INERTIAL SENSORS FOR VISUALIZATION CONTROL

A DISSERTATION SUBMITTED TO THE UNIVERSITY OF MANCHESTER

FOR THE DEGREE OF MASTER OF SCIENCE

IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2010

Jonathan Kirkham

School of Computer Science

Contents

Words: 19,809

Abstract

3D input devices are yet to make a significant impact in a computer interaction world dominated by keyboards and 2D mouse devices. Inertial sensors have recently become a popular 3D input device in products such as the Nintendo Wii and various mobile phones for use in gaming. This project assesses the capabilities of the Nintendo Wii controller with a view to using the device for computer interaction. It is concluded that the device makes a poor 6DOF tracking device because of error prone positional output, although the orientation data output is adequate. Software has been developed which integrates the Wii controller into the trackd 3D input device framework.

Scientific visualization software is a class of software that is often best used in 3D environments. Visualization applications often require interactive manipulation of software parameters by the user. Existing 3D interfaces are not optimal for this type of application control interaction. A custom interface has been designed and implemented which utilizes 3D input devices to provide efficient application control for visualisation software. Taking into account the deficiencies of the Wii controller, the designed interface operates solely on orientation data.

The developed interface can be used with cheap inertial sensor input devices and can be of benefit to any 3D applications requiring application control. An evaluation suggests that performance of the interface when used with the Wii controller is on par with existing 3D and 2D interfaces for visualization control.

Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I would like to thank my supervisor Martin Turner, and George Leaver for their helpful advice and guidance throughout the project. Thanks to James Perrin for his technical assistance. Many thanks to everybody who participated in all parts of the evaluation process.

I would also like to thank all my family and friends for their support over the duration of the project.

1 Introduction

Scientific visualization is the process of viewing complex data in various different forms. The data visualised is often 3D and therefore is easier to view with a 3D output device and easier to manipulate with a 3D input device. However, unlike other 3D applications, visualization is an interesting case because there is a stronger focus on user interactions with objects which are not part of the 3D scene. For example, in virtual reality environments, users navigate the environment and interact with virtual objects within the scene. Users of visualization software manipulate abstract parameters to alter the properties of the scene. Therefore, to maximise the efficiency of a visualization application, a 3D input device must allow intuitive controls for both viewpoint manipulation and application control for adjusting parameters. Unfortunately, these parameters typically have less than 3 dimensions and thus 3D input devices are less suited to controlling them. For example, selecting an integer value from a range of numbers only requires a single degree of freedom rather than the six degrees of freedom a typical 3D controller provides. 3D input devices are therefore a benefit to visualization systems as long as a suitable interface exists.

Research into 3D user interfaces for application control is not as developed as that of object selection/manipulation, the standard interaction of a virtual reality system. Interface designers have typically either utilised these developed interaction techniques for parameter control and/or converted existing 2D interfaces into 3D. Conversions project 2D interfaces into the 3D world. A 3D input device can then be used to generate a heading vector which imitates a 2D mouse pointer. This method is employed by the

VRMenu interface, an existing 3D interface used in visualization applications. This approach is suboptimal for a number of reasons:

- A highly accurate and noise free motion tracker is required. This can prevent cheaper, less accurate controllers from being used and thus hinders adoption of these technologies.
- The extra dimensions provided by a 3D input device make pointing more difficult and parameter control less efficient.
- The different capabilities of the controller are ignored. The 3D data can be mapped more intuitively to the interface.

Motion trackers can be used to control virtual reality environments and other interactive computing systems but have yet to become a standard or common input method. The high cost and speciality of the systems has prevented their widespread use. Inertial sensors offer a cheaper and more available range of devices which allow 3D user input. One such device is Nintendo's Wii controller which has revolutionised the home entertainment market. This system uses two forms of inertial sensor, accelerometers and gyroscopes, to detect motion. This project investigated the properties of the Wii controller and its ability to interact with existing user interfaces. Software was developed to allow the Wii controller to be used within an existing framework for 3D input devices.

This project tackles two interesting problems:

1. Using inertial sensors (in the form of the Wii controller) to control a visualization application.
2. Using 3D input devices for efficient application control.

Having identified the deficiencies in current interfaces and analysed the capabilities of the Wii controller, a custom user interface for visualization control was designed, implemented and evaluated. This design which is well suited for use with 3D input devices allows both viewpoint manipulation and parameter control using solely inertial sensors. The custom interface can also be used with traditional tracking devices. While application control is vital to visualization systems it is useful in almost all applications. The interface developed could be adapted to benefit any application which needs to be operated in a 3D environment.

The rest of this document is organised as follows. Chapter 2 introduces relevant background information for visualization, inertial sensors and 3D user interfaces. Chapter 3 describes the work carried out investigating the Wii controller and the development of the software library used to interface with the controller. Chapters 4 and 5 detail the design and implementation of the application control interface and the viewpoint manipulation interface respectively. The integration of the Wii controller and custom interface into the existing 3D input device framework is described in chapter 6, as well as the advantages this provides. An evaluation of the custom interface is presented in chapter 7. Finally, chapter 8 includes the conclusions for project and ideas for future work.

2 Background

The background research for this project can be split broadly into three main areas:

- Visualization systems.
- Inertial sensors (focusing in part on the Wii controller).
- User interface design (particularly 3D user interfaces).

This chapter provides an outline of the research carried out in these three areas during the project. An overview of the scientific visualization process, details of existing interfaces used in visualization applications, and details on why 3D input devices are desirable for visualization interaction are provided. The advantages of inertial sensors are highlighted and information providing expectations for the performance of the Wii controller as well as methods for maximising that performance are presented. Research into user interfaces describes why the current visualization interfaces are inadequate, as well as highlighting some existing designs. Finally, the core tenants of user interface design which were followed are presented.

2.1 Scientific Visualization

Visualization applications comprise a set of tools designed to allow users to view complex data in various ways. The software constructs visual representations of the complex data to make more information available from the data. The data can come from a variety of sources including computer simulations and various imaging devices (e.g. an MRI scan). The visual representation created allows the data to be more easily explored and new information to be discovered. For computer simulations, this can then lead to either further simulation based on the new information or a set of final results. Computing power (and storage capacity) has drastically increased in recent years. The quantity and complexity of data produced in computer simulations and scientific experiments has grown accordingly. Easy to use visualization software is now vital to the experiment cycle.

The Upson Analysis Cycle [1], which is based on the Haber and McNabb model [2], outlines the data flow in a visualization process (shown as the lower cycle in Figure 1). Data from a simulation is filtered down to the more informative data, mapped onto geometrical primitives and those primitives are rendered to produce a 3D scene. If the simulation involves a series of changing data, rendered images can be played back in sequence. Naturally each of the phases can take a number of different forms; the benefit of a visualization tool is that common tasks are made available to the user like software libraries in computer programming. This way, creating a custom visualization for each new experiment, written in low level data manipulation and graphics code, can be avoided.

2.1.1 User Interfaces

Modern tools, such as Advanced Visualization System's AVS/Express, provide a graphical interface for designing visualizations. A series of self-contained units has been designed which provide common functions with standardised input and output data types. These units are presented visually to the user as **Modules** within a network editor. The user places the required modules and connects compatible modules to define data flow through the network. Most of the modules have **Parameters** to customise their operation.

The user interaction in a visualization process can be separated into two distinct phases as shown in Figure 1:

1. Designing the network of modules which will make up the analysis cycle.
2. Configuring the parameters of the components while viewing the results interactively.

The designers of AVS/Express expected that once the network had been designed and tested that the user would then spend most of their time actually utilizing the visualization, simply adjusting parameters within the network where needed [1]. They imagined users would only return to the network editor occasionally for small edits or when an entirely new network is to be constructed.

This separation of tasks is exploited in AVS/Express with separate user interfaces: a network editor to place modules and connect them, and a data viewer for interacting with the visualization. The data viewer interface shows the current visualization image (if one exists) and provides quick access to the parameters of the modules currently loaded.

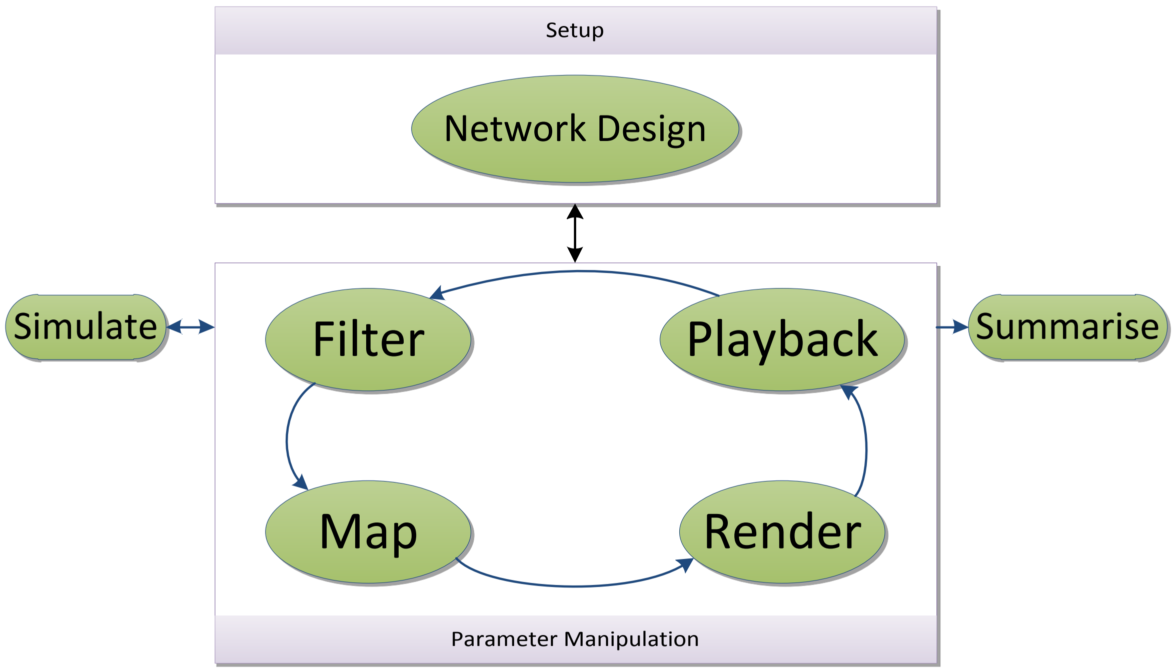


Figure 1: An annotated version of the Upson Analysis Cycle **[1]** showing the user interactions with a visualization system split into two distinct phases.

2.1.2 The Need for 3D

Because complex data is often three dimensional, it is preferable to use 3D hardware to view and interact with visualizations. 3D output devices allow the user to better visualise the 3D nature of the data. Input devices with more degrees of freedom or more natural input methods such as movement tracking, can offer easier and more intuitive ways for users to interact with the 3D scene than the standard 2D mouse. Also, most immersive environments prevent easy use of a standard keyboard and mouse for input. For example, a user cannot easily use these devices whilst standing, or with their view obscured by a head mounted display.

Output devices can include head mounted 3D vision systems, large scale 3D and 2D screens, CAVE style immersive environments [3], 3D workbenches, as well as standard desktop monitors. Input devices come in a variety of different forms but the Nintendo Wii controller will be the focus of this project due to the advantages provided by inertial sensors (section 2.2).

This project is only concerned with the performance of the data viewer interface when using various input/output devices. The network editor works well with the current interface. Using 3D input/output devices would probably decrease performance in this instance as the task maps more naturally to 2D input/output. Interaction involves selection of modules, positioning them in 2D and connecting them. A 3D version of the network display would add little benefit as depth information would be unused. A 3D input device again offers no benefits as the tasks only require 2D input. Furthermore, unlike the data viewer, the user does not need a large/3D output device and thus is not prevented from using a keyboard and mouse for interaction.

The standard data viewer interface within AVS/Express is designed for use with 2D input and output devices and is therefore suboptimal for use with 3D devices. VRMenu is an existing 3D interface that is used throughout this project for comparison purposes. The VRMenu interface [4] uses a set of 3D interface widgets which behave as standard AVS/Express objects within a 3D scene. This interface requires position and orientation inputs for control (it can also be used with a mouse). Because the interface is built using 3D objects it can be used with 3D output devices (unlike the standard 2D interface).

2.2 Inertial Sensors

Inertial sensors measure the effect of different forces upon a device. This project focuses on two of the most common types: accelerometers and gyroscopes. An accelerometer is an inertial tracking device used to measure linear acceleration. Linear acceleration is the change in velocity, in a straight line, over time. Three such accelerometers are typically combined, aligned orthogonally, to provide accelerations in all three dimensions. Gyroscopes measure rotational velocity with regards to a single axis. As with accelerometers, they are typically combined to allow velocity to be measured in 3 dimensions. Combining accelerometers and gyroscopes allows for “6 degrees of freedom” (6DOF), 3 positional measurements and 3 orientation measurements, theoretically allowing true 3D tracking.

Accelerometers and gyroscopes have only recently, with the advent of Microelectromechanical systems (MEMS), become small enough and cheap enough to be standard features in a number of electronic devices. For example, the ADXL335 3-axis accelerometer from Analog Devices (successor to the Nintendo Wii accelerometer) can be bought for around $3 (when purchased in bulk) and measures $4mm^2$ [5].

2.2.1 Advantages of Inertial Sensors

“Inertial trackers might appear to be the closest thing to a silver bullet” in motion tracking technology [6]. Inertial systems have many advantages when compared with other tracking systems [7] [8].

Inertial sensors do not require a reference point or complimentary emitter/sensor to operate and no line of sight must be maintained. This means that they have undiminished accuracy over their operating range. Only the length of the communication medium limits range; either wireless range or wire length. This is in comparison to other sensors such as magnetic or acoustic trackers, these sensors both need to be in range of a base emitter and suffer from interference in the environment. This means that their accuracy degrades as distance from the emitter increases.

High sample rates, low latency and a low level of jitter are standard in inertial technology. Further, sensors can now be manufactured with very low error rates, gyroscopes can have errors as low as 0.001°/s and accelerometers can be produced with accuracies in 10's of μg.

Finally, one of the main advantages of inertial sensor systems is cost. Magnetic positioning systems, such as the Polhemus Fastrak, cost upwards of £4000 and the Polhemus Minuteman (which only reports orientation) costs around £1000 [9]. Compare this with the Nintendo Wii Controller which in combination with the Motion Plus extension can be bought for around £40 and a standard Bluetooth receiver can be bought for as little as £2.

2.2.2 Nintendo Wii Controller

Along with various mobile phones, the Nintendo Wii controller has popularised inertial sensor input for gaming. Initially only including a 3-axis accelerometer, the recently released Wii Motion Plus extension adds a 3-axis gyroscope, again for use in games. The specifications of the Wii controller are not officially published by Nintendo. However, reverse engineering and disassembly of the controller by various community groups has produced a near complete set of specifications and various libraries to allow interaction with the remote [10] [11]. The Wii controller provides an ideal test bed for inertial sensor research, being both cheap and available. As well as including accelerometers and gyroscopes it provides a standard wireless interface (Bluetooth), a set of 7 buttons, and an 8 way direction pad.

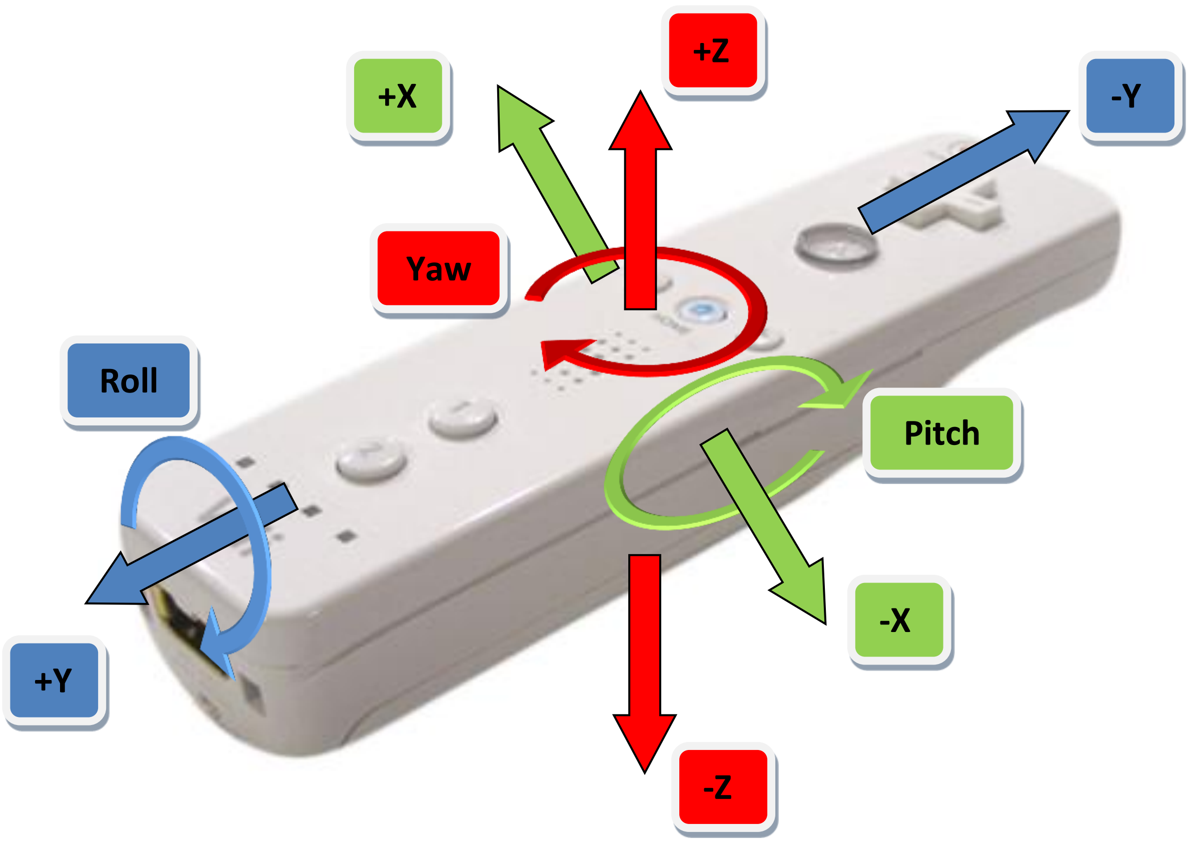


Figure 2: The coordinate systems of the accelerometers and gyroscopes in the Nintendo Wii Controller

2.2.3 Accelerometer Data

Acceleration data can theoretically be used to calculate both the orientation and absolute position of a device. Accelerometers measure all acceleration *apart* from the acceleration due to gravity. Therefore, when at rest, the accelerometer aligned with gravity will show the positive restoring force, equal to 1g, which is keeping the object stationary. The output is zero when the object is in free-fall. With a 3-axis accelerometer, the direction of the restoring acceleration will be known when the device is at rest and trigonometry can be used to calculate the devices orientation. This method can only provide absolute orientation information for two axes. One axis (depending on the orientation of the device) will be independent of gravity. For example, in the normal orientation (Figure 2), the yaw of the device does not change the output from the accelerometers. Also, this method is obviously less effective when the controller is in motion as the gravity response component cannot be isolated as easily.

Absolute positioning of the device can be calculated by double integration of the accelerometer data. There are two main problems with this approach; isolating the acceleration due to movement and dealing with drift.

To calculate the position of a device the gravity response component of the output must be removed leaving just the acceleration due to movement. Because the accelerations are in a fixed coordinated frame, there are problems when the device is rotated; the contribution the gravity response makes to each of the acceleration components will be altered. As described above, when the controller is at rest, the orientation of the gravity response can be recorded and this record used while the device is in motion. Obviously, this does not account for any rotation which occurs during the motion. Further, knowing when the controller is at rest is another challenge. Accelerometers cannot determine the difference between zero acceleration due to lack of movement and zero acceleration due to movement at constant speed.

Drift due to error accumulation is the other major problem when trying to track absolute position. Any errors in the output of the accelerometers will begin accumulating as soon as tracking starts, and with no reference to reset the position data the error continues to grow as time passes. Errors in accelerometer data can stem from incorrect sensor alignment, bias (acceleration at rest), noise, calibration, digitisation, and any errors due to the estimate of the gravity response component. Magnetic and acoustic systems do not suffer from this problem because they are measuring distance from a fixed reference point.

Errors are compounded by double integration; errors in acceleration cause error accumulation in the calculated velocity which in turn causes drift in the calculated position. Errors accumulated in the speed can be removed by zeroing the speed when the device is known to be still. However, there is no way to reset the absolute position of the device without the user returning the device to a known position and prompting a reset.

2.2.4 Gyroscope Data

Gyroscope data can be used to calculate the orientation of the device. This is achieved by integrating the rotational velocities produced by the gyroscopes. This technique suffers from one of the same problems as accelerometer data, namely drift due to accumulation of error. Similarly, when used independently, there is no reference for the device to reset the orientation to. Gyroscopes can also detect when a device is still with regards to rotational motion. Unlike accelerometers, gyroscopes measure velocity rather than acceleration. This means gyroscopes can detect when a device is still with regards to rotational motion as any response indicates the device is rotating.

2.2.5 Sensor Fusion

Combining accelerometers and gyroscopes can alleviate some of the problems mentioned above:

- Combined data from the devices makes it easier to detect when the device is still, allowing improved estimations of the gravity response component to be taken.
- These improved estimates can be used to reset (two axes) of the orientation of the device to counter the drift present in the gyroscopes.
- When the device is rotating during motion, the gyroscopes can update the estimation of the gravity response component. Thus, the estimation of

acceleration solely due to movement when the device is rotating will be improved.

However, errors will still exist and accumulate in the accelerometer data and in the unreferenced axis of the gyroscope data.

Even when using these techniques, very small levels of error in the device can lead to large amounts of drift. An accelerometer with just 1 mg error will have drifted 4.5m in 30 seconds [6]. The Wii controller has measurement resolutions of only 0.04g and 0.05°/s for the accelerometers and gyroscopes respectively. This will lead to much higher levels of error and drift.

Inertial sensors have been used for navigation and guidance systems for over 50 years. However, in these situations errors of the scale of 1 mile are typically acceptable and the devices themselves can be large and expensive (e.g. on board military submarines) [7]. For the much smaller human movement scale; "We may reasonably ask the question whether purely inertial 6-DOF motion tracking will ever be viable" [8].

Because of the problems outlined above, inertial sensors are rarely used independently. Typically, they are combined with other complimentary sensors. One example of this is combining accelerometers with an IR sensor in the Wii controller. The accelerometers can provide motion and orientation analysis and the IR sensor allows the device a fixed reference point to allow it to reset the position and orientation data as the user is forced to periodically direct the IR sensor at a fixed IR emitter. This negates the advantages of inertial sensors, line of sight must now be periodically established and the range of the device becomes limited.

2.2.6 Kalman Filter

A Kalman filter predicts the state of a system when the model used is not precisely defined. It has been used extensively in navigational systems where multiple sources of input are used to estimate position. The filter operates by using known noise and error models for the sensors together with their outputs to predict the current position and orientation. In the next cycle of the filter the difference between the prediction and the reality is recorded and the variables of the systems are updated to allow better predictions in the future. The filter also chooses the most likely result from the given information when there are two sources, for example, estimating orientation using information from both the accelerometers and the gyroscopes.

Such a system is ideal for the needs of this project, and has been successfully shown to improve results when detecting orientation [12]. However, designing Kalman filters and generating error models to give satisfactory results requires a great deal of time, skill and experience. Designing a Kalman filter to generate positional information from the Wii controller would take up an entire project in itself. Systems such as the Xsens Mti [13] use similar quality parts to the Wii controller (with a few additional sensors) to provide orientation data. However, because of the proprietary sensor fusion software (an extended Kalman filter) the system costs £1000's compared to the Wii controllers £40.

2.3 Application Control in 3D

Interaction in 3D applications can be broken down into object selection/manipulation, viewpoint manipulation and application control [14]. Object selection/manipulation is not as relevant to visualization systems as it is to virtual reality environments. The system exists primarily to view the data not alter it. Object selection/manipulation has however been used with 3D widgets to provide application control as discussed below.

Altering the parameters of the various modules within a visualization application can be considered application control. Unfortunately this is the least developed and researched area of 3D interaction with most developers preferring to transfer existing 2D interfaces into 3D. Manipulating parameters within the scene involves selecting first the module to which it belongs, selecting the required parameter from the module and then altering it. Selecting the module and parameter is regarded as a menu system [15], a 'menu hierarchy with a depth of 2'. Existing 3D interface designs for this problem are discussed below. Because the number of modules could be infinite, for some menu designs it might be beneficial to introduce another layer of hierarchy. In this case the modules are already grouped into categories in AVS/Express which makes this a menu hierarchy according to the above taxonomy.

2.3.1 Menu Interfaces

It is difficult to use traditional 2D menus with 3D input devices. One system which attempts this is XiVE [16]. XiVE places existing 2D windows in a 3D scene as textured interactive polygons. While this is convenient and allows rapid development of 3D applications it is difficult to interact with as the menus are designed for pointer input. 5DOFs (the roll of the controller is not used) are being used to select essentially a 2D element.

Because of this difficulty and because of the problems with accuracy of positional data from the Wii controller, this section focuses on interfaces which could be operated using orientation as their primary input data. Most of these designs are variations on the **ring menu**; each menu item is laid out on a horizontal ring which is rotated to select the correct item. This is a good design because it is easy for users to remember the location of an object in a 1D menu, operating the menu is a small and fast wrist movement and only orientation must be maintained to select an object, not position [17]. Rather than using all three orientation components, a ring menu can be operated using only a single DOF, a further reduction when compared to the 5DOF required for pointing. The problem with this base system is its lack of support for a menu hierarchy.

The Spin Menu [18] adds hierarchy to a standard ring menu is 3 different ways (Figure 3).

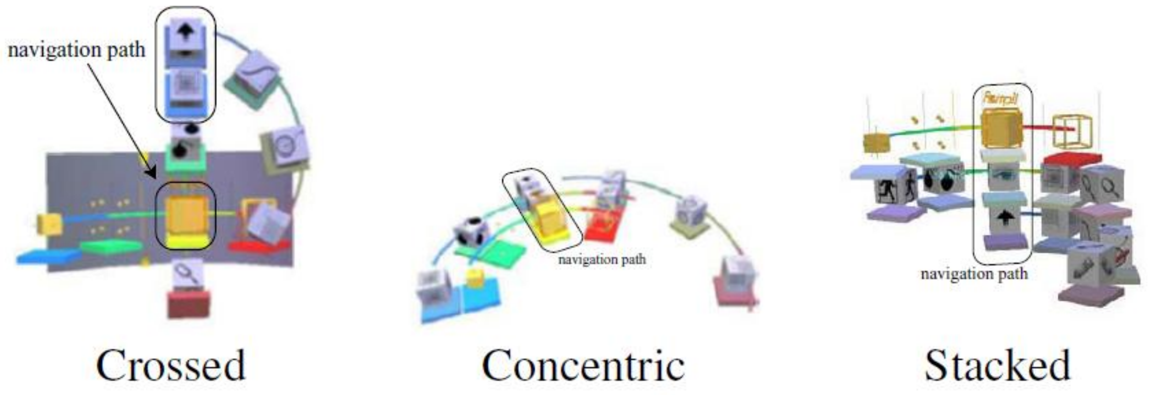


Figure 3: The Spin Menu based hierarchies **[18]**

The crossed menu displays submenus orthogonal to parent menus and performs poorly in comparison to the other menus. Concentric menus display submenus outside the current menu, while stacked menus display them above. These two systems seem to perform reasonably and would work with the visualization menu hierarchy.

Collapsible Cylindrical Trees [19] uses vertical rings instead of horizontal ones (Figure 4). Like spin trees this design has the advantage of showing the path the user took to reach their current selection. This aids with memory training and also navigation as the user knows where they are in the hierarchy.

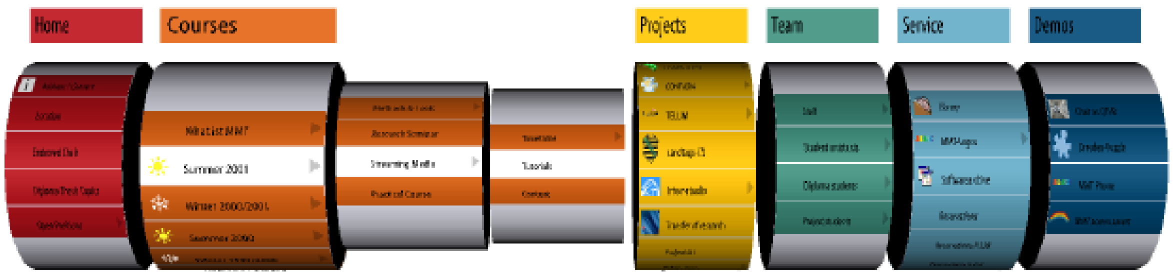


Figure 4: A Collapsible Cylindrical Tree **[19]**

Submenus extended from the current cylinder in a telescopic fashion. This system uses multiple interaction techniques; the user must perform one action to select a cylinder and another to choose an item on that cylinder. Controls for selecting a cylinder do not map well to orientation data.

HoloSketch [20] is an old design based on similar principles. It was designed for use with a pointing device but could be adapted to use orientation. Submenus open overlaid on the parent menu and as another full circle. The items are not placed on a single ring at each level but on several concentric rings. This allows a large amount of information to be displayed at once. It may work well for a pointing device but could introduce delays and complications when using an orientation device as the user has to select the correct ring.

All three menu designs offer possibilities for parameter manipulation but no implementations have been developed. For binary parameters the choices can simply be items on the menu, for 1DOF parameters the menu could be reconfigured such that each position on the ring corresponds to a value.

These existing interfaces do not take into account a user's need to view the scene while the menu is active as is the case in visualization software. The latter two interfaces benefit from being able to display more information on the screen at a time but require additional input beyond a single component of orientation. Because of the need to save screen space in visualization programs, simpler input was preferred over displaying many menu items. For this reason, the spin menu (concentric or stacked) was used as the basis of the interface design developed.

2.3.2 3D Widgets

The problem of manipulating the parameters is dependent on the parameter itself, for example, whether it is a binary choice or a range of values. It is also dependent on the input device.

The **3D Widgets** system [21] [22] is an attempt to standardise 3D elements for use within user interfaces. In the same way that 2D interfaces have a standard set of buttons, sliders, check boxes, etc.; this system specifies 3D shapes and controls that can be linked to them. Because the geometries of the objects are 3D they can be integrated into 3D scenes and controlled 3D input devices. Typically this system has been used to place widgets as additional objects within a scene to allow object manipulation. The user can remain immersed within the scene and still interact with it; they do not need to access menus outside of the scene to manipulate objects.

Object manipulation techniques are mature for positioning objects and orienting them in 3D. Therefore, these techniques are used to alter the widgets directly within the scene which will in turn alter the scene objects they are connected to. For example, a scale widget similar to a 2D slider bar can be attached to an object. The user can then select the widget and position it to alter the scale of the attached object in all three dimensions. Without widgets the user must select the object and bring up a menu to alter the scale in another interface.

For visualization systems, this type of interface is not always ideal. The parameters that need to be altered within a visualization application are often abstract and have no relation to objects in the scene. However, the widgets can also be used to create application control interfaces, such as the ring menus above. The widgets are held in a menu structure and not placed directly within a scene and widgets themselves are used to construct the menus. This technique is used by the VRMenu interface for the parameter manipulation tasks with AVS/Express. The only problem with current interfaces using this idea, including VRMenu, is that they rely on object selection and manipulation methods that inertial sensors cannot support or methods that are inefficient. Pointing, absolute position and orientation are required to interact with the widgets. The Wii controller cannot produce accurate positional data and pointing using a 3D input device is inefficient for essentially 2D selection.

2.3.3 Inertial Sensors for User Interaction

There is very little research into using purely inertial sensors for application control tasks. Typically, as mentioned, inertial sensors are used in conjunction with other sensor types. This is the case, for example, in the Nintendo Wii, the sensors are combined with an IR sensor and it is this input which is used for the majority of application control tasks.

2.4 Designing 3D Interfaces

Having identified that a 3D user interface is preferable when interacting with a visualization process, it is not a trivial task to implement a good user interface or even quantify what ‘good’ means in this context. There are important design considerations beyond those of a traditional 2D interface to take into account.

Standard 2D interface designs [23] should:

- minimise the memory load on the user
- be easy to learn
- be safe to use
- be efficient, minimise the actions required for each task
- have high utility, offer more functionality to the user to make their task easier

Users are considerably more mobile when using 3D interfaces and also have the possibility of their view being obscured or altered. Therefore, designing for safety in 3D interfaces is even more relevant. Efficiency is also possibly even more important as in a 3D interface not only will efficiency improve the performance of the design but also reduce the strain on the user (see User Comfort below).

‘3D User Interfaces Theory and Practise’ [7] identifies feedback, constraints, two-handed control and user comfort as the principle concerns when designing 3D interfaces.

Feedback can refer to any form of information relayed to the user, including information from the user’s own body, the system and the environment. For this project, visual, audio and vibration feedback were available. At its most basic good visual feedback matches real world actions to on screen actions, for example, tilting a controller left tilts the object left. For example, the Nintendo Wii menu system emits a small noise and vibration from the controller when menu boundaries are crossed. Low latency feedback is also paramount, even small amounts of latency can greatly decrease user performance. Also, both latency and poor feedback in general can actually lead to 'cyber-sickness' in users.

Constraints are important in this context in reducing the number of DOF's. Because the majority of tasks in parameter manipulation are 1D, it is vital that the 6 DOF's offered by the controller are constrained in some way to make operation easier. This can also be seen in traditional 2D interfaces, for example, a vertical scroll bar is a 1D control and so the 2D input from the mouse is constrained such that any horizontal movement has no effect on the scrolling. There is obviously a concern that too many constraints and the benefits of a higher DOF controller will be lost, as such, the constraints must match the task. Different 'helper' constraints can also be used in menu systems, for example, slowing a user's movement when they approach the centre of a menu item can aid in selection.

Two handed control can increase performance in certain user interface designs. This stems from observing that in reality humans naturally use two hands for the majority of tasks. Having identified two main tasks for interaction, viewpoint manipulation and application control, it would have been convenient for the user to be able to use one hand for each and to be able to do both tasks simultaneously. This technique was used successfully in an interactive 3D environment. Users used one hand to manipulate the camera and the other to perform object selection and application control tasks. The majority of users preferred and, after practise, performed better with the two handed design [24]. As mentioned, while multiple Wii controllers can theoretically be connected at once technical issues ruled out this possibility.

User comfort can be considerably harder to achieve when a 3D user interface is used. Typically the user will be standing to operate the system and use more movement in their body with less support. Two features of the Wii controller help it perform well in this area. A wireless interface and relatively low weight mean comfort is already optimal in those areas. However, there are still issues with fatigue. A design needs to limit free space interaction, having to hold your hand out in a certain pose for even short amounts of time is tiring. The design should require the user to interact for short periods with resting in between.

2.5 Summary

Visualization applications are well suited to 3D viewing but also require interaction. 3D interaction requires a motion tracking device as well as an interface which can support both view point manipulation and application control in the form of parameter manipulation.

Unfortunately, application control in 3D is not a well-developed research area and current designs suffer from a lack of constraints making them inefficient. The main design considerations which were followed in the development of a custom interface have been identified.

Most existing 3D input devices are expensive. The Wii controller provides a cheaper option for motion detection. It uses inertial sensors which provide the primary benefits of being unencumbered by range and cheap cost. The main disadvantage of this technology is that accuracy of positional data they can provide appears to be poor.

3 Investigation of the Wii Controller

For the Wii controller to be used to control visualization software the controller-computer interface needed to be created and the properties of the controller's output data had to be understood. This chapter describes the process used to investigate the capabilities of the Wii Controller. Several experiments were carried out to test the different components of the device. The results of those experiments and the effect those results had on the remainder of the project are presented. Details of the software library developed to interface with the Wii controller and relevant parts of its functionality are also included.

3.1 Position and Orientation

The majority of the user interface research using motion trackers focuses on devices which output position and orientation data. The existing VRMenu interface for AVS/Express also uses these inputs. Therefore initially, the ability of the Wii controller to provide this data was investigated. Using the techniques outlined in section 2.2, a software library was built to take the inputs of the Wii controller and supply position and orientation as the outputs. The results of this work were in line with previous research. The positional data suffered greatly from drift, to the point where it is almost unusable even for a few seconds.

3.1.1 Accelerometer Error

The acceleration data from the Wii controller has a multitude of error sources, including:

Digitisation error. The analogue signal from the accelerometer must be digitised before it can be stored in the Wii controller's memory. This digitisation process causes some of the data to be lost, resulting in error. The accelerometer in the controller is rated to measure at least ±3g [25] on each axis and it reports these measurements as 8 bits per axis. Therefore, the controller can only report accelerations in 0.02g increments at best. The specific controller used for the experiments was actually able to sense ±5g, because this entire range is reported, the resolution is 0.04g. This digitisation error is a problem if we recall that a constant error of 0.001g causes 4.5m of drift after 30 seconds. An error of 0.001g on the Wii controller could change the output ±0.04g, a constant error of 0.04g leads to 176.4m of drift in 30 seconds. This is a large error considering the range of human movement is likely to be only ±2m in each direction.

Jitter. Digitisation error also makes it difficult to assess the true jitter of the sensors. In certain positions there is no spatial jitter (change in the output when the device is still) whereas in other positions there is high frequency jitter. The digitisation error either masks or amplifies the jitter. For example, if the accelerometer is reporting a value of 1g but the true sensor output is 1.019g it only takes an increase of 0.001g and the output will change directly to 1.04g. This results in the jitter appearing to be exactly 0.04g or 0g. Without knowing the true sensor output the only thing that can be said for certain is that the maximum jitter magnitude is less than 0.02g, if it was any larger than this it would be impossible to find positions where there was no apparent jitter in the output. Without knowing the characteristics of the jitter it cannot be removed from the output.

Calibration error. The controller stores two calibrations per axis in on-board memory. The zero calibration specifies the output value which corresponds to 0g and the gravity calibration specifies the value of 1g. The gravity calibration and nonlinearity of the data will only affect the accuracy of the position data and not the drift error as these errors will be equal in the positive and negative directions. Therefore these errors are not too much of a concern and can be scaled as necessary to provide accurate position.

Nonlinearity refers to the case where the output from the controller does not scale linearly with the actual acceleration. The zero calibration data however will lead to a constant bias error which can cause serious drift as mentioned. The calibrations are stored in 8 bits and when the device is in motion this is not precise enough.

An experiment was designed to test the drift error from the accelerometers and the bias error. The controller was moved along its Z axis 90cm and back to its original position several times. The controller's movement was constrained in the other two directions so that rotation and gravity were not a factor.

When using the on-board calibrations for the Z axis, 119, if the accelerometers are aligned with gravity and orthogonal to it, they measure an almost constant 1g and 0.04g respectively so the drift was expected to be in the regions discussed earlier (176m over 30 seconds). As Figure 5 (a) shows, this was the case; 217m of drift after 40 seconds. All of the actual movements are obscured by the drift. To determine a more accurate calibration value, the velocity was manually reset to zero when the controller was known to be still in order to minimise error accumulation. Periods of non-movement had been included in the test so that it was obvious when the controller was still; the data was reset based on this prior knowledge, automatic detection of the controller being still was not used. After 40 seconds of motion using the on-board calibration there was still 17.2m of drift (Figure 5 (b)). Using a calibration of 188.1 (which cannot be stored using the same scale in 8 bits) on the same data gives a drift of just 0.006m. This more precise calibration was calculated by recording the raw output from the controller and then adjusting the calibration until the drift was as low as possible. Subsequent tests showed that this was the optimal calibration level.

It is impossible to calculate this value when the device is still because of digitisation error. For example, the controller will report 118 when the real value is 118.1 and will not change unless a relatively large change takes place (±0.02g). This means that an average of the sensor output when the device is still simply yields 118. If automatic detection of the controller being still and the estimation of the gravity component orientation were accurate, the calibration value could be obtained automatically. However, as described in section 3.1.3, small inaccuracies in these two systems lead to large errors. If a user moved the controller for a period and returned the controller to its starting position, software could determine the calibration value which would result in the lowest drift amount in each dimension.

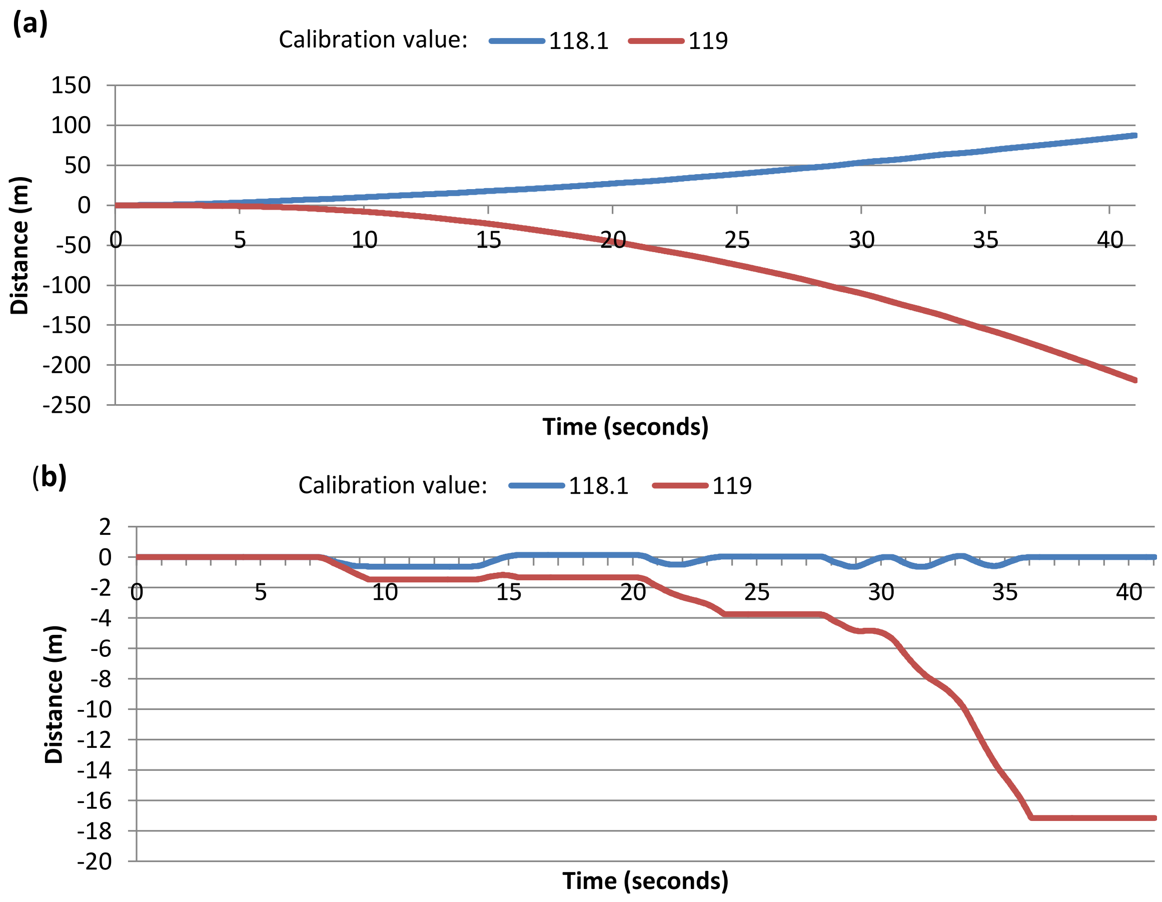


Figure 5: Graphs showing the drift in position over time when using two different calibration settings: (a) the data is untouched; (b) the velocity data was reset when the device was known to be still.

3.1.2 Gyroscope Error

The gyroscope output data is reported as 14 bits per axis and thus does not suffer from digitisation errors to the extent that the accelerometer does. There may be calibration data present for the gyroscopes in the controller memory but this information has yet to be reverse engineered. From the gyroscope data sheet and experimentation it is known that the gyroscopes have two scales depending on the speed of rotation. When the controller is moving slowly the controller reports rotational velocity in increments of 0.05°/s while at high speed it uses steps of 0.25°/s. It reports the scale used interleaved with the gyroscope data. This allows the output to be accurate at low speed and also cover a larger range of speeds.

The zero values for the gyroscopes were calculated through experimentation. The values were measured while the controller was at rest on a table and also at rest in a user's hand. The output from the gyroscopes whilst at rest on a still surface shows considerable jitter. This jitter is normally distributed with varying distribution parameters per axis as shown in Figure 6. The yaw noise standard deviation is roughly half that of the pitch and roll (0.8, 1.5 and 1.6°/s respectively), this is because the yaw sensor is a different model to the other two axis, presumably a more expensive and accurate sensor has been used to compensate for not being able to reset yaw from gravity.

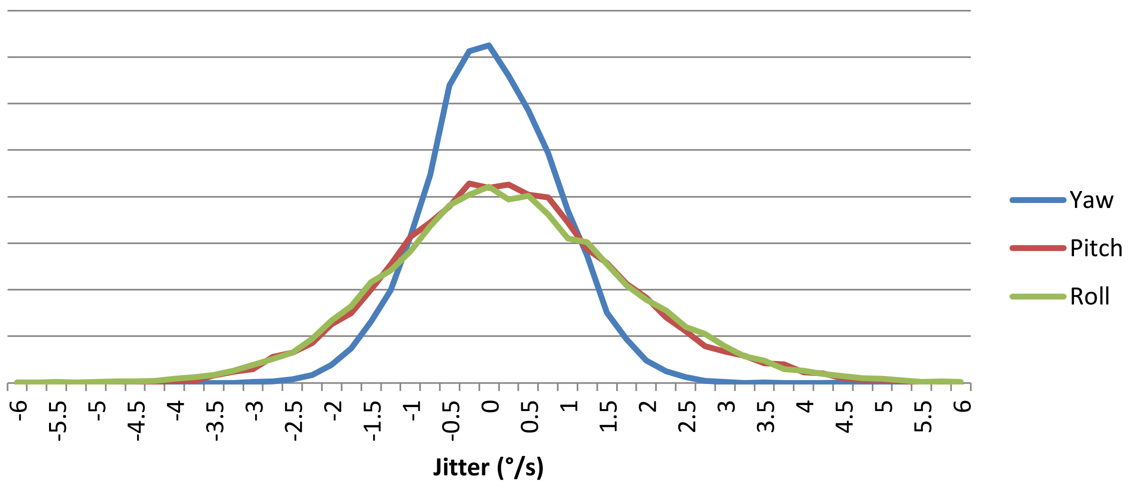


Figure 6: The frequency of jitter from each the gyroscopes over 5 minutes when at rest on a table.

The output was recorded whilst holding the controller still in a user's hand to better assess whether the controller is not moving as there is a significant difference in output compared to when the device is still on a table.

It was found that the drift was more significant when the high speed mode had been used, making it necessary to reset the orientation from gravity. Testing the yaw and pitch components using a low speed motor for 10 revolutions there was 50° of drift for both axes over a period of roughly 30 seconds. Any readings below 3 standard deviations of the jitter frequency for each axis were ignored which is why the pitch result is comparable to the yaw result despite an inferior sensor.

3.1.3 Total Error

The errors above do not represent the true accuracy of the controller for its intended use. When the inputs are combined together to calculate position and the movement of the controller is unconstrained, the error increases.

The main problem is differentiating a still controller from a moving one. In Figure 5 (b) we can see that the drift is still almost 100m even using the superior calibration when the velocity is not reset. The pattern of the actual movement which took place is completely lost in error due to drift. Several techniques were tried to automatically detect a still controller. For example, detecting when the magnitude of acceleration is roughly 1g for a short period and the gyroscopes report no movement (outside of their standard jitter range). In reality it is a difficult balancing act between ignoring very subtle but important accelerations and error accumulation due to not resetting the speed values.

Errors in the estimation of the gravity component orientation are the next biggest cause of error. X° of error in this estimation leads to 1g*sin(X°) error in the accelerometer data, so even 1° of error can cause 0.017g error.

These two errors combined lead to an unusable system. Depending on how the movement detection is calibrated the system either outputs incorrect/minute position data or data which suffers from drift in the hundreds of meters after 30 seconds.

The orientation information on its own is much more useable as expected. All three axes show very little sign of drift for short periods of time. The difficulty encountered when attempting to detect whether a controller is still is less of a problem when used to trigger a reset of orientation from gravity. For acceleration, if the velocity is not reset at the correct time (either too soon or too late), large errors occur which cannot be undone. More time can be allowed to assess whether the controller is still in the gyroscope case because the orientation is being reset to an absolute reference value. It must be remembered that the yaw component cannot be reset from gravity.

3.2 Software Library

While the Wii controller uses standard Bluetooth to communicate, it does not adhere to standard controller interfaces. Therefore, a computer will recognise the device and create a connection to it but cannot access any of the data from the controller. A library called ‘WiiYourself!’ [26] was used to interface with the controller. This library, written in C++, handles the connection to the controller and input/output of data to and from the controller’s memory. It exposes both the raw values from the sensors and some simple manipulations of these values, for example, a method to convert acceleration to orientation. At the time of writing it is the only library available with reliable support for the Motion Plus extension.

The software library developed uses its own calibration data for the gyroscopes to calculate the rotational velocities. As well as the calibration data, the gyroscope scale information is required to convert the raw output into rotational velocities. The ‘WiiYourself!’ library was therefore modified to expose the gyroscope scale information. This also allowed the developed library to monitor whether the controller had used the lower accuracy, and more drift prone, scale.

The developed library can report either rotational velocities or orientation data. However, it only provides acceleration data due to the problems with generating positional data. The final software library includes the following features:

3.2.1 Automatic Gyroscope Calibration

The exact properties of each Wii controller are different, including zero calibrations for each Motion Plus accessory. Therefore, an automatic calibration method was developed for the library which records 500 samples of data from the gyroscopes (while the device is at rest on a still surface), and uses the average values as the zero calibration. The standard deviation of the jitter is also recorded. When the library is used, any gyroscope readings less than 3.5 times this standard deviation are ignored. This is done per axis because, as mentioned, the different gyroscope axes have different jitter characteristics.

3.2.2 Orientation Tracking with Automatic Reset

The library can output both rotational velocity and orientation. The pitch and roll components of the orientation are reset using the accelerometer values when all of the following are true:

- The magnitude of the accelerometers vector is 1 or less for at least two consecutive updates
- The rotational velocity has been below 10 standard deviations of the jitter data for two consecutive updates
- The high speed gyroscope mode has been used

The first two conditions attempt to ensure the controller is still enough that only the gravity response component is being reported. The last condition prevents resets when they are not necessary.

3.2.3 Acceleration Due to Movement

The gravity component of the acceleration data is removed leaving only the acceleration due to movement. This is achieved by rotating the gravity component using data from the gyroscopes. The gravity component is reset when the conditions listed above are met. This method suffers from the errors describe previously.

3.2.4 Multiple Controllers

The library developed has the ability to support as many controllers as can be connected via Bluetooth. The library monitors all connected controllers and makes their data available. Unfortunately, due to issues with the Microsoft Windows Bluetooth implementation, it was never possible to reliably connect more than a single controller.

3.3 Summary

While it is theoretically possible to generate absolute position from the Wii controller, the level of tuning required and the resulting low accuracy mean this is not the best way to utilise the controller. It also means it would be difficult to use the controller in existing interfaces. While the orientation data produced is more accurate, it still suffers from drift and the yaw component cannot be reset. For these reasons and some design considerations (sections 4 and 5), rotation velocity is used as the input to the custom interface developed.

A software library was developed which allows simple access to the Wii controller data. An auto calibration method was developed to provide superior calibrations to those used by the 'WiiYourself!' library. These calibrations allow the developed library to provide accurate orientation data or rotational velocity data with reduced jitter. The poor accuracy of the positional data generated by the accelerometers meant that the Wii controller could not be used with traditional 3D interfaces. Therefore, the custom 3D interface developed for visualization applications was designed to operate solely on orientation/rotational velocity data.

Designs for two handed control, involving multiple controllers, had to be abandoned for technical reasons.

4 Developing a Suitable Menu Interface

The limitations of the Wii controller mean that it cannot easily be used with a traditional 3D interface. Existing interfaces are also inefficient due to their lack of constraints. This chapter contains information regarding the development of an interface for manipulating a visualization using the Wii controller. It details the key decisions made in the design of the interface with regards to the demands of a visualization system, the limitations of the Wii controller, and the feedback received from users during prototyping. The development of the completed interface is described and the novel features are highlighted.

4.1 Interface Requirements

The following requirements were identified for a visualization interface:

- The interface must allow the selection of modules and adjustment parameters within each module. Parameters can be:
    - Choices (including binary on/off choices).
    - Values (continuous or discreet).
    - Symbolic input (text).
- Menus must be able to support an infinite number of menu elements and menu nesting to an infinite depth.

Symbolic input is not provided by the developed interface, this is an input method which is much better suited to a keyboard. However, consideration for a full onscreen keyboard operated by orientation is made; see dial-pad in section 4.3.3.

4.2 Design Considerations

In addition to those presented in section 2.4, the following considerations were identified:

- The interface should obscure the users view of the visualization as little as possible,
- A user should be able to switch between menu navigation/parameter manipulation and viewpoint control at any time.

The notion of constraints was heavily used. Two tasks were required by the interface: selecting an item(s) from a list and selecting a value from a range. Strictly speaking, both of these tasks require only 1DOF. Selecting items from a list *could* be implemented as a 2D item array. This would allow more items to be displayed at a time than a 1D list and the time taken to select an item might be reduced because the maximum distance between elements is reduced. For example, a menu with 9 elements has a maximum distance of 9 in a straight line compared to 4.2 ($\sqrt{3^2 + 3^2}$) if the elements are laid out in a square. However, value selection for a single parameter can only be a 1D task. Menu selection was constrained to 1DOF for the following reasons:

- Consistency; a linear menu matches the control and aesthetics of the value selection control.
- Screen space; a 1D menu takes up less screen space than a 2D menu.
- Constraints; a user can theoretically be more accurate when required to use fewer DOFs
- Subgroups; by using subgroups (AVS/Express modules have already been identified as valid subgroups) the benefits of a higher dimensional menu are reduced. Subgroups reduce the number of elements which need to be displayed in

any given menu. Subgroups reduce the maximum distance between menu elements but also increase time spent traversing menus. If elements which are used together frequently are provided in the same subgroup then any menu traversal time is also reduced.

Therefore, the entire interface (with one minor exception see dial-pad in section 4.3.3) is constrained to 1 DOF. This is more constrained than both a traditional mouse interface which uses 2 DOF and the VRMenu interface used with a traditional tracking device which requires 5DOF.

User comfort was also heavily considered when designing the interface. Thanks to the advantages of inertial sensors (not requiring line of sight or to be in range of an emitter) and the design of the interface, the user can operate the interface sitting down and with minimal movement. Control of the interface and the camera is based on rotational velocities rather than orientation. This means that the users can adopt any pose as their neutral pose. For example, the controller does not need to be level in the physical sense for the selection to be central in the menus. The user can operate the interface with the controller pointed to the floor as their neutral pose if they choose; the same interactions will still work (although they may not be as intuitive for a first time user). The range of movements required can all be achieved using wrist rotations. Therefore, the user's arm can remain stationary allowing them to rest it upon a support. The limited range of movement required also increases the user's safety when using the interface. These considerations do not mean that the interface cannot be used standing up at a large distance away from the output device. In fact, the above example shows that when a user is standing, the interface still offers comfort. By using the device pointed down as the neutral position, the user does not have to hold their arm out to use the interface when standing, reducing strain. They can still use all of the benefits of 3D control to interact with the camera at the same time.

4.3 Implemented Features

Based on the requirements of a visualization system, three menu elements were created:

1. Menu hierarchies, allowing elements with similar functionality to be grouped together.
2. Choice rings, providing support for options. Can be configured to allow selection from any number of choices.
3. Value selection rings, for choosing a value from a specified range.

The design and operation of these elements, including the various additional features designed to aid the user, are described below. The controls for the interface are shown in Table 1. The home button is used to allow the user to change between the menu interface and camera control at any point.

Button	Menu / Choice Ring	Value Selection Ring	Out of Menu
A	Select menu items / Toggle choice item	Select current value	Adjust camera rotation (hold)
B	Go back one menu level	Close selection ring	Adjust camera position (hold)
Home	Show/Hide the menu		
1		Specify zoom region (hold)	Enables camera centre selection mode (A button to confirm)
2		Opens dial-pad	Switch to direct control camera
+		Increment value	
-		Decrement value	

Table 1: The various controls used in the custom interface.

4.3.1 Menu Hierarchies

The concentric Spin Menu was used as a basis for the design of the menu hierarchies. In order to minimise the amount of screen space the menu required the rings used were elongated and oriented almost parallel to the user. A larger ring means that the same amount of menu elements can appear on screen but the menu takes up less space as the visible curvature is reduced. In the final design the curvature of the ring is almost unperceivable (when using a 2D output device) but the menu ring and its elements are in fact 3D objects. The menus can be oriented either horizontally or vertically. Vertical menus were used in the evaluation because standard output devices are wider than they are tall. Thus the number of available on screen elements was sacrificed to increase the amount of space available to view the visualization.

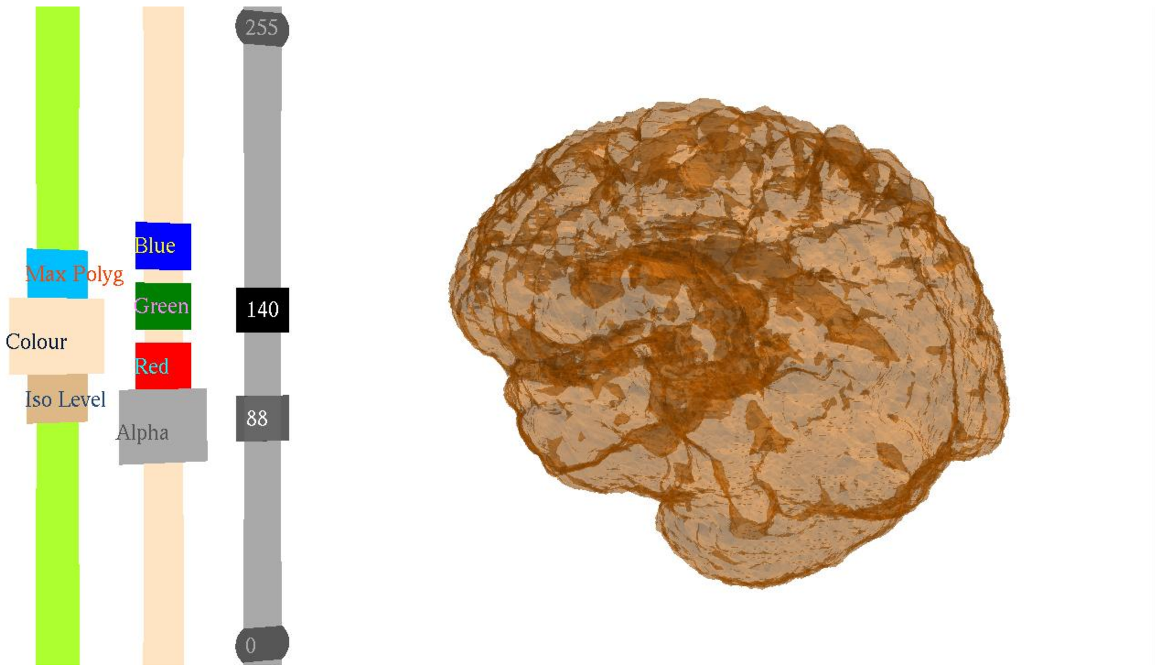


Figure 7: The menu hierarchy interface and a value selection ring showing the visual feedback used. Previous menu choices are highlighted. Rings are the colour of their parent menu choice. The limits of selection, current selection and stored selection are all shown on the value ring. The currently open menu is the largest amount of screen space the menu can occupy (with the exception of the dial-pad, section 4.3.3).

Several visual feedback techniques, shown in Figure 7, are employed to aid the user navigate the menus. The menu elements themselves are different colours and for submenus, the colour of the ring itself is the same as the menu element which opened it. This additional visual information is designed to reduce the memory load on a user. Submenus ‘grow’ out from the parent menu with a smooth transitional animation and the element selected in the parent menu remains highlighted. This indicates which submenu the user is navigating, analogous to how a 2D drop down list submenu grows from its parent and the parent remains highlighted. When the user exits a menu, the menu simply disappears, no animation is used. During prototyping it was found that while users appreciated the animation on the opening of menus, when they were exiting a menu, the animation was unnecessary and was slowing down use of the interface. When the user navigates to menus more than 3 levels deep the entire menu shifts to remove the highest level menu from the screen. This ensures that even with an infinite number of submenus, the menu cannot grow to obscure the screen. This shift is again animated smoothly to indicate to the user what has taken place. The menu is shifted back into position when the user returns towards the top level menus.

Control of the menus is linked to changes in the pitch of the controller (or yaw if horizontal menus are used). The direction of physical movement matches the movement of the selection. Menu elements are highlighted as the selection passes over them. The movement speed of the selection was configured to allow the user to reach the top and bottom of the menu within the comfortable range of hand orientation. To support an infinite number of menu elements some elements must be off-screen at any given time. Therefore, for the user to access these elements comfortably a virtual travel type of control is needed. When the user reaches close to the top of the menu (and the limit of their comfortable range) the menu scrolls in the opposite direction to reveal the hidden elements. The speed at which the ring rotates is dependent on how far the user goes beyond the limit; tilting the controller further makes the ring spin faster. When the user returns to a position below the limit the ring becomes stationary and control over the selection is resumed.

An alternate method of control was implemented where changes in the orientation of the controller controlled the orientation of the ring with the selection area remaining static. The previously described control was preferred because users found it more intuitive, it matched the control method used for the value selection rings, and because the virtual travel technique used above is confusing to users with this system. If a user tilted the controller to the limit and the ring continues to spin (with the selection area remaining in the middle of the screen), when the user comes to the area they require and tilts the controller back past the limit, the controller is oriented near the limit but the selection area remains in the middle. To move the ring so as to select the next item, the user cannot simply tilt the controller because this will take them beyond the limit and trigger virtual movement again.

4.3.2 Choice Rings

Choice elements are derived from standard menu elements. The elements are added to a menu and are displayed in the same way as sub-menu elements. Each ring of choice elements can be configured as either a check or toggle choice. Toggle menus allow only a single element to be selected at a given time while check menus allow any number of elements to be active. The check menu has the advantage of allowing multiple binary decisions to be grouped onto a single ring. Toggle menus provide support for any decision with greater than two options.

4.3.3 Value Selection Rings

Value selection rings can be configured to support any range of numbers and any increment size. For example, both a continuous selection from 0 to 1 and a discreet selection from 0 to 255 in steps of 1 can be configured. The limits of the range are displayed at the limits of the ring. The stored value and the current value at the position of the cursor are also both displayed. While the user moves the controller the current value is updated on the screen, this may seem trivial but can improve a user's performance see section 7.3.3.

The value at the cursor is committed to the stored value when the user presses the A button. If the A button is held, the value at the current position is continuously stored. This can be disabled which is useful in instances where committing a value triggers an expensive computing operation. This is common is visualization where altering certain values can cause the entire geometry of the scene to be recomputed. If the user was allowed to alter the value continuously, large blocks of computing would be queued causing the system to become unresponsive. (This is not the case in my simulation environment as any current work is stopped if a new value is selected. However this may not be the case for example, in AVS/Express).

This ring offers increased efficiency when compared to a 2D slider bar projected into 3D. Repeated value adjustments are expected in visualization applications as parameters are explored. Using the value selection ring, once a ring has been chosen (using 1DOF in the menus), multiple adjustments can be made without needing to reselect the parameter. In a converted design, 5DOF are required to select the slider bar's current position (typically a small target) and for each subsequent value change the bar must be selected again.

Two advanced controls are included to aid the user in selecting a value, zooming and a dial pad.

The movement speed of the current selection area is calculated based on the range. This means that the user can navigate the range 0 to 200 with the same degree of movement as navigating the range 0 to 5000. This ensures the user does not have to move the controller outside of a confortable range, minimising strain. However, when a large range is used, small movements of the controller cause large changes in the value, making it hard to pick a certain value precisely. Zooming allows the user to specify a new range by selecting a region of the current range. By choosing a smaller range the movement speed is reduced and the user can be more precise with their selection.

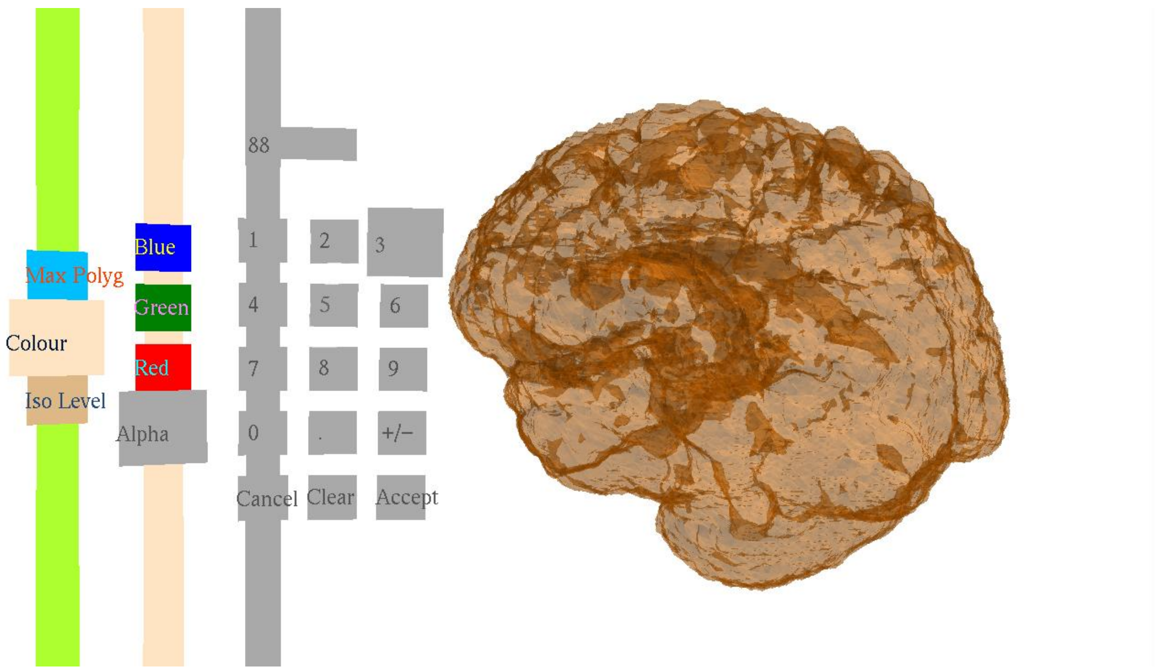


Figure 8: The dial-pad interface can be used to input the desired value

Another option available to the user is to open up a dial-pad (Figure 8). This is useful if the user knows the exact number they want to enter or if they want to enter it to a high level of precision. A menu with buttons for each of the ten numeric characters as well as buttons to clear the value, add a decimal point, cancel the entry, accept the entry, and a negation button is presented. The user navigates this menu in the same way as the menus except that the yaw of the controller is now used to move across the rows of the menus. If necessary, the author feels this approach could be extend to include all alphanumeric keys for true symbolic input. Unlike the menu hierarchy, a 2D menu structure was deemed to be superior to a 1D structure in this case. There are a number of reasons for this:

- A 2D structure means the familiar 3x3 numerical layout can be used.
- The dial-pad is only open briefly (in comparison to the menu), and while it is open the visualization cannot change. Therefore nothing is lost by obscuring the users view.
- Unlike a menu hierarchy there is no relationship between the menu elements, all buttons have an equal chance of being pressed, and any sequence is equally likely. Therefore you cannot use submenus to make sure that related items are close together in the menu. As such with a 1D structure there may have been much traversal of the menu between selections. This problem is compounded by the fact that the fifteen elements required could not all be displayed simultaneously in the 1D menus used.
- Users will select multiple items within a short space of time, unlike in the menus. Therefore, distance between elements must be minimised to improve efficiency.

4.4 Background Menu Movement

The decision to continue showing the previous level of a menu to aid the user presents an interesting problem when the user navigates between menu levels. The initial selection when the user enters a menu can cause both physical strain and confusion. Physical strain occurs when the initial selection is in the opposite position to the current orientation of the controller. For example, if the controller is pointing as low as is comfortable for the user and a new menu is entered where the initial selection is the top most menu item, the user cannot select any of the lower menu items without strain. This is easily solved when advancing forwards in the menus by passing the controller orientation information forward (Figure 9). The user enters the menu at the same position as they left the last one and can navigate the full range of the menu without strain.

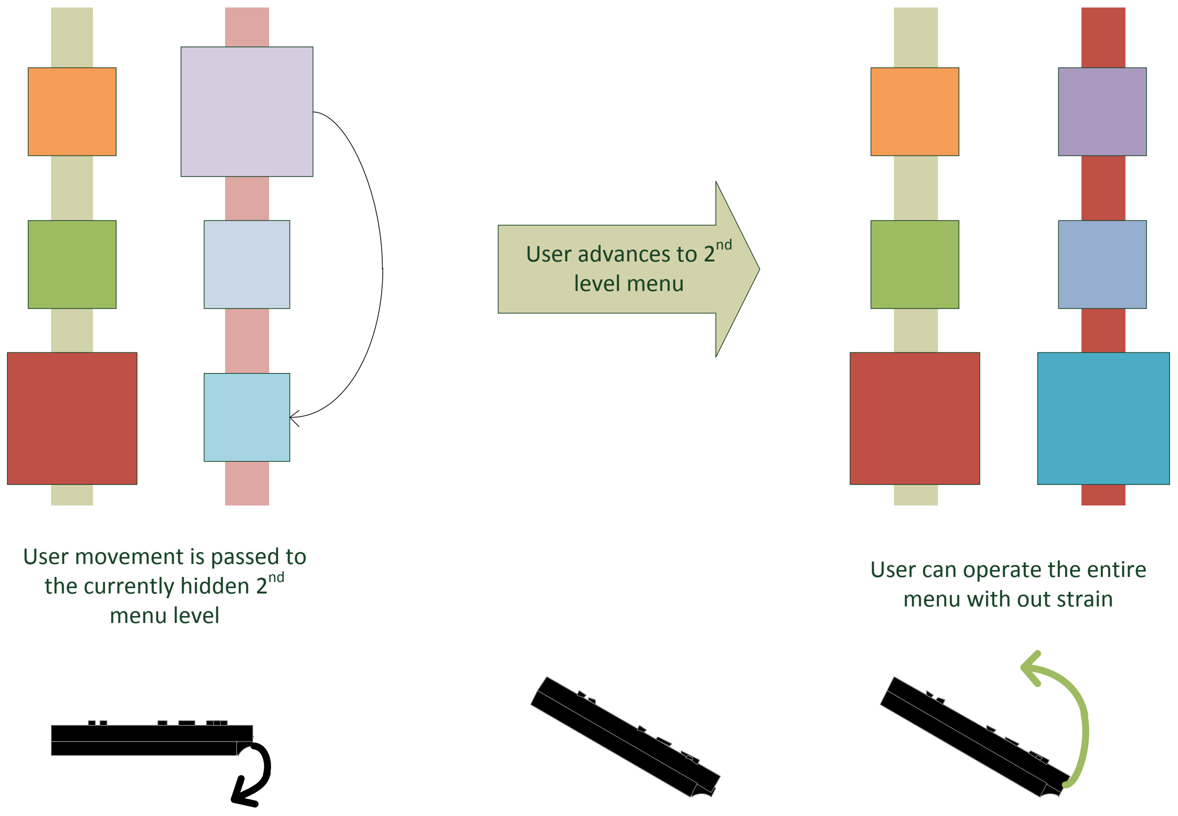
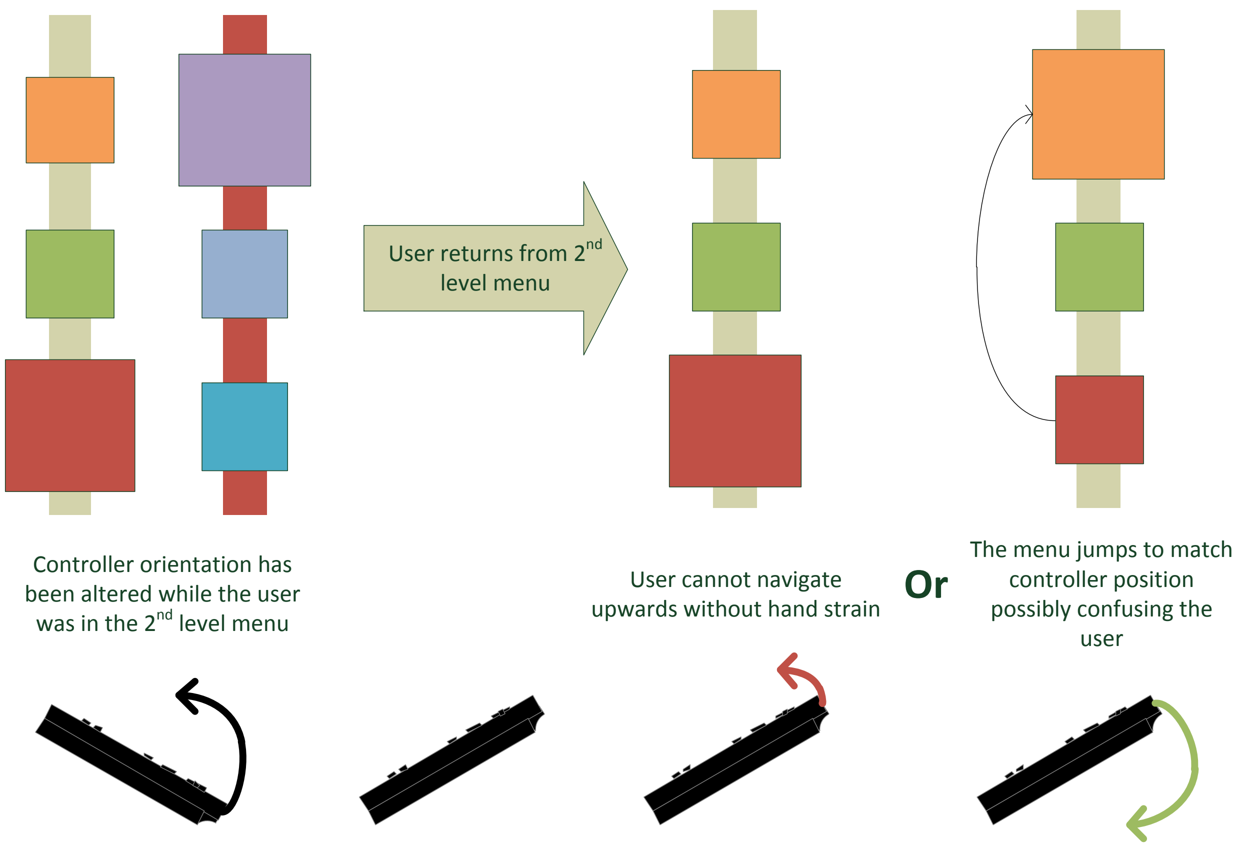

User advances to 2nd level menu
User movement is passed to the currently hidden 2nd menu level
User can operate the entire menu with out strain

Figure 9: Background movement when opening a menu.

However, when the user returns from a menu, the problem is not so easily solved, see Figure 10. Because the previous menu selection is displayed on the screen, the user expects that the current selection will be centred on that item when they return to that menu. In the evaluation of the interface, the system was configured such that the selection changed from the previously selected item to a position where the users hand orientation was taken into account. This gave the impression that the selection was moving 'on its own', causing users to compensate by tilting the controller in the opposite direction to the movement. The effect is more apparent on menus with fewer elements. As the animation does not highlight each element along its path, a large menu makes it is easier to see that the position jumps, with a small menu it can appear as if the position is moving.



User returns from 2nd level menu
Controller orientation has been altered while the user was in the 2nd level menu
User cannot navigate upwards without hand strain
Or
The menu jumps to match controller position possibly confusing the user

Figure 10: Problems when using background movement as a menu is closed.

This is fine in the sense that orientation has been taken into account and hand strain has been avoided. However, some users frequently moved off the item they required by compensating for the movement, forcing them to move back to the original position to reacquire the target, reducing the efficiency of the interface with unnecessary movement.

The author proposes two ways to alleviate this problem:

- Alter the animation or simply do not animate the automatic movement. If the selection instantaneously changes to the correct position rather than appearing to move there, the user will not feel like they are causing the automatic change in selection.
- Have a secondary visual effect to signify which element the user *would* be selecting based on their hand orientation. The current selection can then animate to this point with the user knowing why it's moving and where to. The user can then continue to navigate the menu while the animation catches up. This secondary visual effect can be hidden when the current pointing position and the current select are synchronised.

Users who were more experienced at using the system and understood what was occurring when a menu was exited suffered less from this problem. Some users used the system to their advantage. By changing their hand position slightly prior and during the exit of a menu, these users were already targeting the selection they required when the animation completed.

4.5 Summary

The developed interface uses rotation velocity to allow a user to control a visualization application in 3D. The design considerations identified were followed to create controls for menu navigation, choice selection and value selection.

- The efficiency of the design for value selection is better than converted 2D designs which use 3D pointers for control.
- The input of the 3D device is constrained to match the tasks (1DOF used at almost all times).

- The range of movement required to operate the interface has been minimised to reduce physical strain. Strain is further reduced by the use of relative orientation.
- Increased utility, in the form of zooming and a dial-pad, has been provided for the commonly used value selection ring.
- Use of visual feedback attempts to reduce the memory load on the user.

The key issue of background movement has been identified when using rotation velocities instead of absolute orientation for control.

5 Developing a Camera Interface

As discussed, it is possible to correct errors in the orientation output more so than positioning data. Orientation from inertial sensors has therefore been used more often in existing systems. For example, inertial sensors have been used to track head orientation for use in virtual environments. This is a form of viewpoint manipulation using inertial sensors. However, attaching the sensors to a user's head reduces the sensors ability to be used for application control tasks. This method is also less than ideal for visualization as to orbit an object of interest the user would have to be constantly physically spinning around.

A different approach was therefore taken to operate the camera in this interface. This chapter describes two camera modes used in the custom interface and some of the problems encountered.

5.1 Orbit Camera

A standard virtual camera within a scene requires 6DOF to control fully. In many applications, the camera is often constrained in some way based on the type of application and the control devices in use. For example, computer games with a direct control camera (sometimes known as first person camera) are often limited to 4DOF. Users do not typically need roll their heads or significantly alter their Y-position above the floor (i.e. fly), and this maps nicely onto the typical dual joystick (4DOF) controllers used.

In visualization applications, the standard camera manipulation use case is to explore a 3D model. The model is not positioned in a scene, or virtual environment, it is typically the only object rendered. To make exploring the model as efficient as possible the focus must remain on the model. As such, in this design, the centre of the camera (the position it is directed at) has been fixed within the model itself. Movement of the controller affects the rotation of the camera around this point. Because of the duality of camera systems, if the direction of rotation is reversed this is equivalent to the camera being fixed and the object rotating in the same direction as the controller. This was found to be more intuitive than rotating the camera during prototyping.

Using this system there only needs to be 1 positional control. Because the centre is fixed, movements of the camera up/down and left/right are analogous to rotating the camera in those directions. Movement forwards/backwards is equivalent to zooming in/out to the model. Because positional data from the Wii has been discounted, this 4DOF (roll, pitch, yaw, and zoom) camera was mapped onto the 3 orientations of the Wii controller. The rotational controls are mapped as expect and zoom is mapped to the pitch of the controller when a different button is held.

The problem with this camera model is the positioning of the centre. Having a fixed centre makes it impossible to view the model from certain angles. Without positional data a novel approach had to be taken. Two methods were tested for moving the centre within the scene. The first used the movement controls from the direct control camera model to control the centre; however this method has some usability problems (section 5.2 below). The second method uses the idea that the position of the centre in the x and y positions (relative to the viewpoint) can be set by rotating the camera, the only problem is setting the depth of the centre along the z-axis, into the screen. Two methods were implemented to solve this problem:

1. Attach the camera to the centre as the user moves it along the z axis. This allows the user to see how far they have travelled. This approach works well for hollow 3D models or positioning the centre outside of the model. However, the types of

model used in visualizations are typically volumetric and thus have geometry within in them. This can sometimes make it impossible to tell how far in the model a user has actually travelled as their view becomes obscured.

2. Render a plane aligned with the screen positioned on the centre. As the centre (and thus the plane) travels away from the user more of the model is revealed through the plane allowing the user to tell how far the centre has travelled though the model. This method only works well for selecting up to the point within the model which corresponds to largest part of the model from the current camera perspective. After that no more of the model is revealed and thus the distance travelled cannot be determined.

A further method is proposed but was not implemented:

3. Show another view of the scene offset from current camera. This would allow the user to view the centre travelling along the z axis. The problem now becomes how to position the camera. If it is to be positioned automatically, it is necessary to know the size of the model so that the camera is not positioned inside the model, too far away so that the user cannot be precise, or too close such that the entire model is not in view. The first problem with this is that the size of the model may not be easy to determine. The size of the data being used is known but the visualization options may only be causing parts of this data to be rendered. The second problem is that the user may wish to alter the view, for example to zoom into a large model to precisely position the centre. This means giving the user another set of commands and controls to remember. In testing it was found that even implementation 2 above required too many controls and actions for a first time user to remember.

5.2 Direct Control Camera

There are scenarios in visualization software where the orbit camera is insufficient. A direct control camera model was developed to investigate whether such a camera could be controlled using just orientation data. This work is also important if the Wii controller is to be used in other 3D applications where direct control cameras are the standard. The orientation of the camera is controlled as expected using the orientation of the device. A second mode was introduced to allow the same orientation data to be used to manipulate the position of the camera. The data mapping here is somewhat arbitrary because the physical motions (rotation) do not match the desired result (translation). However, the configuration which felt most natural was using yaw to control left/right movement, pitch to control forwards/backwards movement, and roll to adjust the height of the camera.

Because the motions do not match the intended result, it is difficult to manipulate all three positional components simultaneously with this method. It becomes very difficult to calculate what action will achieve the required result. In light of this, the camera is limited to moving in one direction at a time. When rotation in the controller is detected the component with the largest value is selected as the direction of travel and this direction is locked until the user releases the control button. Initially the direction was unlocked when the controller was still; however, this did not test well. When users paused momentarily for any reason and the direction became unlocked it became confusing. When the user resumed their rotations expecting to continue in the same direction their rotations were often fine adjustments making it more difficult to distinguish the required direction. Therefore, a different direction could be selected and the user would have to pause to unlock and try again. It was more efficient for the user if they had manual control over the direction lock.

5.3 Virtual Travel

In virtual reality systems the main concern for camera control is how to travel large distances through the scene without the user having to actually physically move the same distance. Obviously physical space is limited (when compared to virtual space) so the user cannot move around great distances to move the same distance in virtual reality. While positional data is not being used in this interface, the same problem is encountered. These systems typically use ‘virtual travelling’, where input corresponds to velocity rather than position. If the visualization model is very large the virtual travel may become necessary for zooming when using the orbit camera. Virtual travel is also necessary for large movements using the direct control camera. Two methods were implemented, both with unsatisfactory results.

1. If the orientation of the controller does not alter the position of the camera but rather the velocity of the camera, the user can travel at the speed they require. This allows them to cover large distances quickly. However, this technique does not match the camera orientation controls. Users found it confusing when forced to change between direct control of camera orientation and velocity control for position. Users also found it harder to stop the position movement where they required even when a larger dead zone had been added to the controls (a region of orientation where the velocity is set to zero).
2. The system used in the menus of the custom interface was applied to travel. Orientation directly controlled position until the user passed a limit. Distance past the limit then controlled the velocity of positional movement. The problem with this approach occurs when the user attempts to end virtual travel. Users expected that velocity would return to zero when the controller was level. Initially this was not the case, virtual travel ended when the users crossed the limit. Therefore as

the user continued towards levelling the controller the camera moves in the opposite direction as direct control has resumed. When the controls were modified to disable travel once the user levelled the controller the users expectation differed again from the reality. Because virtual travel had been active as they passed the limit and returned to the level position, when the user tilted the controller back towards the limit they expected this to be controlling velocity, when in fact direct control had been resumed.

The second implementation shows promise but requires more work to match users expectations. Another possible option is to have both direct control and velocity control but keep them totally separate. The user can then decide what is needed. However, this does add additional complexity to the controls.

5.4 Relative Camera Movement

A virtual camera within a scene has a position and orientation, and as such, control of the camera can be mapped naturally onto a standard 6DOF controller. For a system with a head mounted display method and head tracking it is easy to see that this mapping makes sense, the user simply moves around the scene as if it was a real environment. However, while a one-to-one mapping of the controls is a natural way to map the controls, it may not always be the easiest or most intuitive system to use. A system with a fixed screen would require the user to manipulate the controller into unnatural positions in order to rotate the camera fully. When using a hand held device the user will expect to be able to relax and not have to maintain a pose to hold a camera fixed. If the systems allows the user to disconnect control for these periods, when control resumes the camera will jump to a new orientation if the controller is not in the previous position, which could be confusing for the user. This situation also arises when interacting with menus, if the same device is used and camera control is not disabled the camera will be altered by menu interaction. Likewise, if it is disabled in-menu, when the user exits the menu the view will be altered.

The camera controls in this project are therefore all relative rather than absolute. Changes in orientation the user makes while holding down a button are added to the current view. To ensure that rotations remain related to the current orientation of the controller (Figure 11), rotations are stored in a temporary value while the button is held. The temporary rotations are multiplied into the scene when it is rendered to ensure the view updates with the user's movement. However, the temporary value is not multiplied into the total stored rotation for the scene until the button is released.

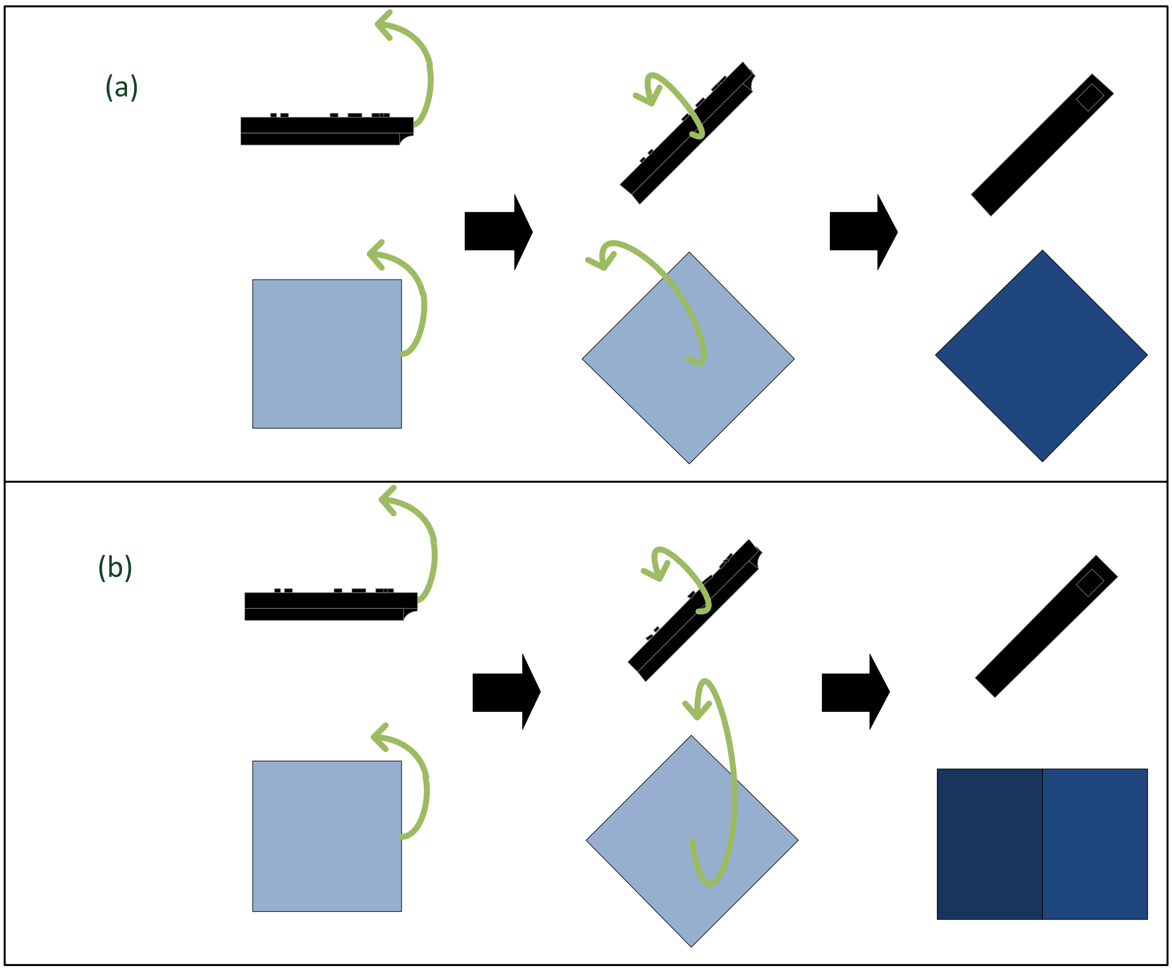


Figure 11: The effects of controller rotation on an object within the scene; (a) the effect when using temporary rotations, the rotations are relative to the controller, (b) the effect when all rotations are stored as they happen, the rotations are relative to a fixed coordinate frame.

5.5 Summary

The camera controls used in the custom interface use rotation velocity to manipulate the viewpoint. Two camera modes are offered; the orbit camera which should cover the majority of viewpoint manipulation within a visualization application and the direct control camera for the minority of times when the orbit camera is insufficient.

Selecting the centre of the orbit is identified as a major problem when positional control is not available. More work is needed in this area to develop a viable solution for all situations.

Emulation of position using orientation was implemented using virtual travel with varying results. Again more work is needed, mainly identifying what a user expects from the interface.

As with the menu interface, strain has been minimised by using relational orientation input. At its simplest the camera only requires two buttons to operate reducing memory load on the user. Efficiency of the camera orientation control is better than that of a mouse interface, all three camera orientation components can be adjusted simultaneously (compared to two with a mouse). However, control over the camera position is less efficient than both a mouse interface and an interface which uses position data from a 3D tracker. Only one position can be adjusted at a time compared to 3 with a 3D tracker and 2 with a mouse. The specific needs of a visualization application (exploited by the orbit camera) mask this deficiency.

6 Integration with Existing Systems

The test system available for the project typifies a visualization setup for complex data sets and also for other immersive 3D systems. The visualization calculations; filtering, mapping and rendering, are performed on a high performance computer while the user input and the output of the visualization is delivered to/from a standard desktop. This allows the user to work with large data sets from the comfort of their own desk without needing a large computer/server in the room. A framework for using novel input devices with immersive interactive tools is used to standardise this interaction.

Trackd [27] is a standard framework which enables the sending of 'tracker' data collected on a client computer to a server computer. There are two benefits in using this standardised framework. Firstly, a controller integrated into the trackd client software allows it to control existing interfaces with support for trackd. Secondly, any interface designed with trackd support can make use of any device which has trackd support. This acts a layer of separation between the hardware control and the interface, preventing code cross over and making sure the interface is not tied too closely to the hardware.

The trackd framework operates by storing input data to a shared memory address. Applications then read from this memory location to access the data. To operate on remote systems, a network address rather than a memory location can be used. Another instance of trackd is run on the remote computer which reads data from the network and stores it in shared memory.

Software which utilises both ends of this framework was developed during this project. The Wii controller library reports its data to the common framework and the custom interface reads its controller inputs from the framework. The extent of the integration into this existing framework and some of the limitations are described in the remainder of this chapter.

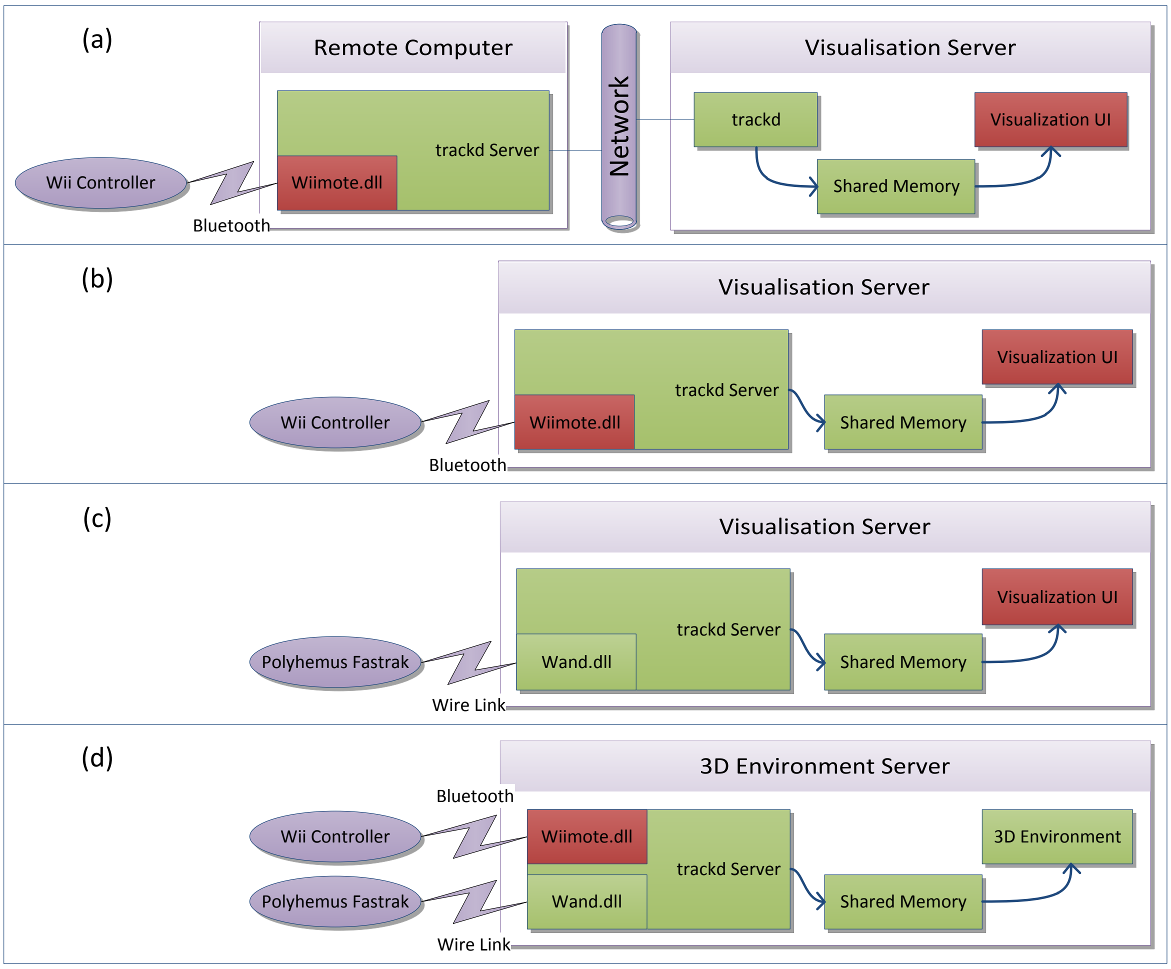


Figure 12: Some possible combinations of hardware and software made available using the trackd framework (elements in red were written as part of this project): (a) the custom interface using a Wii controller connected to a remote computer; (b) same as (a) but with the device connected locally; (c) existing hardware connected to the custom interface; (d) the Wii controller connected to existing software.

6.1 Wii Controller Integration

The Wii controller library was written using the trackd SDK which allows the trackd system to read the data of the Wii controller (Figure 12 (a) (b) (d)). The 7 buttons of the controller are reported as well as the 8 way directional pad which is reported as 2 valuators. Where trackd expects position and orientation data, the raw accelerations and rotational velocities are reported respectively. The library was written this way such that the developed interface could work with the raw controller data. Unfortunately, this prevents the device from being used with existing tools expecting position and orientation. To integrate with existing tools, the library can be configured to report orientation instead of rotational velocity. This would allow the device to be used in any interface which required orientation data and/or a set of buttons and has support for trackd. Although this integration is possible it has never been tested.

The library is also configurable to allow a number of gestures to be sent as binary data. The gestures are sent as additional buttons allowing them to be used by any existing software like any other button. To test this functionality a set of gestures which were simple thresholds of the acceleration data in each dimension were used and successfully transmitted to another computer. Neither the final library nor the custom interface make use of this functionality.

6.2 Interface Integration

The custom interface uses the trackd API to read input data from a shared memory location (Figure 12 (a) (b) (c)). The data can come from any device supporting the trackd interface, theoretically allowing any of these devices to be used with the interface. In reality two restrictions apply:

1. The device must report either rotational velocity or absolute orientation
2. The device must have at least 3 buttons

However, these restrictions need not be fulfilled by a single device. A number of devices can be combined using trackd, for example, a pure orientation tracking device with a single button could be used in combination with any device which has a further two buttons.

A special version of the user interface was developed to accept absolute orientation data instead of rotation velocities. This allows devices which report absolute orientation (such as the Polhemus Fastrak) to be used with the developed interface. The software converts the absolute orientation data into rotational velocities simply by differentiating the input. The difference between the previous orientation and the current one is divided by the time between updates.

The button limitation is to ensure that the basic menu functionality can be achieved, see Table 1. The 3 buttons are used to emulate the A, B, and Home buttons on the Wii controller which are required for basic use of the menus. A further 2 buttons emulate the 1 and 2 buttons, if present, for some of the more advanced functionality. A device with 7 buttons has the same functionality as the Wii controller, with the last two buttons emulating the + and – buttons. This system has been successfully tested using the Polhemus Fastrak system, with the basic 3 button functionality.

6.3 Summary

Thanks to integration with trackd, the custom interface is able to accept input from a number of devices, connected both locally and/or remotely, provided a small set of restrictions are met. On the other side of the interface, the Wii controller can be used with any tool which supports trackd. Because of a lack of positional data, the extent to which this would be useful is dependent on the individual tool.

The good ergonomics, large number of buttons, price, and wireless interface of the controller make it an attractive device to use in combination with other devices within trackd as described earlier. It could be easily used to add a set of buttons into a system with devices which may provide tracking data but no buttons, for example, a head tracker.

7 Evaluating the Interface

The project followed a usability engineering process [28] shown in Figure 13. This methodology involved task analysis, expert analysis, a cycle of formative evaluation and improved prototypes, and finished with a summative evaluation.

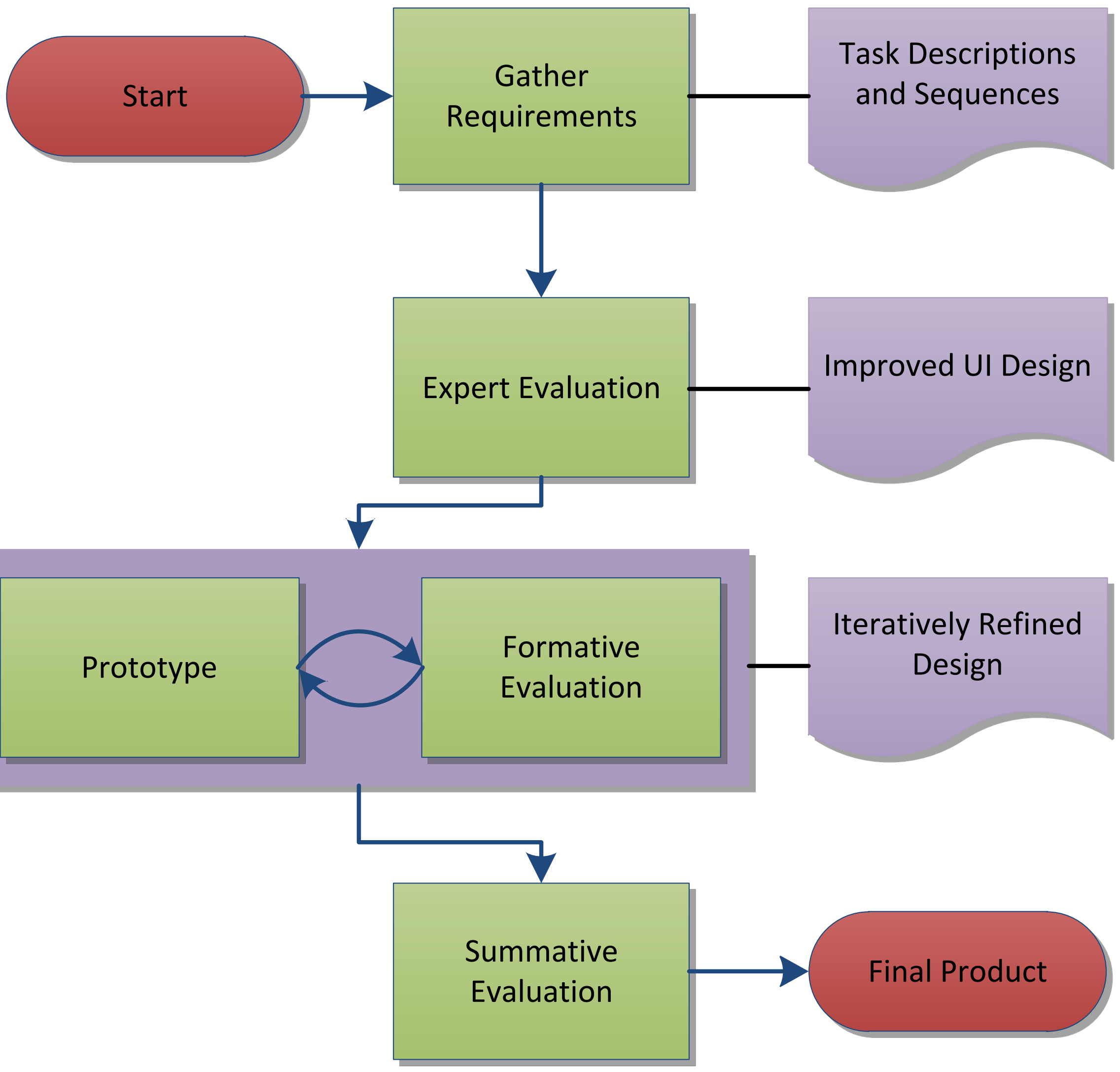


Figure 13: Usability Engineering Process **[28]**

The requirements for the interface have already been discussed in section 4. That section also described the decisions taken from the iterative prototyping and formative evaluation cycle which took place. During the evaluations in these cycles, users were given tasks to carry out using prototypes of the interface. This was followed by an informal discussion with the group of users to establish which interaction method, out of a selection, they preferred. This information was used to steer the direction of future prototypes.

Typically in an expert analysis phase, multiple user interface experts evaluate the design based on a set of guidelines in order to produce an improved and more streamlined design. This system is less effective for 3D user interfaces as there is, of yet, no standard guideline framework for 3D interfaces. Because of this and because such resources were not available, the design was constantly evaluated by the author based on the criteria outlined in section 2.4.

Information about evaluation techniques, details of the summative evaluation carried out, and the results of that evaluation are presented in the following sections.

7.1 Evaluation Process

There is no simple metric of performance in a user interface. There are various empirical metrics that can be used to rate performance but there are also subjective indicators such as the 'feel' or 'intuitiveness' of a design. This makes a design difficult to evaluate.

Evaluating new user interfaces can be especially difficult when novel input devices are involved. There is a question of potential users; visualization is a fairly specialised domain and also few users have experience when using 3D input/output devices. Compounding this, typically all 3D interfaces are somewhat unique; there is no one system which dominates 3D UIs in the same way that WIMP (Windows Icons Menus and Pointing devices) dominates the 2D field.

Many 3D interface evaluations focus on a user's feeling of 'presence' within the system because the applications are typically virtual reality based. This is an abstract idea which describes how immersed the user is in the experience. This is naturally hard to quantify but fortunately for the application being used this is a secondary concern. The application focuses on usability rather than immersion, visualization software offers functional tools to expose information from data. Data from simulations do not typically represent a real world scene and it is unnecessary for the user to feel like they are in a virtual world.

7.2 Summative Evaluation

A user study involving the custom interface was carried out to evaluate its performance. Qualitative information in the form of questionnaire answers and quantitative data from automatic metrics in the system were recorded. At this point the interface had already been tested to ensure that it could provide the desired functionality. The criteria for success was slightly strange in this evaluation as the custom design did not necessarily have to outperform the existing VRMenu interface (when used with a mouse or the Polhemus Fastrak Device) for it to be considered successful. The custom interface and Wii remote combination can be used in immersive environments which a mouse device cannot, and is much cheaper than the Polhemus Fastrak device. These separate advantages mean that performance needed only to be as good as/close to the exiting interface.

A set of ten users completed a small number of visualization tasks using the Wii controller and the custom interface. A subset of the same tasks was also carried out by the users using a VRMenu interface with both the mouse and the Polhemus Fastrak device.

A test program was developed containing a small subset of visualization software functionality. This program allowed the user to load different 3D data sets, select the rendering mode, and adjust the parameters for each rendering mode. Various options for altering global setting such as lighting were also included. All of this functionality was exposed using the custom interface. 6 data sets containing volumetric data [29] were used in the evaluation. Users used two of the models to acclimatise themselves with the various functions of the system and the controls. The remaining four were used in the evaluation tasks.

For the evaluation involving the VRMenu interface (shown in Figure 14), an approximation of this test program was built using AVS/Express. Because of the differences in interface style and the different level of functionality available in AVS/Express, the menus constructed did not match the custom interface identically. The evaluation was designed to test the best case performance of the custom interface against the best performance of an existing system. Therefore, the menus were constructed to play to the strengths of each system. The global options as well as some of the rendering options were removed in the VRMenu interface. The VRMenu interface was also used to evaluate the mouse as the menu controls are essentially identical to the standard AVS/Express interface but all unnecessary menus and options are hidden from the user. The camera controls are the same in the AVS/Express standard interface and the VRMenu interface. This ensures a fairer comparison with the other two input devices.

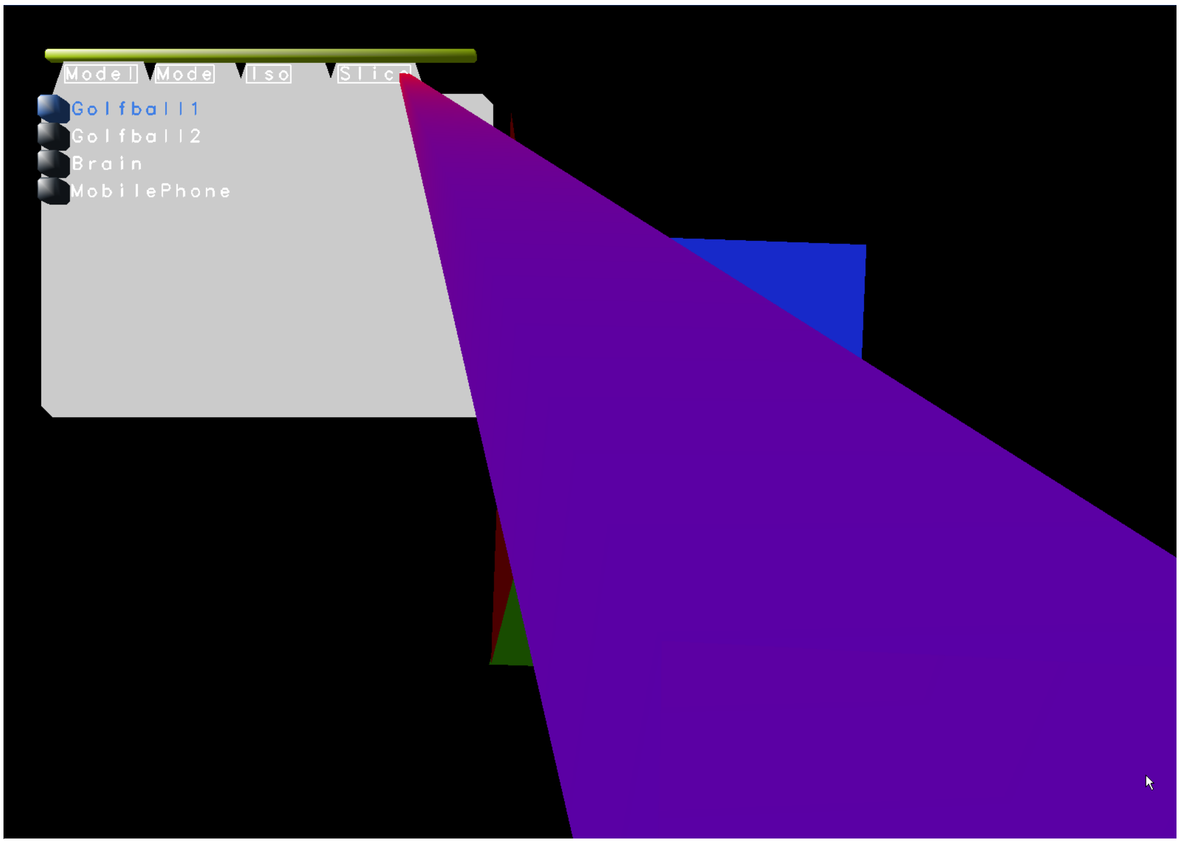


Figure 14: The VRMenu interface used in the evaluation. The image shows how not only the menu, but also the visual representation of the pointer (purple area), can take up a large area of the screen.

The test evaluations were performed with each interface running on a computer with a trackd instance receiving the user input over the network from the Wii controller/Polhemus Fastrak connected to a separate computer. This setup represents the worst case scenario for latency in the system.

The users were asked to complete 4 tasks which in total required them to use all aspects of the interface. The four tasks emulated small visualization workflows and in total, took an average of 15 minutes to complete. Performance analysis focused on 3 factors:

1. Navigating to/selecting the required menu item.
2. Adjusting parameter values.
3. Manipulation of the camera.

Each of these factors was measured by timing the users as they completed the tasks. Additionally for the evaluation of the custom interface and Wii controller the number of errors the users made as well as their accuracy (measured in the number of menu elements users missed the required elements by) was recorded. Each user filled out a questionnaire following their evaluation to gather subjective data. Users were requested to comment on:

- Each interfaces intuitiveness/ease of use.
- Particular frustrations encountered.
- The perceived responsiveness of each interface.
- Any physical strain experienced.
- Their experience with similar interfaces, devices, and the Nintendo Wii.

7.3 Results

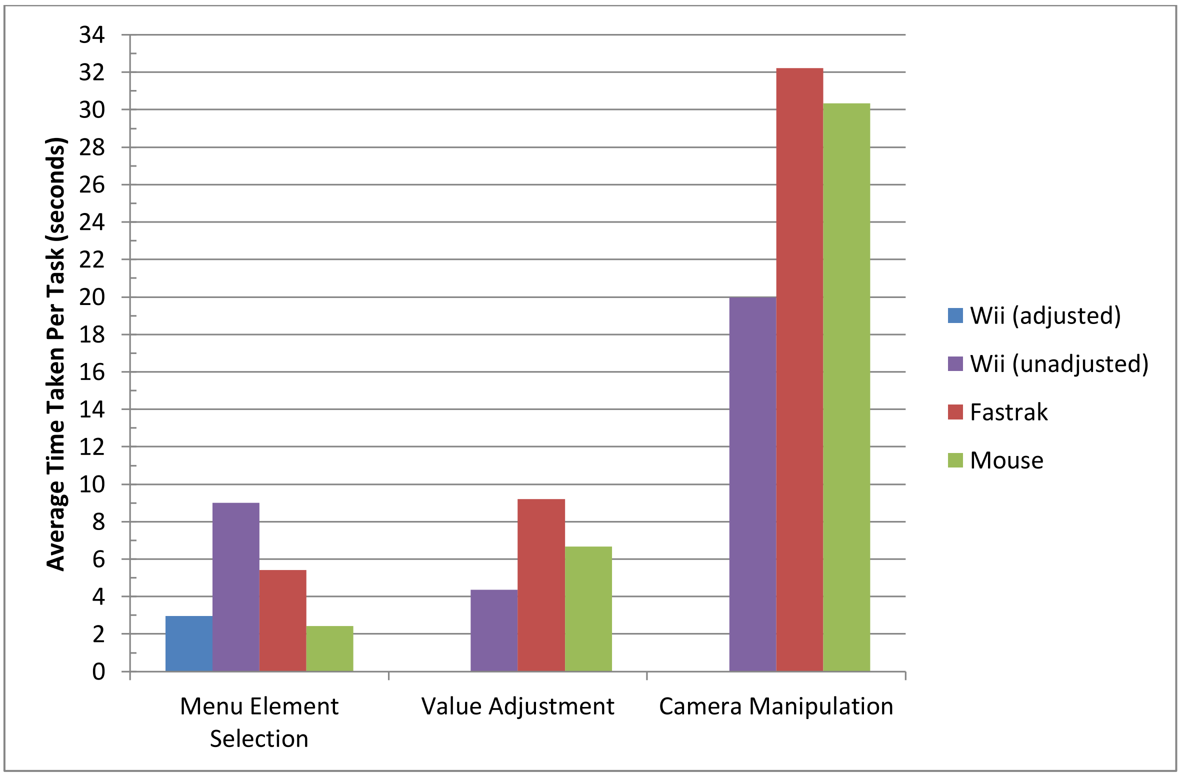


Figure 15: The average time users took to complete the various types of task. The average time for menu selection tasks using the Wii controller has been adjusted to reflect the number of number of menus traversed.

Figure 15 shows the average time users took to complete tasks using the three different systems. The tasks have been separated based on the factors above. Because of the differences in the interface styles the number of actual menu element selection tasks performed per task is not consistent. For example, to select the rendering mode menu in the VRMenu interface was a single selection because it is a top level option. In the custom interface two selections are required as the option is part of the Visualization submenu. Additionally, tasks in the VRMenu interface were timed per selection whereas for the custom interface timings were taken from the beginning of a complete task up to the final selection. To reflect this difference, for menu selection tasks, an adjusted Wii value has been included which crudely divides the time take for these total tasks by the number of selections required.

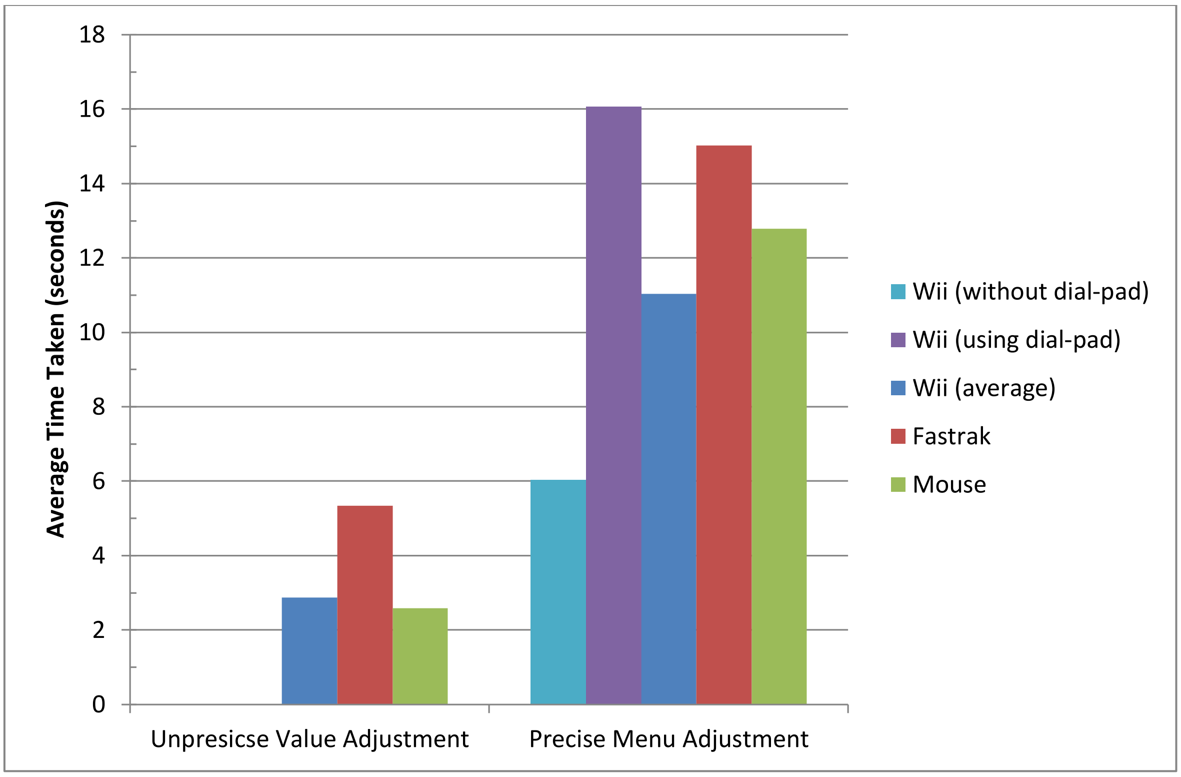


Figure 16: A breakdown of the average time taken to complete value adjustment tasks. During the evaluation users were asked to use the dial-pad feature of the custom interface to select a precise value in some of the tasks and use the value selection ring in others. The Wii average value is an average of all the value selection tasks regardless of the selection mechanism employed.

During the evaluation, the tasks required users to adjust the parameters of various properties. Some tasks required the user to select a specific value (e.g. set the isolevel to value 10) while others required the user to pick an approximate value (e.g. set the orthoslice plane value to around 50%). This was designed to reflect the exploratory nature of visualization interaction. The user may not know in advance the exact value they wish to use but are instead proceeding by trial and error to find the correct value. When using the custom interface, users were asked to use the standard value selection ring for some of the precise selections and use dial-pad for the others. A breakdown of timing results for these value selection tasks is shown in Figure 16.

	Experience Users	First Time Users	All Users
Average number of errors made per selection	0.25	0.57	0.44
Average number of inaccuracies per selection	0.25	2.04	1.33
Average magnitude of inaccuracies	0.28	2.66	2.00

Table 2: Measurments taken from users evaluating the custom interface.

The number of errors (selecting the wrong menu element), the number of inaccuracies (instances where the user missed the desired menu element), and the magnitude of those inaccuracies (measured by the number of elements missed by) are shown in Table 2. These measurements were taken only for the custom interface.

Experienced users made, on average, 50% less errors per selection compared to first time users. First time users missed their targets, on average, just over 8 times as many times as experienced users and the margin by which they missed was over 10 times larger. While the sample size for this evaluation was very small, it is clear that within that sample, experience significantly improved a user's ability to operate the interface. Fewer errors and fewer/smaller inaccuracies will lead to more efficient menu usage and a reduction in operating time. Therefore with experienced users it may be expected that the menu selection task times would have been reduced for the custom interface and Wii controller system. The same observation may be true of the VRMenu interface and the Polhemus Fastrak system as most evaluation participants had not used these systems before. However, the VRMenu interface is a projection of a 2D style interface into 3D and as such users may have been more comfortable using the interface as they have experienced this style of interface before. Therefore, the possible improvement in selection times gained from experience may not be of the same magnitude for this interface. All users have extensive experience in using the mouse device and thus improvement from further experience for this interface/device combination may be lower still.

7.3.1 Custom Interface and Wii Controller

All users completed the tasks successfully using this interface/device combination. All users commented that the Wii Controller and custom interface was responsive and accurate, while one user also commented that it was not as accurate as a mouse. No users complained of any jitter effects. The Wii controller software library succeeded in masking the jitter of the gyroscopes without affecting the responsiveness of the controller. No users experienced any noticeable latency between their physical actions and the corresponding movement onscreen. Although the evaluation only took a short amount of time, no users experienced any serious physical strain. Two users initially found manipulating the camera was forcing them to move the controller into uncomfortable positions. However, after some practise using the interface they began to take advantage of the relative control system, i.e. releasing the camera control once an uncomfortable position had been reached, returning to a comfortable position, and continuing from there.

The camera controls were well received by those people with 3D interface experience. Two users with no experience of working in a 3D environment really struggled to position the camera, generally reverting to using a random movement trial-and-error approach.

Those users who had previous experience of the Nintendo Wii system had trouble adjusting to the movements required. The Nintendo Wii typically requires large fast movements in games (measuring responses with the accelerometer) and uses the IR pointing mechanism for menus. This led to these users using a mixed of positional movements and pointing techniques whilst carrying out the tasks. Interestingly, although these users were operating this interface as if the IR sensor and accelerometers were in use, they managed to complete the tasks and had no complaints regarding the interface. Evidently the design of the menus emulates a pointing device to such an extent that the users did not perceive a difference. While this was not the intention of the interface it is a helpful side effect if users are able to familiarise themselves with the interface more quickly because it emulates an interface style they already know. Regarding positional movement, it was noted that when users were moving the controller rather than rotating it, they did actually unknowingly rotate the controller naturally. This caused a response from the interface, reinforcing the user's impression that positional movement was required. The unintended rotations actually matched what the user was attempting to achieve with positional movement. This was an unintended but welcome result. For example, when trying to zoom in and out, the users were moving the controller forwards and backwards. However, when moving the controller forwards they also naturally tilted the device down and tilted it up when moving it toward themselves. This causes the correct response from the interface. While this is not the most efficient way to use the interface and will cause unnecessary strain in the long term, it does indicate that the movements required are intuitive to the user.

7.3.2 VRMenu and Mouse/Keyboard

All users completed the tasks successfully using this interface/device combination. The mouse was perceived to be the most accurate device of the three as expected. There were also no reports of any physical discomfort. The camera controls received a lot of criticism from first time users. Experienced users were able to use keyboard shortcuts to change between camera modes. The alternative was to select the modes from a toolbar. In line with expectations, rotations proved difficult using the mouse. This was compounded by the fact that the interface translates the velocity of the mouse when the mouse button is released into the velocity of the object. Therefore, if the mouse is not stationary when the button is released the object continues to move. Several users managed to lose the object from view in this way. Users highlighted negatively the large number of small movements required with the mouse to achieve the required camera positions. This was caused both by the need to translate 2D movement into 3D and because of the limited space afforded by the mouse pad.

7.3.3 VRMenu and Polhemus Fastrak

All users completed the tasks successfully using this interface/device combination. During this evaluation, two problems with this interface were identified which could be solved without totally redesigning the interface:

1. Destructible menus; because the menus are objects within the scene, they can be selected and manipulated as any other object can. Thus they can be rotated and moved. Some users managed to accidentally select parts of the interface and alter their positions/orientations, effectively destroying the menu. This could be solved by restricting general interaction to scene objects.
2. Intermediate value selection; the value selection slider bars can be set in two modes. One where each movement triggers an update of the network and the current value display. This can causes long load times for every slight movement. The other mode updates the network only when the user releases control. Unfortunately the current value is also not updated until this time so the user cannot see what value they are selecting until after they select it. This leads to multiple selections as the user tries to find the

correct value. A mode is needed where the network is not updated until the user releases control but the current value indicator is updated for every movement.

Users found camera manipulation with this device to be difficult. As shown in Figure 15, users were actually slower using this device for camera manipulation than when using the mouse. This is surprising given the 3D nature of the Polhemus Fastrak device and the fact that the camera model was almost identical to the one used in the custom interface. Some users commented that the ergonomics of the device and the wire interface was causing problems for them both when manipulating the camera and when pointing. The grip used on the controller does not match the grip one would use when pointing a wand. So the physical grip did not match the virtual representation.

7.4 Summary

This user evaluation serves to prove the concept that inertial sensors can be used for visualization control. The results above show that the custom interface coupled with the Wii controller is a match for two existing interface/device combinations when measuring time to complete tasks. It is hoped that the efficiency of the custom interface is an improvement on existing designs. Unfortunately, this claim remains untested. Subjectively however, users did find the custom interface easier to use. A more in depth evaluation with a larger group of users is needed to make any further comment about the performance of the interface.

One interesting point is that of user experience. All users in the evaluation have had years of training while using mouse devices and interfaces deigned for mouse input. Some users involved have had experience with the Wii controller (although never used in this way) and/or the Polhemus Fastrak device but not to the extent of the mouse experience.

More evaluation is needed to assess whether with further use and training, the performance of users with the 3D input devices would improve. The initial tests above suggest that even with a small amount of experience (compared to experience gained with a mouse) user performance was noticeably better. However, a very small sample was used in the evaluation and a larger group would need to be evaluated over a longer period to certify this claim.

8 Conclusions and Future Work

This project has shown that inertial sensors can be successfully used to control visualization applications. Software has been developed that allows the Wii controller to be used within the trackd framework and a custom interface was developed to allow visualization control using solely orientation data.

The developed interface outperformed (in terms of time) the existing VRMenu interface when used with a more expensive 3D tracking device for all types of task analysed (if the adjusted metrics are used) in the small final evaluation performed. The custom interface was also very near to the performance of the mouse controlled interface for application control tasks and outperformed it for camera manipulation tasks. However, further evaluation is needed to cement these claims. Qualitative feedback suggests that users preferred the custom interface overall. Metrics such as efficiency (number of button presses and amount of user movement), memory load, and physical strain were only improved upon in the custom interface theoretically. Further evaluation is needed to prove these improvements, although testing for these metrics is typically difficult.

To take the custom interface forward and improve upon the design a deeper evaluation must take place. A longer user trial of the interface within a real visualization environment will allow better understanding of the deficiencies of the interface. Various features such and audio and haptic (vibration) feedback should be investigated to determine their effect on user performance.

Further work is needed to make use of the accelerometer data. Either more sophisticated techniques, such as a Kalman filter, need to be employed to generate accurate positional data, or the acceleration data should be used in another way. One possible use is in the generation of gesture data, this is the technique used within the Nintendo Wii itself to make use of the accelerometer data.

The ability for the custom interface developed to be used in other 3D applications beyond visualization should be investigated. Many applications do not require more interaction methods than the interface currently provides. Lack of full symbolic input is the interfaces largest current deficiency in the author’s opinion if it is to be used in other applications. The dial-pad interface should be extended as suggest to bring full symbolic input to the interface. Also, the custom interface itself needs to be integrated into an existing visualization application such as AVS/Express.

9 Bibliography

1. *The Application Visualization System: A Computational Environment for Scientific Visualization.* **Upson, Craig, et al.** July/August 1989, IEEE Computer Graphics and Applications, pp. 30-42.

2. *Visualization Idioms: A Conceptual Model for Scientific Visualization Systems.* **Haber, H. B. and McNabb.** s.l. : IEEE Computer Science Press, 1990, Visualization In Scientific Computing, pp. 74-93.

3. **Cruz-Neira, C., Sandin, D. J., DeFanti, T. A., Kenyon, R. V., and Hart, J. C.** The CAVE: audio visual experience automatic virtual environment. *Communications of the ACM.* June 1992, Vol. 35, 6, pp. 64-72.

4. VRMenu User Guide. *AVS/Express.* [Online] [Cited: 07 09 2010.] http://www.avs.com/mpe/mpe62/mpe_vrmgTOC.html.

5. **Analog Devices.** ADXL335. *Analog Devices.* [Online] [Cited: 15 05 2010.] http://www.analog.com/en/sensors/inertial-sensors/adxl335/products/product.html.

6. *Motion Tracking: No Silver Bullet, but a Respectable Aresnal.* **Greg Welch, Eric Foxlin.** s.l. : IEEE, November/December 2002, Computer Graphics and Applications.

7. **Bowman, Doug A., et al.** *3D User Interfaces Theory and Practice.* s.l. : Addison Wesley, 2004.

8. **Foxlin, Eric.** *Handbook of Virtual Environment Technology.* s.l. : Lawrence Erlbaum Associates, 2002.

9. Motion Capture/Tracking from Inition. *Inition.* [Online] [Cited: 18 04 2010.] http://www.inition.co.uk/inition/products.php?CatID_=11.

10. *WiiBrew.* [Online] [Cited: 18 04 2010.] http://wiibrew.org/wiki/Main_Page.

11. **Lee, Johnny Chung.** Hacking the Nintendo Wii Remote. *IEEE Pervasive Computing.* July-September 2008, pp. 39-45.

12. *Measuring orientation of human body.* **Luinge, H. J. and Veltink, P.H.** 2, s.l. : Springer Berlin / Heidelberg, 2005, Medical and Biological Engineering and Computing, Vol. 43, pp. 273-282.

13. **Xsens.** Miniture Attitude and Heading Sensor MTi. *Xsens.* [Online] [Cited: 2010 05 03.] http://www.xsens.com/en/general/mti.

14. *A Survey of 3D Interaction Techniques.* **Hand, Chris.** 5, 1997, Computer Graphics Forum, Vol. 16, pp. 269-281.

15. *A Survey and Taxonomy of 3D Menu Techniques.* **Hübner, R. Dachselt and A.** 2006. Eurographics Symposium on Virtual Environments.

16. **Belleman, Robert.** XiVE: X in Virtual Environments. [Online] [Cited: 12 05 2010.] http://www.science.uva.nl/~robbel/XiVE/.

17. **D. Gerber, D. Bechmann.** Design and evaluation of the ring menu in virtual environments. [Online] 2004. [Cited: 07 05 2010.] http://www710.univ-lyon1.fr/~sbrandel/en/research/VR/ipt_gb04_web.pdf.

18. *The Spin Menu: A Menu System for Virtual Environments.* **D. Gerber, D. Bechmann.** 2005. IEEE Virtual Reality Conference. pp. 271-272.

19. *Collapsible Cylindrical Trees: A Fast Hierarchical Navigation Technique.* **Raimund Dachselt, Jürgen Ebert.** s.l. : IEEE Computer Society , 2001. Proceedings of the IEEE Symposium on Information Visualization .

20. *HoloSketch: a virtual reality sketching/animation tool.* **Deering, Michael F.** s.l. : ACM, 1995. ACM Transactions on Computer-Human Interaction. pp. 220-238.

21. *3D widgets for exploratory scientific visualization.* **Herndon, Kenneth P. and Meyer, Tom.** UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology : ACM, 1994. pp. 69-70.

22. *Three-dimensional widgets.* **Conner, Brookshire D. and Snibbe, Scott S. and Herndon, Kenneth P. and Robbins, Daniel C. and Zeleznik, Robert C. and van Dam, Andries.** Cambridge, Massachusetts, United States : ACM, 1992. Proceedings of the 1992 symposium on Interactive 3D graphics .

23. **Dr Helen Sharp, Professor Yvonne Rogers, Dr Jenny Preece.** *Interaction Design: Beyond Human-computer Interaction* . s.l. : John Wiley & Sons, 2007.

24. *Exploring bimanual camera control and object manipulation in 3D graphics interfaces.* **Ravin Balakrishnan, Gordon Kurtenbach.** s.l. : ACM, 1999. pp. 56-62.

25. **Analog Devices.** ADXL330. *Analog Devices.* [Online] [Cited: 12 05 2010.]

26. WiiYourself! *Glitter.org.* [Online] [Cited: 03 05 2010.] http://wiiyourself.gl.tter.org/.

27. **Mechdyne.** trackd. *Mechdyne Corperation.* [Online] [Cited: 2010 5 2010.] http://www.mechdyne.com/integratedSolutions/software/products/trackd/trackd.htm.

28. **Joseph L. Gabbard, Deborah Hix, J. Edward Swan.** User-Centered Design and Evaluation of Virtual Environments. *IEEE Computer Graphics and Applications.* November/December 1999, pp. 51-59.

29. Viz For HPC. *Research Computing Community Wiki.* [Online] [Cited: 29 08 2010.] http://wiki.rcs.manchester.ac.uk/community/Viz_for_HPC.