

VR Juggler: A Framework for Virtual Reality Development

Immersive Projection Technology Workshop

Christopher Just, Allen Bierbaum, Albert Baker, Carolina Cruz-Neira

(cjust@icemt.iastate.edu, allenb@icemt.iastate.edu, baker@cs.iastate.edu, cruz@iastate.edu)

Iowa Center for Emerging Manufacturing Technology

Iowa State University

Abstract

In this paper, we outline the structure and progress of VR Juggler, a development environment for Virtual Reality (VR) applications. The paper explains what VR development environments are used for and points out the main strengths in VR Juggler as compared to other currently available libraries. The paper then goes on to describe the VR Juggler development environment in detail.

Introduction

The VR Juggler project has been underway at the Iowa Center for Emerging Manufacturing Technology (ICEMT) since the beginning of 1997. ICEMT works with a wide variety of academic and industrial users, helping them apply Virtual Reality (VR) technology to their problems. Through this experience, we have identified certain significant technical barriers to the development of effective and extensible VR applications. These barriers have led us to develop a framework for the next generation of virtual reality applications. This framework, VR Juggler, simplifies application development while

providing hardware flexibility and scalability to applications and retaining the utmost in performance.

VR Juggler is designed to “juggle” any combination of devices, graphics APIs, and platforms that users may want to use. VR Juggler provides researcher with a framework to create any application they desire without having to worry about the low-level details of VR.

Why VR Development Environments are Needed

Simplify application development

The overwhelming reason for using a VR development environment is to make it simple to write VR applications. VR applications are inherently complex: developers must worry about machine details, graphics rendering, device interaction, and more. Creating an application from scratch would overwhelm any but the most skilled and daring programmers. Development environments for VR applications work in the same way as windowing or networking toolkits: they hide the common complexities behind easily understood interfaces and abstractions. Therefore, the developer can worry less about the details of devices, displays,

and so on, and concentrate on the business at hand: creating an effective, interactive, virtual world.

VR development environments need to be easy to use so that people other than dedicated VR developers can use them. VR is a tool that allows insight into the problems of many different fields. Unfortunately, it has been very difficult in the past for experts in other fields to make use of VR technology. These users may be very knowledgeable in their own fields, but often are not skilled computer programmers, and are only rarely familiar with the sort of complexities required for VR. In order for these people to benefit from VR, they must have development systems that are easy to learn and that will enable them to produce highly interactive and visually complex results.

Rapid development

While interest in VR applications has increased in recent years, developing them remains difficult. This is because users constantly want more features and more abilities. This has made it much more difficult to develop VR applications in a reasonable time frame. Therefore, one of the reasons for having a VR development environment is to decrease development time and costs. Rapid development environments will allow for faster application development. An added benefit is that normally if the development environment enables rapid development, the applications developed are based on components, which can lead to applications that are more robust.

Unified interface

Common interfaces make applications easier to learn and use. This has been demonstrated by progress in graphical user interfaces (GUIs), where a common language of symbols and concepts – buttons, windows, menus, and pointers – has made learning new applications a process that relies more on intuition and applied experience than on thick manuals. This was only able to happen because application programmers were given development tools that gave them access to those common GUI elements. The same adoption of common user interfaces can happen in VR, but only if development environments are created which give developers the necessary tools. The result will be VR applications that are easy to learn, and users who will be able to

develop an intuitive understanding of how an application “ought to” respond to their input.

Extensibility and portability

By using a common VR development environment, it is possible to extend applications to use new VR devices. It is also possible for VR applications to be ported to other systems.

VR Juggler in Brief

VR Juggler is an application framework and set of C++ classes for writing virtual reality applications. It is designed to allow the developer direct access to various graphics APIs for maximum control over applications, while still providing a generalized, easy to understand view of displays as well as input and output devices.

VR Juggler supports a variety of virtual reality hardware, including projection systems like the CAVE™ and C2, projection tables, simple workstation monitor displays, and traditional head-mounted displays. Our tracking classes include support for the Ascension Technologies Flock of Birds™, Logitech 3D mouse, and Fakespace BOOM™. Our input devices also include Immersion boxes, the Virtual Technologies CyberGlove™, and the Fakespace PINCH™ gloves. A set of generic device interfaces allows new devices to be added easily, and simulators exist to replace any unavailable devices.

VR Juggler will include support for distributed applications. The processes that make up an application can be divided among two or more machines, to take advantage of additional processing power, input devices, and graphics systems. We are currently exploring techniques to make this distribution as transparent to the developer as possible.

Benefits of Using VR Juggler

Development of virtual reality applications has always been overly difficult. Current VR development environments require the application

developer to have a deep understanding of system-level programming. One of our goals is to open the field of VR development to researchers and artists without requiring them to learn all the usual technical minutia. VR Juggler is designed to provide a level of abstraction above the complex inter-workings of distributed computing, shared memory, multiprocessing, and device I/O. At the same time, the framework does not sacrifice performance or flexibility.

Hardware abstraction

We are trying to simplify and abstract use of the wide range of VR hardware in use today. Many current VR development libraries, such as the CAVE™ library [Cruz-Neira95] and WorldToolkit [Sense8-97] have interfaces that rely on or were designed around specific underlying devices. For example in WorldToolkit, the function calls for using a head-mounted display (HMD) are different from those used for a CAVE™, and an application designed for one must be modified to use the other. VR Juggler shields application programmers from the specifics of device hardware. The same applications run in a CAVE™ or on a head-mounted display with no change. Input devices are given common interfaces; data from two functionally similar devices will look the same to the application, regardless of the hardware details. In implementing this interface, we have eliminated hardcoded restrictions wherever possible. For example, there are no fixed addresses or maximum numbers for trackers or other input devices.

Performance

Maximizing performance was one of the top priorities in VR Juggler's design. Immersive environments need to maintain a high frame rate (15 Hz or better) and respond quickly to user input. Poor performance can lead to unpleasant side effects including disorientation and motion sickness [Kawalsky93].

VR Juggler is designed to take the best advantage of all system resources. We have worked to give it the smallest possible performance footprint, and to let it use the hardware at maximum efficiency.

Performance tuning

Promising high performance is one thing, but to really deliver, the development environment must provide tools that will allow the developer to fine-tune an application's performance. The first and most important step is to be able to accurately and precisely measure the performance of various parts of the system.

VR Juggler's design includes built-in support for collecting performance data. Users of Juggler-based applications can view performance metrics at run-time and also dynamically reconfigure the system.

VR Juggler includes a GUI-based program called the Environment Manager, which can monitor and graph performance data. For example, the simplest metric might be a graph showing how the frame rate of the application varies over time. Watching this display while navigating in the virtual world might indicate places where performance is compromised because of the amount of geometry that needs to be drawn. Another display might show the latency between when the head tracker returns new positional data and when that data is reflected in the display. A third example would be a display to show the relative load among different processors of an application running on a multiprocessor architecture.

In addition to displaying performance measurement data, the Environment Manager GUI can be used to dynamically reconfigure a running Juggler application. In the above example, after displaying the processor loads, the user might choose to redistribute processes himself, even dynamically moving certain processes to different machines. This is generally transparent to running applications, which might only demonstrate a momentary graphics glitch while their configuration is being rearranged.

Run time flexibility

In the course of this work, the run-time environment for Juggler applications has been made as fluid and dynamic as possible. Most VR systems are static: once an application is started, the configuration cannot be changed without stopping the program. VR Juggler applications are dynamic: every element of the system can be reconfigured, or even stopped and restarted, without disturbing a running

application. We have already suggested how this ability might be used to tune application performance, but that is only one of many uses for this powerful feature. Juggler's run-time flexibility allows developers to switch input devices, change display properties, or even restart "dead" trackers, all without restarting the application itself. The system abstraction is designed in such a way that the application is never aware of the foundations shifting beneath it.

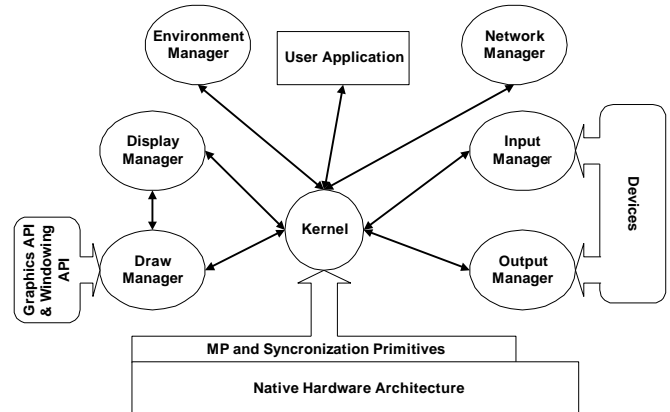
Cross-Platform

With the wide range of machines available today, a VR development environment that is not cross-platform is at a major disadvantage. For example, developers may want to be able to write applications on the Windows NT™ machines on their desktops and seamlessly run the application on SGI Reality Monsters controlling CAVEs. VR Juggler encourages such portability. It is designed upon abstract system primitives using standard C++. This allows VR Juggler, and applications using it, to be easily ported to new platforms. VR Juggler currently runs on the Silicon Graphics, Inc. (SGI) platform, and is being ported to Hewlett Packard (HP) and Windows NT.

Extensibility

Virtual reality is a rapidly changing field, and as such there are new devices and interfaces being developed every day. To keep from becoming obsolete, a VR development environment must be easily extended. With this in mind, we have made extensibility a priority for VR Juggler. Juggler's hierarchy of devices can be added to with relative ease. For example, all tracking devices are controlled by subclasses of the Position class. Adding support for a new tracker merely involves implementing a new subclass of Position. The new tracker device's interface is identical to all the others – it is the standard interface defined by Position. Therefore, once the new device is added, it can be used by existing applications without any coding changes. Because of the hardware abstraction in its design, VR Juggler can be extended to support new devices without ever forcing changes to application.

VR Juggler Description



VR Juggler is an object-oriented system, designed to take advantage of the power and flexibility of C++. Most of the functionality of Juggler has been divided into a handful of distinct and well-defined components called Manager. Each Manager encapsulates and hides a specific set of system details. For example, there is a set of Managers to deal with different windowing systems, and another to handle aspects of drawing with different graphics systems. A single Manager oversees all input devices, while another control output devices. Other Managers control networking and system configuration. The Managers are united and controlled by a small kernel, which brokers all communication in the system.

Input Manager

Every input device in the system, including trackers, joysticks, mice, and keyboards, is controlled by the *Input Manager*. Input devices are divided into several broad categories. Tracking systems are represented by Position devices, which provide one or more sets of position and orientation data. There are also Analog devices (a steering wheel is a good example) and Digital devices (like the buttons on a joystick or mouse).

A layer of abstraction is maintained between the application and the actual device processes. When an application requests access to a particular device (e.g. "Give me the position input for the user's head tracking"), it actually receives a proxy. The application talks to the proxy, and the proxy keeps

track of the most recent data from the device process. Proxies are one of the mechanisms that allow running applications to be reconfigured without any disruption. The device hidden by an input proxy can be stopped, started, or replaced with another of the same basic type. There are also simulators for each of the major types of input. For example, this lets us replace a faulty tracker with a keyboard-based simulator. The application never sees the changes that occur under the cover of the input proxies

Output Manager

The *Output Manager* works in much the same way as the Input Manager except the data is given to the devices to output. Examples of output are sound, LCD displays, and motion bases.

Similar to the Input Manager, the Output Manager will use proxies to hide the implementation details of the output device. VR Juggler will present a simple uniform interface to each type of output (sound, digital, etc). This interface can then be used by an application without the application having direct access to the actual output device.

One of the first output devices to be supported by VR Juggler is an LCD screen for displaying text output. We are also adding capabilities for audio output.

Draw Managers

There are numerous Draw Managers. Each one encapsulates a specific graphics Application Programming Interface (API). Currently, Draw Managers exist to support OpenGL and Iris Performer.

An application instantiates only one Draw Manager, based on which graphics API it is written for. The Draw Manager handles various details of the graphics operations. It sets up viewing parameters for each frame drawn, calls drawing functions defined by the application, and handles the details of stereo viewing, buffer swapping, and so on. The Draw Manager can be configured to set up viewing parameters for a variety of different display systems, from CAVEs to HMDs (the distinctions of which are generally hidden from the application developer).

The Draw Manager also manages whatever API-specific windowing and graphic system initialization are required. For example, the Draw Manager is responsible for the GLX library calls that are required to initialize OpenGL using the X Window System.

VR Juggler can be extended to support additional graphics APIs by creating new Draw Managers (and creating new application framework classes to instantiate them). The rest of the library adapts seamlessly.

Display Manager

Display Managers encapsulate all information about graphics windows, displays, and windowing APIs. As with the Draw Managers, there are several kinds of Display Managers, but only one is instantiated at any time. Whereas the choice of Draw Managers was based upon the application, which Display Manager is chosen depends on the windowing system used by the underlying architecture. For example, there are separate display managers for Windows and for Unix machines using X. Unlike the Draw Manager, the application never has direct contact with the Display Manager.

The information the Display Manager encapsulates includes the size, location, graphics pipeline, and viewing parameters to be used. The exact details depend on the underlying windowing system. For example, The X Display Manager understands about X display names, like "titanic:0.0".

The Display Manager uses this information to create windows or open displays, change video modes, and so on, as necessary. Control is then turned over to the Draw Manager, which creates rendering contexts and then handles the actual drawing.

The Display Manager can add displays to and remove them from a running application without interruption, when ordered to by the Environment Manager.

Network Manager

The *Network Manager* is a generic object-oriented interface to the networking abstraction of the system.

Because the Network Manager is an abstraction, it may use any networking method to distribute the data. For example, it might use a simple sockets-based protocol or it may be implemented on MPI.

The Network Manager provides the system with all necessary networking for application and kernel use. The Network Manager can be used directly by an application, or an application can use other generic distribution methods provided by the library. In either case, all of the networking goes through the Network Manager.

The Network Manager will handle all the needs of distributed applications. In the future, we plan to extend the Network Manager to support distributed environments (that is, connecting separate VR systems, possibly at remote locations, so that their users can interact with each other).

Configuration Manager

A full-fledged VR setup is going to have a great deal of configuration information – display device name, video modes, window locations, tracker serial port setup and calibration file names, application-specific data, and much else besides. The *Configuration Manager* is an extensible database designed to organize all of that information.

The largest unit of configuration information in VR Juggler is the “chunk.” A chunk contains all the configuration information for a particular part of the VR system. For example, the chunk for a particular kind of tracker might include information about serial port settings, position and orientation offsets, and the location of a calibration file.

Chunks are divided into properties. Each property has a type and one or more values. Continuing the previous example, the tracker might have a “serial port” property which contains a single string value. In a Unix system, the value might be something like “/dev/ttyd45”. The positional offset might be a property with three floating point values, for the offsets along the x, y, and z axes.

Applications can use the Configuration Manager to define and store their own configuration information. The different kinds of chunks are defined in a plain

text file; new chunk types can be defined by editing that file or using a convenient GUI utility. The application’s configuration data will be loaded with the rest of Juggler’s configuration information, and the application can query the Configuration Manager for the data. Juggler’s GUIs are entirely able to support user-defined chunks, giving applications access to the same sort of simple, graphical configuration tools as the VR Juggler environment itself, and also giving the application direct access to Juggler’s powerful dynamic reconfiguration capabilities.

Environment Manager

The *Environment Manager* is VR Juggler’s run-time control system. It actually consists of two components: an object similar to the other managers, and a graphical interface tool which allows users to examine and control Juggler applications.

The library side of the Environment Manager is built around a network connection that is used to talk to the GUI. It collects configuration and performance data from the rest of the system and passes it across the network to the graphical interface. It can also receive instructions from the GUI. These instructions might include such things as “Restart tracker number two” or “Create a new display with this configuration...”

The GUI side of the Environment Manager has been written in Java for maximum portability. Currently, it is an application, but the possibility of making it an applet accessible through a web browser is being considered.

The GUI gives the user the ability to examine and alter every piece of configuration information in the system. It also allows control over all running processes, such as the ability to start and stop displays or trackers. It will also be able to display a variety of performance data, such as display frame rates, processor loads, and input device data latency.

Low-level Foundations

A set of low-level objects form the foundation on which VR Juggler’s Kernel and Managers are built. These objects control process management, process

synchronization, and memory management. Most of the platform and operating system-specific details of Juggler's operation are buried in these classes. Therefore, porting VR Juggler to a new architecture consists primarily of rewriting these low-level classes to use whatever synchronization, process management, etc., primitives are available on the new system. Confining system-specific details to these classes aids porting of the Juggler environment and reimplementing these classes for each architecture helps ensure maximum performance for each one.

The Kernel

The Kernel binds all the rest of VR Juggler's components together. It controls the whole of the run-time system, and brokers all communications between the various Managers.

A major advantage of this design is that the Managers are, for the most part, loosely-coupled and very independent. This has proven to be a great aid in the development process, because most modifications and additions to the system have been very localized. This makes adding features, such as a Draw Manager for a new graphics API, easier to write, easier to keep track of, and, most importantly, easier to debug.

The Application

VR Juggler includes several application framework classes, one for each available graphics API. The fundamental difference between them is which Draw Manager will be instantiated. A developer creates a new application by choosing one of these framework classes and creating a subclass of it. The framework includes functions for drawing frames and performing and pre- and post-drawing computations, which should be filled in by the developer.

The application only speaks directly to the Kernel, but function calls in the Kernel's API give the application access to the Draw Manager, devices controlled by the Input and Output Managers, and configuration information stored in the Configuration Manager.

Current State of the Software

At this writing (late April 1998), an initially version of VR Juggler for the SGI architecture is being prepared for internal use and evaluation at ICEMT. This version supports OpenGL and Iris Performer applications, and supports many of the input devices available to the development team.

Display support has so far emphasized the C2, a CAVE-like projection screen system. Support for HMDs is forthcoming.

Design work for the Network Manager is ongoing, as we explore various possibilities for enabling high-performance distributed applications and environments without unduly burdening developers. Full support for distributed applications is slated for the second major release of VR Juggler, tentatively scheduled for late 1998.

Performance measurement capabilities for the Juggler classes are currently being implemented.

The basic network functionality and configuration management features of the Environment Manager GUI are functional, and work continues on the run-time control functions (starting and stopping of displays and devices, and so on).

Conclusion

The VR Juggler architecture is meant to set a new standard for VR development APIs. Our new object-oriented design is very portable, and presents a simple, easy-to-learn interface to developers. We expect that VR Juggler's performance will equal, and perhaps exceed, that of most current VR solutions, and we are working to support our expectations with a detailed performance analysis. We hope that VR Juggler's introduction will encourage the development of highly portable and scalable VR applications, and allow developers to concentrate on the worlds they want to create, instead of the systems that run them.

References:

[Cruz-Neira93] Cruz-Neira, C., Sandin, D.J., and DeFanti, T.A., *Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE*. ACM SIGGRAPH 93 Conference Proceedings. August 1993, pp 135-142.

[Cruz-Neira95] Cruz-Neira, C. "Virtual Reality Based on Multiple Projection Screens: The CAVE and Its Applications to Computational Science and Engineering." Ph.D. Dissertation, University of Illinois at Chicago. May 1995.

[Ghee97] Ghee, S. *Programming Virtual Worlds*. ACM SIGGRAPH 97 Course #29 notes. July 1997.

[Sense8-97] Sense8 WorldToolkit R8 Reference Manual, 1997.

[Kawalsky93] Kalawsky, R, *The Science of Virtual Reality and Virtual Environments*. Addison-Wesley, 1993.