

2011

Developing virtual reality applications: The design and evaluation of virtual reality development tools for novice users.

David J. Kabala
Iowa State University

Follow this and additional works at: <https://lib.dr.iastate.edu/etd>

 Part of the [Computer Sciences Commons](#)

Recommended Citation

Kabala, David J., "Developing virtual reality applications: The design and evaluation of virtual reality development tools for novice users." (2011). *Graduate Theses and Dissertations*. 10231.
<https://lib.dr.iastate.edu/etd/10231>

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

**Developing virtual reality applications:
The design and evaluation of virtual reality development tools for novice users**

by

David Kabala

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Major: Human Computer Interaction

Program of Study Committee:
Julie Dickerson, Major Professor
Eve Wurtele
Stephen Gilbert
David Fernández-Baca
Sunghyun Kang

Iowa State University

Ames, Iowa

2011

Copyright © David Kabala, 2011. All rights reserved.

DEDICATION

For Amber, Mom, Sarah, and Tony.

TABLE OF CONTENTS

LIST OF TABLES	viii
LIST OF FIGURES	ix
LIST OF ALGORITHMS	xi
ACKNOWLEDGEMENTS	xii
ABSTRACT	xiii
CHAPTER 1. GENERAL INTRODUCTION	1
Motivation	1
Research Goal	1
Dissertation Overview	1
Challenges	2
Single GUI framework for desktops and clustered VR environments	2
Support for real-time 3D application features in clustered environments	2
Expert user interface for creating virtual environments of scenes	2
Novice user interface for creating virtual environments	3
Background	3
Immersive Virtual Reality	3
VR Systems	4
Software Licenses	6
References	7

CHAPTER 2. OPENSNGTHOOLBOX: A TOOLKIT FOR EFFICIENT DEVELOPMENT OF 3D USER INTERFACES FOR VIRTUAL REALITY APPLICATIONS

APPLICATIONS	8
Abstract	8
Introduction	8
Background	9
VR System Architectures	9
Scene Graphs	10
Related Work	11
GUIs	11
VR Toolkits	12
Game Engines	13
Scene graphs	13
Implementation	13
OpenSGToolbox with scene graph	16
Event-driven programming	16
Reflexive event production definition	17
Look-and-feel	18
Simulation of WIMP-like input in 6-DOF tracker based systems.	18
Cross-platform	19
Limitations	20
Sample Applications	20
Conclusion And Future Work	21
Acknowledgments	21
References	22

CHAPTER 3. KABALA ENGINE: A VIRTUAL ENVIRONMENTS AUTHORIZING FRAMEWORK FOR NOVICE USERS

Abstract	23
--------------------	----

Introduction	23
Related Work	24
Modules	25
Animation	25
Morphs and Deformable Geometries	27
Advanced Particle System	27
Importing Digital Assets	30
Scripting Language	30
Sound	31
Video	32
Physics	32
User Interface	33
Generic FieldContainer GUI	33
KabalaEngine Architecture and Interfaces	36
Project Player	36
Project, Scenes, SceneObjects, Effects, Behaviors	36
Control Loop	38
Run-time debugger	39
Content View Panel	40
Scene graph GUI tree	41
Utility Tab Panel	41
Main Menubar	41
Conclusions	42
Future Work	43
Acknowledgements	43
References	43

CHAPTER 4. CREATING A GRAPHIC USER INTERFACE FOR VIRTUAL REALITY AUTHORIZING: AN OVERVIEW AND ANALYSIS OF THE KABALAENGINE WORLD BUILDER	45
Abstract	45
Introduction	45
Related Work	47
Design Principles	48
KabalaEngine Builder Interface	48
Materials and Methods	56
Overview of the Study	56
Apparatus	57
Procedure	57
Measures	58
Participants	58
Results	58
Created Users Projects	60
Discussion	62
What Worked Well?	62
What Did Not Work?	63
Limitations and Future Work	64
Acknowledgements	64
References	64
CHAPTER 5. EXAMPLE WORKS THAT USE OUR CONTRIBUTIONS	66
Overview	66
MetaBlast	66
Quarterback Development System	68
CHAPTER 6. GENERAL CONCLUSIONS	70
Challenges Addressed	70

Single GUI framework for desktops and clustered VR environments	70
Support for real-time features in clustered environments	71
Expert UI for creating virtual environments of scenes	71
Novice UI for creating virtual environments of scenes	72
Outcome Concerning Research Goal	72
Future Research	72
Limitations	73
Final Thoughts	74
APPENDIX A. NOVICE USER SURVEY	75
APPENDIX B. KABALA ENGINE BUILDER USER SURVEY RAW DATA	78
APPENDIX C. NOVICE USER TASKS	83
APPENDIX D. KABALA ENGINE BUILDER INTERFACE SKETCHES	91

LIST OF TABLES

Table 3.1	Particle properties	28
Table 3.2	Asset types	37
Table 3.3	Project Events	37
Table 3.4	Scene Events	37
Table 3.5	Specific Types of SceneEffects	38
Table 3.6	Scene Object Effect Events	38
Table 4.1	3D Scene View Panel Toolbar Buttons	51
Table 4.2	Main Toolbar Buttons	56
Table B.1	Survey Questions	79
Table B.2	Survey Results	79
Table B.3	Survey Results (continued)	80
Table B.4	Survey Results (continued)	81
Table B.5	Survey Results (continued)	82
Table B.6	Survey Results (continued)	82

LIST OF FIGURES

Figure 2.1	A four-sided Virtual Reality system.	11
Figure 2.2	Graph structure of a scene graph used to represent a 3D scene.	12
Figure 2.3	Example GUI components supported in the OpenSGToolbox user interface toolkit	14
Figure 2.4	Complex GUI components supported in the OpenSGToolbox user interface toolkit	15
Figure 2.5	Example 3D Graphical User Interface.	17
Figure 2.6	Examples of the OpenSGToolbox user interface toolkit used in other applications.	21
Figure 3.1	Class hierarchy of Animations	26
Figure 3.2	Class Hierarchy of Deformable Geometries and Morphs	28
Figure 3.3	Class hierarchy of Particle System	29
Figure 3.4	Class hierarchy of Sounds	32
Figure 3.5	Class hierarchy of Video	33
Figure 3.6	Class hierarchy of Physics	34
Figure 3.7	Generic FieldContainer Editing Interface	35
Figure 3.8	Class hierarchy for Project	38
Figure 3.9	KabalaEngine Debug Interface	40
Figure 4.1	KabalaEngine Builder Interface	49
Figure 4.2	Transformation Direct Manipulation Tools	52
Figure 4.3	Transformation Direct Manipulation Tools	53

Figure 4.4	Transformation Direct Manipulation Tools	54
Figure 4.5	Participant Experience	58
Figure 4.6	Task Survey	59
Figure 4.7	Project Realism	59
Figure 4.8	Software Freetime Use	60
Figure 4.9	Project created by a single participant in 50 minutes	61
Figure 4.10	Scenes created by participants during 50 minute task	62
Figure 5.1	Inside a Leaf Cell in the Metablast Application	67
Figure 5.2	The Main Menu of the Metablast Application	67
Figure 5.3	An Information Log GUI in the Metablast Application	68
Figure 5.4	The QDS Playbook Manager Interface	69
Figure 5.5	Warping the video playback in QDS	69
Figure D.1	Version 1	92
Figure D.2	Version 2	92
Figure D.3	Version 3	93
Figure D.4	Version 4/Final	93

LIST OF ALGORITHMS

Algorithm 3.1 Window main loop pseudocode	39
---	----

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those that have helped me on my journey. Firstly, I wish to thank and express my deepest gratitude to my advisor Dr. Julie Dickerson for providing me a tremendous graduate education experience. I owe a major share of my success to her constant encouragement, continuous support, and her belief in my abilities. I am also very grateful for having a wonderful doctoral committee and wish to thank Drs. Stephen Gilbert, David Fernández-Baca and Sunghyun Kang for providing me valuable input.

A great deal of thanks is due to the Virtual Reality Application Center (VRAC) at ISU. The staff have been tremendously helpful with organizing and with fixing technical problems that have cropped up along the way. The facilities at Virtual Reality Application Center (VRAC) have been world class and extremely useful.

Thanks also go to my work colleagues on Metablast: Achyuthan Vasanth, Daniel Guilliams, Robert Goetz, Eric Langkamp, Will Schneiller, Amy Dixon, PJ Campbell, and David Naylor. You provided a supportive and positive work environment. Thanks goes out to my friends, especially Karen. You have helped me stay on course, and given me time to relax in the fleeting moments when it was possible.

Finally, I would like to thank my wife Amber, mother, sister Sarah, and brother Tony. My wife, you have been there for me at all those times that no one knows about that kept me going. Mother, you have supported me in all my endeavors, I can still remember that your's was the voice that I could always hear when I would run a race. Sister, you are the reason I started working a VRAC; you have provided me with intelligent critique of ideas. Brother, you have solidified our families cohesion with your tireless work for Mom, Sarah, and myself.

ABSTRACT

Developing applications for Virtual Reality(VR) systems is difficult because of the specialized hardware required, complexity of VR software, and the technical expertise needed to use both together. We have develop tools and applications that support the authoring of virtual reality applications. The tools will support development of VR applications based on common requirements of the hardware and architecture used in VR systems.

We developed support for animations, geometry morphs, deformable geometry, advanced particle systems, importing digital assets, embedding a scripting language virtual machine, sound library wrappers, video library wrappers, and physics library wrappers for the OpenSG framework. The KabalaEngine was developed to use the supporting libraries previously mentioned in a clustered VR system using OpenSG's clustering capabilities. The KabalaEngine has an expert graphical user interface that can be used for developing virtual environments. Finally, we developed a graphical user interface for novice users of the KabalaEngine. We found that users of the KabalaEngine were able to use the interface to produce three different complex virtual environments with 10-15 different 3D objects arranged in a meaningful way in fifty minutes.

CHAPTER 1. GENERAL INTRODUCTION

Motivation

Virtual reality applications are useful. They have been used for psychiatric treatment, training skilled disciplines, prototyping architecture, teaching visual-spacial skills, teaching history, entertainment, and many other applications. Developing applications for Virtual Reality systems is difficult because of the hardware required, complexity of Virtual Reality software, and the technical expertise required to use them. The scope of users that could utilize virtual reality systems could be greatly increased with better development tools for expert and novice users.

Research Goal

The goal of this research is to develop tools and applications that support the authoring of virtual reality applications. The tools will support development of VR applications based on common requirements of the hardware and architecture used in VR systems. Building on these tools we will develop an application for novice and expert users to develop virtual reality applications using a graphical user interface. The goal of the application for novice users is to reduce the time and technical skill required to develop VR applications.

Dissertation Overview

This dissertation is organized as a series of three manuscripts that have been submitted or will be submitted to peer-reviewed journals and conferences. Chapter 1 frames the motivation and goals of the research, and provides a background of virtual reality and why it is important.

Chapter 2 is a manuscript describing the design and implementation of a 3D graphical user interface library for OpenSG. Chapter 3 is a manuscript of the design and implementation of the KabalaEngine, an application for authoring virtual environments, and supporting libraries for the KabalaEngine. Chapter 4 is a manuscript of the design and evaluation of a novice graphical user interface for the KabalaEngine. Chapter 5 describe works that have used the contributions we've developed. Chapter 6 discusses the general conclusions that can be made from these works, limitations, and future work that could follow.

Challenges

Single GUI framework for desktops and clustered VR environments

Creating a graphical user interfaces that can be used on desktop and VR systems is difficult. There are many mature libraries for graphical user interfaces for desktop systems, and there are some that can be used in 3D. However, there is a need for graphical user interfaces that can be used in both desktop and VR systems. The primary difficulty is handling clustered architectures used by many VR systems.

Support for real-time 3D application features in clustered environments

There are many common features of real-time applications that are not directly supported for clustered VR systems. These include animations, morphs, deformable geometries, particle system, physics, sound, video, importing assets, and scripting. OpenSG is a scene graph library that directly supports efficient clustered VR application development. We will detail our contribution to OpenSG to support the previously mentioned features.

Expert user interface for creating virtual environments of scenes

If the challenges for supporting clustered graphical user interfaces and the other previously mentioned features are met, then those feature can be used to create a application to support expert users creating virtual environments. The application can support expert developers by

moving the definition of content and behavior from compiled code to data that is created using a graphical user interface and saved.

Novice user interface for creating virtual environments

It is a goal of virtual reality research to make authoring virtual environments easier. Ideally, anyone that has an idea for a virtual environment should be able to create that environment easily. There are many barriers to this: the hardware is expensive and complicated, the software for running the hardware is also expensive and/or complicated, multimedia content is expensive or nonexistent, and there is not a large community of novice virtual reality developers. Novice developers could be supported by an application that uses a graphical user interface to simplify many of the complex tasks required.

Background

Immersive Virtual Reality

Virtual reality systems were first introduced in the 1960's when a cinematographer released a system called Sensorama Simulator. This device used stereo video of riding a motorcycle, stereo audio, haptic feedback through vibrations, and wind through fans. This resulted in the first serious work on virtual reality.

Virtual Reality has been applied to many problems. Virtual reality attempts to create immersion in a virtual world to help solve some problems. Immersion is realized by providing input to users that gives an experience of being inside a real place and situation outside of their actual location and situation. To create an immersion experience, users are presented with input to some subset of their senses; visual, audial, touch (pressure, temperature, and pain), smell, and taste. The simulation is also an important aspect of immersion. This means providing an environment that behaves in an expected manner, gravity makes things fall, people don't walk through walls, fires burn things and produce smoke, and many other things. Interaction is also important; users need to be able to effect the environment in order to feel immersed in it.

Virtual reality has been demonstrated to be useful in the following disciplines.

- Simulations for military and civilian training when training under the actual circumstances would be too dangerous, costly, or literally infeasible[Taffinder et al. 1998].
- Psychiatric treatment[Krijn et al. 2004].
- Teaching visual spatial skills.
- Prototyping large architectural and construction projects.
- Entertaining through video games, interactive storytelling, or artistic/cultural exhibits.
- Gaining new insights on data through data visualization.
- Providing cathartic experiences to provide stress relief.
- Teaching historical or contemporary fact through virtual recreations of environments, civilizations, cultures, and structures that no longer exist or cannot be experienced directly.

VR Systems

The hardware used for virtual reality systems is varied. There is specialized hardware for visual, audio, touch, smell, and taste displays. The hardware used in VR systems varies by the scope, level of immersion, cost, and technology available.

Visual Display Visual displays can range in size from very small, 1 cm, to very large > 10 meters. A small project may use a PC and monitor; this is relatively inexpensive but does not provide the depth of immersion that more complex systems offer. Recently 3DTV technologies have brought 3D, i.e. stereo displays, to consumer level HDTVs. However, this is little different than using a larger monitor as most HDTVs still do not cover a large field of view. Head mounted displays (HMDs) provide more visual immersion by presenting separate images to each eye and also suffer from a small field of view. Larger displays use projectors and combinations of smaller visual displays.

Multi-visual display systems combine and synchronize a collection of visual displays. A CAVE system is a multi-visual display arrangement that encloses, or partially encloses, users in a room where the walls of the room are visual displays. The visual displays in CAVEs cover all or most of the users field of view. CAVEs require a large space footprint, complex construction, and expensive projection hardware. A theater arrangement uses a screen setup similar to cinema theaters, but often with higher end projectors in order to display stereo images. The expense and complexity of large and high fidelity visual displays keeps them out of reach at the consumer level. This makes VR development more difficult because the VR systems are difficult to access.

Audio Display The complexity of an audio system depends on the frequency range, clarity, amplitude, and 3D positioning support of the display. To handle 3D positioning of audio more speakers are needed to increase the number of physical positions that virtual sounds can come from. Additionally more computation is needed as more speakers are added; so that the correct audio output is sent to each speaker. The decibel output of the audio system scales with the number of simultaneous users that a system is designed for. The expense and complexity of large and high fidelity audio displays is not as high as for visual displays. However, this still makes VR development more difficult because the VR systems are difficult to access.

Computation The computational power of the system running a VR environment is another important hardware factor for VR systems. Large visual displays can only be fully utilized if they receive a high fidelity, low latency stream of graphical output from the computers running the VR application. The computational component of VR systems receive input from the input devices, process that input via computation, and then package and stream output to all of the output displays. VR systems organize the computation hardware may be either one large server or a clustered collection of smaller computers.

VR systems that use a cluster of computers, synchronized with software and hardware to run a VR application. Clusters can scale to larger display sizes because several computers can

render a separate view of the scene in parallel. The cost of a clustered system is significantly larger than a desktop system with just a single computer.

Input Input for VR systems has primarily been concerned with tracking the position and orientation of users. This information is required for immersive stereoscopic display when calculating the correct viewpoint for each eye. 6-degrees of freedom (6-DOF) tracking, 3D position and orientation, are necessary for this calculation. There are four main types of devices that have been produced for 6-DOF tracking; magnetic, sonic, gyroscopic, and image based. Recently there has been research into using low cost cameras and visual fiducials in a fully enclosed cave for tracking []. This technique has been shown to be accurate, with moderate latency. Using visual fiducials is currently limited by the requirement to be used in fully enclosed cave environments, and additional processing on the image data from the camera.

Other than tracking devices, VR environments use common computer interaction devices, the keyboard and mouse being the most prevalent. There is also use of touch screen devices (tablets), gamepads, 3-degree of freedom trackers (Wii mote, PS3 controller), and microphone (audio processing).

Software Licenses

When considering the use of software and libraries for development of VR applications, it's important to understand how the product it is affected by the licensing. The licenses of dependent libraries and applications can be barrier to application not only in understanding the licenses but also in the legal requirements imposed by them. Together this can decrease the incentives for developing applications. Bruce Perens describes four main categories of software licenses: proprietary, gift, sharing with rules, and in-between licenses [Perens 2009].

Proprietary software is licensed such that it may not be modified or used in another software package, doing so would be copyright infringement. The Open-source gift licenses like the Apache license [Apache] allow modification and use of the software in any derivative work including proprietary software. Open-source sharing with rules licenses allow modification and

use of software as long as the derivative work is also shared. The General Public License version 3(GPL3) [GPL3] is an example of an open-source sharing with rules license. Open-source “in-between” licenses like the Lesser General Public License version 3(LGPL3) [LGPL3] allow modification and use of the software in derivative work, including proprietary software, with the condition that the original software code be made available with the derivative work.

There may be software that provides the functionality needed in a new application, but because of licensing, that software may not be legally usable. This makes the general use of proprietary software and systems inaccessible to many. But under some open-source licenses, there is legal ground for users to use, modify, and share derivative work. All of the contributing libraries and software we are presenting are released under the LGPL and GPL.

References

- KRIJN, M., EMMELKAMP, P. M. G., OLAFSSON, R. P., AND BIEMOND, R. 2004. Virtual reality exposure therapy of anxiety disorders: A review. *Clinical Psychology Review* 24, 3, 259–281.
- PERENS, B. 2009. How many open source licenses do you need?
- TAFFINDER, N., SUTTON, C., FISHWICK, R. J., MCMANUS, I. C., AND DARZI, A. 1998. Validation of virtual reality to teach and assess psychomotor skills in laparoscopic surgery: results from randomised controlled studies using the mist vr laparoscopic simulator. *Studies in health technology and informatics* 50, 124-30,
- .

CHAPTER 2. OPENSNGTHOOLBOX: A TOOLKIT FOR EFFICIENT DEVELOPMENT OF 3D USER INTERFACES FOR VIRTUAL REALITY APPLICATIONS

A paper submitted to *UIST 2011*

David J. Kabala^{1 2}, Julie Dickerson³

Abstract

Graphical User Interfaces(GUI) are difficult to develop for clustered Virtual Reality(VR) applications, because GUI toolkits do not distribute the rendering and input between the computers in the cluster. Because of this, GUIs are often coded separately for each VR application. OpenSGToolbox is an open-source, cross-platform graphical user interface toolkit that can render a 3D projection of a GUI across a clustered VR system. The toolkit has been designed to support an event-driven programming model, data and method reflexivity, and a wide variety of input devices common to both VR and desktop systems. We will present the methods we used to develop a GUI toolkit using the cluster-supporting scene graph library OpenSG.

Introduction

Developing a VR application that has a GUI is difficult. VR systems can widely vary on their computing architecture, display system, input devices, and the underlying operating system used. Developing a GUI for a VR application can be inflexible if moved to a different VR

¹Human Computer Interaction Program, Iowa State University, Ames, Iowa

²Author for correspondence

³Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa

system than the original. The requirements for the OpenSGToolbox are to efficiently synchronize a 3D GUI over clustered system, common to VR systems. It is important to draw the the GUI as a 3D object in VR systems because there are often multiple displays in the system that are not on the same physical plane. Drawing the GUI on a plane within the 3D virtual space allows the GUI to be rendered with the correct projection across displays. There are toolkits that make this easier for 3D content, but not for GUIs. Using the existing technology OpenSG for developing clustered VR applications, we have developed the OpenSGToolbox user interface toolkit. The OpenSGToolbox user interface toolkit that helps users develop GUIs that can be used across a wide array of clustered VR systems and desktop environments.

In the following sections we will give background information on VR architecture and it's differences with single-computer systems. The toolkits available for GUI and VR development will be evaluated. We will describe the design and implementation of the OpenSGToolbox user interface toolkit and how it can be used to develop VR applications.

Background

VR System Architectures

Many virtual environment systems connect multiple visual displays together that must be synchronized. A CAVE system is a multi-display arrangement that encloses, or partially encloses, users in a room where the walls function as displays[Cruz-Neira 1995][Cruz-Neira et al. 1993]. The displays in CAVEs cover all or most of the user's field of view.

VR systems can be comprised of a large number of displays. These systems can be limited by the display bandwidth required to fill the resolution of all of the displays at an interactive rate. Solutions to this have involved a single powerful server or a distributed cluster of computers. Single server configurations have fallen out of favor because of cost, complexity, and availability. Clusters have the advantage of employing consumer-level hardware to provide the displays with the visual information. However, clustered VR systems must synchronize the input and output to and from the nodes in the cluster. This in turn can lead to additional bandwidth issues.

The computational power of the platform running a Virtual Environment is another important hardware factor for VR systems; large displays can only be fully utilized if they receive a high quality, low latency stream of graphical output from the computers running the VR application. The computational component of VR systems receive the input from the input devices, process that input in the VR application, and then stream output to all of the output devices. Many VR systems use a cluster of computers, which are synchronized with software and hardware, to run a VR application. Clusters can scale to larger display sizes because several computers can share the load of rendering different parts of a scene in parallel.

Input for VR systems has primarily been concerned with tracking the position and orientation of users. This information is required for immersive stereoscopic display when calculating the correct viewpoint for each eye. Six degree-of-freedom (6-DOF) tracking devices provide the 3D position and orientation data that are necessary for this viewpoint calculation. Other than tracking devices, VR environments use common computer interaction devices such as a keyboard, mouse, touch screen devices, gamepads, three degree-of-freedom trackers (Wiimote, PS3 controller), and microphones (audio processing). Figure 2.1 shows an example of a four-sided cave with four projectors and a tracking device in the users hand.

Scene Graphs

Given the complexity of the visuals created for VR applications scene graphs are often used for efficiency. Scene graphs are graphs that are used to logically and/or spatially represent graphical objects in a scene. Scene graphs are collections of nodes that are connected as a graph and in some cases a tree data structure. Scene graphs can achieve greater efficiency by traversing the graph and only rendering or checking for collision with those nodes that intersect with the view volume or collision geometry. Often times these lead to large efficiency gains, because all of the geometry outside of the view volume is not pushed further through the rendering pipeline. Figure 2.2 shows an example of a scene graph for a robot scene with nodes containing transformations, geometry, and children nodes.

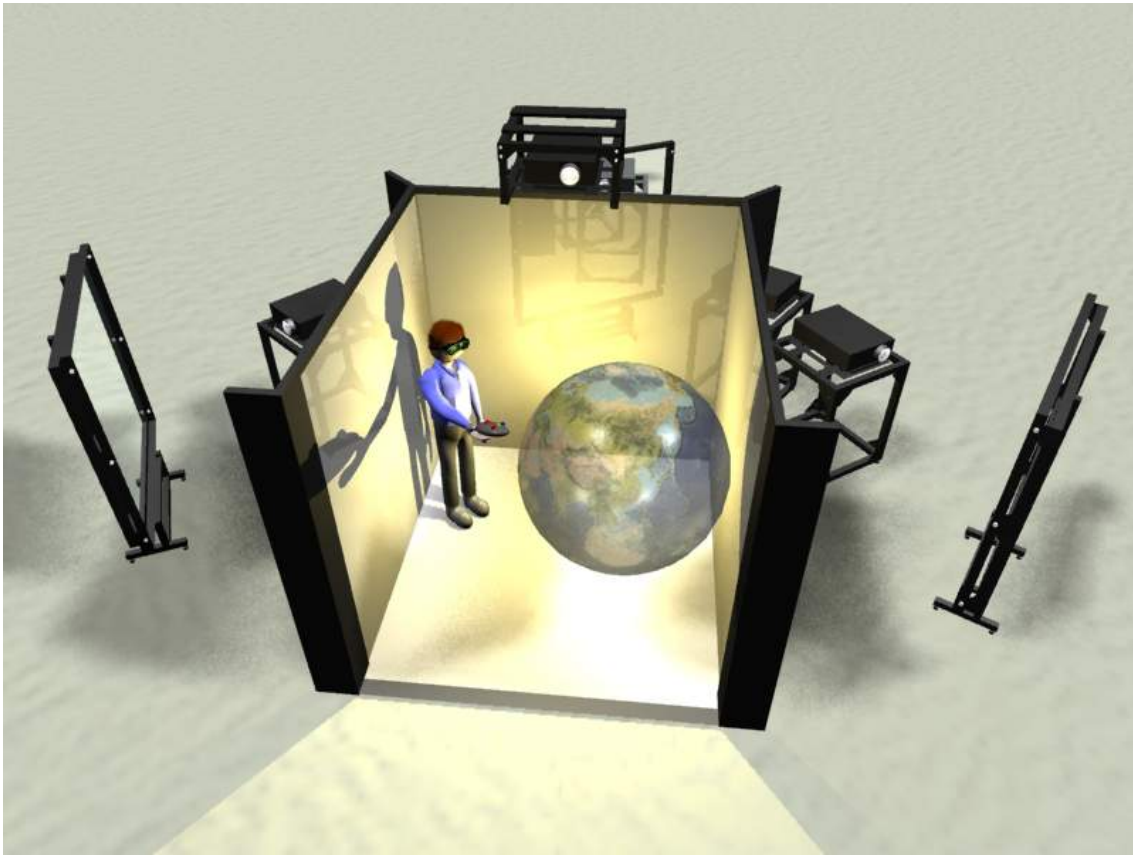


Figure 2.1 A four-sided Virtual Reality system.

Related Work

Several toolkits were assessed for adding a GUI into the scene graph of a clustered application. There are some commercial and noncommercial toolkits that efficiently synchronize data in a clustered system or can render a GUI as a 3D object in the cluster, but no toolkit was found that can do both. We assessed which toolkit would be the best suited for adding a GUI into the scene graph of a clustered application. The toolkits assessed ranged from VR toolkits, game engines, and scene graph libraries.

GUIs

The GUI toolkits evaluated were WIN32(Windows), Cocoa(OS X), X(Primarily Linux/Unix), and QT(cross-platform). Synchronizing a GUI over a clustered VR system is problematic be-

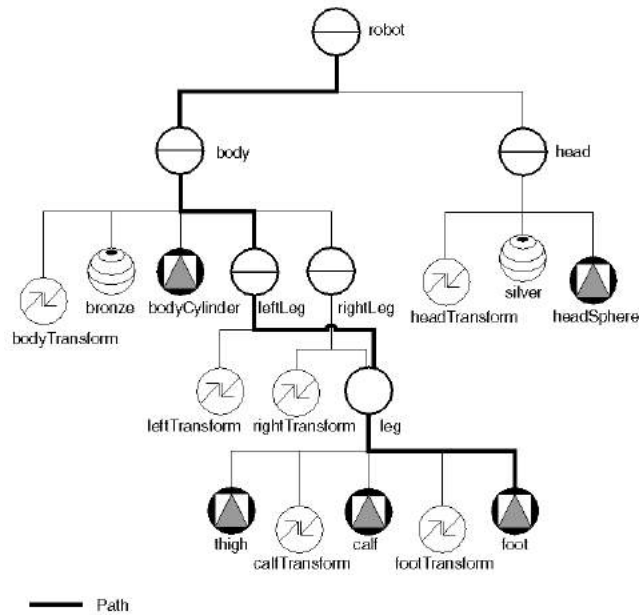


Figure 2.2 Graph structure of a scene graph used to represent a 3D scene.

cause these GUI toolkits were not designed to do this. These GUI toolkits are designed to receive mouse and keyboard input from the OS they are running on and are not designed to share the response to input and drawing responsibilities between separate computers. A clustered VR system that has nodes of different platforms means that the GUI library needs to be cross-platform as well, neither WIN32 nor Cocoa can do this. The QT and X toolkits are cross-platform but can not be directly rendered in a 3D clustered system.

VR Toolkits

Vizard and VRJuggler are toolkits for developing a 3D VR application that can be used in a clustered architecture. In both cases, the GUI cannot be attached to the scene graph or drawn in 3D space, it is only drawn on the foreground. Vizard is closed-source and does not give flexibility. Additional GUI elements cannot be added using Vizard plugins or by any other means since it is closed source, in addition the GUI is not rendered in the scene graph so can't be rendered in 3D. VRJuggler does not have a GUI library, although it does handle input from a wide variety of devices.

Game Engines

Game engines provide much of the functionality needed for developing a GUI. We reviewed Delta3D, Panda3D, Torque3D, Unity, Unreal, and Crytech. Delta3D is the only engine that has been used in a clustered VR environment without synchronization problems, the others do not have support for clustering in a VR system. None of these engines can be used to draw the GUI as a 3D object in the scene using the scene graph, but instead with the window and is drawn as a foreground.

Scene graphs

OpenSceneGraph and OpenSG were reviewed for use in a clustered VR environment. OpenSceneGraph does not directly support clustering at the scene graph level, but applications can use OpenSceneGraph and VRJuggler together for clustered applications. For OpenSceneGraph, clustering is usually handled by starting the same application on all the nodes of the cluster and synchronizing the input and output of all of the nodes. This can provide good results for simple applications but often has many synchronization issues as the application gets complex. OpenSG, however, directly supports multithreaded and cluster safe observation and modification of data for a VR application[[Reiners 2002](#)]. Because of this OpenSG can run the VR simulation on a single node and synchronize the data with the other nodes in the cluster. This is a more efficient use of the cluster resources. Both OpenSceneGraph and OpenSG can also be combined with VRJuggler to handle VR input devices. Neither OpenSceneGraph nor OpenSG have a GUI library.

Implementation

Because OpenSG provides direct support for clustering VR applications, we selected it as a base toolkit to develop our user interface toolkit. Use of OpenSG as a backend allows for clustering without additional code. OpenSG has the additional important features that class member data are reflexive, can be used with VRJuggler for building applications for VR systems, and is easily modified and extended because it is open-source.

The OpenSGToolbox user interface library is approximately 240,000 lines of c++ source code (excluding comments and whitespace). The OpenSGToolbox user interface toolkit is made up of Component, ComponentContainer, Layout, Border, Layer, Font, LookAndFeel, and DrawingSurfaces. Component is the base class for all GUI elements, often called widgets in other GUI toolkits. Figure 2.3 shows the basic Window, Icon, Mouse, Pointing device (WIMP) GUI elements implemented as Components, Figure 2.4 shows the complex WIMP GUI elements implemented as Components. ComponentContainers are Components that contain other Components, our user interface library supports the common GUI element containers: Tab Panel, Split Panel, Scroll Panel, and Windows.



Figure 2.3 Example GUI components supported in the OpenSGToolbox user interface toolkit

ComponentContainers contain Components and use Layouts to define the position and size of Components they contain. The Layout base class is an abstract class that defines

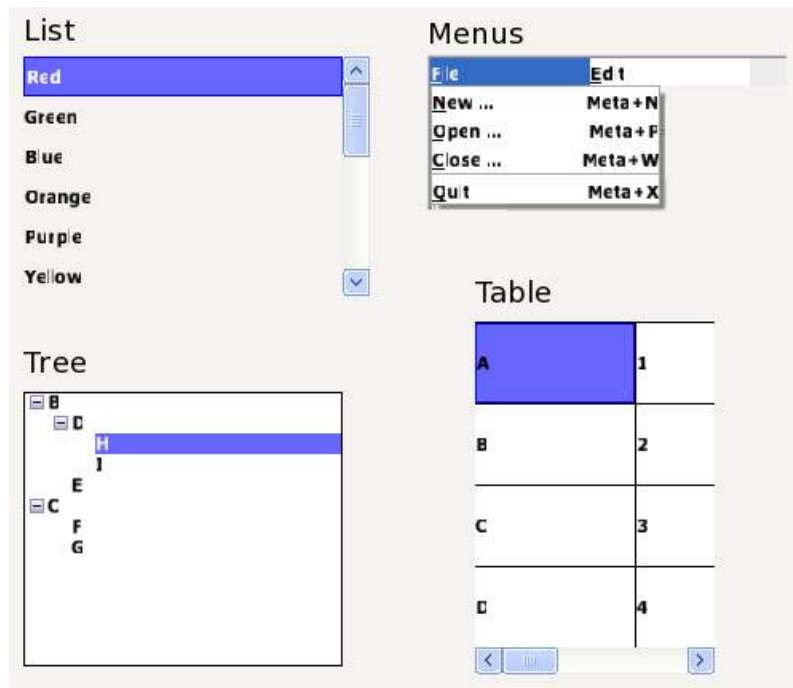


Figure 2.4 Complex GUI components supported in the OpenSGToolbox user interface toolkit

polymorphic methods for setting the position and size of Components that are contained within some ComponentContainer. Every ComponentContainer must have a specific Layout attached. The following specific Layouts are defined by our user interface library.

- **AbsoluteLayout:** set the position and size of Components with user-defined values.
- **FlowLayout** and **BoxLayout:** position the Components vertically or horizontally within the ComponentContainer.
- **BorderLayout:** position and size Components into north, south, east, west, and center areas of a ComponentContainer.
- **SpringLayout:** set the position and size of Components by using user defined directional relationships, or constraints, between the edges(east, west, north, south, vertical center, horizontal center) of components.
- **GridLayout:** sets the position and size of Components by a grid with user defined number

of horizontal and vertical subdivisions.

In addition to the implementation of common GUI library components, the OpenSGToolbox user interface library contributes the features of attaching a GUI as a node in the scene graph, reflexive definition of produced events, support for clustered applications with a single GUI, and simulation of WIMP-like input with 6-DOF trackers and gamepads.

OpenSGToolbox with scene graph

The user interface library can define a GUI and use it flat on the foreground of a window or put into the 3D projection within the scene graph. The same interface can be used on a desktop or in a VR system. Connecting the GUI to the scene graph allows the GUI to be arranged like any other object in the scene graph and displayed across multiple output displays in a clustered VR system fulfilling one of our design requirements. As an example, the same window is rendered in the foreground in Figure 2.5(a) and as a 3D object in the scene graph in Figure 2.5(b). In the bottom of the figure, the camera is moved to an oblique angle with respect to the surface of the GUI window so that the effect of the perspective transformation can clearly be seen on the GUI window. The mouse location is projected into the scene when the GUI is attached to the scene graph.

Event-driven programming

The OpenSGToolbox user interface toolkit was developed to provide an event-driven programming model to developers. This was implemented using object oriented programming with event producers, events, event details, and event handlers[Meyer 2004]. Event producers are concrete instances of objects that are defined to produce specific events. Event details encapsulate the specific details attributed to the invocation of an event. Event handlers are objects that “listen” for events produced by a specific instance of an event producer. When an event is produced by the event producer all attached event handlers are invoked and sent the event details of the event. This allows applications developers to write closures of code that handle how the application should respond to an event.

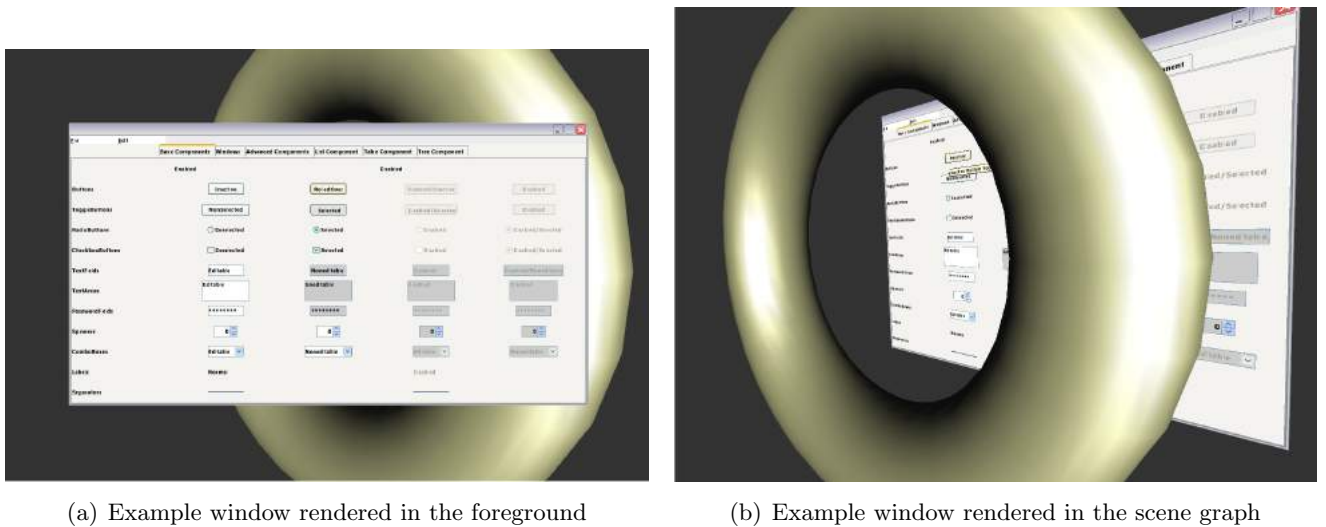


Figure 2.5 Example 3D Graphical User Interface.

Reflexive event production definition

Reflexivity is a property in which a computer application can observe and modify its structure at run-time. Reflection can be broken down further in the parts of the system that can be observed or modified [Smith 1982]. While FieldContainers in OpenSG are data-reflexive, there is no definition for method-reflexivity. This is a problem for the addition of a data-driven architecture for events with OpenSG, because events require a reflexive interface so that generic algorithms can be used for connecting to specific events. We added an interface to FieldContainers in OpenSG for Method-reflexivity. The boilerplate code for each FieldContainer that produced events is automatically generated by extending OpenSG's code generating tools. As an example, user code could connect an event handler to a key pressed event by searching all instantiated FieldContainers for one that produces an event called "KeyPressed". The interface for produced events consists of the following methods:

- Query the number of events produced by a FieldContainer.
- Query the type of data produced by a produced event of a given index
- Attach a method to a produced event that is invoked when the event is produced by an

instance of that FieldContainer

Reflection is not a feature of the C, or C++ languages which are used to program OpenSG and the OpenSGToolbox user interface toolkit. OpenSG was designed to handle multithreaded data in an easy way and that extends to clustering quite naturally. OpenSG defines containers which are protected against simultaneous access from more than one thread. Nearly every OpenSG specific class related to data storage is derived from the FieldContainer. A FieldContainer holds a collection of fields. A field is the smallest construct of data attached to a FieldContainer, and consists of the type of data held by the field, the data of the field, name of the field, and other properties. FieldContainers can be considered as having data-reflexivity, because the fields of a FieldContainer can be observed and modified at run-time by using an abstract interface defined by the FieldContainer base class. This allows the definition of methods that act on FieldContainers generically instead of creating specific code for every concrete type, and future type, of FieldContainer.

Look-and-feel

The OpenSGToolbox user interface library can change the look-and-feel of a GUI. This is done using the prototype pattern[[Gamma et al. 1995](#)]. A Look-and-feel manager class is used to store and define the prototypes used for all of the components of a particular look-and-feel. Only a single look-and-feel can be active at once. As an example, when a Button needs to be instantiated, the new instance is copied from the data of the prototype button for that look-and-feel. This allows users to change the look-and-feel of their interface by simply changing the prototypes used by the look-and-feel manager.

Simulation of WIMP-like input in 6-DOF tracker based systems.

Simulating WIMP-like input for a GUI in a VR system is useful because the WIMP model is familiar, it can be used across many platforms, decreases development time, and has been extensively studied. Many users are familiar with the WIMP interface model. It allows the creation of interfaces that can be used either in desktop or VR systems. Development time

can be decrease dramatically because the GUI can be created and tested on a desktop system using familiar GUI components and then used in a VR system.

The WIMP model requires input for directing the location of interacting with objects in the GUI. This is most commonly accomplished using a mouse-cursor model, or a touch based model. Mouse and touch-based input methods are not commonly used in VR systems because they are awkward to use. The OpenSG user interface library implements two primary methods for directing the location of interaction with objects in the GUI in VR systems depending on the input available: 6-DOF trackers and gamepads.

For VR systems with 6-DOF trackers and devices with analog buttons, mouse-like functionality can be simulated. The location and orientation of a tracker that a user has in their hand is used to cast a ray into the 3D simulation and is intersected with the quadrilateral that the 3D GUI is drawn on. The point of intersection is used as the position of the cursor. The buttons of the device with analog buttons are mapped to specific mouse buttons. A 6-DOF tracker is necessary for constructing the ray for the intersection test, because a ray is defined with a position and direction.

For VR systems with gamepad input, mouse-like input may be unnecessary. The buttons can control which GUI component has focus. Only a single GUI component in a window can have focus, and the focused component can be manipulated further with the gamepad. As an example, a button that has focus can be triggered by moving the focus to the button in the GUI, and then pressing a button on the gamepad to trigger it. This approach is similar for other GUI components. This model is used by many console-based video games, because the gamepad is often the only input device used.

Cross-platform

OpenSG already provides cross-platform support for threading, network communications, and data endianness. The Window class of OpenSG needed to be extended to ensure that The OpenSGToolbox user interface toolkit maintained cross-platform support for Windows, OS X, and Linux platforms. The bridge design pattern was used for this[[Gamma et al. 1995](#)]. An

abstract `WindowEventProducer` inherits from OpenSG's `Window` class and defines an abstract interface used by the user interface library for managing the operating system specific window and events. Concrete classes were implemented for WIN32, Carbon, and X11 that implemented the abstract `WindowEventProducer` interface for those respective platforms. The C++ boost libraries were used for cross-platform support for filesystem paths and date-time objects.

Limitations

The described configuration for simulating the desktop mouse allows the same user interface to be used across desktop and VR platforms with some limitations. Mouse simulation with a 6-DOF tracker requires the user to continuously(actively) keep the cursor in its location(cannot release the tracker, unless it is attached in some way). However, in desktop systems a mouse controlled cursor maintains its position when the user releases the mouse. Mouse directed input has greater precision than many 6-DOF trackers, this can make using the interface more difficult on VR platforms. Precision of the simulated mouse is further exacerbated on VR platforms as the distance of the 3D GUI from the viewer increases . This can be mitigated by constraining the 3D location of the GUI to follow the viewer.

Sample Applications

The `OpenSGToolbox` user interface library has been used by several separate projects. Figure 2.6(a) is an application used for authoring a VR application; it is using `Button`, `SplitPanel`, `TextField`, `Checkbox`, `Spinner`, `Menubar`, `Label`, `TabPanel`, `ProgressBar`, `Tree`, and `OpenGLCanvas` components provided by the library. Figure 2.6(b) from the `Meta!Blast` educational game that can be used in a VR system or desktop. The `Meta!Blast` game contains a GUI for answering questions inside of an interactive plant cell[Call et al. 2007][Wurtele et al. 2010]. These question GUIs are rendered in the 3D scene so that they can be used in a VR system and because the GUI is localized in 3D space to the area of the cell that the question is relevant to.

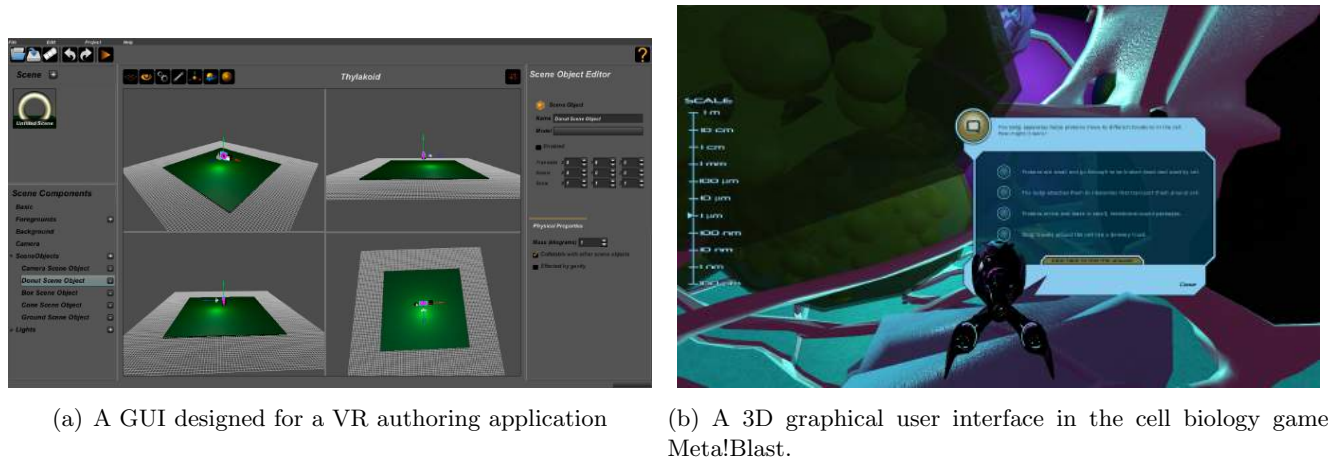


Figure 2.6 Examples of the OpenSGToolbox user interface toolkit used in other applications.

Conclusion And Future Work

This user interface library is part of the OpenSGToolbox collection of libraries (<http://www.opensgtoolbox.org>), and is released under the [Lesser General Public License version 3](#). We have contributed a toolkit for building clustered VR applications with a GUI. This includes all of the basic functionality of modern GUIs, connection of the GUI to a scene graph, reflexive event production, and simulation of WIMP-like input with 6-DOF tracker or gamepad input devices. Future work with the OpenSG user interface toolkit could include rendering GUI components with 3D filling geometry or as a textured surface over non-planar geometry instead of rendering the GUI over a flat planes, support for additional direct manipulation metaphors in VR environments, through body-based gestures, and support for easier input of textual information such as through tablets or other mobile devices.

Acknowledgments

Work on this article was sponsored in part by SEPA - NCR - NIH, the National Science Foundation, and the GDCB PSI College of LAS at Iowa State University. The authors would like to thank Achyuthan Vasanth, Jonathan Flory, Lee Zaniewski, Alden Peterson, Gerrit Voss, David Naylor, and Aaron Cronk for their contributions of source code.

References

- CALL, A., HERRNSTADT, S., WURTELE, E. S., DICKERSON, J., AND BASSHAM, D. 2007. Meta!blast virtual cell: A pedagogical convergence between game design and science education. *Journal of Systematics, Cybernetics, and Informatics* 5, 27–31.
- CRUZ-NEIRA, C. 1995. Virtual reality based on multiple projection screens: The cave and its applications to computational science and engineering. Ph.D. thesis, University of Illinois at Chicago.
- CRUZ-NEIRA, C., SANDIN, D. J., AND DEFANTI, T. A. 1993. Surround-screen projections-based virtual reality: The design and implementation of the cave. In *ACM SIGGRAPH 93*.
- GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- MEYER, B. 2004. The power of abstraction, reuse and simplicity: an object-oriented library for event-driven design. In *Essays in Memory of Ole-Johan Dahl*. Vol. 2635. 236–271.
- REINERS, D. 2002. Opensg: A scene graph system for flexible and efficient realtime rendering for virtual and augmented reality applications. Ph.D. thesis, Technischen Universität Darmstadt.
- SMITH, B. C. 1982. Procedural reflection in programming languages. Ph.D. thesis, Massachusetts Institute of Technology.
- WURTELE, E., BASHAM, D., DICKERSON, J., KABALA, D., SCHNELLER, W., VASANTH, A., AND STENESON, M. 2010. Meta!blast: A serious game to explore the complexities of structural and metabolic cell biology. In *ASME 2010 World Conference on Innovative Virtual Reality*.

CHAPTER 3. KABALA ENGINE: A VIRTUAL ENVIRONMENTS AUTHORING FRAMEWORK FOR NOVICE USERS

A paper submitted to *ISVC 2011*

David J. Kabala^{1 2}, Julie Dickerson³

Abstract

Developing applications for Virtual Reality systems is difficult because of the hardware required, complexity of Virtual Reality software, and the technical expertise required to use them. The KabalaEngine is a application development framework designed to support the development of Virtual Reality applications. The KabalaEngine is an open source application that allows users to build virtual environments with multimedia content and interactive components into a user-created application. Once the relationships have been designed, the engine can then run the user-created application in a virtual reality system equipped with a VR Juggler configuration.

Introduction

It is a goal of virtual reality research to make authoring virtual environments easier. Ideally, anyone that has an idea for a virtual environment should be able to create that environment easily. There are many barriers to this: the hardware is expensive and complicated, the software for running the hardware is also expensive and/or complicated, multimedia content is expensive

¹Human Computer Interaction Program, Iowa State University, Ames, Iowa

²Author for correspondence

³Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa

or nonexistent, and there is not a large community of novice virtual reality developers. We have developed the KabalaEngine and supporting libraries to ease the complexities of authoring for virtual reality experiences. The KabalaEngine was built using OpenSG for clustered support. Because of this, several libraries were contributed to OpenSG that add features and still provide clustered support. The libraries added include support for animations, particle systems, 3D graphical user interfaces, sounds, videos, scripting, and physics. The KabalaEngine uses these libraries to construct projects from a collection of scenes. The KabalaEngine can play the projects in a virtual reality system that has a VR Juggler configuration.

Related Work

The existing tools available to users for developing Virtual Reality applications provide many useful features but have some limitations. There are frameworks such as Vizard, CAVE [Cruz-Neira 1995], VR Juggler [Bierbaum et al. 2001], OpenSG [Reiners 2002], Virtools, Alice, and inVRs [Kepler 2009] that provide Application Program Interfaces(APIs) for developing applications. All of these require programming experience to use. Other features that can be important in a general purpose framework for VR application development are animation, character models, cluster support, modern graphics support, particle systems, physics(rigid and soft body), scripting languages for faster development, sound, user interfaces, and video. There are many libraries available that provide some of these features. Vizard supports many of these features but does not provide an interface for non-programmers.

3DVIA Virtools is a large commercial product for authoring VR applications. Virtools has support for modern 3D graphics, physics, animations, scripting (via a proprietary, in-house language), particle systems, clustering support, and many others. Virtools is closed-source, potentially very costly, and targeted more to expert users. The price for deploying Virtools increases with the number of GPUs present in the VR systems it is deployed on.

Alice is a teaching tool designed for introducing programming to students using a 3D drag-and-drop interface[?]. Alice was originally developed as a tool for fast prototyping for VR

systems[Conway et al. 2000] but has changed for use as a programming teaching tool. Alice is not clusterable, has moved to a desktop-only application. Alice is open-source but does not take contributions. No physics, morphs, skeletal morphs, particles, clustering, or video playback on 3D objects in the scene are supported. The software does allow users to create videos from the animation.

Modules

The KabalaEngine was written to support many of the expected features in real-time 3D applications. These include modern 3D graphics, animation, physics, particle systems, 3D model file importing, video playback, sound playback, scripting, graphical user interfaces, and scene management. Some of these features are supported by OpenSG and other libraries. The KabalaEngine and supporting libraries were developed to work with OpenSG. The following sections details our contributions of some of these features to OpenSG as modular libraries, and their integration into the KabalaEngine.

Animation

The data reflexive features of OpenSG were used to implement a generic animation library. The animation library is made up of Animations, Animators, and KeyframeSequences. Animations are can be applied to data that is supported by an attached Animator. Because of this, Animators and KeyframeSequences are independent of the data that they may be applied to. As an example, if a specific instance of a keyframe sequence is made for 32-bit float data; this keyframe sequence could then be attached to any OpenSG field that is a 32-bit float. So, for all of the data types supported by keyframe sequences, any OpenSG field of those types can have an animation attached to them. Figure 3.1 shows the UML class hierarchy of the animations library.

Keyframed animations can be applied using different interpolation methods. Step, linear, and cubic interpolation is implemented for all math types, and step is implemented for all other types. Keyframed animations can be applied using different methods for how to replace

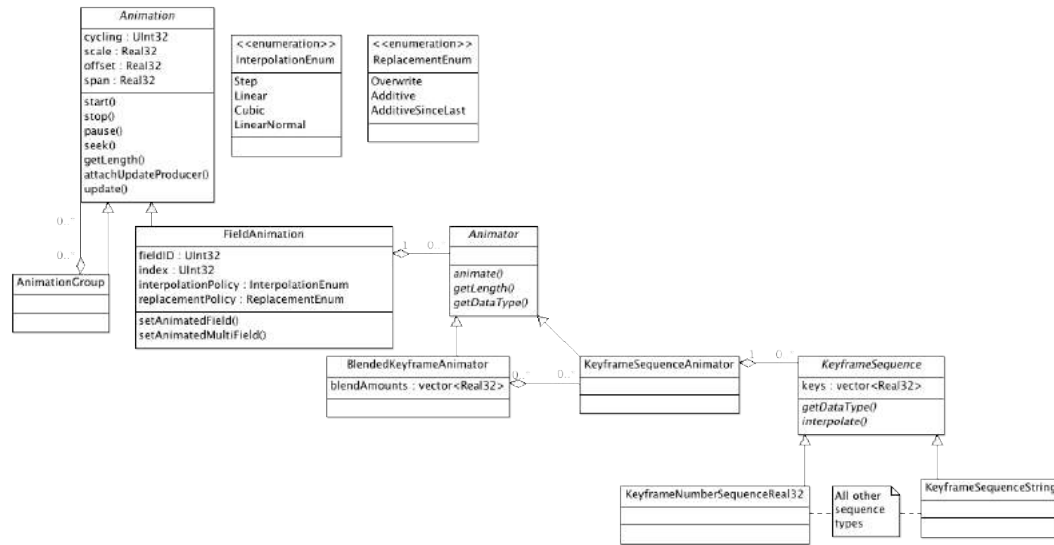


Figure 3.1 Class hierarchy of Animations

the data they are animating; these are overwrite, additive, and additive-since-last. Overwrite will replace the value of the data being animated. Additive adds the value interpolated by the keyframe sequence to the data being animated (only defined for types with an addition operator). Additive-since-last adds the difference of the value interpolated by the keyframe sequence last frame and the current frame to the data being animated (only defined for types with an addition operator).

Multiple animations that act on the same field can be blended using the `BlendedKeyframeAnimator`. The `BlendedKeyframeAnimator` is a realization of the `Animator` interface that contains a collection of `KeyframeAnimators` and floating point weights for each. Only `Animators` that work on types that have addition and multiplication operators defined on them can be used by the `BlendedKeyframeAnimator`. `BlendedKeyframeAnimator` calculates the weighted sum of the interpolated value for animator it contains. The weighted sum is used as the value used by the replacement policy of the animator.

Morphs and Deformable Geometries

Deformable geometries are dynamic 3D geometries [?]. They are created by weighting the vertices of a mesh geometry to transformation nodes. The transformation nodes attached to a deformable geometry are called joints, and the total collection of joints called a skeleton. When constructing the deformable geometry the geometry is bound to the skeleton in the skeleton's bind pose. The bind pose represents the transformations of joints in the skeleton that would not apply any transformational change to the geometry, i.e. identity transformations. The geometry is bound to the skeleton by defining weights to each of the joints in the skeleton, for most joints in the skeleton the weight on a particular joint will probably be zero. For joints that are close to the vertex, then the weight value may be greater than 0. If a joint has a 0 weight affect on a vertex, then the joint weight is not attached to save memory. When joints in the skeleton move then the vertices, and normals of the geometry need to be updated. We developed support for deformable characters for the OpenSG libraries.

Geometry morphs are dynamic 3D geometries. Morphs are constructed from a base geometry, and any number of target geometries. There are weights that can be assigned for each of the target geometries. The geometry rendered for a morph is the sum of the base geometry and the weighed difference of the target geometries with the base geometry. Animating the weights associated with each target geometry can smoothly bring out the feature differences of the target geometry compared to the base geometry. Figure 3.2 is the UML class diagram for morph and deformable geometries. Figure 3.2 is the UML class diagram for morph and deformable geometries.

Advanced Particle System

Particle systems can be used to simulate and render complex particle phenomena. Particle systems consist of a group of particles that each have the properties listed in Table 3.1. Particle systems also contain generators for the creation of new particles and affecters for modifying the properties of particles. A particle system is managed independently from how it is drawn. Figure 3.3 shows the UML class diagram of the particle system library.

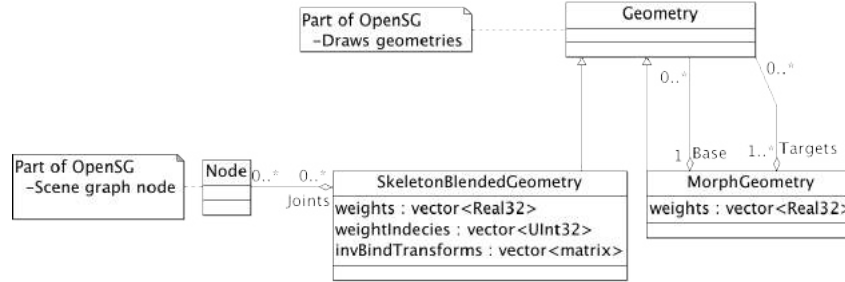


Figure 3.2 Class Hierarchy of Deformable Geometries and Morphs

Table 3.1 Particle properties

Property	Type
Position	3D point
Secondary Position	3D point
Normal	3D vector
Color	4D Color (RGBA)
Size	3D vector
Age	float
Lifespan	float
Velocity	3D vector
Secondary Velocity	3D vector
Acceleration	3D vector
Attributes	String to unsigned integer map

Particle generators generate new particles. The properties of new particles from are generated from 1,2,or 3D distributions. Such as lines, boxes, spheres, cylinders, Gaussian, convex volumes, 3D-Surfaces, etc. Each particle property can be generated from a different distribution. So the position can be generated from a box, velocity from the surface of a sphere, and color from a box localized in RGB colorspace. Particle generators can be attached to a beacon node in the scene graph. The local-to-world transformation of the beacon node is used to transform the generated position, normal, velocity, and acceleration properties of the generated particle. The rate of particle generation is also handled by particle generators. The `RateParticleGenerator` generates a given number of particles per second. The `BurstParticleGenerator` generates a given number of particles the instant it is attached to a particle system. Particle generators are attached to a particle system and are updated every frame to see if

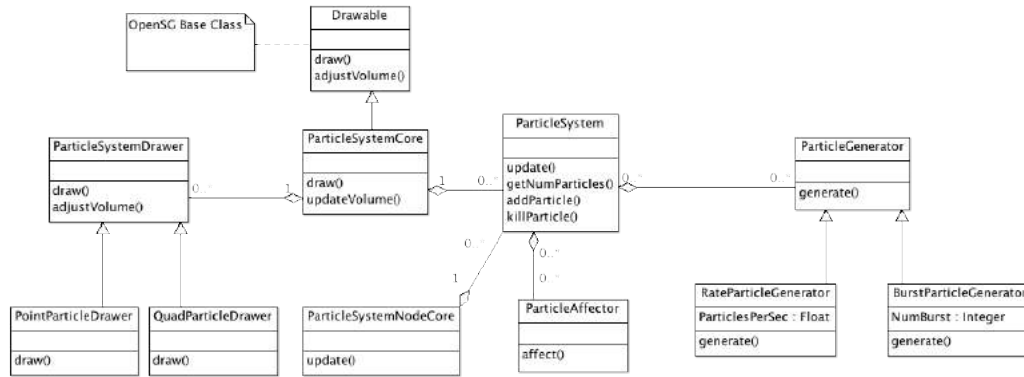


Figure 3.3 Class hierarchy of Particle System

new particles are generated. When a particle generator is finished generating particles it is detached from the particle system.

Distributions are the class of objects used by particle generators to produce a 1,2, or 3 dimensional vector of floats from a specific distribution. Every time a distribution is called to generate a value, usually by a particle generator, a new random value within its distribution is generated using pseudo-random techniques. As an example, a box distribution will generate a point within a box, it may produce a different point the next time it is called to generate a point, but the distribution of points generated will have a uniform distribution within the box. This can be used by ParticleGenerators to use a Gaussian distribution for a particles position, a Spherical distribution for a particles velocity, a Box distribution for a particles color, etc. The dimensions of the distribution attached to a specific property of a particle system must match the dimensions of the property, so only 3D distributions can be attached to the position property of a ParticleGenerator, and only a 1D distribution can be attached to the lifespan property of a ParticleGenerator. Table 3.1 indicates the dimension of each property's type.

ParticleAffectors. A class of objects that affect the properties of particles in a PS. ParticleAffectors affect a particle independently of all other particles. ParticleSystemAffectors are a class of objects that affect the properties of particles of a PS as a whole. ParticleSystemAffectors can affect particles with respect to all other particles in a particle system. An example of this is the GravityParticleSystemAffector that models a PS as a system of point

masses, and calculates the gravitational force applied on each particle by every other particle. ParticleEffectors are attached to a particle system and are updated every frame. When a ParticleEffectors is finished it is detached from the particle system.

Particle drawers are a new OpenSG node type that are attached to a particle system to draw it. There are particle drawers for drawing particle systems as points, lines, quadrilaterals, disks, or as cloned nodes of the scene graph. ParticleDrawers are a class of objects that can draw a particle system in a specific way. Particle systems can be used to simulate all kinds of particle phenomena. These include an explosion, fire, smoke, Meteorological effects like rain, snow, and hail, a school of fish, and many others.

Importing Digital Assets

COLLADA is a royalty-free XML schema for digital asset exchange. COLLADA is used by many real-time 3D developers to exchange assets from digital content creation tools into real-time 3D engines. OpenSG already has support for importing geometry and material content from COLLADA files. We extended the OpenSG COLLADA importer to support the importing of animation elements, shader elements for Cg effects, and controller elements morphs and skins.

Scripting Language

To facilitate the data-driven design of the KabalaEngine a scripting language was integrated. Scripting languages considered were Lua, Python, ECMAScript (Javascript), ActionScript, PHP, Ruby, Perl, Java, Tcl, and Go. Lua was chosen as the scripting language. This is because Lua is a small and relatively easy language that is still Turing complete, creating bindings is easier than most other languages, Lua was designed from the ground up to be embedded, and it has fast and small interpreters and just-in-time compilers. Lua bindings were generated using the Simplified Wrapper and Interface Generator(SWIG), and a SWIG interface definition file. Other scripting languages, such as Python, Ruby, Java, and more, could be integrated by using the SWIG interface definition file and SWIG to generate the bindings

for that language.

The data reflexive features of OpenSG was utilized to simplify the Lua bindings. A generic method for getting and setting the value of a named field of a FieldContainer was written. Because Lua is a dynamically typed language the getting and setting methods could return, and receive parameters of different types without defining a method for each type as would be required by a statically typed language. This allows the get method to return an number when the queried field is a number, and a string when the queried field is a string. This also greatly simplifies the process of using OpenSG libraries that were not constructed with Lua bindings. This is because the getting and setting methods are generic for any FieldContainer instance and so would work for any new FieldContainer type.

Sound

The sound library integrates sound playback with the scene graph. An abstract interface was written to control sound playback with an abstract SoundManager and Sound. The SoundManager handles creating concrete sounds, making the real-time updates to the sound subsystem, tracking the listener position, and initializing/deinitializing. The new Sound FieldContainer is an abstract interface for sounds that can be used to observe and modify sounds. SoundEmitters are a new scene graph node that can have a Sound emitted from them. SoundEmitters play a sound at the virtual location in 3D space that the emitter is located in the scene graph. When SoundEmitters are updated each frame, the emitted sounds are updated to take into account the changes in position and orientation of the listener and SoundEmitter node. Moving a sound can be achieved by changing the transformation of the SoundEmitter node in the scene graph. The node used by the scene graph for the camera is used as the listener position. The SoundManager and Sound abstract interfaces were used to add support for the Fmod sound library, and could be used to support other low level sound libraries. Figure 3.4 shows the UML class diagram of the primary components of the sound library.

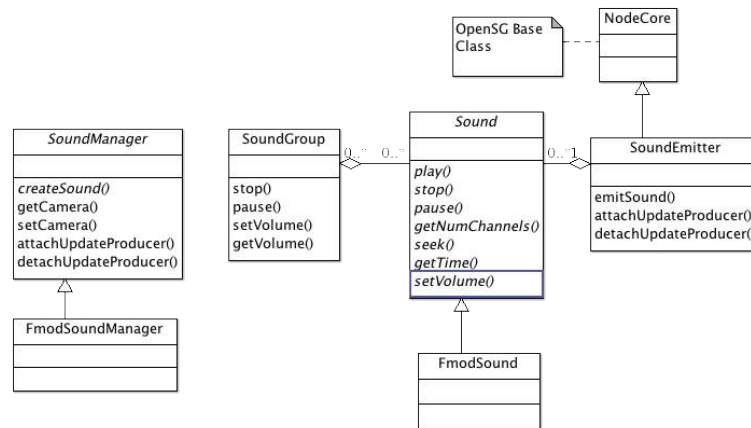


Figure 3.4 Class hierarchy of Sounds

Video

The video library integrates video playback with OpenGL textures. An abstract interface was written to control video playback. VideoTextureObj is a new type of TextureObj in OpenGL that can be used anywhere a texture can be used. In this way a video can be used for data as though it was a texture. The abstract interface was implemented for the VideoLAN (VLC) libraries, which is cross-platform supported, and the DirectShow framework, which only has Microsoft Windows support. Figure 3.5 shows the UML class diagram of the primary components of the video library.

Physics

The physics library OpenDynamicsEngine(ODE) was integrated with OpenGL. ODE supports real-time simulation of rigid body dynamics and collision detection. Behboud Kalantary created a physics library that combined the ODE library with OpenGL 1.8. Nodes in the scene graph can have physics characteristics attached to them. The PhysicsBody attachment contains rigid body definitions of ODE, including mass and center of gravity. PhysicsGeom attachments contain the definitions for collision geometry. PhysicsJoints attachments define how Physics-

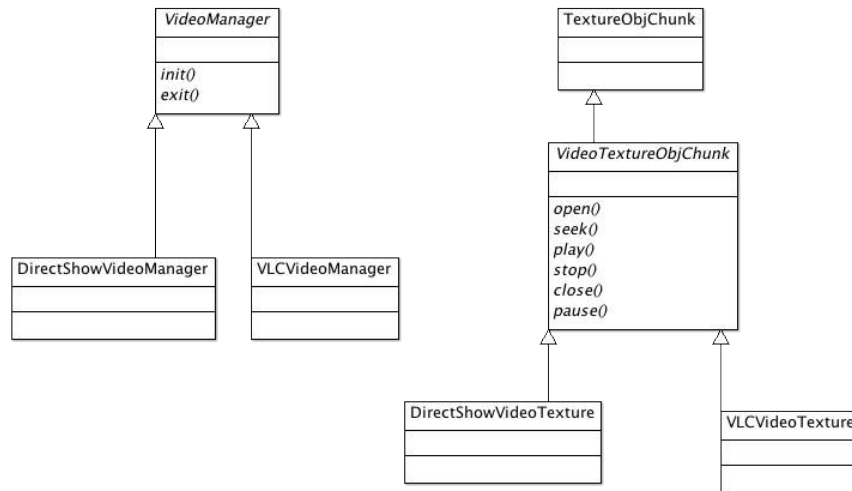


Figure 3.5 Class hierarchy of Video

Bodies can be constrained with other PhysicsBodies or the simulated physics world. The ODE physics simulation is executed by a new scene graph traversal that iterates the physics collision and simulation loop. After updating the collision and physics simulation the traversal updates the transformations of nodes in the scene graph that have a PhysicsBody attached to them. We added support for using ODE with OpenSG 2.0, drawing the physics characteristics, the generation of events from collisions based on the nature of the collision. Figure 3.6 shows the UML class diagram of the primary components of the physics library.

User Interface

The user interface library used by the KabalaEngine is a library build on top of OpenSG. A separate paper describing this library has been submitted to UIST 2011.

Generic FieldContainer GUI

The graphical user interface library for OpenSG was used to make a generic editor for FieldContainers. Most classes in OpenSG are derived from FieldContainer. They contain a collection of fields, where each field has a name, type, description, and concrete data value. FieldContainers are data-reflexive because the definition of the fields present on a FieldCon-

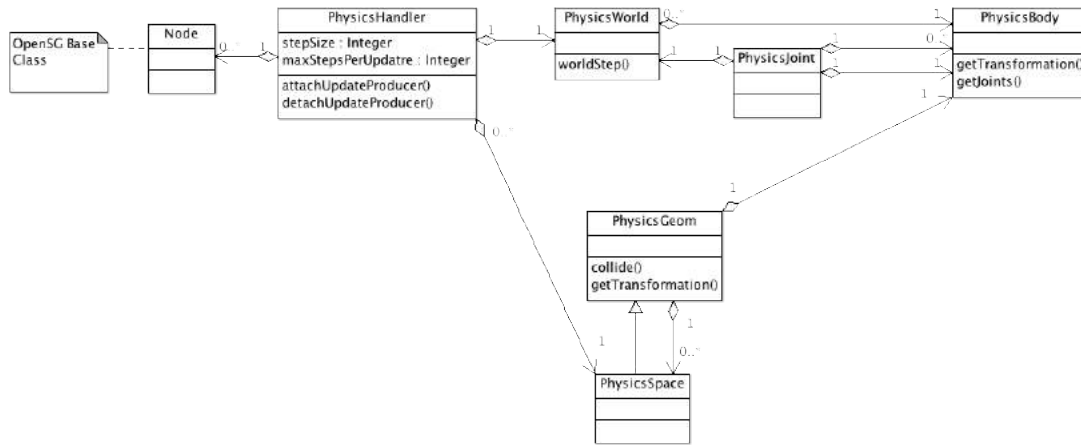


Figure 3.6 Class hierarchy of Physics

tainer can be queried at run-time. The data-reflexive nature of FieldContainers was utilized to implement a generic editor for FieldContainers. Figure 3.7 shows the generic FieldContainer GUI attached to the viewport of a scene.

Each field in a FieldContainer has a specific type, but the values of a field can be observed and modified by converting their values to and from a string. The generic FieldEditor uses only a textfield. The text of the textfield is filled with the text conversion of the field attached to it. Using this generic FieldEditor every field can be edited. However, there are usually better GUI elements that can be used. FieldEditors were made for number types that use a spinner GUI element, multi-dimensional vector types use a spinner for each dimensional element, color types that use a color selection dialog, matrix types that use a spinner for each of the translation, rotation, and scale affine decompositions of the matrix.

FieldEditors are can only be created from the FieldEditorFactory. The FieldEditorFactory takes a pointer to a FieldContainer and the index of the field to create a FieldEditor for. The FieldEditorFactory returns an instance of a concrete FieldEditor that can be inserted into a GUI panel. The FieldEditorFactory contains a map that stores the type of FieldEditor to use

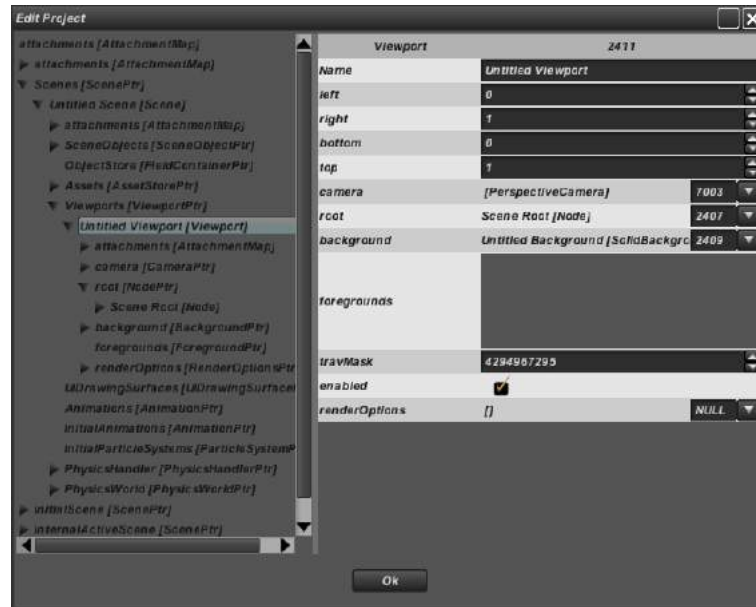


Figure 3.7 Generic FieldContainer Editing Interface

for a field type, there can be more than one type of FieldEditor used for the same field type. For each field type there is a default FieldEditor type associated with it, this can be changed at run-time by user code. For field types that have more than one type of FieldEditor, the FieldEditors are differentiated by names.

Using the FieldEditorFactory a generic FieldContainerEditor was created. The generic FieldContainerEditor takes a pointer to a FieldContainer, it then iterates over all the fields for that type of FieldContainer. For each field the generic FieldContainerEditor uses the FieldEditorFactory to create a FieldEditor for that specific field. The returned FieldEditor is placed in a GUI panel. The FieldContainerEditorFactory behaves the same as FieldEditorFactory except if uses FieldContainer types and FieldContainerEditor types.

The right panel in Figure 3.7 shows an example of a generic FieldContainerEditor attached to a viewport FieldContainer. The names of the fields are listed left, right, bottom, top, camera, root, background, foregrounds, travMask, enabled, and renderOptions. Next to the names of the fields are the specific field editors for that field. For the left, right, bottom, and top fields there are spinner FieldEditors because those fields are numbers. The enabled field uses a checkbox FieldEditor because it has a boolean type. The camera, root, background,

and foregrounds fields use a FieldEditor specific for fields that have a type that points to a FieldContainer.

KabalaEngine Architecture and Interfaces

Project Player

The KabalaEngine constructs applications into projects. The KabalaEngine uses the Player feature to play a project. The play attaches the project to the root window of the window the KabalaEngine is running in. The Player handles starting, stopping, resetting, and pausing a project.

Project, Scenes, SceneObjects, Effects, Behaviors

A project is made up of a collection of scenes. Only one scene can be active at a time in a project; the project manages the transitions between scenes by entering and exiting them. Projects also hold a collection of global assets, listed in Table 3.2, that can be accessed by any scene. Figure 3.8 shows the UML class diagram of projects, scenes, scene objects, effects, and behaviors.

There are several events that a project can produce. Table 3.3 lists the type of events that projects can produce. The Started event of a project is produced when the Player starts the project. The Stopping event is produced when the Player is about to stop a project, and the Stop event when the Player has stopped the project. Reset is produced by a project when the reset method is called on the project, this may be called by a script. SceneChanged is produced by a project when the project has changed the active scene. ScenesChanged is produced when a scene is added, removed, or moved in a project.

Scenes contain a collection of SceneObjects, scene specific assets, viewports, and scene-wide physics properties. Table 3.4 shows the types of events that scenes produce; in addition, scenes also produce all of the mouse, key, and window events by connecting the origin producers of these events. When a project makes a scene active, it first checks that the scene has been started, if not then it invokes the start method of that scene. Then the project invokes the

Table 3.2 Asset types

Scene graph nodes
Foregrounds
Backgrounds
Images
Textures
Sounds
Particle systems
Materials
Animations
Graphical user interfaces
Videos

Table 3.3 Project Events

Started
Stopping
Stopped
Reset
SceneChanged
ScenesChanged

Table 3.4 Scene Events

Entered
Exited
Started
Ended
Reset

enter method to the scene. This allows scenes to initialize assets only once, in the start method. When a project deactivates a scene it invokes the exit method on the scene. Finally, when a project is reset or stopped it calls the ended method of all scenes that have been started. When these respective methods are invoked the corresponding event is produced as listed in Table 3.4.

SceneObjects hold a pointer to a 3D node representing the object, SceneEffects, and Behaviors. The 3D node of a SceneObject is attached to the scene graph used by the scene. Table 3.5 shows the types of scene object effects available. Table 3.6 shows the events that Scene object effects can produce. SceneEffect is an abstract class that has start, stop, pause, and reset methods. The SceneEffect produces the corresponding events when these methods are invoked. In addition, SceneEffects produce the Finished event as defined by the specific type of effect they are emulating. For the SoundSceneEffect the Finished event is produced when the sound played has reached the end. The SequentialSceneEffect and ConcurrentSceneEffect contain a collection of SceneEffects. The SequentialSceneEffect, when started, starts the first SceneEffect in its collection, waits till that effect has produced a Finished event, and then starts the next in the sequence until it reaches the last. When it reaches the last then it produces its own Finished event. The ConcurrentSceneEffect, when started, loops through all of its SceneEffects and starts them. It then waits for all of them to produce their Finished event before it produces its own Finished event.

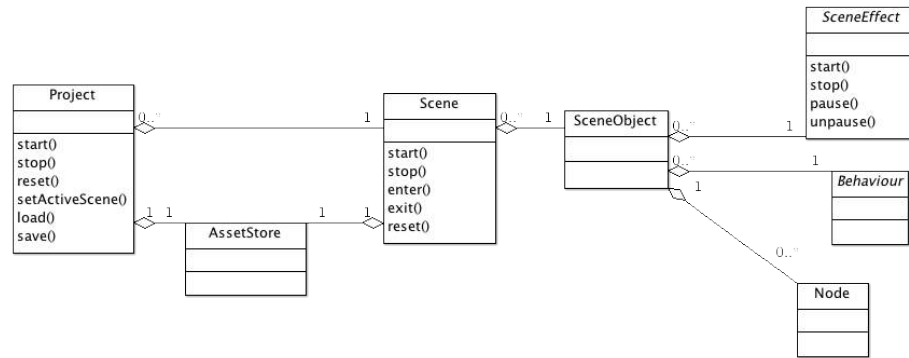


Figure 3.8 Class hierarchy for Project

Class hierarchy of Project, Scene, SceneObject, SceneEffect, and Behaviour

Table 3.5 Specific Types of SceneEffects

Play/stop/pause/unpause sound.
Start/stop/pause/unpause animation.
Physics impulse.
Lua script.
Particle system effect.
Sequential effect.
Concurrent effect.

Table 3.6 Scene Object Effect Events

Started
Stopped
Paused
Unpaused
Finished

Control Loop

The main loop of the KabalaEngine is managed by the root window of the application. The window loops through handling events, updating, and rendering as shown in pseudocode listing 3.1. The handling of events by a window involves producing any input or window specific events. These can include mouse, key, mouse wheel, and window events. The update method simply produces an update event that encapsulates the elapsed time since the last time the update event was produced by the window. Finally, the window invokes the render method attached to it and iterates the loop.

Projects attach event handlers to the mouse, key, mouse wheel, and window events produced by the root window. The project reproduces these events, with the same details, itself.

Algorithm 3.1

Window main loop pseudocode

```
while the application is running do
  while the event queue is not empty do
    handleEvent(getNextEvent())
  end while
  produceUpdateEvent(getElapsedTime())
  render()
end while
```

However, projects have a `BlockInput` flag that, if true, will cause the project not to reproduce any events. Scenes also attach event handlers to the mouse, key, mouse wheel, and window events and reproduce them in the same way as the project. However, scenes only reproduce events from the root window when they are active. This is useful because specific event handlers can be attached locally to the mouse, key, mouse wheel, or window events of a scene or project. Then if the project or scene block input of those events, it does not cause the blocking of event handles attached to the root window or other scenes. Using this, event handlers can be attached to specific scenes, but do not need to be detached when the scene is no longer active because that scene will not reproduce those events unless it is active.

Objects that need to be updated can attach to the window, project, or scene. Examples of objects that require updates are `SoundManager`, dynamic `ParticleSystem`s, active `VideoWrapperTextures`, active `Animations`, `PhysicsHandlers`, `SceneEffects`, and any user-defined `LuaBehavior` that depend on the update event. The update is produced once per loop of the main loop before the scene is rendered.

Run-time debugger

A Run-time debugger interface was created for expert users to make modifications and introspect properties of a running project. This interface was defined for moderate to expert users. When a project is running, a configurable input sequence can bring up the debugger interface, drawn as a foreground, in the root window that the project is running in. Figure 3.9 shows the debugger interface. The debugger interface is divided into three primary regions:

scene graph tree (left), utilities tab panel (bottom), and the content view panel (center). The debugger interface also includes a menubar at the top of the window.

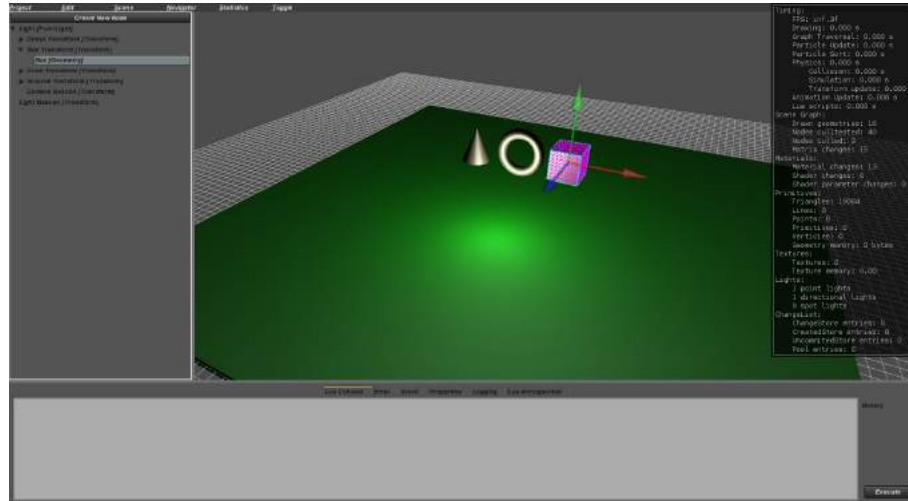


Figure 3.9 KabalaEngine Debug Interface

KabalaEngine Debug Interface for Intermediate-Expert Users

Content View Panel

Direct manipulation of objects in the scene. Users can directly manipulate scene graph nodes in the scene. This is done by first selecting a scene geometry, light, camera, or particle system. A node can be selected by clicking on the screen drawing of its geometry directly or selecting the appropriate node in the GUI tree. Once selected the triangle mesh of the geometry, bounding volume, and local coordinate frame are rendered. If the selected scene graph node can be transformed then the manipulator interface is rendered. There are translation, rotation, and scale manipulators. Only one of these types of manipulators is active at once, the user can switch between the type of manipulator by using the t, r, and s keys. The user can use the mouse to click and pull on the manipulators to apply changes to the transformation of the selected scene graph node.

Scene graph GUI tree

The structure of the scene graph can be changed by the user. This includes creating new nodes, removing nodes, copy and pasting nodes, importing a model file, hiding/showing a node, and apply advanced graph traversals. The scene graph is presented in a tree GUI element in the left panel of the debugger interface shown in Figure 3.9. The user can expand/collapse nodes in the GUI tree to observe the children of nodes in the scene graph. A right-click menu provides actions for hiding/showing a node, copy, cut, paste, and deletion of a node, importing model files into the scene graph, exporting to a file, and several graph traversal methods.

Utility Tab Panel

Lua debugging utilities The bottom portion of the debugger interface shown in Figure 3.9 contains a tab panel with a Lua console, Lua error text field, Lua stack trace, a Lua data introspection tree, log panel, and scene graph properties panel. The Lua console contains a text field where Lua code can be typed and an execute button for executing the Lua code. The Lua error text field provides a detailed description of a Lua error if an error occurs when executing error code. Also, the Lua stack trace panel provides a stack trace of the Lua functions when a Lua error is produced. The Lua data introspection tree is a tree GUI element that is rooted at the global data table of the running Lua context. The introspection tree can be used to observe the names, types, and values of all values that are attached to the global Lua context.

Main Menubar

The main menubar of the debugger interface contains Project, Edit, Scene, Statistics, and Toggle menus. The main menu bar can be seen at the top of Figure 3.9.

Project Menu The following is a list of the menu items in the Project menu actions preformed when selected.

- Open Project. Opens a file selection dialog window to select a project to load.

- Save Project. Saves the current project to a file.
- Save As Project. Opens a file save dialog window to save the current project.
- Reset. Exit the currently active scene, stop all scenes that have been started, reset all scene, and then activates the initial scene.
- Force Quit. Closes the KabalaEngine.

Undoable actions. An important consideration from direct manipulation theory are that user actions can be undone if there is an error. The run-time debugger implements most user actions in a way that can be undone. This is done by extending the use of the strategy design pattern as described by [Gamma et al. 1995]. All user actions are encapsulated into objects of Command or UndoableCommand types. Command defines an abstract interface for executing some action. Commands are executed by a CommandManager. UndoableCommands inherited from Command and add an interface to undo and redo the action performed when the CommandManager executes the UndoableCommand. An UndoManager stores UndoableCommands as they are executed so that they can be undone or redone according to the order they were originally executed.

An example of an undoable command is the manipulation of the transformation of a node in the scene graph. Users can directly translate, rotate, and scale nodes in the scene graph. When this is done an UndoableCommand is created with the details of the manipulation. The UndoableCommand is executed and added to the UndoManager. The user could then undo and subsequently redo the application of this transformation by invoking the undo/redo menu items.

Conclusions

The source code repository for OpenSG can be obtained at http://github.com/djkabala/OpenSGDevMaster_Toolbox. The source code repository for the OpenSGToolbox can be ob-

tained at <http://github.com/djkabala/OpenSGToolbox>. The source code repository for the KabalaEngine can be obtained at <http://github.com/djkabala/KabalaEngine>.

Future Work

The interface uses primarily a WIMPS-base style which is very useful for work on desktops, it doesn't have a specialized interface for using the Builder in a virtual reality system or hand-held devices. The Builder does not have a easy user interface for dynamic relationships such as animations, particle systems, and scripts. The KabalaEngine run-time debugger supports this through a generic "expert" user interface. Currently there is not a large user community for the KabalaEngine. The importing of assets can be cumbersome. To run a project in a virtual reality system the system requires VR Juggler.

Acknowledgements

The authors would like to thank Achyuthan Vasanth, Daniel Guilliams, Robert Goetz, Eric Langkamp, and David Naylor for there contributions of source code. The authors would like to thank William Schneller, Amy Dixon, and Heidi Sinsel for their work designing the look and feel and the KabalaEngine logo. The user interface library is released under the General Public License version 3.

References

- BIERBAUM, A., JUST, C., HARTLING, P., MEINERT, K., BAKER, A., AND CRUZ-NEIRA, C. 2001. Vrjuggler: A virtual platform for virtual reality application development. In *Proceedings IEEE Virtual Reality*.
- CONWAY, M., AUDIA, S., BURNETTE, T., COSGROVE, D., CHRISTIANSEN, K., DELINE, R., DURBIN, J., GOSSWEILER, R., KOGA, S., LONG, C., MALLORY, B., MIALE, S., MONKAITIS, K., PATTEN, J., PIERCE, J., SHOCHET, J., STAACK, D., STEARNS, B., STOAKLEY, R., STURGILL, C., VIEGA, J., WHITE, J., AND WILLIAMS, G. 2000. Alice: Lessons learned from building a 3d system for novices. In *Conference on Human Factors in Computing Systems: Proceedings of the SIGCHI conference on Human factors in comput.*

- CRUZ-NEIRA, C. 1995. Virtual reality based on multiple projection screens: The cave and its applications to computational science and engineering. Ph.D. thesis, University of Illinois at Chicago.
- GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- KEPLER, J. 2009. A collaborative interaction framework for networked virtual environments. Ph.D. thesis, Universität Linz.
- REINERS, D. 2002. Opensg: A scene graph system for flexible and efficient realtime rendering for virtual and augmented reality applications. Ph.D. thesis, Technischen Universität Darmstadt.

CHAPTER 4. CREATING A GRAPHIC USER INTERFACE FOR VIRTUAL REALITY AUTHORING: AN OVERVIEW AND ANALYSIS OF THE KABALAENGINE WORLD BUILDER

A paper to be submitted to *ACM VRST 2010*

David J. Kabala ^{1 2}, Julie Dickerson ³, and Stephen Gilbert ⁴

Abstract

Developing applications for Virtual Reality systems is difficult because of the hardware required, complexity of Virtual Reality software, and the technical expertise required to use them. The KabalaEngine Builder has been designed to support the development of Virtual Reality applications by novice users. The KabalaEngine is an open source application that allows users to build virtual environments with multimedia content and interactive components into a user-created application. Once the relationships have been designed, the engine can then run the user-created application in a virtual reality system equipped with a VR Juggler configuration.

Introduction

It is a goal of virtual reality research to make authoring virtual environments easier. Ideally, anyone that has an idea for a virtual environment should be able to create that environment

¹Human Computer Interaction Program, Iowa State University, Ames, Iowa

²Author for correspondence

³Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa

⁴Department of Psychology, Iowa State University, Ames, Iowa

easily. There are many barriers to this: the hardware is expensive and complicated, the software for running the hardware is also expensive and/or complicated, multimedia content is expensive or nonexistent, and there is no large community of novice virtual reality developers. There is a lack of VR frameworks that contain a comprehensive set of multimedia and computational for novice users of the framework. Designers of the KabalaEngine define novice users as those who are comfortable using graphical user interface(GUI) applications like office productivity software and internet browsers but may not necessarily have experience with computer programming or other technical computer training. We have chosen this group of users because there is currently no means for such a user to develop a Virtual Reality(VR) application. We define VR systems as systems that achieve a high level of immersion through graphics, sound, haptics, other displays, and a simulated environment that affects users with a high level of presence.

We have developed the KabalaEngine and supporting libraries to ease the complexities of authoring for virtual reality experiences. The KabalaEngine was built using OpenSG for clustered support. Because of this, several libraries were contributed to OpenSG that add features and still provide clustered support. The libraries added animation, particle system, 3D graphical user interface, sound, video, and physics support. The KabalaEngine uses these libraries to construct projects from a collection of scenes. The KabalaEngine can play the projects in a virtual reality system that has a VR Juggler configuration.

Using the KabalaEngine as the supporting architecture we have designed and implemented a graphical user interface for novice users called the KabalaEngine Builder. The builder presents an interface that users can use to create virtual environments. The cross-platform and data-driven design of the KabalaEngine, and supporting libraries, makes the builder available on Windows, Linux, and OS X operating systems. The projects can be created on one system and then moved and edited on another system. This is useful because users can create virtual environments on low cost, easily accessible systems and then move the created project onto complicated virtual reality systems when the systems are accessible.

In the following sections we will detail related work, the design and implementation of the

Builder interface, describe our evaluation study, and discuss the findings from the study.

Related Work

3DVIA Virtools is large commercial product for authoring VR applications. Virtools has support for modern 3D graphics, physics, animations, scripting (via a proprietary, in house language), particle systems, clustering support, and many others. Virtools is closed-source, potentially very costly, and targeted more to expert users. The price for deploying virtools increases with the number of GPUs.

Alice is a teaching tool designed for introduction to programming using a 3D and drag-and-drop interface[Conway et al. 2000]. Alice was originally developed as a tool for fast prototyping for VR systems[Pausch et al. 1995]. Alice is not clusterable, has moved to a desktop-only application. Alice is open-source but does not take contributions. Alice does not support physics, morphs, deformable geometry, particle systems, clustering, or video playback as textures in scene. The software does allow users to create videos from animation they create.

Croquet and Cobalt Croquet is an Software Developer’s Kit (SDK) for collaborative work between teams of workers that is implemented in Squeak Smalltalk. Croquet uses OpenGL for rendering and includes support for many GUI elements: buttons, textboxes, windows, etc. Croquet was built for communication, collaboration, resource sharing, and synchronous computation among multiple users. Applications created with the Croquet SDK can be used to support scalable collaborative data visualization, virtual learning and problem solving environments, 3D wikis, online gaming environments (MMORPGs), and privately maintained/interconnected multiuser virtual environments[Smith et al. 2004]. Croquet is no longer under development, but now the open Cobalt project[Lombardi and Lombardi 2010] has taken up Croquet and is in alpha release.

Design Principles

We defined a set of principles to use when designing the builder interface. These design principles were selected and modified from best practices described by [Shneiderman 1987](#)

- *The interface should be familiar* to users. This is an effort to keep novice users comfortable by providing an interface with interaction methods and a display format that users have some experience with.
- *The interface should use direct manipulation* for modifying graphical, 3D properties. The editing of 3D properties of objects is complicated and error prone through command line and menu-stye interfaces. A direct manipulation interaction style would allow users to immediately observe the changes made to objects in 3D space.
- *The interface should assist the user with errors* by allowing actions to be easily reversible. It is important to reduce the user anxiety of doing something wrong by making actions reversible and to have the capability of saving the project to a file.
- *The interface should assist novice users of VR authoring.* We will attempt to reduce short-term memory load by keeping the interface simple and dividing it into logical sections.
- *Use menus for separating similar but distinct components.* For actions where there is potentially a large number of options to choose from, menus will be used to logically categorize them. This is an important part of keeping the interface simple and uncluttered

KabaleEngine Builder Interface

The interface we developed for novice users is a graphical user interface that makes up what we call the KabalaEngine Builder. The KabaleEngine Builder interface was designed to support the following features: create projects, create/remove scenes, add/remove 3D objects in the scene, move objects in the scene, direct manipulation of 3D objects, reversable errors, modify physical attributes of objects in scene, and can save/play a project. In addition the

interface should allow editing properties of cameras, lights, scene objects, scenes, backgrounds, and foregrounds.

Figure 4.1 shows the final design of the GUI for the KabalaEngine Builder. The Interface is divided up into four primary sections, and other utility areas. The four primary sections are the project view panel (Figure 4.1A,) editor view panel (Figure 4.1B,) 3D scene view panel (Figure 4.1C,) and the scene components panel (Figure 4.1D). The interface is designed for editing a project. Only one scene of a project can be edited at any one time. Users must select which scene is to be edited, this is called the selected scene.

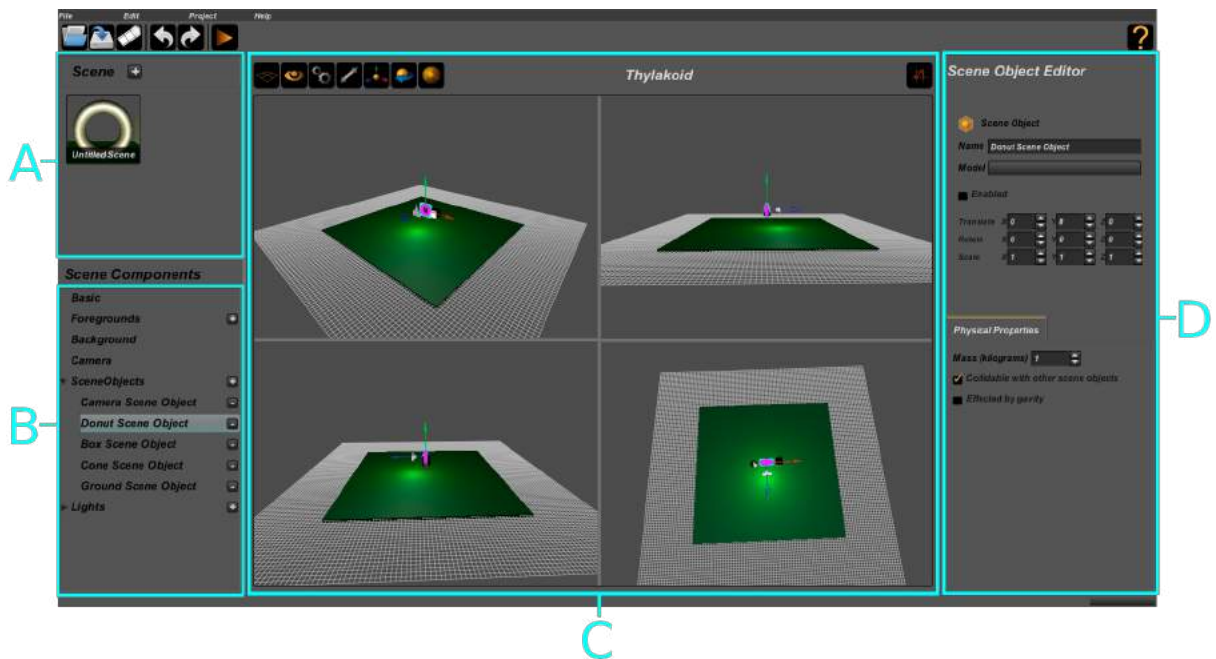


Figure 4.1 KabalaEngine Builder Interface

(A.) Project editor panel. (B.) Scene components panel. (C.) 3D scene view panel. (D.) Editor view panel.

Project view panel (4.1A) The project view panel provides an interface for users to add and remove scenes from the project, and to change the currently selected scene. Scenes are arranged in a table format in the order they were created, the order of the scenes in the table is not significant. Each of the scenes in the table have a display image and label to represent them. The display image for a scene is a rendering of the scene. The display images of scenes

that are not selected are rendered darker than normal to indicate they are not selected. To add a scene users press the “+” button next to the “Scenes” label. A scene can be removed by pressing the “-” button that appears on a scenes display image when the mouse is hovered over the display image for the scene. A scene can be selected for editing by pressing the scene display image.

Scene components panel (4.1B) The editor view panel provides users with an interface to select a logical component of the selected scene to edit and for adding/removing components. Users can edit the basic scene properties of foregrounds, background, scene camera, scene objects, and lights. The components are arranged in a tree interface with a depth of two. Components that a scene can have multiples of, foregrounds, scene objects, and lights, can be expanded in the tree interface to show all of their instances. Users can create new instances of foregrounds, scene objects, or lights by pressing the “+” button next to the component type. Also, the components can be removed from the scene by pressing the “-” button to the left of the corresponding component.








Selecting a scene component will fill the editor view panel with the specific editor interface for that component. When the camera, a scene object, or a light component is selected in the tree, then that component will also indicate it’s selection in the 3D scene view panel. For the scene object components, the mesh of the geometry is rendered in pink and the bounding box in teal.

3D scene view panel (4.1C) The 3D scene view panel provides users with an interface to select and move scene objects, lights, and the camera. The 3D scene view uses visual annotations to give the user information about the editing area and the objects they are editing. None of these visual annotations are rendered when the project is played. The grid visual annotation is located on the XZ-plane; it is used to orient the user with the origin in world space and is used as a spatial reference when moving objects around. The camera for the scene has a visual annotation, 3D model of a camera, that is drawn at the location and orientation of the camera. Lights also have visual annotations similar to the camera, but use

different graphical representations based on the type of the light (point, directional, or spot).

The 3D scene view panel has a toolbar at the top for applying actions to the 3D view. The icons and actions of each of the buttons is listed in Table 4.1. The split toolbar button can split the view into a single, two-by-one horizontal, one-by-two vertical, or a two-by-two view. Figure 4.1 shows the 3D view panel with the two-by-two split configuration, however, by default the application starts with the single, or no split, configuration.

Table 4.1 3D Scene View Panel Toolbar Buttons

Button	Action
	Splits the view between four configurations
	Focus view on selected component
	Edit the view configuration
	Switch to fullscreen mode
	Activate the translate transformation mode
	Activate the rotate transformation mode
	Activate the scale transformation mode

Objects in the scene can be selected using the left-mouse button. The object in the scene that has the closest intersection with a ray cast as a projection of the mouse is selected. Selecting a scene object will cause the selection visual annotations pink mesh, teal bounding box, for that scene object to be rendered. In addition, the corresponding element in the scene components panel will also be selected. Depending on the type of the transformation mode (translate, rotate, or scale) the visual manipulators for that transformation mode will be rendered.

There are three types of transformation modes for scene objects, cameras, and lights; they are translation, rotation, and scale. Figure 4.2 lists the transformation modes and the graphical

manipulators used for editing them. The axis that the manipulator component is constrained to is denoted by its color: red for the x-axis, green for the y-axis, and blue for the z-axis. The Scale transformation node has an additional manipulator that is teal for scaling in all 3 axes uniformly. The manipulators are used to edit the local space transformations of the object they are connected to.

Users can change the view position and orientation in the 3D scene view panel. The view can be moved by dragging with the left mouse button; this will translate the view in a direction perpendicular to the view direction. The view can be rotated by dragging with the right mouse button; this will rotate the camera around the point it is focused on. The view can be moved towards and away from the focus point with the mouse scroll wheel. Additionally, the user can focus the view on the selected component (scene object, light, or camera) by using the “Focus view” toolbar button listed in Table 4.1. The “Focus View” toolbar button has an icon that looks like an eye.

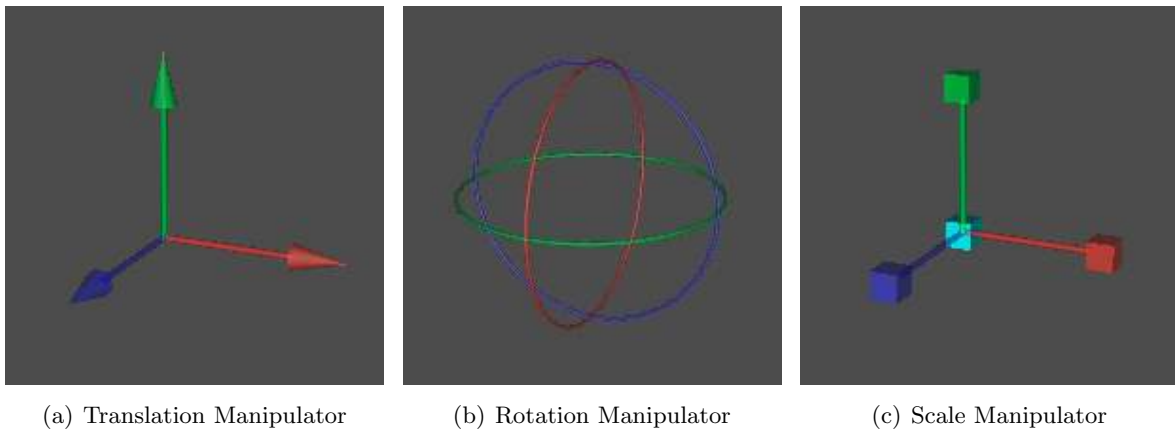


Figure 4.2 Transformation Direct Manipulation Tools

The user interacts with the manipulators with the mouse. When the mouse hovers over the manipulator, then the manipulator color is changed to a dull yellow and reverted back when the mouse is no longer over the manipulator. The area that the mouse can hover over the manipulator is larger than the rendered geometry; there is a separate collision geometry that

is scaled larger than the rendered geometry. When the user interacts with a manipulator by clicking and subsequently pulling, the manipulator is drawn in a bright yellow.

The manipulators are drawn to follow the object they are editing in screen space. This means that the rendered size of the translation, rotation, or scale manipulators does not change based on the distance from the editing object and the view camera.

Editor view panel (4.1D) The editor view panel provides users with an interface to edit the specifics of the particular scene component that is selected in the scene components panel. The editor changes when the selection of the scene components panel changes. There are different interfaces that fill the panel depending on the type of the scene component that is selected. There are interfaces for scene object, light, background, foreground, scene, and camera. Figure 4.3 and Figure 4.4 show the interfaces for those component types.

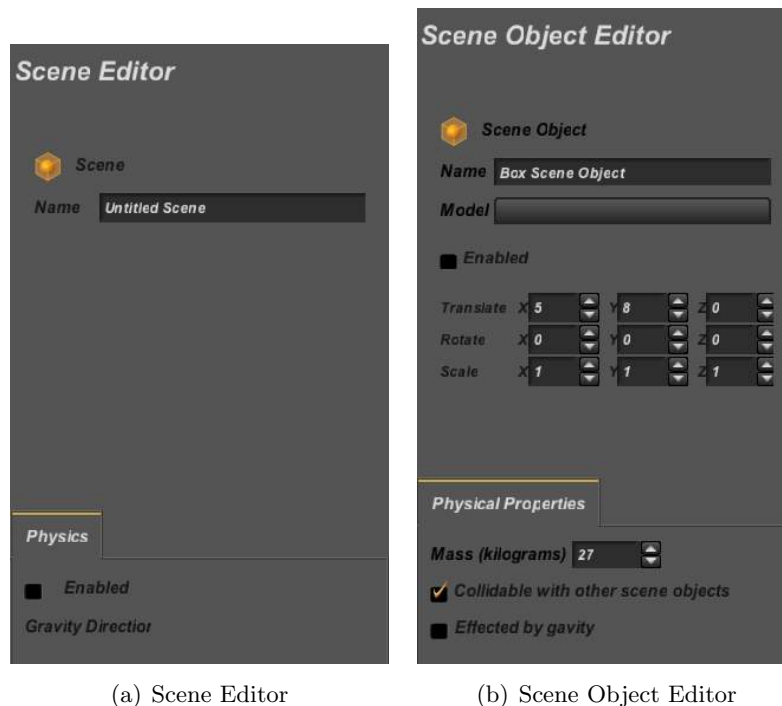


Figure 4.3 Transformation Direct Manipulation Tools

The scene editor interface in Figure 4.3(a) can be used to edit the name and physics



Figure 4.4 Transformation Direct Manipulation Tools

properties of a the selected scene. The physical properties of a scene that can be edited are the direction of gravity in world space and whether the physics simulation is enabled for the selected scene.

The scene object editor interface in Figure 4.3(b) can be used to edit the name, transformation, and physics-base properties of a the selected scene object. The translate, rotate, and scale controls can be used to alter the local space transformation of the scene object. These are the same properties that the transformation manipulators in the 3D scene view panel edit, except that instead of using direct manipulation the user inputs the numerical values directly. The physical properties of a scene object that can be edited are the mass, whether the object is collidable with other objects in the scene, and whether the object is affected by gravity.

The background editor interface in Figure 4.4(a) can be used to edit the name, type, and color properties of the selected background. The combobox GUI element can be used to change the type of the background, when the type of the background is changed then the icon to the left of the type also changes. Pressing the button for the background color will open up a color chooser dialog window that can be used for selecting the color.

The camera editor interface in Figure 4.4(b) can be used to edit the name, type, field of view, and the transformation of the camera. The translate, rotate, and scale controls are used in the same way as for scene objects. Changing the type of the camera behaves the same way as for backgrounds.








The light editor interface in Figure 4.4(c) can be used to edit the name, type, enabling, colors, and the transformation of the light. The translate, rotate, and scale controls are used in the same way as for scene objects. Changing the type of the camera behaves the same way as for backgrounds. The ambient, diffuse, and specular color of the light open a color chooser dialog window when pressed.

Main toolbar and menu Table 4.2 lists the icons and actions present in the main toolbar. There are also corresponding menus in the main menubar for the window. The bottom of the window contains a status bar where the status of operations performed by the builder are notified to the user with use of text and a progress bar.

Undoable actions Most of the actions performed by the user are reversible. Using the undo or redo button in the main toolbar or the undo or redo menu in the main menubar. The undo and redo buttons and menus provide a short description of the command to be undone/redone. For example, if the user creates a new scene in the project then the undo button will have a tooltip with the text "Undo create new scene" and similar text for the redo button if applicable.

The design of the builder interface went through four iterations. Appendix D shows the sketches of the builder interface from version 1 to the final version used in the study. During the design and evaluation of versions 1, 2, and 3 the sketches were presented to a handful of

Table 4.2 Main Toolbar Buttons

Button	Action
	Open project from filesystem
	Save project to filesystem
	Create new project
	Undo last edit
	Redo last undone edit
	Run the current scene in player mode
	Open the help window

undergraduate students, digital artists, and Human Computer Interaction graduate students. The input from these students was used to refine the subsequent versions.

Materials and Methods

Overview of the Study

We developed a user study to test the effectiveness of the Builder user interface, determine the strengths and weaknesses, and to evaluate the complexity of virtual environments that users could create in a limited time. Participants were given a set of introductory tasks, with instructions, to study the effectiveness of the interface and to give the participants experience with the application. After the introduction tasks, the participants were given the task of developing a scene that could be used as a virtual environment. The users were given a post-study questionnaire about (1) The effectiveness of the interface to perform the tasks, (2) the quality of the scenes they created, and (3) the use of the KabalaEngine as a creative tool.

Apparatus

Participants used a 2008 MacBook Pro laptop with an NVIDIA GeForce 9600M GT graphics card. A mouse was used because the KabalaEngine builder interface uses right mouse input for some actions.

Procedure

Potential participants were contacted through e-mail from university mailing lists, fliers placed around campus, and the psychology research student pool. Individuals that responded were sent an email with a copy of the consent form, and times they could choose for participation. Upon arrival to the study, participants were given a hard copy of the consent form to read and sign. Participants that consented were given a pre-survey about their age, sex, and computer experience.

One experimenter was present in the user study lab. When participants arrived they were given a document that contained an overview of the software, and a series of tasks to complete. [Appendix C](#) is the overview and task document. The tasks were arranged into two sections. The first 40 minute section contained five introductory tasks, these tasks were designed to introduce the users to the features of the software. The participants were then given up to a 10 minute break. Next, participants started the second section task. The second section gave users 50 minutes to use the software to create a new project with up to three scenes. For each scene, participants were asked to create a scene from a set of scenarios. Participants were asked to put 10-20 scene objects in each scene. After the second section participants were given the exit survey with questions about the effectiveness/quality of the interface and their opinions on it's use. [Appendix A](#) was the survey given. The participants were finally given a debriefing document and allowed to ask any further questions before leaving.

Participants were provided with a set of 3D models for use when performing the tasks. The models were made up of animals, the video game Halo, indoor furniture, famous monuments, generic humans, space, the movie Star Wars, and vehicles. The models were gathered from the publicly available 3D model repository at <http://sketchup.google.com/3dwarehouse/>.

Measures

Participants were measured by means of a written questionnaire containing 5 point Likert scale questions, and three written questions. [Appendix A](#) was the questionnaire given. While the participants were using the application, their mouse input, keyboard input, and screen were recorded with Morae screen recording software. The saved file for the project they created was kept for analysis.

Participants

There were nineteen participants (9 male, 10 female; $M(\text{age}) = 19.3$, $SD(\text{age}) = 2.3$). Some of the participants received course credit for their participation. Most (95%) of participants rated themselves as moderate to expert computer users. Most (84%) of participants played video games monthly or more, while 16% played video games rarely or never. Figure 4.5 shows the responses of users to the computer experience question (Figure 4.5(a)) and the video game experience question (Figure 4.5(b)).

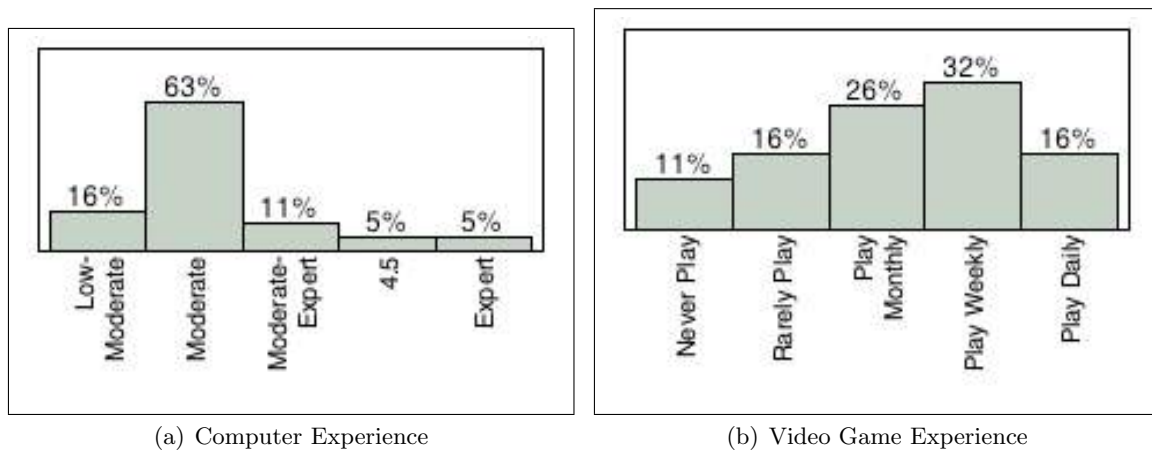


Figure 4.5 Participant Experience

Results

The raw data from the survey is listed in [Appendix B](#).

Task difficulty Most participants (79%) described the tasks as very easy, easy, or neutral in difficulty, 21% described the tasks as difficult, and no participants described the tasks as very difficult. All participants described the solution to performing the tasks to be straightforward half the time, frequently, or always. Figure 4.6 shows the distribution of user's answers to the task difficulty (Figure 4.6(a)), and task solution straightforwardness question (Figure 4.6(b)).

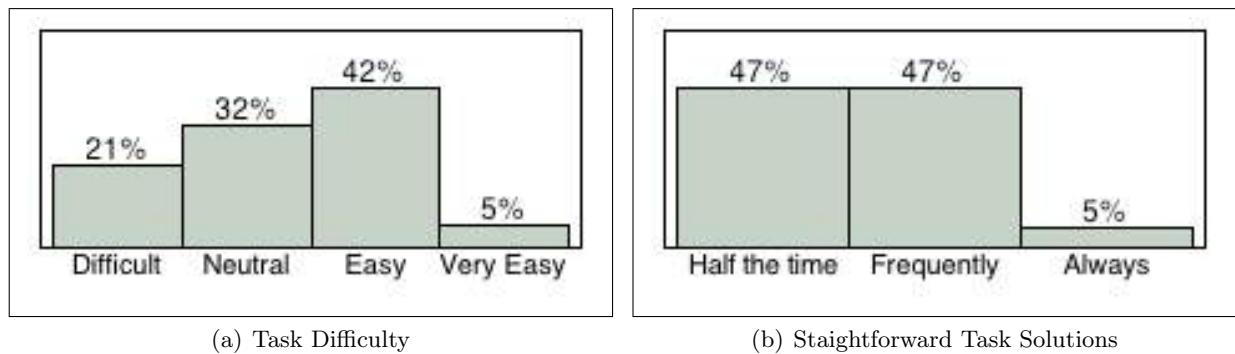


Figure 4.6 Task Survey

The response that participants gave for how engaging the virtual environments they created were varied; 34% described it as not at all realistic or somewhat realistic, 50% neutral, and 17% realistic-very realistic. Figure 4.7 shows the distribution of user's responses.

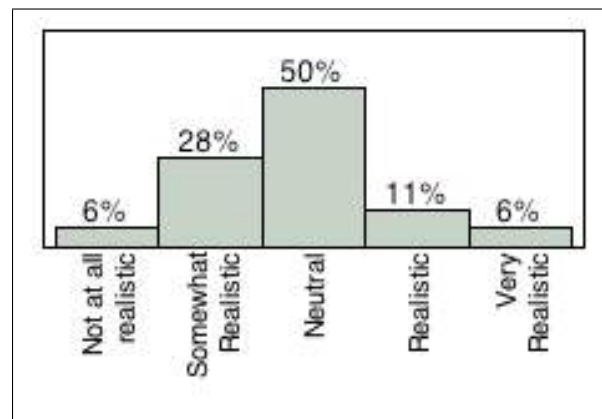


Figure 4.7 Project Realism

With the question of whether they would use the KabalaEngine Builder in their free-time outside of work, 53% of users would use it monthly or weekly, while 48% would use it rarely or never. Figure 4.8 shows the distribution of user's answers.

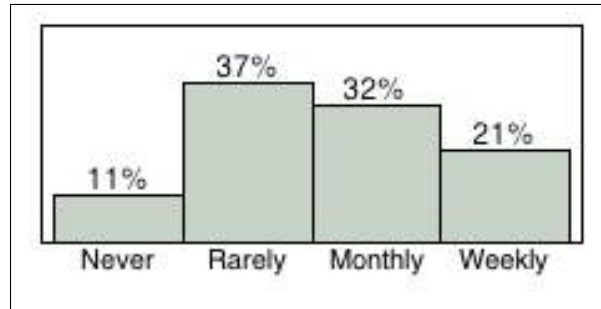


Figure 4.8 Software Freetime Use

Created Users Projects

Figure 4.9 shows screenshots from the three scenes created by a single participant during the 50 minute scene creation task. The screenshots were taken by moving the view so that all of the objects in the scene were visible. For some scene objects, such as the green ground plane, the screenshots cut off parts that were unnecessary. Figure 4.9(a) appears to be an aquatic scenario, possibly a fish-tank-like scenario; the orca, clownfish, penguin, and the great sphinx model were used. The clownfish appear to be schooling as predatory orca and penguin pursue them. Figure 4.9(b) appears to be a generic park scenario. The park scene has a dog and owner playing with a hoop and another dog and owner next to a car, oddly there is a Harrier jet in the scenario. Figure 4.9(c) is a space battle made from models of a famous science fiction movie. The scenarios created by this participant show that the user was comfortable and skilled at placing, orienting, and scaling objects to create interesting environments. The participant also changed the background color for each scenario to an appropriate color.

Figure 4.9 shows screenshots from scenes created by four different participants. The screenshots were taken in the same way as in previous figures. Figure 4.10(a) shows a space scene where the user placed objects, changed the background, and changed the light. The light

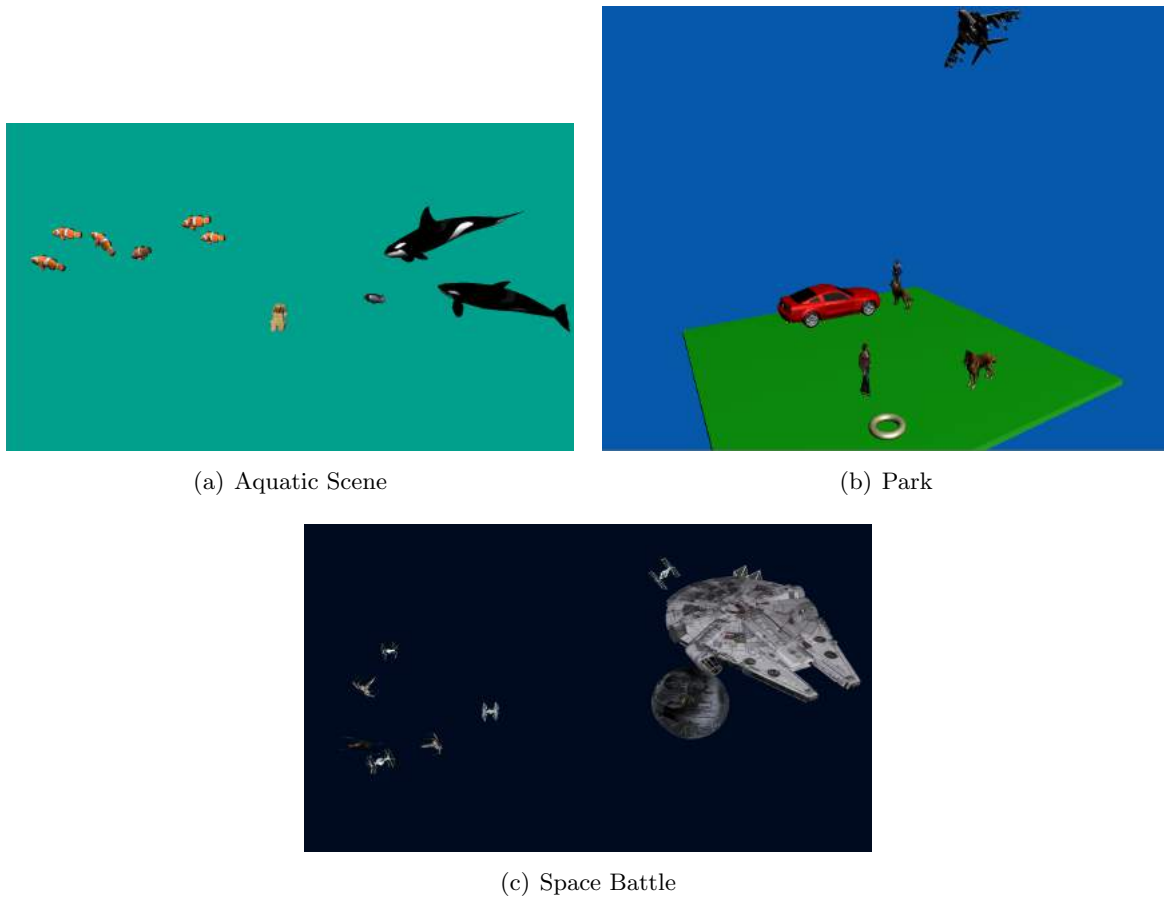


Figure 4.9 Project created by a single participant in 50 minutes

changes brought down ambient values and changed the hue of diffuse and specular light toward orange/yellow. Figure 4.10(b) is a relatively complex social scene, where there are a group of individuals interacting, a lone individual in a dominant and awkward position of standing on a table, and one individual resting on a couch. Figure 4.10(c) is a scene that appears to represent a typical scenario in the Halo video game. The primary humanoid, Master Chief, is being surrounded by enemy forces, Covenant. Figure 4.10(d) is an example of one of the bizarre scenes created. There are oddly proportioned dogs, a human on the nose of an orca that is coming out of the ground, some flowers, and a clownfish.

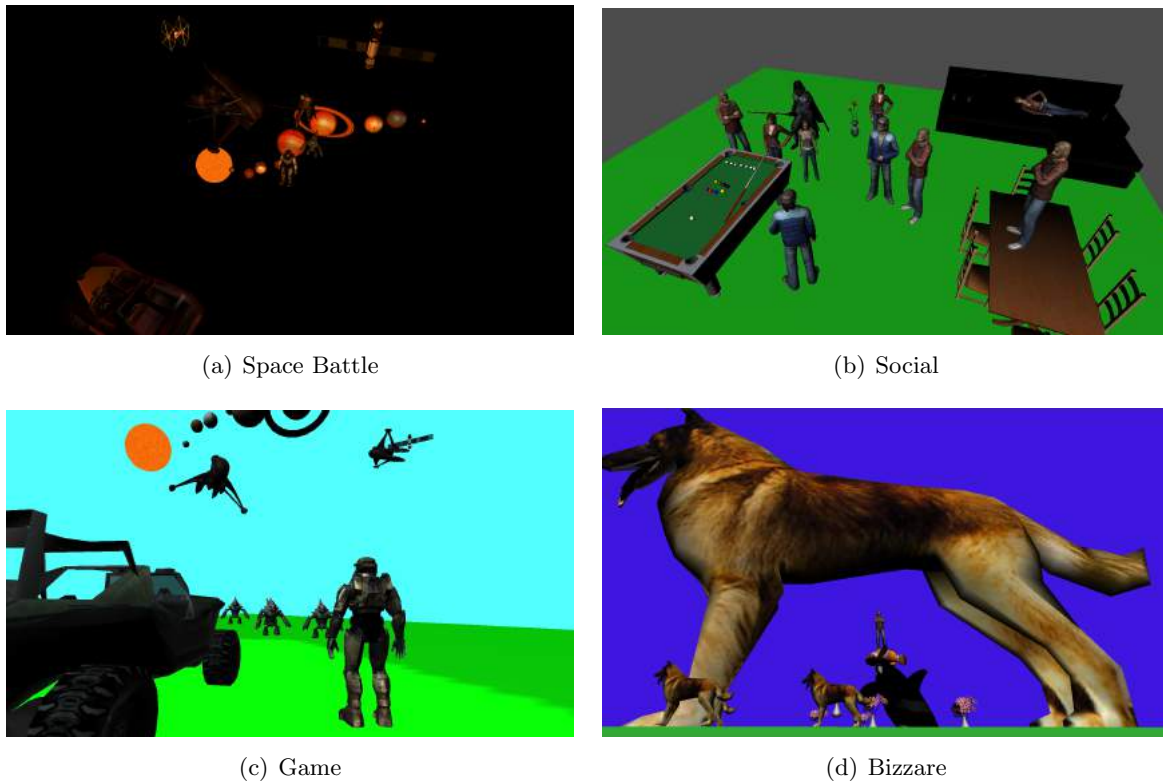


Figure 4.10 Scenes created by participants during 50 minute task

Discussion

The source code repository for OpenSG can be obtained at http://github.com/djkabala/OpenSGDevMaster_Toolbox. The source code repository for the OpenSGToolbox can be obtained at <http://github.com/djkabala/OpenSGToolbox>. The source code repository for the KabalaEngine can be obtained at <http://github.com/djkabala/KabalaEngine>.

What Worked Well?

The study tasks were not too difficult because all of the users were able to complete all of the tasks. This included the task of creating the scenes in the second section, 50 minute task. The scenes that users created were fairly complex given the 50 minute time frame they had to produce them. Most participants created scenes with many objects arranged logically,

changed lighting characteristics, and changed physical properties.

We received many encouraging remarks from users from the "What did you find engaging about using this software?" question of the questionnaire. Several of the users wrote that they enjoyed creating their own scenes. There were comments about the builder providing very rapid development, many editable parameters of objects, and that objects could collide. Users liked being able to control the physics and lighting characteristics of the scenes.

What Did Not Work?

The written response from users indicates that there are problems with the view navigation in the 3D view panel. This is because the navigation is using an unconstrained trackball model. This allows rotation around the views z-axis, which is very disorienting for users. The navigation was changed so that the camera would not roll and provided better results. There were some users that had difficulty using the rotation manipulator. Some users indicated that they felt uncomfortable not knowing how everything worked or having no experience with constructing virtual environments.

Some improvements suggested by participants were: add easy cloning of scene objects, provide some pre-defined view orientations, add basic shapes, and to include more models including terrain. All of these suggestions can be incorporated into the Builder. Adding easy cloning could be as simple as adding a copy/cut/paste mechanism to the Builder. Pre-defined view orientations could be added as a menu next to each of the 3D viewports. Basic shapes can be added to the new scene object button, there are implementations in OpenSG for creating the shapes: sphere, box, cone, cylinder, plane, teapot, and torus. Adding more models and terrain can be done by finding more models in 3D model repositories. The participants were limited to the number of models available so that they would not spend the majority of their time looking for which model to use.

Limitations and Future Work

The interface uses primarily a WIMPS-base style which is very useful for work on desktops, it doesn't have a specialized interface for using the Builder in a virtual reality system or hand-held devices. The Builder does not have a easy user interface for dynamic relationships such as animations, particle systems, and scripts. The KabalaEngine run-time debugger supports this through a generic "expert" user interface. Currently there is not a large user community for the KabalaEngine. The importing of assets can be cumbersome. To run a project in a virtual reality system the system requires VR Juggler.

Acknowledgements

The authors would like to thank Achyuthan Vasanth, Daniel Guilliams, Robert Goetz, Eric Langkamp, and David Naylor for there contributions of source code. The authors would like to thank William Schneller, Amy Dixon, and Heidi Sinsel for their work designing the look and feel and the KabalaEngine logo. The user interface library is released under the General Public License version 3.

References

- CONWAY, M., AUDIA, S., BURNETTE, T., COSGROVE, D., CHRISTIANSEN, K., DELINE, R., DURBIN, J., GOSSWEILER, R., KOGA, S., LONG, C., MALLORY, B., MIALE, S., MONKAITIS, K., PATTEN, J., PIERCE, J., SHOCHET, J., STAACK, D., STEARNS, B., STOAKLEY, R., STURGILL, C., VIEGA, J., WHITE, J., AND WILLIAMS, G. 2000. Alice: Lessons learned from building a 3d system for novices. In *Conference on Human Factors in Computing Systems: Proceedings of the SIGCHI conference on Human factors in computing systems*. –.
- LOMBARDI, J. AND LOMBARDI, M. 2010. *Opening the Metaverse*. Springer, London.
- PAUSCH, R., BURNETTE, T., CAPEHAR, A. C., CONWAY, M., COSGROVE, D., DELINE, R., DURBIN, J., GOSSWEILER, R., KOGA, S., AND WHITE, J. 1995. A brief architectural overview of alice, a rapid prototyping system for virtual reality. *IEEE Computer Graphics and Applications*, 195–203.

SHNEIDERMAN, B. 1987. *Designing the user interface : strategies for effective human-computer-interaction*. Addison Wesley, University of Maryland, College Park.

SMITH, D. A., RAAB, A., REED, D. P., AND KAY, A. 2004. Croquet: a menagerie of new user interfaces. In *Second International Conference on Creating, Connecting and Collaborating through Computing*. 4–11.

CHAPTER 5. EXAMPLE WORKS THAT USE OUR CONTRIBUTIONS

Overview

The KabalaEngine was designed to support the needs of many different types of projects. The real-time features can be used by video game projects, Metablast is a video game that uses the KabalaEngine. The libraries developed for OpenSG and the KabalaEngine were used by the commercial software Quarterback Development System.

MetaBlast

The KabalaEngine has been used by the Metablast video game. Metablast is video game that puts players in an interactive plant leaf cell. The player can move around the game environment in a miniature bioship. Figure 5.1 is a screenshot of Metablast with the bioship in a leaf cell. The component of the cell, organelles, can be seen surrounding the ship. This scene uses the physics integration with OpenSG to propel the ship in a convincing fashion, the ship also realistically collides with boundaries of the organelles. The 3D models were created in the Autodesk Maya software and were exported to the COLLADA file format. The extension of the OpenSG COLLADA importer allowed the same CgFX material definition that the artists used in Maya to be replicated exactly by the engine. The back of the bioship emits bubbles, represented with a particle system, when the ship is powering itself forward.

The Metablast video game takes advantage of all the features created for the KabalaEngine. This includes the animations, COLLADA file importing, GUI library, Lua scripting, particle systems, physics, sound, and video.

Figure 5.2 shows the main menu of Metablast. The main menu contains a set of buttons, that animate a pulse effect when the mouse hovers over them. There are also background

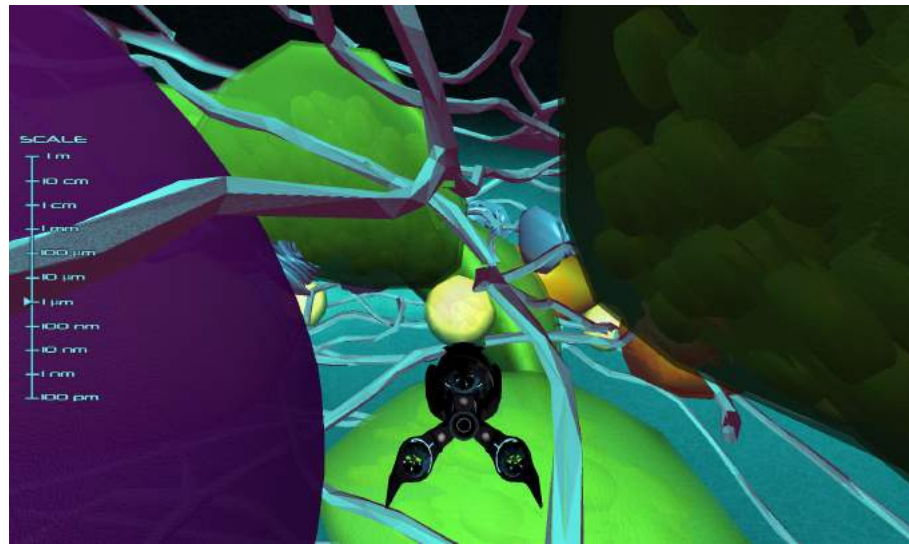


Figure 5.1 Inside a Leaf Cell in the Metablast Application

components that animated fluidly to give the interface an organic feel. The user interface was made using the graphical user interface library. The animations were created using the animation library. Lua scripts were attached to the invocation of the buttons.



Figure 5.2 The Main Menu of the Metablast Application

Figure 5.3 shows the log menu of Metablast. The log menu in Metablast supplements material presented in the interactive game with greater detail. The functions and descriptions

of important molecules, proteins, and organelles are presented. The KabalaEngine provides support for this with the GUI, and the video playback library.



Figure 5.3 An Information Log GUI in the Metablast Application

Quarterback Development System

The Quarterback Development System (QDS) software uses OpenSG, the user interface library, and the video playback library. The QDS software is a commercial package designed for managing American football plays in a playbook, and playing high field of view videos back interactively. Figure 5.4 shows playbook manager interface; the interface is using the OpenSG user interface library described in Chapter 2. Figure 5.5 shows warping of the video for playback. The videos are captured with a high ($\sim 170^\circ$ horizontal) camera. The scene graph is used to draw geometry that the video is then drawn on using the video library. The geometry used to render the video on is designed to remove the warping in the video caused by the camera lens when viewed in the 3D scene.



Figure 5.4 The QDS Playbook Manager Interface



Figure 5.5 Warping the video playback in QDS

CHAPTER 6. GENERAL CONCLUSIONS

In the previous chapters we have described contributions to facilitate the authoring of virtual reality applications. In Section 1 we laid out the research goals we strove to complete. In Section 1 we laid out the challenges we would address to reach our research goals. We will describe how the challenges were addressed and future research that could come from our work. We will finish with an overview of the limitations of our work and final thoughts.

Challenges Addressed

Single GUI framework for desktops and clustered VR environments

Creating a graphical user interfaces that can be used on desktop and VR systems is difficult. There are many mature libraries for graphical user interfaces for desktop systems, and there are some that can be used in 3D. However, there is a need for graphical user interfaces that can be used in both desktop and VR systems. The primary difficulty is handling clustered architectures used by many VR systems.

In Chapter 2 we discussed our contribution of a graphical user interface library to OpenSG. This library can be used in clustered architectures as well as desktop systems. In addition the library supports features common to most GUI libraries. This can make creating a GUI, or switching from another library to this one easier. The same GUI can be attached to a foreground or in the 3D scene. This makes switching from a desktop to a distributed system consistent and faster. The addition of reflexive event production to OpenSG facilitates the advantages of using an event driven programming architecture. The implementation of the OpenSG user interface library provides a straight forward method for simulating mouse events in a VR system using a 6-DOF tracker.

Support for real-time features in clustered environments

There are many common features of real-time applications that are not directly supported for clustered VR systems. In Chapter 3 we discussed the our contributions of libraries for animations, morphs, deformable geometries, particle system, physics, sound, video, importing assets, and scripting to OpenSG.

The animation library provides a generic approach for creating animations that can be applied to any field of a FieldContainer in OpenSG. The particle system library can simulate and render complex particle phenomena using highly configurable particle generators and particle affectors. Morphs and deformable geometries are useful contributions to OpenSG for making complex, dynamic geometries such as using morphs for simulating mouth movements during speech or using deformable geometries to model avatars with walking animations. The contribution to the OpenSG COLLADA loader to handle animations, morphs, deformable geometries, and CgFX materials streamlines the process of digital asset creation in digital content creation programs into a real-time 3D engine like the KabalaEngine. Embedding a Lua virtual machine and creating bindings to the c++ objects of OpenSG with Lua was contributed. This provides an easier method for creating VR applications because Lua code can be changed at run time. Wrappers for sound, video playback, and physics were contributed. Finally a generic GUI for editing FieldContainers at run-time was contributed to OpenSG. This can move the definitions of data used by an OpenSG application to using the generic GUI instead of coding in c++. These contributed features have significantly increased the breadth of features available to developers of clustered VR applications using OpenSG.

Expert UI for creating virtual environments of scenes

In Chapter 3 we discussed our contribution of the KabalaEngine, an application that can support expert developers by moving the definition of content and behavior from compiled code to data that is created using a graphical user interface and saved to a file. The KabalaEngine utilizes the libraries contributed from the previously discussed challenges. The expert GUI provides interfaces for changing the scene-graph, executing Lua code, observing errors and the

stack trace from Lua, and a direct manipulation interface for modifying objects in the scene graph.

Novice UI for creating virtual environments of scenes

It is a goal of virtual reality research to make authoring virtual environments easier. Ideally, anyone that has an idea for a virtual environment should be able to create that environment easily.

In Chapter 4 we discussed our development of the KabalaEngine to simplify the complexities of authoring virtual reality applications. We contributed a novice-oriented user interface to the KabalaEngine called KabalaEngine Builder. This interface allows users to construct a project using scene objects, backgrounds, foregrounds, camera, lights, and other scenes.

The KabalaEngine Builder interface was tested with 19 users. We found that participants were able to finish all assigned tasks. Participants could use the interface to produce three different complex virtual environments with 10-15 different 3D objects arranged in a meaningful way in 50 minutes. Survey results revealed that 53% of participants would use the software outside of work.

Outcome Concerning Research Goal

[[[The goal of this research was to develop tools and applications that support the authoring of virtual reality applications. The tools will support development of VR applications based on common requirements of the hardware and architecture used in VR systems. Building on these tools we will develop an application for novice and expert users to develop virtual reality applications using a graphical user interface. The goal of the application for novice users was to reduce the time and technical skill required to develop VR applications.]]]

Future Research

Future work with the OpenSG user interface framework could include rendering GUI components with 3D filling geometry or as a textured surface over non-planer geometry instead

of rendering the GUI over a flat planes, support for additional direct manipulation metaphors in VR environments, through body-based gestures, and support for easier input of textual information such as through tablets or other mobile devices.

Future work for KabalaEngine could include better management of assets using a database backend instead of a file-base one. Future work for the KabalaEngine Builder could involve creating an interface for dynamic behaviors. Additional studies could be made to with two part studies where users can create projects on a desktop in the first part, and then experience them in a virtual reality system in the second part.

Limitations

3D GUI library The described configuration for simulating the desktop mouse allows the same user interface to be used across desktop and VR platforms with some limitations. Mouse simulation with a 6-DOF tracker requires the user to continuously(actively) keep the cursor in its location(cannot release the tracker, unless it is attached in some way). However, in desktop systems a mouse controlled cursor maintains its position when the user releases the mouse. Mouse directed input has greater precision than many 6-DOF trackers, this can make using the interface more difficult on VR platforms.

OpenSG Support Libraries There are a number of limitations of the support libraries for OpenSG. The following is a description of some of the most relevant. The animation library. Morphs and deformable geometries cannot be combined. The particle system library is implemented to be run by the CPU, extending to also have a GPU implementation can improve performance under some common circumstances. The extensions to the COLLADA importer do not fully implement the entire specification of the COLLADA file format, and their is no COLLADA exporter. There are other scripting languages other than Lua that developers may find more appropriate for their circumstances. The physics library uses OpenDynamicsEngine. ODE is stable, but not being further developed so does not support more modern real-time physics features such as soft-body physics.

KabalaEngine Expert GUI The primary limitation of the expert GUI is that it is an expert GUI. It has a high learning curve. To truly master the use of the expert GUI users must have a deep understanding of OpenSG and the our supporting libraries. The GUI also suffers from some user interface design issues that more user studies could help.

KabalaEngine Novice GUI The KabalaEngine Builder provides a easy means for novice users to modify basic aspects of a virtual environment. Support for a novice interface for the advanced features we created with the OpenSG support libraries could greatly increase the complexity of the virtual environments that could be constructed.

Final Thoughts

This research has been more difficult and time consuming than originally expected. Developing general purpose libraries requires good requirement definitions, good software design, good programming, good maintenance, and good software building tools. Sometimes complex algorithms can interact “very strangely”, until understanding is gained, with the architecture of dependent libraries. Because of this, it has been a tremendous learning experience about the best practices of software design and programming. Given these hardships, the work has been exceedingly rewarding and well worth it for me. The fact that projects such as Metablast and even commercial software use our contributions has solidified my conviction that these are important contributions to VR, and real-time 3D developers. I look forward to continuing work to bring more people tools for authoring virtual reality environments.

APPENDIX A. NOVICE USER SURVEY

The following is the Internal Review Board approved survey used for the study conducted for the manuscript of [Chapter 4](#).

Exit Survey

Participant random id: _____

Age: ____

Sex: Male / Female

What is your major/program of study: _____

Using the scale below, rate your technical expertise with computers:

Low (need assistance for many daily computing tasks)	Low-Moderate (can do most things but occasionally need assistance)	Moderate (able to complete daily computing tasks)	Moderate-Expert (teach others "expert" features)	Expert (fluent in one or more programming languages)
1	2	3	4	5

Using the scale below, please rate how often you play video games:

Never Play	Rarely Play	Play Monthly	Play Weekly	Play Daily
1	2	3	4	5

----Stop here, you will complete the rest of this survey after you have completed all tasks.---

Exit Survey

Using the scale below, how difficult were the tasks you just completed:

Very Difficult	Difficult	Neutral	Easy	Very Easy
1	2	3	4	5

Using the scale below, could tasks be performed in a straight-forward manner?

Never	Infrequently	Half the time	Frequently	Always
1	2	3	4	5

Using the scale below, rate how engaging do you feel the virtual environment you created was:

Not at All Realistic	Somewhat Realistic	Neutral	Realistic	Very Realistic
1	2	3	4	5

Using the scale below, would you use this software outside of your job as a creative tool?

Never	Rarely	Monthly	Weekly	Daily
1	2	3	4	5

What did you find frustrating about using this software?

What did you find engaging about using this software?

Using the spectrum below, how seriously did you take these tasks?

Not Seriously at all	Not Seriously	Neutral	Seriously	Very Seriously
1	2	3	4	5

Please share any additional comments you may have on the back.

APPENDIX B. KABALA ENGINE BUILDER USER SURVEY RAW DATA

The following tables are the raw data collected from the user survey listed in [Appendix A](#) for the study conducted for the manuscript of Chapter 4.

Table B.1 Survey Questions

Question #	Question
1	Please rate your technical expertise with computers.
2	How often do you play video games?
3	How difficult were the tasks you just completed?
4	Could tasks be performed in a straight-forward manner?
5	How engaging do you feel the virtual environment you created was?
6	Would you use this software outside of your job as a creative tool?
7	How seriously did you take these tasks?

Table B.2 Survey Results

ID	date	age	sex	major	Question 1	Question 2
101	11/15/10	18	male	Architecture	Moderate	Play Monthly
22866	11/15/10	20	male	Math ED	Moderate	Play Weekly
45793	11/15/10	18	female	English	Moderate-Expert	Rarely Play
48414	11/17/10	19	female	Psychology	Moderate	Play Monthly
13276	11/17/10	19	male	Business	Moderate	Play Daily
17215	11/16/10	25	female	Computer Science	Moderate	Play Daily
49739	11/16/10	19	female	AMD & Journalism	Moderate	Never Play
28398	11/16/10	18	female	Computer Science	Moderate-Expert	Play Monthly
13382	11/18/10	18	female	Psychology	Moderate	Never Play
15798	11/18/10	18	male	Electrical Engineering	Expert	Play Weekly
36416	11/18/10	19	male	Construction Engineering	Moderate	Play Weekly
10355	11/18/10	26	male	Electrical Engineering	Moderate-Expert	Play Monthly
35684	11/19/10	18	male	Psychology	Moderate	Play Weekly
45527	11/19/10	19	female	Design	Moderate	Rarely Play
11487	11/30/10	18	male	Biology	Low-Moderate	Play Weekly
35220	11/23/10	18	female	Business Econ	Low-Moderate	Play Weekly
22001	12/1/10	19	male	Aerospace	Moderate	Play Daily
33291	12/2/10	19	female	Graphic Design	Moderate	Play Monthly
11324	12/3/10	19	female	Horticulture	Low-Moderate	Rarely Play

Table B.3 Survey Results (continued)

ID	Question 3	Question 4	Question 5	Question 6	Question 7
101	Easy	Half the time	Neutral	Weekly	Neutral
22866	Easy	Half the time	Somewhat Realistic	Weekly	Seriously
45793	Difficult	Half the time	Neutral	Rarely	Seriously
48414	Difficult	Half the time	Neutral	Monthly	Seriously
13276	Neutral	Frequently	Not at all realistic	Rarely	Seriously
17215	Difficult	Half the time	Neutral	Weekly	Very Seriously
49739	Neutral	Frequently	Neutral	Rarely	Seriously
28398	Neutral	Half the time	Neutral	Never	Not Seriously
13382	Neutral	Half the time	Somewhat Realistic	Rarely	Very Seriously
15798	Easy	Frequently	Realistic	Monthly	Seriously
36416	Very Easy	Frequently	Neutral	Weekly	Seriously
10355	Easy	Always	Realistic	Monthly	Very Seriously
35684	Easy	Frequently	Neutral	Rarely	Neutral
45527	Neutral	Half the time	Very Realistic	Rarely	Seriously
11487	Easy	Frequently	Somewhat Realistic	Monthly	Seriously
35220	Difficult	Half the time	Somewhat Realistic	Never	Very Seriously
22001	Easy	Frequently	Somewhat Realistic	Rarely	Seriously
33291	Neutral	Frequently	Realistic	Monthly	Seriously
11324	Easy	Frequently	Neutral	Monthly	Seriously

Table B.4 Survey Results (continued)

ID	What did you find frustrating about using this software?
101	Hard to navigate (for me)
22866	The rotating mechanism using alt and the mouse was difficult to control
45793	I didn't like that I couldn't move things by clicking and dragging like I'm used to, and that they only moved in one direction at a time.
48414	Not understanding how to use it fully.
13276	It took awhile to understand how to use
17215	2 axis camera maneuvering!!!! (left click) VERY FRUSTRATING!
49739	Rotating you view became a little difficult
28398	No set camera positions for views. Couldn't add basic shapes. Didn't have ability to clone objects
13382	I'm not very good with virtual things, so just not knowing different tools.
15798	Occasionally I wanted to do more advanced work than the software allowed
36416	Sometimes when I wanted to move objects they would move in the opposite direction I moved the mouse.
10355	I couldn't create multiple objects at one time. The axis of movement on the mouse didn't adjust relative to camera angle when positioning an object.
35684	You couldn't add hills or adjust the models poses, you can only transform models.
45527	It sort of would freeze up at times
11487	Wanted to add my own shapes and lines. I'm just not used to the spatial aspect of putting the shapes in the correct place.
35220	I found the rotation and movement controls frustrating
22001	The rotate tool. Pivoting around the point in space other than the center of object.
33291	Not knowing how everything worked.

Table B.5 Survey Results (continued)

ID	What did you find engaging about using this software?
101	Cool 3D software, fun to play with
22866	The light effect portrayed to give a more 3D feeling
45793	I enjoyed designing environments.
48414	It was interesting and unique.
13276	It created a challenge
17215	Physics, lighting, simplicity of menus
49739	Being able to control everything & create your own scene
28398	Different models
13382	It was fun to mess around with different objects & being able to create your own environment.
15798	Very rapid development
36416	It was very easy to change camera angles and to rotate objects to get them to face the right direction.
10355	Many parameters of objects where adjustable. Allowed free Range of creativity.
35684	the interface was easy to get used to and the transformations that are possible are easy to learn.
45527	You can make realistic scenes
11487	The opportunities are endless and software is basic enough
35220	The ability to be able to create virtual worlds/scenes
22001	When objects collided with one another it seemed realistic.
33291	being able to zoom in and be in the image
11324	I thought creating my own scenes was fun.

Table B.6 Survey Results (continued)

ID	Please share any additional comments you may have.
28398	I think it has potential, it's just still very basic at this point.
15798	This will be fantastic as more features are added, bugs are fixed, and UI is tweaked.
35684	Make the camera easier to see once selected maybe give it a model of some sort it was hard for me to locate at first
11324	I enjoy the program and even though I am usually bad with technology I found it pretty simple to use.

APPENDIX C. NOVICE USER TASKS

The following is the task document used by study participants for the study conducted for the manuscript of [Chapter 4](#).

Kabala Engine: Overview

The software you are using is called the KabalaEngine. You will be using the WorldBuilder of the KabalaEngine to construct scenes in a 3D environment. The WorldBuilder has four primary areas for editing.

Project view
Holds a view of the currently created scenes in the project.

WorldBuilder



3D Scene View
3D view of the selected scene. You can navigate in this view, and manipulate objects in the scene.

Editor view
Holds a view of the editor for the component of the scene selected for editing.

Scene overview
Holds a view of the components of the current scene that can be edited.

Task: Navigation

For this task you will learn how to navigate in the 3D editor.

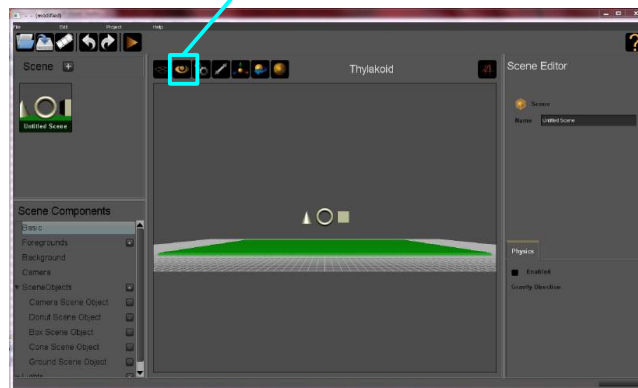
To do this you will need to use the controls for moving the camera.

Rotate: Press and hold the left mouse button and move the mouse.

Move: Press and hold the right mouse button and move the mouse.

Zoom: Move the mouse scroll wheel forwards and backwards.

Focus on specific object: You can focus the camera on a specific object in the scene by left clicking on the object to select it, and then pressing the Focus tool. The Focus tool is above the 3D viewing region and looks like an eye.



Tasks:

1. Rotate the camera 180 degrees around the doughnut shape.
2. Move the camera through the center hole of the doughnut.
3. Rotate the camera 180 degrees
4. Move the camera back through the center hole of the doughnut.
5. Finally, bring the whole scene back into view by using the focus tool to focus on the green ground.

Task: Arrange Objects in Scene

For this task you will learn how to arrange objects in the 3D editor. You may need to use the navigation techniques learned earlier to find a camera location that makes the arranging easier.

To do this you will need to use the controls for moving scene objects.

Select a scene object: Left click on an object in the scene. This will select the object.

Once a scene object is selected there will be a red, green, and blue manipulator handle. You can use the manipulator handle to move, rotate, or scale the scene object.

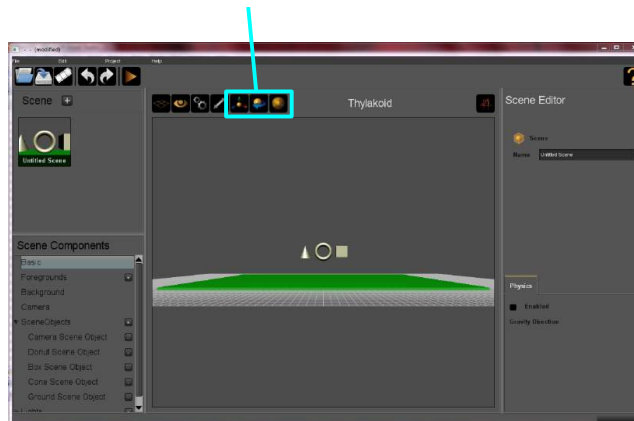
Move: Select a scene object. Next select the move tool. Now you can click and move the scene object.

Rotate: Select a scene object. Next select the rotate tool. Now you can click and rotate the scene object.

Scale: Select a scene object. Next select the scale tool. Now you can click and scale the scene object.

Alternate method: You can also move, rotate, or scale a scene object by selecting the scene object and then inputting the values directly in the editing area on the left of the interface.

Move, Rotate, and Scale tools



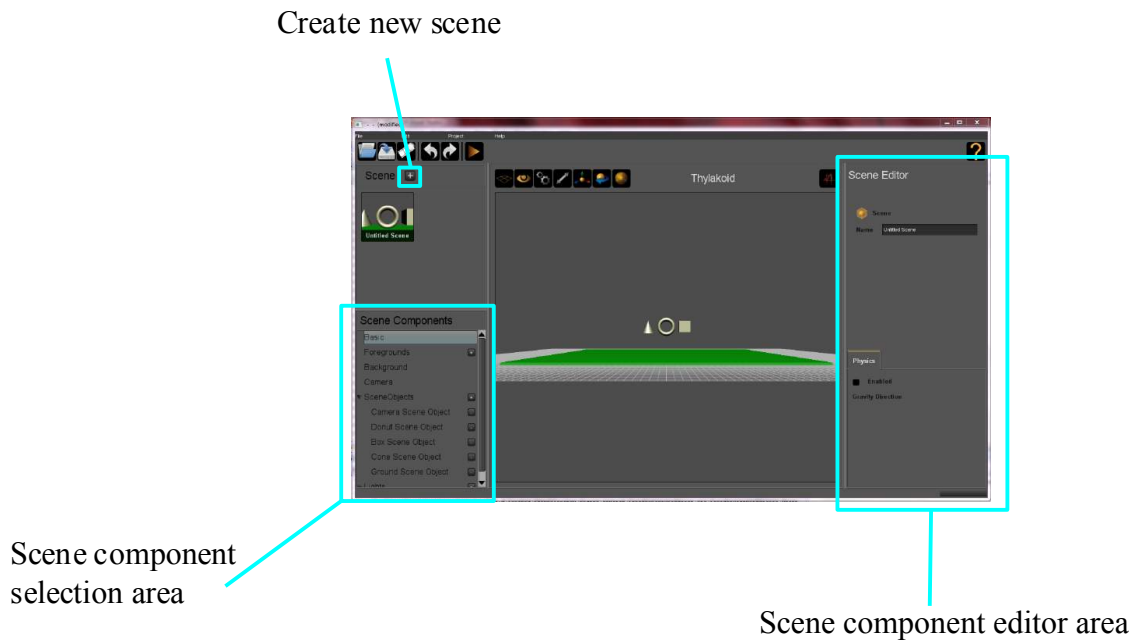
Tasks:

- Move the doughnut to the left of the cone
- Move the box to the left of the doughnut
- Point the tip of the cone toward the doughnut.

Task: Scene Creation

For this task you will learn how to create and edit various aspects of a scene.

1. Create a new scene.
2. Create four new scene objects and place them in the scene such that they are not colliding.
3. Change the background color of the scene.
4. Change the location of the camera in the scene.
5. Change the ambient, diffuse, and specular colors of the light in the scene.



Task: Navigation in player mode

For this task you will use the player mode of the application.

Controls for player mode

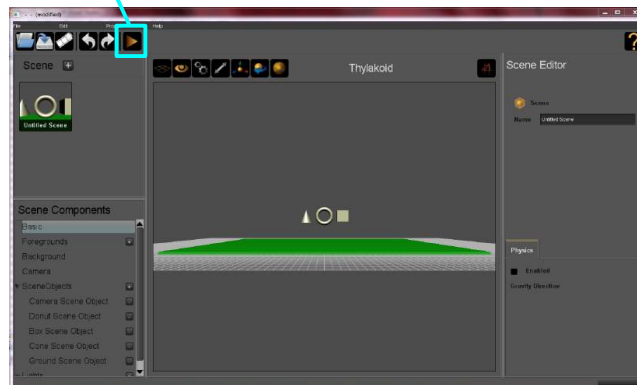
Start Player: Press the play button. This button looks like a triangle and is near the top of the interface.

Exit the player: Press the escape key to exit from the player mode back to the WorldBuilder.

Rotate: Left click and pull to rotate your view.

Move: Use the arrow keys to move forward, backward, left, or right.

Start player mode



Tasks:

1. Start the player mode
2. Move around the scene using the navigation controls
3. Move around so that you run into the box, cone, and doughnut shapes
4. Exit the player mode back to the WorldBuilder

Task: Change Physical properties of scene objects

For this task you will use the interface to change the physical properties of scene objects.

Tasks:

- Create a new scene.
- Add a new scene object to the scene
 - Make the scene object collidable and effected by gravity
 - Use the player mode to see if this was successful
- Make the doughnut, cone, and box collidable and effected by gravity.
- Arrange the objects such that:
 - The Box is at the bottom
 - The flat end of the cone will fall onto the box
 - The hole of the doughnut will fall through the point of the cone
 - Move the camera so that it is viewing your arrangement
- Use the player mode to see if you were successful

Task: Create 3 Scene Project

For this task you will freely use the interface for 50 minutes to create a new project.

- Create at least 3 scenes
- For each scene attempt to create one of the following scenarios
- Put 10-20 scene objects in each scene

Scenarios

- Space Battle
- Indoor social event
- Vehicle race somewhere famous
- Sporting event
- Bizarre world
- A comical situation
- Historical event
- Other

APPENDIX D. KABALA ENGINE BUILDER INTERFACE SKETCHES

The following figures are the design sketches for the KabalaEngine Builder interface describe in the manuscript of [Chapter 4](#).

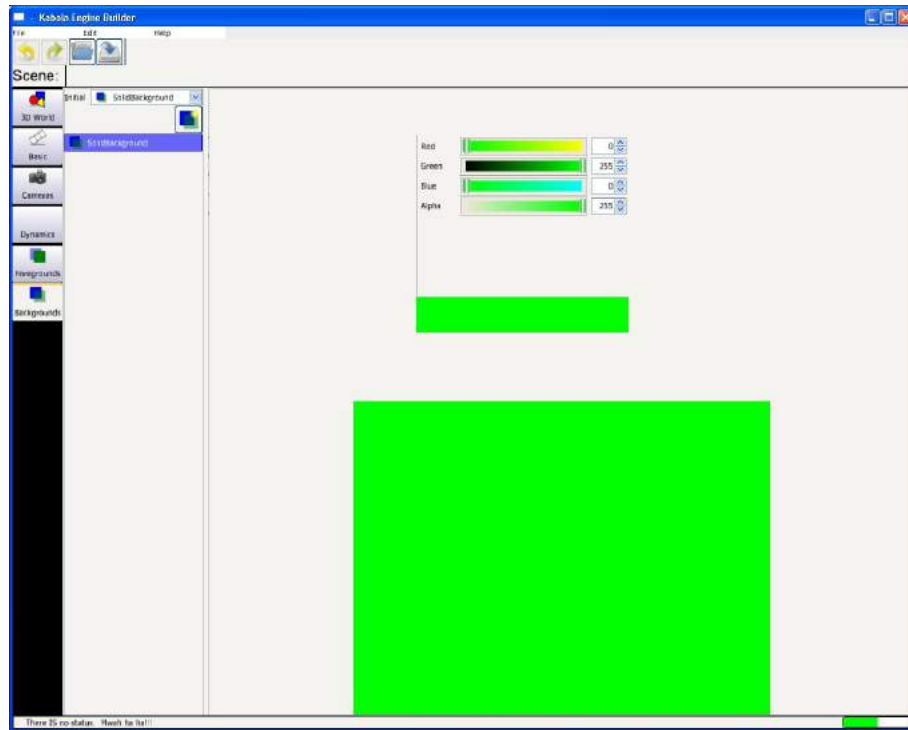


Figure D.1 Version 1

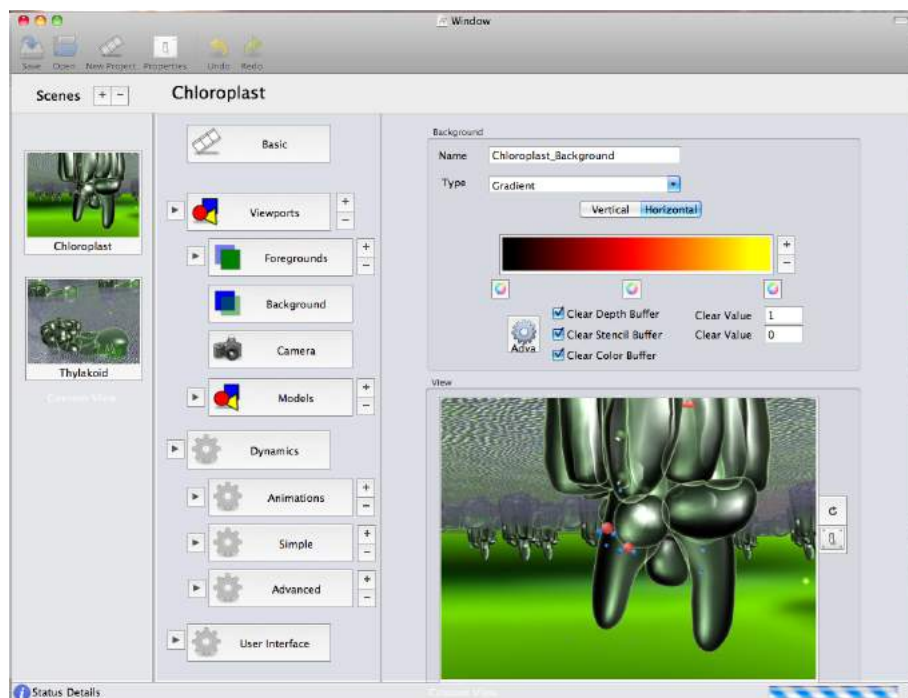


Figure D.2 Version 2

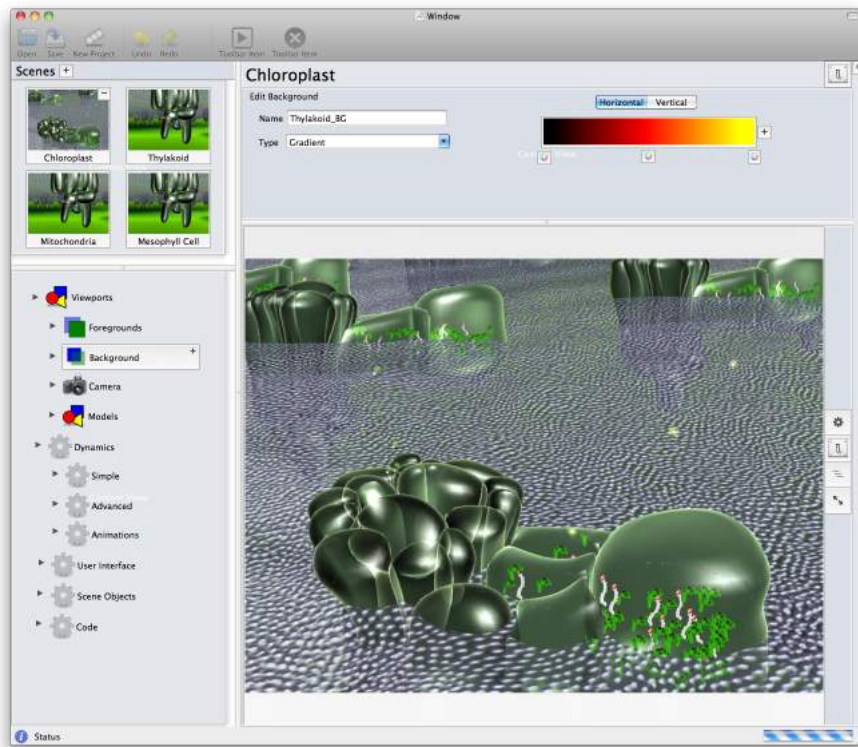


Figure D.3 Version 3

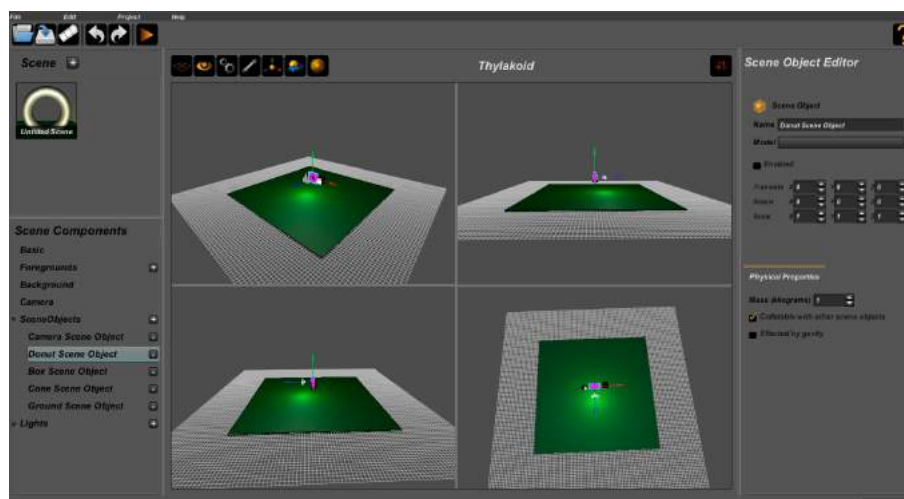


Figure D.4 Version 4/Final