An Open Source Framework for CAVE Automatic Virtual Environments

Carl Flynn, B.A.

A Thesis Submitted for the Degree of
Master of Engineering

Dublin City University
School of Electronic Engineering

Supervisors:
Prof. Noel E. O'Connor
Dr. Hyowon Lee

September 2014

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Master of Engineering is entirely my own work, and that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: _____ ID No.: 10113401 Date: _____

Acknowledgements

I would like to give special thanks to my supervisors Prof. Noel O'Connor and Dr. Hyowon Lee. I am deeply grateful to both of them for their constant support and encouragement and for making my time at DCU such an enjoyable and truly worthwhile experience. The 'Dream Team' of supervisors if ever there was such a thing.

Thanks to my colleagues Andrew, Julie and Ron. They may not have realised it but their support made a big difference.

Thanks also to the participants who took part in the experiments. The spirit of generosity is alive and well.

Finally, a big thank you to all my family for all their love and support. In particular my wife Agnese who has had to put up with so much over the last few years but who never complained and never showed anything other than love and encouragement. Ti voglio bene Pico!

Contents

Abstract

A CAVE Automatic Virtual Environment (CAVE) is a 3D interactive environment that enables a user to be fully immersed in a virtual world and offers a unique way to visualise and interact with digital information.

The primary goal of this thesis is to report on the development of a new open source CAVE software framework that provides greater access to both professional and amateur CAVE environments to users of all levels.

In the first part of this thesis, the history and evolution of virtual environments as well as the generic affordances of the modern day CAVE are characterised. This is followed by a detailed discussion of the factors that influence immersion, the different methods and devices of interaction, and a small project to develop a CAVE specific interaction device is described.

In the second part of this thesis, the focus is on novel work to develop a new open source platform for CAVE environments and the implementation of this platform in the visualisation of real-world sensor data. This data is collected from a range of residential and educational buildings across a local community and is used in support of an ongoing 'smart energy' project which is also described in detail.

In conclusion, this thesis argues that through the development of new, easy-to-use, open source software and the ongoing reduction in key hardware technology costs, CAVE environments can be an effective and affordable visualisation tool for both experienced and novice users. The CAVE environment need no longer remain the sole preserve of well-funded educational and business organisations. Rather, through technology innovations such as proposed in this work, the era of the much vaunted low-cost CAVE is 'virtually' upon us.

List of Figures

List of Tables

List of Publications

- Carl Flynn, David Monaghan, and Noel E. O'Connor. "SCReen Adjusted Panoramic Effect: SCRAPE." In Proceedings of the 21st ACM International Conference on Multimedia[1], pp. 859-862. ACM, Barcelona, Spain, 21-25 Oct, 2013.
- Carl Flynn, Noel E. O'Connor, Hyowon Lee and John Loane. "Visualising Better Energy Communities in a CAVE Automatic Virtual Environment." The 7th Annual Irish Human Computer Interaction Conference, Dundalk, Ireland, 12-13 Jun, 2013.
- Carl Flynn, Hyowon Lee and Noel E. O'Connor. "Visualising and Interacting with a CAVE using Real-World Sensor Data." The 5th Annual Irish Human Computer Interaction Conference, Cork, Ireland, 8-9 Sep, 2011.

[1]Peer reviewed and accepted for publication as part of the annual ACM MM Open Source competition

List of Acronyms

3D Three Dimensional

ACM Association for Computing Machinery

API Application Programming Interface

BER Building Energy Rating

CASALA Centre for Affective Solutions for Ambient Living Awareness

CAVE CAVE Automatic Virtual Environment

CAVELib CAVE Library

CEMS Community Energy Monitoring System

DkIT Dundalk Institute of Technology

DOF Degrees Of Freedom

D-pad Directional pad

EVL Electronic Visualization Laboratory

FOR Field Of Regard

FOV Field Of View

GNH Great Northern Haven

HCI Human Computer Interaction

HMD Head Mounted Display

I2C Inter-Integrated Circuit

IDE Integrated Development Environment

IR Infrared

ITQ Immersive Tendencies Questionnaire

MIT Massachusetts Institute of Technology

MPE Most Pixels Ever

NPV Net Present Value

OCD Obsessive Camera Direction

P5 Processing

PDE Processing Development Environment

PQ Presence Questionnaire

SCRAPE SCReen Adjusted Panoramic Effect

SEAI Sustainable Energy Authority of Ireland

SEED Sustainable Energy Efficiency Dundalk

SIGGRAPH Special Interest Group on GRAPHics and Interactive Techniques

SUS System Usability Scale

VE Virtual Environment

VIEW Virtual Interactive Environment Workstation

VR Virtual Reality

VRAC Virtual Reality Applications Center

VW Virtual World

Chapter 1

Introduction

Three Dimensional (3D) technology has been with us in some form or another ever since a stereographic photograph of Queen Victoria first appeared at The Great Exhibition in London in 1851. More than a century and a half later, we continue to be fascinated by 3D. From photographs, to film, to television and video games, 3D delights and disappoints in equal measure. Our love-hate relationship with 3D seems to follow a decennial cycle where it is continually repackaged alongside the latest media technology only to slowly languish before its next reincarnation. One thing remains steadfast however, for over 150 years we have continued to be interested in the concept of being able to visualise images (both still and animated) in a way that replicates our natural ability to view the world. One viewpoint is that 3D technology is nothing more than a novelty or toy that we repeatedly tire of. Whatever the verity of this view, it is unlikely that our 150 year love affair with 3D technology will end anytime soon. Without question, 3D technology will continue to be developed and improved upon until such a time that we may even find it difficult to differentiate between the real and the virtual. While that vision may be some way off, today's 3D technology already has a lot to offer. This thesis discusses how existing 3D technology is already being put to good use. In particular, it investigates a specific piece of 3D technology known as a *CAVE Automatic Virtual Environment* (CAVE) and how the development of a new open source CAVE framework (in combination with reduced hardware costs) aims to assist in the democratisation of once costly and cumbersome 3D CAVE systems.

A 'typical' CAVE can essentially be described as a cube shaped room which offers a multi-person, multi-screen, high-resolution 3D video and audio interactive environment; a full characterisation of which is provided in Chapter 2.

1.1 Motivation

The motivation for this work came about primarily as a result of the author's work with the Centre for Affective Solutions for Ambient Living Awareness (CASALA), a research organisation based at the Dundalk Institute of Technology (DkIT). One of CASALA's principal goals is to investigate how different technologies can help in our understanding of people's daily lives. This ranges from projects that explore how sensor technologies can enhance older people's sense of well-being and independence, to projects that examine how sensor technologies can improve our understanding of energy use across homes, businesses and institutions. As part of this research, CASALA uses a state of the art 3D CAVE to visualise and interact with realistic and abstract 3D worlds using the vast quantities of data collected from the different projects. Over a period of four years, many different CAVE software applications were used to develop these Virtual Worlds (VWs), however, the high cost and/or inaccessibility of most remained a constant source of frustration. At the same time, many of the traditionally costly hardware elements that are essential to a CAVE were becoming far more affordable. All these factors combined to provide substantial motivation for the development of a new software framework which could contribute to the creation of the next generation of affordable and accessible CAVE systems.

1.2 Research Contributions

There are three key contributions which this thesis aims to provide:

1. A comprehensive documentation of 3D technologies and the key factors that need to be considered in relation to CAVE environments.

2. The development of a new open source software framework to assist in making CAVE platforms more accessible.

3. A demonstration of the benefit of the proposed open source framework. Specifically, as an illustration of its application, it is used as the technological platform for a comparative user evaluation to test the effectiveness of a CAVE in a range of tasks in comparison to a traditional desktop system. Such tasks and experiments are commonplace in data visualisation research.

The thesis also provides an additional minor contribution through the customisation of a unique interaction device for CAVE interactions.

1.3 Structure

This thesis is structured as follows: Chapter 2 provides a history of the birth and development of 3D technologies over the past half century, as well as definitions of common terms and a full characterisation of an existing CAVE. Chapter 3 discusses the importance of immersion in a CAVE and the elements that combine to augment it. Chapter 4 highlights the key considerations in relation to choosing the optimum interaction modalities; it also provides a comprehensive taxonomy of interaction devices and presents a small project that integrates a new type of controller device into a CAVE. Chapter 5 takes a brief look at some key CAVE frameworks and highlights their current deficiencies, before leading on to Chapter 6 which describes in detail a project to develop an entirely new CAVE framework. Chapter 7 describes a project that is collating and assessing real-world data gathered across a local community and then demonstrates how this data is being applied and visualised in a CAVE. Chapter 8 provides details of a user evaluation experiment that compares a CAVE's performance and usability in comparison to a traditional desktop system. Finally, Chapter 9 provides a brief conclusion of the work and results obtained as well as a description of potential future work.

Chapter 2

Virtual Environments: Overview

2.1 A Brief History of ...

In 1961, the philosopher, filmmaker and inventor, Morton Heilig filed a patent for a device called the *Sensorama Simulator*[1] (see Figure 2.1). The device was very similar in appearance to the video arcade cabinets that would be commonplace several decades later and its purpose was to take users into 'another world' by means of its multi-sensory capabilities. The user sat in the machine, placed their arms on an armrest in front of them and leaned forward, with their eyes peering into a viewer. They were then presented with a stereoscopic 3D film which took them on a pre-determined journey such as a motorbike ride through Brooklyn in New York. As they were taken on this 3D journey they would experience the sensation of a drive in the city through movement and vibrations in their seat and armrest. Even the sounds, breezes and smells of the city were conveyed to the user through the use of speakers, fans and perfume vessels contained within the unit itself. It was hugely innovative for the time but interest from the public and investors was lacking and it never took off.

Despite *Sensorama's* commercial failure, Morton Heilig's invention is considered of historical importance and Heilig himself is often credited with having given birth to the idea of Virtual Reality (VR). This reputation is further enhanced by some of his other work, most notably his *Telesphere Mask* (see Figure 2.2) which he filed as a patent as early as 1957[2]. The *Telesphere Mask* was designed to present stereo images, sound and simulate gusts of wind through a device that was worn on the head and is the first known example of the Head Mounted Display (HMD). HMDs are in common use in many of today's VR systems and are once again 'in vogue' with the recent

[1] http://www.mortonheilig.com/SensoramaPatent.pdf
[2] http://www.mortonheilig.com/TelesphereMask.pdf

Figure 2.1: The Sensorama Simulator
Source: (Mortonheilig.com, 2011a)

development of a low-cost, wide Field of View (FOV) device known as the Oculus Rift[3]. The *Telesphere Mask* itself, however, never made it into commercial production and only a prototype was ever built.

Despite the commercial failures, there is no doubting that Morton Heilig's work was hugely important in the formation of VR as we know it today. Although recognition was slow to arrive (His seminal paper entitled 'Cinema Of The Future' (Heilig, 1992) was only first published in English in the Massachusetts Institute of Technology (MIT) Presence journal in 1992, some thirty seven years after he first wrote it), he is now considered by many people to be the father of VR.

In 1961, Charles Comeau and James Bryan working for the Philco Corporation built the first fully manufactured HMD called *Headsight*. Unlike Morton Heilig's *Telesphere Mask* prototype, the Philco device only used a single CRT image display but it did use a magnetic tracking system that was linked to a remote camera (see Figure 2.3). The camera would adjust its viewing position based on the user's head movements and relay the remote image back to the HMD (a type of set-up that can be referred to as telepresence). The system was built with a view to being used as a security surveillance system and soon after, Comeau and Bryan set up Telefactor Corporation to work on products based on their telepresence research.

The 1960's was an important time in the birth of VR and no one contributed more to

[3]http://www.oculusvr.com/

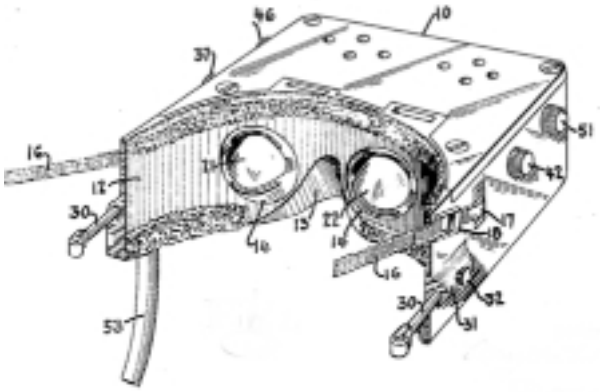


Figure 2.2: The Telesphere Mask
Source: (Mortonheilig.com, 2011b)

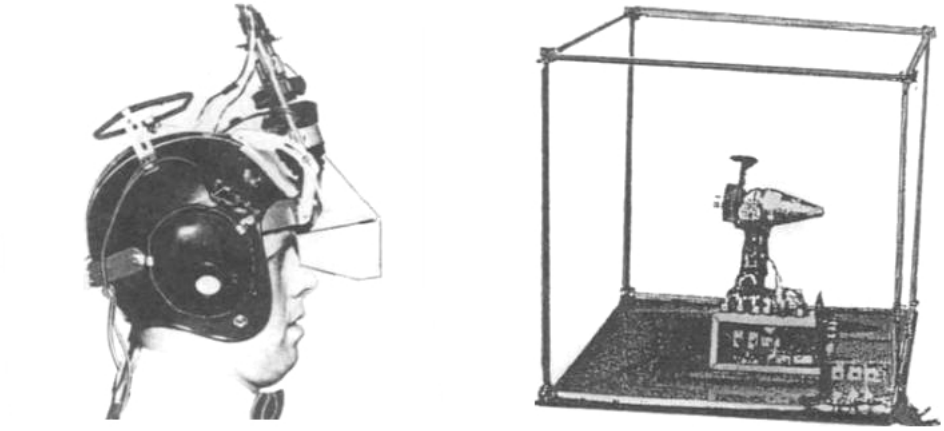


Figure 2.3: The Headsight HMD (left) and linked remote TV camera (right)
Source: (Ellis, 1994)

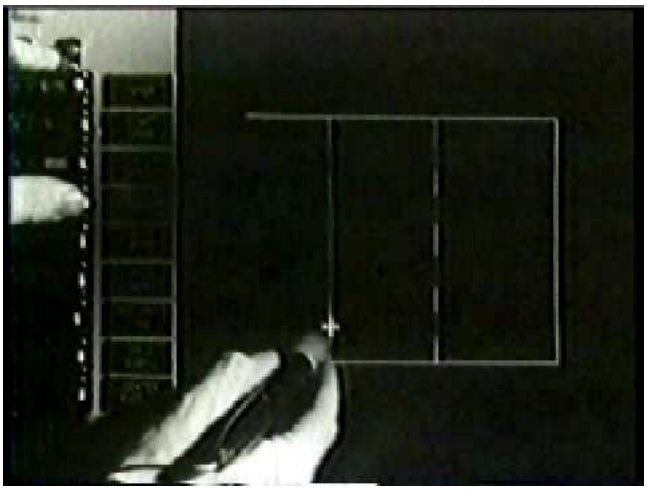


Figure 2.4: The Sketchpad System
Source: (Kay, 2011)

that early development than Ivan Sutherland. Before Morton Heilig's work became more widely recognised, Ivan Sutherland was generally considered the father of VR (and to many he still is). Ivan Sutherland is both an engineer and academic and he was particularly interested in computer graphics and interaction at a time when both were virtually non-existent. His creation of the *Sketchpad* system (see Figure 2.4) in 1963 allowed users to draw directly on to a computer monitor using a light pen. It was revolutionary and was the precursor to the ubiquitous GUI systems that we use today.

In 1965 Ivan Sutherland wrote a paper entitled 'The Ultimate Display' (Sutherland, 1965) in which he laid out a vision for the future of Human Computer Interaction (HCI). It was a truly prophetic piece of work for its time, describing many of the possible future uses of technology with surprising accuracy. In the final paragraph he provided us with one of his most celebrated quotes in which he summed up the Ultimate Display as:

> *"a room within which the computer can control the existence of matter. A chair displayed in such a room would be good enough to sit in. Handcuffs displayed in such a room would be confining, and a bullet displayed in such a room would be fatal. With appropriate programming such a display could literally be the Wonderland into which Alice walked."*

In 1968 he followed through on many of the visionary ideas set out in 'The Ultimate Display' by creating the world's first real-time computer generated HMD. The device was known as the *Sword of Damocles* due to the fact that the HMD unit was suspended from the ceiling by a metal frame (a reference to the Greek myth where Damocles was seated on the Kings throne with a sword hung by a single hair over his head). The device was crude by today's standards, showing only basic wireframe overlays to the

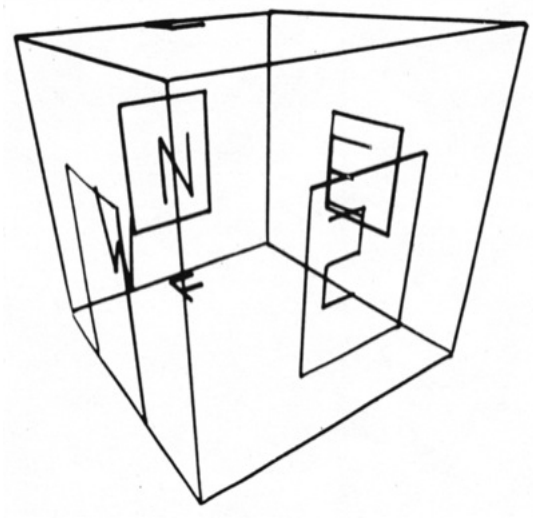


Figure 2.5: Ivan Sutherland's wireframe room as it looked from the outside
Source: (Sutherland, 1968)

wearer (see Figure 2.5) but it did provide some of the basic elements which still exist in today's HMDs such as separate image views to each eye and head tracking to enhance the sense of immersion.

In many respects Ivan Sutherland is more famous for his *Sketchpad* system than his work on VR but this just highlights the incredible contributions made over a long and distinguished career. This was recognised in 1988 when Sutherland was awarded the Turing award for his contributions to the computing community by the Association for Computing Machinery (ACM).

In 1974, the computer scientist and artist, Myron Kreuger created a truly unique interactive environment called *Videoplace* (Krueger et al., 1985). By using a camera to display a high contrast image of a user onto a projected screen, the user could interact with computer generated images in real-time. His computing system was able to recognise and analyse the silhouetted images and allow them to interact directly with the computer generated images (see Figure 2.6). Kreuger's work was important in terms of advancing HCI and his work directly influenced many of the interaction devices and methods available today, not only for VR, but for all forms of HCI. Modern day gaming devices such as the Sony EyeToy or the Microsoft Kinect offer experiences very similar to those demonstrated by *Videoplace* nearly forty years earlier.

Another significant event in the development of VR and how humans interact with VWs was the creation of the dataglove. A dataglove quite simply is a glove that allows the actions made by the hand and fingers to be translated and input into a computer (see Figure 2.7). In 1977 Thomas Defanti and Daniel Sandin from the Electronic

Figure 2.6: Videoplace - A human silhouette interacting with a graphical object
Source: (Krueger et al., 1985)

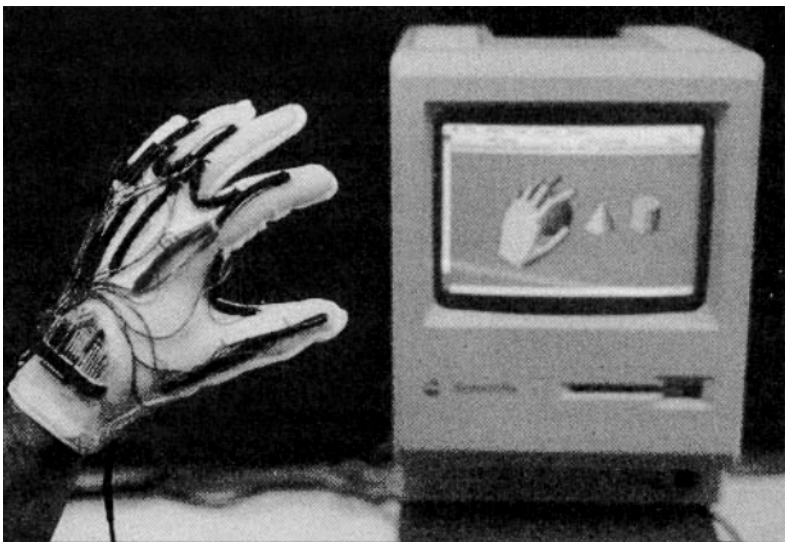


Figure 2.7: Dataglove
Source: (Zimmerman et al., 1987)

Visualization Laboratory[4] (EVL) based at the University of Illinois at Chicago created the world's first dataglove called the *Sayre Glove*. It was named after their colleague Rich Sayre who had come up with the original idea. The glove was lightweight and inexpensive and was able to detect hand movements through the use of flexible tubes, light sources and photocells on the ends of each tube. As the user bent their fingers the amount of light reaching the photocells would decrease, providing a control device similar to a set of sliders.

In 1980, MIT developed the *MIT LED Glove*. LEDs were placed at different points on the glove and then cameras were used to detect the hand and finger motion. It was an interesting project but it was used for motion capture rather than as a control device

[4]http://www.evl.uic.edu

and was never fully developed.

In 1983, Gary Grimes, an engineer at Bell Laboratories developed the *Digital Data Entry Glove*. Touch sensors were embedded strategically in the glove in order to be able to detect eighty unique combinations of sensor readings and it was designed specifically to recognise the single hand manual alphabet for the deaf. Once again, it was not fully developed but it was another step in the evolution of the dataglove.

Finally in 1987, Thomas Zimmerman an inventor and researcher developed a dataglove which was given the (not so original) name *DataGlove*. Zimmerman's *DataGlove* (Zimmerman et al., 1987) was able to monitor ten finger joints as well as the hand's position and orientation in real-time. It was also lightweight, comfortable to wear and relatively inexpensive. All these elements combined to make Zimmerman's device the first truly complete interactive glove and was the first example of a commercially successful dataglove. There have been many different types of glove developed since then, but the work started in 1977 by Defanti and Sandin leading up to Zimmerman's *DataGlove* ten years later, laid an important foundation and helped inspire the many forms of interaction that users have with Virtual Environments (VEs) today.

Around the same time that all this work was going on in relation to datagloves, an interaction designer named Scott Fisher working in NASA's Ames Research Centre was developing the *Virtual Interactive Environment Workstation (VIEW)* (Fisher et al., 1987). *VIEW*, which was developed in 1987, was certainly not the first VR system but it was important because it is generally considered to be the first affordable, fully integrated VR system. It used a HMD with LCD displays to provide a high quality image with a wide FOV to its users. It was relatively light and comfortable in comparison to earlier HMDs. It incorporated real-time head tracking and also integrated the recently developed *DataGlove* to provide for a truly interactive experience. It is this combination of HMD and Dataglove such as the one developed by Scott Fisher that conjures up the classic image that many of us have of what constitutes a VR system (see Figure 2.8)

Another important name in the history of VR is Jaron Lanier. A multi-talented computer scientist, composer, musician, writer and artist, Lanier is probably most famous for having coined the term 'Virtual Reality' in the 1980's. In 1984 Lanier set up a company called VPL Research to develop and sell VR products. These included a HMD called the *EyePhone*, a visual programming language called *Body Electric*, a 3D real-time render engine called *Isaac* as well as Thomas Zimmerman's *DataGlove*. Together these devices could be integrated to create the full VR experience and it was the first commercial example of such a tightly integrated VR system (see Figure 2.9).

Figure 2.8: VIEW at NASA's Ames Research Centre
Source: (Fisher et al., 1987)

Figure 2.9: VPL Research VR Kit
Source: (Pape, 2011)

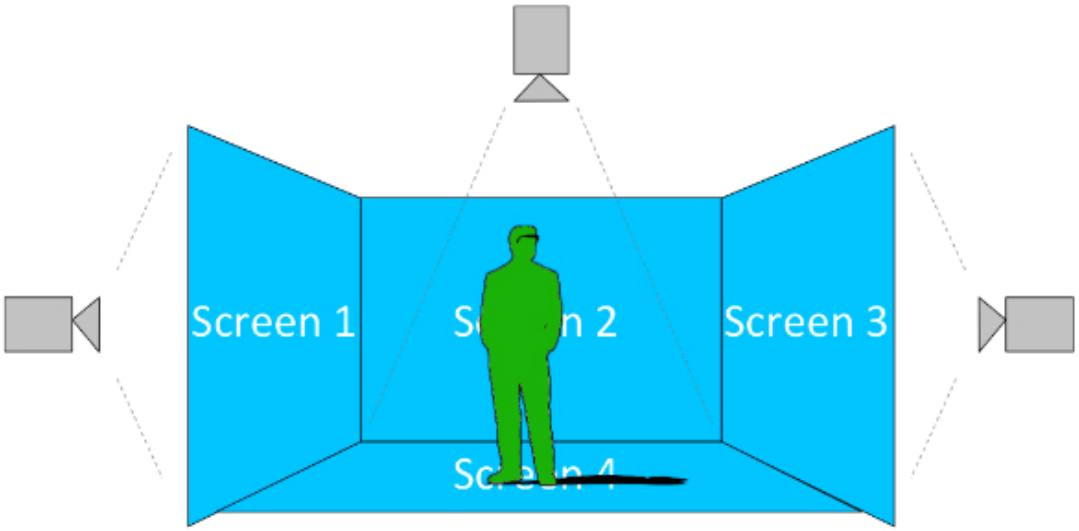


Figure 2.10: A standard 4 wall CAVE configuration

In 1992, Carolina Cruz-Neira, Daniel Sandin and Thomas DeFanti, from the EVL research lab at the University of Illinois at Chicago created the world's first CAVE (Cruz-Neira et al., 1992). A typical CAVE consists of four large rear projected screens placed around the user to form an incomplete cube (see Figure 2.10). Stereoscopic projectors present the user with stereographic images and the user wears 3D glasses in order to correctly view them. Interaction takes place through the use of glove, wand or joystick type controllers and immersion is enhanced through the use of head tracking. Multiple speakers are also placed around the CAVE to provide audio from various directional sources.

The unusually named CAVE, which is short for *CAVE Automatic Virtual Environment*, is a recursive acronym that refers back to Plato's fable *The Republic*. In this fable, Socrates describes how prisoners locked up and restrained in a CAVE since childhood, could perceive shadows cast on the CAVE's walls to be real things and not just reflections of a reality that they are unable to see due to the conditions of their restraint.

The CAVE was a major step away from the typical HMD configuration used for VR. It allowed for a much wider FOV for users, freed the user from wearing a cumbersome HMD and also allowed the user to interact with the VW while still being able to fully contextualise themselves within it. This is unlike most HMD environments that either separate the user from their body or need to generate virtual reconstructions of themselves. The first CAVE was presented at the ACM SIGGRAPH (Special Interest Group on GRAPHics and Interactive Techniques) conference in Chicago in 1992 and the fundamental design has changed very little since then. It could also be argued that the advent of the CAVE instigated the use of the term 'Virtual Environment'. The term

'Virtual Reality' was closely associated with the classic images of legacy HMD systems but the term 'Virtual Environment' was one which could comfortably encompass both types of systems. Cruz-Neira, Sandin and DeFanti have continued to contribute significantly to the development of CAVE systems as well as many other VE/VR related technologies and have already secured their places as true pioneers in the field.

There have been many other contributions in the birth and development of VEs: people such as Edwin Link who developed the first interactive flight trainers used in the 1930's, Frederick Brooks who started the GROPE (Brooks et al., 1990) project in 1967 to develop haptic feedback interfaces and Michael McGreevy from NASA's Ames Research Centre who probably deserves as much recognition as Scott Fisher for his work on VR. More recent work such as the LAIR (Denby et al., 2009) project by Barry Denby, Abraham Campbell, Hamish Carr and Greg O'Hare also provide an important evolutionary step in making VEs more accessible to the mainstream. The list of people goes on and one could argue over the omission of certain names, however, despite this, there can be no doubt that the majority of names highlighted thus far can be rightly considered outstanding pioneers of VEs.

2.2 Definitions

Terms such as 'Virtual Environment' (VE), 'Virtual Reality' (VR) and 'Virtual World' (VW) are used throughout this thesis and the peculiarities and differences between them is not always clear. There are a multitude of different explanations or definitions available in the literature, many of which seem either at odds with each other or appear to be direct translations of the same thing depending on the source (and to some extent the time frame within which they were defined). It therefore seems appropriate to offer a clear definition for each one for the sake of clarity within this document. Although one could argue over the merits of the provided definitions, for the purposes of this thesis, the terms are defined as specified in the following subsections. These definitions are either taken from a generally accepted interpretation of a term, or, an entirely new definition offered by the author.

2.2.1 Virtual Environment

The first term to be defined is 'Virtual Environment'. It is a widely used term but probably one of the least clearly defined. For the purpose of this thesis the following definition is proposed by the author:

A Virtual Environment is a collection of objects, technologies and spaces that when used together both on and off the human body, present a three dimensional computer generated artificial environment which a user can view, sense and with which a user can interact.

2.2.2 Virtual Reality

When Jaron Lanier coined the term 'Virtual Reality' it was around the same time that his company, VPL Research, started to manufacture and sell VR products, which were typically in the form of HMDs and glove controllers. It is probably for this reason, that for many people, the term 'Virtual Reality' conjures up the image of VPL Research's HMD products even though the definition which follows is very similar to the one defined for VEs:

The computer-generated simulation of a three-dimensional image or environment that can be interacted with in a seemingly real or physical way by a person using special electronic equipment, such as a helmet with a screen inside or gloves fitted with sensors[5].

In essence the only real difference between VR and VEs is that VR is typically associated with the interaction of immersive VWs through HMD type systems, whereas, VEs tend to incorporate not only the HMD systems but any other systems in which people interact with immersive VWs (such as a CAVE). These terms are also often used in a context which has no link to any type of hardware or software technologies but simply to refer to a 3D world, however, this is not how they will be used in this thesis and instead the term 'Virtual World' will be used to refer to a computer generated 3D world, a definition of which is proposed next.

2.2.3 Virtual World

The term 'Virtual World' is used in many different contexts from VR to websites to video games to social networks. Unfortunately, a definition which incorporates all these elements does not necessarily satisfy the requirements for a definition of a VW in the context of this thesis which focuses primarily on VEs. For that reason the following simple definition is proposed by the author:

A concrete or abstract computer generated synthetic environment which people can visualise and with which they can interact.

[5]Source: Oxford Dictionary

Figure 2.11: The CAVE at CASALA, a technology research centre at the Dundalk Institute of Technology

2.3 CAVE Focus

Having provided a brief overview of VEs and defined various related terms, the focus of this thesis now turns specifically to CAVE VEs. For the purpose of this thesis, many of the examples and experiments in relation to CAVEs are carried out using the CASALA CAVE at DkIT, a characterisation of which is provided next.

2.4 CASALA CAVE Characterisation

The CASALA CAVE is a relatively standard configuration in terms of overall CAVE implementations (Cruz-Neira et al., 1993). It consists of a four wall rear-projection screen configuration with each screen measuring approximately 2.0 metres by 2.6 metres (see Figure 2.11). Three vertical screens are each allocated their own 120hz 3D stereoscopic projector which is placed approximately ten metres behind the CAVE on a purpose built stand which is approximately 2.5 metres in height. The fourth floor screen is also allocated a 120hz 3D stereoscopic projector, however, it is attached directly to the CAVE frame and makes use of a ceiling mirror to project images on to the floor. The CAVE also consists of five workstations and one standard PC which are connected via CAT5e ethernet cables and a Cisco gigabit switch. The five workstations are dedicated to generating and managing the VW (four slave workstations for each screen and one master workstation to act as systems manager). Each workstation runs on Windows XP x64, uses an Intel Xeon quad core processor, 12GB of RAM, a terrabyte of disk space and an NVIDIA Quadro graphics card, all of which assists in the visualisation of highly complex 3D worlds across the distributed framework. The

standard PC then controls the tracking system as well as acting as a hub for the three Infrared (IR) emitters. These are placed strategically behind the screens in order to cover 180 degrees of direction and movement by the user. The tracking system works by means of ten NaturalPoint IR cameras[6] which are spread out along the upper edges of the three vertical CAVE screens with each pointing to a specific area of the CAVE space in order to provide full area coverage. The tracking system identifies objects using reflective spheres which are placed onto 3D glasses and are primarily used to track a user's head movements. They can, however, also be used to track multiple tagged objects at any given time. The cameras are then linked to the tracking PC (via USB cables) which combines the data from each of the cameras to build up a full picture of tracked objects within the CAVE. This information is then fed to a tracking client which resides on the Master workstation and can be utilised by the VE management software to adjust the on-screen visualisations in real-time.

Users can interact with the VWs presented to them by various means. The standard methods for CAVE environments tend to be either dataglove, joystick, joypad or wand style controllers. Currently a joypad, a Microsoft Kinect and a modified Nintendo Wii Nunchuck are used to interact with the CASALA CAVE, all of which are discussed in more detail in Chapter 4.

In terms of software, the CASALA CAVE currently uses a range of systems including VR4MAX[7], WebGL[8] and SCRAPE in order to view and interact with VWs. Each of these platforms offers a specific capability which can be useful for generating distinct VWs and scenarios. In particular, a program which is suitable for an architectural style environment may not be suitable for an abstract data visualisation environment. This is discussed in more detail in Chapter 4. In Chapter 6 a novel open source framework for CAVEs entitled SCRAPE is presented. Based on Processing[9], it has been developed specifically to enable CAVE users to quickly and easily develop useful data visualisations for all types of CAVE configurations.

2.5 Conclusion

This chapter discussed some of the key people and technologies behind the evolution of VEs over the last half century. The thesis contributes to the knowledge of key terminologies associated with VEs by clarifying many of the ambiguities which currently exist and by providing new definitions where required. The key elements that

[6] http://www.naturalpoint.com/
[7] http://www.tree-c.nl/vr4max
[8] http://www.khronos.org/webgl/
[9] http://processing.org

constitute the CASALA CAVE are also characterised in order to clearly illustrate the composition of a ‘typical’ CAVE.

The following chapter expands upon this theme by focusing upon the important role that immersion plays in CAVE environments, the definitions associated with it and the key elements that influence it.

Chapter 3

Enhancing Immersion

3.1 Introduction

There is strong empirical evidence to prove that increased levels of immersion impact positively upon a user's ability to carry out tasks in a CAVE and demonstrate superior performance results when compared with traditional desktop applications (Pausch et al., 1997). Doug Bowman from the 3D Interaction Group at Virginia Tech University is one of the leading authorities on VEs and has written many articles, books and papers on the subject. In his paper entitled 'A Method for Quantifying the Benefits of Immersion Using the CAVE' (Bowman and Raja, 2004), Bowman provides us with a practical guide as to how we can measure and compare the impacts of immersion. In his article 'Virtual Reality: How Much Immersion Is Enough?' (Bowman and McMahan, 2007) he clearly lays out the benefits of Immersion and then in multiple other publications (Bowman et al., 2005, 2009, Narayan et al., 2005, Schuchardt and Bowman, 2007, Sowndararajan et al., 2008) he demonstrates the impact that levels of immersion have on a range of different VE scenarios.

A primary objective of any CAVE environment, therefore, is to provide as immersive an experience as possible into a VW. In other words, it is about trying to create an environment that absorbs the user so that they become unaware of the physical CAVE and its surrounds and fully experience and 'believe' the three dimensional world that is generated for them.

In Janet H. Murray's book *'Hamlet on the Holodeck'* (Murray, 1997) she describes immersion as being:

> *"a metaphorical term derived from the physical experience of being submerged in water. We seek the same feeling from a psychologically immer-*

sive experience that we do from a plunge in the ocean or swimming pool: the sensation of being surrounded by a completely other reality, as different as water is from air, that takes over all of our attention, our whole perceptual apparatus."

This is an eloquent definition, however, in relation to CAVE environments it is insufficient. There are other terms such as 'Presence' and 'Telepresence' which are often used interchangeably with immersion to describe the same thing, but which are not. Mel Slater from University College London who is a recognised authority on VE immersion has written extensively on this subject. He argues in his article 'Measuring Presence' (Slater, 1999) and in his note on presence terminology (Slater, 2003) that a clear definition and separation of the terms is essential to avoid confusion. With this in mind, the following subsection provides a full definition for each of the three terms. This is followed by an overview of the ways in which presence can be measured and the ways in which immersion can be augmented.

3.2 Definitions

- **Immersion**
  Slater proposes that the term 'Immersion' should stand for what the technology delivers from an objective point of view. The more a system delivers in terms of number of screens, tracking, audio, 3D & display fidelity, haptic feedback etc., the closer it gets to replicating its equivalent real-world sensory modalities and the more that it is immersive.

- **Presence**
  According to Slater, presence can be interpreted as a human reaction to immersion. In other words the sensation of 'being there'. Therefore, we may consider that a key objective for any CAVE researcher is not simply to provide an immersive environment, but to provide the highest level of immersion possible in order to augment the level of presence that a user experiences.

  Slater considers that there are three main aspects to presence:

  1. The sense of 'being there' in the environment depicted by the VE.
  2. The extent to which the VE becomes the dominant one i.e. that participants will tend to respond to events in the VE rather than in the 'real world'.
  3. The extent to which participants remember their VE experience as having

visited a 'place' rather than just having seen images generated by a computer.

- **Telepresence**
  The term 'Telepresence' was coined by Marvin Minsky in an article to OMNI magazine in 1980 (Minsky, 1980). In it he described how people could use special devices that translate the movement of an arm, hand and fingers and reproduce it in another place by mobile, mechanical hands. The term Telepresence, according to Minsky, conveys the idea of these remote control tools, emphasises the importance of high quality sensory feedback and suggests future instruments that will feel and work so much like our own hands that we won't notice any significant difference.

  Even though technology may have moved on somewhat since 1980 and the scope of the original definition of the term 'Telepresence' may be broadened beyond remote mechanical arm manipulation, Minsky's description of telepresence still makes sense in relation to controlling or manipulating remote objects in a VE.

  In Jonathan Steuer's paper 'Defining Virtual Reality: Dimensions Determining Telepresence' (Steuer, 1992) he defines telepresence as:

  > *The experience of presence in an environment by means of a communication medium.*
  >
  > *In other words, presence refers to the natural perception of an environment, and telepresence refers to the mediated perception of an environment.*

Steuer's definitions of presence and telepresence are somewhat at odds with what was previously described by Slater and Minsky respectively. In reality, Minsky's definition of telepresence is more to do with teleoperation and what Slater defines as presence is in many respects what Steuer defines as telepresence.

For the purposes of this thesis, it is considered that Slater's simple definitions of immersion and presence will suffice and are the most relevant in the context of a CAVE environment; immersion being what technology can deliver in terms of replicating its equivalent real-world sensory modalities and presence as being the human reaction to immersion (or sense of 'being there'). Having clarified the differences and associations, the next step is to consider the way in which levels of presence can be measured.

3.3 Measuring Presence

As stated in Wijnand IJsselsteijn's paper 'Measuring Presence: A Guide to Current Measurement Approaches' (Baren and IJsselsteijn, 2004), there are two primary categories of measurement that are used for determining presence: Subjective and Objective Measures. Subjective Measures require participants to consciously provide a judgement of their state with regard to their experience within a VE, whereas, Objective Measures attempt to measure a participant's state automatically and without conscious deliberation, but which can somehow be correlated with measurable properties.

3.3.1 Subjective Measures

Simply asking someone how immersive their VE experience was may prove highly subjective and ineffective in accurately determining levels of presence. That being said, most of the research carried out in this area looks at qualitative rather than quantitative measures (usually through the use of questionnaires). In Thomas Sheridan's paper 'Musings on telepresence and virtual presence' (Sheridan, 1992) he notes a number of subjective measures. The primary category is considered to be presence questionnaires but he also defines 4 other subjective measures: Continuous Assessment, Qualitative Methods, Psychophysical Measures and Subjective Corroborative Measures, which are discussed in the following:

- **Questionnaires**
  Two of the best known and widely used questionnaires are the Presence Questionnaire (PQ) and the Immersive Tendencies Questionnaire (ITQ) as set out by Witmer and Singer in their article 'Measuring Presence in Virtual Environments: A Presence Questionnaire' (Witmer and Singer, 1998).

  1. **The Presence Questionnaire (PQ)**
     The PQ is used to measure the degree to which individuals experience presence in a VE and the influence of possible contributing factors on the intensity of this experience.

  2. **The Immersive Tendencies Questionnaire (ITQ)**
     The ITQ is used to measure the capability or tendency of individuals to be involved or immersed in a VE. in other words the ITQ does not ask specific questions about the VE but rather the person's ability to involve themselves in common activities.

Both of these questionnaires use a seven point scale to rate the answers. So, for example, the first PQ question that Witmer and Singer propose is: 'How much were you able to control events?'. Answers are then rated on a scale ranging from 'Not Very Much' to 'Moderately' to 'A Lot'.

Some other well referenced subjective VE presence questionnaires that have been developed over the years include Slater, Usoh and Steeds' presence questionnaire (Slater et al., 1994), Barfield and Weghorsts' 10 point questionnaire (Barfield and Weghorst, 1993) and Nichols physical presence questionnaire (Nichols et al., 2000).

- **Continuous Assessment**
  Continuous Assessment aims to measure presence by requiring VE users to continually rate their sense of presence during the actual experience itself. The level of presence is recorded through the use of a slider which can be continually adjusted by the user along a scale during the experiment. This allows fluctuations in perceived levels of presence to be measured at any given time and helps prevent against memory recall problems that may be an issue through the use of post experiment questionnaires. Another similar method is proposed by Mel Slater in his paper 'A virtual Presence Counter' (Slater and Steed, 2000) in which he describes how users were required to call out 'Now!' to report transitions from the virtual world to the real world. Critics of these type of assessments, however, point to the fact that the act of indicating current presence levels disrupts the user experience and, consequently, the level of presence that would otherwise be felt.

- **Qualitative Measures**
  Qualitative Measures are a much less formal method of assessment and in general refer to open ended discussions or interviews with VE users about their experience with no set list of questions or required responses. This may provide unexpected insight into the user's experience and the factors that impact their sense of presence, however, the open ended nature of the responses makes them difficult to assess and compare, and may be considered more as a supporting tool rather than a principal method of assessment.

- **Psychophysical Measures**
  Psychophysical Measures require VE users to rate their perceived level of presence based on specific stimuli which are provided for them in the VW. A typical experiment may require users to complete certain tasks where different measurable stimuli such as frame rate levels, FOV, head tracking etc., are modified and then levels of presence are measured and compared against those changes. This experiment helps identify which factors may have greater impact in the levels of

presence felt by the user.

- **Subjective Corroborative Measures**
  Corroborative Measures do not attempt to directly measure presence but instead look at different elements which are considered to be related to presence. Different measures include; levels of attention and awareness of external stimuli, memory levels of different aspects of a VW, levels of motion sickness, task time duration etc. The theory behind Subjective Corroborative Measures is that the results of these experiments can help ascertain levels of presence indirectly. Increased levels of attention to a VW and a lower level of external stimuli awareness could be considered a reasonable indicator of increasing levels of presence.

3.3.2 Objective Measures

As stated previously, objective measures attempt to measure a participant's state automatically and without concious deliberation but which can be correlated with measurable properties. In Martijn J. Scheumie's paper 'Research on Presence in Virtual Reality' (Schuemie et al., 2001) he proposes two types of objective measures: Behavioural and Physiological.

- **Behavioural Measures**
  Scheumie argues that people tend to respond to stimuli in a VE as if they are unmediated stimuli when they experience a high level of presence. He therefore proposes that by observing and measuring specific reactions of participants to specific stimuli, it may be possible to gauge levels of presence. A practical example of this type of behaviour can be observed in the CASALA CAVE when participants are navigated at speed towards a specific object (such as a virtual tree). They often react by attempting to move out of its path or by raising their arms and hands to protect themselves. By measuring these types of responses it may be possible to provide a more objective reading of a participant's level of presence.

- **Physiological Measures**
  Scheumie argues that it may be possible to gauge a participant's level of presence by measuring specific aspects of the participant's body, such as skin temperature, heart rate, skin conductance etc. It does appear that some physiological measures can indicate increased levels of presence, particularly in tests involving people with specific phobias that are then placed in a VW designed to trigger those fears. These tests can then be compared with tests where participants perform similar tasks on a desktop PC to help corroborate any findings.

Figure 3.1: Slater's Pit Room
Source: (Slater, 2002)

Slater talks specifically about both Behavioural and Physiological measures in his paper 'Presence and the sixth sense' (Slater, 2002). In this he makes reference to the 'Pit Room' (see Figure 3.1) which was designed specifically to elicit a reaction from users in a VE. Essentially it consists of two virtual rooms, one room (from which the user initially starts) and a second room which can be accessed via a doorway. In the first room there is nothing of any particular interest. It is a simple rectangular shaped room with wooden floors, painted walls, four chairs and a doorway. In the second room, however, there is an eighteen metre precipice in the centre. Users navigate around the relatively uninteresting first room then make their way to the second room. It is at this point that they are confronted with the precipice. Reactions such as users jumping back to prevent themselves from the perceived danger of falling, as well as physiological measures such as increased heart rate are not unusual, all of which would indicate a high level of presence felt by the users. This is a somewhat extreme example designed specifically to induce a reaction. In reality, it may be far more difficult to apply behavioural and physiological measures to a more sedate virtual experience.

3.3.3 Other Considerations

One factor that hasn't been considered so far with regards to measuring presence is the relationship between presence and a person's cognitive characteristics. In Corina Sas and Greg O'Hare's paper 'Presence Equation: An Investigation Into Cognitive Factors

Underlying Presence' (Sas and O'Hare, 2003) it is argued that the greater a person's level of absorption, creativity, imagination and willingness to experience presence, the more positive the impact in terms of that person's sense of presence. While research into this particular area of presence is still in its infancy, it does raise some interesting questions that at the very least should be considered in the context of CAVE environments and the levels of presence that users' experience.

Having looked at many of the different ways in which to measure presence, the next consideration in this thesis is to investigate the ways in which presence can be augmented via changes in the levels of immersion.

3.4 Augmenting CAVE Immersion

As argued by Slater, presence is a human reaction to immersion. Seeing as CAVE designers have ultimate control over a CAVE's immersive technologies (i.e. numerous screens, object tracking, display fidelity etc.), it is therefore important to understand the factors that influence immersion in order to maximise a user's sense of presence. These are outlined as follows:

- **Photorealism**
  One could be forgiven for assuming that the level of immersion generated in a CAVE is closely linked to the level of photorealism within a VW, however, this is not the case. The sense of depth and space that the virtual world conveys is arguably far more important (Durand, 2002), and implies that a cartoon or abstract world can be just as immersive as one that is photorealistic. Techniques such as ambient lighting or global illumination within a 3D world can have a significant impact on the sense of depth and space conveyed to the user (Mortensen et al., 2007).

- **Reduced Ambient Light**
  Not to be confused with the use of ambient lighting within a 3D scene, the ambient light referred to here, is any light that is not generated by light thrown on to the screens by its projectors. The more ambient light there is, the more the user is made aware of the physical CAVE surrounds. This is likely to distract the user's focus on the VW and therefore their immersion into the VW. Ideally, CAVEs should be set-up in windowless rooms where any artificial lighting can be fully controlled.

- **3D Projectors**
  Projectors that are capable of displaying crisp and clear stereo 3D images are one

of the most important aspects in enhancing the immersive effect of the CAVE. It is the 3D projections (when combined with 3D glasses) that provide the overriding illusion of depth in the virtual world. The resolution provided by the projectors can also help to support that effect, with higher resolutions providing more clearly defined images.

- **3D Glasses**
  Flicker-free 3D shutter glasses or high quality polarized glasses ensure the illusion of depth is as effective as possible. A high quality 3D stereo projector with poor quality glasses is likely to have a significant negative impact in the immersion level experienced by users.

- **Screens and Seams**
  The number of screens in a CAVE impacts the level of immersion. A standard four screen CAVE provides a high level of immersion, particularly when a user is facing the centre screen. This ensures that the user's entire horizontal FOV (which is generally anywhere between 160 to 180 degrees) will be fully covered by the CAVE's three vertical screens. It is only if the user rotates his/her head a significant angle facing away from the centre screen, or up towards the ceiling, that there will be a break in the screen coverage. This is also referred to as a person's Field of Regard (FOR). CAVEs with six screens can provide complete coverage.

  Another important consideration is the way in which CAVE screens are linked together. Any material other than the projection screen itself will interfere with the 3D illusion, particularly in the low ambient light conditions used in CAVEs. Many CAVE environments do not use visible seams and instead join the seams behind the screens outside the path of a projector's beam of light.

- **Audio Set-up**
  Depending on the type of 3D world with which the user is interacting, the type of audio set-up may be of greater or lesser importance to the user. Imagine the scenario of a VW which is designed to assess a user's ability to identify and locate traffic sounds. Having multiple speakers placed at different points around the CAVE would allow for the generation of traffic noise from multiple points and help users to correctly identify which direction a vehicle is coming from. Providing sounds that appear to come directly from the source as indicated in the VW will contribute to the sense of immersion. The importance of this can depend on the VW that is being explored and the tasks that the users are required to perform. In many cases simply adding sound effects (not necessarily from the direction of the perceived source) can help augment immersion.

- **Hardware and Software Specifications**
  Ensuring that CAVE hardware and software can successfully render and manage the VW is essential. A 3D world that jitters and struggles to keep up with the user's movements will not provide for a highly immersive experience.

- **Object Tracking**
  Object tracking can add to the immersive experience in different ways. Most commonly, object tracking is used to determine the positional co-ordinates of a user's head within the CAVE. This data is then used to adjust the image that the user sees on the screen. Imagine standing in the centre of a CAVE looking at an image of a wall that is shoulder height on the screen in front of you. Behind the wall some distance away is the image of a tree. In the real world if we hunker down behind a wall, the tree behind it will be hidden from view. In the CAVE, head tracking replicates this scenario, so as the user hunkers down, the view will descend and the tree will no longer be visible behind the wall. In order to achieve this, the CASALA CAVE uses ten IR cameras placed around the top edge of the three vertical screens. These are able to detect the exact position of three reflective balls that are placed on the user's 3D glasses providing the x, y and z co-ordinates of the user's head within the CAVE space and pass that information back to the CAVE software. This is a widely used object tracking set-up, however, technologies such as Microsoft's Kinect can also provide an interesting and much lower cost alternative to traditional tracking implementations.

- **CAVE Floor Area**
  One of the limitations of a CAVE is the fact that its usable area is generally quite small (typically $2m^2$ to $3m^2$). This means that a user's movements are significantly restricted. Ironically, due to the highly immersive nature of CAVEs, this can sometimes lead to users wandering towards the screens and becoming disorientated. This is a significant limitation of CAVE environments and their ability to fully immerse a user.

- **Haptic Feedback**
  Alongside the restrictive size of CAVE environments, realistic haptic feedback poses a huge challenge in realising the ultimate immersive dream of many CAVE researchers (i.e. replicating the Star Trek Holodeck). The ability to be able to touch and feel virtual objects would significantly enhance the sense of immersion. This can be achieved to some limited extent through the use of force feedback gloves or joysticks but the technology is not yet mature. However, research continues apace, for example, Disney Research presented a paper at SIGGRAPH 2013 demonstrating their new AIREAL haptic feedback device (Sodhi et al.,

2013) which takes a highly novel approach to haptic interactions[1].

- **Interaction Modalities**
  Finally, the methods by which users interact with CAVE VWs can also have an impact on their sense of immersion. The more natural and intuitive the interaction method is, the more likely users will 'lose themselves' within the VW. Interaction modalities and the impact they have on our abilities and experiences in CAVE VWs are dealt with in detail in the following chapter.

3.5 Conclusion

This chapter identified the way in which immersion can impact on a CAVE user's ability to 'believe' the VW in which they are interacting. It clearly defines seemingly synonymous terms associated with immersion and describes the ways in which presence (the human reaction to immersion) can be measured. It also lays out the different methods that can be employed to augment the level of immersion. All of which, it should not be forgotten, is ultimately aimed at positively impacting on a user's ability to perform in a CAVE.

Having previously characterised the elements that constitute a CAVE and discussed the unique contribution that immersion provides, the next step is to look at the different interaction modalities that can be employed. The following chapter investigates the different methods of CAVE interaction and the key factors to consider in choosing the right one.

[1] http://www.disneyresearch.com/project/aireal/

Chapter 4

CAVE Interaction Modalities

4.1 Introduction

'CAVE Interaction Modalities' refers to the different input methods by which a user can alter the state of the computer generated environment with which they are engaging. The following chapter investigates how the different interaction devices being used in a CAVE affect the VW and how these different interactions affect our abilities and experiences within that world.

There are many possible ways in which to interact with CAVE VWs. Typical devices include: Datagloves, joypads, wands, IR tracking devices and off-body gesture controllers. Some interaction devices such as joypads and wand controllers have been around for many decades, whereas, other devices such as the Microsoft Kinect have only been available in the last few years. As technologies change, so too do the methods and possibilities in which to interact with CAVE environments. This is a topical issue for many CAVE researchers (Otaduy et al., 2009, Sutcliffe et al., 2006) as the choice of modalities has the potential to fundamentally change how a user engages with a VW and, therefore, impacts on both the experience itself and what can be understood and learnt from it.

4.2 Key Considerations

In order to ensure that the correct interaction methods are chosen for the relevant VW scenarios, there are two key considerations which should be taken into account before deciding upon any particular solution and these are discussed in the following subsections.

4.2.1 Visualisation Scenario

The first consideration relates to the type of visualisation scenario being employed, and in particular, whether or not it is a 'Built Environment' style visualisation or an 'Abstract' style visualisation. For the purposes of this thesis, a *Built Environment Visualisation* can be defined as:

> *A visualisation scenario where the user engages with a three dimensional representation of the real world. A world that contains familiar objects such as buildings, cars, people etc., and where, for the most part, the general laws of physics are replicated within the VW.*

An *Abstract Visualisation* on the other hand has no such constraints and is often used to visualise raw data. In this scenario it is up to the creator of the VW to determine how the data should be represented and what interaction possibilities should be applied. The term 'Abstract' is itself defined as:

> *Relating to or denoting art that does not attempt to represent external reality, but rather seeks to achieve its effect using shapes, colours, and textures*[1].

By fully understanding the type of visualisation scenario required, we are in a better position to assess the likely freedoms and constraints associated with it and, therefore, provide for a better choice of interaction modalities.

4.2.2 Degrees of Freedom

The second key consideration is known as the Degrees Of Freedom (DOF) of interaction (see Figure 4.1). This is an important factor in assessing the suitability of different interaction devices for different types of VWs and scenarios. A device which can provide six DOF is considered to have total freedom of movement. DOF is more accurately defined as follows:

> *In engineering terms, DOF describes flexibility of motion. A mechanism that has complete freedom of motion (even if only in a limited area, or envelope) has six degrees of freedom. Three modes are translation - the ability to move in each of three dimensions and three are rotation, or the ability to change angle around three perpendicular axes.*

[1] Source: Oxford Dictionary

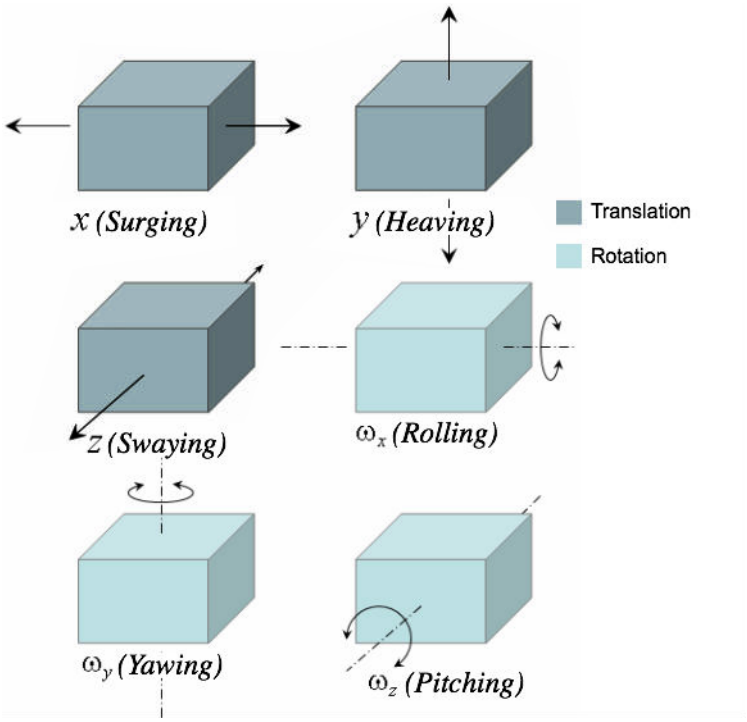


Figure 4.1: Six Degrees of Freedom - Translations and Rotations
Figure adapted by author from: (Diracdelta.co.uk, 2012)

> *To put it in simpler terms, each of the following is one degree of freedom:*
>
> 1. *Moving up and down (heaving)*
> 2. *moving left and right (swaying)*
> 3. *moving forward and back (surging)*
> 4. *tilting up and down (pitching)*
> 5. *turning left and right (yawing)*
> 6. *tilting side to side (rolling)*[2]

To provide a practical example of DOF, imagine a scenario of two CAVE VWs. In one world the user controls a car travelling along a road, in the other world the user controls an aeroplane in mid-flight. Due to the fact that the car is fixed to the road by the simulated forces of gravity and the mechanics of how a car travels, its movement is restricted to two controllable DOFs (surging and yawing). An aeroplane in mid-flight, however, does not have as many restrictions and therefore has four controllable DOFs available to it (surging, pitching, yawing and rolling). See Figure 4.2. Due to the difference in DOFs between the two scenarios very different interaction devices may be required. Where a simple digital directional pad (D-pad) style controller might suffice to control the car it may not provide sufficient flexibility of motion for the aeroplane.

[2]Source: http://www.fact-index.com/d/de/degrees_of_freedom.html

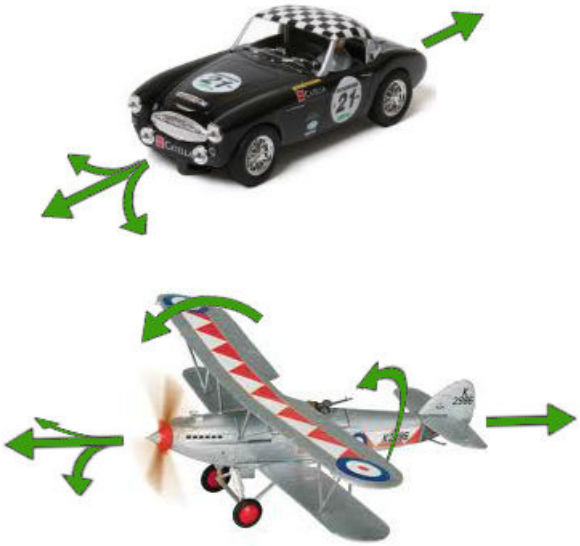


Figure 4.2: Different DOF requirement examples

This is a simplistic example and does not take into account the particular requirements of each simulation or any aspects particular to the CAVE. It does, however, highlight the basic principle of DOF and its importance in relation to interaction requirements. It also demonstrates how scenarios which fit within the same category of visualisation (in this case the Built Environment) may require very different methods of interaction. This is not to mention the possibilities offered by 'Abstract' visualisations where more complex interaction methods may be required if they offer more DOFs to the user.

In choosing appropriate interaction devices it is often tempting to opt for those devices that provide the greatest freedom of movement, however, this is not always recommended. In most cases we should be looking to implement those devices that provide just the required amount of DOF and no more. This assists in providing a more tightly controlled interaction experience and avoids any unnecessary confusion for the user.

4.3 CAVE Affordances

The concepts and examples discussed so far could just as easily apply to a VW on a desktop PC as they could to a CAVE, therefore, it is important to identify and understand the specific affordances that are unique or in some way different for a CAVE. Factors to consider include:

1. **User Mobility** - A CAVE user has freedom of body movement (albeit within the restrictions of the CAVE itself) unlike the fixed position of a desktop PC

user. How does this impact interaction? For a start, wired devices may pose a challenge within a CAVE whereas an off body controller may prove more difficult for a PC user.

2. **Seated Vs Standing** - Considering the fact that a PC user is normally seated and a CAVE user is normally standing, does this impact our ideal method of interaction? Possibly yes, particularly if we are thinking of incorporating tracking, wired devices or fixed position devices etc.

3. **Field of View** - Does the wide FOV within a CAVE impact on interaction? By significantly reducing the tunnel effect as compared to a VW scene viewed on a standard desktop monitor, it may do. Observation of the environment often becomes easier through movement of the user's horizontal head position rather than through the movement of the environment around the user. This may therefore impact on the choice or set-up of interaction devices.

4. **Immersion Levels** - By its very nature, through the use and wrap-around position of large screen displays, wide FOV, surround sound, low ambient light levels and stereo 3D, a CAVE aims to be more immersive than a standard PC. Interaction methods that support and augment immersion may be considered important to CAVE users (e.g. head, hand and body tracking)

5. **Stereoscopic Displays** - Although stereo 3D is widely and cheaply available on PCs (e.g. NVIDIA VISION[3]), generally speaking, it is not a standard feature of most computers. Displaying a VW in stereo 3D adds depth for the user and opens up new interaction possibilities. For example, a gamepad style controller combined with tracking can be used to generate a virtual laser pointer which in turn allows the user to select and interact with objects easily across different depth layers (see Figure 4.3)

6. **Surround Sound** - Typically a CAVE will have multiple speakers placed behind and around the screens as well as behind the main open space to provide full surround sound and help enhance immersion. This could play a role if interaction by voice and sound direction is important.

7. **Screen Real Estate** - The combined size of all CAVE screens provides users with a lot of screen real estate in which to view their VWs and this in turn can affect how the world is presented. So, for example, the large amount of CAVE screen space could impact how on-screen menus are displayed in comparison to a PC screen. If this is the case, it is important to consider whether or not this impacts on how the user interacts.

[3] http://www.nvidia.com/object/3d-vision-main.html

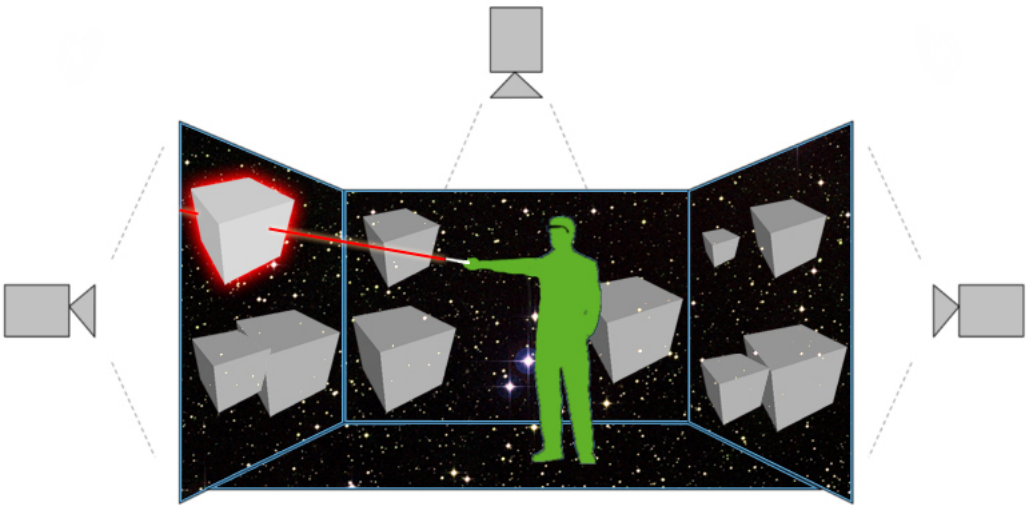


Figure 4.3: Virtual Laser Pointer

8. **Hardware and Software** - A CAVE generally requires a significant financial investment in expensive high-tech equipment which means that CAVE users often have some unique hardware and software tools at their disposal. This may provide an opportunity to interact in different ways, for example, through the use of expensive object tracking kit, glove controllers, force feedback devices or even custom built controllers.

4.4 Choosing Modalities

In order to assess which methods of interaction would be most suitable for a particular scenario the following key questions should be answered:

- What is the purpose of the virtual world?
  If the purpose of the virtual world is to assess a user's movements within a scene then perhaps it is necessary to ensure that the interaction method is a very natural one.

- Who is that world created for?
  If the VW is specific to older people, for example, then perhaps it should implement a less dexterous method of interaction than for young adults.

- What type of world is being generated?
  If the world being generated is an abstract environment such as a data model then it may require a navigation control that provides for greater DOF.

- What level of interaction is required?
  If a user needs to manipulate objects in a scene then perhaps this will require a specific interaction method that provides features to point, select, drag and stretch in some way that is specific to the CAVE environment.

- Have the CAVE's unique attributes been considered?
  Do the chosen interaction devices make the most of the CAVE's wide FOV and stereo 3D.

The ideal type of interaction strategies for the CAVE will be different depending on the envisaged usage scenarios. The following example demonstrates how experience and understanding of a virtual world can be impacted by the method of interaction.

4.4.1 CASALA CAVE Example

To illustrate some of the concepts and considerations of the previous sections, in this section a specific example (based on the author's own experience) is presented. Using the CAVE at CASALA, users can interact with a variety of VWs. One of these is the to-scale replication of the Great Northern Haven (GNH) apartments that are designed specifically for the needs of older people. Users can navigate around these virtual apartments using a standard Playstation style gamepad controller providing two DOF movements (surging and yawing) through the use of one of its analog thumbsticks. Additional actions such as position resets and screen menu actions are provided through the use of the gamepads buttons. This is the standard method for allowing users to interact with the VW and it provides the primary method of interaction by which users can understand and interpret the VW. To introduce another interaction method into the mix, the user will continue to use the gamepad controller in the same way as before, except this time, head tracking will also be incorporated. Multiple IR cameras which are positioned around the top edge of the CAVE screens track three reflective balls which are attached to the user's 3D glasses and adjust the on-screen image based on the user's head position and orientation. This will impact the user in two key ways. Firstly, it should enhance the sense of immersion that the user experiences by augmenting the parallax effect, similar to that which would be experienced in the real world. In other words, the position of nearby objects relative to far away objects will appear different based on the viewers position. Secondly, by engaging head tracking the user will observe the environment according to their own particular height. Any older users that have mobility issues and require the use of a wheel chair, for example, will instantly see the VW from the correct height perspective. This could be a crucial consideration if the objective of the VW is to assist in the design of homes for older

people. Important feedback on items such as counter heights, switch levels and access requirements could be missed. Even at its most basic level, head tracking would permit any user to easily hunch down and look underneath a virtual table to discover a previously occluded object. These scenarios could be catered for by pre-setting height levels or could of course be simulated through the use of a standard gamepad controller. It would, however, be more difficult to operate (as the use of more controls such as a second thumbstick or D-pad would come in to play) and be far less natural and intuitive for the end user.

4.5 Taxonomy of Interaction Devices

Having outlined some of the considerations in choosing appropriate modalities, it is important to enumerate some of the currently available interaction methods and their general characteristics. The aim of this is to enable CAVE developers to give due consideration to a wide range of interaction possibilities before choosing any particular one. This list is intended to be as comprehensive as possible but is by no means exclusive.

 Gamepad Controller (Drilnoth, 2013)	Gamepad controllers are already a familiar device for video gamers. Typically the user holds the device with both hands and a number of push buttons and thumbsticks are conveniently accessible under thumbs and forefingers. At CASALA they are used to provide two DOFs in architectural style visualisations, with the D-pad used for navigation and its buttons acting as triggers for further interactions within the VW. In more abstract style visualisations (that may require more DOF), this can be achieved through the use of the controller's analog thumbsticks as well as its D-pad. While these controllers are extremely adaptable and tend to prove very effective for those accustomed to them, they can appear overly cluttered and non-intuitive to inexperienced users.

 Wand Controller (Ar-tracking.com, 2013)	'Wand Controller' is a somewhat generic term to define a particular style of interaction device such as the Flystick2 (pictured), Wanda, Playstation Move, Nintendo Wii-mote etc. While each controller has its own particular traits, they tend to possess some key characteristics: (i) they have a similar 'remote control' style form, (ii) they are held or gripped in a similar way in one hand, (iii) they feature trigger buttons and (iv) they usually support high degrees of DOF through tracking or on-board accelerometers/gyroscopes. The intuitive yet flexible nature of wand controllers makes them ideal for many CAVE VW scenarios and they are probably the most popular method of CAVE interaction.

 Interactive Glove (Peregrine, 2013)	There are multiple glove controllers available on the market such as the Peregrine[4] (pictured), the 5DT Glove[5] and the Control Glove[6]. Some gloves incorporate touch sensitive points that act as the equivalent of a key press or button press, others, such as the 5DT measure the bend movement of specific fingers. When combined with tracking mechanisms, glove controllers offer the possibility to navigate through VWs and manipulate and interact with objects. Anyone who has seen the movie *Minority Report* can realise the potential. Unfortunately, most currently available glove controllers do not possess the same level of functionality, accuracy and sensitivity that would be expected of other controllers. In spite of this, it is worth noting that interactive gloves are still widely used in CAVEs.

[4]http://theperegrine.com/
[5]http://www.5dt.com/
[6]http://fibrestructures.com/innovations.php

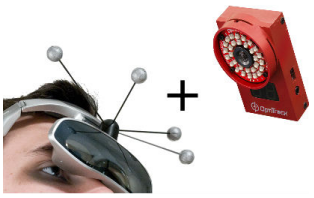
 Fixed Position Joystick	An integrated joystick and stand (with perhaps one or two trigger buttons) provides an interesting method for interacting with a virtual world, particularly if it is used in a VW where only a few degrees of freedom are required. Its fixed position helps users to focus on the virtual environment and eliminates the need to hold and carry a cumbersome controller, however, this may limit its usefulness in many situations. Also, the fact that the joystick is placed on a fixed stand will restrict movement, will block some of the user's view of the floor screen and may cast additional shadows.

 Dance Pad	A 'Dance Pad' is a flat electronic controller device that is placed on the ground and provides the user with interaction through the placement of feet on specific areas of the pad. They are typically used at home and in games arcades for dance based videogames and can be easily replicated through the use of floor sensors attached to a microcontroller such as the Arduino[7]. The advantage of using a dance pad (or similar floor sensing devices) in a CAVE is that they do not require the use of hand-held or on-body controllers and have the potential for enabling simple and intuitive interactions. The disadvantage, however, is that interaction via foot placement alone may be restrictive and force the user to concentrate on their foot movements rather than on the VE itself.

[7] Arduino is an open source electronics prototyping platform based on flexible, easy-to-use hardware and software. It is intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

 Voice Control (Friedrich, 2013)	Using voice commands to control specific aspects of a VW adds a natural and intuitive interaction modality suitable for CAVE environments. Through the use of voice recognition software such as Microsoft Speech, it may be possible to better integrate users into the CAVE by recognising key words and sentences and converting them into direct actions. Although voice control is generally not considered a primary interaction method, it provides huge potential when combined with other natural interaction methods such as controller-free gesture recognition or haptic/aural feedback.
 Body Sensor-Based Recognition (Emotiv.com, 2011)	The possibility of being able to interact with a CAVE by thought alone is an exciting prospect. Devices such as Emotiv Systems EPOC neuroheadset[8] offer the promise of thought based interaction via neuro-signal information which is collected from multiple sensors placed on the user's head. Unfortunately, trials carried out by the author as part of a 5 day course on 'multimodal interaction in virtual environments' at Aalborg University in Denmark in 2011 demonstrated poor levels of reaction and accuracy. There are, however, many other types of body sensors that are proven to be reliable and accurate which also offer the possibility of interacting with CAVE environments (e.g. sensors for monitoring heart rate, body temperature, fall detection, etc.). These sensors are often low cost and easy to integrate with microcontrollers such as the Arduino. This allows for interesting interaction possibilities such as fall detection scenarios within a virtual home or dynamic reactions within a VW scenario based on a user's vital sign changes.

[8] emotiv.com

 Phone and Tablet Apps Integration (Android.com, 2013)	In certain situations the use of a smartphone or tablet device could be a useful way to interact with a CAVE. Using custom-built apps, the device could be used to interact with a 'Built Environment' simulation in real-time in a CAVE. So, for example, the device could simulate a home automation control panel (e.g. turning lights on or off, activating an alarm or opening and closing blinds). This is potentially a useful way to simulate and test real-world home interaction scenarios. In terms of interacting with abstract data visualisations the device could also be used to feed data directly back into the CAVE and adjust the data in real-time rather than through a computer placed outside of the CAVE space.
 Object Tracking (Iotracker.com, 2012)	Object tracking is most often achieved through the use of IR cameras and reflective spheres. By strategically placing a selection of spheres on a CAVE's 3D glasses within view of special IR cameras, a user's head movements can be tracked. Because it is possible to know the exact positional co-ordinates of someone's head or any other object that is being tracked within the CAVE space, it is then possible to use this information to interact with the VW in other ways. One way is to use object tracking to generate a virtual pointer from a controller device allowing the user to select, move and manipulate objects and menus in the virtual world (refer back to Figure 4.3). Another way could be to trigger certain actions when a tracked object is within specific co-ordinate positions within a CAVE space, such as opening a virtual door.

 Computer vision-based gesture recognition (Alphathon, 2013)	An alternative way of recognising human gestures in a CAVE environment is by using computer vision techniques to provide 'controller-free' interaction, whereby, the user does not have to hold any physical device or sensor. In this regard, the Microsoft Kinect probably offers the most exciting new method of interaction. Through the use of its RGB camera, IR depth sensor and microphone array, the Kinect is able to determine a user's exact position within a defined range and track their detailed skeletal movements. Then, using tools such as the Microsoft Kinect SDK[9] these movements can be translated into actions in the virtual environment, similar to the configuration already implemented in the CASALA CAVE[10].
 Custom Controllers (Arduino.cc, 2013)	If none of the available interaction devices fits specific requirements, or budget, then a bespoke custom solution can be built (see Section 4.6 where the author's own work in this regard is described). Low-cost, accessible microcontrollers such as the Arduino make building or integrating new devices relatively easy. The integration of the Nintendo Wii Nunchuck in the CASALA CAVE (see below) is a perfect example of this.

4.6 Nunchuckoo: a novel interaction device for CAVEs

As part of a larger open source CAVE visualisation project called SCRAPE, which is discussed in detail in Chapter 6, a new easy-to-use interaction device in the CASALA CAVE is proposed and described here. Entitled 'Nunchuckoo', the objective of this project is to integrate a Nintendo Wii Nunchuck controller, an Arduino microcontroller and the Processing visualisation programming tool to provide an intuitive, yet effective, method of interaction for the CAVE (see Figure 4.4). It also serves as an experiment on the ability to easily integrate new devices into the SCRAPE CAVE system, and as an example of employing appropriate interaction modalities based on the particular requirements of a given system.

The low-cost Nintendo Wii Nunchuck controller is employed due to its unique de-

[9]Microsoft Kinect SDK: http://www.microsoft.com/en-us/kinectforwindows/develop/
[10]See a video demonstrating this at: http://youtube.com/watch?v=fxnMMVy9tFU

Figure 4.4: The Nunchuckoo Controller

sign and versatility, which makes it ideal for all types of users across a wide range of different CAVE VE scenarios. This simple ergonomic device is held in one hand and implements just one analog thumbstick and two trigger buttons, yet allows for the possibility of six DOF (due to its inbuilt accelerometer). This enables it to be useful in a wide range of both built environment and abstract data interactions, and as such, presents an interesting alternative to the standard gamepad controller. It also incorporates an I2C[11] serial connection which makes the job of connecting it to non compatible devices (such as an Arduino) significantly easier.

In support of this work, a brief summary of the set-up process is provided next, with basic sketch test code provided in **Appendix A** (Full and detailed instructions are available at **https://github.com/c-flynn/Nunchuckoo)**

4.6.1 Set-Up Summary

Required hardware and software :

- A Nintendo Wii Nunchuck
- An Arduino microcontroller and Arduino software[12]. An Arduino Duemilanove or newer (e.g. Arduino UNO) should work fine.
- A WiiChuck adapter[13]. Alternatively, wires can be connected directly between the Nunchuck and Arduino (see Figure 4.5).

[11] Inter-Integrated Circuit - Widely used communications protocol invented by Philips in the 1980's
[12] http://arduino.cc
[13] search on: http://amazon.co.uk or https://www.sparkfun.com

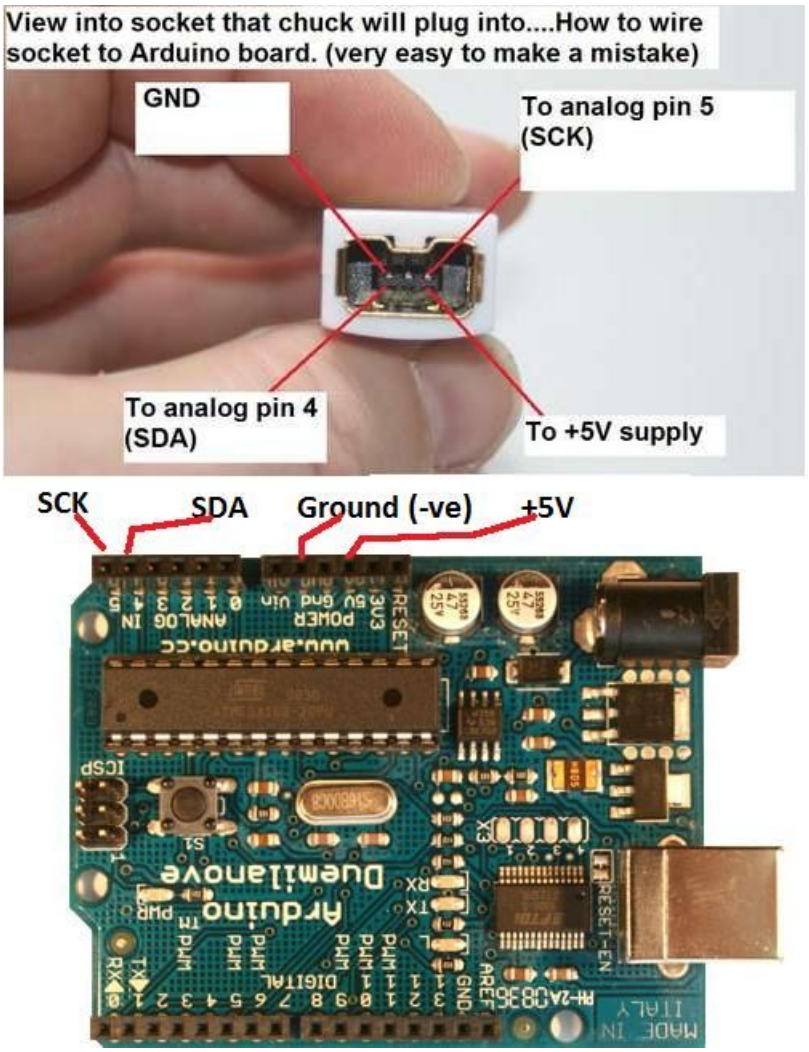


Figure 4.5: Nunchuck Arduino Direct Wire Connection
Source: (Instructables.com, 2013)

- A USB printer cable
- A copy of Processing[14]

Step 1 - Connect the Wii Nunchuck to the Arduino board via the WiiChuck wires as follows: (see Figure 4.6).

Black Wire - Arduino Gnd
Red Wire - Arduino 3V3
White Wire - Arduino Analog 4
Grey Wire - Arduino Analog 5

Step 2 - Connect the Arduino to a computer via the USB printer cable and launch the Arduino software.

[14]http://processing.org/download

Figure 4.6: WiiChuck Adapter

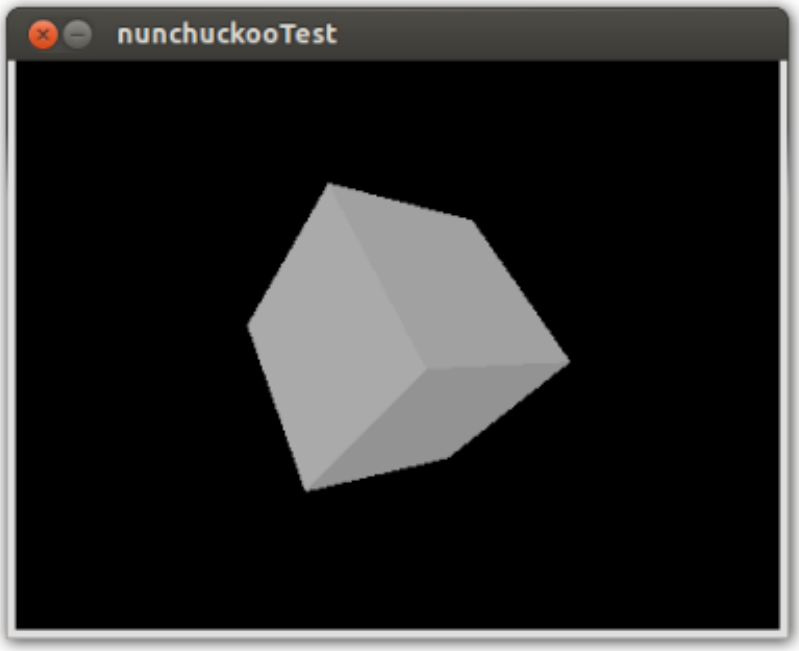


Figure 4.7: Nunchuckoo Test Sketch

Step 3 - Add Nunchuck datastream code to the Arduino Integrated Development Environment (IDE) in order to read the Nunchuck's controller serial data. This code is widely available on a number of Arduino related websites[15]. If all is working correctly a seven column readable datastream should be visible in the Arduino's serial monitor.

Step 4 - Add the Nunchuckoo test code (see Appendix A) to Processing. Ensure the Nunchuck and Arduino are connected, launch Processing and open the NunchuckooTest sketch. This will open a small window which displays a grey three dimensional cube in the centre of the window on a black background (see Figure 4.7). If everything is set up correctly, the cube can be rotated in different directions using the Nunchuck's

[15] http://pragprog.com/titles/msard/source_code

thumbstick. In order to move the cube using the Nunchucks accelerometers hold down the the Z trigger button on the controller. The Nunchuckoo is now ready for use in any Processing related projects.

4.7 Conclusion

As with any human-computer interaction, the choice of interaction modality is an important factor in determining a user's experience. This chapter addressed this matter in relation to CAVEs by highlighting the importance of visualisation scenarios and DOF when choosing modalities. It characterised many of the elements that make CAVEs unique and highlighted some basic questions to answer before choosing a specific method of interaction. A taxonomy of interaction devices was provided to assess their potential suitability for use in a CAVE and, finally, the integration of a novel interaction device was described.

The following chapter focuses on CAVE software, assesses some of the key software tools available and provides a critical analysis of their strengths and deficiencies. The concept of developing a low-cost CAVE is introduced and the suitability of existing software in any such system is discussed.

Chapter 5

CAVE Development Frameworks

5.1 Introduction

Ever since the invention of the CAVE in the early 90's, software development frameworks have been available to assist in the creation of exciting new VEs that exploit the unique characteristics and requirements of a CAVE. The number of CAVE frameworks available has grown steadily over the years with the result that today there is a healthy selection of frameworks from which to choose. Rather than present a full list (which a visit to Wikipedia[1] already provides), this chapter highlights a few key frameworks that are considered worthy of note and which may prove useful (in particular to CAVE newcomers). This chapter also highlights some of the deficiencies of these frameworks in relation to accessibility and cost, and lays the foundation for the introduction of a new framework which aims to address some of these concerns and supports the development of low-cost and accessible CAVE systems.

5.2 Key Frameworks

5.2.1 CAVElib

When Carolina Cruz-Neira, Thomas A. Defanti and Daniel J. Sandin developed the CAVE in the early 90's they also developed the CAVE Library (known as CAVELib[2]). CAVELib is a low-level Application Programming Interface (API) that enables developers to focus on developing their VW, while CAVELib handles most of the CAVE

[1] http://en.wikipedia.org/wiki/Cave_automatic_virtual_environment
[2] http://www.mechdyne.com/cavelib.aspx

system specific elements such as, machine clustering, multi-screen displays, virtual camera positioning and stereoscopic viewing. CAVELib merits a special mention as it is the first piece of software dedicated to the operation and development of VEs within a CAVE. CAVELIb is still available today, however, since 1996 it is a commercially owned product that is sold and distributed by Mechdyne (one of the largest suppliers of CAVE and CAVE type solutions worldwide). The materials, hardware and software for the CASALA CAVE were all originally supplied by Mechdyne and included CAVELib as part of the initial package. Although no longer used at CASALA, CAVELib remains an important entry in the history of CAVE software and continues to be widely deployed on many CAVE systems around the world.

5.2.2 VRJuggler

Following on from the development and subsequent sale of CAVELib to Mechdyne in 1996, Carolina Cruz-Neira and a team of students from the Virtual Reality Applications Center (VRAC) at Iowa State University went on to develop an open source and freely available virtual reality application development framework called VRJuggler[3]. The first alpha version of VRJuggler was released in September 1998, followed by the beta version in July 1999, with a full release being made available in October of the same year. VRJuggler followed on very closely from the work carried out on CAVELib and provides a reliable alternative to the commercially available CAVELib software. As with CAVELib, VRJuggler provides a low-level framework for CAVE systems that can handle clustering, multiple displays, head tracking, stereo viewing etc., thus enabling developers to remain solely focused on the creation of VWs.

5.2.3 VR4MAX

Developing graphically rich, extensive and interactive VEs with tools such as CAVELib and VRJuggler requires a significant investment of time and technical resources. If this is not an option (or if a user simply wants a graphically rich VE), then VR4MAX[4] from Tree C Technologies provides an interesting alternative. VR4MAX's key strength is its ability to port 3D objects created in the popular 3DS MAX graphics software package directly into a CAVE. This enables rich 3D worlds to be created and transferred to a CAVE in a very short period of time. It also comes with a rich list of features such as collision detection, terrain binding, lighting, animations, physics and atmospherics. Similar to CAVElib and VRJuggler, VR4MAX also handles machine clustering,

[3]https://code.google.com/p/vrjuggler/
[4]http://www.tree-c.nl/vr4max

multiple screens, interaction devices, head tracking and stereoscopic viewing.

A criticism that is often levelled at higher level frameworks is a lack of flexibility or programmability. While this is often true, VR4MAX provides a useful middle ground option, particularly as it also allows users to extend functionality through user developed scripts (using LUA[5]).

5.2.4 MiddleVR & getReal3D

MiddleVR[6] from 'I'm in VR' and getReal3D[7] from Mechdyne are two software products that are designed to work directly with the Unity[8] game engine. Unity is one of the world's most popular software tools for the development and running of video games across multiple hardware and software platforms and is ideally suited for the creation and running of CAVE VWs. Unity allows for the easy importation of 3D models and is designed to assist in the creation of highly interactive worlds with inbuilt support for items such as collision detection, physics, lighting, head-tracking, multiple controller devices and terrain binding. Essentially MiddleVR and getReal3D are two competing products that enable Unity to be extended so as to be compatible with the particular requirements of a CAVE environment similar to the frameworks discussed previously. This enables developers to concentrate solely on their VW and not on the CAVE specific elements of their developments.

5.3 Framework Comparisons

Each framework has its own particular strengths and weaknesses. One strength which they all share is their proven ability to afford rich interactive CAVE VEs when implemented with the necessary expertise. A key concern, however, relates to the ease with which VEs can be developed. In particular, frameworks that require expertise in programming languages such as C, C++ and OpenGL, and require significant work on the command line, have a smaller pool of potential users that can avail of them. This makes many frameworks immediately inaccessible or overly complex at the very least.

Another key concern, relates to their suitability for use outside of the typical CAVE scenario. CAVE environments in their traditional form are costly to implement and require highly skilled resources in order to develop and maintain. This tends to limit their

[5]http://www.lua.org/
[6]http://www.imin-vr.com/middlevr/
[7]http://www.mechdyne.com/getreal3d.aspx
[8]http://unity3d.com/

use to educational institutions and big business. However, as the cost-performance ratio of hardware continues to improve (particularly in relation to processing power and 3D projectors) so too does the opportunity for setting up lower cost CAVEs. Nonetheless, if the software remains too costly then this inevitably limits its adoption by a wider user base.

Considering that accessibility and cost are key considerations for this thesis, Table 5.1 is included to facilitate a comparison of each framework's key attributes.

CAVE Framework Comparison Table							
	Open Source	Cost	GUI	Component Sofware	Complexity	Platforms	Language
CAVElib	No	$\sim$ €3000 - €5000 per node + 10% annual fees	No	OpenGL OpenGL Performer Open SG Open Scene Graph	High	Windows Linux	C
VRJuggler	Yes	€0	No	OpenGL OpenGL Performer Open SG Open Scene Graph	High	Windows Linux Mac OS X	C++
VR4MAX	No	$\sim$ €3300 per node + €1900 annual fees	Yes	3ds Max	Medium - Low	Windows (Dev + Runtime) Linux (Runtime Only)	Lua
MiddleVR	No	$\sim$ €4000 per node + 20% annual fees	Yes	Unity	Medium	Windows Linux Mac OS X	UnityScript C# Boo

Table 5.1: CAVE Framework Comparisons

5.4 Conclusion

CAVE environments enjoy a reasonable level of choice in terms of software frameworks, from traditional code intensive applications such as CAVELib and VRJuggler, to more recent game engine based frameworks such as getReal3D and Middle VR. Despite the level of choice and different options, CAVE frameworks still remain relatively inaccessible in terms of both cost and development. The following chapter presents a new open source project to develop an entirely new CAVE framework which aims to address the key deficiencies of existing frameworks. The proposed framework can easily integrate into traditional CAVE environments without compromising any current framework implementations, but is designed specifically for new CAVE users that traditionally would not have implemented CAVE systems due to cost or other technical barriers.

Chapter 6

Project SCRAPE

6.1 Introduction

The CASALA CAVE is utilised in a range of different VW scenarios from built-environment style visualisations to abstract visualisations of real-world sensor data. These are used primarily to assist in the research and development of new 'independent living' technologies for the elderly and to improve our understanding of energy usage across communities. Over a period of four years, many different CAVE software applications were trialled against required VW scenarios, however, the high cost and/or inaccessibility or unsuitability of most remained a constant source of frustration. It became clear that there was a need and an opportunity to develop a new CAVE framework. The **need** for CASALA was clear but the **opportunity** was to develop an open source application which could be used not only in traditional CAVE systems but also in low-cost or amateur CAVEs. Interest expressed directly to the author by people from a wide range of disciplines regarding the possibility of setting up low-cost CAVEs was a significant factor in the drive towards this goal. This initiated the development of a new open software application that would enable both amateurs and professionals to easily set up and create interactive VWs. In order for any such application to work, it would need to be easy to configure, easy to develop for, and usable in CAVEs of differing sizes, dimensions and screen numbers. It would also need to facilitate a multitude of interaction devices.

After significant research into a suitable platform on which to develop a new CAVE framework, it was decided to use a software platform called Processing[1] (P5) and to call the project *SCReen Adjusted Panoramic Effect* or SCRAPE for short. Other languages, frameworks and libraries were investigated as part of the research process,

[1] http://processing.org

with the key focus being on cost and accessibility. With this in mind, many software applications that might typically be considered suitable for a new CAVE framework were not chosen for SCRAPE. OpenGL[2], for example, is an industry standard API for graphics rendering but it is complex to work with and was therefore not considered suitable for SCRAPE. A similar issue arises with regard to the use of scene graphs such as OpenSG[3] or OpenSceneGraph[4] both of which were considered overly convoluted for SCRAPE. In essence, SCRAPE required an existing platform or framework on which it could easily build. With this in mind, Java3D[5], VPython[6] and OpenFrameworks[7] were considered. Both Java3D and VPython provide relatively straightforward access to 3D graphics, however, they lacked the community support and fine tuned accessibility of P5, with its simplified Java syntax, excellent documentation and large community supports. OpenFrameworks, on the other hand, is an open source toolkit which could make the basis for an excellent CAVE framework, with a proven track record, excellent documentation and an active support community. In many respects it is the big brother of P5 (with its creators citing P5 as its precursor). While the development team behind OpenFrameworks have gone out of their way to make it accessible, it still requires knowledge of C++ and was therefore considered unsuitable for this project. Above all, It was important for SCRAPE to abstract itself from lower level APIs, libraries and languages and make itself more accessible. The P5 open source programming language and data visualisation tool platform, created by Casey Reas and Benjamin Fry, holds as its tenets the fundamental concepts of being freely available and easy-to-use by non-experts. While P5 has some limitations (principally, its use of Java over C/C++, its notably basic IDE and lack of graphical interface toolkit), it is an excellent and accessible tool for developing 2D and 3D interactive visuals. It also has a strong user community and a wealth of useful software libraries. For all these reasons, P5 was chosen as the platform of choice for SCRAPE.

The following section provides an overview and technical description of P5. This is followed by a detailed analysis of the SCRAPE framework in Section 6.3, full installation and configuration instructions in Section 6.4 and an assessment of SCRAPE's suitability for low-end CAVE implementations in Section 6.5

[2]http://www.opengl.org
[3]http://www.opensg.org
[4]http://www.openscenegraph.org
[5]http://java3d.java.net
[6]http://vpython.org
[7]http://www.openframeworks.cc

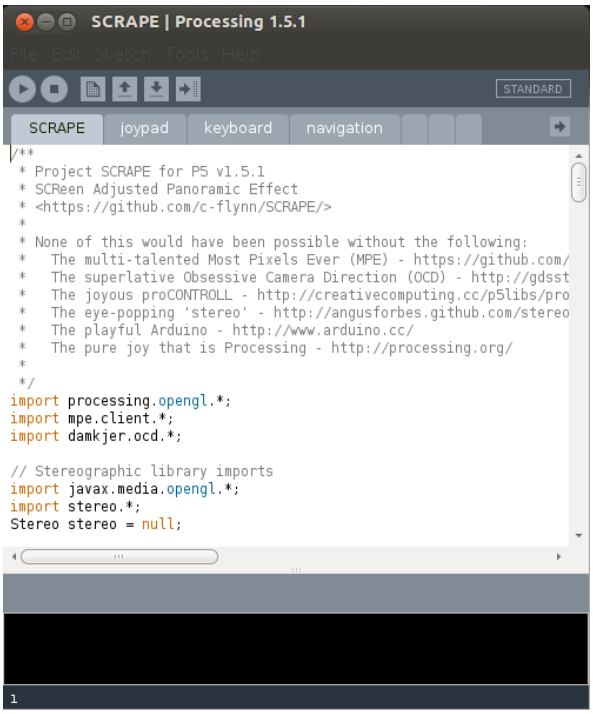


Figure 6.1: Processing Development Environment (PDE)

6.2 Processing

6.2.1 Overview

P5 provides users with a basic IDE which is better known as the Processing Development Environment (PDE) (see Figure 6.1). In reality the PDE is not much more than a notepad with some basic functions and is intended to evoke the concept of a sketchpad where users can easily begin to sketch out their ideas in code and quickly visualise them on-screen. It is for this reason that P5 applications are referred to as sketches rather than programs, with each sketch being self contained in a folder of the same name. Each sketch may contain multiple sub-sketches (which are really inner classes of the main sketch class) and these are represented as tabs on the PDE. The P5 software can be downloaded directly from the *processing.org* website and it works out of the box with no real set-up or install required. It is also compatible with Windows, Mac and Linux operating systems.

6.2.2 Versions

During the development of SCRAPE a new version of P5 was released (P5 v2.0). There are some significant changes to this version of P5 over previous versions, particularly in relation to the integration of OpenGL 2 as a core library of P5 v2.0. While the

integration of OpenGL 2 is a logical progression in the development of P5, it does pose some issues in terms of the use of active shutter 3D with the existing P5 stereo library. With this in mind, it was decided to remain with P5 v1.5.1 as any potential gains in modifying the existing system were regarded as relatively small considering the time investment required to make the necessary upgrade to SCRAPE. It is fully expected that SCRAPE will be upgraded to function fully with P5 v2.0 in due course, however, for the purposes of this thesis, P5 v1.5.1 is the version used.

6.2.3 Technical Analysis

The P5 environment is written in Java and programs written in the P5 programming language are translated to Java and run as Java programs. Java was chosen by Reas and Fry because it was seen as the best option in terms of compatibility, ease of use, portability and performance. P5 differs from Java through its graphics library and simplified programming style but fully accepts standard Java code. There is also a sister project to P5 called Processing.js[8] which is based on JavaScript and is intended to bring the power of P5 to the web. While Processing.js is an interesting project, it was not considered for SCRAPE primarily due to its incompatibility with quad-buffered stereo 3D. That being said, web based projects such as Processing.js, WebGL[9] and three.js[10] certainly have potential for the future.

P5 has essentially two modes of operation, a 2D mode and a 3D mode. For the purpose of the CAVE it is the 3D mode that is of interest and in version 1.5.1 of P5 there are essentially two ways in which a 3D sketch can be invoked. The first mode is called P3D which is the default 3D renderer developed by the creators of P5. It is more than sufficient for many 3D sketches, however, it is recommended that the other option of OpenGL is employed. This enables sketches to avail of the full power of OpenGL which is the industry standard graphics API for high performance 3D graphics. It also opens up the possibility of providing stereographic images to viewers. The render mode is determined when specifying the screen dimensions of the sketch within the size() function:

```
size(800, 600, OPENGL);
```

It is important to note that if OpenGL is specified in this function it is also necessary to import the relevant OpenGL library :

```
import processing.opengl.*;
```

[8] http://processingjs.org
[9] http://www.chromeexperiments.com/webgl/
[10] http://threejs.org/

Most sketches in P5 consist of two core functions: setup() and draw():

setup() is called once and is generally used to set some initial parameters such as the size of the sketch window, the type of renderer to use etc...

```
void setup(){
  size(800, 600, OPENGL);
}
```

draw() is called repeatedly and is used to handle the animated sketch such as rotating an object on screen. The following demonstrates a simple sketch which uses both setup() and draw() to present a rotating box to the viewer:

```
import processing.opengl.*;
float angle = 0.0f;

void setup() {
  size(400, 300, OPENGL);
}

void draw() {
  // Clear the screen
  background(0);

  // Centre, rotate and draw box
  translate(200,150,0);
  rotateY(angle);
  box(80);
  angle += 0.01f;
}
```

This basic program demonstrates the simplicity with which 3D objects can be animated and presented to the user (see Figure 6.2).

6.3 The SCRAPE Framework

Having chosen P5 as the development platform for SCRAPE, the next step was to develop a flexible framework that would meet the very specific demands of both standard and non-standard CAVE systems.

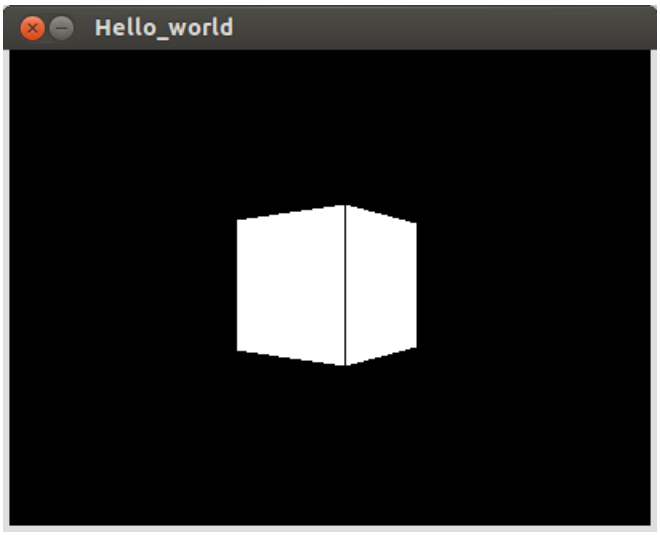


Figure 6.2: Hello World - Basic sketch demonstrating 3D rotating cube

In some respects the SCRAPE framework is similar to many typical P5 projects. All the relevant SCRAPE code is contained within a folder of the same name and within that folder there exists a number of classes (.pde files), a configuration file (mpe.ini) and a folder named *code* which contains the relevant imported libraries. Unlike most P5 projects, however, SCRAPE runs multiple sketch instances across multiple machines and screens using a range of classes and functions that are highly specific to a CAVE's unique characteristics. Figure 6.3 provides a basic illustration of the SCRAPE code structure, followed by a brief description of each of the SCRAPE classes and the specific functions they perform. **A more detailed breakdown of the SCRAPE code is provided in Appendix B.**

6.3.1 Core Classes

- **SCRAPE.pde**
  This is the main class file that initialises the program, checks and applies the necessary configurations, calculates and assigns the required camera parameters, applies the relevant camera template and calls the necessary inner classes.

- **scene.pde**
  Contains the code for a given scene. So for example, someone wanting to draw an animated box would do so in scene.pde.

- **stereoscene.pde**
  Essentially a copy of scene.pde but with modifications to enable the scene to render in stereoscopic 3D.

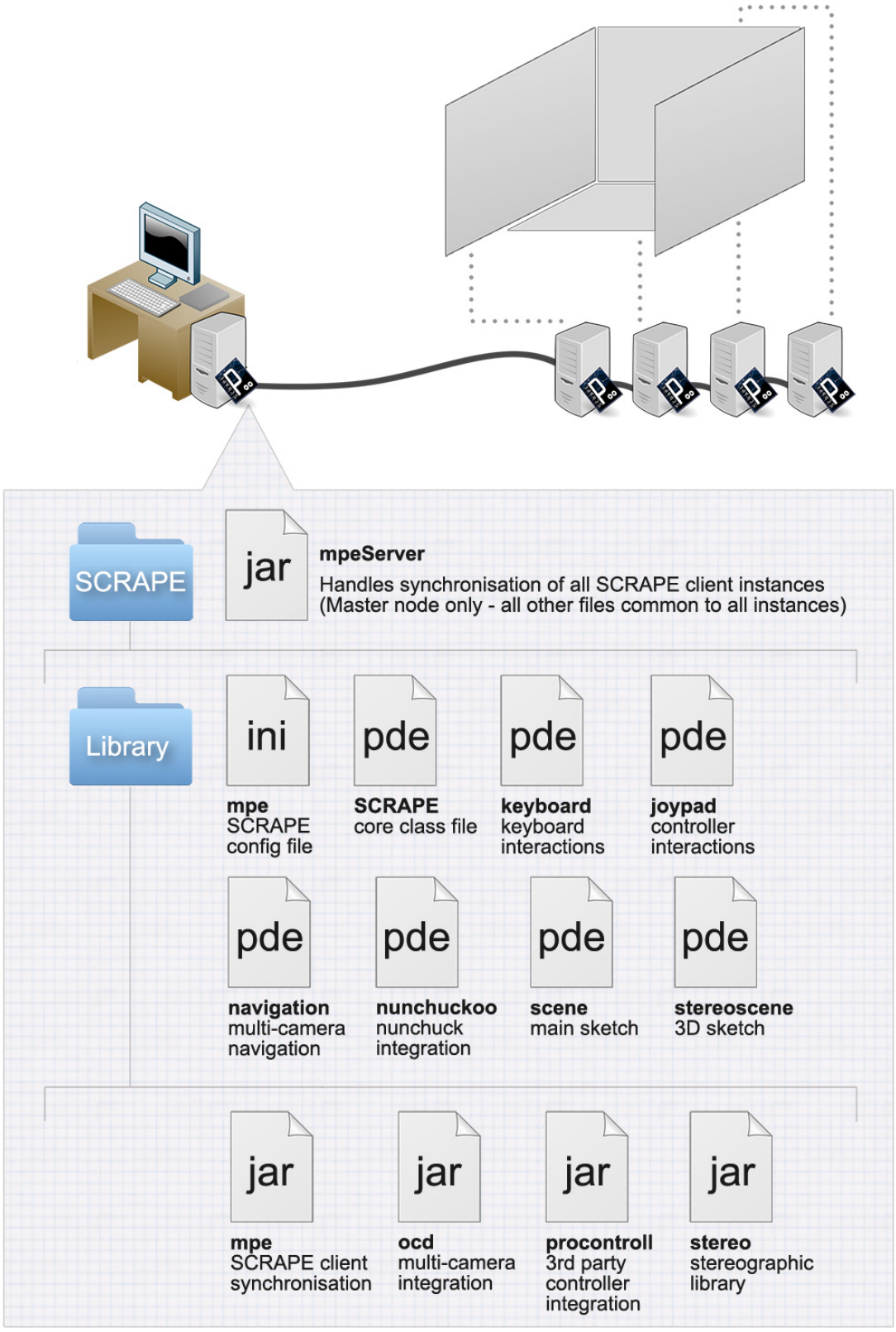


Figure 6.3: SCRAPE Code Structure

- **keyboard.pde**
  Defines what movements or actions to broadcast to other sketch instances based on keyboard navigations

- **joypad.pde**
  Checks for installed controller devices, assigns controllers (& corresponding controls) and broadcasts movements and actions.

- **nunchuckoo.pde**
  Communicates with an Arduino microcontroller and Nunchuck controller. Reads and interprets movement data received from Nunchuck controller via Arduino then broadcasts relevant actions.

- **navigation.pde**
  Applies the relevant navigation movements and scene actions based on received navigation messages and according to camera selection.

6.3.2 Additional Classes for Data Integration

- **DBconnect.pde**
  Provides the necessary code to enable SCRAPE to connect and query a standard MySQL open source database.

- **initialisation.pde**
  Enables scene variables to be initialised outside of the core SCRAPE class for cleaner code organisation. Also used for initial calls to database and threads where required.

- **DBthread.pde**
  Creates new thread for accessing database while SCRAPE is running. Making calls to a database from within SCRAPE's scene class would cause each instance to pause momentarily during data access. By creating a separate thread to call on the required database functions, scene specific variables can be updated without causing any interruption to the running of the main SCRAPE program.

Having outlined the different SCRAPE class files, the following subsections describe the key functions that these classes (both individually and in combination) make possible.

Figure 6.4: Most Pixels Ever at the IAC Manhattan
Source: (Shiffman, 2012)

6.3.3 Multi-Screen Displays

Unlike most standard P5 projects, SCRAPE is required to function across multiple screens and machines. In order to do this SCRAPE employs the use of a library called Most Pixels Ever (MPE). The MPE library[11] for P5 was developed by Daniel Shiffman alongside Jeremy Rotsztain, Elie Zananiri & Chris Kairalla as part of a New York University project to display students' interactive projects on a large multi-screen video wall (measuring 12' x 120') at the IAC building in Manhattan (see Figure 6.4). Daniel Shiffman has been a key figure in the P5 community for many years and has contributed significantly not only in terms of his code contributions with libraries such as MPE but also through the publication of his book *learning processing*[12] which is a 'must have' for anyone looking for a useful introduction to P5.

Essentially, MPE enables SCRAPE to be spanned across multiple screens by enabling multiple instances of the same sketch to run synchronously on either the same machine or on separate network connected machines. There are two key elements to MPE:

1. The first is to ensure that all instances of the sketch run in time synch. Each sketch communicates with a dedicated service which ensures that sketches render the same scene at exactly the same moment. So, in the case of the 'Hello_world' box sketch if there are three instances of Hello_world running and displaying the same scene segment, all three instances will run their sketch in perfect synch with the others (see Figure 6.5).

2. The second element is to allow each sketch to render a specific portion of a scene. So for example, if a scene contains a long rectangular object such as a table that

[11] github.com/shiffman/Most-Pixels-Ever/
[12] www.learningprocessing.com/

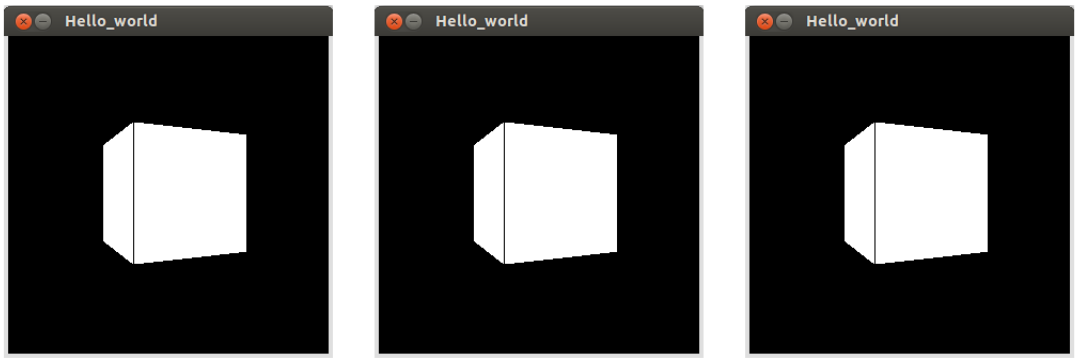


Figure 6.5: Multi-Sketch Synchronisation

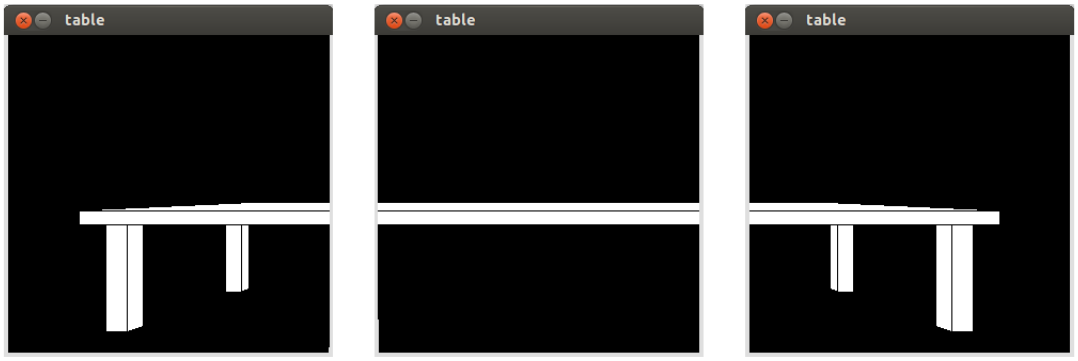


Figure 6.6: Multi-Sketch Segment Rendering

needs to be rendered across multiple screens then MPE can instruct each sketch which part of the scene it needs to display (see Figure 6.6). It is important to note that each sketch is rendering the full scene but only displays the portion specified. This could have performance implications in scenes containing many objects or objects with high polygon counts but most CAVEs (including low-end set-ups) are likely to have sufficiently capable processors and graphics card for this not to be a significant issue. Many CAVEs incorporate the use of multiple workstations which can also be used to help spread the load.

In order to achieve this, a dedicated MPE service is required. This service is written in Java and runs directly from the command line.

```
$ java -jar mpeServer.jar -framerate30 -screens3
```

Each SCRAPE instance then references a separate configuration file called mpe.ini which points back to the MPE service (on whatever machine it is running). This allows each instance to obtain synchronisation with the others via the MPE service and also to inform the service which instance it is communicating with, what the overall screen dimensions are, what dimensions each individual screen is and the offset by which they are rendered. The following example demonstrates a typical (non-SCRAPE) mpe.ini file configuration:

```
//−−− MPE Configurations −−−//
// Client ID (0,1,2,etc.)
id=2;

// Server address
server=192.168.1.x;

// Server port
port=9002;

// What are the client window dimensions?
localScreenSize=1024,768;

// Where is the client located in the master dimensions?
localLocation=2048,0;

// What are the master dimensions?
masterDimensions=3072,1024;

// Turn to 0 to turn off all the console printing
debug=0;
```

As traditional CAVE environments are made up of a multitude of different screens, MPE provides a relatively simple method of breaking sketches up into separate components and ensuring synchronisation. This, however, is not sufficient for a CAVE. MPE was primarily developed for displays sitting side by side or on top of one another on a flat 2D surface. It was never designed for screens sitting on different axes and sitting at right angles to one another and it is not designed to render images from different positions in full 3D space. To develop a fully functional CAVE system it is essential to be able to render views of a scene from different angles and to be able to manipulate and blend screen images in such a way as to provide a seamless representation of a scene across multiple screens (as if the screens were one large amalgamated window onto the world). To create this illusion, each sketch which renders the scene for an individual screen needs to be allocated it's own camera which can then be manipulated in multiple ways. As there are a total of six screen positions possible in a CAVE (front, back, left, right, top and bottom), SCRAPE needs to be able to accommodate a maximum of six separately configurable cameras. SCRAPE provides for this by including an additional configuration option to the standard mpe.ini configuration file as follows:

```
// Specify screen option: front,stern,left,right,top,bottom
wall=front;
```

Due to the fact that SCRAPE makes use of cameras, the 'localLocation' screen position parameter is no longer required and is removed from the standard configuration file.

```
/##––– Removed –––##/
// Where is the client located in the master dimensions?
localLocation=2048,0;
```

As each SCRAPE instance has its own configuration file, it is therefore able to assign it's own specific camera position. For each sketch instance that is launched, the associated SCRAPE.pde class file will apply one of six possible camera assignments based on the camera option selected. It will also gather information from the rest of the configuration file to determine screen size, camera axis and camera position, as well as calculate and apply the horizontal and vertical field of view, the aspect ratio and required clipping ranges. While it is possible to work with cameras using P5's camera() and perspective() functions, SCRAPE uses Kristian Linn Damkjer's Obsessive Camera Direction (OCD)[13] library which provides a more succinct method of managing multiple cameras.

Please note: The OCD library in its original format does not (at the time of writing) fully support camera placement on all virtual scene axes. Due to this the OCD library was modified by the author and recompiled in order to resolve this issue. This modified library is included as part of the overall SCRAPE package.

6.3.4 Scene Generation

By using P5, SCRAPE enables users to easily develop 3D worlds using either standard JAVA code or the P5 programming language, both of which are widely used, well documented and accessible to users of of varying programming abilities. In general, SCRAPE scenes are built similar to standard P5 3D sketches with only the inclusion of references to some CAVE specific functions distinguishing it from a standard 3D sketch. These are key strengths of SCRAPE in comparison to some other frameworks which require users to code in C and reference the OpenGL API directly, which, although powerful, can be very time consuming and difficult for non-expert users.

[13] gdsstudios.com/processing/libraries/ocd/

6.3.5 Stereographics

The ability to display VWs in Stereoscopic 3D is an important function of any CAVE software application and SCRAPE is no exception. In order to accommodate stereo displays, SCRAPE includes a stereo configuration option to the mpe.ini configuration file as follows:

```
// Set to 'on' to activate stereographics
activeStereo=on;
```

If the 'activeStereo' option is activated, SCRAPE will use the stereoScene.pde class file which wraps the entire VW scene code into one render function. SCRAPE then uses a library named 'stereo' (created by Charlie Roberts and Angus Forbes[14] and based on code by Paul Bourke[15]) which calls upon the render function for both the left eye and right eye on each frame loop and then applies the relevant OpenGL functions for active, passive or anaglyph stereo. Although graphics cards increasingly support stereo 3D, it is by no means a default feature and therefore, any potential CAVE set-up will need to verify compatibility before activating SCRAPE in 3D

6.3.6 Device Interaction

As discussed in Chapter 4, interaction devices play an important role in any CAVE system. In order to maximise the number of input devices that can be used such as keyboard, mice, joypads, joysticks etc., as well as the ease with which they can be accessed, SCRAPE enables the integration of a wide range of USB devices. To do this, SCRAPE firstly incorporates the following two configuration options in the mpe.ini configuration file:

```
// Set to 'on' if adding controllers
controller=on;

// Controller name
controllerName=Saitek Saitek P2900 Wireless Pad;
```

For each instance of SCRAPE that activates the 'controller' setting in the configuration file, the joypad.pde class can then perform the following key functions:

- Detect and list connected devices

[14] https://github.com/CreativeCodingLab/stereo/
[15] paulbourke.net/

- Detect and list accessible parameters
- Assign a device
- Assign device parameters
- Read device actions
- Broadcast actions to all other SCRAPE instances

Although each instance can assign its own controller, typically only one SCRAPE instance will activate a device at any given time. To assist in the detection and assignment of interaction devices, SCRAPE incorporates the use of a library called proCONTROLL[16] developed by Christian Riekoff. This enables SCRAPE to check for all installed input devices and their controllable actions. Once these are accessed, SCRAPE can then assign the required device and apply the necessary parameters. A broadcast function is also used to instantly communicate any movements or actions to other sketch instances. This ensures that movements or actions are applied to all SCRAPE instances and that everything remains in sync across all screens.

As detailed in Chapter 4, SCRAPE also supports a novel interaction device entitled 'Nunchuckoo' which provides for the integration of a Wii Nunchuck controller and an Arduino microcontroller. This option is incorporated in the mpe.ini configuration file as follows:

```
// Set to 'on' if adding arduino and nunchuck controller
nunchuck=on;
```

Activating this option calls upon the nunchuckoo.pde class which is specifically developed to access serial data from devices such as the Arduino. The nunchuckoo class performs a series of functions. Firstly it checks to see if any serial device is connected to the computer and if so prints out a list of connected devices and their associated ports to the P5 console window. Armed with this information the user can identify which port the Arduino is attached to and specify the appropriate port. The nunchuckoo class will then read the incoming data, translate controller action integer arrays into scene actions and broadcast navigation instructions to all other SCRAPE instances to create seamless VW interactions.

In the situation where no USB controller devices (including the Nunchuckoo) are connected, or are simply not required, SCRAPE defaults to standard keyboard input through the basic keyboard.pde class. This broadcasts navigation instructions to all

[16] creativecomputing.cc/p5libs/procontroll/

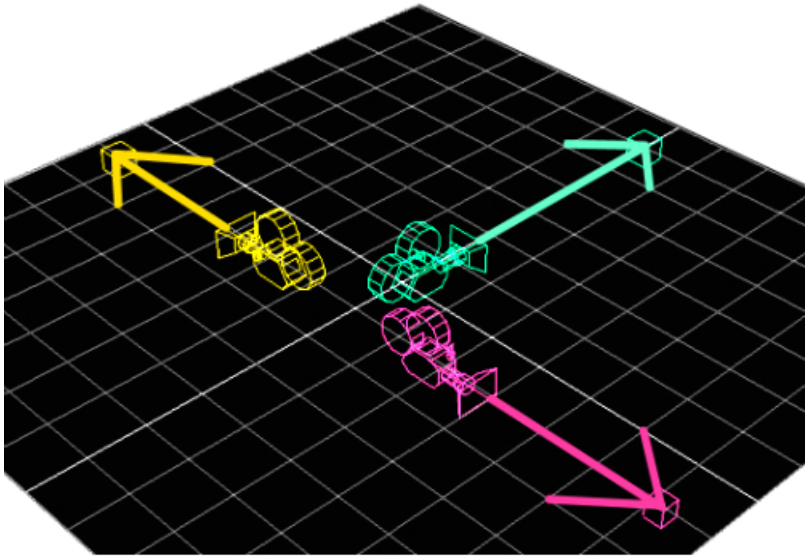


Figure 6.7: Cameras moving relative to themselves

instances in order to facilitate basic control functionality which may be useful during set-up, testing or demo scenarios.

6.3.7 Multi-Camera Navigation

One of the primary objectives of SCRAPE is to ensure that users are presented with a seamless virtual display across multiple screens in three dimensions at all times. The use of multiple cameras sitting on different axes and orientations within 3D space is key to this. There is, however, a complication. Whenever a user interacts with a SCRAPE instance via a keyboard, USB controller or Nunchuckoo device, it will automatically broadcast these actions to all SCRAPE instances. So, for example, if we are navigating around a VW and we tilt the Nunchuckoo downward in order to move forward relative to the centre screen, then a 'forward' motion instruction is sent to all SCRAPE instances. If each instance was to employ a camera sitting on the same axis, with the same orientation, this would not be an issue, however, as SCRAPE uses cameras sitting on different axes pointing in different directions, then a forward motion by all cameras will cause the cameras to decouple (see Figure 6.7). This would instantly break the illusion of a seamless VW. To prevent this from happening, SCRAPE provides the navigation.pde class which ensures that each camera moves according to the axis upon which it sits and its primary orientation. So, for example, navigating forward on a CAVE centre screen will cause it's associated camera to move **forward** along the z axis, whereas, the cameras associated with the CAVE's side screens will move **sideways** along the z axis and retain their primary orientation (see Figure 6.8). This ensures that all camera movements remain in sync at all times.

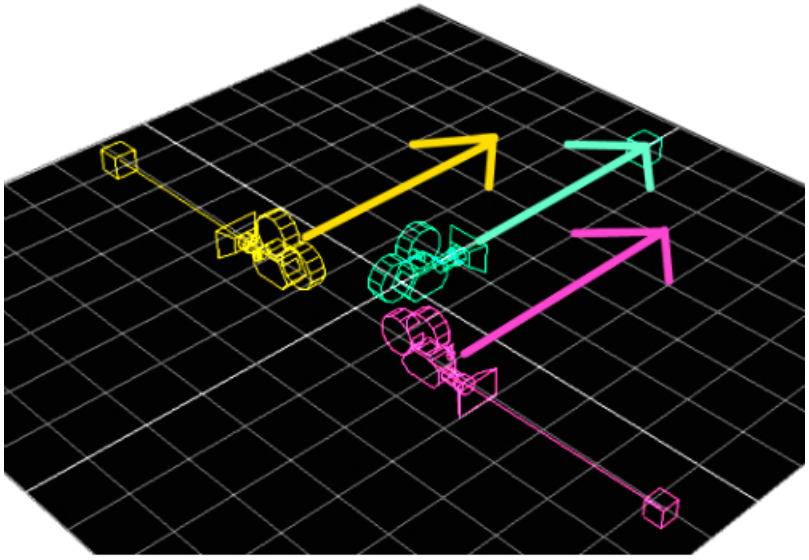


Figure 6.8: Cameras moving relative to axes and primary orientation

6.3.8 Data Access

If users wish to incorporate data from a MySQL open source database into their VW then SCRAPE provides them with the means to do so. This is a basic requirement for practically all data visualisations and therefore an important addition to the core SCRAPE code. Through the use of the DBconnect.pde class file, users can specify a database, provide the necessary credentials and define the required queries. Unfortunately, calling upon the DBconnect class directly from within the main scene or stereoscene classes will cause the VW to pause momentarily each time the database is accessed. In order to address this issue, SCRAPE provides the DBthread.pde class file which creates a separate P5 thread. It is this thread which then calls upon the DBconnect class and updates the necessary global variables without causing any interruption to the main sketch.

In addition to the two key database access classes, SCRAPE also provides the initialisation.pde class file. This is used to initialise scene variables outside of the core SCRAPE class, provide for better code organisation, make the initial calls to database functions and launch the database thread on start-up. While not used exclusively in relation to database initialisation and associated variables, it performs a useful function in relation to VWs that require access to stored data.

A practical implementation of incorporating live data into a SCRAPE VW is demonstrated in Chapter 7 where the CASALA CAVE is used to visualise live and historic electricity usage readings from a range of buildings and homes across the local community.

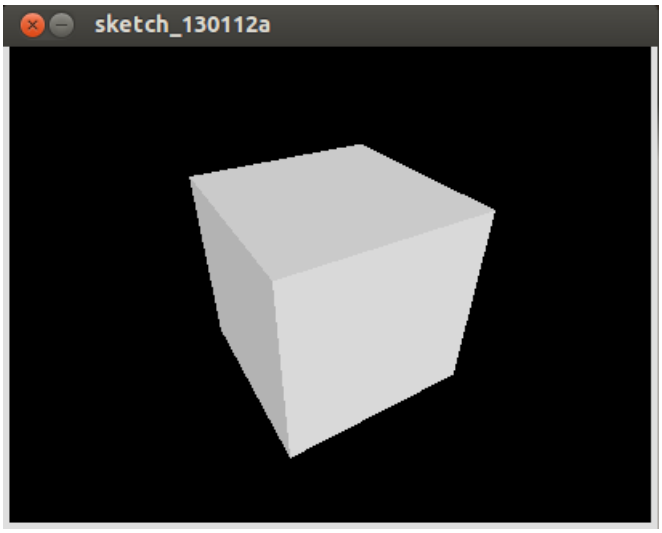


Figure 6.9: Processing set-up test image

6.4 SCRAPE Installation and Configuration

Given the inner workings of SCRAPE, this section describes how SCRAPE can be installed and configured in a CAVE (of up to six screens). The instructions below apply to both Windows, Linux and Mac operating systems.

6.4.1 Key Steps

6.4.1.1 Step 1

Download and install P5 version1.5.1 from the processing.org[17] website onto each of the machines that is used to serve images to the CAVE walls/screens. It is highly recommended that the 32bit version of P5 is used as a starting point regardless of the operating system being used.

In order to test that P5 works on any given machine simply launch the program, paste the code below into the blank sketch code window then click on the run button (Ctrl+R). All going well, a rotating cube (similar to Figure 6.9) should appear on screen.

Test sketch

```
import processing.opengl.*;
float angle = 0.0f;
```

[17] processing.org/download/

```
void setup() {
  size(400, 300, OPENGL);
}

void draw() {
  background(0);lights();
  translate(200,150,0);
  rotateX(−PI/5);
  rotateY(angle);
  noStroke();
  fill(255);
  box(120,120,120);
  angle += 0.01f;
}
```

6.4.1.2 Step 2

Download SCRAPE_v1.5.1 from **https://github.com/c-flynn/SCRAPE** onto each machine. Open the 'SCRAPE_ v1.5.1' folder and copy the 'SCRAPE' sub-folder to the P5 Sketchbook path. P5 sketches are always stored in the 'sketchbook' folder and the sketchbook path can be viewed/modified by launching processing and going to *File > Preferences*.

If individual CAVE walls/screens share a workstation then install one copy of P5 on the machine and run multiple instances by placing the SCRAPE code in separate sub-folders within the 'sketchbook' folder. For example:

- *path/to/sketchbook > SCRAPEcontainer > leftScreen > SCRAPE*
- *path/to/sketchbook -> SCRAPEcontainer > frontScreen > SCRAPE*
- *path/to/sketchbook -> SCRAPEcontainer > rightScreen > SCRAPE*
- *path/to/sketchbook -> SCRAPEcontainer > floorScreen > SCRAPE*

Double-clicking on the 'SCRAPE.pde' file within each 'SCRAPE' folder will open a separate P5 instance. If a separate command workstation is used to manage the CAVE screens then P5 can be installed there as well. SCRAPE works with up to six different screen views but in theory there is no limit to the number of SCRAPE installations.

6.4.1.3 Step 3

In order for SCRAPE to function correctly it is necessary to run a server application so that all the individual SCRAPE sketches can communicate and stay in synchronisation with one another. The server application can be on any one of the workstations that are used to render a SCRAPE sketch or on any other machine as long as they are all on the same network.

Download the file mpeServer.jar from within the 'SCRAPE_v1.5.1' folder. The file can be placed anywhere but if P5 is pre-installed on the machine then it is a good idea to place it in the *processing/lib* directory. Open a command line terminal then navigate to the folder containing 'mpeServer.jar' and run the following command:

```
$ java −jar mpeServer.jar −framerate30 −screens5
```

Be sure to modify the 'framerate' and 'screens' options as required. In this example the configuration is for a four walled CAVE using one workstation per CAVE screen and a master node, therefore five screens have been specified. The option '-debug1' can also be added to debug any issues. Once the command is run a message similar to the following should appear:

```
WallServer: framerate = 30, screens = 5, debug = false
Starting server: ubuntu−home/127.0.1.1 9002
```

IMPORTANT: All machines that intend running at least one instance of SCRAPE need to be able to communicate to the server node. A good way to test this is by logging on to each machine and ensuring that they can telnet to the server node's IP address on port 9002:

```
$ telnet 192.168.1.x 9002
```

6.4.1.4 step 4

Before launching SCRAPE it is necessary to modify the 'mpe.ini' configuration file for each instance. The 'mpe.ini' file included with SCRAPE is an extension of the MPE configuration file and additional SCRAPE configurations as outlined previously in this chapter. The file is located in the 'SCRAPE' folder and looks similar to the following:

```
//−−− SCRAPE Configurations−−−//
// Specify screen. Options are: front,stern,left,right,top,bottom
wall=front;
```

```
// Set to 'on' if adding controller
controller=off;

// Controller name
controllerName=Saitek Saitek P2900 Wireless Pad;

// Set to 'on' if adding arduino and nunchuck controller
nunchuck=off;

// Set to 'on' to activate stereographics.
activeStereo=off;


//--- MPE Configurations ---//
// Client ID (0,1,2,etc.)
id=0;
// Server address (localhost for testing)
server=localhost;

// Server port
port=9002;

// What are the screen window dimensions in pixels?
localScreenSize=1024,768;

// What are the master dimensions?
masterDimensions=1024,768;
```

6.4.2 Configuration Checklist

Most of the configuration options in the 'mpe.ini' file are self explanatory but there are a few important considerations to be aware of:

- Each instance must have a unique client ID. Start with 0 and work your way up.
- Modify the 'server' address option to the appropriate IP.
- The server 'port' option can be modified if required.

- The ‘localScreenSize’ and ‘masterDimensions’ will be the same except for floor screens and ceiling screens. This is explained in more detail in Section 6.4.3
- Only activate the ‘controller’ or ‘nunchuck’ options on the machine that a controller is connected to.
- If planning to use the ‘activeStereo’ option, ensure it is supported by the associated computer’s graphics cards. Currently SCRAPE is only tried and tested to work with active shutter glasses.
- If running SCRAPE on a command workstation, any ‘wall’ option can be chosen regardless of whether or not it has also been selected for another machine. It is only the client ID’s that must remain unique.
- If running SCRAPE on a command workstation it may only be required to display a small window on-screen. If so, simply specify smaller ‘localScreenSize’ and ‘masterDimensions’ sizes and ensure that P5 isn’t run in ‘Present’ mode.
- For the initial SCRAPE activation it is highly recommended that the ‘controller’, ‘nunchuck’ and ‘activeStereo’ options are set to ‘off’ until the initial launch has been confirmed successful.

6.4.3 Camera Configurations

SCRAPE has been designed to manage the different cameras required for a CAVE with little input required from the CAVE operator. The key concern is to ensure that the ‘localScreenSize’ and ‘masterDimensions’ settings in the ‘mpe.ini’ file are set correctly. For the most part these settings will be the same, however, in the case of ‘floor’ and ‘top’ screens they are likely to be slightly different. The settings for these screens should be as follows:

- **localScreenSize** will be the local screens resolution (which should be in proportion to its physical dimensions)
- **masterDimensions** will be the same resolution as the horizontal screens (e.g. front, left, right and back)

So, for example, if the ‘front’ screen has a physical screen size of 4 x 3 metres and a resolution of 1024 x 768 pixels and a floor screen which is 4 x 4 metres with a resolution of 1024 x 1024 pixels, then the ‘floor’ screens ‘mpe.ini’ file settings will be as follows:

front

- localScreenSize=1024,768;
- masterDimensions=1024,768;

floor

- localScreenSize=1024,1024;
- masterDimensions=1024,768;

These settings will ensure that the 'floor' screens field of view calculations are set correctly in relation to the other horizontal screens and that the images displayed on all screens are correctly aligned.

6.4.4 Troubleshooting and Additional Information

Additional installation and configuration instructions are fully available on SCRAPE's github repository[18]. This provides a comprehensive wiki and issues list to assist users in deploying SCRAPE. As a general rule, however, the best approach to setting up SCRAPE for the first time is to deploy and test on one machine initially with all controller and stereo options deactivated and build from there. It is also recommended that a basic test sketch is used until all framework functionalities have been fully tested. By using this approach any issues should be easily identified and by consequence easier to address.

6.5 Low-End CAVE Implementation

Up to this point, the CASALA CAVE has been used as the primary test platform for SCRAPE. While this is highly useful, it does not address one of the key objectives of SCRAPE i.e. the development of an open source CAVE framework that is suitable for both high-end and low-end CAVE implementations.

The CASALA CAVE is a high-end CAVE implementation and therefore it is not all that surprising to discover that SCRAPE runs effortlessly across a cluster of quad core Xeon workstations each with 12GB of RAM and an NVIDIA Quadro graphics card. In order to ensure that SCRAPE is suitable for low-end CAVE implementations it was extensively tested on a relatively standard PC (Intel Core 2 Duo + 4GB RAM +

[18]https://github.com/c-flynn/SCRAPE

NVIDIA GeForce GTX) to simulate a six screen CAVE. In other words six instances of SCRAPE running at the same time on the same machine. While this is at the extreme of low-end CAVE implementations it helps to demonstrate the possibilities of SCRAPE for low-end set-ups. Six copies of the main SCRAPE folder were placed in six key folders (named: Front, Back, Left, Right, Top, Bottom) within a main folder named 'SCRAPE-low-test'. Each sketch instance used a copy of the same basic test scene and a Nunchuckoo controller was used as the main control device. The mpe.ini configuration file was modified as necessary for each instance and the MPE service was then activated for a total of six screens.

```
$ java −jar mpeServer.jar −framerate30 −screens6
```

Each sketch instance was then launched, one by one, until all six instances were running. Figure 6.10 illustrates how the images are displayed in sync across six windows which represent each CAVE screen.

The test VW ran smoothly across all screens and although the test sketch is basic and the test windows are small, it helps demonstration of the light-weight nature of SCRAPE. It also helps support the argument that SCRAPE is suitable for low-end CAVE environments. A short video of this test is available on the Carl Flynn YouTube channel[19]

In order to test SCRAPE further, a more intensive sketch test was implemented on the same desktop as previously (Intel Core 2 Duo + 4GB RAM + NVIDIA GeForce GTX). This test used 4 instances of SCRAPE to replicate a more typical CAVE scenario and involved increasing the number of polygons per sketch until such time that performance slowdown became noticeable. In this test it was possible to run with approximately 36,000 polygons per instance (i.e. 144,000 polygons in total) at a frame rate of 30 frames per second before the on-screen display became noticeably jittery. While SCRAPE is not expected to compete with the performance of some other dedicated CAVE frameworks, it does, nonetheless, indicate that SCRAPE has the capacity to provide sufficient performance even with low specification hardware. See figure 6.11 for an illustration of the performance test in action.

Note: Normally sketches are displayed at different angles to each other in a cube shaped environment and not on a flat surface. This accounts for the distorted angles which are apparent on the floor plane between screens in Figure 6.10 and Figure 6.11. This is not an error but rather an important feature to ensure that SCRAPE functions correctly within a CAVE.

[19]https://www.youtube.com/watch?v=eqK2TW1vxf4

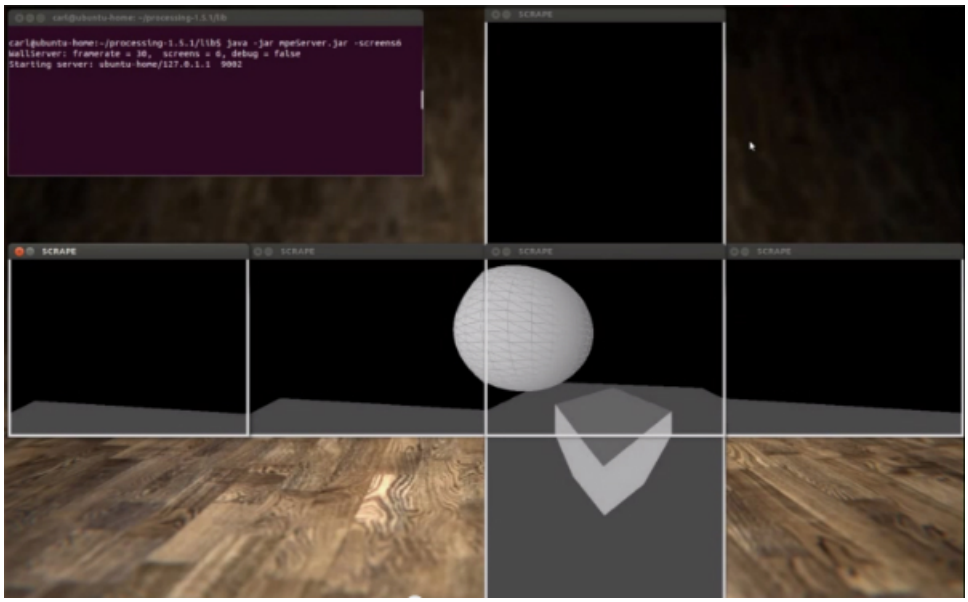


Figure 6.10: SCRAPE running across six screens on standard desktop PC

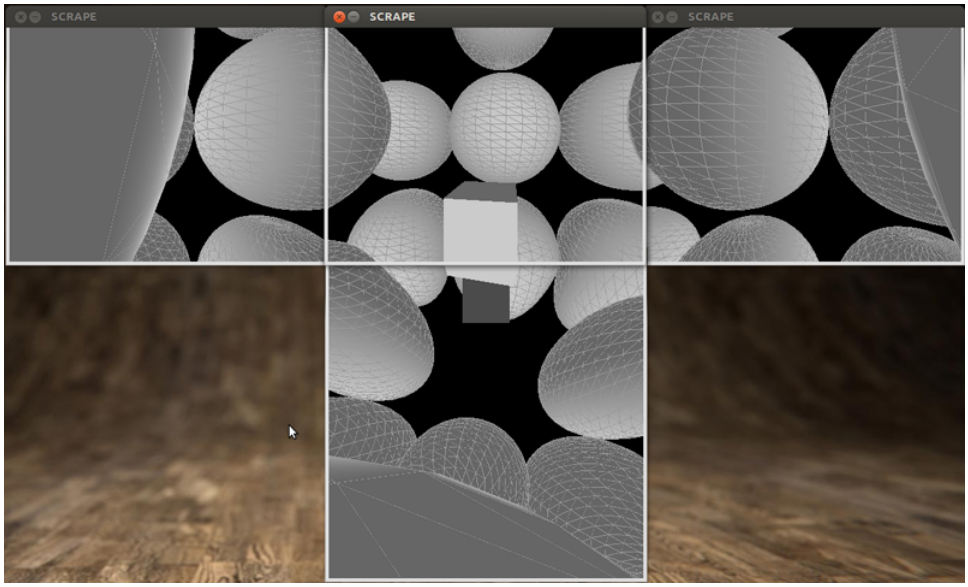


Figure 6.11: SCRAPE performance test on standard desktop PC

6.6 SCRAPE Framework Comparisons

In Chapter 5, Table 5.1 was presented in order to facilitate the comparison of key CAVE frameworks across a range of different attributes. In Table 6.1 the original table from Chapter 5 is replicated, but this time, it also includes the relevant attributes from SCRAPE. Using this table it is possible to compare it against other key frameworks across a range of different measures and in particular in relation to areas of key interest such as cost and indicators of accessibility.

CAVE Framework Comparison Table							
	Open Source	Cost	GUI	Component Sofware	Complexity	Platforms	Language
CAVElib	No	$\sim$ €3000 - €5000 per node + 10% annual fees	No	OpenGL OpenGL Performer Open SG Open Scene Graph	High	Windows Linux	C
VRJuggler	Yes	€0	No	OpenGL OpenGL Performer Open SG Open Scene Graph	High	Windows Linux Mac OS X	C++
VR4MAX	No	$\sim$ €3300 per node + €1900 annual fees	Yes	3ds Max	Medium - Low	Windows (Dev + Runtime) Linux (Runtime Only)	Lua
MiddleVR	No	$\sim$ €4000 per node + 20% annual fees	Yes	Unity	Medium	Windows Linux Mac OS X	UnityScript C# Boo
SCRAPE	Yes	€0	Basic IDE	Processing	Low	Windows Linux Mac OS X	Processing

Table 6.1: CAVE Framework Comparisons

6.7 Conclusion

This chapter described the motivation behind the development of SCRAPE, characterised the technologies that underpin it, looked in detail at how the technology functions, provided the necessary resources to set it up, demonstrated its potential to function on low-end CAVE systems and compared it in relation to other frameworks across a range of different attributes. All in order to provide the reader with a comprehensive understanding of exactly what SCRAPE is and how (& where) it works.

In summary, SCRAPE rivals and improves upon the existing 'state of the art' in a few key areas:

- Ease of implementation.
- Accessibility for both professional and novice CAVE developers.
- Flexibility to function in all types of CAVE Environments.
- Fully open source.
- Access to large support resources through use of Java and P5.
- Interaction device integration.

The following chapter expands upon our understanding of SCRAPE by providing a practical illustration of how SCRAPE is currently employed within the CASALA CAVE in order to visualise and interact with energy data as part of a local community energy efficiency project.

Chapter 7

Real-World Application of SCRAPE

7.1 Introduction

As stated previously, the CASALA CAVE is utilised in support of a range of different research projects. One of these projects is called *Sustainable Energy Efficiency Dundalk* or SEED for short. This chapter presents an overview of SEED and demonstrates how SCRAPE is being employed to visualise SEED data and contribute to our understanding of energy usage across a local community.

7.2 Project SEED

7.2.1 Overview

In 2012, CASALA set up the SEED project to leverage the learnings, networks and relationships from previous local projects and apply them to energy related ventures. In September of the same year, SEED was awarded a 'Better Energy Communities' pilot project by the Sustainable Energy Authority of Ireland (SEAI)[1]. The 'Better Energy' programme is Ireland's national upgrade initiative to retrofit building stock and facilities to high standards of energy efficiency, thereby reducing fossil fuel use, running costs and greenhouse gas emissions. The purpose of these projects is to support and pilot innovative approaches to achieving high quality and efficient delivery of improvements in energy efficiency within Irish communities (Howley et al., 2012) (KEMA, 2008) (McAuley and Polaski, 2011). For this pilot, SEED (in conjunction with community partners) was required to fully implement energy saving changes and install

[1] http://www.seai.ie/Grants/Better_Energy_Communities

monitoring equipment in a range of residential and public buildings across the local community. The pilot was fully implemented by December 2012, providing live energy data readings from all community participants directly to CASALA's data acquisition system at DkIT.

7.2.2 Related Work

Energy security has been an important research topic for many years, with reports from agencies such as the SEAI highlighting increasingly stringent energy related goals. In order to ensure energy security for the future, it is important to understand how energy can be generated and used as efficiently as possible. In the book 'Energy Efficiency: Towards the End of Demand Growth' (Kesting and Bliek, 2013), Stephanie Kesting and Fritz Bliek describe a smart energy pilot project in the Netherlands where dynamic real-time pricing is employed to optimize the energy system in a decentralised way. End users are both producers and consumers of energy in a market system that allows them to either use energy themselves when needed or sell excess energy to their neighbours in a system known as a smart grid community. In a key paper on energy consumption (Darby, 2006), Sarah Darby demonstrates how clear feedback educates energy users to be more efficient over a long period of time and how direct feedback in combination with frequent, accurate billing can provide sustained energy reductions. In the paper 'Real-Time Recognition and Profiling of Appliances through a Single Electricity Sensor' (Ruzzelli et al., 2010), Ruzzelli et al. present a single sensor system that can recognise multiple electrical appliances in real-time. It demonstrates the huge potential for easy-to-use, low-cost sensors & systems that can provide important feedback on energy usage and help promote energy awareness and generate efficiencies. Another article on energy efficiency in Sweden (Henryson et al., 2000) also argues that feedback, while important, does not guarantee efficiencies in of itself and that it is paramount to continue to educate and inform users about their energy usage as well as look at their motivations to change. Important research studies such as these are helping to inform decisions on projects such as SEED and help guide its development into the future.

7.2.3 Project Details

As part of the 'Better Energy Communities' pilot, SEED committed to delivering significant energy efficiency upgrade works, increased renewable energy generation and implementation of an integrated energy data collection platform across a group of ten

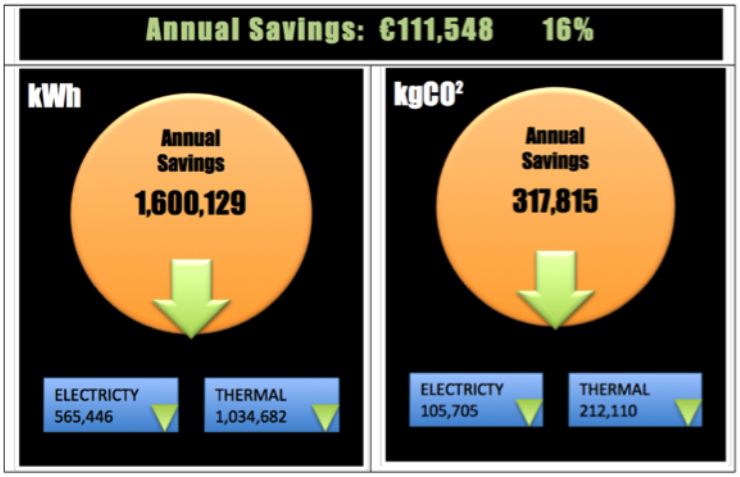


Figure 7.1: Projected Annual Savings
Source: (Flynn et al., 2013)

homes, a community house, a secondary school as well as a third level college. The actual locations are as follows:

Residential Community

- Muirhevna Mór, 10 homes in conjunction with Dundalk Town Council and Electric Ireland
- Croabh Rua, Community House, Muirhevna Mór in conjunction with Youth Work Ireland

Public Buildings

- O'Fiaich Secondary School in conjunction with Louth VEC and the Department of Education
- Carrolls Building, DkIT

The specific deliverables defined within the project required an investment of €808,270 by the partners to achieve an overall energy target of 16% in energy efficiency, and deliver savings of 1,600,129 kWh and 317,815 kgCO2 per annum (see Figure 7.1). The payback period has been calculated at 7.25 years, with a positive Net Present Value (NPV) of €707,704 into perpetuity at a discount of 4%. A result which, if met, represents excellent value for money for the level and scale of works undertaken. On project initiation, Building Energy Rating (BER) assessors were appointed to establish the baseline energy ratings and this was repeated on completion to establish the reduction in BERs for all project buildings.

In order to confirm if the 16% energy savings target is being met, accurate measurements are crucial to the verification of such savings. This is achieved by monitoring

energy usage through the use of smart metering[2] and the measurable improvement in BER ratings for the domestic and non-domestic buildings involved in the projects.

The overall benefits are multi-faceted and include empowering the local community in terms of managing energy efficiency and cost effectiveness; the educational and research & development community in terms of applied research and Living Lab learnings; the local and regional economy in terms of increased access to research & development funding (and related job creation) as well as contributing to a model of leadership for future energy related projects.

7.3 SCRAPE Application of SEED

Having characterized the SEED project, this section explains how SCRAPE is being used to visualise SEED data in a CAVE.

Before any data is visualised, it must first be processed in order to make the data accessible and readable to SCRAPE. In its raw form, the large amounts of data collected as part of SEED (over 300,000 records per day) is impractical to work with directly. In order to overcome this, the CASALA data acquisition system firstly organises the relevant data into specific tables and fields within the database that are quick and easy to access, and only contain the necessary data for visualisations. Of particular interest are the current energy readings and total readings from each particular meter (or zone) which can be broken down into type such as gas or electricity. Once the data is processed, SCRAPE is then able to assign data from specific tables and fields to its relevant on-screen representation, providing viewers with current and historic output of SEED data in real-time.

In its current implementation, SCRAPE represents specific fields from the SEED database tables as individual coloured cubes within the visualisation (see Figure 7.2). Each of the relevant database table fields contains information such as the current and total energy readings of a particular SEED energy meter (or zone) and this is fed back to SCRAPE. Using this information, SCRAPE then displays the current and historic readings for a particular meter (or zone) within a cube and uses colour variations to highlight readings of individual cubes dynamically. The cubes also have the potential to increase or decrease in size relative to their energy usage which can also be used to help identify and compare usage levels. Importantly, each cube belongs to an array of cubes that represent the overall reading for a parent grouping (e.g. electricity readings

[2] A smart meter is an electronic device that can measure the consumption of energy, record and store more information than a conventional meter and provide real-time information to the customer on usage and costs.

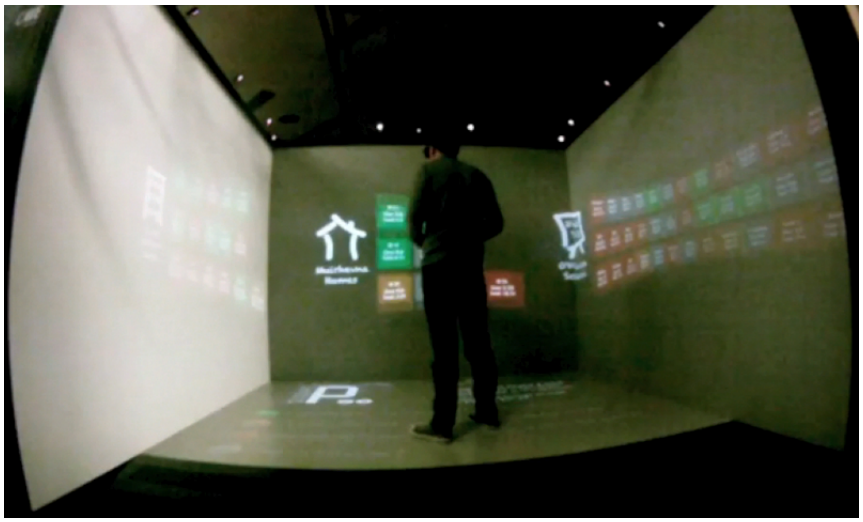


Figure 7.2: SCRAPE visualising SEED data

of classrooms in a school). These arrays can then be visualised in blocks alongside other groupings for comparison and analysis. Using this system, it is therefore possible to compare multiple groupings and understand at a glance the energy usage of both related and unrelated data on different levels. So, for example, it is possible to examine the electricity usage of an individual home in isolation or in relation to other homes by analysing different aspects of their visual representations such as size, shape and colour. This can help identify those homes that may be using energy more efficiently or inefficiently than others and help understand the different behaviours that lead to these variations. It can also help to demonstrate how material disparities (e.g. electrical devices, insulation types, building materials, heating systems etc.) between buildings impact usage.

This type of analysis can also be applied across different building types by comparing the energy use of an individual Muirhevna Mór home to, say, that of the O'Fiaich school or DkIT, which may provide an interesting comparator of energy usage across entirely different buildings. The levels of immersion provided by SCRAPE within the CAVE enables users to contrast and compare the large number of SEED data readings side by side, one on top of the other and layer by layer, providing the viewer with a unique visualisation of community energy data.

As an educational tool, being able to visualise community energy data using SCRAPE offers CASALA researchers an opportunity to engage with the students, teachers and householders of both O'Fiaich, DkIT, Muirhevna Mór and GNH. SCRAPE can help demonstrate how material changes and behavioural adjustments can make a significant difference in how energy is consumed in a way that is very different to standard application reports or web dashboards. In short, it offers a unique tool to demonstrate the tangible benefits that SEED is achieving and provide impetus for others to follow.

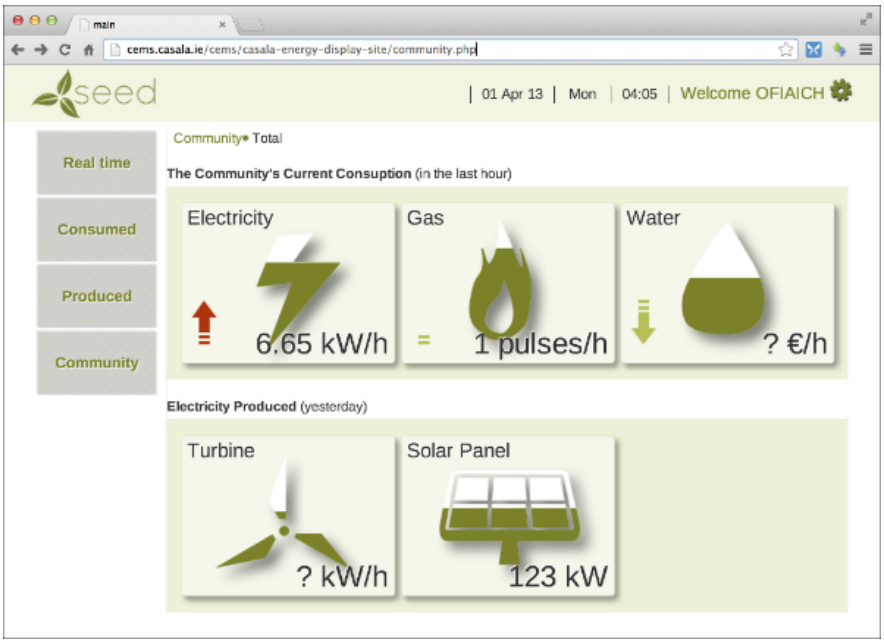


Figure 7.3: Community Energy Monitoring System

7.4 Usage Scenarios

The following subsections demonstrate just some of the current and envisaged usage scenarios that leverage the power of SCRAPE:

7.4.1 Researchers

Up until recently the CASALA research team visualised key energy usage data indicators solely through the use of the *Community Energy Monitoring System* (CEMS). See Figure 7.3. CEMS is a web based system that displays important live and historical energy output and usage readings. This is ideal for viewing energy readings from a particular building or zone at a specific point in time. So, for example, if researchers need to know exactly how much electricity the O'Fiaich school is currently using, this is quickly and easily identifiable through CEMS. What CEMS and similar web based systems are not so good at doing is visualising data from a wide range of different datasets across all community buildings (and zones) simultaneously then contextualising them in a manner that is visually coherent. A CAVE on the other hand, is more than capable of visualising large segments of data and displaying them in a coherent manner due to its unique characteristics as outlined in previous chapters.

At this early stage of development, CASALA researchers are primarily using SCRAPE to compare readings across similar groupings to help identify unusual differences between similar environments as well as to help ensure data integrity. For example,

using the CAVE it is possible to visualise complex data from over forty different zones throughout O'Fiaich and easily identify those areas that use energy more intensively than others at any particular point in time. This based on the shape, colour and size of their on-screen representations (both current and historic). The visualisations can also be interacted with through the use of the Nunchuckoo controller, enabling users to zoom in or highlight specific areas of interest. In conjunction with the O'Fiaich school it is possible to suggest likely factors for any significant differences in energy usage and identify how material or behavioural changes might be applied in order to improve efficiencies further. This can be continually visualised and assessed over time in an ongoing process of improvement. This applies equally to individual Muirhevna Mór homes in terms of understanding how ten homes of similar size and insulation levels vary in their energy usage at different stages of the day, and help identify areas for further improvement.

Once energy usage is better understood at this level, it can then be interesting to compare usage across entirely different buildings or building types. This can be useful in helping to identify key energy demand times and how energy usage can be spread more efficiently across different users. For example, energy usage is more likely to be lower for O'Fiaich and DkIT in the evening and higher for the Muirhevna Mór homes as people switch on lights and televisions and prepare their evening meals. Using the SCRAPE visualisations it is possible to highlight these interactions and point towards interesting ways in which to use energy across the community, perhaps similar to some of the smart grid communities seen in the Netherlands.

7.4.2 Stakeholders

Another key scenario for visualising SEED data in a CAVE is promotional. SCRAPE offers CASALA the ability to promote the work being carried out as part of the SEED project both to investors (such as SEAI) and the various community stakeholders in a compelling way. By visualising and navigating complex data in a 3D space using SCRAPE (as described in Section 7.3), it is easy to demonstrate the work that has been carried out and the progress that has been made to date. One of the key goals of SEED is to continue to expand its integration of energy data across the North East region and a great way to do this is to demonstrate to potentially interested parties the scale and value of the work carried out so far and the benefits both to themselves (particularly financially) and to the community at large. The visualisation and interaction scenario is similar to that used by the CASALA researchers but with a particular focus on visuals that highlight the economic and environmental benefits that are being made through both material and behavioural changes.

7.4.3 Students

A key element to making projects such as SEED a success is not just in material changes to buildings but also in changing behaviours and mindsets particularly in relation to younger generations. CASALA regularly provides tours, talks and exhibitions that involve primary, secondary and third level students on a range of different research subjects. SCRAPE provides CASALA with an excellent opportunity to educate students on the benefits of energy conservation for the future.

A scenario which is currently envisaged is to develop an interactive CAVE demonstration using SCRAPE based on SEED learnings. This can be used to demonstrate the difference that they as individuals can make in conserving energy and the impact that could be made if every student across the region or country made a similar change. Each student interacts with the sketch using a simple controller device (such as the Nunchuckoo) which enables them to modify different elements within an energy usage simulation. This mixes real data with a test scenario that enables the student to increase or reduce their energy usage based on a range of different criteria such as appliances being used, times of use, insulation levels etc., in a simulated household or building. The simulation then projects back to the student the impact that their individual changes make and extrapolates this to show how similar changes made by a multiple of students is likely to impact on a much larger scale. This type of interactive simulation scenario can prove to be a useful educational tool in bringing about important behavioural change not only with students but also with parties engaged both directly and indirectly with SEED.

7.5 Conclusion

SCRAPE has demonstrated its ability to be a useful tool in the visualisation of real-world data through its implementation as part of the SEED project. By doing so, it strengthens its ability to be viewed as a realistic alternative to existing frameworks and brings us one step closer to implementing CAVE environments that are both accessible and affordable to all.

The following chapter adds additional support to this argument by carrying out a comparative user evaluation of SEED data using both SCRAPE and a traditional desktop based system.

Chapter 8

A Comparative User Evaluation using SCRAPE

8.1 Introduction

The previous chapter provided a practical demonstration of how SCRAPE is employed in a CAVE to visualise and interact with real-world sensor data. This chapter investigates how effectively SCRAPE performs this task in comparison to traditional systems. In order to answer this question an experiment was devised to compare user performance (Lazar et al., 2010) in both a CAVE and a standard desktop based system. The visualisation scenario chosen for the CAVE is the same as that described in Chapter 7, Section 7.3 and for the Desktop it is the same CEMS system as highlighted in Chapter 7, Section 7.4.1.

Sixteen volunteers were chosen from DkIT during induction week of the college's new academic year and they were provided with a set of identical tasks in both the CAVE (using SCRAPE) and on a desktop computer (using CEMS). CEMS was chosen as it is the 'gold standard' within CASALA in terms of visualising energy data and has an established group of users. The CASALA CAVE was chosen for SCRAPE due to its availability and ability to facilitate a robust experiment in a timely manner. None of the sixteen volunteers had previous experience of using either a CAVE or of using CEMS.

The volunteers ranged in age from nineteen to forty eight and there was a concerted effort to avail of as many mature students as possible in order to spread the average age base. Volunteers were also selected from the widest possible range of courses from Business Studies to Humanities to Multimedia to Music. It was envisaged that the experiment would have an equal number of male and female volunteers in order to

eliminate any gender bias, however, this was not possible from the available pool of volunteers who were predominately male. Therefore, it should be noted that only 25% of selected volunteers were female.

The volunteers were broken up into groups of two and each group was provided with a time slot in which to make themselves available during the day. On arrival, each group were informed that they would be provided with a set of tasks on two different systems. They were also informed that the tasks were based on energy usage data that is collected from the nearby O'Fiaich secondary school. All volunteers were asked if they had any questions before commencing and were reminded that they were free to stop participating in the experiment at any stage. Refreshments and snacks were also provided.

Each volunteer was required to perform a set of tasks using one system then the other. By organising volunteers in groups of two it was possible to maximise the number of volunteers that could be assessed during the day. One volunteer started in the CAVE and the other started with the desktop before switching over to the other system. This ensured that an equal number of volunteers started with either system first so as to not skew the overall results due to familiarity with the data on just one system.

Although sixteen volunteers made themselves available for the experiment, one of the volunteers was unable to properly visualise the data within the CAVE using the active stereo 3D glasses. Due to this, the recorded results are based on fifteen volunteers and not the intended sixteen. This non-completion is discussed in the final section of this chapter.

8.2 Experimental Set-Up

Hypotheses

1. Given the same amount and type of data, SCRAPE provides for easier exploration of the data than a desktop.
2. Given the same amount and type of data, SCRAPE provides for faster information retrieval times than a desktop.
3. Given the same amount and type of data, SCRAPE provides an overall higher level of satisfaction of interaction than a desktop.
4. Given the same amount and type of data, SCRAPE provides for a reduced error rate on information retrieval tasks than a desktop.

User Tasks

The user tasks are based on energy data that is retrieved from the O'Fiaich secondary school in Dundalk Co. Louth. Each volunteer participates in five tasks for both CAVE and Desktop systems. The prescribed tasks are as follows:

- **Training Task:** Record the **Real Time** and **Today's Total** energy consumption readings for 'Science Room 1' the 'Art Room' and the 'Boiler House' of the O'Fiaich school.
- **Task One:** Record then add-up the **Real Time** energy consumption readings for all the computer rooms in the O'Fiaich school.
- **Task Two:** Record then add-up the **Today's Total** energy consumption for all lighting readings in the O'Fiaich school.
- **Task Three:** Find and record the room that has the highest **Today's Total** energy consumption in the O'Fiaich school.
- **Task Four:** Record then add-up the **Today's Total** energy consumption costs for all 'ESB' rooms in the O'Fiaich school.

Measures

1. Task Completion Times. **See Appendix C**.
2. System Usability Scale (SUS) Questionnaire. **See Appendix D**.

Method

Each volunteer takes part in a number of identical tasks on both CAVE and Desktop systems. Half the volunteers start with the Desktop based system (CEMS) before moving on to the CAVE system (SCRAPE) and vice versa. Upon entering the CAVE or sitting at the desktop, the volunteer is provided with seven stapled sheets, a pen, a table upon which to write and any necessary system tools (such as mouse, joypad or 3D glasses). On the first page is the initial training task which is used to familiarise them with the system. No timings are recorded for this task and volunteers are informed that they are not being assessed during the training task. They are also free to ask as many questions as they wish and are provided with as much assistance as required. Once the volunteer has completed the training task and is comfortable with how the system works, they are then asked to complete the four main tasks as described previously. On completion of each of these tasks the volunteer informs the facilitator and a completion time is noted. Upon completion of all tasks they are then required to complete a SUS usability questionnaire and encouraged to provide any feedback on how usable

Figure 8.1: User interacting with SCRAPE experiment data in the CAVE

(or unusable) they found the system. Once the volunteer has completed this process on one system they then repeat it on the other.

Error Rate

Incomplete answers are recorded for additional comparative analysis.

Note: Due to the constantly changing nature of the data, volunteers are not assessed upon the accuracy of the actual energy readings or costs that they provide but rather on the accuracy of room selections and number of answers provided. For example, if a specific task expects three answers and only two are provided then this will be considered a task error. By providing the correct room name as part of the answer, it is possible to accurately assess a volunteers ability to correctly navigate and retrieve the required information. For this reason the accuracy of the data itself is not deemed to be of crucial importance, however, in future experiments it could be included to add further weight to result findings.

User Satisfaction

At the end of the experiment, each volunteer is asked to provide feedback with regard to their experience using either system. This information is used to provide additional qualitative data in support of the SUS questionnaire and the quantitative tasks data.

Equipment

Desktop
Dell i7 desktop computer
22" high resolution LCD monitor
Two button optical mouse
CEMS web application software on Google Chrome browser

CAVE
Joypad controller
5 Dell Xeon workstations
3 rear projection screen (3 metres x 2 metres) + 1 floor screen
Active shutter stereo 3D glasses
3 IR emitters for stereo glasses synchronisation
SCRAPE application software

Trial Run

A 'trial run' using a volunteer without previous CAVE or CEMS experience was instigated a few days before the main experiment. This was used to iron out any potential issues or difficulties that may not have been anticipated.

8.3 Results

In the case of the CAVE, volunteers used SCRAPE to collate the necessary information by navigating around groupings of multi-coloured cube shaped objects in a 3D space that represent data from different sources (e.g. O'Fiaich school, DkIT, GNH or the Muirhevna homes). On the desktop system, users gathered the required information by navigating the different sections of CEMS using a Google Chrome browser. Then at the end of all four tasks for each system, users completed a SUS questionnaire to provide qualitative data on the user experience.

The following subsections provide the recorded results for the task timings, SUS questionnaires and error rates respectively. All 'void' entries relate to volunteer no. 7 who was unable to complete the experiment as stated previously.

8.3.1 Task Timings

From the results displayed in Tables 8.1 & 8.2 and Figure 8.2, we can clearly see that the CAVE provided for faster completion times for each of the individual tasks, with

the combined totals for all tasks showing a 46% difference between the two systems. This provides strong support for the original hypotheses that SCRAPE provides for faster retrieval times and easier exploration of the data than the desktop system.

Desktop Task Timings (Seconds)						
Vol No.	**Sex**	**Task 1**	**Task 2**	**Task 3**	**Task 4**	**Totals**
1	M	137	129	79	129	474
2	M	67	110	140	80	397
3	M	200	170	120	103	593
4	M	93	130	254	143	620
5	F	150	170	170	195	685
6	M	210	125	120	93	548
7	M	void	void	void	void	void
8	M	110	125	62	80	377
9	M	185	115	135	96	531
10	F	90	85	115	70	360
11	M	90	110	250	210	660
12	M	97	103	195	100	495
13	F	102	91	51	82	326
14	M	188	166	399	90	843
15	M	255	110	255	192	812
16	F	84	79	128	85	376
Average Task Completion Times in Seconds						
		137	121	165	117	540
Average Task Completion Times in Minutes & Seconds						
		02:17	02:01	02:45	01:57	09:00

Table 8.1: Desktop Timings Table

CAVE Task Timings (Seconds)						
Vol No.	**Sex**	**Task 1**	**Task 2**	**Task 3**	**Task 4**	**Totals**
1	M	76	90	20	52	238
2	M	63	120	46	105	334
3	M	93	76	65	121	355
4	M	132	94	55	84	365
5	F	182	112	40	72	406
6	M	138	115	35	108	396
7	M	void	void	void	void	void
8	M	135	118	30	124	407
9	M	86	76	53	131	346
10	F	115	104	43	90	352
11	M	92	90	75	80	337
12	M	98	102	55	74	329
13	F	83	119	47	98	347
14	M	91	123	54	82	350
15	M	70	81	43	82	276
16	F	95	62	26	44	227
Average Task Completion Times in Seconds						
		103	99	46	90	338
Average Task Completion Times in Minutes & Seconds						
		01:43	01:39	00:46	01:30	05:38

Table 8.2: CAVE Timings Table

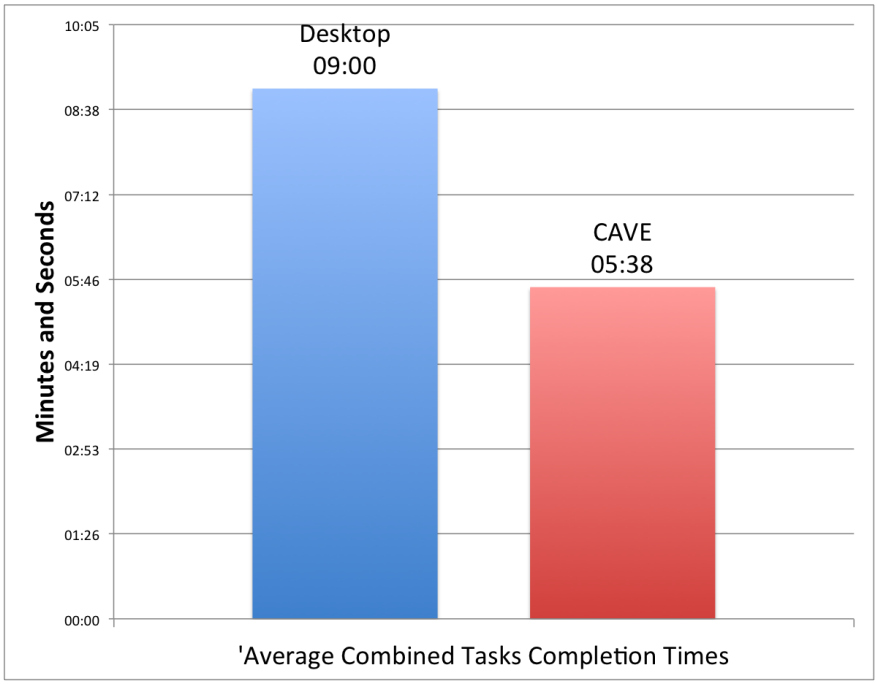


Figure 8.2: Task Comparison Chart

8.3.2 SUS Questionnaire

In 1986 John Brooke developed the SUS questionnaire (Brooke, 1996) as a 'quick and dirty' method for measuring system usability. Since then, it has become an industry standard and is widely used as a reliable method for assessing system usability.

As well as collating quantitative data in the form of the task timings, the data gathered by the SUS questionnaires as part of the experiment adds further evidence in support of using SCRAPE. From the results displayed in Figure 8.3 and Tables 8.3 & 8.4, we can see that the CAVE obtained a SUS score of 88 and the desktop obtained a score of 60. These scores support the hypothesis that in this particular evaluation, the CAVE provides an overall higher level of satisfaction of interaction than the desktop system. SUS scores are rated on a scale of zero to one hundred. The higher the score, the more usable a system is considered to be. According to Tom Tullis and Bill Albert in their book *Measuring The User Experience* (Tullis and Albert, 2008), any score under sixty can generally be considered poor, while any score over eighty is considered quite good. With this in mind, the obtained SUS scores for desktop and CAVE of 60 and 88 respectively, would suggest that both systems are suitable for the task in hand but with the CAVE system showing itself to be on the upper end of the scale.

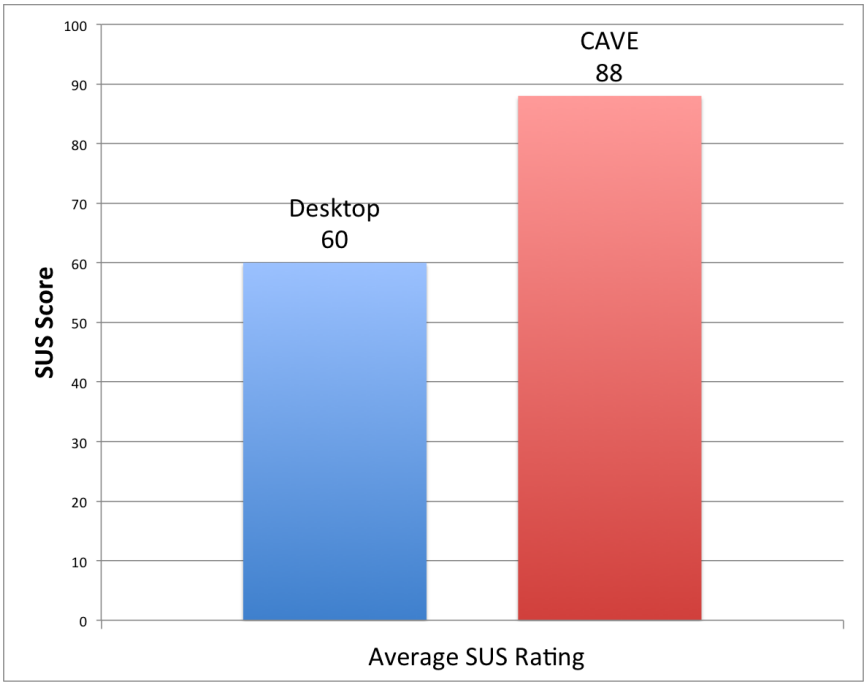


Figure 8.3: SUS Score Chart

Desktop SUS Questionnaire Scores																
Vol No.	Sex	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Totals				Score
1	M	2	1	1	4	1	2	1	2	3	4	21	x	2.5	=	52.5
2	M	1	1	2	3	2	1	3	0	3	3	19	x	2.5	=	47.5
3	M	2	3	2	3	2	2	4	3	2	3	26	x	2.5	=	65
4	M	0	2	1	2	1	1	1	4	1	1	14	x	2.5	=	35
5	F	1	3	1	1	2	1	1	1	1	1	13	x	2.5	=	32.5
6	M	3	4	4	4	3	4	4	4	4	4	38	x	2.5	=	95
7	M	void	void	void	void	void	void	void	void	void	void					
8	M	3	4	4	4	4	3	3	4	4	4	37	x	2.5	=	92.5
9	M	3	4	3	4	3	4	2	3	2	2	30	x	2.5	=	75
10	F	4	4	4	4	3	4	4	3	4	4	38	x	2.5	=	95
11	M	2	1	2	0	4	3	1	2	3	0	18	x	2.5	=	45
12	M	2	4	3	4	3	3	4	1	3	3	30	x	2.5	=	75
13	F	2	2	2	1	2	3	0	2	2	2	18	x	2.5	=	45
14	M	2	1	1	2	2	2	3	1	2	3	19	x	2.5	=	47.5
15	M	0	2	1	4	2	2	4	1	1	3	20	x	2.5	=	50
16	F	1	1	3	3	1	3	0	3	3	1	19	x	2.5	=	47.5
										Desktop SUS Score						**60**

Table 8.3: Desktop SUS Table

CAVE SUS Questionnaire Scores																
Vol No.	Sex	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Totals				Score
1	M	4	4	4	4	4	4	4	4	4	4	40	x	2.5	=	100
2	M	2	4	4	4	3	2	4	3	4	4	34	x	2.5	=	85
3	M	3	4	4	4	4	4	4	4	4	4	39	x	2.5	=	97.5
4	M	1	4	4	4	4	3	4	4	2	4	34	x	2.5	=	85
5	F	4	3	2	4	4	4	4	4	3	4	36	x	2.5	=	90
6	M	1	4	4	4	2	4	4	3	4	4	34	x	2.5	=	85
7	M	void	void	void	void	void	void	void	void	void	void					
8	M	4	4	4	4	3	2	3	4	4	4	36	x	2.5	=	90
9	M	4	4	4	4	4	4	4	4	4	4	40	x	2.5	=	100
10	F	2	4	2	4	4	4	3	2	2	4	31	x	2.5	=	77.5
11	M	4	1	4	4	4	1	4	0	4	1	27	x	2.5	=	67.5
12	M	2	4	4	3	2	4	3	3	3	3	31	x	2.5	=	77.5
13	F	4	4	4	2	4	4	4	4	3	4	37	x	2.5	=	92.5
14	M	3	4	3	2	3	4	4	4	3	4	34	x	2.5	=	85
15	M	3	1	4	4	3	4	4	4	4	4	35	x	2.5	=	87.5
16	F	4	4	4	4	4	4	4	4	4	4	40	x	2.5	=	100
										CAVE SUS Score						**88**

Table 8.4: CAVE SUS Table

8.3.3 Error Rates

Error rates were calculated based on volunteers recording the expected room names for each task and the correct number of results for each task. Every time a volunteer incorrectly recorded a result or failed to record a result this was noted and assigned one point. The accuracy of the recorded result was not considered due to the issues in accurately verifying the actual result reading at the point when the users noted it.

Even though the difference in recorded error rates is marginal, the results do add further weight in support of the hypothesis that the CAVE running SCRAPE provides for a reduced error rate on information retrieval tasks, with error rates 5% lower in the CAVE than for the desktop system. Full results can be seen in Figure 8.4 and Tables 8.5 & 8.6.

Figure 8.4: Error Rate Comparison Chart

Number of Desktop Errors						
Vol No.	**Sex**	**Task 1**	**Task 2**	**Task 3**	**Task 4**	**Totals**
1	M	0	0	0	0	0
2	M	0	0	0	0	0
3	M	1	1	1	0	3
4	M	0	0	0	1	1
5	F	0	0	0	0	0
6	M	0	1	0	1	2
7	M	void	void	void	void	void
8	M	0	0	0	0	0
9	M	0	0	0	0	0
10	F	0	1	0	1	2
11	M	0	0	1	0	1
12	M	0	1	0	0	1
13	F	0	1	0	0	1
14	M	0	0	0	1	1
15	M	0	0	0	0	0
16	F	0	0	0	0	0
					Total Errors	**12**

Table 8.5: Desktop Errors

Number of CAVE Errors						
Vol No.	**Sex**	**Task 1**	**Task 2**	**Task 3**	**Task 4**	**Totals**
1	M	1	0	0	0	1
2	M	0	0	0	1	1
3	M	0	0	0	0	0
4	M	0	1	0	1	2
5	F	0	0	0	0	0
6	M	0	0	0	0	0
7	M	void	void	void	void	void
8	M	0	0	0	0	0
9	M	0	0	0	0	0
10	F	0	0	0	1	1
11	M	0	0	1	1	2
12	M	0	0	0	0	0
13	F	0	0	0	0	0
14	M	0	0	0	1	1
15	M	0	0	0	0	0
16	F	0	0	0	1	1
					Total Errors	**9**

Table 8.6: CAVE Errors

Timing data errors were not removed from the overall timing results as illustrated in Section 8.3.1. These figures simply record the actual time it took users to complete all tasks. Tables 8.7 & 8.8, however, provide data on the average completion times with all timing errors removed. These revised timings result in a 41% difference in result timings as opposed to a 46% difference in the original unrevised results. This represents a narrowing of the difference between systems but still demonstrates a significant performance benefit for the CAVE over the Desktop system.

Desktop Task Timings	
Average Completion Time in Seconds	**437**
Average Completion Time in Minutes	**07:17**

Table 8.7: Revised Desktop Timings Table

CAVE Task Timings	
Average Completion Time in Seconds	**289**
Average Completion Time in Minutes	**04:49**

Table 8.8: Revised CAVE Timings Table

8.3.4 Feedback

Upon completing all tasks, volunteers were requested to provide feedback on their experience. Their views were generally positive towards both systems, however, they did highlight positive and negative aspects to both. A number of volunteers expressed issues in particular with the inability to sort data. While this is a valid complaint, it is currently a limitation on both systems and, therefore, does not particularly add anything to the comparison debate at this point in time. Of more interest was the feedback in relation to the resolution of the CAVE screens. Some users found the data visualisations in the CAVE somewhat hard to read. This could be attributed to the fact that the current CAVE screens rely on somewhat dated WVGA analog stereo projectors, however, further study would need to be carried out to confirm if this is the only factor.

In relation to the desktop system, some volunteers expressed the opinion that it was slower, more cumbersome and specifically “less fun” to use than the CAVE system. Overall, most volunteers expressed a preference for the CAVE system, commenting in particular that it was both easier to use and easier to retrieve the data. While the feedback provided by the volunteers is subjective, the overall sentiment tends to sup-

port the quantitative data. Data which errs in favour of the CAVE as providing greater performance and usability for its users for this particular scenario.

8.3.5 Statistical Validity

A sample size calculation was not carried out as part of the comparative user evaluation. As such, it should be treated as a pilot study rather than a comprehensively valid statistical assessment. For this pilot study, the 16 volunteers that participated in the experiment were deemed sufficient in order to explore research questions around the volunteers' experience of both CAVE and desktop systems, and to provide sufficient data for basic comparisons and further investigations.

8.4 Analysis & Findings

Looking at the different result categories, it can be seen how each one supports all four original hypotheses. The data collected in relation to task timings as well as user feedback confirms that, for this evaluation, SCRAPE provides for easier exploration of the data and faster information retrieval times than CEMS. The SUS questionnaire strongly supports the view that SCRAPE provides for higher levels of satisfaction of interaction and the recorded task results show slightly lower error percentage rates.

While the overall results of the user evaluation lean favourably towards the CAVE system, there are some specific caveats which should be noted:

Firstly, in this evaluation, volunteers were encouraged to provide written feedback but were not strictly requested to do so. While some volunteers did, the number was small and therefore one could argue that the feedback provided which was typically oral does not add sufficient weight to the overall hypothesis.

Secondly, the evaluations for SCRAPE were carried out on the CASALA CAVE which is a high-end CAVE system. Throughout this thesis we have argued the importance of SCRAPE to work on all types of CAVE systems both high-end and low-end. While the author does not believe that carrying out the evaluation on an equivalent low-end CAVE system would drastically alter the results, it is accepted that this could be open to question.

Thirdly, as mentioned at the beginning of this chapter, for one volunteer the CAVE proved to be completely unusable. While not the norm, there are a small number of people for whom the CAVE is not a viable solution. Inability to visualise 3D scenes,

motion sickness, headaches and loss of balance are just a few potential issues. Sixteen volunteers is a relatively small sample, it is therefore difficult to accurately estimate the percentage of users that are likely to experience difficulty using a CAVE. However, based on these figures (i.e. one in sixteen), it suggests that over 6% of any potential CAVE users may be unable to use a system like SCRAPE.

Finally, It should be noted that whilst the experimental results are themselves interesting, in that they support the concept of CAVEs for data visualization, they are not particularly unexpected. More extensive experiments of a similar nature (Arns et al., 1999, Prabhat et al., 2008, Ye et al., 1999) have provided ample evidence of VEs performing better than their desktop counterparts. It may not, however, be as clear cut as that. It could be argued that direct comparisons of fundamentally different systems using traditional assessment methods do not provide accurate results. In Doug Bowman's paper 'A Survey of Usability Evaluation in Virtual Environments: Classification and Comparison of Methods' (Bowman et al., 2002) he argues that VE evaluations should be assessed differently than traditional GUI systems and that traditional comparisons are inaccurate. In addition, the SCRAPE experiment also raises other questions that require further investigation (e.g. whether the same user evaluation results are obtained using a low-end CAVE). While both of these considerations are important, they are considered outside the scope of this thesis which focuses on the development of SCRAPE. Rather, the experiment is presented in this chapter as a strong indicator of the practical validation, usefulness and applicability of SCRAPE in a research context.

Chapter 9

Conclusion

In Chapter 1, three key contributions were identified in relation to the development of an open source framework for data visualisation in a CAVE. The first was to provide the reader with a comprehensive documentation of 3D technologies and the key factors that need to be considered in relation to CAVE environments. In Chapters 2, 3 & 4 this was provided through the provision of the following:

- A comprehensive overview of the history of VR.
- A discussion of the key elements of immersion (a vital ingredient in any CAVE).
- The key factors to consider in relation to interaction modalities.
- A taxonomy of CAVE interaction devices.
- The proposal of a novel interaction device suitable for a CAVE's unique characteristics.

The second key contribution was to provide a new open source software framework to assist in making CAVE platforms more accessible to both amateur and experienced users alike. Chapter 5 firstly identified and discussed some existing frameworks that were considered worthy of note in relation to CAVE environments. Chapter 6 then presented in detail the development of SCRAPE, an open source framework ideally suited to visualising large datasets in a CAVE. SCRAPE was also set up in the CASALA CAVE and employed successfully to visualise real-world sensor data from the SEED project. Full details of this project and its application in the CAVE were then provided in Chapter 7.

It should be noted that a direct assessment of SCRAPEs 'ease of use' in comparison to other CAVE frameworks was not carried out as part of this work. While this could be viewed as a basic omission, it is considered that the core tenet of P5 (upon which

SCRAPE extends itself) which is based on promoting accessibility to all users, offers sufficient credentials in of itself that a detailed comparison was not requisite. It is, however, accepted that without a direct scientific comparison this could still be open to question.

The third key contribution was to prove whether or not a CAVE using SCRAPE can be more effective in a range of tasks when compared to a traditional desktop system. The goal was to demonstrate the applicability of SCRAPE in quickly designing and running user evaluations that are a key aspect of data visualisation research. This was addressed in Chapter 8 where a user evaluation was carried out which compared the CAVE (using SCRAPE) with the browser-based desktop system that is typically used to view information for the SEED project. In this experiment, the CAVE provided for greater performance across a range of different qualitative and quantitative measurements.

Finally, there was an additional minor contribution relating to the customisation of a unique interaction device for CAVE interactions. In Chapter 4 an overview of this work (entitled 'Nunchuckoo') as well as set-up instructions were provided. Additional test code is also included in Appendix A.

9.1 Future Work

The creation of SCRAPE was not the first foray into the development of an open source framework for CAVE environments but it is the first to offer a development framework that users of all programming abilities can master. Nonetheless, while it has already proven its worth in relation to visualising data from SEED, it is far from perfect. SCRAPE is expected to be redeveloped for compatibility with Processing v2 which has some significant upgrade features from previous versions. It is also envisaged that the SCRAPE code will be reorganised so as to provide greater separation between the framework itself and the sketches that run on it. In other words, currently there is some overlap between the code that is core to SCRAPE and code that is purely sketch related. This may cause some confusion to novice users and runs the risk of important framework based code being inadvertently removed. With this in mind, it is envisaged that the framework based code will be abstracted from direct view but still made easily available if required.

Ideally SCRAPE would be set up as a standalone installation that will simplify even further the set-up and configuration process for end-users. SCRAPE is ideal for visualising data in abstract ways but it is not necessarily the ideal tool for creating highly

developed built environment style VWs. Whatever form SCRAPE takes in future developments, it is certain that it will continue to be used in conjunction with projects such as SEED, and will be further developed to provide ever richer VWs that assist in data visualisation.

9.2 Final Thoughts

A CAVE in its current format will never be a replacement for traditional methods of visualising data, even with open source frameworks and reduced infrastructure costs. It is unlikely that we will see one in every office any time soon, however, for those that do use a CAVE the opportunities are exciting. As demonstrated at CASALA, a CAVE can be a useful data visualisation tool across a range of different projects. By fully understanding the elements that make a CAVE unique and through the continuous development of improved technologies and reduced costs, the fortunes of the CAVE can only improve.

As the revival of head mounted 3D displays in the form of the Oculus Rift has demonstrated, it often takes only small changes in an existing technology to make a big difference. In 2014 the Oculus rift is paving the way for the video games community but who is to say that in 2015 it won't be evolutions in a CAVE that will bring us the next step forward in terms of interactive 3D technologies. The CAVE has already proven its worth and value to projects such as SEED and it is this author's belief that when the infrastructure costs become low enough and glasses free 3D becomes a reality, then CAVE technology will be the ultimate experience many of us have been striving for. The vision of the Star Trek Holodeck may not be all that far away...

References

Alphathon (Accessed: December 20, 2013). Image: Kinect sensor. http://commons.wikimedia.org/wiki/File:KinectSensor.png.

Android.com (Accessed: December 20, 2013). Image: Smartphone and tablet devices. http://android.com/phones-and-tablets.

Ar-tracking.com (Accessed: December 20, 2013). Image: Wand controller. http://www.ar-tracking.com/products/interaction-devices/flystick2/.

Arduino.cc (Accessed: December 20, 2013). Image: Arduino microcontroller. http://store.arduino.cc/index.php?main_page=index&cPath=11.

Arns, L., Cruz-Neira, C., and Cook, D. (1999). The benefits of statistical visualization in an immersive environment. In *Proceedings of the IEEE Virtual Reality*, VR '99, pages 88–, Washington, DC, USA. IEEE Computer Society.

Baren, J. v. and IJsselsteijn, W. (2004). Measuring presence: A guide to current measurement approaches. presence-research.org and ispr.info. OmniPres project for The International Society for Presence Research (ISPR) and Eindhoven University of Technology.

Barfield, W. and Weghorst, S. (1993). The sense of presence within virtual environments: A conceptual framework. In *HCI (2)*, pages 699–704.

Bowman, D., Gabbard, J. L., and Hix, D. (2002). A survey of usability evaluation in virtual environments: Classification and comparison of methods. presence: Teleoperators and virtual environments. pages 404–424.

Bowman, D. A. and McMahan, R. P. (2007). Virtual reality: How much immersion is enough? *Computer*, 40(7):36–43.

Bowman, D. A. and Raja, D. (2004). A method for quantifying the benefits of immersion using the cave. *Presence-Connect*, 4(2).

Bowman, D. A., Raja, D., Lucas, J., and Datey, A. (2005). Exploring the benefits of immersion for information visualization. In *Proceedings of HCI International*.

Bowman, D. A., Sowndararajan, A., Ragan, E. D., and Kopper, R. (2009). Higher levels of immersion improve procedure memorization performance. In *Proceedings of the 15th Joint virtual reality Eurographics conference on Virtual Environments*, pages 121–128. Eurographics Association.

Brooke, J. (1996). SUS: A quick and dirty usability scale. In *Usability evaluation in industry*. Taylor and Francis, London.

Brooks, Jr., F. P., Ouh-Young, M., Batter, J. J., and Jerome Kilpatrick, P. (1990). Project grope - haptic displays for scientific visualization. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '90, pages 177–185, New York, NY, USA. ACM.

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

Cruz-Neira, C., Sandin, D. J., DeFanti, T. A., Kenyon, R. V., and Hart, J. C. (1992). The cave: audio visual experience automatic virtual environment. *Commun. ACM*, 35:64–72.

Darby, S. (2006). The effectiveness of feedback on energy consumption. *A Review for DEFRA of the Literature on Metering, Billing and direct Displays*, 486.

Denby, B., Campbell, A., Carr, H., and O'Hare, G. (2009). The lair: Lightweight affordable immersion room. *Presence: Teleoperators and Virtual Environments*, 18(5).

Diracdelta.co.uk (Accessed: February 24, 2012). Image: Six degrees of freedom. http://www.diracdelta.co.uk/science/source/s/i/six%20degrees%20of%20freedom/source.html.

Drilnoth (Accessed: December 20, 2013). Image: Gamepad controller. http://commons.wikimedia.org/wiki/File:PlayStation_3_gamepad.svg.

Durand, F. (2002). An invitation to discuss computer depiction. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, NPAR '02, pages 111–124, New York, NY, USA. ACM.

Ellis, S. R. (1994). What are virtual environments? *Computer Graphics and Applications, IEEE*, 14(1):17–22.

Emotiv.com (Accessed: January 23, 2011). Image: Epoc neuroheadset. http://www.emotiv.com/epoc/features.php.

Fisher, S. S., McGreevy, M., Humphries, J., and Robinett, W. (1987). Virtual environment display system. In *Proceedings of the 1986 workshop on Interactive 3D graphics*, I3D '86, pages 77–87, New York, NY, USA. ACM.

Flynn, C., O'Connor, N. E., Lee, H., and Loane, J. (2013). Visualising better energy communities in a cave automatic virtual environment. In *Proceedings of the 7th Annual Irish Human Computer Interaction Conference*, Dundalk, Ireland.

Friedrich, A. (Accessed: December 20, 2013). Image: Voice control. http://commons.wikimedia.org/wiki/File:Microphone_slant.svg.

Heilig, M. L. (1992). El cine del futuro: the cinema of the future. *Presence: Teleoper. Virtual Environ.*, 1(3):279–294.

Henryson, J., Hakansson, T., and Pyrko, J. (2000). Energy efficiency in buildings through information - swedish perspective. *Energy Policy*, 28(3):169 – 180.

Howley, M., Dennehy, E., Ó Gallachóir, B., and Holland, M. (2012). Energy in ireland 1990 2011. Technical report, Sustainable Energy Authority of Ireland (SEAI).

Instructables.com (Accessed: February 23, 2013). Image: Nunchuck to arduino direct wire connection. http://www.instructables.com/files/deriv/FOA/0I6U/GFRWRNI0/FOA0I6UGFRWRNI0.LARGE.jpg.

Iotracker.com (Accessed: February 26, 2012). Image: Object tracking. http://iotracker.com/index.php?q=rigid_body_targets.

Kay, A. (Accessed: December 4, 2011). Image: Sketchpad system. http://youtube.com/watch?v=495nCzxM9PI.

KEMA (2008). Demand side management in ireland - evaluating the energy efficiency opportunities. Technical report, Sustainable Energy Authority of Ireland (SEAI).

Kesting, S. and Bliek, F. (2013). Chapter 14 - from consumer to prosumer: Netherlands powermatching city shows the way. In *Energy Efficiency*, pages 355 – 373. Academic Press, Boston.

Krueger, M. W., Gionfriddo, T., and Hinrichsen, K. (1985). Videoplace - an artificial

reality. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '85, pages 35–40, New York, NY, USA. ACM.

Lazar, J., Feng, J., and Hochheiser, H. (2010). *Research Methods in Human-Computer Interaction*. Wiley, Indianapolis, IN.

McAuley, D. and Polaski, K. (2011). energy research in ireland 2004 - 2010 - people, funding and technologies. Technical report, Sustainable Energy Authority of Ireland (SEAI).

Minsky, M. (1980). Telepresence. *OMNI*, pages 44–52.

Mortensen, J., Khanna, P., Yu, I., and Slater, M. (2007). Real-time global illumination in the cave. In *Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, VRST '07, pages 145–148, New York, NY, USA. ACM.

Mortonheilig.com (Accessed: December 3, 2011a). Image: Sensorama simulator. http://mortonheilig.com/InventorVR.html.

Mortonheilig.com (Accessed: December 3, 2011b). Image: Telesphere mask. http://mortonheilig.com/TelesphereMask.pdf.

Murray, J. H. (1997). *Hamlet on the Holodeck: The Future of Narrative in Cyberspace*. The Free Press, New York, NY, USA.

Narayan, M., Waugh, L., Zhang, X., Bafna, P., and Bowman, D. A. (2005). Quantifying the benefits of immersion for collaboration in virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '05, pages 78–81, New York, NY, USA. ACM.

Nichols, S., Haldane, C., and Wilson, J. R. (2000). Measurement of presence and its consequences in virtual environments. *International Journal of Human-Computer Studies*, 52(3):471 – 491.

Otaduy, M. A., Igarashi, T., and LaViola, Jr., J. J. (2009). Interaction: interfaces, algorithms, and applications. In *ACM SIGGRAPH 2009 Courses*, SIGGRAPH '09, pages 14:1–14:66, New York, NY, USA. ACM.

Pape, D. (Accessed: December 11, 2011). Image: Vpl virtual reality equipment. http://commons.wikimedia.org/wiki/File:VPL_Eyephone_and_Dataglove.jpg.

Pausch, R., Proffitt, D., and Williams, G. (1997). Quantifying immersion in virtual reality. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 13–18. ACM Press/Addison-Wesley Publishing Co.

Peregrine (Accessed: December 20, 2013). image: Glove controller. http://theperegrine.com\.

Prabhat, Forsberg, A., Katzourin, M., Wharton, K., and Slater, M. (2008). A comparative study of desktop, fishtank, and cave systems for the exploration of volume rendered confocal data sets. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):551–563.

Ruzzelli, A. G., Nicolas, C., Schoofs, A., and O'Hare, G. M. (2010). Real-time recognition and profiling of appliances through a single electricity sensor. In *Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on*, pages 1–9. IEEE.

Sas, C. and O'Hare, G. M. (2003). Presence equation: An investigation into cognitive factors underlying presence. *Presence: Teleoperators and Virtual Environments*, 12(5):523–537.

Schuchardt, P. and Bowman, D. A. (2007). The benefits of immersion for spatial understanding of complex underground cave systems. In *Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, VRST '07, pages 121–124, New York, NY, USA. ACM.

Schuemie, M. J., van der Straaten, P., Krijn, M., and van der Mast, C. (2001). Research on presence in virtual reality: A survey. *Cyberpsy., Behavior, and Soc. Networking*, 4(2):183–201.

Sheridan, T. B. (1992). Musings on telepresence and virtual presence. *Presence: Teleoper. Virtual Environ.*, 1(1):120–126.

Shiffman, D. (Accessed: June 24, 2012). Image: Most pixels ever at the iac manhattan. https://github.com/shiffman/Most-Pixels-Ever-Processing.

Slater, M. (1999). Measuring presence: A response to the witmer and singer presence questionnaire. *Presence: Teleoper. Virtual Environ.*, 8(5):560–565.

Slater, M. (2002). Presence and the sixth sense. *Presence: Teleoper. Virtual Environ.*, 11(4):435–439.

Slater, M. (2003). A note on presence terminology. *Presence-Connect*, 3.

Slater, M. and Steed, A. (2000). A virtual presence counter. *Presence-Teleoperators and Virtual Environments*, 9(5):413–434.

Slater, M., Usoh, M., and Steed, A. (1994). Depth of Presence in Virtual Environments. *Teleoperators and Virtual Environments*, 3:130–144.

Sodhi, R., Glisson, M., and Poupyrev, I. (2013). Aireal: tactile gaming experiences in free air. In *ACM SIGGRAPH 2013 Emerging Technologies*, SIGGRAPH '13, pages 2:1–2:1, New York, NY, USA. ACM.

Sowndararajan, A., Wang, R., and Bowman, D. A. (2008). Quantifying the benefits of immersion for procedural training. In *Proceedings of the 2008 workshop on Immersive projection technologies/Emerging display technologiges*, IPT/EDT '08, pages 2:1–2:4, New York, NY, USA. ACM.

Steuer, J. (1992). Defining virtual reality: Dimensions determining telepresence. *Journal of communication*, 42(4):73–93.

Sutcliffe, A., Gault, B., Fernando, T., and Tan, K. (2006). Investigating interaction in cave virtual environments. *ACM Trans. Comput.-Hum. Interact.*, 13:235–267.

Sutherland, I. E. (1965). The ultimate display. In *Proceedings of the Congress of the Internation Federation of Information Processing (IFIP)*, volume volume 2, pages 506–508.

Sutherland, I. E. (1968). A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), pages 757–764, New York, NY, USA. ACM.

Tullis, T. and Albert, W. (2008). *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Witmer, B. G. and Singer, M. J. (1998). Measuring presence in virtual environments: A presence questionnaire. *Presence: Teleoperators and virtual environments*, 7(3):225–240.

Ye, N., Banerjee, P., Banerjee, A., and Dech, F. (1999). A comparative study of assembly planning in traditional and virtual environments. *IEEE Transactions on Systems, Man, and Cybernetics*, 29:546–555.

Zimmerman, T. G., Lanier, J., Blanchard, C., Bryson, S., and Harvill, Y. (1987). A hand gesture interface device. In *Proceedings of the SIGCHI/GI Conference on Human Factors in Computing Systems and Graphics Interface*, CHI '87, pages 189–192, New York, NY, USA. ACM.

Appendices

Appendix A

Nuchuckoo Test Code

```
1  // Access the arduino + nunchuck control data
2  import processing.serial.*;
3
4  final int LINE_FEED = 10;
5  final int BAUD_RATE = 19200;
6  float angleX = 0.0f;
7  float angleY = 0.6f;
8
9  Serial arduinoPort;
10
11 void setup(){
12   size(400,300,P3D);
13     try{
14       String[] ports = Serial.list();
15       if(ports.length < 1){
16         println("WARNING: Your serial list is empty!");
17       }
18       else{
19         println("NOTICE: serial list is : ");
20         println(ports);
21         arduinoPort = new Serial(this, Serial.list()[0], BAUD_RATE);
22         arduinoPort.bufferUntil(LINE_FEED);
23       }
24     }
25     catch(Exception e){
26       println(e);
```

```
27        println("WARNING: Check Nunchuckoo connections!");
28      }
29  }
30
31  // Draw scene
32  void draw(){
33    background(0);
34    lights();
35
36    noStroke();
37    fill(204);
38    translate(200,150,0);
39    rotateY(angleY);
40    rotateX(angleX);
41    box(100);
42  }
43
44  // Arduino + nunchuck actions read and rotation variables modified
45  void serialEvent(Serial port) {
46    final String arduinoData = port.readStringUntil(LINE_FEED);
47    if (arduinoData != null) {
48      final int[] data = int(split(trim(arduinoData), ' '));
49      if (data.length == 7) {
50        // NOTE: array selections may change
51        if (data[0] < 60) {
52          angleY -= 0.02f;
53        }
54        else if(data[0] > 190) {
55          angleY += 0.02f;
56        }
57        else if(data[1] > 190) {
58          angleX += 0.02f;
59        }
60        else if(data[1] < 60) {
61          angleX -= 0.02f;
62        }
63        else if (data[2] < 400 && data[5] == 1) {
64          angleY -= 0.02f;
65        }
```

```
66          else if(data[2] > 600 && data[5] == 1) {
67            angleY += 0.02f;
68          }
69          else if(data[3] > 600 && data[5] == 1) {
70            angleX += 0.02f;
71          }
72          else if(data[3] < 450 && data[5] == 1) {
73            angleX −= 0.02f;
74          }
75          else if(data[6] == 1){
76            angleX = 0.0f;
77            angleY =0.6f;
78          }
79        }else {
80          println(”WARNING: Data length incorrect!”);
81        }
82    }
83  }
```

Appendix B

SCRAPE Code

The following pages step through the SCRAPE code files and include relevant comments.

SCRAPE.pde
Core SCRAPE Class

List of libraries to be imported:

```
import processing.opengl.*;
import mpe.client.*;
import damkjer.ocd.*;
import javax.media.opengl.*;
import stereo.*;
```

Majority of reference variables and assignments required to run SCRAPE:

```
Stereo stereo = null;
Camera camSelected;
TCPClient client;
boolean start = false;
Properties props;
String wall;
String controller;
String controllerName;
String nunchuck;
String activeStereo;
char wallChar;
float fov;
float fov2;
```

```
float aspRatio;
float nrClip = 0.01f;
float frClip = 2000f;
float rotVar = 0.03f;
float linVar = 2.5f;
float rolVar = (rotVar*180)/PI;
float angle = 0.0f;
String dir = "";
float joyMove = 0.0f;
float x;
float y;
float eyeSep = 3.1f;
```

Define environment properties within the setup() function:

```
void setup() {
```

Read SCRAPE configurations file (mpe.ini) and assign values to relevant variables:

```
try {
 props = new Properties();
 props.load(new FileInputStream(sketchPath("mpe.ini")));
 wall = props.getProperty("wall").replaceAll(";$", "").toLowerCase();
 controller = props.getProperty("controller").replaceAll(";$", "").toLowerCase();
 controllerName = props.getProperty("controllerName").replaceAll(";$", "").
      toLowerCase();
 nunchuck = props.getProperty("nunchuck").replaceAll(";$", "").toLowerCase();
 activeStereo = props.getProperty("activeStereo").replaceAll(";$", "").toLowerCase()
      ;
}
catch(IOException e) {
  println("unable to read config file...");
}
```

Make a new MPE client using mpe.ini file configs:

```
client = new TCPClient(sketchPath("mpe.ini"), this);
```

Determine aspect ratio and vertical field of view (fov)[8] based on screen size settings in mpe.ini:

[8] See paulbourke.net/miscellaneous/lens/ for full explanation

```
aspRatio = (float)client.getLWidth()/(float)client.getLHeight();
println("Aspect ratio is: " + aspRatio);
fov = 2*atan(((float)client.getLHeight()/2)/((float)client.getLWidth()/2));
println("Vertical fov is: " + fov);
```

Use active stereo for MPE client if set in config file:

```
if(activeStereo.equals("on")) {
 size(client.getLWidth(), client.getLHeight(), "stereo.ActiveStereoView");
 stereo = new Stereo(this, eyeSep, fov, nrClip, frClip, Stereo.StereoType.ACTIVE);
 println("Active stereo is set to 'on'");
}
else {
 size(client.getLWidth(), client.getLHeight(), OPENGL);
 println("Active stereo is set to 'off'");
}
```

Create OCD camera based on screen option specified in config file:

```
if(wall.equals("left")) {
 camSelected = new Camera(this, 0, 0, client.getLWidth()/2, -1, 0, client.getLWidth
     ()/2, 0, 1, 0, fov, aspRatio, nrClip, frClip);
}
else if(wall.equals("front")) {
 camSelected = new Camera(this, 0, 0, client.getLWidth()/2, 0, 0, 0, 0, 1, 0, fov,
     aspRatio, nrClip, frClip);
}
else if(wall.equals("right")) {
 camSelected = new Camera(this, 0, 0, client.getLWidth()/2, 1, 0, client.getLWidth()
     /2, 0, 1, 0, fov, aspRatio, nrClip, frClip);
}
else if(wall.equals("bottom")) {
 camSelected = new Camera(this, 0, 0, client.getLWidth()/2, 0, 1, client.getLWidth()
     /2, 0, 0, 1, fov, aspRatio, nrClip, frClip);
}
else if(wall.equals("top")) {
 camSelected = new Camera(this, 0, 0, client.getLWidth()/2, 0, -1, client.getLWidth
     ()/2, 0, 0, -1, fov, aspRatio, nrClip, frClip);
 }
else if(wall.equals("stern")) {
```

```
 camSelected = new Camera(this, 0, 0, client.getLWidth()/2, 0, 0, client.getLWidth(),
       0, 1, 0, fov, aspRatio, nrClip, frClip);
}
else {
 throw new IllegalArgumentException('\n' + "Camera name not correctly specified
      " + '\n' + "Use one of the following in mpe config file: left, right, front, back,
      top & bottom" + '\n');
}
 println("Currently displaying the " + wall + " screen");
```

Set screen char variable for use in switch statements:

```
wallChar = wall.charAt(0);
```

Random seed set for MPE. Must be identical for all clients:

```
randomSeed(1);
```

Call nunchuckoo() controller function if activated in config file:

```
if(nunchuck.equals("on")) {
 println("Nunchuck is set to 'on'");
 nunchuckoo();
}
 else {
 println("Nunchuck is set to 'off'");
}
```

Call joypad() controller function if activated in config file:

```
if(controller.equals("on")) {
 println("Controller is set to 'on'");
 joypad();
}
 else {
 println("Controller is set to 'off'");
}
```

Start the MPE client and close the setup() function:

```
client.start();
}
```

draw() function required but not used. frameEvent() function ensures synchronised frame rendering instead:

```
void draw() {
}
void frameEvent(TCPClient c) {
```

Call OCD camera feed() function for selected camera, select scene and close frameEvent() function:

```
camSelected.feed();
if(activeStereo.equals("on")) {
 caveSceneStereo();
}
else {
 caveScene();
 }
}
```

keyboard.pde

Inner class to assign basic keyboard interactions

Arrow keypad actions broadcast on key press:

```
void keyPressed() {
 if(key == CODED) {
  if(keyCode == LEFT) {
   dir = "LEFT";
   client.broadcast(dir);
  }
  else if(keyCode == RIGHT) {
   dir = "RIGHT";
   client.broadcast(dir);
  }
  else if(keyCode == UP) {
   dir = "UP";
   client.broadcast(dir);
  }
  else if(keyCode == DOWN) {
   dir = "DOWN";
   client.broadcast(dir);
  }
```

```
 }
```

Spacebar keypad action to be broadcast on key press:

```
 else if (key == 32) {
  dir = "RESET";
  client.broadcast(dir);
 }
}
```

joypad.pde

Inner class to access control devices using proCONTROLL

Import libraries:

```
import procontroll.*;
import java.io.*;
```

Create object reference variables:

```
ControllIO controll;
ControllDevice device;
ControllStick stick;
ControllButton button;
```

joypad() function called if specified in config file:

```
void joypad() {
```

Check for installed input devices and their corresponding controls then print results:

```
 controll = ControllIO.getInstance(this);
 controll.printDevices();
 for(int i = 0; i < controll.getNumberOfDevices(); i++) {
  ControllDevice device = controll.getDevice(i);
  println(device.getName()+" has:");
  println(" " + device.getNumberOfSliders() + " sliders");
  println(" " + device.getNumberOfButtons() + " buttons");
  println(" " + device.getNumberOfSticks() + " sticks");
  device.printSliders();
  device.printButtons();
  device.printSticks();
 }
```

Select device specified in config file, assign parameters and call joypadAction() broadcast function if no errors:

```
try {
 device = controll.getDevice(controllerName);
 stick = device.getStick(1); //stick no. chosen from list
 stick.setTolerance(0.8f);
 button = device.getButton(2); //button no. chosen from list
 joypadAction();
 }
 catch(IndexOutOfBoundsException e) {
  println(e);
  println("NOTICE: Ensure that your stick, button and slider
  settings are correct for your controller. If unsure, check under
  <<available proCONTROL devices>> in the log text above" + '\n');
 }
 catch(RuntimeException e) {
  println(e);
  println("NOTICE: Ensure that your controller is connected and that
  you named it correctly. If unsure, check that the name specified
  matches exactly one of the names listed under
  <<available proCONTROL devices>> in the log text above" + '\n');
 }
}
```

joypadAction() function checks stick values and button press actions then broadcasts accordingly:

```
void joypadAction() {
 if(device != null) {
  x = stick.getX();
  y = stick.getY();
  if(joyMove > y) {
   dir = "UP";
   client.broadcast(dir);
  }
  else if(joyMove < y) {
   dir = "DOWN";
   client.broadcast(dir);
  }
  else if(joyMove > x) {
```

```
    dir = "LEFT";
    client.broadcast(dir);
   }
   else if(joyMove < x) {
    dir = "RIGHT";
    client.broadcast(dir);
   }
   else if(button.pressed()){
    dir = "RESET";
    client.broadcast(dir);
   }
  }
}
```

nunchuckoo.pde

Inner class to access Nunchuckoo controller

Import library, assign line feed, assign baud rate and create object reference variable:

```
import processing.serial.*;
final int LINE_FEED = 10; // ASCII code 10 denotes a line feed
final int BAUD_RATE = 19200;
Serial arduinoPort;
```

nunchuckoo() function called if specified in config file:

```
void nunchuckoo() {
```

Check if a serial device is connected. If so, print results, specify port and assign delay before serialEvent() function is called, otherwise abort:

```
try{
 String[] ports = Serial.list();
 if(ports.length < 1) {
  println("NOTICE: uh oh....your serial list is empty. You don't
  appear to have your arduino and nunchuck connected!" + '\n');
 }
 else {
  println("NOTICE: serial list is : ");
  println(ports);
  println("If your nunchuck and arduino are plugged in and not
  working, check that you are specifying the correct serial list
```

```
  port and that BAUD rates are correct on the computers COM port
  and also that there is no RXTX Version mismatch on processing
  output window" + '\n');
  arduinoPort = new Serial(this, Serial.list()[0], BAUD_RATE);
  arduinoPort.bufferUntil(LINE_FEED);
 }
}
catch(Exception e) {
 println(e);
 println("NOTICE: Could not run nunchuckoo! Ensure that your
 nunchuck and arduino are connected correctly" + '\n');
 }
}
```

Arduino + Nunchuck actions read, interpreted and broadcasted if connected and transmitting data:

```
void serialEvent(Serial port) {
 final String arduinoData = port.readStringUntil(LINE_FEED);
 if (arduinoData != null) {
  final int[] data = int(split(trim(arduinoData), ' '));
  if (data.length == 7) {
   // data[0] is joystick left/right
   // data[1] is joystick forward/back
   // data[2] is Tilt left/right
   // data[3] is Tilt forward/back
   // data[4] is not currently used
   // data[5] is z button
   // data[6] is c button
   if (data[0] < 60) {
    dir = "LEFT";
    client.broadcast(dir);
   }
   else if(data[0] > 190) {
    dir = "RIGHT";
    client.broadcast(dir);
   }
   else if(data[1] > 190) {
    dir = "UP";
    client.broadcast(dir);
```

```
    }
    else if(data[1] < 60) {
     dir = "DOWN";
     client.broadcast(dir);
    }
```

Check accelerometer data thresholds and trigger values, then broadcast action if conditions met:

```
    else if (data[2] < 400 && data[5] == 1) {
     dir = "LEFT";
     client.broadcast(dir);
    }
    else if(data[2] > 600 && data[5] == 1) {
     dir = "RIGHT";
     client.broadcast(dir);
    }
    else if(data[3] > 650 && data[5] == 1) {
     dir = "UP";
     client.broadcast(dir);
    }
    else if(data[3] < 450 && data[5] == 1) {
     dir = "DOWN";
     client.broadcast(dir);
    }
    else if(data[6] == 1) {
     dir = "RESET";
     client.broadcast(dir);
    }
   }
  }
}
```

navigation.pde

Inner class to listen and react to broadcast messages

Long switch statement to handle different actions depending on OCD camera selection:

```
void navActions() {
 if (client.messageAvailable()) {
```

```
String[] way = client.getDataMessage();
switch(wallChar) {
```

Navigation actions for front screen:

```
case 'f':
 if (way[0].equals("LEFT")) {
  camSelected.pan(-(rotVar));
 }
 else if (way[0].equals("RIGHT")) {
  camSelected.pan(rotVar);
 }
 else if (way[0].equals("UP")) {
  camSelected.dolly(-(linVar));
 }
 else if (way[0].equals("DOWN")) {
  camSelected.dolly(linVar);
 }
 else if (way[0].equals("RESET")) {
  camSelected.jump(0,0,client.getLWidth()/2);
  camSelected.aim(0,0,0);
 }
 break;
```

Navigation actions for rear(stern) screen:

```
case 's':
 if (way[0].equals("LEFT")) {
  camSelected.pan(-(rotVar));
 }
 else if (way[0].equals("RIGHT")) {
  camSelected.pan(rotVar);
 }
 else if (way[0].equals("UP")) {
  camSelected.dolly(linVar);
 }
 else if (way[0].equals("DOWN")) {
  camSelected.dolly(-(linVar));
 }
 else if (way[0].equals("RESET")) {
```

```
      camSelected.jump(0,0,client.getLWidth()/2);
      camSelected.aim(0,0,client.getLWidth());
     }
     break;
```

Navigation actions for left screen:

```
    case 'l':
     if (way[0].equals("LEFT")) {
      camSelected.pan(-(rotVar));
     }
     else if (way[0].equals("RIGHT")) {
      camSelected.pan(rotVar);
     }
     else if (way[0].equals("UP")) {
      camSelected.truck(linVar);
     }
     else if (way[0].equals("DOWN")) {
      camSelected.truck(-(linVar));
     }
     else if (way[0].equals("RESET")) {
      camSelected.jump(0,0,client.getLWidth()/2);
      camSelected.aim(-1,0,client.getLWidth()/2);
     }
     break;
```

Navigation actions for right screen:

```
    case 'r':
     if (way[0].equals("LEFT")) {
      camSelected.pan(-(rotVar));
     }
     else if (way[0].equals("RIGHT")) {
      camSelected.pan(rotVar);
     }
     else if (way[0].equals("UP")) {
      camSelected.truck(-(linVar));
     }
     else if (way[0].equals("DOWN")) {
      camSelected.truck(linVar);
```

```
    }
    else if (way[0].equals("RESET")) {
     camSelected.jump(0,0,client.getLWidth()/2);
     camSelected.aim(1,0,client.getLWidth()/2);
    }
    break;
```

Navigation actions for top screen:

```
   case 't':
    if (way[0].equals("LEFT")) {
     camSelected.roll(radians(rolVar));
    }
    else if (way[0].equals("RIGHT")) {
     camSelected.roll(-(radians(rolVar)));
    }
    else if (way[0].equals("UP")) {
     camSelected.boom(linVar);
    }
    else if (way[0].equals("DOWN")) {
     camSelected.boom(-(linVar));
    }
    else if (way[0].equals("RESET")) {
     camSelected = new Camera(this, 0, 0, client.getLWidth()/2, 0, -1, client.
          getLWidth()/2, 0, 0, -1, fov2, 1, nrClip, frClip);
    }
    break;
```

Navigation actions for bottom screen:

```
   case 'b':
    if (way[0].equals("LEFT")) {
     camSelected.roll(-(radians(rolVar)));
    }
    else if (way[0].equals("RIGHT")) {
     camSelected.roll(radians(rolVar));
    }
    else if (way[0].equals("UP")) {
     camSelected.boom(-(linVar));
    }
```

```
      else if (way[0].equals("DOWN")) {
       camSelected.boom(linVar);
      }
      else if (way[0].equals("RESET")) {
       camSelected = new Camera(this, 0, 0, client.getLWidth()/2, 0, 1, client.
            getLWidth()/2, 0, 0, 1, fov2, 1, nrClip, frClip);
      }
      break;
   }
  }
}
```

scene.pde

Inner class that lays out basic non stereographic scene

caveScene() function called in FrameEvent() function loop if stereo set to 'off' in config file:

```
void caveScene() {
```

Clear the screen and assign basic scene parameters:

```
 background(0);
 lights();
 noStroke();
 fill(150);
```

Position, colour, draw and rotate plane, box and sphere objects:

```
 translate(0,200,0);
 box(800,0,1000);
 translate(0,−100,0);
 fill(204);
 rotateY(angle);
 box(80);
 translate(400,−150,0);
 stroke(153);
 sphere(200);
```

Assign rotation speed:

```
 angle += 0.02f;
```

Call navigation and joypad functions then close caveScene() function:

```
 navActions();
 joypadAction();
}
```

stereoScene.pde

Inner class that lays out basic stereographic scene

caveScene() function called in FrameEvent() function loop if stereo set to 'on' in config file:

```
void caveScene() {
```

Clear the screen:

```
 background(0);
 lights();
```

Apply OpenGL stereo assignments based on camera selection and call render() function for each eye. pushMatrix() and popMatrix() functions are used to save and restore the coordinate system and ensure the image remains in sync for both eyes. OpenGL uses a different cartesian co-ordinate system from P5 and therefore different axis configurations are required:

```
 ActiveStereoView pgl = (ActiveStereoView) g;
 GL gl = pgl.beginGL(); {
  if(wall.equals("bottom") || wall.equals("top")) {
   stereo.start(gl,
        0f, 0f, 0f,
        0f, 0f, 1f,
        0f, −1f, 0f);
  }
  else {
   stereo.start(gl,
        0f, 0f, 0f,
        0f, 0f, −1f,
        0f, 1f, 0f);
  }
  stereo.right(gl); // right eye rendering
  pushMatrix();
  render(gl);
```

```
  stereo.left(gl); // left eye rendering
  popMatrix();
  render(gl);
  stereo.end(gl);
  }
   pgl.endGL();
}
```

Render scene for individual eye:

```
void render(GL gl) {
```

Push transformation matrix on to the matrix stack then use scale() function to mirror image. This compensates for change in cartesian co-ordinates:

```
pushMatrix();
if(wall.equals("bottom") || wall.equals("top")){
  scale(1,−1,1);
}
else {
  scale(1,−1,1);
}
```

Position, colour, draw and rotate plane, box and sphere objects:

```
noStroke();
fill(150);
translate(0,200,0);
box(800,0,1000);
translate(0,−100,0);
fill(204);
rotateY(angle);
box(80);
translate(400,−150,0);
stroke(153);
sphere(200);
```

Assign rotation speed:

```
angle += 0.01f;
```

Pop transfomation matrix off stack, call navigation and joypad action functions then close caveScene() function:

```
 popMatrix();
 navActions();
 joypadAction();
}
```

NOTE: The following code listings represent additional functionality to the core SCRAPE system and provide for the integration of live database interactions. Certain modifications will be needed in order to customise for new implementations e.g. database connection details, new SQL queries and new variable assignments. The basic structure however should not require any major modification. Also a call to the initialisation() function will need to be added to the setup() function of the core SCRAPE class in order to initialise the database classes.

DBconnect.pde

Inner class that provides Database Access.

Library imports and variable assignments:

```
import de.bezier.data.sql.*;
MySQL msql;

String IP = "192.168.x.x";
String user = "username";
String pass = "password";
String database = "some_DB";

int[] identAry;
String[] localeAry;
String[] descAry;
float[] liveAry;
float[] totalAry;
double[] threshAry;
```

Initial DB connection function:

```
void DBInitial(){
```

Assign variables:

```
  int i = 0;
```

```
  msql = new MySQL(this,IP,database,user,pass);
```

Connect to database, count rows and set counter:

```
  if(msql.connect()){
    msql.query("SELECT COUNT(*) FROM some_table");
    msql.next();
    i = msql.getInt(1);
    println("Count result is "+i);
```

Set required array lengths:

```
    identAry = new int[i];
    localeAry = new String[i];
    descAry = new String[i];
    liveAry = new float[i];
    totalAry = new float[i];
    threshAry = new double[i];
```

Close DB connection:

```
    msql.close();
   }else {
    println ("Initial DB Connection Failed!");
   }
}
```

Function to query database:

```
void DBReadings(){
```

Variable assignments:

```
    int i = 0;
    int identifier;
    String location;
    String description;
    float live_reading;
    float daily_total;
    double daily_threshold;
```

Connect to database, query and assign values to arrays:

```
    if(msql.connect()){
      msql.query("SELECT id, location, 'desc', format(some_field,2),format(
          some_field,2), format(some_field,2) FROM some_table");
      while(msql.next()){
        identifier = msql.getInt("id");
        location = msql.getString("location");
        description = msql.getString("desc");
        live_reading = msql.getFloat("format(some_field,2)");
        daily_total = msql.getFloat("format(some_field,2)");
        daily_threshold = msql.getDouble("format(some_field,2)");
        identAry[i] = identifier;
        localeAry[i] = location;
        descAry[i] = description;
        liveAry[i] = live_reading;
        totalAry[i] = daily_total;
        threshAry[i] = daily_threshold;
        i++;
      }
```

Close DB connection:

```
      msql.close();
    }else {
      println ("Loop DB Connection Failed!");
    }
}
```

DBthread.pde

Inner class that creates new thread to facilitate database calls without interruption to main sketch

Initialise thread

```
class DBthread extends Thread {
  boolean running = false;

  void start() {
    running = true;
    println("Starting DB thread. Waiting for all clients to connect");
    super.start();
```

```
  }
```

If all clients have connected and are still running then query database, reset timer and wait for specified time before running again:

```
  void run () {
    while (running) {
      if(allClientsConnected == true){
        DBReadings();

        timer = 30000;
        println("DB accessed! Next access in "+timer/1000+" seconds");

        try {
          sleep((long)(timer+2000));
        } catch (Exception e) {println("Sleep thread exception "+e);}
      }
    }
    println("End of thread!");
  }
```

Function to call in order to quit thread:

```
  void quit() {
    System.out.println("Quitting thread!");
    running = false;
    interrupt();
  }
}
```

initialisation.pde

Inner class used to assign key variables and load images outside of main SCRAPE class as well as initialise the database thread and make start-up calls to database functions. Modifications will need to be made for custom environments e.g. image references and timer settings.

Reference variables and assignments:

```
DBthread sThread;

PImage bg;
PImage P5;
```

```
PImage SEED;
PImage GNH;
PImage MUR;
PImage OFI;

int timer = 30000;
int savedTime = millis();
int totalTime = 1000;

boolean endOfGroup = false;
double efficiencyNum;
float decimalMinutes;

boolean allClientsConnected = false;
```

Load images, set texture modes and assign colour options:

```
  void initialisation(){
    bg = loadImage("starfield.png");
    P5 = loadImage("p5-logo.png");
    SEED = loadImage("seed-logo.png");
    GNH = loadImage("gnh.png");
    MUR = loadImage("muirhevna.png");
    OFI = loadImage("ofiaich.png");

    textureMode(NORMAL);
    colorMode(HSB,360,100,100,100);

    color colBlue = color(204, 102, 0);
    color colGreen = color(204, 102, 0);
    color colOrange = color(204, 102, 0);
    color colRed = color(204, 102, 0);
    color colGrey = color(204, 102, 0);
```

Connect to database:

```
    DBInitial();
    DBReadings();
    println("ID Array is:\n"+Arrays.toString(identAry)+"\n");
    println("Location is:\n"+Arrays.toString(localeAry)+"\n");
```

```
println(”Live Reading Array is:\n”+Arrays.toString(liveAry)+”\n”);
println(”Total Reading Array is:\n”+Arrays.toString(totalAry)+”\n”);
println(”Daily Threshold Array is:\n”+Arrays.toString(threshAry)+”\n”);
println(”hour is ”+hour()+”:”+minute());
```

start DB thread:

```
sThread = new DBthread();
sThread.start();
}
```

Appendix C

Evaluation Questionnaires

The following pages list the 5 question sheets that volunteers were required to complete for both CAVE and desktop system evaluations.

TRAINING TASK

Volunteer Number: _____

Record the **Real time** and **Today's total** energy consumption readings for the following rooms in the OFiach school:

SDB4 Science 1

Real time: _____ kW/h

Todays total: _____ kW/h

SDB2 Art Room

Real time: _____ kW/h

Todays total: _____ kW/h

SDB1 Bolier House

Real time: _____ kW/h

Todays total: _____ kW/h

TASK 1

Volunteer Number: _____

Record and then add-up the **Real time** energy consumption readings for all **Computer** rooms in the OFiach school:

Note: There may be less rooms than spaces provided below.

Room name: _____ Reading: _____ kW/h

Room name: _____ Reading: _____ kW/h

Room name: _____ Reading: _____ kW/h

Room name: _____ Reading: _____ kW/h

Room name: _____ Reading: _____ kW/h

Room name: _____ Reading: _____ kW/h

Room name: _____ Reading: _____ kW/h

Room name: _____ Reading: _____ kW/h

Total: _____ kW/h

TASK 2

Volunteer Number: _____

Record and then add-up **Todays total** energy consumption for all **Lighting** readings in the OFiach school:

Note: There may be less rooms than spaces provided below.

Room name: _____ Reading: _____ kW/h

Room name: _____ Reading: _____ kW/h

Room name: _____ Reading: _____ kW/h

Room name: _____ Reading: _____ kW/h

Room name: _____ Reading: _____ kW/h

Room name: _____ Reading: _____ kW/h

Room name: _____ Reading: _____ kW/h

Room name: _____ Reading: _____ kW/h

Total: _____ kW/h

TASK 3

Volunteer Number: _____

Find the room that has the highest **Todays total** energy consumption reading in the OFiach school:

Room name: _____ Reading: _____ kW/h

TASK 4

Volunteer Number: _____

Record and then add-up **Todays total** energy consumption costs for all **ESB** rooms in the OFiach school:

Note: There may be less rooms than spaces provided below.

Room name: _____ Cost: _____ €

Room name: _____ Cost: _____ €

Room name: _____ Cost: _____ €

Room name: _____ Cost: _____ €

Room name: _____ Cost: _____ €

Room name: _____ Cost: _____ €

Room name: _____ Cost: _____ €

Room name: _____ Cost: _____ €

Room name: _____ Cost: _____ €

Total: _____ €

Appendix D

System Usability Scale

Industry standard ten point questionnaire for subjective assessments of system usability.

1. I think that I would like to use this system frequently

Strongly Disagree				Strongly Agree
1	2	3	4	5

2. I found the system unnecessarily complex

Strongly Disagree				Strongly Agree
1	2	3	4	5

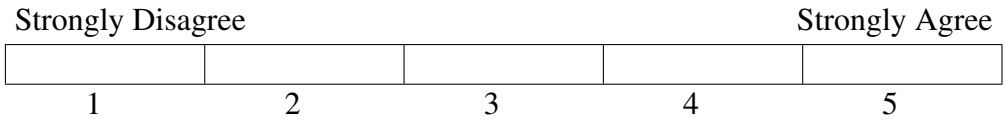
3. I thought the system was easy-to-use

Strongly Disagree				Strongly Agree
1	2	3	4	5

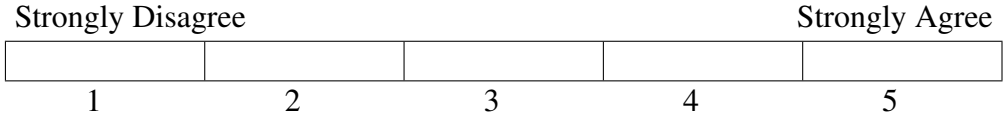
4. I think that I would need the support of a technical person to be able to use this system

Strongly Disagree				Strongly Agree
1	2	3	4	5

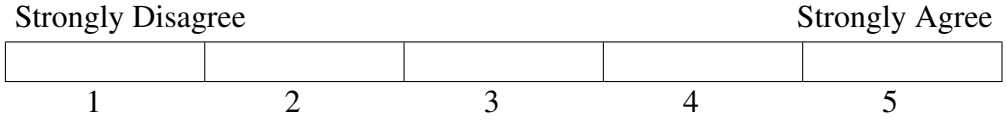
5. I found the various functions in this system were well integrated



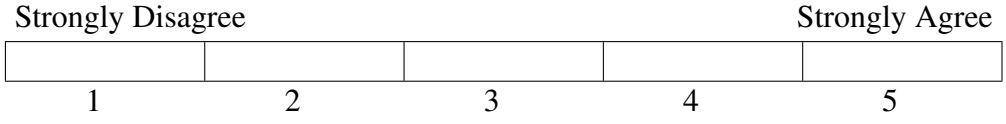
6. I thought there was too much inconsistency in this system



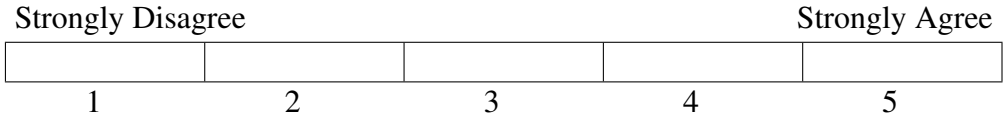
7. I would imagine that most people would learn to use this system very quickly



8. I found the system very cumbersome to use



9. I felt very confident using the system



10. I needed to learn a lot of things before I could get going with this system