

WIIISUALISATION



A THESIS SUBMITTED TO THE NATIONAL UNIVERSITY OF IRELAND, CORK  
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF  
MSc IN APPLIED SCIENCE (INTERACTIVE MEDIA)  
IN THE FACULTY OF SCIENCE

October 2008

Conor Crowley  
Department of Computer Science

# Contents

<b>Abstract</b>	<b>6</b>
<b>Declaration</b>	<b>7</b>
<b>Acknowledgements</b>	<b>8</b>
<b>1 Introduction</b>	<b>9</b>
1.1 Motivation . . . . .	9
1.2 Objectives . . . . .	10
1.3 Similar Work . . . . .	10
1.3.1 Wii projects . . . . .	10
1.3.2 Stereoscopic Views . . . . .	12
1.3.3 Medical Imaging . . . . .	13
1.3.4 Collaborative Learning and New Technologies . . . . .	14
<b>2 Analysis</b>	<b>15</b>
2.1 The Challenge . . . . .	15
2.2 Project Overview . . . . .	15
2.2.1 The Visualisation . . . . .	16
2.2.2 Control . . . . .	16
2.2.3 Display . . . . .	16
<b>3 Design</b>	<b>17</b>
3.1 Options . . . . .	17
3.1.1 Independent Application . . . . .	17
3.1.2 WiiMote Aware Osirix . . . . .	18
3.1.3 WiiMote Aware Osirix Plug in . . . . .	18
3.2 Platform . . . . .	18
3.3 Technologies . . . . .	18
3.3.1 Objective C and Cocoa . . . . .	18
3.3.2 Visualisation Toolkit . . . . .	19
3.3.3 Bluetooth . . . . .	20
3.4 Hardware . . . . .	20
3.4.1 WiiMote . . . . .	20

<b>4</b>	<b>Implementation</b>	<b>22</b>
4.1	WiiMote Framework . . . . .	22
4.1.1	Connection . . . . .	22
4.1.2	Connection Stream and Events . . . . .	23
4.1.3	Fundamental Controls . . . . .	24
4.2	Osirix Views . . . . .	24
4.2.1	Building VTK . . . . .	25
4.2.2	The Different Views . . . . .	25
4.2.3	User Interface . . . . .	26
4.2.4	Creating the View . . . . .	26
4.3	Stereoscopic Modes . . . . .	27
4.3.1	Coloured Anaglyph . . . . .	29
4.3.2	Active Glasses . . . . .	29
4.3.3	Polarised Displays . . . . .	30
4.3.4	Testing and Outcome . . . . .	31
4.4	Integration . . . . .	32
4.4.1	Control . . . . .	33
4.4.2	Test and Debug Functionality . . . . .	36
4.5	Issues . . . . .	37
<b>5</b>	<b>Evaluation</b>	<b>39</b>
5.1	System Tests . . . . .	40
5.2	User Tests . . . . .	41
5.2.1	The Users and their Scores . . . . .	42
5.2.2	Results of Analysis . . . . .	43
5.3	Overall . . . . .	44
<b>6</b>	<b>Conclusions</b>	<b>45</b>
6.1	Success of Work . . . . .	45
6.1.1	3D Medical Visualisations . . . . .	45
6.1.2	WiiMote Control . . . . .	45
6.1.3	Stereoscopic Display . . . . .	46
6.1.4	Overall . . . . .	46
6.2	Planned Future Work . . . . .	46
	<b>Bibliography</b>	<b>47</b>
<b>7</b>	<b>Appendix</b>	<b>49</b>
7.1	Appendix A: Wiisualisation Controls . . . . .	49
7.2	Appendix B: System and User Test Results . . . . .	50
7.3	Appendix C: On The DVD . . . . .	52
7.4	Appendix D: Websites . . . . .	53
7.5	Appendix E: Source Code . . . . .	54

# List of Tables

3.1	System Specifications . . . . .	20
4.1	Example of a Colour Lookup Table with Opacity . . . . .	27
5.1	Results Of Systems Tests for GameDev . . . . .	40
5.2	Results Of Systems Tests for SciViz . . . . .	40
5.3	Results of OpenMark Benchmarking . . . . .	40
5.4	User 1's Scores . . . . .	42
5.5	User 2's Scores . . . . .	43
5.6	User 3's Scores . . . . .	43
5.7	User 4's Scores . . . . .	43
5.8	Average User Scores . . . . .	44
7.1	Results Of Systems Tests . . . . .	50
7.2	Results of User Tests . . . . .	51

# List of Figures

1.1	An Example of a Stereoscopic Image . . . . .	12
1.2	Demonstration of how eyes converge and diverge to focus on objects near or far away . . . . .	12
1.3	Structure of Osirix' components . . . . .	13
3.1	WiiMote control Layout . . . . .	21
3.2	WiiMote Peripherals. Classic controller on the left, Nunchuck on the right . . . .	21
4.1	Example of Upper Torso and Head Model Rendered in Surface Rendered View .	24
4.2	Example of Upper Torso and Head Model Rendered in Volume Rendered View .	25
4.3	A Pair of Colour Anaglyph Glasses . . . . .	29
4.4	A Pair of Active Glasses . . . . .	30
4.5	Lineaer Polarised Light . . . . .	30
4.6	A pair of Polarised Anaglyph Glasses . . . . .	30
4.7	Circular Polarised Light . . . . .	31
4.8	The 3 different Axis . . . . .	34
5.1	Example of Skull Model Rendered in Wiisualisation . . . . .	41
5.2	Example of Upper Torso and Head Model Rendered in Wiisualisation . . . . .	42

# Abstract

Advances in stereoscopic displays and methods of interaction have opened up several areas of possible research. This coupled with advances in graphical computing power led to the development of this project.

Wiisualisation is a plugin designed for the open source DICOM viewer Osirix. It was designed to generate 3D medical models from various existing medical scans such as MRI or CT scans. When the model is generated the plugin connects to and accepts input from a wiiMote. The wiiMote is the controller for Nintendo's recent console release, the Wii. It is aware of both position and motion and so has provided a new novel form of control. The project concentrates on the development of a large stereoscopic display which can be used for group viewings. This would allow for the demonstration of highly detailed medical models in a group learning environment.

It is hoped that well a designed and implemented stereoscopic display coupled with the novel method of control in the form of the wiiMote will result in a more effective means of teaching. To aid in this outcome research was conducted into how groups learn. It is believed that when groups are presented with new technologies then they become more motivated to learn. It is this particular aspect of groups behaviours when learning which should cause students to be more engaged using Wiisualisation.

# Declaration

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

Signed:  
Conor Crowley

# Acknowledgements

There are several people who must be acknowledged for their part in the creation of this thesis.

Firstly is Orla Forde whos input, proof readings and support throughout the project were key to the result.

Another key aid throughout the project was Michael McAuliffe. His aid in the setup of the projectors as well as many proof readings and help in working out the awkward relationships in the code was invaluable.

My supervisor, Dave Murphy, helped greatly with his key insights and experience with Osirix development helped whenever I got stuck.

I also need to acknowledge my family's support and patience over a busy summer.

# Chapter 1

## Introduction

The aim of this project is to develop a stereoscopic 3D medical visualisation system which can accept control input from the wiiMote for use in group learning environments. The project will be developed and deployed on an Apple Macintosh. The goal is to have a functioning system which will allow a demonstrator or lecturer to present highly detailed and accurate medical case studies to groups of students in full 3D. It will allow the demonstrator or lecturer to interact with the model wirelessly from a distance and, where appropriate from among the group, show key areas of interest on or in the model.

This will allow students to become more familiar with the body's internals as well as allowing them to recognise and understand medical conditions by displaying them in an easily recognisable format.

### 1.1 Motivation

The project was undertaken due to a keen interest in computer interaction and novel control methods. The plan was to develop a system which would allow intuitive user control of an unfamiliar interface. The release of the Wii and with it the wiiMote has prompted a surge in independent developments in the area of interaction and visualisations. Some of the most notable developments in this field have been made by Johnny Chung Lee who has gained worldwide recognition for his development of multipoint tracking and a head tracking system to suggest a 3D environment. It was projects like these which prompted this project's development.

Medical visualisations were chosen because of Osirix's high quality models. The quality and detail of these models are very impressive and gives the user unprecedented insight into the biological makeup of the human body. It is hoped that allowing an intuitive interaction with these models in 3D space will give the user a very immersive experience and promote a deeper understanding of the subject being displayed. It is hoped that this project can also lead to an improved level of surgery preparation through showing a surgeon exactly what is to be expected in an operation. The wiiMote would offer a more comfortable method for interacting with and exploring these models.

This project could also have implications for teaching. Allowing lecturers and demonstrators

to show a class highly detailed medical models in stereoscopic 3D. This should allow a class to familiarise themselves with the intricate details of biology.

## 1.2 Objectives

The objectives of this Masters thesis are as follows:

- Create a highly detailed 3D Medical Model

This can be a generic Model or if possible built from existing records as in Osirix. Allowing exported models from Osirix to be read is a possibility.

- Allow interaction with the model using the wiiMote

The control should be intuitive and functional. Extensive testing should be employed to ensure that the control is accessible to a wide variety of users with very little alteration.

- Create a Stereoscopic 3D display

The display should be stereoscopic where available. However this should be an option to allow those who do not have the facility to display stereoscopic content use the project anyway.

## 1.3 Similar Work

The research for this project was divided into four key areas. It was decided to look at other projects which use the wiiMote first. This would give a good overview of what the wiiMote is really suited for and how the control has been implemented for the best results. The next area which will be covered are stereoscopic views and how they are generated. Current advancements in the area will also be covered. Careful attention will be paid to this area through the project's life cycle as new developments are being released on almost a weekly basis. After this Medical imaging will be addressed. A high level of familiarity with this area will be required to discover exactly what will be required for this project. Osirix's handling of image sets and generations of images will be closely analysed. Alternative methods will also be covered. Finally the area of collaborative learning will be covered. This will be key in understanding what is required when demonstrating a concept to a group. It will also be useful to see how people learn in a group environment.

### 1.3.1 Wii projects

When Nintendo released the Wii on November 19th 2006 it introduced a new method of control to the video game industry. This novel form of control came in the form of the Wii Remote or wiiMote. The wiiMote is a location aware controller which introduced motion as a form of control. Its use of Bluetooth as a means of connection meant that it could be easily connected to other systems. Use of the Service Discovery Protocol reveals that the wiiMote works in a

similar way to the standard Universal Serial Bus Human Interface Device (USB HID). Because of this it appears as a standard input device to any Bluetooth Host.

Over time the traffic on the connection was analysed and is now almost entirely identified. Libraries for the connection and control of the wiiMote were developed on all operating systems. For windows the main method of connection is an application called GlovePie. It was originally developed to emulate keyboard and joystick controls using the essential Reality P5 Glove [Ken08]. On Apple's OS X platform the base functionality was developed by the Darwiin Project [Whe08]. The Darwiin Project has been in active development since December 6th 2006. At the time of printing, version three of the project was in the alpha stages of development. The project concerns itself with the the handling of traffic between the wiiMote and the host system. In this capacity it mainly facilitates other developers to create new projects. The sourceforge site for the Darwiin Remote project encourages developers to share details of projects they are creating using the Wii Remote Framework developed for the Darwiin Remote.

Projects which now incorporate control from the wiiMote are widely varied. One of the most influential personalities in the area is Johnny Chung Lee who gained world wide recognition through demonstrating his projects on YouTube, a social video sharing site. With the success of his interactive white board created for less than US\$50, he Was invited to give a talk for TED [Lee08a]. TED stands for Technology, Entertainment, Design. It is an annual conference which invites leaders of these areas to give eighteen minute talks. It has been running since 1984 [TED08]. Johnny Chung Lee has three main projects which are the general types of projects being developed by the community.

#### **Multi-Point tracking**

The wiiMote is capable of tracking multiple points using the in-built infra-red camera. For this project reflective tabs were attached to the tips of the users wingers and an array of infra-red LEDs were shone out from the wiiMote. For this and indeed all of these projects the wiiMote is not interacted with and the multi point tracking is exploited. With the lighting array turned on the reflective points are easily identified by the wiiMote and this information is delivered to the application which facilitates the interaction [Lee08b].

#### **Interactive Whiteboard**

This project puts the concepts from the original project to use. However, for tracking this time a simple IR LED pen is developed. This is comprised of an LED and a power source. The position tracking was routed to control the mouse and a flicker of the light was identified as a mouse click. This was the project discussed in the TED talk. It would provide functionality of a far more expensive system for a fraction of the cost [Lee08b].

#### **Head Tracking for Desktop VR Display**

This project is also based on a similar principal. With the head's position known it is possible to alter the display to suggest depth. The head is referenced using two IR LEDs to give position, distance and orientation. With this perspective, it can be derived and used to alter the display on screen [Lee08b].

A project similar to this was developed with the wii remote framework from the Darwiin Remote Project. There have also been attempts to create games which take advantage of the wiiMote as a form of control. The wii remote framework was integrated into one of these in the

Torque Game engine. These projects are all listed on the Darwin Remote Sourceforge website [Whe08].

### 1.3.2 Stereoscopic Views



Figure 1.1: An Example of a Stereoscopic Image

Stereoscopic views are views which are comprised of separate channels for both the left and the right eyes [Sex99]. They are made up of a display for the channels and some manner of separating them for each eye. The three regularly used methods for this are colour anaglyph, polarised anaglyph and active glasses. The displayed image usually looks like the image in figure 1.1. The figure shows how the two images are overlaid. The splitting is performed by glasses which the user wears. This particular image is based on the colour anaglyph method of recombining the images.

The creation of stereoscopic displays is performed by analysing how far into the foreground or background the object protrudes. As demonstrated in figure 1.2, the position of the object will alter the angle of convergence between the eyes. It is these angles which cause the user to infer distance. The different perspectives on the view being sent to each eye cause these alterations.

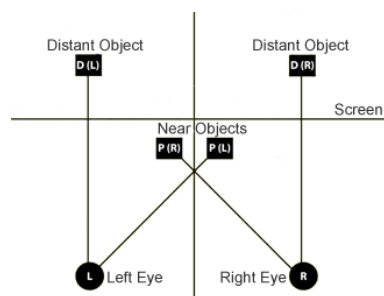


Figure 1.2: Demonstration of how eyes converge and diverge to focus on objects near or far away

Stereoscopic views have been explored for use in medical situations before. The potential of using an auto stereoscope in micro-surgery was examined and it was found that it improved the users accuracy slightly [CTL<sup>+</sup>99]. There were some issues with discrepancies in user results but with modern advancements in displays this issue would be reduced.

### 1.3.3 Medical Imaging

The area of medical imaging and visualisation has had many advancements recently. The cost of high performance graphical hardware is dropping and now even desktop machines are powerful enough to display complex medical models [WG07]. What once would have required an expensive and dedicated imaging machine can now be performed by regular desktop machine.

For creating medical visualisations one method is used above all others to gather the data. A dataset is comprised of several 2D slices which are collected using an instrument like a Computed Tomography (CT) Scanner or Magnetic Resonance Imaging (MRI) scanner [LAGS05]. To compile a 3D rendering of the subjects, each of these slices is read and analysed. The 2D slice is represented as a greyscale image by using different levels of white to represent the different intensities at which the scanner encountered a structure. Bone has a different intensity to skin so the two are easily distinguishable [LAGS05].

The datasets themselves are stored under the DICOM image format. DICOM stands for Digital Imaging and Communications in Medicine. The standard was created due to the move towards having digital records in medicine. DICOM was designed around a concept called the Service Object Pair. Services are the details or meta data that accompanied the objects which are the the images [MD04]. Over time it has been incorporated into a larger range of devices.

Due to this standardisation and the increased graphical performance of desktop computers, DICOM image viewers are developed without needing a dedicated setup. One such viewer which was developed to run on Mac OS X is Osirix. It is the follower to a previous viewer called Osiris which ran on all platforms [MMP04]. It was developed partially to allow access to these dataset to a much larger range of users. It was developed with open source methodologies and is freely available, this has made it a popular choice among active users and developers alike [MMP04]. Osirix itself is composed of several different components which are used in the areas they are most suited. These components are layered on top of each other so the system is designed to take advantage of all available performance from the hardware up. The structure is outlined in figure 1.3. The components this project will be concentrating most on are VTK and the DICOM reader. These will invoke the lower level components as they are necessary.

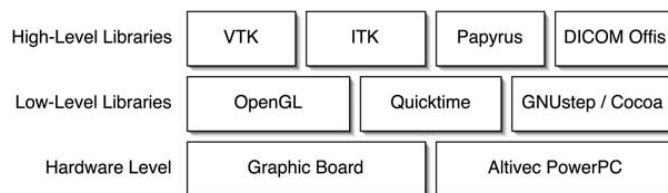


Figure 1.3: Structure of Osirix' components

2D image slices have been used for several years to help in computer aided diagnosis (CAD). This has been especially effective with mammograms and breast cancer detection [Doi07]. Any CAD system is not intended to replace a physicians diagnosis but instead aims to supplement and assist them [Doi07]. Recently there has been research into how a rendered volumetric representation of the data can help in a diagnosis [HHR<sup>+</sup>08]. Forced-choice reader study was designed using MR brain images from 36 subjects. They found that the rendering was an aid

in the diagnosis [HHR<sup>+</sup>08].

### 1.3.4 Collaborative Learning and New Technologies

In learning there are times when we develop skills through trial and error. In these cases we discover the best methods to accomplish a task through experimentation. This can be very rewarding and effective with tasks which are relatively simple. This method becomes awkward and inefficient when dealing with larger more complex tasks. To cope with these we learn through collaborative means. In general the person learning takes on the role of the novice and the more experienced person becomes the expert [Cro94]. In these situations the expert does more than just show the novice how to perform the task. Through the interaction the of solving the act together both learn and the novice will become more adept at the particular task faster than if it was merely demonstrated to them. When computers are introduced into this situation it is found that lively discussions are engaged in. This stimulates thought and improves learning [Cro94]. In teaching there are many ways to introduce computers as a form of learning. These range from typing using a word processor to creating simulations to allow students to interact and thus encourage learning. The creation of simulations are considered to be one of the most effective forms of encouraging class interest in a subject [Mai96].

The introduction of a new form of interaction can generate enthusiasm within a class. In 1997 the introduction of basic computing into the classroom environment was enough to instantly bolster interest and enthusiasm for both students and teachers [San97]. Today it requires more than basic computer usage to encourage interest. New technologies are needed as it is the introduction to the new and unfamiliar which grabs our attention. This is why it is believed that the used of stereoscopic models will keep the interest of the students.

New technologies can also encourage greater initiative in students. With the introduction of the computers it was found that students would go beyond the requirements for assignments, they would even volunteer to extra work which involved the new technologies [San97]. If this can be carried over and applied to this project it should be a welcome addition for lecturers.

## Chapter 2

# Analysis

### 2.1 The Challenge

The challenge which presents itself here is how to demonstrate a highly detailed medical model to a group of users in order for those users to be able to identify key regions of interest. Current technology allows for the creation of these models as well as limited interaction. The issue which this project is concerned with arises when the model is being demonstrated to a large group of people. The main issue here is the control of the model. To allow a group freedom of movement to more closely inspect the model or gain a different vantage point it is no longer feasible to use a keyboard and mouse for interaction. This method of control requires a podium or some other hard surface from which to interact and at that point the user is fixed to that point.

In modern lecture theaters it is freedom of movement which is key. The main user or lecturer must move and interact with their audience. Even with current wireless technology they are tethered to the podium or table, or if they move interaction on counters not designed specifically for the task can make these interaction awkward. To allow the user or lecturer to concentrate on their main task the control of the model should be intuitive and unobstructed. They need to be able to interact with the model from within the crowd to ensure their audience is viewing the model correctly.

The other challenge is the improvement of the immersion of viewing these models. Advances in stereoscopic displays have made high quality 3D displays more widespread. 3D cinema is growing and these technologies can be applied here to improve the user experience. A model display in full 3D will allow the user to gain a better insight into any conditions being shown to them or even diagnose diseases. This summer a development in the identification of colon cancer using a virtual colonography was published by the American College of Radiology Imaging Network (ACRIN) [Joh08]. They found that use of a highly detailed scan could be used to assist doctors in identifying certain types of colon cancers.

### 2.2 Project Overview

To address all of these challenges the project will need to be comprised of several parts.

### **2.2.1 The Visualisation**

To be of any use to a group of users the model will have to be very high quality. If it all possible it will be rendered from an existing DICOM dataset in the standard manner, by reading through each 2D slice to build the model [LAGS05]. If this is not possible then the next option is to use a pre-rendered model. This would limit how the plug in is used however, as a new model would have to be generated for each dataset a user wants to view. Even the facility to accept new models would be a difficult and awkward process. The user would need to have experience in a separate application. For these reasons it will be necessary to either set a specific model and accept the limitations or develop the ability to read DICOM datasets to render the model.

### **2.2.2 Control**

The control as outlined above has very specific requirements. The recent release of the wiiMote caters to all of these. The Bluetooth connectivity will allow the demonstrator to walk among their audience while still being connected to the host machine. The wiiMote does not require a surface to lean on, so interactions can still be performed with the model. The main limitation is the line of sight requirement for the infra-red pointer functionality. This may also be an issue if there is sunlight shining directly on the IR LEDs which are being used as reference points for the wiiMote's IR Camera. It is the dependency on these IR points which will probably be more of a limiting factor on range than the Bluetooth radio.

### **2.2.3 Display**

The hardware required for the stereoscopic display is already in place. Its details will be covered in the Design chapter. The challenge here is to create an application which fully takes advantage of how a stereoscopic image is generated. The application will be developed with this in mind at all times. Stereoscopic displays work best when the subject is brightly lit against a dark background. This is what will dictate most of colours chosen for the display. The subject will need to be easily identifiable so brighter colours will be required.

Even with these factors catered to, the hardware setup can cater for multiple types of stereoscopic display. The three best suited to this area would be colour anaglyph, active glasses or polarised anaglyph. Each of these must be assessed to determine which gives the best result.

# Chapter 3

## Design

This chapter outlines how the overall decision for the project's design were reached. It details the various options available and give each option's characteristics. Details what will be involved in developing the selected option and detailing the hardware and software restriction which must be adhered to.

### 3.1 Options

Several options were considered for this project. They are:

1. Development of an independent application which can use input from the wiiMote to interact with a pre-rendered medical model.
2. Alter the existing Osirix build to allow for wiiMote connection and interaction throughout the program's functionality.
3. Create a plug in for Osirix to allow the above interaction on a much more limited subsection of the functionality.

The benefits and drawbacks of each of options are outlined in the following section.

#### 3.1.1 Independent Application

The stand alone application would allow for the greatest amount of customisation. Being a whole new application, features to be implemented could be carefully selected and designed to meet the requirements of the users. The overall look and feel could be built to specifically cater to the applications purpose.

However, there would also be issues with the length of time required to develop such an application. There are also technical issues with the implementation of various versions. The goal of the project is to develop a system which can be widely deployed across different systems. This option would require the development of both stereoscopic and non-stereoscopic versions. The development of the medical model would also be an issue. While researching how it would be possible to create a model with sufficient accuracy for use in an educational situation, it was

found that the models generated in the DICOM viewer Osirix could be exported. This would result in highly accurate and detailed models. This raised another important issue, the option to develop a custom wiiMote aware version of Osirix.

### **3.1.2 WiiMote Aware Osirix**

Osirix is an open source DICOM viewer. It was already planned to use the model created by this viewer so it was decided to analyse the feasibility of building a version of Osirix which could be controlled by the wiiMote. Doing this can remove a lot of the opportunities for customisation which were available in developing a new application. However it would allow the usage of the established features of Osirix and give the project an established point to build from. Osirix is a widely used DICOM viewer so many professionals would be familiar with its operation. It would also be beneficial to any student to become familiar with it.

The main issue with a custom Osirix build was the sheer scale of the project. Writing wiiMote controls into each aspect of the program or even gaining an understanding of the program would take several months to accomplish. In examining the program it was found that plug ins could be developed for the application, and so offering a third option for the project.

### **3.1.3 WiiMote Aware Osirix Plug in**

Osirix plug ins have access to all of the functionality of Osirix. The plug in allows the focusing of efforts into the creation and control of a medical model. The plug in can use the set of classes Osirix has dedicated to the generation of 3D models from DICOM image sets. The key for this option is allowing a plug in have access to an alternate control scheme. The inputs or signals need to be routed to the commands in Osirix. Whether this is possible through a plug in is yet to be seen. The plug in option appears to be the most viable as there are only minor drawbacks.

## **3.2 Platform**

The wiiMote aware Osirix plug in was chosen for this project. This dictated several things. Firstly, the development will have to be carried out on an Apple Mac, because Osirix was developed exclusively for Macs. Secondly, the plug in will be developed in Objective-C mixed with various parts of C or C++ as required. This is the same language used for the development of Osirix itself.

## **3.3 Technologies**

The project is composed of a mix of several different technologies working together.

### **3.3.1 Objective C and Cocoa**

The development of this project will require the ability to program and debug using Objective C. This will have to be learned during the design and planning of the project as currently no

Objective C is known. Objective C is in essence an Object Orientated version of the C programming language. It incorporates many of the functions of another language called smalltalk to improve the accessibility of the C language to the developer. Some of the concepts which are added to C are as follows [inc08]:

- Messages

These are how methods are accessed and variables are passed. As C does not incorporate object oriented methodologies Objective C uses messages to control instances of classes.

- Defined Types

These are also required for object oriented design. It is important to identify an objects type to know who it should respond for different commands.

- Classes

Classes are used to define objects. Every object is a type of class. this once again allow the developer to alter the behavior of all object of a particular type of class simply by altering the class' definition.

- Categories

These allow the functionality of a class to be extended or altered in a specific way. These are especially useful if you have no access to the the original class.

They are some of the added functionality Objective C has over C. It also allows for the addition of C or C++ code as required. This becomes important in the building of the models as Osirix uses the Visualisation ToolKit (VTK) in their generation which requires C++ for its interactions in this development environment. Objective C 2.0 was developed with Cocoa for rapid application development on Mac OS X [inc08]. Cocoa is the Application Programming Interface (API) for developing these applications [ACM08]. In this environment, what are generally referred to as libraries are called frameworks. Cocoa allows you to use these frameworks to access any functionality you might need very quickly. It makes the creation of new applications a simple process. The final aspect for this development is the Interface Builder. This is a WYSIWYG (What You See Is What You Get) editor for creating Graphical User Interfaces (GUI). It uses a simple drag and drop interface to position interactive widgets and design layouts. It is then possible to connect these elements to the code through their associated properties [ACM08].

### 3.3.2 Visualisation Toolkit

The Visualisation ToolKit(VTK) is an open source, object-oriented system for computer graphics and visualisation. It is based on OpenGL and designed to be system agnostic. That means that it can be implemented on a wide variety of systems or with different level languages like C or Java [AsBB<sup>+</sup>04]. This project, like Osirix, will be implementing VTK using C. As with Objective C, VTK has to be learnt from basics for this project and while it is a large and highly complex system the manner in which it was developed allows for rapid learning once the basics have been covered in detail.

	<b>GameDev</b>	<b>SciViz</b>
Processor	Intel Core Duo 1.83GHz	2 x Intel Quad Core Xeon 3GHz
RAM	1GB	8GB
Graphics Card	128MB ATi Radeon X1600	1536MB NVIDIA Quadro FX 5600
Operating System	OS X 10.5.4	OS X 10.5.4

Table 3.1: System Specifications

### 3.3.3 Bluetooth

The wiiMote uses Bluetooth to connect to the system it controls. Bluetooth is a wireless communication method which allows for high speed and reliability over Relatively short distances, within around ten meters usually. The built in Bluetooth in most Macs and the drivers used make connecting the wiiMote to any Mac a simple task.

Bluetooth is a wireless technology which uses a frequency-hopping physical layer to allow portable devices to form short-range ad-hoc networks [Haa00]. An ad-hoc network is one created between devices without needing a dedicated infrastructure to support it. They can be created and ended as necessary. Bluetooth development was begun when in 1994 the Ericsson Mobile Communications AB in Lund, Sweden started to research a low powered wireless means of communication for mobile devices to remove the need for wires to connect mobile phones to their peripherals [Haa98]. Bluetooth operates on the Industrial-Scientific-Medical (ISM) radio band, 2.5GHz in Europe. This band is free to use and so Bluetooth must be able to deal with interference. This is the reason for its frequency hopping [Haa98]. Frequency hop splits the frequency band into several segments. Once interference is detected on a frequency the system hops. The result is a ‘*low-cost, narrow band transceivers with maximum immunity to interference*’ [Haa98]. Bluetooth would allow two separate systems to be operated in close proximity without interference. This would be useful if the system was implemented in multiple rooms in the same building.

## 3.4 Hardware

This project was developed on two main machines which would represent the high and low scale of systems running Osirix. Their specifications are in table 3.1.

GameDev is displaying at 1400 x 900 on an integrated LCD Screen. SciViz is outputting to two projectors in order to achieve the stereoscopic visuals. Through the rest of this document these two machines will be referred to by name.

### 3.4.1 WiiMote

The wiiMote was selected as the method of controlling this project. The layout of the controls can be seen in Figure 3.1. As mentioned above it is connected by Bluetooth and delivers a continuous stream of updates for each of its various sensors through this connection. The development community is very active in repurposing this controller to other means and has created a custom framework for its use with Objective C. This was a product of the Darwiin



Figure 3.1: WiiMote control Layout

Project. The wiiMote opens several possible options for control. The expansion port at the base of the controller can accept several different peripherals. Of these there are two which could serve to be truly useful for this project.

- **Classic Controller**

This is comprised of two analogue thumbsticks, a directional pad, four buttons on its face and four on its shoulders. The downside to this controller is the fact that it requires both hands to use effectively. However, through experience, it is believed that the use of two thumbsticks to control an object in 3D space is quite effective. This controller will not be used as the functions of the wiimote cannot be used comfortably at the same time. The classic Controller can be seen in figure 3.2.



Figure 3.2: WiiMote Peripherals. Classic controller on the left, Nunchuck on the right

- **Nunchuck**

The Nunchuck is an attachment which compliments the existing functions of the wiiMote. It consists of an analogue thumbstick and two buttons. It has a three axis accelerometer of its own as well. If there are issues with the control the nunchuck could be an excellent candidate for offering extra options to route extra controls. The nunchuck can also be seen in figure 3.2.

## Chapter 4

# Implementation

This Chapter outlines the work involved in the development of the project. It will address the various aspects and elements of the project and detail the issues encountered and their solutions. The project itself can be divided into three key areas. The final product of this project is the result of the integration of these three aspects into a functioning plug in. The three areas will be handled individually first with the integration addressed later.

### 4.1 WiiMote Framework

The first goal of the project was to connect the wiiMote to the machine. This is an area with a robust support community. The initial development is the Darwiin Remote. From this a Wii Remote Framework has been developed. This can be easily implemented into any project in theory. It handles all events from wii controllers allowing the developer to use them to drive their own applications. This framework has become the cornerstone of wiiMote development on the Mac platform.

Seeing how this framework was implemented it was decided to create something similar for Osirix. The plug in's main class handles the wiiMote events and routes them to a separate class which will build what is displayed on screen as well as any calculations or warnings. A class is a name for types of objects in programming. This structure allows for others to quickly develop other applications based on accepting control from the wiiMote.

This part of the project was given several stages to complete. The project concentrates on a lone wiiMote with no peripheral connected to the wiiMote's expansion port. The first stage planned is a successful connection.

#### 4.1.1 Connection

Both of the systems involved in the development of this project have Bluetooth modules. This allowed the initial connection to be completed quickly and easily.

Bluetooth connections are established by a receiver and a pager. In this instance the receiver is the wiiMote, the pager is the main machine. The pager sends several requests to connect to the receiver. These requests contain a code based on a timestamp. The timestamp is the

exact time an event or request is created. To allow for a difference between the two timestamps several versions of the code are sent. Once the receiver respond to one of these requests the pager sends the next packet. This packet contains an acknowledgment and its system clock. At this point the two are synchronised, the paging units system clock is used for the rest of the connection [Haa98].

To connect the wiiMote it must first be placed in ‘discovery’ mode which is accomplished by pressing the 1 and 2 buttons simultaneously.

This was implemented in a sample Osirix plug in. Osirix has a pre-defined waiting window which was used here to notify the user to attempt to connect the wiiMote. To identify when the wiiMote had been successfully connected the ability to customise certain functions on the wiiMote were used. It is possible to alter which of the numbered LEDs at the base of the wiiMote is lit, any combination from none to all four. The third LED was selected to identify that the wiiMote was successfully connected. To review the location of the LEDs on the wiiMote please refer to Figure 3.1.

A successful connection was established as well as a facility to catch unintended disconnects so that the plug in would close itself and not cause the system to lock. It was necessary as the connection tended to end with no apparent cause on GameDev. The second attempt to connect was more successful. Careful analysis of the data stream from both successful connections and connections which yielded immediate disconnects showed no apparent differences. These disconnects affected all wiiMote projects that were run on the GameDev machine. Connecting to the SciViz machine occurred without incident.

#### 4.1.2 Connection Stream and Events

A continuous stream of data is transmitted once a wiiMote is successfully connected. There are four key sensors which send updates continuously. This process is similar to the traffic from a mouse thus lending itself to an interaction similar to a mouse. There are then the twelve buttons which send either on or off signals. The framework allows for these events to be easily identified and addressed. A sample of the packet stream from both a successful and unsuccessful connection can be found on the accompanying disc. The packet logger used to collect this data gives the information both in its hex coded and a translated format. This helped in trying to identify how the wiiMote was communicating and also aided in understanding how to use this data for interactions.

Events are created with variables based on what is happening with the wiiMote. These variables are all floating point numbers allowing for very precise control. The three accelerometers can detect very minute alterations in the position and level of the wiiMote. This actually resulted in the plug in being far too responsive and unusable at one point.

To ensure the project was feasible it was necessary to have the wiiMote issue commands to an Osirix plug in. The concept was to create a duplicate view. This is outlined as the tutorial in learning how to build a new plug in. The plan consisted of establishing a connection to the wiiMote, have the press of the A button on the wiiMote cause the view to duplicate, close the connection on the press of the Home button.

### 4.1.3 Fundamental Controls

At this point it was decided to try and develop a simple interface for the wiiMote and an Osirix plug in. Initially this was to aid the development of the project and allow for rapid alterations to the code for testing and improvements. Once the majority of the events were being handled it was decided to implement controls for the aspects of a wiiMote which are not planned to be used in this plug in but which are catered for by the wii remote framework for future development. The facility to accept input from expansion peripherals was also included with the view to allowing further interaction using the wii Nunchuck should the plug in require more complex control.

The wii Nunchuck is a peripheral which connects through the expansion port. It has three accelerometers of its own along with a analogue controls in the form of two buttons which register various levels of resistance and a directional joystick to be controlled with the thumb.

At this point the the ability of the wiiMote to control the plug in was established. The command to duplicate the current window was tested with each button. The feed from the accelerometers were tested by writing a feed containing the various values to the console.

## 4.2 Osirix Views

More research was put into the creation of the plug in itself once the control was developed. Examples of plug ins dealing with the generation of a 3D display are rare. In Osirix all 3D displays are built using VTK. A segment of a plug in from Center for Medical Image Science and Visualization (CMIV) for viewing regions of interest has a custom designed Volume rendered View. This was used as a starting point for the development of the new plug in.

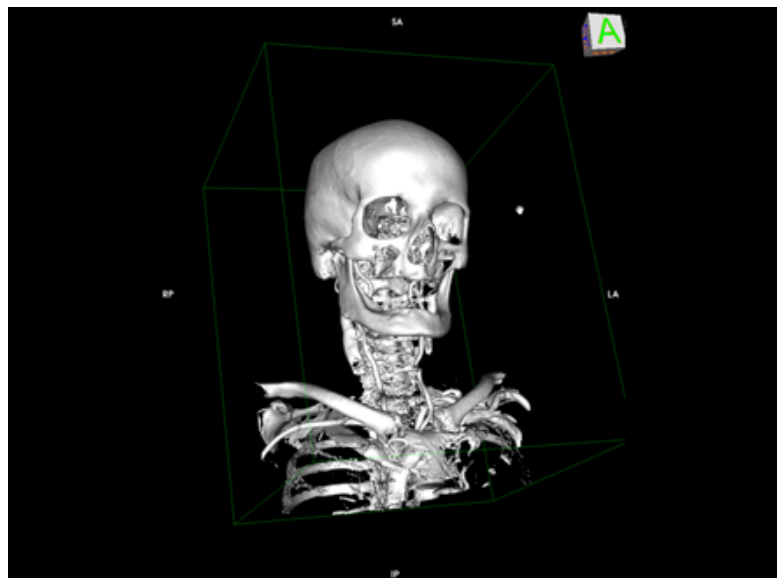


Figure 4.1: Example of Upper Torso and Head Model Rendered in Surface Rendered View

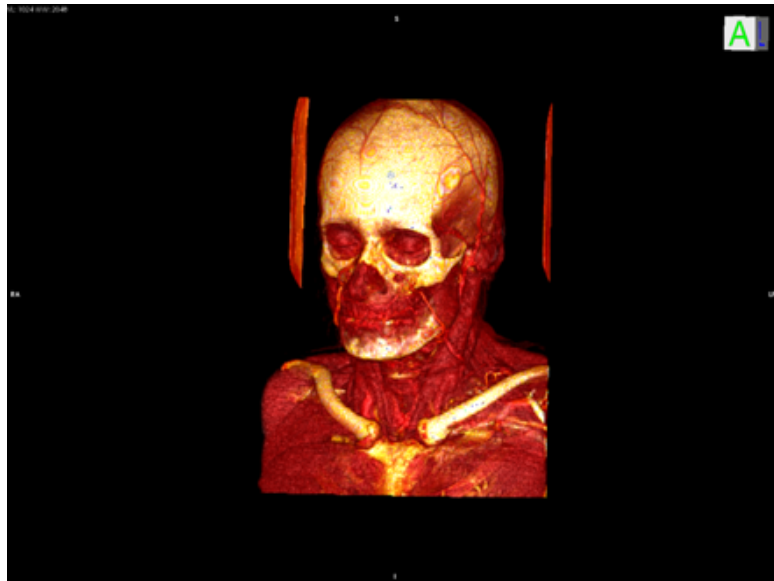


Figure 4.2: Example of Upper Torso and Head Model Rendered in Volume Rendered View

#### 4.2.1 Building VTK

The development of this plug in requires the building of the VTK libraries. VTK is an open source library that uses OpenGL for simulations and visualisations. It has regular updates on the VTK home page and the build used for this project was taken on July 9th 2008. VTK is built using CMAKE. CMAKE is a portable build system, it allows a developer to prepare one version of their source and then build it on a variety of platforms [Lov06]. Once the VTK libraries are built they are imported into the plug in as necessary.

#### 4.2.2 The Different Views

Osirix has two main types of 3D view. Each has its advantages and disadvantages. Both of these views work by creating a grid in 3D space. The grid's resolution is dictated by the resolution of the images taken. The height and width of the images dictate the depth and breadth of the model, the resolution for the height of the model is dependent on the number of images in a series. The actual height is set in the meta data for the DICOM image set.

##### Surface Rendered View

The surface rendered view is based on selecting a particular intensity level. The view is built by rendering a surface of polygons of this intensity. The initial calculation and rendering is longer for this view but once the model is built no more rendering is required. If the initial selection is accurate, then the performance advantages are significant as only the outer surface is created. The problem with this view is the lack of detail. It renders one intensity level at a time. As a consequence only one or two details can be displayed at once. This means that while it looks impressive its limitations in displaying the interactions between details at different intensities, means that it is not the best option for demonstrating to a medical class or group. An Example of a model of the upper torso and head rendered in this view can be

found in figure 4.1.

### **Volume Rendered View**

A volume rendered view reduces the performance, requiring more power to display a 3D model. In a volume rendered view each voxel, point in 3D space, is given a value. This view renders all of these points. The values for each point's opacity and colour is set using a windowing function and a colour lookup table (CLUT). A windowing function will limit what intensities are displayed. A volume rendered view can support many windows with different CLUTs allowing for several different features to be identified and highlighted at once. This view offers the most detailed representation, and is best suited to familiarising a group with different features. For the increase in detail performance is sacrificed. Everytime the model is moved it must be re-rendered. An Example of a model of the upper torso and head rendered in this view can be found in figure 4.2.

For this project the volume rendered view's detail was deemed worth the trade off in performance. The issue with performance will have little to no effect on the main machine, SciViz. This may reduce the plug in's compatibility on certain setups and machines however for the intended release it will work. The code was developed in such a way that it should be possible to apply the control scheme to a surface rendered view.

### **4.2.3 User Interface**

The user interface was planned out once the view had been decided upon. Osirix's user interfaces are built using cocoa, DETAILS ON COCOA. Cocoa was used to develop the user interface for this plug in. The user interface itself is very sparse. One custom view was created to take up the entire screen. Objective-C stores the user interface and window setups in .nib and .xib files including the properties of windows and any links to variables in the code. Certain properties of these displays are accessible in the code. This is used to ensure that the window fills the screen as much as possible. The option to display the plug in in full screen was explored. This would be far more immersive for the user but will have an impact on the required hardware. For both options the view itself is created in the same way.

### **4.2.4 Creating the View**

The volume Rendered view was selected for the project. The following are the steps in creating the view:

- The DIACOM image set is loaded into memory

The DIACOM image set contains multiple images taken at various slices though the selected body part or whole body. These are handled by `FILL IN DATA SET DETAIL HERE`

- A property Dictionary List is Created

A property dictionary list contains details of both the colours and opacity of items at different intensities, it is made up of an opacity level, colour lookup table, and ranges. This will be applied later.

Red	0	1	1	1
Green	0	0	1	1
Blue	0	0	0	1
Opacity	0	0.2	0.5	1
X	0	1024	1500	2048

Table 4.1: Example of a Colour Lookup Table with Opacity

- The Intensities from the DICOM are loaded into VTK

The voxels are given intensities based on the stored values from the DICOM image set. These values are given space in memory for storage. At this point there has been nothing rendered.

- The voxels are coloured based on their intensities

The property Dictionary List is used to colour the various Voxels. This is accomplished using the colour and opacity transfer functions in VTK. With these and the set intensities level it is possible to select specific parts of the model and outline them clearly.

After these steps a model is rendered on screen. The current property list results in the displaying of a skull with features such as the muscles and blood vessels shown with partial opacity. Because it is created using an altered version of the standard volume rendered view the plug in accepts mouse and keyboard interactions. The next aspect of the plug in is the stereoscopic view.

Development of an accurate representation of the model and an overall working understanding of how the system works was hampered by a lack of documentation in the area. In the end a custom method of rendering the model was created to give more access to various aspects of the model. The existing property table was altered to include a line to set opacity dependant on intensity. While this gives greater control during the coding and initial rendering of the model, it also means that it is too different from the original osirix method to allow alteration later using established tools.

The table itself now has five rows, Red, Green, Blue, Opacity and X. X refers to the intensity. The system searches for the appropriate X value and applies the appropriate values from the corresponding four rows. Graduations are also catered for by taking averages between the two nearest values. A table can specify as many intensities as it requires but it was found that at least four points were required to give the model good colour definition. A sample lookup table can be seen in table 4.1.

### 4.3 Stereoscopic Modes

Stereoscopic modes are visuals which emulate a three dimensional image on a flat surface, which, is usually accomplished by sending a different view to each eye. There are three methods used for accomplishing this in the project. Osirix supports colour anaglyph as standard, unfortunately Osirix does not support any other from stereoscopic view. However VTK does support active glasses, but this was not available within Osirix. The VTK based Views in Osirix were not

setup for this when they were created. In research for a way to enable this support, ‘method swizzling’ was discovered. Method swizzling allows an objective-C developer to add lines of code into the beginning of an already existing method. Unlike a category which replaces the whole method swizzling allows the ‘patching’ of code into a class you don’t have access to [Coc08]. Below is an example of the code used to implement this technique.

```
inline void QBSSwizzleInstanceMethod(Class klass,
SEL originalSelector,
SEL newSelector)
{
    Method originalMethod = nil, newMethod = nil;

    originalMethod = class_getInstanceMethod(klass,
        originalSelector);
    newMethod = class_getInstanceMethod(klass, newSelector);

    // If both are found, swizzle them
    if (originalMethod && newMethod)
    {
        IMP originalMethodImp;

        // Swap method implementations
        originalMethodImp = originalMethod->method_imp;
        originalMethod->method_imp = newMethod->method_imp;
        newMethod->method_imp = originalMethodImp;
    }
    else
    NSLog(@"Error: Methods not swizzled! (Old: -%@ New: -%@
Class:%@)",
        NSStringFromSelector(originalSelector),
        NSStringFromSelector(newSelector),
        NSStringFromClass(klass));
}
```

While attempting to add the extra functionality into the existing plug in, a previous piece of work was discovered which had all these features enabled already. The plug in, quad-buffered stereo, uses method swizzling to enable improved stereoscopic visuals on all 3D views. This extends to the view used for this plug in. The quad-buffered stereo is vulnerable to alterations in the source for Osirix. The method swizzling alters existing methods in an existing Osirix class. If these classes are altered in future revisions of Osirix the plug in will be rendered inoperable [Mah07].

### 4.3.1 Coloured Anaglyph

Coloured anaglyph is the cheapest and easiest form of stereoscopic display. An anaglyph image allows the perception of depth when observed through colored glasses [Dub01]. The Source image is created from two different viewpoints to represent the two eyes. Each of these images is then colour coded to key them to their individual eye. The colours used are cyan and red due to how these are perceived by the visual system. Because the images are separated using colour it is possible to display the result on a standard screen and the separation work is performed by glasses worn by the user. These glasses are very simple and cheap to manufacture consisting of simply a frame, generally made from cardboard, and two different coloured transparent films to be held in front of each eye. Figure 4.3 shows an example of a pair of colour anaglyph glasses.



Figure 4.3: A Pair of Colour Anaglyph Glasses

This method is effective but causes a loss of colour detail due to the separation method. Osirix has a button to activate it on the regular 3D window. Its activation in the plug in was achieved easily.

### 4.3.2 Active Glasses

Active glasses are a mechanical method of separating the two channels intended for the separate eyes. It requires a screen capable of a very high refresh rate (the speed at which the screen is updated). The channels for each eye are displayed on screen alternately on each refresh. So if a screen were to refresh at 120 Hz, or 120 times a second, then all the even refreshes would be for the left eye and the right eye's image would be displayed on the odd refreshes. This results in a complete 3D image being displayed at 60 Hz. To ensure that each eye only sees the image intended for it the user must wear a pair of 'active' glasses. These glasses cover the eye when the image intended for it is not being displayed. Once the speed is fast enough the brain perceives one image viewed in 3D. To keep the rate of the active glasses in time with the screen displaying the content a control box is required. Modern glasses do not need to be tethered to the machine which is creating the visuals, now the rate of changed is generally communicated through an infra-red connection to the control box. This means that there must be a direct line of sight connection between the box and the user. There is a sample pair of active glasses in figure 4.4.

Active glasses are a very expensive method of creating a stereoscopic View. They require a lot of equipment and a very high end display. Current refresh rates on LCD (Liquid Crystal Display) monitors are not sufficient for this method. A CRT (Cathode Ray Tube) monitor would be needed. A pair of Active glasses would be needed for each member of the group using the system.



Figure 4.4: A Pair of Active Glasses

### 4.3.3 Polarised Displays

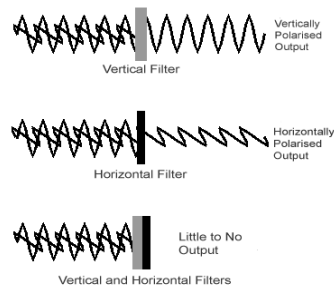


Figure 4.5: Linear Polarised Light

The third method of creating a stereoscopic display is to use a polarised projector system and a matching pair of polarised glasses. Polarised lenses alter the light output from a projector forcing the light to follow one wavelength. Another polarised lens with its axis set  $90^\circ$  to the original lens will cut out all light which has passed through the original lens. If the axis of the second lens is the same as the original lens then no light which passed through the original lens will be blocked by the second lens. This allows for the output from each projector to be channeled to a specific eye. The next step is sending the feed of the image for the correct eye to the correct projector. Light can be polarised either circularly or linearly. Different types of polarisation are used in different fields. Figure 4.5 demonstrates how linear polarising filters work and how they are used to control the light. An example of a pair of polarised anaglyph glasses can be found in figure 4.6.



Figure 4.6: A pair of Polarised Anaglyph Glasses

This method was chosen for this project for several reasons.

1. The Use of projectors allows for a very large display which will increase immersion.
2. The projectors can be mounted behind the screen so as to allow the users to approach the screen without disrupting the image.

3. The relative cost of the glasses for a group is quite low, especially compared to Active Glasses. This allows the system to be implemented quickly, and cheaply for most situations.

This project used circular polarised lenses. Polarised lenses used commonly today in areas such as sunglasses are linear polarised lenses and work as outlined above. This can be seen as a shift in colours and reflections as a lenses horizontal angle alters. Circular polarised lenses remove this issue. This is required to further enhance the sense of immersion. Circular polarisation comes in two forms, right and left circularity. As with linear polarisation mixing the two lenses will

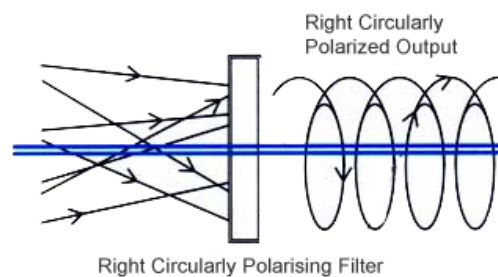


Figure 4.7: Circular Polarised Light

#### 4.3.4 Testing and Outcome

To decide the best option for the stereoscopic aspect of this project each of the above were tested. The only option which will work on all setups of Osirix is the colour anaglyph. This is because no special hardware is needed for the display. For this reason it was the only option which was tested on the regular machine, GameDev. The ability to activate this form of display was already present in Osirix's own Volume Rendered View. It was a simple matter to activate this code using a button event. The activation and de-activation of the stereoscopic view worked flawlessly and the effect was immediately noticeable. The main issue was the loss of colour detail.

After this was implemented and tested, it was then decided to implement the active glasses technique. After researching the various methods of creating stereoscopic views with VTK it was found that many graphics cards will identify if they have two monitors connected when an active glasses signal is being displayed and if this is found then they will split the different signals to the separate displays. With this in mind the two projectors were aligned as close as possible and the code was updated to activate the active glasses method of creating the stereoscopic view. The Quad-Buffered Stereo plug in was also running to enable the stereo throughout the various 3D views. When running a flicker was perceived in one eye but the feeds for left and right were being split successfully. The flicker was the feed intended for the active glasses but was reduced and even removed when the model was being moved.

The active glasses were tested first. It was found that the refresh rate of the display was insufficient and the effect was never truly realised. For this reason this method of creating the

view was abandoned and it was decided to concentrate on the Polarised view. Apart from the flicker the 3D effect was realised effectively. This was particularly noticeable under the Surface Rendered View. This was expected due to the fluidity of the movement in that view along with its overall performance.

## 4.4 Integration

After all aspects of the project had been analysed they were integrated into a functioning plug in. The custom built View forms the basis of the plug in. After the decision to develop the wiiMote controls in a separate class the connection between the wiiMote and that custom view was quite straightforward. The plug in follows these following basic steps while running

- Begin creation of Custom View
- Begin searching for any wiiMote attempting to connect
- Creating 3D Model
- Finalise connection to the wiiMote
- Begin using the stream of events from the wiiMote to control the 3D model
- Listen for disconnect or request to close
- On close request or sudden disconnect end program and clear memory

With this plan outlined, implementing this control was performed step by step. As connection and disconnection were already implemented in the wiiMote plug in from earlier, a second line of code called the custom view plug in which is now implemented as a class of the wiiMote plug in. This allows for the building of the view while the wiiMote connection is being established.

The following is an example of how the wiiMote object is created separately to the second class which is called wiiInteractor in this occasion.

```
- (int) WiiR:(ViewerController *) vc
{
int err=0;

self = [super init];

//Prepare for wiiMote Connection
wii = nil;
discovery = [[WiiRemoteDiscovery alloc] init];
[discovery setDelegate:self];
[[NSNotificationCenter defaultCenter]
addObserver:self
//Allow for additions through expansion Port.
```

```

//Possible added Functionality with accessories
selector:@selector(expansionPortChanged:)
name:@"WiiRemoteExpansionPortChangedNotification"
object:nil];

retries = 1; //Retry Count For Connections
//Create a new WiiInteractor object. This is the view where commands are sent
wiiInteractor = [[WiiInteractor alloc] init];

//Begin Loop and Wiimote Interaction
NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
{
    [self doDiscovery: retries];
}

err = [wiiInteractor showVRPanel:vc:self];
if(!err)
    currentController=wiiInteractor;
[pool release];
return err;
}

```

#### 4.4.1 Control

The next step was the exposing of the controls for the view to the wiiMote plug in. While the mouse control was already implemented this did not translate well into wiiMote control. The control used by Osirix is a mix of both the regular VTK solution as well as some custom tools for specific work within Osirix. It was possible to take advantage of some of these custom controls. Firstly 3D rotation was enabled using the ‘Vertical’ and ‘Azimuth’ commands created by Osirix. The values read from two of the three accelerometers were used to control how these worked. Those values had to be altered slightly to reduce the sensitivity of the controls to allow them to be usable. The 3 axis needed for basic control of the model are shown in Figure 4.8.

The first control issue arose when attempting to enable horizontal rotation. It seems that due to the several layers of abstraction many of the VTK commands are not fully accessible by a plug in. Several attempts were made to expose the commands of VTK so as to easily route commands from the wiiMote, the results were repeated failures. The decision was finally made to use the infra-red sensor to allow a form of mouse pointing.

As discussed before the wiiMote has an infra-red sensor as one of its inputs. This sensor or camera is intended to use a sensor bar to orientate itself and give out a position being pointed at. The sensor bar is simply two groups of IR LEDs (infra-red Light Emitting Diodes) which are used to derive a point. These groups are located by the IR camera which records their position using an x and y co-ordinate. These co-ordinates are recorded for up to four IR points. The wiiMote Framework outputs these four points as an array or will also output a single set

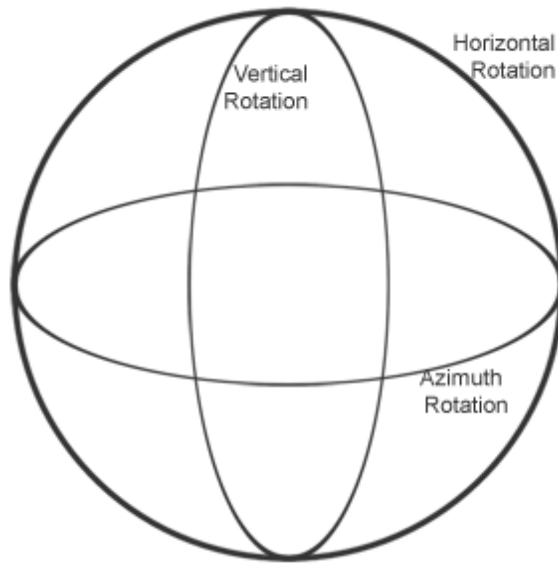


Figure 4.8: The 3 different Axis

of co-ordinates which refer to a position on screen. This point is attained through a series of calculation and derivations performed within the framework. Once that is complete the plug in refines the output to ensure it is within the screen and more importantly the window used to display the model.

After allowing the wiiMote to give a position on screen a mouse event was created which equated to the mouse moving to a new point. Due to the speed at which the wiiMote updated its position the motion of the pointer was very smooth and fluid. This meant that the wiiMote was an operational replacement for the mouse. Mouse clicks and key modifiers could now be assigned to the various buttons to allow for the control which was already accessible through the mouse.

At this point flags were created for each of the wiiMotes buttons. It was decided that the key tools from Osirix to do with models were first the rotation, which could not be implemented using any of the accelerometers. The other two tools which were used were the ability to move the model around the screen which was accomplished by panning the camera, and zooming which dollies the camera in and out from the model. The three of these commands are activated through combinations of key presses on the keyboard. Events for these keypresses are created when buttons on the wiiMote are pressed along with the event for the mouse movement and click.

In this code segment the mouse pointer is being moved. The variable

`point`

actually contains two variables which are x and y co-ordinates. The CGEvent recognises this. The variable values are set in the wiisualisation class and are handed through when the method is called.

```
//Mouse Pointer Movement
```

```

if (!isLeftButtonDown && !isRightButtonDown){
CFRelease(CGEventCreate(NULL));
// this is Tiger's bug.

CGEventRef event = CGEventCreateMouseEvent(NULL,
kCGEventMouseMoved,
point,
kCGMouseButtonLeft);

CGEventSetType(event, kCGEventMouseMoved);
// this is Tiger's bug.
// see also: http://lists.apple.com/archives/Quartz-dev/2005/Oct/msg00048.html

CGEventPost(kCGHIDEventTap, event);
CFRelease(event);
}

```

With the mouse movement implemented more code was developed to quickly test and implement new commands based on mouse presses. After testing the various movement controls it was found that it was very easy to lose the model off the screen. This was especially true when the user is zoomed in and attempts to rotate the model about the focal point. To remedy this it was decided to develop a reset button. When the 2 button was pressed the model is immediately returned to its starting position. This was possible through the use of several existing methods which were found while researching the CMIV Tagged Volume Rendering plug in. The plug in refers to the four key positions which would be viewed. One was selected and the method responsible was simply activated once the 2 button was pressed. When this was implemented it was found that this was called twice, once for the press and once for the release. There is a boolean variable associated with the event. Boolean variables are either true or false. Flags were used to track which buttons are pressed in the plug in. All methods are set to only run when a true value is set for their corresponding buttons.

Another issue which was identified while testing is a control issue in many computer game control schemes. The issue arises for whether when the user points the wiiMote up the model begin to rotate towards, or away from the user. There is no definitive decision on which would be correct, nor is there a consensus between those who used the plug in which feels correct. For this reason the ability to invert the direction was included in the code and the inversion can be activated at any point during the operation of the plug in. This control was assigned to the 1 button.

The code contained here is concerned with normalising the values from the accelerometer to avoid the system being too sensitive. The inversion code is implemented here too

```

- (void) accelerationChanged:(WiiAccelerationSensorType)type
accX:(unsigned short)accX accY:(unsigned short)accY accZ:(unsigned short)accZ
{
float angleX, angleY, angleZ;

```

```

angleX = accX;
angleX -=258;
angleX /=20;

angleY = accY;
angleY -=260;
angleY /=20;

angleZ = accZ;
angleZ -= 300;
angleZ /= 20;

//B rotate set for IR Testing
if(aButton && !bButton)
{
if(invert){
angleY = -angleY;
}

[wiiInteractor accelCameraMove:-angleX :angleY];
}
}

```

Below are the controls for the plug in:

- **Home Button** - End Plugin and disconnect wiiMote.
- **- or + Buttons** - When Pressed with the **B Button** activates the zoom function.
- **A Button** - Activates the Accelerometer movement.  
When pressed with the **B Button** activates the pan function.
- **B Button** - Rotates the model on the horizontal axis
- **1 Button** - Toggles the Inversion of the horizontal axis
- **2 Button** - Resets the Model to its original position

#### 4.4.2 Test and Debug Functionality

When the plug in was operational the opportunity was taken to develop other functionality and test the system. Code was developed for testing new code quickly. A tag was assigned to each of the four buttons on the directional pad. A method was written which recognised these tags allowing for simple commands to be written, deployed and tested rapidly. This was used to experiment with both new functions and new ways to create current functions more efficiently. It was when running these experiments that the aspect of Objective-C called categories was

discovered. Categories allow new methods to be included in existing classes. It was thought that this could be used to add the required support for creating stereoscopic views as outlined above. Categories were used to implement the method swizzling.

Categories can also alter existing methods. This was explored for the toggling of stereoscopic views. There was an existing method in Osirix's VRView class to accomplish this. The existing method activated the colour anaglyph stereoscopic view. A category was used to replace the existing method with one which activated active glasses stereoscopic view.

This code is the the new version of SwitchStereoMode which replaces the version in the volume rendered view in Osirix

```

-(IBAction) SwitchStereoMode :(id) sender
{
if( [self renderWindow]->GetStereoRender() == false)
{
[self renderWindow]->StereoRenderOn();
[self renderWindow]->SetStereoTypeToCrystalEyes(); // Mode For Active Glasses
}
else
{
[self renderWindow]->StereoRenderOff();
}

[self setNeedsDisplay:YES];
}

```

Categories can also add new functionality. In an attempt to allow access to more controls and tools a new method was created. This method alters the current tool active for the view. The method was designed to cycle through the available tools. The current tool variable alters the result of a mouse click. The rotate, pan and zoom functions are all examples of these tools. The following code is the new method which is implemented into the volume rendered view. It can access and alter existing variables but cannot create any new global variables

```

-(void)swapTools:(short)tool
{
fprintf(stderr, "Changing Tool\n");
[self setCursorForView: tool];
[self setCurrentTool:tool];
}

```

## 4.5 Issues

As always there were issues both with the control and the code itself. A main issue which affected most of the development was the abstraction between the plug in and VTK. This removed the ability to use direct VTK control causing the final control scheme to not be

exactly as planned. It also took a lot of time to explore and diagnose. The errors and problems were not immediately obvious and any attempts to address the issues had no success for no apparent reason. It was these issues that caused the development to concentrate on using the wiiMote as a mouse.

The other issue which caused confusion was in the use of categories. It was unknown why certain variables were inaccessible to the category. This was discovered when code was copied directly from the original VRView class. It was thought that a category would have full access to all the existing variables of its parent class. This was not the case and so the functionality which could be implemented through a category was limited, which took a lot of time to analyse.

## Chapter 5

# Evaluation

This Chapter details the tests performed on the project. The tests have been designed to examine both the system and the usability of the project. The systems tests consist of tracking the CPU usage, the memory usage and the load on the graphics card on both machines. The usability tests consisted of the project being demonstrated to a group of four users of varying ability and familiarity with both computers and medical imaging. After this, each of the users were asked to use the plug in and grade it on several aspects.

The same steps were used in all tests in order to maintain consistency. This allowed for the results from the system tests to be relevant to each other and gave the user tests some structure. The steps were:

- Open Osirix
- Select Data Set
- Start Visualisation Plug in
- Perform 360° Azimuth Turn
- Perform 360° Vertical Turn
- Perform 360° Horizontal Rotation
- Use pan function to move the model to the 4 corners and back to the center
- Use zoom to reduce the model to a pinpoint
- Use zoom to have the model fill the screen
- Use reset
- End Visualisation Plug in
- Close Osirix

	GameDev			
	Run1	Run2	Run3	Average
CPU (%) (Min)	12.2	13.24	7.84	<b>11.09</b>
(Max)	97	100	100	<b>99.00</b>
(Average)	65	54.64	65	<b>61.55</b>
Memory (MB) (Min)	660.07	653.69	835.87	<b>716.54</b>
(Max)	850.54	913.1	1008.42	<b>924.12</b>
(Average)	739	828	926	<b>831.00</b>
VRAM Free (MB) (Min)	54.49	57.14	56.94	<b>56.19</b>
(Max)	91.17	98.73	98.72	<b>96.21</b>
(Average)	60.05	57.64	72.49	<b>63.40</b>

Table 5.1: Results Of Systems Tests for GameDev

	SciViz			
	Run1	Run2	Run3	Average
CPU (%) (Min)	1.11	1.36	1.72	<b>1.40</b>
(Max)	97	62.55	60.87	<b>62.10</b>
(Average)	28	17	24	<b>23.00</b>
Memory (MB) (Min)	783.13	672.78	732.79	<b>729.57</b>
(Max)	1080	898.21	924.69	<b>967.63</b>
(Average)	1011	890	910	<b>937.00</b>
VRAM Free (MB) (Min)	1423	1419	1429	<b>1423</b>
(Max)	1460	1490	1437	<b>1462</b>
(Average)	1435	1431	1433	<b>1433</b>

Table 5.2: Results Of Systems Tests for SciViz

## 5.1 System Tests

The project was tested using Instruments. Instruments is part of the Xcode suite for developers. It allows for multiple aspects of a system to be inspected while the program is being run. The values are recorded and graphed so any changes can be spotted immediately. Table 5.1 and table 5.2 contain summaries of the recorded values. The full details are contained on the original trace files which are stored on the disc accompanying this document. The table outlines each system's usage of the CPU, the memory and to track the visual performance the remaining VRAM (Video Memory) was in absence of a better metric.

To give a better awareness of the difference in the performance of each system graphically a benchmark was taken on each. OpenMark was the application used to perform the benchmark. This application runs a test with an increasing number of polygons and tracks the changes in frame rate, the number of time the screen is refreshed. It then outputs a log and gives the system a score. This was run three times on each system in order to ensure a fair scoring. The results are outlines in Table 5.3.

	Machine	Run1	Run2	Run3	Average
OpenMark Score	GameDev	8,520	8,520	8,520	<b>8,520</b>
	SciViz	27,676	27,320	27,320	<b>27,439</b>

Table 5.3: Results of OpenMark Benchmarking

As is evident in table 5.3 SciViz has a significant advantage over GameDev. This allows it to render much larger datasets. The dataset used for this test was from the base of the nose to the top of the head. It consisted of 48 images or slices. An example of this model can be seen in Figure 5.1. This was the largest dataset that the GameDev machine could render and still perform to a reasonable standard but it still caused the processor to require 100% usage at certain points.

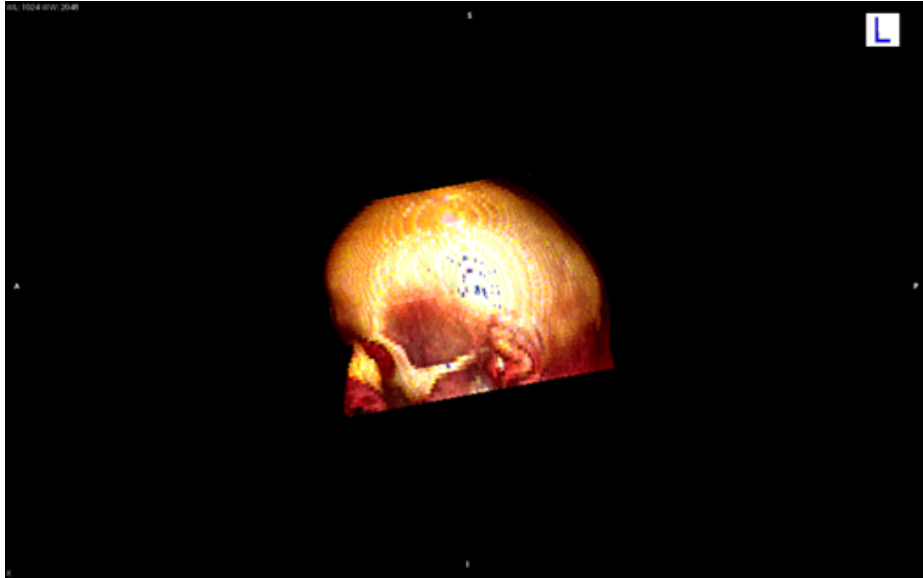


Figure 5.1: Example of Skull Model Rendered in Visualization

The SciViz machine showed no signs of being unable to perform the required at any point during the tests. A larger set was also rendered on this machine. It was the upper torso and head and consisted of 460 images. An example of this model can be seen in Figure 5.2. While some slowing in frame rate was perceived when the model was being moved quickly, the overall performance was still very fluid and smooth. No figures were taken for this test and the result was simply the users opinion.

## 5.2 User Tests

For the User Tests a group of four users were gathered. The size of the group was limited by the available testing area. The plug in was initially demonstrated to them by a user who was familiar with it. They were shown the various controls and features before being asked to interact with the system themselves. They were then asked to score the system in three ways, Control, Visuals and finally they were asked to give the project an Overall score. To ensure a fair scoring they were asked to also rate the plug in using the mouse interaction and lastly they were asked to rate Osirix on its own doing the same tasks.

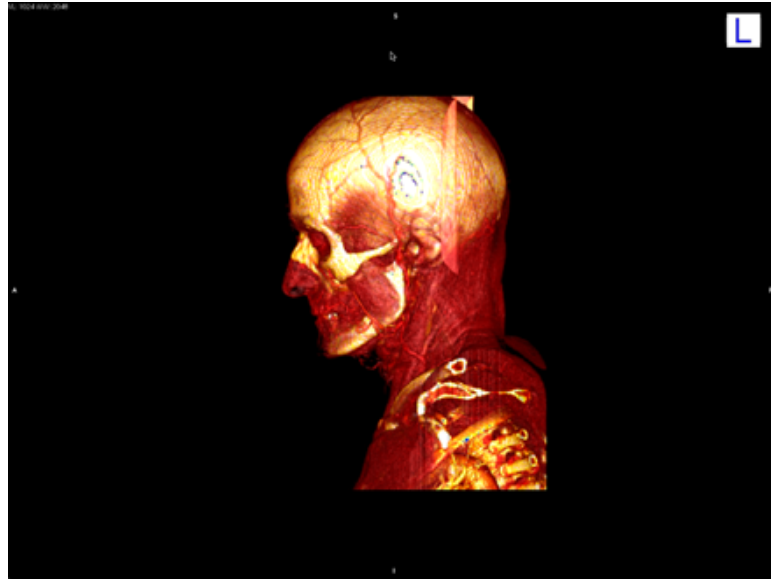


Figure 5.2: Example of Upper Torso and Head Model Rendered in Wiisualisation

		Wiisualisation	Mouse Control	Osirix
GameDev	Control	9	8	7
	Visuals	7	8	9
	Overall	8	7	6
SciViz	Control	8	6	8
	Visuals	7	6	9
	Overall	8	6	8

Table 5.4: User 1's Scores

### 5.2.1 The Users and their Scores

**User 1** has an Masters in Geography. They are familiar with scientific visualisations but are less familiar with alternate versions of control. They have moderate skills with computer interaction but would have less than other users in this test. This user is the shortest in the group. This caused issues in controlling the plug in with the wiiMote on the large projector. This was due largely to the setup and it is hoped that this could be remedied in a larger room. Their scores are stored in table 5.4.

**User 2** is a civil engineer. They are very familiar with working with models. They are also very familiar with alternate methods of computer interaction and control. This user also fully understands the concepts of frame rates and delay. They have no experience with medical models however. This user had issues with frame rate on both machines. This was most noticeable when moving the models quickly. Their scores are stored in table 5.5.

**User 3** is a journalist. Their knowledge of computer interactions is similar to user 1's. They would also have very little experience with scientific models. They were chosen to get the opinion of a student who would have little to no experience with this form of teaching or interactions. Their scores are stored in table 5.6.

**User 4** is a masters student in physics who has experience in biomedical instruments. They

		Wiisualisation	Mouse Control	Osirix
GameDev	Control	9	8	6
	Visuals	6	6	5
	Overall	9	7	5
SciViz	Control	7	6	7
	Visuals	5	5	9
	Overall	8	7	6

Table 5.5: User 2's Scores

		Wiisualisation	Mouse Control	Osirix
GameDev	Control	9	8	7
	Visuals	7	6	8
	Overall	8	7	8
SciViz	Control	6	8	8
	Visuals	6	7	10
	Overall	7	7	9

Table 5.6: User 3's Scores

are familiar with medical models but not the ability to diagnose through them. They would be very familiar with various forms of computer interaction and have similar knowledge of frame rates and delay to user 2. Their scores are stored in table 5.7

### 5.2.2 Results of Analysis

These tests highlighted some issues. It was found that the current setup of the SciViz machine is not ideal. The performance of the stereoscopic display and overall control of the project would be much easier in a larger room. At the moment it is quite awkward to control as you must aim slightly higher than is intuitive. Another surprise result is the visual score for Osirix on its own on SciViz. This is probably because the Surface Rendered Version of the model was demonstrated to the group. This version looks very impressive and the stereoscopic effect is much more pronounced. There are also no issues with frame rate making this perfect for showing the system's capabilities. Table 5.8 outlines the average user scores.

		Wiisualisation	Mouse Control	Osirix
GameDev	Control	9	7	6
	Visuals	8	8	7
	Overall	8	7	7
SciViz	Control	8	7	7
	Visuals	8	6	9
	Overall	9	6	8

Table 5.7: User 4's Scores

		Wiisualisation	Mouse Control	Osirix
GameDev	Control	<b>9</b>	<b>7.75</b>	6.5
	Visuals	<b>7</b>	<b>7</b>	7.25
	Overall	<b>8.25</b>	<b>7</b>	6.5
SciViz	Control	7.25	7.75	<b>7.5</b>
	Visuals	6.5	6	<b>9.25</b>
	Overall	8	7.5	<b>7.75</b>

Table 5.8: Average User Scores

### 5.3 Overall

In general these tests confirmed several assumptions about this project. It was believed that the performance would be an issue on lower end machines such as GameDev. This came into play with the limited size of a dataset. The machine's performance was also topped out. However this was not an issue with the SciViz machine. As long as the project is run on a machine with adequate memory available then it should have no issues running.

The other aspect which has been an area of concern through the project was the choice for the view type. As seen when demonstrating the surface rendered view in stereoscopic mode to the group it is far more impressive. At the beginning of the project it was decided to go with the more functional volume rendered view but now it seems that may have been a mistake. However with the method of separating control from function in the code it would not be a difficult task to attempt to route the control to a surface rendered view.

# Chapter 6

## Conclusions

In this chapter the project results are analysed and compared to the goals set out at the start of the project.

### 6.1 Success of Work

In general the project was a success. The initial goal was to have a 3D medical visualisation controlled using the wiiMote with stereoscopic visuals. The three key points in this statement were completed. The plug in itself created a custom version of Osirix's existing volume rendered View. This accepted control from the wiiMote to interact with the resulting model. The stereoscopic visuals were the weakest aspect of the project.

#### 6.1.1 3D Medical Visualisations

Osirix already created accurate and highly detailed medical models. The greatest challenge here was analysing how these models were built and getting access to that process. The commenting in Osirix's code is very sparse making it very difficult to follow the logical flow of the program. This coupled with the lack of detailed documentation made the creation of a custom version of the view a very difficult and drawn out process. The discovery of the CMIV's Tagged Volume Rendering was of great assistance for this aspect of the project.

#### 6.1.2 WiiMote Control

This was the most straight forward aspect of the development process. The community support and documentation of the wii remote framework made the connection and event handling a very simple process. These controls were easy to implement up until they required new versions of controls for Osirix. Because the development was occurring in a very specific environment, a plug in, it did not have easy access to all aspects of Osirix. There were several occasions, such as the attempts to implement rotation using the accelerometer, where an ability to alter existing classes of Osirix would have allowed the task to be completed in a much shorter time.

However without attempting to develop a whole new version of Osirix it is unknown exactly what challenges would have been met there.

### **6.1.3 Stereoscopic Display**

A lot of time was devoted to getting a stereoscopic display functional. The discovery of the Quad-Buffered Stereo plug in which had completed and tested all the method swizzling which was being started on this project was fortunate. The other issues which plagued this area of development were hardware related. The Room being used for the display is not fully suitable. If a new area were selected the stereoscopic display would be significantly improved. Most of the time spent on this area of the project was devoted to aligning the two projectors. This was also hampered by the room size, forcing the projectors to be too close to the screen to allow for the disparity in their positions. To improve and develop a proper stereoscopic display a new room would be needed.

### **6.1.4 Overall**

The decision to separate control from function means that this project can help the Osirix plug in development community to implement these controls into a wide variety of other projects. This project itself serves as a useful method of demonstrating Osirix in a group situation.

## **6.2 Planned Future Work**

It is thought that if it were to be developed as a new build of Osirix then extra functionality could be incorporated. Control could be seamless. Ideally once the application is started and the connection to the wiiMote is successful the mouse and keyboard would no longer be necessary. However this level of development and integration would require more time than was available to this project.

The Stereoscopic views could also be analysed and improved. The setup of the system should be carefully analysed and researched to take the intended forms of interaction to be facilitated.

It would also be interesting to see exactly how the wiiMote controls are adopted if at all by the osirix development community. This form of control has been widely adopted by nearly all other aspects of independent development.

# Bibliography

- [ACM08] Bert Altenberg, Alex Clarke, and Philippe Mouglin. *Become an Xcoder: Start Programming the Mac Using Objective-C*, 2008.
- [AsBB<sup>+</sup>04] Lisa S. Avila, sebastien Barre, Rusty Blue, Berk Geveci, Amy Henderson, William A. Hoffman, Brad King, C. Charles Law, Kenneth M. Martin, and William J. Schroeder. *VTK User's Guide*. Kitware Inc, 2004.
- [Coc08] CocoaDev. Method swizzling, lightweight modification of a single method at runtime. <http://www.cocoadev.com/index.pl?MethodSwizzling>, 2008.
- [Cro94] Charles Crook. *Computers and the Collaborative Experience of Learning*. Routledge, 1994.
- [CTL<sup>+</sup>99] P. Chios, A. C. Tan, A D Linney, G. H. Alusi, A. Wright, G. J. Woodgate, and D. Ezra. The potential use of an autostereoscopic 3d display in microsurgery. *Springer-Verlag Berlin Heidelberg*, 1999.
- [Doi07] Kunio Doi. Computer-aided diagnosis in medical imaging: Historical review, current status and future potential. *Elsevier, Computerized Medical Imaging and Graphics*, 2007.
- [Dub01] E. Dubois. A projection method to generate anaglyph stereo images. *Acoustics, Speech, and Signal Processing, 2001. Proceedings.*, 2001.
- [Haa98] Jaap Haartsen. Bluetooththe universal radio interface for ad hoc, wireless connectivity. *Ericsson Review*, 1998.
- [Haa00] J.C. Haartsen. The bluetooth radio system. *Personal Communications, IEEE*, 2000.
- [HHR<sup>+</sup>08] Rolf Heckemann, Alexander Hammers, Daniel Rueckert, Richard Aviv, Christopher Harvey, and Joseph Hajnal. Automatic volumetry on mr brain images can support diagnostic decision making. *BMC Medical Imaging*, 8(1):9, 2008.
- [inc08] Apple inc. *The Objective-C 2.0 Programming Language*, 2008.
- [Joh08] Johnson, C. Daniel and Chen, Mei-Hsiu and Toledano, Alicia Y. and Heiken, Jay P. and Dachman, Abraham and Kuo, Mark D. and Menias, Christine O. and Siewert,

- Betina and Cheema, Jugesh I. and Obregon, Richard G. and Fidler, Jeff L. and Zimmerman, Peter and Horton, Karen M. and Coakley, Kevin and Iyer, Revathy B. and Hara, Amy K. and Halvorsen, Robert A., Jr. and Casola, Giovanna and Yee, Judy and Herman, Benjamin A. and Burgart, Lawrence J. and Limburg, Paul J. Accuracy of CT Colonography for Detection of Large Adenomas and Cancers. *N Engl J Med*, 359(12):1207–1217, 2008.
- [Ken08] Carl Kenner. Glovepie homepage, 2008.
- [LAGS05] David Levina, Usaf Aladlb, Guido Germanoc, and Piotr Slomkac. Techniques for efficient, real-time, 3d visualization of multi-modality cardiac data using consumer graphics hardware. *Elsevier, Computerized Medical Imaging and Graphics*, 2005.
- [Lee08a] Johnny Chung Lee. Johnny Lee: Creating tech marvels out of a \$40 wii remote. [http://www.ted.com/index.php/speakers/johnny\\_lee.html](http://www.ted.com/index.php/speakers/johnny_lee.html), 2008.
- [Lee08b] Johnny Chung Lee. Projects - Wii. <http://www.cs.cmu.edu/~johnny/projects/wii/>, 2008.
- [Lov06] Tanner Lovelace. Cmake: The cross platform build system. *Linux Magazine*, 2006.
- [Mah07] Jonathon Mah. Quad-buffered stereo project. <http://www.jonathonmah.com/devetc/projects>, 2007.
- [Mai96] Pat Maier. *Using technology in Teaching and Learning*. Interactive Learning Center, 1996.
- [MD04] James D. Thomas MD. The dicom image formatting standard:what it means for echocardiographers. *Journal of Digital Imaging*, 2004.
- [MMP04] Antoine Rosset MD, Luca Spadola MD, and Osman Ratib MD PhD. Osirix: An open-source software for navigating in multidimensional dicom images. *Journal of Digital Imaging*, 2004.
- [San97] Judith Haymore Sandholtz. *Teaching with Technology : Creating Student Centered Classrooms*. Teachers College Press, 1997.
- [Sex99] P. Sexton, I. Surman. Stereoscopic and autostereoscopic display systems. *Signal Processing Magazine, IEEE*, 1999.
- [Shu62] William A. Shurcliff. *Polarized Light : Production and Used*. Harvard University Press, 1962.
- [TED08] TED. Technology, entertainment, design. <http://www.ted.com/>, 2008.
- [WG07] Xiao Hui Wang and Walter F. Good. Real-time stereographic rendering and display of medical images with programmable gpus. *Elsevier, Computerized Medical Imaging and Graphics*, 2007.
- [Whe08] Guy Wheeler. Darwiin remote homepage. <http://sourceforge.net/projects/darwiin-remote/>, 2008.

# Chapter 7

## Appendix

### 7.1 Appendix A: Wiisualisation Controls

#### WiiMote Button Function

A	Activate Accelerometers
B	Rotate
1	Toggle Camera Inversion
2	Reset Model
A & B	Move Model Around Screen
+ or - & B	Activate Zoom

## 7.2 Appendix B: System and User Test Results

The following are the results of the usability and performance tests conducted on the project.

		Run1	Run2	Run3	Average
GameDev	CPU (%) (Min)	12.2	13.24	7.84	<b>11.09</b>
	(Max)	97	100	100	<b>99.00</b>
	(Average)	65	54.64	65	<b>61.55</b>
	Memory(MB) (Min)	660.07	653.69	835.87	<b>716.54</b>
	(Max)	850.54	913.1	1008.42	<b>924.12</b>
	(Average)	739	828	926	<b>831.00</b>
	VRAM Free(MB) (Min)	54.49	57.14	56.94	<b>56.19</b>
	(Max)	91.17	98.73	98.72	<b>96.21</b>
	(Average)	60.05	57.64	72.49	<b>63.40</b>
	OpenMark Score	8,520	8,520	8,520	<b>8,520</b>
SciViz	CPU (%) (Min)	1.11	1.36	1.72	<b>1.40</b>
	(Max)	97	62.55	60.87	<b>62.10</b>
	(Average)	28	17	24	<b>23.00</b>
	Memory(MB) (Min)	783.13	672.78	732.79	<b>729.57</b>
	(Max)	1080	898.21	924.69	<b>967.63</b>
	(Average)	1011	890	910	<b>937.00</b>
	VRAM Free(MB) (Min)	1423	1419	1429	<b>1423</b>
	(Max)	1460	1490	1437	<b>1462</b>
	(Average)	1435	1431	1433	<b>1433</b>
	OpenMark Score	27,676	27,320	27,320	<b>27,439</b>

Table 7.1: Results Of Systems Tests

			Wiiisualisation	Mouse Control	Osirix
User 1	GameDev	Control	9	8	7
		Visuals	7	8	9
		Overall	8	7	6
	SciViz	Control	8	6	8
		Visuals	7	6	9
		Overall	8	6	8
User 2	GameDev	Control	9	8	6
		Visuals	6	6	5
		Overall	9	7	5
	SciViz	Control	7	6	7
		Visuals	5	5	9
		Overall	8	7	6
User 3	GameDev	Control	9	8	7
		Visuals	7	6	8
		Overall	8	7	8
	SciViz	Control	6	8	8
		Visuals	6	7	10
		Overall	7	7	9
User 4	GameDev	Control	9	7	6
		Visuals	8	8	7
		Overall	8	7	7
	SciViz	Control	8	7	7
		Visuals	8	6	9
		Overall	9	6	8
Averages	GameDev	Control	<b>9</b>	<b>7.75</b>	6.5
		Visuals	<b>7</b>	<b>7</b>	7.25
		Overall	<b>8.25</b>	<b>7</b>	6.5
	SciViz	Control	7.25	7.75	<b>7.5</b>
		Visuals	6.5	6	<b>9.25</b>
		Overall	8	7.5	<b>7.75</b>

Table 7.2: Results of User Tests

## 7.3 Appendix C: On The DVD

Root - Source - VTK

Osirix

Wiiisualisation - WiiRemote

Viewer

Images

WiiRemote.Framework

Build

Video - 01082008013.mp4

12082008015.mp4

25092008020.mp4

28082008017.mp4

Images - Whiteboard Images

Screenshots

Osirix - Install Files

Plugins - QuadBufferedStereo.plugin

wiiisualisation. osirixplugin

Sample Datasets

Logs - Bluetooth

System Tests

OpenMark

Thesis - Images

Research Papers

ConorCrowleyThesis.pdf

## 7.4 Appendix D: Websites

The Following are websites which were integral in the development of this project

- <http://3dwii.blogspot.com> - Development Blog for the Project
- <http://developer.apple.com/> - Apple Developer Connection
- <http://www.vtk.org/> - The Visualisation Toolkit Homepage
- <http://www.osirix-viewer.com/> - Osirix Homepage

## 7.5 Appendix E: Source Code

### Wiisualisation.h

```
//
// Wiisualisation.h
// Accept Input from the Wiimote and interact with 3D model
//
// Created by conordjpc on Firday July 25th 2008.
// Copyright (c) 2008 Conor Crowley. All rights reserved.
//
#import <Cocoa/Cocoa.h>

#import <Foundation/Foundation.h>
#import "PluginFilter.h"
//#import "WiiInteractor.h"

//Wii CLASSES
#import <CoreFoundation/CoreFoundation.h>
#import "WiiRemote/WiiRemote.h"
#import "WiiRemote/WiiRemoteDiscovery.h"

@interface Wiisualisation : PluginFilter {

IBOutlet NSWindow *window;
NSMutableDictionary *dataOfWizard;
NSObject* currentController;

// WiiRemote Framework classes
    WiiRemoteDiscovery *discovery;
    WiiRemote *wii;
    int retries;
id waitWindow;
ViewerController *new2DViewer;

//Control Variables
bool aButton, bButton, plusButton, minusButton;
bool upButton, downButton, leftButton, rightButton;
bool oneButton, twoButton, invert;

//Mouse Control Variables
CGPoint point;
```

```
bool leftMouse, rightMouse;

}

- (long) filterImage:(NSString*) menuName;
- (int) WiiR:(ViewerController *) vc;
- (void) doDiscovery:(int)retries;

//Mouse and Keyboard Events
- (void)sendKeyboardEvent:(CGKeyCode)keyCode keyDown:(BOOL)keyDown;

@end
```

## Wiisualisation.mm

```
//
// Wiisualisation.m
// Accept Input from the Wiimote and interact with 3D model
//
// Created by conordjpc on Firday July 25th 2008.
// Copyright (c) 2008 Conor Crowley. All rights reserved.
//

#import "Wiisualisation.h"
#import "WiiInteractor.h"
//#import "VRView.h"
//#import "CLUTOpacityView.h"

//Wii CLASSES
#import <CoreFoundation/CoreFoundation.h>
#import "WiiRemote/WiiRemote.h"
#import "WiiRemote/WiiRemoteDiscovery.h"

@implementation Wiisualisation

WiiInteractor *wiiInteractor;
//IBOutlet VRView *vrViewer;
//IBOutlet CLUTOpacityView *wlwwWindow;

- (void) initPlugin
{

}

- (long) filterImage:(NSString*) menuName
{
//Make sure the Current Controller is Clear
if(currentController)
[currentController release];
currentController=nil;
int err=0;

//If the Wiisualisation option is selected from the menu
//Begin the rendering and rest of plugin
if ( [menuName isEqualToString:NSLocalizedString
```

```

   (@"Wiisualisation", nil)] == YES)
err = [self WiiR:viewerController];

return err;
}

- (int) WiiR:(ViewerController *) vc
{
int err=0;

self = [super init];

//Prepare for wiiMote Connection
    wii = nil;
    discovery = [[WiiRemoteDiscovery alloc] init];
    [discovery setDelegate:self];
    [[NSNotificationCenter defaultCenter]
        addObserver:self

//Allow for additions through expansion Port.
//Possible added Functionality with accessories
selector:@selector(expansionPortChanged:)
name:@"WiiRemoteExpansionPortChangedNotification"
object:nil];

        retries = 1; //Retry Count For Connections - NOT IMPLEMENTED YET!

//Create a new WiiInteractor object. This is the view where commands are sent
wiiInteractor = [[WiiInteractor alloc] init];

//Begin Loop and Wiimote Interaction
NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init]; {
[self doDiscovery: retries];

    }

err = [wiiInteractor showVRPanel:vc:self];
if(!err)
currentController=wiiInteractor;

```

```

[pool release];

return err;

}

//Wii Methods
- (void)doDiscovery:(int)r /*retries*/
{
    retries = r;
    // fix infinite loops
    if(retries <= 0) {
        retries = 1;
    }
    [discovery start];
    waitWindow = [viewerController startWaitWindow:
@" Waiting for Wiimote... Please press 1 + 2 buttons"];
}

// =====
#pragma mark -
#pragma mark WiiRemote notification
// ~~~~~
- (void)expansionPortChanged:(NSNotification *)nc
{
    char *devStr, *stsStr;

    // Check that the Wiimote reporting is the one we're connected to.
    WiiRemote* tmpWii = (WiiRemote*)[nc object];
    if (![tmpWii address] isEqualToString:[wii address]){
        return;
    }

    WiiExpansionPortType epType = [wii expansionPortType];
    switch(epType) {
        case WiiNunchuk:
            devStr = "WiiNunchuk";
            break;
        case WiiClassicController:

```

```

        devStr = "WiiClassicController";
        break;
    case WiiExpNotAttached:
        devStr = "WiiExpNotAttached";
        break;
    default:
        devStr = "Unknown Expansion Unit";
        break;
}

if ([wii isExpansionPortAttached]) {
    [wii setExpansionPortEnabled:YES];
    stsStr = "enabled";
} else {
    [wii setExpansionPortEnabled:NO];
    stsStr = "disabled";
}

fprintf(stderr, "Expansion port: %s %s.\n", devStr, stsStr); fflush(NULL);

}

// =====
#pragma mark -
#pragma mark WiiRemote delegates
// ~~~~~
- (void) wiiRemoteDisconnected:(IOBluetoothDevice*)device
{
    // not doing [wii release] because dealloc will do it
    // However view Must be ended if the wiimote disconnects

    [wiiInteractor disconnect];
}

// ~~~~~
- (void) irPointMovedX:(float)px Y:(float)py
{
    //From Darwiin Remote With Small Tweaks - Prototype - now in wii Interactor
    int dispWidth = CGDisplayPixelsWide(kCGDirectMainDisplay);
    int dispHeight = CGDisplayPixelsHigh(kCGDirectMainDisplay);
}

```

```

float sens2 = .5;

float newx = (px*1*sens2)*dispWidth + dispWidth/2;
float newy = -(py*1*sens2)*dispWidth + dispHeight/2;

//Keep Points Within the screen
if (newx < 0) newx = 0;
if (newy < 0) newy = 0;
if (newx >= dispWidth) newx = dispWidth-1;
if (newy >= dispHeight) newy = dispHeight-1;

float dx = newx - point.x;
float dy = newy - point.y;

float d = sqrt(dx*dx+dy*dy);

// mouse filtering
if (d < 20) {
point.x = point.x * 0.9 + newx*0.1;
point.y = point.y * 0.9 + newy*0.1;
} else if (d < 50) {
point.x = point.x * 0.7 + newx*0.3;
point.y = point.y * 0.7 + newy*0.3;
} else {
point.x = newx;
point.y = newy;
}

if (point.x > 0 && point.y > 0)
[wiiInteractor IRMouse:point :leftMouse :rightMouse];

}

// ~~~~~
- (void) rawIRData:(IRData[4])irData
{
/*
    Insert code for individual IR points
*/

```

```

}
// ~~~~~
- (void) buttonChanged:(WiiButtonType)type isPressed:(BOOL)isPressed
{

if( (type == WiiRemoteHomeButton) && isPressed )//End Program
{
    [wii closeConnection ];
    [wiiInteractor endView];
    }

//A and B Buttons
if( (type == WiiRemoteAButton))
{
aButton = isPressed;
}

if( (type == WiiRemoteBButton))
// Rotate/Pan/Zoom (mouse Tool Function)
{
bButton = isPressed;
leftMouse = isPressed;
if (aButton)//Pan
{
[self sendKeyboardEvent:56 keyDown:NO]; //Shift
[self sendKeyboardEvent:59 keyDown:NO]; // Control
[self sendKeyboardEvent:55 keyDown:isPressed]; // Comand
}
else if (plusButton || minusButton)//Zoom
{
NSLog(@"Zoom ON");
[self sendKeyboardEvent:56 keyDown:isPressed]; //Shift
[self sendKeyboardEvent:59 keyDown:NO]; // Control
[self sendKeyboardEvent:55 keyDown:NO]; // Comand
}
else //Rotate
{
[self sendKeyboardEvent:56 keyDown:NO]; //Shift
[self sendKeyboardEvent:59 keyDown:isPressed]; // Control
[self sendKeyboardEvent:55 keyDown:NO]; // Comand
}
}
}

```

```

}

//1 and 2 Buttons
if( (type == WiiRemoteOneButton))//Toggle Inversion
{
oneButton = isPressed;
if(isPressed)
{
if(invert)
{
invert = false;
}else{
invert = true;
}
}
}

if( (type == WiiRemoteTwoButton))//Reset Image
{
[wiiInteractor reset];
twoButton = isPressed;
}

//Plus and Minus Buttons - Zoom with B button using IR
if( (type == WiiRemotePlusButton))
//Zoom
{
plusButton = isPressed;
}

if( (type == WiiRemoteMinusButton))
//zoom
{
minusButton = isPressed;
}

//D-Pad Controls
if(type == WiiRemoteUpButton)
//CLUTOpacity 0
{

```

```

upButton = isPressed;
[wiiInteractor CLUTOpacityChange:0 :isPressed];
}

if(type == WiiRemoteDownButton)
//CLUTOpacity 1
{
downButton = isPressed;
[wiiInteractor CLUTOpacityChange:1 :isPressed];
}

if(type == WiiRemoteLeftButton)
//CLUTOpacity 2
{
leftButton = isPressed;
[wiiInteractor CLUTOpacityChange:2 :isPressed];
}

if(type == WiiRemoteRightButton)
//CLUTOpacity 3
{
rightButton = isPressed;
[wiiInteractor CLUTOpacityChange:3 :isPressed];
}

}

// ~~~~~
- (void) analogButtonChanged:(WiiButtonType)type
amount:(unsigned short) press
//For Nunchuck
{
    //fprintf(stdout, "analog %d %d\n", (int)type, press); fflush(NULL);
}

// ~~~~~
- (void) joyStickChanged:(WiiJoyStickType)type
tiltX:(unsigned short)tiltX
tiltY:(unsigned short)tiltY
//for Nunchuck
{
    //fprintf(stdout, "joystick %d %d %d\n", (int)type, tiltX, tiltY);
}

```

```

}

// ~~~~~
- (void) accelerationChanged:(WiiAccelerationSensorType)type
accX:(unsigned short)accX
accY:(unsigned short)accY
accZ:(unsigned short)accZ
{
float angleX, angleY, angleZ;
angleX = accX;
angleX -=258;
angleX /=20;

angleY = accY;
angleY -=260;
angleY /=20;

angleZ = accZ;
angleZ -= 300;
angleZ /= 20;

//B rotate set for IR Testing
if(aButton && !bButton)
{
if(invert){
angleY = -angleY;
}

//fprintf(stderr, "camera Move\n"); fflush(NULL);
[wiiInteractor accelCameraMove:-angleX :angleY];
}
}
// =====
#pragma mark -
#pragma mark WiiRemoteDiscovery delegates

// ~~~~~
- (void) WiiRemoteDiscoveryError:(int)code
{
fprintf(stderr, "WiiRemote discovery error code=%x.\n", code);
fflush(NULL);
[discovery close];
}

```

```

retries -= 1;
if(retries > 0) {
    // try again
    if(wii) {
        [wii release];
    }
    [discovery release];
    discovery = [[WiiRemoteDiscovery alloc] init];
    [discovery setDelegate:self];
    [[NSNotificationCenter defaultCenter]
     addObserver:self
     selector:@selector(expansionPortChanged:)
     name:@"WiiRemoteExpansionPortChangedNotification"
     object:nil];
    [self doDiscovery: retries];
} else {
CFRunLoopStop(CFRunLoopGetCurrent());
}
}

// ~~~~~
- (void) willStartWiimoteConnections
{
    fprintf(stderr, "WiiRemote discovered. Starting connection...\n");
    fflush(NULL);
// Close the waiting window
[viewerController endWaitWindow: waitWindow];
}

// ~~~~~
- (void) WiiRemoteDiscovered:(WiiRemote*)wiimote
{
    // the wiimote must be retained because the discovery
    //provides us with an autoreleased object
    wii = [wiimote retain];
    [wiimote setDelegate:self];
//Setup options for wiimote
    [wiimote setLEDEnabled1:NO enabled2:NO enabled3:YES enabled4:NO];
    [wiimote setMotionSensorEnabled:YES];
    [wiimote setIRSensorEnabled:YES];

// Finish Discovery

```

```

    [discovery close];
}

// ~~~~~
//Mouse and Keyboard Interactions
// ~~~~~

- (void)sendKeyboardEvent:(CGKeyCode)keyCode keyDown:(BOOL)keyDown
//
{
    CFRelease(CGEventCreate(NULL));
    // this is Tiger's bug.

    CGEventRef event = CGEventCreateKeyboardEvent(NULL, keyCode, keyDown);
    CGEventPost(kCGHIDEventTap, event);
    CFRelease(event);
    usleep(10000);
}

@end

```

## WiiInteractor.h

```
/*=====
Author: Conor Crowley

Program: Wiimote Interface Wiisualisations

Copyright (c) 2008

=====*/
#import <Cocoa/Cocoa.h>
#import <Foundation/Foundation.h>
#import <AppKit/AppKit.h>
#import "PluginFilter.h"
#import "Wiisualisation.h"
#import "VRView.h"
//#import "CLUTOpacityView.h"

#define id Id
#include <vtkCamera.h>
#include <vtkColorTransferFunction.h>
#include <vtkRenderer.h>
#include <vtkVolumeCollection.h>
#include <vtkVolume.h>
#include <vtkVolumeProperty.h>
#include <vtkVolumeMapper.h>
#include <vtkImageData.h>
#include <vtkPlane.h>
#include <vtkCallbackCommand.h>
#include <vtkObject.h>
#include <vtkPlaneWidget.h>
#include <vtkVolumeRayCastMapper.h>
#include <vtkVolumeRayCastCompositeFunction.h>
#undef id

@interface WiiInteractor : NSObject
{
//Window Outputs
IBOutlet NSWindow *window;
IBOutlet NSWindow *leftWindow;
IBOutlet NSWindow *rightWindow;

```

```

IBOutlet NSColorWell *colorControl;
IBOutlet VRView *vrViewer;
//IBOutlet CLUTOpaCityView *wlwwWindow;

//Image property Storage
int imageWidth,imageHeight,imageAmount;
ViewerController *originalViewController;
float wholeVolumeWL,wholeVolumeWW;
NSMutableArray *propertyDictList;
NSMutableDictionary *curPropertyDict;
NSMutableArray *toolbarList;
NSMutableArray *clutViewPoints;
NSMutableArray *clutViewColors;
NSMutableArray *mutiplePhaseOpacityCurves;
NSMutableArray *mutiplePhaseColorCurves;
NSMutableArray *imagesFor4DQTVR;

//VTK Data
float *originalVolumeData;
float maxInSeries,minInSeries;
vtkRenderer *renderOfVRView;
vtkVolumeCollection *volumeCollectionOfVRView;
vtkVolume *volumeOfVRView;
vtkVolumeProperty *volumePropteryOfVRView,*myVolumeProperty;
vtkColorTransferFunction *myColorTransferFunction;
vtkPiecewiseFunction *myOpacityTransferFunction;
vtkPiecewiseFunction *myGradientTransferFunction;
vtkVolumeMapper *volumeMapper;
vtkVolumeMapper *fixedPointVolumeMapper;
vtkVolumeMapper *rayCastVolumeMapper;
vtkImageData *volumeImageData;
vtkCamera *aCamera;
vtkPlane *clipPlane1;
vtkCallbackCommand *clipCallBack;
vtkPlaneWidget* clipPlaneWidget;
vtkVolumeRayCastMapper* myMapper;
vtkVolumeRayCastCompositeFunction* myCompositionFunction;

float verticalAngleForVR;
float osirixOffset;
unsigned short* realVolumedata;
long maxMovieIndex;

```

```

int isSegmentVR;
Wiisualisation* parent;
NSMutableArray *inROIArray;

//Wii Variables For Controls
float opacityLvl;
int toolNumber;
int propertyNumber;

//Mouse And Keyboard events
//CGPoint point;
bool isLeftButtonDown, isRightButtonDown;

}

- (IBAction)setClipPlaneOrigin:(id)sender;
- (IBAction)switchBetweenVRTorMIP:(id)sender;
- (IBAction)changeToLinearInterpolation:(id)sender;
- (int) initVRViewForSegmentalVR;
- (int) initVRViewForDynamicVR;
- (int) showVRPanel:(ViewerController *) vc :(Wiisualisation*) owner;
- (void) applyOpacity;
- (void)reHideToolbar;

//to cheat VRView
- (float) minimumValue;
- (float) maximumValue;
- (ViewerController*) viewer2D;
- (void)resetClipPlane;
-(NSString*)osirixDocumentPath;

//Wii Controls
- (IBAction) endView;
- (void) disconnect;
- (IBAction)CLUTOpacityChange:(int)tag :(bool)isPressed;
- (void)accelCameraMove:(float)amountX :(float)amountY;
- (void)reset;
- (void)stereoToggle:(bool)pressed;
- (void)IRMouse:(CGPoint)point :(bool)leftMouse :(bool)rightMouse;
- (void)setTool:(int)tag :(CGPoint)point;
- (void)placeROI:(CGPoint)point;
@end

```



## WiiInteractor.mm

```
/*=====
Author: Conor Crowley

Program: Wiimote Interface Wiisualisations

Copyright (c) 2008

=====*/

#import <Cocoa/Cocoa.h>
#import <Foundation/Foundation.h>
#import <AppKit/AppKit.h>
#import "PluginFilter.h"
#import "Wiisualisation.h"
#import "VRView.h"
//#import "CLUTOpacityView.h"

#define id Id
#include <vtkCamera.h>
#include <vtkColorTransferFunction.h>
#include <vtkRenderer.h>
#include <vtkVolumeCollection.h>
#include <vtkVolume.h>
#include <vtkVolumeProperty.h>
#include <vtkVolumeMapper.h>
#include <vtkImageData.h>
#include <vtkPlane.h>
#include <vtkCallbackCommand.h>
#include <vtkObject.h>
#include <vtkPlaneWidget.h>
#include <vtkVolumeRayCastMapper.h>
#include <vtkVolumeRayCastCompositeFunction.h>
#undef id

@interface WiiInteractor : NSObject
{
//Window Outputs
IBOutlet NSWindow *window;
IBOutlet NSWindow *leftWindow;
IBOutlet NSWindow *rightWindow;
}
```

```

IBOutlet NSColorWell *colorControl;
IBOutlet VRView *vrViewer;
//IBOutlet CLUTOpaicityView *wlwwWindow;

//Image property Storage
int imageWidth,imageHeight,imageAmount;
ViewerController *originalViewController;
float wholeVolumeWL,wholeVolumeWW;
NSMutableArray *propertyDictList;
NSMutableDictionary *curProperdyDict;
NSMutableArray *toolbarList;
NSMutableArray *clutViewPoints;
NSMutableArray *clutViewColors;
NSMutableArray *mutiplePhaseOpacityCurves;
NSMutableArray *mutiplePhaseColorCurves;
NSMutableArray *imagesFor4DQTVR;

//VTK Data
float *originalVolumeData;
float maxInSeries,minInSeries;
vtkRenderer *renderOfVRView;
vtkVolumeCollection *volumeCollectionOfVRView;
vtkVolume *volumeOfVRView;
vtkVolumeProperty *volumePropteryOfVRView,*myVolumeProperty;
vtkColorTransferFunction *myColorTransferFunction;
vtkPiecewiseFunction *myOpacityTransferFunction;
vtkPiecewiseFunction *myGradientTransferFunction;
vtkVolumeMapper *volumeMapper;
vtkVolumeMapper *fixedPointVolumeMapper;
vtkVolumeMapper *rayCastVolumeMapper;
vtkImageData *volumeImageData;
vtkCamera *aCamera;
vtkPlane *clipPlane1;
vtkCallbackCommand *clipCallBack;
vtkPlaneWidget* clipPlaneWidget;
vtkVolumeRayCastMapper* myMapper;
vtkVolumeRayCastCompositeFunction* myCompositionFunction;

float verticalAngleForVR;
float osirixOffset;
unsigned short* realVolumedata;
long maxMovieIndex;

```

```

int isSegmentVR;
Wiisualisation* parent;
NSMutableArray *inROIArray;

//Wii Variables For Controls
float opacityLvl;
int toolNumber;
int propertyNumber;

//Mouse And Keyboard events
//CGPoint point;
bool isLeftButtonDown, isRightButtonDown;

}

- (IBAction)setClipPlaneOrigin:(id)sender;
- (IBAction)switchBetweenVRTorMIP:(id)sender;
- (IBAction)changeToLinearInterpolation:(id)sender;
- (int) initVRViewForSegmentalVR;
- (int) initVRViewForDynamicVR;
- (int) showVRPanel:(ViewerController *) vc :(Wiisualisation*) owner;
- (void) applyOpacity;
- (void)reHideToolbar;

//to cheat VRView
- (float) minimumValue;
- (float) maximumValue;
- (ViewerController*) viewer2D;
- (void)resetClipPlane;
-(NSString*)osirixDocumentPath;

//Wii Controls
- (IBAction) endView;
- (void) disconnect;
- (IBAction)CLUTOpacityChange:(int)tag :(bool)isPressed;
- (void)accelCameraMove:(float)amountX :(float)amountY;
- (void)reset;
- (void)stereoToggle:(bool)pressed;
- (void)IRMouse:(CGPoint)point :(bool)leftMouse :(bool)rightMouse;
- (void)setTool:(int)tag :(CGPoint)point;
- (void)placeROI:(CGPoint)point;
@end

```



## VRView\_wiiMote.h

```
//
// VRView+wiiMote.h
// Wiisualisation
//
// Created by Conor on 08/08/2008.
// Copyright 2008 Conor Crowley. All rights reserved.
//
#import <VRView.h>
#import </usr/include/objc/objc-class.h>

@interface VRView (wiiMote)

- (void)swapTools:(short)tool;
-(void)addROIPoint;

//-----
//Altered VRView Methods
//-----
- (IBAction) SwitchStereoMode :(id) sender;
@end
```

## VRView\_wiiMote.mm

```
//
// VRView+wiiMote.m
// Wiisualisation
//
// Created by Conor on 08/08/2008.
// Copyright 2008 Conor Crowley. All rights reserved.
//

#import "VRView_wiiMote.h"

@implementation VRView (wiiMote)

-(void)swapTools:(short)tool
{
printf(stderr, "Changing Tool\n");
[self setCursorForView: tool];
}
```

```

[self setCurrentTool:tool];
}

-(void)addROIPoint:(CGPoint)point
{
// Only Run if NOT pointing at an existing point
if (![self isAny3DPointSelected])
{
double x = point.x;
double y = point.y;

NSLog(@"Adding Point");
// add a point on the surface where wiiMote is pointing
NSLog(@"Point Co-ordinates : %f, %f", point.x, point.y);
long pix[ 3];
float pos[ 3], value;

if( [self get3DPixelUnder2DPositionX:x Y:y
pixel:pix
position:pos
value:&value])
{
[self add3DPoint: pos[0] : pos[1] : pos[2]];
//[controller add2DPoint: pix[0] : pix[1] : pix[ 2] :pos];
}
NSLog(@"Point Placed, Apparently");
[self setNeedsDisplay:YES];
}
}

//-----
//Altered VRView Methods
//-----
//Switch Stereo

-(IBAction) SwitchStereoMode :(id) sender
{
if( [self renderWindow]->GetStereoRender() == false)
{
[self renderWindow]->StereoRenderOn();
fprintf(stderr, "1. - Stereo Render Started\n");
// Mode For Active Glasses

```

```
[self renderWindow]->SetStereoTypeToCrystalEyes();
fprintf(stderr, "2. - Stereo Set to Crystal Eyes\n");
}
else
{
[self renderWindow]->StereoRenderOff();
fprintf(stderr, "1. - stereo render off\n");
}

[self setNeedsDisplay:YES];
}

@end
```