Summary

This project aims to allow a user to draw free hand in 3D space by representing the movement of a tracking device, as a line. The system was designed as tool to be used by artists, to explore the possibilities of drawing in 3D space. As a result the system uses a simple intuitive interface so not to disrupt the creative process, instead of offering a wide range of features typical of a program designed from a CAD perspective.

The project has investigated previous research conducted in a similar area, which helped to inspire the design of this system. The system utilises the tracking hardware to draw a representation of the shape drawn by the user in real time. It also allows the manipulation of the scene to make the difficult task of drawing in 3D space easier. User's can load and save their drawings so that they can work on a piece over a period of time; there is also the option of exporting to VRML allowing users to view their work using a web browser.

On completion of the project the system offered a good base upon which to add features allowing the concept of drawing 3D space to be developed further, some possible enhancements have been highlighted.

Acknowledgements

I would like to thank John Stell for his guidance and support throughout, and for arranging the opportunities that have contributed so greatly to the success of this project.

I would also like to thank John Hodrien for guiding me through the IRIX operating system, and for his invaluable feedback throughout the interface design.

Finally I would like to thank Claude Heath for making the effort to travel the length of the country to evaluate the system, and for the excellent feedback that he provided. I only hope that the trip was as useful for him as it was for me.

Contents

1. Introduction

Digital art is a relatively new area offering a vast range of possibilities for artists to experiment with all kinds of technology. Paul (2003) makes the distinction between art that is created using digital technology e.g. photography, and art that uses the technology "as its very own medium." This project is concerned with developing a system that will allow users to create art that falls into the latter category. Digital Art does not exclusively encompass the development of tools with which to draw in the digital medium, for example William Latham created a computer program that created sculptural three dimensional (3D) forms based upon the genetic properties that he programmed. Another example was Osmose, created by Charlotte Davies (1995), a fully immersive virtual environment that allowed 'imersants' to float around, and explore her digital world, her work of art. The aim of the project described in this report is to allow artists to create their art using the digital medium; once saved to disk, the piece can evolve further or be used as the basis of new work.

1.1. The Problem to Be Solved

Currently the majority of drawing software allows the user to draw in two dimensional (2D) space; the artist Claude Heath would like software to be developed to allow the user to work in 3D space. A useful analogy is to think how a person sketches using a pen and paper; Claude would like to use a system that would allow him to draw in a similar manner, but in 3D space.

The first step towards solving the problem is to establish the features that such a system should have, based on requirements specified by Claude, and research into similar projects to gain inspiration and to analyse other approaches to implementing such a system.

The fundamental requirement of the system is to allow the user to draw in 3D space; this will require the use of tracking hardware that is able to measure a device's position relative to a transmitter. Research will be conducted to establish how to use this hardware, and then to develop the application further to exploit it to its full potential.

The data read from the tracking device will be in the form of a series of co-ordinates in 3D space; these then need to be manipulated so that they represent a line. The line can be drawn using different types of mathematical functions, these will be researched and a representation appropriate for the system shall be chosen.

It is important that the system has a simple intuitive user interface (UI), designed so that users who lack experience working with computer systems are able to concentrate fully on drawing, rather than struggling to use the application. Designing interfaces for 3D environments is a relatively new area of Human Computer Interaction (HCI), without any experience or formal guidelines to follow, the area will be investigated to aid the implementation of a usable product.

1.2. The End User

Claude Heath is an artist based in London. He has had a number of solo exhibitions since 1995, was included in "Young British Artists VI" at the Saatchi Gallery London in 1996, and was the 2002-03 Kettles Yard Artist Fellow at Christ's College, Cambridge. His work is included in a number of collections including the Arts Council Collection at the Hayward Gallery, The British Museum, Deutsche Bank, the Henry Moore Institute and the Walker Art Gallery. More details and a discussion of some of his drawings can be found in Kinston (2003).

1.3. Background Reading

Throughout the project extensive research was conducted to gain a better understanding of the problem to be solved. Wherever this research has been used, the information has been cited and attributed to its source.

At the outset, Fores (2004) was consulted to find projects from previous years that could be of use when conducting this project. Using these reports, a list of appropriate literature was compiled to act as a starting point from which to begin the research. The Electronic Journal, ACM (Association for Computing Machinery) was searched to retrieve articles concerned with research conducted in a similar area to inspire ideas as to how the system could be implemented. Throughout the project texts relevant to the task in hand were found by consulting course reading lists and using the University of Leeds Library catalogue search. The World Wide Web also provided much valuable information, though due care was taken to check that the information came from an appropriate and trustworthy source.

1.4. Methodology Used

When developing a new system, it is important that the project follows an appropriate software development methodology, giving the developer a framework upon which to build a system that will meet the user's needs. Pressman (2000) outlines the development methodologies that were considered. Each has its own strengths and weaknesses, and is particularly suited to the development of a particular type of system.

An adaptation of the Rapid Application Development (RAD) methodology appears to be appropriate for the development of this project; its iterative nature does not necessitate the specification of all of the system's requirements at the beginning of the project. Having no past experience of developing similar systems, the option to add and refine the requirements throughout development will be essential. Upon the completion of each iteration the project stakeholders develop a better understanding of the problem, enabling a better set of requirements to be defined for the subsequent iteration, (Avison and Fitzgerald, 2002).

When implementing a system using new technology it is inevitable that there will be initial implementation problems that will slow down the progress of the project. Pressman (2000) indicates that this does not complement the RAD methodology, where strict deadlines are enforced to keep the forward momentum of the project; but the RAD methodology allows some flexibility by ranking the requirements by order of priority, if the project looks like it will fail to meet an iteration deadline, then the least important requirements will be jettisoned to keep the project on schedule.

1.5. Project Schedule

The initial project schedule is outlined below; this is represented by means of a Gantt chart shown in Appendix B. Throughout the course of the project some slight changes were made to this schedule, these alterations are shown in Appendix C.

- *Research – 11/10/2004 – 09/01/2005 (13 weeks)* – Throughout this phase most of the background reading regarding how the system is to be implemented will be conducted. This involves deciding upon an appropriate software development methodology, a review of similar projects, and extensive research into the design of 3D UIs.
- *Specify user requirements – 29/11/2004 – 05/12/2004 (1 week)* – After researching similar projects and consulting with the end user, a set of plausible requirements can be finalised. Once the requirements are established, the research can focus on how they will be implemented.
- *Implement 2D prototype – 27/12/2004 – 30/01/2005 (5 weeks)* – The first step towards implementing the system is to develop a prototype that allows the user to draw freehand using a desktop configuration. The initial UI design will be carried out but this will be subject to change.

- *Christmas break and exams – 13/13/2004 – 23/01/2005 (6 weeks) –* Over this period less time will be spent on the project, allowing for preparations for the exams in January.
- *Convert 2D prototype to 3D – 31/1/2005 – 20/02/2005 (3 weeks) –* The next step is to integrate the tracking hardware into the system, and to alter the UI to work in 3D space.
- *Add functionality to the prototype – 21/02//2005 – 20/03/2005 (4 weeks) –* Once the tracking hardware has been integrated, the remaining development time will be concerned with implementing the user requirements, and developing the UI to make the system as easy to use as possible.
- *Testing – 21/03/2005 – 27/03/2005 (1 week) –* Upon completion of the implementation, testing will be carried out to find and rectify any problems that have arisen.
- *Evaluation – 21/03/2005 – 10/04/2005 (3 weeks) –* Once implemented the system will be evaluated, this will comprise of usability testing involving students from the university followed by an extended evaluation carried out by the end user.
- *Write up – 04/04/2005 – 27/04/2004 (4 weeks) –* This period shall be concerned with writing up the project report ready for submission. It can also act as a buffer in the event that the project falls behind schedule.

2. Analysis

This chapter outlines the process by which the system's requirements were defined, and the decisions made concerning the implementation of the system. Defining the requirements involved researching projects conducted in a similar area, and discussing with the end user what such a system should deliver and how the user should interface with it. Once a list of possible requirements was drawn up, it was submitted to the other project stakeholders for confirmation. The research into hardware and software available is outlined followed by research into how best the shapes drawn by the user could be represented.

2.1. Similar Projects

In the past there have been no attempts within the school to implement a system with similar functionality; Boulton (2004) implemented a virtual environment on the Holobench, which gave some useful insight into how to implement a system on the hardware available. Rickus (2003) was used as a reference to how to conduct a software development project.

Outside of the school, some similar applications have been developed; there are two approaches to solving the problem: one approach is to develop a system with Computer Aided Design (CAD) in mind; the other is to develop a system that can act as a tool to let artists experiment with the possibilities of drawing in 3D space. It is important to distinguish between the two approaches, as each requires different functionality.

An application developed from the CAD perspective was "3-Draw: A Tool for Designing 3D shapes," (Sachs et al, 1991), where a user could draw in 3D space using a combination of a tablet and a six Degree of Freedom (DOF) motion sensor. The author concluded that the system was a success with its high performance resulting from "the simultaneous use of both hands to provide complex 3D information." The process of drawing in 3D was simplified further by distancing transformation tasks from the process of actual shape capture.

Deering (1995) developed a system very closely related to this, allowing the user to, amongst other things, draw freehand in 3D space. The functionality offered was similar to a simple drawing package allowing the user to draw geometric shapes as well as freehand lines. The author highlights the difficulty of designing a user interface for a 3D environment, and outlines the reasoning behind the decisions made. The research concluded with feedback from an artist who used the system for an extended period of time; amongst other things it was established that when drawing the artist needed instant feedback as to the shape drawn, giving a greater feeling of control over the shape being created.

A project that took a more artistic approach to drawing in 3D was the Tacitus Project; the aim of which was to develop a system that allowed artists and craftsmen to carry out their practices in a 3D virtual environment. The system was implemented using a stereo monitor combined with a force feedback haptic device. Two aims of the project were "to explore the potential advantages of being able to work, think and respond to physical and visual stimuli, in a virtual, fully three-dimensional, non-gravity context," and, "to develop viable software applications and virtual 'hand tools' to enhance the creative practice of applied artists," (Shilito et al, 2001).

Schkolne (2002) developed a system that allowed the user to draw in 3D space using their hand. The system drew surfaces rather than lines, the shapes of which were determined by the shape of the user's hand gesture. The application allowed drawn objects to be edited, using a range of tools; for example, to stretch an object the user grabbed it using a pair of kitchen tongs, and then dragged it to the desired size. The author highlights that when people attempt to draw in 3D they very quickly learn how draw crude lines, but to create something complex requires a lot of experience. Another characteristic of the drawing in 3D space is that many artists approach the system thinking in terms of 2D, taking time to adjust to thinking in terms of depth whilst drawing.

Keefe et al (2001) were responsible for developing a system called CavePainting that fully immersed the artist inside a Cave Automatic Virtual Environment (CAVE) giving them a range of real artists' tools, fitted with tracking devices and buttons, with which to paint in 3D space. This research developed some interesting ideas regarding interaction with the system, and provided inspiration as to the sort of functionality this system could offer. Different artists used the system and gave positive feedback regarding its usability and the potential of using virtual reality as a medium for creating works of art.

2.2. Requirements Gathering

RAD requires participation from all stakeholders to define the requirements, and to outline which requirements should be tackled in which iteration. An efficient way to carry out this would be for all stakeholders to meet together in a Joint Application Development (JAD) workshop, (Avison and Fitzgerald, 2002). Such a meeting was impractical for this project, as Claude Heath was based in London, so the requirements were initially defined, and then passed on via e-mail to John Stell and Claude Heath for improvement, and a decision upon the order of priority.

It was decided that the minimum requirements for the system were:

- To allow the user draw a freehand curve in 3D space.
- To allow the user to save a scene that had been drawn.
- To allow the user to load a scene that had been drawn in a previous session.
- The system should have a simple, intuitive interface.

For each iteration of the RAD process a set of requirements must be specified, these are then given an order of priority in which they are to be completed. This order was based upon the 'MoSCoW rules', (Avison and Fitzgerald, 2002). Appendix D details the requirements specification for each iteration of the implementation.

2.3. Hardware and Software

A traditional desktop computer was unsuitable for a system of this nature therefore Roy Ruddle was consulted to establish the input and output technologies available within the school. The feedback given was that for an application such as this it was appropriate to use six DOF tracking devices, and to complement this, a display that could provide stereo output to give the illusion of depth. John Hodrien provided information regarding the hardware on which all of these devices were run and the platforms used.

The tracking hardware chosen was the Intersense IS-900 system, which provided four devices that could be tracked independently. The four devices were: a wireless wand, with five buttons and a joystick, one head tracking device, and two hand devices. The IS-900 system provided extremely accurate tracking data by the triangulating a device's position using ultrasonic technology. The position was tracked using six transmitters positioned strategically around the Holobench display. Although extremely accurate, it was possible for the system to lose track of the device if the user positioned the device so that the 'lines of sight' of four of the transmitters were blocked leaving the system unable to triangulate its position.

As stated before the IS-900's transmitters were situated around the edge of the Holobench display, giving no choice as to the output device used. Nevertheless the Holobench provided the stereo output that the system required. Stereo output provides a powerful depth cue by projecting two images alternately, one for each eye; meanwhile the user wears a set of stereo-glasses that cover each eye alternately to correspond with the display, (Bowman et al, 2004). A drawback of using the Holobench was that the projector displaying the output had a tendency to deteriorate, sometimes giving a poor quality display.

Initially the system was implemented using this combination of devices, however mid way through the implementation the IS-900 system developed a hardware fault making it unusable and consequently another set of devices had to be used.

The replacement used was the Ascension ‘Flock of Birds’ which was older technology that allowed the user to input data with a stylus and other tracking devices. The stylus had four buttons and a joystick, and was wired to the base unit. The device’s position was tracked accurately using a magnetic field, emitted from transmitter located above the user’s head. Tracking the position using a magnetic field meant that the ‘Flock of Birds’ system could not lose the device as magnetic fields pass through all objects, but it could suffer from jitter (unwanted signal variation). The change in input device allowed for a change to a better quality cathode ray tube (CRT) stereo projector.

All of the hardware above ran on the Silicon Graphics Onyx 3400, a high performance machine capable of stereo rendering. The Onyx 3400 runs the IRIX operating system, a variant of UNIX for Silicon Graphics machines.

The system was programmed in C++, the extensibility that Object Oriented Programming (OOP) offers, is suited to this system, allowing large problems to be divided into smaller ones, the components of which can be easily added to the system when ready (Deitel and Deitel, 2001). The graphical output was written using the OpenGL application programming Interface (API), this was chosen due to the programmer’s previous experience of using it.

2.4. Drawing a Curve

When a user draws a line, the system will read the data from the tracking device at certain time intervals. This data can then be fed into a mathematical function to plot a curve to represent the user’s motion. There are a number of approaches that can be used, advice was sought from Matthew Hubbard and Royce Neagle, as to which method was best suited to this application. A review of some of the concepts and methods used when plotting curves is outlined below.

When drawing a curve consisting of line segments, it is important that it appears to be smooth and drawn with a single line; how well this illusion is applied can be expressed in terms of parametric continuity, (Angel, 2003). The most fundamental continuity constraint states that the end points of two adjoining segments coincide, requiring that the three parametric components are equal as shown in fig. 2.1; this is referred to as C^0 parametric continuity, whereby,

$$m(1) = \begin{bmatrix} m_x(1) \\ m_y(1) \\ m_z(1) \end{bmatrix} = n(0) = \begin{bmatrix} n_x(0) \\ n_y(0) \\ n_z(0) \end{bmatrix}.$$

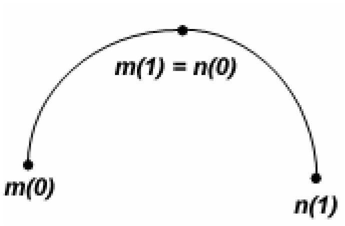


Figure 2.1 - A curve demonstrating C^0 parametric continuity at its join point

Making sure that the end points of the line segments meet is not enough to ensure the smoothness of a curve. Fig. 2.2 shows an example of a curve, comprised of smaller curves; although all of the lines join up, it could not be described as a smooth curve.

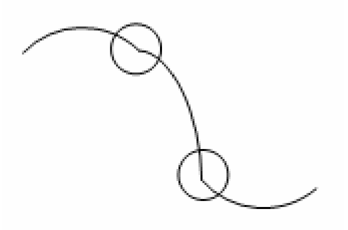


Figure 2.2 - A curve that fails to show C^1 parametric continuity

To ensure that the shape of the curve appears smooth, the derivatives of two consecutive segments must be equal at the point where they join. Once the derivatives of both segments are equal at the join point, the curve is said to have C^1 parametric continuity, whereby,

$$m'(1) = \begin{bmatrix} m'_x(1) \\ m'_y(1) \\ m'_z(1) \end{bmatrix} = n'(0) = \begin{bmatrix} n'_x(0) \\ n'_y(0) \\ n'_z(0) \end{bmatrix}.$$

This concept can be extended further to higher derivatives.

The simplest way to represent the curve is using piecewise linear interpolation (Burden and Faires, 2001), whilst giving the shape of the curve, it fails to demonstrate C^1 parametric continuity , as a result the curve is displayed as a jagged line, see fig. 2.3.

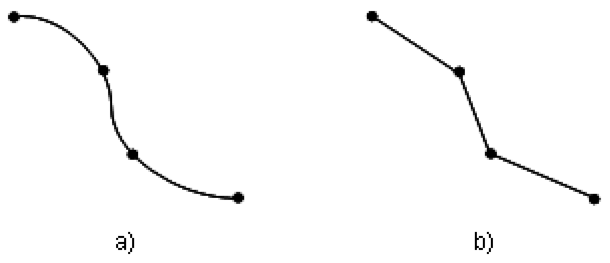


Figure 2.3 - A demonstration that piecewise linear curves fail to show C^1 parametric continuity

A slightly more complex approach is to represent the curve as a series of Bezier curves, Watt (2000) gives a detailed account of the mathematical theory on which they are based. Fig. 2.4 shows how a curve could be drawn using Bezier segments, notice that to enforce C^1 continuity, the control points either side of the join point, and the join point itself must be aligned. For this reason Bezier curves were inappropriate for this application; the control points would be points that were read from the tracking device, and there would be no way of enforcing this alignment. Another drawback was the fact that the data had to be used in groups of four, leading to the issue of how to deal with the inevitable situation whereby the number of data points on the line was not a multiple four.

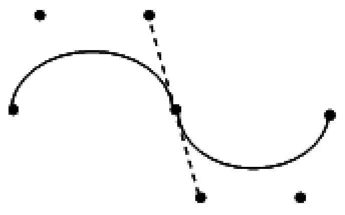


Figure 2.4 - A curve comprised of smaller Bezier curves

Notice that using Bezier curves the line no longer interpolates all of the data points, to do so would require higher order polynomials, which become computationally expensive to calculate, (Angel, 2003). The same can be said when drawing all curves, not just Bezier curves.

It is important to understand that a Bezier curve can be expressed in terms of a set of four blending functions, (Angel, 2003). Other literature uses the term “basis function”, but the term “blending function” will be used throughout this report. The first point that needs to be established is that the curve is evaluated over the distance between the start point and the end point, referred to as u. Weisstein (2005) points out that the blending functions for Bezier functions are the Bernstein polynomials, and explains the interesting properties that these functions show. Fig. 2.5 shows a plot of the Bernstein polynomials, adapted from Joy (1997), as used as the blending functions for Bezier curves.

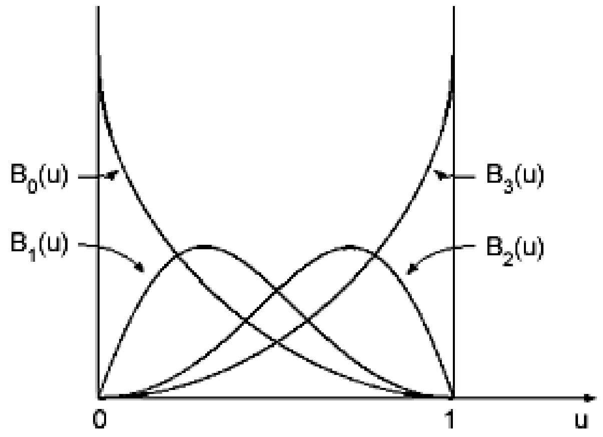


Figure 2.5 - The Bernstein polynomials used as the blending functions for Bezier curves

The curve is calculated by evaluating all of the blending functions at points in u. Each of the evaluated blending functions is then multiplied by the one of the four control points, p_i, associated with it. All of the results are added together to give the evaluated value of u, this is represented in the formula; fig. 2.6 shows the situation being evaluated.

$$\sum_{i=0}^{3} b_i(u)p_i = b_0(u)p_0 + b_1(u)p_1 + b_2(u)p_2 + b_3(u)p_3 \ ,$$

(Angel, 2003)

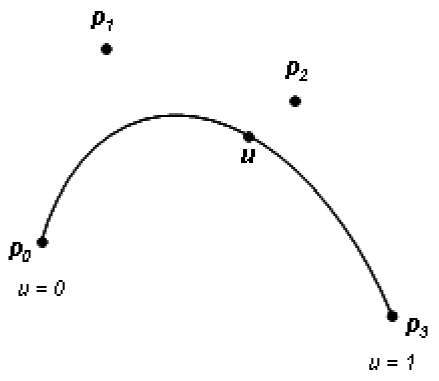


Figure 2.6 - An example of a Bezier curve

Blending functions can be used to represent many curves; one such curve is the B-spline which could be used as an improvement upon constructing a curve out of many smaller Bezier curves. Using a B-spline, none of the data points are interpolated, but they all have a degree of control over the shape of the curve in the area local to them. Fig. 2.7 shows how one segment of a B-spline curve is influenced by four control points, another way of looking at this is that each control point has influence over four segments. B-splines can be as higher order a curve as the developer wishes, the higher the order of the spline, the closer the curve will pass to the control points, but it will require more processing power to be calculated.

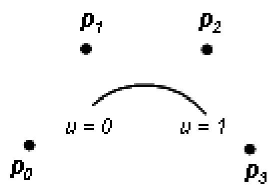


Figure 2.7 - A segment of a B-spline

The benefit of using a B-spline to represent a curve is that at the join points will have C^2 continuity, as demonstrated by Angel (2003).

The first point to consider is if the curve does not pass through any of the data points, then will it not interpolate the end points of a line? The answer to this is no, not when using the concept of knots; Watt (2000) demonstrates that when evaluating the blending functions, individual control points can be used more than once to give them a greater degree of influence over the curve. To pin a B-spline down to the end point, the end point must be used as the value for all of the control points when the blending functions are evaluated.

Another drawback of using B-splines is that they require the distance between consecutive pairs of points to remain constant at all times. Watt (2000) explains that this is because the same blending function is used for the whole curve, and each blending function used is a translate of the original. This makes B-splines unsuitable for this application as the distance between data points is relative to the speed at which the user moves the tracking device. An alternative is to use non-uniform B-splines that offer all of the benefits of the standard B-spline but do not require equally sized segments; as a result the blending functions for each segment have to be specified before they can be evaluated.

These blending functions can be constructed using the Cox-De Boor recursion method, as laid out by Angel (2003). Using the formula below, a set of blending functions can be determined for each segment of the curve.

$$B_{k0} = \begin{cases} 1, u_k \le u \le u_{k+1}; \\ 0, otherwise \end{cases}$$

$$B_{kd} = \frac{u - u_k}{u_{k+d} - u_k} B_{k,d-1}(u) + \frac{u_{k+d} - u}{u_{k+d+1} - u_{k+1}} B_{k+1,d-1}(u)$$

This computation will require a lot of processing power, especially if it needs to be rendered in real time. The process can be made more efficient by keeping the order of the curve low and by reducing the number of points to be evaluated.

The decision as to which method the system will use to draw the curves comes down to two choices: piecewise linear interpolation and non-uniform B-splines.

Using linear interpolation the curve would lack C^1 continuity, the result being that what was intended to be a smooth curve, will look like a line composed of smaller lines; this can be improved by reducing the size of the line segments resulting from increasing the number of points on a line. The benefit of using linear interpolation would be its simplicity to implement, and the fact that no processing power would be required, as there is no need to evaluate a function responsible for drawing the curve.

If the curves were to be drawn using a non-uniform B-spline then the result would be a single smooth curve, it would come at a cost; requiring many calculations this method may not be suited to drawing in real time, but it could be used to create a more aesthetically pleasing image once the user has finished editing it.

3. Design

This chapter is concerned with detailing how the system was to be implemented. A description of the classes from which the program was composed is given, along with details of how they were to interact with each other. This is followed by a description of the evolutionary design of the user interface, detailing the decisions made and the theories they were based upon.

3.1. Description of Classes

When using the object oriented programming paradigm it is important to understand what each of the classes is used for and how they interact with each other. Fig. 3.1 shows a class diagram that demonstrates how the classes related to each other. This is followed by a brief description of the classes and their methods.

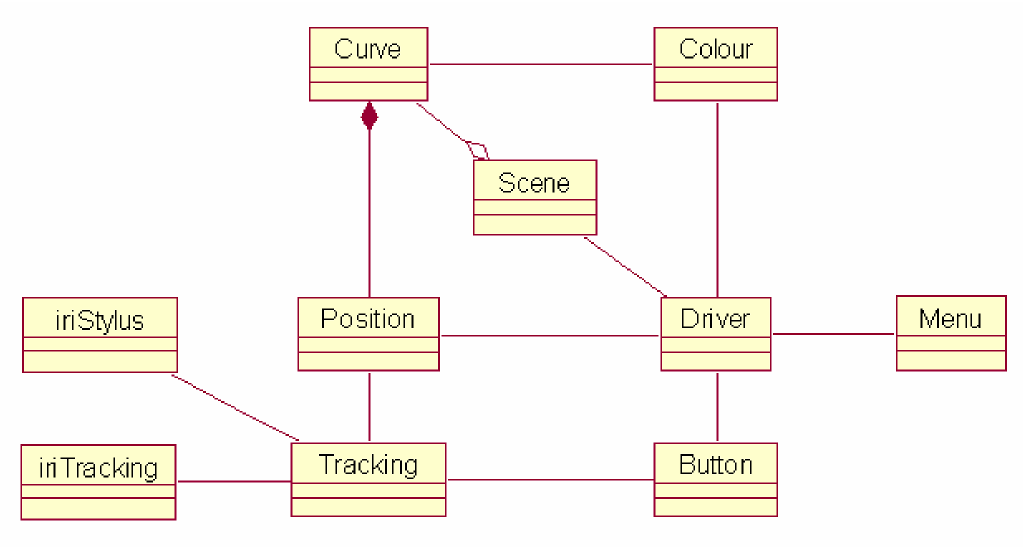


Figure 3.1 - A class diagram representing the modules of which the system is comprised

Curve

The Curve class stored all of the properties and data that comprised an individual curve in the scene. A line was drawn by interpolating a number of points read from the tracking device; these points were stored in a vector. In addition to the geometry of the line, the Curve object stored its colour and width. At the time when a curve was drawn, the offset of the scene along the x and the y axes was stored; this information was used when the program rendered the curve to make sure that all of the curves in the scene were correctly rotated relatively to each other.

Scene

The Scene class stored all of the curve objects that were drawn by the user in one drawing. The class only provided methods to add a new curve to the scene, and to delete the last curve drawn.

Colour

The Colour class was created to make the process of passing colour information between methods more efficient. The class stored the red, green, and blue components of a colour; the alpha attribute stored the opacity of the curve. It was decided that it would be simpler and tidier to program the application using one colour parameter instead of four.

Tracking

The Tracking class was created to read the tracking data from the input device and to monitor the state of the buttons. It was created to act as an interface between the system and Roy Ruddle's classes, iriStylus.c++ and iriTracker.c++, which were used to access the data from the devices. The class returns the positional data in terms of a Position object, and the state of each of the buttons and the joystick in terms of a Button object. This class was designed to improve the readability of the code by grouping all of the tracking device code together and creating methods that could return the tracking data using a higher level of abstraction.

Position

The Position class was a user defined data type that stored a position in 3D space using Cartesian coordinates. The class was composed of three attributes, used to store x, y, and z coordinates individually.

Button

The Button class was used to express the state of the buttons and the joystick on the input device using one object. It comprised of eight attributes, four of which stored the state of the buttons on the device; the other four specified whether or not the joystick was positioned in the up, down, left or right state.

Menu

The Menu class held all of the methods used to draw the UI. It also stored the methods responsible for loading the textures used for the buttons and messages. In order to maintain readability, it was decided that the code used to draw the interface should be held separately from the main workings of the system for the reason that defining the UI using OpenGL primitives required many lines of code.

Driver

The Driver class held most of the functionality offered by the system, such as the methods used to draw the lines, rotate the scene around the x and the y axes, and the methods responsible for loading, saving and exporting the scenes. The driver class was responsible for calling all of the methods from the other classes and passing the data between them. Most of the code was called from the display loop, including code used for displaying the curves drawn by the user, and all of the conditions based on cursor positions and button status that determined which part of the menu system was to be displayed. All of the OpenGL programming was contained in this class, such as the initialisation of the lighting and the other properties of the scene.

iriStylus and iriTracker

These classes existed prior to the implementation of the system, they were developed by Roy Ruddle to access data from the input devices.

3.2. Interface Design

This section discusses how the interface design developed through the three iterations of the project. It is important that a system has a good interface, making the user feel comfortable using the system, allowing them to concentrate fully on the task in hand, (Dix et al, 2004).
Scali et al (2003) explain why WIMPs are unsuitable to use in a 3D environment, requiring developers to invent new interaction techniques. Bowman et al (2004) recommend a trial and error approach, due to the lack of formalised design guidelines for 3D UI's. When a new feature was added to the interface its usability was evaluated, and a judgement passed on whether it could be improved. If improvements could be made, then research was conducted as to how similar problems had been solved in the past.

3.2.1. Iteration One

The first iteration was concerned with developing a system that would allow a user to draw lines in 2D space, using a desktop configuration. At this early stage the interface was designed to work using a two DOF input device with a progression to using a six DOF input device kept in mind. Fig. 3.2 shows the interface upon the completion of this iteration.

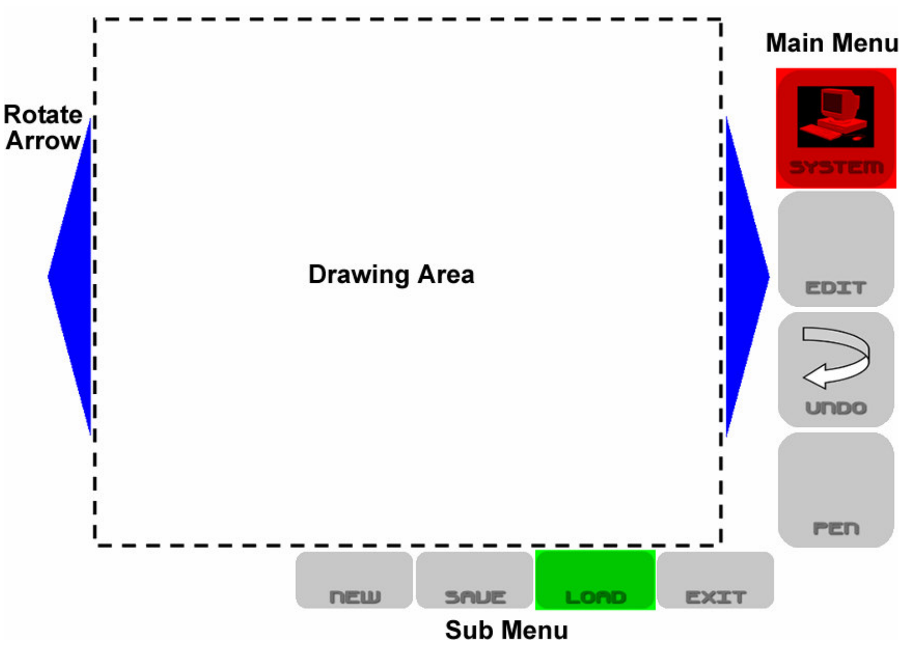


Figure 3.2 - The interface upon completion of the first RAD iteration

To improve the action of option selection, the buttons were designed with Fitts' Law in mind, which principally states that big targets are acquired faster than small targets, accordingly they were made to be big and bold. They were positioned to the side and the bottom of the screen because the edges are, "infinitely targetable because of the boundary created by the edges of the screen," (Zhao, 2002), another benefit of this placement was that they did not occlude the drawing area. The interface used the concept of rollover buttons giving feedback as to when the user had positioned the cursor over the button to be selected. The device button was pressed to select the icon, once pressed the icon turned red, giving extra feedback that the user had completed their action.

Mine et al (1997), suggest that the menu's occlusion of the display can be minimised by attaching it to a device button, with that in mind UI was attached to the right mouse button, allowing the user to press it once to show the menu, and to press it again to hide it. At this stage, when the menu was visible the drawing area was shrunk and translated to the top left, so that the entire scene remained visible at all times.

The arrows on either edge of the screen controlled the rotation of the scene, to rotate the scene the user hovered the cursor over the arrow pointing in the appropriate direction. The reasoning behind the arrows was that when the system evolved to using 3D input, they would be easy for the user to find, exploiting the phenomenon of proprioception, “a person’s sense of the position and orientation of his body and limbs,” (Mine at al, 1997). It was thought that users would find it intuitive to hover their hand to their side to rotate the scene in that direction. Initially this feature did not work well, it was found that the user positioned the cursor over the arrow to get the rotation they required, they then had to move the cursor away from over the arrow, resulting in more rotation. It was clear that the calling the scene rotation function would have to be controlled via a button on the input device.

The ‘edit colour’ menu consisted of nine boxes filled with nine colours, giving the user a very small set of colours to choose from. Even at this early stage it was evident that there should be a wider range of colours available, meaning that this option would need to be redesigned.

3.2.2. Iteration Two

Iteration two was concerned with converting this 2D implementation to a 3D implementation making use of the six DOF Input device. The output device used was a Holobench, consisting of two displays set orthogonally to each other, only the back display was used for this application. Each of these screens was surrounded by a wooden frame; these frames were used as a point of reference for the 3D interface. The interface was fitted to the Holobench so that each edge of the virtual workspace was aligned to the corresponding edge of the horizontal display, with the virtual representation being shown on the back display. Fig. 3.3 shows the UI upon the completion of this iteration, the rotate arrows have been omitted from the plan view so not to occlude the rest of the interface.

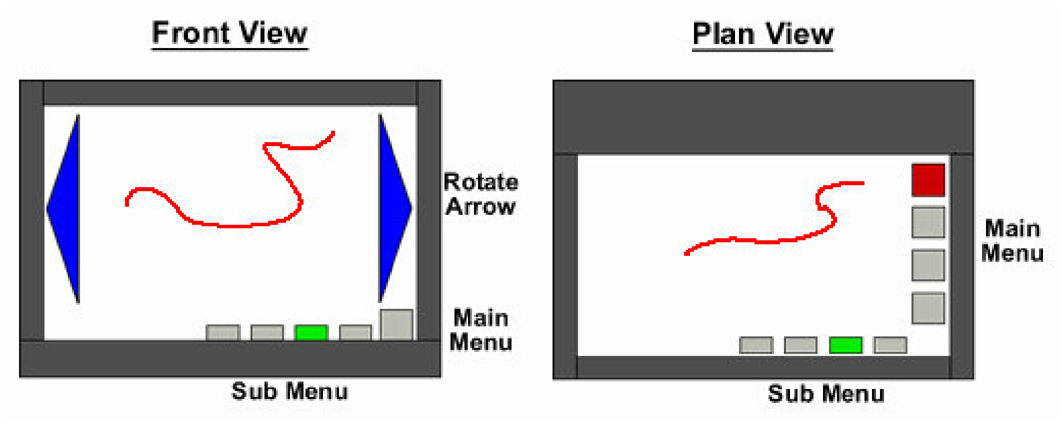


Figure 3.3 - The interface upon completion of the second RAD iteration

The major difference between developing an interface in 2D, and developing an interface in 3D was the amount of extra space available to work with. It was decided that to make use of this extra space the menu system would be visible at all times. After the first iteration the main menu was displayed up the y-axis, it was now positioned along the z-axis instead, without intruding on the drawing space. This change also dictated that the 2D icons became 3D boxes, making it possible for the user to select them with the cursor.

The transition from 2D to 3D required thought as to how to deal with the issue of selection of objects. Mine (1995) classifies 3D selection techniques under two headings: 'local' where, "objects are chosen for selection by moving a cursor (typically attached to the user's hand) until it is within the objects selection region"; or 'at-a-distance' whereby, "the selection of objects that fall outside of the immediate reach of the user can be accomplished with laser beams or spotlights which project out from the user's hand and intersect with the objects in the virtual world." In both cases an object should be finally selected by the user confirming the action with a pre-defined gesture or button press. Bowman et al (2004) provide a more detailed explanation of these techniques; because all of the selection needed by this application takes place within arms reach, the 'simple virtual hand' method was chosen.

Bowman et al (2004) recommend that when designing a system control menu, that it uses "an appropriate spatial reference theme," in an effort to follow this principle the interface was designed so that the menu options were aligned with the frames of the horizontal display and were virtually 'sat upon' it. The intention was to give the user a point of reference to help find the area where the menu options were situated, and to prevent the user from moving the cursor underneath the icons. The user had to position the device where they would expect the icon to be in real world space, they received feedback from the position of the cursor on the display, but new users found it difficult to cope with. The changing colours of the icons representing their state helped to make the system more usable, giving the user extra feedback as to when the cursor was over their intended target. Another benefit of the 'rollover' icons was that they allowed the user to know that the cursor was still positioned over the icon, even when the cursor became obscured by the icon itself.

Mine (1997) highlights the reasons as to why precise manipulation is so difficult in a virtual environment; one such reason is that without haptic feedback (the sensation of touching an object) the user is not aware that they are holding an object or not. Another is owing to the limited precision offered by the input devices, using a whole handed input device means that the user has to interact through a "boxing glove style interface," Zhai (1996). Zhai demonstrated that users take longer to complete a 3D docking task when the input device excludes the use of fingers. To help the user with the process of selection, the menu option boxes were made as large as possible, the gap between the boxes was designed to be large enough that the user could place the cursor in between them, reducing the risk of accidentally choosing the wrong option, but small enough that the user did not have to move the cursor too far between choices.

A drawback of using the Holobench as a spatial reference to the boundary of the scene was that at times the user felt that they were stretching to make the most of the available space, where they would have preferred to have been working within arm’s reach. This problem carried through to selection of menu items, the worst scenario involved selecting the ‘system menu’ icon which was located close to the back display of the Holobench due to size and distribution of the menu option icons; taller users had to stretch to reach it, whereas shorter users could not reach it at all.

The rotate arrows worked well using the Holobench, the outline of the work space was clearly in line with the edges of the displays and users quickly understood that to find the arrow they simply had to hover the cursor above the edge of the horizontal display. Because they covered the whole depth, and most of the height of the virtual scene users found them very easy to find with the device. The drawback of them being so easy to find was that when users drew too close to the edge of the scene they would inadvertently move the cursor into one of the arrows thus causing an unwanted rotation of the scene.

3.2.3. Iteration Three

Having altered the system to use the six DOF input device, the focus of the project was to improve the system’s usability, and to add some extra functionality. Just after starting this iteration the IS-900 tracking equipment developed a hardware fault requiring that the ‘Flock of Birds’ was used. The change of equipment meant that the project was no longer constrained to using the Holobench; a decision was made to move the system to a better quality display consisting of a single vertical screen. As a result the interface developed so far became unusable without the Holobench to act as a reference.

The first problem to be addressed was how the user would be positioned relative to the interface, using the Holobench it was clear that the edge of the scene was aligned to the edge of the display, but there was no longer anything to act as point of reference. An advantage of designing the system to work on the Holobench was that everything was almost within arms reach, so the position of the interface problem was solved by allowing the user to calibrate the tracking device to their own point of reference (the centre of the scene) before they began. It was assumed, based on proprioception, the user would realise that if they moved the cursor down and to the right, the cursor would be situated near to where the 'virtual icons' were located.

Even with this new point of reference, the menu system failed to work well; new users found the icon boxes difficult to locate, and without a horizontal surface to 'sit' the boxes on users had to make a conscious effort to prevent themselves moving the cursor out from underneath them.

Bowman et al (2004) suggest using an 'adapted 2D menu', whereby the menu is displayed as 2D icons positioned at the front of the scene. Adapted 2D menus improve usability because of their similarity to a desktop Window Icon Menu Pointer (WIMP) interface, "almost all users will instantly recognise these elements as menus and will understand how to use them." After implementing this approach the menu was attached to a device button, to view the menu the user clicked the appropriate button, pressing it again hid the menu. Selection of these icons was made easier by restricting the cursor to only two DOF whenever the menu was visible. This change dramatically improved the system's usability; users felt more comfortable selecting options from a style of interface familiar to them, allowing them to complete tasks quickly allowing them to concentrate more on drawing.

To maximise the adapted 2D menu's usability, the buttons were designed to be extremely big, based on Fitts' Law. It had been noted that the users tended to keep the cursor around the area that they set as the point of reference, to exploit this the buttons were positioned just above and below the centre of the scene, meaning that on average users would only have to move a small distance, if at all, to acquire their target, (Zhao, 2002). The use of the adapted 2D menus came at the cost of the occlusion of the screen, but it was decided that this approach made the menu system easier to use, requiring less thought on the part of the user, allowing them to return to the process of drawing more quickly.

The 'undo' feature that originally was called via the menu was attached to a button on the input device. Prior iterations had shown that due to the difficult nature of drawing in 3D, users made many mistakes that they wanted to undo, therefore it made sense to move this function to a button on the device rather than have the user navigate through the menu system to complete this task each time a mistake was made.

Feedback from earlier prototypes indicated that users wanted to be able to rotate the scene around the x-axis as well as the y-axis already offered. Initially another set of rotate arrows was added to the top and the bottom of the screen, mimicking the behaviour of the arrows responsible for rotating the scene around the y-axis. In practice the functionality worked, but the result was that the display seemed to be dominated by these arrows.

The first solution was to make the arrows appear as 2D icons at the front of the scene, but still allow the user to press the button whilst the cursor was positioned anywhere behind them to rotate the scene. The result was that the system became unusable; it became impossible to tell whether or not the cursor was within the area occupied by these arrows, and users regularly mistakenly moved the device into one of these areas causing the scene to rotate.

The problem was fixed by abandoning the rotate arrows completely and moving the rotate functionality to the joystick situated on the top of the input device. This joystick gave input signals for when it was either in the up, down, left or right positions, making it the perfect replacement for the arrows. The changes made the system far easier to use, with the rotate arrows removed from each side the display became less cluttered, giving a sense of a much larger space to draw within. It also meant that users could devote full concentration to drawing rather than to avoiding the rotate arrows.

One of Nielson's (1994) ten usability heuristics states that effort should be made to "minimize the user's memory load". At this stage all but one of the buttons on the input device were attached to a function, consequently users found it difficult to remember which button controlled what function. A solution was provided by annotating a diagram of the stylus and attaching it to the remaining button. This gave the users a quick reference to find out what button they had to press to complete the task. Two diagrams were drawn, one for when the system was in drawing mode, the other for when the system was displaying the menu, due to the fact that buttons behaved differently depending on the state of the system.

It was decided earlier that the 'edit colour' option needed redesigning to give the user a greater choice of colours, so a palette was designed that allowed the user to choose from a greater range of colours along a continuous scale, a second similar palette was added to give the choice of greyscale. This palette was displayed in a similar way to the menu system, at the front of the scene with the cursor restricted to only two DOF. The tone of the colour palette only changed along its x-axis so it was decided that the selection of colour should be restricted to only one DOF. This was implemented by changing the DOF once the user had positioned the cursor inside of the colour palette, at which point the cursor was replaced by a pointer controlled by the tracking device's x position, the benefits of this approach are highlighted by Shaw and Green (1994). This data controlled the position of a pointer that acted like a slider bar along the length of the palette; once the user reached the colour desired, they pressed the button to select the colour and then selected the 'OK' icon on the interface to confirm the choice. When the user selected a colour, the colour of the pointer changed to the selected colour to give feedback that a different colour had been chosen, if the user was not happy they could reselect the colour as many times as they liked before confirming their selection.

The 'Load' and 'Save' options required the input of text, due to the semi-immersive nature of this application it was decided that symbolic input should be entered through the keyboard.

4. Implementation

This chapter concentrates on explaining the process by which the system was implemented; using the RAD methodology the implementation was divided into three iterations. Throughout the process, each iteration raised different problems that needed solving, an explanation of each problem and its solution is provided. Any programming techniques were used are explained to help give a better understanding of the system. The chapter concludes with an outline of the testing procedure and details of any issues that arose from it. A Photograph showing the end product in use can be found in Appendix E.

4.1. Iteration One

4.1.1. Drawing a Curve

The first issue to be addressed when creating this system was to establish how the lines were going to be stored. The data had to read from the input device and stored in a dynamically sized data structure; C++ offers the Standard Template Library (STL) which offers a number of different data structures that could have been used. From the available options the vector container was chosen because: a) it stored the data sequentially so that when the data was read, it was stored in the order it was written; b) it allowed rapid insertions making the process of storing a line as efficient as possible, (Deitel and Deitel, 2001).

A class called Curve was created to store all of the data needed to plot the curve and the other attributes associated with it. A collection of Curve objects were stored in a class called Scene. The Scene class was essentially another vector, but a class was used so that methods could be written to allow the manipulation of its data.

A curve was drawn by the user holding down the left mouse button and dragging the cursor to form the shape of the line. It was decided that the system did not need to store the cursor's position for every display loop, but every five. Another vector was used to store the points of the curve as it was drawn, allowing the system to render the curve in real time. The process of reading in a line's data is shown in pseudo-code below.

```
If button pressed
    Create temporary curve object
    Read curve properties from system and store in temporary curve
    Add device position to temporary curve
    Add device position to current curve vector

While button held
    Every five display loops
        Add device position to temporary curve
        Add device position to current curve vector

If button released
    Add device position to temporary curve
    Copy temporary curve to the Scene object
    Delete temporary curve
```

The system rendered the scene by iterating through the all of the Curve objects stored in the Scene object. For each curve, variables had to be set to the draw the properties of that curve, then the coordinates that comprised the curve were read sequentially and a straight line drawn between each pair of successive points. Due to its simplicity to implement piecewise linear interpolation was chosen to represent the curves. Once all of the curves stored in the Scene object had been rendered, if there was data stored in the `currentCurve` vector then this was used to render the curve being drawn in real time. Fig. 4.1 shows a sequence diagram representing the messages past between the classes from when a line is drawn to when it is displayed.

4.1.2. Event Driven Programming

Programming graphics applications requires an event driven approach; using OpenGL the program loops round every time it is determined that the display should be redrawn, the program checks for changes in the state of the devices and acts accordingly, (Angel, 2002). There are some flags that determine the system's state, its behaviour is based upon these flags, three of which are detailed below.

cursorOver

Each button on the UI was assigned a constant value, whenever the user positioned the cursor over a button the variable `cursorOver` was assigned that constant value. When these buttons were rendered the texture used depended on whether or not the cursor was hovering above them. This variable was responsible for the rollover buttons when selecting the menu options.

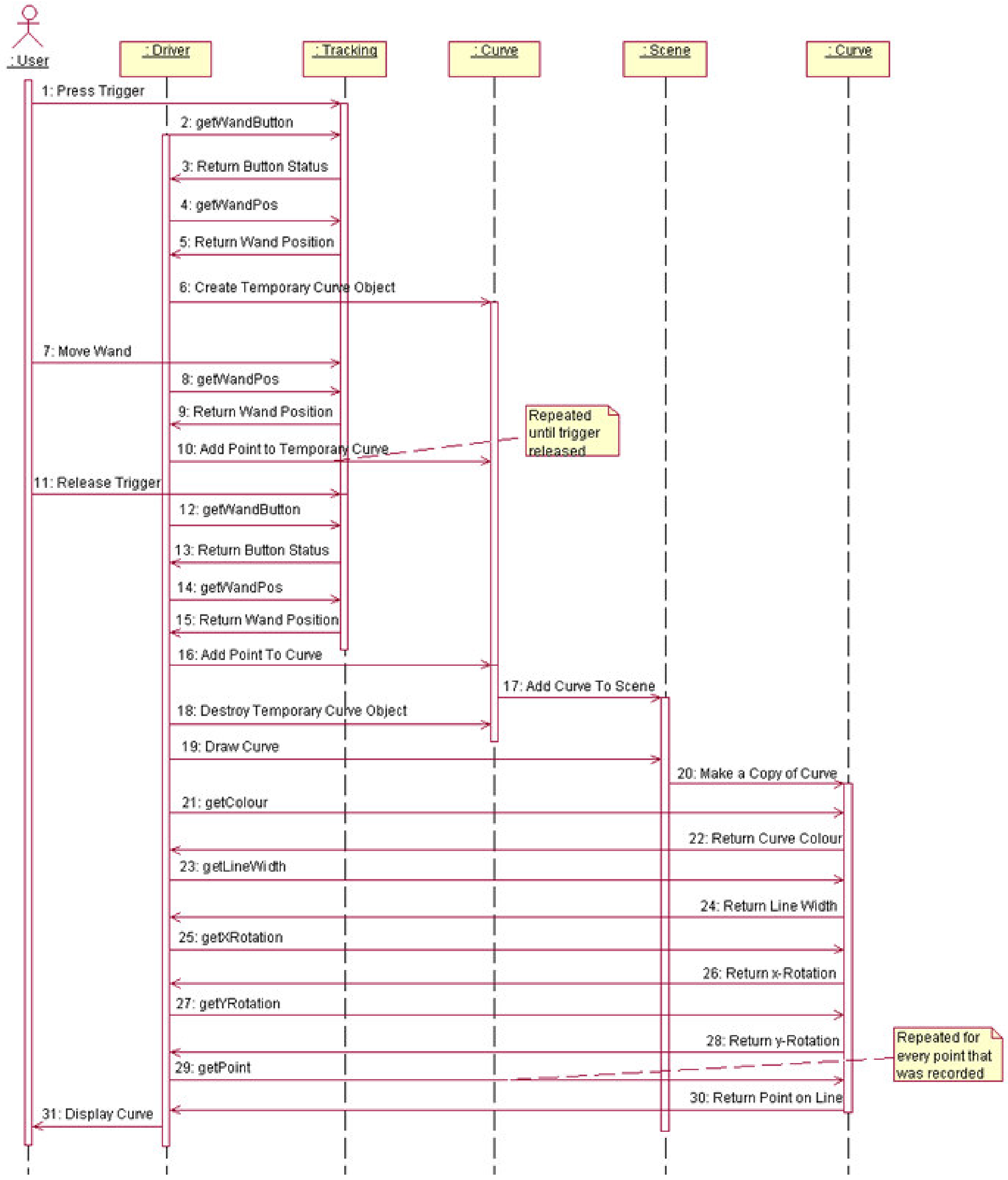


Figure 4.1 - A state diagram showing the messages past between the classes when a curve is drawn

menuDisplayed

There were a number of different menus contained within the menu hierarchy; each was assigned a different constant value. When the user selected an option from a menu, the `menuDisplayed` variable was assigned to the value associated with the user's choice. The `display` function contained a number of conditions based on the value of `menuDisplayed` that called the appropriate function from the Menu class. If `menuDisplayed` was assigned the value ‘Nothing’, then no method from the Menu class was displayed, thus no menu was displayed.

penMode

When a menu was displayed the user selected an option by hovering over it and pressing the trigger, the same button that was pressed when drawing a line. It was important that when selecting an option from the menu the system did not add extra marks to the drawing. This was resolved by adding the penMode flag; when the trigger was pressed a new curve object was only created if the penMode flag was set to true, when the menu was displayed `penMode` was set to false, preventing any drawing. It was decided that there should be a separate variable for this action when `menuDisplayed` could have been used, to leave the option of implementing an object selection function that would have used the trigger.

4.2. Iteration Two

4.2.1. Making the Program Run

The first stage of changing the system to run using the 3D input devices was to the make the system compile and run on the Onyx 3400. John Hodrien was consulted to explain where the OpenGL libraries were located on the IRIX system, and helped to create a makefile to compile and link all of the components of the system together.

Once the system got to a stage where it compiled, it was unable to run due to errors whilst reading the texture bitmap files; the cause was the issue of ‘endianess’ “which refers to the order in which bytes in multi-byte numbers should be stored,” Cobas (2003) provides a detailed description of the problem. The difficulty arose because the 2D implementation was written and compiled on a Microsoft Windows machine that used the Intel x86 architecture which followed the ‘Little-Endian’ convention, whereas the Onyx 3400 used the MIPS architecture, which followed the ‘Big-Endian’ convention. The problem was fixed by a process called byte swapping, whereby the order in which the bytes that make up the number is reversed.

4.2.2. Integration of Devices

The next step was to integrate the tracking devices to work with the system. Roy Ruddle provided an example program which used the devices, the source code for this was closely examined and was used to create the Tracking class that was responsible for reading the data from the devices and passing the on to the application.

The rest of this iteration was concerned with adjusting the interface to work using 3D tracking data, the details of which can be found in section 3.2.2.

4.3. Iteration Three

4.3.1. Switching Tracking Hardware

Just as this iteration was beginning the IS-900 equipment developed a fault that made it unusable. At this point a decision had to be made as to which direction the project would now progress. There were two options: the first was to revert back to the desktop version and attempt to develop a 3D drawing system based on that; the second was to take a step backwards and to make the system work with the 'Flock of Birds' tracking equipment, meaning that much of the work carried out in the previous iteration would have to be repeated. It was decided that the input of 3D data was crucial to the project and so it was decided that the system would use the 'Flock of Birds' tracking hardware. The transition was not particularly smooth with remnants of the IS-900 code causing initial problems; once these matters were resolved it was clear that the right decision had been made allowing the project to carry on from where it had left off.

Once the system had been moved from the Holobench to the CRT projector, the tracking equipment began to return incorrect data. The reason behind this was the change in the position of the user relative to the transmitter. Both of the tracking devices operated using the concept of hemispheres to convert the data from the device to the co-ordinate system used by the application, (Ruddle and Holbrey, 2004). The centre of all of the hemispheres was the transmitter; the choice of hemisphere dictated the direction of the axes of the coordinate system relative to it, the hemisphere to be used was defined the file workspace_example.cfg, (Ascension, 2002). Using the Holobench the device transmitters were fixed to the display and used the 'Forward' hemisphere. When Roy Ruddle's IS-900 program was adapted, no thought was given to how the hemisphere used would make a difference to the data. However, using the CRT projector the transmitter was located above the user's head, as a result the 'Lower' hemisphere needed to be used. This simple change rectified the problem.

4.3.2. The use of latches

Because the system ran in a loop, problems arose with the processing of events when they were attached to the device buttons. The example of the 'show menu' button will be used here, but the same fault occurred with each of the device buttons. It is important to remember that the same button responsible for both 'showing' and 'hiding' the menu. Whenever the user pressed the 'show menu' button, the next time the program looped round the menu was displayed; however due to speed at which the display was refreshing, when the next loop was processed the user would still be holding the ‘show menu’ button. Because the menu was already shown, the function of the button was in fact to ‘hide’ the menu. The result was, whenever the user pressed the ‘show menu’ button, the menu would flicker on the display and there was a 50% chance that the menu would be displayed when the user released the button.

The problem was fixed by using a programming technique called a latch, which works on the same principle as a latch on the door. When a door is shut the spring in the latch is released to keep it shut. Each button on the device had a latch attached to it, when a button was pressed, the function attached to that button was completed, and then a flag was raised to show that the function had been completed. Every call to a function attached to a device button was preceded by a condition that checked for the state of the latch. The function was only executed if the flag had not been raised, this made sure that every function was only called once. When the device button was released the flag was set to its original state allowing the function to be called again.

4.3.3. Displaying in Stereo

OpenGL uses the process of double buffering to produce a smooth animated output, by drawing to the back buffer, whilst displaying the front buffer. Once the drawing to the back buffer is completed, the contents of the back buffer are moved to the front buffer to be displayed, whilst the next frame is drawn, (Angel, 2003). When the system displayed in stereo, OpenGL completed this process once for each eye, the scenes were drawn to the `GL_BACK_RIGHT` and `GL_BACK_LEFT` buffers. The difference between the images for the left and right eyes was a translation along the x-axis of half of the interocular distance (IOD). The IOD chosen for this system was 65mm, as used by Crvarich (1995). Once the image was translated to the position of each eye, the scene had to be rotated slightly so that although each image was slightly different the eyes still focused on the same point. McReynolds and Blythe (1997) give details as to how this was implemented in OpenGL.

The stereo output was fixed to look at one point, from a defined viewpoint, as a result when the user moved his head the image appeared to move around. Ideally the stereo output would be controlled by the position and orientation of the user’s head through the use of a head tracking device. With this extra data the image displayed can be translated so that the 3D image represented is correct from any position with the user looking in any direction. Foley et al (1997) outlines how to implement a head tracked stereo display, and the mathematics behind it. The simpler stereovision was used because there were problems getting the ‘Flock of Birds’ head tracking to work properly, but it was decided that implementing simple stereo output would be better than using no stereo output at all.

4.3.4. Rotating the Scene

Allowing the user to rotate the scene and to continue editing with the scene remaining in the position that the user left it required some thought. It was simple enough to rotate all of the curves in the scene, but when the user tried to draw a new curve, it was not drawn from the cursor position, but from a rotation of the cursor’s position. The problem was fixed by rotating the scene as a whole, but then rotating each individual curve relative to the scene back in the opposite direction.

Whenever the user called a function that rotated the scene around an axis, a variable kept track of the scene’s angle of rotation. When a new curve was drawn by the user, the current rotation of the scene was added to its properties. When the system rendered a scene it was rotated by the value of the scene’s rotation, then as each curve was rendered, they were rotated in the opposite direction by the angle that the scene was positioned in when they were drawn. This ensured that all of the scene’s curves were rotated correctly relative to each other.

4.3.5. Collision Detection

All selection within the system was based on simple collision detection. If the user positioned the cursor so that one edge of the cursor overlapped an edge of the button to be selected, then in the eyes of the system, the cursor is hovering over the button. For there to be a collision, the cursor had to overlap the button on the x and y axes. Fig. 4.2 shows how the collision detection algorithm worked.

Every time the program looped around the collision detection algorithm was called, so an effort was made to reduce the computation required when processing these collisions. When designing the UI, all of the buttons were aligned to improve the aesthetics; an advantage of this was that all of the buttons x-axis boundaries were aligned. This meant that when testing for collision detection, if the condition for testing the position of the cursor along x-axis was met for one of the buttons, then it was known that it would be met for all of the buttons, so the condition based upon the cursor’s x position only needed to be tested once, instead of for each individual button.

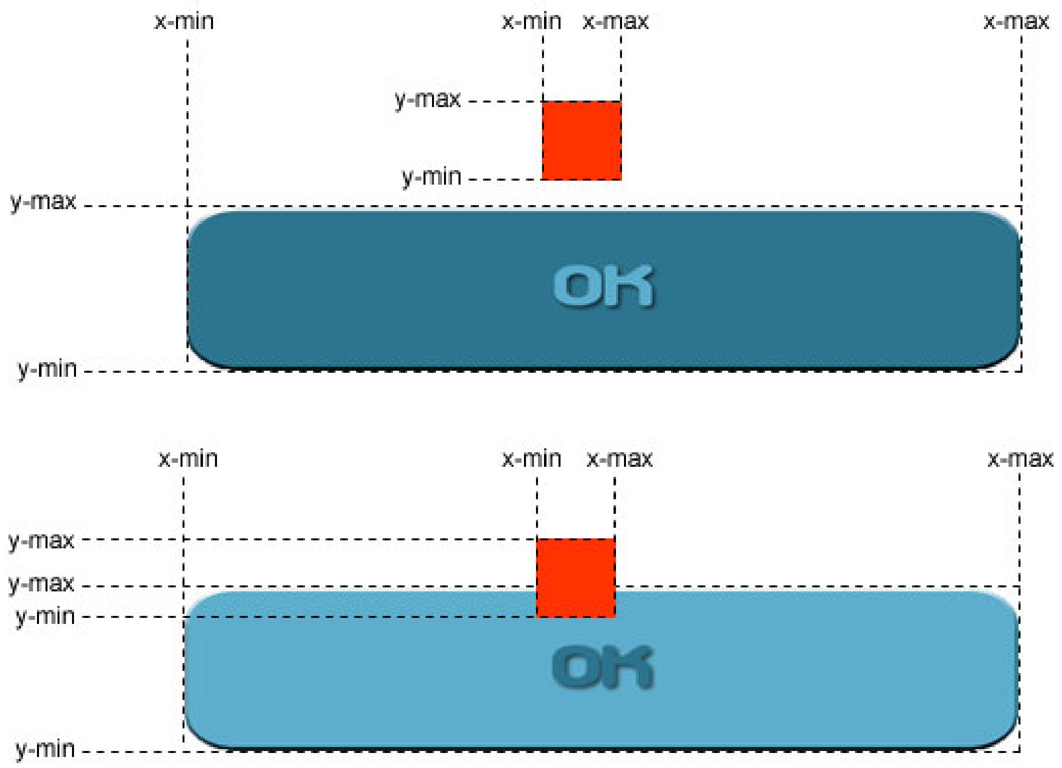


Figure 4.2 - A graphical representation of the algorithm used to check if the cursor was positioned over a button

Although collision detection was originally implemented in the second iteration, the bulk of the algorithm was implemented and refined during the third iteration.

4.3.6. The Selection of Colour

To increase the user's choice when selecting colour, it was decided that a sliding scale should be used. Fig. 4.3 shows the colour bar and the available choice, the red, green and blue (RGB) values are shown below. This was drawn using OpenGL's `GL_QUADS` command, with different colours assigned to each side of the rectangle.

The transition of the colour was automatically calculated by OpenGL when the object was rendered. It is important to note that only one component of the RGB measure changed inside each rectangle. To select a colour, the user positioned the cursor inside the colour bar, at which point only the position along the x-axis was measured. When the user pressed the trigger on the device, collision detection was used to establish which colour area the cursor was in, the value of the variable component was calculated by determining the x position of the cursor relative to the outline of rectangle.

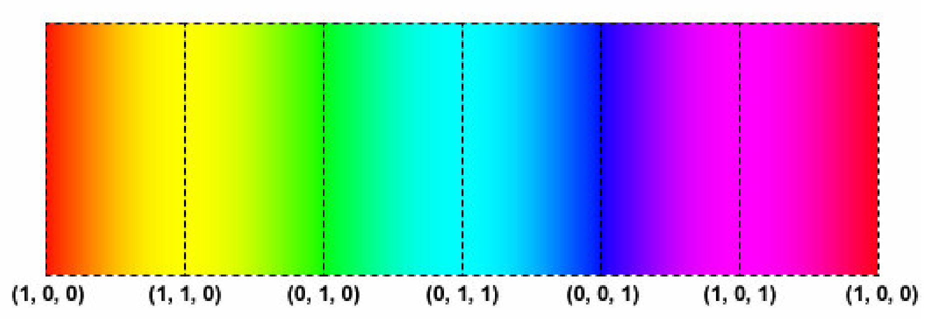


Figure 4.3 - The choice of colour available to the user, the dashed lines represent the polygons used to draw the colour bar

4.3.7. Command Line Arguments

When developing the system it was necessary to test new features under different sets of conditions; originally these conditions were altered by editing the source code. Rather than editing the source code it was decided that it would be more efficient to set changes by means of command line arguments. Below the two command line arguments are given along with the reasons as to why they were implemented.

-f	Controlled whether or not the program display occupied the full screen. It was preferable to test the system using the full screen output, but when a problem was encountered, diagnostic messages were used, requiring that the terminal be visible.
-s	Controlled whether the display used stereo output. When a feature was developed mono output would be used to make sure that it worked as expected, saving the need to have to wear the stereo glasses.

4.3.8. Writing to Files

When the 'Load' and 'Save' functions were implemented the first approach used was to overload the C++ operators '<<' and '>>' for use with the different classes, as outlined by Deitel and Deitel (2001). This approach did not work because the compiler installed on the IRIX system was outdated, and did not recognise that the functions could be overloaded for user defined classes. To get around this, the functions had to implemented using C; having had no previous experience of programming in C, Jamsa (1998) was consulted, and the problem was resolved.

4.3.9. Exporting to VRML

A requirement that emerged in this iteration was that the system should be able to export the scenes drawn by the user into the VRML 2.0 format allowing them to be viewed using a web browser. Without any previous experience of using VRML before, Hartman and Wernecke (1996) was referred to in order to gain a basic understanding. For the purposes that this system needed to use the VRML language, modelling was essentially the same as modelling in OpenGL. A line was drawn by specifying the coordinates of which it was composed and the browser rendered the lines between them. Unfortunately VRML 2.0 did not offer support for varying the thickness of lines, as a result when a user drew a scene and then exported it, variations in line width were not shown.

4.4. Testing

Upon the completion of the implementation the system was tested thoroughly to discover if there were scenarios where the system did not work as expected. A test plan was drawn up to test all of the functionality offered by the system; this consisted of navigating to each option of the UI and trying to make the system fail. Due to the simple nature of the system and the fact that users could primarily only input data through the tracking device it was impossible to test the data for extreme values. The user could enter data through the keyboard when the system read from and wrote to files, the input of extreme data was tested for these situations. Appendix F shows the results from the test plan.

A few minor bugs were discovered during the testing, all were fixed bar one. The main problem arose whilst navigating the interface and pressing buttons on the device that should have been disabled. The most notable example occurred when the system displayed the device calibration page; if the user pressed a button other than the device trigger, then the system would display the appropriate menu option behind it. Eventually when the user pressed the trigger, the menu appeared rather than the blank canvas that should have been displayed. It was important to fix this error, as it was very confusing for new users who often pressed the wrong button on the device at this early stage.

Another problem highlighted by the testing was the input of filenames; when the user typed in an incorrect filename, the string input was not deleted by the system when the disk access failure occurred. Consequently any subsequent attempts to access a file would fail because the new string was appended to the incorrect string. This was easily fixed by clearing the string after failed file access attempts. Also when the user tried to delete the filename entered, there was no limit to how many characters they could delete; this led to the deletion of the file path, causing the subsequent file accesses to fail. If the user deleted the entire path then the system would crash due to trying to delete a character from an already empty string. These problems were fixed by restricting the deletion of characters to only those entered by the user.

The only bug that could not be fixed was the system's failure to display the text entered by the user. Earlier on in the implementation the system displayed the text as expected, but at some point this problem arose. The reason behind it could not be ascertained; as a result the problem could not be fixed.

5. Evaluation

When evaluating a system it is important to do so from a number of viewpoints as a system may be strong in some areas but weak in others. Firstly the success of the system shall be evaluated based upon the minimum requirements set out at the start. Next, the usability of the system shall be evaluated, this shall comprise of a heuristic evaluation, an explanation of the formative evaluation that was used throughout the UI design, and the feedback from the usability testing. The end user, Claude Heath, came to evaluate the system over the 11th and 12th April 2005; the details of his extended evaluation are detailed below. It is also worth evaluating the 3D drawing experience, the success of the project could be measured by how the user dealt with drawing in 3D space, and whether the system helped or hindered the process. Finally the physical effects of using the system were evaluated.

A simple criterion by which to evaluate the system was the end user's satisfaction regarding the system. An extract from the feedback given by Claude Heath is given below, the complete feedback from the evaluation can be found in Appendix G.

> *"Working with the three-dimensional freehand drawing system that Ben Hammett has designed has been a thoroughly rewarding experience for myself. As a practicing artist whose interest lies in the field of drawing and finding new ways of describing space, this software, which I was lucky enough to be given the chance to work with, has opened doors for me by providing new insights into the recording of space as it is experienced in real time."*

Over the course of the evaluation phase, a number of other people from outside of the department expressed an interest in using the system. David Walker Barker, a lecturer from the department of Contemporary Art Practice used the system briefly, the feedback given was extremely positive. He was particularly struck by the speed with which he was able to learn how to use the system, and the control that he had whilst drawing. The system was also demonstrated to the Spatiality in Design research cluster, one of 21 clusters funded by AHRB/EPSRC under the Designing for the 21st Century Programme, (EPSRC, 2004); again the feedback was positive, with members of the group showing an interest into the possibilities of how the concept of drawing in 3D could be extended for use into other areas of recreation and design.

5.1. Evaluation Against Minimum Requirements

A criterion by which the system was evaluated was to establish whether the minimum requirements specified in section 2.1 were fulfilled. Below each of the requirements are specified followed by a statement of how they were met.

- *To allow the user draw a freehand curve in 3D space* – Upon the completion of the second iteration of implementation, the system was capable of reading data from the tracking hardware and constructing a line based upon the data.
- *To allow the user to save a scene that had been drawn* – The ability to save a scene that had been drawn was implemented during the third iteration of the implementation.
- *To allow the user to load a scene that had been drawn in a previous session* – The function responsible for loading a previously saved scene was implemented during the third iteration of the implementation.
- *The system should have a simple, intuitive interface* – Section 3.2 outlines the evolution of the design of the UI. The evaluation of the interface's usability is discussed throughout section 5.2.

All of the minimum requirements stated at the start of the project, examples of how these minimum requirements were exceeded are given throughout section 4.

5.2. Usability Evaluation

Dix et al (2004) outline a number of methods for evaluating a system's usability, each having their individual strengths and weaknesses. For this reason the usability evaluation comprised of three of the methods described, so that each method's area of weakness was mitigated by another method's area of strength.

5.2.1. Heuristic Evaluation

A heuristic evaluation is a method that does not involve the users, whereby developers evaluate the system based on a set of ten heuristics laid out by Nielsen (1994). Bowman et al (2004) point out that these guidelines were designed to evaluate 2D UIs, and so are not entirely appropriate for evaluating 3D UI's. Below are the results of the evaluation based on Nielsen's heuristics, followed by a reflection on their appropriateness. These heuristics are intended to be of a generic nature; when evaluating against them, an effort was made to clarify how they applied to this system. To finish, the system was evaluated against some proposed heuristics for evaluating 3D user interfaces.

Simple and Natural Dialogue
The system rarely entered into dialogue with the user, though when it did, the messages used were standard regarding the actions they were related to e.g. the loading and saving of files. The only situation where the system displayed two consecutive messages occurred when the user tried to begin a new scene at which point they were prompted to save the existing scene. If the user wished to save the existing scene then the save dialogue was shown. These messages appeared in a logical order, matched to the order of the task. At all times the information given in these messages was relevant to the task ahead.

Speak the user's language
On the rare occasions when the user entered into dialogue, the language used was kept simple. The first dialogue box, prompting the tracker calibration caused most confusion; not because the language was complicated, rather that users were unfamiliar with the process so they were unsure as to what they had to do, and the reasons behind it. With that exception the system used understandable language throughout.

Minimize the user's memory load
Because the majority of data was input through the tracking device scenarios that tested user's memory were infrequent. The one situation where the user was required to remember information occurred whilst trying to load a file; the system failed to provide a list of available filenames, requiring that the user remember the name of the file they wished to load.

Consistency
The system communicated with the user using the same language throughout, and followed a standard menu layout. The menu system used the same colour scheme throughout to avoid possible confusion regarding different colourings having different meanings.

Visibility of System Status
The system worked on the principle that if a task had completed successfully then the user could carry on as normal, if an error occurred, the user was alerted to it. When the user wished to load a complex scene there was a slight delay, the system failed to provide any feedback concerning the tasks completion.

A situation where the lack of feedback could have confused the user was when the user chose to save a scene before beginning a new one. Upon the completion of the save there was no confirmation that the scene had been saved successfully, the system just cleared the screen, leaving the user unsure as to whether or not the task had been executed.

There needed to be a clearer distinction between when the system was in 'drawing mode' and in 'menu mode', especially because the function of the device buttons were dependent on the mode. A simple way to have provided this feedback would have been to change the cursor depending on the mode the system was in, perhaps a smaller cursor to allow more precision when drawing, and a bigger cursor to improve selection when in 'menu mode'.

When using the menu system, 'rollover' buttons were used to indicate when the cursor was hovering over a button. Normally buttons were dark blue with light blue text, but when the user hovered the cursor over a button its colour changed to light blue with dark blue text.

The system lacked feedback when editing the line width; to select a line the user clicked on their choice and then pressed 'OK'. This procedure of selection could be improved by giving feedback of the selection by drawing a line somewhere on the display that uses the width of the line that is currently selected in the colour that is currently being used. There would be two benefits of using this approach: the first being that the user could see how their choice would be displayed, and would give them the option to change their selection before they left the dialogue; the second would be that they could see when they have successfully selected a width of line, without this feedback users are left unsure as to whether their action was executed.

A similar situation arose when editing the colour of the line; the user picked a colour from one of the two colour bars, with a pointer following the x position of where the cursor would be. When the user pressed the trigger on the device to select a colour, the pointer changed to the colour selected. This did not work well, the result being that the pointer just blended into the colour bar; the user only noticed, if at all, that the pointer had changed colour when it was moved into an area of more contrasting colour. A better solution would have been to adopt a similar strategy to that proposed for the editing of the line width, by displaying a line of the colour selected using the current line width.

The system lacked any feedback when typing filenames to be saved, loaded and exported; when the text was inputted into the program there was a bar that should have been used to display text, but when text was entered nothing changed. It is important that a user can read the name of the file typed in, if they were to make a spelling mistake, the file would be saved under the incorrect name; when the user later tried to load this file using the correct name, they would be unable to do so and an error would occur due to the system trying to open a file that did not exist.

An important task when drawing is to be able to draw lines stemming from another; the system could have employed better feedback to help the user join lines in terms of depth. It was fairly easy to line up the cursor with a line in the horizontal and vertical directions, but it was extremely difficult to align the depth of the cursor and the line. The system offered some feedback as to when the cursor intersected a line, with the line appearing to disappear inside the cursor. But further feedback could have been provided by changing the colour of either the cursor or the line when the two intersected.

For the rest of system feedback works on the principle that is an action leads to a change in state of the system then that change is displayed, if it does not then nothing would happen.

Clearly marked exits

This system did not contain any processes that took more than three steps to execute, and all of these processes were fairly self explanatory, if a user found themselves in the process of executing a function that they did not wish to, then they could easily leave that process.

When navigating the menu system the user could leave it at any time by pressing the show menu button again. Although not being clearly marked, it was explained when users first used the system that the menu button was pressed once to show the menu, and was pressed again to hide it.

When users entered a dialogue that would change the state of the system, a cancel button was displayed. Although the user could exit this sequence by pressing the menu button, this was added to ensure that users did not feel that they were forced to make a change just to leave the menu system.

When the ‘Help’ pages were displayed the system adopted a slightly different principle; when the help button was pressed for the first time it displayed the help for ‘Drawing Mode’, when pressed for the second time the help regarding ‘Menu Mode’ was displayed, on the third press of the button the help pages were hidden and the system returned to its original state. Although this process was slightly different from the menu system, it still followed the idea of pressing a button once to show something and hiding it by pressing the same button again.

Shortcuts

Due to the simplicity of the system there was no real need for shortcuts, the processes that the user would want to use most often were attached to buttons on the input device. The one shortcut that was implemented was related to the navigation of the menu system. If a user accidentally selected a wrong menu option they could press the left button to return to the main menu. This was designed to help users who were unsure as to what option they needed to select from the main menu to find the sub-menu they wanted. Without this, if a user selected the wrong option then they would have had to restart the process by hiding the menu and then showing it again.

This function could have been improved by editing its function to display the menu above where the user was located, for instance if the user entered the ‘Load’ dialogue when they meant to enter the ‘Save’ dialogue, it would have been more efficient for the button to return them to ‘System’ sub-menu rather than the main menu, but it would still have retained the functionality of helping users who were unsure as to which sub-menu to select when searching for a function.

Good Error Messages

The system only had one error message read, “There was an error reading the file.” This message was fairly generic and could have been improved upon because there were only two possible causes for this message; the first being when reading from a file the name specified did not match an existing file, and the second being when writing to a file the directory to be written to did not exist, but if the program was set up correctly then this should not have occurred.

The issue of reading from a file that did not exist could easily have been fixed by displaying a list of all of the available files and let the user choose from that list.

The system needed a message to warn users that they were about to overwrite a file, if the user wanted to overwrite it then they could carry on with the save, but if they did not, then they needed to be warned that their action would cause them to lose previously saved work.

Prevent Error Messages

The fact that the system only needed two error messages showed that a lot of work went into preventing errors from occurring in the first place. Also the simple nature of the system’s interface prevented users from being able to enter the incorrect data that could lead to errors.

Help and Documentation

The system made use of a tracking device, the shape and use of which may have confused users who were accustomed to using a standard three button mouse. To get around this, a diagram was provided as a reference to each of the button's function depending on the mode the system was in. These diagrams were attached to a button on the device so were always accessible regardless of the task being completed. The downside to this was that the user had to remember the functionality of this one button to find out the functionality of the others.

These diagrams were only a reference and could have been improved by making them more interactive, for example, by letting the user click on each of these buttons to display a brief description of the function attached to that button.

The system provided a user manual, see Appendix H, to help users learn about the system before using it and to act as a reference later on.

Reflection on Nielsen's Heuristics

These ten heuristics are designed to be general to cover any sort of user interface, and evaluated the system's usability well, but there were issues that the heuristics did not cover when evaluating the usability of a 3D system.

One such issue was that of object selection, one of the most fundamental project objectives was to make the interface as usable as possible. Most 2D UI's take the approach to design, whereby objects are selected by hovering the cursor over the object and pressing a button. Scali et al (2003) explain that the WIMP style interface is unsuitable for 3D UI design, so developers have to think of different ways to carry out this most basic of tasks, inevitably the process can be implemented to a range of standards. For this reason when evaluating a 3D user interface the heuristic, "selection of objects should be accurate" could be added to the above list.

Another issue is of maintaining visual continuity whilst the user is immersed in a virtual environment. If the system jumps unexpectedly, the user can be left feeling disorientated, and confused as to why the jump occurred. To increase the user comfort the system should change between states smoothly without the user noticing. Another heuristic to add to the list is, "maintain visual continuity at all times." The evaluation based on these heuristics is as follows:

Selection of objects should be accurate

Selection of options using the menu system was very easy. The movement of the cursor was restricted to vertical and horizontal motion in an effort to try to make the system behave more like a WIMP system that the users would feel comfortable with.

The buttons to be selected were very large so even if the user struggled to use the device, they had a large target to aim for. If the user felt at ease using the input device then the large buttons sped up the use of the system, with the user only having to move the cursor a shorter distance to select an option. When the cursor hovered over an option the icon changed colour, this feedback meant that the user did not have to concentrate so hard on positioning the cursor, once the icon changed colour they could press the trigger, and the option was selected.

When selecting the width of line to draw with, the user had to pick the appropriate width from a sliding scale. In an effort to make things easier the motion of the cursor was restricted to 1 DOF, when the cursor was positioned within the area used for selection. This gave the desired effect simplifying the process of editing the width of line, but the drawback was that no feedback was given as to the vertical position of the cursor, and it was too easy for users to move the cursor out of the selection area by mistake.

A similar situation arose when selecting colours, but led to greater confusion on the part of the user. The two colour bars both used the one DOF sliding scale approach to selection, with a small gap between the two. There was only a limited amount of space to place these two colour bars, and to accommodate them both their height had to be reduced, resulting in a narrower selection area. Selection from either of these colour bars was made more difficult because it was harder to keep the cursor within this smaller range along the y-axis, and the situation was exacerbated due to the lack of feedback as to the cursor's position on the y-axis.

Maintain visual continuity at all times

The system did well at maintaining visual continuity when the user was in the process of drawing and manipulating the scene. There were only two minor problems, both related to issue of restricting the input device's freedom of motion.

When editing the width of the line the user manoeuvred the cursor into the selection area, at which point the cursor disappeared and was replaced with a pointer that moved horizontally across the area behaving like a slider bar. The system could probably get away with this slight jump; it may have surprised the user when it happened for the first time, but afterwards they would know to expect it.

The problem arose when the user moved the cursor out of the selection area, when this occurred, the cursor reappeared in the correct position but the pointer jumped to the point where the user last pressed the trigger within the selection area. This jump could be quite confusing and could leave the user unsure as to what happened, and why the cursor jumped to where it did. A better approach may have been to hide the pointer until the cursor entered the selection area, and then to track the position of the cursor along the x-axis. To solve the problem of the lack of feedback regarding position of the cursor along the y-axis would, a small dash showing the position of the cursor could be used, so that the user could see when the cursor was approaching the edge of the selection area and take the appropriate action.

When selecting a colour a similar problem occurred, but it was made worse by the fact that when the user crossed the gap between the two colour bars, the cursor did not appear, instead the pointer jumped to the horizontal centre of the gap; when the user moved the cursor into the other colour bar the pointer then jumped to the x-position of the cursor, and to the normal vertical position of the slider bar. The situation was exacerbated because the bars were quite narrow so it was harder for the user to keep the cursor within the boundaries of the selection area.

5.2.2. Formative Evaluation

Bowman et al (2004) describe formative evaluation as, “an observational empirical evaluation method applied during the evolving stages of design.” When designing the UI, formative evaluation was used throughout, resulting in a highly usable product, see section 3.2. After implementing a new feature it was evaluated, allowing users to give their opinions as to how it could be improved. If the need for any changes arose, then these were implemented and the task was re-evaluated. This process was carried out repeatedly, leading to an iteratively refined design.

5.2.3. Usability Testing

Kuniavsky (2003) believes that usability testing is the most efficient method for discovering usability problems. By testing the interface it could be established whether users understood the design in the way that was intended. The usability tests followed a format whereby a test subject was given a brief introduction to the system, then given some time to get used to it. Once they felt comfortable they were given a task (see Appendix I) requiring the use of most of the functions offered by the system, whilst they completed the task they were observed to see how they coped with interacting with the system on their own, upon the completion of the task a brief interview was conducted to ascertain the user's opinions of the system. The test subjects were students from a number of departments of the University, chosen to represent a wide range of artistic and computing abilities.

Jordan (1998) suggests effectiveness, efficiency and user satisfaction as three criteria against which systems should be evaluated. Upon completion of the task the test subjects were asked to pass comment about the system in terms of these criteria.

The effectiveness of the system was measured by establishing whether the test subjects were able to complete the task. All of the users were able to complete task to some degree, all managed to draw a representation of the object, though some failed to draw it in terms of 3D space. This was not a fault of the system rather the user's failure to grasp the concept of drawing in terms of 3D space. All of the users were able to edit the properties of the line and to load and save their work.

The test subjects of a more artistic persuasion commented on the lack of precision when using the cursor, preferring to use a smaller cursor allowing them to align the cursor with the lines more accurately. One subject commented that the lack of precision was perhaps more the fault of the device rather than the cursor, preferring to be able to draw using the finger tips rather than the whole hand.

The users found the task difficult, especially joining lines together to construct the geometric shapes. A few of the users noticed that the lines disappeared inside of the cursor, giving some feedback as to when the cursor and a line coincided. But all of the test subjects said that they would have liked to have been given more feedback signalling this event, perhaps either the line or the cursor changing colour.

The efficiency was measured by how quickly the test subjects were able to complete the task. All of the users completed the task on the first attempt, and in similar times.

User satisfaction was evaluated against the system's ease of use. All of the test subjects were able to use the system within minutes of picking up the device. Initially they struggled to remember the functions attached to buttons on the device, but once they had established what each button did, they were able to start using the system as intended.

Each subject found the UI easy to use, those with a background in Computing seemed to quickly establish the hierarchy of the menu system, and understood where a specific menu option would be located, whereas users from outside of the department seemed to have a little difficulty remembering where options were located. They found the large buttons easy to hit, and the few that noticed that the movement of the cursor had been restricted to two planes (horizontal and vertical) commented on how much easier they found it than trying to control the cursor whilst simultaneously thinking in terms of depth when in drawing mode.

Some of the test subjects struggled when selecting different colours for two reasons: firstly the process of having to select the colour followed by confirming the selection seemed to confuse some users, especially in the absence of any feedback to highlight that the selection had occurred; the second reason was because there were two colour bars squeezed into the space where there should have been one, some users found it difficult to keep the cursor within the selection area, and did not know whether they had moved the cursor out of the top or the bottom, so were unsure as to what action to take. This could be fixed using the extra feedback as to the cursors vertical positioned as laid out in section 5.2.1.

5.3. End User Evaluation

The end user evaluation followed a different pattern to the previous usability tests. No task was set; instead Claude was left to his own devices to see what he could come up with. Upon the completion of the evaluation, an interview was conducted to discover his opinions of the system. Jordan's (1998) three criteria were used again as the basis for evaluation.

In terms of effectiveness the system performed well, Claude was happy with the work he created using the system. He did not feel limited by the system; rather he struggled to find ways to make the best use of it.

At times he found that the system lacked precision, and would have preferred the option of zooming in on the scene to add more detail to the drawings. He commented that at times he felt hindered by the fact that the scene could only rotate around its centre, and would have liked to be able to have more control over its manoeuvrability. He also felt that the device was responsible for a lack of precision, preferring to draw using his fingers rather than using his whole hand.

He would have liked to have been able to choose from a set of different brush strokes, finding the simple line a bit limiting. When asked to expand on this point he suggested features such as a spray can effect and the possibility of replacing the line with a ribbon to give a better feeling for the 3D shape of the scene.

Claude found the system very efficient to use, and throughout the course of the two days was able to create a large number of pieces.

Claude found the interface extremely intuitive and commented that its simplicity allowed him to concentrate fully on drawing, the task for which the system had been created. Initially he encountered some of the problems that arose in the usability tests, for instance finding the selection of colour a bit fiddly, but after the extended use he clearly understood how the UI worked.

5.4. Evaluation of Drawing in 3D

With the exception of Claude Heath, for all of the users that evaluated the system it was their first attempt at drawing in 3D space. This group of new users could be categorized into two sub-groups concerning the way in which they approached the drawing task. The first group failed to grasp the context of drawing in space, drawing the object as it would look like from the front, forgetting to think in terms of depth. When the scene was rotated ninety degrees around the y-axis the picture was a series of lines bearing no relation to the shape they were asked to draw. The second group managed to draw the object correctly in space by drawing on a plane, rotating the scene by ninety degrees and drawing the next face of the object adjoining the first. Using this combination of lines and rotations the users were able to draw a representation of the shape that occupied the same space as the shape would have.

A common theme throughout the testing was that users did not feel comfortable with moving around and utilising the space available. The more artistic users realised that they had lots of space to work with, in terms of the vertical and horizontal directions, and made more use of that, whereas the other users felt constrained to working within a much smaller area. None of the users seemed to want to move their feet and walk into the drawing space, when asked why this was, the answer was always that the thought had not occurred to them.

The only user that exploited the space available was Claude, the explanation for this being his past experience of drawing in 3D space whilst working with the Tacitus project. Claude took a completely different approach to using the system, by setting up a work area and positioning objects to act as points of reference. He has an interest in drawing blind; as a result he did not use the stereo display option, and very rarely looked at what he had drawn, relying more on his memory of what he had previously drawn. The result was a better quality end product, with the pictures being much more spatially correct, he clearly understood how to work with the medium.

Obviously none of the test subjects who used the system were going to draw to the same standard of Claude Heath, being a professional artist and having experience of using similar systems in the past, but interesting comparisons can be drawn when observing the different approaches.

5.5. Physical Effects

The only problem that became apparent through the usability testing was that the stereo glasses easily fell out of sync with the display, causing the glasses to flicker. This was caused by the signal between the transmitter behind the display's screen and the receiver on the glasses being blocked. The problem was fixed for later evaluations by replacing the single transmitter with two either side of the screen.

6. Future Enhancements

In this section features that could be added to enhance the system and the 3D drawing experience are discussed. Firstly the requirements that were specified and assigned a ‘could have’ priority but were not implemented are discussed. This is followed by new ideas that arose during the end user evaluation, and during the usability testing.

6.1. ‘Could Have’ Requirements

The ‘could have’ requirements were specified at the start of the project, but due to the lack of time the decision was made that the quality of the system would not be affected without them.

The next step to improve interaction with the system is to implement stereovision based on the position and orientation of the user’s head; this will give the user a better sense of the 3D space in which the object they are creating in situated. It will also allow the user to look at the objects from whatever point of view they wish, allowing the possibility of looking at an object from the side without having to rotate the scene.

Claude Heath commented that he felt hindered by interacting with the drawing from one viewpoint, and would have liked to have been able to zoom into the scene and work on areas in greater detail. The same could be said for zooming out, some of Claude’s drawings were too big to be contained within the viewing frustum; this problem could have been solved simply by zooming out. Once the zoom feature has been added the scene can then be rotated around points other than the centre of the scene, but Claude would like to able to have had further control regarding the point within the scene about which it was rotated.

The system needs to offer a choice of line representations, currently the width of the line displayed is determined by a number of pixels on the screen, it’s relative depth within the scene has no bearing on its width. The next step is to represent the lines using either polygons to give a ribbon effect, or by using a combination of cylinders and spheres, to create a truly 3D line. With the use of polygons normals can be used to add shading to the lines, giving a better sense of the 3D shape of the object. This also means that the depth cue will be added so that objects that are deeper inside the scene will appear smaller than those situated at the front. The representation of lines do not have to be limited to these options, Claude said that he would like to draw using a spray can effect, or dashed lines, the length of dash could be set or based on temporal data.

The system would be improved by allowing the user to edit a curve once it has been drawn; options could range from editing a curve's width or colour, to allowing translation operations to be carried out on it, to curve deletion.

The properties of the curves could be improved to allow the opacity of the curve to be defined by the user. An attempt to implement this was made, but difficulties arose and it was abandoned in favour of other functions. In a similar vein the choice of colour could be enhanced to allow for the selection of a colour's hue in addition to its tone as is currently implemented.

During the analysis phase of the project an investigation was carried out to find a method to improve the representation of the line. It was concluded that non-uniform B-splines were best suited to the application, but due to time constraints the functions required to calculate the data were never implemented. It is probable that the extra calculations required to render each line would have an effect on the performance of the system, so the code will have to be altered to run more efficiently, reducing the effect of the extra calculations.

6.2. Future Enhancements

Currently the code written is inefficient; OpenGL display lists should be used to reduce the computation required. Once the curve has been drawn it can be stored as a display list, so when the curve is rendered the pre-built curve can be retrieved from the memory, instead of the application being required to build the curve each time it needs to be rendered, (Angel, 2002).

To make the task of drawing in 3D space easier a reference grid could be incorporated into to the system, to assist with the alignment of curves. This feature should be optional, allowing for it to be shown when the user needed it, and then turned off when it was no longer required. The scale of the grid should be able to be specified by the user to fit with the scale of the current drawing. The grid feature could be enhanced so that the cursor could 'snap' to the gridlines, assisting with the task of making lines join together.

The system could give better feedback as to when the cursor intersects a previously drawn line; this could be done using auditory cues, or by changing the colour of the cursor when it passes through another line. Special care would have to be taken, as this could be prove to be more of a hindrance in a situation where the lines are densely packed together.

To improve the depth cues offered by the system, an ‘atmosphere’ could be added so that curves nearer the front of the scene would have greater colour intensity than those situated towards the back.

During the end user evaluation Claude Heath showed an interest in the possibility of importing previous drawings in to the scene that he was working on. If this were to be implemented then thought would be required as to how to deal with possible difference in scale between the two drawings, giving the option to scale one scene to fit the other.

A possible feature that Claude was very interested in was to be able to export the drawings that had been drawn into a format that could be imported into a CAD program that created a 3D model through the use of rapid prototyping. Initial research was conducted into this option, but the enormity of this task quickly became clear. If this could be implemented then an interesting problem concerning how a 3D model of a drawing could support itself would arise, the solution would involve thought from the artist, but the system would have to provide support to make the task as easy as possible.

Finally Claude expressed an interest in the possibilities of changing the perspective in which the scene was drawn. The options could include isometric, dimetric, trimetric, and orthographic, perspectives and the system should allow the user to have control over them. OpenGL only provides single point perspective support, so research would have to be conducted into the maths behind these perspective options and into how these could be implemented to work within the system.

References

Ascension (2002) *Flock of Birds – Installation and Operation Guide* [Online]. [14/03/2005] Available from World Wide Web: ftp://ftp.ascension-tech.com/PRODUCTS/FLOCK_OF_BIRD/Flock_of_Birds_Manual-RevB.pdf.

Angel, E. (2002) *OpenGL: a primer.* Addison-Wesley.

Angel, E. (2003) *Interactive computer graphics: a top-down approach with OpenGL.* Addison Wesley.

Avison, D. and Fitzgerald, G. (2002) *Information systems development: methodologies, techniques and tools*. McGraw Hill.

Boulton, R. C. (2004) *Virtual Reality On The Holobench*. School of Computing, University of Leeds.

Bowman, D., Kruijff, E., LaViola, Jr, J. J., and Poupyrev, I. (2004) *3D User Interfaces: theory and practice.* Addison Wesley.

Burden, R. L. and Faires, J. D. (2001*) Numerical Analysis*. Brooks Cole.

Cobas, J.C. (2003) *Basic concepts on Endianness* [Online]. [13/02/2005] Available from World Wide Web: http://www.codeproject.com/cpp/endianness.asp.

Crvarich, G. B. (1995) *An Exploration of Techniques to Improve Relative Distance Judgments within an Exocentric Display* [Online]. [12/03/2005] Available from World Wide Web: http://www.hitl.washington.edu/publications/crvarich/home.html.

Davies, C. (1995) *Osmose* [Online]. [30/03/2005] Available from World Wide Web: http://www.immersence.com/osmose/osmose.htm.

Deering, M. F. (1995) *HoloSketch: a virtual reality sketching/animation tool.* ACM Transactions on Computer-Human Interaction, vol 2(3) pp 220-238.

Deitel, H.M. and Deitel, P.J. (2001) *C++ how to program.* Prentice Hall.

Dix, A., Finlay, J., Gregory D. Abowd, G. D., and Russell Beale, R. (2004) *Human Computer Interaction.* Prentice Hall.

Engineering and Physical Sciences Research Council (2004) *Designing for the 21st Century* [Online]. [21/04/2005] Available from World Wide Web: http://www.design21.dundee.ac.uk/home.html.

Foley, J. D., van Dam, A., Feiner, S. K. and Hughes, J. F. (1997) *Computer graphics: principles and practice.* Addison-Wesley.

Fores, S. (2004) *Titles from Previous Years* [Online]. [10/11/2004] Available from World Wide Web: http://www.comp.leeds.ac.uk/tsinfo/projects/UG/previous-years.html.

Hartman, J, and Wernecke,J. (1996) *The VRML 2.0 handbook: building moving worlds on the web.* Addison-Wesley.

Jamsa, K, and Klander, L. (1998) *C/C++ programmers bible: the ultimate guide to C/C++ programming.* Jamsa Press.

Joy, K. (1997) *On-Line Geometric Modelling Notes - Bernstein Polynomials* [Online]. [23/02/2005] Available from World Wide Web: http://graphics.cs.ucdavis.edu/GraphicsNotes/Bernstein-Polynomials/Bernstein-Polynomials.html.

Jordan, P. W. (1998) *An introduction to usability.* Taylor & Francis.

Keefe, D. F., Feliz, D. A., Moscovich, T. L., David, H. and LaViola, Jr, J. J. (2001) *CavePainting: a fully immersive 3D artistic medium and interactive experience.* Proceedings of the 2001 symposium on Interactive 3D graphics, ACM Press, pp 85-93.

Kingston A. eds. Guning, L., Heath, C., and Smith, R (2003) *What is Drawing? Three Practices explored*. Black Dog Publishing.

Kuniavsky, M. (2003) *Observing the user experience: a practitioner's guide to user research*. Morgan Kaufmann.

McReynolds, T. and Blythe, D. (1997) *Programming with OpenGL: Advanced Rendering* [Online]. [22/03/2005] Available from World Wide Web: http://www.opengl.org/resources/tutorials/advanced/advanced97/notes/notes.html.

Mine, M. R., Brooks, Jr., F. P. and Sequin, C. H. (1997) *Moving objects in space: exploiting proprioception in virtual-environment interaction.* SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques, ACM Press, pp 19-26.

Mine, Mark R. (1995) *Virtual Environment Interaction Techniques*. Dept. of Computer Science, University of North Carolina.

Nielsen, J (1994) *Enhancing the explanatory power of usability heuristics.* CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM Press, pp 152-158.

Paul, C. (2003) Digital Art. Thames & Hudson.

Pressman, R. S. (2000) *Software engineering: a practitioner's approach*. McGraw Hill.

Rickus, N. M. (2003) *Provision of coursework information within SIS.* School of Computing, University of Leeds.

Ruddle, R. and Holbrey, R. (2004) *Peripheral Device Servers User Manual*. School of Computing, University of Leeds.

Sachs, E., Roberts, A. and Stoops, D. (1991) *3-Draw: A Tool for Designing 3D Shapes.* IEEE Computer Graphics and Applications, vol 11(6), pp18-26.

Scali, S., Wright, M. and Shillito, A.M. (2003) *3D Modelling is Not for WIMPs* [Online]. [10/11/2005] 10/11/2004] Available from World Wide Web: http://www.eca.ac.uk/tacitus/papers/HCI2003_camready2.pdf

Schkolne, S. (2002) *Drawing with the Hand in Free Space: Creating 3D Shapes with Gesture in a Semi-Immersive Environment.* Leonardo vol 35(4), pp 371-375.

Shaw, C. and Green, M. (1994) *Two-handed polygonal surface design.* UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology, ACM Press, pp 205-212.

Shillito, A.M., Paynter, K., Wall, S. and Wright, M. (2001) *Tacitus Project: Identifying Multi-Sensory Perceptions in Creative 3D Practice for Development of a Haptic Computing System for Applied Artists* [Online]. [10/11/2004] Available from World Wide Web: http://www.eca.ac.uk/tacitus/papers/tacituseurohaptics.pdf

Watt, A. (2000) *3D Computer Graphics.* Addison Wesley.

Weisstein, E. W. (1999) *Bernstein Polynomial* [Online]. [23/02/2005] Available from World Wide Web: http://mathworld.wolfram.com/BernsteinPolynomial.html.

Zhai, S., Milgram, P. and Buxton, W. (1996) *The influence of muscle groups on performance of multiple degree-of-freedom input*. Proceedings of the CHI `96 Conference on Human Factors in Computing Systems, ACM Press, pp 308-315.

Zhao, H. (2002) *Fitts' Law: Modelling Movement Time in HCI* [Online]. [05/01/2005] Available from World Wide Web: http://www.cs.umd.edu/class/fall2002/cmsc838s/tichi/fitts.html.

Appendix A – Personal Reflection

I believe that the project was a great success and that it met its original objectives as documented in the project plan. I feel that the end product met the primary requirement that the system should have a simple intuitive interface, acting as a tool for artists to work with, rather than a computer program simulating drawing. A personal high point in the project was when Claude Heath came to evaluate the system over an extended period of time. During the evaluation he was able to use the system to draw a number of pieces that showed the real potential of art in the digital medium.

The low point in the project was when the tracking hardware broke, at the time I was unable to see how to continue. When I was given the option to either continue to develop the system using a traditional 2 DOF device, or to take a step back and redo the work that had taken the last couple of weeks to integrate the 'Flock of Birds' hardware, neither seemed particularly appealing. In retrospect the correct decision was taken, although the implementation of the 'Flock of Birds' proved trickier than it should have been, the process would have been a lot harder, had Roy Ruddle not designed the modules that provided interfaces to the two different hardware technologies in a similar manner. Overall the problem caused the implementation to slip behind schedule by about three weeks, but by putting the extra hours in, the project was back on schedule in time for the progress meeting.

I feel that I have gained a lot of valuable experience throughout the course of the project; I have learnt how to conduct research effectively and efficiently. My expertise in programming has improved greatly, having come across a wide range of programming errors I now understand the reasons behind them, and feel equipped to deal with them.

Throughout the project a number of lessons have been learnt, these are outlined below.

Pick a project that appears interesting – A year is a long time to spend working on one piece of work, it is vital to choose a project that you will find stimulating as the end product will be a reflection of the time and effort you put in. Another benefit is that other people will also show an interest in what you are doing, getting their opinions and ideas can open up new avenues, which can only enhance the quality of the final product.

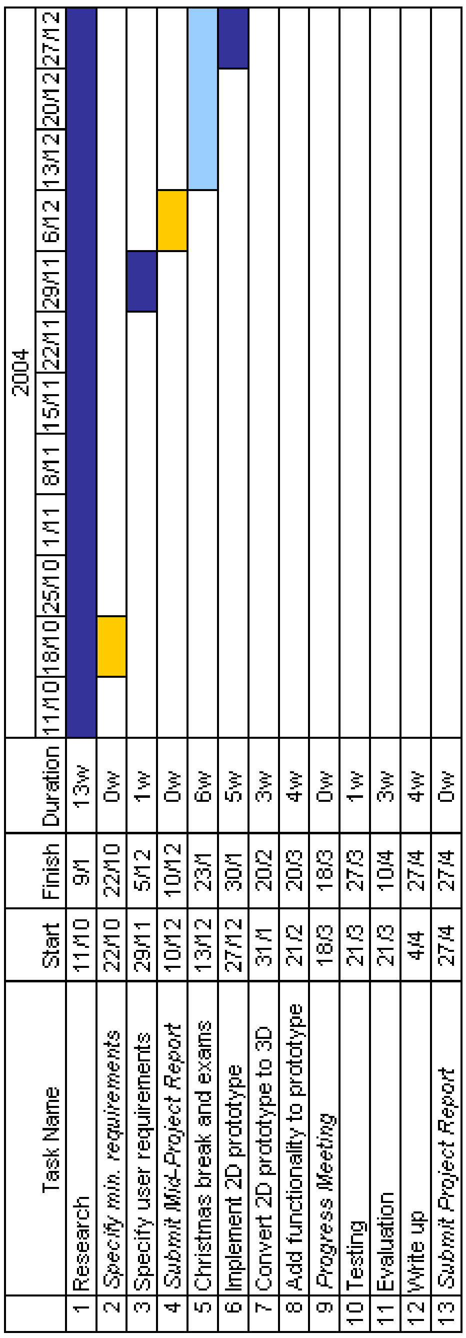
Keep perspective of what you are trying to achieve – The main aim of this project was to develop a tool to allow artists to draw, at points the development did not always advance towards this goal. It was easy to fall into the mindset of developing a system that leaned more towards a CAD style of drawing, whereby the system could have improved what the user was trying to draw, rather than improving the act of drawing itself.

Adding functionality is easy, making it usable is not – When implementing any aspect of the system, only a small amount of time was spent actually implementing the feature, the rest was spent trying to offer the user maximum control whilst exerting the minimum effort. Do not be afraid to take a trial and error approach to adding a feature to the interface, sometimes something may require a little tweaking to perform better, at other times the approach will clearly fail, at which point it is time to start anew and think of a different way to tackle the problem. There has been a lot of research conducted into the area of 3D UI development, if a suitable approach cannot be found, strong points from other people's research can be combined with a little imagination to provide an adequate solution. I would recommend that throughout the implementation the interface should be tested by other people to gain feedback, it is easier to change things quickly and get them right, as opposed to being told at the end that a number of aspects of the interface could use some work.

Do not be afraid to ask for advice – The University is full of people who are only too willing to help with any problems that you encounter provided that you have clearly made an initial effort towards solving the problem. It is easy to waste time by thinking that you are near to finding the solution to a problem, only to find that you were heading in the wrong direction from the start, instead take the initiative and speak to somebody who specialises in the problem area. After a short meeting the problem will usually be sorted allowing the project to progress towards its goal.

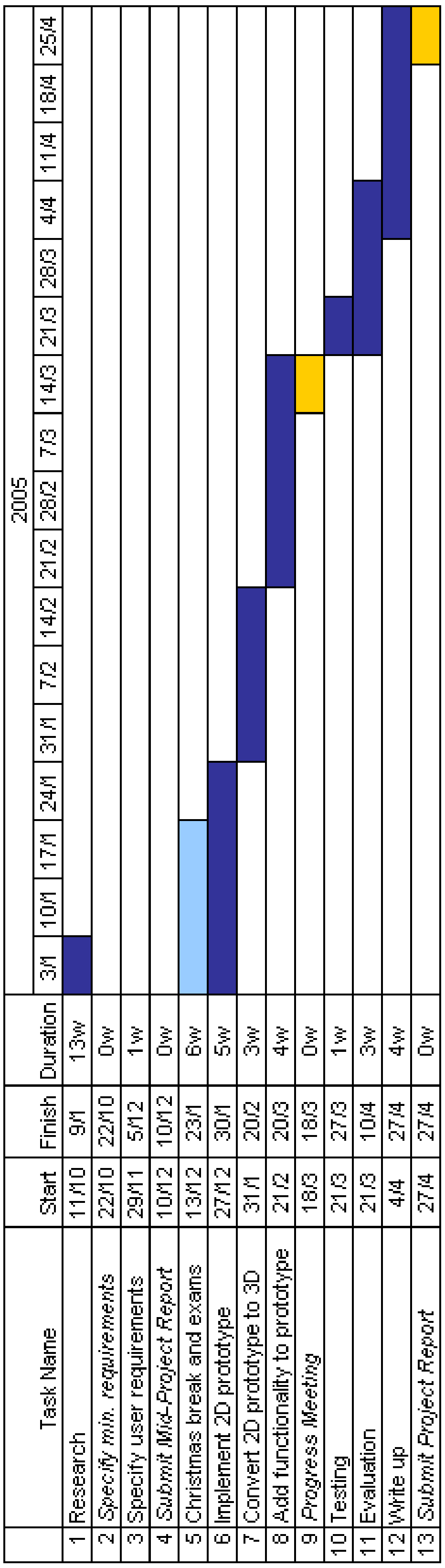
Take every opportunity that arises – When John Stell arranged the demonstration of the system to the Spatiality in Design research cluster I was initially unsure as to how this would benefit the project. I approached it with the attitude that it could be used to gain experience to help with my future career. In retrospect the time spent proved to be invaluable, giving me the opportunity to speak to a number of people who had conducted research in a similar area, learning about their opinions towards the system and their personal research proved to be extremely thought provoking in preparation for the end user evaluation.

Appendix B – Initial Project Schedule



2004

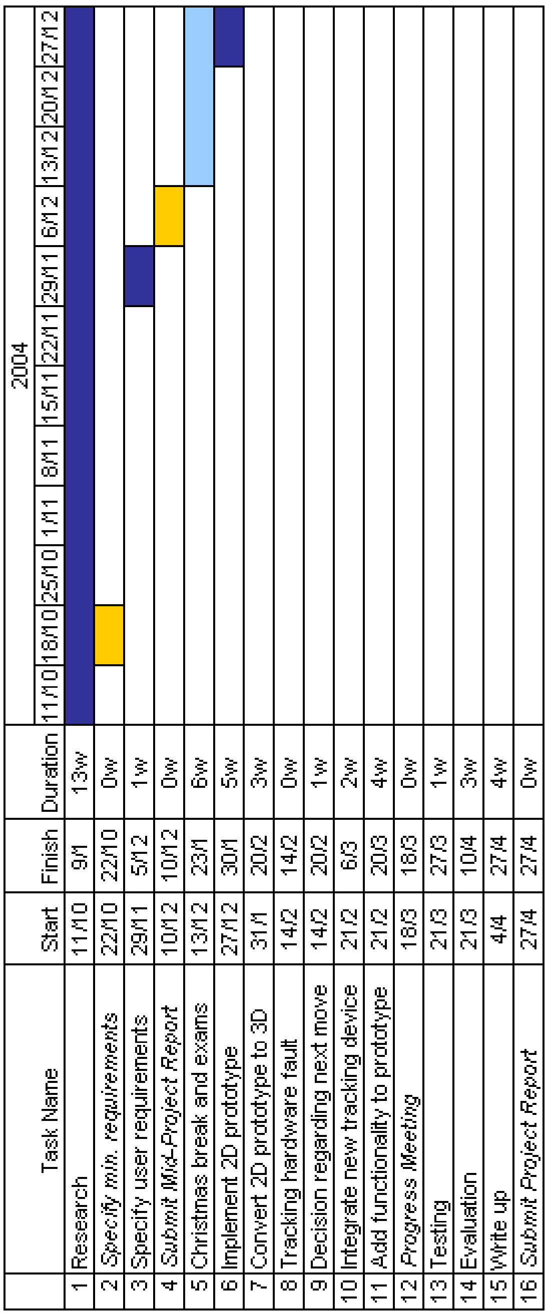
	Task Name	Start	Finish	Duration
1	Research	11/10	9/1	13w
2	*Specify min. requirements*	22/10	22/10	0w
3	Specify user requirements	29/11	5/12	1w
4	*Submit Mid-Project Report*	10/12	10/12	0w
5	Christmas break and exams	13/12	23/1	6w
6	Implement 2D prototype	27/12	30/1	5w
7	Convert 2D prototype to 3D	31/1	20/2	3w
8	Add functionality to prototype	21/2	20/3	4w
9	*Progress Meeting*	18/3	18/3	0w
10	Testing	21/3	27/3	1w
11	Evaluation	21/3	10/4	3w
12	Write up	4/4	27/4	4w
13	*Submit Project Report*	27/4	27/4	0w



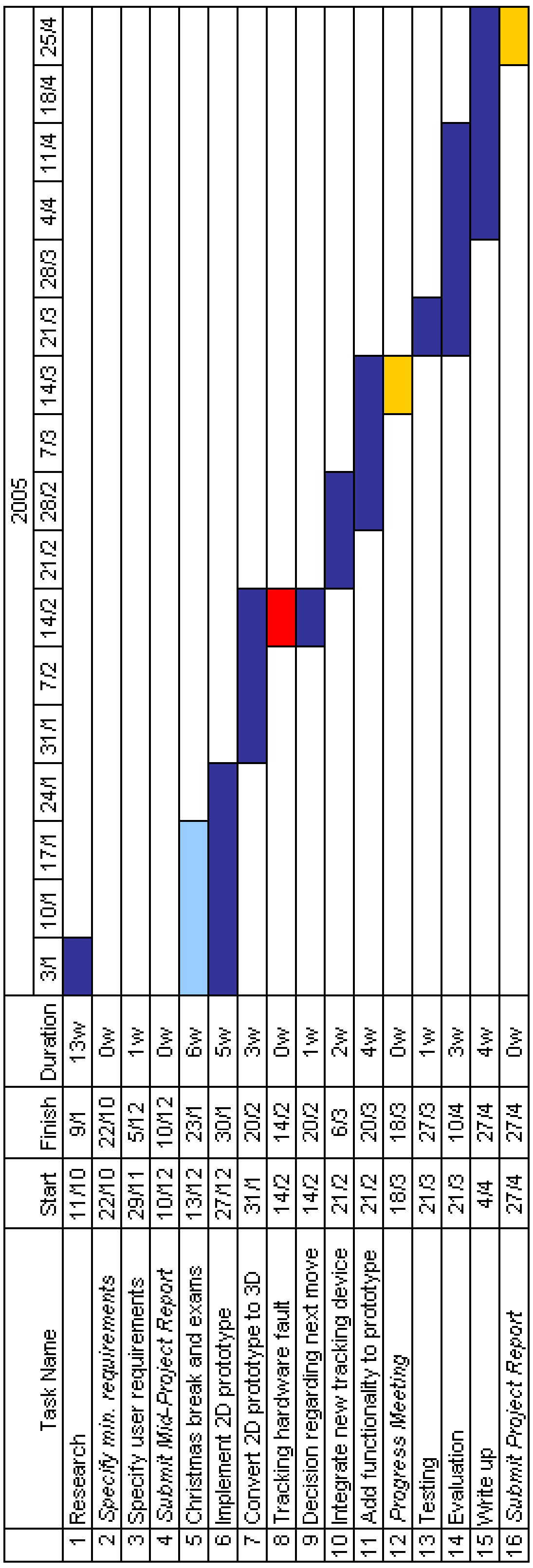
2005

	Task Name	Start	Finish	Duration
1	Research	11/10	9/1	13w
2	*Specify min. requirements*	22/10	22/10	0w
3	Specify user requirements	29/11	5/12	1w
4	*Submit Mid-Project Report*	10/12	10/12	0w
5	Christmas break and exams	13/12	23/1	6w
6	Implement 2D prototype	27/12	30/1	5w
7	Convert 2D prototype to 3D	31/1	20/2	3w
8	Add functionality to prototype	21/2	20/3	4w
9	*Progress Meeting*	18/3	18/3	0w
10	Testing	21/3	27/3	1w
11	Evaluation	21/3	10/4	3w
12	Write up	4/4	27/4	4w
13	*Submit Project Report*	27/4	27/4	0w

Appendix C – Revised Project Schedule



	Task Name	Start	Finish	Duration
1	Research	11/10	9/1	13w
2	*Specify min. requirements*	22/10	22/10	0w
3	Specify user requirements	29/11	5/12	1w
4	*Submit Mid-Project Report*	10/12	10/12	0w
5	Christmas break and exams	13/12	23/1	6w
6	Implement 2D prototype	27/12	30/1	5w
7	Convert 2D prototype to 3D	31/1	20/2	3w
8	Tracking hardware fault	14/2	14/2	0w
9	Decision regarding next move	14/2	20/2	1w
10	Integrate new tracking device	21/2	6/3	2w
11	Add functionality to prototype	21/2	20/3	4w
12	*Progress Meeting*	18/3	18/3	0w
13	Testing	21/3	27/3	1w
14	Evaluation	21/3	10/4	3w
15	Write up	4/4	27/4	4w
16	*Submit Project Report*	27/4	27/4	0w



	Task Name	Start	Finish	Duration
1	Research	11/10	9/1	13w
2	*Specify min. requirements*	22/10	22/10	0w
3	Specify user requirements	29/11	5/12	1w
4	*Submit Mid-Project Report*	10/12	10/12	0w
5	Christmas break and exams	13/12	23/1	6w
6	Implement 2D prototype	27/12	30/1	5w
7	Convert 2D prototype to 3D	31/1	20/2	3w
8	Tracking hardware fault	14/2	14/2	0w
9	Decision regarding next move	14/2	20/2	1w
10	Integrate new tracking device	21/2	6/3	2w
11	Add functionality to prototype	21/2	20/3	4w
12	*Progress Meeting*	18/3	18/3	0w
13	Testing	21/3	27/3	1w
14	Evaluation	21/3	10/4	3w
15	Write up	4/4	27/4	4w
16	*Submit Project Report*	27/4	27/4	0w

Appendix D - Requirements Specification

Iteration One – 2D Implementation

Must Have

- The user will be able to draw a freehand curve in 2D space; the system must store the curve and display it from then on.

Should Have

- The system will display the curve being drawn in real time.
- The system will be able to save a scene drawn by the user.
- The system will be able to load a scene previously drawn by the user.
- The user will be able to rotate the scene around the y-axis.
- The user will be able to undo the most recently drawn curve.

Could Have

- The user will be able to edit the colour of the line.
- The user will be able to edit the width of the line.
- The user will be able to edit the opacity of the line.
- The user will be able to edit the style of the line e.g. a dashed line.
- The user will be able to select curves and edit their properties.
- The user will be able to delete selected curves.
- The user will be able to translate selected curves.
- The user will be able to zoom in on the scene.
- The user will be able pan the view around the scene.
- The user will be able to edit the background of the scene.

Iteration Two – Conversion of 2D implementation to 3D

Must Have

- The user will be able to draw a freehand curve in 3D space; the system must store the curve and display it from then on.

Should Have

- The system will be able to save a scene drawn by the user.
- The system will be able to load a scene previously drawn by the user.
- The user will be able to rotate the scene around the y-axis.
- The system will display the output in stereo.

Could Have

- The user will be able to edit the width of the line.
- The user will be able to edit the opacity of the line.
- The user will be able to edit the style of the line e.g. a dashed line.
- The user will be able to select curves and edit their properties.

- The user will be able to delete selected curves.
- The user will be able to translate selected curves.
- The stereo output will be based on the position and orientation of the user's head.
- The user will be able to zoom in on the scene.
- The user will be able pan the view around the scene.
- The user will be able to edit the background of the scene.

Iteration 3 – Improvement upon the 3D implementation

Must Have

- The system will be able to save a scene drawn by the user.
- The system will be able to load a scene previously drawn by the user.

Should Have

- The user will be able to rotate the scene around the *y*-axis.
- The user will be able to rotate the scene around the *x*-axis.
- The system will display the output in stereo.
- The user will able to export the scene to VRML format.

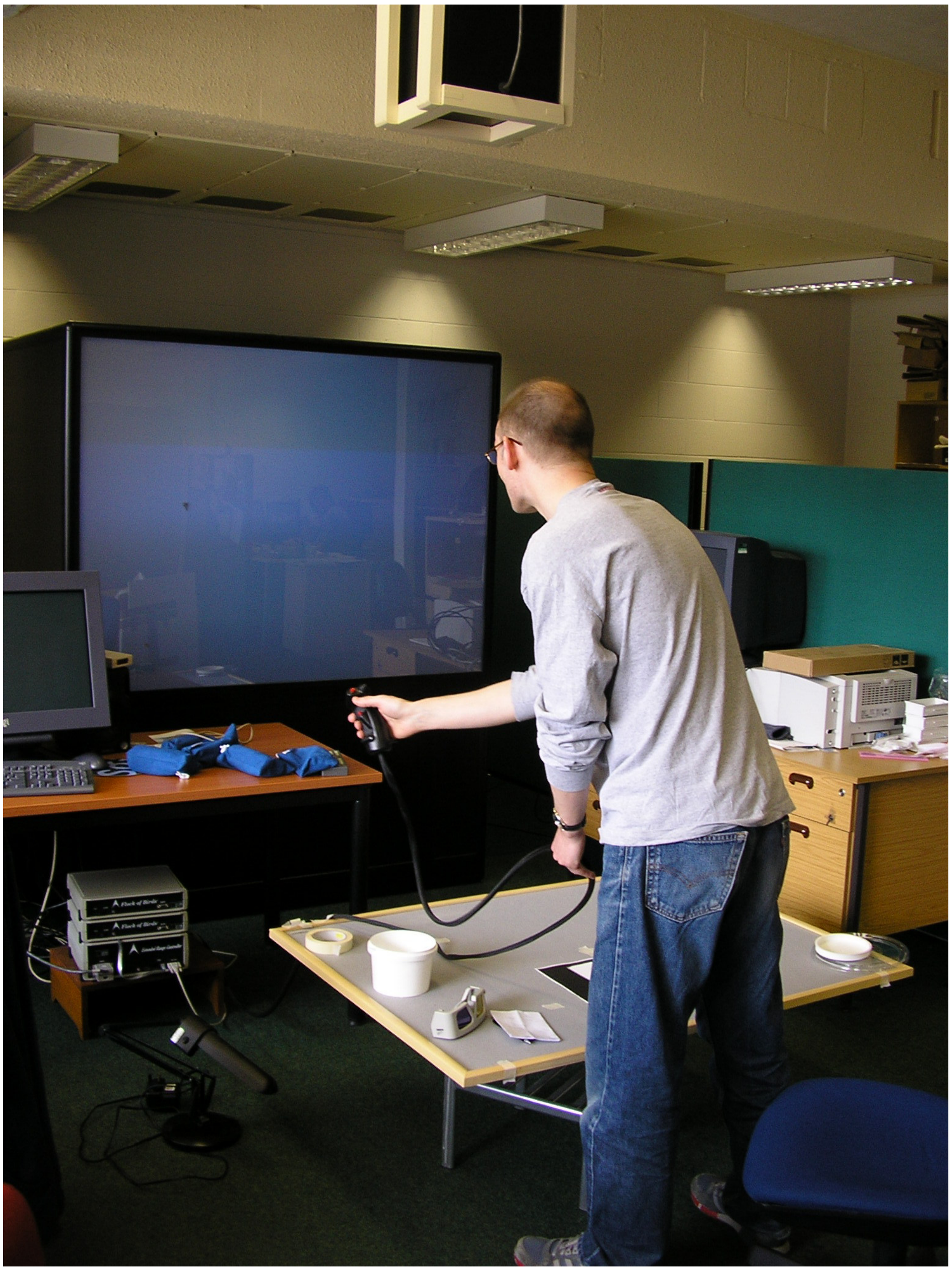
Could Have

- The user will be able to edit the width of the line.
- The user will be able to edit the opacity of the line.
- The user will be able to edit the style of the line e.g. a dashed line.
- The user will be able to select curves and edit their properties.
- The user will be able to delete selected curves.
- The user will be able to translate selected curves.
- The stereo output will be based on the position and orientation of the user's head.
- The user will be able to zoom in on the scene.
- The user will be able pan the view around the scene.
- The user will be able to edit the background of the scene.
- The system will display a grid to act as reference to help users with the process of drawing in 3D space.

The 'would have' requirements were not specified for each iteration, but below is a list of requirements that the system would have, but are beyond the scope of this project.

- The system will able to display the scene using a variety of perspective options e.g. isometric, axonometric.
- The user can edit the lighting properties of the scene, for instance the position, colour and attenuation.
- The system will export to a .STL file to enable a drawn 3D mark into a sculptural one through the use of rapid prototyping.

Appendix E – Photograph of the System in Use



taken on 12/04/2005

Appendix F – Test Plan

	Test	**Expected output**	**Pass/ Fail**	**Description of the problem**	
Command Line					
1	Run the program with no flags	The program runs in a window displaying mono output.	Pass		
2	Run the program with -f flag	The program runs using the full screen displaying mono output.	Pass		
3	Run the program with -s flag	The program runs in a window displaying stereo output.	Pass		
4	Run the program with -s and -f flags	The program runs using the full screen displaying stereo output.	Pass		
Calibration Mode					
5	Press the trigger whilst in the calibrate stylus dialogue	The drawing space appears with the cursor in the middle	Pass		
6	Press the top button whilst in the calibrate stylus dialogue	Nothing	Fail	When the stylus is calibrated the menu appears rather than the drawing space	Fixed
7	Press the left button whilst in the calibrate stylus dialogue	Nothing	Pass		
8	Press the bottom button whilst in the calibrate stylus dialogue	Nothing	Fail	When the stylus is calibrated the help page appears rather than the drawing space	Fixed
9	Move the stylus in 3D space	Nothing	Pass		
10	Move the joystick left	Nothing	Pass		
11	Move the joystick right	Nothing	Pass		
12	Move the joystick up	Nothing	Pass		
13	Move the joystick down	Nothing	Pass		

Drawing Mode					
14	Hold the trigger whilst moving the device	System draws a line following the path of the device	Pass		
15	Press the top button	Display the main menu	Pass		
16	Press the left button	Undo the last line drawn	Pass		
17	Press the bottom button	Display the drawing mode help diagram	Pass		
18	Move the joystick left	Rotate the scene anti-clockwise around the y-axis of the scene	Pass		
19	Move the joystick right	Rotate the scene clockwise around the y-axis of the scene	Pass		
20	Move the joystick up	Rotate the scene clockwise around the x-axis of the scene	Pass		
21	Move the joystick down	Rotate the scene anti-clockwise around the x-axis of the scene	Pass		
22	Move the stylus in 3D space within a metre cube around the calibration point	The cursor follows the position of the stylus	Pass		
23	Move the stylus in 3D space outside of a metre cube around the calibration point	The cursor disappears	Pass		
Main Menu					
24	Move the cursor over a menu option	The menu option changes colour to signify that the cursor is positioned over it	Pass		
25	Press the top button	Hide the menu	Pass		
26	Press the left button	Nothing	Pass		
27	Press the bottom button	Display the drawing mode help diagram	Pass		
28	Move the joystick left	Nothing	Fail	The scene rotates in the background	Fixed
29	Move the joystick right	Nothing	Fail	The scene rotates in the background	Fixed
30	Move the joystick up	Nothing	Fail	The scene rotates in the background	Fixed
31	Move the joystick down	Nothing	Fail	The scene rotates in the background	Fixed
32	Press the trigger over the 'System' option	The system menu is displayed	Pass		
33	Press the trigger over the 'Edit' option	The edit menu is displayed	Pass		

34	Press the trigger over the 'New' option	The save prompt is displayed	Pass		
35	Press the trigger when no option is selected	Nothing	Pass		
Edit Menu					
36	Move the cursor over a menu option	The menu option changes colour to signify that the cursor is positioned over it	Pass		
37	Press the top button	Hide the menu	Pass		
38	Press the left button	Return to the main menu	Pass		
39	Press the bottom button	Display the drawing mode help diagram	Pass		
40	Move the joystick left	Nothing	Fail	The scene rotates in the background	Fixed
41	Move the joystick right	Nothing	Fail	The scene rotates in the background	Fixed
42	Move the joystick up	Nothing	Fail	The scene rotates in the background	Fixed
43	Move the joystick down	Nothing	Fail	The scene rotates in the background	Fixed
44	Press the trigger over the 'Line' option	The edit line menu is displayed	Pass		
45	Press the trigger over the 'Colour' option	The edit colour menu is displayed	Pass		
46	Press the trigger when no option is selected	Nothing	Pass		
System Menu					
47	Move the cursor over a menu option	The menu option changes colour to signify that the cursor is positioned over it	Pass		
48	Press the top button	Hide the menu	Pass		
49	Press the left button	Return to the main menu	Pass		
50	Press the bottom button	Display the drawing mode help diagram	Pass		
51	Move the joystick left	Nothing	Fail	The scene rotates in the background	Fixed
52	Move the joystick right	Nothing	Fail	The scene rotates in the background	Fixed
53	Move the joystick up	Nothing	Fail	The scene rotates in the background	Fixed
54	Move the joystick down	Nothing	Fail	The scene rotates in the background	Fixed
55	Press the trigger over the 'Export' option	The export scene dialogue is displayed	Pass		
56	Press the trigger over the 'Load' option	The load dialogue is displayed	Pass		

57	Press the trigger over the 'Save' option	The save dialogue is displayed	Pass		
58	Press the trigger over the 'Exit' option	The save prompt is displayed	Pass		
59	Press the trigger when no option is selected	Nothing	Pass		
Save Prompt					
60	Move the cursor over a menu option	The menu option changes colour to signify that the cursor is positioned over it	Pass		
61	Press the top button	Hide the menu	Pass		
62	Press the left button	Return to the main menu	Pass		
63	Press the bottom button	Display the drawing mode help diagram	Pass		
64	Move the joystick left	Nothing	Fail	The scene rotates in the background	Fixed
65	Move the joystick right	Nothing	Fail	The scene rotates in the background	Fixed
66	Move the joystick up	Nothing	Fail	The scene rotates in the background	Fixed
67	Move the joystick down	Nothing	Fail	The scene rotates in the background	Fixed
68	Press the trigger over the 'Yes' option	The save dialogue is displayed	Pass		
69	Press the trigger over the 'No' option	The system exits / a new, empty scene is displayed	Pass		
70	Press the trigger over the 'Cancel' option	Return to drawing mode	Pass		
71	Press the trigger when no option is selected	Nothing	Pass		
Save Dialogue					
72	Move the cursor over the cancel option	The menu option changes colour to signify that the cursor is positioned over it	Pass		
73	Press the top button	Hide the menu	Pass		
74	Press the left button	Return to the main menu	Pass		
75	Press the bottom button	Display the drawing mode help diagram	Pass		
76	Move the joystick left	Nothing	Fail	The scene rotates in the background	Fixed
77	Move the joystick right	Nothing	Fail	The scene rotates in the background	Fixed
78	Move the joystick up	Nothing	Fail	The scene rotates in the background	Fixed
79	Move the joystick down	Nothing	Fail	The scene rotates in the background	Fixed

80	Press the trigger over the 'Cancel' option	Return to drawing mode	Pass		
81	Press the trigger when no option is selected	Nothing	Pass		
82	Type in text	The text should be displayed in the coloured bar	Fail	No text appears	
83	Press Backspace	Delete the last character typed	Pass		
84	Press Backspace multiple times	Delete all of the characters typed in the user	Fail	Starts deleting the path of the file	Fixed
85	Press Enter	The scene is saved under the name specified by the user	Pass		
Load Dialogue					
86	Move the cursor over the cancel option	The menu option changes colour to signify that the cursor is positioned over it	Pass		
87	Press the top button	Hide the menu	Pass		
88	Press the left button	Return to the main menu	Pass		
89	Press the bottom button	Display the drawing mode help diagram	Pass		
90	Move the joystick left	Nothing	Fail	The scene rotates in the background	Fixed
91	Move the joystick right	Nothing	Fail	The scene rotates in the background	Fixed
92	Move the joystick up	Nothing	Fail	The scene rotates in the background	Fixed
93	Move the joystick down	Nothing	Fail	The scene rotates in the background	Fixed
94	Press the trigger over the 'Cancel' option	Return to drawing mode	Pass		
95	Press the trigger when no option is selected	Nothing	Pass		
96	Type in text	The text should be displayed in the coloured bar	Fail	No text appears	
97	Press Backspace	Delete the last character typed	Pass		
98	Press Backspace multiple times	Delete all of the characters typed in the user	Fail	Starts deleting the path of the file	Fixed
99	Press Enter after entering a correct filename	The scene specified by the user is loaded	Pass		
100	Press Enter after entering an incorrect filename	The file error message is displayed	Fail	The next time a file name is entered it is appended to the incorrect filename	Fixed

Export Dialogue					
101	Move the cursor over the cancel option	The menu option changes colour to signify that the cursor is positioned over it	Pass		
102	Press the top button	Hide the menu	Pass		
103	Press the left button	Return to the main menu	Pass		
104	Press the bottom button	Display the drawing mode help diagram	Pass		
105	Move the joystick left	Nothing	Fail	The scene rotates in the background	Fixed
106	Move the joystick right	Nothing	Fail	The scene rotates in the background	Fixed
107	Move the joystick up	Nothing	Fail	The scene rotates in the background	Fixed
108	Move the joystick down	Nothing	Fail	The scene rotates in the background	Fixed
109	Press the trigger over the 'Cancel' option	Return to drawing mode	Pass		
110	Press the trigger when no option is selected	Nothing	Pass		
111	Type in text	The text should be displayed in the coloured bar	Fail	No text appears	
112	Press Backspace	Delete the last character typed	Pass		
113	Press Backspace multiple times	Delete all of the characters typed in the user	Fail	Starts deleting the path of the file	Fixed
114	Press Enter	The scene is exported under the name specified by the user	Pass		
Error Message					
115	Move the cursor over the cancel option	The menu option changes colour to signify that the cursor is positioned over it	Pass		
116	Press the trigger over the 'OK' option	Return to drawing mode	Pass		
117	Press the trigger when no option is selected	Nothing	Pass		
Line Width Selection Menu					
118	Move the cursor over the cancel option	The menu option changes colour to signify that the cursor is positioned over it	Pass		
119	Press the top button	Hide the menu	Pass		
120	Press the left button	Return to the main menu	Pass		
121	Press the bottom button	Display the drawing mode help diagram	Pass		

122	Move the joystick left	Nothing	Fail	The scene rotates in the background	Fixed
123	Move the joystick right	Nothing	Fail	The scene rotates in the background	Fixed
124	Move the joystick up	Nothing	Fail	The scene rotates in the background	Fixed
125	Move the joystick down	Nothing	Fail	The scene rotates in the background	Fixed
126	Press the trigger over the 'Cancel' option	Return to drawing mode	Pass		
127	Press the trigger when no option is selected	Nothing	Pass		
128	Move the cursor over the line selection area	The cursor should disappear and the pointer should track the devices horizontal movements	Pass		
129	Move the cursor from over the line selection area	The cursor should reappear, and the pointer should jump back to the position where the user last pressed the trigger	Pass		
130	Press the trigger whilst over the line selection area	Nothing	Pass		
131	Press the trigger over the 'OK' option	The next line drawn is of the width chosen by the user	Fail	The system does not use the thinnest line when selected	Fixed
132	Press the trigger over the 'Cancel' option after selecting a line width	Return to drawing mode, the line width unchanged	Fail	The action that should have been cancelled is not	Fixed
Colour Selection Menu					
133	Move the cursor over the cancel option	The menu option changes colour to signify that the cursor is positioned over it	Pass		
134	Press the top button	Hide the menu	Pass		
135	Press the left button	Return to the main menu	Pass		
136	Press the bottom button	Display the drawing mode help diagram	Pass		
137	Move the joystick left	Nothing	Fail	The scene rotates in the background	Fixed
138	Move the joystick right	Nothing	Fail	The scene rotates in the background	Fixed
139	Move the joystick up	Nothing	Fail	The scene rotates in the background	Fixed
140	Move the joystick down	Nothing	Fail	The scene rotates in the background	Fixed
141	Press the trigger over the 'Cancel' option	Return to drawing mode	Pass		
142	Press the trigger when no option is selected	Nothing	Pass		

143	Move the cursor over either of the colour selection areas	The cursor should disappear and the pointer should track the devices horizontal movements	Pass		
144	Move the cursor from over either of the colour selection areas	The cursor should reappear, and the pointer should jump back to the position where the user last pressed the trigger	Pass		
145	Press the trigger whilst over either of the colour selection areas	The pointer should change to the colour selected by the user	Pass		
146	Press the trigger over the 'OK' option	The next line drawn is of the colour chosen by the user	Pass		
147	Press the trigger over the 'Cancel' option after selecting a colour	Return to drawing mode, the line colour unchanged	Pass		

Appendix G – Claude Heath's Evaluation

Working with the three-dimensional freehand drawing system that Ben Hammett has designed has been a thoroughly rewarding experience for myself. As a practicing artist whose interest lies in the field of drawing and finding new ways of describing space, this software, which I was lucky enough to be given the chance to work with, has opened doors for me by providing new insights into the recording of space as it is experienced in real time.

The work we made with the system speaks for itself in terms of what was achieved in this area. There was a concurrent discussion held about aspects of the work which in itself was useful as an indicator of further avenues of exploration which his work has begun to open up.

I was able beforehand to provide a wish-list of attributes which I would have liked to see within such an application, a brief that ensured that the work carried out by Ben Hammett was not unduly influenced by any other work in this field which we were aware of. Hence the system as I found it has exhibited some unique features which have their own characteristics, which have been brought about by an original approach. This is all the more remarkable because the steepness of the learning curve for Ben has been so great.

The speed and spontaneity of working with the system was especially striking, as it became clear that there was to be no impediment to making progressively more complex and physically challenging work. This clearly was because Ben had made some good decisions about how the movements of the drawing hand were to be encoded and displayed during the process of working.

In the course of two days of drawing carried out in order to test out the capabilities of the software, and while getting used to the specific hardware which happened to be connected up to it, I also found the learning curve to be present, in the form of working out the best ways to use the system. The difficulty though was in finding correct usages rather than in learning complicated software procedures - this is because Ben has managed to design a very intuitive interface which once it has been briefly explained is extremely quick to pick up and work with. In other words it is a testament to the system which he has designed that I was primarily taken up with my own drawing interests and problems, rather than struggling to master a new application. This made the time spent drawing with it most worthwhile and rewarding, whetting the appetite for further work with it. (Claude Heath. April 2005.)

Appendix H – User Manual

1. Introduction

The application has been designed to allow the user to draw freehand in 3D space in a manner that is as simple and intuitive like drawing using a pen and paper. The system offers a range of functionality to aid the user in drawing to a decent standard. It also offers the option of displaying in stereo mode, when combined with stereo glasses the result is excellent depth cue to let user view what they really drew in 3D space.

2. The Basics

2.1. Running the Application

Before the application can be executed two servers must be running to read the data from the tracking hardware. To run the system the following instructions must be followed.

1. Turn on the tracking hardware.

2. To run the tracking server:
   1. Open a new shell.
   2. Type in the following commands,
      - `LD_LIBRARYN32_PATH=/usr/local/iri/iri-server/lib/`
      - `cd testsrc`
      - `/usr/local/iri/iri-server/server  peripheral_example.cfg tracker0`

3. To run the tracking server:
   1. Open a new shell.
   2. Type in the following commands,
      - `LD_LIBRARYN32_PATH=/usr/local/iri/iri-server/lib/`
      - `cd testsrc`
      - `/usr/local/iri/iri-server/server peripheral_example_stylus.cfg stylus0`

4.. To run the application:
   1. Open a new shell.
   2. Type in the following commands,

- `LD_LIBRARYN32_PATH=/usr/local/iri/iri-server/lib/`
- `cd testsrc`
- `./3Dsketch`

There are two flags that can be set via the command line.

Flag	Effect
-f	sets the application to run in full screen mode.
-s	sets the application to run using stereo output (requires the use of stereo glasses).

2.2. The Input Device

Fig. 1 shows a diagram that gives the details of what functions are attached to which buttons on the input device, depending on the state of the system.

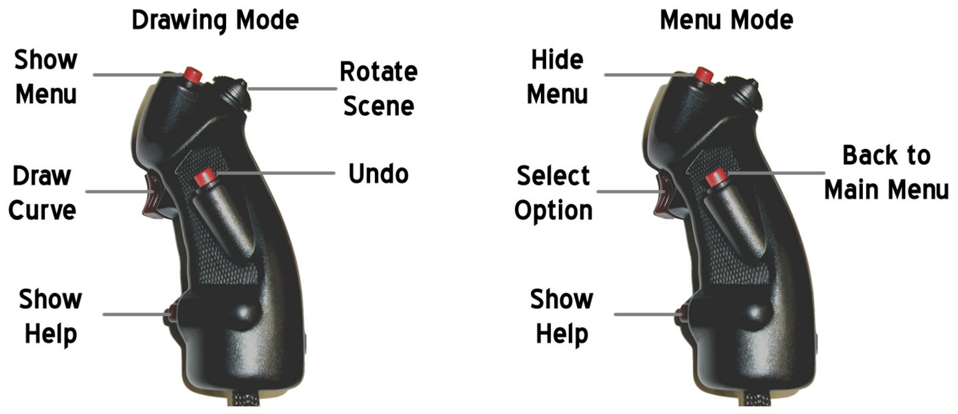


Figure 1 – A diagram of the input device, annotated with the functions attached to each button

2.3. Tracking device calibration

Before the system can be used for drawing, the tracking hardware must be calibrated to the centre of the workspace. This involves holding the tracking device where the centre of where the desired workspace would be, and pressing the trigger. From the position where the trigger was pressed the workspace extends in the dimensions shown in fig. 2.

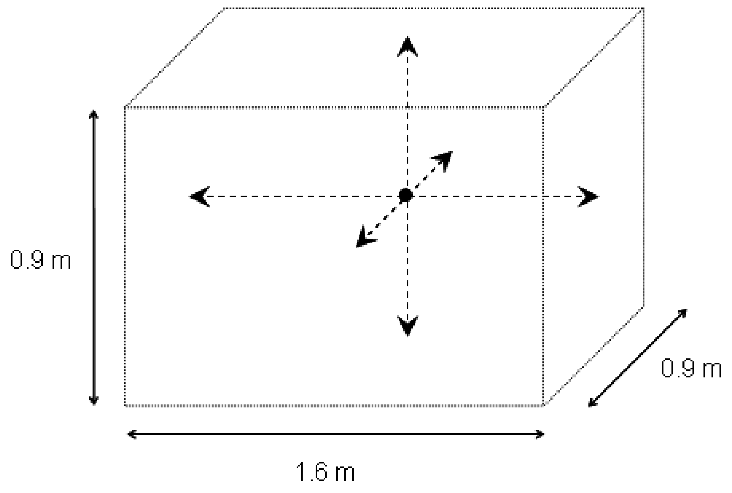


Figure 2 - The workspace available after the calibration of the tracking device

2.3. Drawing a Line

To draw a line, the trigger on the front of the device must be pressed and held down. Once the trigger is held the system tracks the movement of the device drawing a line tracing the path of the user's movement. The line is finished by releasing the trigger.

2.4. Rotating the Scene

Once a line has been drawn, the scene can be rotated. To rotate the scene in a direction the joystick situated on the top of the device should be moved towards that direction. To stop rotating the scene, return the joystick back to its original position. Once the correct rotation has been achieved the user can carry on drawing from the new point of perspective.

2.5. Undo Last Line Drawn

To undo any mistakes made, the last line drawn can be deleted from the scene by pressing the button located on the left of the device. Take care when using this feature because once a line is deleted there is no way of retrieving it.

2.6. Displaying the Help Pages

To help remember the functions of the buttons on the device, the diagram shown in fig. 1 is available when using the system. To display it, press the button located by the little finger; the first press shows the drawing mode diagram, the second shows the menu-mode diagram, and the third hides the help and returns the application to drawing mode.

2.7. Using the Menu

To use other options outside of drawing mode the menu system must be used. The main menu is displayed by pressing the button located on the top of the input device.

To select an option, position the cursor over the top of the button and press the trigger. Whenever the cursor is over a menu option the button will change colour to let the user know when the cursor is positioned correctly.

The menu system can be hidden at any time by pressing the button on top of the device (the same one used to show the menu).

If the incorrect menu option has been chosen, then the button situated on the left of the device can be pressed to jump back to the main menu.

3. Edit Line Properties

3.1 Edit Line Colour

Navigate to the 'edit colour' option by selecting **COLOUR** from the **EDIT** sub-menu. Fig. 3 shows the 'edit colour' screen. To select a colour, position the cursor over either of the colour selection areas. Once over a selection area the cursor will be replaced with a pointer that follows the horizontal position of the device, if the device is moved out of a selection area then the cursor will reappear.

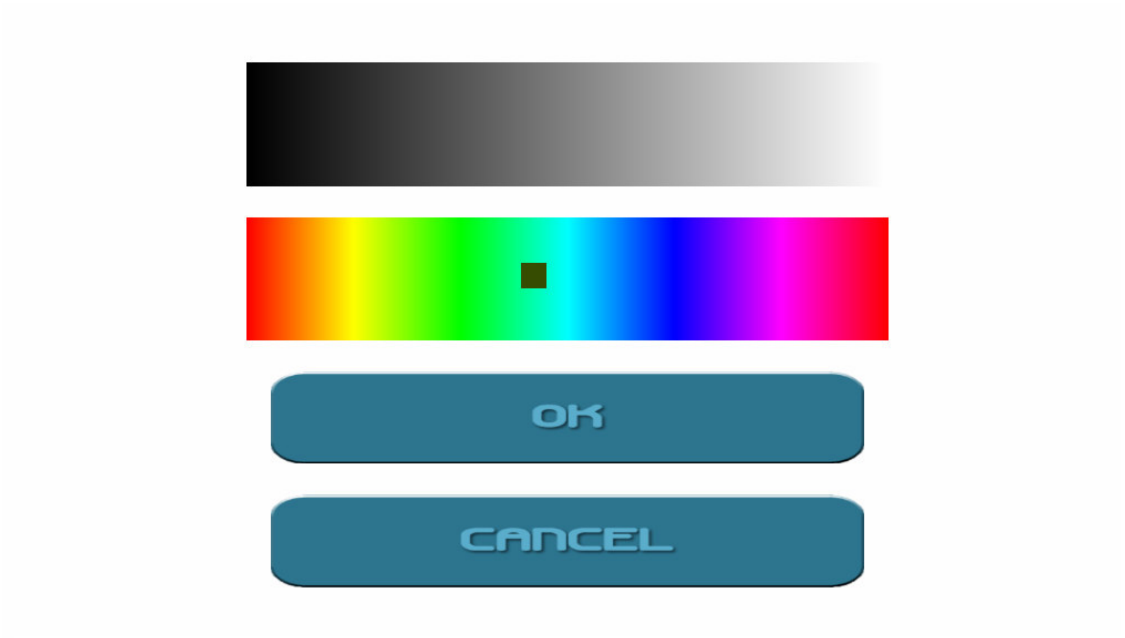


Figure 3 – Selection of colour using the slider along the colour bar

To select a colour, position the pointer in the relevant place and press the trigger. The pointer should change to the colour that was selected. If an incorrect colour was selected, reposition the pointer and press the trigger to select a colour again. Once the correct colour has been selected, confirm the choice by pressing the 'OK' button.

3.2. Edit Line Width

Navigate to the 'edit line' option by selecting **LINE** from the **EDIT** sub-menu. Fig. 4 shows the 'edit line' screen. To select a line width position the cursor over the line width selection area. Once over the selection area the cursor will be replaced with a pointer that follows the horizontal position of the device, if the device is moved out of a selection area then the cursor will reappear.

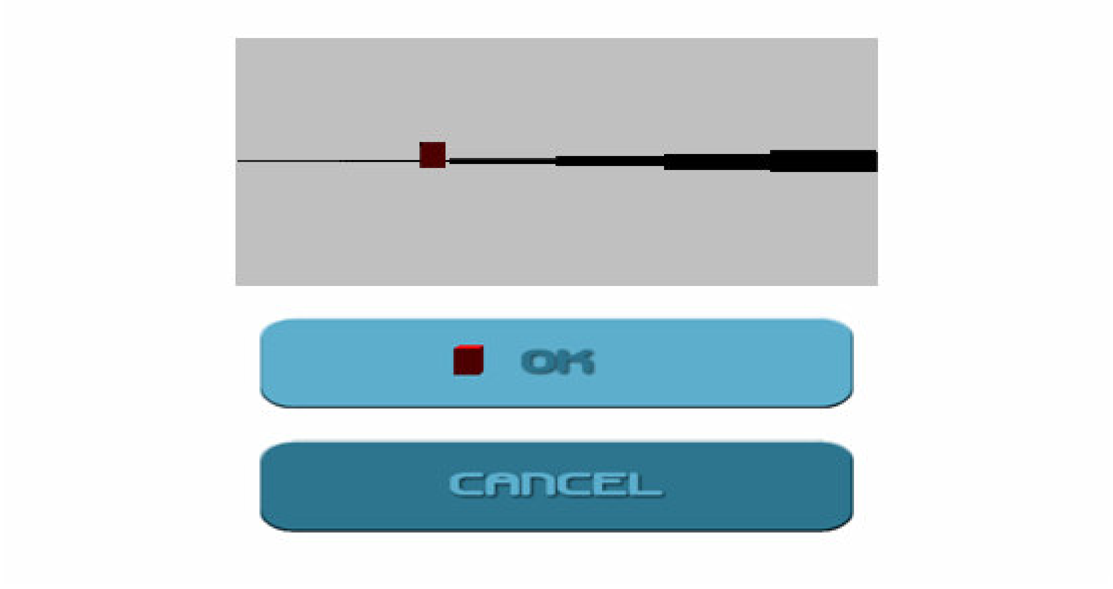


Figure 4 - Selection of line width using the slider along the line selection area

To select a line width, position the pointer in the relevant place and press the trigger. If an incorrect width was selected, reposition the pointer and press the trigger to select another line width again. Once the correct line width has been selected, confirm the choice by pressing the 'OK' button.

4. System Functions

4.1. Saving a Scene

The application can save a scene to allow further work at a later time. Navigate to the 'save scene' option by selecting **SAVE** from the **SYSTEM** sub-menu. Type in the name of the file and press the 'OK' button.

4.2. Loading a Scene

The application can load previously saved scenes to allow further work to be continued from a previous session. Navigate to the ‘save scene’ option by selecting **LOAD** from the **SYSTEM** sub-menu. Type in the name of the file to be loaded and press the ‘OK’ button.

4.3. Exporting a Scene

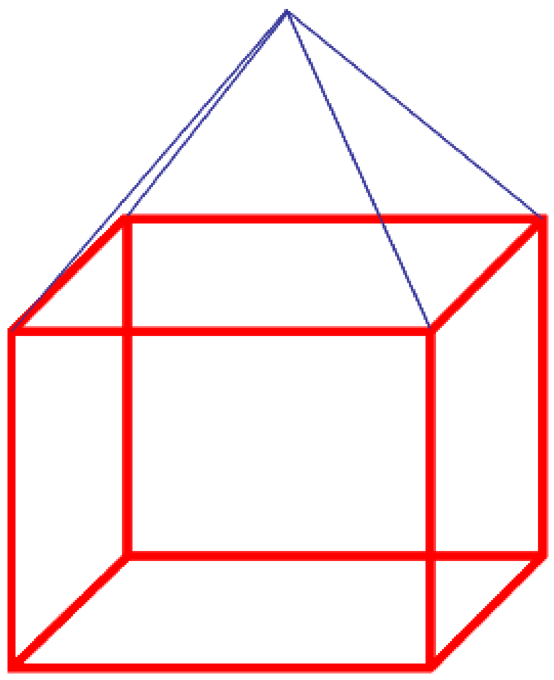
The application can export scenes in to VRML format allowing them to be viewed using a web browser with the appropriate plug-in installed. The plug-in is available for download from http://www.parallelgraphics.com/products/downloads.

Navigate to the ‘export scene’ option by selecting **EXPORT** from the **SYSTEM** sub-menu. Type in the name of the file and press the ‘OK’ button.

4.4. Exiting the Application

To exit the application select **EXIT** from the **SYSTEM** sub-menu.

Appendix I – Usability Test Task



1. Try to draw the above shape (a pyramid on top of a cube).
2. Colour the two objects differently and use different line widths.
3. Save the object.
4. Clear the screen.
5. Re-load the object.