A Review of Three Software Frameworks for Immersive Virtual Reality System

Shi Yin

University of Illinois at Chicago

Abstract

In order to help researchers and developers by accelerating the development of VR applications, a lot of software frameworks have been developed since the first appearance of CAVElib more than 20 years ago. Some VR middleware could only be used in a very narrow area, while some other of them were aimed to become a feature-rich framework that provides both usability to researchers with different requirements and specific features that meet the need of its developers.

VR Juggler was one of most impact development platforms among virtual reality community. It proposed a concept called "virtual platform", aiming to provide a virtual reality system-independent operating environment. Years later, [Morillo et al. 2008] did a great job analyzing the performance of VR Juggler.

FreeVR is another mature integrated VR library. A recent paper about FreeVR [Sherman et al. 2013] focused particularly on how its features serve to restore applications of the past into working condition and to aid in providing longevity to newly developed applications.

Recently, Schulze et al. published a paper about CalVR [Schulze et al. 2013], their new virtual reality software framework. They described CalVR thoroughly, focusing on introducing the philosophy behind CalVR, its features that suit the development of technology and meet their own need, as well as the programming interface and how it works internally.

In this paper, we mainly describe these three frameworks, describing their architectures and comparing their unique features. We hope this work would be helpful in keeping developing track of virtual reality middleware as well as in making future systems.

1 Introduction

Since the debut of CAVE Automatic Virtual Environment (CAVE) of Electronic Visualization Laboratory (EVL) in SIGGRAPH 1992, many research organizations adopted this virtual reality (VR) system and developed a variety of virtual reality applications using CAVElib, the software library specially developed by EVL for CAVE-like virtual reality systems. The dominance of CAVE system made CAVElib popular among researchers, and CAVElib in turn ensured the proliferation of applications that run on CAVE system.

After the CAVE system, CAVElib and the name 'CAVE' were licensed for commercial use, many organizations started to produce their own in-house VR software libraries [Sherman et al. 2013]. One of the early attempts was VR Juggler. Lead by EVL alumna Dr. Carolina Cruz-Neira, VR Juggler project has been under active development and maintenance for more than fifteen years. As one of the most important virtual reality frameworks in history of virtual reality, the most important contribution of VR Juggler was that it was designed entirely in an object-oriented manner in order to abstract implementation details for application developers and to decouple components of the library in order to allow run-time reconfiguration of hardware and software.

Another library used by many developers was FreeVR. FreeVR project was started by William Sherman as an open-source alternative of CAVElib, trying to halt or reverse the balkanization of the community after the commercialization of CAVElib. One of its primary purposes was to allow existing applications continue to run in newer VR systems. Same as VR Juggler, FreeVR is also still under active maintenance and update today.

But as the technology moves forward, new libraries are kept being developed to make the most out of state-of-the-art hardware and to meet the increasing demand from different areas of scientific research. Immersive Visualization Lab (IVL) of California Institute for Telecommunications and Information Technology (Calit2), one of the top institutions in immersive virtual reality area, recently published a paper, introducing their new 3rd-generation software framework CalVR, which includes common standard features as well as a bunch of new unique features that were designed specifically for today's research need.

Besides above three frameworks that we are going to discuss in this paper, there are much more frameworks, all of which provided unique features for their specific requirements, in addition to common features that shared by almost every virtual reality software toolkit. The most unique feature that makes Vrui [Kreylos 2008] different from other frameworks was that it provided a unified user interface to be used by both 3D immersive environment and 2D WIMP (Window, Icon, Menu, Pointer) interface. The Syzygy [Schaeffer and Goudeseune 2003] toolkit implemented a distributed scene graph model to improve performance for cluster-based rendering.

The rest of this paper is organized as follows: Section 2 introduces VR Juggler based on [Bierbaum et al. 2001], focusing on its object-oriented design to build the framework as a virtual platform that set developers free from low-level details. Also discussed is the later growth of VR Juggler. Section 3 discusses features of FreeVR [Sherman et al. 2013], paying special attention in the lessons learned during the fifteen-year-long course of development. Next, in Section 4 we introduce CalVR, delivering a general idea about today's most advanced libraries. Finally, Section 5 discusses shortly about these libraries as a whole before Section 6 concludes the paper.

2 VR Juggler

An important concept that played a core role in the design of VR Juggler was virtual platform (VP). According to [Bierbaum et al. 2001], a VP should make it possible for application developers to avoid dealing with complex details in hardware architecture, operating systems, and available system configurations, and thus be able to focus only on their job, writing and testing virtual reality applications independently of the underlying technology.

In [Bierbaum et al. 2001], Bierbaum et al. gave a minimum set of challenges that a VP should address. It should 1) be scalable and flexible to run under different hardware configurations, such as different number of displays, computer systems, tracking devices, networks, etc. Also it should 2) abstract the complexities of virtual reality system. To allow developers to build and test advanced virtual environments, a VP should also 3) be able to run multiple applications simultaneously, 4) allow for run-time modification of environment configuration and 5) allow the developers evaluate and tune the impact that different layers have on the performance of the application.

VR Juggler was developed under this list of requirements as a guideline. It used a specialized microkernel architectural pattern [Buschmann et al. 1996], consisting of a kernel object, several internal and external managers, and application interfaces.

2.1 Kernel

The core of VR Juggler's microkernel architecture was the kernel object. It was responsible for synchronizing all managers and applications plugged on the system. A manager or application only processes when the kernel calls a method of it or releases a signal to its thread to continue. The kernel was implemented on top of a series of low-level primitives that handle hardware-dependent issues such as process management, synchronization, etc.

2.2 Configuration

In VR Juggler, configuration information was stored as *chunks*. Each chunk included all the configuration information for a component of the system or application. For example, window chunk stored information about the name of the window, the size, and the position of it.

2.3 Internal Manager

To keep the kernel as small as possible and to increase the decoupling of different parts of the framework, VR Juggler implemented some core functionality that is not easily handled in the kernel object in internal managers.

The Input Manager, one of internal managers, controls input devices for the kernel. It contained a base class with must-have functions for all devices as well as a base class for each of the four categories of devices it supports, including position devices, digital devices, analog devices and glove devices. Input Manager creates device proxies to facilitate communication between application and devices. An application receives proxy of a device when asking for input data. Also, all the device drivers were stored in a factory object, separate from the main library. In this way, applications could be device-independent, and users could change devices or add new device drivers while the system was actively running.

Another internal manager was the Display Manager. It manages all configuration information about display screens and windows, thus facilitating application independence from the display being used as well as allowing reconfiguration of displays (add or remove).

The last internal manager was the Environment Manager, which was used by the kernel to communicate information about state of the system to external programs. The importance of Environment Manager was to allow users to reconfigure the system at run-time.

2.4 External Manager

VR Juggler also had several Draw Managers, each of which is implemented for a specific graphics API. They were called external managers because they had interfaces that applications can communicate with. They managed all the API-specific details, such as configuring the API settings, setting up parameters and rendering the view, one for each API. By keeping all the graphics API-specific details away from the application, the draw manager maintains the portability of VR Juggler.

2.5 Application

One of the most important features of VR Juggler was that all applications were built as objects, like other components. An application only had access to the kernel and the draw manager for the graphics API that used by this application but nothing else. What application developers do to build an application is just to implement a set of pre-defined interfaces. There were no main() function in applications. By calling the member functions of the interface, the kernel and draw manager communicate with the application to do processing or ask for information. From the entire system point of view, applications were also components like other parts of the system. This made possible for VR Juggler to change applications during run-time.

2.6 Others

Each manager in VR Juggler had an interface to add or remove configuration chunks for current configuration. When the kernel receives a request to change configuration, it passes the request to corresponding manager or application, which will then fulfill the request or pass the request to its dependencies in the same manner.

VR Juggler also included built-in performance monitoring capabilities, such as collecting data about time spent by different processes and measuring tracker latency. This helped developers to evaluate and tune his/her application in a convenient way.

2.7 Evolution

According to the record at VR Juggler project website, VR Juggler has been under active update since its first release in 2001. The different components of VR Juggler were separated into modules in late 2001, increasing the decoupling even more. These modules included VPR, a portable runtime that provided platform-independent abstractions for threads, sockets and serial I/O primitives, JCCL, a configuration and control library that allowed monitoring application performance, and Gadgeteer, a device management system that provides management of input data from VR devices. During 2002 to 2003, VR Juggler added support for clustering, called ClusterJuggler. Later, it also included a Python module to allow writing application objects in Python.

2.8 Evaluation

Since [Bierbaum et al. 2001] was an introduction paper written right after the initial release of VR Juggler, it did not include any real world experience of this framework as evaluation. The authors mentioned that they did some performance evaluations for applications built on top of VR Juggler as response to the popular belief that ‘object-oriented abstraction fails to keep good performance’, but no real result of performance evaluation was showed until ClusterJuggler was developed.

In 2008, Morillo et al. [Morillo et al. 2008] published a paper to introduce their work on implementing ClusterJuggler to evolve VR Juggler with clustering support as well as to present performance analysis results. ClusterJuggler combined advantages of four different existing clustering techniques. Its main contributions were to provide application portability and scalability from high-end systems to commodity clusters by hiding the clustering from developers and to allow users to customize the clustering methods according to their specific demand.

In [Morillo et al. 2008], the authors presented a performance evaluation of ClusterJuggler. They used real VR applications with different graphics-intensities and computation-intensities for the evaluation. According to the result they presented, the performance was linearly reduced as more nodes are added to the cluster and the performance was much worse for computationally intensive scenes than graphically intensive ones. Generally speaking, ClusterJuggler was an efficient tool to simulate multi-screen immersive visualization environments on a cluster of commodity computers as of that time.

3 FreeVR

Another software library contemporary with VR Juggler was FreeVR. It was first designed and developed fifteen years ago, as an open-source alternative of CAVElib. It focused on providing a consistent interface that would allow VR applications, no mater existing ones or newly developed ones, to stay functional longer in the future, even if the technology of VR system would inevitably evolve. Like VR Juggler, FreeVR is also still under active maintenance and update today.

Recently in 2013, William Sherman, the main developer of FreeVR, and his colleague published a paper as a detailed interim report about the experience gained during this long course, including good choices and those that were that good; jobs well done by FreeVR as a toolkit for developing VR applications; successful adoptions of the library to multiple VR facilities, and helps given to applications in the pursuit of longevity [Sherman et al. 2013].

The motivation of building FreeVR toolkit was similar to VR Juggler but not exactly the same. The primary goals of FreeVR were to keep the library as simple as possible, avoiding complex dependencies; to permit application developers the freedom to choose VR contents (world simulation and rendering) of their own preference and, most importantly, to keep existing applications alive for as long as possible, allowing researchers and practitioners to continue benefiting from experiences of history. From the list of requirements they created for FreeVR, we can see most of their targets reflected this philosophy very well, such as ‘support applications on just about any computing system’, ‘be convenient to import existing VR applications’, ‘provide robust interfaces to specialized VR devices’, ‘limit its functionality to particular needs of virtual reality’ and ‘compile quickly and with little difficulty’.

3.1 Standard Features

FreeVR handled all basic requirements for a virtual reality integration library. Firstly, in the same way as the CAVElib did, FreeVR supported multiprocessing by creating forked processes or new threads for different operations that had to be done simultaneously. Secondly, as a VR framework that had ambition of being adopted by many VR systems, FreeVR provided functionality to allow flexible configuration and to handle most major input devices. Thirdly, even though FreeVR did not implement a fully usable user interface, a rendering of the VR space, including a user's head and devices in his/her hands was presented. This feature provides a means to test applications in desktop without using a real VR system. Lastly, as more and more virtual reality systems are taking advantages of clustering of personal computers to drive multi-screen displays, FreeVR, after standing in the camp of symmetric multiprocessing (SMP) system a long time, finally decided to embrace cluster system.

The long course of developing VR applications served as catalysis for the reveal the potential improvements in front of FreeVR team. Based on their years of experience, they kept adding unique features into FreeVR.

3.2 Multi-User Capabilities

FreeVR used a concept called *eyelist* to naturally handle multiple-user scenarios. According to the authors, an eyelist had "a list of eyes, each of which could be associated with any 'user objects' in the configuration, and each of which would have be masked by a particular color. Different screens could then be using different eyelists, or different users' eyes could be assigned different color masks". In fact, CAVElib had a similar feature to support multi-users simultaneously, but that technique would reduce the light that comes into each users' eyes severely. And at that time, projection technology could not produce images of high brightness originally. Imagine how the quality of images would be after the light was reduced. FreeVR allowed different users standing at different positions in the VR system to see the virtual world from their respective perspectives. This especially unique feature of FreeVR was achieved by assigning his/her own copy of 'travel matrix' and 'perspective matrix' to each user. An application that benefited from this feature is [Koepnick et al. 2010]. However, without detail explanation of the underlying principle, this part of FreeVR paper was very difficult to follow.

3.3 Render Data Conduit

The utilization of data conduit in FreeVR rendering system was another elegant feature. The goal of this feature was to pass necessary information from FreeVR main loop to other functions. The procedure of this conduit was described briefly. The first step was to populate in FreeVR main loop a structure containing information about rendering, such as what window was being rendered to, which eye was being rendered for and a matrix converting between real world and virtual world coordinate systems. Then it was passed as an argument from main loop to rendering routine in a FreeVR application, where the application could use it as an argument to call other functions of FreeVR library. In this way, FreeVR fulfilled the task to pass necessary information around without the bad practice of using global parameters, or popular (but bad) technique of allowing rendering callback to know about particular rendering loop parameters through tricks such as locally scoped global variables.

3.4 Inputs in the Rendering

One important rule the FreeVR team learned was that input should only be read within the world-simulation portion of the application, not too early or too late. This was to keep all rendering processes sampling an input at the same time, otherwise visual scene would probably be different in different screens or for different eyes. One major exception of this principle was that FreeVR would use the 6-sensor of the user's head to do pre-calculation before start rendering the scene.

3.5 Choice of Graphics Libraries

As a practice of FreeVR's philosophy of minimalism, it offered OpenGL interface to give application developers access to all OpenGL functionality, rather than preparing a particular set of graphics libraries to application developers. Another wish in this design was to ease the porting of non-immersive OpenGL applications to virtual reality, but other issues impeded this effort. When first released in 1998, FreeVR also had SGI Performer version to keep with the tradition of CAVElib and thus allow old applications to be easily implanted to FreeVR framework. This was because many earliest applications written for CAVE made use of the Performer library. Later, when SGI and Performer became less popular in VR community, and as the demand for native support of popular graphics libraries increased, FreeVR started to integrate more graphics libraries as options for applications developers, such as OpenSG and OpenSceneGraph (OSG).

3.6 Others

There were also some features that were not unique in terms of functionality, but were implemented in a unique way. One of them was the configuration feature. Unlike dividing configuration information into chunks in VR Juggler, FreeVR had a single configuration file for the whole system. Each component will then ask for information relevant to its functionality when initialized. Users were allowed to set necessary configuration parameters as environment variables through a special external menu system before an application is launched. For example a flag could be marked to run an application in monoscopic mode instead of stereoscopic.

Another such feature was the ability to connect to a running application, which was also included in VR Juggler. In FreeVR, the connection to the application was established via telnet protocol. Developers could inquire about and tweak any configuration parameter of the application through either command line tool or a GUI interface implemented in FreeVR.

3.7 Enhancements

In this detailed paper, the authors also discussed six special enhancements made to FreeVR based on feedback from application developers. For example, the first enhancement, ability to use Euler angles, was not included in FreeVR since it is not an essential component for building a toolkit 'as simple as possible'. Instead, it was later added into the library when FreeVR team found that some old applications that were trying to continue thrive under the help of FreeVR cannot achieve their goal since there were no support for use of Euler angles. In process of metabolism like this, FreeVR evolved while the team kept learning from new experience.

Today, FreeVR didn't stop its steps moving forward. Currently the team is trying to make some new components more robust and elegant, and more importantly, trying to make sure FreeVR can interface to low-cost off-the-shelf tracking devices and stereo TVs.

4 CalVR

As hardware technology became more and more advanced, no matter how much effort was put into the improvement of an existing software system, it cannot meet the increasing demand of the community forever. In a recent published paper [Schulze et al. 2013], the team from Calit2 introduced their most advanced software framework CalVR. This paper was written in a similar structure as the FreeVR paper [Sherman et al. 2013]. The authors at first briefly introduced the background and motivation of development of CalVR, and then presented standard features of CalVR, followed by unique features.

The motivations behind the development of CalVR were to solve issues that did not handled well by first and second generations of virtual reality libraries, such as CAVElib, VR Juggler and FreeVR; and to realize new functionality that was necessary for today's VR systems and was difficult to be added into old libraries without changing inner code. To achieve this goal, CalVR combined standard features of existing framework as well as added new features based on research demand of Calit2 and other institutions.

4.1 Standard Features

In general, CalVR implemented standard features that has been basic requirements for a virtual reality integration library since CAVE, including hardware driver interface to different input devices, OS interface to all display devices, multiprocessing support and configuration for different VR system. They also implemented features that were not considered as standard until years later.

CalVR included cross-platform support, working perfectly on Linux and Mac OS. It also could be used in Windows platform, but without cluster support. CalVR was written in C++, using an object-oriented class hierarchy that similar to VR Juggler. Another feature of CalVR was the support for distributed or clustered computing system, which FreeVR could not do at that time. Frame buffer swaps were synchronized over TCP, and user input would be distributed to all rendering nodes to synchronize views. Because of the exponential increase in hardware capability, such as resolution of displays and computing power of GPUs, CalVR was recommended to run in at least 1Gbps network. CalVR chose OpenSceneGraph as its graphics library due to its popularity and rich-feature nature. Features of OSG library in which CalVR was particularly interested were the import routines for graphics file types, intersection testing and a bunch of node kits for additional functionality. Through OSG, CalVR was able to support different stereo modes (active, passive, interlaced, etc), which were particular important for CalVR and applications developed in Calit2. All configuration information of the system was stored in the same way as FreeVR, except that this time the single configuration file was written in XML, the popular modern markup language. Other standard features of CalVR included support for several popular tracking libraries and support for three default navigation modes (fly, drive and scale).

4.2 Unique Features

One of the most innovative features of CalVR was the concept of "SceneObject". This class handles navigation, interaction and context menu of a specific object in the scene. SceneObjects can be nested so that application developers can interact with multiple different geometrical objects in the scene in almost any way they want.

Besides, CalVR also provided the possibility to use four different rendering modes on top of OSG. The first one was auto-stereoscopic mode, which was realized through a library by Robert Kooima [Kooima et al. 2007]. Besides, an anti-aliased mode, and omnistereo mode and a mode in which head position could be specified differently for each screen of the VR system. The last mode was considered as important by the authors in that it provided possibility of creating convex tiled display screens. But as of today, only few, if not no, publications about hardware benefits from this technique were published.

Above two features together allowed special care to be given to tiled display walls with monoscopic displays, which was another unique feature of CalVR. By keeping the "TiledWallSceneObject", which derived from "SceneObject", always in the wall plane, CalVR provided support for 2D displays.

As for user input, like other components, it was also implemented as a class. In CalVR, all user input was handled as events. When user released a command through input devices, an event was generated in the head node. It would be sent through each node's event pipeline after synchronization was done. Events of different actions had different priorities in the even pipeline. Menu system had first access, followed by SceneObjects, then plugins, and navigation had lowest priority.

CalVR also included a plugin system that allowed add or move applications during run-time, the detail of this system was not included in [Schulze et al. 2013], but the mechanism was like what VR Juggler had done. In one word, no information about uniqueness of this feature can be found in this paper.

Other unique features of CalVR included the support for multiple tracked users, even with multiple input devices in each; support of asynchronous tracking to avoid information loss when frame rate is low; a custom cull visitor that decreases loading time of a scene; a layered menu system that comes with many item types, such as action button (trigger events), check box (turn on or off) and range (select a value); a collaborative mode brought by a native plugin that can connect different sessions in different facilities that runs CalVR over network; and support of terrain rendering through osgEarth that would help scientists from other disciplines such as geoscience.

5 DISCUSSIONS

Due to the complex nature of VR systems, a VR software library must include a good amount of features in a variety of aspects. This leads to the result that, at the first look, all three papers look like clutter collections of facts and details. But if we omit all the technical details and look only for similarities and uniqueness in terms of content (what features these systems have), structure (how these papers were written) and trend (how VR software frameworks evolve), we can conclude following observations:

Some of the unique features of a toolkit would be inherited by newer frameworks, or replaced by better implementations of same idea, while other features would fall into the ashes of history. And this generally depends on how the progress in hardware goes, along with some other factors. Take CAVE as an example. The 4-screen projection system once dominated the virtual reality research organizations for several years. But as the technology evolves, LCD displays became more and more popular. Projection systems are usually expensive, requires constant maintenance, and cannot produce image with high contrast, especially in environment that no lit well [Sherman et al. 2013]. EVL recently built a next generation CAVE, which consists of 72 custom 3D micropolarized LCD panels [Febretti et al. 2013]. Calit2 also built NexCAVE over StarCAVE [DeFanti et al. 2009].

Another lesson we can learn from the track of evolution of VR software libraries is that no matter how much effort were put into making a framework as flexible as possible, and no matter how much independence it offered, it could not always serve researchers, developers and other users well. There would always be new systems that were developed from scratch. How to make most benefit with least cost, is a permanent question that should be kept in mind when designing and develop new software libraries.

6 CONCLUSIONS

In this review, we discussed three important virtual reality software frameworks in the history of VR research. VR Juggler was among the most important libraries in the course of evolution of immersive VR software frameworks. It was not only a useful tool for developing VR applications at that time, but also a model that affected its successors a lot. FreeVR, looked back from today’s point of view, offered a lot of lessons that could benefit today’s VR practitioners. It was not only a good alternative of CAVElib at that time, but still trying to offer good tools to today’s VR application developers. CalVR is among the most advanced frameworks today. It not only includes old standard features, but also aims to the new trend of virtual reality and visualization research in the future.

References

BIERBAUM, A., JUST, C., HARTLING, P., MEINERT, K., BAKER, A., AND CRUZ-NEIRA, C. 2001. VR Juggler: a virtual platform for virtual reality application development. *Proceedings of Virtual Reality 2001 Conference (VR ’01)*, 89–96.

BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P., AND STAL, M. 1996. *Pattern-Oriented Software Architecture: A System of Patterns*. .

DEFANTI, T.A., DAWE, G., SANDIN, D.J., ET AL. 2009. The StarCAVE, a third-generation CAVE and virtual reality OptIPortal. *Future Generation Computer Systems 25*, 2, 169–178.

FEBRETTI, A., NISHIMOTO, A., THIGPEN, T., ET AL. 2013. CAVE2: a hybrid reality environment for immersive simulation and information analysis. *Proceedings of IS&T/SPIE Electronic Imaging, The Engineering Reality of Virtual Reality 2013*, 864903.

KOEPNICK, S., HOANG, R. V, SGAMBATI, M.R., COMING, D.S., SUMA, E.A., AND SHERMAN, W.R. 2010. RIST: Radiological Immersive Survey Training for two simultaneous users. *Computers & Graphics 34*, 6, 665–676.

KOOIMA, R.L., PETERKA, T., GIRADO, J.I., GE, J., SANDIN, D.J., AND DEFANTI, T. A. 2007. A GPU Sub-pixel Algorithm for Autostereoscopic Virtual Reality. *2007 IEEE Virtual Reality Conference*, Ieee, 131–137.

KREYLOS, O. 2008. Environment-Independent VR Development. *Proceedings of the 4th International Symposium on Advances in Visual Computing*, Springer-Verlag, 901–912.

MORILLO, P., BIERBAUM, A., HARTLING, P., FERNÁNDEZ, M., AND CRUZ-NEIRA, C. 2008. Analyzing the performance of a cluster-based architecture for immersive visualization systems. *Journal of Parallel and Distributed Computing 68*, 2, 221–234.

SCHAEFFER, B. AND GOUDESEUNE, C. 2003. Syzygy: Native PC Cluster VR. *Proceedings of the IEEE Virtual Reality 2003*, IEEE Computer Society, 15–22.

SCHULZE, J.P., PRUDHOMME, A., WEBER, P., AND DEFANTI, T.A. 2013. CalVR: an advanced open source virtual reality software framework. *IS&T/SPIE Electronic Imaging*, 864902–864902–8.

SHERMAN, W.R., COMING, D., AND SU, S. 2013. FreeVR: honoring the past, looking to the future. *IS&T/SPIE Electronic Imaging*, 864906–864906–15.