

COOL-VR: a Virtual Environments Toolkit

Rob Kooper, Brian Wills, Kevin Hamilton, Don Allison, Larry F. Hodges

College of Computing
Georgia Institute of Technology
801 Atlantic Dr.
Atlanta, GA 30332-0280
{kooper, bwills, kevinh, don, hodges}@cc.gatech.edu

Abstract

This paper provides an overview of the Common Object Oriented Library for Virtual Reality (COOL-VR) toolkit, a multi-platform, multithreaded toolkit that allows for easy adaptation to different hardware interfaces. Where the common definition of an object in a scene graph is limited to a graphical representation, this toolkit extends the idea of a scene graph to include objects with different and possibly multiple representations (e.g. audio, graphical etc). Each of these different representations is 'rendered' by an output device (screen, HMD, headphones, speakers etc). The toolkit allows the easy addition of new object representations as well as the addition of any new output devices needed for the output of these representations. COOL-VR also provides an easy interface for adding new file loaders that may be needed to load these new representations. In order to provide portability, the toolkit provides an abstraction layer that hides the inconsistencies between differing platforms, and provides the user with a single programming interface. Applications developed using the COOL-VR toolkit can be built on one platform and, without changing any code, be recompiled to run on a completely different platform (for instance from Unix to Windows). When running the application on a different platform, the toolkit maintains the look and feel of that new platform, but the interaction with the application stays the same across the different platforms.

Introduction

The field of virtual reality is a young and rapidly evolving discipline. The introduction of new hardware devices (e.g. I/O, audio, position tracking, force feedback, etc.) coupled with rapid increases in available computing power at lower costs creates opportunities to build sophisticated applications on traditionally 'low-end' platforms. This rapidly changing computing environment also has a detrimental effect on development in that it makes it

difficult to have up-to-date user tools that take advantage of these advancements. These rapid changes are particularly visible on PCs running Windows NT or Windows 95/98, where the current graphics cards are equal to and sometimes surpassing the performance of low and middle end graphics workstations.

One of the major motivations behind the design and implementation of COOL-VR (Common Object Oriented Library for Virtual Reality) was to create a toolkit that could easily be extended to incorporate new hardware platforms and graphics APIs. Since non-visual modalities are becoming more common and more important to virtual environments, we also extend our rendering model to immediately incorporate the design of audio and allow for the addition of different sensory modalities such as haptics (force and tactile feedback). These different output methods require the addition of various representations for each object within the virtual environment. In this paper, we discuss this extended concept of rendering in COOL-VR as it applies to different sensory modalities. Hence, a renderer, in the context of the COOL-VR tool-kit, is any module that takes a representation of an object and outputs it to some device. This module can be a graphics renderer, an audio renderer, a haptics renderer and even a network renderer.

In this paper we first look at other toolkits, both noncommercial and commercial, that are currently available, give an overview of the functionality they provide, and explain some of the differences between these toolkits and COOL-VR. The following section goes into more detail about the design of COOL-VR and gives an update on its current status that explains what has already been implemented and what work is currently in progress. Finally, we discuss future plans for additional functionality to be incorporated into COOL-VR.

Related Work

So, why build another toolkit? Currently there are several noncommercial and commercial toolkits

available on the market. In this section, we will describe some of these toolkits and look at their advantages and disadvantages. We also point out some shortcomings of current toolkits that we intend to resolve in COOL-VR and some of the useful features of these toolkits that we can incorporate into COOL-VR. The rest of this section is divided in two groups, noncommercial and commercial Virtual Environments toolkits.

Noncommercial Toolkits

Most noncommercial toolkits are from universities and are used as research platforms. All of these toolkits that we will discuss run on Windows workstations and some of them also run on SGI workstations. Most of the toolkits focus mainly on graphics, while providing minimal functionality for audio. Since all of these toolkits are developed at universities, they are constantly undergoing modification. Some applications might use features of a toolkit that become obsolete. When such modifications are made, they might break our application. It should also be kept in mind that some of the shortcomings of specific toolkits pointed out in this paper may have been resolved at the time of publication. Our goal is to create a toolkit that allows us to use the different platforms that are currently available in our lab while retaining a high level of adaptability to new platforms. Additionally, the toolkit should be stable and allow for easy integration of better rendering engines, as they become available.

The MR toolkit [10] from the University of Alberta has been around the longest of all the toolkits described in this section. This toolkit was developed to hide the intricacies of developing an application for virtual environments. The toolkit decouples the computation that is used to present the view to the user from the computation of the application. The application can conceptually be split into four components, the master process, a server process for input, a computation process, and a slave process for output. When the application is executed, different processes are created for each of the components. This allows the toolkit to achieve interactive speeds even when the simulation is complicated. The disadvantage of the MR toolkit is that switching renderers is not easy and audio is not tightly integrated into the toolkit.

DIVE [2] is a VE toolkit developed at the Swedish Institute of Computer Science. DIVE is specifically tuned to multi-user applications, where several networked participants interact over the Internet. This toolkit provides functions for easily joining remote worlds. Another advantage of DIVE is the integration

of a scripting language (Tcl) that is used to code the behaviors of objects, and allows these object behaviors to be sent to other users connected to the virtual world. The developer can easily modify these behaviors and send an update to all other nodes. A disadvantage of this system is that the renderers used in DIVE are hard to modify and cannot easily be replaced.

Development of Alice [7] started at the University of Virginia but is now being continued at Carnegie Mellon University. Alice was developed to provide a simple interface for novice programmers for the development of 3D environments. Alice uses Python, an interpretative language, as the programming language in which the user creates the virtual world. The use of Python makes it easy to change the virtual world since no recompilation is needed. The disadvantage of using a scripting language is that it is slower than a compiled language due to the overhead of the interpreter. Currently, Alice only runs on PCs running Windows NT or 95/98.

Bamboo [11] is a toolkit currently under development at the Naval Postgraduate School. The goal of Bamboo is to produce a portable system supporting real-time, networked, virtual environments. Bamboo's design includes some of the same objectives as COOL-VR, such as easy addition of new modules. While in COOL-VR we have a clear model of objects with representations, it is not clear how Bamboo is going to support different representations.

Currently in our group, we work with the Simple Virtual Environments toolkit (SVE) [5] as our primary development environment. Development of SVE was started in 1993, and it has evolved over the years to expand in functionality as needed to support our VR applications. It currently is being maintained at its present state of development. SVE runs on both SGI and HP workstations as well as PCs running Windows NT and Windows 95/98. Some drawbacks in SVE include that the toolkit does not allow for multi-threading, does not allow for easy modification of the different renderers (graphics and audio), nor does it allow for the addition of any new renderers.

Commercial Toolkits

There are a few commercial VE toolkits on the market, including dVS/dVise from Division, OpenGVS from Gemini Technology and WorldToolkit from Sense8. The biggest advantage of these toolkits is that the companies are specialized in the development of VR toolkits, and can devote large amounts of resources (people, time, capital, etc) to the development of their toolkits. The commercial

development model also brings stability and better error handling with the downside of long development cycles between feature revisions. Additionally, it is often not possible to obtain source code for commercial toolkits, making it impossible to access and/or modify the internals of such a toolkit. This is a prime motivation for research groups to develop their own toolkits instead of relying on a commercial vendor. By having access to the internals of a toolkit, we can make modifications to low-level calls, allowing us to experiment with different APIs (graphics, audio, haptics etc.) and with routines that may speed up or improve rendering.

WorldToolkit from Sense8 [9] has been around for perhaps the longest of any of the commercial toolkits currently available. It provides the application developer with a high level API to quickly develop a virtual environment. WorldToolkit behaves similarly to the SVE toolkit, but it allows the programmer to use multiple threads and processors. The world toolkit allows users to easily add lights and lighting effects (i.e. fog) as well as audio. Unfortunately, as with all the commercial toolkits, the disadvantage for us is that we can not change underlying code. Also, audio is seen as a separate entity in the toolkit. By not allowing lights as well as audio to be attached to an object, when the object moves, these entities stay at their old location and do not move with the object.

OpenGVS from Gemini Technology is a toolkit that helps the developer to quickly build an application that can be easily ported to different platforms, and different graphics toolkits. The biggest disadvantage of OpenGVS is that the toolkit is targeted at graphical rendering. It does not have support for trackers or head mounted displays. Again, audio is not tightly integrated into the toolkit.

The last of the commercial toolkits we will discuss is dVS/dVise [4] from Division. Division splits their toolkit in two separate components, a low-level toolkit, dVS, that is like most other toolkits described in this section and the previous section, and a higher level component of the toolkit, dVise, that is comparable to VRML [8]. dVise allows the user to implement the environment in a text file. This text file contains references to models, behaviors and constraints. The file can also contain audio references that can be attached to an object. Using the low-level API, the developer has more control over the toolkit and can now implement new interaction methods and other unsupported features. This combination of a high level language to describe the scene and a low-level API to add new features to the toolkit is a powerful combination. Unfortunately, we plan on

changing the renderers, and this requires access to the internals of the toolkit.

General Design

COOL-VR is based on the idea of having different types of renderers (see Figure 1). A renderer in the context of COOL-VR is a module that gives representation to an abstract object. For example, a graphical renderer provides a visual representation for graphical objects by displaying them on a screen or in an HMD. In the development of COOL-VR, we have extended this view of renderers to other sensory domains as well. We have created an audio renderer, which renders an object to speakers or to headphones. The same metaphor can also be extended to a haptic renderer, which renders the object to a force feedback device. We are currently investigating if we can extend this metaphor to a network renderer, which would send and receive objects across a network. This approach to design allows for easy incorporation of new renderers in future revisions of COOL-VR. If, for example, a hardware device for incorporating olfactory output into a virtual environment became available, a new COOL-VR renderer could be easily added for the association of aromas to virtual objects.

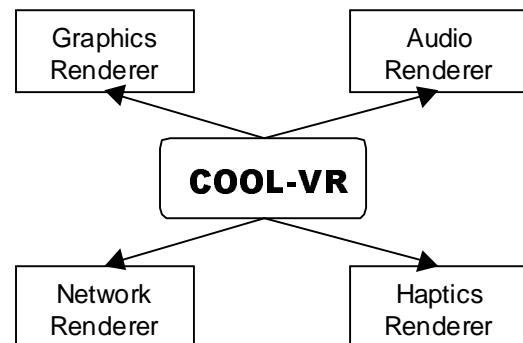


Figure 1 Render modules in COOL-VR

In COOL-VR, each renderer is designed as a separate module. This separation allows for updates to renderers by switching out one module for another without impacting other aspects of COOL-VR. For instance, the graphics renderer is currently written in OpenGL, but we also have a simple raytracer. By running a configuration utility, we can easily switch between the OpenGL renderer and the raytracer.

We use a standard object representation model [5]. In order to use multiple types of renderers for an object, we associate different representations with an object (see Figure 2). This multiple association of representations allows us, for example, to give an

object an audio, a graphic and a haptic representation, or just a graphical representation.

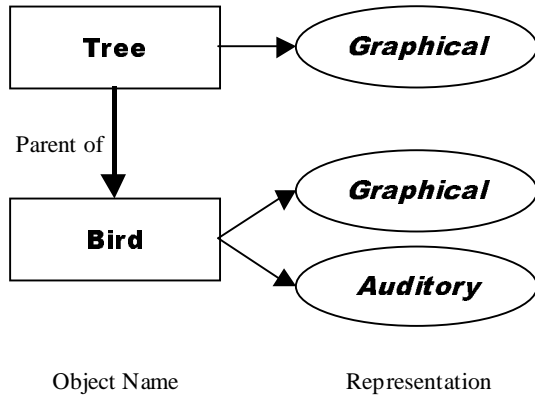


Figure 2 Object hierarchy

If, in the example above, we were to move the bird around, not only would we see a visual representation of the movement, but the audio renderer would also provide spatialized audio so that we would hear the changing position of the bird as it moved around the rendered scene. Notice that to do this all we have to do is move the bird object. The audio and the graphical representations are associated with the bird and don't have to be explicitly moved. Additionally, since the bird is a child object of the tree, if we move the tree in the world, the bird would move with the tree, and so would both graphical and audio representation of the bird.

This object hierarchy, including associated representations, is stored in a scene graph module. The scene graph allows for easy addition, removal and re-parenting of all of the various objects within a scene. When adding an object to the scene graph each renderer is updated and informed of the addition. The renderer can then decide if the new object has any representations that it can render. This way the renderer only has to check objects it needs to represent and can ignore all others. For instance, if an object only has a graphical representation, the audio renderer can safely ignore this object and optimize its own list of audio objects accordingly.

An application using COOL-VR consists of three main components, the `CVR_Main` and `CVR_Done` functions and a third component that is a subclass of `CVR_VEApp`. The `main()` (or `WinMain()` in the case of Win32 applications) function in either C or C++ is encapsulated in the COOL-VR toolkit. When a COOL-VR program is executed, the toolkit first loads the configuration file and passes control to the user component of the program (the `CVR_Main`

function). The user can then change parameters of the configuration and create an application instance (which is a subclass of `CVR_VEApp`). Once the application is done, the toolkit calls a final function in the application (`CVR_Done`) that allows the application to close all files and clean up after itself. The toolkit also performs some clean up operations when control returns from the application.

Once `CVR_Main` returns an application instance to the toolkit, COOL-VR then runs this application instance as the real application with the configuration parameters specified. It first creates all the renderers that the user specified in the configuration and then passes control back to the application instance to load the environment, request events and attach trackers. Once the user application returns, the toolkit takes over and runs all the renderers, and then starts polling all the devices for events. Every 100msec (this is configurable as well) the toolkit calls a function in the application instance. The application can now perform some work of its own. This separate work does not interfere with the renderers, since each renderer runs in its own thread. If the application needs to do a complicated computation, the user does not notice since the renderers continue presenting the user with the correct view

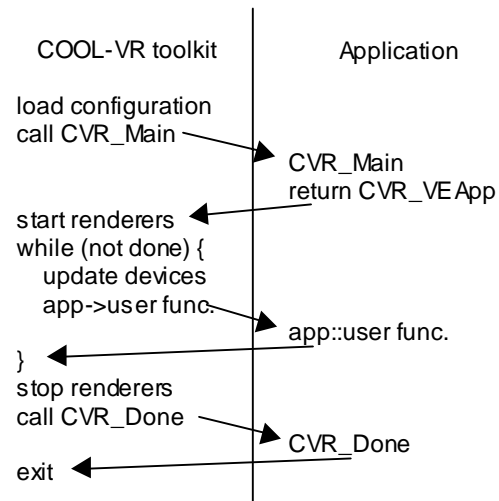


Figure 3 Control flow in COOL-VR

Specifics

The COOL-VR toolkit is being implemented simultaneously on PCs running Windows NT and on SGI workstations. We also maintain the toolkit on Windows 95/98 and, with a lower priority, on HP workstations and PCs running Linux. The toolkit is written in C++, which allows us to easily partition the toolkit in separate modules, and allows for addition

of different representations. We have considered implementing the toolkit in Java [3] but when we started there was no support for 3D. This is changing with the introduction of Java3D, but the speed of Java is still too slow for a highly interactive application (i.e. VR). In the near future, we may have to reconsider Java as it evolves as an alternative, but for now we will continue with the C++ implementation. This section describes more details of some specific components of the toolkit, specifically loading of files, threads and renderers.

File Loaders

COOL-VR recognizes certain files automatically. Currently, it recognizes RGB and BMP image formats, AIFF and WAVE audio formats and the Wavefront obj object format. Most other toolkits import these and other file types, either directly or by way of a conversion step. Since COOL-VR was designed to be easily extendable, we decided to make the file loaders into easily extensible modules. This allows easy addition of new file formats to COOL-VR. When writing a file loader, the developer needs to provide COOL-VR with functions that will recognize files (either by extension or by magic number), and functions to load and save files. Most of the file loaders we have implemented so far only load files and do not save the files from the application. The developer can register a loader with COOL-VR and COOL-VR can then use the loader. If the user wants to load a file, COOL-VR goes through its list of file loaders and asks all of them if they can load the file. If a file loader responds positively to this question, COOL-VR passes a pointer to an object that can hold the data once it is loaded. The loader can now load the data and store it in the object.

The file loaders don't need to be in the library itself. An application, for instance, might use some proprietary data and it can include that file loader with the application itself. When COOL-VR calls the application for the first time, the application can register that file loader with the toolkit. Now, when the application wants to read the proprietary data format, it can use the same calls as for standard data formats.

Threads and Mutexes

All the renderers in COOL-VR run in their own thread of execution. This allows us to make use of multiple processors and guarantees that the renderers continue running even if the application is busy. To implement this, we built a wrapper around the threads package that comes with the specific OS, or in some

cases we use separate processes to simulate threads. All the threads share the same data pool, so all internal data structures are available across all threads. The COOL-VR thread class is modeled after the thread class used in JAVA. The thread class has a `run` function that the user implements, a `start` function to start the thread, a `stop` function to stop the thread, and functions to `pause` and `resume` the thread. All the renderers in COOL-VR, for example, are implemented using this thread class. This allows all the renderers to run independently of each other. Once COOL-VR has finished loading all the objects, it starts all the threads, and all the renderers start rendering their appropriate objects.

To protect memory from being accessed by multiple threads, we have created a simple mutex class. This class is again implemented on top of the mutex provided by the specific OS, or on a simple spin lock (which is not completely safe). For instance, the scene manager has a mutex that protects the scene graph. If a thread wants to move an object or wants to manipulate an object, it would lock the scene graph and change the object. After it is finished, it would unlock the scene graph.

Renderers

As explained before, COOL-VR has various renderers of different types and functions. Each of these renderers is implemented on a base renderer class. This base class provides functions to add objects to and remove objects from the renderer. It also has a simple run function. For each renderer, the developer needs to override these standard functions and provide their own functions. We have currently implemented an OpenGL renderer and a raytracer to do the visual rendering, as well as an audio renderer to do the auditory rendering of objects [6]. The audio renderer had to be implemented differently on SGI and Windows, while the OpenGL and raytracer could be the same on both platforms. For example, Figure 4 shows the OpenGL renderer used on both SGI and Windows NT. As can be seen, there is no difference between the objects rendered on the PC and on the SGI (besides a difference in gamma values used on both PC and SGI). Having separate renderers allows us to easily configure COOL-VR to use any special renderer. Once an application is built, we can choose what renderer to use at the time of execution. If in the future we improve the toolkit with better renderers, we can reconfigure the application without recompiling in order to take advantage of these new renderers.

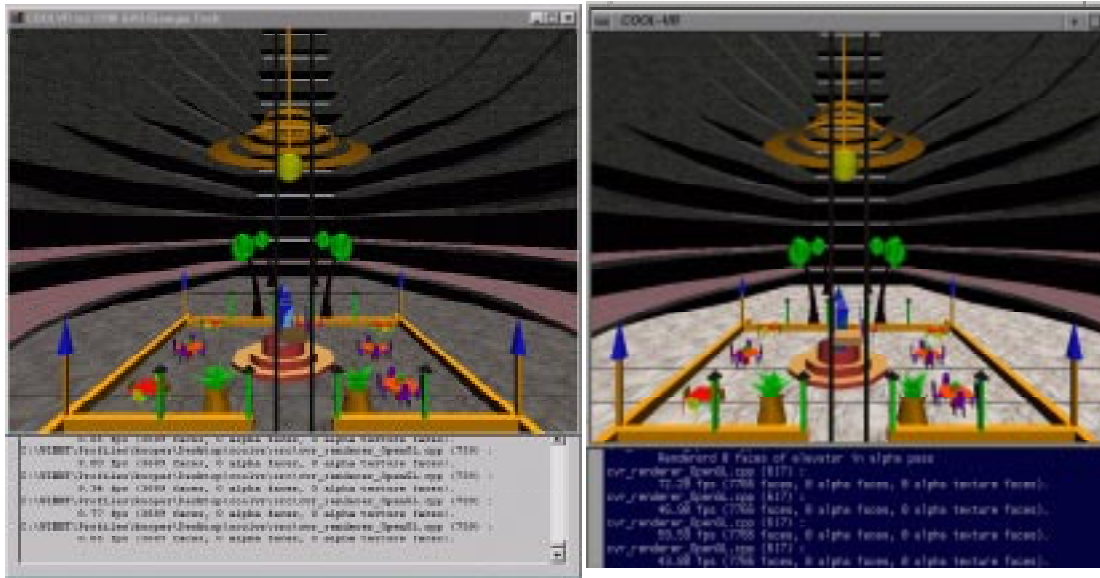


Figure 4 Model rendered on Windows NT (left) and SGI workstation (right)

Current status

Currently, COOL-VR is running on both SGI workstations and PCs running Windows NT and Windows 95. We are also intermittently testing COOL-VR on PCs running Linux and as well as HP workstations. Testing COOL-VR on these different platforms helps to insure that we keep the toolkit portable to new platforms in the future.

We have implemented the audio renderer on windows NT/95 and SGI workstations. We have a demo version of the audio renderer on Windows NT/95 that does spatialized audio, but this needs to be incorporated into the real version of COOL-VR. Later, we will implement the spatialized audio renderer on the SGI workstations.

We have implemented the graphical renderer in OpenGL, which works with both SGI workstations and Windows NT/95 PCs. This renderer is capable of rendering faceted objects (polygons) and light sources. Figure 4 shows the graphics renderer running on an SGI workstation and a PC running Windows NT. Besides the OpenGL renderer, we have also implemented a basic raytracer (see Figure 5), which provides the environment with shadows and more realistic lighting effects, but at the loss of the 'real-time', interactivity within the application. Notice in the images below that the sphere in the right image has a small specular spot, and there are shadows cast by the different objects. To switch to the raytracer we did not have to make any changes to

the application code. All we had to do was run the configuration utility and select the raytracer renderer.

On both UNIX and Windows platforms, we have implemented a utility to configure COOL-VR. This utility allows us to choose what renderer to use, set values specific for the renderer and make changes to generic variables of COOL-VR.

Future Work

First, as with all toolkits for VR, it needs to be faster, faster, faster. Also, we need to finish the audio renderer and implement spatialized audio for both Windows and SGI. We also need to implement audio renderers for the HP workstations and PCs running Linux. COOL-VR has currently no network support. We have postponed implementing the network support until we had a better understanding about what the renderers look like and how they interact. We are planning on first implementing simple network support that is fully connected, but hope to add a network renderer with more sophisticated networking, for example SPLINE [1]. Support for trackers is very rudimentary and needs to be improved. We are still thinking about an intuitive method for the user to add transmitters and receivers for tracking to an environment and have them arbitrarily oriented and located. We are also planning on adding more file readers to the toolkit, specifically a 3DS file reader and a VRML file reader. Currently, files have to be stored locally. We want to be able to use a URL in the future for file pointers. This way we can load files that are stored on a network.

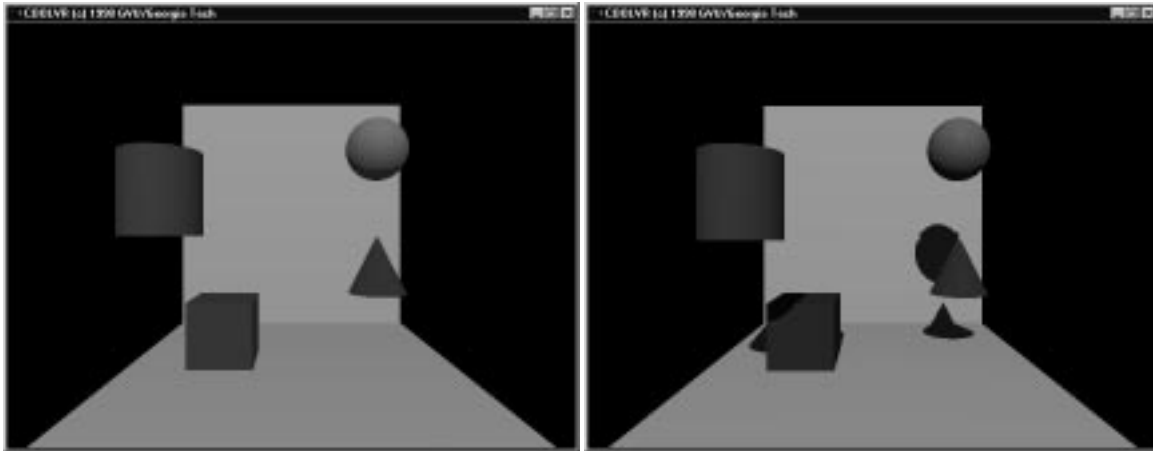


Figure 5 OpenGL renderer and raytracer.

When finished with the basics of the toolkit that will allow us to load environments, render them to either screen, HMD or workbench and connect across the network, we would like to add capabilities to use a scripting language to modify the behaviors of objects. A possible solution for the scripting language would be Java. Besides using Java for behavior scripting, it would be interesting if the user application could be written in Java as well. This would allow us to send precompiled applications across the network. The only requirement then is that the user has the COOL-VR library compiled for their specific platform. The binaries shipped across the network would run on a virtual machine, using native methods to access the COOL-VR library. This would also help if we switch to implementing COOL-VR completely in Java.

References

1. Barrus, J.W., Waters, R.C. and Anderson, D.B., *Locales and Beacons: Efficient and Precise Support for Large Multi-User Virtual Environments*, IEEE Virtual Reality Annual International Symposium, 1996, Santa Clara, California, pp. 204-213.
2. Carlsson, M. and Hagsand, O., *DIVE - A Multi User Virtual Reality System*, IEEE Virtual Reality Annual International Symposium, 1993, Seattle, Washington.
3. Flanagan, D., *Java in a Nutshell, Second Edition*, O'Reilly & Associates, 1997.
4. Ghee, S., *dVS: A distributed VR System Infrastructure*, ACM SIGGRAPH 95 Course Notes.
5. Kessler G.D., Kooper R., Hodges L.F., *The Simple Virtual Environment (SVE) Library: Version 2.0 Users Guide*, GVU Tech Report GIT-GVU-98-13.
6. Pair, J. and Kooper, R., *COOL-VR: Implementing an Audio Renderer for VR*, ICAD'97.
7. Pausch, R., Burnette, T., Capehart, A.C., Conway, M., Cosgrove, D., DeLine, R., Durbin, J., Gossweiler, R., Koga, S. and White, J., *Alice: Rapid Prototyping for Virtual Reality*, IEEE Computer Graphics & Applications, May, 1995, pp. 8-11.
8. Pesce, M., *VRML - Browsing & Building Cyberspace*, New Riders, 1995.
9. Rahn, S., *WorldToolKit Release 8 Technical Overview*, Feb 1998, Sense8.
10. Shaw, C., Green, M., Liang, J., Sun, Y., *Decoupled Simulation in Virtual Reality with the MR Toolkit*, ACM Transactions on Information Systems, July 1993, VolVol. 11, Number 3, pp. 287-317.
11. Watsen, K. and Zyda M., *Bamboo - A Portable System for Dynamically Extensible, Real-time, Networked, Virtual Environments*, IEEE Virtual Reality Annual International Symposium, 1998, Atlanta, Georgia, pp. 252-259.

COOL-VR: a Virtual Environments Toolkit

Rob Kooper, Brian Wills, Kevin Hamilton, Don Allison, Larry F. Hodges

College of Computing
Georgia Institute of Technology
801 Atlantic Dr.
Atlanta, GA 30332-0280
{koop,er, bwills, kevinh, don, hodges}@cc.gatech.edu

Abstract

This paper provides an overview of the Common Object Oriented Library for Virtual Reality (COOL-VR) toolkit, a multi-platform, multithreaded toolkit that allows for easy adaptation to different hardware interfaces. Where the common definition of an object in a scene graph is limited to a graphical representation, this toolkit extends the idea of a scene graph to include objects with different and possibly multiple representations (e.g. audio, graphical etc). Each of these different representations is 'rendered' by an output device (screen, HMD, headphones, speakers etc). The toolkit allows the easy addition of new object representations as well as the addition of any new output devices needed for the output of these representations. COOL-VR also provides an easy interface for adding new file loaders that may be needed to load these new representations. In order to provide portability, the toolkit provides an abstraction layer that hides the inconsistencies between differing platforms, and provides the user with a single programming interface. Applications developed using the COOL-VR toolkit can be built on one platform and, without changing any code, be recompiled to run on a completely different platform (for instance from Unix to Windows). When running the application on a different platform, the toolkit maintains the look and feel of that new platform, but the interaction with the application stays the same across the different platforms.

Keywords: Virtual Environment, Object Oriented, Development Toolkit, Rendering, Hierarchical Object Model.