VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Navigating 3D Worlds via 2D Multi-Touch Interfaces.

Daniel Cope

Supervisor: Stuart Marshall

Submitted in partial fulfilment of the requirements for
Bachelor of Software Engineering.

Abstract

The navigation of 3D worlds is difficult on muti-touch devices due to the limitation of the 2D displays, making movement along all six degrees of freedom hard to achieve. As multi-touch devices become more common the want for an intuitive means to do this becomes greater. This paper presents three gesture styles which meet these criteria. The gestures provide a common interface for navigation on 2D touch displays which scale well to any size of device.

Acknowledgments

I would like to acknowledge a number of key people who help with the completion of this project. Firstly, Stuart Marshall, my supervisor, who made the entire project possible and supported my through out the year. Thanks to Craig Anslow, for all his advice and technical support. I wish him the best of luck with the rest of his PhD. Also thank you to the HCI group as a whole, for there advice and allowing me to demo my prototypes at their meetings.

I want to thank all my test participants, without them this project would have been far less successful, you know who you are. Finally, I would like to thank all my fellow students for their camaraderie, shared support and collaboration to see everyone through their projects.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As multi-touch devices become more powerful and widespread the types of tasks users desire from them become more complex. One such area of difficulty is navigating 3D worlds. The limitations of a 2D touch surface mean that providing movement for all six degrees of freedom becomes a challenge. Existing solutions often restrict movement along particular axes or provide gestures which only make sense in the context of their program. No standardized set of gestures exist for 3D navigation yet exist.

Even in the non-touch world, 3D navigation is still a complex task. Graphic modeling programs, such as Blender or Autodesk Maya, require an array of key strokes to perform simple navigation tasks. While these programs perform functions other than navigation, it still highlights the complexities of remaining in control while navigating through a 3D world.

Another example is space or flight simulators. They typically use physical controls to mimic the control set of real world vehicles. The goal in these simulations is to use the controls to navigate a craft through a 3D world. The physical controls are designed to help conceptualize how the movement of each control relates to the movement of the craft in the world.

A flight simulators navigation could not be considered pure. It still relies on the limitations of physics and other factors to decided how the navigation will perform. These limitations unbalance the effectiveness of each movement, such as tilting a plane up and down affect its speed. While most of these effects are undesired for pure 3D navigation, they may help a user remain in control and on course.

This leaves two problems to address, how to best navigate a 3D world and what is the best interface for this on a 2D touch screen. To simplify the problem the navigation task are broken into subsets, depending on the implementation. This means each subset can be mapped to different controls, allowing bias towards using the simpler controls for the more frequently used tasks.

1.1 Contributions

This paper presents three sets of gestures which can each be used for manipulating the camera in a 3D scene. These are the indirect, static and dynamic gesture sets. Each gesture set aimed to be intuitive, robust and easy to use.

The indirect gestures mimic a physical interface by providing on-screen touch sensitive controls. These controls can each be manipulated separately, but force orientation and specific positions for each gesture on the user.

The static and dynamic controls provide the user with more freedom, as they can be performed anywhere on the touch surface. The static controls directly manipulate the movement of the camera while the dynamic controls manipulate the velocity of each movement.

Each gesture was refined through a peer review process and pilot tested before a larger user study was conducted. Twelve users participated in the study, testing how well each gesture performed when navigating along a series of paths. A survey of the participants was also conducted so feedback could be received on the much prior experience they had and how the felt each gesture performed.

Main Contributions The following are the main contributions for the project:

- Designed three gesture sets for using 2D touch surfaces to navigate 3D worlds.
- Designed and implemented a prototype to demonstrate and test the three gesture sets.
- Conducted a user study with twelve participants to determine which gesture set was the most intuitive and easiest to use.

1.2 Results

Due to the small group of participants, complex statistical analysis could not be performed. The quantitative data that was collected showed the indirect gesture are the most consistent at producing a faster time for each path. The dynamic gestures also had a tighter set of values, but was slower overall. The static covered a broader range of times for the more complex paths, possibly indicating some users struggled with this gesture set more.

The qualitative data shows that most users found the indirect gesture set performed the best for fine-grained and navigational control. The dynamic gesture set was favored for navigational control, but had mixed results for fine-grained navigation. The static gesture were also well received but went seen as performing on the same level as the indirect.

The rest of this paper is structured as thus; background and related works are discussed in the following chapter, the specific goals and problems of the project are talked about in chapter 3, the design of each gesture is discussed in chapter 4, the implementation of both the system and the software are explained in chapter 5. Finally, the evaluation and results are looked at in chapter 6, conclusions are drawn in chapter 7.

Chapter 2

Background and Related Works

The main focus of the project has been on navigating 3D worlds via 2D touch interfaces, however it is important to remember that both 2D interfaces and programs with 3D navigation have existed for over a decade.

A computer desktop environment is a perfect example of where a 2D interface has been successful. It is obvious that anyone who has used a computer in the last ten years would have come into contact with this style of interface. It is important to consider this when designing what the user would be expecting to see on an interface, regardless of other features the surface might have.

Navigating and viewing 3D worlds would be a concept familiar with most users, as a number of computing related areas fall into this category such as, computer gaming or graphics modelling. While there is no standardized way to navigate or control these types of programs, it is common for the full range of motion to be limited in some way.

A full motion is considered to include the six degrees of freedom. These are translation along the X, Y and Z axis and rotation around the X, Y and Z axis. Most programs recognize the difficulties of provide all of these motions and put constraints on there range of movement to make navigation as a whole easier to manage. This limitations commonly include restricting the user to a single plane of movement or enforcing realistic physics.

Similar to what this project is trying to achieve in terms of navigation are programs like flight or space simulators or graphic modelling tools such as Blender [2] or Autodesk Maya. Navigating in these types of programs is still a daunting task, even when using a mouse and keyboard.

2.1 Background

Knowledge of a number of other projects is required to understand the concepts put forward in the paper.

Multi-Touch for Java. Mutli-Touch for Java, or MT4J, is an open source framework for the development of touch sensitive applications. These applications run on a variety of hardware and operating systems. The framework provides high level functionality such as gesture event handling and a basic tool-kit for developing both 2D and 3D feature rich applications. MT4J applications also handle listening for touch tracking information, often sent via the TUIO protocol.

A paper authored by Laufs, Ruff, Zibuschka [10], discusses the motivations for developing the MT4J framework. The paper goes into detail about the architecture of the framework and the different layers of abstraction. The functionality of the framework is subdivided into different layers. The layers provide flexibility with how each input is received and processed. Events can then be filtered down to the presentation layer where the user defined components reside.

Multi-touch table design. The touch table used in this project uses diffuse illumination to detect touch points on a glass surface. Infrared lights are shone from below the surface, through a diffuser material. The diffuser acts as a fliter for light shining back through the surface. When an object is placed on the table it reflects more infrared light back through the surface than the diffuser. A camera is then used to detect and record the difference [13, 16, 12]. Figure 2.1 illustrates this process.

The nuigroup provide information on a series of alternate technologies that could have been used in place of diffuse illumination [14]. They discuss the pro and cons of the different table design. As the table for this project was prebuilt, using an alternate technology was difficult and unneccssary.

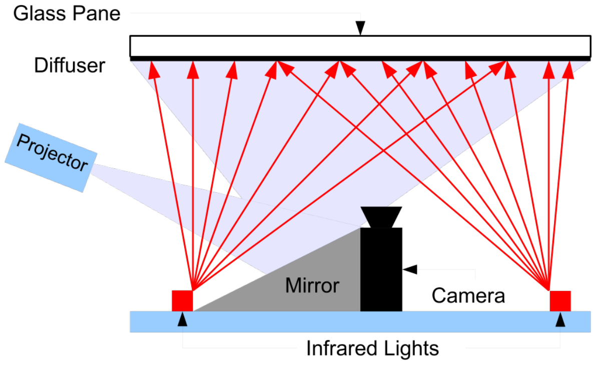

Figure 2.1: An example of a touch table using diffuse illumination.

Community Core Vision. Community Core Vision (CCV) is a free software package which interfaces with the camera in the multi-touch table. The images which are received are analyzed for blobs, shapes created from the reflection of objects on the table surface. These blobs are the tagged and tracked as they move around the table. CCV can then broadcast the tracking information using either the TUIO protocol, flash xml, or binary top. CCB is currently maintained by the nuigroup [15].

TUIO Protocol. TUIO is a simple protocol specifically designed to meet the requirements of table-top tangible user interfaces. The protocol has support for both finger tracking and token objects, which are distinctive shapes which can be mapped to a controller [8]. Originally TUIO was designed for the reacTable project, however missing features prompted the extension of the protocol outside of a single project. TUIO has now become widely used, particularly in the open source community [7].

2.2 Related Works

A number of other projects cover similar areas to this project.

2.2.1 Alternate Multitouch Frameworks

Kammer et al [3] present an overview of nine free multi-touch frameworks currently available. They provide a list of criteria to classify how each framework works including; which platforms a framework supports, the frameworks event systems, support for tangible objects, how multiple touches are handled, how gestures are processed, how the framework allows gestures to be extended and if the framework provides visualization support. This paper covers MT4J as well as the frameworks discussed below.

PyMT Python Multi-Touch, or PyMT, is a cross platform framework for the Python language. PyMT is very like MT4J in that it provides a toool-kit which lets developers quikly prototype and experiment with interaction techniques, as well as build feature rich applications. PyMT supports a range of input including TUIO and mouse devices, it processes each of these into the events which it dispatches to the root component. The root component then decides whether to handle the event or dispatch it down the stack [6].

LibTISCH Echtler and Klinker present LibTISCH, a multi-touch software architecture [4]. It aims to be a generic multi-touch framework to link different input hardware on one hand to different graphical tool-kits on the other. Rather than supporting a protocol such as TUIO it interface with the hardware its self, using abstraction layers to process the input into touch events.

Microsoft Surface SDK Microsoft Surface SDK is a combine software and hardware package. The system handles both the processing of touch events and the output, coming preinstalled with a range of applications. Development can be done directly on the device, using the Windows Presentation Foundation [1]. The Surface offers the most comprehensive touch solution but falls well outside the scope of this project.

2.2.2 Similiar Projects

Perhaps the most similar piece work is by Fu, Goh and Allen [5], who investigate efficient exploration of large-scale astrophysical simulations, using touch gestures. To deal with the problem of navigating large spatial spaces they introduce six gestures for navigation.

They use a tapping gesture for selection and de-selection of a star to show information about that star. They apply the 'rolling ball' mechanism to rotate around a preset point via a single touch and drag. They use the two finger gesture of moving the fingers closer or further apart to control camera zoom. They allow for rotating around any star by first selecting the star via the tap gesture, then placing a finger over that star and dragging a second finger across the surface to produce a 'rolling ball' motion which rotates the camera.

Given the large spatial scale their data covers the introduce a power-of-10 ladder which is displayed when two touches are located vertical of one another, a third touch can then be used to select the power-of-10 which can be dragged back and forth on the ladder to scale the simulation.

They also implement a many-finger gestures where the user swipes across the screen with all fingers to control the pan and tilt of the camera. A many-fingered twisting motion will rotate the camera.

Unlike this project, Fu, Goh and Allen's project is aimed at a touch board for use in public spaces and planetarium installations. Their implementations are specialized for use with astrophysical data with large spatial spaces. This project is more aimed at the general navigation of a 3D scene, where all the objects are located in the immediate area.

Another project, conducted by Reisman, Davidson and Han [11], looks at the direct manipulation of 2D and 3D objects via a touch surface. They developed a four degree of freedom manipulator. This restricts the degrees of freedom that can be used at any one time based on the current gesture.

A one finger touch allows translation along the X and Y axis. A two finger touch enables full 3D translation and rotation about the Z axis. They extend this to six degrees of freedom by adding a three finger gesture, which enables rotation about the X and Y axis. From these initial gestures they discovered a number of intuitive gestures which could be used to manipulate the object.

The primary difference between the work presented in this paper and Reisman, Davidson and Han's work is that they focus on manipulating 3D objects.

This project is focused on navigating around 3D objects. Their gesture control system offers a novel approach to this, which could translate to orientating the viewpoint rather than an object. They likewise target a touch table as their primary device.

Kim et al [9] produced a short paper that looks at using an iPhone or iPod Touch as a way of controlling a virtual environment system. They use a gesture called finger-walking-in-plane (FWIP), which works like human legs on a treadmill. The faster you do the gesture the faster you move in the virtual environment. They also implemented a rotation-in-place, where the user places one stationary touch and then a second touch is dragged to the left or right.

Kim's work discusses using repeated gestures to continue motion and the limitations of this, such as the users fingers going of the edge of the touch surface. The static gesture set, a solution discussed in this paper, explores a similar idea.

Chapter 3

Problem and Solution

This chapter looks at the problems with 2D touch gestures and 3D navigate and proposes three gestures sets that can be used to combined these two areas. The design constraints placed on the project and the methodology followed is also discussed.

3.1 Original Problem

This project covered two distinct areas of interaction, 2D touch gestures and navigation through a 3D scene. Each area had specific problems which had to be addressed. The key then was to devise a way to effectively combine both 2D gestures and 3D navigation. The outcome of project was expected to stack the 2D gestures on top of the 3D navigation, providing a touch interface for the 3D world. It is important to note that the goal was to manipulate the camera in the 3D scene and not the objects themselves.

Multi-touch surfaces come in a variety of sizes and implementations. The quality of each touch surface can be a major factor in how well gestures will perform. Outside interference on the touch surface or confusion with detecting the points can cause unexpected behavior with the gestures. This could of either be dealt with at the gesture level, only providing gestures that were resistant to interference; or at the software level, by building in corrective measures.

While this project only focused on developing gestures for a large touch table, it was important to consider how well the gestures could scale to different screen sizes. Smaller screen size would limit the space the gestures had to perform in. This mean any gesture movements would have to be responsive in this space and still have a reasonable effect on the program. Likewise, larger screen sizes would mean that relatively smaller movements would still have to be effective, not requiring the user to strain there hands or arms.

Not all touch devices are created equal, some devices support as little as 2 touch points. While a gesture set should make use of more touch points when available, it was important for the core gestures to be performed with one to two touch points at most. It is likely that this would feel more natural to the user, because they would only need to associate each arm with a touch point rather than multiple fingers.

One of the goals of this project was to provide full 3D navigation, using all six degrees of freedom. While each movement is easy to implement by itself, the combination of all six prove to be more complicated. In particular, human beings are used to traveling upright through the world. This still applies when they travel through a 3D space. Users can quickly become disorientated when they rotate to quickly in one direction, as everything associated with the natural upwards direction will be in relatively different places. Features to combat this are important in any implementation of a 3D world where movement and navigation are allowed.

The final problem to overcome was combining the 2D gestures onto the 3D world. Due to the variability in screen sizes, the maximum amount of space should be used to display the 3D world. Any visual aides for the gestures should be discrete and not obscure any parts of the screen. This also meant the visual aides should avoid being obscured by the hands themselves. Proper use of translucence is important here as it also helps to signify that the element is separate from the 3D scene.

The gestures themselves should make it obvious that user is manipulating the camera and not any of the objects in the scene itself. If the user was required to interact with any on-screen controls, it must be obvious that they are part of the navigation controls. The gestures would also map to the effects occurring on-screen where ever possible.

3.1.1 Gesture Criteria

These problems lead to a series of criteria that each gesture set would have to meet:

- Be intuitive. The motion of each gesture should indicate the effect the gesture will have.
- Be robust. The gestures should be easy to correct if the wrong gesture is performed.
- Be simple. The gestures should not require complex multi-touch movements, one or two touch points are best.
- Scale well to large or smaller displays. This includes the effects of the gestures as well as the movements required for the gestures.
- Be resistant to interference. This can be achieved by limiting the speed of movement or recognizing and ignoring erratic movements.

The 3D world had to be sufficiently complex to demonstrate and test each gesture. It also had to provide a means to regain bearings if the user become disorientated.

3.2 Proposed Solution

This paper proposes three sets of gestures to overcome these problems, the indirect, static and dynamic gesture sets. The indirect gestures are a set of onscreen controls, which mimic a physical interface already used for 3D interaction such as a game pad. The static gestures are pure gesture movements which can be performed anywhere on the screen. The dynamic gestures are the same gesture movements as the static controls but effect the velocity of the camera rather than just the movement.

Indirect Gestures. The indirect gestures could be considered intuitive, as they are based on physical controls which have been around for many decades. To those that aren't familiar with the physical controls, labels can be included to indicate what each control does. The labels also make the gestures simple to use, as does the ability to split the controls apart based on functionality. The localized interaction also make the controls harder to interfere with as the user will be aware to what is going on around each control.

The indirect gestures can not be considered to be robust, it would not be as obvious what gesture caused which movement and would be hard for the user to reset back to the original position. The control would also have trouble scaling, particularly to small screen devices where the controls would have to be redesigned to fit on the screen.

Static Gestures. The static gesture set is mostly intuitive, actions such as zooming and rotating are performed instinctively. Both the panning/tilting and strafing/lifting actions would be performed with the same motion, causing problems with which the users expects to be performed in which way. The gestures do however require no more than two touch points meaning they are simple to use. This can cause issues around which two points to use if many are active, that makes these gestures susceptible to interference.

A rotation circle are the first touch point makes the gestures more robust, particularly when performing individual gestures. The visual aide also indicates the original distance from the first and second finger, allowing correction if the user zooms in or out to far. The aide would scale well to both smaller and larger screens as it appears smaller when the touch points are closer together. Each gesture is also easily repeatable is limited screen space was prevent the full range of motion.

Dynamic Gestures. The dynamic gestures are much like the static, only differing in the effect each gesture has on the camera. Instead of moving the camera in steps over the course of the gesture, the dynamic gesture increase the velocity of the movement as the gesture is performed. This means smaller more control movements can be performed for a better effect. This scales very well to both larger and smaller devices as the user is only required to move there hands as much as they are comfortable. The movement of the gesture will continue if they leave there hands stationary on touch surface.

The dynamic gestures have indicators for the points of the first and second touch points. This are mainly to indicate where the movement is being calculated from, but also make the gestures more robust. The user is quickly able to return the fingers to their original positions, where they can then perform gesture in the opposite direction to return to their original bearing.

The design of each gesture set is discussed in the next chapter, chapter 4.

3.3 Design Methodology

Over the course of this project a two week iterative development cycle was followed. The complexity of each phase increased as progress was made in the project. The early stages dealt with conceptual ideas, before moving into early prototypes of individual gestures and then onto to the combined gestures presented in this paper. This saw the development of the indirect gestures first, followed by the static gesture which were modified to become the dynamic gestures.

At the end of each two week phase the latest prototype was presented at Human Computer Interaction (HCI) group meetings. The group members were all experienced with user interface design and many had worked on touch screen prototypes themselves. This group session acted as a peer review process for each iteration of the prototype, highlighting different parts of the prototype that needed to be improved in the next phased. Members of this group also acted as participants in the pilot studies for the prototype.

The budget for provided for this project was small, however only $100 needed to used. This was spent on two $50 vouchers to be award to the test participants; one for the fastest test completion time and the other for a random participant. As the touch table was already available, built by a PhD student, and the software for operating it was free, no extra costs were incurred.

3.4 Design Constraints

A number of design constraints were inherited from the systems available for this project. As discussed in chapter 2, section 2.1; the touch table uses diffused illumination to detect objects on the table surfaces. This makes the table more susceptible to hover interference, where an unexpected part of the hand or piece of clothing is picked up by the sensor despite not being in direct contact with the surface. The granularity of the detection can be tweaked to be more sensitive, however this makes it more difficult for smaller targets to picked up accurately.

Another limitation of the table comes from the camera. The camera records images at between 30 and 60 frames per second. This can vary based on the power of the computer being used. The low sample rate makes it difficult for the CCV system to track fast moving touch points. In most instances the points will seen as multiple points, as i the user had raised and replaced their finger. This makes gestures such as flicking more difficult to perform and can causes multiple smaller gesture to be performed in its place.

Chapter 4

Gesture Design

This project saw the design and implementation of three gesture sets. They aim to be intuitive, easy to use and provide all six degrees of freedom.

This chapter looks at these three gesture systems that were designed for the project; Indirect gestures in section 4.1, static gestures in section 4.2 and finally dynamic gestures in section 4.3.

4.1 Indirect Gesture Set

The indirect gesture set consists of onscreen controls which are touch sensitive, allowing them to be individually manipulated by the user. Two different control types were developed; a joystick like control which allows the user to move the stick along the X and Y axis within the bounds of a box. Also a rotatable dial control which can be rotated about a central point.

The arrangement of these controls are shown in Figure 4.1. The left and right sides are a mirror of each other, with a joystick control on the outside and a dial on the inside. The effect of each control is different however, the left joystick controls panning and tilting, the left dial controls camera roll, the right dial controls camera zoom and the right joystick controls lifting and panning.

The indirect gesture set aims to mimic physical controls from flight or space flight simulators by emulating them on the touch display. These simulators provide the most realistic approach to navigating 3D worlds that already exists. These simulations usually require the user to manipulate a set of physical controls to navigate. This is usually with a specific piece of hardware such as a joystick or game-pad. The idea is that a user who is familiar with the physical controls would be able to map this knowledge to the touch controls with ease. This gesture set has been called indirect as the user is not directly manipulating the camera but instead using the touch controls to indirectly control the camera.

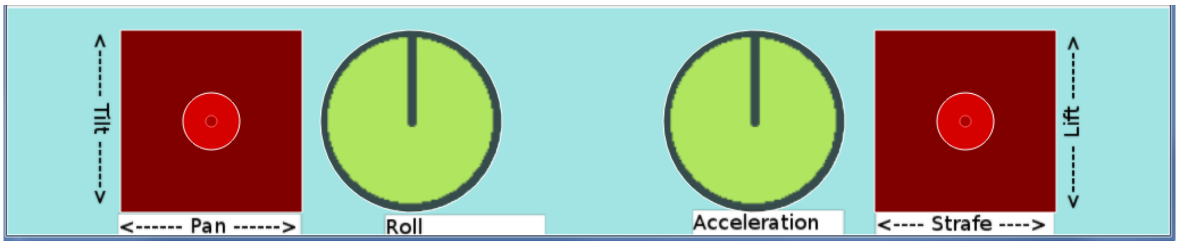

Figure 4.1: The control interface for the indirect gesture set.

4.1.1 Joystick Control

The leftmost joystick control deals with the tilt (looking up and down) and panning (looking left and right). The rightmost joystick control deals with the strafing (moving directly left and right) and the lifting (moving directly up and down).

It was decided that the joystick controlling pan and tilt should be on the opposite side to the control which handles the zooming as early pilot tests showed these were the two favoured controls. By placing them on the opposing left and right side it allowed the user to more easily manipulate them both at the same time.

Each of the joystick controls consists of three components, the stick, the dead-zone and the border.

The stick is presented as a translucent red circle and is the only active part of the joystick control. The user can select the stick with a touch and drag the stick around, within the confines of it's border. The further the joystick is moved from the center of the control, the more velocity that movement will gain. If the touch point is dragged beyond the confines of the box the stick will remain on the edge, at the point where the touch point left the box area.

Originally the stick would retain is position when the touch point was released, however this coupled with a small dead-zone made it very difficult to completely stop movements in any direction. This made manipulation of the camera more tedious as a constant slow drift in each direction would occur after use. To prevent this the stick is now immediately returned to the center when the touch point is released, ceasing movement coming from that control.

The dead-zone is represented by the small black dot in the middle of the control. The size of this dot represents the distance that the joystick could move from the center of the control without the movement actually manipulating the camera in any way. The purpose of this is twofold, to prevent 'jitter' if a touch point has locked onto the joystick in the center and to make it easier for a user to return the control to a neutral position.

The border is represented by the dark red square around the outside of the control. The size of the square represents the maximum distance the joystick can be moved from the center of the control. This is important to give a scale to the range of motion of the joystick and to better simulate the limitations of its physical counter part.

Originally a circle was used as the border for the stick, however it was quickly discovered that this limited the amount of movement possible simultaneously along the X and Y axis. Instead, a square shape is used for the border as the corner of the squares allow the full range of motion of X and Y at the same time.

4.1.2 Dial Control

The left dial control deals with camera rolling or rotation around the cameras center point.Turning this dial to the left or right causes left or right rolling of the camera respectively. The right dial controls the forward and backwards acceleration of the camera. Turing this dial to the right will accelerate the camera forward and vice veras for turning to the left. Each dial control consists of just one part, the dial.

The dial is presented as a green circle with an indicator mark. When a touch point is placed on the circle the mark will rotate to align with the touch point. The further to the left or right of the initial top point the mark gets, the faster the action is performed.

Variations on how the control should perform when it rotates greater than 180 degrees were tested. Allowing the user to continuing to increase acceleration by rotating the dial continually caused confusion when trying to slow down and reverse. Stopping the dial at the halfway point also causes problems as the dial would no longer track to the users finger properly.

With the current implementation if the dial is rotate past the half-way point then the direction of movement is reversed as if rotated backwards to that point. This effectively splits each side of the control to do different things.

4.1.3 Control Variations

Originally the joystick control was stacked on top of the dial control to make one single control on each side. The joystick was scaled down slightly to allow contact with the dial. The reason was to allow all controls on each side to easily fit under a hand. It was design so the joystick could be manipulated with a finger while the dial controlled with the thumb.

After much review with a peer group it was found that this made the dial a less obvious control, only being discovered by accident. Indeed many users believed it to be the limit for the stick on the joystick control. Rather than redesign the control, they were simply repositioned to make each more obvious.

It would be worthwhile in future work to look at different controls that could be used in place of the dials, such as sliders. This would certainly make more sense as a velocity control.

4.2 Static Gesture Set

The static and dynamic gesture sets are quite different from the indirect gesture controls. They allow the user to directly manipulate the camera rather than using a set of controls. This provides a more direct link between what the user is doing at how it is effecting the camera. It is our belief that this could offer a more intuitive control over the camera.

It is important to note that all gestures result in the *camera* being manipulated and not the objects in the world. It becomes clearer in the following implementation chapter, but it was communicated to the user that they should imagine they are dragging the center cross hair around, rather than any object in the world.

Pan and Tilt Gestures. A single point drag gesture is the simplest type of gesture that can be implemented. Figure 4.2 shows an example of this gesture. The dashed circle represents the initial touch point for the finger. The arrow indicates the direction of movement the finger is dragged to the current position of the touch point, displayed as a solid circle.

This gesture would result in the camera panning towards the right. Due to its simplicity, a decision had to be made around which movement should be controlled by this gesture. The strafing and lifting gestures were initially implemented using the single drag however pilot tests showed users more frequently required the use of pan and tilt controls.

To control the remaining movements two fingers gestures are used. As multiple gestures can be manipulated by using two fingers at the same time, a system to avoid confusion was developed. When starting a two finger gesture the first touch point acts as a base and the seconded finger is used to execute the movements. The first finger should only be moved if executing a strafe or lift gesture. There are four gestures that can be performed using two fingers, zooming, strafing, lifting and rotating. These cover the remaining ranges of motion.

Zooming Gesture. The zoom gesture controls movement forward and backwards of the screen. It is performed by dragging the second finger towards the first. This will move the camera towards objects in front of the screen.

Figure 4.3 shows an example of the zooming gesture. The first finger is numbered one and remains stationary. The second finger is numbered two and starts to the right of the first as shown by the dashed circle. The arrow then shows the second touch point's direction of movement toward the first point. To move backwards away from objects on the screen the opposite gesture is perform, by moving the second finger away from the first.

The zooming gesture is inspired by the two finger zooming gesture common to 2D touch applications. However, the gesture has been added in reverse to emphasize that the camera is being manipulated and not the world itself.

Rotate Gesture. The rotate gesture controls the rotation around the center of the screen to create a rolling camera effect. The gesture is performed by dragging the second finger in an arc around the first finger. Figure 4.4 shows an example of the rotation gesture. The first fingers remains stationary at position one. The second finger is placed to the right and dragged down around to the left. This would result in the camera rolling to the right.

Strafe and Lift Gestures. The strafing and lifting gestures are performed in a similar manner. These gestures control movement to the left and right of the screen. Figure 4.5 shows an example of a strafing gesture. Both the first and second fingers are dragged to the right. This would result in the camera moving to the right as well. If both fingers are moved above or below their initial positions the lifting gesture is performed instead. These gestures are the two finger equivalents of the drag gesture.

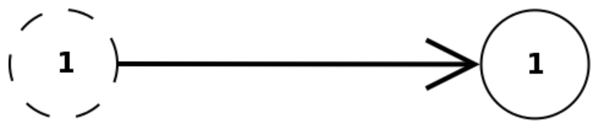

Figure 4.2: A single finger drag motion. Only one finger is used, the dashed circle represents the initial touch point. The arrow shows the direction of the movement as the finger is dragged to its new position, denoted by the solid circle. This would result in the camera panning to the right.

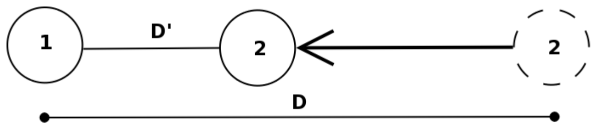

Figure 4.3: A two finger zooming gesture. The first finger is represented by the solid circle labeled one. The second finger starts at the dashed circle and is moved in towards the first finger, stopping at the solid circle labeled two. The distance traveled is based on the scale ratio, calculated by taking D / D′. This gesture will result in the camera moving forward into the screen.

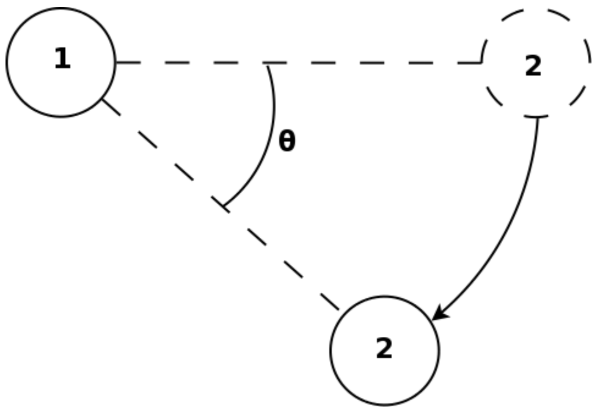

Figure 4.4: A two finger rotate gesture. The first finger is placed and remains stationary, denoted by the solid circle labeled one. The second finger moves in an arc around the first finger, creating a change in the angle. The angle is calculated in steps, not necessarily based on the initial position of the second finger. The dashed circle represents the position of finger two at the start of the step and the solid circle represents the finger at the end of the step.

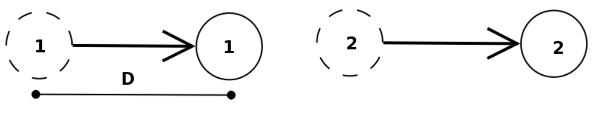

Figure 4.5: A two finger strafing gesture. Both the first and second fingers are placed on the table and then dragged to the right. Like with the rotate gesture in Figure 4.4 the distance is calculated in steps. The dashed circle represents the starting touch points, moving in the direction of the arrow to their respective solid circles. The distance is calculated using the minimum distance D from a fingers start point to its end point. This gesture results in the camera strafing to the right.

4.3 Dynamic Gesture Set

The dynamic gesture set uses the same gestures as the static gesture set, with one major difference. The static gestures set uses the distance the touch point has moved in steps to determine the movement of the camera. The dynamic gesture set instead uses the distance from the initial touch point to the current position of the finger. This causes the gesture to change the velocity of the cameras movement, rather than just translate its position. In particular, this means the cameras motion will continue even if the users finger is stationary on the table.

Pan and Tilt Gestures. Figure 4.6 shows an example of the altered gesture, although the touch point does not move any further from the initial position the camera will continue to pan to the right. A green circle reminds the user where the touch point originated from, as the distance from this point is used to calculate the velocity of the panning movement.

Zoom Gesture. The two finger gestures have been altered in a similar way, the gestures remain the same but velocity is calculated rather than a stepping distance. Also, a red circle is used to signify the initial point of the second finger. In the case of the zoom gesture the distance between the first and second point is still used, however it is now used to calculate the velocity of the movement. Figure 4.7 shows the updated gesture as compared to the earlier Figure 4.3.

Rotate Gesture. The rotate gesture is performed in the same manner but now a velocity is calculated based on the difference of the rotation angle. The initial angle is calculated from the first and second fingers initial points, the red and green markers. A second angle is calculated based on the current positions of the first and second fingers. The difference in these two angles is used to set the velocity for the gesture. Figure 4.8 illustrates this point.

Strafe and Lift Gestures. The strafing and lifting gestures are also performed in the same way, but now the velocity is calculated based on the minimum distance either of the two touch points have moved from their respective initial points as shown in Figure 4.9. This has been implemented to allow better control when using multiple gestures at the same time. For example, if the second finger is far from its initial point then the velocity of the strafe or lift can be controlled by moving the first finger away from its initial point.

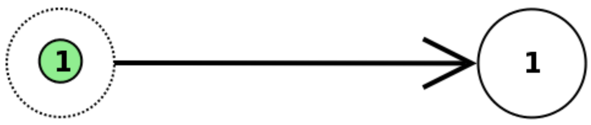

Figure 4.6: A single finger dynamic drag gesture. Unlike Figure 4.2 the drag gesture is not calculated in steps but always from the initial point represented by the dashed circle. This point is displayed to the user as a green circle. The velocity of the movement is calculated based on the distance from this initial point to the current position.

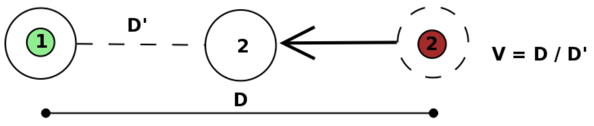

Figure 4.7: A two finger zoom gesture. This differs from Figure 4.3 by using the scale ratio to calculate velocity rather than stepping distance. This allows the user to drag the second finger then stop, but have the camera continue its motion forward.

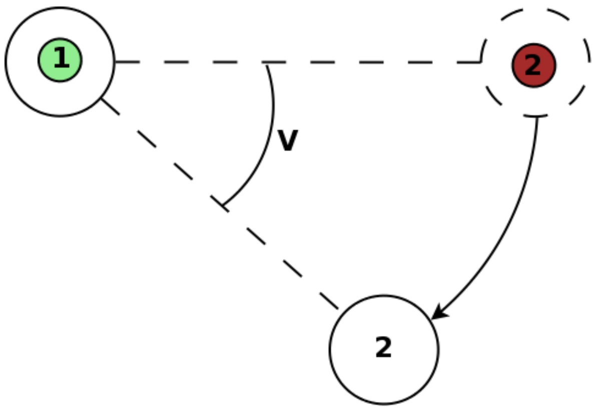

Figure 4.8: A two finger dynamic rotate gesture. Rather than using steps like the static gesture in Figure 4.4 the angle of rotation is calculated by the difference between the angles between the green and red points and the angle between the current position of the first and second finger. This difference is used to calculate the velocity of the movement. N.B. It is still a valid gesture if the first finger has moved from it's initial position.

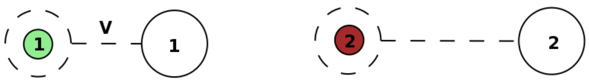

Figure 4.9: A two finger dynamic strafing gesture. This gesture works like the static strafe in Figure 4.5 but instead uses the minimum distance V to calculate the velocity of the movement.

Chapter 5

Implementation

The next section describes the physical and system setup of the touch table required so that the prototype application is able to recieve touch information.

5.1 System Setup

It is important to understand some of the limitations put on the gestures by how the touch table is implemented and the finer points of the tables use. The touch tables uses infrared light to detect objects on or close to the tables surface. Infrared light is shone up from below the screen through a diffuser material. Objects near the surface then reflect the light back, becoming brighter than the diffuser material. This difference can be picked up by the camera.

To use this system, the camera is connected to a computer running software that can analyze the images and turn them into touch points. This project used a free software package called Community Core Vision (CCV) which takes the live video from the camera and processes it to produce 'blobs' which can then be tracked as touch points. The locations and other information is then sent from CCV to the prototype program using the common TUIO protocol. Figure 5.1 illustrates this process.

CCV requires calibration to work with the available screens, this is performed from directly within CCV. For this project the touch table was used as a primary screen and the laptop's main screen as a secondary monitor. This allowed CCV to be calibrated for the touch screen and to run the prototype while the secondary screen could be used to monitor the experiments.

Unfortunately, the touch table design has a number flaws. Unexpected touch points will often occur if the user's palm or a piece of clothing is too close to the screen. These will interfere with any gesture being performed, possibly confusing the user. While it is possible to tweak the settings for CCV, this normally leads to ignoring the smaller blob sizes and make it more difficult for an object to be picked up.

This correlates with another issue, where users with smaller fingers have a hard time registering a touch point on the table. This is usually where the blob size is too small to be registered. The simplest solution is to use multiple fingers to try and increase the size of each blob. This is the trade off that was encourage to users for this project.

A better solution is to tweak the CCV settings for each individual user. This would lead to the best user experience, however a means to quickly calibrate new settings when a user switches will also have to be included. This highlights an inherit problem with the diffuse illumination touch table, where the controls have to be reset for every user rather than anyone being able to pick up and play and expect the same performance.

For more on the touch table design, CCV and TUIO see chapter 2, section 2.1.

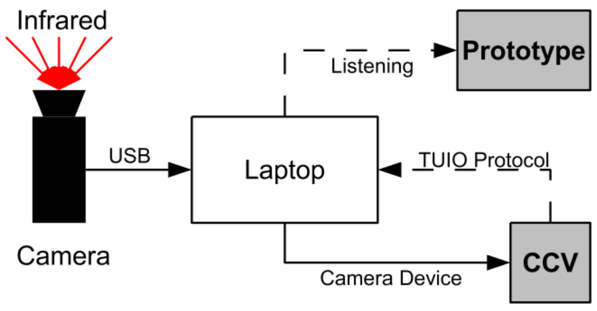

Figure 5.1: The system chain between the touch table and prototype.

5.2 Framework Selection

The prototype was developed in Java using the Multi-Touch for Java (MT4J) framework. The Java language was chosen in part due to the familiarity with the language, but was primarily dictated from the chosen framework. MT4J was favored for a number of reasons; in particular it had the best support providing extensive online documentation, tutorials and example.

As MT4J is an open source project the source code is also provided with every release, this enabled bugs to fix locally and provided a better understanding of how everything worked under the hood. All of the MT4J demo projects are also included with the source. This helped to speed up initial development as an existing program could be used to prototype different features of the final program.

MT4J was also familiar to other members of the Human Computer Interaction (HCI) group within Victoria University.They were able to help with common problems and share their fixes. MT4J also provides a graphics library inside the framework, which has been specially design to allow objects to respond to touch events. The objects can either be rendered with OpenGL or D3D.

Kammer et al [3] provides a good review of other multi-touch frameworks, including MT4J. Chapter 2 discussed MT4J in section 2.1 and alternate frameworks in section 2.2.1.

5.2.1 Framework Limitations

During the implementation of the prototype MT4J was found to have a number of limitations and bugs, particularly around the scene camera. Perhaps the most severe problem was with the way the camera is implemented. The camera use two points to determine its position and orientation. The first point is the current position of the camera and the second point is used for the camera to look at. This causes many problems with rotation.

The second camera point was not made to be used as a means for moving the cameras rotation, it was only meant to allow the camera to pick a point to face. This meant it did not respect rotation in all directions and would flip the camera if rotated too far. This was, in a sense, to always keep the camera ‘upright’. The solution to this was to set the second point at an appropriate distance and then rotate everything thing else in the world around the first camera point. This would make it appear that the camera was moving itself.

A second problem with the camera was to do with the clipping distance. Objects in front of the camera would begin to disappear as the object was approached, rather than appear to move past the camera. Adding an offset to the cameras position alleviated some of the problem.

5.3 Software Design

Programs using MT4J are made by extending *MTApplication* and adding an *AbstractScence* to the implementation. The implementation of the scene can then be used to load objects into the world and set how they respond to touch events.

The prototype was designed to be extensible beyond the three planned gesture sets. To achieve this a template design pattern was used for the gesture controls. Each gesture set was created as an 'Environment' which implemented the *AbstractTestEnviroment*. The methods which setup the gestures controls are defined in the each child environment and then added to scene by the abstract environment.

The *AbstractTestEnviroment* used layers to manage all the visual elements. The background and scene layers were built by the abstract environment, the controller layer was handled in the environment child. If the test variable was set a fourth testing scene replaced the scene layer. The test scene was created by the test suite class.

This design fitted nicely with the iterative design methodology discussed in chapter 3, section 3.3. It allowed for each gesture to be implemented one at a time and still have all scene changes propagated into the new and old environments.

5.4 Interface Design

By default every MT4J component has scale, rotate and drag gestures preregistered into their event handlers. As for this project it was undesirable to manipulate objects individually, there were two options to avoid this. The simplest was to group the objects in an component group and then turn off picking for the top level object. This meant objects in the group would not respond to any touch points placed on their surface.

The alternative was to remove all event handlers from object after initialization. This can be a tedious process when many components are in use, however it is the only way to have custom gesture events work with a component. This alternative was only used where component were required to be touch sensitive.

Each scene was broken down into a series of layers. The top layer contained a special overlay component and was used for setting up the components that each gesture required. The overlay is a special object that will always be displayed in front of the camera, drawing over any objects behind it. The middle scene layer contains all of the objects which make up the world to be navigated through. When an evaluation is occurring this middle layer is replaced by the test path layer by the test suite.

Early in the design a background layer was also included. It was discovered that a bug in the background components was causing the program to slow down and stutter. For these reasons the background was removed.

The indirect or controller interface uses a number of components to build a 2D control set. Two different types of composite components were built, the joystick and the dial control. The joystick consists of a dead-zone, a border and the stick itself, which is the only component that is touch sensitive. The dial control only uses one component, the dial, which is also touch sensitive. The use of these controls are described in the previous section, chapter 4 section 4.1.

Both the dynamic and static gestures add a transparent layer to the overlay. This layer is used to register the custom touch events, which can be performed anywhere on the screen.

While both gesture sets have four distinct gestures, this has only been implemented using only two touch events, drag and scale.

Originally the rotate event was also being used but a bug, relating to the background components, meant the rotation event would only get picked up when hitting a component outside of the overlay. Instead new rotation code was built into the scale gesture, which now deals with zooming, strafing, lifting and rotation. The drag gesture is left to deal with the panning and tilting.

5.5 Gesture Implementation

The computation of each gesture are handled in different ways. Two sets of coordinates are used, the world and touch coordincates. The world coordinates refer to the 3D vector positions in the scene itself. The touch coordinates refer to the 2D vectors of the touch screen.

The static drag gesture calculates the change in distance on both the X and Y touch axes using the 'to' and 'from' positions provided by the gesture event. This distance is calculated in steps as the finger is dragged across the table.

For the calculation of panning and tilting the negation of the step along the X axis is used to calculate the degree of pan. This rotates the camera around the Y axis. The Y step is then used to calculated the degree of tilt. Which rotates the camera around the world Z axis. The dynamic gesture set uses the same calculations but calculates the distance from the initial point, rather than the using stepping value.

The static zoom gesture uses the scale ratio provided from the scale event. The scale factor from both the X and Y touch axes are applied to move the camera forward and back along the world X axis. This is to ensure the full range of motion is translated from gesture to camera, no matter if the second finger is dragged left to right or from top to bottom.

The dynamic gesture set uses the ratio of the distance between the first and second fingers initial points compared to the distance between the first and second fingers current points. If the ratio is greater than one then the camera will move backwards or if smaller, forwards. This calculation results in a velocity instead of a movement step.

As mentioned in the previous section, a bug prevents the rotation gesture from using the rotation event, instead the scale event is used. calculating the rotation manually. First, the angle is calculated between the current position of both fingers. The static gesture takes the difference between this and the previous angle, while the dynamic gesture takes the difference from the angle between the initial positions instead.

For the strafing and lifting gestures, the static gesture looks at the difference between the current and previous position of each finger. The minimum change is then used as a step for the gesture. Again, the dynamic gestures use the initial points rather than the previous points to calculate a velocity instead.

5.5.1 Visual Design

As an aid to help the user with more accurate gestures, a number of additional user interface elements were included. The first is a rotation aid. This appears when a two finger gesture is being performed.

When a second finger is place on the table, a yellow circle is drawn around the first finger. The radius of the circle is determined by the distance between the first and second touch points when the second finger was first placed. This circle primarily acts as a rotation aid, but also gives an indication of which direction to drag the finger to perform the scaling gesture.

When using the dynamic gestures the initial touch points of each finger are important. To signify this and remind the user where those points were, coloured circles are placed at their positions. The solid green circle is the initial position of the first finger, while the red is the position of the second. These are set when each finger first makes contact with the touch surface and removed if the finger is raised.

At the center of each window a small black dot can be seen. This is the cross-hair for the camera. If this point is aligned with a target, moving towards that point will eventually result in the camera reaching the target. It was communicated to users that the gestures could be considered to 'drag' this cross-hair around as opposed to the gestures directly manipulating anything actually inside the scene.

Figure 5.2 shows an example of these additions. It is also worth noting that the blue circles represent the current positions of the fingers. These are provided by MT4J as default and provide a good indication of whether the touch system is calibrated properly and how well the touch system is reading a users input.

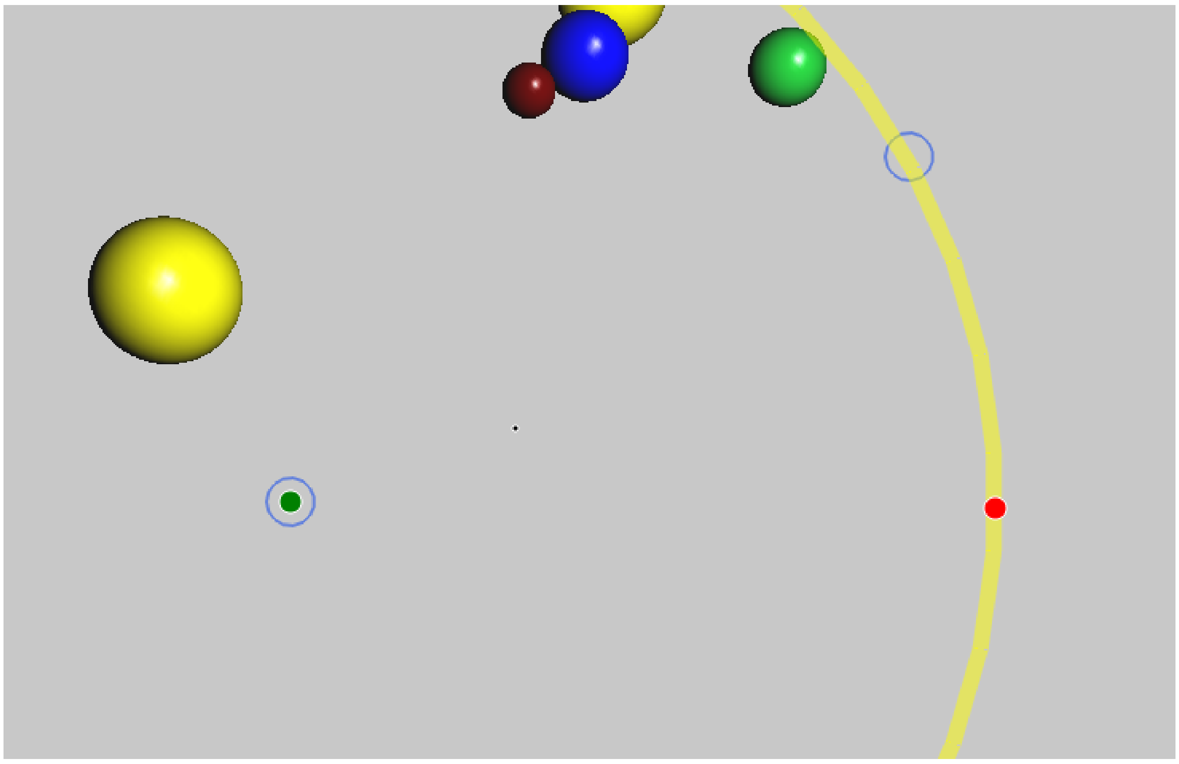
Figure 5.2: This is an example of the dynamic rotation gesture. The yellow circle is a rotation aid which appears around the first finger when the second finger is placed on the touch screen. The red and green circle represent the initial position of each finger.

Chapter 6

Evaluation

The goal of the evaluation was to test the intuitiveness, robustness and simpleness of each gesture. That is looking at how instinctively the users perform each gesture, how easy the user can correct any errors and stay on the path, and how they perceive the usability of each gesture.

The twelve participants where between the ages of 21 to 29, primarily male with two female test subjects. The participants were pulled from a wide range of backgrounds and many had not had much contact with multi-touch technology.

To evaluated each gesture set a testing process was devised. Key to this evaluation where two prototype scene types; the demonstration and test path scenes. The demonstration scene only contained static objects while the test scene contained the test paths, which could be completed. The demonstration scene allowed all the gestures to be used and practiced without the constraints of a test environment.

6.1 Demonstration Design

The demonstration scene, pictured in Figure 6.1, shows some of the elements the user can navigate around. All of the objects are static and can be clipped through. The purpose of the scene was to allow the user to practice each gesture at their leisure.

The different gesture sets could be switched to by using keyboard. Key 1 displayed the indirect gesture set, key 2 displayed the static gesture set and key 3 displayed the dynamic gesture set. The testing phase could be start with key T and ended prematurely with key Y.

These actions were implemented on the keyboard, as opposed to the touch screen, to better separate the user and tester domains. This allowed to the tester to manipulate the demonstration without having to interfere with the user.

6.2 Test Design

The gestures have two main use cases, orientation and navigation. Orientation involves the finer degree of control possible with the gestures. This is using small movements to align the camera or rotate it around objects in certain ways.

Navigation looks at moving the camera through the scene, often over large distances. This doesn't require as large a degree of control because there is more tolerance for errors. The possible variations of movements that can be perform but still result in the same location is also bigger.

Originally this project aimed to test both of these areas however due to time constraints,

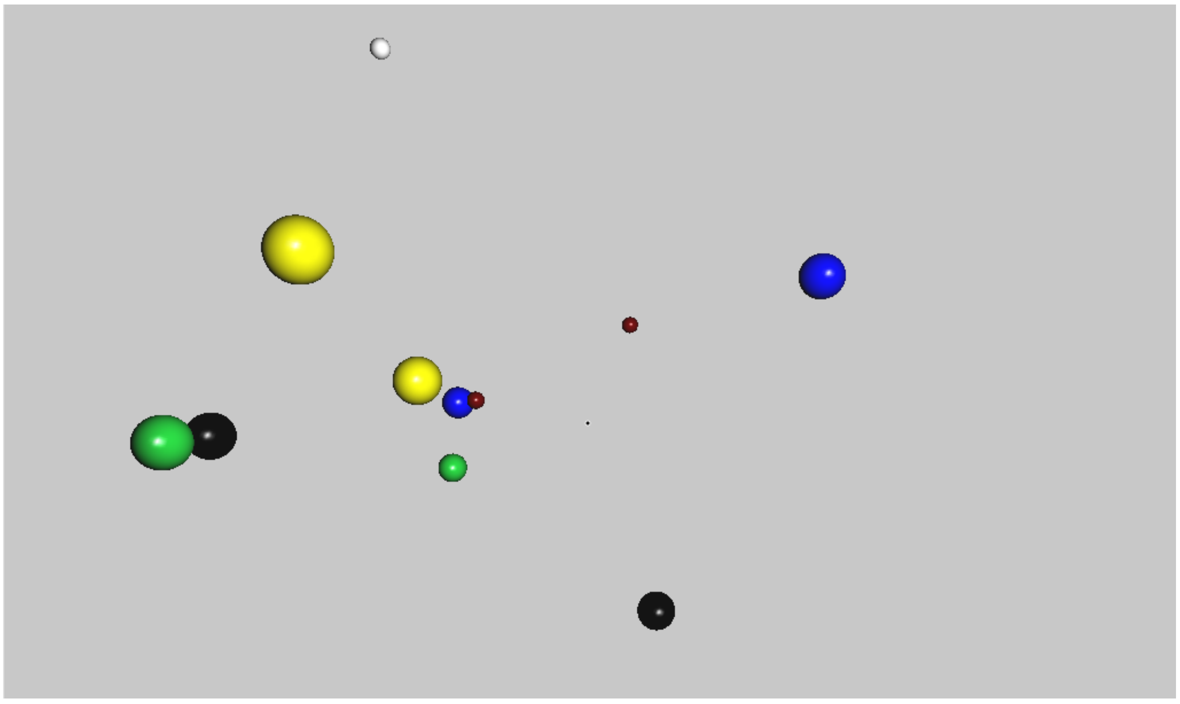

Figure 6.1: This is the gesture demonstration scene. It is filled with coloured static spheres. The black circle in the center of the image is the cross-hair for the camera. The cross-hair is the exact center of the screen.

navigation was favored over orientation. The tests developed for the prototype place orientation into a secondary role to the main objective.

6.2.1 Path Design

To test the navigation, way-point paths were created. Each gesture is tested once on each of the five different paths. The paths were designed to be a mix of easy to difficult movements. Each of these paths consisted of a set of five green spherical nodes. The first node in each path was coloured orange to indicate it was first. Each node is linked by a green line. An example of this can be seen in Figure 6.2.

The five paths were designed to be of varied difficultly. Each having a different degree of sharp bends and turns. The entire set of paths was made to allow the use of any of the six degrees of freedom. The paths are labeled zero to four.

Path zero can be seen in Figure 6.3. The user would start facing the lowest node, with the linked directly behind the first. After moving through the first and second a sharp turn the left is required to pass through the third node. Shallows turns to the left place the user through the final two nodes. The path is 1326 units long.

Path one, shown in Figure 6.4 is identical to path zero, except the user has to take a sharp turn upwards, instead of too the left. The purpose of this is to see if a user will simplify the path by rolling the camera or if they will using the other control instead. The path is 1326 units long.

Path two, shown in Figure 6.5, starts at the lowest node. The path consist of the series of moderate left and right turns and design to test how well a user can continue a gesture movement through each node. The path is 1428 units long.

Path Three, shown in Figure 6.6, starts at the lowest center node. The user must make a sharp left turn upon passing the first node. They must then turn very sharp to the right after node two. The both turns left and dips after node three. The dip rises again after node four, up to the final point. The path is 1789 units long.

Path four, shown in Figure 6.7, starts at the lowest central node. The path consist of a series of sharp left and right turns. The path is 1296 units long.

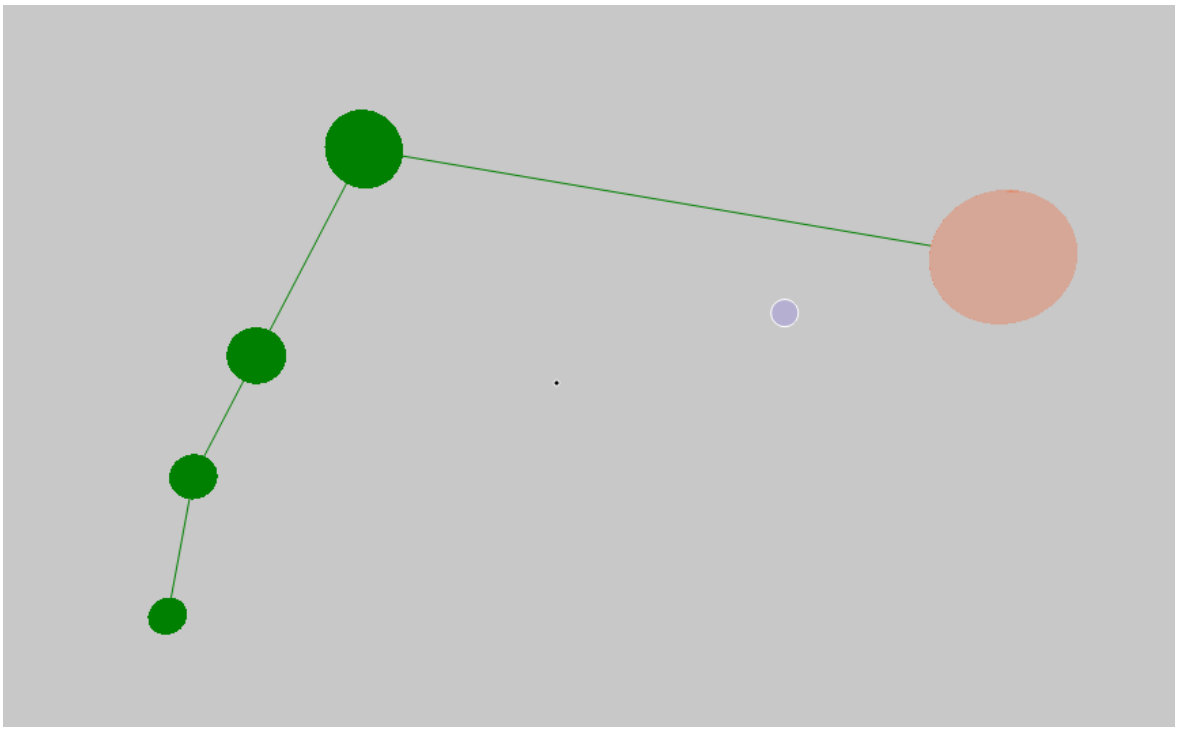

Figure 6.2: An example of a test path. The orange node is the first node in the path, with the linked green spheres following it. The purple circle between the black cross-hair and orange node is the hint indicator.

Waypoint Vectors (X,Y,Z)

	Waypoint 1	Waypoint 2	Waypoint 3	Waypoint 4	Waypoint 5	Distance
Path 0	(200,0,0)	(600,0,0)	(800,200,0)	(1000,400,0)	(1200,700,0)	1326
Path 1	(200,0,0)	(600,0,0)	(800,0,200)	(1000,0,400)	(1200,0,700)	1326
Path 2	(200,0,0)	(400,200,200)	(800,400,400)	(1000,600,600)	(1200,700,700)	1428
Path 3	(200,0,0)	(400,400,0)	(600,0,0)	(800,0,-400)	(1000,0,0)	1789
Path 4	(200,0,0)	(400,-200,0)	(200,-400,0)	(400,-600,0)	(0,-800,0)	1296

Table 6.1: Vector positions of way-points for each path. Every path starts at the same position in front of the camera. The distance is the sum of the distance between each connect points. The points are connected from waypoint one to waypoint two, waypoint two to waypoint three, so on and so forth.

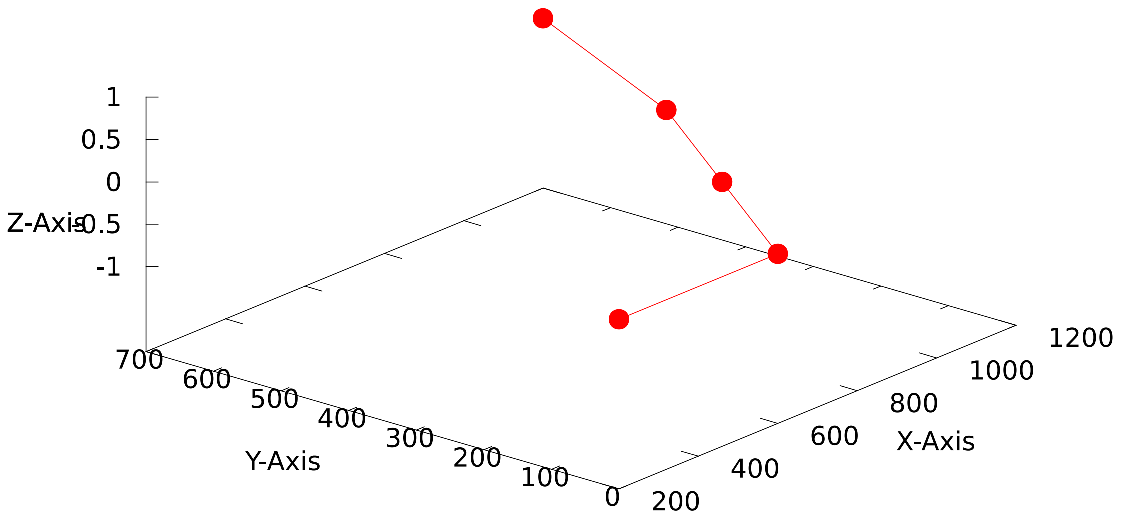

Figure 6.3: These are the waypoints for path 0. All nodes are located at the same Z value, attributing the small Z axis. The start node is the lower front-most node. The camera starts facing node one aligned with node two. This path is deisgned to be easy to follow and the entire path can be viewed from the original starting location.

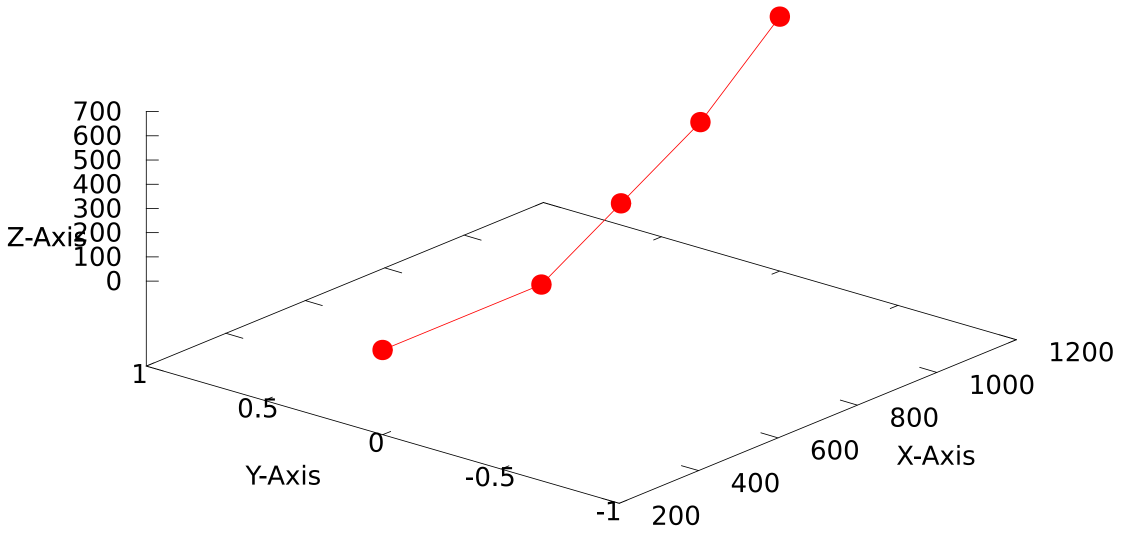

Figure 6.4: These are the waypoints for path 1. This path is designed to mimic path 0, to test uesrs recognistion and how they handle navigating the path compared to path 1. Notice the similarity between values on the Y, comapred to the values of the Z axis in path0.

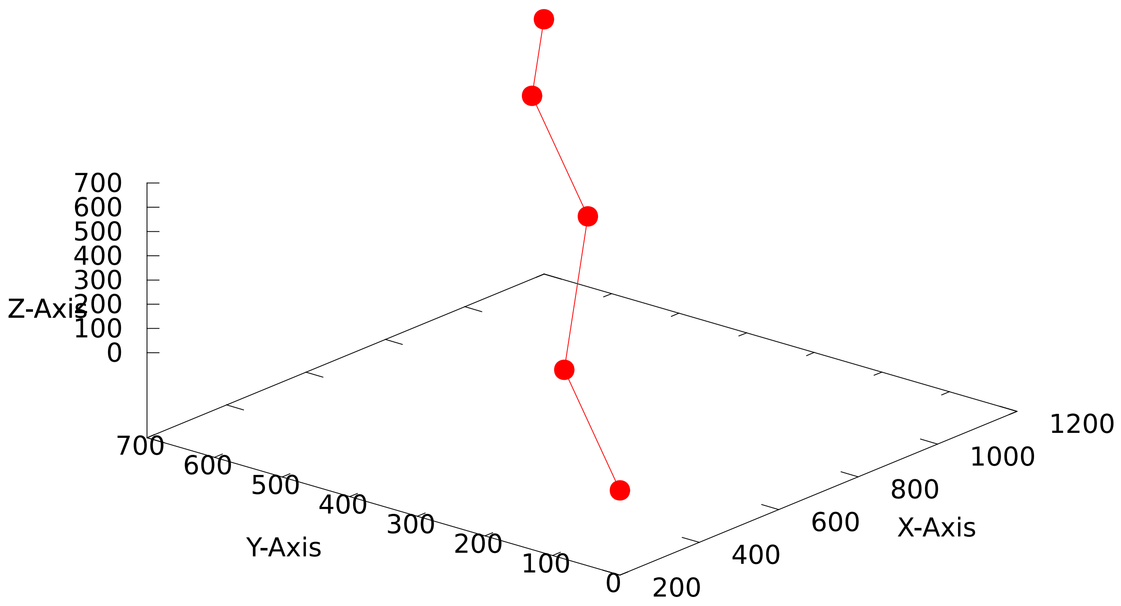

Figure 6.5: These are the waypoints for path 2. The path follows a zig-zag with medium sized curves. The nodes also move up the Z axis as the travel along.

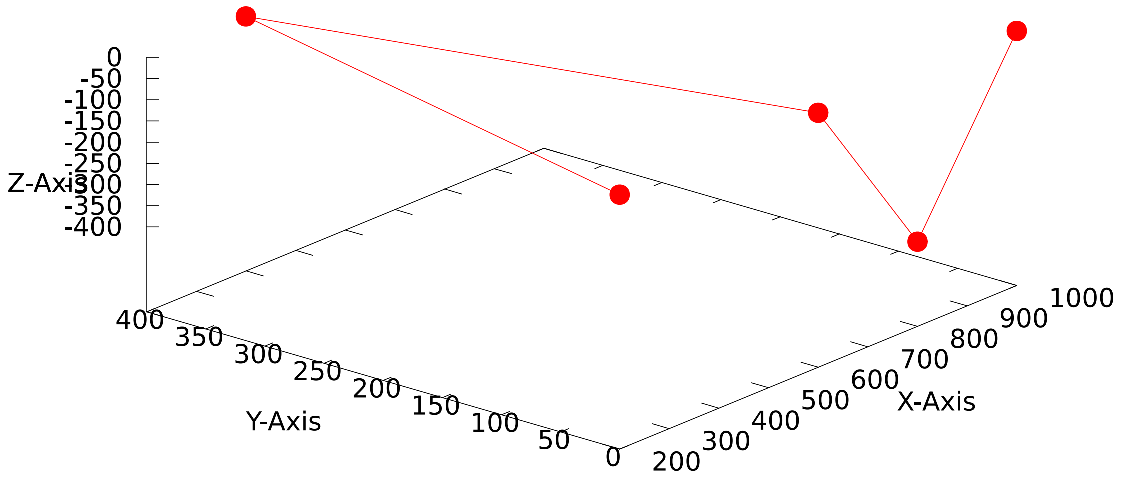

Figure 6.6: These are the waypoints for path 3. This path is the most complex out of all the paths test. The first not is the loewr center node. It features a very sharp bend around the second node and a dip from node three to node four and back up to node five.

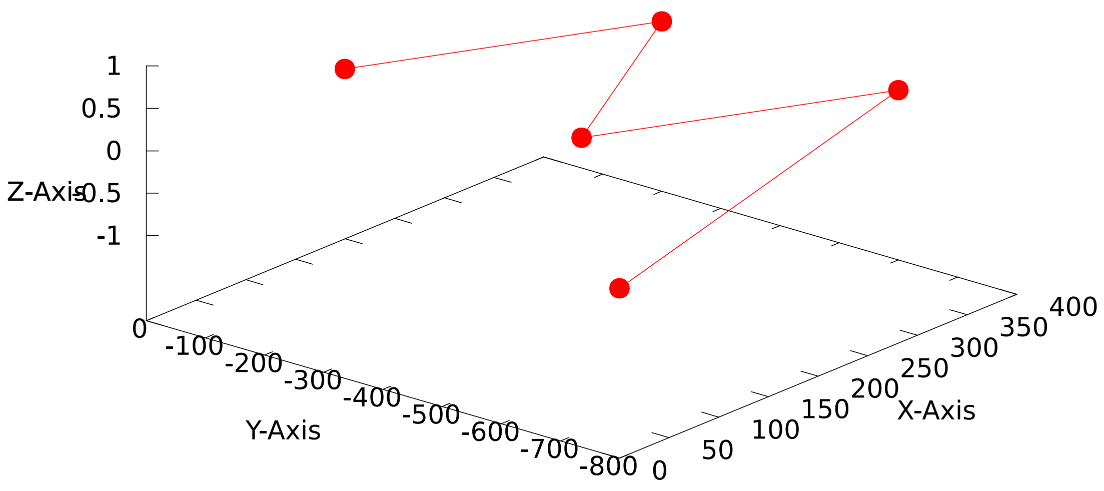

Figure 6.7: These are the waypoints for path 4. This is a more extreme version of the zig-zag, with sharper turns. This path stays on the same Z value, only alternating over the X and Y planes. The lower most node is the starting node.

6.2.2 Testing Aids

After initial pilot tests it was found that if the user became disorientated and lost they had no way of returning to the path without forfeiting the test. To help avoid this a hint indicator was developed. This can be seen in Figure 6.2, the purple circle between the black cross-hair and orange node is the indicator.

The hint indicator shows which direction the camera would need to pan or tilt in to be facing towards the orange node. This can some what simplify the navigation task as the user is only required to align the black cross-hair onto the purple circle. As the indicator is only on a 2D plane it is possible for the cross-hair to align to the indicator but have the target directly behind the camera.

If the camera were to pass through the first orange node, that node would disappear and the next node in the path would then be coloured orange. This indicates the next node in the path to pass through. The user must continue passing through nodes till the end of the path. When the final node is passed through the test will end.

Due to problems mention earlier in the previous chapter, with the MT4J camera, it can be hard to tell when the camera is passing through an object. To provide more of an indication a sound is played when the active node is intersected. This is to complement the node disappearing and to prevent users wasting time trying to move back and check if the node had been completed.

To maximise user engagement the entire test procedure was automatic. A screen was displayed at the beginning of each path. The screen told the user which gesture set they were using and which path the were up too. The user could then touch a blue square to begin the test for real. This allows the user to think about the next gesture set they will be using and also offers the tester the opportunity to step in and further explain if something is going wrong.

During the test the time, users performance and actions are recorded for each movement. While the actual gestures being performed by the user are not recorded the resulting manipulation of the camera is. This would allow the exact path to be recreated to see where a test participant went wrong.

The gestures themselves could not be recorded as four of the degrees of freedom use the same scale event to pick up the gesture movement. Recording that the scale event was trigger then become meaning less. Other information such as the overall time and time to reach each node are recorded.

6.3 Testing Procedure

The testing procedure was performed in three stages; the demonstration and practice stage, the evaluation stage and a survey stage, for how well the users think the performed. The firsts two stages were completed on the table, with the survey being completed on paper.

Demonstration and Practice. The demonstration stage involved the user practicing each gesture in turn. The tester would demonstrate how to perform each gesture and show the effect they would have on the demo. The user was then given the option to play around with the gesture for as long as they liked. It was explained to the user that they would be required to navigate and pass through the center of spherical nodes for the evaluation stage. The users were then encouraged to attempt this using the spheres in the demonstration scene. When the user had grown comfortable with the gesture they would indicate they were ready to move on to the next.

The indirect gestures were demonstrated first, followed by the static and dynamic gestures. This order was chosen so that the easier controls were learned first, allowing the user to become familiar with the program tasks before having to grapple with the more difficult controls.

Testing. The testing phase was performed automatically by the user. The gestures were tested in a random order, to prevent any bias around which gesture set was used first. The five paths were also randomly ordered for the same reason. The tester would start by describing the task required in more detail. They would explain how the directional indicator works and how the path way-pointing works with the sound when a node is intercepted. These indicators are particularly important as they were not included in the demonstration scene. A screen was displayed before each test allowing the user to continue each test at their own pace. When all the tests had been completed a blank screen would appear.

During each test path the users movements and other events are recorded. At the end of each test the record is outputted to a log file which also records which gesture set the test was in and which path. A time stamp of the start and finish of the test are also included. The file name is formatted to include the users name, the gesture set, the path and the time when the log was created.

Survey. The survey covers two main areas, the demographic information and qualitative questions. The demographic information included basic data such as age and gender, but also included how much prior experience each user had with touch screens or navigating through 3D scenes.

The main focus of the survey asked users to differentiate between how well the gestures performed for navigation and how well they performed for more fine-grained orientation.

An open feedback section at the end meant users could also communicate problems they had with the gesture sets or the setup of the table.

6.4 Results and Discussion

Two types of data were collected, a quantitative set of performance data for each gesture and a qualitative information from the survey.

6.4.1 Testing Feedback

Given the small set of participants, it was meaningless to perform complex statistical analysis of the data. Simple analysis of the data have been performed in its place. This includes a box plot of performance times for each of the three gestures and tables which looking at the fastest, slowest, mean and median times for each gesture set. The box plots have had the outliers removed to be show the general trends. These outliers are instead included with the tables, particularly when looking at the slowest times.

The indirect gestures performance box plot, see Figure 6.8 shows the performance for each path are tightly clustered. That is the upper and lower quartiles are relatively small. This indicates that the performance times tend towards the faster end of the scale for this gesture set. The overall performance of the users, even with the more complex paths such as three and four, suggests that these controls were intuitive and simple to use.

The static gestures, see Figure 6.9, are much looser compared to the indirect gestures. The first three paths are comparable, especially when ignoring the upper quartile. This shows for simple paths that the static gestures can perform well. The tighter grouping for these paths suggest the gesture remained intuitive.

Paths three and four, however show no relation to these paths with the indirect gestures. The large spread of the data over these points indicates the many more users struggled to us the static gesture effectively. This could either be due to an increased amount of errors or difficult for some user to quickly repeat the gestures necessary to travel the more complex paths.

Much like the indirect gestures, the dynamic gestures have much tighter groupings, shown in Figure 6.10. The lower quartiles of each path are similar to the indirect gestures but otherwise, the times indicate that the dynamic gesture is slower overall compared to both the other gesture. It is interesting to note that more confidence is shown using the dynamic gestures with paths three and four still being tightly group compared to the static gestures performance over the same paths. This could further suggest that the repetitive nature of the static gestures hampers their performance.

Both the indirect and dynamic gesture sets seem to perform well with path one. The curiosity here is that the path is identical to path zero, only rotating on its side. This would require the user to tilt up and down as opposed panning left and right. As both tilt and pan are performed using the same gesture in both sets, it is interesting that the two paths do not have a similar ranges. Due to the limited sample size it is impossible to conclude whether users perform better using tilting instead of panning.

While the tables provided do include the outlier values, they are still useful for looking at how well users performed overall. Unsurprisingly the indirect gesture set proved to have the fastest time per path, the fastest average time and the lowest median time out of all of the gesture sets. These are shown on tables 6.2, 6.3 and 6.5. The indirect gesture set was only beaten on path four for the fastest time, path one for the average fastest time, and path two for the lowest median times. In all three of these instances indirect gestures were beaten by the dynamic gesture set.

The static gesture portrays the opposite story, with the highest of the fastest times on all but path 3. It also has the slowest average times on all but path 4 and the highest median times for all paths. While this doesn't suggest that the gesture isn't simple and intuitive, it does prove user do not become comfortable using it at speed after only five tests.

As expected the slowest times prove to be a mixed bag, see table 6.4. These times are more indicative of the difficulties user had either performing the gesture or completing the task. Static gestures are slowest on paths one, three and four. Indirect gestures are slowest for path one and dynamic is slowest for path one. It is worthwhile noting that the static gestures still performed badly overall. This would suggest most users had difficulty getting to grips with the gesture itself rather than problems with performing each task.

It would be safe to conclude that the indirect gesture set is simple and intuitive to use. The lack of large outliers would also indicate that this gesture set is reasonably robust. Due to the nature of the controls, it is obvious that this set is the most resistant to interference. Unfortunately it is also the least likely to map to devices of different size without diminishing some of the controls. This would require further analysis.

These initial data sets suggest the static gestures set does not come intuitively to people, especially compared to the other two gesture sets. Both the dynamic and static gestures are simple by definition, only using a maximum of two touch points. The large outliers of the static set could indicate that the gesture is not robust, however further study would have to be performed before this is decided.

Although slower overall than the indirect gesture set, the dynamic gestures still had a tight clustering of values across all paths. This could indicate that the gestures are intuitive and robust. Both the static and dynamic gestures would scale well to different size device, but unfortunately neither are very resistance to interference.

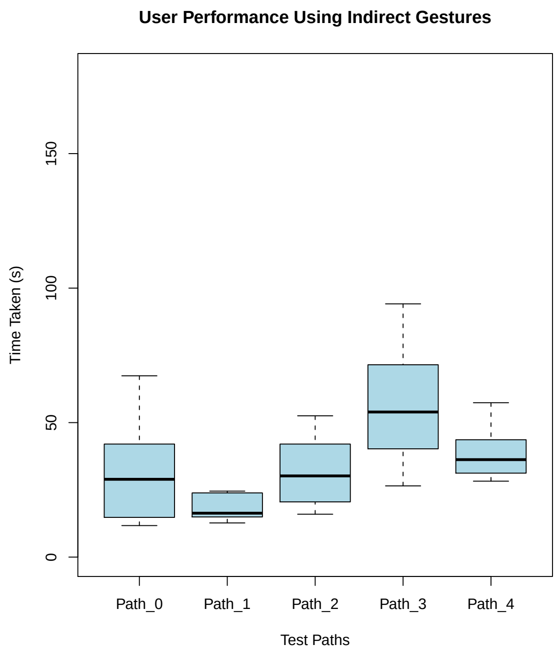

Figure 6.8: Box Plot of Indirect Gesture Performance. Outliers have been ommitted.

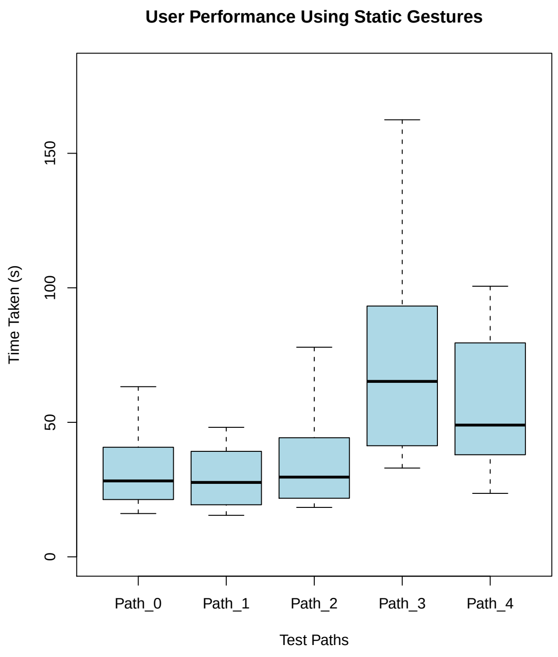

Figure 6.9: Box Plot of Static Gesture Performance. Outliers have been ommitted.

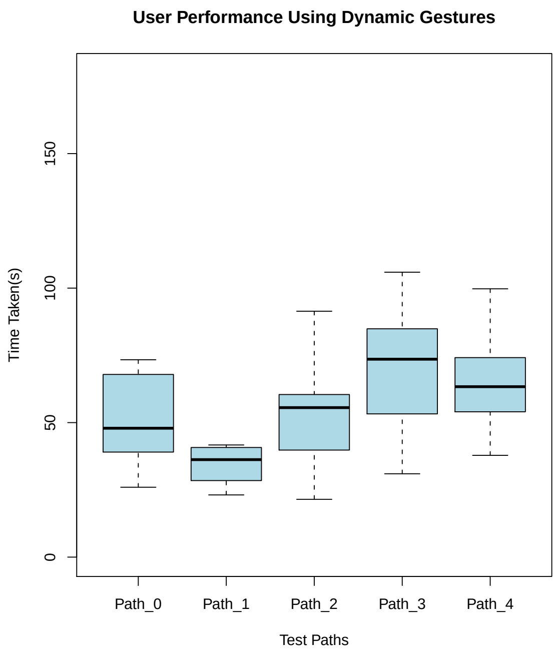

Figure 6.10: Box Plot of Dynamic Gesture Performance. Outliers have been ommitted.

Fastest times per path per gesture (ms)

	Indirect	Static	Dynamic
Path 0	**11718**	25974	16090
Path 1	**12702**	23118	15435
Path 2	**15949**	21485	18371
Path 3	**26471**	30981	32994
Path 4	28235	37830	**23591**

Table 6.2: The fastest times across all test participants. The fastest time for each path is highlighted.

Mean times per path per gesture (ms)

	Indirect	Static	Dynamic
Path 0	**33982**	57856	35321
Path 1	39522	51905	**29293**
Path 2	**35069**	73260	36182
Path 3	**60403**	78835	71738
Path 4	**30589**	67229	70831

Table 6.3: The mean times across all test participants for each path. The fastest averages are highlighted.

Slowest times per path per gesture (ms)

	Indirect	Static	Dynamic
Path 0	83412	**117592**	91762
Path 1	**260275**	214112	48143
Path 2	88434	**314228**	77926
Path 3	128949	**200132**	162463
Path 4	57396	112806	**258645**

Table 6.4: The slowest times across all test participants. The slowest time for each path is highlighted.

Median times per path per gesture (ms)

	Indirect	Static	Dynamic
Path 0	**28927**	47912	29777
Path 1	**16327**	36235	27656
Path 2	30185	55570	**29631**
Path 3	**53941**	73570	65210
Path 4	**36237**	63352	48977

Table 6.5: The median times across all test participants for each path. The lowest medians are highlighted.

6.4.2 Survey Feedback

The survey results help to clarify what was seen in the test results. The indirect gestures are well liked for both fine-grained orientationa and navigating. This reflects how the performed in the user test, the participants obviously found them easy and intuitive to use. The static gestures are more of a mixed bag, the results spread between 'Strongly Agree' and 'Disagree' for both fine-grained orientation and navigation. The participants seem to struggle with the static gestures the most, confusing which gesture did what.

The dynamic gestures show to be the most liked for navigation but the opinion on fine-grained control spread. This backs up the early results that the dynamic geture can produce a quicker movement, but lose out on accuracary. This is likely what stopped the gestures from performing better than the indirect controls in the user tests.

The pool of test participants provided a good range of prior experience with multi-touch devices. The participants were mainly males in there early twenties with a lot of experience navigating 3D worlds. This was mainly from computer games as opposed to modeling software.

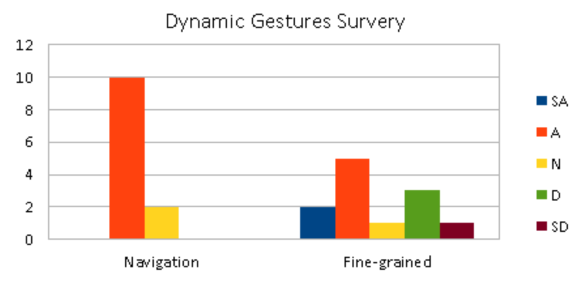

Figure 6.11: User feedback on the dynamic gesture set.

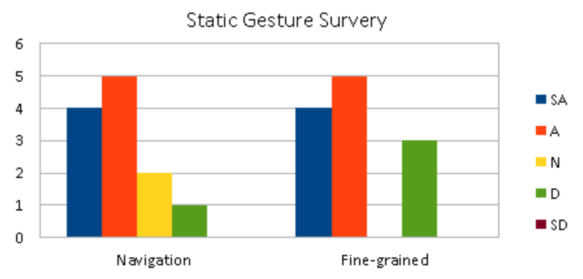

Figure 6.12: User feedback on the static gesture set.

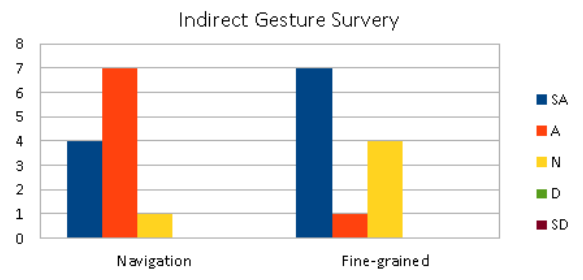

Figure 6.13: User feedback on the indirect gesture set.

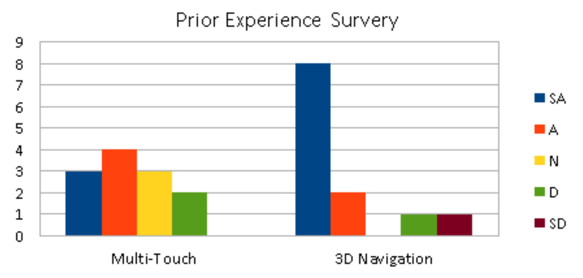

Figure 6.14: Users prior experience with multi-touch and 3D navigation.

Chapter 7

Conclusions

This paper explored the problems associated with both 2D mutli-touch gestures and navigation through 3D space. Existing solutions, such as graphic modeling programs or flight simulators can be hard to use, even with a keyboard and mouse. Any gestures that were developed would need to avoid the complexity. A list of criteria for was defined, so that each gesture could be evaluated and measured against the project goals.

Three gesture sets were developed. The indirect, static and dynamic sets. The indirect uses an on-screen control panel to mimic a physical set of controls on the touch display. Each control was touch sensitive, allowing them to be manipulated separately. The static gestures consisted of a set of four gestures, using both one and two touch points. The dynamic gesture set used the same gestures as the static set, but changed the velocity of the movement as opposed to just translating the position.

To demonstrate these gestures a prototype program was developed. This allowed the conclusion of a number of aids to help control the gestures. For the static and dynamic gestures, a yellow circle appears around the first finger when both fingers are placed on the table. For the dynamic gestures, a green circle is placed at the initial position of the third finger and a red circle is placed at the initial position of the second finger. These circles illustrate were the dynamic gestures are calculating their velocity from, making the user control easier.

To evaluate the gestures the prototype was extended to include a testing suite. The suite contained a set of paths that the user would have to navigate along. This allowed a user study with twelve participants to be conducted. Each of the paths were tested against every gesture, and the time and performance data recorded.

Due to the small sample size of user, no complex statistical analysis could be perform. Instead simple analysis was undertaken. These tests suggested that the indirect gesture set was the fastest overall, followed by the dynamic then static. Both the indirect and dynamic gestures had tight value groupings, indicating the users were confident and consistent in the use of those gestures. The static gesture set had far looser groups, particularly for the more complex paths three and four. This suggests this gesture become harder to control when the task was more difficult, it is possible the repeating nature of this role slows this down.

7.1 Future Work

One requirements of the gesture sets was that they were scalable to both small and large devices. This was not tested as part of the evaluation, however this is still and important criteria. This one of a number of areas that would benefit from further testing. Another area is specifically testing both orientation and navigation. One problem with solely doing a navigation test is that it does stress the use of all degrees of freedom. Often a user will get accustom to using one style of navigation and not be forced to try anything different.

Overall, the main improvement would to test more expansively on a larger set of people. This would allow for proper statistical analysis of the data, leading to some more concrete results.

Bibliography

[1] Developing applications for microsoft surface 2.0. Tech. rep., Microsoft.

[2] BLENDER FOUNDATION. *Blender Manual: 3D Interaction*, 2.5 ed.

[3] D. KAMMER, M. KECK, G. F., AND WACKER., M. Taxonomy and overview of multi-touch frameworks: Architecture, scope and features. In *Workshop on Engineering Patterns for Multi-Touch Interfaces* (June 2010).

[4] ECHTLER, F., AND KLINKER, G. A multitouch software architecture. In *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges* (New York, NY, USA, 2008), NordiCHI '08, ACM, pp. 463–466.

[5] FU, C.-W., GOH, W.-B., AND NG, J. A. Multi-touch techniques for exploring large-scale 3d astrophysical simulations. In *Proceedings of the 28th international conference on Human factors in computing systems* (New York, NY, USA, 2010), CHI '10, ACM, pp. 2213–2222.

[6] HANSEN, T. E., HOURCADE, J. P., VIRBEL, M., PATALI, S., AND SERRA, T. Pymt: a post-wimp multi-touch user interface toolkit. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces* (New York, NY, USA, 2009), ITS '09, ACM, pp. 17–24.

[7] KALTENBRUNNER, M. reactivision and tuio: a tangible tabletop toolkit. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces* (New York, NY, USA, 2009), ITS '09, ACM, pp. 9–16.

[8] KALTENBRUNNER, M., BOVERMANN, T., BENCINA, R., AND COSTANZA, E. TUIO - A Protocol for Table Based Tangible User Interfaces. In *Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation (GW 2005)* (Vannes, France, 2005).

[9] KIM, J.-S., GRACANIN, D., MATKOVIC, K., AND QUEK, F. iphone/ipod touch as input devices for navigation in immersive virtual environments. In *Virtual Reality Conference, 2009. VR 2009. IEEE* (2009), pp. 261 –262.

[10] LAUFS, U., RUFF, C., AND ZIBUSCHKA, J. MT4j - A Cross-platform Multi-touch Development Framework. *ArXiv e-prints* (Dec. 2010).

[11] REISMAN, J. L., DAVIDSON, P. L., AND HAN, J. Y. A screen-space formulation for 2d and 3d direct manipulation. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology* (New York, NY, USA, 2009), UIST '09, ACM, pp. 69–78.

[12] SCHÖNING, J., BRANDL, P., DAIBER, F., ECHTLER, F., HILLIGES, O., HOOK, J., LCHTEFELD, M., MOTAMEDI, N., MULLER, L., OLIVIER, P., ROTH, T., AND VON ZADOW, U.

Multi-touch surfaces: A technical guide. Tech. Rep. TUM-I0833, University of Munster, 2008.

[13] Schöning, J., Hook, J., Bartindale, T., Schmidt, D., Oliver, P., Echtler, F., Motamedi, N., Brandl, P., and von Zadow, U. *Tabletops - Horizontal Interactive Displays*. 2010, ch. Building Interactive Multi-touch Surfaces, pp. 27–49.

[14] Seth. Getting started with multitouch, May 2008.

[15] Seth. Community core vision, 2011.

[16] Teiche, A., Rai, A. K., Yanc, C., Moore, C., Solms, D., Cetin, G., Riggio, J., Ramseyer, N., D'Intino, P., Muller, L., Khoshabeh, R., Bedi, R., Bintahir, M. T., Hansen, T., Roth, T., and Sandler, S. Multi-touch technologies. http://nuigroup.com/, 2009.

Appendix A

Raw Tables

Indirect Gesture times per user per path (ms)

Alias	Path 0	Path 1	Path 2	Path 3	Path 4	Average	Total
User 1	67387	16821	32353	57919	30631	41022	205111
User 2	15123	14471	15949	41019	52827	27878	139389
User 3	27737	16477	18382	50893	37457	30189	150946
User 4	28243	15399	33818	61622	57396	39296	196478
User 5	13963	15697	52525	35085	33848	30224	151118
User 6	29610	45169	28686	56201	42628	40459	202294
User 7	14425	24526	17851	39490	28235	24905	124527
User 8	44466	260275	88434	94137	44631	106389	531943
User 9	32109	23200	22681	81374	35016	38876	194380
User 10	83412	12702	31683	128949	38332	59016	295078
User 11	39588	13354	50234	51681	31800	37331	186657
User 12	11718	16176	28236	26471	30589	22638	113190
Average	33982	39522	35069	60403	38616	41519	207593

Table A.1: The time taken per participant for the indirect gesture set. The averages for each path and user, along with a total time or all paths is included.

Static Gesture times per user per path (ms)

Alias	Path 0	Path 1	Path 2	Path 3	Path 4	Average	Total
User 1	26658	42297	37208	89683	258645	90898	454491
User 2	16363	18172	19769	32994	28095	23079	115393
User 3	16090	15435	23138	33128	49199	27398	136990
User 4	36279	27124	77926	53779	48755	48773	243863
User 5	21109	15815	21805	44050	23591	25274	126370
User 6	63253	20533	51345	61556	69973	53332	266660
User 7	29777	25700	28079	74259	89079	49379	246894
User 8	91762	48143	67448	162463	100594	94082	470410
User 9	33623	28188	36163	104752	47779	50101	250505
User 10	45204	31680	21753	68863	47901	43080	215401
User 11	22257	41345	31182	96775	58241	49960	249800
User 12	21480	37084	18371	38554	28120	28722	143609
Average	35321	29293	36182	71738	70831	48673	243366

Table A.2: The time taken per participant for the static gesture set. The averages for each path and user, along with a total time or all paths is included.

Dynamic Gesture times per user per path (ms)

Alias	Path 0	Path 1	Path 2	Path 3	Path 4	Average	Total
User 1	41921	39039	34293	66617	64706	49315	246576
User 2	28472	36029	46323	41274	68844	44188	220942
User 3	43286	36304	41750	86786	55532	52732	263658
User 4	58295	29852	37830	105914	37830	53944	269721
User 5	25974	39815	60820	39748	99733	53218	266090
User 6	112298	27046	59772	71630	60512	66252	331258
User 7	52537	41688	91410	79264	112806	75541	377705
User 8	117592	214112	314228	200132	79481	185109	925545
User 9	62456	36165	60063	75510	63477	59534	297671
User 10	38528	72913	54051	82960	63227	62336	311679
User 11	73358	26782	57089	65206	48091	54105	270526
User 12	39559	23118	21485	30981	52506	33530	167649
Average	57856	51905	73260	78835	67229	65817	329085

Table A.3: The time taken per participant for the dynamic gesture set. The averages for each path and user, along with a total time of all paths is included.

Feedback from user survey.

	User 1	User 2	User 3	User 4	User 5	User 6	User 7	User 8	User 9	User 10	User 11	User 12
I have a lot of experience with multi-touch devices.	D	D	N	A	SA	SA	A	A	N	N	A	SA
I have a lot of experience with navigating in 3D worlds either in games, or similar environments (such as modeling).	A	SA	SA	SA	SA	D	SA	SD	SA	A	SA	SA
The dynamic Hestures were easy to control in supporting navigation and target acquisition.	A	A	A	A	N	A	A	A	A	A	N	A
The static gestures were easy to control in supporting navigation and target acquisition.	A	A	D	N	A	A	SA	A	SA	SA	SA	N
The indirect gestures were easy to control in supporting navigation and target acquisition.	A	A	A	A	A	SA	SA	SA	A	N	SA	A
The dynamic gestures supported fine-grained control for navigation and target acquisition.	D	SA	A	A	D	D	N	A	A	A	SD	SA
The static gestures supported fine-grained control for navigation and target acquisition.	SA	A	D	D	A	A	SA	A	SA	SA	A	D
The indirect gestures supported fine-grained control for navigation and target acquisition.	N	N	A	SA	N	SA	SA	SA	SA	N	SA	SA

Table A.4: The user answers to each survey question.