

VR Juggler – An Open Source Platform for Virtual Reality Applications

Carolina Cruz-Neira, Allen Bierbaum, Patrick Hartling, Christopher Just, Kevin Meinert
Virtual Reality Applications Center
Iowa State University
{carolina, allenb, patrick, cjust, kevn}@vrac.iastate.edu

Abstract

This paper describes VR Juggler, an Open Source platform used to develop and run virtual reality applications. We emphasize VR Juggler's ability to provide a uniform VR application environment and to allow extendibility to new devices without affecting existing applications. These features enable VR applications to evolve along side other technologies with minimal or no new developmental efforts.

1 Introduction

The use of virtual reality (VR) to interact with data is becoming routine practice in many domains. One domain where VR is being used currently is engineering design, analysis and evaluation. This domain is pushing VR technology to become more accessible and easier to implement. Engineers and scientists are concerned with their problem areas, and in order to fully accept the use of VR into their workflow, they need tools that simplify the use of the complex VR systems¹ they utilize.

There have been many efforts to create tools for the development and execution of VR applications. Some tools are developed and maintained by academic groups while others are created directly in commercial settings. Many of the commercial tools originally started out in a research setting and then were migrated to a commercial venue. Most of these tools focus on specific hardware technologies with customized display and interaction methods thus making them hard to expand to new methods and configurations. Examples of existing tools include CAVELib, WorldToolKit, Avango, and Lightning.

The CAVELib [1] was designed as part of the initial work with the CAVE system, and it was

expanded to be a toolkit to develop applications for multiscreen systems using a procedural programming approach. Its design does not separate the application from the library, so as the CAVELib expands, applications need to continue being updated to keep up with the new features.

WorldToolkit [2] is designed to support a variety of VR technologies, however, the programmers must specify in the code the particular devices being used for the application. This means that if, in the future, the application needs to run in a different VR system, the application code needs to be modified to accommodate the new hardware setup.

Avango [3] and Lightning [4] are two other tools for creating immersive applications. Avango was developed at the German National Research Center for Information Technology, GMD; Lightning was developed at the Fraunhofer IAO. Both packages are based on OpenGL Performer, thus the applications are limited to SGI systems² only.

All of these packages are now commercial tools, which limits VR users to the features available. As the technology advances, users must wait for the tools to be updated by the supporting companies. If the company does not or cannot provide desired updates, then there is no clear way to migrate current applications to the new technologies.

2 VR Juggler Design

VR Juggler is an object-oriented software system for the development and execution of virtual reality applications. Its architecture makes use of several software engineering techniques including design patterns [6]. The techniques employed aid in encapsulating the many elements of a VR system. The goal of VR

¹ Within this paper, we use the term “VR system” to refer to the hardware that makes up a VR environment. When we refer to the software that manages the hardware, we will use the term “VR software system”.

² OpenGL Performer is now also available for the Linux operating system, so both packages may be available for PC platforms

Juggler is to introduce a software layer between the application and the hardware to provide a hardware-independent software system. This layer contains a set of well-defined interfaces to the VR components based on their functionality. For example, a 6-degrees-of-freedom tracking system is handled through a positional device interface. In this way, if the application needs to use a different tracking system, it will not need to know about the change. As long as the new device is controlled through the positional interface, the application would not need any code changes. With this design, applications are concerned only with the type of information that they need to receive from and send to the VR system. VR Juggler takes care of converting that information into the underlying representation needed for each particular device and hardware component.

1 Technical Design

To achieve hardware independence, the VR Juggler architecture is based on a microkernel, which uses a set of managers, each one dedicated to specific tasks. The microkernel's main responsibility is to be the mediator for the managers. This involves the following:

- Sustain the interactive performance of the system and the applications
- Coordinate interactions among the managers
- Manage the communications between the managers and the applications
- Keep the system components synchronized.
- Handle runtime reconfiguration of the VR system
- Direct the execution of multiple applications

Figure 1 shows a diagram of the VR Juggler kernel, the current managers and the relationships between them.

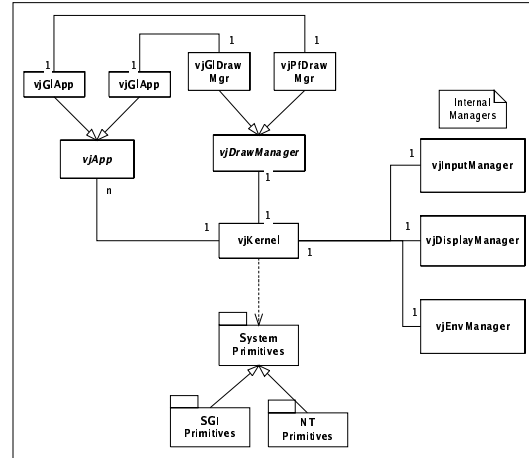


Figure 1: VR Juggler design

1 VR Juggler Managers

The managers in VR Juggler control specific components of the VR system. The managers are separated into two categories: internal managers and external managers. The internal managers handle the core functionality beyond the scope of the kernel; the external managers deal with the interfaces to the application.

The Draw Manager is the external manager that provides the interface for rendering. The Draw Manager has two layers. The top layer specifies the common interface for all rendering tasks. This includes initialization of the graphics environment, definition of viewing parameters, specification of light sources and so on. The second layer extends the top layer to include customizations required for specific graphics subsystems. For example, VR Juggler supports several graphics application programming interfaces (APIs) for the development of applications. Each of these graphics APIs has an associated Draw Manager that contains the API-specific interface extensions and base layer implementation. Currently, VR Juggler includes Draw Managers supporting OpenGL, OpenGL Performer, and OpenSceneGraph.

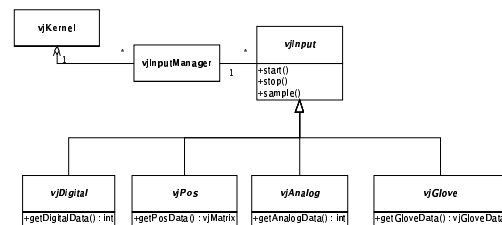


Figure 2: Input manager structure

The Input Manager is the manager that controls all input devices in the system. Figure 2 shows a diagram of the Input Manager structure. The Input Manager deals with devices based upon the kind of input that an application can receive. Built-in types that VR Juggler supports include positional data (tracking), digital data (buttons), gesture data (gloves), analog data (joysticks), and others. At its lowest level, the Input Manager accesses all devices through device-specific drivers. All references to the real device types are hidden by the Input Manager. When retrieving data from a device, the Input Manager transforms the data into one of the basic types for use by the application.

The primary advantage of this structure is that applications are independent of any specific physical device because they only deal with the basic input types. Since applications are not tied to the devices of a specific VR system, device substitution, reconfiguration, and VR system migration can occur without code modification.

VR Juggler can be configured to simulate a VR system on the desktop. This allows VR Juggler to be used in a traditional desktop setup. For example, a positional input, conceptually designed for 6-degree-of-freedom tracking devices, can be mapped to a combination of mouse and keyboard through the use of a simulated positional device. The mouse and the keyboard generate the 3D position and orientation information that the application expects; therefore the application does not need to be notified of this new mapping.

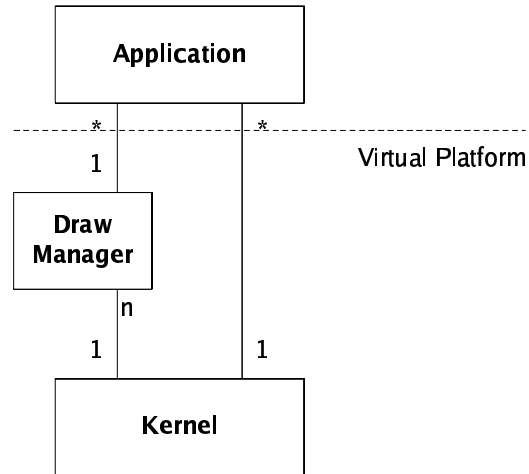


Figure 3: Virtual Platform

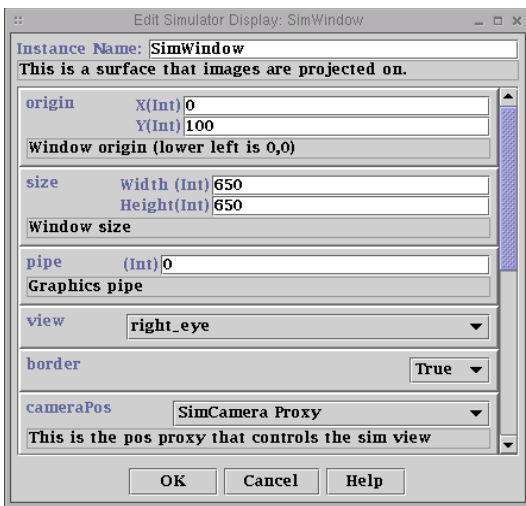
Figure 3 shows how applications interact with the VR Juggler managers. Most of the communications go through the kernel, which as we mentioned above, handles the information flow within the system. The only manager that applications need to have direct interaction with is the Draw Manager.

The VR Juggler Draw Manager allows applications to directly interact with the underlying graphics APIs. Most graphics APIs make it very difficult to completely isolate the application from them. Some use scenegraphs, others require applications to handle their geometry directly. To develop applications with advanced features such as collision detection, complex lighting models, object manipulation, and computational updates to the geometry, applications must use graphics API calls. Any attempts to abstract these features result in limiting the capabilities available to applications and restricting the flexibility available to application developers.

VR Juggler acts as a virtual platform, shielding applications from changes to the underlying system. For example, new managers can be added or removed without impacting user code. Because VR Juggler is Open Source, the managers' design allows other groups in the VR community to expand the manager structure to accommodate for the particular needs at their sites. This allows applications to evolve transparently with the technology and prevents them from becoming obsolete as new devices and systems become available.

2 VR Juggler Parameterization

VR Juggler has been parameterized to accommodate the many possible configurations of devices, components, and software APIs. The active configuration specifies what parts of the VR Juggler system are needed for an application to run. The information flowing through the VR Juggler system has been grouped into “chunks” where each chunk contains the parameters that are related functionally. For example, there is a “simulator display chunk”, which allows users to emulate an immersive display in a desktop environment. Figure 4 shows a view of chunk parameters being edited in a graphical user interface (GUI). The application user can specify the size of the window, the window’s location in the desktop, and some viewing parameters.



3 Figure 4: Simulator chunk

4 Platform Independence

Through the use of a microkernel design, managers, and parameterization, VR Juggler successfully isolates applications from the underlying software and hardware. The internal and low-level external managers can be implemented to match different operating systems and hardware architectures while preserving a consistent external software interface to applications. With this capability, VR Juggler scales across multiple platforms and VR systems. One example of this scalability is

that developers can create applications in a simulator using the resources available on their desktop PCs without needing access to the actual VR system until the application is near completion. This enables a more efficient use of the available resources at VR sites, allowing for the development of VR applications on standard PCs and migrating the application to the actual VR system with minimum developer effort.

3 Using VR Juggler

Developing and running applications with VR Juggler requires an understanding of how applications operate within its framework. Applications are executed by the VR Juggler virtual platform. The application is treated as another object that the kernel must handle. All configurable elements of the application and the platform are parameterized and are configured by the “chunks” of information introduced in Section 2.3.

5 Application Development

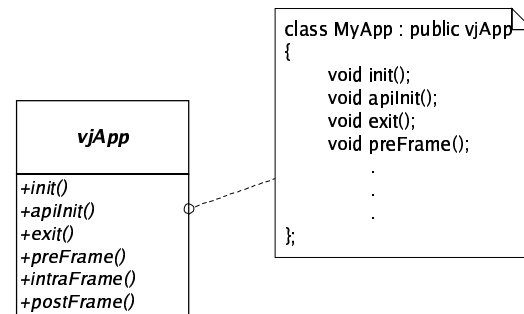


Figure 5: vjApp interface

All VR Juggler applications are developed as C++ objects known as “application objects”. These application objects are handled directly by the kernel through an interface named **vjApp**, shown in Figure 5. This interface specifies the actions an application needs to define in order to present a virtual world to users. These actions include any computations to process user inputs, to update virtual element behavior, to detect collisions with virtual objects, and to provide other characteristics of the virtual space. Developers have full control of the graphics API, so they can set rendering methods, add sounds, manage the geometry, and incorporate any other visual feature required by the application.

An advantage of using kernel-managed application objects is that VR Juggler can support multiple applications running simultaneously. We can think of this feature similar to the document concept that most word processors provide. We can start the word processing environment once, and from within that environment, we can maintain multiple documents and switch between them without having to restart.

This feature is particularly useful for VR installations that are subject to many visitors and demonstrations. All the showcased applications can be integrated into a single VR Juggler environment. This integration saves a significant amount of time because there is no need to shutdown and restart the VR system for each application. In a later section, we describe our switcher environment, an implementation of this feature developed in our research center.

6 Running VR Juggler

Once applications are developed, they can run in a variety of VR settings ranging from single-computer desktop environments to multi-screen, supercomputer-driven, high-performance VR systems. The same application code will scale among all these systems through the data given in the information chunks. Running an application requires configuring the components of the VR Juggler system and the application object or objects being run by the system.

7 Configuring VR Juggler

To simplify the creation and editing of the information chunks, VR Juggler provides a configuration editor written in Java called VjControl. VjControl presents a GUI to the chunks that are grouped by functionality. Users can edit the information chunks without being concerned with the internal format or valid options. VjControl takes care of those details, presenting users with the possible options or ranges for all the available parameters of a given configuration. Figure 6 shows a typical configuration window for VjControl. This window shows the available configuration chunks for a particular application set.

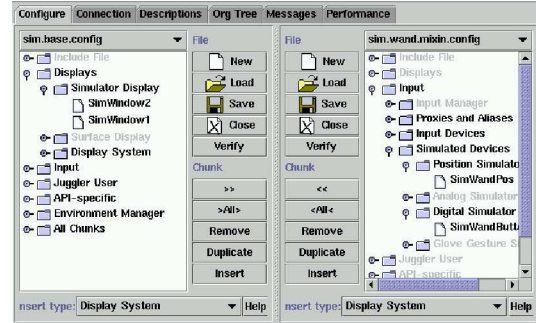


Figure 6: VjControl

Figure 7 shows a VjControl window that specifies the parameters for the desktop simulator chunk. Users can map key presses on the keyboard to device actions. In this example, the user is defining key press mappings for head motion actions such as forward, backward, up and down.

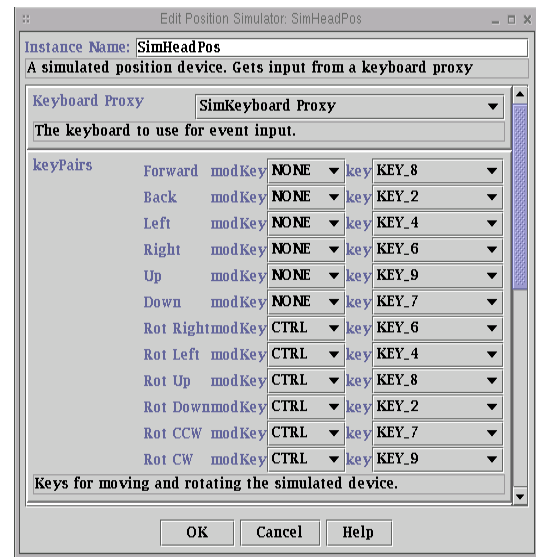


Figure 7: Example of a VjControl configuration chunk window

VjControl can be run standalone to edit a configuration file for an application, or it can be run with a live connection to the VR Juggler kernel. When VjControl is connected to the VR Juggler kernel while an application is running, it can provide live performance monitoring of the applications and allows for reconfiguring the application at run time.

By analyzing the live performance data provided by VjControl, users can identify where performance bottlenecks are located in the code.

It can also help to monitor the VR Juggler system performance.

Sometimes, applications may need to be reconfigured at run time. A component of the VR system may fail and need to be replaced by another component. An application may need to run in a different display, or a new interaction device may need to be added. When these situations arise in conventional VR systems, users need to terminate the VR software, reconfigure the application, and restart the entire environment. VR Juggler allows the system to continue running while using VjControl to simply change the configuration parameters of the system component that needs to be changed. It also allows for remote application-level configuration such as adding additional geometry into the virtual world.

4 VR Juggler Applications

VR Juggler has been used in a variety of application domains. This section covers a small set of representative applications developed at the Virtual Reality Applications Center (VRAC) in the past year. Detailed coverage of VR Juggler applications is found at the VRAC web site³.

8 The Application Switcher

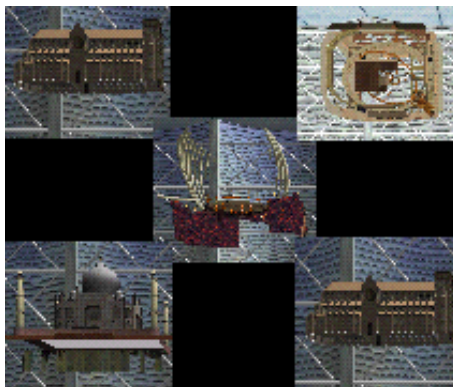


Figure 8: Application Switcher

We have built a VR Juggler environment that allows us to interactively switch among any number of applications at run time. The switcher has a VR Juggler core that manages each application object based on input from the wand available in our VR systems. Applications are presented to the user sequentially through a 3D

³ <http://www.vrac.iastate.edu>

miniature representation of their content, as shown in Figure 8. Currently there are 12 different applications integrated into the switcher. As new applications are developed, we continue integrating them into the switcher to facilitate their use by all the members of our lab.

9 The Nexus

Terry Welsh and Jeremy Eccles developed the Nexus application to explore the gravity-free environment provided by immersive systems shown in Figure 9. Anchored somewhere in the depths of space, the Nexus treats its viewers to an exploration of impossible gothic architecture and giant twisting machinery. Through VR Juggler, the Nexus has been shown in projection-based VR systems, head-mounted displays, and large audience theaters, with no additional programming efforts on the developers.

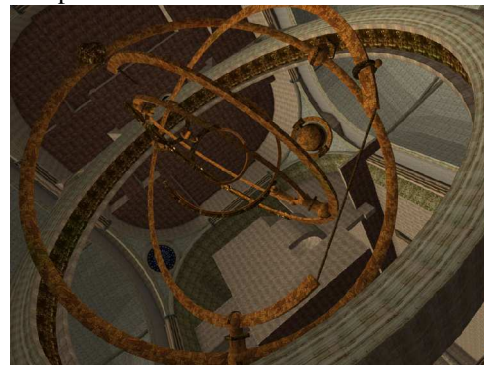


Figure 9: The Nexus

10 Virtual Tornadoic Thunderstorm

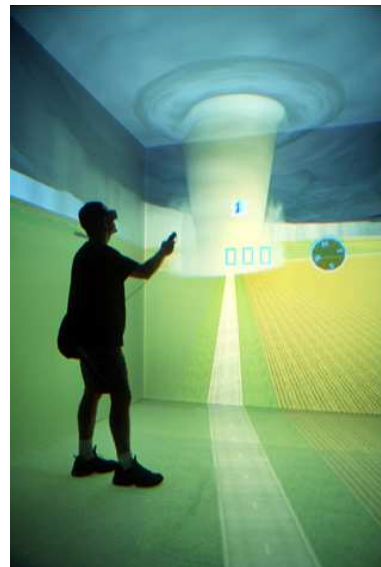


Figure 10: Virtual tornado

Shown in Figure 10, the tornado application was created to facilitate improved student understanding of both the atmospheric dynamics involved and the visual characteristics of a tornadic storm. This project has resulted in a new visual tool that will allow student-centered learning in this challenging area of meteorology. The flexibility offered by VR Juggler has allowed its use in introductory courses taken by non-majors and upper-level major meteorology courses using standard PC desktop environments. We have also been able to use it in advanced seminars using our fully immersive VR systems to navigate throughout the thunderstorm in a virtual tornado chase.

5 Conclusions

VR Juggler has been a very successful software system for the development and execution of scalable immersive applications. There is a growing community of users around the world and, because of its Open Source free availability, there are also a growing number of contributors expanding its capabilities. Several commercial software companies have started to release products that are VR Juggler compatible.

6 Acknowledgments

The authors would like to thank the VR Juggler users at the Virtual Reality Applications Center for their useful feedback as VR Juggler continues growing.

7 References

[1] Cruz-Neira, C., *Virtual Reality Based on Multiple Projection Screens: The CAVE and its Applications to Computational Science and Engineering*, PhD Dissertation, University of Illinois at Chicago, May 1995.

[2] <http://www.sense8.com>

[3] H. Tramberend, "Avocado: A Distributed Virtual Reality Framework", in *Proceedings IEEE Virtual Reality '99 Conference*, Houston, March 1999.
http://imk.gmd.de/docs/ww/ve/projects/proj1_2.mhtml

[4] Lightning Homepage,
<http://vr.iao.fhg.de/vr/projects/Lightning/OVERVIEWen.html>

[5] Bierbaum, A., *VR Juggler: A Virtual Platform for Virtual Reality Application Development*, M.S. Thesis, Iowa State University, 2000.

[6] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. *Pattern-Oriented Software Architecture: A System of Patterns*, Addison-Wesley, 1996.