Both are required. Note that there is no space between the variable name and its value. If the named variable did not exist before, it will pop into existence. Otherwise, you overwrite the old setting with the new one. If the variable was already among your current shell's environment variables, the `export` command is not necessary.

To print the value of an environment variable, use the following command:

```
% echo $VARIABLE_NAME
```

Getting the value of a variable works the same way.

Paths are specified as a colon-separated list. An example of this is:

```
% echo $PATH
/bin:/sbin:/usr/bin:/usr/sbin
```

For these types of shells, a “permanent” setting for a given variable should usually be done in the `.profile` file in your home directory or in your shell's “rc” file. Different shells have different names for this file. Examples are `.bashrc` for BASH and `.zshrc` for Zsh. Please refer to your shell's documentation for more information. In any case, the file will be in your home directory.

## DOS Shell

The typical syntax for setting an environment variable from the command line (in a DOS shell window) under Win32 is:

```
C:\ set <VARIABLE_NAME>=<value>
```

Here, `<VARIABLE_NAME>` is the name of the environment variable to be set, and `<value>` is the value being assigned to that variable. If the named variable did not exist before, it will pop into existence. Otherwise, you overwrite the old setting with the new one.

To print the value of an environment variable, use the following command:

```
C:\ set <VARIABLE_NAME>
```

Referring to a variable, however, is done using the following syntax:

```
C:\ cd %VARIABLE_NAME%\bin
```

Paths are specified as a semicolon-separated list. An example of this is:

```
C:\ set PATH  
C:\WINDOWS;C:\bin;C:\
```

For some versions of Windows, a “permanent” setting for a given variable should usually be done in `C:\AUTOEXEC.BAT`. In newer versions (Windows ME in particular) and in the Windows NT line of operating systems, the setting is done using the Control Panel. Please refer to the next section for more information on that method.

## Win 32 GUI

Before reading this section, please be sure to have read the section called “DOS Shell”. This is necessary because the Win32 GUI for setting environment variables is simply a front-end to that older method and thus uses the same conventions and syntax. The versions of Windows to which this subsection applies are indicated individually since each is a little different. For more detailed information, please refer to the Windows online help system and search for “environment variables”.

## Windows 2000

In the Control Panel, open the System icon. Under the Advanced tab, there is a button labeled Environment Variables. Here, you can set variables for yourself and, if you have the access privileges, for all users.

## Windows NT 4.0

In the Control Panel, open the System icon. The window that is opened has a tab labeled Environment Variables. Here, you can set variables for yourself and, if you have the access privileges, for all users.

## Syntax Used in this Document

To avoid tying this documentation to a single style of environment variable creation, assignment and reference, the following syntax will be used exclusively from this point onward. Please read this carefully before proceeding.

## Naming Environment Variables

When naming an environment variable in the plain text of this document, the variable will be referred to by its name only. For example, to talk about the environment variable containing your path, we will talk about it as `PATH`.

## Creating/Setting Environment Variables

The syntax to set an environment variable is:

```
% <VARIABLE_NAME> = <value>
```

Setting an environment variable also creates it if it is not already present in the current shell's environment.

## Printing an the Value of an Environment Variable

Printing an environment variable's value to standard output (stdout) is done as follows:

```
% echo $VARIABLE_NAME
value
```

## Referring to the Value of an Environment Variable

To get the value of an environment variable when it needs to be expanded, the following syntax will be used:

```
% cd $VARIABLE_NAME/bin
```

Here, the reference to the value is \$VARIABLE\_NAME.

## Required Environment Variables

### VJ\_BASE\_DIR

The environment variable `VJ_BASE_DIR` tells a VR Juggler application where to find important data files. It is required to compile and run any Juggler app. It should be set to the base directory of the installed VR Juggler library. For example, if you downloaded a UNIX version of VR Juggler 1.0 and extracted it to the directory `/home/software/`, you would set `VJ_BASE_DIR` with this command:

```
% VJ_BASE_DIR = /home/software/vrjuggler-1.0
```

The last component of the path depends on the particular version of Juggler you have downloaded.

If you downloaded and built VR Juggler from the source code, the compilation creates a directory called `instlinks` which can be used as a VR Juggler base:

```
% VJ_BASE_DIR = $HOME/juggler/my_build_dir/instlinks
```

In any case, on a Win32 platform, you should use `/`'s as the path separator for `VJ_BASE_DIR` rather than `\`'s. The compiler tools can handle either, and the utilities in `juggler-tools` will behave much better if UNIX-style paths are used. It is safe to use the drive letter at the start of the path (e.g., `C:/software/vrjuggler-1.0.5`).

### JDK\_HOME

The `JDK_HOME` environment variable is required by the script that starts `VjControl`, the VR Juggler configuration program. If Java is installed on your system, `JDK_HOME` may already be set. If not, it needs to be set to the base of the Java installation.

### LD\_LIBRARY\_PATH (UNIX/Linux only), LD\_LIBRARYN32\_PATH (IRIX only), LD\_LIBRARY64\_PATH (IRIX only)

UNIX/Linux systems use these environment variables to find dynamically loaded libraries, such as `libJuggler.so`. Unless you are building everything with static libraries, you will need to set these to include the VR Juggler library directory (under `VJ_BASE_DIR`). IRIX supports several Application Binary Interfaces (ABIs). VR Juggler supports only the N32 and 64 formats, and there are different library path variables for each. The N32 ABI uses the `LD_LIBRARYN32_PATH` variable, and the 64 ABI uses `LD_LIBRARY64_PATH`. An example of setting the library path is as follows:

```
% LD_LIBRARY_PATH = $VJ_BASE_DIR/lib
```

## Note

On some SGI systems running IRIX, users of the MIPSpro Compilers (version 7.3) will need to add another directory as follows:

```
% LD_LIBRARY_PATH = $LD_LIBRARY_PATH:/usr/lib32/cmplrs:$VJ_BASE_DIR
```

## Optional Related Environment Variables

PATH

If you intend to use the VjControl program, you may want to add the \$VJ\_BASE\_DIR/bin directory to your PATH as follows:

```
% PATH = $PATH:$VJ_BASE_DIR/bin
```

VJ\_DEBUG\_NFY\_LEVEL

This variable can be used to control the amount of diagnostic information a VR Juggler application outputs. Its value is a number between 0 (only very important messages are printed) and 7 (vast amounts of data) inclusive. Non-hackers are advised to use levels 0 through 3, as higher debug levels become increasingly cryptic and *can severely impact application performance*. The default is level 1—only errors and critical information are output. An example of setting a value for this variable is:

```
% VJ_DEBUG_NFY_LEVEL = 3
```

VJ\_DEBUG\_CATEGORIES

This variable can be used to control which components of VR Juggler are allowed to output diagnostic data. If for some reason you set VJ\_DEBUG\_NFY\_LEVEL to 5 or higher, this variable can be used to filter the output. The value of VJ\_DEBUG\_CATEGORIES is a space-separated list of Juggler debug component names (defined in \$VJ\_BASE\_DIR/include/Kernel/vjDebug.h). The default value is "DBG\_ALL", which performs no filtering whatsoever. Examples of setting it are as follows:

```
% VJ_DEBUG_CATEGORIES = DBG_ERROR  
% VJ_DEBUG_CATEGORIES = "DBG_KERNEL DBG_INPUT"  
% VJ_DEBUG_CATEGORIES = "DBG_CONFIGDB DBG_ENV_MGR"
```

---

# Chapter 3. VR Juggler Sample Applications

VR Juggler comes with several sample applications in its `samples` directory tree. Many of them are very simple and are designed to demonstrate a specific feature of VR Juggler or a technique to use when writing your own applications. This chapter lists the current sample applications as of this writing and gives a quick description of what you as a potential developer might find interesting in the code. Those users who just want to run applications can safely skip this chapter.

## OpenGL Applications

The OpenGL sample applications are in `$VJ_BASE_DIR/share/samples/ogl`. They are as follows:

- **Analog:** A very simple demo that allows you to test out analog input to VR Juggler. Use an analog simulator, or a real analog device (like a potentiometer hooked up to an IBOX, etc), you can move a cube from the floor to 6 feet above the floor.

### Note

This demo is well documented within the header comments. Thus, it would also be a good application for you to learn VR Juggler. It is, however, a very basic application without any interesting features.

- **Cones:** A  $10 \times 10 \times 10$  array of cones in space. The cones are created using GLU quadrics. Navigation is done using quaternions and is not recommended as a basis for navigation in most applications.
- **Cubes:** A  $10 \times 10 \times 10$  array of cubes in space. The cubes are created using OpenGL display lists. This is the best example of using display in VR Juggler and should be studied carefully. Navigation is done using quaternions and is not recommended as a basis for navigation in most applications.
- **Torus:** A very, very simple that renders a purple torus at a key point in space. That is, the torus is positioned such that in a CAVE-like device, the center of the torus is at the lower left corner of the device where the front wall, left wall, and floor meet. It is designed specifically to test VR Juggler's handling of multiple walls and projection alignment (in an extremely crude manner). Lighting of the torus is done with a flashlight on the wand but should not be used as an example of doing lighting in VR Juggler. The behavior of the flashlight is very hard to predict.
- **Wand:** A trivial program showing how to get input from a wand using digital inputs (buttons). This application is designed primarily as a tool to test wand tracking and button input in a VR system.
- **Texture:** An application demonstrating the use of OpenGL texture objects (and display lists). It includes code for loading many different texture formats: SGI RGB and RGBA, BMP, TGA (24 and 32 bit), and PCX.

### Note

This demo is well documented within the header comments. This is a good example for you to start learning VR Juggler.

- **Combo:** A collection of Cubes, Torus, and Wand that demonstrates how to cycle between three VR Juggler OpenGL applications.
- **Glove:** An example of picking up objects with a glove and navigating through an environment.
- **Simple Glove:** A very simple application that shows how to use the VR Juggler glove interface with a gesture interface.
- **Digital Glove:** Based on Simple Glove, this is another simple application that shows how to use the VR Juggler glove interface but this time with digital interfaces for each finger.

## OpenGL Performer Applications

Examples of OpenGL Performer applications can be found in `$VJ_BASE_DIR/share/samples/pf`. These are for more advanced developers who are familiar with Performer and some of the more complicated aspects of VR Juggler. There are two main programs there:

- `pfNav`: A starting point for basic VR Juggler Performer applications that need to load a model and navigate through it. Users implement their application by inheriting from a provided class, `simplePfNav`. This is a good place for Performer beginners to start because `simplePfNav` hides many of the complicated details (which actually makes that class far from simple).
- `pfConfigNav`: A more advanced example of a VR Juggler Performer application that can be given its model through a VR Juggler configuration chunk.

---

# Chapter 4. Compiling a VR Juggler Sample Program

Now that you have VR Juggler installed and you have your environment all configured, it is time for the fun to begin. No, seriously. You are now ready to compile and run VR Juggler applications, and that is the whole point, right? This chapter explains how to compile the applications provided in the directory `$VJ_BASE_DIR/share/samples`.

## Required Reading

Before reading any further, make sure you have already read the instructions on how to install VR Juggler (in Chapter 1, *Installing VR Juggler*) and on how to configure your environment (in Chapter 2, *Environment Variables*). That information will not be repeated, and it is assumed that you already know what we mean by `VJ_BASE_DIR`. You should also have a basic understanding of how `make(1)` works, but in these examples, nothing more will be necessary than typing `make` on the command line. Refer to the `make(1)` manual page for more information about it.

## Compiling an Application

There are two ways to compile VR Juggler applications: from the command line or with Microsoft Visual Studio. Compiling an application at the command line will work on all supported platforms including Win32. Using Microsoft Visual Studio will only work on Win32.

## Compiling from the Command Line

All the sample programs in `$VJ_BASE_DIR/share/samples` use the same basic steps to compile unless otherwise noted. Always refer to the top of the sample application's `Makefile` for information that may be specific to building that application. In general, though, all applications' makefiles are compatible with the standard version of the `make(1)` utility. This includes the `nmake` command provided with Visual C++.

The example used here will be the Torus application found in `$VJ_BASE_DIR/share/samples/ogl/torus`. It is an OpenGL-based application that will compile and run on all platforms supported by VR Juggler. Begin by changing into the directory `$VJ_BASE_DIR/share/samples/ogl/torus` in a command shell.

To compile Torus, simply enter the following:

```
% make
```

or on Win32 without Cygwin, enter:

```
% nmake
```

The compile process will then begin. If you have your system set up properly, it will complete with an executable `torus` file (or `torus.exe` on Win32) in the directory. Now that you have a program compiled, it is time to learn how to run it. Readers who are not using Visual Studio can skip ahead to Chapter 5, *Running a VR Juggler Sample Program*.

## Compiling Using Microsoft Visual Studio

All OpenGL sample applications are shipped with pre-configured Microsoft Visual C++ workspaces. This is done to help new users get started with compiling VR Juggler applications and to give experienced Visual Studio users a

starting place for their application development. To use the workspace for the Torus application, begin by opening the folder containing the source code and double-clicking on `torus.dsw`.

Visual C++ will open, and the Torus project will be loaded. The unexpanded class view will appear as shown in Figure 4.2 when Visual C++ first loads.

Before proceeding, the program arguments must be set. This is done using the Settings item under the Project menu. This is shown in Figure 4.3.

Selecting this item opens the “Project Settings” dialog, shown in Figure 4.4. In this window, choose the Debug tab. There will be an empty text entry field under the heading Project arguments. Here, enter the full paths to the VR Juggler configuration files that will be used to run the torus application. The `VJ_BASE_DIR` environment variable cannot be used here, unfortunately, so the *full path* to *every* file must be used. The following shows the beginning of the program arguments listing `sim.base.config`, `sim.wand.mixin.config`, and `sim.displays.config`:

Note that in this example, the `/` directory separator is used instead of `\` for the program arguments. This is not strictly required in this case, but doing it this way maintains consistency with other examples.

Once the program arguments are set up, compile the application. Under the Build menu, choose the Build `torus.exe` item as shown in Figure 4.5.

Visual C++ will compile the application, and if you have everything configured properly on your computer, the compiling will complete successfully. Once it is done, execute the torus program by choosing the Execute `torus.exe` item from the Build menu, shown below in Figure 4.6.

## Additional Notes

In some cases, VR Juggler applications have makefiles that require the use of GNU make. This is always noted in the heading comments of Makefile in the application's directory. GNU make is free software that can be downloaded from the GNU Project [<http://www.gnu.org/>]. Some operating system vendors provide pre-compiled packages for easy installation of GNU software.

---

# Chapter 5. Running a VR Juggler Sample Program

It is important to note that the same VR Juggler application can be run in simulator mode or in a full-scale VR system with no modifications. What does change is the configuration files used when starting the program. In `$VJ_BASE_DIR/share/Data/configFiles`, you can find many basic configuration files including those for running in simulator using a mouse and keyboard to simulate VR input devices and some example files based on those used for the VRAC C2 systems. In the directory, you will see some files with names containing “mixin”. These are special files that provide a specific capability not necessarily needed by all applications. They can be mixed in (hence the name) with other configuration files as needed. The configuration files found in the `configFiles` directory will be referenced in the examples provided, so be sure you know where they are.

## Required Reading

Before reading any further, make sure you have already read the instructions on how to install VR Juggler (see Chapter 1, *Installing VR Juggler*) and on how to configure your environment (see Chapter 2, *Environment Variables*). That information will not be repeated, and it is assumed that you already know what we mean by `VJ_BASE_DIR` and `LD_LIBRARY_PATH` to name two environment variables. At this point, it is also assumed that you already have compiled an application (Torus in the case of the examples provided), so you should be sure to have read about how to compile a sample VR Juggler application (in Chapter 4, *Compiling a VR Juggler Sample Program*) before proceeding.

## Running an Application in Simulator Mode

Running in simulator mode means that your input is simulated and your display windows have limited functionality. (By, “simulated input”, we mean that input is provided through windows that take keyboard and mouse input and translate that into transformations in the virtual world.) Simulator displays are limited primarily in that they cannot display stereo graphics. It is important to note that a simulator display is a special kind of VR Juggler display. Instead of basing its viewpoint on the head position of one of the users, the viewpoint is controlled by a separate “camera” that is just another positional device. The Sim Display also draws certain objects to help visualize the environment. For example, the heads of users are represented as blue spheres with gray eyes, and a wand (if present) is drawn as a green cone. Besides these common simulator objects, surface displays can be drawn. These are Juggler Displays representing projection screens or HMD viewing projections and are drawn as translucent rectangles.

As mentioned, several simulator configuration files are provided with a VR Juggler distribution. These files provide a complete simulation of an immersive environment. Please note that this documentation reflects the state of the configuration files at the time the documentation was written. For more information about the configuration files and how to view or modify the configuration, refer to the *VjControl Guide*. (Using VjControl is the best way to find out how a specific configuration file is set up.) The configuration files of interest for simulator mode are as follows:

- `sim.base.config` - The basic configuration file needed by all applications when run in simulator mode. It defines commonly used VR Juggler concepts that are beyond the scope of this particular book. It also defines simulated head movement using the keyboard. For now, it is sufficient to know that it is required to run the sample applications in simulator mode.
- `sim.displays.config` - The basic simulator display configuration file needed by all applications when run in simulator mode. This file defines the display windows where the rendering magic occurs. Two simulator display windows are configured by this file: a small one that is active by default and a larger one that is inactive initially.
- `sim.analog.wandmixin.config` - A “mix-in” configuration file that defines simulated analog input using the keyboard. This is only required for applications where analog input is used and needs to be simulated

when in simulator mode.

- `sim.analog.mixin.config` - This version of the analog simulator opens its own window. See the previous file (`sim.analog.wandmixin.config`) for other details.
- `sim.c6displays.mixin.config` - A “mix-in” configuration file that defines the surface displays of VRAC's C6. This is not required for any application but can be used to test opening multiple display windows (both surface and simulator) before running in a multi-screen VR system.
- `sim.digital.glove.mixin.config` - A “mix-in” configuration file that defines simulated digital glove input using the keyboard. This is only required for applications where digital glove input is used and needs to be simulated when in simulator mode.
- `sim.glove.mixin.config` - A “mix-in” configuration file that defines simulated gesture-based glove input using the keyboard. This is only required for applications where gesture-based glove input is used and needs to be simulated when in simulator mode.
- `sim.wand.mixin.config` - A “mix-in” configuration file that defines simulated wand input using the mouse. This is only required for applications where wand input is used and needs to be simulated when in simulator mode.

For the Torus application, we only need the base configuration file, the displays configuration file and the wand mix-in configuration file.

Now it is time to run the application—finally! Make sure that all your environment variables are set properly before trying to start the application. Once you are ready, specify the name of the application and all the configuration files it needs. An example of this is:

```
% torus sim.base.config sim.displays.config sim.wand.mixin.config
```

You will notice that no paths are specified for finding the three configuration files. This is intentional to shorten the command line given in the example. In general, you always have to give the full path to the files. VR Juggler 1.0 does not have any logic for defining a path that would contain configuration files<sup>1</sup>. Beginning users will typically want to reference the example configuration files in `$VJ_BASE_DIR/share/Data/configFiles`. As you get more comfortable with VR Juggler and its configuration system, you may want to make your own modified files and put them in the directory `$HOME/.vjconfig`. To simplify running applications, you may want to make a shell script (or batch file as appropriate) that does all the work of passing configuration files and common command-line arguments. For now, though, use the path `$VJ_BASE_DIR/share/Data/configFiles` for each of the configuration files you pass on the command line.

As the application starts, you will see a plethora of output (more or less depending on how you have `$VJ_DEBUG_NFY_LEVEL` and `$VJ_DEBUG_CATEGORIES` set), and then one moderately sized simulator display window will open on the left side of your screen while three blank keyboard input windows open on the right side of your screen. The display window will be titled “SimWindow1”, and the keyboard input windows will be titled “Head Keyboard”, “Sim View Cameras Control” and “Wand Keyboard” (in order from the top of the display to the bottom). Do not worry that the keyboard windows are black—that is normal. The display window will have a purple torus, a cyan sphere and a green cone. The torus is what you have come to see; the sphere is the simulated user's head; and the cone is the simulated user's wand. In Figure 5.1, we show what this looks on an IRIX 6.5 desktop for comparison with what you are seeing. Note that the head and wand are only rendered in the simulator windows. They are present because head and wand input are being simulated, and it is typically quite helpful to see the results of that simulated input.

So now you are probably wondering what you can do with this fancy application. Interaction is done through the keyboard windows on the right side of your desktop. Moving the head is done with the keyboard in “Head Keyboard”; moving the camera is done with the keyboard in “Sim View Cameras Control”; and moving the wand is done with the mouse in “Wand Keyboard”. As noted above, this information is current as of this writing. For in#

---

<sup>1</sup>This has been fixed for VR Juggler 1.1 and beyond.

formation on how to verify these settings and to view the current configuration, refer to the *VjControl Guide*. The following list of tables provides all the keyboard and mouse controls for the simulator when using these particular configuration files. Note that it is possible to reconfigure the simulator to suit your preferences. This is provided mainly for those who just want something that works now.

**Table 5.1. Moving the simulated head**

<b>Transformation</b>	<b>Key Press</b>
Move head backward	2 on keypad
Move head left	4 on keypad
Move head right	6 on keypad
Move head forward	8 on keypad
Move head down	7 on keypad
Move head up	9 on keypad
Turn head up	CTRL+2 on keypad
Turn head left	CTRL+4 on keypad
Turn head right	CTRL+6 on keypad
Turn head down	CTRL+8 on keypad
Rotate head clockwise	1 on keypad
Rotate head counter-clockwise	3 on keypad

**Table 5.2. Moving the simulated wand**

<b>Transformation</b>	<b>Mouse Input/Key Press</b>
Move wand backward	ALT+move mouse backward
Move wand forward	ALT+move mouse forward
Move wand left	CTRL+move mouse left
Move wand right	CTRL+move mouse right
Move wand up	CTRL+move mouse forward
Move wand down	CTRL+move mouse backward
Rotate wand left	SHIFT+move mouse left
Rotate wand right	SHIFT+move mouse right
Rotate wand up	SHIFT+move mouse forward
Rotate wand down	SHIFT+move mouse backward
Rotate wand clockwise	Right arrow
Rotate wand counter-clockwise	Left arrow
Wand button #1	Left mouse button
Wand button #2	Middle mouse button
Wand button #3	Right mouse button
Wand button #4	4
Wand button #5	5
Wand button #6	6

**Table 5.3. Moving the camera**

Transformation	Key Press
Move camera backward	2 on keypad
Move camera left	4 on keypad
Move camera right	6 on keypad
Move camera forward	8 on keypad
Move camera down	7 on keypad
Move camera up	9 on keypad
Turn camera up	CTRL+2 on keypad
Turn camera left	CTRL+4 on keypad
Turn camera right	CTRL+6 on keypad
Turn camera down	CTRL+8 on keypad
Rotate camera clockwise	1 on keypad
Rotate camera counter-clockwise	3 on keypad

Before continuing on to running an application in a full-scale VR system, we provide two asides: using a simulated glove and using a simulated analog device. The examples provided thus far have not discussed this because the information was not relevant to the particular sample application being used. Knowing how to use these simulated devices is important, however, and it is treated separately as a reference for your future endeavors in running VR Juggler applications.

## Using a Simulated Glove

If you include the `sim.glove.mixin.config` file, your application will also have access to a simulated glove, with position and gesture inputs. The glove is controlled by a window titled “Glove Keyboard”. This window lets you control the glove position and selected gesture. Movement control of the glove uses the mouse and is the same as that of the wand. The mouse buttons are used to select gestures. The mapping of the gesture numbers to actual hand positions is controlled by the “training file” for the sim glove. The default training file is `$VJ_BASE_DIR/share/Data/gesture/simpleSimGestures.dat`.

## Using a Simulated Analog Device

If you include the `sim.analog.wandmixin.config` file, your application will also have access to a set of four analog devices (devices with a value in a range from 0.0 to 1.0). The analog devices are also controlled using the “Wand Keyboard” window which means that their configuration file requires the wand configuration file.

### Note

A separate file, `sim.analog.mixin.config`, is provided for analog input from a separate simulator window.

The key presses used for controlling the analog devices are listed in Table 5.4.

**Table 5.4. Analog Device Simulator Keys**

Analog Device Action	Key Press
VJAnalog0 increase	Q

---

Analog Device Action	Key Press
VJAnalog0 decrease	A
VJAnalog1 increase	W
VJAnalog1 decrease	S
VJAnalog2 increase	E
VJAnalog2 decrease	D
VJAnalog3 increase	R
VJAnalog3 decrease	F

## Running an Application in a VR System

Running an application full-scale in a VR system is more complicated than running in simulator mode. The reason for this is that VR systems tend to differ in configuration and in available hardware. VR Juggler is flexible enough to handle most any configuration you throw at it, but those configurations need to be put together first. Examples of configuration files used in VRAC's C2 system are provided, and they are used in this documentation. It should be noted, however, that for any particular system, custom configuration files will probably have to be written. The idea behind this section is to provide a basic understanding of what is needed to get started with running in a VR system.

The example configuration files in the directory `$VJ_BASE_DIR/share/Data/configFiles` modeled after those used for VRAC's C2 system are as follows:

- `C2.base.config` - The basic configuration file needed by all applications when run in the C2. It defines commonly used VR Juggler concepts that are beyond the scope of this particular book.
- `C2.displays.config` - The basic display configuration file needed to run with all four walls active and rendering stereo graphics. This defines only the four surface displays to be opened.
- `C2.flock.config` - The Ascension Flock of Birds configuration file that defines which bird provides input for the head and for the wand.
- `C2.ibox_buttons.config` - The IBOX configuration file that handles the digital wand button inputs.
- `C2.mono.displays.config` - The same configuration as `C2.displays.config` except that the walls are opened to render mono graphics.

Running the application is the same as in simulator mode except that the configuration files given on the command line are different. For example, to run Torus in the C2 with stereo graphics, the following command would be used:

```
% torus C2.base.config C2.displays.config C2.flock.config C2.ibox_buttons.config
```