An Evaluation of Visual Gesture Based Controls for Exploring Three Dimensional Environments

by

Amy Davidson, B.A.(Mod)

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

August 2012
I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Amy Davidson

August 31, 2012
Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Amy Davidson

August 31, 2012
Acknowledgments

I would like to thank my supervisor Michael Manzke for his advice and suggestions throughout this project. I would also like to thank Rachel McDonnell, Daniel O’Byrne and my family for their time, insight and support. Finally I would also like to thank all the participants who helped to test and evaluate this project’s application.

Amy Davidson

University of Dublin, Trinity College
August 2012
An Evaluation of Visual Gesture Based Controls for Exploring Three Dimensional Environments

Amy Davidson
University of Dublin, Trinity College, 2012

Supervisor: Michael Manzke

Over the last few years, visual gesture based interfaces have received much attention from the academic world, as well as the hobbyist and commercial spaces [1, 2, 3, 4]. With the arrival of the Microsoft Kinect in particular, a host of gesture controlled games and applications have been released [5]. However the vast majority of these have been extremely limited in the scope and complexity of interaction afforded to the user.

This research project aims to evaluate how suitable gesture based controls might be to more complex tasks and interactions, such as the navigation of 3D virtual environments. To do this, a 3D testbed application was developed specifically to evaluate the use of gestures for 3D navigation, focussing on some of the most common 3D navigational tasks, such as freely controlling a virtual avatar and manipulating an orbiting camera.
This application was then used in a series of experiments to gather quantitative and qualitative data pertaining to the relative strengths and weaknesses of gesture controls, using other control types as a comparison point. Data was gathered from participants through a combination of questionnaires, researcher’s observations and statistical data recorded by the application itself.

The results of this study highlight a number of potential limitations associated with the use of visual gesture controls for complex tasks. One of these was a lack of fine-tuned control, which is linked to the current generation of technology’s inability to accurately detect small or fast movements. The non-discrete nature of gestures can also cause issues when a large set of controls are required, as gestures can overlap and interfere with each other. These issues can be further magnified when coupled with the lack of haptic feedback provided by visual gesture controls.

These results identified a number of potential issues with using the current generation of visual gesture based controls for complex tasks, such as the navigation of 3D virtual environments, and that further research into the area is needed.
# Contents

- Acknowledgments iv
- Abstract v
- List of Tables xi
- List of Figures xii

## Chapter 1 Introduction 1
- 1.1 Motivation 2
- 1.2 This Report 3

## Chapter 2 State of the Art 4
- 2.1 History of Input Controllers 5
- 2.2 Natural Mappings 7
- 2.3 Gesture Controls 8
  - 2.3.1 Previous Gesture Based Interfaces 9
  - 2.3.2 Microsoft Kinect 11

## Chapter 3 Design 13
- 3.1 Choosing Input Methods 13
  - 3.1.1 The Kinect 14
  - 3.1.2 Input Methods to Compare Against 14
- 3.2 Choosing A Game Engine 15
  - 3.2.1 Commercial Game Engines 15
  - 3.2.2 Open Source Rendering Engines 16
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3 Extending Ogre to an Experiment Framework</td>
<td>16</td>
</tr>
<tr>
<td>3.4 Separating Three Navigation Techniques</td>
<td>17</td>
</tr>
<tr>
<td>3.4.1 First Person Camera</td>
<td>18</td>
</tr>
<tr>
<td>3.4.2 Third Person Camera</td>
<td>19</td>
</tr>
<tr>
<td>3.4.3 Orbiting Camera</td>
<td>20</td>
</tr>
<tr>
<td>3.5 First Person Camera Tasks</td>
<td>22</td>
</tr>
<tr>
<td>3.5.1 Travel and Look At Task</td>
<td>22</td>
</tr>
<tr>
<td>3.5.2 Collection Task</td>
<td>23</td>
</tr>
<tr>
<td>3.6 Third Person Camera Tasks</td>
<td>24</td>
</tr>
<tr>
<td>3.6.1 Travel and Look At Task</td>
<td>25</td>
</tr>
<tr>
<td>3.6.2 Collection Task</td>
<td>25</td>
</tr>
<tr>
<td>3.7 Orbiting Camera Tasks</td>
<td>25</td>
</tr>
<tr>
<td>3.7.1 Zooming Task</td>
<td>26</td>
</tr>
<tr>
<td>3.7.2 Panning Task</td>
<td>27</td>
</tr>
<tr>
<td>3.7.3 Rotating Task</td>
<td>27</td>
</tr>
<tr>
<td>Chapter 4 Implementation</td>
<td>29</td>
</tr>
<tr>
<td>4.1 Expanding Ogre</td>
<td>29</td>
</tr>
<tr>
<td>4.1.1 Input Management</td>
<td>29</td>
</tr>
<tr>
<td>4.1.2 Audio and Physics</td>
<td>30</td>
</tr>
<tr>
<td>4.1.3 GUI</td>
<td>30</td>
</tr>
<tr>
<td>4.1.4 The Ogre and Experiment Framework</td>
<td>31</td>
</tr>
<tr>
<td>4.2 Cameras</td>
<td>32</td>
</tr>
<tr>
<td>4.2.1 BaseCamera</td>
<td>33</td>
</tr>
<tr>
<td>4.2.2 Derived Camera Classes</td>
<td>33</td>
</tr>
<tr>
<td>4.3 State Based Application</td>
<td>34</td>
</tr>
<tr>
<td>4.3.1 ExperimentApplication</td>
<td>35</td>
</tr>
<tr>
<td>4.3.2 AppStateManager</td>
<td>35</td>
</tr>
<tr>
<td>4.3.3 AppState</td>
<td>35</td>
</tr>
<tr>
<td>4.3.4 Menu States</td>
<td>36</td>
</tr>
<tr>
<td>4.3.5 GameState</td>
<td>36</td>
</tr>
<tr>
<td>4.4 Experiment Task States</td>
<td>37</td>
</tr>
<tr>
<td>4.5 FAAST</td>
<td>42</td>
</tr>
</tbody>
</table>

viii
Chapter 5  Analysis
5.1 Testing Procedure ................................................. 44
5.2 Timed Results ..................................................... 45
5.3 Pre and Post Test Questionnaires .............................. 46
5.4 Between Control Scheme Questionnaires ..................... 46
5.5 Gesture Choices ................................................. 47
5.6 Data Storage ...................................................... 48

Chapter 6  Results and Evaluations ................................. 50
6.1 Participant Sample .................................................. 50
6.2 Features of Each Input Method .................................. 51
   6.2.1 Enjoyment ...................................................... 51
   6.2.2 Interpretation and Response Time ......................... 51
   6.2.3 Mental and Physical Fatigue ............................... 52
   6.2.4 Presence ...................................................... 53
   6.2.5 Ease of Use .................................................... 54
   6.2.6 Ease of Learning ............................................. 54
6.3 Fine-Tuned Controls ................................................ 55
6.4 Overlapping Gestures ............................................. 56
6.5 Effect on Learning .................................................. 57
6.6 Post-Test Opinions ................................................. 58
6.7 Natural Mappings .................................................. 59
6.8 Feedback to User ................................................... 60
6.9 User Opinions of Gestures Based Navigation ................. 60
6.10 Preferred Set of Gestures ....................................... 62

Chapter 7  Conclusions .................................................. 63
7.1 Using Gestures for Complex Tasks .............................. 63
7.2 Limitations ......................................................... 64
   7.2.1 Sample Size and Diversity ................................. 64
   7.2.2 Binary Gesture Identification .............................. 65
   7.2.3 Experiment Design .......................................... 65
   7.2.4 User Choice of Gestures ................................... 66
7.3 Future Work ................................................. 66
  7.3.1 Larger, More Varied Sample Size ................. 66
  7.3.2 Separate Gesture Set Study ......................... 67
  7.3.3 Tailored Gesture Recognition System ............. 67
  7.3.4 Streamlined Gesture Selection ..................... 67
  7.3.5 Full Body Versus Hands Only Gesture Control .... 68

Appendix A Questionnaires ................................. 69

Bibliography .................................................. 79
List of Tables

5.1 Gesture Options for Each Camera Control ............................................. 49
6.1 Enjoyment for Each Input Method ...................................................... 51
6.2 Interpretation and Response for Each Input Method ......................... 52
6.3 Mental and Physical Fatigue for Each Input Method ............................ 53
6.4 Sense of Presence for Each Input Method ........................................... 54
6.5 Ease of Use and Learning for Each Input Method ................................. 55
6.6 Average Times Taken for Each Task .................................................. 56
6.7 Grouped Average Times Taken for Each Task ..................................... 57
6.8 Post-Test Voting Averages ................................................................. 58
List of Figures

3.1 Examples of First Person Cameras .......................... 18
3.2 Examples of Third Person Cameras .......................... 19
3.3 Examples of Orbiting Cameras .............................. 20
3.4 First Person ‘Travel and Look At’ Task .................. 22
3.5 First Person ‘Collection’ Task ............................. 23
3.6 Third Person Camera Tasks ................................. 24
3.7 Orbit Camera ‘Zoom’ Task .................................. 26
3.8 Orbit Camera ‘Pan’ Task .................................... 27
3.9 Orbit Camera ‘Rotate’ Task ................................. 28

4.1 Chaining Structure of Menu and Game States ............. 32
4.2 Hierarchy of the Multiple Camera Types .................. 33
4.3 Class Diagram of the Application’s Structure .......... 34

5.1 Taxonomy of Camera Controls ............................ 48
Chapter 1

Introduction

Today, 3D graphics applications and virtual environments are becoming increasingly common. Since the introduction of 3D computer graphics in the late seventies, the popularity and fidelity of these graphics have improved almost exponentially over time as the technology and rendering capabilities of modern computers have increased to support them. They have now become central to a wide range of application fields, including scientific visualisation, product design and prototyping, army and medical simulation or training, psychiatric treatment, movies and computer games.

However, these three dimensional virtual environments (VE) are generally only shown on a two dimensional display, and as such, we cannot move around the virtual world ourselves to see different parts of, or obtain different views of, the virtual environment. Instead, we must view these worlds through the eyes of virtual cameras which we can then control and move through the environment.

These cameras can move through the world in three dimensions, however the input devices available upon the introduction of 3D virtual environments were better suited to 2D applications [1]. This extra dimension and extra functionality available to the user in these new 3D VEs added an extra layer of complexity to the control scheme used by these applications. But a simple and effective input method is of principal importance to the success of a 3D graphics application, as this affects our sense of presence in the virtual world and overall enjoyment and satisfaction of that environment [6, 7].

Previously, the approach taken to tackle these issues was to create more complex controllers. In the twenty years from 1985 to 2005, games controllers went from a
single d-pad and two buttons (NES controller), to ten buttons, two analog sticks and a D-Pad (Xbox360 controller). Meanwhile in 3D PC applications (which already had numerous mapping options due to the keyboard), these complex controls manifested as combinations of simultaneous key presses and mouse movements.

In recent years this pattern has reversed however, toward input types with more natural and intuitive input methods. The release of the Nintendo DS introduced to mainstream gaming the touch screen and stylus interface which enables the user to interact directly with the content onscreen rather than indirectly though buttons or a mouse \[8\]. This was followed by the Nintendo Wii games console with its new Wiimote controller design. The design of the controller and console peripherals allowed the Wii to determine the motions of the Wiimote in 3D space \[9\]. This enabled players to interact spatially with these games using more natural mappings, such as swinging their arm to swing a tennis racket \[9, 10\].

Microsoft has since continued this trend toward simpler and more intuitive input methods with the release of the Kinect. This device allows users to interact with the Xbox360 or PC using only their body, without the need for a controller at all. These newer control methods have made it significantly easier for casual and novice users to just “pick up and play” games \[11, 12, 13\].

1.1 Motivation

With the backing of various companies such as PrimeSense, Microsoft, Nintendo, Sony, and Leap Motion, motion controls in various forms are becoming some of the main input methods in the entertainment space \[14, 15, 16\]. These motion controls supposedly provide more natural and intuitive mental mappings, especially for casual players. One of the currently most popular and prevalent gesture control devices is the Microsoft Kinect, which performs full body skeletal tracking that enables users to use their whole body to play video games and interact with PC applications \[17\].

However there are doubts as to whether or not these new methods of input are capable of handling the complex interactions that we find in modern 3D virtual environments. There have been indications of the limitations of the Kinect to identify small gestures or fast gestures due to the resolution, frame rate and accuracy of the skeletal tracking of the device \[18\]. Its sensitivity to sunlight and its environment requirements,
such as particular distances at which the user must stand from the device, also pose
problems for developers [19, 20]. There have also been recorded issues of the device
not recognising or misinterpreting some gestures [21].

The goal of this project was to evaluate how well gesture based interfaces can handle
more complex control sets than those found in the current generation of commercially
released Kinect applications - specifically the navigation of 3D virtual environments.
In order to achieve this, a 3D testbed application was designed and implemented for
use with both the Kinect and other, more traditional input methods. Several different
input methods were used to allow for comparisons between them, while taking into
account the user’s previous familiarity with a given input type.

It was decided to test three separate travel techniques, in order to get a broader
view on the suitability of gesture based navigation in 3D environments. A number of
participants were then tested using the testbed application with three different input
methods. Each input method was then rated according to a number of different features
which affect the effectiveness of a user interface in a 3D computer application.

1.2 This Report

This report will contain a review of the current state of the art in gesture based inter-
faces as well as a brief history of game controllers and the current trend toward more
natural and intuitive input methods as they relate to this project.

Some of the major design decisions made during the course of this project will
then be explained, as will the reasoning behind them. Then a detailed summary of
the implementation and development of the experiment framework, the 3D testbed
application, the three travel techniques as well as the tasks designed for each travel
 technique will be given.

The details pertaining to the analysis of the chosen input methods used with this
testbed application will then be given. This will be followed by an explanation and
evaluation of both the quantitative and qualitative results obtained from the exper-
iment participants. Finally, the conclusions drawn from these results concerning the
feasibility of gesture based interfaces for complex tasks, such as the navigation of 3D
virtual environments, will be presented, along with a brief outline of the future work
to be carried out.
Chapter 2

State of the Art

As discussed in the previous chapter, 3D graphics applications and virtual environments are becoming ubiquitous. 3D virtual environments in a variety of forms, can now be found on almost any modern desktop machine or smartphone [22]. However most VEs contain a world much larger than can be displayed from a single viewpoint, and as such, the user must navigate within these VEs in order to acquire different views of the virtual world presented to them. From this we can say that a 3D environment is only as useful or enjoyable as the user’s ability to navigate through it and interact with it effectively [7].

An effective user interface is therefore paramount for the success of any 3D virtual environment based application. These are known as 3D user interfaces (UI) and consist of input devices and interaction techniques for effectively controlling 3D computer generated content [23]. Since the introduction of 3D computer graphics in the seventies, much research has gone into improving 3D UIs through almost countless user studies. However, originally most of this work was focused on interaction techniques [24, 25, 7, 26] instead of input devices.

Unfortunately many of the traditional 2D input devices (e.g. mouses, keyboards and D-Pads) are unsuited to many of the tasks required in 3D applications as they require a mapping from 2D input to 3D positions [27]. Many users find them unwieldy and unnatural to use in 3D applications, possibly due to the attempt to map 6 degrees of freedom (three positional and three orientation) onto such 2 dimensional devices [1]. This has resulted in the evolution of new and existing input devices in order to better
facilitate the user’s interaction with 3D VEs. The history of video game controllers provides an interesting look at this evolution towards input devices better suited to exploring 3D VEs.

2.1 History of Input Controllers

The Paddle  The first recorded computer game was Tennis for Two (1958) [28]. This was controlled using a paddle (a single button and a turning wheel [29]) which allowed one dimensional input from the user. The paddle was used as the main controller for the first video games console, the “Magnavox Odyssey” (1972), and later for Atari’s Pong (1975).

The Joystick  The Atari 2600 (1977) introduced the joystick which became the standard input controller during the late seventies and early eighties [30]. The joystick was a simple controller making it accessible to more casual gamers. It was also well suited to the dynamics of the popular games of that era which featured simple two-dimensional, single-screen game worlds where players were rarely asked to do anything more than move and fire in two dimensions [28].

The D-Pad  With the release of the Nintendo Entertainment System (NES, 1985) came the traditional gamepad, which consisted of a “D-pad” (directional pad) and two buttons. While joysticks could in theory capture greater sensitivity to movements, financial considerations restricted them to very limited directional tracking. A D-pad provided a comparable amount of directional control for considerably less effort on the part of the user.

The next generation of consoles, Sega Genesis and the Super Nintendo Entertainment System (SNES), didn’t provide any major interface innovations except shoulder buttons on the SNES controller [30].

The Analog Stick and Introduction of 3D Environments  From 1995, with the launches of the Sega Saturn, Sony PlayStation, and Nintendo 64 (N64) the computational power and memory space required to make large three-dimensional worlds was now available [30]. To cope with the extra dimension and functionality of play,
more buttons were added to the original gamepad design, but the D-pad (designed to navigate two-dimensional spaces) was poorly suited to the challenges presented by the extra dimension. The movements afforded by it seemed clumsy and restrictive when used in a 3D world. To combat this, Nintendo integrated an “analog stick” into the controller for the N64, enabling the user to provide a vector as directional input and allowing different speeds of movement [28]. This innovation proved so successful that Sony introduced the PlayStation Dual Analogue controller, which added two analog sticks to the original design of their PlayStation controller (generally one for travel and the other for camera control).

The trend of adding more buttons to gamepads continued with future consoles and with the release of Microsoft’s original Xbox in 2001, the Xbox controller consisted of 10 buttons, a D-Pad and two analog sticks. As games and controllers both continued to grow in complexity, the dexterity and advanced mental mapping required of users to play the game also increased [30]. Over time, game interfaces became tailored to hard-core gamers, often alienating the casual player [10].

3D PC Controls  PC input types didn’t mirror this trend of increasing complexity as the keyboard already provided a wide variety of potential mappings. However many find the mapping of keyboard keys to tasks in 3D VEs to be unnatural and difficult to use, requiring even more mental effort than the use of a controller. Some research has gone into creating better input controls for 3D applications, but these are generally designed for specific applications and tasks (e.g. Safe 3D Navigation [31] for 3D modelling suites). Other approaches have designed elaborate input devices (e.g. 3D mouses, joysticks and gamepads) but again these are often better suited to specialised applications (e.g. a joystick suites a flight simulator very well but is not appropriate for many other genres of games).

Reversing the Trend  The Nintendo DS (2004) combined the existing D-pad and button input methods with a touch screen and stylus. The stylus could take the place of many instruments such as a pen or scalpel and allowed users to carry out certain actions and have those actions reflected in the game by the character (a natural mapping, see Section 2.2). Following this trend towards more natural mappings, the Wiimote (the controller for the Nintendo Wii (2006)), was designed with multiple accelerometers
inside it giving it the ability to sense what motions the player is putting it through.

The orientation and movements of the Wiimote can be tracked relatively accurately in the real world and these motions can then be mapped directly into a 3D VE, allowing users to interact with games spatially (for example, swinging a bat, tennis racket, or golf club) [10]. When used in conjunction with the sensor bar, the Wiimote also becomes a pointing device, which can be visualised by the user as an extension of their arm. Due to the Wiimote’s more natural movements, less complex mental mapping is required on the part of the user to master it [32]. By simplifying the interactions required of the user, this revelation in input design opened up the world of games to more casual users [11], who had previously been excluded due to the complex mental mappings required by other control schemes.

The Kinect Continuing this movement to simpler and more intuitive input methods, Microsoft released the Kinect, a motion sensing input device, in 2010. The Kinect allows players to interact with the Xbox360 console using body gestures, without the need for a controller at all. However since the Kinect can also be connected to a PC, many applications outside of the games industry have been developed that take advantage of both the natural interface it provides and the lack of any need for a controller [4, 33]. More information on these applications and the device will be given in Sections 2.3.2 and 3.1.1.

2.2 Natural Mappings

These evolutions in input devices have brought a variety of more intuitive and natural mapping of controls in 3D environments to the user. Natural mapping is typically thought of as how closely actions carried out in the real world correspond to the resulting action in the application [34]. Some research has claimed that more naturally mapped input controllers should allow users to quickly access mental models of real-world behaviour, thereby providing more information about how to interact with the environment in both accuracy and quantity [35]. For example, swinging the Wiimote in WiiSports Tennis [36] is very close to the real-world behaviour of swinging a tennis racket, this allows players to immediately play the game without learning unnatural mappings in the form of button combinations.
The results of experiments carried out by Skalski et al.\cite{32} have provided evidence that the use of more natural input controls can improve both a user’s spatial presence and enjoyment. Furthermore, experiments have shown that gestural controls are perceived as more natural (than a gamepad), especially for arm and hand gestures\cite{32,37}. Overall, these results indicate that gestures have the potential to provide very natural mappings in many situations, which in turn can provide users with a more pleasing experience when using 3D applications.

2.3 Gesture Controls

Even from a young age we use gestures as a form of communication and hence, gestures may be considered a very natural communication channel\cite{38,39}. In much of the current literature, gestures are defined as the spatial position and pose of a person’s body (or part of it, such as the hand and fingers)\cite{1}, and is split into two distinct subgroups; Static gestures, where the user is not moving (e.g. posing for a photograph), and dynamic gestures, defined by the motion path of the body (e.g. a swing or wave of an arm)\cite{40}.

As the technology for gesture recognition has improved in the last decade or so, experiments using gesture control for video games and 3D design environments are providing increasing amounts of evidence that gesture control is perceived by users as natural and intuitive, and that it provides an engaging and immersive experience for the user\cite{38,41,42,2,43}. Due to their natural mappings, the learning curve associated with understanding and using gestures as a form of input is reduced\cite{42,1}. Additionally, gestures could theoretically provide a more natural interface for those who find it difficult to interact with computers using conventional methods\cite{44} (e.g. the elderly or disabled).

The large scale implementation of gesture based interfaces has been slow for a number of reasons however. Many devices do not have the computational capabilities necessary as they were not specifically designed for such purposes (gesture tracking can be extremely resource intensive) and differing environmental conditions (e.g. lighting) present difficulties to producing generalised solutions\cite{15} p. 4. As well as this, some gesture tracking requires specific hardware solutions (e.g. high definition cameras, wearable accelerometers) which few users have access to. Also previous APIs used very
low level libraries and algorithms which increased the perceived difficulty in working with them and lead to little widespread development and a relatively small community of active developers and researchers. In addition to these restrictions, gesture recognition has to overcome the significant challenge of temporal, spatial and style variations between users [46].

2.3.1 Previous Gesture Based Interfaces

Gesture based human computer interaction has been a research area for over 30 years. One of the first widely known gesture controlled interfaces was discussed in “Put that There” [47] in 1980. The system was a combination of pointing gestures, calculated using a sensor cube worn on the user’s hand or wrist and speech recognition. Another approach used magnetic trackers worn by the user at the waist and head to identify a leaning gesture as a form of navigation through a VE [48].

Accelerometer Based Approaches

In their study of accelerometer-based gesture control of design environments, Kela et al.[38] noted that gesture commands were perceived as “natural” by users, especially those used in 3D space. In order to discuss the design issues associated with 3D spatial gestures Payne et al.[41] developed their own inertial motion tracking device. The results of their study showed that users preferred games with simple gestures, allowing them to immediately understand how to play.

After the release of Nintendo’s Wii, researchers made use of the relatively cheap Wiimote and its accelerometers for gesture recognition. Kratz et al.[40] used hidden Markov models to overcome the issue of different user styles, while Chen et al.[49] modified the Wiimote in an attempt to seamlessly support both 2D and 3D interfaces. Takala et al.[2] used modified Wiimotes as 6 DOF controllers to evaluate their new user interface system. The results of their tests indicated that the 3D UI with gestural inputs was more fun, intuitive and realistic to use than a traditional mouse interface. However an inertial sensor, such as those above, can only identify gestures that it itself is being put through.
2D Vision Based Approaches

Vision based techniques on the other hand have the potential to allow the recognition of gestures almost anywhere within the camera’s field of view. Hand and arm gesture control has been a popular approach for interaction with computers in the past [50, 44, 51] as most of this interaction is only in two dimensions and can be identified from two dimensional images.

Starner et al. [52] developed a wearable IR camera that could detect hand gestures, even in the dark. Their system could be used to control home automation, however because it was a wearable device, it could also be used as a monitoring system for the elderly or disabled (e.g. by detecting a tremor of the hand). Two dimensional full-body gesture recognition has been in development since the late 90’s [53], but it was first brought to mass market with the release of the PlayStation EyeToy in 2003. This USB peripheral for the PlayStation2, could “translate your body movements into game controls allowing you to physically interact with games using your body” [54].

3D Vision Based Approaches

Unfortunately, as mentioned previously, 2D input devices were not well suited for many of the tasks required in 3D applications. The development of 3D vision gesture recognition techniques has been held back due to the computationally expensive calculations required for such techniques, but the idea is not a new one [55].

By 2009 Varona et al. [46] had developed a system which could identify the position of a user’s arms and torso in three dimensional space using the input from two cameras. Applying their system to a game of Tetris, the system correctly identified 86% of the participants’ gestures and obtained over 95% accuracy in classifying identified gestures.

In developing their system they tackled some of the common challenges associated with gesture recognition. For spatial variations they normalized the identified joint positions, temporal variations were managed using a temporal gesture representation and style variations were solved using a non-parametric scheme for the learning and recognition of gestures.
2.3.2 Microsoft Kinect

With the release of Microsoft’s Kinect (in 2010), 3D gesture recognition has become a viable input method for many applications and the academic world has performed much research and development of gesture based user interfaces and navigation techniques in recent years.

As a few examples; Stannus et al. [1], Boulos et al. [56] and Francese et al. [43] have used the Kinect to manipulate and navigate GoogleEarth and Microsoft Bing Maps in 3D while DSouza et al. [57] attempted to use gestures to navigate architectural models as a more natural interface for architectural clients. Takala et al. [2] have recently added the Kinect as an input device to their UI system and Papadopoulos et al. [58] created an interface for 3D navigation using the Kinect (however their interface uses only the users hands).

In the commercial world however, there are relatively few example of using the Kinect to freely navigate 3D environments. Many of the titles developed for use with the Kinect are relatively simplistic when compared to titles designed for use with traditional control methods. Several game designers have put this down to the fact that a number of traditional game design rules and paradigms must be severely altered when using the Kinect [59]. Many studios have found it extremely difficult to adjust to the additional requirements and limitations developing for the Kinect can have on the design process [60].

An example of these difficulties can be seen during the development of “Puss In Boots”, when Blitz Game Studios had to completely rethink their approach to the game’s design in an effort to better incorporate the strengths and weaknesses of the Kinect once it was decided to produce the title as a Kinect game [59]. They were faced with questions many developers would not give huge consideration to when developing for a controller based game such as “How do we get the player to do even basic tasks like, moving or looking around?” . Twisted Pixel also spent weeks discovering the capabilities and limitations of using the Kinect for gesture controls before designing their game play mechanics around this knowledge[60].

These and many of the other existing games released for use with the Kinect place the user on a fixed path through the game world, so the user no longer has the freedom to explore the game on their own. There are some exceptions to this, for example
Kinect Disneyland Adventures [61] allows the player to navigate the park in 3D through gestures, but the game play is kept separate from this navigation. These design choices are generally ascribed to the Kinect’s inability to identify small or fast gestures, the lack of control developers have over the end user’s environment (as the Kinect can be sensitive to sunlight), and the tendency for user’s to slowly alter a gesture over time until it no longer works [18].

The number of applications in the academic world using gestures when navigating 3D environments compared with the number in the commercial world raises the question of how feasible gesture based 3D user interfaces truly are for a wide variety of 3D navigation tasks, given the current level of the technology found in the Microsoft Kinect and similar devices.
Chapter 3

Design

This chapter will discuss some of the major design decisions made during the course of this project and a few of the major challenges presented by this project. Firstly, the choice of input methods used in this project will be discussed and justified. Secondly the choice of game engine with which to construct this project’s testbed application will be examined and the need to extend this engine to create an experiment framework will be explained. The decision to separate the study between three navigation techniques will then be covered. Lastly, the design of each of the experimental tasks for each of the three navigation techniques will be discussed.

3.1 Choosing Input Methods

As the Microsoft Kinect is currently the most prolific and widely available full body gesture based input device, it was decided early on in development to use the Kinect as the input device for visually identifying gestures. However, in order to better evaluate the suitability of gesture based control for the complexities of 3D navigation, it was decided to compare gesture based controls against other, more traditional forms of input, such as the keyboard and mouse and a modern gaming control pad (an Xbox360 controller was chosen for this).
3.1.1 The Kinect

The Kinect is a motion sensing input device released by Microsoft in November 2010 which went on to sell over 8 million units in its first 60 days on the market [5]. The device is capable of identifying and tracking two human body shapes simultaneously through the use of its depth sensor and RGB camera. The depth sensor technology created by Zalevsky et al. [62] from PrimeSense is used to create a 3D reconstruction of the room using an infrared (IR) projector and sensor with an RGB camera. This reconstruction is then used to interpret specific gestures, enabling users to interact with electronic devices without the need for a physical controller. By identifying a user’s gestures, the Kinect can, in theory, provide a more natural user interface for a variety of applications and electronic devices.

Although, the Kinect was originally designed and created to expand the audience of Xbox360 users to include more casual gamers, it has since proved to be extremely popular amongst hobbyist, as well as academic and commercial programmers, as an input device for a wide range of applications [63, 1, 56, 43, 57, 2, 58, 3]. This popularity stems from the fact that the Kinect provides full body motion tracking at a fraction of the price of previous solutions and without the need for the user to wear or hold anything. The release of Microsoft’s “Kinect for Windows” SDK and other open source drivers further reduced the barrier for entry for Kinect users to create their own natural interface controls and applications.

3.1.2 Input Methods to Compare Against

Today, almost everyone who uses a computer interacts with it using a combination of a keyboard and a mouse (though touch screens are becoming increasingly common with the increased popularity of tablet PCs and smart phones). In contrast, gesture based interfaces are still in their infancy and not a common form of interaction with desktop computers.

Given the likely familiarity most participants would have with a keyboard and mouse input system (when compared to a gesture based interface), a keyboard and mouse were chosen to compare against the gesture based interface. A traditional games console controller (specifically an Xbox360 controller) was also chosen as it is the usual control type for most 3D games today.
These inputs were chosen to give a good baseline for the comparisons against gesture based controls and to allow the exploration of the Kinect’s supposed natural mapping advantages over these more traditional and familiar devices.

Other input methods, such as a 3D mouse and a joystick, were considered for this evaluation, but these input devices are generally designed and highly optimised for use with particular applications (namely, 3D modelling suites and flight simulator games). Another factor which led to the exclusion of these devices from the experiments was the increase in time that would be required for the development of this project’s testbed application and the extra time required to perform the evaluations.

3.2 Choosing A Game Engine

One of the first design decisions required before any experiments could be created was the choice of a game engine in which to create the necessary 3D environments. While it would have been possible to design and code the entire system from scratch, using an existing game engine would serve to simplify and accelerate the development process by providing an existing framework in which to develop the testbed application.

The chosen game engine would be required to provide a scene-graph (a structure designed to hold and arrange the contents of a graphical scene in a hierarchical manner), a rendering engine for 3D graphics, collision detection, and audio playback for each experiment. The chosen game engine would also be required to integrate with the Kinect and the chosen gesture recognition library. Fortunately, there are a large number of game engines freely available for non-commercial use and a number of potential game engines were considered before eventually deciding on one.

3.2.1 Commercial Game Engines

Among the researched engines were the Unreal Engine, CryENGINE and Unity Pro. All three of these engines have proven their capabilities and popularity in multiple commercial products, and each has its own free version available for non-commercial projects (the Unreal Development Kit [64], CryENGINE 3 SDK [65] and Unity [66]). However some of these engines are better suited to particular genres of game design (e.g. First Person Shooters) and as the exact format each experiment would take was not
known at this point of development, it was decided to explore more open and generic options as these engines might have placed restrictions on the experiment design.

3.2.2 Open Source Rendering Engines

In order to avoid prematurely restricting the experiment designs, it was decided to explore lower level engines which could grant more flexibility to the later design stages.

Irrlicht, the first of these engines to be considered, is a cross-platform, open source real-time 3D graphics engine written in C++. It is platform independent and can use Direct 3D, OpenGL or its own renderer to render 3D scenes. While Irrlicht is primarily a rendering engine it also includes a number of extra features such as collision detection. However it has not been significantly updated or maintained in several years [67].

Ogre (Object-Oriented Graphics Rendering Engine), a scene-orientated 3D engine also written in C++ was also investigated. Similar to Irrlicht, Ogre is primarily a rendering engine allowing rendering using either OpenGL or Direct 3D, but unlike Irrlicht, Ogre does not provide collision detection out of the box and also has a steeper learning curve associated with getting to know the engine. However there are many libraries and plug-ins designed and implemented for Ogre which provide any extra functionality. This makes Ogre extremely flexible for use in any graphical application by allowing the user to decide which extra components are necessary for their own application e.g. physics, input management, audio and GUI libraries. This also keeps projects as lightweight as possible by not including a large list of features unnecessary for smaller applications such as this project.

Ogre is also stable and well maintained with a large and mature development community [68]. It was for these reasons, coupled with the researcher’s previous experience with the engine, that OGRE was the used for the development of this project’s testbed application.

3.3 Extending Ogre to an Experiment Framework

From the list of requirements for this project’s game engine (stated at the start of Section 3.2), Ogre of course provides a rendering engine and it also provides a scene-graph in the form of its SceneManager class. But, by default, it is missing some key
higher level functionality that was required for this project’s testbed application.

These features include a physics engine for collision detection, the ability to play sound effects to enable audio feedback to experiment participants and input management. However these features are often needed in 3D applications and the Ogre community provides a number of options for each, in the form of additional libraries and wrappers.

In order to allow the rapid development of any number of unique experiments later in the project, it was decided to create an experiment framework consisting of all common functionalities that the experiments were expected to require from the outset. This framework would consist of the main game engine as well as a menu system, a GUI system, input management and a camera navigation system.

A flexible and reusable menu system would be maintained by creating each scene of the testbed application as a state (derived from a base AppState class), all of which are managed in a stack by a state manager. All common variables and functionality expected of each experiment (including a camera navigation system and GUI components), could then be placed in a single parent state. This structure will be discussed in further detail in Chapter 4.

### 3.4 Separating Three Navigation Techniques

Within the realm of 3D applications, there exist a number of techniques and metaphors used to navigate and manipulate the environments presented to the user, but there is no “best” technique for all forms of 3D navigation. The results of experiments carried out by Ware et al. [24] indicated that their “Scene-In-Hand” metaphor was well suited to manipulating an object but was not very good for navigation, whereas the opposite was true of their “Flying Vehicle Control” metaphor. Therefore the technique used for a particular 3D application is typically chosen due to it’s suitability for the tasks required by the application.

In order to get a broader view on the suitability of gesture based control of exploration in 3D environments, it was decided to test the commonly used travel techniques in video games (a free-moving walking metaphor) and 3D modelling suites (an orbiting metaphor with a movable centre of rotation) separately. The walking metaphor was further broken down into a first person navigation technique and a third person nav-
Separating navigation into three distinct sets of controls made it possible to determine if gesture based controls are better suited to any one of the 3D navigation techniques and hence better suited to the tasks associated with that technique (i.e. model investigation with the orbiting metaphor, or travelling for either the first or third person navigation techniques).

For the remainder of this report, “travel” will refer to the translation of the viewpoint in the VE relative to a given co-ordinate system, while “orientate” will refer to rotating the viewpoint around a given point in 3D space. Navigation technique will also be referred to as a camera type.

### 3.4.1 First Person Camera

First person cameras have become increasingly common in computer games with some of the most popular games today, such as Call of Duty, utilising this camera type. This camera type is a method used to control a character’s movement through the game world from that character’s viewpoint, so that the screen displays what the character would see with their own eyes (see Figure 3.1 for examples).

A first person camera orientates around its own position in 3D space and its coordinate system is created with X and Z axes parallel to the ground plane and Y axis parallel to the character’s head. The viewpoint is fixed at the character’s eye position, and the camera moves in response to the character’s movements, allowing players to feel immersed in the game world.

Figure 3.1: Examples of First Person Cameras

(a) The Elder Scrolls V: Skyrim, 2011, Bethesda
(b) Mirror’s Edge, 2008, Electronic Arts
(c) Call of Duty 4: Modern Warfare, 2007, Activision
perpendicular to the ground plane. The Z axis is in line with the direction the camera is facing at all times while X runs perpendicular to this direction.

When using this camera type the user may travel forward, backward, left or right, and may rotate the viewpoint freely around the Y-axis (yaw). To prevent gimbal lock and reduce disorientation for the user, the camera may not be pitched (rotation around X axis) more than positive or negative 60 degrees from the horizon. If the combination of yaw and pitch applied from a particular update results in a roll of the camera (i.e. rotation around the Z axis), the rotations are adjusted to remove the rolling effect as this can be very disorientating for users.

3.4.2 Third Person Camera

In many older 2D games a third person camera was a viewpoint placed to the side of the user’s character that followed the character as they moved across the screen. However in most modern computer games, a third person camera is defined as a viewpoint which is rendered from a fixed distance behind and slightly above the user’s character, allowing users to see a more strongly characterized avatar.

This project uses the former type, shown in Figure 3.2. The co-ordinate system for this third person camera is created in a similar manner as the first person camera, but it is relative to the character and not the viewport. The centre of rotation for the character is at its current position in space while the centre of rotation for the camera is placed slightly above this point.
As with the first person camera, the user may move forward, backward, left or right, however this translation is applied to the character while the camera remains a fixed distance from it (giving the effect that the camera is following the character). The camera remains at a fixed yaw rotation relative to the character at all times and any input to rotate the camera from left to right is instead applied to the character. In contrast the camera may change its pitch to rotate above or below the character while the character’s pitch is fixed. Rolling of the camera is removed using a similar approach to the one used for the first person camera. Lastly, the distance from the character at which the camera is positioned may be adjusted by zooming the camera in or out.

3.4.3 Orbiting Camera

This camera type is modelled on the navigation of a viewport through a 3D scene used in 3D modelling suites and some 3D computer games, which gives the user almost complete freedom to position and orientate the camera wherever they wish (see Figure 3.3 for examples). This navigation is typically split into three techniques: “rotating” around a centre point, “zooming” toward and away from this point, and “panning” to move this point parallel to the plane created by the screen.

The co-ordinate system of the orbiting camera is an extension of the typical screen-space co-ordinate system. The X axis runs along horizontally to the screen and Y runs vertically. The Z axis, perpendicular to both these axes, then runs into the screen. The centre of rotation is the position of the camera’s target which always appears in
the centre of the screen. Travel in this camera system refers to the translation of the target point with respect to the X and Y axes and the translation of the camera centre along the Z axis (both of which translate the viewpoint).

When this camera is used, the user may rotate freely around the Y-axis while rotation about the X-axis is limited to positive and negative 89 degrees from the horizon to prevent gimbal lock. This camera cannot move forward and backward but rather zooms toward and away from its target point (i.e. centre of rotation). However, unlike the previous two camera types, this camera may move up and down as well as left and right. This is done by moving the camera’s target point parallel to the screen plane, though the target point may not move along the Z-axis.

Nine Unique Experiments

It was decided to create three separate cameras, each of which would use three different input methods, resulting in nine distinct experiments. This approach was decided on to give a broad range of results and allow for in-depth comparisons based on several factors. However, the time frame given for the development of this project’s testbed application would not allow for the separate development of all nine experiments.

By using the experiment framework described in Section 3.3, the time required to create each experiment could be drastically reduced, allowing more time for the refinement and testing of each of the experiments. This was achieved by grouping the functionality common to all experiment states into a single parent state from which all experiment task states could inherit, allowing more time to be spent designing and implementing the tasks, rather than working on the basic 3D environment and GUIs required in each.

In the interest of comparing each input method equally, the same tasks for each camera type would be used for all three input methods (i.e. the tasks designed for the orbiting camera would be the same for all three input methods). However, the ordering in which participants used each input method was randomised to prevent a bias toward the later controls where the user’s previous familiarity with the task could serve to provide an advantage.
3.5 First Person Camera Tasks

To test the first person navigation technique, two separate tasks were designed. Each task would require the user to travel through the 3D environment and re-orientate the viewpoint (both separately and simultaneously). The design of these tasks were inspired by, and based on, Bowman’s Travel Testbed evaluation [25] and D’Souza’s evaluation of their own Kinect interface [57].

3.5.1 Travel and Look At Task

The user’s goal in this task is to travel through the 3D environment to a highlighted area and once inside this area, rotate the viewpoint to face a particular target object. The user may carry out these two subtasks in any order but the conditions of both subtasks must be satisfied simultaneously in order to finish this task.

Once the conditions of both subtasks are satisfied a progress bar on the screen begins to fill. This bar continues to fill up so long as both subtasks are satisfied, however if any of the conditions for either subtask are no longer satisfied the progress bar is reset to zero. The progress bar is there to ensure that the user has consciously and continuously satisfied both subtasks and has not fulfilled either subtask only momentarily by entering and overshooting the target area or desired viewpoint orientation.

A transparent crosshairs is overlaid in the centre of the screen as a guide to users.
for the “look at” subtask. When the user is looking at the target object through the crosshairs, this subtask is satisfied.

To let the user know when they have satisfied one of the subtasks, feedback in the form of a noticeable colour change is applied to both the highlighted area and the target object once the conditions of the respective subtasks have been met. Additionally, the crosshairs change colour when the user is looking correctly at the target object as additional feedback. This can be seen in Figures 3.4a and 3.4b.

### 3.5.2 Collection Task

The user’s goal in this task is to collect a number of objects by re-orientating the viewpoint to find them and by then travelling through the 3D environment to collect them. For the purpose of this task “collecting” an object simply involves walking into it.

When the user collides with an object, a short audio clip is played to inform the user that they have successfully collected that object. To prevent the user having to spend concentration on remembering how many objects they have collected and how many there are in total, the number of objects they have currently collected and the total number of objects to collect are displayed at the bottom corner of the screen (see Figure 3.5).
The positions of each object are predefined for each repetition, rather than randomised, in order to reduce any bias which may occur between tests if these objects were placed randomly. There is no fixed order in which the user must collect all of the objects, however the distance of each object to the previous and next object is constant, when considered in the optimal order.

To ensure the user fully understands this task and its audio feedback, a simple practice level with only a single collectable object (placed directly in front of the user’s starting position and orientation) is used as a “practice test”. The data obtained from this “practice test” is discarded so as not to bias the later results.

3.6 Third Person Camera Tasks

To test the third person navigation technique the same two tasks that were employed for the first person camera evaluations are used. The decision to do this was based on the fact that many 3D computer games use either (or sometimes both) of these camera types. By performing the same evaluation on both camera types, we can compare the suitability of gesture based control to either of these camera types equally, without having the difficulty or user’s enjoyment of the tasks affecting their opinion of the navigation technique.
3.6.1 Travel and Look At Task

Again the user must travel to within the highlighted area and orientate the camera to face the target object. The main difference of this version of the task from the first person version is that it is the character which must be within the highlighted area, not the user’s camera (see Figure 3.6a). The user must still align the target object in the centre of the screen, using the crosshairs as a guide.

3.6.2 Collection Task

Similarly to the first person version of this task, the user must navigate through the 3D environment to collide with the collectable objects in order to collect them all. However, in the third person version of this task the character must collide with the objects in order to collect them, rather than the camera colliding with them (see Figure 3.6b).

3.7 Orbiting Camera Tasks

It was decided to test each of the main camera manipulation controls required in a 3D modelling application separately, as it was felt that a task requiring all of them at once would be too difficult and abstract a notion for users who have never used such an application before.

The camera controls used for the testbed application were the three main camera manipulation tasks required in 3D modelling applications. These controls are rotating around a target point, zooming toward and away from this target point, and panning across the screen (moving this target point).

The 3D environment developed for these three tasks consist of a transparent ground plane with a grid texture, akin to those seen in most 3D modelling suites, and a simple model of a dice. A dice was chosen for its simple and obvious sides allowing for clear and concise instructions to users, as well as most users’ likely familiarity with the object.
3.7.1 Zooming Task

As its name suggests, the goal for this task is to zoom the camera to a particular distance from the dice. The desired distance is presented to the user through the use of a semi-transparent silhouette of the dice, overlaid above the initial rendered image of the 3D environment (Figure 3.7a).

Due to the effect of a perspective camera lens, the silhouette of a dice changes at different zooms. Therefore, each level of this task was manually chosen, with the corresponding silhouette assets produced. At this point, choosing fixed values for each zooming level was not considered a negative aspect of the design, as this helped to ensure that all users were given identical tasks, with equal levels of difficulty.

When the user moves the camera to the correct zoom (within a margin of error proportional to the zoom), the silhouette undergoes a noticeable colour change as feedback to the user (Figure 3.7b). Similarly to the “travel and look at” task, a progress bar also begins to fill when the user is at the correct zoom. This progress bar will fill up so long as the user remains at the correct zoom, while resetting if the camera is zoomed out of the allowed margin of error. The time required to fill the progress bar ensures that the user is intentionally at the correct zoom and has not briefly passed through the correct zoom and overshot it.
3.7.2 Panning Task

The goal of this task is to pan the camera so the dice appears in the centre of the screen. To display to the user how the scene should appear when the dice is in this location, another semi-transparent silhouette was used. As with the zooming task, the starting offset for the camera was not chosen randomly so as to prevent any unfairness of levels between participants.

When the camera is positioned so that the dice fits snugly within the silhouette, the colour of the silhouette is noticeably changed and a progress bar begins to fill (Figure 3.8b). As described in previous tasks, if the user moves the camera so the dice no longer fits in the silhouette, this progress bar is reset and the user must try again to complete this task.

3.7.3 Rotating Task

The user’s goal in this task is to orbit the camera around the dice to face a particular side of it. The user is deemed to be facing a side of the dice if it is in the centre of the screen. The side which the user must rotate to in order to complete this task is displayed at the top of the screen.

In order to illustrate to the user which side of the dice they are currently facing, the side they are focused on is highlighted by changing the colour of that side. A crosshairs
is also overlaid on the screen to aid the user in placing the correct side in the centre of
the screen. The correct side of the dice is highlighted a green colour when it is focused
on, while all other sides of the dice are highlighted with a red colour (see Figures 3.9a
and 3.9b).

Once again, the user must hold the camera facing the correct side while a progress
bar fills. If the camera is moved to focus on a different side of the dice, the bar is reset.
Chapter 4

Implementation

The previous chapter illustrated some of the major design decisions made during the course of this project and this chapter will describe the structure, architecture and implementation of the resulting application. Firstly, the expansion of the Ogre engine to a full experiment framework will be described. Next the implementation of the multiple navigation techniques will be explained. Then the overall architecture of the application and how it saved development time and shortened the experiment session length will be discussed. Lastly the implementation of each of the experiment tasks and the chosen gesture recognition software will be explained.

4.1 Expanding Ogre

As discussed in the previous chapter, Ogre required a number of extra features and functionality not provided with the base engine in order to meet all this project’s requirements. These features include a physics engine, input management and the ability to play music and sound effects. The application designed for this project also required a menu system, a GUI system and a camera navigation system which will all be discussed in the following sections.

4.1.1 Input Management

Included within the Ogre SDK, and used by many of the example applications and tutorials, is the OIS library. This is an open source, cross-platform input library that
supports keyboard, mouse and joystick inputs. It was decided to make use of this library rather than developing an input manager from scratch as this made the code from many of the tutorials and examples provided with the engine, and by the Ogre community, compatible with this application.

A short online tutorial [69] provided a small wrapper class for Microsoft’s XInput library, which was added to this application in order to read input from an Xbox360 controller. Lastly, to interpret the gestures identified by the Kinect, the Flexible Action and Articulated Skeleton Toolkit (FAAST) was used (and will be discussed in detail in Section 4.5 of this report).

4.1.2 Audio and Physics

Some of the tasks discussed in the previous chapter require the ability to play sound effects in order to provide audio feedback to the experiment participants. There are many sound managers provided by the Ogre community, including Audiere, wrappers for OpenAL (OgreAL, OgreOggSound), and Fmod sound managers. Due to the researcher’s previous experience with the library, an Fmod SoundManager was used with this project.

There were also a number of options available for a physics engine to use with Ogre; OgreNewt, OgreBullet and NxOgre to name a few. In the end it was decided to use a minimal collision detection class (CollisionTools [70]) as this was the only physics engine feature actually required by the experiments. The use of a more fully featured option would have only served to increase the complexity of the project’s code base and increase its development time. The CollisionTools class detects collisions by ray tracing through the scene (usually from a user’s previous position to their new position) to find the first intersected object in the scene. If this ray intersects with any objects bounding box, further tests are then performed to determine if the ray has intersected with the mesh associated with this bounding box.

4.1.3 GUI

All GUI components used within this application were created using either the SDK-Trays library (provided with the OgreSDK) or Ogre::Overlays. The SDKTrays library provides the implementation of many commonly used GUI elements, such as buttons,
labels, check boxes, progress bars, drop-down lists and sliders. The library also controls
the positioning of these elements and provides the textures used for each component.

However the *SDKTrays* library does not give much freedom to the developer to
create their own GUI elements or to edit the look and feel of the ones it provides.
Therefore any component required by this application which was not provided by *SDK-
Trays*, or which required a GUI component with no border/background shading (which
*SDKTrays* automatically adds) was created using custom made *Ogre::Overlays*. This
included the crosshairs, which were required for a number of tasks, and the silhouette
overlays, required in the Orbit camera tasks.

### 4.1.4 The Ogre and Experiment Frameworks

The *OgreFramework* class (taken from Ogre tutorials [71]) was edited to include some
additional functionality required by this project’s application. This class is created
as a singleton class, and was designed to allow any state of an application access
to its functionality and members. These members include a render window, log file,
keyboard and mouse states, GUI manager and other classes common to many 3D
computer applications. It is also responsible for initialising all of its member variables
at the start-up of an application to allow its members to be used across the application.

Functions to transfer a point in 3D space to 2D screen co-ordinates were added to
this class as well as the ability to cleanly and properly remove an object (and all of its
children) from a scene. An *Fmod SoundManager* was also added to this class in order
to give any state the ability to play music or sound effects.

In order to keep the experiment task details as abstracted as possible, another sin-
gleton class was implemented for this project. This class, *ExperimentFramework*, was
designed to hold all information concerning each task’s fixed values, the details of state
chaining, the current user’s ID as well as a separate log file designed to store experi-
ment data; essentially any information or functionality associated with the application
and pertaining to only the experiments.

The details concerning each task included its number of repetitions, the XML file
containing the arrangement of its scene elements and instructions to display to the
user on how to complete its associated task. The *ExperimentFramework* class is also
responsible for logging messages containing the time the user has taken to complete
a task and for informing each experiment task which state is to follow it when it exits. This functionality allowed for the repetition of each task and the chaining of all states to be automated, significantly reducing the time it took to perform each experiment and reducing the amount of direct input needed from the researcher during these experiments. An overview of this chaining architecture can be seen in Figure 4.1.

4.2 Cameras

The application for this project required three separate cameras with different navigation techniques. As this is a relatively low number, creating three completely detached classes (with a small amount of repeated code) was a feasible solution. However, as mentioned in Section 3.3, an instance of this system would be necessary for the single application state containing all common functionality of each experiment. Therefore all three required cameras were created in a hierarchy with a single abstract base class (see Figure 4.2).
4.2.1 BaseCamera

The *BaseCamera* class provides an interface to the common functionality found in the camera navigation system for each of the experiment states. This class stores all members common to each of its derived classes, such as movement and rotation speeds, collision detection details and a pointer to the *Ogre::Camera* class which is required by the viewport in order to render to the screen.

Implemented in this class is an *Update()* function (which is called every update by each state which requires a particular navigation technique), a *setCollisionTools()* function (which sets the collision detection details associated with a particular scene) as well as functions to pass the state of all input controllers to the navigation system.

4.2.2 Derived Camera Classes

Each derived class implements their own *InterpretInput()* and *MoveCamera()* functions. *InterpretInput()* serves to identify which key on the keyboard or button on the Xbox controller is currently being pressed and translates these key presses and button presses from multiple controllers into a single set of travel and re-orientation instructions. These instructions are then decoded by the *MoveCamera()* function to edit the
camera’s position and orientation according to its type (see Sections 3.4.1 - 3.4.3).

Having a single `InterpretInput()` method to convert user input to a set of instructions which would be later used to move the camera, rather than having input directly affecting the camera’s position and orientation, enabled new input methods to be added at any point without creating extra camera classes or duplicated code to deal with the new inputs.

### 4.3 State Based Application

As previously mentioned in Section 3.3, the application developed for this project is made up of a number of scenes, such as the main menu, experiment selection menu, and a number of experiment task scenes. Each scene is implemented as a separate state of the application, an `AppState`, all of which are stored and managed by a single `AppStateManager`. An overview of this architecture can be seen in Figure 4.3.
4.3.1 ExperimentApplication

The *ExperimentApplication* class provides the main entry point for this application. The only tasks required of this class take place during the applications start-up. These tasks are to instantiate an *AppStateManager*, create all *AppState* needed by the application, pass them to the *AppStateManager* and to then start the main update loop.

4.3.2 AppStateManager

The *AppStateManager* operates as a state machine for this application, providing the typical push, pop, pause and resume functionality associated with one. The class contains a list of all *AppState* required for the current applications as well as a stack of all currently active states. This class also provides the function `findByName(string)` which returns a pointer to the *AppState* with the given name, allowing any *AppState* to be referenced by its name instead of its variable or memory address.

However, the most important function of this class is the `startLoop()` function, which creates the main game loop. This function sets the initial *AppState* as the current state and then loops until the application is shutdown. Within this loop this function captures and stores input from the keyboard and mouse, updates the currently displayed *AppState* and renders a single frame.

4.3.3 AppState

The *AppState* class inherits from a number of interfaces and abstract classes for input from the keyboard and mouse as well as GUI events. This is an abstract class that defines the functionality required by all states, including; `enter()` (which loads any resources when changing to this state), `exit()` (which unloads those resources when exiting this state) and `update()` (which takes the time passed since the previous update and updates the state).

This class also contains an *Ogre::SceneManager* which acts as a scene graph to store all objects within a scene as well as providing functions to manage these objects. Lastly, the *AppState* class provides functions to find a state by name and push or pop a state through the *AppStateManager* it belongs to.

Most *AppState* do not instantiate their member variables with their correct values
when they themselves are instantiated; instead they are only given their appropriate values when the state is entered. It is at this stage that any resources are loaded and variables correctly instantiated.

4.3.4 Menu States

This project contains a menu hierarchy where each node in the hierarchy is created as a separate class, inheriting from AppState. These classes implement their own enter(), exit() and update() functions according to the needs and requirements of each menu. However, as each menu state contains an empty scene graph with only 2D GUI components, no navigation technique is required and so a simple Ogre::Camera is used to render the empty scene.

4.3.5 GameState

GameState inherits directly from AppState and was designed to contain all functionality common to every experiment task state as mentioned in Section 3.3. This class implements enter() and exit() functions which set up and remove basic elements of a 3D scene and GUI components necessary to many or all experiment states (e.g. a progress bar) as well as taking note of the CPU time once everything is loaded.

This class also contains a BaseCamera pointer (instantiated by its inheriting classes), an instance of the collision detection class (CollisionTools) and a Timer to record how long a task took the user to complete. However this class is not designed to be instantiated due to the design of the update function described below.

The update() function implemented by GameState is designed to be used by all inheriting classes. This function updates the camera, and task variables of a particular experiment task (updateTaskVariables()). If the conditions required for the current task are met (areTaskConditionsMet()), the progress bar starts to fill. Once the progress bar is completely filled, a repetition counter in the ExperimentFramework is incremented and any final responsibilities required of the state at the end of a task are carried out (taskCompleted()) before changing to the next state. These three functions were created as pure virtual functions, allowing each experiment task state to use the same update function with their own task completion criteria and finishing responsibilities.
4.4 Experiment Task States

This section will cover the implementation of each of the task states which inherits from `GameState`. This consists mainly of resource loading when the state is entered and checks for task completion as the rest of their functionality is carried out by `GameState`.

**Free-Roaming States**

The three free-roam states all inherit directly from `GameState`. The purpose of these three classes is to allow the user time to practice the current input method with the chosen camera type. This is particularly important for user’s who have not used a particular input method or camera type before and who might be unaware of the possible ways to manipulate the camera. The `updateTaskVariables()` function does nothing in these three classes, and both `areTaskConditionsMet()` and `taskCompleted()` return false at all times so the user may practice for as long as they require.

Each class implements the most basic features required for each camera’s testing environments. For a first person camera this is a single ground plane with a neutral repeating tile texture applied to it as well as a darkly coloured skybox. The third person class also requires a single ground plane with repeating tile texture and a darkly coloured skybox. However this class also loads a simple player model to which a third person camera is attached. Lastly the orbit camera requires a dice model made of six separate planes and a single ground plane textured with a transparent grid pattern.

**taskCompleted()**

The `taskCompleted()` function is very similar in each of the remaining inheriting `GameState` classes, and so will only be covered once here. When the current task is deemed completed, the current CPU time is noted and the task start time is subtracted from this. A string containing this time and an identifier for the current task is sent to the `ExperimentFramework` to be logged. If the number of repetitions of the current task is above or equal to the value stored in the `ExperimentFramework` for the current task, the class queries the `ExperimentFramework` for the next `AppState`, otherwise the class is set to repeat.
First Person ‘Travel and Look At’

When this state is entered, a column and penguin model are added to the scene and their default red materials are applied to them. In addition to this, a first person camera is instantiated, a crosshairs overlay is created and the current time is noted. This class’s `updateTaskVariables()` function then checks for each subtask separately.

A transparent column was used to display to the user the target location as the check to determine whether the user is within a circular area is a very simple and efficient calculation. If the camera’s current position is identified as being within this column, a particular boolean is set to true and the material of the column is changed to green. If not, the boolean is kept as false and the column material is kept red.

Secondly the test to determine if the user is looking towards the penguin first checks whether the penguin’s axis aligned bounding box is visible to the camera. If the bounding box is within the screen the position of the penguin in 3D space is translated to screen co-ordinates. If this position is at the centre of the screen (with a threshold of 10% of the screen resolution in any direction) a separate boolean is set to true and the penguin material is altered. Otherwise the boolean is kept false and the penguin material is kept red.
remains a red colour (as shown in Figure 4.4a). A simple conjunction of the two mentioned booleans makes up the areTaskConditionsMet() function for this class.

**First Person ‘Collection’**

Upon entering this state, a number of gold sphere meshes are loaded and added to the scene and a first person camera is instantiated (shown in Figure 4.4b). Spheres were chosen for the collectable objects as the calculations for collisions against them are simple and fast to compute.

During each update this class’s updateTaskVariables() tests for collisions of the user with any remaining, visible collectables. This was originally done using the CollisionTools class to ray trace from the user’s previous positions to their new position. However when walking through an environment, we as humans expect to intersect with objects slightly to the side of our view as our bodies have width. Unfortunately the ray trace through the virtual environment will not identify these collisions which we expect to occur off the centre of the screen, as a ray is one dimensional and cannot identify these collisions at our side. Therefore, the collision detection checks were changed to sphere-sphere intersection tests. The radius and position of each collectable is checked against the user’s current position (with a radius equal to the near clip plane distance).

Once a collision is detected, a message containing the collided object’s name and the time passed since entering this state is sent to the ExperimentFramework to be logged and the collided object is then made invisible to the user. A sound effect is also played to inform the user that they have successfully collected an object. This ensures that the user is still aware of collisions which occur off-screen where they are unable to see the object disappear (e.g. moving backward into an object behind them).

The areTaskConditionsMet() function of this class checks if the number of objects collected is equal to the value stored in the ExperimentFramework for the current repetition of this class’s task. If all objects have been collected, this class completely fills the progress bar in a single update as there is no need to hold the current viewport steady (unlike with most of the other tasks).
Third Person ‘Travel and Look At’

Similarly to the first person version of this task, a transparent red column and red penguin model are added to the scene when this state is entered and the `updateTaskVariables()` function checks for each subtask independently. However, a third person camera is instantiated instead of a first person camera.

A very similar test is carried out to determine if the user is facing the penguin correctly through the crosshairs but in contrast to the first person travel task, it is the user’s character, not the user’s camera, which must be inside the highlighted area to satisfy this subtask (see Figure 4.4c). The same material changes are applied to each of the models when their respective subtasks are satisfied and the corresponding booleans set to true. Lastly, as in the first person version, the progress bar begins to fill up when both booleans are true.

Third Person ‘Collection’

As with its first person equivalent, gold spheres are added to the scene when entering this state. A third person camera is then instantiated instead of a first person camera (see Figure 4.4d).

In the third person version of this task, the user must direct the player to collide with the spheres, rather than colliding the camera with them. The bounding radius of the player entity and its current position are used to do the sphere-sphere intersection tests. Once a collision is detected, the same events which occur in the first person version are then carried out. Finally, once the number of objects collected is equal to the value stored in the `ExperimentFramework`, the progress bar is filled in a single update analogous to the first person collection task.

Orbit ‘Zoom’

When entering this state, six finite `Ogre::Planes` are loaded and a particular material applied to each, corresponding to each side of a dice. A transparent ground plane with a grid texture is also loaded and an orbit camera is instantiated. Lastly, a full screen overlay with the desired dice silhouette for the current repetition is created and given a red colour (as shown in Figure 4.5a).
To prevent expensive distance calculations, the offset of the camera from the world’s X-axis is used to denote the desired and current zoom of the camera. As zooming takes place on a single axis, and rotation and panning is locked during this task, this form of measurement works successfully and efficiently. The desired distance is stored in the *ExperimentFramework* and the camera’s current position on the X-axis is used as the current distance. The *updateTaskVariables()* function determines if these two values are equal (to within a threshold of 6.67% of the desired distance) and sets a particular boolean to true or false accordingly. If this boolean is true, the colour of the silhouette overlay is changed to green, otherwise the silhouette remains red. The *areTaskConditionsMet()* function of this class simply returns the value of the previously mentioned boolean.

**Orbit ‘Pan’**

When this state is entered, the same seven planes described in the previous section are created and an orbit camera is instantiated. Another full screen red silhouette overlay is created, however the same one is used for all repetitions of this task. The camera is then translated so the task begins offset from the world origin (see Figure 4.5b for an example of this).

To determine if the camera has been panned to the correct position, the screen space co-ordinates of the dice’s centre (i.e. the world origin) are calculated in the *updateTaskVariables()* function. If this position is in the centre of the screen, with a threshold of 2.5% of the screen resolution in any direction, another boolean is set to true. If this boolean is true, the material of the dice silhouette is altered to a green
colour, and if not, the silhouette stays a red colour. Similarly to the Orbit Zoom task class, the \texttt{areTaskConditionsMet()} function returns the value of the boolean mentioned above.

**Orbit ‘Rotate’**

In a similar manner as the previous two sections, the same seven planes are added to the scene and an orbit camera is instantiated. However this class creates a crosshairs overlay instead of a dice silhouette overlay.

During this class’s \texttt{updateTaskVariables()} the side of the dice currently being faced by the user is highlighted. In order to determine which side this is, a ray is traced from the centre of the screen into the scene in the direction the camera is currently facing. The nearest plane to intersect this ray is returned by the \texttt{CollisionTools} class. The number of this side is noted and its material changed accordingly. If the current side is equal to the desired side stored in \texttt{ExperimentFramework} for the current repetition of this task the plane is highlighted green and a new boolean is set to true, otherwise it is highlighted red and the boolean is false (as shown in Figure 4.5c). Once again the \texttt{areTaskConditionsMet()} function simply returns the value of this boolean.

### 4.5 FAAST

The Flexible Action and Articulated Skeleton Toolkit (FAAST) is middleware software which facilitates the integration of full body gesture control with virtual reality and video games \[^{72}\]. This software can be used with depth sensors compliant with either the OpenNI or Microsoft “Kinect for Windows” skeleton tracking software \[^{73}\].

FAAST was developed to aid the development of virtual reality applications using one of the depth sensors described above and to simplify the incorporation of gesture based controls in existing video games. They achieve this by providing access to the skeletal information provided by your chosen skeleton tracking software and an input emulator to generate virtual keyboard and mouse events.

Gestures can be described using relative positions of body joints, a short list of poses (leaning, twisting and jumping), angular constraints (of the elbows or knees) or velocity constraints of any joint. This project made extensive use of the FAAST
software to identify the poses used by experiment participants and to bind them to virtual key presses and mouse clicks.

The gestures used to navigate the 3D environments of the experiment are chosen by each participant individually. In order to facilitate the user’s choice of gestures, a number of predefined gestures were identified by the researcher for each of the controls in each navigation techniques (which will be explained further in Section 5.5). These predefined gestures are created as small XML snippets which, once chosen by the participant, were placed in a single XML file for a particular navigation technique. The FAAST application can then read in this file and identifies only the gestures placed in that XML file.

The process of placing these snippets in an XML file according to the participants’ choices during the experiment was time consuming but unavoidable for this project, as each participant required the thresholds of each gesture to be adjusted due to different body shapes and sizes.
Chapter 5

Analysis

The testbed application developed for this project was designed to evaluate the feasibility and suitability of gestures as an input method to performing complex tasks - specifically navigating 3D virtual environments. This chapter will first describe the testing procedure undertaken for each experiment session. The data obtained from the application and the participants will then be explained. Lastly, the methods taken to store and evaluate this data will be discussed.

5.1 Testing Procedure

Participants were recruited from around the researcher’s academic institution, through a combination of email-lists and individual discussions with colleagues and family friends. Before each test began, participants were given a brief description of the project as well as its goals, and were asked to sign a consent form giving the researcher permission to record their data anonymously (in the case of minors, the parent or guardian was asked to sign a separate consent form). Lastly, before the test could start, each participant was asked to fill out a short questionnaire.

At the start of each session, participants were given time to practice the current camera travel technique with the current input method in a free-roaming environment without any goals to satisfy. Each participant would then complete each camera’s tasks for each input method (a total of nine separate tests).

Unfortunately, the participants were required to be at different distances from the
display for different input methods. This disparity in distance results in a difference in the amount of a participant’s field of view the display occupies and this may have affected the participants’ sense of presence, enjoyment and overall opinion of the various input methods. In an attempt to counteract this effect, a projector was used to display the application at different sizes (i.e. a large display for the gesture controls, where the participant is forced to stand a significant distance from the display, and a smaller windowed display for the keyboard and gamepad controls where the participant is situated nearer to the display).

Between each camera’s tasks and the differing input methods, the participants were asked to fill out another short questionnaire pertaining specifically to the input method for the tasks they had just completed. Finally, once all nine tests were completed, an additional, more broadly focused questionnaire was filled out by each participant. Examples of these questionnaires can be found in Appendix [A].

5.2 Timed Results

During each experimental task, the application takes note of when an experiment state is entered and when it is completed. The difference in these times is saved to a log file as the time taken for the participant to complete a particular task.

Each experiment task was repeated five times before changing to the next task, or menu state, in order to obtain a more accurate result for each participant’s completion time through the use of a weighted average.

A small program was also developed to parse each user’s log files to identify only the relevant timing data (taking into account pauses and waiting times to ensure the user did not accidentally satisfy a particular task’s requirements). The times extracted from these log files could then be used to generate averages for each task and each input method.
5.3 Pre and Post Test Questionnaires

Pre-Test Questionnaire

The pre-test questionnaire was designed to obtain general profiling information about each participant, such as their age and sex, as well as information pertaining to their previous experiences with similar technologies, in an attempt to identify any biases they may have towards a particular input method or camera control technique. These included questions such as “How often do you play 3D computer games?”, “How often do you use 3D modelling suites?” and “How often do you use a gamepad control system?”. Participants were asked to pick one answer for each question on a four point rating scale (Never, Rarely (<10hrs a month), Sometimes (<10hrs a week) and Often (>10hrs a week)).

Post-Test Questionnaire

At the end of the experiment, each participant was asked to rate each of the three input control methods they had just used relative to each other (with values 1, 2 and 3, where 1 represents a better score). The input controls were rated on a number of different criteria, each of which affects a 3D UI’s effectiveness. These features were ease of use, ease of learning, ease of remembering, comfort or fatigue, a sense of presence and overall enjoyment and satisfaction. The purpose of this post-test questionnaire was to remove the ability for a participant to rate all three controls equally in all of the “Between Control Scheme Questionnaires” (see Section 5.4).

5.4 Between Control Scheme Questionnaires

Between each camera’s tasks and for each individual input method, participants were given a questionnaire to fill out. The purpose of this questionnaire was to determine the participant’s opinion of the current input method for the current camera travel technique, with respect to a variety of different features which affect a 3D UI’s effectiveness. The list of features used for the questionnaire were inspired by the work of Bowman in 3D UI and 3D travel technique evaluations [26, 23]. The list of features consists of overall enjoyment, ease of use, ease of learning, physical and mental fatigue,
response time, correct interpretations and a sense of presence. Participants were asked to rate the current input method with the current camera travel technique using a five point Likert scale in response to a given statement for each feature (where a value of 5 represents strongly agreeing with the given statement and a value 1 represents strongly disagreeing with the given statement).

Participants were also informed to not take into account the mapping chosen for the current input method (e.g. using the WASD keys instead of the four arrow keys or inverted axis rotations) and to answer with respect to the input method as a whole (i.e. the pressing of keys with precise mouse controls instead of holding a gamepad using mostly their thumbs, or the use of their whole body). They were asked to do this in an attempt to minimise any bias that the chosen mapping may have on a participant’s opinion of the current input method. It was felt that this was necessary as many games today (especially on the PC) allow the user to choose their own mappings and give the user the option to invert the camera axis rotations to a scheme that they are familiar with.

There is also a section of the questionnaire which gives participants the opportunity to provide any other opinions they may have about the current input method with the current camera travel technique as qualitative feedback.

5.5 Gesture Choices

The mappings chosen for the keyboard and gamepad controls were based on those commonly used in 3D games and 3D modelling suites for their respective camera travel techniques. However there does not yet exist a common set of full body gestures used for 3D navigation and, as a result of this, it was decided not to select a set of gestures for all participants to use, but to allow the participant to choose their own mappings for each camera travel technique. Allowing the participants to choose their own mappings would also provide data regarding which gestures are the most popular for the control of each camera type, and this data could be used as a guideline for further applications which plan to use gestures to navigate 3D environments.

Experiment participants were first given a choice of gestures to control each camera travel technique. The participant could then choose a gesture set immediately, or they could try all of the available options before choosing. Regardless of their choice,
participants were given time to practice with their chosen gestures in the free-roaming environment for each travel technique before starting the timed tasks.

Once the participant was happy with their chosen gestures, the participant’s choices were then noted by the researcher on the corresponding questionnaire. The available controls can be seen in the taxonomy in Figure 5.1 and the corresponding options for each control in Table 5.1.

**5.6 Data Storage**

All of the quantitative data obtained from each participant was placed in a single database, using a collection of unique user IDs. The records within the database were
Table 5.1: Gesture Options for Each Camera Control

<table>
<thead>
<tr>
<th>Fwd/Bkwd</th>
<th>Strafe</th>
<th>Yaw</th>
<th>Pitch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hands F/B</td>
<td>Hands out L/R</td>
<td>Twist</td>
<td>Lean F/B</td>
</tr>
<tr>
<td>Feet F/B</td>
<td>Feet L/R</td>
<td>Lean L/R</td>
<td>Left Hand U/D</td>
</tr>
<tr>
<td>Lean F/B</td>
<td>Lean L/R</td>
<td>Left Hand L/R</td>
<td>Right Hand U/D</td>
</tr>
<tr>
<td>Left Hand F/B</td>
<td>Left Hand L/R</td>
<td>Right Hand L/R</td>
<td>Right Hand L/R</td>
</tr>
<tr>
<td>Right Hand F/B</td>
<td>Right Hand L/R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Left Foot F/B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Right Foot F/B</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Zoom</th>
<th>Pan U/D</th>
<th>Pan L/R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lean F/B</td>
<td>Left Hand U/D</td>
<td>Lean L/R</td>
</tr>
<tr>
<td>Left Hand I/O</td>
<td>Right Hand U/D</td>
<td>Left Hand L/R</td>
</tr>
<tr>
<td>Right Hand I/O</td>
<td>Left Hand U/D</td>
<td>Right Hand L/R</td>
</tr>
<tr>
<td>Left Hand U/D</td>
<td>Right Hand U/D</td>
<td>Right Hand L/R</td>
</tr>
<tr>
<td>Right Hand U/D</td>
<td>Left Foot F/B</td>
<td></td>
</tr>
<tr>
<td>Right Foot U/D</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Then split into multiple tables, one for each of the sections listed above. These tables then used each user’s unique ID in order to associate the data from one table with the data from another table.

This database could then be queried to visually explore the data captured, and collate useful statistics such as; average times taken to complete a particular task, proportions of values given to each question of the “Between Control Scheme Questionnaires”, the average times taken by participants who played many computer games verse those who do not etc.
Chapter 6

Results and Evaluations

This chapter will discuss and evaluate the results obtained from this project’s tests. Firstly, a summary of the participants who took part in the experiments will be given. Next the results pertaining to the each of the features in Section 5.4 will be explained. Then an overview of some of the limitations of gesture based controls will be examined. Lastly some opinions given by participants will be discussed.

6.1 Participant Sample

A group of 18 participants (15 male, 3 female) took part in the experiments for this project. The average age of participants was 23.56 (the youngest being 15 while the eldest was 27 ). Each participant finished all of the tests for each input method and camera travel technique and completed all of the associated questionnaires. 5 of the participants played 3D computer games less than 10 hours a month while 9 of the participants played them more than 10 hours a week. Of the 18 participants, 12 used 3D modelling software less than 10 hours a month, while only 2 used some 3D modelling software more than 10 hours a week. All participants used a keyboard and mouse input method more than 10 hours a week, but only 6 of the participants used gamepad controls more than 10 hours per week.
Table 6.1: Count of Likert values given for Enjoyment

6.2 Features of Each Input Method

6.2.1 Enjoyment

Table 6.1 shows the number of times each input method was rated a particular value in response to the statement “I enjoyed using this control scheme”. As explained in Section 5.4 of this report, these results were extracted from the questionnaires based on a common Likert scale [74], where the participants could strongly agree, strongly disagree, agree, disagree or remain neutral in regards to a given statement (where strongly disagreeing with the statement results a value of 1 and strongly agreeing results in a value of 5).

The results seen here are distributed fairly evenly amongst the three control types, indicating that the participants did not find motion controls to be fundamentally more enjoyable than either the keyboard or gamepad input methods.

6.2.2 Interpretation and Response Time

Tables 6.2a and 6.2b display the number of times each input method was rated a particular value in response to the statements “The system correctly interpreted my controls” and “The system responded quickly to my controls” respectively. Here it is
clear that the gesture based controls did not compare very well against the keyboard and gamepad input methods (as indicated by the larger count of neutral, disagreeing or strongly disagreeing values). This however was to be expected as gesture based interfaces require extra time to identify the user’s pose and ensure that they are purposely using that pose (i.e. that they are not satisfying the pose conditions only momentarily as they move to another pose).

6.2.3 Mental and Physical Fatigue

Tables 6.3a and 6.3b display the number of times each input method was rated a particular value in response to the statements “The controls were physically tiring to use” and “The controls were mentally tiring to use” respectively. The keyboard and gamepad input methods were not considered tiring (i.e. disagreeing or strongly disagreeing with both of the above statements) by most participants, while the results for gestures were mixed across all values indicating that gestures were considered more tiring, both mentally and physically.

This was not entirely unexpected as full body gesture based controls inherently require more physical effort to use than either a keyboard or gamepad control scheme, since they require the user to be standing instead of seated (as is the case for the other two input methods).
However it was predicted at the start of this project that the gesture based controls would be a more natural and intuitive input method for users, resulting in less mental fatigue. However some of this extra mental fatigue may be attributed to the learning curve associated with a new input method. All participants who took part in this evaluation were very familiar with keyboard controls and most of the participants were at least slightly familiar with gamepad controls, and this familiarity would lower the mental fatigue acquired while using these input methods.

### 6.2.4 Presence

Table 6.4 shows the number of times each input method was rated a particular value in response to the statement “I felt a sense of presence in the 3D world (feeling of “being there” in the virtual environment)”.

These results indicate that motion controls can have a slight advantage over the other two input methods when it comes to aiding the users’ immersion in a 3D environment (as more participants strongly agreed with the relevant statement). The relatively small magnitude of this advantage was surprising however, as one of the key advantages often ascribed to gesture controls over other control types is the increased sense of presence granted by their natural mappings [40].
Table 6.4: Count of Likert values given for Sense of Presence

6.2.5 Ease of Use

Table 6.5a shows the number of times each input method was rated a particular value in response to the statement “The controls were easy to use”. It is clear from the table that gestures were not considered easy to use by most participants, shown by the number of times participants did not agree with the above statement for gesture-based controls (corresponding to values of 1, 2, or 3). This may be due to several factors, including the choices given to users for each control, the capabilities of the Kinect to identify the user’s pose, the capabilities of the FAAST toolkit to identify the user’s gestures, or an inherent weakness of gestures to map to certain tasks (see Section 7.1 for further information).

6.2.6 Ease of Learning

Table 6.5b shows the number of times each input method was rated a particular value in response to the statement “The controls were easy to remember”. Most participants agreed with this statement for all three input methods, despite having a much larger familiarity with both keyboard controls and gamepad controls. This may indicate that the gestures were easy for participants to learn and remember even though most did
6.3 Fine-Tuned Controls

Most of the participants displayed a significant amount of frustration while using the gesture based controls. This frustration was most obvious when more fine-tuned controls were needed to place the camera in a specific position or orientation. This was especially prevalent during the ‘orbit camera’ tests. Participants could quite easily get the dice near the target location in both the zooming and panning tasks but they would overshoot the target position multiple times before getting it perfectly aligned. A similar effect was noticeable during the ‘travel and look at’ tasks, where participants could easily make their way to the highlighted area but aligning the camera with the penguin proved more difficult and they would overshoot the target direction multiple times before completing the task.

The effect of this lack of fine-tuned control can be seen in the average times taken to complete each of the experiment tasks with each of the input methods (see Table 6.6). All values for gestures were significantly higher than either the keyboard or gamepad input methods (which both scored comparatively equally in all tasks). The time taken to complete each task with gestures ranged between 3 and 6 times slower than the keyboard and gamepad times.
Table 6.6: Average times taken for each task using each input method

A small amount of the extra time taken when using gestures can be attributed to the slower response times associated with gesture based interfaces. However observations made by the researcher during the experiments can attribute much of this time to the time spent fine-tuning the position or orientation of the camera to the desired values when using gesture controls.

6.4 Overlapping Gestures

During the tests which used gestures as the input method, the gesture recognition system would sometimes misinterpret the controls or intentions of the participants. This misinterpretation of a user’s gestures and poses is a common problem for gesture based interfaces, and while their resolution and tracking algorithms will undoubtedly improve as the technology matures, the issue of body parts occluding each other will remain.

One of the larger issues with gesture controls identified during this project’s experiments is the non-discrete nature of gestures themselves, which lead to the misinterpretation of a user’s intentions. For example, one of the participants chose to use their hands to move the camera forward, backward, left and right and used leaning to alter
the camera’s pitch. However when they leaned forward, their shoulders moved forward as well, placing their hands behind their shoulders. This was interpreted by the system as moving backwards (and changing the pitch) when the intention of the participant was only to look downwards.

This non-discrete nature of gesture controls allows for different gestures to interfere with each other, confusing the gesture recognition system (and the user). This may create a number of difficulties for applications requiring more than a small handful of input commands; difficulties almost never encountered by keyboard and gamepad control systems as they use a combination of discrete input controls.

### 6.5 Effect on Learning

Table 6.7 shows the average time taken in each of the camera travel techniques, with each input method, separated into two groups within each graph. These groups represent the users who play 3D computer games less than 10 hours a month (on the left) and more than 10 hours a week (on the right). From these tables we can see that those who played more 3D computer games achieved better times using all three input con-
controls in all three travel techniques and the ratios of time taken for each input method between groups are very similar.

If gestures were indeed a more natural and intuitive input method, the ratio of task completion times for gesture controls between groups should be reversed, or at least nearer to a 1:1 ratio. However they were not, providing no indication that the Kinect gestures were quicker to master than those using either the keyboard or gamepad, nor do they provide any indication that the Kinect eases the learning experience of new users (despite participants’ roughly equal opinions of all three input methods (Section 6.2.6)).

### 6.6 Post-Test Opinions

Table 6.8 shows the average vote given to each input method by participants in each of the six categories described in Section 5.3 after they have completed all nine tests (a lower vote indicating a better score). Most of the scores here conform with participants opinions given in Section 6.2.

The Kinect was rated as less comfortable as well as harder to learn, remember and use than either the keyboard or gamepad input methods. This coincides with the fact
that the time taken to complete each task using gestures was also considerably higher than the keyboard or gamepad (see Table 6.6).

Despite the participants’ preference for keyboard or gamepad input methods in ease of learning, ease of use, ease of remembering and comfort, gestures still managed to score well in overall enjoyment and achieved the highest average scores for presence (i.e. gestures gave them the greatest sense of presence). However the enjoyment score may be due to the novelty of a new input method. The keyboard and gamepad input methods scored approximately equivalently in their timed tasks, however the gamepad input method was slightly favoured by participants at the end of the experiments (as seen by the lower scores in Table 6.8).

6.7 Natural Mappings

One of the advantages associated with gesture based user interfaces is that they theoretically provide more natural mappings (i.e. interaction methods similar to those used in the real world) when compared with other control interfaces such as a keyboard or a gamepad. This allows users to more easily access their own mental models of the particular system’s interaction techniques as they are closer to the actions they would expect to use in the real world.

While the Kinect can provide more natural mappings for certain actions, such as swiping to move an object across the screen (and even more direct mappings such as jumping to make your avatar jump), it still creates a number of restrictions to navigating a 3D world. The most obvious of these is the spatial restrictions applied to the user. When using the Kinect, or any visual gesture recognition system with a fixed position, the user must be positioned within the camera’s field of view in order for it to identify the user’s gestures. This causes actions requiring traversal to be mapped to less natural gestures (such as running on the spot, or abstractly holding out a leg to move forward).

This problem also extends to the rotations of the user. If a user has access to a 360 degree display of the virtual world, for example from a HMD or CAVE (cave automatic virtual environment) set up, the most natural mapping would be to rotate their entire body to change their orientation. However a fixed visual gesture recognition system, such as the Kinect, is unable to identify whether the user is facing toward or away from
the camera, and as a result will identity the user’s left side as the right and their right side as the left if they are orientated more than 90 degrees from a front-on view to the camera.

Therefore, in reality, many of the supposed advantages that the Kinect (as a gesture based input system) provides as a natural and intuitive input method are removed for 3D navigational tasks.

6.8 Feedback to User

As previously mentioned, many of the experiment participants could quite easily get near the target location or orientation for a task but the lack of fine-tuned control caused them to overshoot or undershoot the target multiple times before completing the task when using gestures. This occurrence was much less prevalent when participants were using the keyboard or gamepad input methods.

The tendency for users to overshoot their target can also be attributed to the lack of feedback given to the user (in conjunction with the lack of fine-tuned control). The keyboard and gamepad input methods provide haptic feedback to the user through button presses and resistive forces but the only feedback provided to participants when using gestures was the change in viewpoint when the appropriate gesture had been identified.

Unfortunately, there is also an innate delay between the user’s visual identification of a moving screen and the execution of an action to move their body \cite{75}. The lack of any haptic feedback given to the user to indicate when a gesture has been identified and the slower response time associated with gesture recognition (see Section \ref{sec:6.2.2}) further add to this delay. By the time the user has shifted their pose to prevent the current gesture from being identified again in the following frames they have often overshot their target.

6.9 User Opinions of Gestures Based Navigation

One of the participants stated that they found gestures to be “fun”, however they claimed that this fun was due to “the novelty and ridiculousness of them” and that
“doing a real task using gestures would not be fun at all”. This sentiment was also mirrored by a number of other participants. Another participant believed that the Kinect should not be used to move around a virtual world as it “doesn’t add anything to the experience” and that “it would be better suited to supporting actions” such as picking up or selecting a particular item in a scene.

Many of the participants also commented on how annoying it was when they put their hand down at the end of a task and this action was detected as looking or panning downwards which moved the camera outside of the accepted positions or orientations to complete the task, forcing them to recomplete the task.

A lot of participants also pointed out that the speed at which the camera moved and turned were not proportional when using gestures. They often mentioned that this made it “confusing” or “annoying” when trying to rotate while travelling forward. This stems from the inability to scale the speed at which the user wishes to travel or rotate in a particular direction when using gestures (note: a slower rotation speed was chosen to make the “look at” subtask possible to complete). However the mouse and analog sticks provide the user with this ability to alter the rotation and travel speed of the camera, when using the keyboard and gamepad input methods. Many participants mentioned also that the gestures were quite “sensitive” in that the camera or character would suddenly start to move although the participant only moved their body a small amount.

These issues are partially a result of the implementation of the gesture recognition system used for this project. The chosen system, FAAST, does not allow gestures to be described or identified with a magnitude. For example, the users who chose to turn their body to rotate the camera to the left and right all expected the camera to turn faster if they turned their body more to the side. However the toolkit only allows a gesture to be described using a single angle and only identifies ‘if’ the user is turned more than that angle, not ‘how much’ more.

The effect of identifying the extent to which a gesture is performed may be approximated by describing a number of gestures using a different angle for each. However each of these gestures must then be mapped to separate keyboard events, which would then also need to be identified by each camera type by adding every extra key press into the InterpretInput() function of every camera type.
6.10 Preferred Set of Gestures

Due to the number of possible gestures available to control various elements of each camera type, shown in Table 5.1, there were a large number of possible combinations that could form a set of gesture controls for each camera. This number was far larger than the number of participants that took part in this project’s experiments, and as a result of this no clear pattern of preferred gesture sets emerged.
Chapter 7

Conclusions

This project’s goal was to explore the feasibility of using gesture based controls for complex tasks and interactions, such as those found in the navigation of 3D virtual environments. In order to do this, a 3D testbed application was implemented in which gesture controls could be evaluated and compared with existing control methods, such as the keyboard and gamepad.

The results of this study highlighted a number of potential limitations associated with the use of visual gesture controls for complex tasks, relating to the areas of fine-tuned control, haptic feedback, overlapping gestures and a general lack of natural mappings due to the technology’s spatial restrictions.

7.1 Using Gestures for Complex Tasks

Many participants of the study had trouble completing the various experiment tasks when using gestures, but not when using either of the other two input methods. This was primarily due to the lack of fine-tuned control available when using the Kinect for full body gesture recognition as it can’t accurately identify small motions [18]. At a resolution of 320x240 (the resolution required for skeletal tracking with the Kinect) roughly 40% of the depth camera pixel values actually contain IR depth data while the other 60% are merely guesses or approximations [76]. Given this resolution, and possible errors in the depth data, it is easy to see how the skeletal tracking could not accurately identify small movements in the user’s joints.
The ability for gestures to overlap and interfere with each other also created problems for some participants, especially when using gestures which changed the axes of their upper body, such as leaning or twisting, as they expected their gestures to be identified using their body’s new axes. Unfortunately, the gesture recognition system used with this project only identified gestures according to the Kinect’s view plane.

For example, the relative position “behind”, which is used in the logic of a number of gesture types, corresponds to a greater depth value. So the gesture “right hand behind right shoulder” checks if the right hand has a greater depth value than the right shoulder, not whether the right hand is “behind” the plane created by the user’s torso as they would expect. This resulted in misinterpreted controls, which served to confuse and frustrate many of the experiment participants.

Due to the restrictions on a user’s movement when using a visual gesture recognition system, a gesture based 3D navigational interface still requires various movements to be mapped to unnatural gestures, rather than those used in the real world. For example, user’s cannot simply “walk” forward to move an avatar in a 3D VE as they would quickly leave the Kinect’s view and most likely collide with the display. Even if these “unnatural” mappings are no less mentally tiresome than those used with keyboard or gamepad controls, users are still likely to prefer the later input methods as they produce less fatigue and generally require less physical exertion.

The lack of any haptic or direct visual feedback given to the participants was also a likely contribution to the poor performance of gestures in task completion times, as well as negatively affecting the participants’ opinions of gestures as an input method. Without feedback, participants could not know how close they were to the correct pose for a given gesture, or how their attempts at one gesture could be identified as another. While this information could be presented visually to a user, this would draw their attention away from the virtual world and their presence within it.

### 7.2 Limitations

#### 7.2.1 Sample Size and Diversity

The number of participants who took part in this study would have preferably been larger. While some of the data collected converged to give indicative results, some data
queries required too many variables both within and between subjects and the resulting group sizes became quite small. For example, when attempting to correlate popular gesture choices with the effect of previous experience with certain input methods the sample size proved too small to yield conclusive results. While the data in these groups may still indicate a particular result, the group sizes are too small to give truly conclusive results.

Secondly, the variety of participants was also quite poor. Most of the participants were males in their mid to early twenties with much 3D gaming experience with both the keyboard and gamepad input methods. Ideally, the sample of participants would have included more users with little or no 3D gaming experience as well as users from a more diverse age bracket.

7.2.2 Binary Gesture Identification

The implementation of the gesture recognition for this project may have affected some of the participant’s opinions. The implementation could identify whether a particular gesture was recognized or not, but could not identify the extent to which it was recognized. This resulted in jerky movements in the application environment when using gestures, as the user could not scale the speed of their travel or rotation.

One approach to rectify this would be to identify the extent or magnitude to which the user is satisfying the current gesture and scale the movement and rotation speeds accordingly. Another possible approach to artificially creating more fine-tuned control with the chosen gesture recognition system would be to scale down the movements of the user as they approach their target. However this approach requires there to be a target which the user is currently aiming for, which defeats the purpose of providing the user with the freedom to explore a 3D world off-rails.

7.2.3 Experiment Design

The design of this project’s application was that of a bare testbed environment in order to prevent distractions to the participants. This also served to reduce the development time required to make the testbed application. However a richer and more game like environment may have provided slightly different results due to an increased sense of overall presence and immersion.
The character used with this project’s third person camera was a simple player model made entirely of spheres, which prevented the character from being meaningfully animated. The lack of animations or an identifiable character could also have had a negative effect on the participants’ experience within the virtual environment.

7.2.4 User Choice of Gestures

It was hoped that by letting users chose their own set of gestures to navigate the 3D environment, the most popular and successful gestures could be identified. However by giving participants this choice, they were exposed to a very wide number of possible controls and then expected to quickly choose one particular set for use in a timed task. This may have caused some confusion for participants, as not only were the actions required to carry out a gesture a new form of interaction, but the particular set of gestures chosen was also new and remembering which they had chosen presented extra difficulties in using gestures to control the camera or character for certain participants.

An additional problem presented by allowing the participants to choose their own set of gestures, were the number of combinations that ended up containing interfering gestures. These sets prevented users from achieving better results in the timed tasks and most likely negatively affected their opinions of the gesture based input method as a whole.

7.3 Future Work

7.3.1 Larger, More Varied Sample Size

A larger and more varied sample of participants would help to provide more conclusive evidence as to the suitability of gestures to navigating 3D virtual environments. It could also give better insight into the effects that previous experiences with other input methods and navigation techniques have on both the user’s opinions of, and performance with, gesture based controls. A larger and more varied sample of participants would also provide better indications of the effect that the different gesture choices could have on the user’s performance and enjoyment.
7.3.2 Separate Gesture Set Study

Another possible area for future work would be to perform a separate study aimed at identifying the ideal gesture set (or sets) for each of the 3D travel techniques explored in this report. This knowledge could then be used to re-evaluate and further examine the suitability of gesture based control to navigate 3D VEs.

7.3.3 Tailored Gesture Recognition System

One of the most obvious areas of future work for this project would be to create a gesture recognition system better suited to the specific requirements of navigating 3D VEs identified in this report. Ideally a full gesture recognition system would have been written from scratch to suit the needs of this project’s application; however the time constraints on this project prevented this as a possibility.

This system would include the ability to continuously scale the user’s travel and rotation speeds. For example, the user would rotate faster, the further they turn their body to the side, or move forward faster by placing their foot further in front of their body. This would help to provide more fine-tuned control to the user.

7.3.4 Streamlined Gesture Selection

In order to save time during each experiment session, the selection of gesture choices could be further streamlined. Currently, the process of creating a unique set of gestures tailored to the needs of a particular participant requires a significant amount of direct input from the researcher (the XML file needs to be directly edited and recompiled each time a participant wishes to use a different gesture). Creating a drop down list of options within the application for the control of each camera type would save a significant amount of time in setting up the gesture controls for each participant.

However this approach would require the system to automatically scale the thresholds for each gesture to suit each user (i.e. the angles to which the user must rotate or the distance from their shoulder the user must place their hand for the gesture to be identified). This would involve accounting for the differences in body shapes and sizes, similar to the dynamically generated detection values used in D’Souza’s paper [57].
7.3.5 Full Body Verses Hands Only Gesture Control

Another potential study to be done in the future would be to look specifically at the comparisons between a set of full body gestures and those which use only the user’s hands to navigate a 3D virtual environment (e.g. Leap Motion technology[77], or the Kinect for Windows when used in “near mode”[78]). By using two forms of gesture control this may identify what effect using one’s entire body has on the performance or sense of presence within a virtual world, when compared to using only one’s hands. Given that this project indicates that the act of navigating 3D VEs reduces the amount of natural mappings available to full body gesture control, hand based gesture control may prove preferred due to the reduced amount of physical exertion needed.
Appendix A

Questionnaires

Pre-Test Questionnaire

<table>
<thead>
<tr>
<th></th>
<th>Never</th>
<th>Rarely (&lt;10hrs month)</th>
<th>Sometimes (&lt;10hrs week)</th>
<th>Often (&gt;10hrs week)</th>
</tr>
</thead>
<tbody>
<tr>
<td>How often do you play 3D computer games?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How often do you use 3D modeling suites? (e.g. 3DS Max, Blender)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How often do you use a Keyboard and Mouse control system</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How often do you use a Gamepad control system (e.g. Xbox controller, Joystick)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Post-Test Questionnaire

Please rate the controls in order of preference (1 to 3) according to the factors on the left

<table>
<thead>
<tr>
<th></th>
<th>Keyboard &amp; Mouse</th>
<th>Games Controller</th>
<th>Kinect Gestures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of use (where 1 is easiest to use)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ease of learning (where 1 is easiest to learn)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Confusion (where 1 is easiest to remember the controls)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enjoyment/Satisfaction (where 1 is most enjoyable)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comfort/Fatigue (where 1 is least tiring)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sense of presence (where 1 has the most presence) (feeling of “being there” in the virtual environment)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Between Control Scheme Questionnaire

*To be filled in by Evaluator:*

<table>
<thead>
<tr>
<th>Control Scheme</th>
<th>Gesture</th>
<th>Camera Type:</th>
<th>First</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward/Back</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strafe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yaw</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pitch</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*To be filled in by Participant:*

How much do you agree with the following statements for the task you’ve just completed (select one per statement)?

<table>
<thead>
<tr>
<th>Statement</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Undecided</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>I enjoyed using this control scheme</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to remember</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were physically tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were mentally tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system correctly interpreted my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system responded quickly to my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I felt a sense of presence in the 3D world (feeling of “being there” in the virtual environment)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any extra comments about the control scheme:

______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
# Between Control Scheme Questionnaire

**To be filled in by Evaluator:**

<table>
<thead>
<tr>
<th>Control Scheme</th>
<th>Gesture</th>
<th>Camera Type:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Third</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gesture</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward/Back</td>
<td></td>
</tr>
<tr>
<td>Strafe</td>
<td></td>
</tr>
<tr>
<td>Yaw</td>
<td></td>
</tr>
<tr>
<td>Pitch</td>
<td></td>
</tr>
<tr>
<td>Zoom</td>
<td></td>
</tr>
</tbody>
</table>

**To be filled in by Participant:**

How much do you agree with the following statements for the task you’ve just completed (select one per statement)?

<table>
<thead>
<tr>
<th>I enjoyed using this control scheme</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Undecided</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>The controls were easy to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to remember</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were physically tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were mentally tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system correctly interpreted my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system responded quickly to my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I felt a sense of presence in the 3D world (feeling of “being there” in the virtual environment)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any extra comments about the control scheme:

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________
**Between Control Scheme Questionnaire**

*To be filled in by Evaluator:*

<table>
<thead>
<tr>
<th>Control Scheme Gesture</th>
<th>Camera Type: Inspection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pan Up/Down</td>
<td></td>
</tr>
<tr>
<td>Pan Left/Right</td>
<td></td>
</tr>
<tr>
<td>Yaw</td>
<td></td>
</tr>
<tr>
<td>Pitch</td>
<td></td>
</tr>
<tr>
<td>Zoom</td>
<td></td>
</tr>
</tbody>
</table>

*To be filled in by Participant:*

How much do you agree with the following statements for the task you’ve just completed (select one per statement)?

<table>
<thead>
<tr>
<th>Statement</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Undecided</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>I enjoyed using this control scheme</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to remember</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were physically tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were mentally tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system correctly interpreted my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system responded quickly to my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I felt a sense of presence in the 3D world (feeling of “being there” in the virtual environment)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any extra comments about the control scheme:

______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
### Between Control Scheme Questionnaire

*To be filled in by Evaluator:*

<table>
<thead>
<tr>
<th>Control Scheme</th>
<th>Keyboard &amp; Mouse</th>
<th>Camera Type:</th>
<th>First</th>
</tr>
</thead>
</table>

*To be filled in by Participant:*

How much do you agree with the following statements for the task you’ve just completed (select one per statement)?

<table>
<thead>
<tr>
<th>Statement</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Undecided</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>I enjoyed using this control scheme</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to remember</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were physically tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were mentally tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system correctly interpreted my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system responded quickly to my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I felt a sense of presence in the 3D world (feeling of &quot;being there&quot; in the virtual environment)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any extra comments about the control scheme:__________________________________________
______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
___________________________________________________________
# Between Control Scheme Questionnaire

*To be filled in by Evaluator:*

<table>
<thead>
<tr>
<th>Control Scheme</th>
<th>Keyboard &amp; Mouse</th>
<th>Camera Type:</th>
<th>Third</th>
</tr>
</thead>
</table>

*To be filled in by Participant:*

How much do you agree with the following statements for the task you’ve just completed (select one per statement)?

<table>
<thead>
<tr>
<th>Statement</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Undecided</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>I enjoyed using this control scheme</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to remember</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were physically tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were mentally tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system correctly interpreted my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system responded quickly to my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I felt a sense of presence in the 3D world (feeling of “being there” in the virtual environment)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any extra comments about the control scheme: __________________________________________
______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
# Between Control Scheme Questionnaire

**To be filled in by Evaluator:**

<table>
<thead>
<tr>
<th>Control Scheme</th>
<th>Keyboard &amp; Mouse</th>
<th>Camera Type:</th>
<th>Inspection</th>
</tr>
</thead>
</table>

**To be filled in by Participant:**

How much do you agree with the following statements for the task you’ve just completed (select one per statement)?

<table>
<thead>
<tr>
<th>Statement</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Undecided</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>I enjoyed using this control scheme</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to remember</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were physically tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were mentally tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system correctly interpreted my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system responded quickly to my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I felt a sense of presence in the 3D world (feeling of “being there” in the virtual environment)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any extra comments about the control scheme: ___________________________________________________________
_______________________________________________________________________________________________
_______________________________________________________________________________________________
_______________________________________________________________________________________________
_______________________________________________________________________________________________
_______________________________________________________________________________________________
_______________________________________________________________________________________________
_______________________________________________________________________________________________
_______________________________________________________________________________________________

_______________________________________________________________________________________________

_______________________________________________________________________________________________
# Between Control Scheme Questionnaire

*To be filled in by Evaluator:*

<table>
<thead>
<tr>
<th>Control Scheme</th>
<th>GamePad</th>
<th>Camera Type:</th>
<th>First</th>
</tr>
</thead>
</table>

*To be filled in by Participant:*

How much do you agree with the following statements for the task you’ve just completed (select one per statement)?

<table>
<thead>
<tr>
<th>Statement</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Undecided</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>I enjoyed using this control scheme</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to remember</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were physically tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were mentally tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system correctly interpreted my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system responded quickly to my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I felt a sense of presence in the 3D world (feeling of “being there” in the virtual environment)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any extra comments about the control scheme:______________________________

______________________________________________________________________________

______________________________________________________________________________

______________________________________________________________________________

______________________________________________________________________________

______________________________________________________________________________

______________________________________________________________________________
# Between Control Scheme Questionnaire

*To be filled in by Evaluator:*

<table>
<thead>
<tr>
<th>Control Scheme</th>
<th>GamePad</th>
<th>Camera Type:</th>
<th>Third</th>
</tr>
</thead>
</table>

*To be filled in by Participant:*

How much do you agree with the following statements for the task you’ve just completed (select one per statement)?

<table>
<thead>
<tr>
<th>Statement</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Undecided</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>I enjoyed using this control scheme</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to remember</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were physically tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were mentally tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system correctly interpreted my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system responded quickly to my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I felt a sense of presence in the 3D world (feeling of “being there” in the virtual environment)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any extra comments about the control scheme:________________________________________
______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
Between Control Scheme Questionnaire

To be filled in by Evaluator:

Control Scheme | GamePad | Camera Type: | Inspection

To be filled in by Participant:
How much do you agree with the following statements for the task you’ve just completed (select one per statement)?

<table>
<thead>
<tr>
<th>Statement</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Undecided</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>I enjoyed using this control scheme</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were easy to remember</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were physically tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The controls were mentally tiring to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system correctly interpreted my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The system responded quickly to my controls</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I felt a sense of presence in the 3D world (feeling of “being there” in the virtual environment)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any extra comments about the control scheme: ____________________________________________
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
Bibliography


[57] L. D’Souza, I. Pathirana, D. McMeel, and R. Amor, “Kinect to architecture,”


