

# Integrating an Open Source Game Engine Into a Virtual Reality Application Framework

## ABSTRACT

Increasingly, 3D game engine technology is being utilized in the virtual reality community. Game engines enable researchers and developers to quickly author scenes and easily exploit the rapidly advancing capabilities of PC 3D graphics cards. However, game engines lack key features required by virtual reality applications such as support for tracking, stereoscopic graphics, and multiple displays. We describe our experience extending an open source game engine, OGRE (Object Oriented Graphics Rendering Engine) [15] for use in our immersive virtual reality research project, FlatWorld. This task required the integration of the engine into an existing virtual reality application framework. Our completed system utilized OGRE for stereoscopic 3D graphics rendering, OpenAL [16] for audio processing, ACE [1] for network communications, and Boost.Spirit [3] for parsing configuration files. Our process of adding multi-screen, tracked stereoscopic display capabilities to the engine is also described. Furthermore, benefits and limitations of 3D games engines compared to dedicated VR toolkits are discussed. We also describe a prototype application built with our game engine based system.

## Categories and Subject Descriptors

I.3.7 [Three-Dimensional Graphics and Realism]: Virtual reality;  
I.3.2 [Graphics Systems]: Distributed/network graphics.

## General Terms

Design, Experimentation

## Keywords

Virtual reality systems and toolkits, education, entertainment, military applications

## 1. INTRODUCTION

The FlatWorld project [17] at the University of Southern California Institute for Creative Technologies is inspired by the Hollywood practice of building large stage sets using decorated modular wall units called “flats”. FlatWorld merges this practice with virtual reality technology by integrating immersive display screens with physical props to simulate rooms, buildings, and streets.

In 2001, a single room, proof of concept FlatWorld system was implemented. The OpenGL based prototype could not scale beyond this one room demonstration. The project quickly faced a need for a flexible software architecture which would allow using multiple screens in reconfigurable locations.

FlatWorld is intended to be relatively inexpensive. The hardware is based on low cost off-the-shelf projectors and consumer-grade PC's. The software component must also be affordable and accessible while facilitating the development of a broad range of entertainment, educational, and training applications with a level of visual realism equivalent to current interactive 3D games.

Below we describe an approach to build a FlatWorld software system which incorporates a 3D game engine into our prior developed virtual reality application framework. We conclude by mentioning the benefits and limitations of using game engine technology uncovered as a result of our experience.

## 2. RELATED WORK

Recently, game engines have begun to play a role in scientific research [11]. Quake and Unreal are popular choices as platforms for building simulations. These applications are authored by exploiting an engine's built-in scripting language. The underlying graphics engine is kept in closed proprietary form and the researcher cannot usually access the engine source code. One notable immersive display software solution, CaveUT [9], is built upon the Unreal Tournament engine. It is a set of modifications to the Unreal scripting language that sets correct viewing and perspective transformations for each display screen acting as a “Spectator”, where an acting character stays invisible on a server. CaveUT inherits the restrictions of the underlying Unreal engine. For example, it does not directly support stereoscopic rendering and tracking devices.

VR Juggler [5] is an effort to overcome limitations imposed by commercial and proprietary graphics engines by building an open-

source interface layer between VR application and hardware devices.

In describing the architecture of the VR Juggler platform, the authors reviewed the following toolkits for building VR applications: CAVELib, WorldToolkit, Avango, and Lightning. All of these solutions are oriented to popular immersive display systems such as the CAVE [6] and the responsive workbench [10]. In many cases, the developer is allowed to choose the 3D graphics rendering platform. It is up to the developer to customize the interface between the toolkit and the graphics software.

The FlatWorld software architecture is similar to VR Juggler in that it relies on open-source components, provides input device abstraction, and a platform independent networking API. Though VR Juggler would have been an appropriate platform to support our enhanced, game engine based FlatWorld system, using it would have required a significant reworking of our existing application code base.

### 2.1 Motivation

We were satisfied with our system’s device handling and network communications framework, and therefore we had no need to replace it. However, our proof of concept system used a simple OpenGL application library for rendering graphics which severely hindered our ability to present convincing 3D visuals. We did not have the ability to display complex animations, easily import content from 3D modeling programs, or use real time shaders, features common in current 3D game engines. Consequently, we decided to initiate a development effort to integrate a game engine into our existing FlatWorld virtual reality application framework.

## 3. SYSTEM COMPONENTS

In this section we describe the general architecture of the FlatWorld system to provide an understanding of the modifications necessary to utilize a 3D game engine for use in a virtual reality application.

### 3.1 Hardware

The FlatWorld system consists of the following components (Figure 1): stereoscopic rear-projection screens, surround sound speakers, floor shaking speakers, a position tracker, air blowing fans, strobe lights, and X10 sensors. The processing load is distributed among dedicated rendering machines, a tracker PC, and a central controller computer. As the system expands in the future, we have the option of adding additional computers to manage tasks such as vision sensor processing and speech input handling. The system is entirely PC based with all machines running Windows XP.

### 3.2 Software Architecture

All subsystems are implemented as separate processes, communicating via network messages. Our communications model is set up with one controller talking to many clients. The controller process provides a user interface which assigns display walls to virtual cameras and sends event data to clients. Clients are configured to perform specific roles. Thus far we have implemented clients to handle graphics display, audio processing, and X10 device control. The graphics display clients render scene content according to their assigned virtual cameras. Audio clients use OpenAL to provide dynamic 3D audio. Fans and strobe lights

are triggered by a client utilizing the X10 device automation protocol.

### 3.3 Passing Network Messages

Concurrent networking programming is a well studied area with several established software approaches [19]. We decided to use the popular ACE library in our implementation. ACE is open-source and portable across multiple platforms. On top of ACE we created NetMsg, a network message exchange library. NetMsg uses Loki [2] for implementing the message factory pattern and Boost for callback functions.

NetMsg features a design with one router (master) and many listeners (slaves). The router can broadcast messages to all registered listeners or to a single listener. Its multithreaded architecture handles automatic reconnection, message buffering, and application notifications via C++ functors. Messages are implemented as C++ classes that can be added to the system independently from each other as long as Message IDs are unique. For time critical events, delivery time can be assigned to each message. For computer clock synchronization we rely on the Network Time Protocol [13].

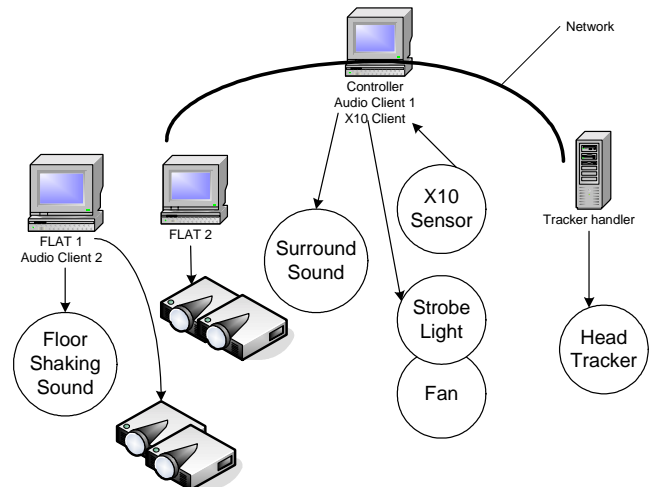


Figure 1. A typical FlatWorld hardware configuration

### 3.4 Graphics Rendering

Visual realism can play an important role in contributing to a user’s sense of presence. As a result, in a state of the art virtual reality application, the choice of graphics rendering engine is critically important. The engine should support hierarchical scene representation and modern graphics hardware APIs (namely, OpenGL and DirectX). It should be well documented and provide tools to support 3D modeling and animation packages. Additionally, the engine should be frequently updated to support new graphics card features. For our project, we selected OGRE (Object Oriented Graphics Rendering Engine) [15]. OGRE is well designed, object-oriented, and thoroughly documented. It is actively supported, and it has been used in a number of completed game projects.

### 3.5 User Interface

The FlatWorld controller user interface (Figure 2) allows an operator to assign display clients (flats) to virtual cameras, select scenes, control animations, and enable input devices. The

controller is implemented using an open-source cross-platform C++ user interface (UI) framework, wxWidgets [21].



**Figure 2. Controller user interface: OGRE 3D window shows a bird’s-eye view of the virtual scene. The view is selected using 2 radio-buttons in the upper-left portion of the screen.**

The OGRE generated 3D scenes are controlled by a sequence of states defined by a simple configuration file. Listing 1 provides an example of the syntax used to define a single action in the OGRE scene.

```

action
{
  id=1
  event { label="Scene"      int=1 }
  event { label="Car Smoke" int=1 }
}

```

**Listing 1. Fragment of actions configuration file.**

The action has a numeric ID, corresponding to a keyboard shortcut and events defining different actions in the scene. Each event corresponds to a button on the user interface. In this example, the controller interface’s “Scene” radio button is selected to 1 (City) and the “Car Smoke” button is set to a state 1 (pressed). To parse this file we used the Boost.Spirit library. It allowed the description of the syntax in C++, reducing development time and code maintenance problems (Listing 2).

```

revt = strp("event") >> '{' >> *tevt >>
}';
tact = rid | rname | revt;
ract = strp("action") >> '{' >> *tact >>
}';
tactions = *ract;

```

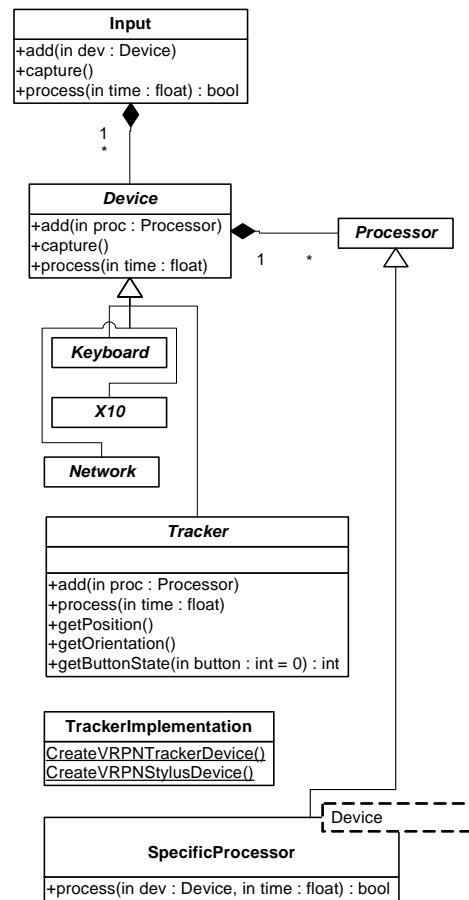
**Listing 2. Simplified extract from action configuration file syntax definition.**

### 3.6 Input Device Class Framework

Game engines are limited in that they only support common, consumer-grade input devices, such as joysticks, gamepads, keyboards, and mice. Microsoft’s DirectInput API provides uniform support for these devices.

VR systems often utilize highly specialized or custom built input devices such as head trackers, data gloves, and video sensors. To use OGRE or any other game engine in a VR application, it is usually necessary to link the engine to a specialized VR device handling system.

There are several software systems that provide an abstraction layer encapsulating different devices of the same type and uniform interfaces to them. These systems include VRPN [20], OpenTracker [18], and VR Juggler’s Gadgeteer. OpenTracker supports a variety of different tracking devices. VRPN and Gadgeteer each support different subsets of available inputs. Both Gadgeteer and OpenTracker treat VRPN interfaces as a device type. In a sense, VRPN has become a device interface standard in the VR development community. In all of these systems, the application determines how the device information is processed. FlatWorld applications include a simple device handling system designed to provide a plug-and-play architecture for device data processing. A simplified UML diagram of the system is shown in Figure 3.



**Figure 3. Simplified UML structure diagram of Input class framework hierarchy.**

Each input type is described by a descendant of the device class. The device state can be captured and processed. Also, a device class can be tailored to the capabilities and features of a specific device. For example, a 6DOF tracker device would include a GetPosition method. For each input device, one or more

implementations can be defined. Device input processing is conducted by a sequence of attached processors which are independent from a specific device implementation. This scheme facilitates re-use of device handling and processing code. Our system currently supports keyboards, joysticks, and mice using DirectInput, a 3rd Tech HiBall tracker via VRPN, and X10 devices using a serial protocol define by the X10 device manufacturer.

#### 4. STEREOSCOPIC GRAPHICS

Unlike traditional video games, immersive VR applications often require stereoscopic display on multiple networked screens.

FlatWorld utilizes the dual-screen capability of current NVIDIA video cards to render stereoscopic images displayed on two projectors with passive polarizing filters. To support this method of stereoscopic display, the game engine must provide two viewports within a single window to render left and right images to different halves of this window. The game engine must also support an off-axis view frustum and account for the observer's position. In order to create the stereoscopic illusion of objects floating in front of the screen, the engine must also allow the programmer to set the projection plane distance.

OGRE, being an open-source project targeted at traditional 3D game development, lacked these capabilities. However, OGRE's object oriented architecture and well written documentation allowed us to implement the needed stereoscopic display features in a relatively straight-forward manner. Our changes have been submitted to the OGRE open source community to support others who may want to use this game engine in virtual reality applications.

#### 5. CONTENT DEVELOPMENT

After completing our modifications and extensions to OGRE, we developed a demonstration application. The demonstration utilized two wall sized screens with tracked stereoscopic graphics. We authored 3D content using the Maya animation and modeling package. Though the OGRE community supplies Maya export plug-ins, we found that they lacked the ability to reliably preserve texture coordinates and animation paths.

Our final content development pipeline involved saving models as Maya files, converting them to the .OBJ format, and then using a shareware application to convert the .OBJ files into OGRE's native .mesh format.

We animated objects in the scene programmatically by setting keyframes using OGRE's animation class. Though this process was tedious, it provided satisfactory results.

#### 6. PROTOTYPE APPLICATION

Our completed OGRE based application was intended to illustrate our system's basic capabilities as an urban combat training tool. Our application's virtual world consists of an urban combat environment complete with vehicles, soldiers, and debris (Figure 4).



**Figure 4. City environment used in our OGRE based FlatWorld demonstration application.**

Wearing polarized glasses and a 3<sup>rd</sup>Tech HiBall position tracking device, the user views the exterior world by opening a real window and door (Figure 5). Using the controller user interface (Figure 2), a behind the scenes operator can trigger a wide variety of events. For example, an operator can change the time of day, order helicopter flyby's, trigger explosions, or start thunderstorms. These events are intensified through the use of fans, strobe lights, and floor shaking speakers controlled using our framework's X10 interface. The user can request unmanned aerial vehicle (UAV) flights to conduct reconnaissance missions. The UAV's virtual camera is viewed on a notebook computer allowing the user to detect potential threats in the exterior urban environment.



**Figure 5. User viewing OGRE scene presented on two wall sized displays with physical door and window props. An unmanned aerial vehicle (UAV) appears in the window.**

#### 7. BENEFITS AND LIMITATIONS

Based upon our experience with OGRE, we found that game engines can serve as an adequate foundation for developing virtual reality applications. However, game engines require substantial modifications to support tracking, stereoscopic graphics, and other features necessary for creating an immersive virtual environment. Though OGRE lacked the high level content

authoring tools available with commercial game engines, it was well documented, provided source code and it performed reliably. More importantly, it was free, and it is supported and updated by an active community of developers.

## 8. FUTURE WORK

A primary advantage of using game engines is that they are updated continuously to take advantage of the ever expanding feature set of PC 3D graphics cards. In our future work with OGRE we would like to develop a virtual environment which fully exploits OGRE's real time lighting and shading capabilities. We feel that the resulting enhancement in visual realism will significantly improve our system's ability to immerse users in the virtual world. Furthermore, to facilitate easier content creation, we plan to use LUA [8] or another equivalent language to implement a scenario scripting capability.

## 9. ACKNOWLEDGMENTS

The authors would like to thank Alex Okita, Ernie Eastund, and LoQuan Seh for creating the art and animation used in our prototype application.

The effort depicted was sponsored by the U.S. Army Research, Development, and Engineering Command (RDECOM). The content of this information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

## 10. REFERENCES

- [1] The Adaptive Communications Environment (ACE), <http://www.cs.wustl.edu/~schmidt/ACE.html>.
- [2] Alexandrescu, A. *Modern C++ Design: Generic Programming and Design Patterns Applied*, Addison-Wesley 2001.
- [3] Boost, free peer-reviewed portable C++ source libraries, <http://boost.org>.
- [4] CaveUT 2004 v1.1 Freeware for Low-Cost Integrated Multi-Screen Displays Using Unreal Tournament 2004, <http://planetjeff.net/ut/CaveUT.html>.
- [5] Cruz-Neira, C., Bierbaum, A., Hartling, P., Just, C., and Meinert, K.. VR Juggler – An Open Source Platform for Virtual Reality Applications. In *Proceedings of 40th AIAA Aerospace Sciences Meeting and Exhibit 2002*, Reno, Nevada, January 2002.
- [6] Cruz-Neira, C., Sandin, D. J., and DeFanti, T. A. Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. In *Computer Graphics Proceedings*, Annual Conference Series, ACM SIGGRAPH, 1993, 135-142.
- [7] CYVIZ 'Stereo 3D converters': <http://www.cyviz.com/products.html>
- [8] Ierusalimschy, R., de Figueiredo, L. H., and Filho, W. C. "Lua-An Extensible Extension Language, *Software: Practice & Experience* 26 #6, 1996, 635-652.
- [9] Jacobson, J. and Lewis, M. Game Engine Virtual Reality with CaveUT, *IEEE Computer*, 2005/Vol.38, No. 4.
- [10] Krueger, W. and Froehlich, B. The Responsive Workbench. *IEEE Computer Graphics and Applications*, May 1994, 12-15.
- [11] Lewis, M. and Jacobson, J., Game Engines in Scientific Research, *Communications of the ACM*, January 2002/Vol.34, No. 1.
- [12] Loki, C++ library, <http://sourceforge.net/projects/loki-lib/>.
- [13] NTP: The Network Time Protocol, <http://www.ntp.org/>.
- [14] NVIDIA 3D Stereo, [http://www.nvidia.com/object/3d\\_stereo.html](http://www.nvidia.com/object/3d_stereo.html).
- [15] OGRE, Object-oriented Graphics Rendering Engine, <http://www.ogre3d.org>.
- [16] OpenAL, Cross Platform 3D Audio Library, <http://www.openal.org>.
- [17] Pair, J., Neumann, U., Piepol, D., and Swartout, B. FlatWorld: Combining Hollywood Set Design Techniques with VR. *IEEE Computer Graphics and Applications*. January/February 2003, 12-15.
- [18] Reitmayr, G. and Schmalstieg, D. An Open Software Architecture for Virtual Reality Interaction. In *Proceedings of the 2001 ACM Symposium on Virtual reality Software and Technology*, November 2001, 47-54.
- [19] Schmidt, D., Huston, S. *C++ Network Programming*, Addison-Wesley, 2003.
- [20] Taylor II, R. M., Hudson, T. C., Seeger, A., Weber, H., Juliano, J., and Helser, A. T. VRPN: A Device-Independent, Network Transparent VR Peripheral System, In *Proceedings of the 2001 ACM Symposium on Virtual reality Software and Technology*, November 2001, 55-62.
- [21] wxWidgets Cross Platform GUI Library, <http://www.wxwindows.org/>.