MAGE-VR: A Software Framework for Virtual Reality Application Development

Amit Mathew, Leonidas Deligiannidis
Department of Computer Science
University of Georgia
{mathew,ldeligia}@cs.uga.edu

Abstract

In this paper, we describe the architecture and implementation of MAGE-VR, a modular, high-performance software framework for developing virtual reality (VR) applications. MAGE-VR excels in developing highly realistic and interactive virtual environments (VE) by wrapping the specialized functionality of mature third-party libraries into one cohesive package. MAGE-VR is flexible because it allows the user to choose the libraries that are used to provide the underlying functionality through its plug-in architecture. The framework is extensible because it provides the means for the user to use their customized libraries with the framework. We also demonstrate that MAGE-VR significantly outperforms Java3D [8] and adds very little overhead to the libraries it encapsulates. The source code is freely available under the GNU Lesser General Public License (LGPL) [4].

Keywords: Virtual Reality, VR framework

1. INTRODUCTION

Virtual reality application development is complex because it requires the integration of many complex technologies such as graphics, input, audio, collision detection, physics, and haptic feedback. In designing a library that would integrate these technologies, we had several important goals that we felt were not fully met by other available frameworks. First, we understood that building our own versions of these technologies, such as graphics, would be re-inventing the wheel. Instead, we provide interfaces to third-party libraries that are already available and mature. Secondly, we wanted to give the users the flexibility to use libraries that we did not directly support to provide the underlying functionality of their VR applications. We addressed this issue by defining an interface that allows a user to write wrapper classes that encapsulate their favorite libraries. Thirdly, we wanted our system to support the latest technologies to give users the power to realize realistic and immersive worlds while still maintaining high frame rates. We accomplished this goal by encapsulating several modern, high-performance libraries without adding significant overhead. Fourthly, we wished to use high-level abstractions of the VE, so that the user is shielded from the complexities of the underlying libraries. Finally, we wanted users to view, modify, and distribute the source code of MAGE-VR, so we are releasing it under an open-source license.

2. RELATED WORK

Several other VR frameworks and platforms are available to developers, but in our opinion, do not achieve all the goals set forth for MAGE-VR. VR-Juggler [3] is an open source VR platform created by researchers at Iowa State University that provides support for many input and output devices. VR-Juggler on its own is fairly low-level, allowing the developer to input OpenGL commands. Through use of external libraries, like OpenSceneGraph [15], VR-Juggler gains some high-level abstractions, but these are not integrated into the core library. Furthermore, VR-Juggler lacks the advanced physics capabilities available in MAGE-VR.

Panda3D [16] is an open source game and simulation rendering engine, originally developed by Disney, and now maintained by the Entertainment Technology Center at Carnegie Mellon University. Although Panda3D is very mature and boasts many features, much of its functionality is built into the engine and modern technologies take longer to be integrated into the engine. For example, at the time of writing, support for advanced graphical techniques like normal mapping and sophisticated physics was absent from the engine. These features are

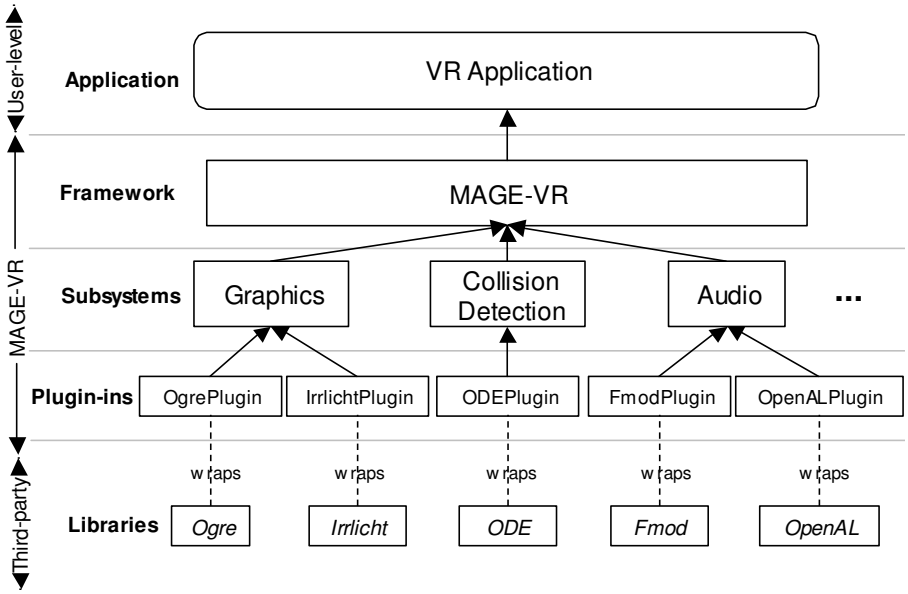


Figure 1: High-level architectural diagram of MAGE-VR

available in the libraries supported by MAGE-VR.

Although not exclusively made for VR, Java3D is popular among VR researchers because it offers high-level abstractions of world objects that allow developers to rapidly develop VEs. Java3D has functionality for graphics, audio, and collision detection, giving it a similar scope as MAGE-VR. Java3D, however, suffers from poor performance, especially when rendering large, complex worlds. We discuss our performance comparisons later in this paper. Furthermore, it lacks some of the advanced features provided by MAGE-VR, such as accurate collision detection, reflections, and shadowing.

3. OVERVIEW OF THE ARCHITECTURE

MAGE-VR is an object-oriented system written in C++ that creates an abstraction layer between the user and various subsystems that work together to create a realistic VE. Subsystems include graphics, input, audio, collision detection, and physics. Figure 1 shows how these subsystems use third-party libraries packaged as plug-ins to provide the implementation. The libraries at the bottom of the figure encapsulate low-level APIs and hardware. For example, the graphics libraries encapsulate OpenGL and DirectX. The plug-ins are loaded at run-time and, due to the abstraction layer, can be changed without recompiling the application by using configuration files. Also, the user is free to use only the subsystems that are required for their applications by loading only selected plug-ins at run-time.

At the application level, the framework is broken into two parts: (1) the static, or world, geometry that encloses the VE, and (2) scene nodes, which are used for movable, discrete elements of the world. Scene nodes retain transform information for the objects that are attached to them. A wide variety of objects can be attached to scene nodes such as meshes, lights, particle systems, cameras, collision proxies, and sounds. Figure 2 shows an example of a simple scene composed of objects attached to scene nodes. Attaching a mesh to a scene node gives the object a visual representation. For example, the *tree_mesh* from the figure would load a tree model into the scene node to which it is attached. Similarly, attaching sounds to a scene node gives the scene node an aural representation. Attaching collision primitives to the scene node creates bounding volumes that enable the scene node to report collision events between the world and itself. The scene nodes provide hierarchical transformations to the attached objects, so when the *motorcycle_node* is translated, its two tires and the associated graphical, aural, and physical components are moved as well.

MAGE-VR currently supports several libraries for providing the implementation of various

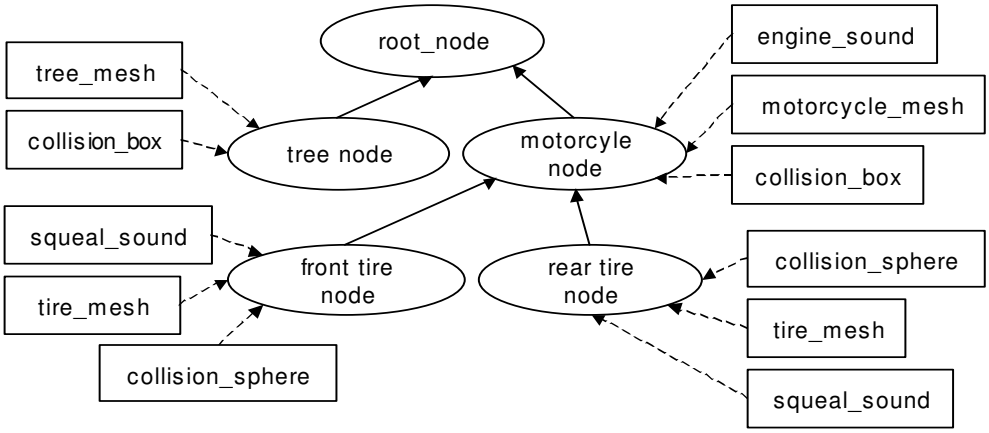


Figure 2: A simple scenegraph, with scene nodes in ovals and attachable objects in rectangles.

subsystems. OGRE [12] and Irrlicht [7] are supported open-source graphics engines that power MAGE-VR's graphics subsystem. We employ the open-source OpenAL [13] library and the commercial fmod [5] library for spatial audio and the open-source ODE [14] for collision detection and physics. For input, we use the keyboard and mouse functionality provided to us by the graphics libraries and we are currently porting our existing Java drivers for VR input devices, such as 3D trackers and gloves.

4. BENEFITS OF APPLICATION DEVELOPMENT WITH MAGE-VR

MAGE-VR provides numerous benefits to VR application developers by giving them more usability, power, and flexibility than ordinary VR libraries and toolkits.

Common Interface and Encapsulation

MAGE-VR's object-oriented architecture provides a common interface between the user and the underlying libraries that supply the functionality for the various subsystems. This architecture protects the user from changes in the libraries or from switching libraries. The user is only exposed to the common interface, which is consistent and predictable. Plug-ins also shield the user from the complexities of the underlying hardware. The user can easily change graphics cards, audio cards, or VR input devices without needing to modify their code.

Rapid Development

One of our main goals is to allow the user to rapidly develop VR applications. Our standard distribution contains support for a variety of VR input devices and several graphics, audio, and physics libraries. These subsystems provide the necessary components to build full-fledged VR worlds and are available in a single package, saving the user the hassle of downloading, building, and installing multiple libraries themselves. The example applications included with the distribution demonstrate the simplicity of creating MAGE-VR applications.

Furthermore, MAGE-VR provides a high-level abstraction of the VE, allowing the developer to control world objects directly and more efficiently. Figure 3 exhibits a code sample that demonstrates how a few lines of code can manipulate world objects in powerful ways. The code plays streaming music, adds and plays sounds, and creates and animates meshes that cast shadows.

Extensibility and Flexibility

Due to its plug-in architecture, MAGE-VR is both extensible and flexible. If the user has special needs that the supported libraries do not provide, the user can write plug-ins that wrap their own custom libraries. Similarly, users can easily add support for VR input devices that are not already supported. The only requirement is that new plug-ins and VR input devices conform to the MAGE-VR interface. Another benefit is that plug-ins allow developers to switch between libraries without having to rewrite their code. This is useful for testing different libraries or for enabling the user to change libraries during the development of their VR application if they find another library that suits their needs better.

Feature-Rich

By using third-party libraries to provide the subsystem functionality, the user is able to

```
audioSys->getStream("music")->play();

meshNode = gfxSys->getRootSceneNode()>createChild("robot");
Mesh* mesh = gfxSys->createMesh("robot", meshNode, me);
meshNode->setPosition(0, 0, 50);
mesh->setCastShadows(true);
mesh->getAnimation("walk")->setEnabled(true);

Sound* sound = audioSys->createSound("footsteps", meshNode,
"footsteps");
sound->setLoop(true);
sound->play();

lightNode = gfxSys->getRootSceneNode()->createChild("light0");
Light* light = gfxSys->createLight("light0", lightNode);
lightNode->setPosition(0, -50, 40);
```

Figure 3: Sample source code for creating a music stream, a mesh, a sound, and a light.

benefit from the features of these libraries. Through the use of third-party graphics libraries, we are able to support several shadow techniques, advanced shader support, particle systems, and several world types (indoor, outdoor, etc.). The physics libraries provide efficient collision detection, rigid-body simulation, and support for various types of joints. The audio libraries support spatial audio, Doppler effects, and streaming music. Individually, these libraries can take years to develop, but by harnessing the functionalities of pre-existing libraries, we are able to focus our development efforts on MAGE-VR.

With these additional technologies, we allow the users to experiment with new ways to create highly immersive environments. Consider the treatment of different phobias using VEs. Modern technologies provided by MAGE-VR's libraries can be used in interesting ways to allow patients to confront their phobias. Arachnophobia is a phobia that VEs can help treat [1]. For arachnophobia, shaders can be use to create realistic hair on spiders, a characteristic that makes them look more life-like. If a patient can overcome their fear of spiders in a VE where the spiders appear realistic, we believe it is more likely they will overcome their fear of spiders in real life. Another example is fear of flying [10]. Three-dimensional audio can be used to simulate engine noise and turbulence as the patient takes off in the virtual plane. This would increase the environment's realism and would more accurately simulate the situations that the patient must overcome in real life.

Many existing VR libraries concentrate on graphics, which is undoubtedly an important aspect of a VE. However, other technologies can have a substantial impact on immersion. Brooks notes in his survey of VR installations that audio can sometimes be more immersive than graphics and that collision detection and collision response were requested features [6].

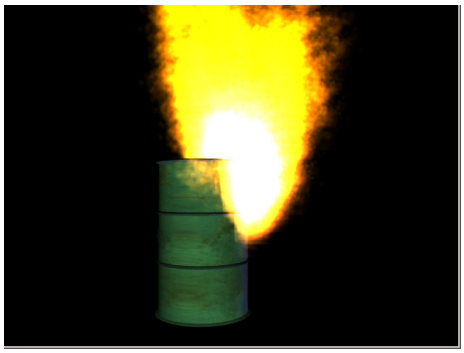
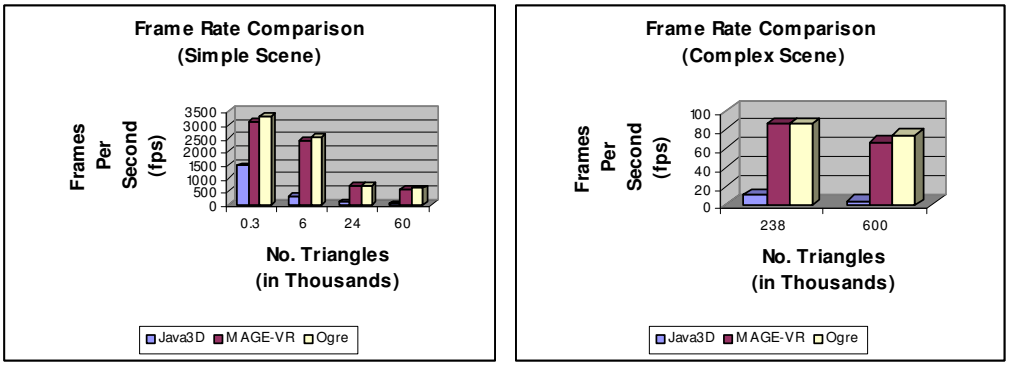


Figure 4: A scene showing realistic fire using particle systems. (Model courtesy of Platinum Pictures Multimedia)

High Performance

The performance of VR applications is critical for people immersed in the VE. Low performance will result in a drop in frame rate, which often causes VE users to become nauseated and disoriented [17]. Users in a VE cannot easily leave the environment, especially when using head-mounted displays. For this reason, high performance is an important characteristic for the third-party libraries that are supported by MAGE-VR. Our initial tests, detailed later in the paper, demonstrate that MAGE-VR can handle complex environments and special effects while still maintaining high



(a) (b)

Figure 5: Comparison of frame rates for Java3D, MAGE-VR, and OGRE for (a) simple scenes and (b) complex scenes.

frame rates. Figure 4 shows MAGE-VR rendering realistic fire using a particle system while still maintaining good frame rates.

Scene-Independent

MAGE-VR is designed to be scene-independent, meaning that it is impartial to the type of world the user wants to create. The user is free to create large outdoor scenes with vast landscapes, cramped indoor environments, underwater worlds, or scenes in outer space. Even though MAGE-VR does not prefer one scene type to another, the underlying graphics libraries can, and often do, optimize certain types of scenes.

Open Source

MAGE-VR is open source software, available under the GNU Lesser General Public License (LGPL) [4]. This license, in summary, allows the user to freely create and distribute applications with MAGE-VR, as long as changes to MAGE-VR itself are made available. This makes MAGE-VR suitable for use in non-commercial and commercial environments. Although the supported libraries are not all under open source libraries, we support at least open source library for each subsystem, giving the user the flexibility to use a completely open source solution. We support certain closed-source libraries if they provide more functionality than their open-source counterparts.

5. RESULTS

In this section, we describe the results of our tests comparing the frame rates of identical scenes rendered in MAGE-VR, Java3D, and OGRE. The purpose of these tests is to demonstrate the superior performance of MAGE-VR compared to Java3D. Also, we compare MAGE-VR to OGRE, the underlying library that powers MAGE-VR's graphics, and show that in most cases MAGE-VR adds little overhead. Since OGRE only handles graphics and basic input, MAGE-VR incurs a small performance penalty from adding an extra layer of abstraction, but allows it to handle VR input devices, audio, collision detection, and physics in a single integrated framework.

Frame rates are affected by other applications and operating system services running on the test machine. For this reason, frame rates represent an estimation of performance of the libraries being tested. Furthermore, we realize that graphics are just one part of a VR system that can cause latency, but we believe they are indicative of the relative performance between MAGE-VR and Java3D.

The test machine we used for our experimentations was a single-processor 3.2 GHz Pentium 4 with 1 gigabyte of RAM and a 256 MB Geforce 6800 graphics card. The operating system used was Windows XP. For the MAGE-VR tests, we used OGRE as the underlying graphics library and we disabled all other subsystems. For Java3D, we used the JWSU toolkit for handling basic scene creation and input handling [11]. For each test, we fixed the camera on one or more instances of a single model. The model was loaded in the OGRE mesh file format for MAGE-VR and OGRE and in VRML for Java3D. DirectX was used for all tests and the graphics are rendered in a 640 x 480 window with 32-bit color.

Our first four tests, shown in Figure 5a, compared the frame rates obtained while

rendering relatively simple scenes. In the first test, our scene was composed of one simple model made up of 308 triangles. In this test, MAGE-VR's frame rates are 2.11 times that of Java3D's. MAGE-VR added a modest overhead of 7% compared to OGRE's frame rates. The second test, with approximately 3000 triangles, showed MAGE-VR's frame rates to be 7.88 times that of Java3D's frame rate and with 4% overhead compared to OGRE by itself. The third test with about 24,000 triangles showed MAGE-VR 7.51 times the frame rate of Java3D and MAGE-VR's frame rate was 0.4% below OGRE's. The final simple test was a scene with 60,000 triangles and MAGE-VR maintained 15.49 times the frame rate as Java3D with an 8.4% overhead compared to OGRE.

Figure 6: One of the models used in the tests. It is composed of 6,000 triangles. (Model courtesy of De Espona Infografica)

Our final two tests, shown in Figure 5b, demonstrated complex scenes that have a large number of triangles on screen at once. In the first test, a scene with approximately 238,000 triangles on screen at once was used. The test showed MAGE-VR having 7.78 times the frame rate as Java3D with an overhead of less than 1%. The final stress test was a scene with 600,000 triangles, and MAGE-VR obtained 16.70 times the frame rate as Java3D with an overhead of 9.5%.

Our comparisons show that MAGE-VR outperforms Java3D in both simple and complex scenes and incurs little overhead over just using OGRE graphics. Furthermore, our tests show that MAGE-VR generally scales better than Java3D as the scene gets more complex. We believe this is an important factor that will allow VR application developers using MAGE-VR to implement highly detailed worlds while still maintaining high frame rates.

6. FUTURE WORK

MAGE-VR is under active development and we plan to include many more features to allow VR developers to fully realize their worlds. We intend to add drivers for additional VR input/output devices so that the user can test their application with multiple devices without modifying their code. We also plan to port MAGE-VR from Windows to Linux and SGI environments. We do not anticipate much difficulty with porting since the underlying libraries are all cross-platform and MAGE-VR does not have many platform-specific interactions. Next, we plan to add a GUI subsystem for displaying 2D widgets in the VE and a networking subsystem to provide high-level networking capabilities for distributed VR applications. Furthermore, we wish to give the users the flexibility to write code with MAGE-VR in their favorite language, without losing the performance benefits of C++. One way to accomplish this is to create language bindings for MAGE-VR. We initially plan to create bindings for Java and Python. Finally, since the process for transferring art assets to the development environment is often tedious, we intend to provide real-time, library-independent importing of a variety of model and scene formats.

7. CONCLUSION

We hope that MAGE-VR's features enable VR developers to craft complex worlds in new and interesting ways. We believe MAGE-VR represents the new generation of software driving VR development by supporting a variety of technologies through open source libraries. We are optimistic that its flexibility, extensibility, and openness will allow it to adapt to a variety of development environments.

REFERENCES

[1] Albert Carlin, Hunter Hoffman, Suzanne Weghorst, "Virtual reality and tactile augmentation in the treatment of spider phobia: a case report." *Behaviour Research and Therapy*, 35(2), 153-158, 1997.

[2] Allen Bierbaum, Christopher Just, Patrick Hartling, Kevin Meinert, Albert Baker, and Carolina Cruz-Neira, "VR Juggler: A Virtual Platform for Virtual Reality Application Development." IEEE VR 2001, Yokohama, Japan, March 2001.

[3] Christopher Just, Allen Bierbaum, Albert Baker, and Carolina Cruz-Neira, "VR Juggler: A Framework for Virtual Reality Development." 2nd Immersive Projection Technology Workshop (IPT98), Ames, Iowa, May 1998.

[4] GNU Lesser Public License. http://www.gnu.org/copyleft/lesser.html

[5] fmod. http://www.fmod.org

[6] Frederick Brooks, "What's Real About Virtual Reality?" *IEEE Computer Graphics and Applications* 19(6), 1999.

[7] Irrlicht. http://irrlicht.sourceforge.net

[8] Java3D. http://java.sun.com/products/java-media/3D/

[9] John Kelso, Lance E. Arsenault, Steven G. Satterfield, Ronald D. Kriz, "DIVERSE: A Framework for Building Extensible and Reconfigurable Device Independent Virtual Environments." Proceedings of IEEE Virtual Reality 2002 Conference.

[10] Larry F. Hodges, Barbara O. Rothbaum, Benjamin Watson, G. Drew Kessler, and Dan Opdyke, "A Virtual Airplane for Fear of Flying Therapy." Proc. VRAIS '96, IEEE Virtual Reality Annual Symposium (San Jose, April), 86-93.

[11] Leonidas Deligiannidis, Gamal Weheba, Krishna Krishnan, and Michael Jorgensen, "JWSU: A Java3D Framework for Virtual Reality." Proc. of the International Conference on Imaging Science, Systems, and Technology, June 2003.

[12] Object-Oriented Graphics Rendering Engine (OGRE). http://www.ogre3d.org

[13] OpenAL. http://www.openal.org

[14] Open Dynamics Engine (ODE). http://www.ode.org

[15] OpenSceneGraph. http://www.openscenegraph.org

[16] Panda3D. http://panda3d.org

[17] Randy Pausch, Thomas Crea, Matthew Conway, "A Literature Survey for Virtual Environments: Military Flight Simulator Visual Systems and Simulator Sickness." *Presence* 1(3), MIT Press, 1992, pp. 344-363.