Rui Manuel Moreira

Integrating a 3D application server with a CAVE

UMinho | 2011

Universidade do Minho
Escola de Engenharia

Rui Manuel Ferreira de Carvalho Azevedo Moreira

Integrating a 3D application server with a CAVE

Outubro de 2011

Universidade do Minho
Escola de Engenharia

Rui Manuel Ferreira de Carvalho Azevedo Moreira

Integrating a 3D application server with a CAVE

Tese de Mestrado
Mestrado em Engenharia Informática

Trabalho efectuado sob a orientação do
Professor Doutor José Creissac Campos

Outubro de 2011

DECLARAÇÃO

Nome
Rui Manuel Ferreira de Carvalho Azevedo Moreira

Endereço Electrónico
rui.moreira17@gmail.com

Número do Bilhete de Identidade
13447659

Título da Dissertação
Integrating a 3D application server with a CAVE

Orientador
José Creissac Campos

Ano de Conclusão
2011

Designação do Mestrado
Mestrado em Engenharia Informática

Universidade do Minho, Outubro 2011

Assinatura: Rui Manuel Ferreira de Carvalho Azevedo Moreira

This work was carried out in the context of the APEX project (Agile Prototyping for user EXperience), funded by FCT under contract PTDC/EIA-EIA/116069/2009.

Acknowledgements

I would like to dedicate this work to my parents and sister, for the support they have given me during my academic course and for having the patience to endure my absence at times.
To Jorge, Hélder, Pedro, Francisco, Hugo and Diogo, for their friendship during these past years and for always helping me when I needed the most.
To my supervisor, Professor José Creissac Campos, for his endless availability, dedication and support. Without his guidance, it would had been a lot more difficult to decide which paths to take for this work.
To José Silva, for giving me advice and helping me understand the APEX framework, and to Professor Michael Harrison, for giving his opinion on several aspects of this work.
To the people at the CCG, especially Bruno Aragão, Catarina Mendonça and Jorge Santos, for providing information about the technology present there and for having the availability to show how experiments are done at the Laboratory of Vision and Perception.
To all these people, I want to express my deepest gratitude. If it were not for them, this work would not be completed.

Abstract

Title: Integrating a 3D application server with a CAVE

The prototyping of ubiquitous environments has never been an easy task. Under normal circumstances, this type of system can only be tested during deployment, resulting in complications in the development process. To solve this issue, developers at the Department of Informatics in the University of Minho created the rApid Prototyping for user EXperience (APEX) framework, that relies on a 3D application server to create simulations for ubiquitous environments. The goal is to use these simulations as prototypes of the actual environments, allowing early assessment of how users will experience the proposed design.

The focus of this dissertation is in integrating a 3D application server with a Cave Automatic Virtual Environment (CAVE), for a more realistic approach when using the APEX framework. 3D application servers provide the means to develop virtual worlds. The CAVE is a Virtual Reality (VR) theatre where a user can interact with an immersive environment, through sensors and mobile devices. This dissertation shows a set of solutions to achieve the desired integration, as well as a number of interaction devices that can be used for this purpose.

Fields: Human-computer Interaction, Rapid Prototyping, Ubiquitous Environments, Virtual Reality
Keywords: 3D Application Servers, APEX Framework, CAVE, Motion Capture, OpenSimulator, Second Life, VR Frameworks, Wiimote

Resumo

Título: Integração de um servidor de aplicações 3D com uma CAVE

A prototipagem de ambientes ubíquos nunca foi uma tarefa fácil. Por norma, este tipo de sistema só pode ser testado durante o seu uso efetivo, o que acaba por resultar em complicações no processo de desenvolvimento. Para resolver este problema, foi criada a framework rApid Prototyping for user EXperience (APEX), no Departamento de Informática da Universidade do Minho, que utiliza um servidor de aplicações 3D para criar simulações de ambientes ubíquos. O seu objetivo é utilizar estas simulações como protótipos dos ambientes, possibilitando desde cedo uma avaliação da sua experiência de utilização.

O objetivo desta dissertação consiste na integração de um servidor de aplicações 3D com uma Cave Automatic Virtual Environment (CAVE), no sentido de se obter simulações mais realistas durante a utilização da framework APEX. Um servidor de aplicações 3D providencia meios para o desenvolvimento de mundos virtuais, enquanto que uma CAVE é um sistema de realidade virtual que permite ao utilizador interagir com um ambiente imersivo, por intermédio de sensores e dispositivos móveis. Nesta dissertação, apresenta-se um conjunto de soluções para obter este tipo de integração, assim como alguns dispositivos de interação que podem ser utilizados para este propósito.

Campos: Ambientes Ubíquos, Interação Humano-Computador, Prototipagem Rápida, Realidade Virtual
Palavras-chave: CAVE, Framework APEX, Frameworks de Realidade Virtual, Captura de Movimento, OpenSimulator, Second Life, Servidores de Aplicações 3D, Wiimote

Table of contents

Acknowledgements **v**

Abstract **vii**

Resumo **ix**

Table of contents **xi**

List of figures **xv**

List of tables **xvii**

List of abbreviations **xix**

1 Introduction **1**

1.1 Context 1

1.2 Main objectives 3

1.3 Structure of the dissertation 3

2 State of the art **5**

2.1 Introduction 5

2.2 CAVE 5

2.3 3D application servers 6

2.3.1 Second Life 8

2.3.2 OpenSimulator 9

2.3.3 Comparison 10

2.4 Second Life clients 11

2.4.1 Mesh Viewer 12

2.4.2 Dale's SL Viewer 14

2.4.3 CaveSL 15

2.4.4 Comparison 17

2.5 Virtual reality frameworks . . . 19
2.5.1 VR Juggler . . . 20
2.5.2 Avango . . . 21
2.5.3 CAVELib . . . 22
2.5.4 Mixed Reality Toolkit . . . 23
2.5.5 Comparison . . . 23
2.6 Summary . . . 24

3 A Second Life client for a CAVE 27
3.1 Introduction . . . 27
3.2 Solution I: Using VR Juggler . . . 27
3.2.1 VR Juggler architecture . . . 27
3.2.2 Creating a Second Life client using VR Juggler . . . 29
3.2.3 Invoking Second Life application objects . . . 29
3.3 Solution II - Using CaveSL . . . 31
3.4 Solution III - Using Dale's SL Viewer . . . 34
3.5 Solution IV - Using NVIDIA 3D Vision Surround . . . 36
3.6 Summary . . . 39

4 CAVE deployment 41
4.1 Introduction . . . 41
4.2 Configuring a triple-head setup . . . 41
4.3 CAVE characteristics . . . 44
4.4 Deploying the proposed solutions . . . 45
4.4.1 CaveSL . . . 45
4.4.2 Dale's SL Viewer . . . 46
4.5 The library case study . . . 48
4.6 Summary . . . 50

5 Interacting with mobile devices and sensors 53
5.1 Introduction . . . 53
5.2 Input devices . . . 53
5.2.1 Keyboard and mouse . . . 53
5.2.2 Wiimote . . . 55
5.2.3 Motion capture . . . 59
5.3 Summary . . . 61

6 Conclusion and future work **63**
6.1 Overall analysis 63
6.2 Objective completion 66
6.3 Future work 67

Appendices **69**
Appendix I - Wiimote GlovePIE script (adapted from [Chari (2009)]) . . . 69
Appendix II - NTP configuration files 73

References **77**

List of figures

2.1 User inside a CAVE 6
2.2 Rendering of mesh objects (from [Linden Research, Inc. (2011c)]) . . 13
2.3 Data and display synchronization in CaveSL (from [Haynes (2010)]) . 16

3.1 VR Juggler interfaces for application objects (adapted from [Iowa State University (2007)]) 28
3.2 Using VR Juggler's interfaces for OpenGL applications (adapted from [Iowa State University (2007)]) 30
3.3 Communication between different machines with CaveSL 33
3.4 CaveSL with three clients running simultaneously 35
3.5 Selecting stereo mode in Dale's SL Viewer 35
3.6 Dale's SL Viewer in anaglyph stereo mode 35
3.7 VESA miniDIN-3 connector (from [Bungert (1999)]) 36
3.8 Setting Dale's SL Viewer in window mode 36
3.9 Adjusting display bezels with NVIDIA drivers 38
3.10 Using TitaniumGL 39

4.1 Triple-head setup 43
4.2 CAVE in the LVP 44
4.3 CaveSL implementation for the CAVE 46
4.4 CaveSL running in the CAVE 47
4.5 Dale's SL Viewer running in the CAVE 48
4.6 University of Minho's library plant at Gualtar campus (from [SDUM (2011)]) 49
4.7 Library created in the OpenSimulator server 50

5.1 Wiimote (from [Nintendo (2009)]) 56
5.2 Communication between clients and server using NTP 61

List of tables

2.1 3D application servers comparison 10
2.2 Second Life clients comparison 18
2.3 VR frameworks comparison 23

5.1 Second Life clients' key binding 54
5.2 Wiimote button mapping with GlovePIE 57

List of abbreviations

2D Two-dimensional

3D Three-dimensional

API Application Programming Interface

APEX rApid Prototyping for user EXperience

AR Augmented Reality

AV Augmented Virtuality

CAVE Cave Automatic Virtual Environment

CPN Colored Petri Net

CLI Common Language Infrastructure

DVI Digital Video Interface

FCT Fundação para a Ciência e a Tecnologia

FOV Field of View

GPIO General Purpose Input/Output

GUI Graphics User Interface

HID Human Interface Device

HMD Head-Mounted Display

LAN Local Area Network

LGPL GNU Lesser General Public License

LVP Laboratory of Vision and Perception

MPI Message Passing Interface

MR Mixed Reality

NTP Network Time Protocol

OSG OpenSceneGraph

RAM Random Access Memory

SLI Scalable Link Interface

SMP Simmetric Multi-Processing

SNTP Simple Network Time Protocol

VR Virtual Reality

WTK WorldToolKit

Chapter 1

Introduction

In this chapter, a contextualization is made about the rapid prototyping of ubiquitous environments. The chapter explains how Three-dimensional (3D) application servers and Virtual Reality (VR) systems, such as the Cave Automatic Virtual Environment (CAVE), can contribute for a better assessment of user experience when testing the prototypes of this type of system. The main objectives of the work are delineated and the structure of the dissertation is described.

1.1 Context

Due to the constant growth of information technologies throughout the years, there has been a tendency to equip places of our daily life with ubiquitous computing devices. In ubiquitous computing, a variety of computer devices are commonly used to assist and automate human tasks and activities in the physical world [Poslad (2009)]. For example, public displays might change the information displayed based on the user's location or on where he has been (see [Singh et al. (2006)] for a case study about museums).

Currently, the main issue of developing such environments is that developers have to "rely on either low-fidelity techniques (such as paper prototypes and mental walk-throughs) or simply wait for a full scale deployment" [Singh et al. (2006)] for testing and validation, making it extremely inconvenient and perhaps costly to actually build them. In many cases, it is not possible to install prototypes, as that would be disruptive of ongoing activities (consider the case of a hospital, or an airport). Hence, it is of utmost importance to find means of prototyping ubiquitous environments in a rapid fashion without disrupting key services of human society. The use of immersive environments combined with 3D application servers should enable us to accomplish this.

The usefulness of an ubiquitous computing application can be determined by the developers' skills in identifying users' requirements and expectations. It is exceedingly important that these requirements and expectations are met. By designing prototypes and letting users test them, a developer can better assess the potential impact and relevance that the system may have on a real scenario. [Singh et al. (2006), Hodges et al. (2006)]

Even though prototypes are not intended as final products, they must be both robust and refined for a fair evaluation [Hodges et al. (2006)]. The more detailed and realistic a prototype is, the more a user is able to discern its future utility.

The rApid Prototyping for user EXperience (APEX) framework is a tool, developed at the University of Minho[1], for the rapid prototyping of ubiquitous environments. According to Silva et al. (2010), it achieves this by allowing the user to create Colored Petri Net (CPN) models to define properties and behaviors in the simulation created by a 3D application server. This simplifies the whole process of defining ubiquitous environment prototypes, because the developer can easily add events that simulate devices and sensors that react to the position, movements and commands of the avatar.

3D application servers offer developers a set of tools to develop and explore virtual world environments. These tools allow the creation of geometrical figures such as terrain, objects, characters and also let an undefined number of avatars (users) join a virtual world to interact with it and with each other. This gives developers the flexibility desired to create complex worlds and behaviors.

One of the goals of the APEX project is that the tool must provide prototypes that feel real, by giving the user the means to interact with the virtual environment in a way that feels natural. This could be accomplished by taking advantage of the functionality of a CAVE, which is a VR theatre where a user can be immersed with stereoscopic 3D imaging and many devices to interact with. At the start of this project, however, APEX supported interaction via the traditional Two-dimensional (2D) desktop client applications only.

By definition, VR makes use of computer graphics to create physical world simulations that are responsive to its users' actions, be it through their movements or by using devices [Burdea and Coiffet (2003)]. This means that a VR environment immerses a user in a virtual world and lets him interact with it in a similar way to the real world. VR is a good way to configure scenarios adequate to immerse users with virtual objects around them.

[1]Project supported by the Fundação para a Ciência e a Tecnologia (FCT) under contract PTDC/EIA-EIA/116069/2009 (97 KEuro)

To integrate a 3D application server with a VR environment, such as the CAVE, it is required that the latter must provide simple methods of running applications that use different graphical engines and toolkits (e.g. OpenGL, OpenSG, OpenSceneGraph (OSG), etc.). It is also necessary that such an environment must possess the capability to communicate with many kinds of digital devices, giving the user the possibility to interact with the simulation.

1.2 Main objectives

The expected result of this project should bring forth a solution that permits its users to interact with the simulated scenarios, of a 3D application server, with a much higher sense of immersion than what is currently provided by APEX. Not only does this contribute to make the experience feel more realistic, but it can also, as a consequence, help analyze the viability of the prototypes created by using the APEX framework. More speciffically, the main objectives of this work are:

- To investigate ways to integrate a 3D application server with a CAVE, to use in the context of the prototyping framework being developed in the APEX project;
- To explore means of interaction with the CAVE.

1.3 Structure of the dissertation

Besides this introductory chapter, this dissertation is composed by the following chapters:

- Chapter 2, which concerns the state of the art on the subject of this dissertation. In this chapter, various VR frameworks, 3D application servers and clients are described and compared. A description of the CAVE is also made. The chapter explains why some solutions are adequate for this project while some others are not;
- Chapter 3 presents a set of solutions based on choices made in the previous chapter. Features such as stereoscopic 3D and multiple display support are taken into consideration in order to contribute to user immersion. A ponderation is made whether it is viable or not to create a different client for a 3D application server (using the capabilities of a VR framework, for instance) or to use an already existing one. The advantages and drawbacks of each solution

are discussed and the possibility of combining several of them is also taken into account;

- Chapter 4 relates to the deployment of the solutions described in Chapter 3, into the CAVE. First, it is checked how they work out in a triple-head setup and whether it is viable to use it. Then, the solutions are verified in the CAVE at the Laboratory of Vision and Perception (LVP), where they are tested using a cluster instead of a single machine. The tests are made using a library case study, which is meant to simulate the library at the University of Minho;

- Chapter 5 introduces the use of interaction devices in the CAVE. A solution based on the Wiimote is described, as well as the Motion Capture system at the LVP. Each system provides specific benefits for user interaction which are discussed in this chapter;

- Chapter 6 presents some conclusions and discusses the solutions found, taking into consideration the objectives defined for this project. The benefits and drawbacks of each solution are analyzed to determine which ones provide the best results. Furthermore, it also describes some of the future work that could be done to minimize the drawbacks of the chosen solutions.

Chapter 2

State of the art

2.1 Introduction

This chapter presents the state of the art for this work. Technology that may be used for the prototyping of ubiquitous environments is described here in detail. Some comparisons are made to determine which solutions may contribute to the objectives of this project. In concrete, this chapter makes a description of the CAVE, and of existing VR frameworks, 3D application servers and clients. At the end of each section (except for the one regarding the CAVE), we will determine which solution should be adopted. Then, in Chapter 3, the chosen solutions will be explored in further detail to see whether they are viable for this work.

2.2 CAVE

The CAVE is a system, designed by the Electronic Visualization Laboratory at the University of Illinois at Chicago, for the reproduction of VR environments, by making use of multiple rear projection screens to display image. With this kind of setup, it is possible to surround the user with the visualization, without having the problem of casting shadows to the screens [DeFanti et al. (2011), Creagh (2003), Cruz-Neira et al. (1993)]. Fig. 2.1 shows an example where the user is surrounded by three displays and everything regarding visualization is settled on the backstage. Typically, a CAVE has at least three rear projection screens on the left, right and front of the user. This can be complemented with displays on top and bottom of the user, for example, but it may not always be the case. Some setups may have multiple projectors point at a single flat rear projection screen (which is currently the case of the CAVE described in Section 4.3). Ideally, the user should not have direct contact with the hardware that handles this type of system, except for the

interaction devices that may be used.

Cruz-Neira et al. (1993) and DeFanti et al. (2011) state that the process of engineering a system of this kind involves some of the following goals:

- The ability to display images in higher resolution and higher detail;
- The need to use physical devices to interact with the visualization. For instance, detecting the user's movements results in actions that have an effect on the visualization;
- The need to produce graphics, in stereoscopic 3D, that can be easily and realistically perceived by the human eye;
- The capability to use clusters for VR environment processing.

All these goals aim for a system capable of providing an experience where visualization should be as realistic as possible and where interaction should be as responsive as possible. Having screens surrounding the user, adding to the capability of providing stereoscopic 3D, allows the user to see objects around him with the realism that other systems cannot achieve. This is desirable for this work.

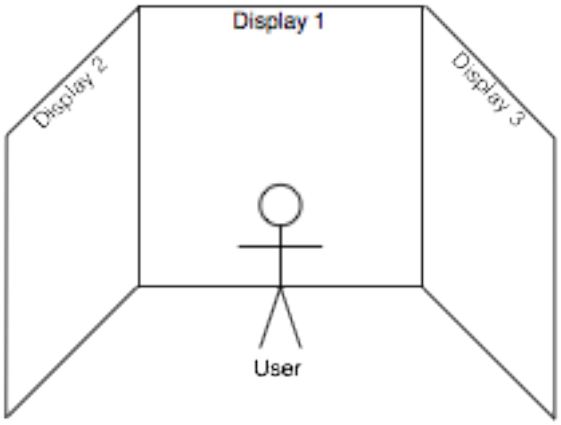


Figure 2.1: User inside a CAVE

2.3 3D application servers

3D application servers are used for creating virtual worlds, to which many users are able to connect, and in which they can interact with each other. There are three aspects that must be handled by a 3D application server: avatars, regions and the centralized grid. An avatar is a virtual object that represents a user, in other words,

it may be seen as a virtual personification of the user. Usually, avatars are shown in the form of a 3D model or a 2D picture. A region is the virtual physical place where avatars move and interact. It may contain land, with buildings, water and mountains, customized by its own users. A centralized grid contains information about the regions and their location in the virtual word. It has information about things that transcend a region (for instance, a user's inventory is part of a grid, so that it can be accessed in any region). [OpenSimulator (2011), Linden Research, Inc. (2011a)]

To illustrate the relevance of 3D application servers, we describe some practical applications of virtual worlds, created by different people (note that they all apply to Second Life, but they could be created in any virtual world with the minimum required features):

- Lucia et al. (2009) have developed a virtual campus in Second Life where students may interact with each other in a number of ways. The idea was based on the e-learning platforms currently used by many universities. Although current e-learning systems provide a way to share documents and post comments about several academic subjects, the authors thought they did not offer enough interaction between their users. They wanted to create a platform where users could interact in a more realistic way, so they designed several virtual spaces in Second Life to simulate various areas of the campus. Furthermore, they have equipped their virtual world with tools that allow students to access contents, such as papers, videos and books. The authors developed a Second Life component and a Moodle plug-in in order to achieve this. The results of this project indicated that the students found that using Second Life as a virtual campus was more appealing than using traditional platforms on the Internet. In their opinion, it made communication between students and teachers much easier and more personal. It also allowed them to get to know their classmates better.

- Doherty et al. (2006), members of the Explonatorium Science Museum, have created an interactive science museum in Second Life. Initially, a replica of an amphitheater, in Turkey, was created in Second Life, equipped with a screen to broadcast the solar eclipse that hit it in 2006. The general opinion of the viewers indicated that the experience was almost like feeling the eclipse directly. This was the authors' inspiration to create the virtual museum called 'Splo, because they saw the science teaching possibilities offered by this kind of environment. Hence, the amphitheater was expanded with a planetarium

that helps viewers understand eclipses. In this museum, an avatar can walk or fly around the Earth or to the Moon, giving the sensation of distance, for instance. Other examples involve reproducing illusions that affect the user in real life. The Depth Spinner is one example. Once a user sees it for a few seconds in the virtual world and then stares at an object in the real world, that object will appear to grow.

One of the advantages of creating a museum in Second Life is that it offers the possibility to represent macro and micro environments in a scale that a user is able to perceive, like planets and cells, respectively. The user can freely move around them and change their size according to his desire. That is one of the purposes of their museum: to show scientific occurrences that are not easily represented with traditional models on a normal scale. The more detail given to an exhibit, the more a person is able to understand it and describe it.

- There is also a case study involving an e-health service in Second Life (for more information see [Gorini et al. (2008)]).

To sum up, 3D application servers are great tools that can be used for a variety of purposes, applicable to real life, such as: modeling and/or prototyping buildings and scenarios that would otherwise be difficult to produce in the real world; eliminating social barriers that often occur in our daily life, e.g., educational and medical environments; creating communities designed to help people with specific problems such as illness; and much more.

In the remaining of this section, an analysis is made of two existing 3D application servers: Second Life and OpenSimulator. This analysis will focus on a number of aspects that are relevant to the current work. In particular, regarding the limitations imposed by the servers on the creation of virtual worlds. There needs to be freedom in the occupation of regions and in the creation of objects, so that we are able to create the simulations we desire. The server must be capable of being executed in a local network, because we want the simulations to be responsive to the user (i.e. no network latency issues). An open source solution may also be necessary, to deal with coding issues that might eventually come up.

2.3.1 Second Life

Second Life was created by Philip Rosedale, the founder of Linden Lab, which is the company that has been responsible for the development of Second Life over the past years, and is still working on it. Second Life had its debut in 2003. At that

time, the service was supported by sixteen servers but, three years later, the results showed a remarkable growth. Second Life had thousands of servers and about a million registered users [Rymaszewski et al. (2007)].

Second Life offers the user the ability to shape the virtual world based on his own ideas. Users can create a vast number of objects such as cars, buildings, furniture, clothes and toys [Rymaszewski et al. (2007)]. However, this ability comes with a price. In order to use land, the user has to buy it first. Likewise, if the user desires to use some very specific objects, he has to buy them first as well.

The trading currency in Second Life is called Linden dollars. This kind of currency can only be obtained by trading it with real money, or through accomplishments in the virtual world [Linden Research, Inc. (2011d)]. Obviously, this limits the creation of a virtual world, because time and money need to be spent in order to do it.

Second Life is a service that acts on the Internet only. There is no way to create a genuine Second Life server in a local network, because Linden Lab does not provide the software to do that. The user needs to access Second Life's public servers in order to obtain access to a region. The software in the server-side of Second Life is strictly proprietary, so it cannot be used or modified by the common user.

2.3.2 OpenSimulator

The success of Second Life made people realize that the potential of such environments should be exploited by an open source solution. Many of the attempts at creating open source solutions for 3D virtual environments failed, because they envisioned the coding of both client and server sides [OpenSimulator (2011)]. While the creation of a 3D application server was feasible, the creation of a client involved many difficulties because it required the programming of a graphics engine to render the virtual world, and the creation of a simple enough Graphics User Interface (GUI) to encompass all the functionalities that users need for manipulation, e.g., creating and using objects, building lands and customizing user preferences.

The issue of creating a client for an open source 3D application server persisted until January 2007, when Linden Lab decided to release Second Life Viewer with an open source license. This was when the project of OpenSimulator, started by Darren Guard, was born [OpenSimulator (2011)].

OpenSimulator can be used to create virtual environments to simulate scenarios of the real world. Since it is available for download on its website, it can be executed in a local server, allowing the user to customize the virtual world as he sees fit. OpenSimulator comes with the possibility to create lands and objects, provided that the user has the required models for them. The virtual world can be manipulated

	Second Life	OpenSimulator
Terrain Occupation	Limited	Unlimited
Object Creation	Limited	Unlimited
Local Network Support	No	Yes
License Type	Proprietary	BSD

Table 2.1: 3D application servers comparison

using the client from Second Life as well.

OpenSimulator is available in an open-source fashion, under a BSD license. It can be altered and encapsulated to work within another application. It comes with an Application Programming Interface (API) that provides a number of libraries that can be used to interact with the server. The more relevant ones are:

- **OpenSim**, that contains calls responsible for OpenSimulator's region server and console;
- **OpenSim.ApplicationPlugins**, that allows the integration of modular pieces of code to add new functionality to the region server;
- **OpenSim.Framework.Servers**, for the OpenSimulator server and http servers;
- **OpenSim.Grid**, which includes methods for grid manipulation;
- **OpenSim.Region**, which is related with anything that has to do with region manipulation.

There are other libraries that may be used but are not specified here. For more information see [OpenSimulator (2011)].

2.3.3 Comparison

Some of the differences between Second Life and OpenSimulator are represented in Table 2.1 [Freire et al. (2010)].

As explained above, terrain occupation and object creation are limited in Second Life. A user has to first buy land so that he can customize it however he wants. Whilst Second Life may have more projection, making created lands more visible, it certainly is a disadvantage to have to pay for land in order to construct whatever we want in it, especially if we only want to use it on a local computer or network. Likewise, some objects may require additional payment, when needed to be used. OpenSimulator, as opposed to Second Life, does not have any limitations when it

comes to occupying terrains and creating objects. The user is free to create his own customized land (by importing a terrain image file, for instance) and then place the objects he desires in it, with no monetary costs.

OpenSimulator will come in handy because the server can be ran locally. Not only does this remove the dependency over Linden Lab's services, but it is also particularly useful if we want to access a virtual world with a client on a single computer or inside a local network.

Another difference between Second Life and OpenSimulator lies in their software licensing. Second Life is licensed as proprietary software while OpenSimulator has a BSD license. The question here is how licensing will limit our capability to deal with issues that might come up during development. If a user chooses Second Life, he will not have to worry about having to fix things because Linden Lab will do it for him for an additional fee. This means, however, that the user will be dependent on Linden Lab. Maintenance may not always be instantaneous. Another problem is that it is not possible to alter the code of Second Life to satisfy our needs. OpenSimulator is open source and free to use. This means that, while the level of support might not be as high as for Second Life, users are able to change the source code and suit a particular system to their needs. Considering that we want to rely on a CAVE for visualization and its devices for interaction, alterations may be necessary, so an open source application is advisable.

Therefore, OpenSimulator is the best choice for this project. Having to pay for land and objects would definitely hinder the efforts to prototype ubiquitous environments. The APEX framework is meant to be used inside a CAVE, so its usage should not go beyond a local network. Using an open source server is the best choice for development, because modifications to the code can be made to support the APEX framework's needs.

2.4 Second Life clients

From the moment Linden Lab declared its client, Second Life Viewer, as open source, a number of adapted clients for Second Life began to emerge, each one having the characteristics desired by the authors who created it. The result is a variety of clients, available to the public, that can be used to connect to Second Life and OpenSimulator servers, and that can be customized by anyone, provided that the open source license of Linden Lab, for Second Life Viewer, is not breached.

For this work, we are looking for the following characteristics in a client:

- **Stereoscopic 3D support:** it is typical of a VR environment to surround a

user with floating objects that almost feel like they can be touched. Hence, the capability to produce stereoscopic 3D graphics is a needed feature, if we want to add realism to the simulations;

- **Multiple display support:** the ability to support more than one display is required, in order to support the different displays in the CAVE;

- **Client version:** the client version must be preferably recent, for better compatibility with other recent clients and to include all the features provided by a server;

- **License type:** the license must be open source if we want to make modifications to the code of the client;

- **Capability to import mesh objects:** this feature goes beyond the scope of this work. However, considering that the APEX framework requires realistic models, this is also taken into consideration.

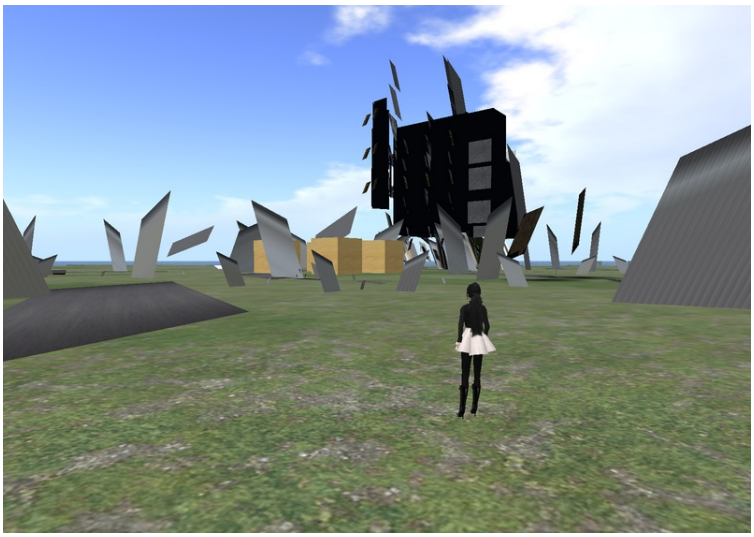
In this section, an analysis is made of the following clients: Mesh Viewer, Dale's SL Viewer and CaveSL. They are all based on the source code of Linden Lab's Second Life Viewer, even though they were created from different versions. A comparison is made between them to see if they support the required features. The possibility to integrate the source code of several of them is also considered. All the viewers are compatible with any kind of server running either Second Life or OpenSimulator.

2.4.1 Mesh Viewer

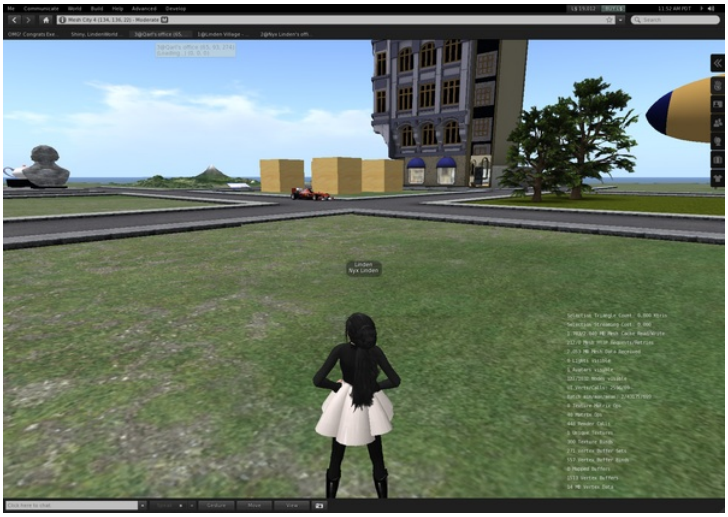
With Mesh Viewer, the user is capable of importing models, designed with mesh creating tools, into Second Life [Linden Research, Inc. (2011b)]. These models are stored in COLLADA files and can virtually represent anything that we would otherwise see in the real world, such as: prototypes for buildings, house furniture, daily used objects like hair brushes, pencils and laptops, automobiles, etc. Importing meshes represents a major advantage, by granting free access to a wide range of developed objects (e.g. 3D Google Warehouse). This adds the ability to construct much more complex virtual worlds. The more realistic the prototypes are, the more accurate is the representation of a ubiquitous environment, contributing to the immersion of the user (one of the objectives of the APEX framework is to make the prototyping of ubiquitous environments closer to the real world, using virtual environments only). According to Linden Research, Inc. (2011b), some of

the supported mesh creating tools are: 3DS Max, Blender, Maya, Sketchup and ZBrush.

Once the model is created and saved in the desired format, it can be imported into Second Life using Mesh Viewer's import functionality. The model is then processed by the server so that it can be represented in the virtual environment and accessed in the future. Note that only Mesh Viewer is capable of interpreting mesh objects. If a user tries to access the server with a viewer that has no support for mesh, the objects will not be rendered correctly. For instance, if we import a model with mesh objects, Fig. 2.2 (a) would be a representation of how the object is rendered without Mesh Viewer, while Fig. 2.2 (b) would be the representation using Mesh Viewer.



(a) Without using Mesh Viewer



(b) Using Mesh Viewer

Figure 2.2: Rendering of mesh objects (from [Linden Research, Inc. (2011c)])

However, Mesh Viewer does not come with any kind of features that would allow a representation of the world in multiple displays or in stereoscopic 3D. So, this client would have to be merged with another to actually fulfill the objectives for this work. In Section 2.4.4, we explain why this is not easily feasible.

2.4.2 Dale's SL Viewer

One of the aspects required for this project is immersion. An effective way of doing that is to provide the user with stereoscopic 3D visualization. There are no Second Life clients capable of doing that except for Dale's SL Viewer. This client makes use of the source code of an older version of Second Life Viewer, but it is compatible with OpenSimulator and it comes with a set of stereoscopic modes that can be used. According to Glass (2011), these modes are:

- **Anaglyph Stereo:** the sensation of depth is achieved through the use of red/cyan glasses. By separating the image with two different colors for both eyes, the user can see objects "pop" out of the screen when wearing the appropriate glasses. While this mode is highly compatible with most displays, it lacks the quality of other stereoscopic modes. This lack of quality is caused by color distortions that may confuse the human brain;

- **Passive Stereo:** to obtain passive, polarized stereo, the use of two projectors with polarized filters is required. Both projectors are used for displaying image to the same screen, each one for a different eye. This mode provides the most quality for the user, because it does not rely on color to provide image depth. However, passive stereo implicates the use of two projectors or an Head-Mounted Display (HMD), which means that it is not compatible with LCD/CRT monitors or any other displays of the same type.

  Both images must be evenly separated so that each eye can process the adequate one to produce the sensation of depth. This is done by the polarized filters that can filter the light meant to be seen by each eye. For example, projector one has a polarized filter that sends light for the left eye of the user, and projector two has a polarized filter that sends light for the right eye of the user. Since the images overlap, and the polarized glasses let the user only see the light that is meant to be seen for each eye, stereoscopic 3D is created.

  This mode also requires the use of compatible hardware that allows the enabling of stereo in replicated mode (i.e. both projectors have to show the same image at the same time). Usually, this is supported by the appropriate graphics drivers for a certain graphics card;

- **Active Stereo:** with this mode, the user can view image depth with the use of shutter glasses. Unlike the passive stereo mode, active stereo can be produced using a single monitor. This is possible by separating half of the frames produced for each eye [Gateau (2009)], which means that a monitor with a high refresh rate is required if we want to have quality. For instance, if the monitor's refresh rate is 60Hz, this means that thirty frames will be processed for the right eye and the other thirty will be processed for the left eye. This would be like having a monitor with a refresh rate of 30Hz, possibly causing a lot of flicker for the human eye. There are monitors that support refresh rates up to 120Hz. Using them would result in sixty frames processed for the human eye, which is the minimum desired to produce image without flickering (even though some people could use an even higher refresh rate). Although this mode allows the use of monitors, it still requires specific hardware for its functioning. A graphics card with a 3-pin din connector is needed as well as supporting shutter glasses.

The benefit in using Dale's SL Viewer is that the issue of stereoscopic 3D is completely addressed by it. It provides support to different stereo modes, with at least one that is compatible with any system: anaglyph stereo. However, this solution is not oriented for multiple display systems and since it uses an older version of Second Life Viewer's source code, it might prove difficult to use it in harmony with other, more recent, solutions.

2.4.3 CaveSL

CaveSL is a solution created by Kip Haynes at the University of Southern California's Institute for Creative Technology [Haynes (2010)]. This project, as the name suggests, is an adaptation of the traditional Second Life Viewer for clustered systems such as the CAVE and other multi-projector systems. CaveSL uses Second Life Viewer's source code (see Project Snowstorm (2011) for more information about it) as a means to provide access to Second Life's basic functionalities whilst having the necessary modifications to work with several displays instead of just one.

As stated by Haynes (2010), CaveSL uses several logins on a specific server in order to display image through an unlimited number of machines. A client-server relationship is established where a leader (main viewer) controls all its followers (other viewers). This is how camera synchronization is achieved. Whenever the leader viewer issues a command to move forward, the follower viewers also move accordingly. The following aspects of each camera are constantly adjusted to al-

ways match the leader's commands and each viewer's initial configuration: camera rotation and Field of View (FOV). In other words, the user input that the main viewer receives is also received by the other viewers. In Fig. 2.3, data and display synchronization between all viewers is represented.

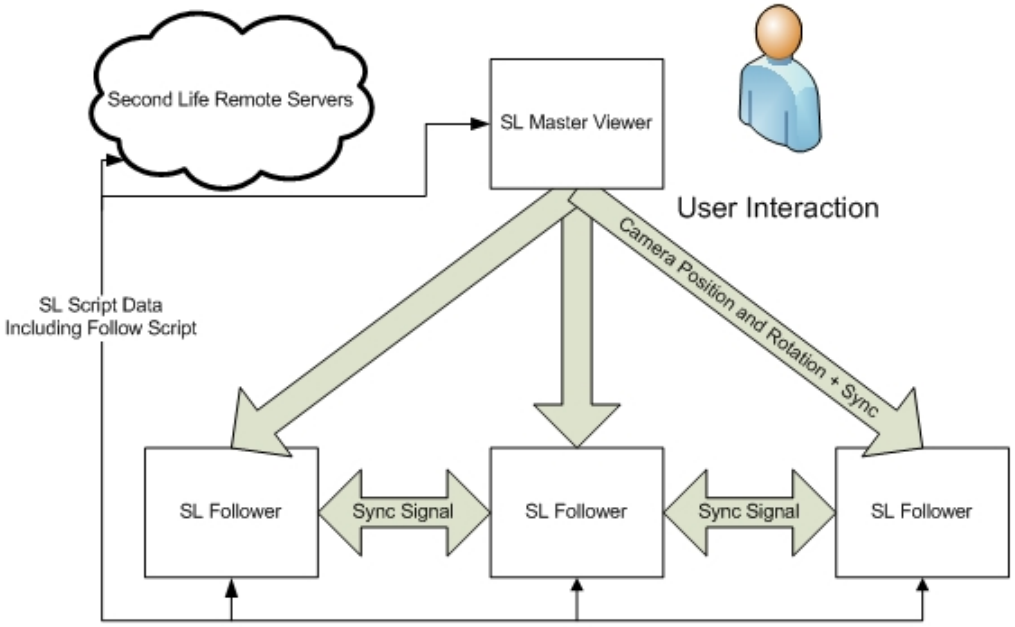


Figure 2.3: Data and display synchronization in CaveSL (from [Haynes (2010)])

Each follower viewer is executed on a separate machine where a local configuration file (*cave.cfg*) is read to provide camera rotation and FOV information. With these two options, it is possible, for example, to put three cameras pointing around an avatar: one forward, one on the left and one on the right. With this, we can better explore the rich worlds that 3D application servers provide by maximizing their immersion capabilities. Since it is possible to use an unlimited number of machines, the capability to immerse a user with this technology is quite flexible (with costs, however). Instead of using three displays, it is possible to use four displays, for instance, where the extra display could show the backside of an avatar.

The synchronization signals are handled by a Message Passing Interface (MPI) program called MPICH2. It allows the parallel computing of the viewers by sharing the desired data across machines: in this case the master viewer commands such as camera rotation, interactions, etc.

Some of the benefits of CaveSL are:

- The capability to provide support for CAVEs and multi projector systems;
- The fact that most implementation difficulties are addressed by this solution (it synchronizes camera position, rotation and FOV of all viewers based on the leader's commands and it does this without limitation on the number of machines desired);
- The fact that it is easy to extend or reduce the number of displays used.

Some of the limitations in using CaveSL are:

- CaveSL is highly oriented to work with several machines as opposed to one only. This could be solved by using several virtual machines or by changing its source code (e.g. allow the support of camera rotation based on a user's login, process number, etc.);
- Since CaveSL is specifically designed for multi projector systems, it only supports horizontal rotation of the follower cameras [Haynes (2010)];
- The user cannot interact directly with the follower cameras, only with the leader camera;
- CaveSL does not support stereoscopic 3D, unlike Dale's SL Viewer.

2.4.4 Comparison

Table 2.2 shows a comparison between the different clients, considering the criteria initially defined: stereoscopic 3D support, multiple display support, mesh object importation, client version and license type. The original Second Life Viewer is included as a base reference. However, it can be seen it does not provide most of the features we are interested in.

Only one of the clients offers the capability to produce graphics in stereoscopic 3D. Dale's SL Viewer supports three kinds of stereo: anaglyph stereo, passive stereo and active stereo. This is an advantage compared to the other clients, because they do not provide any support for this feature.

The problem with Dale's SL Viewer is that in order to transform graphics to active or passive stereoscopic 3D, specific hardware is needed. This problem does not apply to anaglyph stereo, making it highly compatible with many systems. In Section 3.5 we mention a technology, created by NVIDIA Corporation, that allows the use of stereoscopic 3D for a variety of applications, which may help in this scenario.

	Mesh Viewer	Dale's SL Viewer	CaveSL	SL Viewer
Stereoscopic 3D Support		X		
Multiple Display Support	X[2]	X[2]	X	X[2]
Mesh Object Importation	X			
Client Version	3.1.0	1.18.4	2.0.0	3.1.0
License Type	GPLv2[3]	GPLv2	GPLv2	GPLv2

Table 2.2: Second Life clients comparison

When it comes to multiple display capability, CaveSL provides the best support by allowing the use of an undefined number of cameras around the avatar. The user can promptly define the angle and FOV of each camera, which is perfect for a variety of CAVE setups. Mesh Viewer and Dale's SL Viewer have limited support for multiple displays. By limited support, we mean that it cannot use anything beyond the operating system's or the graphics card's capabilities (i.e. maximizing a window in extended mode). Again, there are some technologies, created by graphics cards manufacturers, that permit the use of more than two displays at the same time. For instance, NVIDIA 3D Vision Surround supports three displays at the same time as well as stereoscopic 3D. ATI Eyefinity supports up to six displays at the same time, but it has no stereoscopic 3D support. On one hand, using more than three displays might be good only when using CaveSL, if we want them to surround the participants. On the other hand, if we plan to use no more than three displays in a flat panel, the other clients (Mesh Viewer and Dale's SL Viewer) should suffice.

Mesh Viewer is the client capable of importing objects. This is actually a very important feature, because we can import the prototypes of buildings into Second Life, making the experience much more realistic. If this feature could be somehow integrated with other clients, it would be a major advancement. Having prototypes created by architects or designers along with CaveSL's and/or Dale's SL Viewer's capabilities would definitely result in a more satisfying experience when inside the CAVE.

Every client's source code is distributed in an open source fashion, according to Linden Lab's specifications. However, Mesh Viewer has the following limitation: the code that actually supports mesh objects is not open sourced. So, whoever wants to build Mesh Viewer with that feature needs to implement it using an open source library (see [Nyx Linden (2010)] for more information about it). As for Dale's SL

[2]Only when using extended displays for a desktop, with a maximized window or in full-screen mode.

[3]Source code comes with some limitations due to licensing issues. See `http://community.secondlife.com/t5/Mesh/Mesh-viewer-source-code/td-p/396912` for more information.

Viewer, the source code is old compared to the other viewers. Integrating that source code into Mesh Viewer or CaveSL might prove difficult because the code for graphics rendering has become rather obsolete [Linden Research, Inc. (2010)].

The ideal solution here would be to combine the features of the analyzed clients. However, considering the limitations each client has on its source code, merging them would prove to be an arduous task. Therefore, it is best to maintain the focus of this project in exploring the solutions given by Dale's SL Viewer and CaveSL.

2.5 Virtual reality frameworks

In this section, some of the most popular VR frameworks (VR Juggler, Avango, CAVELib and Mixed Reality (MR) Toolkit) are described and compared. We desire a framework that is capable of providing features such as stereoscopic 3D and multiple display support when accessing to a 3D application server. However, it is also important to find out whether these frameworks can be integrated in conjunction with a Second Life client, to avoid the issue of creating a new one from scratch. VR frameworks also provide some other features that typical clients do not support.

Before proceeding to the analysis of the VR frameworks, it is important to mention that some of the already existing ones (Alice and WorldToolKit) were not taken into consideration due to the following reasons:

- Alice [Alice (2010)] is highly oriented towards users initiating themselves in object-oriented programming and, because of that, simple applications can be created very fast but not complex applications;
- WorldToolKit (WTK) appears to have become obsolete. The last reference manual encountered dates back to 1999 [Engineering Animation, Inc. (1999)] and currently there is no available website for it.

So, in resemblance to some of the characteristics analyzed by Bierbaum (2000), the main goal is to find the ideal VR framework that balances all these features:

- **Flexibility**. One of the main goals is to make the VR framework communicate with an application that is completely independent of it. Flexibility is a required characteristic that must allow this kind of integration. By having it, the VR framework is compatible with multiple graphics engines and altering the code for graphics rendering cannot compromise other factors, such as the interaction with external devices. However, in the best case, this feature should not come in detriment of other important characteristics;

- **Multi-display and stereoscopic 3D support**, through the use of projectors and monitors;
- **External device compatibility**, because a good VR framework must have some way of communicating with mobile devices and movement sensors at the very least, so that it may interact with users when needed;
- **Open source license**, to reduce costs and to allow structural changes that may be needed for the task at hand.

Flexibility is an especially important feature that we want on a VR framework, because of the possibility of creating a new client for a 3D application server. However, the other features are also important. Multi-display and stereoscopic 3D support, as well as external device compatibility are also a requirement if we want to realistically simulate ubiquitous environments.

2.5.1 VR Juggler

In VR Juggler [Iowa State University (2007)], all applications are written as objects that are handled by a specifically designed kernel. The basis of VR Juggler is that it must be a dynamic system capable of being flexible and extensible, allowing developers to add functionality and to change services without having to reinvent the wheel. The kernel acts like an operating system, with the difference that it is designed solely for running VR applications. With it, VR Juggler implements a number of services that are essential for the development of VR applications. According to Bierbaum (2000), these services provide support to input devices, displays and may also have configuration information.

In order to communicate with different types of applications, there are external managers that are capable of handling them. These external managers can be extended to make the kernel compatible with different kinds of applications. A developer may use them to add application support for a different graphics API, for instance.

The VR Juggler kernel also works as a mediator [Bierbaum (2000)] by discriminating and encapsulating the interaction between components. For instance, the kernel can act as a mediator when an input device needs to interact with a graphics rendering engine, which makes sense since the kernel has the knowledge to do so (it contains information about the devices and the external managers). Not only this makes devices independent of graphics libraries (one does not need to have information on another, so they can be altered without compromising any services)

but it also allows the kernel to indicate the timing of calls made to the managers [Bierbaum (2000)]. This is good for performance because important calls can be prioritized, reducing their waiting time for processing.

Basically, VR Juggler's flexibility allows its applications to have longevity, because most of the refinements made to any components hardly have any repercussions to the rest of the system, not being necessary to rewrite code when they occur. It also permits the addition of new functionality to the framework (this is particularly good for this work, since it is intended to integrate a 3D application server from scratch). The kernel already provides means of interacting with external devices and takes into account some performance concerns. VR Juggler has multiple display support through the use of projectors (e.g CAVE), and it is also possible to set it in stereoscopic 3D mode using its configuration files.

In Section 3.2.1 of Chapter 3, the VR Juggler application architecture is discussed and explored with more detail.

2.5.2 Avango

One of the distinguishing features of Avango is its capability for developing distributed applications for VR environments. The distribution of tasks to several machines not only makes the execution of algorithms more efficient but it also allows the projection of image to several displays without overloading a single computer.

Unlike VR Juggler, Avango does not have a kernel to handle multiple graphics libraries. Instead, it is built on top of a scene graph framework, known as OSG, to render the virtual world. Each node object of the graph structure represents an aspect of the environment which may be a geometrical object, a user command or an object that produces three-dimensional sound. Avango extends the scene graph concept with field connections [Fraunhofer-Gesellschaft (2010)]. These fields represent object state information [Tramberend (1999)] and may be linked with the fields of another object, giving developers many ways of defining complex behaviors through networks of objects (e.g. touching a light switch would result in turning an electric light on or off).

Besides node objects, Avango also has sensor objects [Tramberend (1999)]. Sensor objects are capable of transferring information between Avango and external devices, which means that the fields of these objects have data collected through device interaction. The device data obtained from sensor objects can be used to interact with the virtual world using the network of referenced objects through fields. Indeed, the fields of sensor objects may also be linked with the fields of node objects that are part of the scene graph, giving developers the possibility of creating behaviors

based on the devices that they see fit. The usage of fields in the objects of Avango allows the developer to create applications with complex behaviors and with the capability to interact with external devices.

Avango developers were able to map its features to a scripting language using Python [Fraunhofer-Gesellschaft (2010)]. This allows users to create and manipulate virtual worlds in a very simple and fast manner, with immersive 3D, being another one of the main advantages of using this framework.

Despite all these features, Avango has the flaw of being highly dependent on the OSG library for graphics, which means that this framework does not have support for other graphics libraries. Compatibility with existing Second Life clients (or any other client that does not use OSG) is not possible.

2.5.3 CAVELib

CAVELib is a VR framework that works by providing an API with a set of callback functions that are available for application development. There are three types of callback functions in CAVELib: display, frame update and initialization [VRCO, Inc. (2004)]. The display callback functions are used for rendering and this can be done with multiple kinds of graphics libraries [Mechdyne (2010)]. Any kind of drawing is rendered by the chosen graphics library. The frame update callback functions are called before the rendering of each frame. They may be used for behavior control by retrieving data from external devices. This facilitates user interaction with the virtual world, since the commands are being processed by the system before rendering the frames, making it highly responsive. Initialization callback functions are called on the beginning of execution of the application. Similarly to some graphics libraries, such as OpenGL, this happens before looping the callback functions responsible for data retrieval/update and rendering (frame update and display callbacks), and it is meant for initializing any configurations that the application might need before starting the rendering cycle.

According to Scientific Computing World (2008), CAVELib is mainly focused on providing image through projection screens, specifically for a CAVE, but it may also be used for desktop applications, HMD systems and PowerWalls.

This framework is commercially available for research and many other fields of interest [Scientific Computing World (2008)], which means that it is not free of charge. Despite that, it is a good solution mostly due to its simplified programming, platform independence, and compatibility with several tracking devices. It provides support for stereoscopic 3D and multiple graphics channels (e.g. multiple projectors) [Mechdyne (2010)].

	VR Juggler	Avango	CAVELib	MR Toolkit
Flexibility	X		X	
Multiple display support	X	X	X	
Stereoscopic 3D support	X	X	X	
External device compatibility	X	X	X	
Open Source	X	X		X

Table 2.3: VR frameworks comparison

2.5.4 Mixed Reality Toolkit

The concept of MR is to combine elements of the real world with the virtual world. This can be done in the following ways: Augmented Virtuality (AV) where objects from the real world are incorporated into virtual worlds, or Augmented Reality (AR), where computer graphics are used to complement images taken from the real world [Freeman (2005)]. This can be achieved by forming shapes over live video, for example around tables, boxes, etc. If a user decides to put virtual objects on a real table and decides to physically move it, those virtual objects will move accordingly, provided that the shapes were correctly drawn. By describing aspects of the real world, the user can place virtual objects according to the ones in the description and have the program detect collisions or create effects [Freeman (2005)].

So, this framework provides a slightly different concept than the others, since MR intertwines both realities: real and virtual. This can be considered as an advantage, because it enables users to interact with objects from the real world and see the results obtained in the virtual world, which is a good aspect for immersion. However, since the MR Toolkit depends on live video and currently there is no support for multiple cameras and camera tracking [Freeman (2004)], it can hardly be called an immersive environment[4]. This also means that the virtual world is attached to the real world, so it is not possible to create simulated scenarios for ubiquitous environments.

2.5.5 Comparison

Table 2.3 shows a comparison between the described frameworks. The most important characteristics are compared: flexibility, external device compatibility, multiple display and stereoscopic 3D support, and license type (open sourced or proprietary).

- In terms of flexibility, VR Juggler satisfies the specified requirements thanks to

[4]One of the requirements of this project is to add the capability for the user to see virtual objects around him, but MR Toolkit does not provide that kind of support.

its kernel-based architecture, allowing the use and implementation of multiple graphics libraries. Similarly, the CAVELib also supports multiple graphics libraries.

- As for multiple display and stereoscopic 3D support, VR Juggler, CAVELib and Avango can send this type of image transmission to the projectors of a CAVE.

- All of the frameworks, except for MR Toolkit, present external device compatibility, permitting users to interact with the virtual worlds created.

- Only the CAVELib is not classified as open source. The others are licensed under a GNU Lesser General Public License (LGPL).

Considering the advantages that VR Juggler presents on Table 2.3, compared to the others, and since it has most flexibility options and multi-language support due to its kernel capabilities, it is the best framework to serve as a client for the 3D application server. Not only it makes integration possible, but it also has good characteristics for providing an immersive experience for the user, and with the possibility to use external devices and sound systems.

There is a problem, however. Currently, there is no Second Life client capable of running alongside with VR Juggler. In order to use this solution, it is necessary to create a new Second Life client supporting VR Juggler. This can be done by accessing Linden Lab's source code for Second Life Viewer. The main issue here is how much time will it take to build a stable client to support all the required features. Section 3.2, in Chapter 3, shows how that integration can be made, and what kind of costs it brings.

2.6 Summary

Regarding 3D application servers, the best solution turns out to be using OpenSimulator, because it is free of charge, and it can be executed on a local server, as opposed to Second Life that requires an Internet connection, and it is open source. All of these features make OpenSimulator much less limiting than Second Life, in which real money needs to be paid to get land and objects, and since OpenSimulator is compatible with Second Life clients, the problem of creating a new one from scratch is potentially removed.

Therefore, it is important to know which clients may satisfy some of the needs of this project. Two clients stand out: Dale's SL Viewer and CaveSL. The first one

is capable of providing stereoscopic 3D in three diferent modes: anaglyph, active and passive stereo. The second one has the ability to be configured in clusters, with an undefined number of machines. It provides support for multiple displays. Furthermore, the user can define angles and fields of view according to the positions of the several displays in the cluster.

Based on the VR frameworks that are analyzed, we come to the conclusion that using VR Juggler is the best option among them. It provides enough flexibility to create a new client from it, possibly using existing code from Second Life Viewer. Doing this may add the support for stereoscopic 3D, multiple displays and interaction devices that we desire, while keeping the communication between the 3D application server.

With these solutions in mind, we proceed to Chapter 3, where they are explored into more depth to create a solution that satisfies this project's objectives.

Chapter 3

A Second Life client for a CAVE

3.1 Introduction

In this chapter, a set of solutions are explored for the integration of a Second Life client with a CAVE. These solutions try to bring the best of the CAVE capabilities by exploring features like stereoscopic 3D and multiple display support. The first solution involves the usage of VR Juggler's architecture to create a new client that supports those features; the second solution involves the usage of CaveSL; the third shows how Dale's SL Viewer works; and the fourth solution shows some of the advantages of using NVIDIA 3D Vision Surround technology. In the end, a ponderation of each solution is made, and whether it is beneficial to use combinations of several of them.

3.2 Solution I: Using VR Juggler

3.2.1 VR Juggler architecture

To better understand how VR Juggler can be used to develop a client for a 3D application server, one has to grasp the portion of its internal structure that supports adding new functionality. According to Iowa State University (2007), VR Juggler can execute application objects that "contain the visuals and the interactions that make up a virtual world". These objects are handled by the kernel and are created with the intent to override some of the interfaces shown in Fig. 3.1. By calling their implementation methods, the kernel can act as a scheduler passing control to different application objects[5].

[5]Each application object is executed in a different process and it may assume control through context switching issued by VR Juggler's kernel.

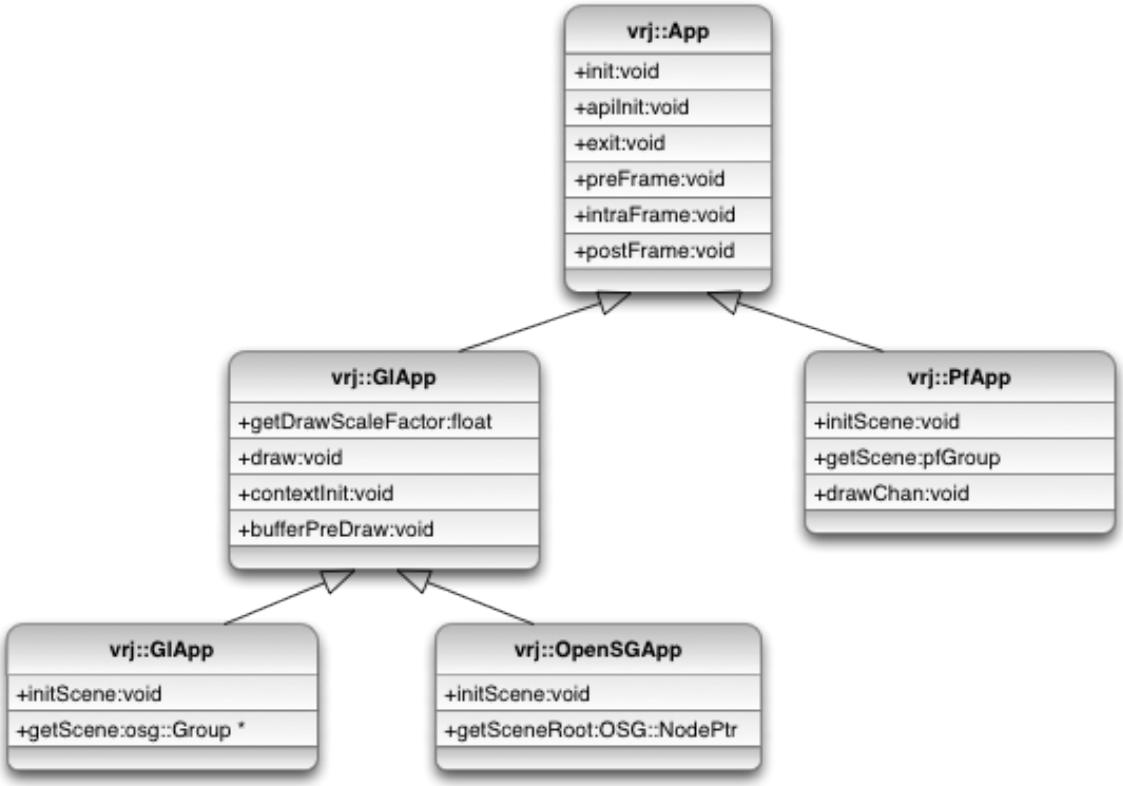


Figure 3.1: VR Juggler interfaces for application objects (adapted from [Iowa State University (2007)])

To implement an application in VR Juggler, developers need to code the methods of the appropriate interfaces. Although VR Juggler provides default implementations, in most cases they may not do anything relevant and require further customization in order to work as intended. If no customization is made, VR Juggler assumes that the default implementation is to be executed. In Fig. 3.1, the current interfaces provided by VR Juggler, for application objects, are displayed. "The kernel expects a base class for the interface *vrj::App* that specifies methods for initialization, shutdown and execution of the application" [Iowa State University (2007)]. Through its interface hierarchy, VR Juggler has Draw Manager interfaces for specific graphics libraries that are capable of rendering the virtual environment, such as *vrj::PfApp* and *vrj::GlApp* that are relative to OpenGL Performer and OpenGL respectively. These interfaces are used in *vrj::*App* classes that have the method implementations of the Draw Manager selected (e.g. OpenGL) and its parent interfaces (e.g. *vrj::App*).

The benefits of using application objects based on this structure are as follows [Iowa State University (2007)]:

- **Run-time reconfiguration** by the kernel allows the switching of applications (since they run on different processes) and the starting and assembling of new devices. The process of reconfiguration can then be sent to application objects so that they become aware of the changes;

- **Robustness and extensibility** because any modification issued to one object does not bring implications to other objects, provided that the interface itself remains unchanged. This results in product compatibility in the long term and gives room for adding or changing functionality;

- **Implementation changes** to the virtual platform do not alter the correct execution of applications. Such changes may bring performance improvements to VR Juggler's internal structure or increased device compatibility, for instance;

- **Support for multiple languages** is very important. VR Juggler kernel is written in standard C++ but since it treats applications as objects, it is possible to create programs in languages such as C#, VB.NET or Python. This is supported through VRJ.NET, that contains a set of C/C++ libraries and .NET assemblies with bindings to the Common Language Infrastructure (CLI), and through PyJuggler, a Python module [VR Juggler Development Team (2010)].

3.2.2 Creating a Second Life client using VR Juggler

Perhaps the simplest way to make VR Juggler act as a client for OpenSimulator is to take advantage of the existing source code for Second Life Viewer and use it to integrate VR Juggler's code. Since Second Life Viewer is an OpenGL application, we can use *vrj::GLApp* interface to create a class *vrj::*App* that overrides methods *init:void* and *draw:void* with calls to the respective code that initializes and renders the virtual environment created by OpenSimulator.

Fig. 3.2 shows the methods that can be overwritten in the *vrj::GLApp* interface and the relationship it has with VR Juggler's basic methods in the *vrj::App* interface.

3.2.3 Invoking Second Life application objects

To finalize, a short extract of code is presented to analyze how VR Juggler may handle calls to Second Life application objects in C++. The code is shown in Listing 3.1.

The algorithm starts by initializing the kernel of VR Juggler and an application object, named *app*, instantiated under a *SecondLifeApp* class that is defined in ac-

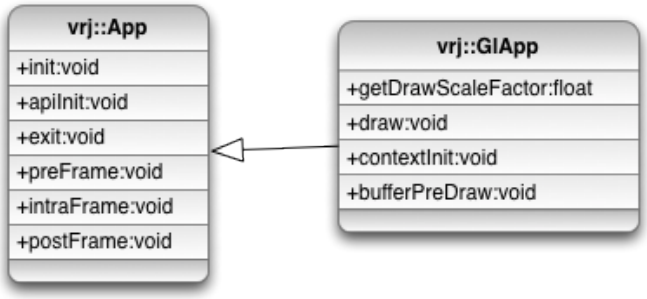


Figure 3.2: Using VR Juggler's interfaces for OpenGL applications (adapted from [Iowa State University (2007)])

```
1  int main(int argc, char *argv[])
2  {
3          vrj::Kernel* kernel = vrj::Kernel::instance(); // Get the kernel
4          SecondLifeApp* app = new SecondLifeApp();// Create the app object
5
6          kernel->loadConfigFile(...);      // Configure the kernel
7          kernel->start();                  // Start the kernel thread
8          kernel->setApplication(app);      // Give application to kernel
9          kernel->waitForKernelStop();      // Block until kernel stops
10
11         return 0;
12 }
```

Listing 3.1: Code for VR Juggler's main function (adapted from [Iowa State University (2007)

cordance with the *vrj::GLApp* interface, specifically designed to work with OpenGL applications. Before starting, VR Juggler's kernel loads any configuration files set by the developer to change any behavior he desires. Once the kernel starts its activities, the developer may pass the application object to it for execution. From this moment on, all activities are handled by the created object's programming and the structures defined in this section. The execution only stops when an exit command is issued to the application object (this is controlled by *kernel->waitForKernelStop() method*).

The major issue of this solution is that the class *SecondLifeApp* needs to be created with methods that override the *vrj::GLApp* interface. There are two possible options: either create a new OpenGL application or send calls to Second Life Viewer's source code. The first option would require the creation of an entirely new graphics engine to render the virtual world. Interaction with the server would have to be handled as well. The second option would eliminate the issues of the first option, because we would be using code of an existing client. However, this option comes with its costs as well. To begin with, the code would have to be put in the correct methods of the interface without any flaws. Then, the dependencies from

VR Juggler and Second Life Viewer would have to be addressed to build the new client. This would require the modification of the files responsible for building VR Juggler. While the second option might be feasible, extensive knowledge on the source code of VR Juggler and Second Life Viewer is needed. If we add the time spent in learning these technologies with the time spent in interpreting Second Life Viewer and VR Juggler's source code, it may turn out to be a very long task. In the context of the current project, it was decided that the risk was too high, and this solution was not further explored.

3.3 Solution II - Using CaveSL

In section 2.4.3, it is mentioned that CaveSL uses MPICH2 to provide parallel computing support to Second Life Viewer, which means that it can work in a CAVE or a similar system with multiple displays. For development purposes, one of the objectives of this project is to try this with a single machine, to analyze the possibility to run this solution on a triple-head setup. Because it is absolutely necessary to use MPICH2 for CaveSL, and its configuration files (such as *cave.cfg*) must be local to each machine, in order to support different camera rotations and fields of view, it is not possible to use it on one single physical machine. Thus, three different approaches must be considered:

- The first approach involves using one single machine with the help of virtual machines. For a triple-head desktop, we can use two additional virtual machines while taking advantage of the physical machine as well;
- The second approach involves the use of several physical machines, connected through a network;
- The third approach is to change CaveSL's source code so that it may be able to support at least three instances of CaveSL on one, and only one, machine. To do this, we have to find a way to alter the way CaveSL interprets its configuration file.

There are, of course, limitations to any of these approaches. The first requires a powerful machine with a lot of processing power and Random Access Memory (RAM), in order to support several virtual machines and several instances of CaveSL running simultaneously. The second, while it may not be required that each individual machine must be as powerful as in the first case, needs as many physical

```
1 pm  1
2 vm1  1
3 vm2  1
```

Listing 3.2: Content of machines.txt

machines as the number of displays we want to have. The third one does not demand the acquisition of more computers. However, it does imply some costs that development may not be able to deal with in the short term (for instance, issues may occur during development that may take a long time to solve).

There is also another question we must put for the third approach: how are we going to alter the code, to make CaveSL capable of running several processes on one single machine? Instead of using a local configuration file on each machine, this may be achievable if we adjust those configurations based on the login of a user (since CaveSL requires one login per process) or perhaps on the process number of each instance of CaveSL in execution. This suggests that instead of having a local configuration file on each machine we must have a generic configuration file with all the required information for all processes. For example, we can have three logins connected to a region in OpenSimulator and for each login instance we set the necessary angles and fields of view to satisfy the requirements of a CAVE in a U form, like the one represented by Fig. 2.1 in Section 2.2.

After considering the three possibilities, it was decided, for initial testing purposes, to opt for the first approach (in Section 4.4.1 of Chapter 4, we use the second approach for the CAVE). The diagram represented in Fig. 3.3 demonstrates in detail how it is possible to execute three processes of CaveSL with one machine that utilizes virtual machines. This method is also compatible with the second approach described before, provided that all computers are connected in an open network. In other words, it can be applied to a desktop and a CAVE running in a cluster.

In this case, we are using three viewers, one ran by the physical machine and the other two by virtual machine 1 and virtual machine 2. The physical machine is responsible for starting MPICH2. SMPD (a Simmetric Multi-Processing (SMP) client) is the application used by MPICH2 to communicate between different machines. It is used for data transfer and synchronization. With MPICH2 we define the path of the application we want to use (in this case, CaveSL), the text file containing the host names we will be using and any extra parameters necessary for the correct functioning of all processes. In this scenario, we have three processes running under the host names pm, vm1 and vm2. So, the machines text file (*machines.txt*) should be defined like Listing 3.2.

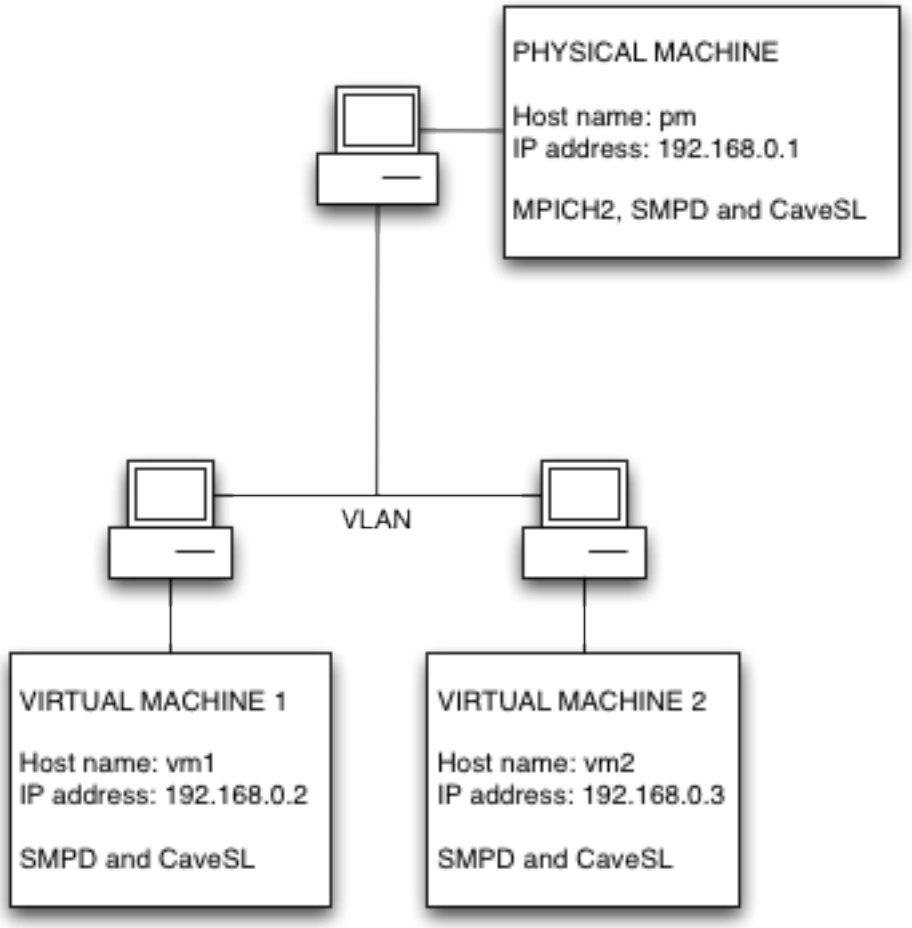


Figure 3.3: Communication between different machines with CaveSL

```
1 smpd -stop
2 smpd -d 0
```

Listing 3.3: Initializing SMPD in debug mode

Each row represents a host name that will be used for CaveSL. In this case, each machine executes one process of CaveSL. Note that the installation of CaveSL must be at the same location on every machine and the data model of the underlying operating systems must be coincidental.

To allow the local computer to initialize GUIs and windows, we must pass the command *-localroot* as a parameter for MPICH2 execution. This gives administrator privileges to the user but only on processes executed on the local host. For the other computers, we must instruct SMPD to allow the creation of GUIs and windows by issuing the commands shown by Listing 3.3.

```
1 mpiexec -localroot -n 3 -machinefile machines.txt secondlife-bin.exe --loginuri
    http://pm1:9000
```

Listing 3.4: Initializing MPICH2

The first command may be skipped if there is no SMPD service running. The second command initializes SMPD in debugging mode, allowing the creation of windows on those computers. This cannot be done without SMP running in debugging mode, because the main machine does not have the privileges to access the other computers as root.

Once all the computers are set up, we must issue the command, shown by Listing 3.4, in the master computer. The parameter *-n 3* designates the number of processes that will be executed (in this case, three). We can indicate the machines file by passing it on the *-machinefile* parameter. Last but not least, we indicate the executable that starts CaveSL and its parameters: *secondlife-bin.exe –loginuri http://pm1:9000*. In this case, we assume that host name pm1 is running a service for OpenSimulator on port 9000, though it can be any machine (even outside the local network).

By issuing this command to the operating system of the leader machine, all processes are executed on their respective machines. Taking this example into consideration, a setup with three displays is provided to the user. As Fig. 2.1 in Section 2.2 suggests, a user can be immersed with three displays around him (i.e. inside a CAVE). The camera for each display is adjusted so that it presents the correct image according to the user's position. For instance, in a three display scenario, one of the adjustments that must be considered is camera rotation. While the camera of display 1 must be pointed to the front, the cameras of display 2 and 3 must be pointed to the left and right of the avatar. These are the camera rotations that can be applied for each display (in the *cave.cfg* file), with values represented in radians:

- Display 2: FOV: 0.88139 rad, angle: 1.03727 rad;
- Display 1: FOV: 0.88139 rad, angle: 0 rad;
- Display 3: FOV: 0.88139 rad, angle: -1.03727 rad.

The final result are three viewers, shown by Fig. 3.4.

3.4 Solution III - Using Dale's SL Viewer

According to Section 2.4.2, Dale's SL Viewer supports three kinds of stereoscopic 3D: anaglyph, passive and active stereo. The user can change the type of 3D stereo

Figure 3.4: CaveSL with three clients running simultaneously

he wants to use by selecting it in the Preferences pane, under the Adv. Graphics tab, as shown in Fig. 3.5. The user can also define stereo separation and stereo focal distance (to better adjust 3D to the user's eyes, taking into account factors such as the distance separating the user from the monitor/projector and the size of the display.

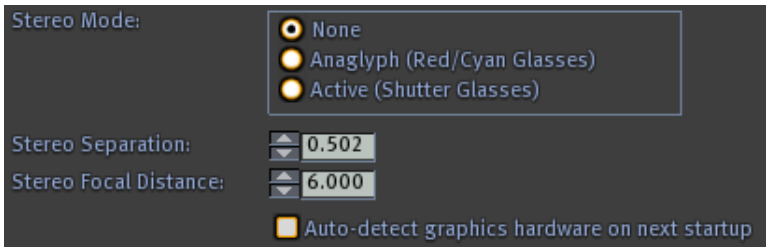


Figure 3.5: Selecting stereo mode in Dale's SL Viewer

By selecting anaglyph stereo and applying settings, the option is activated right away. There are no hardware limitations for this option, so the effects are reproduced on the screen and 3D images can be seen using red/cyan glasses. Fig. 3.6 shows an example of a virtual environment using Dale's SL Viewer in anaglyph stereo mode.

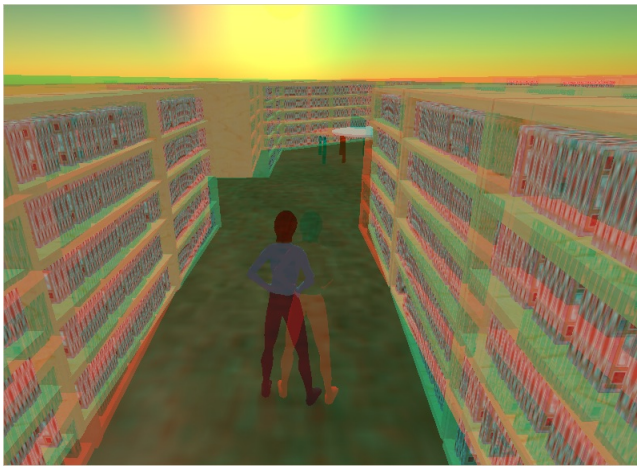


Figure 3.6: Dale's SL Viewer in anaglyph stereo mode

Activating the active stereo mode may be more problematic, because it requires specific hardware for it to function. Dale's SL Viewer website (see [Glass (2011)]) tells that, in order to have active stereo, with compatible shutter glasses, a 3-pin DIN connection on the back of the graphics card is needed (Fig. 3.7 illustrates how a 3-pin DIN connector looks like). A connector is then used to establish the communication between the graphics card and the monitor or projector used.

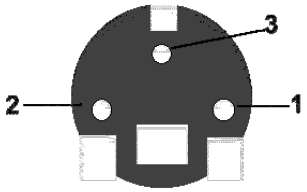


Figure 3.7: VESA miniDIN-3 connector (from [Bungert (1999)])

Since at least two projectors with polarized filters are required for passive stereo, where the image of one overlaps the other, it cannot be used in the CAVE. The projectors inside the CAVE are only meant to be used for setups with multiple displays.

Contrary to CaveSL, Dale's SL Viewer does not support using multiple machines at the same time, where each client has a different angle and field of view for its respective display. However, like any other client, it can run in window mode. Fig. 3.8 shows how the user can set the client running in that mode and how he can set the resolution to eventually fit two or more displays.

Figure 3.8: Setting Dale's SL Viewer in window mode

3.5 Solution IV - Using NVIDIA 3D Vision Surround

NVIDIA 3D Vision is a technology that provides stereoscopic 3D for applications with graphics rendering that would otherwise not be capable of providing it. It has grown in the market because of its simplicity of use and compatibility with digital photography and many games [NVIDIA Corporation (2009)].

According to NVIDIA Corporation (2011a), NVIDIA 3D Vision Surround provides the user with stereoscopic 3D while using up to three displays at the same time. This is a feature that could contribute a lot for user immersion. For this, three displays compatible with 3D Vision and at least two NVIDIA graphics cards connected in Scalable Link Interface (SLI) mode are required.

NVIDIA drivers come with solutions to help solve a number of issues related to using multiple screens. One issue is the need to adjust display bezels, caused by the edges of the screens (see Fig. 4.1 in Section 4.2 for an example with multiple screens). The edges contribute to the non-alignment of the image shown by two screens next to each other. Fig. 3.9 shows how the adjustment can be made using NVIDIA drivers. The road, as it is in this document, may seem that it is cut in half. But in reality, the road is aligned when the left side is displayed on the left screen (3) and the right side on the middle screen (2), due to the gap between the images when they are displayed on the screens. The same happens when the left side is displayed on the middle screen (2) and the right side on the right screen (1). The user can define how much he wants to split the road, to make it look as aligned as possible. In this example, we used 200 as the value to split the road between screens 3 and 2 and 190 to split the road between screens 2 and 1. The bigger the value, the more it is split. This feature will be particularly useful when using the triple-head setup, described in Section 4.2 of Chapter 4.

NVIDIA 3D Vision is a good technology for stereoscopic 3D, because of the already mentioned reasons, but it comes with its drawbacks:

- NVIDIA 3D Vision does not support Second Life (see [NVIDIA Corporation (2011b) for a list of supported games). The reason for this is that Second Life is an OpenGL application, which is not supported unless a NVIDIA Quadro graphics card, with professional OpenGL stereo quad buffer, is used [NVIDIA Corporation (2009),Gateau (2009)];

- Using multiple screens may not always be advantageous, especially if we try to put them in different angles. The image produced is meant to be seen as if it was one flat panel;

- Specific hardware needs to be used to support NVIDIA 3D Vision (it usually comes with a label saying it is compatible with this technology).

Despite these drawbacks, it is always possible to have multiple displays working with this technology, even if producing stereoscopic 3D cannot be achieved.

The OpenGL related limitation of NVIDIA 3D Vision is due to the fact that it only supports DirectX applications when using the NVIDIA GeForce graphics cards.

Figure 3.9: Adjusting display bezels with NVIDIA drivers

While, this may not be a problem when using the solution in the CAVE, it may be a problem when using the triple-head setup. Second Life is an OpenGL application, meaning that it cannot use NVIDIA 3D Vision features, under normal circumstances. There are a few solutions on the Internet that perform the conversion of OpenGL graphics to DirectX graphics - the so called OpenGL to DirectX wrappers.

Two applications that support this functionality were found: GLDirect and TitaniumGL. Each of them contains a dynamic link library called *opengl32.dll* that can be copied to Second Life Viewer's folder. By doing this, the user is altering the OpenGL library used by the application. In this case, the OpenGL library provided by the *opengl32.dll* file has the necessary DirectX API calls for conversion.

The obtained result, by using TitaniumGL to convert Dale's SL Viewer graphics renderer to DirectX, was clearly unsatisfactory and cannot be used as a solution for the stereoscopic 3D problem. This is because TitaniumGL only supports OpenGL 1.x applications [LegendgrafiX (2011)] (Second Life Viewer uses a later version of OpenGL). GLDirect suffers from the same problem and testing resulted in the client not even running. Fig. 3.10 shows the results obtained using TitaniumGL.

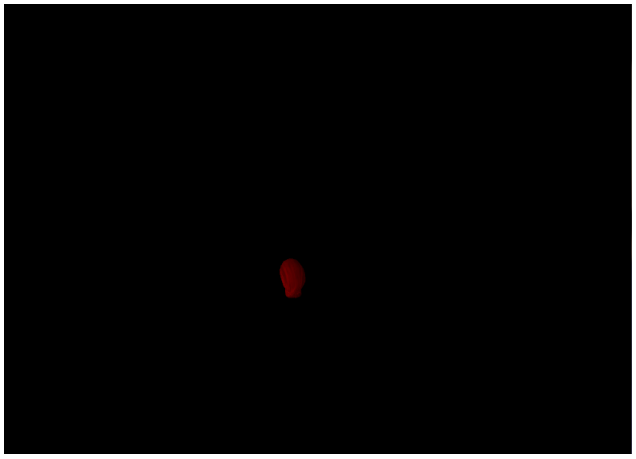


Figure 3.10: Using TitaniumGL

3.6 Summary

This chapter has analyzed a number of solutions to the problem of deploying a Second Life client in a CAVE. Some of which may be used, others may not. In this summary, we explain why.

Solution I brings forth the capability to merge Second Life Viewer's functionality with VR Juggler. By making use of the OpenGL interfaces of VR Juggler, the drawing methods could be rewritten with the Second Life Viewer's source code. This would allow the creation of an application object, to run Second Life Viewer, which is embedded by the kernel of VR Juggler, giving stereoscopic 3D, multi-display support and device interaction independently of the drawing code. The problem of this solution is that it requires extensive knowledge of the source code of both applications and that is very time consuming.

Solution II demonstrates how CaveSL can be configured in a system with three displays. The experiments were made on a single machine, with a virtualized cluster (through the use of two virtual machines). It turns out that using CaveSL for clusters is in fact possible and it has the feature of defining angles and fields of view, which is particularly useful if we want to set different arrangements for the displays (e.g. flat panel or cubic form). CaveSL has a limitation regarding camera rotation. It only rotates horizontally, not vertically. The author of CaveSL says that it is possible to add that functionality by changing the source code.

Solution III explains how stereoscopic 3D can be achieved by using Dale's SL Viewer. While it suggests that anaglyph stereo is compatible with any machine (using red/cyan glasses), the other modes, active and passive stereo, require specific hardware that may not be available on every machine or system. Active stereo requires a graphics card with a 3-pin din connector, while passive stereo requires

two projectors with polarized filters.

Solution IV relates to the usage of NVIDIA 3D Vision Surround technology. With this, it is possible to add multiple display support for any Second Life Viewer or any other client we desire to use (though only on flat panels and not like CaveSL where you can define different angles). Unfortunately, 3D Vision does not have Second Life as one of its supported applications for stereoscopic 3D. It only supports DirectX applications in most cases. Attempts have been made to convert the OpenGL rendering of Second Life Viewer to DirectX rendering, by using TitaniumGL and GLDirect, but the results were unsatisfactory.

In conclusion, the best way to achieve user immersion seems to be to use CaveSL and Dale's SL Viewer as the preferred clients. Using the NVIDIA 3D Vision Surround technology may prove useful when trying to add multiple display support for Dale's SL Viewer and when trying to use stereoscopic 3D inside the CAVE. Chapter 4 explains in further detail how these solutions work out on a triple-head setup and using the CAVE.

Chapter 4

CAVE deployment

4.1 Introduction

In this chapter, the solutions proposed in Chapter 3 are analyzed when used in a triple-head setup and in a CAVE. The process of deployment, to determine whether there are any incompatibilities with the systems used, is described. As with the previous chapters, the focus in this chapter is user immersion. The features of these systems are explored in order to achieve that. A simulation of the University of Minho's library is used.

4.2 Configuring a triple-head setup

For this project, we have to assess whether it is viable or not to use a triple-head setup for the APEX project. The CAVE is unquestionably a good system for providing immersion to the user. The question is, however, how much are we willing to sacrifice user immersion for a low cost solution (when compared to the CAVE). While using three monitors does not nearly surround a user like three projectors, it is definitely better than having just one monitor. The machine used at the University of Minho has the following characteristics:

```
Processor: Intel(R) Core(TM) i7 CPU 960 @ 3.20GHz
RAM: 24.0 GB
Graphics Cards: 2x nVidia GeForce GTX 460 1024 MB connected by SLI
Hard Disk: 500 GB / SATA
Monitors: 3x Samsung SyncMaster 2233 22'' - GeForce 3D Vision Ready
Sound: SoundMAX Integrated Digital HD Audio (Motherboard integrated)
Operating System: Windows 7 Professional Service Pack 1
```

The system configuration is meant to support three monitors at the same time by using the NVIDIA 3D Vision Surround technology. For that, at least two NVIDIA GeForce graphics cards compatible with 3D Vision Surround are required (they both have to be compatible with NVIDIA 3D Vision). NVIDIA drivers allow the user to select the following SLI configurations:

- **Maximizing 3D performance:** where the power of the graphics cards is used to maximize performance in 3D rendering;
- **Expanding monitors with Surround:** supports up to three monitors using NVIDIA 3D Vision Surround technology;
- **Activating all monitors:** supports up to three monitors, but applications may only run in window mode;
- **Deactivating SLI:** this option only uses one of the graphics cards. It only supports up to two monitors at the same time.

The options we are most interested in are expanding monitors with Surround and activating all monitors. The first one allows the user to run applications in full-screen using the NVIDIA 3D Vision Surround technology. This actually makes the computer capable of providing stereoscopic 3D on the three monitors in full-screen mode, treating them as if they were just one monitor. The option to activate all monitors does not support running applications in full-screen, thus it is a requirement to run them in windowed mode and expand them across all monitors. This may be an alternative if the other option fails to work for some reason, but it is not recommended because it does not use NVIDIA's technologies to the fullest (stereoscopic 3D on all displays is not supported in this mode). We may also want to choose maximizing 3D performance if the graphics rendering becomes too consuming.

Fig. 4.1 shows Dale's SL Viewer running in full-screen mode using the SLI configuration to expand monitors with Surround.

There is an issue when using Dale's SL Viewer. If the monitors are not aligned, i.e., like a flat wall, the objects that are not in front of the user may appear longer than they actually are. For instance, if monitor 1 on the left has an inclination of 45 degrees, monitor 2 on the middle has an inclination of 0 degrees and monitor 3 on the right has an inclination of minus 45 degrees, and the user is facing directly at monitor 2, the objects displayed by the other two monitors will look abnormal. This issue is not present when using CaveSL, because the user can define the angles displayed on each monitor.

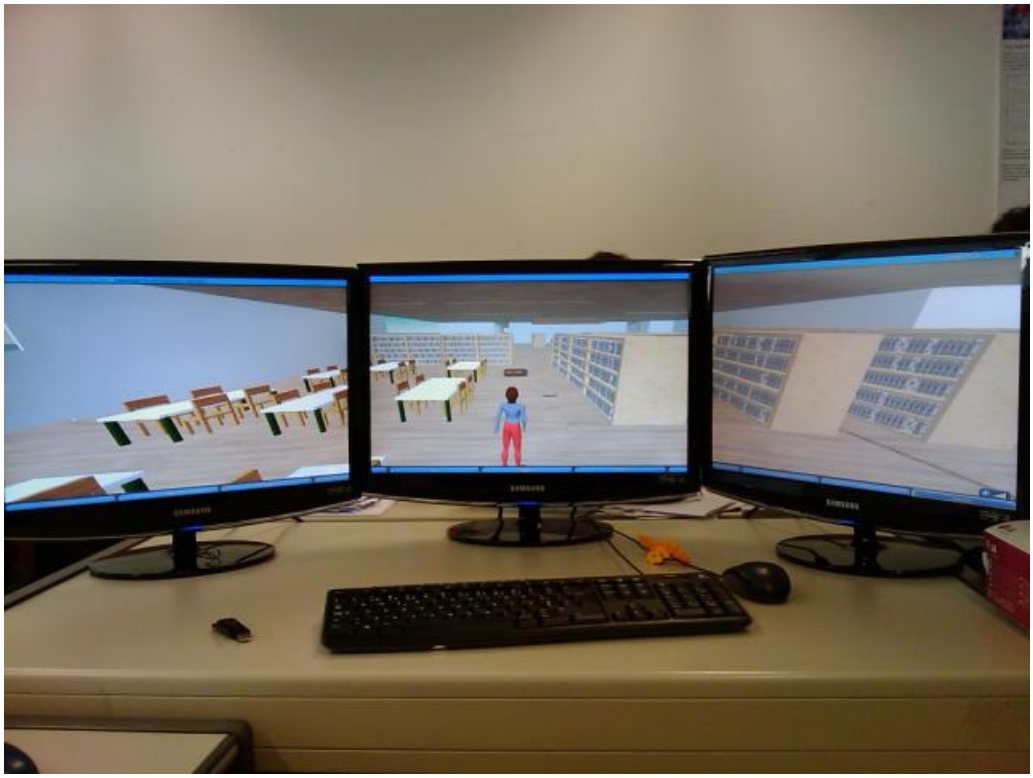


Figure 4.1: Triple-head setup

Another issue is concerned with the "black spots" caused by the borders of the monitors. The user cannot see anything between them and that results in alignment issues. Fortunately, NVIDIA's drivers provide ways to correct the gap between the displays, as shown in Section 3.5. The major limitation about NVIDIA 3D Vision is that it does not support OpenGL applications, when using GeForce cards. The only option is to try Dale's SL Viewer to obtain stereoscopic 3D in the visualization.

The results obtained with this kind of setup are similar to the ones described in Chapter 3. CaveSL can be executed using two virtual machines. The configuration is the same as the one described in Section 3.3. There were no problems when trying this solution on the triple-head setup. The amount of RAM, and the graphics cards processing power are more than enough to make the client run smoothly. Fig. 3.4 illustrates what is obtained when using the three clients at the same time. Dale's SL Viewer only works in anaglyph stereo mode. Active stereo does not work because the GeForce graphics cards do not have 3-pin din connectors, nor do they have OpenGL support with NVIDIA 3D Vision. Fortunately, it is possible to use Surround, allowing the client to be displayed on three screens at the same time. Fig. 4.1 shows Dale's SL Viewer running in the triple-head desktop.

4.3 CAVE characteristics

The CAVE present in the LVP at Guimarães is composed by three rear projection screens in a single flat screen, which means that they are lined up on a single plane right next to each other. Although this setup does not properly surround a user in a cubic fashion, it does provide a larger image that results in a more immersive experience than in a traditional system. This setup, however, will be eventually modified to make a U form.

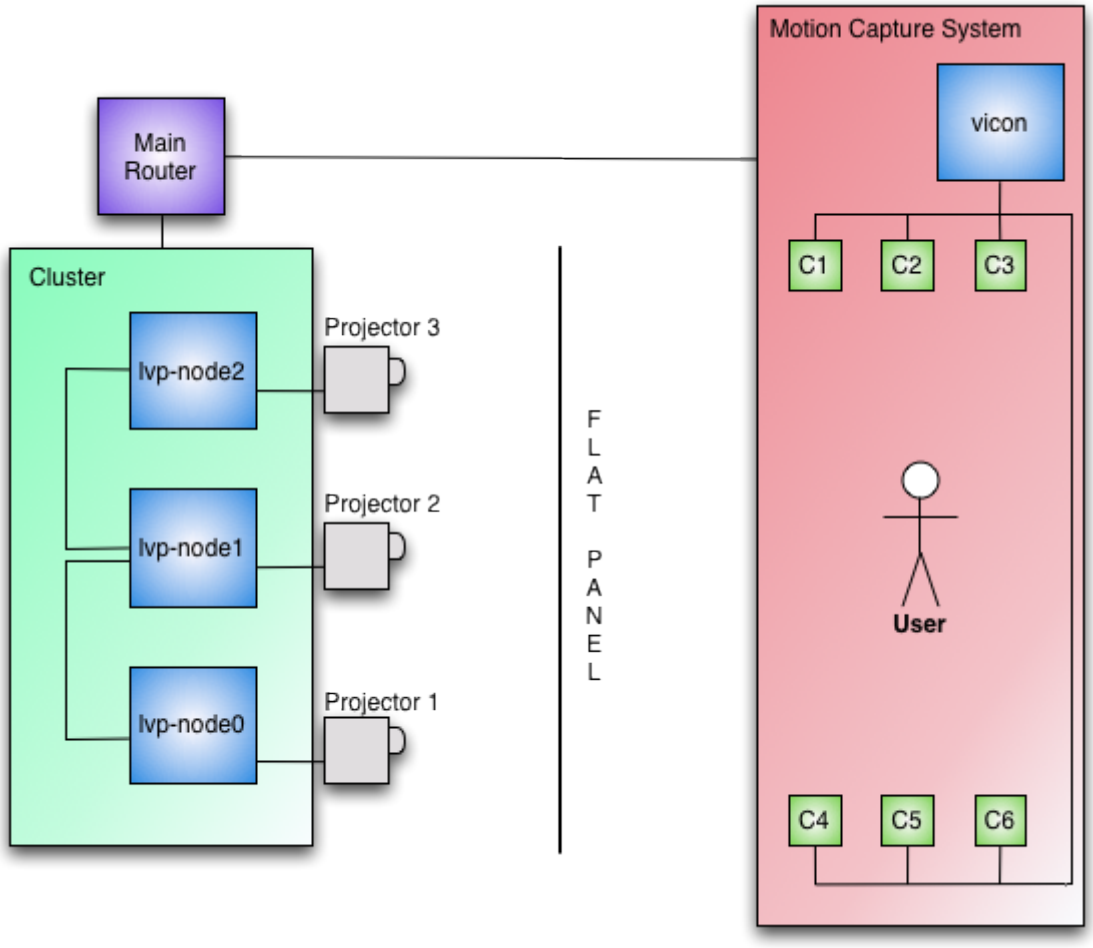


Figure 4.2: CAVE in the LVP

As represented in Fig. 4.2, the system is a cluster with three machines (lvp-node0, lvp-node1 and lvp-node2), each one for each projector, that can be controlled individually with their own input devices or via the network. Each machine has the Linux operating system installed but it is easy to add a new hard drive disk to support other operating systems. In our case, we have installed one new disk on each machine to support our solutions, without disrupting the LVP's daily activities.

Each of the three machines has an output for the respective projector and another for a LCD monitor (the monitors are not represented in the figure). What is reproduced in the projectors can be done using three LCD monitors, if we do not want to drain the projectors' life. This is useful for testing purposes. However, if we want to use stereoscopic 3D, then we must use the projectors because that functionality is not supported on the LCD monitors.

The other system present in the CAVE is the Motion Capture system, where the user's movements can be captured by six infrared cameras (C1, C2, C3, C4, C5 and C6). This system is implemented in a way that the user can move freely around the room, while looking to the visualization displayed on the flat panel. Everything related to the cluster system is hidden from the user. Currently, the Motion Capture system records the movements of the user on a computer (vicon) for posterior analysis. There is no interaction with the cluster visualization whatsoever. In Section 5.2.3, we will present how we are planning to initialize interaction with the user in real time.

4.4 Deploying the proposed solutions

4.4.1 CaveSL

The configuration of CaveSL for the CAVE was very straightforward, because it does not differ much from the configuration used on the triple-head setup using the virtual machines (see Fig. 3.3). However, in resemblance to the configurations made in the file *cave.cfg*, in Section 3.3 of Chapter 3, to use CaveSL's capabilities to the fullest, adjustments to the angle and FOV need to be made, considering that the CAVE has a single flat panel where the visualization is projected. Since one of the projectors at the LVP was being repaired, we used a configuration involving two machines of the cluster:

- lvp-node0: FOV: 0.75049 rad, angle: 0.5236 rad;
- lvp-node1: FOV: 0.75049 rad, angle: -0.5236 rad.

Once the LVP arranges its projectors, to surround the user with displays in a U form, we can change CaveSL's properties, without any issues, to support that. Fig. 4.3 shows CaveSL configuration using the CAVE's settings at the LVP (assuming we could use the three machines). Fig. 4.4 shows CaveSL running at the LVP.

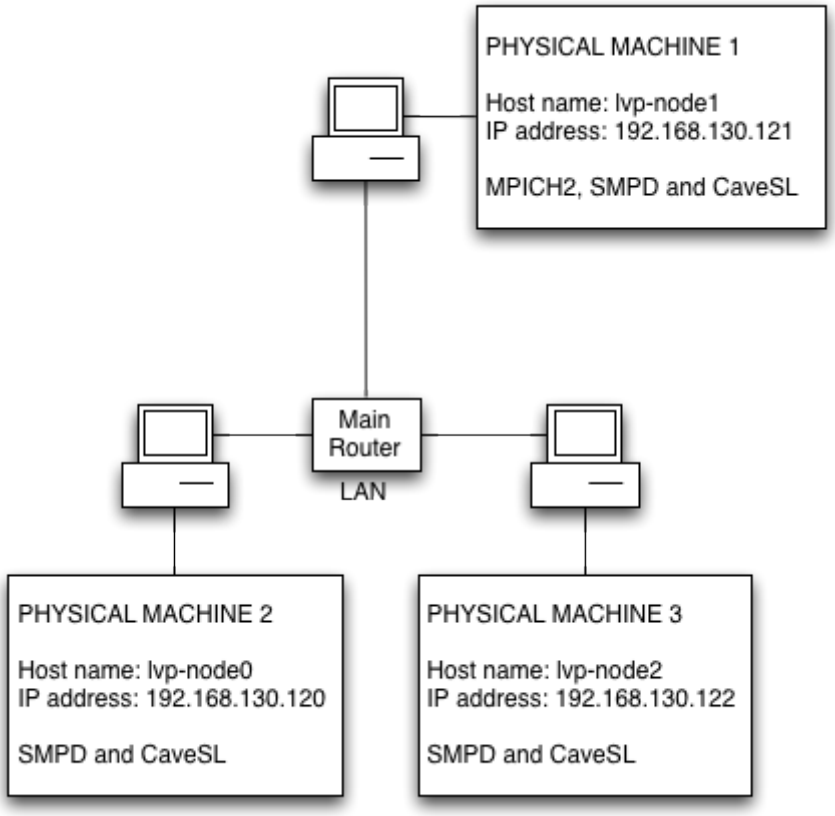


Figure 4.3: CaveSL implementation for the CAVE

4.4.2 Dale's SL Viewer

Before showing the obtained results for Dale's SL Viewer, it is important to mention that each machine of the cluster has a NVIDIA Quadro 4500 graphics card that comes with a 3-pin din connector and two Digital Video Interface (DVI) outputs for visualization. The projectors (each one is a Christie Mirage S+4k) have active stereo support. Besides the DVI, each projector has the following input/output: a General Purpose Input/Output (GPIO) interface, an input and output for RS232, and a connector for RS422. Some of these hardware aspects are relevant for stereoscopic 3D support.

The CAVE at the LVP has support for NVIDIA 3D Vision technology. However, alterations need to be made to support Surround. Each machine of the cluster has two DVI outputs, which means that they only support up to two projectors at the same time. Installing two graphics cards of the same type, in SLI mode, on a single machine, would solve this issue. For instance, if we connected the machine described in Section 4.2 to the projectors inside the CAVE, we would be able to see the image provided by Dale's SL Viewer across the three projectors. At the moment, however,

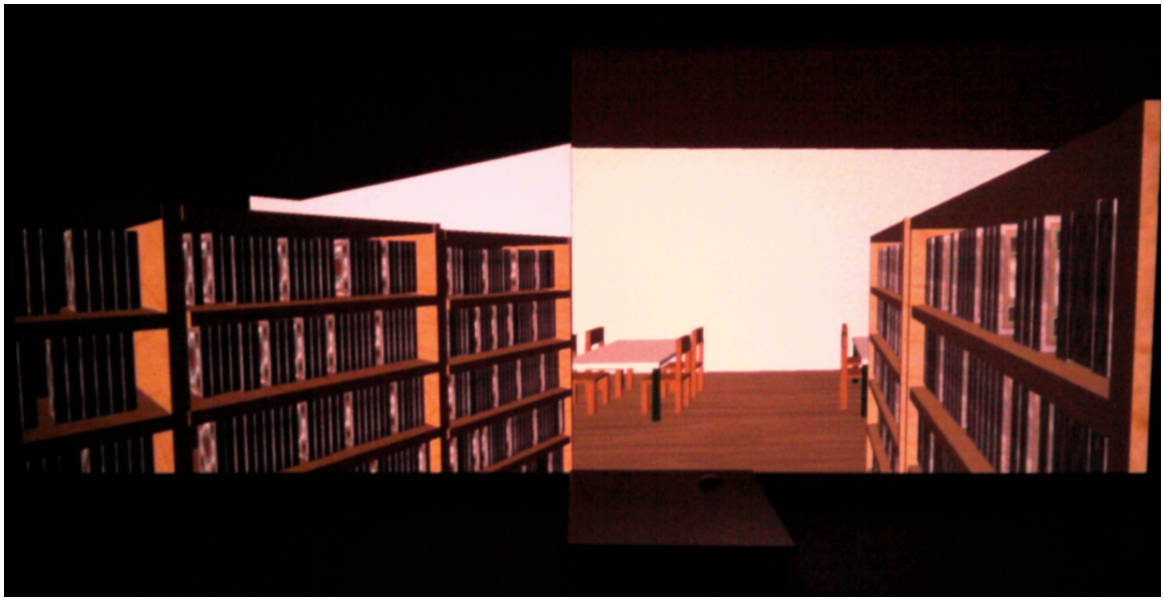


Figure 4.4: CaveSL running in the CAVE

Dale's SL Viewer can only be tested using one projector, because each machine is directly connected to one projector only.

Testing Dale's SL Viewer's stereoscopic 3D modes, inside the CAVE, resulted in the following:

- Anaglyph stereo: since this mode is compatible with almost any kind of system, no problems happened when using it. With the appropriate 3D red/cyan glasses, the user can see objects outside of the screen. Fig. 4.5 shows Dale's SL Viewer running in anaglyph stereo mode inside the CAVE;

- Active stereo: taking into account the hardware used for the CAVE, i.e. the graphics cards have 3-pin din output that can be used to connect to the projectors via GPIO, and the NVIDIA Quadro graphics cards have OpenGL quad-buffered stereo support for NVIDIA 3D Vision, it is possible to obtain stereoscopic 3D using this mode. However, testing did not show significant results. This may be due to abnormal configuration of the projectors and graphics drivers;

- Passive stereo: considering that the projectors do not have polarized filters, this mode is not supported.

Figure 4.5: Dale's SL Viewer running in the CAVE

4.5 The library case study

To test the Second Life clients using the CAVE and the triple-head setup, we created a replica of the library at the University of Minho's Gualtar campus. A library is a good case study for the APEX framework, and, indeed, a small version of the environment had already been developed [Silva et al. (2010)]. The idea was to create a library with the following characteristics: the entrance of the library has a gate and, once the user approaches the gate, it is supposed to open; once the gate is open, the user can enter the library; the user indicates which book he desires to find and then the book lights up, when the user is close by, to facilitate identification. It is a simple case, but it illustrates how a library can be explored in a easier way when it is equipped with ubiquitous computing devices and sensors. Therefore, to make the experience of the user more realistic, it is important to create a replica of a place he already knows and it is also especially important that the library has a reasonable number of books and bookshelves for the user to get lost in.

Fig. 4.6 shows the plants of the second and third floors of the library at University of Minho's Gualtar campus.

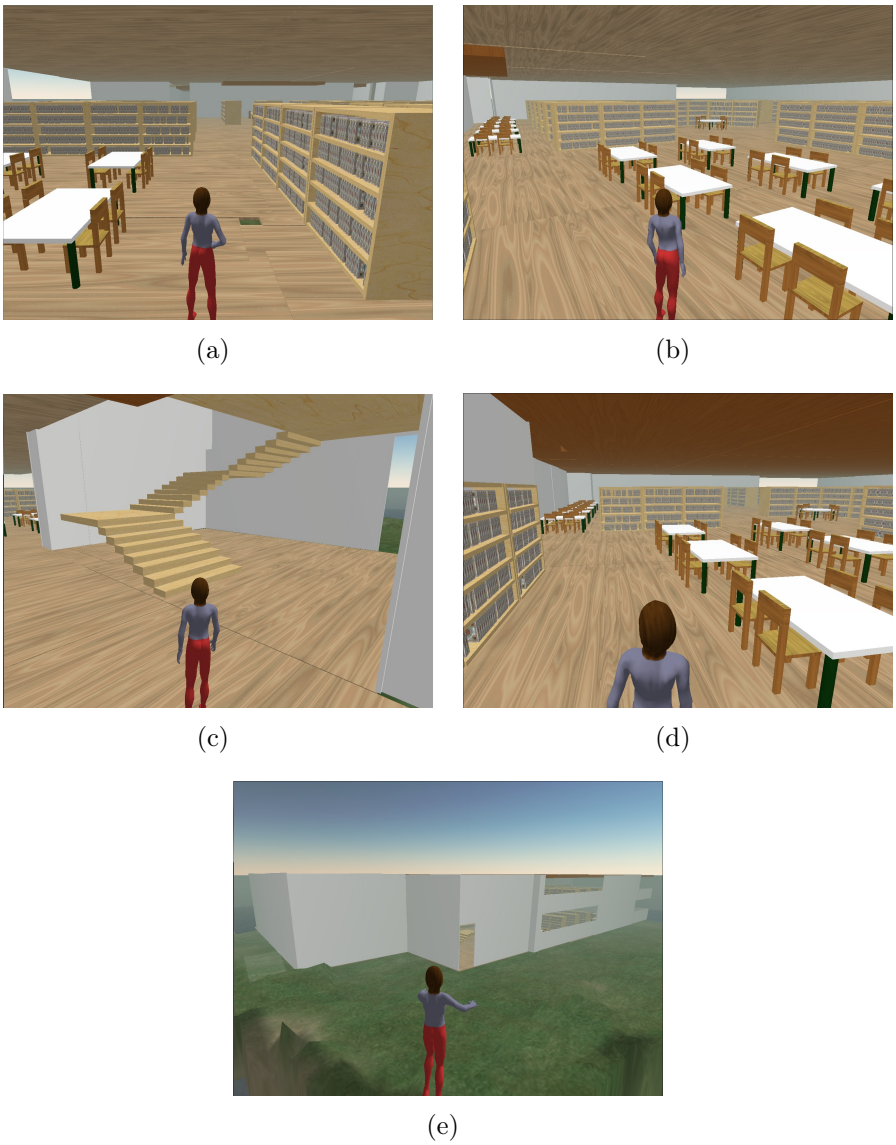
(a) Second Floor of the Library (b) Third Floor of the Library

Figure 4.6: University of Minho's library plant at Gualtar campus (from [SDUM (2011)])

Some of the characteristics of the physical library were taken into account to make the replica in Second Life recognizable by the user, such as:

- The walls surrounding both floors are all white;
- The number of steps someone has to take in the stairs connecting both floors is the same as in the real building (twenty-three);
- Each floor has windows on every corner;
- There are three round tables on the center of each floor (between bookshelves that are placed there); Those tables have a white top and four green legs with their position farthest away from each other;
- There are rectangular tables dispersed inside each floor. Those tables have a white top and four green legs, one in each corner;
- The chairs are made of wood and so is the floor. The chairs are square and have four legs on each corner;
- The bookshelves on each floor are pretty much distributed on a equal manner. The main difference, when observing both plants in Fig. 4.6, are on the top side, where on the third floor four lines of bookshelves are distributed vertically and on the second floor there is only a single line of bookshelves close to the wall, and on the left side, where on the third floor there are only four lines of bookshelves distributed horizontally and on the second floor there are five lines;
- The ceiling is brown colored.

The final result, with the library built in the OpenSimulator server, is demonstrated in Fig. 4.7.



(a) (b) (c) (d) (e)

Figure 4.7: Library created in the OpenSimulator server

4.6 Summary

In this section, we have explored the deployment of two Second Life clients (CaveSL and Dale's SL Viewer) on both a triple-head desktop machine and a CAVE system.

By using the triple-head setup, we have managed to get CaveSL and Dale's SL Viewer working as they are supposed to. CaveSL worked by using two virtual machines where each client worked as a slave. On the native operating system, a

master client was launched, which commands the movements of the avatar on all clients, resulting in simultaneous camera movements. Dale's SL Viewer worked well when using anaglyph stereo mode. Thanks to NVIDIA 3D Vision technology, it was possible to use it with three displays at the same time (with the client running in full-screen mode as if it were just one screen). Unfortunately, 3D Vision does not give stereoscopic 3D for Second Life clients, because of the lack of support to GeForce cards. One of the advantages of the triple-head setup, however, is that it is possible to connect three projectors to it and make it act as a CAVE, provided that there is enough RAM to support the virtual machines.

There are also some interesting results when using the solutions in the CAVE. CaveSL's configuration for clusters is identical to the one used with virtual machines. Since angles can be defined, it is not stuck to flat panels only, though it can be used for that too. Contrary to the triple-head setup, the machines of the cluster have graphics cards with 3-pin din connectors, which means that active stereo for Dale's SL Viewer is supported. Usage of Dale's SL Viewer on a cluster is limited though, because it can only take advantage of a single projector.

The issue of getting a client to work in a CAVE is concluded in this chapter to make way for interaction devices. In the next chapter, solutions are presented to aid user interaction inside the CAVE.

Chapter 5

Interacting with mobile devices and sensors

5.1 Introduction

In this chapter, solutions are presented for user communication with OpenSimulator, via the Second Life clients, in the context of the CAVE. First it is explained why using a keyboard and mouse is not viable in the CAVE. Then, a solution using the capabilities of a Wiimote is presented. Finally, the Motion Capture system at the LVP is explained.

5.2 Input devices

5.2.1 Keyboard and mouse

The usual method to interact with a Second Life client is to use the keyboard and mouse. As shown in Table 5.1, the user can move the avatar using the directional keys of the keyboard or the W, S, A, D keys. The keyboard also presents a number of shortcuts that help the user during his experience using a Second Life client. For instance, the user can tell the avatar to start/stop flying by pressing F. While flying, the user can hold E to order the avatar to fly up and hold C to order the avatar to fly down. Jumping can be done by pressing Space or E. The keyboard also works as a good intermediary for GUI browsing. The manual created by Friedkin (2008) presents some of the shortcuts that can be used for that purpose.

Table 5.1 is not entirely in synchrony with the manual provided by Friedkin (2008), because key bindings can be customized by the user as much as he sees fit. The key bindings shown in the table are the ones used in Dale's SL Viewer (and will

Keyboard/Mouse command	Second Life viewer Action
Keyboard Up / W	Move Forward
Keyboard Down / S	Move Backward
Keyboard Left / A	Turn Left
Keyboard Right / D	Turn Right
Mouse Right Button	Object/Terrain Properties
Mouse Left Button	Select Object
Mouse Up	Cursor Up
Mouse Down	Cursor Down
Mouse Left	Cursor Left
Mouse Right	Cursor Right
Mouse Wheel Up	Zoom In
Mouse Wheel Down	Zoom Out
Jump	E
Fly	F
Fly Up	Hold E
Fly Down	Hold C

Table 5.1: Second Life clients' key binding

be used for reference in other sections), though we can assume that the user can set whatever keys he wants.

The advantages in using the keyboard and mouse as the main input for the Second Life clients are as follows:

- It is an easy to customize solution. Key bindings can be set by the user;
- Keyboard shortcuts provide faster access to certain menus in the GUI;
- Users accustomed to using keyboard and mouse will feel at home when using them in Second Life clients.

The drawbacks in using the keyboard and mouse as the main input for the Second Life clients are as follows:

- Limitations caused by wires and/or having to use flat surfaces, such as tables, to work with the devices. Many of the keyboards and mouses on the market still connect to the computer with USB cables. This can be solved by acquiring a set that connects to the computer through Bluetooth, but it does not solve the problem of needing a flat surface to use the mouse. Using a wireless keyboard only (one arm supporting it and one hand giving commands) would allow the user to move freely around the CAVE, but the experience would definitely be highly restrictive;

- The referred to devices are not originally meant for 3D applications. While the user can control the avatar with them, the experience is not even slightly realistic. Keyboards and mouses lack the feeling of immersion. A device that actually translated the user's movements into input for the Second Life clients would be quite an asset, because it would add realism to the experience.

5.2.2 Wiimote

The Wiimote is a controller, created specifically for the Wii gaming console, that connects to it via Bluetooth. The main idea of this controller is to provide the user with a more familiar interaction than that traditional controllers, keyboards and mouses are able to give. Here, we are particularly interested in taking advantage of its accelerometer to improve interaction (see [Palha (2010), Kela et al. (2006)] for studies on the viability and advantages of using accelerometers as interaction devices).

The Wiimote has directional buttons (up, down, left and right), buttons for interaction and menu browsing, a three-axis accelerometer, an infrared camera, wireless bluetooth connectivity, a vibration motor, a speaker, four blue LEDs and a port to connect external devices like the Nunchuk [Santos et al. (2010), Barbosa and Silva (2010)]. The Nunchuk has the same movement detection mechanism of the Wiimote, but it provides an analog joystick for character movement, i.e., in games that a user controls an avatar, it can make its movements smoother. In this project, however, we only had the Wiimote, so we will not be using the Nunchuk. Fig. 5.1 displays the buttons the Wiimote has.

Considering that the Wiimote has bluetooth connectivity and it does not have any specific requirements regarding encryption and authentication, it can be used on a traditional computer for input. In this project, Dolphin (a Wii emulator) is used to detect and pair the Wiimote with the bluetooth device connected to the computer. During the bluetooth pairing process, the user has to hold both 1 and 2 buttons at the same time. By doing this, a packet is sent, with the information about the buttons that are pressed, to the computer through an Human Interface Device (HID) input report. [Barbosa and Silva (2010)]

In order to instruct the Wiimote to work with a Second Life client, we use an application called GlovePIE. GlovePIE is an application that allows the user to map the Wiimote's buttons, for instance, to regular keyboard and mouse commands. It receives the signals from the Wiimote via bluetooth and then converts those signals to the functionality desired by the user. This can be done using a scripting language that the author created for better customization. The scripting language

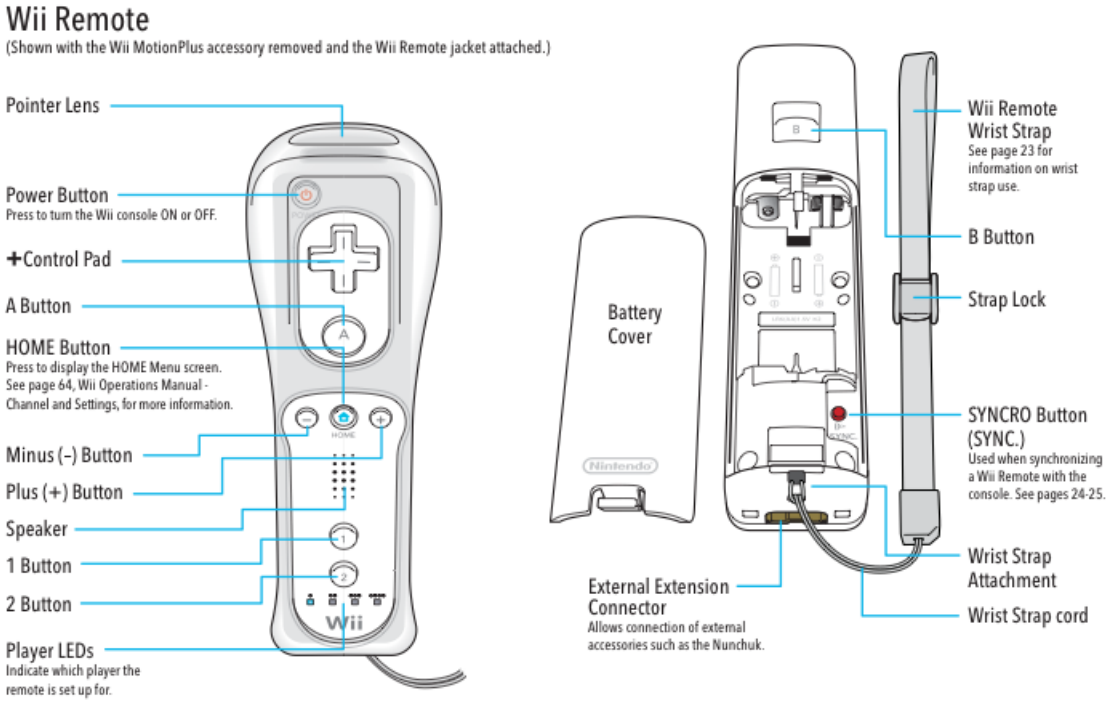


Figure 5.1: Wiimote (from [Nintendo (2009)])

allows the use of assignment statements, which means we can declare variables, the use of if statements, where we can define specific conditions, and the use of loops. Furthermore, it comes with a set of commands and functions that may help development, such as: mathematical functions (sin, cos, tan, log, round, sqrt, etc.), wait commands and say commands (pretty much like printf in C).

Since the scripting language, provided by GlovePIE, has enough features to configure the buttons of the Wiimote, then it is a viable option to use the Wiimote to interact with a Second Life client. Table 5.2 shows how Wiimote buttons are mapped as keyboard and mouse commands for this project. Appendix I [Chari (2009)] presents the code created, in the scripting language of GlovePIE.

Considering the key bindings described in Table 5.1 in Section 5.2.1, the mapping done with GlovePIE allows the user to move the avatar using the directional buttons. It also lets the user select an item by moving the mouse cursor, while holding B button and using the directional buttons. By pressing the Home button, the user can select the object pointed by the mouse cursor. Item and building options can be seen by pressing the A button. Zooming in and zooming out is done with the Plus and Minus buttons, respectively. If a user zooms in until it gets to first-person view, mouse movement is converted to camera rotation (as if the user is watching through the avatar's eyes). With this kind of mapping, the user can do pretty much anything he could with the keyboard and mouse, though he will not be able to use

Wiimote Button	Keyboard/Mouse/Wiimote Command
Up	Keyboard Up
Down	Keyboard Down
Left	Keyboard Left
Right	Keyboard Right
Up & B	Mouse Up
Down & B	Mouse Down
Left & B	Mouse Left
Right & B	Mouse Right
A	Mouse Right Button
B	N/A
Home	Mouse Left Button
Plus	Mouse Wheel Up
Minus	Mouse Wheel Down
1	Enable Wiimote Accelerometer
2	Disable Wiimote Accelerometer

Table 5.2: Wiimote button mapping with GlovePIE

shortcuts or browse the GUI as smoothly.

Another feature, included in the script created for the Wiimote, is the ability to use the accelerometer to control the mouse cursor. By pressing 1, the user enables it. By pressing 2, the user disables it. It is important to have the ability to enable/disable this option because using the accelerometer is not quite easy when manipulating the cursor (our experience tells us that selecting a small object using the accelerometer is a difficult task). When in first-person view, however, using the accelerometer might be a thrilling experience, because it makes the avatar's movements look more natural and the user can actually point to where he wants to look.

There are three axis in which the user can rotate the Wiimote to provide input. The z-axis (pitch) refers to up and down movements, the x-axis (yaw) refers to left and right movements and y-axis (roll) refers to rolling movements, i.e., the y-axis is aligned with the Wiimote, so rotating 180 degrees in that axis means turning the controller upside down, if its initial position was upside up.

So, in order to make the camera move with the accelerometer, we adapted a script from [Chari (2009)], made in GlovePIE, for our needs. Basically, when the user rotates the Wiimote on the z-axis, the camera moves up or down accordingly. If the user rotates the Wiimote on the x-axis, the camera moves left or right accordingly. The sensitivity of the accelerometer suffers adjustments, to avoid sudden camera movements, based on the level of rotation of the respective axis. Equations 5.1 and

5.2 demonstrate how sensitivity plays an important role on mouse cursor control with the Wiimote.

$$MouseX = MouseX \pm \frac{1}{Sensitivity} \tag{5.1}$$

$$MouseY = MouseY \pm \frac{1}{Sensitivity} \tag{5.2}$$

If the value of the *Sensitivity* variable raises, less movement is applied to the mouse cursor. Likewise, if the value of *Sensitivity* lowers, more movement is applied to the mouse cursor. In the script, we have four sensitivity settings to let the user look at places with more precision or to rotate the camera faster. If the Wiimote is slightly inclined (either on the x-axis or z-axis), the mouse cursor will begin to move slowly. By adding more inclination, the script starts using sensitivity settings with lower values, resulting in faster cursor movement. Note that rotations in the x-axis and z-axis of the Wiimote's accelerometer correspond to mouse cursor movements in the x-axis and y-axis, respectively. Also, different sensitivity settings can be applied to the x-axis and y-axis of the cursor, because they are treated independently. This allows simultaneous movement on both axis.

In conclusion, first indications show that the advantages in using the Wiimote, when compared to the traditional keyboard and mouse, are as follows:

- The experience inside a CAVE is far more satisfactory than with a mouse or a keyboard. Holding a keyboard or mouse during the CAVE experience is not comfortable at all nor does it fit well with its immersive setting (the user would have to hold the keyboard with his arm at the very least and would need a table to use the mouse). The Wiimote does not need wires and the user can move freely with it without restrictions;

- The three-axis accelerometer contributes to the immersion of the experience. When in first-person view, the avatar's movements look more natural.

The drawbacks in using the Wiimote, when compared to the traditional keyboard and mouse, are as follows:

- Shortcuts to GUI options are not available. The user has to move the cursor to the options he desires;

- Selecting objects might not be as easy as when doing it with a mouse, especially if they are small.

5.2.3 Motion capture

Although the Wiimote solution seems viable and interesting, the ideal solution would be to be able to interact with the environment without the help of any device. In principle, this can be achieved through a Motion Capture system. However, before Motion Capture can be used, a number of technical details must be dealt with in the CAVE. While solving them was outside the scope of this work, this section identifies the main issues, and puts forward some possible solutions.

As stated above, one of the reasons to use the Motion Capture system at the LVP (see Section 4.3 for a detailed description of the system) is that it does not rely on a specific device for control. Instead, it is possible to convert the user's movements into actions that can be interpreted by computer applications. This would benefit the user when interacting with the Second Life client, because it would allow him to interact with the simulation more freely. For instance, if a user decides to pick an object, he could express that by trying to reach it with one of his hands. This would make the interaction with the world far more realistic but it implies, of course, a lot of work and study about real-time user interaction in a VR environment.

The Motion Capture system captures the movements of a user by using six infrared cameras. The user needs to have several markers attached to his body for camera detection and for computer processing. By using the software from Vicon (used at the LVP), it is possible to define labels for each marker on the body. These labels can define where parts of the body are located, such as the head, chest, arms and legs. The existing software, at the LVP, for Motion Capture control is the following:

- **Vicon Nexus**, which is specifically used for Life-Science experiments. With this software, users can analyze motion and use it for purposes concerning human health and physical condition. Vicon Nexus comes with native real time capability for data analysis; [Vicon (2011)]

- **Vicon iQ**, which is meant for films, games and broadcasts, for instance. This software has the capability to provide very accurate data. It also comes with native real time capability for data analysis. [Vicon (2011)]

The major drawback of the Vicon software is that it is proprietary. Integration with Second Life clients is not an easy task because there is no way to access its code. The only way to do it is to somehow find a way to obtain the output from Vicon in real time and pass it on to the applications we desire. There may be two ways to do this: either find a way to obtain the output from the Vicon software

in real time or use an interface (e.g. GPIO) that the Motion Capture hardware provides.

How to interact with a Second Life client, however, is an issue that must be taken care of later on because, for now, there is another issue at the LVP that needs addressing if we desire real-time interaction. Currently, there is no direct communication between the visualization (cluster) and the Motion Capture system. In practical terms, this means that the capture is started manually by the user, which results in it starting before or after the visualization starts, depending on when he wishes to do it. Not only does this constitute as a problem for data analysis (because there is no reference to compare the occurrences on both ends), but it is also a problem for real-time interaction, because they need to be synchronized for it to work.

To obtain this kind of synchronization, two kinds of approaches are taken into consideration:

- The first one is to synchronize the clocks on the intervenient computers to sub-millisecond accuracy;
- The second one is to start the visualization once the capturing starts.

While the first one does not resolve the issue, it can lead to the second approach, where we may desire to issue a command to start both systems at the same time. Besides, it can facilitate data analysis on both ends. For instance, if we have output of the visualization and the Motion Capture system, we can compare both of them using time as reference. To be able to compare both outputs is also a very needed requirement.

In order to achieve clock synchronization with sub-millisecond accuracy, the Network Time Protocol (NTP) was used. A communication between client-server has been established to make the clocks in the cluster and the Motion Capture system coincidental. Fig. 5.2 shows how the communication is done between the clients and the server at the LVP.

The server (biomose-desktop2) sends packets to the clients (computers at the cluster and the Motion Capture system) to synchronize their clocks according to its own. Since we have the computers connected to each other in a Local Area Network (LAN), sub-millisecond accuracy is achievable and the desired accuracy can be obtained. Appendix II shows the configuration used in all computers in order to make NTP work properly.

It was possible to obtain some satisfying results regarding clock synchronization. Inside a local network, the computers can have their clocks synchronized with very

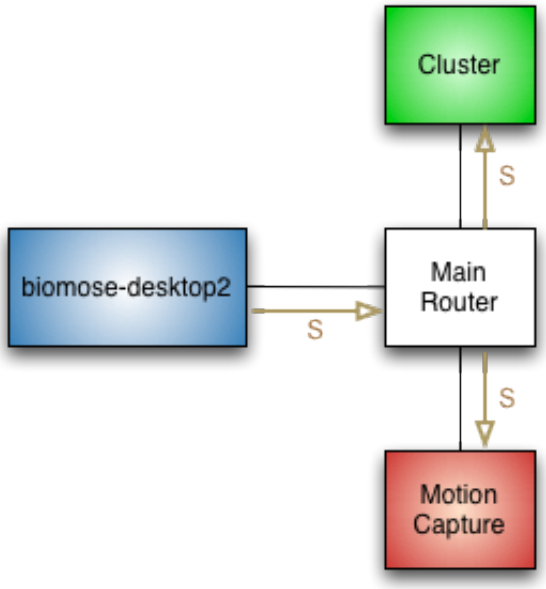


Figure 5.2: Communication between clients and server using NTP

low levels of jitter (e.g. below one millisecond). However, synchronizing clocks between different operating systems, such as Linux and Windows, is not always accurate, because the first one uses NTP and the second one uses the Simple Network Time Protocol (SNTP), which is not as complete. This could be solved by installing an accurate NTP client in the Windows platform.

To start the visualization once the capturing starts, there may be two possible approaches:

- Modify the code of the application dealing with the visualization, to start at the same time as the capture;
- Use the projectors' output to send a trigger that starts the capture.

The first approach involves modifying the code of the desired application (e.g. Second Life client in our case). The second approach involves knowledge on how the Vicon software and hardware work. Further investigation needs to be done to accomplish this task.

5.3 Summary

During this chapter, it was explained why the usage of a keyboard and mouse is not suitable for a CAVE. It limits the user's movements. The familiarity some users have with such interaction (due to usage of these devices on other applications) does

not justify their use in detriment of other solutions. They may be of good use in the background, when the experience is coordinated by a supervisor. The user, however, needs to be totally independent from these devices.

Using the Wiimote is a solution that allows the user to move freely when interacting with the simulation. By using the capabilities of its accelerometer it is possible to make the camera rotations feel more natural. The buttons of the Wiimote are enough for the user to access all the functionality of the Second Life clients (though access to menus is not as fast as when using the keyboard and mouse). Thanks to GlovePIE, it is possible to script functions for each button and for the accelerometer, making the Wiimote highly customizable, and to interact with any client we desire.

The main advantage in using the motion capture system is that it provides a hands-free interaction with the Second Life clients. But before that can be achieved, the system at the LVP has a limitation that needs addressing: it cannot interact with the visualization in real time. The visualization created by the clusters and the motion capture system need to be synchronized in order to make that happen. First step involves synchronizing the clocks of the computers to know when each event occurs. The second step is to initialize both systems at the same time.

Chapter 6

Conclusion and future work

This dissertation, developed in the context of the APEX project, has investigated ways to integrate a 3D application server with a CAVE system, with a view on the prototyping of ubiquitous environments.

In this chapter, an overall analysis of the work done for this thesis is made. A ponderation of each solution is made to check whether the objectives defined, for this project, were achieved. It is important to know the limitations of each one so that they can be minimized in the future.

6.1 Overall analysis

Taking into account the contextualization and objectives defined in Chapter 1, we can say that they have been accomplished, to some extent, with the encountered solutions. In Chapter 2, we defined that using OpenSimulator was the best option for a 3D application server, not only because it is free to use and open source, but also because it has compatibility with Second Life clients. This facilitated the task at hand because it eliminated the issue of creating a complete new client to work with clusters and provide immersion, while still taking advantage of the benefits of OpenSimulator. Indeed, as Section 2.4 in Chapter 2 indicates, there are two clients that can provide useful features such as multiple display support and stereoscopic 3D: CaveSL and Dale's SL Viewer, respectively.

The main issue with this is that we cannot use both features at the same time, unless we somehow combine the source code of both viewers. In Chapter 3, these solutions were explored, along with some others, to counter this issue. The first one suggests that using VR Juggler could add that kind of functionality if we managed to integrate Second Life Viewer's source code using VR Juggler's application architecture. The problem with this solution is that it requires extensive knowledge of

the source code of both applications. Trying to understand and integrate the code turned out to be a very time-consuming task with no obtainable results in the short term. Even if we managed to do it, there is not enough evidence to prove that, once it was done, all the features of VR Juggler (such as stereoscopic 3D), would work immediately without at least making some tweaking. The capabilities of extending VR Juggler code, however, suggest that this is possible. Instead of relying on this solution, we decided to focus on the capabilities of CaveSL and Dale's SL Viewer.

CaveSL does provide very good support for clusters in a multi-display fashion, such as the CAVE. The user can define the angle and field of view for each panel, so it is not only meant to be used in flat panels. It can be used with displays arranged in a cubic form (typical of CAVEs) and use as many clients as we desire (there is no limitation in the amount of displays and machines in the cluster). Dale's SL Viewer, contrary to CaveSL, does not have native multi-display support. But it comes with a few interesting features regarding stereoscopic 3D. It comes with an anaglyph stereo mode, which works with any system we desire, including the CAVE, an active stereo mode, for more refined stereoscopic 3D imaging, and a passive stereo mode.

It is important to mention that NVIDIA 3D Vision Surround technology, discussed in Chapter 3, also takes an important role on the deployment of these solutions, because it comes with features that have multiple display support and stereoscopic 3D as well. For instance, Dale's SL Viewer does not have native multi-display support, but with NVIDIA 3D Vision Surround, we can make it work with up to three monitors or projectors at once. The only limitation is that it is not possible to define the angles and fields of view, for each display, like it can be done in CaveSL. So, for better quality, the displays need to be arranged as if they were a single flat panel.

NVIDIA 3D Vision can also give stereoscopic 3D for applications that don't have native support for it. Unfortunately, it does not support all the existing applications on the market and Second Life Viewer is one of them, which means that CaveSL cannot take advantage of this feature, nor does Dale's SL Viewer (for active stereo) or any other client for Second Life. This is because 3D Vision does not support OpenGL on NVIDIA GeForce cards. Attempts have been made in the CAVE, using NVIDIA Quadro graphics cards, and an approach to convert OpenGL rendering to DirectX was also tested (described in Section 3.5), but without successful results.

With this in mind, we have decided to test these solutions on a triple-head setup and the CAVE (Chapter 4). On the triple-head setup (i.e. a single machine connected to three displays simultaneously), we managed to get the solutions working. CaveSL works by making use of two virtual machines with one client on each and

another client on the native operating system. We can define the angles according to the monitors position. The main drawback of this is that it is required a decent amount of RAM and a good graphics card to make the clients run smoothly. Fortunately, with today's technology, it is possible to purchase a good computer, at a decent price, to support this. Second Life clients are not as resource consuming as some recent 3D graphics applications. The triple-head setup had no performance issues.

Dale's SL Viewer works well on the triple-head setup. It is possible to have stereoscopic 3D using anaglyph stereo mode, with red/cyan glasses, and we can have it run on multiple screens thanks to NVIDIA 3D Vision Surround. The limitation is that the displays need to be aligned perpendicularly to the user, otherwise objects will appear to look bigger on screens with different angles.

In the CAVE at the LVP, installing the clients did not bring any issues. They function pretty much as they do on any other machine. CaveSL works via communication between the machines of the cluster. One of them acts as the master node and the other two as slaves. The configuration is basically the same used for the triple-head setup with the virtual machines. Only the hostnames in the configuration need to be altered to work with the cluster. Defining angles and fields of view turns out to be beneficial when using the CAVE, because we can adjust them to look better when using an undefined number of panels around the user or when using a different amount of projectors, like two instead of three. Unfortunately, there is no stereoscopic 3D support for CaveSL coming directly from the system installed at the CAVE. For CaveSL, we used two projectors pointing at a single flat panel (the third one could not be used because of technical issues), but extending it to three projectors later on will not prove to be difficult.

Dale's SL Viewer is limited to one machine and one display only. It cannot take advantage of the cluster because it was not programmed in that way. However, the machines have graphics cards that have 3-pin din connectors (see Fig. 3.7) that add active stereo support for the client, with the appropriate configuration. Anaglyph stereo works without any problems. The issue of not having multiple display support, with Dale's SL Viewer, can be easily solved if we use the triple-head setup connected to the three projectors. It should work with a decent quality provided that the projectors are all pointed to a single flat panel.

When it comes to interaction, as described in Chapter 5, we came up with a solution using the Wiimote. The Wiimote not only removes the limitations caused by wires (it connects to the computer via Bluetooth), but it also makes the interaction feel more realistic because of the capabilities of the accelerometer. With the Wi-

imote, the user can tell he wants to look up or down and left or right by inclining it vertically or by rolling it, respectively. This is useful when the user is in first-person view mode. The user can do anything he could with the keyboard and mouse by using the Wiimote's buttons, such as moving the avatar around, selecting an object and see an object's properties. This solution turned out to be very advantageous, because it is compatible with any desired client, thanks to the capability to script functions for each button of the controller.

Another approach that can be used for interaction is Motion Capture. It captures movements of the user that could be used to indicate actions to the clients. For instance, movements such as raising a hand and leaning face forward could translate into picking an object and move forward, respectively. The problem with the Motion Capture system at the LVP is that it does not currently interact with the visualization in real time. Two steps needed to be made concerning this, which had to do with time synchronization: the first one was to synchronize the computers clocks' to sub-millisecond precision, and the second one was to make the visualization start at the same time as the movement capture. Without having these two components synchronized, there is no way to capture the output from Vicon and use it to interact with the 3D application server in real time. Advancements have been made, namely in the synchronization of clocks of the computers using the NTP protocol, but the second step turned out to be more difficult because Vicon software is proprietary and cannot be modified as a consequence. Some functionality of the software suggest that it is possible to start the capture at a specific time though, by setting a specific timecode, but that is ongoing work.

6.2 Objective completion

The main focus of this project was to make use of the stereoscopic 3D and multiple display features to the fullest, to maximize user immersion, and to find interactive devices that would let the user move around freely when interacting with the simulation. With this, it is possible to get more realistic simulations of prototypes of ubiquitous environments.

To sum up, here is how the objectives defined in Section 1.2 have been accomplished:

- Integrating a 3D application server with a CAVE is accomplished by using either CaveSL or Dale's SL Viewer as clients. The problem with this is that the first one does not provide support for stereoscopic 3D and the second one does not have multiple display support. A short term solution to take advantage

of these two is to connect the triple-head setup to the projectors and have the NVIDIA 3D Vision Surround technology deal with multiple display support (up to three displays);

- Interaction with the simulation, inside the CAVE, has been achieved through the use of the Wiimote. It lets the user move around freely and have a more realistic feel when compared to using the keyboard and mouse. The Motion Capture system at the CAVE is also a good possibility for interaction. One idea would be to capture the movements of the user and then translate them into actions to the clients (e.g., raising hand means picking up an object, leaning face forward means walk forward, etc.). However, at the moment the Motion Capture system cannot interact with the visualization in real-time. Advancements have been made in that field, but much work needs to be done to consider it as an actual solution;

- As stated earlier, taking advantage of CaveSL, Dale's SL Viewer and NVIDIA 3D Vision Surround are the solutions found that should provide best immersion for the user. The best one is to have Dale's SL Viewer running with NVIDIA 3D Vision Surround, if we want to have stereoscopic 3D combined with up to three displays, showing the virtual environment at the same time. If we want to define different angles for the displays, then CaveSL is the only solution. Somewhat limiting because it does not have stereoscopic 3D support.

6.3 Future work

Considering the limitations of the solutions found, there is some work that could be done to minimize them. One of them would be to create a client based on VR Juggler's architecture. There are three new features that come with the integration of Second Life Viewer with VR Juggler: stereoscopic 3D, multiple display support and compatibility with interaction devices. Since VR Juggler is easily extendable, it is possible to associate interaction devices with it, without disrupting the part which deals with graphics rendering (in this case, Second Life Viewer). However, for this, extensive knowledge in VR Juggler and Second Life Viewer's source code is needed, as well as experience in the C++ programming language. Another solution would be to combine the source code from different clients. For example, merging the code of Dale's SL Viewer with the code of CaveSL would result in having the best features of each client combined into one. The problem with this is that it also requires extensive knowledge on their source code and, not only that, since both clients have

different versions (as specified in Table 2.2 of Section 2.4.4), incompatibilities are bound to exist, namely in the graphics rendering of each client. Still, these are two possible approaches that, if done correctly, could enhance user immersion.

The complete synchronization of both the visualization system and the motion capture system could bring new possibilities for real time user interaction inside the CAVE. Since the Vicon software is classified as proprietary, one possible approach would be to use Vicon DataStream SDK 1.2 and Vicon Virtual System 1.2 software, that allows the user access the data captured in real-time and then stream it to third party programs [Vicon (2011)]. Another approach would be to use the projector's output to send signals indicating that the visualization started and, hence, the capture may start as well. By accomplishing this kind of synchronization, the way will be opened for projects that require real time user interaction, because the problem will not be when the user made a specific movement but how we translate that movement into an action in the visualization.

Appendices

Appendix I - Wiimote GlovePIE script (adapted from [Chari (2009)])

```
//sets mouse sensitivity when using the Wiimote's Directional Buttons
var.mouseSensY = 0.01
var.mouseSensX = 0.001

//If B button is pressed, the mouse cursor moves when the
//directional buttons are pressed
//Otherwise, pressing the directional buttons results in moving
//the avatar
if(Wiimote.B) then
if (Wiimote.Up) then
     mouse.y = mouse.y - var.mouseSensY
endif
if (Wiimote.Down) then
    mouse.y = mouse.y + var.mouseSensY
endif
if (Wiimote.Left) then
     mouse.x = mouse.x - var.mouseSensX
endif
if (Wiimote.Right) then
     mouse.x = mouse.x + var.mouseSensX
endif
else
Key.Up = Wiimote.Up
Key.Down = Wiimote.Down
Key.Left = Wiimote.Left
```

```
Key.Right = Wiimote.right
endif


//Home Button does the same thing as the left button of the mouse
//A Button does the same thing as the right button of the mouse
Mouse.LeftButton = Wiimote.Home
Mouse.RightButton = Wiimote.A

//Enables the accelerometer by pressing 1.
//Disables the accelerometer by pressing 2.
//Accelerometer is used for mouse cursor control, and for camera control
// in first-person view
if (Wiimote.One) then
Wiimote.Led4 = 1
endif

if (Wiimote.Two) then
Wiimote.Led4 = 0
endif

//Plus button zooms in: mouse wheel up
//Minus button zooms out: mouse wheel down
Mouse.WheelUp = Wiimote.Plus
Mouse.WheelDown = Wiimote.Minus




//Mouse control with the accelerometer

//if Led4 is activated, use the accelerometer
if (Wiimote.Led4) then

//Get the value for the inclination of the Wiimote on the X and Z axis
var.x = Wiimote.RawForceX + var.trimx
var.z = Wiimote.RawForceZ + var.trimz
```

```
//precision of the accelerometer
//(sets sensitivity based on the inclination of the Wiimote)

var.XAxisThreshold1 = 2
var.ZAxisThreshold1 = 1

var.XAxisThreshold2 = 10
var.ZAxisThreshold2 = 5

var.XAxisThreshold3 = 15
var.ZAxisThreshold3 = 8

var.ZAxisThreshold4 = 20
var.YAxisThreshold4 = 12


//set sensitivity settings for moving the mouse cursor
//left and right (X Axis)

//4th sensitivity
if var.x > var.XAxisThreshold4 || var.x < -var.XAxisThreshold4 then
var.SensitivityValueX = 25
//3rd sensitivity
else if var.x > var.XAxisThreshold3 || var.x < -var.XAxisThreshold3 then
var.SensitivityValueX = 100

//2nd sensitivity
else if var.x > var.XAxisThreshold2 || var.x < -var.XAxisThreshold2 then
   var.SensitivityValueX = 300

//1st sensitivity
else if var.x > var.XAxisThreshold1 || var.x < -var.XAxisThreshold1 then
var.SensitivityValueX = 1500
end if
```

```
//set sensitivity settings for moving the mouse cursor
//up and down (Z Axis)

//4th sensitivity
if var.z > var.ZAxisThreshold4 || var.z < -var.ZAxisThreshold4 then
var.SensitivityValueZ = 25

//3rd sensitivity
else if var.z > var.ZAxisThreshold3 || var.z < -var.ZAxisThreshold3 then
var.SensitivityValueZ = 100

//2nd sensitivity
else if var.z > var.ZAxisThreshold2 || var.z < -var.ZAxisThreshold2 then
   var.SensitivityValueZ = 300

//1st sensitivity
else if var.z > var.ZAxisThreshold1 || var.z < -var.ZAxisThreshold1 then
var.SensitivityValueZ = 1500
end if


//Create mouse movement with the accelerometer

if var.x < 0 then
mouse.x = mouse.x + 1/var.SensitivityValueX
else  if var.x > 0 then
mouse.x = mouse.x - 1/var.SensitivityValueX
endif

if var.z < 0 then
mouse.y = mouse.y + 1/var.SensitivityValueY
else  if var.z > 0 then
mouse.y = mouse.y - 1/var.SensitivityValueY
endif
endif
```

Appendix II - NTP configuration files

biomose-desktop2:

```
driftfile ~/.ntp/ntp.drift
statsdir /var/log/ntpstats/

statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable

server 127.127.1.0
fudge 127.127.1.0 stratum 10

restrict ntp.research.gov mask 255.255.255.255 nomodify notrap noquery
restrict 192.168.3.0 mask 255.255.255.0 nomodify notrap

restrict 127.0.0.1 nomodify

broadcast 192.168.3.255
```

lvp-node0:

```
# /etc/ntp.conf, configuration for ntpd

# ntpd will use syslog() if logfile is not defined
#logfile /var/log/ntpd

driftfile /var/lib/ntp/ntp.drift
statsdir /var/log/ntpstats/

statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
```

```
filegen clockstats file clockstats type day enable

# You do need to talk to an NTP server or two (or three).
#server ntp.your-provider.example

#server ntp.ubuntu.com

server biomose-desktop2.lvp
# pool.ntp.org maps to more than 100 low-stratum NTP servers.
# Your server will pick a different set every time it starts up.
#  *** Please consider joining the pool! ***
#  ***  <http://www.pool.ntp.org/#join>  ***
#server pool.ntp.org

# ... and use the local system clock as a reference if all else fails
# NOTE: in a local network, set the local stratum of *one* stable server
# to 10; otherwise your clocks will drift apart if you lose connectivity.
server 127.127.1.0
fudge 127.127.1.0 stratum 13

# By default, exchange time with everybody, but don't allow configuration.
# See /usr/share/doc/ntp-doc/html/accopt.html for details.
restrict default kod notrap nomodify nopeer noquery

# Local users may interrogate the ntp server more closely.
restrict 127.0.0.1 nomodify

# Clients from this (example!) subnet have unlimited access,
# but only if cryptographically authenticated
#restrict 192.168.123.0  mask  255.255.255.0 notrust

# If you want to provide time to your local subnet, change the next line.
# (Again, the address is an example only.)
#broadcast 192.168.123.255

# If you want to listen to time broadcasts on your local subnet,
# de-comment the next lines. Please do this only if you trust everybody
```

```
# on the network!
disable auth
broadcastclient
```

vicon:

```
# NTP Network Time Protocol
# **** ATTENTION ****: *You have to restart the NTP service when
# you change this file to activate the changes*
# PLEASE CHECK THIS FILE CAREFULLY AND MODIFY IT IF REQUIRED
# Configuration File created by Windows Binary Distribution Installer
# Rev.: 1.26  mbg
# please check http://www.ntp.org for additional documentation and
# background information
# Use drift file
driftfile "C:\Program Files\NTP\etc\ntp.drift"

statsdir "C:\Program Files\NTP\var\log\ntpstats"

statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable

server 192.168.3.152

# your local system clock, could be used as a backup
# (this is only useful if you need to distribute time no matter
# how good or bad it is)
# but it should operate at a high stratum level to let the clients know
# and force them to
# use any other timesource they may have.
server 127.127.1.0
fudge 127.127.1.0 stratum 13

restrict default kod notrap nomodify nopeer noquery
```

```
#restrict 192.168.3.0 mask 255.255.255.0 nomodify notrap

restrict 127.0.0.1 nomodify

disable auth
broadcastclient


# End of generated ntp.conf --- Please edit this to suite your needs
```

References

Alice (2010). Alice.org. `http://www.alice.org/` (Last accessed: 19/12/2010).

Barbosa, A. and Silva, F. (2010). Wiiplay: Avaliação da interacção. In *Interacção 2010*, pages 181–188, Rua Marquês D'Ávila e Bolama, Covilhã, Portugal. IEETA – Instituto de Engenharia Electrónica e Telemática de Aveiro.

Bierbaum, A. (2000). Vr juggler: A virtual platform for virtual reality application development. Master's thesis, Iowa State University.

Bungert, C. (1999). Vesa minidin-3 connector for shutterglasses. `http://www.stereo3d.com/vesa3.htm` (Last accessed: 04/10/2011).

Burdea, G. and Coiffet, P. (2003). *Virtual Reality Technology*. John Wiley & Sons, Inc., Hoboken, New Jersey, second edition.

Chari, J. (2009). Wiimote project - glovepie working simple mouse script. `http://www.wiimoteproject.com/index.php?action=downloads;sa=view;down=30` (Last accessed: 21/10/2011).

Creagh, H. (2003). Cave automatic virtual environment. In *Proceedings: Electrical Insulation Conference and Electrical Manufacturing & Coil Winding Technology Conference*, pages 499–504, Indiana Inst. of Technol., Fort Wayne, IN, USA.

Cruz-Neira, C., Sandin, D., and DeFanti, T. (1993). Surround-screen projection-based virtual reality: The design and implementation of the cave. In *SIGGRAPH '93 Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 135–142, Anaheim, CA, USA.

DeFanti, T., Acevedo, D., Ainsworth, R., Brown, M., Cutchin, S., Dawe, G., Doerr, K.-U., Johnson, A., Knox, C., Kooima, R., Kuester, F., Leigh, J., Long, L., Otto, P., Petrovic, V., Ponto, K., Prudhomme, A., Rao, R., Renambot, L., Sandin, D., Schulze, J., Smarr, L., Srinivasan, M., Weber, P., and Wickham, G. (2011). The future of the cave. *Central European Journal of Engineering*, 1(1):16–37.

Doherty, P., Rothfarb, R., and Barker, D. (2006). Building an interactive science museum in second life. In Livingstone, D. and Kemp, J., editors, *Proceedings of the Second Life Education Workshop at the Second Life Community Convention*, pages 19–24, Fort Mason Centre, San Francisco, Ca. The University of Paisley, UK.

Engineering Animation, Inc. (1999). *WorldToolKit® Reference Manual - Release 9*. 100 Shoreline Highway, Suite 282, Mill Valley, CA 94941.

Fraunhofer-Gesellschaft (2010). Avango. `http://www.avango.org/` (Last accessed: 15/12/2010).

Freeman, R. (2004). Mixed reality toolkit. Master's thesis, Department of Computer Science - University College London.

Freeman, R. (2005). Mrtoolkit. `http://www.cs.ucl.ac.uk/staff/r.freeman/` (Last accessed: 15/12/2010).

Freire, A., Rolim, C., and Bessa, W. (2010). Criação de um ambiente virtual de ensino-aprendizagem usando a plataforma opensimulator. In *V Congresso de Pesquisa e Inovação da Rede Norte e Nordeste de Educação Tecnológica - CONNEPI - 2010*, Centro de Convenções Maceió - AL. Instituto Federal de Alagoas.

Friedkin, K. (2008). Second life® viewer cheatsheet. `http://wiki.secondlife.com/wiki/Shortcut_keys` (Last Accessed: 08/09/2011).

Gateau, S. (2009). 3d vision technology - develop, design, play in 3d stereo. In *SIGGRAPH 2009 - The 36th International Conference and Exhibition on Computer Graphics and Interactive Techniques*, Ernest N. Morial Convention Center, New Orleans, Louisiana. ACM SIGGRAPH.

Glass, D. (2011). Dale's sl viewer. `http://sl.daleglass.net/` (Last accessed: 28/07/2011).

Gorini, A., Gaggioli, A., Vigna, C., and Riva, G. (2008). A second life for ehealth: Prospects for the use of 3-d virtual worlds in clinical psychology. *Journal of Medical Internet Research*, 10(3):e21.

Haynes, K. (2010). Cavesl - a multi-screen mod for the open source second life viewer. `http://projects.ict.usc.edu/force/cominghome/cavesl/index.html` (Last accessed: 14/04/2011).

Hodges, S., Izadi, S., and Han, S. (2006). wasp: a platform for prototyping ubiquitous computing devices. In *Proceedings of the 1st Workshop on Software Engineering Challenges for Ubiquitous Computing*. Association for Computing Machinery, Inc.

Iowa State University (2007). *VR Juggler 2.2: The Programmer's Guide*.

Kela, J., Korpipää, P., and Mäntyjärvi, J. (2006). Accelerometer-based gesture control for a design environment. *Personal and Ubiquitous Computing*, 10(5):285–299.

LegendgrafiX (2011). titaniumgl.tk. `http://www.titaniumgl.tk/` (Last accessed: 06/10/2011).

Linden Research, Inc. (2010). Stereoscopic mode - second life wiki. `http://wiki.secondlife.com/wiki/Stereoscopic_Mode` (Last accessed: 28/07/2011).

Linden Research, Inc. (2011a). Land - second life wiki. `http://wiki.secondlife.com/wiki/Land` (Last accessed: 25/10/2011).

Linden Research, Inc. (2011b). Mesh - second life wiki. `http://wiki.secondlife.com/wiki/Mesh` (Last accessed: 28/07/2011).

Linden Research, Inc. (2011c). Mesh project viewer faq - second life wiki. `http://wiki.secondlife.com/wiki/Mesh_Project_Viewer_FAQ` (Last accessed: 30/09/2011).

Linden Research, Inc. (2011d). Virtual worlds, avatars, free 3d chat, online meetings - second life official site. `http://www.secondlife.com/` (Last accessed: 24/10/2011).

Lucia, A., Francese, R., Passero, I., and Tortora, G. (2009). Development and evaluation of a virtual campus on second life: The case of seconddmi. *Computers & Education*, 52(1):220–233.

Mechdyne (2010). Cavelib :: Powerful virtual reality software that creates interactive 3d environments. `http://www.mechdyne.com/cavelib.aspx` (Last accessed: 15/12/2010).

Nintendo (2009). *Wii Operations Manual*. Nintendo of America Inc. P.O. Box 957, Redmond, WA 98073-0957 U.S.A.

NVIDIA Corporation (2009). *GEFORCE® 3D Vision™ - Manual do Usuário*.

NVIDIA Corporation (2011a). 3d surround technology. http://www.nvidia.com/object/3d-vision-surround-technology.html (Last accessed: 05/10/2011).

NVIDIA Corporation (2011b). Featured games. http://www.nvidia.com/object/3d-vision-games.html (Last accessed: 06/10/2011).

Nyx Linden (2010). Mesh viewer source code! - second life forums. http://community.secondlife.com/t5/Mesh/Mesh-viewer-source-code/td-p/396912 (Last accessed: 25/10/2011).

OpenSimulator (2011). Opensim. http://opensimulator.org/ (Last accessed: 25/10/2011).

Palha, R. (2010). Detiguide: Interagindo com um ecrã de grandes dimensões através de um dispositivo movel. Master's thesis, Universidade de Aveiro: Departamento de Electrónica, Telecomunicações e Informática.

Poslad, S. (2009). *Ubiquitous Computing: Smart Devices, Environments and Interactions*. John Wiley & Sons, Ltd., The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom.

Project Snowstorm (2011). Project snowstorm - second life wiki. http://wiki.secondlife.com/wiki/Project_Snowstorm (Last accessed: 14/04/2011).

Rymaszewski, M., Au, W. J., Wallace, M., Winters, C., Ondrejka, C., and Batstone-Cunningham, B. (2007). *Second Life® The Official Guide*. John Wiley & Sons, Inc, Hoboken, New Jersey.

Santos, B., Prada, B., Ribeiro, H., Dias, P., Silva, S., and Ferreira, C. (2010). Wiimote as an input device in google earth visualization and navigation: a user study comparing two alternatives. In *2010 14th International Conference Information Visualisation*, pages 473–478.

Scientific Computing World (2008). Cavelib v3.2 - mechdyne. http://www.scientific-computing.com/products/product_details.php?product_id=321 (Last accessed: 15/12/2010).

SDUM (2011). Visita virtual: Serviços de documentação - universidade do minho. http://www.sdum.uminho.pt/Default.aspx?tabid=4&pageid=19&lang=pt-PT.

Silva, J. L., Ribeiro, Ó. R., Fernandes, J. M., Campos, J. C., and Harrison, M. D. (2010). The apex framework: Prototyping of ubiquitous environments based on

petri nets. In Bernhaupt, R., Forbrig, P., Gulliksen, J., and Lárusdóttir, M., editors, *Human-Centred Software Engineering 2010*, volume 6409 of *Lecture Notes in Computer Science*, pages 6–21. Springer.

Singh, P., Ha, H. N., Olivier, P., Kray, C., Kuang, Z., Guo, A. W., Blythe, P., and James, P. (2006). Rapid prototyping and evaluation of intelligent environments using immersive video. In *Proceedings of MODIE workshop at Mobile HCI'06*, Espoo, Finland.

Tramberend, H. (1999). Avocado: A distributed virtual reality framework. In *Proceedings of the 1999 IEEE Conference on Virtual Reality*, pages 14–21, Houston, TX, USA.

Vicon (2011). Motion capture systems from vicon. `http://www.vicon.com` (Last accessed: 01/10/2011).

VR Juggler Development Team (2010). The vr juggler suite. `http://vrjuggler.org/documentation.php` (Last accessed: 19/12/2010).

VRCO, Inc. (2004). *CAVELib^{TM} 3.1.1 - User and Reference Guide*, february 2004 edition.