Low cost motion capture system for the implementation of a real-time two-dimensional and three-dimensional mouse

Name: Patrick Holroyd

Degree: Multimedia and Digital Systems (BSc)

Department: SciTech

Candidate Number: 42729

Project Supervisor: Dr Phil Watten

Year: 2008

Statement of Originality

This report is submitted as part requirement for the degree of Multimedia and Digital Systems at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

(Signature of Student) ____________________________

Acknowledgements

I would like to thank the following people:

Dr Phil Watten for his guidance, patience and motivation.

My family for their support, especially my mother and father for their financial assistance.

David Chabbi for his inspiration and motivation.

Summary

This project aims to research, develop and evaluate a gesture controlled, usable real-time two-dimensional (2D) and three-dimensional (3D) mouse by means of a low financial cost motion capture system.

Motion capture is a technique for digitally recording movement. The motion data recorded can be used to animate a 3D model, control a robot or analyse the movements of the subject for scientific research. The progression of technology is making mocap an expanding field, which is more available to researchers and developers.

This project has been developed using the Cocoa programming environment for Mac OS X. The 2D motion tracking system is designed to use a single high-resolution iSight camera and the 3D motion tracking system uses two cameras; a high-resolution iSight camera and an off the shelf low-resolution web cam. Colour-based motion tracking is used to track a coloured thimble attached to the index finger of the user. Feedback is given to the user via an OpenGL 3D cube, which can be either rotated or moved, and a finger location display showing the tracked location of the finger.

This project has been evaluated for success against the original requirements and by means of a user based questionnaire. Tests for latency and jitter were completed to evaluate the performance of the system.

Although several challenges and obstacles were faced during the implementation of the system, the project has been completed and the challenges successfully overcome. Additionally all the project system requirements have been met. In conclusion, colour based finger tracking could be used to implement a gesture controlled real-time 3D mouse in software. There are some issues with speed, but these will reduce as technology advances.

Contents

1. Introduction 1
    1.1 Project Motivation 1
    1.2 Structure of Report 3
    1.3 Professional considerations 4
2. Requirements Analysis 6
    2.1 Project Aim 6
        2.1.1 Requirements 7
    2.2 Background research 7
        2.2.1 Motion Capture Technologies 7
        2.2.2 Optical Systems 8
        2.2.3 Non-optical Systems 8
        2.2.4 Evaluation of Motion Capture Technologies 10
        2.2.5 Tracking 10
        2.2.6 3D Object 11
    2.4 Specification 11
    2.5 Project Management 12
        2.5.1 Project Implementation Flow Diagram 12
        2.5.2 Project Deliverables 13
        2.5.3 Scheduling 13
        2.5.4 Gantt Chart 14
3. Design 15
    3.1 High Level System Overview 15
    3.2 Input 16
    3.3 Tracking and Analysis 17
    3.4 Output 20
4. Implementation 22
    4.1 Platform 22
    4.2 Implementation Stages 23
    4.3 Output Test-Bench 24
    4.4 Static 2D Motion Tracking System 26
        4.4.1 Image Viewer 26
        4.4.2 Custom Filter 27

4.4.3 Tracking algorithm 29
4.5 2D Video Motion Tracking System 30
4.5.1 iSight Capture 30
4.5.2 Filter Implementation 32
4.5.3 Tracking Algorithm Implementation 33
4.5.4 OpenGL Model Implementation 35
4.5.5 GUI 36
4.5.6 Optimisations 39
4.5.7 2D Motion Tracking System Class Diagram 40
4.6 3D Video Motion Tracking System 41
4.6.1 3D Tracking System Class Diagram 43
4.7 Development Testing 44
4.7.1 OpenGL Model Test 44
4.7.2 Filter Test 44
4.7.3 Custom Filter 45
4.7.4 Tracking Algorithm 45
4.7.5 iSight Capture 46
4.7.6 2D video motion tracking system filter test 46
4.7.7 Memory Allocation 47
4.7.8 USB/Firewire Camera 47
4.7.9 GUI Testing 47
5. Evaluation 48
5.1 Requirements Evaluation 48
5.2 System Performance 49
5.3 User Evaluation 50
5.3.1 Level of Control 51
5.3.2 Graphical User Interface 51
5.3.3 Feedback 52
5.3.4 Overall System 52
6. Conclusion and Further Work 54
6.1 Challenges Overcome For Project Success 54
6.1.1 Developing using Cocoa for Mac OS X 54
6.1.2 Memory Management 55
6.1.3 Thread Safe 55

6.1.4 Streaming Two Cameras Simultaneously 55
6.1.5 Efficiency 55
6.2 The Final System 56
6.3 Future Work 57
6.3.1 Infrared Tracking 57
6.3.2 Multiple Point Tracking 57
7. References 58
8. Appendix A 61
8.1 2D Motion Tracking System 61
8.1.1 xyTracking 61
8.1.2 MyOpenGLView 63
8.1.3 FingerPosition 65
8.2 3D Motion Tracking System 66
8.2.1 xyTracking 66
8.2.2 ZTracking 69
8.2.3 MyOpenGLView 72
8.2.4 FingerPosition 73
8.3 Custom Filter 74
9. Appendix B 76
9.1 Questionnaire 76

1. Introduction

This project aims to research, develop and evaluate a gesture controlled, usable real-time two-dimensional (2D) and three-dimensional (3D) mouse by means of a low financial cost motion capture system.

1.1 Project Motivation

Motion capture or mocap is a technique for digitally recording movement. It is defined as *"The creation of a 3D representation of a live performance and translating it into usable mathematical terms."* in the book Understanding Motion Capture for Computer Animation and Video Games [16]. The data recorded can be used to animate a 3D model, control a robot or analyse the movements of the subject for scientific research. It is an ever-expanding field, which is becoming more accessible to researchers and developers with the rapid increase of available processing power.

There are many potential applications for mocap systems, but only two drew interest into the area. The first and the most common is for computer generated (CG) effects. This application is mainly used by the entertainment industry to create CG 3D characters or CG shots for use within video games, commercials or films. Within the games industry mocap is used to capture the motion of real people or animals with a database of the different motion movements built up e.g. walking, running, fighting. These are then strung together within the game to create the visual effect of the virtual character moving and interacting realistically. The film industry deploys mocap to capture data used for sophisticated CG shots and in recent years this has allowed films to include CG characters; one of the more successful being Gollum (see figure 1.1) from The Lord of the Rings.

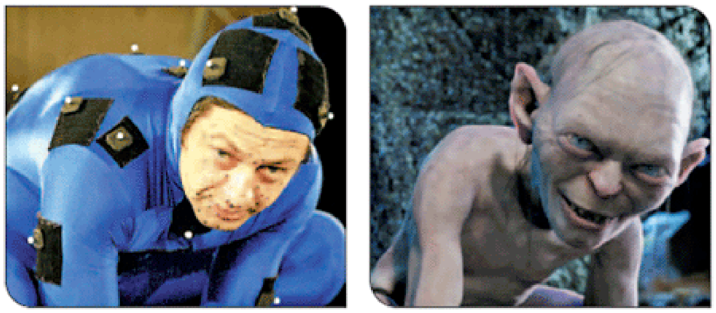


Figure 1.1 Motion Capture being used on The Lord of the Rings

The announced film production of Tintin has recently made news due to its groundbreaking technology, which allows the director to look at a monitor showing a real time fully rendered 3D virtual set, with the actors animated as their characters.

The second, and less common until recently, is advanced user interfaces. The keyboard, mouse and games controller have dominated the area of Human Computer Interaction (HCI) since the late 1970s, but recently there has been movement away from these towards more innovative and natural methods of interaction. Mocap systems are increasingly being used in vision based interfaces such as gesture recognition, for sign language applications. The Nintendo Wii, launched in 2007, gives users the ability to interact with games similar to that of a real world experience. It is supplied with a controller which uses motion capture to track its position in accordance with the television allowing users to use it in a variety of different ways e.g. golf club, tennis racket.

Using motion capture, a system will be developed capable of capturing the real-time motion of a user’s index finger (see figure 1.2). The motion data captured will be mapped to a control interface, which will control the movement of a 3D object, giving the user feedback of the finger being tracked. The system will be designed for a generic user therefore allowing more design focus on the interactive process of motion capture. It will conform to general usability standards allowing the majority of users to correctly operate the system with little training required. Using readily available hardware, with the majority of software being custom produced means the system with have a low financial cost.

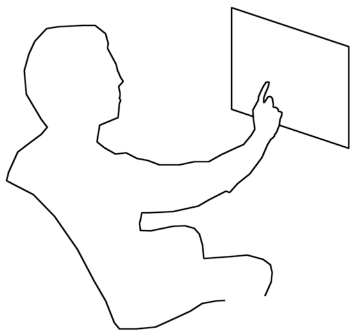


Figure 1.2 Example of user operating the system

1.2 Structure of Report

This report is structured as follows:

Chapter 2 – Requirements Analysis: Examines the requirements and specifications of the project, background research and project management.

Chapter 3 – Design: Examines the conception and design of each module within the system.

Chapter 4 – Implementation: Examines the implementation and testing of the different systems.

Chapter 5 – Evaluation: Examines the outcome of the system against the requirements, system performance and user evaluation questionnaire

Chapter 6 – Conclusion: Examines the challenges faced, the final system and future work of the project.

Chapter 7 – References

Chapter 8 – Appendix A

Chapter 9 – Appendix B

1.3 Professional considerations

A number of issues need to be addressed in order to complete a professional and ethical project. The British Computer Society sets out a Code of Conduct [10] and Code of Practice [11] which both students and professionals must adhere to in order to be a member. The practical application behind this project means that issues involving public interest must be considered. Below are codes which require particular consideration:

"1. In your professional role you shall have regard for the public health, safety and environment." [10]
During the development of the system public health and safety will be considered and attention will be given to any situation which arises that could harm or endanger the public. The finished system will give warnings to the user about the possible dangers and hazards of using such a physical form of computer interaction. All possible design considerations will be taken into account to reduce public risk when the system is in use. The environment will also be a major consideration throughout the development and operation of the system. The considerations extend to the three key factors of recycling which are Reuse, Reduce, and Recycle [22]. Every effort will also be made to reduce the carbon footprint of the entire project by using modern, more energy efficient computers and programming applications which are as streamline as possible therefore reducing processing power which consume large amounts of electricity.

"3. You shall ensure that within your professional field/s you have knowledge and understanding of relevant legislation, regulations and standards, and that you comply with such requirements." [10]
Wherever necessary and possible relevant legislation, regulations and standards will be consulted. This will minimise infringement of such codes. In particular the Disability Discrimination Act 1995 [21] will be reviewed to prevent discrimination which may arise from the development of practical application which involves certain physical requirements.

"14. You shall seek to upgrade your professional knowledge and skill, and shall maintain awareness of technological developments, procedures and standards which are relevant to your field, and encourage your subordinates to do likewise." [10]

Throughout the project the primary aim will be to improve knowledge and skill relating to the chosen topic. The recent release of Mac OS X Leopard will be reviewed and decisions will be made regarding further development of the system using Xcode 2.0 and Objective-C 2.0.

"15. You shall not claim any level of competence that you do not possess. You shall only offer to do work or provide a service that is within your professional competence." [10]
This project has been discussed with supervisors and an agreement made that the level work required is suitable. Regular meetings have also been arranged with supervisors to discuss the work completed and outstanding.

2. Requirements Analysis

This chapter examines the aims, requirements and specifications of the project. It includes the background research conducted during the early stages of the project comprising an overview of the motion capture technologies available, technical requirements analysis and chosen tracking algorithm. It also includes the project management scheduling and delivery dates.

2.1 Project Aim

This project aims to develop a gesture controlled, usable real-time two-dimensional (2D) and three-dimensional (3D) mouse by means of a low financial cost motion capture system. In order to evaluate the system it should be capable of moving and rotating a 3D object in space through specific hand gestures tracked by the motion capture device. It must also provide the user with an input mechanism, similar to a click on a tradition 2D mouse. It will use a two camera system to provide 3D motion data, which once analysed will provide parameters that will be mapped into a control interface. The control interface will then drive the movements of the 3D object through x,y,z co-ordinate parameters specifically using the z axis for input clicks. A graphical user interface (GUI) will be designed to provide the user with a simple easy to use front end, which requires minimal training.

2.1.1 Requirements

- RQ1 - Understand the process and techniques involved in capturing motion data
  - RQ1.1 - Research current methods of motion capture
  - RQ1.2 - Analyse motion capture method most appropriate for the proposed system
- RQ2 - Develop a 3D object for controlling
  - RQ2.1 - Design appropriate object which best displays control effect
  - RQ2.2 - Design object to be controlled from control interface
  - RQ2.3 - Design object to be updated within real-time
- RQ3 - Develop a real-time 2D and 3D motion tracking system
  - RQ3.1 - Capture video stream
  - RQ3.2 - Convert video stream into frame images capable of being analysed
  - RQ3.3 - Filter out unwanted environment
  - RQ3.4 - Track user’s index finger with tracking algorithm
  - RQ3.5 - Map tracking result to control interface
- RQ4 - Develop a control interface to drive the 3D object from captured motion data
  - RQ4.1 - Receive tracking data from RQ3
  - RQ4.2 - Analyse data to determine amount to update object from RQ2
  - RQ4.3 - Update object using results from RQ4.2

2.2 Background research

2.2.1 Motion Capture Technologies

There are two main types of motion capture systems in use within the motion capture industry, optical and non-optical [17]. Both systems have advantages and disadvantages (see table 3.1) but the freedom of movement combined with relatively high levels accuracy usually makes the optical approach the system of choice. Systems can also be separated into real-time (no post processing required) and non-real time capture (post processing required to clean up data). Within recent years interest and development of motion capture has grown exponentially. Several factors can be attributed to this growth but it is mainly due to the increase of available processing power. This increase has lowered the costs of capturing and video processing, meaning that is it now used within many different applications once considered out of reach.

2.2.2 Optical Systems

Optical systems rely on directionally reflective, light emitting or coloured balls referred to as markers, which are attached to the performer body. The system requires several cameras (usually between 3 and 16), which are used to track and triangulate the 3D position of the markers within a specific area, usually called a motion capture stage [25]. With an increase of cameras many markers can be tracked at once producing realistic capture data. The different optical systems involve the types of markers used. Passive markers are either balls coated with a reflective material that reflects light generated near the camera (see figure 2.1) or balled coated with vibrant colour. Active markers are one or multiple LEDs which emit their own light rather than reflecting an externally generated light source. Other optical system markers include time modulated active and semi-passive imperceptible markers [20].

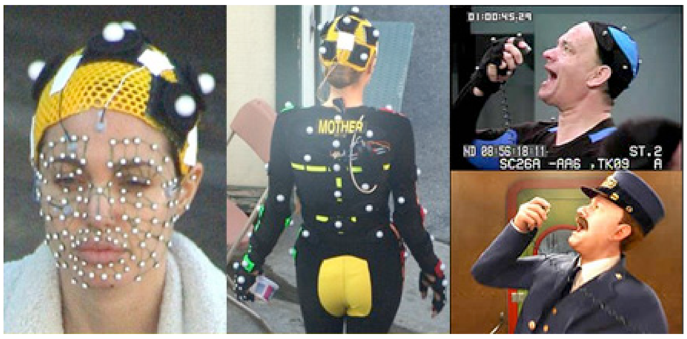


Figure 2.1 Angelina Jolie using optical motion capture in Beowulf and Tom Hanks using it in Polar Express [15] [9]

2.2.3 Non-optical Systems

Non-optical systems involve all other non marker related capture devices. Mechanical motion capture records movements directly via an exo-skeletal type suit attached to the performer (see figure 2.2). As the user moves so does the suit, feeding information into sensors located at each joint. This information is then fed into a computer wirelessly for real time processing. Mechanical motion capture is cheaper than optical systems but the suits tend to be restrictive and only major movements can be recorded. Magnetic systems use sensors placed on the body to measure the low frequency magnetic field generated by a transmitter source. The sensors and source are connected to a control unit which calculates the location of the sensors within the field. Magnetic systems can use six or more sensors to track a user’s movements using inverse kinematics to solve the angles of various body parts. Many problems are associated which magnetic capture mainly inaccuracies due to magnetic interference. Also the data has to be transmitted through cables which make sensors cumbersome to wear and difficult to use [24]. Gyro systems use tiny inertial gyroscopes that are attached to the performers body. These directly record the rotations of the performers body parts. The rotational data is transmitted by radio to a receiver unit where it is processed in real-time and mapped to a CG object. An advantage of the gyro system is that they are very easy to use and do not require a motion capture stage, meaning they can be used on location [2].

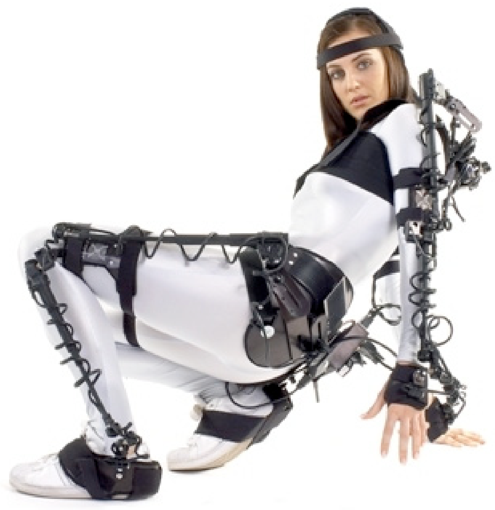


Figure 2.2 Mechnical exo-skeletal suit [1]

Motion Capture System	Advantages	Disadvantages
Optical	• Extremely accurate in most cases • Large number of markets can be used • Performs not constrained by cables • Large performance area • High frequency of capture	• Require extensive post processing • Expensive hardware • Marker occlusion • Capture must be carried out in controlled environment (away from reflective noise)
Non-optical	• Real time data output can provide immediate feedback • Position and orientation data are available without post-processing • Less expensive than optical • Sensors are never occluded • Multiple performance capture	• Tracker's sensitivity to metal can result in irregular output • Performers constrained by cables • Low sampling rate • Smaller capture area compared to optical • Difficult to change maker configurations

Table 2.1 Motion Capture System Advantages/Disadvantages [18] [19]

2.2.4 Evaluation of Motion Capture Technologies

In order for the motion capture system being developed for this project to be low cost, it must be produced from readily available hardware, which can easily be integrated into custom produced software. Non-optical exo-skeletal systems are not suitable for this project because of their complexity and high equipment costs. An optical system is far more suited because it can be produced using standard camera equipment, with the tracking and post processing being done with custom produced software. A passive coloured based marker will be used to track the user's index finger within a small capture stage as it is financially cheaper than using a reflective based marker.

2.2.5 Tracking

There are many sophisticated algorithms developed which are capable of detecting and tracking a number of different things. One of the most famous tracking algorithms developed is the Kalman filter [23]. This algorithm will provide accurate continuously-updating information about the position and velocity of an object given only a limited number of observations, some of which include errors. It is a highly sophisticated algorithm utilised within radar and computer vision systems. A more recent detection algorithm was developed by John F. Canny in 1986 [12] which when applied to a colour image will return an image with the edges of objects within highlighted. These types of algorithms were developed to extract information and produce results from often complex and erroneous data. The tracking method used within the system will remove most of the complex data using chroma key colour separation. A specific coloured marker will be attached to the user's index finger and then using a number of filters, all other colours will be removed. Removing all but one colour reduces the amount of the complex data before a tracking algorithm is being applied. This allows for a simpler, yet still demanding, tracking algorithm to be used. The bases of this algorithm will be the centre of mass equation defined in equation 2.1.

The centre of mass R of a system of particles is defined as the average of their positions r_i, weighted by their masses m_i:

$$\mathbf{R} = \frac{\sum m_i \mathbf{r}_i}{\sum m_i}$$

Euqation 2.1 Centre of Mass Equation

In order to track 3D motion, two cameras will be used which when processed simultaneously will provide the location of the user's finger within the motion capture stage. One camera will be used to track x and y-axis and the other will be used to track the z-axis.

2.2.6 3D Object

Within the system a 3D object will be created which will updated and rendered within real-time. The industry standard application programming interface (API) for creating real-time 3D graphics is the cross platform Open Graphics Library (OpenGL). This will be used to create a 3D object which will be controlled using the motion capture data. The OpenGL API consists of approximately 250 function calls used to draw complex 3D objects from simple primitives. It allows 3D objects to be directly rendered from the graphics card, which enables them to be rendered within real time. Figure 2.3 shows the overall system architecture developed using the background research

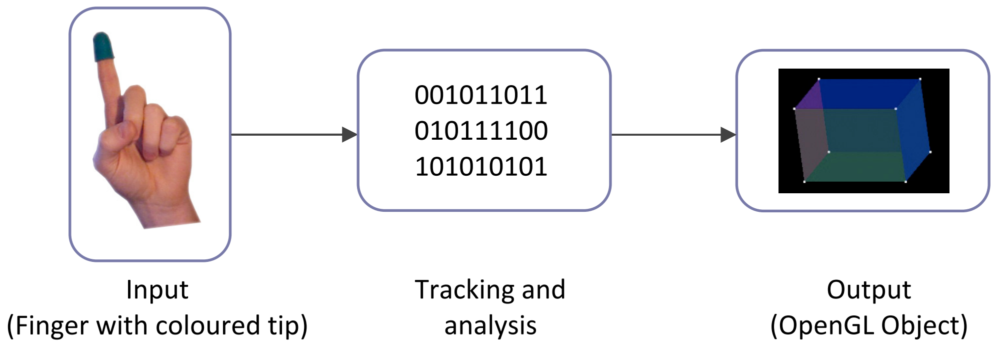


Figure 2.3 High Level System Architecture

2.4 Specification

The following specifications were developed using the requirements and background research outlined in sections 3.1.1 and 3.2 respectively.

SP1 - Develop OpenGL object

SP2 - Control object from SP1 using control interface

SP3 - Specify type of control from SP2 using tracking system

SP4 - Render object from SP1 in real-time

SP5 - Capture video streams from two cameras simultaneously in real-time

SP6 - Convert video streams from SP5 into frame images

SP7 - Remove all but specified colour from images from SP6

SP8 - Apply tracking algorithm to filtered images from SP7

SP9 - Map tracking algorithm result from SP8 to control interface

SP10 - Analyse tracking algorithm result from SP9 to determine amount to update object from SP1

SP11 - Update object from SP1 using result from SP10 in real-time

2.5 Project Management

2.5.1 Project Implementation Flow Diagram

The final system is divided into a number of sub systems which can be developed independently. See figure 2.4.

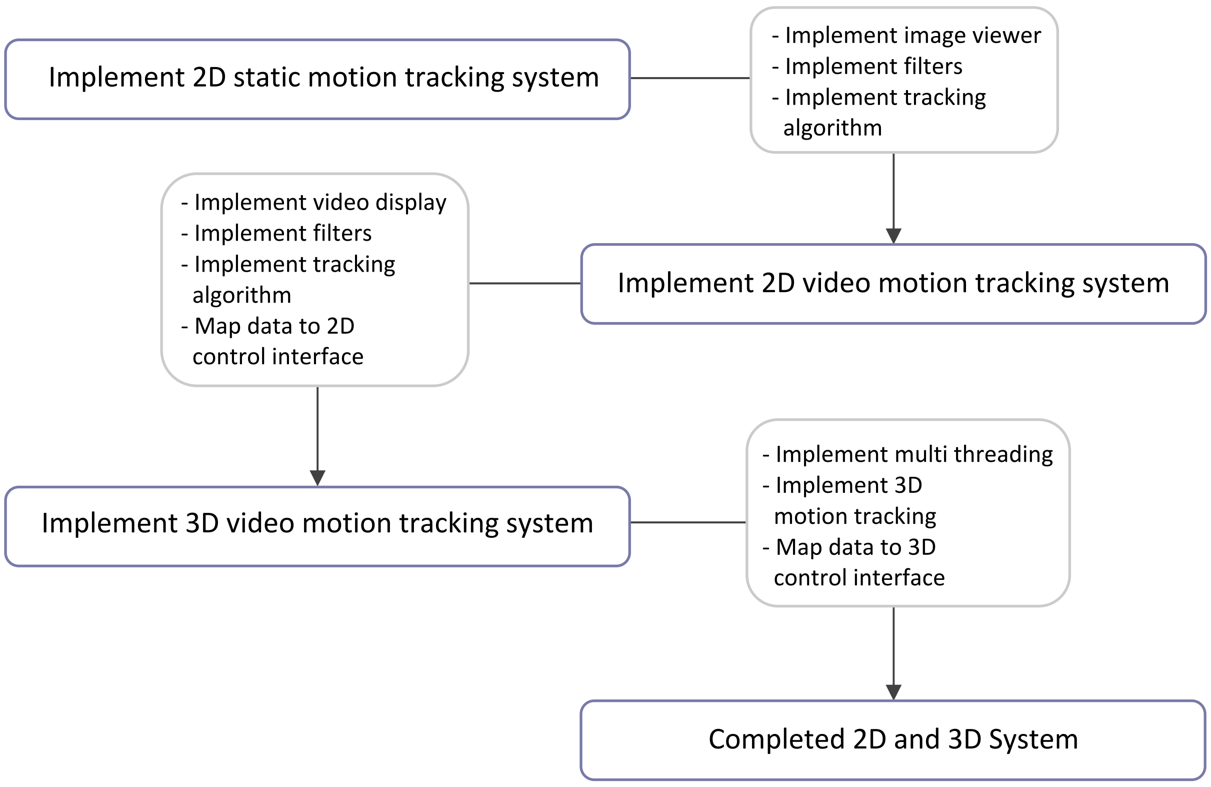

Implement 2D static motion tracking system
- Implement image viewer
- Implement filters
- Implement tracking algorithm
- Implement video display
- Implement filters
- Implement tracking algorithm
- Map data to 2D control interface
Implement 2D video motion tracking system
- Implement multi threading
- Implement 3D motion tracking
- Map data to 3D control interface
Implement 3D video motion tracking system
Completed 2D and 3D System

Figure 2.4 Project Flow Diagram illustrating the development flow of the system

2.5.2 Project Deliverables

Description	Date	Completed
OpenGL Object	09/11/2007	✓
2D Static Motion Tracking System	07/11/2007	✓
Image Viewer	16/11/2007	✓
Custom Filter	31/11/2007	✓
Tracking Algorithm	07/11/2007	✓
2D Video Motion Tracking System	18/01/2008	✓
Video Display	11/01/2008	✓
3D Video Motion Tracking System	01/02/2008	✓
Graphical User Interface	08/02/2008	✓
Final Testing	15/02/2008	✓

2.5.3 Scheduling

The project has been given an overall time scale of 24 weeks. The 24 weeks were broken down into approximately 5 week for research, 11 weeks for implementation and testing and 8 weeks for the project write up and presentation. Section 2.5.4 displays a Gantt chart which outlines the start and finish time of each aspect of the project. It also includes dependences allowing for the continuation of the project should one aspect fail to meet its specified deadline.

The following milestone dates were set:

- Project proposal: 18/10/07
- Begin development: 05/11/07
- Delivery of interim report: 06/12/07
- Completed system: 15/02/08
- Final report completed: 24/04/08
- Presentation: 28/04/08 – 09/05/08

2.5.4 Gantt Chart

3. Design

Chapter 2 established that the camera based optical motion capture technology is best for this project. A passive colour marker will be used to track the user's index finger using two cameras for 3D motion tracking. This chapter examines the conception and design of each module within the 2D and 3D systems. It examines the main components included within the systems with main focuses on the input, tracking and analysis and GUI modules.

3.1 High Level System Overview

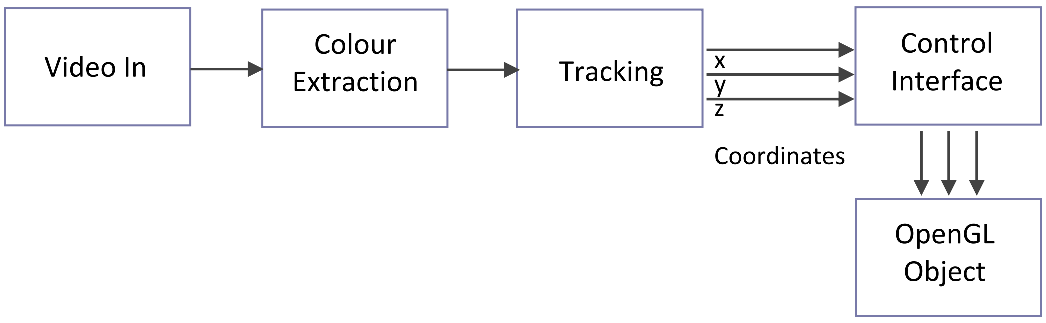


Figure 3.1 – High Level System Overview

Figure 3.1 shows the high level system overview, which consists of three main components: Input, Tracking and analysis and Output. The input component captures the movement of the user's index finger. This information is then passed to the tracking and analysis component which applies the filter and tracking algorithm. The result from the tracking and analysis component is then used to drive the output component; an OpenGL 3D model.

3.2 Input

To produce a low cost motion capture system, readily available equipment is being used throughout this project. An available MacBook computer will be used to implement the project therefore reducing costs. 3D movements is tracked using the x,y and z-axis, see figure 3.2.

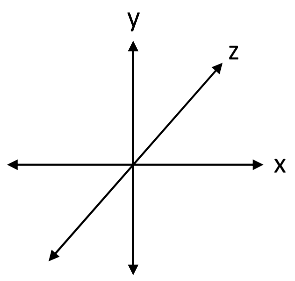


Figure 3.2 3D x,y,z-axis

To capture 3D movement of a user's index finger two standard webcams are being used. The first webcam will be used to track the x and y axis of the user's index finger and the second will deal with the z axis. This will provide the required 3D motion capture data used throughout the systems. A simple set up for the two cameras is to position them at right angles from each other therefore reducing the amount of computational overhead, which would result from correcting erroneous data generated from inaccurate tracking. Figure 3.3 illiterates the positioning of two cameras within a 3D environment.

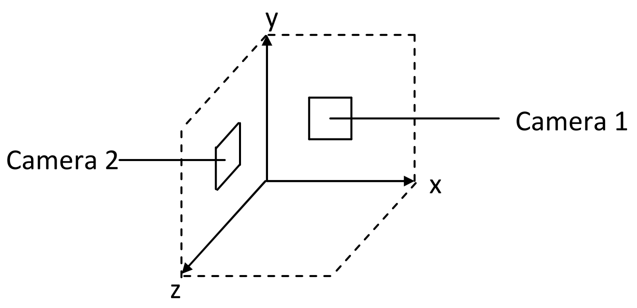


Figure 3.3 Camera and axis locations

Camera 1 will be a high-resolution isight camera already built into MacBook computers and camera 2 will be an inexpensive off the shelf webcam. The input component will have direct, simultaneous access to both decompressed real time streams from the cameras. Once the input module receives a frame from a camera it will resize it, therefore reducing processing required for larger images, and convert it into an appropriate image format. This will then be fed into the tracking and analysis module at a rate of approximately 15 frames per second (fps). Multi-threading will be incorporated into the input module because of the high levels of data being processed by the central processing unit (CPU). Each camera stream will be processed via different threads therefore improving efficiency within the system. See figure 3.4.

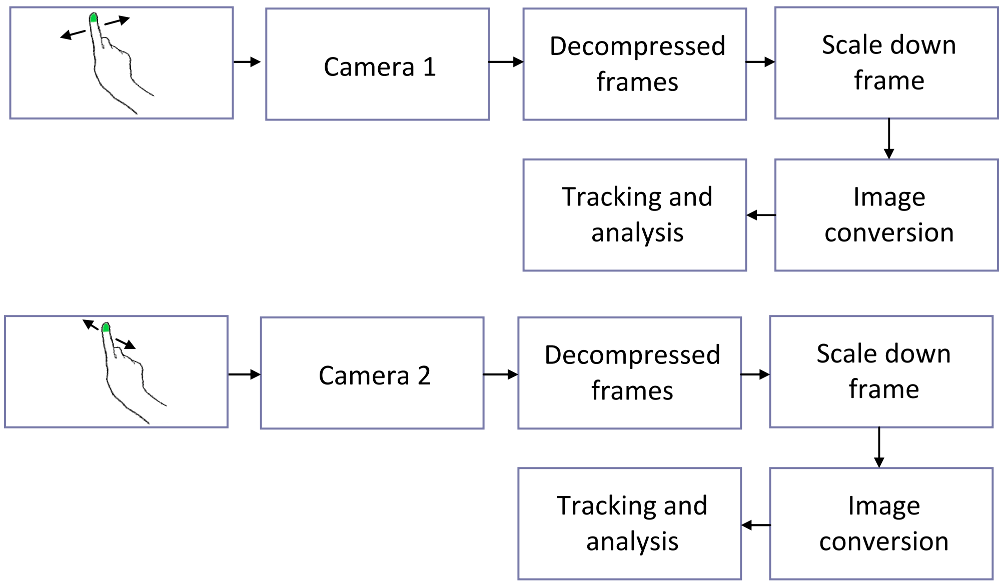


Figure 3.4 Input Module System Diagram

<u>3.3 Tracking and Analysis</u>

The tracking and analysis component will use the stream of images sent by the input component and apply a tracking algorithm. The result produced will be the position of the user's index finger. The tracking algorithm will be colour based so the user will wear a coloured thimble on their finger. The tracking algorithm will be applied to the stream of images sent from the input component and upon receiving an image it will first apply a filter, which will exact a specified colour. The filter will take a detection colour, match every pixel that is similar to that colour, and change the pixels not similar, to black. Pixels that are similar but not an exact match will be changed to the detection colour. This will produce an image with all but the coloured thimble removed. See figures 3.5 and 3.6.

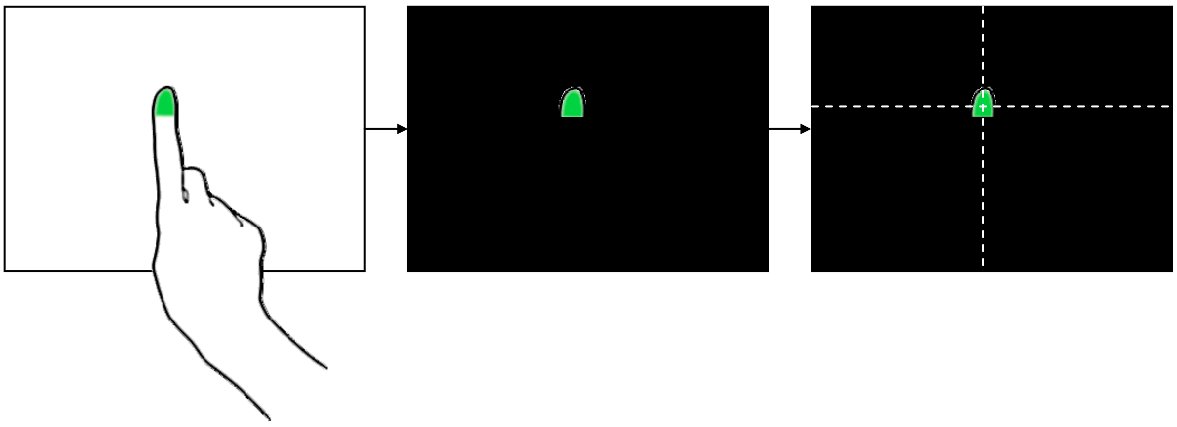


Figure 3.5 Example Colour Extraction and Centre of Mass result

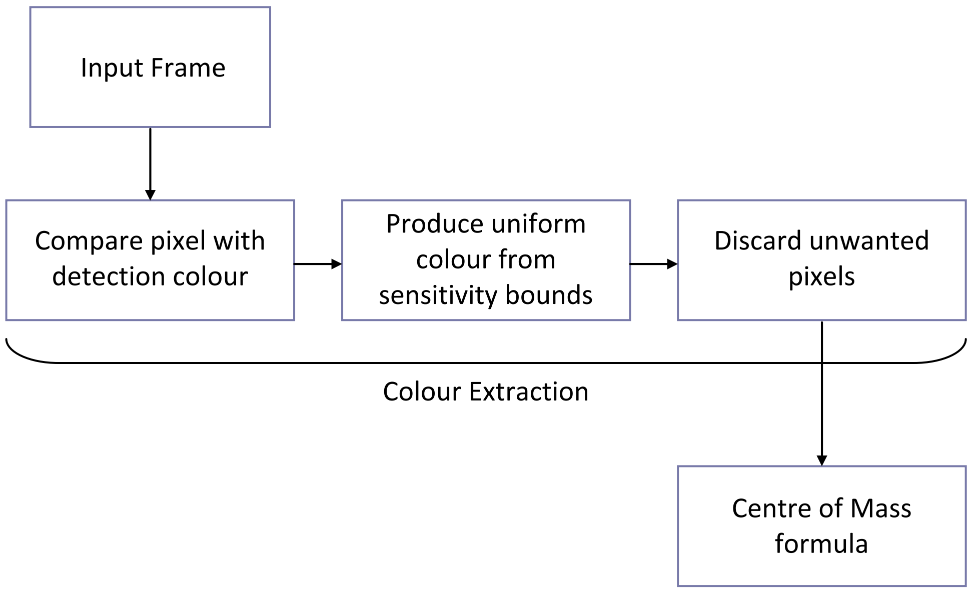

Input Frame
Compare pixel with detection colour
Produce uniform colour from sensitivity bounds
Discard unwanted pixels
Colour Extraction
Centre of Mass formula

Figure 3.6 Colour Extraction System Diagram

The next stage of the tracking will calculate the centre of the thimble. This will be achieved using the centre of mass formula outlined in chapter 2. The mass of a pixel will be represented by its colour therefore any coloured pixel will have a greater mass than a black pixel. This will result in coloured pixels having a greater pull on the centre of mass.

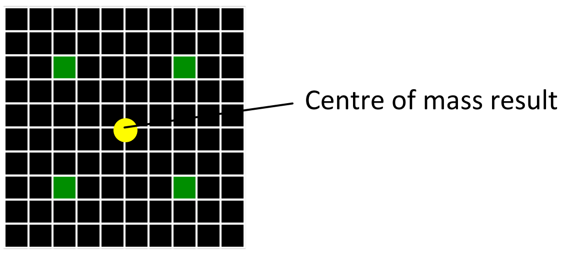


Figure 3.7 Centre of mass example result

Figure 3.7 shows an example result when the centre of mass formula is applied to a 12x12 grid of pixels with 4 green pixels each equidistant from one another. The centre of mass is in the middle of the 4 green pixels because each pixel is exerting an equal force on the centre of mass. If there were another green pixel on the right the resulting centre of mass would also shift to the right because there would be a greater pull in that direction. The coordinate system used for the centre of mass is one where the top left of the scene is represented by 0,0. As the finger moves towards the right, along the x-axis the centre of mass will increase. As it moves down, along the y-axis it will also increase. See figure 3.8.

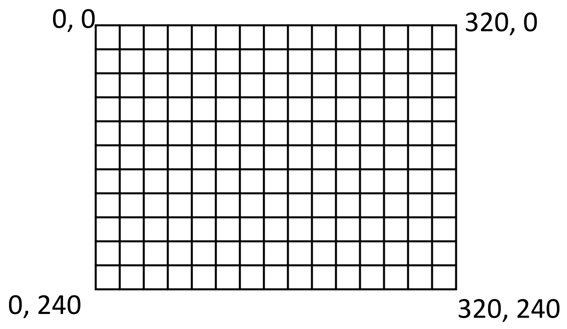


Figure 3.8 Centre of mass coordinate system

The result produced when this formula is applied to each filtered image is the centre of the thimble and because there are approximately 15 images being processed per second, per camera, this will result in a real time position of the thimble which can then be used to drive the output component. Once the thimble has been successfully tracked there will be an analysis on the results produced. This mainly involves calculating the distance the thimble has moved from one image to the next. This is simply achieved by subtracting the previous known location from the current location (see figure 3.9 and 3.10). The camera 1 result from this calculation can be used to move the 3D object. The camera 2 result can be used to check if the user is clicking (along the z axis). To simulate a click the user will quickly move their finger towards camera 1 and back to the original position. This movement will be tracked using camera 2 along the z axis. Therefore if the distance moved between frames on camera 2 is large, it can be assumed that the user has simulated a click.

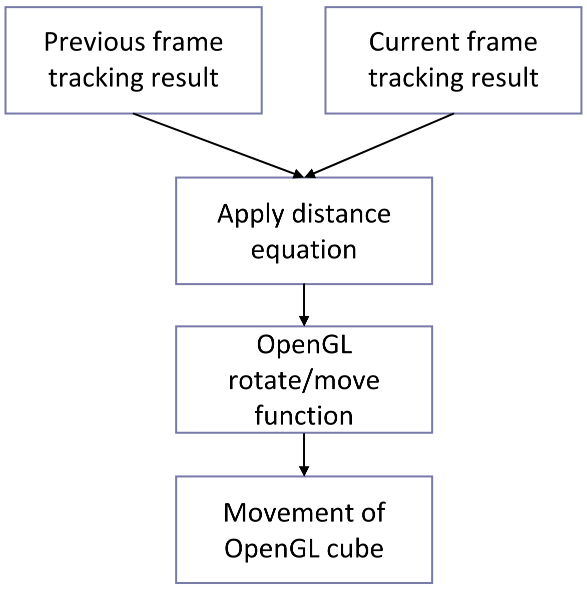

Previous frame tracking result
Current frame tracking result
Apply distance equation
OpenGL rotate/move function
Movement of OpenGL cube

Figure 3.9 Distance Equation System Diagram

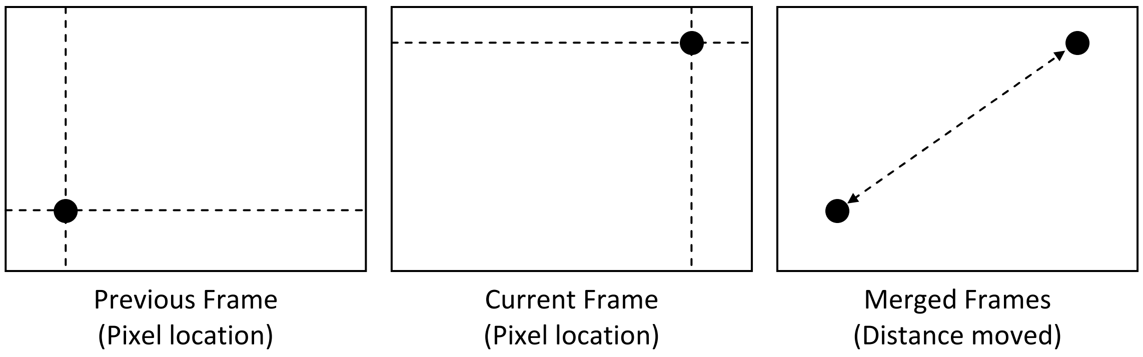

Previous Frame (Pixel location)
Current Frame (Pixel location)
Merged Frames (Distance moved)

Figure 3.10 Example of Distance Equation

3.4 Output

The output component will consist of the real time rendered 3D object which will be rotated or moved in accordance with the tracking results from camera 1, along the x and y axis. The object will be produced using OpenGL, outlined in chapter 2, and displayed on the system GUI. There are many shapes that could be used but because of its simplicity and defined edges a cube will best show the movement effect. The cube will be rotated right or left along its x axis if the user moves their finger to the right or left. It will be rotated up and down along its y axis if the user moves their finger up or down. If the user simulates a click the model will move instead of rotating (still in accordance with the user's finger gestures). If the user clicks again the model will revert back to rotating. See figure 3.11

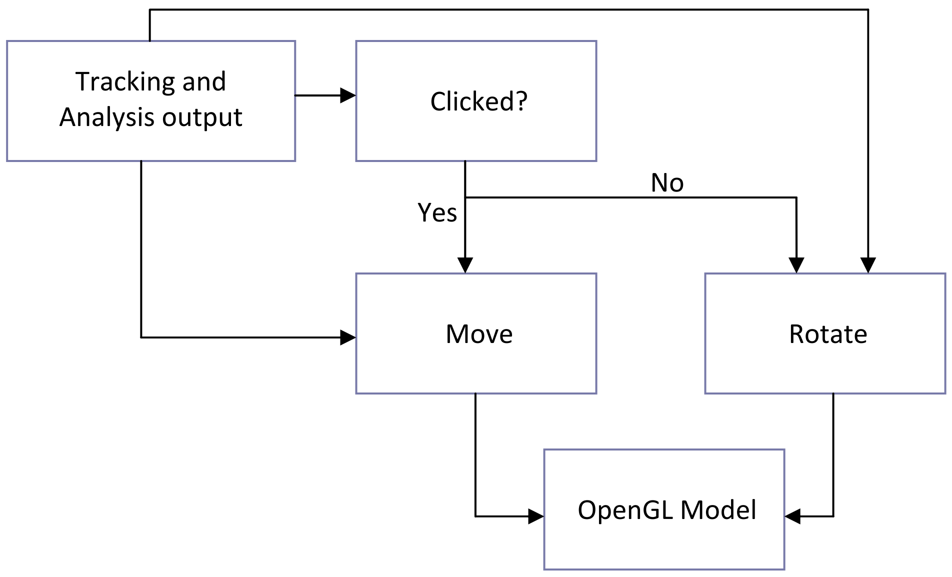


Figure 3.11 Output Data Flow diagram

4. Implementation

This chapter covers the implementation of the various components of the system. It discusses the problems, which arose during the development, and the solutions that were implemented. It also presents the graphical user interface (GUI), and incorporated features.

4.1 Platform

The software written for this project will be for Mac OS X using Apple Computer's XCode development environment. Applications produced will be based upon the Cocoa application framework using Objective-C programming language.

Mac OS X is a powerful operating system developed for Apple Macintosh computers [5]. It will be used to develop the software required for this project due to its image, video, performance and multiprocessor capabilities. Mac OS X is also a fully-conformant UNIX operating system popular for open source development. It provides a high level programming environment Cocoa, which enable developers to produce complex and demanding programs quickly and effectively. To produce programs Mac OS X includes a number of developer applications, primarily Xcode. Xcode is a development environment, which provides developers with a graphical workbench integrated with a professional text editor, debugger and powerful complier [8]. It tracks all the resources which go into an application such as code, image, sound etc..

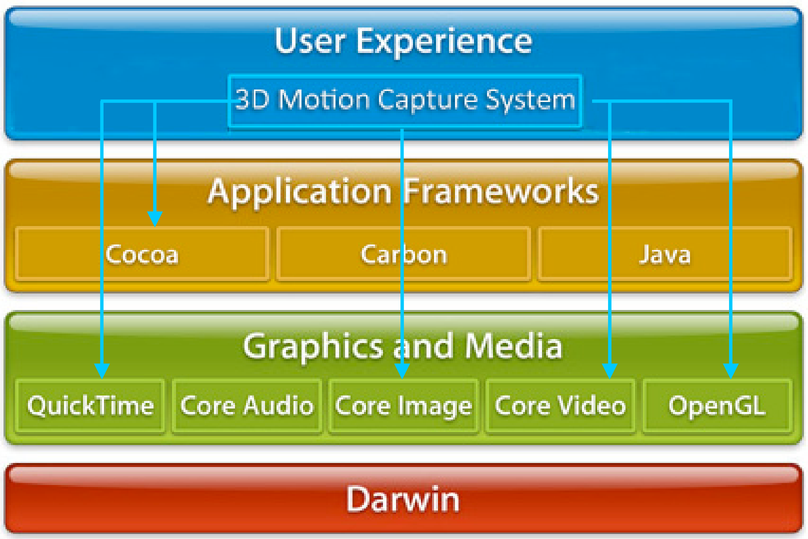
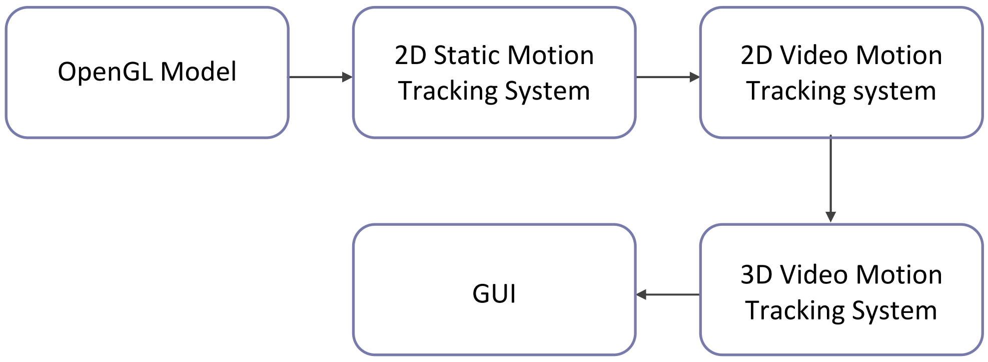


Figure 4.1 Mac OS X System Architecture [6] in relation to this project (User Experience)

The motion capture application will be coded using Objective-C programming language. Influenced by C and Smalltalk, Objective-C is an object or orientated language similar to Java. It is an easy language to learn if the user previously familiar with object-oriented languages such as Java or C++ [14]. To manipulate the video stream, both Core Video and Core Image libraries will be used. The Core Image and Core Video APIs provide access to built-in image filters for video and still image streams [3] (see figure 4.1). They provide a number of pre-coded filters and support for creating custom filters using Core Image Kernel Language. These can be used to filter out the unwanted background environment.

4.2 Implementation Stages

The final 3D video motion tracking system was implemented in five sub stages each built independently. The main stages of development were the implementation of: Output test bench, 2D static motion tracking system, 2D video (dynamic) motion tracking system, 3D video (dynamic) motion tracking system and GUI. See figure 4.2.


OpenGL Model
2D Static Motion Tracking System
2D Video Motion Tracking system
3D Video Motion Tracking System
GUI
Figure 4.2 Implementation Flow

4.3 Output Test-Bench

A test framework was developed which used OpenGL to render an object onto a display area (see figure 4.3). The object was created by defining four vertices, which together make a square polygon. This object could be transformed by moving the vertices within the display area along the x, y or z-axis. The use of 3 axis allowed 3D shapes to be created which could be transformed within the 3D space. This formed an output test-bench for the front-end system. See figure 4.4.

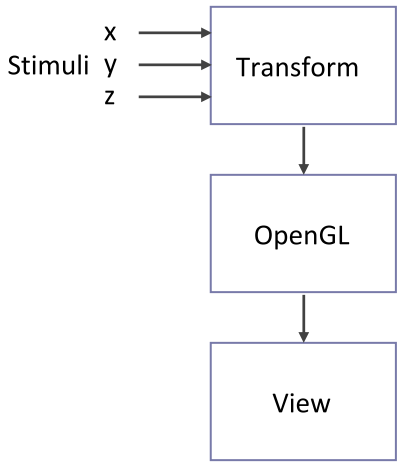
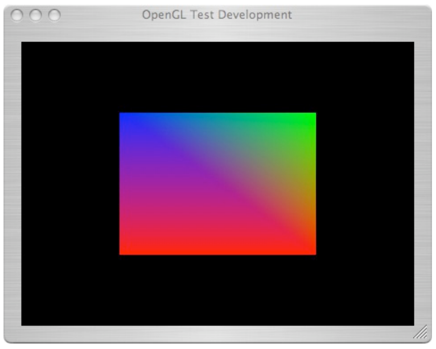


Figure 4.3 Test-Bench System



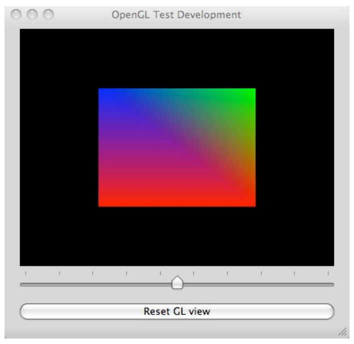
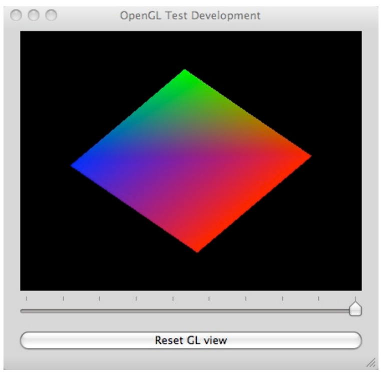


Figure 4.4 OpenGL Test Development

A number of tests was performed using the test-bench. These involved continuously updating the position of the four vertices causing the object to rotate around the z-axis. The update method was also connected to a slider so that as the slider was moved the object rotated (see figures 4.5, 4.6 and 4.7). The final system used a similar update method but was connected to the control interface rather than a slider.



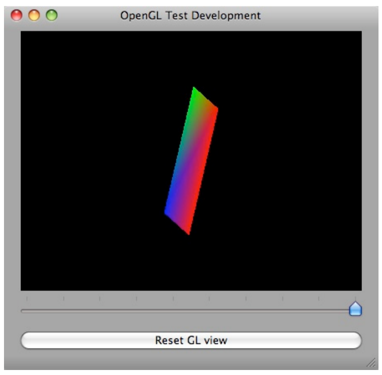


Figure 4.5 Test 1 Rotate around z –axis   Figure 4.6 Test 2 Rotate around x,y,z-axis

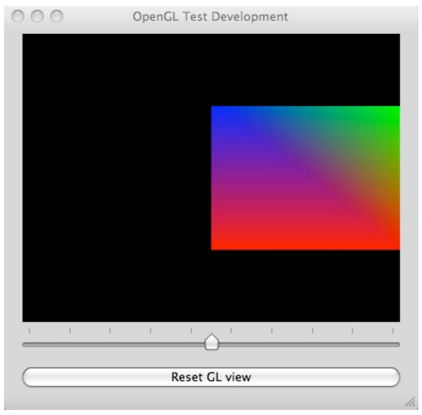


Figure 4.7 OpenGL Test 3: Move on x-axis

4.4 Static 2D Motion Tracking System

4.4.1 Image Viewer

An image viewer was also developed as part of the initial test bench. This read in an image (from a number of different formats), applied a filter, and then displayed the result within a display area, see figure 4.8. This was used to test various filters, which were later developed (see figures 4.9 and 4.10). Once the correct filter was developed it was modified so that it would also display the output result of the tracking algorithm.

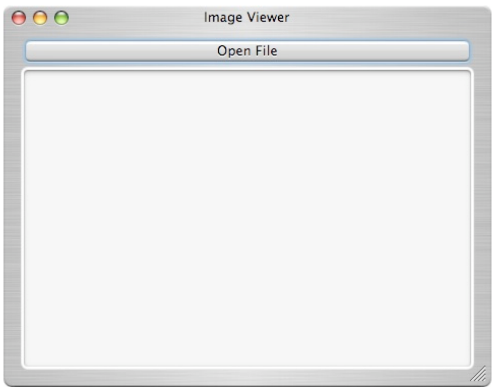


Figure 4.8 Image Viewer Test Bench

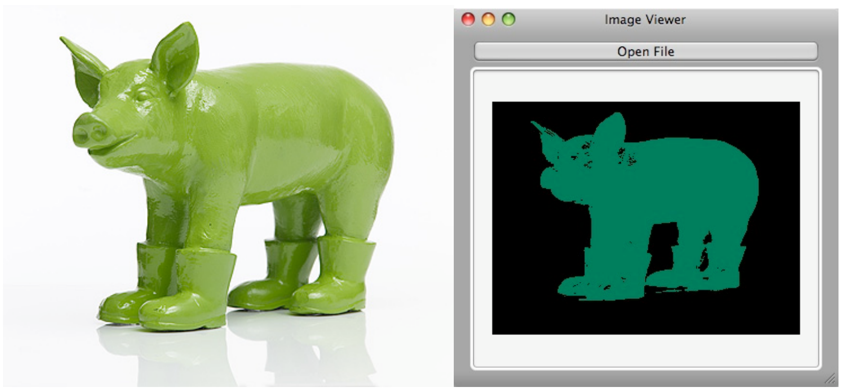


Figure 4.9 Example Image Before and After Filter

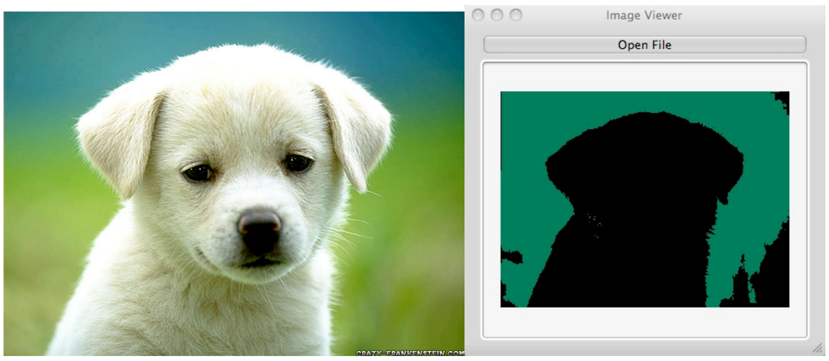


Figure 4.10 Example Image Before and After Filter

4.4.2 Custom Filter

In order to track a user's index finger the user has to wear a coloured thimble. The colour of the thimble is extracted from the image and all other colours set to black. This allows a tracking algorithm to be applied which will be less confused by other objects within the scene. The original design indicated that a combination of Core Image and Core Video filters would be able to produce the effect. Unfortunately after a number of tests using the built-in filters the results were not as expected and it was very difficult to achieve the desired effect. Within Cocoa, Core Image Kernel Language can be used to be to produce custom filters. This was therefore used to produce a custom filter capable of extracting the thimble colour. A benefit of using Core Image Kernel filters is that they are usually handled by the computer graphics hardware providing near real-time processing (see figure 4.11).

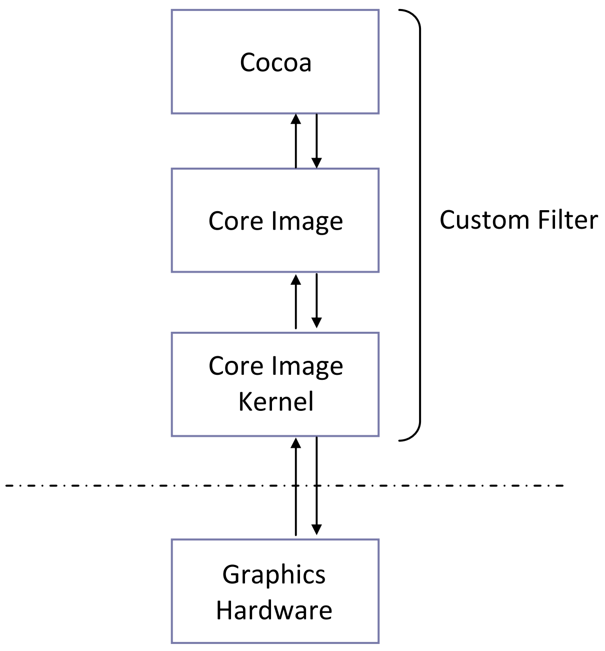


Figure 4.11 Custom Filter

This improved the efficiency of the system and allowed a higher frame rate. When run, a Core Image Kernel filter inspects every pixel within an image and modifies the pixel using a customisable piece of code. The filter used for this project firstly converts the pixel from an RGB (Red, Green, Blue) value to HSV (Hue, Saturation, Value), and then checks the Hue (perceived colour) and Saturation (intensity of colour) against the colour provided. The Compare function is used which returns either a black colour or the specified match colour depending if the augment specified is greater or less than 1. Within this filter the argument specified is a Vec4 component created from the Hue and Saturation results from the pixel. A Vec4 component is a representation of a 4D vector type. Figure 4.12 shows an RGBA (Red, Green, Blue, Alpha) Vec4 representation. This is the same as the one used within the custom filter except the Vec4 is made created from the Hue and Saturation result.

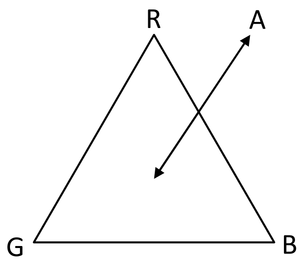


Figure 4.12 RGBA Vec4 representation

There is also a sensitivity variable, which is calculated as a fraction and subtracted from the Hue and Saturation result. This variable is used in order to allow a wider range of similar colours to be changed to the detection colour instead of black. The compare function sets the pixel to black if the argument is greater than 1, therefore if the sensitivity is set to a high value the fraction subtracted will be smaller and more pixels will be changed to black. Once all the pixels have been inspected, the resulting image is one where the coloured object has been highlighted by blacking out all other colours. See figure 4.13.

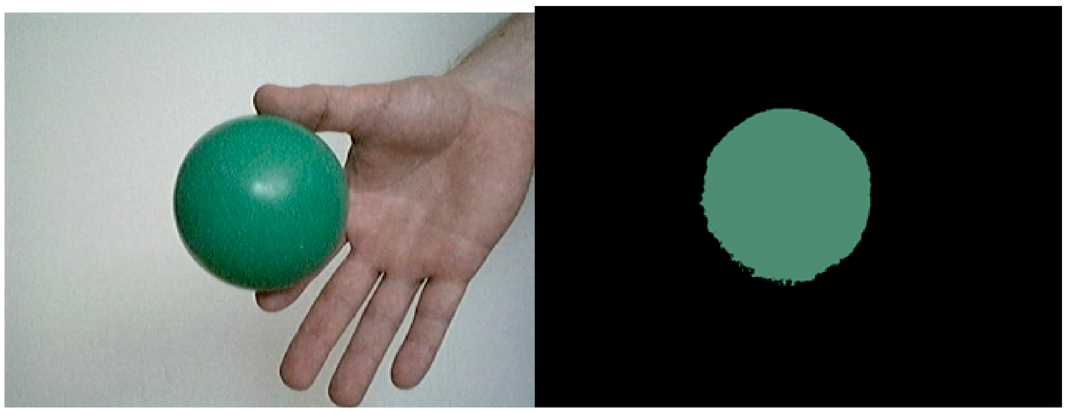
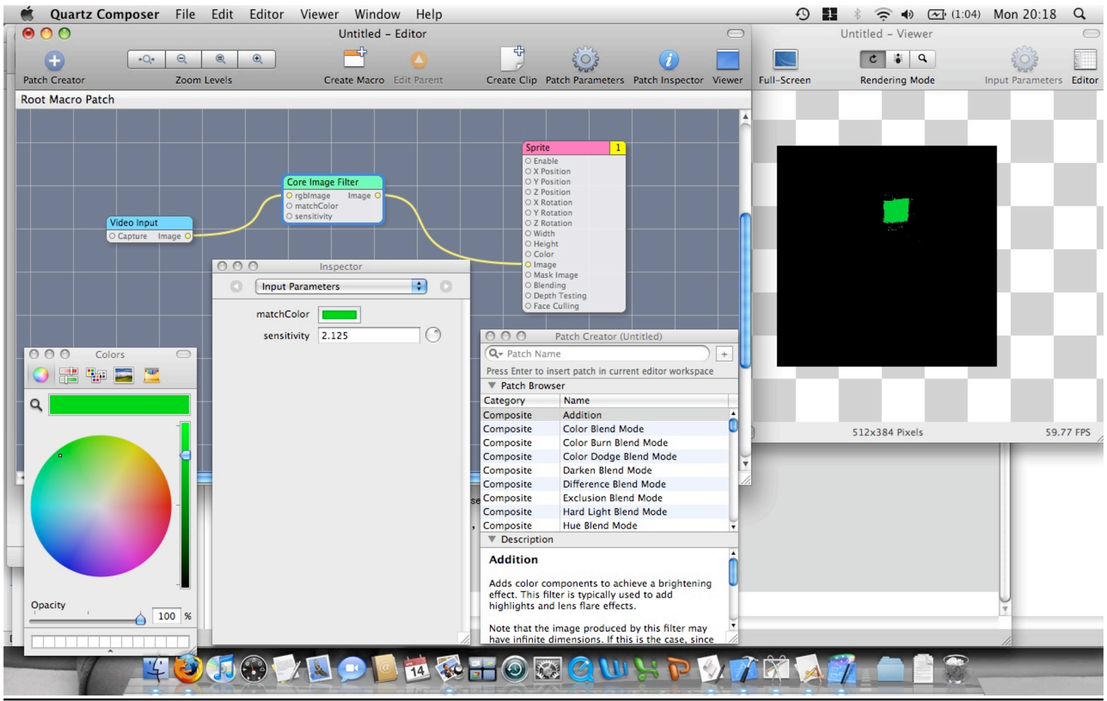


Figure 4.13 Left, image before filter. Right, image after filter

Quartz Composer was used as a test bench for the filter allowing it to be improved before it was packaged (see figure 4.14). This involved creating a Cocoa method that allowed the filter to take in a sensitivity value and output the filtered image. The whole filter was then packaged as releasable (meaning it can be used on any Mac) and then imported into the

MacBook library. Once imported a filter can be instantiated from any program just as any built-in filter would be.



matchColor 🟩
sensitivity 2.125

Category	Name
Composite	Addition
Composite	Color Blend Mode
Composite	Color Burn Blend Mode
Composite	Color Dodge Blend Mode
Composite	Darken Blend Mode
Composite	Difference Blend Mode
Composite	Exclusion Blend Mode
Composite	Hard Light Blend Mode
Composite	Hue Blend Mode

Addition

Adds color components to achieve a brightening effect. This filter is typically used to add highlights and lens flare effects.

Note that the image produced by this filter may have infinite dimensions. If this is the case, since

Figure 4.14 Quartz Composer Ouput

4.4.3 Tracking algorithm

The output of the filtering stage is in the form of a CIImage – a low level image format. In order for the tracking algorithm to determine the Centre of Mass, the image had to be converted into a NSBitmapImage. This allowed the use of the ColourAt method which returns the colour of a specific pixel. The bitmap data is then scanned, one pixel at a time, and the colour extracted. The colour information is then fed into the Centre of Mass equation. This method uses the CPU to extract the pixel colour, which can be inefficient compared to using the graphics hardware. Another idea for implementing the tracking algorithm was to use another custom filter. Unfortunately custom filters cannot accumulate knowledge from pixel to pixel and they can only output a Vec4 component. This therefore left no choice but to use the less efficient method.

The 2D static motion tracking system was developed purely as a test bench allowing for tests to be carried out on different components of the system before they were implemented within the 2D video motion tracking system. This meant that fewer errors would arise during the development of the more complicated 2D and 3D video motion tracking systems.

4.5 2D Video Motion Tracking System

Figure 4.15 shows the overall 2D motion tracking system development flow. Implementing the system in a number of sub stages allowed for the testing of each sub stage before continuation of the overall system development.

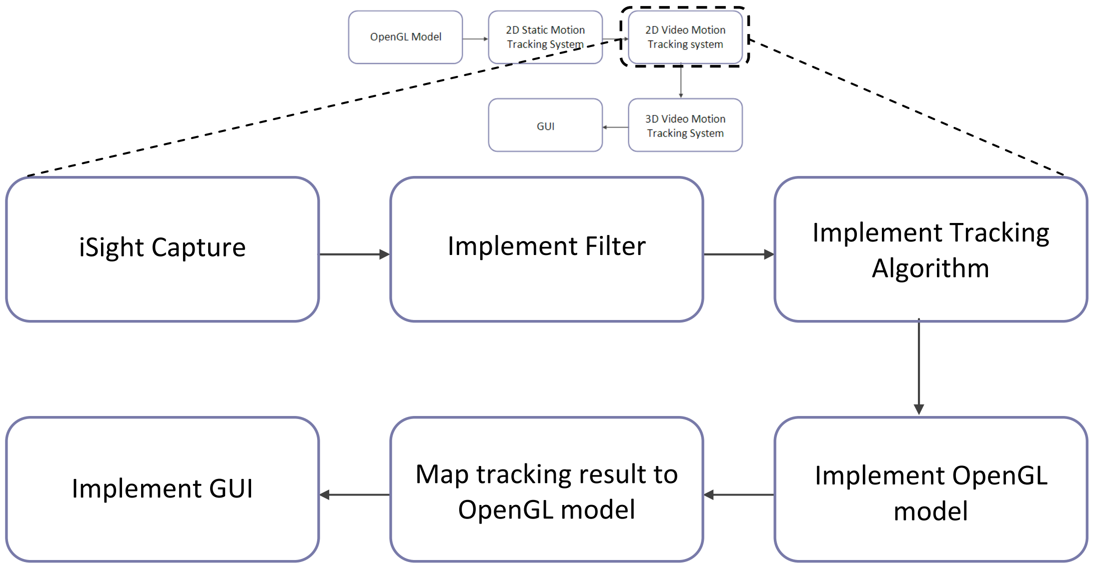

OpenGL Model
2D Static Motion Tracking System
2D Video Motion Tracking system
GUI
3D Video Motion Tracking System
iSight Capture
Implement Filter
Implement Tracking Algorithm
Implement GUI
Map tracking result to OpenGL model
Implement OpenGL model

Figure 4.15 2D Video Motion Tracking System Development Flow Diagram

4.5.1 iSight Capture

Initial development for the 2D video motion tracking system focused on accessing the built in iSight camera included within MacBook computers. The system originally began development using XCode 2.0 for Mac OS X 10.4 (Tigar). Using this XCode 2.0 environment development was concentrated on using Sequence Grabber components included within the QuickTime architecture. Sequencer Grabber components allow applications to obtain digitized data from external sources. As the iSight is connected to the USB High Speed bus it is treated as an external source, just as if an off the shelf camera had been plugged in. Experimental code was produced which could grab frames from the iSight camera and store them within a QuickTime movie. The code produced for this method of capture was unable to process frames real-time and it became apparent that another solution would need to be found. One method was to use a number of Carbon APIs, which could better handle the real-time capture from external sources. This however was difficult because Carbon is a foreign language compared with Cocoa. With the release of XCode 3.0 for Mac OS X 10.5 (Leopard) a number of new components was available for developers. This included QuickTime 7 architecture, with a brand new QTKit framework. QTKit makes it easier to access the underlying QuickTime primitives and directly use more than 2500 functions in the QucikTime procedural API. This in essence enables the real-time capture of the iSight camera using Cocoa methods included within the QTKit framework. The method within QTKit capable of capturing from input sources was the QTCaptureSession. This provides an interface that determines which input device is selected and where the output of the device should be routed. See Figure 4.16.

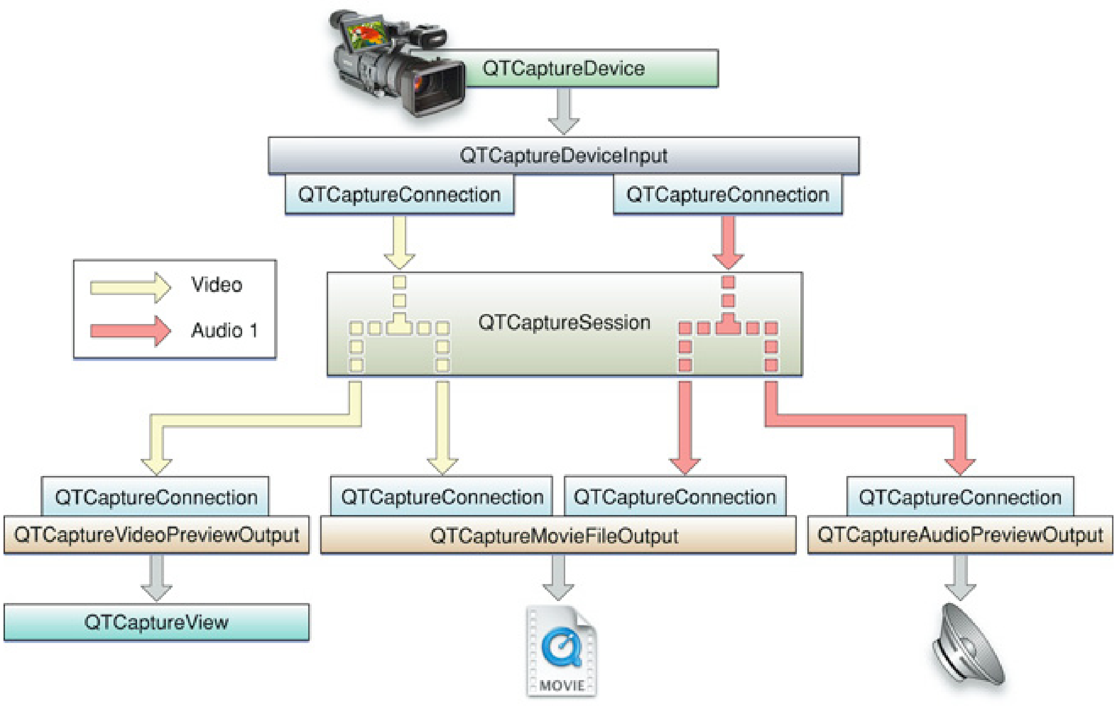


Figure 4.16 QTCaptureSession Visual Display [4]

As the only device connected to the Mac was the iSight camera, the only external input source connected could be the iSight. A connection was set up between the QTCaptureSession and the iSight camera. To output the frames captured, the QTCaptureDecompressedVideoOutput method was used. This method represents an output destination for the QTCaptureSesson and can be used to process decompressed frames for high quality processing. QTCaptureDecompressedVideoOutput functions by using a delegate method despatched as a new thread. This delegate method outputs each frame captured from the iSight as a CIImage. This stream of images can then be used anywhere within the system. A test application was produced using the stream of CIImages, converting them to

NSImages then displaying them on screen using a NSImageView. This showed that the capture method was working and that it was possible to move to the next stage of implementation (see figure 4.17).

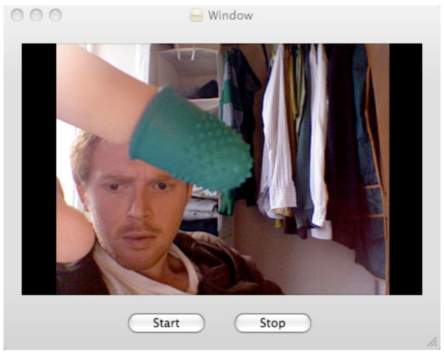


Figure 4.17 iSight Capture Test Application

4.5.2 Filter Implementation

The filter made within the 2D static motion tracking system could easily be imported into this system with the images being filtered in the same way as in the previous system. To apply the filter to the stream of images being captured from the iSight camera, a separate method is called from the QTCaptureDecompressedVideoOutput delegate which applies the filter and converts the image into a NSImage ready to be displayed using another NSImageView. See figure 4.18.

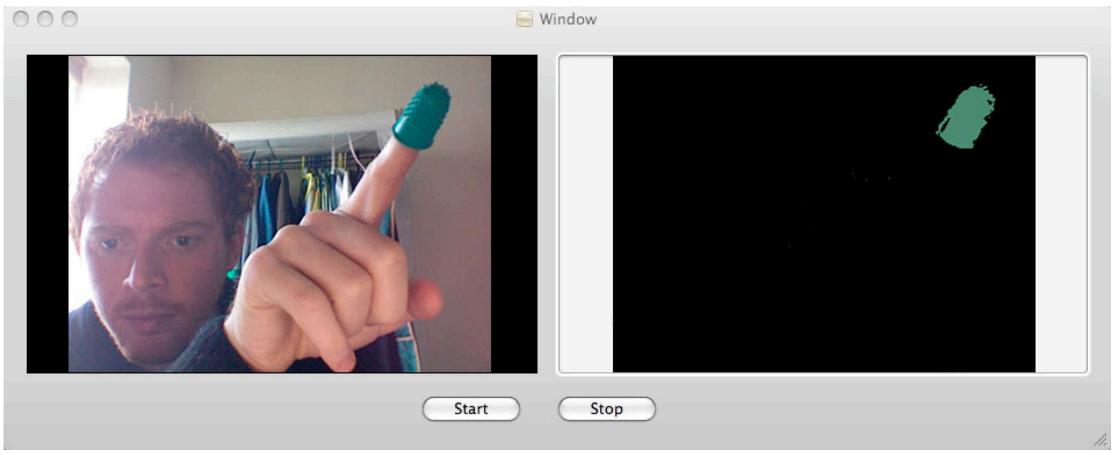


Figure 4.18 2D Video Motion Tracking System Filter Test GUI

A major problem, which arose after implementing the filter, was a Cocoa issue with memory management. Java uses automatic garage collection which cleans up the memory that has been assigned and is no longer being used. If the memory is no longer being used the garbage collection automatically releases it, making it available for other parts of the system to use. If large amounts of memory are assigned and not released, once they have been finished with, then all the memory available to computer fills up and the computer crashes. As this system began development using Mac OS X 10.4, there is no automatic garbage collection. The release of Mac OS X 10.5 included this function but could not be used because parts the system had already been coded using the previous OS version. Major issues arose when large amounts of data, being generated from the stream of images being captured from the iSight, were being stored within system memory. The memory being assigned for captured data was not being released and the system kept crashing. This was not an issue with previously developed modules because there were not the large amounts of data being generated. To overcome this problem Cocoa implements a manual allocation and release of memory, which must be coded into the program. Instances (e.g. CIImage) must be allocated a block of memory, which will be retained until the block is released using a release method. Careful attention needed to be taken not to try and access released memory because this caused an exception, which crashed the system. This problem was made more difficult because QTCaptureDecompressedVideoOutput delegate method was threaded therefore the system needed to be thread safe (meaning that multiple threads weren't accessing the same piece of data simultaneously). One thread could also not access a piece of memory already released by another thread without the memory first being retained using the retain method.

4.5.3 Tracking Algorithm Implementation

The tracking algorithm used was the same one developed for the 2D static motion tracking system. Once the image, being outputted from the QTCaptureDecompressedVideoOutput delegate method, had the filter applied to it, then the tracking algorithm is applied. The result of the tracking algorithm is the centre of mass of the coloured object within the image (user's index finger). The stream of images being generated meant that as each image was examined the centre of mass changed therefore giving a real-time location of the user's index finger. Using a camera located in front of the user, meant that the images generated were a mirror of the real scene. The coordinate system used for the tracking algorithm used the top left corner of the scene as 0, 0. With the mirrored image the coordinate system became one where the top right corner of the scene was 0,0. See figure 4.19.

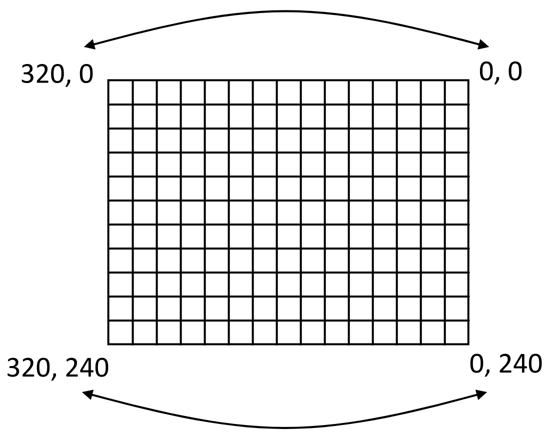


Figure 4.19 Mirrored Image Coordinate System

To reverse this, the result of the tracking algorithm needed to be corrected. This was done using the following formula outlined in equation 4.1.

$$COG\_X = (COG\_X * -1) + 320$$

Equation 4.1 Tracking Algorithm Correction

COG_X = Centre of mass result

One issue that arose from implementing the tracking algorithm used from the 2D static motion tracking system was speed. The number of image format conversions necessary for the use of different methods (CIImage, NSImage, NSBitmapImage) was beginning to slow the system down. The NSBitmapImage method ColorAt, for exacting pixel colour information, was particularly slow. This was changed so that the raw pixel integer data was extracted instead of its colour. This still gave the same result as the previous method but was more efficient. In order to the move the OpenGL model, the distance the user's index finger moved from frame to frame needed to be calculated. To achieve this, the finger location from the previous frame and the current frame were stored. Equation 4.2 is the distance equation being used to calculate the distance the finger moved between frames. Figure 4.20 shows an example of the distance equation being used within the system.

$$d(x), d(y) = \sqrt{[(x2 - x1)^2 + (y2 - y1)^2]}$$

Equation 4.2 Distance Equation

d = distance

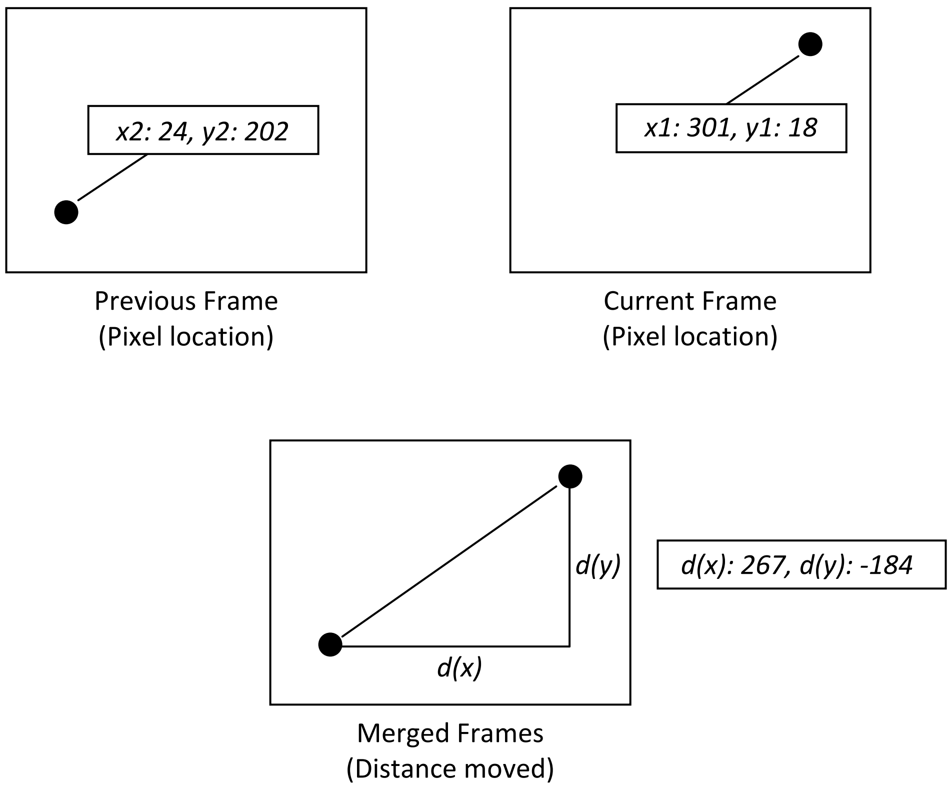


Figure 4.20 Distance Equation Example

When the capture system first loads there is no previous frame, therefore the distance equation cannot be used. A Boolean flag was used to indicate whether it was the first tracking instance and if true, the distance equation is not applied and only the location of user's finger is stored. This can then be used along with the next captured frame to find the distance.

<u>4.5.4 OpenGL Model Implementation</u>

The OpenGL model, controlled using the input module, was designed as a 3D cube. To implement this, each vertex was defined using the OpenGL vertex function call. Four vertices were required to create each side of the cube along with another four for each point, displayed at the corners of the cube. See figure 4.21.

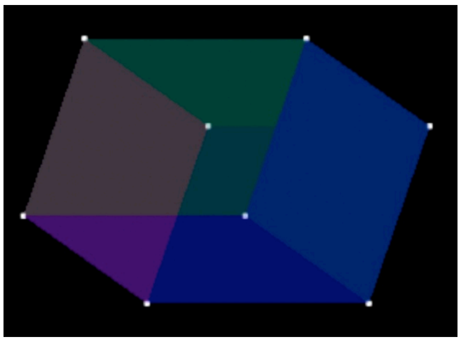


Figure 4.21 OpenGL Cube

The OpenGL model was designed in a separate class, called each time the cube needed to be re-drawn. Within the OpenGL class is a method used to rotate the cube. This method calls the rotate function available from the OpenGL API and, using the result from the distance equation, rotates the cube around either the x-axis or y-axis. An issue that arose during the implementation of the OpenGL model was making the class thread safe. The QTCaptureDecompressedVideoOutput delegate method may not be called by the main thread but from a newly dispatched thread. This meant that it was possible for 2 or more threads to try and access the same draw method within the OpenGL class. If two or more threads try and access the same shared data the system crashes. The lockFocus and unlockFocus methods available within Cocoa were used to make the draw method thread safe. The lockFocus method locks off the draw method from all other threads until it is unlocked by the unlockFocus method. If a thread tries to access the locked draw method, it will just wait until it becomes unlocked, therefore preventing the system from crashing.

4.5.5 GUI

A number of GUIs where developed during the development and testing phase. During early development the GUI was used to give test feedback about different aspects of the system (e.g. finger location, distance moved). It also displayed different stages of image capture including filtered images. Figure 4.22 and 4.23 are examples of the different stages of the GUI. Figure 4.22 shows the GUI during the iSight capture stage with the filter being applied to the stream of images being received. Figure 4.23 shows the GUI during the implementation of the tracking algorithm.

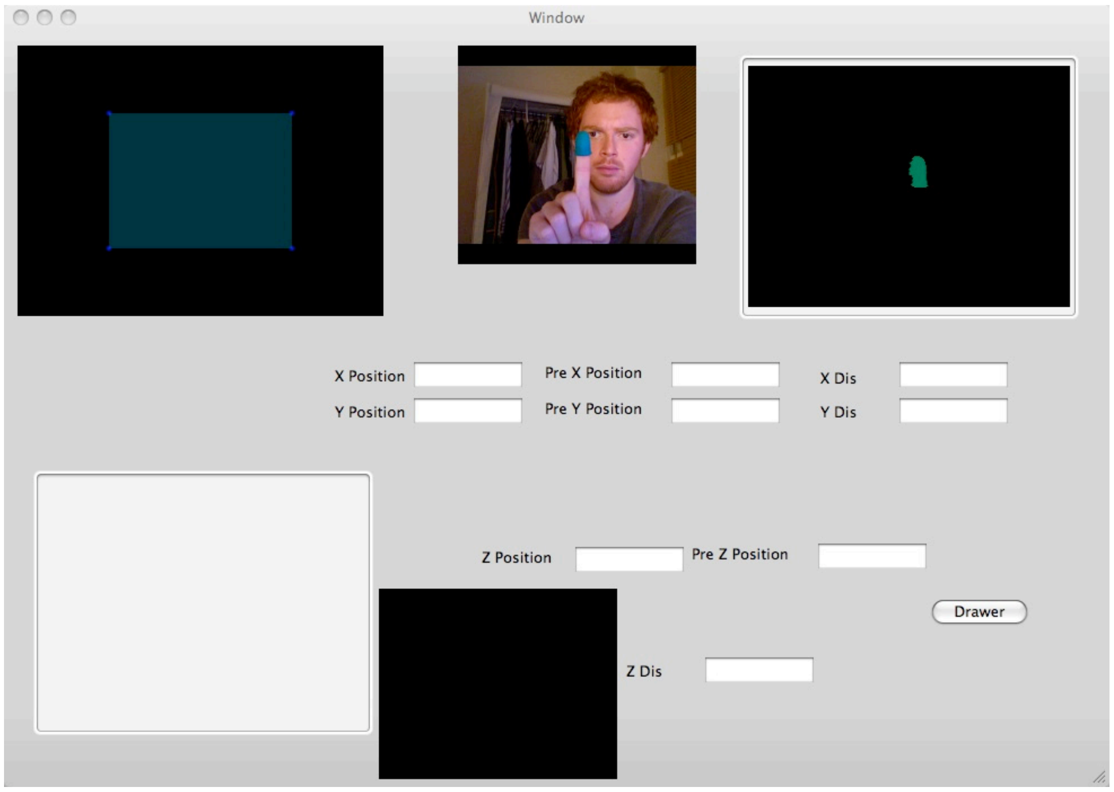
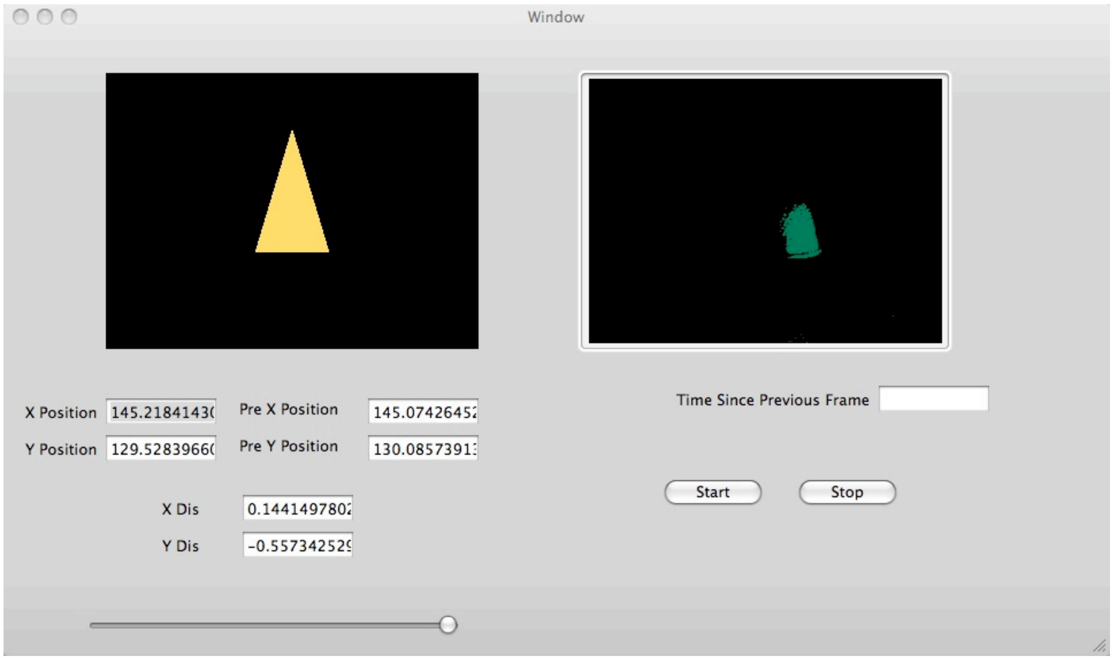


Figure 4.22 GUI during iSight Capture Stage



X Position 145.2184143(

Y Position 129.5283966(

Pre X Position 145.0742645z

Pre Y Position 130.0857391:

X Dis 0.1441497802

Y Dis -0.557342529

Figure 4.23 GUI during Implementation of Tracking Algorithm

The GUI designed for the final systems includes the 3D cube as one of two main features. The other is a feedback window giving the user the finger tracking result as a white point (discussed further in 4.5.6). This advises the user if the system is tracking the location of their finger correctly. Clicking on the Settings button brings out a sliding drawer to the right or left of the window (depending where it is places on screen). This side window is used to adjust the filter sensitivity by click on either the plus or negative buttons. Clicking on capture button displays a frame from the iSight, which has had the filter applied to it. This gives the user the ability to adjust the sensitivity of the filter depending on their surroundings and lets them view the resulting filtered image. Figure 4.24 shows the final 2D video motion tracking system GUI.

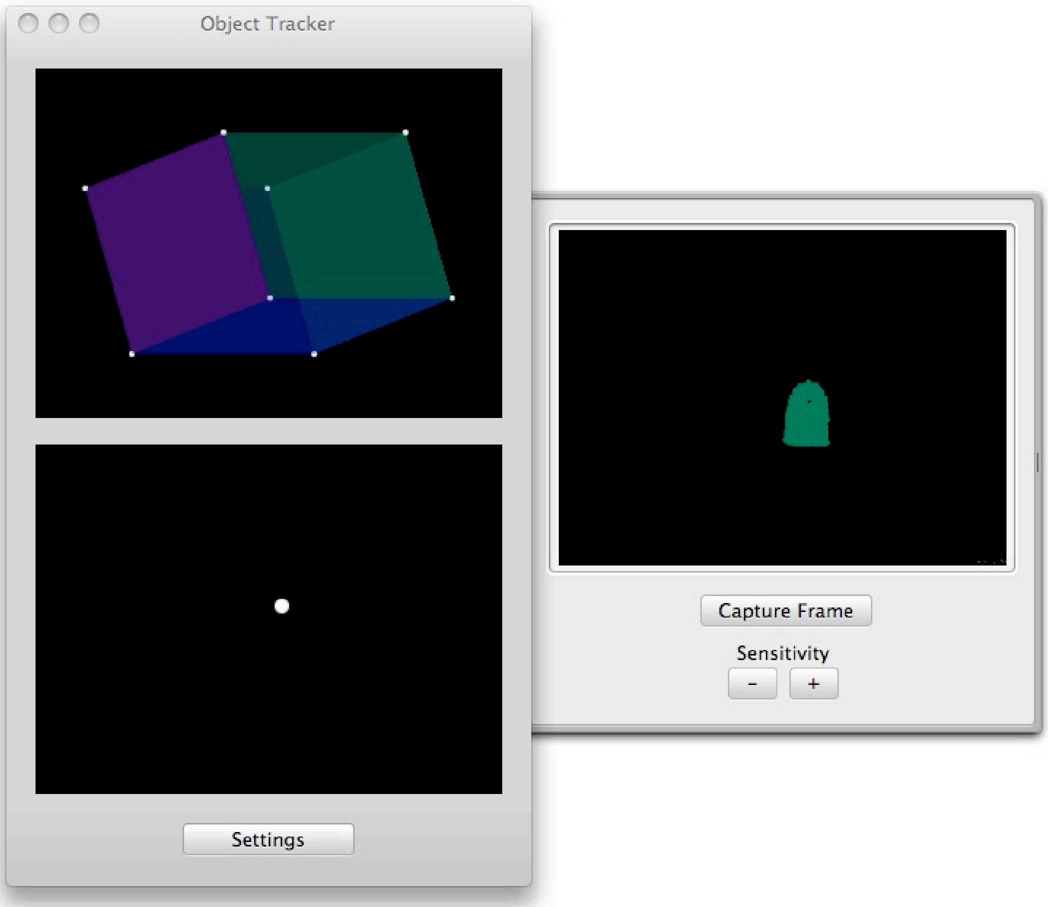


Figure 4.24 Final 2D Video Motion Tracking System GUI

The Settings, Capture Frame, Plus and Minus buttons were also mapped to the computer keyboard as short keys. This enables the user to control the system with one hand while adjusting the filter sensitivity with the other.

4.5.6 Optimisations

Some improvements were made in the final development of the 2D video motion tracking system. The main issue hindering the development of the 3D video motion tracking system was speed. Doubling up the amount of data being processed could crash the system or make the system latency so large it rendered the program useless. The first of two major improvements made to the system was the use of QTCaptureVideoPreviewOutput method instead of QTCaptureDecompressedVideoOutput. Instances of QTCaptureVideoPreviewOutput produce decompressed video frames suitable for preview. Because the output video is intended for preview only, instances may drop frames or reduce output quality in order to improve overall performance of the capture session. [7] QTCaptureDecompressedVideoOutput was designed to never drop frames and to always keep full quality regardless of system performance therefore QTCaptureVideoPreviewOutput is far more suitable. The other major improvement was to exact the colour data from every other pixel, instead of all pixels, thus reducing the computational requirements per frame by half. This does give a slightly less accurate tracking but is a fair trade off for the improved efficiency. The improvements made to the GUI were done because of the lack of feedback being given to the user regarding the perceived location of their finger by the system. This was done using another OpenGL class, using an OpenGL point function to create the white point displayed within the scene. The OpenGL move function was used to move the point using the output result from the tracking algorithm. The coordinate system of the tracking algorithm uses the top left corner of the scene as (0, 0), whereas the OpenGL coordinate system uses the middle of the scene as (0, 0). See figure 4.25.

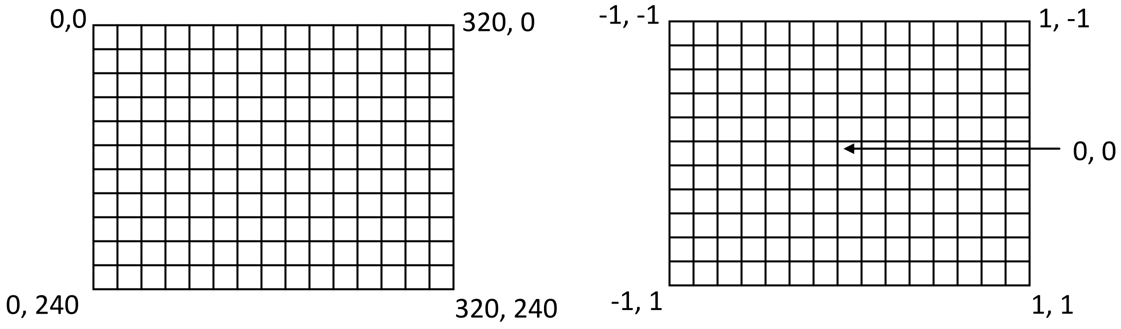


Figure 4.25 Tracking Algorithm and OpenGL Coordinate Systems

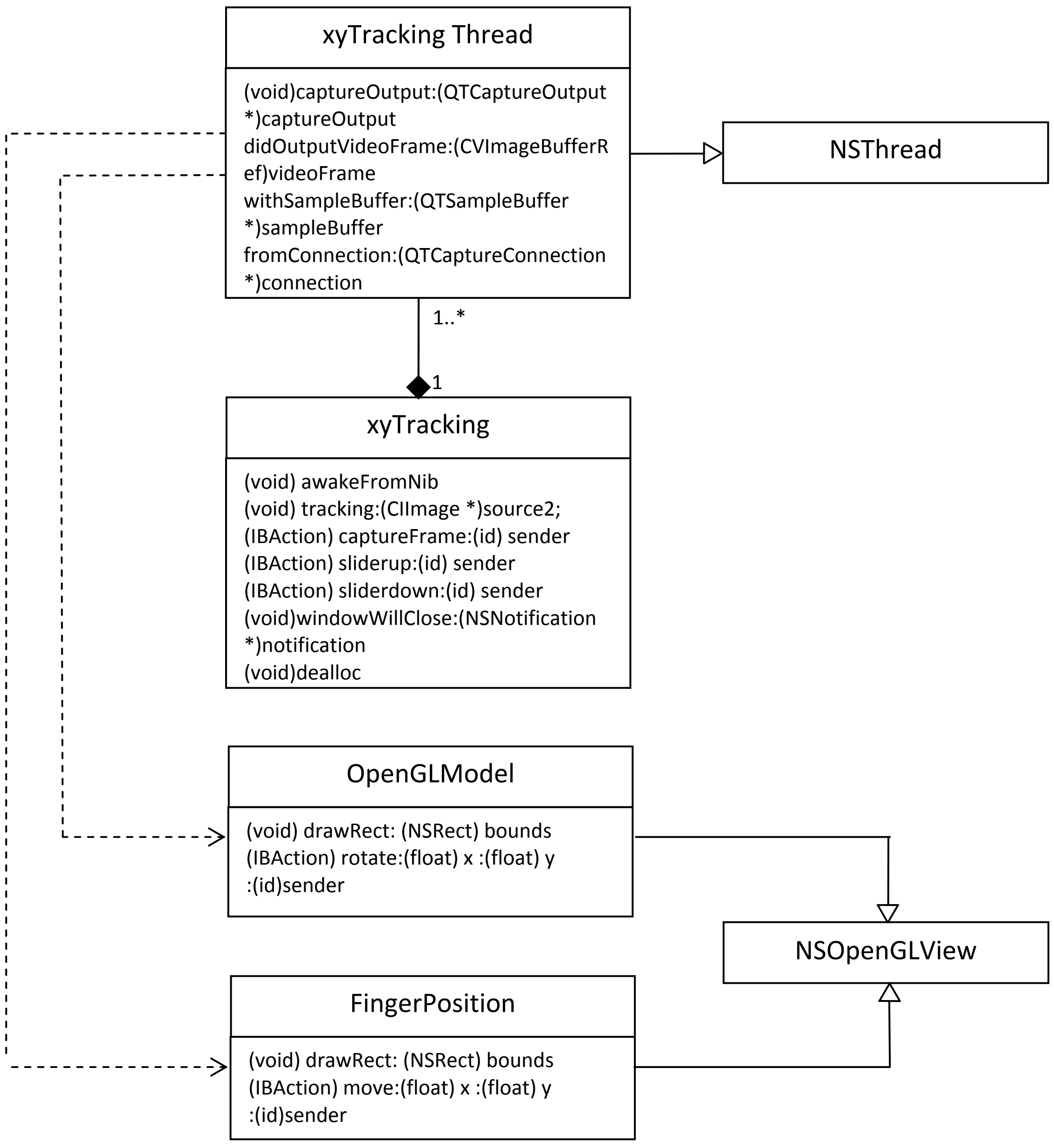
To convert between the tracking algorithm and OpenGL coordinate system a coordinate transform was applied. See equation 4.3:

$$t(x) = (x/160) - 1$$
$$t(y) = (y/120) - 1$$

Equation 4.3 Coordinate Transform Equations

t = transform

4.5.7 2D Motion Tracking System Class Diagram



xyTracking Thread
(void)captureOutput:(QTCaptureOutput *)captureOutput didOutputVideoFrame:(CVImageBufferRef)videoFrame withSampleBuffer:(QTSampleBuffer *)sampleBuffer fromConnection:(QTCaptureConnection *)connection
NSThread
1..*
1
xyTracking
(void) awakeFromNib
(void) tracking:(CIImage *)source2;
(IBAction) captureFrame:(id) sender
(IBAction) sliderup:(id) sender
(IBAction) sliderdown:(id) sender
(void)windowWillClose:(NSNotification *)notification
(void)dealloc
OpenGLModel
(void) drawRect: (NSRect) bounds
(IBAction) rotate:(float) x :(float) y :(id)sender
NSOpenGLView
FingerPosition
(void) drawRect: (NSRect) bounds
(IBAction) move:(float) x :(float) y :(id)sender

4.6 3D Video Motion Tracking System

The 3D video motion tracking system was adapted from the 2D video motion tracking system. It captures from 2 source cameras, the iSight camera and an off the shelf webcam. See figure 4.26.

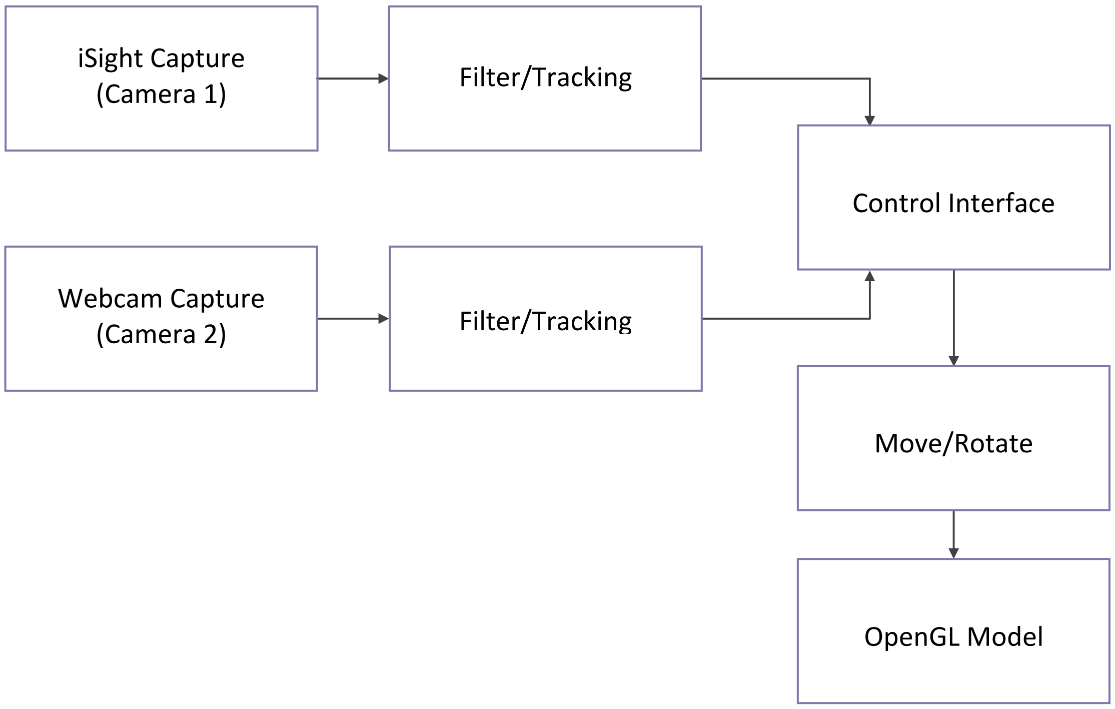


Figure 4.26 3D Video Motion Tracking Data Flow Diagram

To capture two devices simultaneously an array was generated holding the information about the devices connected to the computer. Using this array the two cameras were established as input devices and each connected to a QTCaptureSession. Each QTCaptureSession used a QTCaptureVideoPreviewOutput to output the stream of images, which were filtered using the same code as the 2D video motion tracking system. A major issue which occurred while developing the two camera setup was the inability to stream two cameras connected to the same bus. Because the built in iSight is connected to the High Speed USB bus another High Speed USB camera could not be connected and streamed at the same time. This left the use of a low-resolution standard USB camera or high-resolution Firewire camera. Due to the high costs of Firewire camera a standard USB camera was used. Camera 2 tracks whether the user has simulated a click, which was originally designed to be done using the tracking algorithm and distance equation. If the finger moved a large distance between 2 frames it was assumed they were clicking. To improve efficiency within the system and because of the reduced camera resolution, this method was changed so that the number of green pixels within the scene determined the click. Instead of applying the tracking algorithm to each frame, the number of green pixels was counted and if there were a high number of green pixels it could be assumed the user was clicking. To prevent the system constantly recognising clicks, only when the number of green pixels dropped below a certain number does it recognise clicks again. Clicking determines whether the OpenGL cube is rotated or moved and this was achieved by either selecting the rotate method (within the OpenGL class) or a new move method. See figure 4.27.

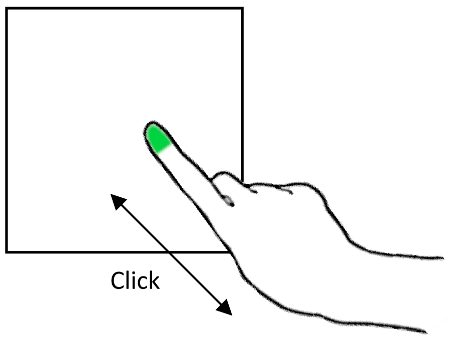
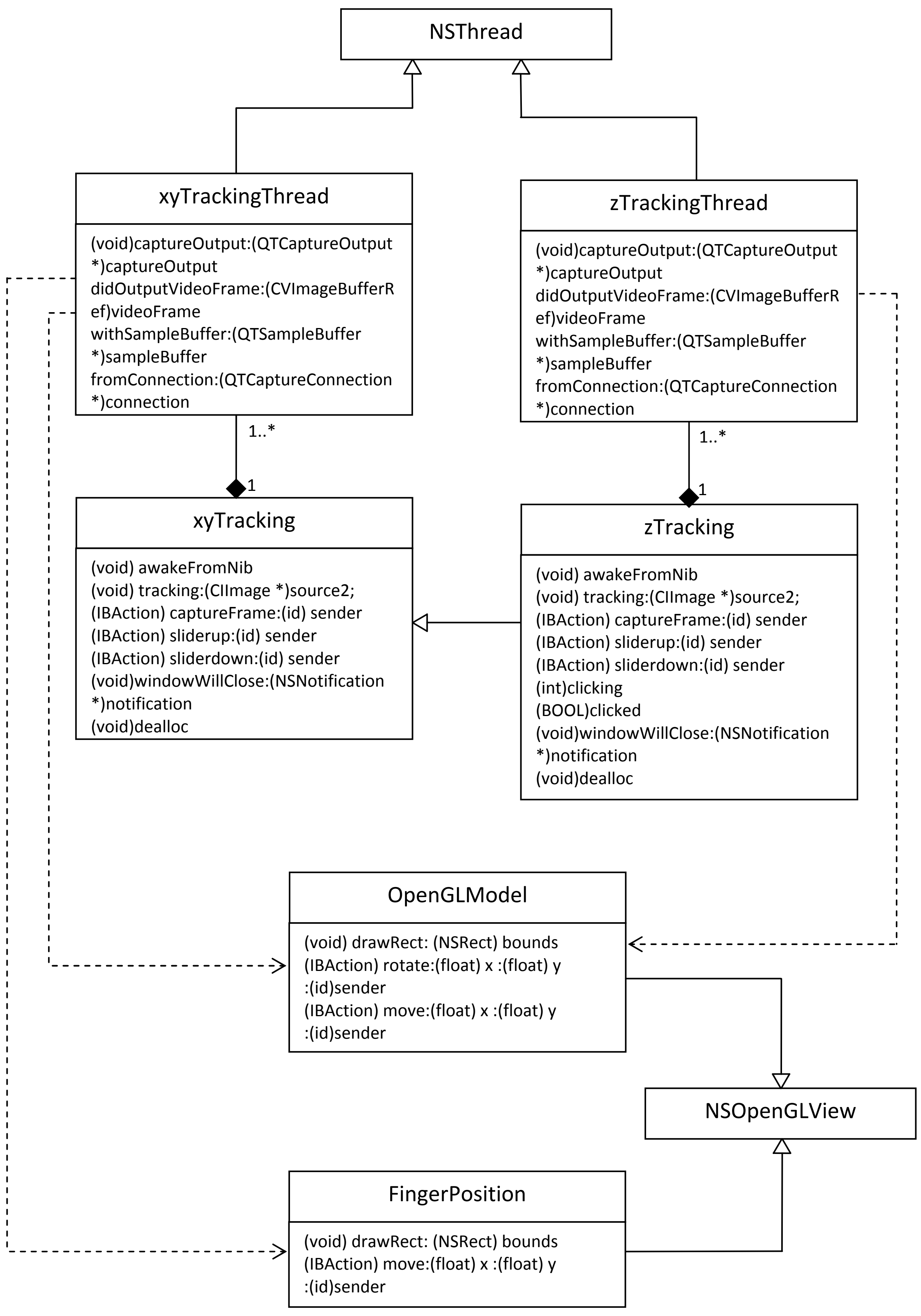


Figure 4.27 Clicking Process

The move method used the OpenGL translate function and the output result from the coordinate transform, used to control OpenGL white point.

4.6.1 3D Tracking System Class Diagram


NSThread
xyTrackingThread
(void)captureOutput:(QTCaptureOutput *)captureOutput didOutputVideoFrame:(CVImageBufferRef)videoFrame withSampleBuffer:(QTSampleBuffer *)sampleBuffer fromConnection:(QTCaptureConnection *)connection
zTrackingThread
(void)captureOutput:(QTCaptureOutput *)captureOutput didOutputVideoFrame:(CVImageBufferRef)videoFrame withSampleBuffer:(QTSampleBuffer *)sampleBuffer fromConnection:(QTCaptureConnection *)connection
1..*
1
1..*
1
xyTracking
(void) awakeFromNib
(void) tracking:(CIImage *)source2;
(IBAction) captureFrame:(id) sender
(IBAction) sliderup:(id) sender
(IBAction) sliderdown:(id) sender
(void)windowWillClose:(NSNotification *)notification
(void)dealloc
zTracking
(void) awakeFromNib
(void) tracking:(CIImage *)source2;
(IBAction) captureFrame:(id) sender
(IBAction) sliderup:(id) sender
(IBAction) sliderdown:(id) sender
(int)clicking
(BOOL)clicked
(void)windowWillClose:(NSNotification *)notification
(void)dealloc
OpenGLModel
(void) drawRect: (NSRect) bounds
(IBAction) rotate:(float) x :(float) y :(id)sender
(IBAction) move:(float) x :(float) y :(id)sender
NSOpenGLView
FingerPosition
(void) drawRect: (NSRect) bounds
(IBAction) move:(float) x :(float) y :(id)sender

4.7 Development Testing

4.7.1 OpenGL Model Test

Description: Evaluate rotate and move functions available from OpenGL API

- The rotate and move functions were tested at the beginning of the development using the OpenGL test bench framework. A 2D object was used and the rotate and move functions connected to a slider to test if the OpenGL model would continually update. The slider automatically generated data which was fed into the OpenGL rotate and translate functions.

Result: The OpenGL model successfully moved, rotated and updated when data was inputted manually and when the slider automatically generated it. See figures 4.28 and 4.29.

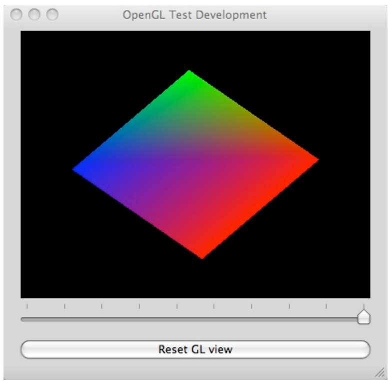


Figure 4.28 Example Test 1: Rotate around z –axis

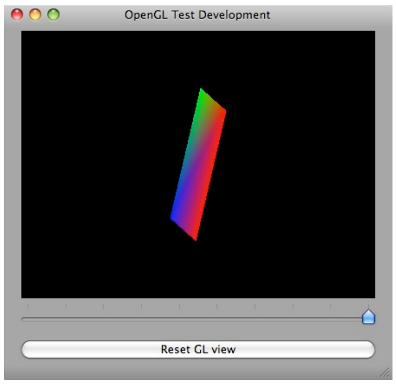


Figure 4.29 Example Test 2: Rotate around x,y,z-axis

4.7.2 Filter Test

Description: Evaluate built-in system filters

- The built-in system filters were evaluated by implementing an image filter application using the image viewer test bench. The following filters were tested:

CIHueAdjust - Changes the overall hue, or tint, of the source pixels

CIColorControls - Adjusts saturation, brightness, and contrast values

CIColorInvert - Inverts the colours in an image

CIColorMatrix - Multiplies colour values and adds a bias factor to each colour component

CIGammaAdjust - Adjusts midtone brightness

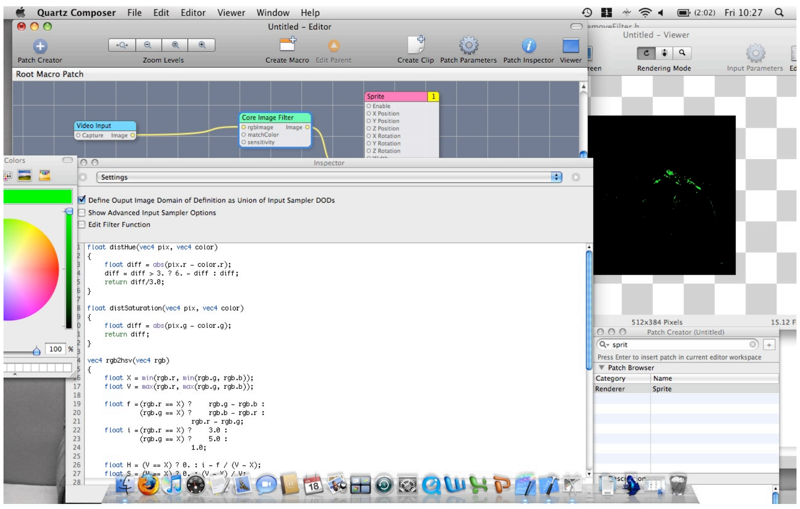
Result: No combination of the built-in system filters offered the correct colour separation therefore a custom filter was produced as previously described in section 4.4.2.

4.7.3 Custom Filter

Description: Test custom filter

- The custom filter was tested using Quartz Composer before it was packaged (see figure 4.30). This allowed rapid optimisation of the filter before development.

Result: Quartz Composer ran filter and improvements were made before the final filter was packaged for release.



Settings

- [✓] Define Ouput Image Domain of Definition as Union of Input Sampler DODs
- [ ] Show Advanced Input Sampler Options
- [ ] Edit Filter Function

```
float distHue(vec4 pix, vec4 color)
{
    float diff = abs(pix.r - color.r);
    diff = diff > 3. ? 6. - diff : diff;
    return diff/3.0;
}

float distSaturation(vec4 pix, vec4 color)
{
    float diff = abs(pix.g - color.g);
    return diff;
}

vec4 rgb2hsv(vec4 rgb)
{
    float X = min(rgb.r, min(rgb.g, rgb.b));
    float V = max(rgb.r, max(rgb.g, rgb.b));

    float f =(rgb.r == X) ?    rgb.g - rgb.b :
             (rgb.g == X) ?    rgb.b - rgb.r :
                            rgb.r - rgb.g;
    float i =(rgb.r == X) ?    3.0 :
             (rgb.g == X) ?    5.0 :
                            1.0;

    float H = (V == X) ? 0. : i - f / (V - X);
    float S = (V == X) ? 0. : (V - X) / V;
```

Figure 4.30 Using Quartz Composer to test filter

4.7.4 Tracking Algorithm

Description: General test to check is algorithm was function correctly

- The tracking algorithm was applied to the stream of frames and the results analysed to confirm it was giving the correct location of the user's finger (see figure 4.31 and 4.32).

Result: Tracking algorithm worked correctly which enabled further development of the system

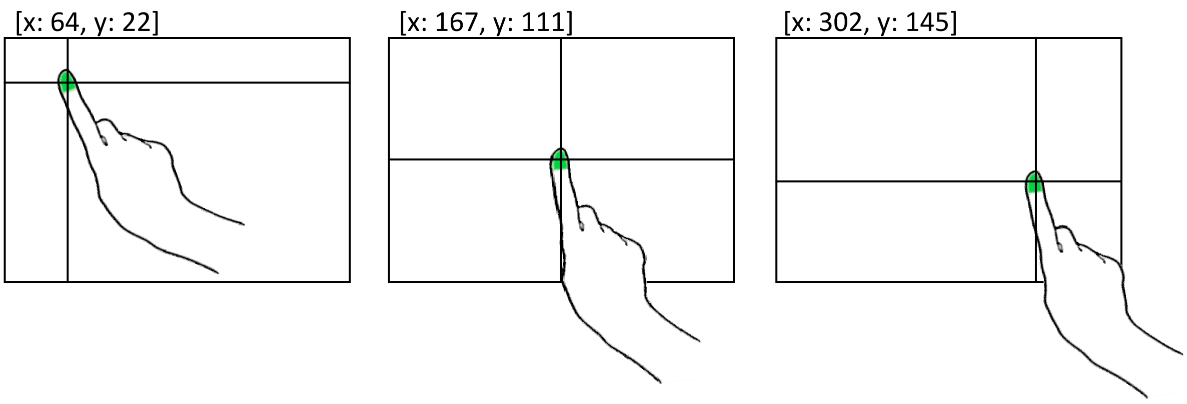


Figure 4.31 Tracking Algorithm Example Test Video Sequence

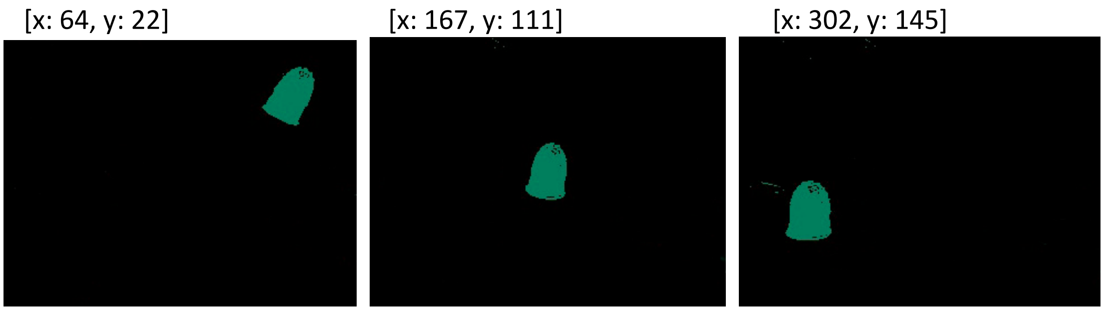


Figure 4.32 Tracking Algorithm Example Test Video Sequence

<u>4.7.5 iSight Capture</u>

Description: General test to check correct capture of iSight

- 2D video motion tracking system test GUI was used to output iSight capture to check if code was functioning correctly.

Result: QTKit was utilised for Mac OS X 10.5 and iSight capture was successfully displayed using an NSImageView with a frame size of 320 x 240 at a 30fps.

<u>4.7.6 2D video motion tracking system filter test</u>

Description: General test to check if filter was functioning correctly

- 2D video motion tracking system test GUI was used to display both the iSight capture and filtered image stream. This was used to analyse the effect of the filter on iSight video stream and enable further development of the system. On the GUI the standard video stream was displayed using an NSImageView along with the filtered video stream (see figure 4.33).

Result: Filter functioned correctly although different light levels do affect filter quality. Having bright light behind the user majorly reduces filter effectiveness. This is due to light not being bounced off the coloured thimble; therefore it does not appear green.

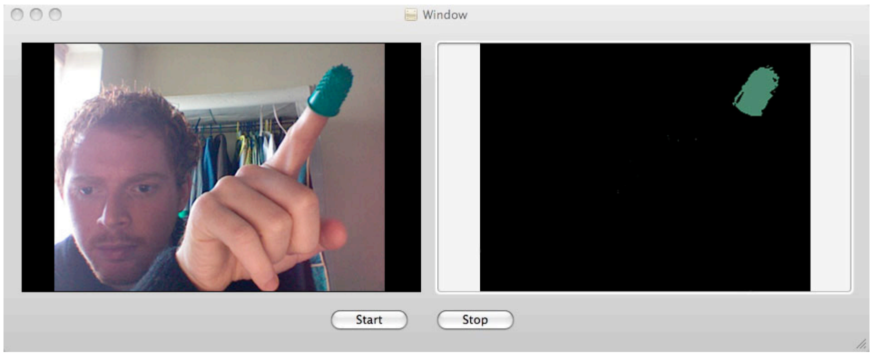


Figure 4.33 iSight and Fiter Video Streams

4.7.7 Memory Allocation

Description: Test for memory allocation and release

- Performance tools Malloc Debug and Leaks were used to check memory leaks within the system. Results from these tools were used to track down memory leaks and fix them.

Result: Many memory leaks were found throughout the system. The majority of these were careless programming errors (due to lack of knowledge regarding Cocoa memory management) which performance tools Malloc Debug and Leaks were unable to track down. All leaks were finally tracked down and fixed.

4.7.8 USB/Firewire Camera

Description: Test standard USB and Firewire cameras for simultaneous streaming

- With the discovery that a High Speed USB camera could not be used testing was carried out on the ability to stream the built in iSight camera and either a standard USB or Firewire camera simultaneously.

Result: Both the standard USB and Firewire camera were successfully streamed simultaneous with the built in iSight camera.

Camera	Fps	Quality	Frame Size
iSight	30fps	High	640 x 480
Standard USB	30fps	Low	352 x 288
Firewire	30fps	High	640 x 480

Table 4.1 Test Cameras Specifications

4.7.9 GUI Testing

Description: General test to check functionally of GUI

- All GUI features were manually tested to confirm system was performing correctly.

Result: System crashed when adjusting sensitivity setting of second camera. This problem was fixed meaning all GUI features function correctly.

5. Evaluation

This chapter discusses the evaluation and outcome of the project. The project was initially evaluated against the requirements outlined in chapter 2. It was then evaluated by testing the latency and jitter of the system and finally by means of a user questionnaire.

5.1 Requirements Evaluation

The project was first evaluated against the requirements outlined in chapter 2. These objectives describe what was required to achieve a successful project.

RQ1 - Understand the process and techniques involved in capturing motion data

Understanding the techniques deployed in modern motion capture was essential in order to implement the input module of the system. Chapter 1 contains a brief outline and description of motion capture. Chapter 2 contains a detailed explanation of the different types of motion capture technologies (RQ1.1), the advantages and disadvantaged of each type of technology and the type of application they might be used in (RQ1.2).

RQ2 - Develop a 3D object for controlling

A 3D object was necessary for the user to have feedback about the capture system. A successfully working 3D model was created using OpenGL in the form of a cube (RQ2.1). This could be controlled using the OpenGL rotate and translate functions (RQ2.2) within real-time (RQ2.3). The design and development of the 3D model is described in sections 3.4 and 4.3 respectively.

RQ3 - Develop a real-time 3D motion tracking program

A real-time 3D motion capture program was successfully developed using low cost equipment and custom produced software. The system produces streams from two cameras (RQ3.1/RQ3.2), filters out unwanted environment (RQ3.3) and uses a centre of mass equation to track movement across x, y and z axis (RQ3.4). This is then mapped to a control interface used to control the OpenGL 3D cube (RQ3.5). The design and development of the 3D motion capture program is described in sections 3.2 and 4.6 respectively.

RQ4 - Develop a control interface to drive the 3D object from captured motion data

A control interface was successfully developed which calculates the distance the user's finger had travelled between frames to drive an OpenGL 3D cube (RQ4.1/RQ4.2/RQ4.3). This objective was exceeded with the inclusion a finger location feedback window. This gave the tracked location of the user's finger as a white point within a black scene. The design and development of the control interface is described in sections 3.3 and 4.5 respectively.

5.2 System Performance

The responsiveness of the system depends on the frame rate of both cameras and overall system latency. Frame rate is the rate at which frames are received and processed by the system from the two cameras. The higher the frame rate the more accurate the system would become because the number of samples being taken of the user's finger is dependent on the frame rate. The fewer samples there are the less accurate the system. The overall system latency is the time delay between when something is initialised to the moment the first effect begins. Within this system is can be defined at the time delay between the user moving their finger and the movement of the 3D model.

The following evaluation tests were completely on a MacBook Intel 2 GHz Core 2 Duo with 1GB of RAM.

Throughout the design and development the target frame rate was approximately 15 frames per second (fps). This would achieve a system that would be responsive to all but extremely fast finger movements. The target rate of 15fps was achieved with the 2D video motion tracking system. Unfortunately with the high levels of data being processed, the frame rate for the 3D video motion tracking system is approximately 7fps per camera. This is well below the target frame rate meaning there is limited response from the 3D video motion tracking system. The limiting factor with increasing the frame rate is the overall available processor speed and memory. Increasing the computer CPU speed and RAM would have a highly beneficial effect on the possible frame rate.

To measure the latency of the system, it was run and the time between a frame being processed to the OpenGL 3D model moving was recorded. With the 2D video motion tracking system the mean latency was 62.2ms over a period of 30 seconds. This is an acceptable latency period giving a smooth and responsive cube movement. The 3D video motion tracking system had a mean latency of 136.5ms over a period of 30 seconds. This is over a two-fold increase on the 2D video motion tracking system giving an unresponsive and jerky user experience. The low frame rate and high latency experienced with the 3D video motion tracking system were foreseen problems and ones which were very difficult to overcome. Only increasing the available processing power and memory of the computer would allow an increase in frame rate and lower latency time.

5.3 User Evaluation

Once the final 2D and 3D video motion tracking systems were developed user evaluation sessions were set up. The targeted user group were ten typical users; all computer literate and all with a general understanding of motion tracking systems. Participants were given an overview of the system and an explanation on how to use the different features. They were asked to test the system and evaluate it based on their level of control, system feedback, GUI and overall experience. Once they had finished they were asked to fill out a questionnaire.

5.3.1 Level of Control

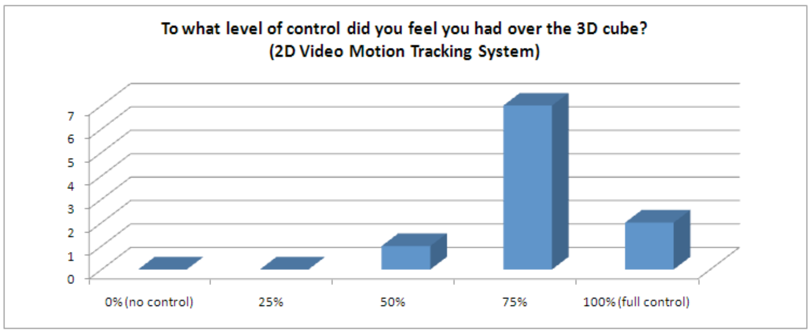


Chart 5.1 Level of Control (2D Video Motion Tracking System)

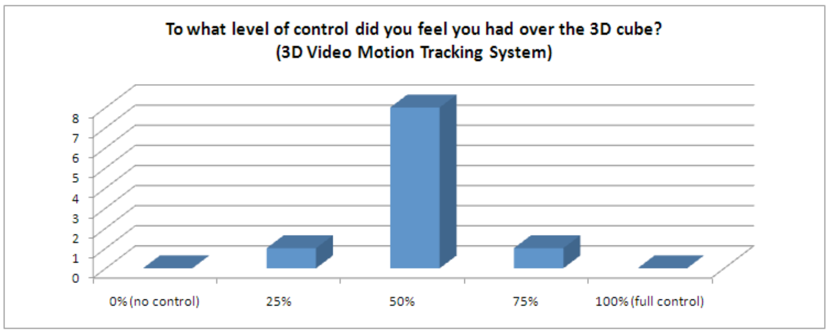


Chart 5.2 Level of Control (3D Video Motion Tracking System)

The results obtained show that participants felt they had a good level of control with 2D video motion tracking system but less control with 3D video motion tracking system. This was mainly due to the low frame rate and high latency experienced with the 3D video motion track system. Results from other questions regarding level of control show that participants felt they could click fairly easily and that there were high levels of accuracy with finger tracking.

5.3.2 Graphical User Interface

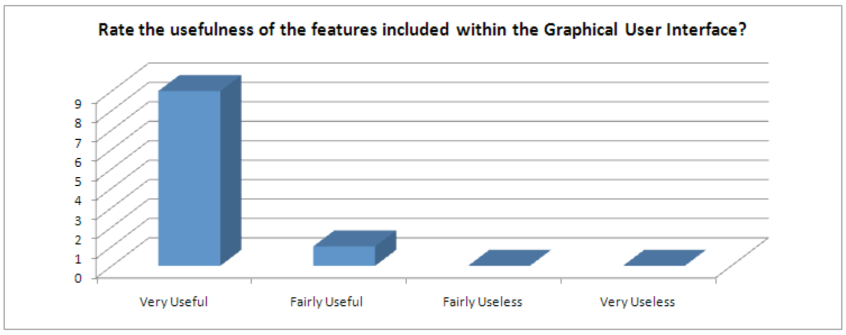


Chart 5.3 GUI (3D Video Motion Tracking System)

Participants found the GUI very useful with all the features being a good idea and implemented to a high standard. They found the sensitivity adjuster very useful especially in different environments e.g. low light conditions. This shows that users were happy to use the system and were pleased with the ease of use.

5.3.3 Feedback

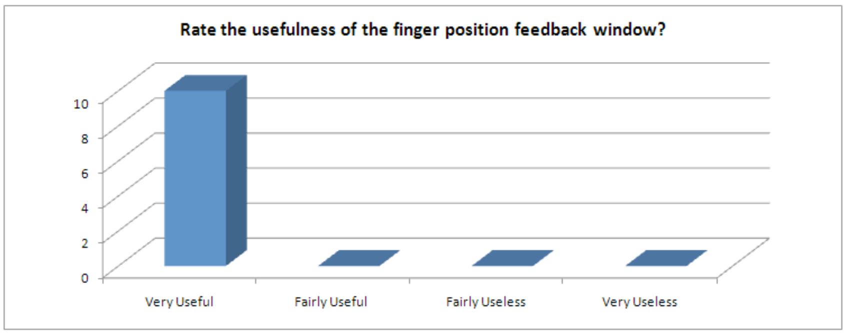


Chart 5.4 Feedback (3D Video Motion Tracking System)

All participants found the finger position feedback window to be very useful showing the inclusion of the extra feedback window was a good idea. Participants also found the cube to be a good shape for best representing the movement effect.

5.3.4 Overall System

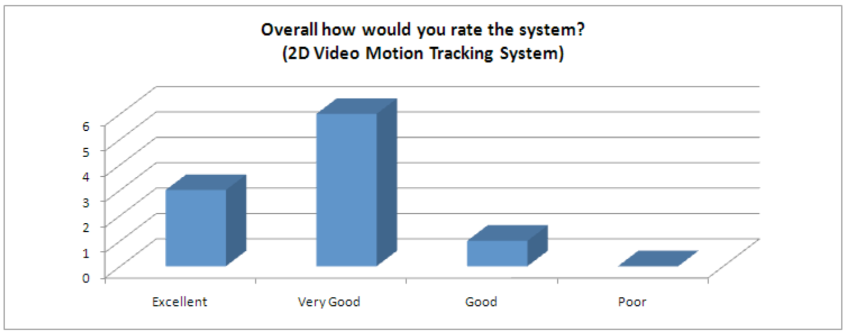


Chart 5.5 Overall System (2D Video Motion Tracking System)

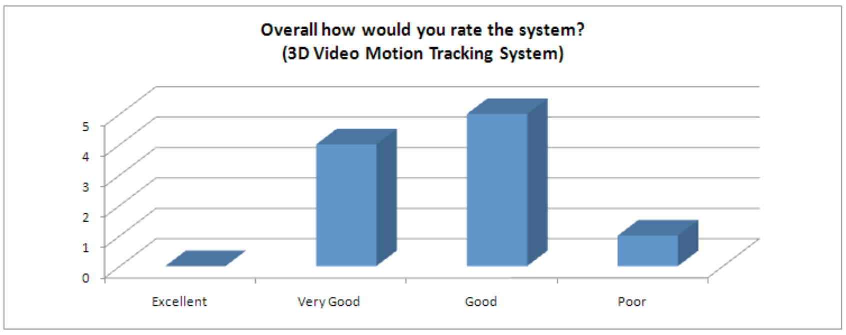


Chart 5.6 Overall System (2D Video Motion Tracking System)

The result obtained about the system overall show that the 2D video motion tracking was well received amongst participants with the majority rating the system as either excellent or very good. Participants were happy with the 3D video motion tracking system, with the majority rating it either very good or good. Unfortunately the low frame rate and latency issues were a problem and this is reflected by the single poor result. Participants were intrigued by the type of control they could have over the cube using just a coloured thimble and reported, despite the speed related problems, that it was an interesting and enjoyable experience.

6. Conclusion and Further Work

The system was developed using Cocoa for Mac OS X operating system giving a large number of possible applications and potential directions. This chapter gives an assessment of the success of the finished system stating the challenges which were faced to successfully achieve the project requirements. It presents the final system and discusses aspects, which could be improved along with possible future work.

6.1 Challenges Overcome For Project Success

During the development of the system many problems and challenges were faced which had to be overcome to successfully complete the system. These challenges also had to be overcome within a limited time frame so that the system was still completed and delivered by the agreed date. The section below outlines the challenges faced during the development of the system and how these were overcome.

6.1.1 Developing using Cocoa for Mac OS X

The understanding and learning of the Cocoa programming language was a very time consuming task. Having only programmed using Java for PC, the jump was larger than expected. Although both languages are object orientated the subtle differences were sometime difficult and frustrating to learn. The increased complexity of the system, compared to previous programming experience, was also sometimes daunting and difficult.

6.1.2 Memory Management

Not having the convenience of automatic garage collection was a major and unforeseen challenge. Initial programming of the system was completed without properly allocating and releasing memory. This meant completing the time consuming task of tracking down and fixing many memory leaks which were abundant throughout the code. Understanding when memory should be released was also a problem because various system components were accessing the same memory from different parts of the system.

6.1.3 Thread Safe

Making the system thread safe meant locking off and unlocking parts of code so that only one thread could access the same data at the same time, therefore preventing the system from crashing. The use of lockFocus, unlockFocus, synchronized and autorelease pool functions were used. One particularly difficult piece of code to make thread safe was the OpenGL cube. This could be re-drawn from a number of different threads and the calling of OpenGL functions kept crashing the system.

6.1.4 Streaming Two Cameras Simultaneously

Being able to stream two cameras simultaneously was a critical aspect of the 3D motion tracking system. Large amounts of time were spent determining why two high speed USB cameras could not be streamed simultaneously. It was later discovered that QTKit was unable to do this, an issue undocumented within the Cocoa API. Testing was then completed establishing that QTKit was able to stream from different buses.

6.1.5 Efficiency

Keeping the system efficient was a challenge which arose throughout the development of the project. Large amounts of data being captured from two cameras, filtering and tracking meant high levels of computation was necessary. Making the system more efficient was an ongoing process requiring creative solutions.

6.2 The Final System

The final system uses colour based motion tracking to implement a usable real-time 3D mouse. A coloured thimble placed on the user's index finger is tracked and the results obtained mapped to a control interface which is used to control an OpenGL cube in real-time. The system successfully tracks the x and y axis using the high-resolution iSight camera built into MacBook computers. It allows the user to simulate a click using a second low cost, low resolution, off the shelf webcam by tracking whether the finger has entered it field of view. The frame rate of the system was unfortunately lower and the latency time higher than specified within the design. This was due to the high levels of data being produced, being unable to be sufficiently processed by the CPU.

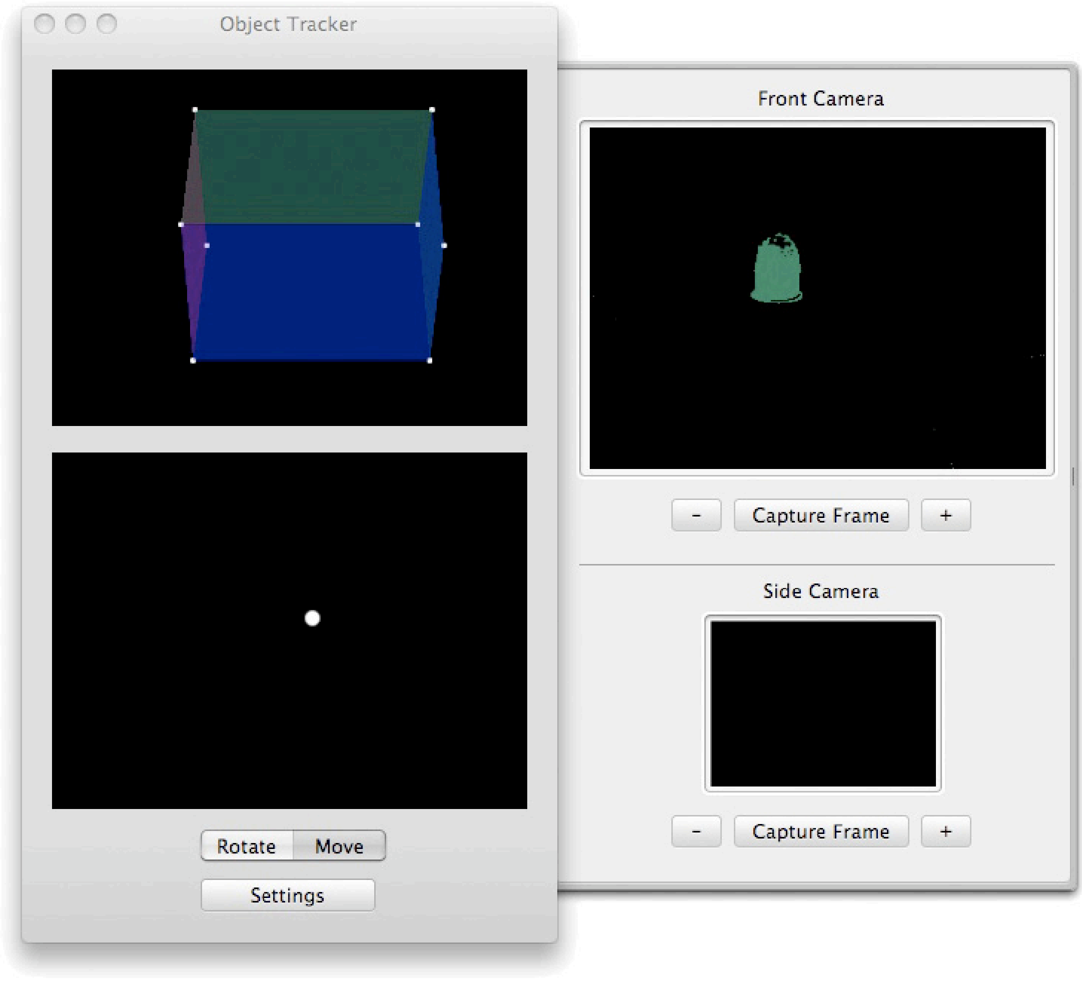


Figure 6.1 Final 3D Motion Tracking System GUI

In conclusion, colour-based finger tracking can be used to implement a gesture controlled real-time 3D mouse in software. There are some issues with possible frame rate and high latency, but these will reduce as technology advances and available processing power increases. Users were able to control the 3D cube and simulate clicks using their finger alone. Participants in the user based evaluation found the system interesting and enjoyable to use, rating it either good or very good, but were disappointed with the latency issues affecting the tracking and 3D cube control. They found the GUI very useful and were pleased with features included, especially the finger location feedback window.

6.3 Future Work

6.3.1 Infrared Tracking

Using infrared to track the location of the user's finger can greatly improve the tracking accuracy and reduce the level computation required. This can be achieved using a standard camera using an infrared filter [13]. This type of filter will only let infrared light pass through it therefore removing the need for a filter to be produced in software. Because the only infrared light being produced will be from an emitter attached to the user's finger, tracking could easily be achieved in almost any condition regardless of the surrounding environment. This type of tracking already has proven potential within commercial applications when it was used as the chosen method of tracking for the Nintendo Wii games console.

6.3.2 Multiple Point Tracking

The ability to track multiple points would greatly increase the functionally and potential applications of the system. To achieve this a more sophisticated tracking algorithm would need to be produced which is capable of tracking multiple points simultaneously. This could then be used for greater user interaction; for example specific two finger gesture controls could be used to scale up or down the 3D cube. Object rotation and movement could be achieved simultaneously along with controlling the cube with one finger while clicking with the other. Overall this would produce far greater system functionality.

7. References

[1] Animazoo (2007) Inertial Gyro Motion Capture Systems. [online] Available from: http://www.animazoo.com/InertialGyroMotionCaptureSystems.aspx [accessed 11/04/2008]

[2] Animazoo (2008) The Mocap Specialists. [online] Available from: http://www.animazoo.com/default.aspx [accessed 18/04/2008]

[3] Apple Developer Connection (2007) Core Image. [online] Available from: http://developer.apple.com/graphicsimaging/coreimage/ [accessed 02/12/2007]

[4] Apple Developer Connection (2007) How QTKit Capture Works. [online] Available from : http://developer.apple.com/documentation/QuickTime/Conceptual/QTKitCaptureProgrammingGuide/GettingStarted/chapter_2_section_3.html [accessed 03/04/2008]

[5] Apple Developer Connection (2007) Mac OS X. [online] Available from:http://developer.apple.com/macosx/ [accessed 02/12/2007]

[6] Apple Developer Connection (2007) Mac OS X System Architecture. [online] Available from: http://developer.apple.com/macosx/architecture/ [accessed 02/12/2007]

[7] Apple Developer Connection (2007) QTCaptureVideoPreviewOutput Class Reference. [online] Available from: http://developer.apple.com/documentation/QuickTime/Reference/QTCaptureVideoPreviewOutput_Class/Reference/Reference.html [accessed 03/04/2008]

[8] Apple Developer Connection (2007) XCode. [online] Available from: http://developer.apple.com/tools/xcode/ [accessed 02/12/2007]

[9] August, J (2005) Digital filmmaking and the paradox of choice. [online] Available from: http://johnaugust.com/Assets/polarex1.jpg [accessed 02/12/2007]

[10] British Computer Society (2006) Code of Conduct for Members. [online] Available from: http://www.bcs.org/server.php?show=conWebDoc.1588 [accessed 02/12/2007]

[11] British Computer Society (2007) Code of Practice. [online] Available from: http://www.bcs.org/server.php?show=conWebDoc.1590 [accessed 02/12/2007]

[12] Canny, J (1986) a Computational Approach To Edge detection. IEEE Trans. Pattern Anal. Mach. Intell. Vol. 8 pp. 679-698

[13] Finny, A (2007) Invisible Light. [online] Available from: http://www.atsf.co.uk/ilight/tech/ilightec.html

[14] Hillegass, A (2004) Cocoa Programming for Mac OS X. 2nd ed., p.4. Pearson Education, Inc.

[15] La Fantasy au quotidian (2007) Beowulf. [online] Available from: http://www.elbakin.net/plume/xmedia/fantasy/news/autres_films/jolie-beowulf-capteurs.jpg [accessed 02/12/2007]

[16] Menache, A (1999) Understanding Motion Capture for Computer Animation and Video Games. 1st ed., p1 Morgan Kaufmann

[17] Menache, A (1999) Understanding Motion Capture for Computer Animation and Video Games. 1st ed., p14 Morgan Kaufmann

[18] Menache, A (1999) Understanding Motion Capture for Computer Animation and Video Games. 1st ed., p20 Morgan Kaufmann

[19] Menache, A (1999) Understanding Motion Capture for Computer Animation and Video Games. 1st ed., p23 Morgan Kaufmann

[20] Meta Motion (2004) Optical Motion Capture Systems. [online] Available from: http://www.metamotion.com/motion-capture/optical-motion-capture-1.htm [accessed [18/04/2008]

[21] Office of Public Sector Information (1995) Disability Discrimination Act. [online] Available from: http://www.opsi.gov.uk/acts/acts1995/ukpga_19950050_en_4#pt3-pb1-l1g19 [accessed 02/12/2007]

[22] Recycling Guide (2007) Reduce, Reuse, Recycle. [online] Available from: http://www.recycling-guide.org.uk/rrr.html [accessed 02/12/2007]

[23] Walsh, G & Bishop G (2006) An Introduction to the Kalman Filter. Department of Computer Science, University of North Carolina at Chapel Hill

[24] Scott Owen, G (1999) A Protical Approach Motion Capture: Acclaim's optical motion capture system. [online] Available from: http://www.siggraph.org/education/materials/HyperGraph/animation/character_animation/motion_capture/motion_optical.htm#An%20overview%20of%20current%20input%20systems [accessed 18/04/2008]

[25] Xsens (2007) Human Motion Analysis. [online] Available from: http://www.xsens.com/index.php?mainmenu=technology&submenu=research&subsubmenu=human_motion [accessed 18/04/2008]

8. Appendix A

8.1 2D Motion Tracking System

8.1.1 xyTracking

```
//  Created by Patrick Holroyd
//  Copyright Patrick Holroyd 2008. All rights reserved.

/* xyTracking */

#import <Cocoa/Cocoa.h>
#import <QTKit/QTkit.h>
#import <MyOpenGLView.h>
#import <FingerPosition.h>

@interface xyTracking : NSObject {

       IBOutlet NSImageView                                   *mImageView;
       IBOutlet NSTextField                                   *xtext;
       IBOutlet NSTextField                                   *ytext;
       IBOutlet NSTextField                                   *xdistext;
       IBOutlet NSTextField                                   *ydistext;
       IBOutlet NSTextField                                   *prextext;
       IBOutlet NSTextField                                   *preytext;
       IBOutlet NSTextField                                   *timetext;
       IBOutlet MyOpenGLView                                  *pOpenGLView;
       IBOutlet FingerPosition                                *pFingerPosition;
       IBOutlet QTCaptureView                                 *mCaptureView;
       IBOutlet NSSlider                                      *slidersens;
       IBOutlet NSButton                                      *settings;
       IBOutlet NSButton                                      *capture;
       IBOutlet NSButton                                      *sliderup;
       IBOutlet NSButton                                      *sliderdown;

       NSImage                                                *image;
       QTCaptureSession                                       *mCaptureSession;
       QTCaptureSession                                       *mCaptureSession2;
       QTCaptureVideoPreviewOutput
       *mCaptureDecompressedVideoOutput;
       QTCaptureVideoPreviewOutput
       *mCaptureDecompressedVideoOutput2;
    QTCaptureDeviceInput                                      *mCaptureVideoDeviceInput;
       QTCaptureDeviceInput                                   *mCaptureVideoDeviceInput2;
       CVImageBufferRef                                        mCurrentImageBuffer;
       CVImageBufferRef                                       imageBuffer;

       BOOL                                                   firsttrack;
       float                                                  pCOG_X;
       float                                                  pCOG_Y;
       float                                                  sensitivity;

}


- (void) MyInstanceThreadMethod:(CIImage *)source2;
- (IBAction) captureFrame:(id) sender;
- (IBAction) sliderup:(id) sender;
- (IBAction) sliderdown:(id) sender;

@end

//  xyTracking.h
//  xyTracking
//
//  Created by Patrick Holroyd
//  Copyright Patrick Holroyd 2008. All rights reserved.

#import "xyTracking.h"

@implementation xyTracking


- (void)awakeFromNib
{

// Create the capture session

       mCaptureSession = [[QTCaptureSession alloc] init];

// Connect inputs and outputs to the session

       BOOL success = NO;
       NSError *error;

// Find a video device

       NSArray *inputs = [QTCaptureDevice inputDevicesWithMediaType:QTMediaTypeVideo];
       QTCaptureDevice *videoDevice = [inputs objectAtIndex:0];
    success = [videoDevice open:&error];

// If a video input device can't be found or opened, try to find and open a muxed input
device

       if (!success) {
          videoDevice = [QTCaptureDevice defaultInputDeviceWithMediaType:QTMediaTypeMuxed];
       success = [videoDevice open:&error];
    }

    if (!success) {
          //Handle error
          videoDevice = nil;
          NSLog(@"No Camera");
       }


//Add the video device to the session as a device input

       mCaptureVideoDeviceInput = [QTCaptureDeviceInput deviceInputWithDevice: videoDevice];
       success = [mCaptureSession addInput:mCaptureVideoDeviceInput error:&error];
       if (!success) {
          //Handle error
          NSLog(@"Cannot add camera to session");
       }

       mCaptureDecompressedVideoOutput = [[QTCaptureVideoPreviewOutput alloc] init];
       [mCaptureDecompressedVideoOutput setDelegate:self];
       success = [mCaptureSession addOutput:mCaptureDecompressedVideoOutput error:&error];
       if (!success) {
                   NSLog(@"Output not added");
       }

       firsttrack = YES;
       sensitivity = 1.5;
       [settings setKeyEquivalent:@"s"];
       [capture setKeyEquivalent:@"c"];
       [sliderup setKeyEquivalent:@"x"];
```

```
        [sliderdown setKeyEquivalent:@"z"];

        [mCaptureSession startRunning];
        [mCaptureSession2 startRunning];

}

/*
Thread method which applies filter, scans pixels and applying Centre of Mass algorithm
*/
- (void) MyInstanceThreadMethod:(CIImage *)inputimage;
{
        @synchronized (self) {

        NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

        [CIPlugIn loadAllPlugIns];
        CIFilter *cFilter = [[CIFilter alloc] init];
        CIFilter *gFilter = [[CIFilter alloc] init];

        [pool addObject:cFilter];
        [pool addObject:gFilter];

//Apply Gaussian Blur filter

        gFilter = [CIFilter filterWithName: @"CIGaussianBlur"];
        [gFilter setDefaults];
        [gFilter setValue: inputimage forKey: @"inputImage"];
        [gFilter setValue: [NSNumber numberWithFloat: 5.0] forKey: @"inputRadius"];
        CIImage *source2= [gFilter valueForKey: @"outputImage"];

//Apply custom filter

        cFilter = [CIFilter filterWithName: @"ColorRemoveFilter"];
        [cFilter setDefaults];
        [cFilter setValue: source2 forKey: @"inputImage"];
        [cFilter setValue: [CIColor colorWithRed: 0.00390 green: 0.41406 blue: 0.29687 alpha:
1.0] forKey: @"matchColor"];
        [cFilter     setValue:     [NSNumber     numberWithFloat:     sensitivity]     forKey:
@"inputSensitivity"];
        CIImage *result = [cFilter valueForKey: @"outputImage"];

//Resize frame

        NSSize xsize = {320.0, 240.0};
        NSCIImageRep *imageRep = [NSCIImageRep imageRepWithCIImage:result];
        [imageRep retain];
        [pool addObject:imageRep];
        [image release];
        image = [[NSImage alloc] initWithSize:xsize];

        [image addRepresentation:imageRep];
        [image retain];
        [pool addObject:image];

        NSData *tiff_data = [[NSData alloc] initWithData:[image TIFFRepresentation]];
        [pool addObject:tiff_data];

        NSBitmapImageRep *bitmap = [[NSBitmapImageRep alloc] initWithData:tiff_data];
        [pool addObject:bitmap];

//Scan over pixels

        int x;
        int y;
        float COG_X;
        float COG_Y;
        float total;
        int totalGreen = 0;
```

```
        for(x = 0; x < 160; x++)
        {
        for (y = 0; y < 120; y++)
        {
            NSUInteger pixelData[3];
            [bitmap getPixel:pixelData atX:x*2 y:y*2];
            float k = (pixelData[0]+pixelData[1]+pixelData[2])/3;

            if(k == 61)
            {
                      totalGreen++;
            }

            COG_X = COG_X + (k*x*2);
            COG_Y = COG_Y + (k*y*2);
            total = total + k;

            }
            }

//Stop tracking if too much green is in frame

            BOOL track = YES;
            if(totalGreen > 3500)
            {
                      firsttrack = YES;
                      track = NO;
            }

//Stop tracking if too little green is in frame

            if(totalGreen < 150)
            {
            firsttrack = YES;
            track = NO;
            }

            if(track)
            {

            if(firsttrack)
            {

//If first tracking only store finger location

                      COG_X = ((COG_X / total) * - 1) +320;
                      pCOG_X = COG_X;
                      COG_Y = COG_Y / total;
                      pCOG_Y = COG_Y;
                      firsttrack = NO;
            }

            else
            {

//If not first tracking apply distance equation

                      COG_X = ((COG_X / total) * - 1) +320;
                      float xdis = COG_X - pCOG_X;
                      COG_Y = COG_Y / total;
                      float ydis = COG_Y - pCOG_Y;

                      //Map data to OpenGL cube

                      [pOpenGLView rotate:xdis :ydis :self];
                      [pFingerPosition move:(COG_X/160)-1 :((COG_Y/120)-1) * -1 :self];
                      [prextext setFloatValue:(COG_X/160)-1];
                      [preytext setFloatValue:(COG_Y/120)-1];
```

```
                    [xtext setFloatValue:COG_X];
                    [ytext setFloatValue:COG_Y];
                    [xdistext setFloatValue:COG_X];
                    [ydistext setFloatValue:COG_Y];
                    pCOG_X = COG_X;
                    pCOG_Y = COG_Y;
                    }
                    }

                    [pool release];
                    }
}

/*
Capture frame for GUI
*/
- (IBAction) captureFrame:(id) sender
{
      [mImageView setImage:image];
}

/*
Increase sensitivity
*/
- (IBAction) sliderup:(id) sender
{
      sensitivity = sensitivity - 0.1;
      [mCaptureSession stopRunning];
      [mCaptureSession startRunning];
      [self captureFrame:self];
}

/*
Decrease sensitivity
*/
- (IBAction) sliderdown:(id) sender
{
      sensitivity = sensitivity + 0.1;
      [mCaptureSession stopRunning];
      [mCaptureSession startRunning];
      [self captureFrame:self];
}


/*
Capture frame from xyCamera
*/
-                     (void)captureOutput:(QTCaptureOutput                    *)captureOutput
didOutputVideoFrame:(CVImageBufferRef)videoFrame            withSampleBuffer:(QTSampleBuffer
*)sampleBuffer fromConnection:(QTCaptureConnection *)connection
{

    // Store the latest frame

      CVImageBufferRef imageBufferToRelease;
    CVBufferRetain(videoFrame);
      @synchronized (self)
      {
         imageBufferToRelease = mCurrentImageBuffer;
       mCurrentImageBuffer = videoFrame;
    }

    @synchronized (self)
      {
       imageBuffer = CVBufferRetain(mCurrentImageBuffer);
    }

      if (imageBuffer)
      {
          CIImage *source = [CIImage imageWithCVImageBuffer:imageBuffer];
//Dispatch thread
          [self MyInstanceThreadMethod:source];
      }

      CVBufferRelease(imageBufferToRelease);
      CVBufferRelease(mCurrentImageBuffer);
}

#pragma mark-

/*
Handle window closing notifications for device inputs
*/
- (void)windowWillClose:(NSNotification *)notification
{

      [mCaptureSession stopRunning];

    if ([[mCaptureVideoDeviceInput device] isOpen])
      {
        [[mCaptureVideoDeviceInput device] close];
      }
}

/*
Handle deallocation of memory for your capture objects
*/
- (void)dealloc
{
      [mCaptureSession release];
      [mCaptureVideoDeviceInput release];
    [mCaptureDecompressedVideoOutput release];
      [super dealloc];
}

#pragma mark-

@end
```

8.1.2 MyOpenGLView

```
//  Created by Patrick Holroyd
//  Copyright Patrick Holroyd 2008. All rights reserved.

/* MyOpenGLView */

#import <Cocoa/Cocoa.h>

@interface MyOpenGLView : NSOpenGLView
{
      float rotX;
      float rotY;
}

- (void) drawRect: (NSRect) bounds ;
- (IBAction) rotate:(float) x :(float) y :(id)sender;

@end

//
//  MyOpenGLView.h
//  MyOpenGLView
//
//  Created by Patrick Holroyd
//  Copyright Patrick Holroyd 2008. All rights reserved.

#import "MyOpenGLView.h"
#include <OpenGL/gl.h>

@implementation MyOpenGLView


/*
OpenGL draw method
*/
-(void) drawRect: (NSRect) bounds
{

       //Lock drawing code
       if( [self lockFocusIfCanDraw] == YES ) {
       {

       //Setup OpenGL
       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
       glShadeModel(GL_SMOOTH);
       glEnable(GL_DEPTH_TEST);
       glDepthFunc(GL_LEQUAL);
       glClearColor(0.0, 0.0, 0.0, 0.0);
       glClearDepth(1.0);
       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
       glEnable (GL_BLEND);
       glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

       glPointSize(3.0);
       glEnable(GL_POINT_SMOOTH);
       glHint(GL_POINT_SMOOTH_HINT, GL_NICEST);
       glEnable(GL_BLEND);
       glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
       glLoadIdentity();

       //Rotate cube using result from tracking
       glRotatef(rotX,0.0,1.0f,0.0f);
       glRotatef(rotY,1.0f,0.0f,0.0f);
       glBegin(GL_POINTS);
       {

          glColor4f(1.0, 1.0, 1.0, 1.0);
          glVertex3f( 0.5f, 0.5f,-0.5f);         // Top Right Of The Quad (Top)
          glVertex3f(-0.5f, 0.5f,-0.5f);         // Top Left Of The Quad (Top)
          glVertex3f(-0.5f, 0.5f, 0.5f);         // Bottom Left Of The Quad (Top)
          glVertex3f( 0.5f, 0.5f, 0.5f);         // Bottom Right Of The Quad (Top)

          glColor4f(1.0, 1.0, 1.0, 1.0);
          glVertex3f( 0.5f,-0.5f, 0.5f);         // Top Right Of The Quad (Bottom)
          glVertex3f(-0.5f,-0.5f, 0.5f);         // Top Left Of The Quad (Bottom)
          glVertex3f(-0.5f,-0.5f,-0.5f);         // Bottom Left Of The Quad (Bottom)
          glVertex3f( 0.5f,-0.5f,-0.5f);         // Bottom Right Of The Quad (Bottom)

          glColor4f(1.0, 1.0, 1.0, 1.0);
          glVertex3f( 0.5f, 0.5f, 0.5f);         // Top Right Of The Quad (Front)
          glVertex3f(-0.5f, 0.5f, 0.5f);         // Top Left Of The Quad (Front)
          glVertex3f(-0.5f,-0.5f, 0.5f);         // Bottom Left Of The Quad (Front)
          glVertex3f( 0.5f,-0.5f, 0.5f);         // Bottom Right Of The Quad (Front)

          glColor4f(1.0, 1.0, 1.0, 1.0);
          glVertex3f( 0.5f,-0.5f,-0.5f);         // Top Right Of The Quad (Back)
          glVertex3f(-0.5f,-0.5f,-0.5f);         // Top Left Of The Quad (Back)
          glVertex3f(-0.5f, 0.5f,-0.5f);         // Bottom Left Of The Quad (Back)
          glVertex3f( 0.5f, 0.5f,-0.5f);         // Bottom Right Of The Quad (Back)

          glColor4f(1.0, 1.0, 1.0, 1.0);
          glVertex3f(-0.5f, 0.5f, 0.5f);         // Top Right Of The Quad (Left)
          glVertex3f(-0.5f, 0.5f,-0.5f);         // Top Left Of The Quad (Left)
          glVertex3f(-0.5f,-0.5f,-0.5f);         // Bottom Left Of The Quad (Left)
          glVertex3f(-0.5f,-0.5f, 0.5f);         // Bottom Right Of The Quad (Left)

          glColor4f(1.0, 1.0, 1.0, 1.0);
          glVertex3f( 0.5f, 0.5f,-0.5f);         // Top Right Of The Quad (Right)
          glVertex3f( 0.5f, 0.5f, 0.5f);         // Top Left Of The Quad (Right)
          glVertex3f( 0.5f,-0.5f, 0.5f);         // Bottom Left Of The Quad (Right)
          glVertex3f( 0.5f,-0.5f,-0.5f);         // Bottom Right Of The Quad (Right)
          }
          glEnd();


          glBegin(GL_QUADS);
          {

          glColor4f(0.0, 0.0, 1.0, 0.2);
          glVertex3f( 0.5f, 0.5f,-0.5f);         // Top Right Of The Quad (Top)
          glVertex3f(-0.5f, 0.5f,-0.5f);         // Top Left Of The Quad (Top)
          glVertex3f(-0.5f, 0.5f, 0.5f);         // Bottom Left Of The Quad (Top)
          glVertex3f( 0.5f, 0.5f, 0.5f);         // Bottom Right Of The Quad (Top)

          glColor4f(0.0, 0.0, 1.0, 0.2);
          glVertex3f( 0.5f,-0.5f, 0.5f);         // Top Right Of The Quad (Bottom)
          glVertex3f(-0.5f,-0.5f, 0.5f);         // Top Left Of The Quad (Bottom)
          glVertex3f(-0.5f,-0.5f,-0.5f);         // Bottom Left Of The Quad (Bottom)
          glVertex3f( 0.5f,-0.5f,-0.5f);         // Bottom Right Of The Quad (Bottom)

          glColor4f(0.0,1.0, 0.0, 0.2);
          glVertex3f( 0.5f, 0.5f, 0.5f);         // Top Right Of The Quad (Front)
          glVertex3f(-0.5f, 0.5f, 0.5f);         // Top Left Of The Quad (Front)
          glVertex3f(-0.5f,-0.5f, 0.5f);         // Bottom Left Of The Quad (Front)
          glVertex3f( 0.5f,-0.5f, 0.5f);         // Bottom Right Of The Quad (Front)

          glColor4f(0.0, 0.0, 1.0, 0.2);
          glVertex3f( 0.5f,-0.5f,-0.5f);         // Top Right Of The Quad (Back)
          glVertex3f(-0.5f,-0.5f,-0.5f);         // Top Left Of The Quad (Back)
          glVertex3f(-0.5f, 0.5f,-0.5f);         // Bottom Left Of The Quad (Back)
          glVertex3f( 0.5f, 0.5f,-0.5f);         // Bottom Right Of The Quad (Back)

          glColor4f(1.0, 0.0, 1.0, 0.2);
          glVertex3f(-0.5f, 0.5f, 0.5f);         // Top Right Of The Quad (Left)
          glVertex3f(-0.5f, 0.5f,-0.5f);         // Top Left Of The Quad (Left)
          glVertex3f(-0.5f,-0.5f,-0.5f);         // Bottom Left Of The Quad (Left)
          glVertex3f(-0.5f,-0.5f, 0.5f);         // Bottom Right Of The Quad (Left)

          glColor4f(0.0, 0.5, 1.0, 0.2);
          glVertex3f( 0.5f, 0.5f,-0.5f);         // Top Right Of The Quad (Right)
          glVertex3f( 0.5f, 0.5f, 0.5f);         // Top Left Of The Quad (Right)
          glVertex3f( 0.5f,-0.5f, 0.5f);         // Bottom Left Of The Quad (Right)
          glVertex3f( 0.5f,-0.5f,-0.5f);         // Bottom Right Of The Quad (Right)
          }

          glEnd();
          glFlush();
          }
          }

          [self unlockFocus];

}

/*
Update point location
*/
-(IBAction) rotate:(float) x :(float) y :(id)sender
{

          @synchronized (self) {
          rotX +=x;
          rotY +=y;
          NSLog(@"rotate");
          [self drawRect:[self bounds]];
          }

}

@end
```

8.1.3 FingerPosition

```
//  Created by Patrick Holroyd
//  Copyright Patrick Holroyd 2008. All rights reserved.

/* FingerPosition */

#import <Cocoa/Cocoa.h>

@interface FingerPosition : NSOpenGLView
{
       float posX;
       float posY;

}

- (void) drawRect: (NSRect) bounds;
- (IBAction) move:(float) x :(float) y :(id)sender;

@end

//
//  FingerPosition.h
//  FingerPosition
//
//  Created by Patrick Holroyd
//  Copyright Patrick Holroyd 2008. All rights reserved.


#import "FingerPosition.h"
#include <OpenGL/gl.h>

@implementation FingerPosition

/*
OpenGL draw method
*/
-(void) drawRect: (NSRect) bounds
{
       //Lock drawing code
       if( [self lockFocusIfCanDraw] == YES ) {
       {
          //Setup OpenGL
          glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
          glShadeModel(GL_SMOOTH);
          glEnable(GL_DEPTH_TEST);
          glDepthFunc(GL_LEQUAL);
          glClearColor(0.0, 0.0, 0.0, 0.0);
          glClearDepth(1.0);
          glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
          glEnable (GL_BLEND);
          glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
          glLoadIdentity();
          glPointSize(10.0);
          glEnable(GL_POINT_SMOOTH);
          glHint(GL_POINT_SMOOTH_HINT, GL_NICEST);
          glEnable(GL_BLEND);
          glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

          //Move cube using result from tracking
          glTranslatef(posX, posY, 0.0);
          glBegin(GL_POINTS);
          {
                    glColor4f(1.0, 1.0, 1.0, 1.0);
                    glVertex3f( 0.0f, 0.0f, 0.0f);
          }

       glEnd();
       glFlush();
}
}
       [self unlockFocus];


}

/*
Update point location
*/
-(IBAction) move:(float) x :(float) y :(id)sender
{

          @synchronized (self) {
          posX =x;
          posY =y;
          [self drawRect:[self bounds]];
}
}


@end
```

8.2 3D Motion Tracking System

8.2.1 xyTracking

```
//  Created by Patrick Holroyd
//  Copyright Patrick Holroyd 2008. All rights reserved.

/* xyTracking */

#import <Cocoa/Cocoa.h>
#import <QTKit/QTkit.h>
#import <MyOpenGLView.h>
#import <ZTracking.h>
#import <FingerPosition.h>

@interface xyTracking : NSObject {

       IBOutlet NSImageView                                  *mImageView;
       IBOutlet NSTextField                                  *xtext;
       IBOutlet NSTextField                                  *ytext;
       IBOutlet NSTextField                                  *xdistext;
       IBOutlet NSTextField                                  *ydistext;
       IBOutlet NSTextField                                  *prextext;
       IBOutlet NSTextField                                  *preytext;
       IBOutlet NSTextField                                  *timetext;
       IBOutlet MyOpenGLView                                 *pOpenGLView;
       IBOutlet FingerPosition                               *pFingerPosition;
       IBOutlet ZTracking                                    *zTracking;
       IBOutlet QTCaptureView                                *mCaptureView;
       IBOutlet NSSlider                                               *slidersens;
       IBOutlet NSButton                                               *settings;
       IBOutlet NSButton                                               *capture;
       IBOutlet NSButton                                               *sliderup;
       IBOutlet NSButton                                               *sliderdown;
       IBOutlet NSSegmentedCell                                        *control;

       NSImage                                                         *image;
    QTCaptureSession                                                   *mCaptureSession;
       QTCaptureSession                                                *mCaptureSession2;
       QTCaptureVideoPreviewOutput
       *mCaptureDecompressedVideoOutput;
       QTCaptureVideoPreviewOutput
       *mCaptureDecompressedVideoOutput2;
    QTCaptureDeviceInput                                     *mCaptureVideoDeviceInput;
       QTCaptureDeviceInput                                  *mCaptureVideoDeviceInput2;
       CVImageBufferRef                                      mCurrentImageBuffer;
       CVImageBufferRef                                                imageBuffer;

       BOOL                                                            firsttrack;
       float                                                           pCOG_X;
       float                                                           pCOG_Y;
       float                                                           sensitivity;

}

- (void) MyInstanceThreadMethod:(CIImage *)source2;
- (IBAction) captureFrame:(id) sender;
- (IBAction) sliderup:(id) sender;
- (IBAction) sliderdown:(id) sender;

@end


//
```

```
//  xyTracking.h
//  xyTracking
//
//  Created by Patrick Holroyd
//  Copyright Patrick Holroyd 2008. All rights reserved.

#import "xyTracking.h"

@implementation xyTracking


/*
Initial wakeup method
*/
- (void)awakeFromNib
{

// Create the capture session

       mCaptureSession = [[QTCaptureSession alloc] init];

// Connect inputs and outputs to the session

       BOOL success = NO;
       BOOL success2 = NO;
       NSError *error;

// Find a video device

       NSArray *inputs = [QTCaptureDevice inputDevicesWithMediaType:QTMediaTypeVideo];
       QTCaptureDevice *videoDevice = [inputs objectAtIndex:0];
       QTCaptureDevice *videoDevice2 = [inputs objectAtIndex:1];

    success = [videoDevice open:&error];
       success2 = [videoDevice2 open:&error];


// If a video input device can't be found or opened, try to find and open a muxed input
device

       if (!success) {
          videoDevice = [QTCaptureDevice defaultInputDeviceWithMediaType:QTMediaTypeMuxed];
       success = [videoDevice open:&error];

    }

    if (!success) {
        videoDevice = nil;
        NSLog(@"No Camera 1");

   }

       if (!success2) {
        videoDevice2 = nil;
        NSLog(@"No Camera 0");

   }

//Add the video device to the session as a device input

       mCaptureVideoDeviceInput = [QTCaptureDeviceInput deviceInputWithDevice: videoDevice];
       mCaptureVideoDeviceInput2      =      [QTCaptureDeviceInput      deviceInputWithDevice:
videoDevice2];
       success = [mCaptureSession addInput:mCaptureVideoDeviceInput error:&error];

       if (!success)
       {
          //Handle error
          NSLog(@"Cannot add camera 1 to session");
        }

        [zTracking main:mCaptureVideoDeviceInput2];

        mCaptureDecompressedVideoOutput = [[QTCaptureVideoPreviewOutput alloc] init];
        [mCaptureDecompressedVideoOutput setDelegate:self];
        success = [mCaptureSession addOutput:mCaptureDecompressedVideoOutput error:&error];

        if (!success)
        {
                        //Handle error
                        NSLog(@"Output not added");
        }

        [mCaptureView setCaptureSession:mCaptureSession];

        firsttrack = YES;
        sensitivity = 1.5;
        [settings setKeyEquivalent:@"s"];
        [capture setKeyEquivalent:@"c"];
        [sliderup setKeyEquivalent:@"x"];
        [sliderdown setKeyEquivalent:@"z"];

        [mCaptureSession startRunning];
        [mCaptureSession2 startRunning];

}


/*
Thread method which applies filter, scans pixels and applying Centre of Mass algorithm
*/
- (void) MyInstanceThreadMethod:(CIImage *)source2;
{

        //Check if clicking
        if([zTracking clicking] < 20)
        {

        @synchronized (self) {
        NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

        [CIPlugIn loadAllPlugIns];
        CIFilter *cFilter = [[CIFilter alloc] init];

//Apply custom filter

        [pool addObject:cFilter];
        cFilter = [CIFilter filterWithName: @"ColorRemoveFilter"];
        [cFilter setDefaults];
        [cFilter setValue: source2 forKey: @"inputImage"];
        [cFilter setValue: [CIColor colorWithRed: 0.00390 green: 0.41406 blue: 0.29687 alpha:
1.0] forKey: @"matchColor"];
        [cFilter     setValue:     [NSNumber     numberWithFloat:     sensitivity]     forKey:
@"inputSensitivity"];
        CIImage *result = [cFilter valueForKey: @"outputImage"];

//Resize frame

        NSSize xsize = {320.0, 240.0};
        NSCIImageRep *imageRep = [NSCIImageRep imageRepWithCIImage:result];
        [imageRep retain];

        [pool addObject:imageRep];
        [image release];
        image = [[NSImage alloc] initWithSize:xsize];
        [image addRepresentation:imageRep];
        [image retain];
        [pool addObject:image];

        NSData *tiff_data = [[NSData alloc] initWithData:[image TIFFRepresentation]];
        [pool addObject:tiff_data];

        NSBitmapImageRep *bitmap = [[NSBitmapImageRep alloc] initWithData:tiff_data];
        [pool addObject:bitmap];

//Scan over pixels

        int x;
        int y;
        float COG_X;
        float COG_Y;
        float total;
        int totalGreen = 0;
        for(x = 0; x < 160; x++)
        {
        for (y = 0; y < 120; y++)
        {
           NSUInteger pixelData[3];
           [bitmap getPixel:pixelData atX:x*2 y:y*2];

           float k = (pixelData[0]+pixelData[1]+pixelData[2])/3;
           if(k == 61)
           {
                   totalGreen++;
           }

           COG_X = COG_X + (k*x*2);
           COG_Y = COG_Y + (k*y*2);
           total = total + k;
        }
        }

//Stop tracking if too much green is in frame

        BOOL track = YES;
        if(totalGreen > 1750)
        {
           firsttrack = YES;
           track = NO;
        }

//Stop tracking if too little green is in frame

        if(totalGreen < 75)
        {
           firsttrack = YES;
           track = NO;
        }
        if(track)
        {

        if(firsttrack)
        {

//If first tracking only store finger location

           COG_X = ((COG_X / total) * - 1) +320;
           pCOG_X = COG_X;
           COG_Y = COG_Y / total;
           pCOG_Y = COG_Y;
           firsttrack = NO;
        }
        else
        {

//If not first tracking apply distance equation
```

```
            COG_X = ((COG_X / total) * - 1) +320;
            float xdis = COG_X - pCOG_X;
            COG_Y = COG_Y / total;
            float ydis = COG_Y - pCOG_Y;

            //Rotate cube
            if([zTracking clicked] == NO)
            {
                      [control setSelectedSegment: 0];
                      //Map data to OpenGL cube
                      [pOpenGLView rotate:xdis :ydis :self];
            }

            //Move cube
            if([zTracking clicked] == YES)
            {
                      [control setSelectedSegment: 1];
                      //Map data to OpenGL cube
                      [pOpenGLView translate:(COG_X/160)-1 :((COG_Y/120)-1) * -1 :self];
            }

            //Map data to OpenGL finger position feedback
            [pFingerPosition move:(COG_X/160)-1 :((COG_Y/120)-1) * -1 :self];
            [prextext setFloatValue:(COG_X/160)-1];
            [preytext setFloatValue:(COG_Y/120)-1];
            [xtext setFloatValue:COG_X];
            [ytext setFloatValue:COG_Y];
            [xdistext setFloatValue:COG_X];
            [ydistext setFloatValue:COG_Y];

            pCOG_X = COG_X;
            pCOG_Y = COG_Y;

            }
            }

        [pool release];

        }
        }
}

/*
Capture frame for GUI
*/
- (IBAction) captureFrame:(id) sender
{
       [mImageView setImage:image];
}

/*
Increase sensitivity
*/
- (IBAction) sliderup:(id) sender
{
       sensitivity = sensitivity - 0.1;
       [mCaptureSession stopRunning];
       [mCaptureSession startRunning];
       [self captureFrame:self];
}

/*
Decrease sensitivity
*/
- (IBAction) sliderdown:(id) sender
{
       sensitivity = sensitivity + 0.1;
       [mCaptureSession stopRunning];
       [mCaptureSession startRunning];
       [self captureFrame:self];
}

/*
Capture frame from xyCamera
*/
-                     (void)captureOutput:(QTCaptureOutput                    *)captureOutput
didOutputVideoFrame:(CVImageBufferRef)videoFrame            withSampleBuffer:(QTSampleBuffer
*)sampleBuffer fromConnection:(QTCaptureConnection *)connection
{

    //Store the latest frame
       CVImageBufferRef imageBufferToRelease;
    CVBufferRetain(videoFrame);

       @synchronized (self)
       {
           imageBufferToRelease = mCurrentImageBuffer;
           mCurrentImageBuffer = videoFrame;
       }

    @synchronized (self)
       {
         imageBuffer = CVBufferRetain(mCurrentImageBuffer);
    }
       if (imageBuffer)
       {
           CIImage *source = [CIImage imageWithCVImageBuffer:imageBuffer];

//Dispatch thread

           [self MyInstanceThreadMethod:source];
       }

       CVBufferRelease(imageBufferToRelease);
       CVBufferRelease(mCurrentImageBuffer);

}

#pragma mark-

/*
Handle window closing notifications for device inputs
*/
- (void)windowWillClose:(NSNotification *)notification
{

       [mCaptureSession stopRunning];
    if ([[mCaptureVideoDeviceInput device] isOpen])
       {
         [[mCaptureVideoDeviceInput device] close];
       }
       if ([[mCaptureVideoDeviceInput2 device] isOpen])
       {
         [[mCaptureVideoDeviceInput2 device] close];
       }

}

/*
Handle deallocation of memory for your capture objects
*/
- (void)dealloc
{
       [mCaptureSession release];
       [mCaptureVideoDeviceInput release];
    [mCaptureDecompressedVideoOutput release];
       [super dealloc];
```

```
}

#pragma mark-

@end
```

8.2.2 ZTracking

```
//  Created by Patrick Holroyd
//  Copyright Patrick Holroyd 2008. All rights reserved.

/* ZTracking */

#import <Cocoa/Cocoa.h>
#import <QTKit/QTkit.h>
#import <MyOpenGLView.h>

@interface ZTracking : NSObject {

       IBOutlet NSImageView                               *mImageView2;
       IBOutlet NSTextField                               *ztext;
       IBOutlet NSTextField                               *zdistext;
       IBOutlet NSTextField                               *preztext;
       IBOutlet NSTextField                               *click;
       IBOutlet NSTextField                               *greenclick;
       IBOutlet QTCaptureView                             *mCaptureView2;
       IBOutlet NSButton                                            *sliderup;
       IBOutlet NSButton                                            *sliderdown;

    QTCaptureSession                                                *mCaptureSession2;
       QTCaptureVideoPreviewOutput
       *mCaptureDecompressedVideoOutput2;
       QTCaptureDeviceInput                               *mCaptureVideoDeviceInput2;
       NSImage                                                      *image2;
       CVImageBufferRef                                   mCurrentImageBuffer2;
       CVImageBufferRef                                             imageBuffer;

       BOOL                                                         firsttrack;
       BOOL                                                         clicked;
       BOOL                                                         reset;
       float                                                        sensitivity;
       float                                                        pCOG_Z;
       float                                                        zdis2;
       int                                                totalGreen;
       int                                                globaltotalGreen;

}

- (void) MyInstanceThreadMethod:(CIImage *)source2;
- (void)main:(QTCaptureDeviceInput *)zVideoDevice;
-                     (void)captureOutput:(QTCaptureOutput                    *)captureOutput
didOutputVideoFrame:(CVImageBufferRef)videoFrame            withSampleBuffer:(QTSampleBuffer
*)sampleBuffer fromConnection:(QTCaptureConnection *)connection;
- (int)clicking;
- (BOOL)clicked;
- (IBAction) captureFrame2:(id) sender;
- (IBAction) sliderup2:(id) sender;
- (IBAction) sliderdown2:(id) sender;

@end

//
//  xyTracking.h
//  xyTracking
```

```
//
//  Created by Patrick Holroyd
//  Copyright Patrick Holroyd 2008. All rights reserved.

#import "xyTracking.h"

@implementation xyTracking


/*
Initial wakeup method
*/
- (void)awakeFromNib
{

// Create the capture session

      mCaptureSession = [[QTCaptureSession alloc] init];

// Connect inputs and outputs to the session

      BOOL success = NO;
      BOOL success2 = NO;
      NSError *error;

// Find a video device

      NSArray *inputs = [QTCaptureDevice inputDevicesWithMediaType:QTMediaTypeVideo];
      QTCaptureDevice *videoDevice = [inputs objectAtIndex:0];
      QTCaptureDevice *videoDevice2 = [inputs objectAtIndex:1];

    success = [videoDevice open:&error];
      success2 = [videoDevice2 open:&error];


// If a video input device can't be found or opened, try to find and open a muxed input
device

      if (!success) {
         videoDevice = [QTCaptureDevice defaultInputDeviceWithMediaType:QTMediaTypeMuxed];
      success = [videoDevice open:&error];

    }

    if (!success) {
       videoDevice = nil;
       NSLog(@"No Camera 1");

   }

       if (!success2) {
       videoDevice2 = nil;
       NSLog(@"No Camera 0");

   }

//Add the video device to the session as a device input

      mCaptureVideoDeviceInput = [QTCaptureDeviceInput deviceInputWithDevice: videoDevice];
      mCaptureVideoDeviceInput2      =      [QTCaptureDeviceInput      deviceInputWithDevice:
videoDevice2];
      success = [mCaptureSession addInput:mCaptureVideoDeviceInput error:&error];

      if (!success)
      {
         //Handle error
         NSLog(@"Cannot add camera 1 to session");
      }
```

```
       [zTracking main:mCaptureVideoDeviceInput2];

       mCaptureDecompressedVideoOutput = [[QTCaptureVideoPreviewOutput alloc] init];
       [mCaptureDecompressedVideoOutput setDelegate:self];
       success = [mCaptureSession addOutput:mCaptureDecompressedVideoOutput error:&error];

       if (!success)
       {
                    //Handle error
                    NSLog(@"Output not added");
       }

       [mCaptureView setCaptureSession:mCaptureSession];

       firsttrack = YES;
       sensitivity = 1.5;
       [settings setKeyEquivalent:@"s"];
       [capture setKeyEquivalent:@"c"];
       [sliderup setKeyEquivalent:@"x"];
       [sliderdown setKeyEquivalent:@"z"];

       [mCaptureSession startRunning];
       [mCaptureSession2 startRunning];

}


/*
Thread method which applies filter, scans pixels and applying Centre of Mass algorithm
*/
- (void) MyInstanceThreadMethod:(CIImage *)source2;
{

       //Check if clicking
       if([zTracking clicking] < 20)
       {

       @synchronized (self) {
       NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

       [CIPlugIn loadAllPlugIns];
       CIFilter *cFilter = [[CIFilter alloc] init];

//Apply custom filter

       [pool addObject:cFilter];
       cFilter = [CIFilter filterWithName: @"ColorRemoveFilter"];
       [cFilter setDefaults];
       [cFilter setValue: source2 forKey: @"inputImage"];
       [cFilter setValue: [CIColor colorWithRed: 0.00390 green: 0.41406 blue: 0.29687 alpha:
1.0] forKey: @"matchColor"];
       [cFilter     setValue:     [NSNumber     numberWithFloat:     sensitivity]     forKey:
@"inputSensitivity"];
       CIImage *result = [cFilter valueForKey: @"outputImage"];

//Resize frame

       NSSize xsize = {320.0, 240.0};
       NSCIImageRep *imageRep = [NSCIImageRep imageRepWithCIImage:result];
       [imageRep retain];

       [pool addObject:imageRep];
       [image release];
       image = [[NSImage alloc] initWithSize:xsize];
       [image addRepresentation:imageRep];
       [image retain];
       [pool addObject:image];
```

```
       NSData *tiff_data = [[NSData alloc] initWithData:[image TIFFRepresentation]];
       [pool addObject:tiff_data];

       NSBitmapImageRep *bitmap = [[NSBitmapImageRep alloc] initWithData:tiff_data];
       [pool addObject:bitmap];

//Scan over pixels

       int x;
       int y;
       float COG_X;
       float COG_Y;
       float total;
       int totalGreen = 0;
       for(x = 0; x < 160; x++)
       {
       for (y = 0; y < 120; y++)
       {
          NSUInteger pixelData[3];
          [bitmap getPixel:pixelData atX:x*2 y:y*2];

          float k = (pixelData[0]+pixelData[1]+pixelData[2])/3;
          if(k == 61)
          {
                    totalGreen++;
          }

          COG_X = COG_X + (k*x*2);
          COG_Y = COG_Y + (k*y*2);
          total = total + k;
       }
       }

//Stop tracking if too much green is in frame

       BOOL track = YES;
       if(totalGreen > 1750)
       {
          firsttrack = YES;
          track = NO;
       }

//Stop tracking if too little green is in frame

       if(totalGreen < 75)
       {
          firsttrack = YES;
          track = NO;
       }
       if(track)
       {

       if(firsttrack)
       {

//If first tracking only store finger location

          COG_X = ((COG_X / total) * - 1) +320;
          pCOG_X = COG_X;
          COG_Y = COG_Y / total;
          pCOG_Y = COG_Y;
          firsttrack = NO;
       }
       else
       {

//If not first tracking apply distance equation

          COG_X = ((COG_X / total) * - 1) +320;
```

```
            float xdis = COG_X - pCOG_X;
            COG_Y = COG_Y / total;
            float ydis = COG_Y - pCOG_Y;

            //Rotate cube
            if([zTracking clicked] == NO)
            {
                        [control setSelectedSegment: 0];
                        //Map data to OpenGL cube
                        [pOpenGLView rotate:xdis :ydis :self];
            }

            //Move cube
            if([zTracking clicked] == YES)
            {
                        [control setSelectedSegment: 1];
                        //Map data to OpenGL cube
                        [pOpenGLView translate:(COG_X/160)-1 :((COG_Y/120)-1) * -1 :self];
            }

            //Map data to OpenGL finger position feedback
            [pFingerPosition move:(COG_X/160)-1 :((COG_Y/120)-1) * -1 :self];
            [prextext setFloatValue:(COG_X/160)-1];
            [preytext setFloatValue:(COG_Y/120)-1];
            [xtext setFloatValue:COG_X];
            [ytext setFloatValue:COG_Y];
            [xdistext setFloatValue:COG_X];
            [ydistext setFloatValue:COG_Y];

            pCOG_X = COG_X;
            pCOG_Y = COG_Y;

            }
            }

        [pool release];

        }
        }
}

/*
Capture frame for GUI
*/
- (IBAction) captureFrame:(id) sender
{
        [mImageView setImage:image];
}

/*
Increase sensitivity
*/
- (IBAction) sliderup:(id) sender
{
        sensitivity = sensitivity - 0.1;
        [mCaptureSession stopRunning];
        [mCaptureSession startRunning];
        [self captureFrame:self];
}

/*
Decrease sensitivity
*/
- (IBAction) sliderdown:(id) sender
{
        sensitivity = sensitivity + 0.1;
        [mCaptureSession stopRunning];
        [mCaptureSession startRunning];
        [self captureFrame:self];
```

```
}

/*
Capture frame from xyCamera
*/
-                    (void)captureOutput:(QTCaptureOutput                    *)captureOutput
didOutputVideoFrame:(CVImageBufferRef)videoFrame            withSampleBuffer:(QTSampleBuffer
*)sampleBuffer fromConnection:(QTCaptureConnection *)connection
{

    //Store the latest frame
       CVImageBufferRef imageBufferToRelease;
    CVBufferRetain(videoFrame);

       @synchronized (self)
       {
           imageBufferToRelease = mCurrentImageBuffer;
           mCurrentImageBuffer = videoFrame;
       }

    @synchronized (self)
       {
        imageBuffer = CVBufferRetain(mCurrentImageBuffer);
    }
       if (imageBuffer)
       {
           CIImage *source = [CIImage imageWithCVImageBuffer:imageBuffer];

//Dispatch thread

           [self MyInstanceThreadMethod:source];
       }

       CVBufferRelease(imageBufferToRelease);
       CVBufferRelease(mCurrentImageBuffer);

}

#pragma mark-

/*
Handle window closing notifications for device inputs
*/
- (void)windowWillClose:(NSNotification *)notification
{

       [mCaptureSession stopRunning];
    if ([[mCaptureVideoDeviceInput device] isOpen])
       {
        [[mCaptureVideoDeviceInput device] close];
       }
       if ([[mCaptureVideoDeviceInput2 device] isOpen])
       {
        [[mCaptureVideoDeviceInput2 device] close];
       }

}

/*
Handle deallocation of memory for your capture objects
*/
- (void)dealloc
{
       [mCaptureSession release];
       [mCaptureVideoDeviceInput release];
    [mCaptureDecompressedVideoOutput release];
       [super dealloc];
}
```

```
#pragma mark-

@end
```

8.2.3 MyOpenGLView

```
//  Created by Patrick Holroyd
//  Copyright Patrick Holroyd 2008. All rights reserved.

/* MyOpenGLView */

#import <Cocoa/Cocoa.h>

@interface MyOpenGLView : NSOpenGLView
{
       float rotX;
       float rotY;
       float posX;
       float posY;
}

- (void) drawRect: (NSRect) bounds ;
- (IBAction) rotate:(float) x :(float) y :(id)sender;
- (IBAction) translate:(float) x :(float) y :(id)sender;

@end

//
//  MyOpenGLView.h
//  MyOpenGLView
//
//  Created by Patrick Holroyd
//  Copyright Patrick Holroyd 2008. All rights reserved.

#import "MyOpenGLView.h"
#include <OpenGL/gl.h>

@implementation MyOpenGLView


/*
OpenGL draw method
*/
-(void) drawRect: (NSRect) bounds
{
       //Lock drawing code
       if( [self lockFocusIfCanDraw] == YES ) {
       {

       //Setup OpenGL
       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
       glShadeModel(GL_SMOOTH);
       glEnable(GL_DEPTH_TEST);
       glDepthFunc(GL_LEQUAL);
       glClearColor(0.0, 0.0, 0.0, 0.0);
       glClearDepth(1.0);
       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
       glEnable (GL_BLEND);
       glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

       glPointSize(3.0);
       glEnable(GL_POINT_SMOOTH);
       glHint(GL_POINT_SMOOTH_HINT, GL_NICEST);
       glEnable(GL_BLEND);
       glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

```
      glLoadIdentity();

      //Move cube using result from tracking
      glTranslatef(posX, posY, 0.0);
      //Rotate cube using result from tracking
      glRotatef(rotX,0.0f,1.0f,0.0f);
      glRotatef(rotY,1.0f,0.0f,0.0f);
      glBegin(GL_POINTS);
      {

         glColor4f(1.0, 1.0, 1.0, 1.0);
         glVertex3f( 0.5f, 0.5f,-0.5f);          // Top Right Of The Quad (Top)
         glVertex3f(-0.5f, 0.5f,-0.5f);          // Top Left Of The Quad (Top)
         glVertex3f(-0.5f, 0.5f, 0.5f);          // Bottom Left Of The Quad (Top)
         glVertex3f( 0.5f, 0.5f, 0.5f);          // Bottom Right Of The Quad (Top)

         glColor4f(1.0, 1.0, 1.0, 1.0);
         glVertex3f( 0.5f,-0.5f, 0.5f);          // Top Right Of The Quad (Bottom)
         glVertex3f(-0.5f,-0.5f, 0.5f);          // Top Left Of The Quad (Bottom)
         glVertex3f(-0.5f,-0.5f,-0.5f);          // Bottom Left Of The Quad (Bottom)
         glVertex3f( 0.5f,-0.5f,-0.5f);          // Bottom Right Of The Quad (Bottom)

         glColor4f(1.0, 1.0, 1.0, 1.0);
         glVertex3f( 0.5f, 0.5f, 0.5f);          // Top Right Of The Quad (Front)
         glVertex3f(-0.5f, 0.5f, 0.5f);          // Top Left Of The Quad (Front)
         glVertex3f(-0.5f,-0.5f, 0.5f);          // Bottom Left Of The Quad (Front)
         glVertex3f( 0.5f,-0.5f, 0.5f);          // Bottom Right Of The Quad (Front)

         glColor4f(1.0, 1.0, 1.0, 1.0);
         glVertex3f( 0.5f,-0.5f,-0.5f);          // Top Right Of The Quad (Back)
         glVertex3f(-0.5f,-0.5f,-0.5f);          // Top Left Of The Quad (Back)
         glVertex3f(-0.5f, 0.5f,-0.5f);          // Bottom Left Of The Quad (Back)
         glVertex3f( 0.5f, 0.5f,-0.5f);          // Bottom Right Of The Quad (Back)

         glColor4f(1.0, 1.0, 1.0, 1.0);
         glVertex3f(-0.5f, 0.5f, 0.5f);          // Top Right Of The Quad (Left)
         glVertex3f(-0.5f, 0.5f,-0.5f);          // Top Left Of The Quad (Left)
         glVertex3f(-0.5f,-0.5f,-0.5f);          // Bottom Left Of The Quad (Left)
         glVertex3f(-0.5f,-0.5f, 0.5f);          // Bottom Right Of The Quad (Left)

         glColor4f(1.0, 1.0, 1.0, 1.0);
         glVertex3f( 0.5f, 0.5f,-0.5f);          // Top Right Of The Quad (Right)
         glVertex3f( 0.5f, 0.5f, 0.5f);          // Top Left Of The Quad (Right)
         glVertex3f( 0.5f,-0.5f, 0.5f);          // Bottom Left Of The Quad (Right)
         glVertex3f( 0.5f,-0.5f,-0.5f);          // Bottom Right Of The Quad (Right)
         }

      glEnd();


         glBegin(GL_QUADS);
         {
                   glColor4f(0.0, 0.0, 1.0, 0.2);
                   glVertex3f( 0.5f, 0.5f,-0.5f);          // Top Right Of The Quad (Top)
                   glVertex3f(-0.5f, 0.5f,-0.5f);          // Top Left Of The Quad (Top)
                   glVertex3f(-0.5f, 0.5f, 0.5f);          //  Bottom  Left  Of  The  Quad
(Top)
                   glVertex3f( 0.5f, 0.5f, 0.5f);          //  Bottom  Right  Of  The  Quad
(Top)

                   glColor4f(0.0, 0.0, 1.0, 0.2);
                   glVertex3f( 0.5f,-0.5f, 0.5f);          //  Top  Right  Of  The  Quad
(Bottom)
                   glVertex3f(-0.5f,-0.5f, 0.5f);          //  Top  Left  Of  The  Quad
(Bottom)
                   glVertex3f(-0.5f,-0.5f,-0.5f);          //  Bottom  Left  Of  The  Quad
(Bottom)
                   glVertex3f( 0.5f,-0.5f,-0.5f);          //  Bottom  Right  Of  The  Quad
(Bottom)
```

```
                    glColor4f(0.0,1.0, 0.0, 0.2);
                    glVertex3f( 0.5f, 0.5f, 0.5f);          //  Top  Right  Of  The  Quad
(Front)
                    glVertex3f(-0.5f, 0.5f, 0.5f);          // Top Left Of The Quad (Front)
                    glVertex3f(-0.5f,-0.5f, 0.5f);          //  Bottom  Left  Of  The  Quad
(Front)
                    glVertex3f( 0.5f,-0.5f, 0.5f);          //  Bottom  Right  Of  The  Quad
(Front)

                    glColor4f(0.0, 0.0, 1.0, 0.2);
                    glVertex3f( 0.5f,-0.5f,-0.5f);          // Top Right Of The Quad (Back)
                    glVertex3f(-0.5f,-0.5f,-0.5f);          // Top Left Of The Quad (Back)
                    glVertex3f(-0.5f, 0.5f,-0.5f);          //  Bottom  Left  Of  The  Quad
(Back)
                    glVertex3f( 0.5f, 0.5f,-0.5f);          //  Bottom  Right  Of  The  Quad
(Back)

                    glColor4f(1.0, 0.0, 1.0, 0.2);
                    glVertex3f(-0.5f, 0.5f, 0.5f);          // Top Right Of The Quad (Left)
                    glVertex3f(-0.5f, 0.5f,-0.5f);          // Top Left Of The Quad (Left)
                    glVertex3f(-0.5f,-0.5f,-0.5f);          //  Bottom  Left  Of  The  Quad
(Left)
                    glVertex3f(-0.5f,-0.5f, 0.5f);          //  Bottom  Right  Of  The  Quad
(Left)

                    glColor4f(0.0, 0.5, 1.0, 0.2);
                    glVertex3f( 0.5f, 0.5f,-0.5f);          //  Top  Right  Of  The  Quad
(Right)
                    glVertex3f( 0.5f, 0.5f, 0.5f);          // Top Left Of The Quad (Right)
                    glVertex3f( 0.5f,-0.5f, 0.5f);          //  Bottom  Left  Of  The  Quad
(Right)
                    glVertex3f( 0.5f,-0.5f,-0.5f);          //  Bottom  Right  Of  The  Quad
(Right)
          }

          glEnd();
          glFlush();

          }
          }

          [self unlockFocus];

}

/*
Update amount to rotate cube
*/
-(IBAction) rotate:(float) x :(float) y :(id)sender
{
          @synchronized (self)
          {
                    rotX +=x;
                    rotY +=y;
                    [self drawRect:[self bounds]];
          }
}

/*
Update amount to move cube
*/
-(IBAction) translate:(float) x :(float) y :(id)sender
{
       @synchronized (self)
       {
          posX =x;
          posY =y;
          [self drawRect:[self bounds]];
```

```
       }
}

@end
```

8.2.4 FingerPosition

```
//  Created by Patrick Holroyd
//  Copyright Patrick Holroyd 2008. All rights reserved.

/* FingerPosition */

#import <Cocoa/Cocoa.h>

@interface FingerPosition : NSOpenGLView
{
       float posX;
       float posY;
}

- (void) drawRect: (NSRect) bounds;
- (IBAction) move:(float) x :(float) y :(id)sender;

@end

//
//  FingerPosition.h
//  FingerPosition
//
//  Created by Patrick Holroyd
//  Copyright Patrick Holroyd 2008. All rights reserved.

#import "FingerPosition.h"
#include <OpenGL/gl.h>

@implementation FingerPosition

/*
OpenGL draw method
*/
-(void) drawRect: (NSRect) bounds
{
       //Lock drawing code
       if( [self lockFocusIfCanDraw] == YES ) {
       {
          //Setup OpenGL
          glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
          glShadeModel(GL_SMOOTH);
          glEnable(GL_DEPTH_TEST);
          glDepthFunc(GL_LEQUAL);
          glClearColor(0.0, 0.0, 0.0, 0.0);
          glClearDepth(1.0);
          glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
          glEnable (GL_BLEND);
          glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
          glLoadIdentity();
          glPointSize(10.0);
          glEnable(GL_POINT_SMOOTH);
          glHint(GL_POINT_SMOOTH_HINT, GL_NICEST);
          glEnable(GL_BLEND);
          glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

          //Move cube using result from tracking
          glTranslatef(posX, posY, 0.0);
          glBegin(GL_POINTS);
          {
                    glColor4f(1.0, 1.0, 1.0, 1.0);
                    glVertex3f( 0.0f, 0.0f, 0.0f);
          }

       glEnd();
       glFlush();
}
}
       [self unlockFocus];
}


/*
Update point location
*/
-(IBAction) move:(float) x :(float) y :(id)sender
{
          @synchronized (self)
          {
          posX =x;
          posY =y;
          [self drawRect:[self bounds]];
          }
}

@end
```

8.3 Custom Filter

```
float distHue(vec4 pix, vec4 color)
{
      float diff = abs(pix.r - color.r);
      diff = diff > 3. ? 6. - diff : diff;
      return diff/3.0;
}

float distSaturation(vec4 pix, vec4 color)
{
      float diff = abs(pix.g - color.g);
      return diff;
}

vec4 rgb2hsv(vec4 rgb)
{
      float X = min(rgb.r, min(rgb.g, rgb.b));
      float V = max(rgb.r, max(rgb.g, rgb.b));

      float f =     (rgb.r == X) ?      rgb.g - rgb.b :
                    (rgb.g == X) ?      rgb.b - rgb.r :
                                                  rgb.r - rgb.g;
      float i =     (rgb.r == X) ?      3.0 :
                    (rgb.g == X) ?      5.0 :
                                                  1.0;

      float H = (V == X) ? 0. : i - f / (V - X);
      float S = (V == X) ? 0. : (V - X) / V;

      return vec4(H, S, V, 1.0);
}

kernel vec4 multiplyEffect(sampler rgbImage, __color matchColor, float sensitivity)
{
      vec4 rgb = sample(rgbImage, samplerCoord(rgbImage));
      vec4 color = unpremultiply(matchColor);

      vec4 hsv = rgb2hsv(rgb);
      vec4 colorhsv = rgb2hsv(color);
      float comp = distHue(hsv, colorhsv);
      float comp2 = distSaturation(hsv, colorhsv);
      comp = sqrt(comp*comp + comp2*comp2) - (1.0)/sensitivity + abs(hsv.b - 0.5);

      vec4 outpix = compare( vec4(comp, comp, comp, 1.0),
                                                  matchColor,
                                                  vec4(0, 0, 0, 1.0));

      return outpix;

}

//
//  ColorRemoveFilter.h
//  ColorRemove
//
//  Created by Patrick Holroyd
//  Copyright Patrick Holroyd 2008. All rights reserved.

#import <Foundation/Foundation.h>
#import <QuartzCore/QuartzCore.h>

//interface that specifies the filter inputs (input parameter must be in same order as
specified filter)
@interface ColorRemoveFilter : CIFilter
{
    CIImage      *inputImage;
```

```
      CIColor                  *matchColor;
    NSNumber     *inputSensitivity;
}

@end

//
//  ColorRemoveFilter.m
//  ColorRemove
//
//  Created by Patrick Holroyd.
//  Copyright Patrick Holroyd 2008. All rights reserved.
//

#import "ColorRemoveFilter.h"
#import <Foundation/Foundation.h>
#import <ApplicationServices/ApplicationServices.h>

@implementation ColorRemoveFilter

//check if CIKernel is already initialised
static CIKernel *ColorRemoveFilterKernel = nil;

- (id)init
{
    if(ColorRemoveFilterKernel == nil)
    {
          //return bundle that loads the CIFilter class
          NSBundle                                    *bundle          =          [NSBundle
bundleForClass:NSClassFromString(@"ColorRemoveFilter")];
          NSString               *code    =    [NSString    stringWithContentsOfFile:[bundle
pathForResource:@"ColorRemoveFilterKernel" ofType:@"cikernel"]];
          //store .cikernel files into an array
          NSArray     *kernels = [CIKernel kernelsWithString:code];
          //initialise .cikernel files stored in the array
          ColorRemoveFilterKernel = [[kernels objectAtIndex:0] retain];
    }
    return [super init];
}

//set filter attributes
- (NSDictionary *)customAttributes
{
    return [NSDictionary dictionaryWithObjectsAndKeys:

        [NSDictionary dictionaryWithObjectsAndKeys:
                    [CIColor  colorWithRed:  0.00784  green:  0.36078  blue:  0.25882  alpha:
1.0],

      kCIAttributeDefault,
           nil],                                @"matchColor",

        [NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithDouble:   0.0], kCIAttributeMin,
                    [NSNumber numberWithDouble:   5.0], kCIAttributeMax,
            [NSNumber numberWithDouble: 2.148], kCIAttributeDefault,
            [NSNumber numberWithDouble:   0.0], kCIAttributeIdentity,
           kCIAttributeTypeScalar,              kCIAttributeType,
           nil],                                @"inputSensitivity",

        nil];
}

//create CIImage object for each inputImage and apply kernel method
- (CIImage *)outputImage
{

    CISampler *src = [CISampler samplerWithImage: inputImage];

      return  [self  apply:  ColorRemoveFilterKernel,  src,  matchColor,  inputSensitivity,
kCIApplyOptionDefinition, [src definition], nil];
}

@end
```

9. Appendix B

9.1 Questionnaire

Evaluation Questionnaire

Level of Control

1. How easy was it to control the 3D cube with the coloured thimble?

   ❑ Very Easy
   ❑ Fairly Easy
   ❑ Fairly Hard
   ❑ Very Hard
   ❑ Impossible

2. To what level of control did you feel you had over the 3D cube?

   ❑ 0% (no control)
   ❑ 25%
   ❑ 50%
   ❑ 75%
   ❑ 100% (full control)

3. At what level of accuracy do you feel the 3D cube is following you finger movements?

   ❑ 0% (no accuracy)
   ❑ 25%
   ❑ 50%
   ❑ 75%
   ❑ 100% (full accuracy)

4. How easy did you find it to click? (3D Motion Capture System only)

   ❑ Very Easy
   ❑ Fairly Easy
   ❑ Fairly Hard
   ❑ Very Hard
   ❑ Impossible

Graphical User Interface

5. Rate the usefulness of the features included within the Graphical User Interface?

   ❑ Very Useful
   ❑ Fairly Useful
   ❑ Fairly Useless
   ❑ Very Useless

6. How easy did you find it to adjust the sensitivity of the filter?

❑ Very Easy
❑ Fairly Easy
❑ Fairly Hard
❑ Very Hard
❑ Impossible

Feedback

7. To what level did you find the 3D cube best represents movement?

❑ 0% (no representation)
❑ 25%
❑ 50%
❑ 75%
❑ 100% (full representation)

8. Rate the usefulness of the finger position feedback window?

❑ Very Useful
❑ Fairly Useful
❑ Fairly Useless
❑ Very Useless

Other Features

9. What additions would you make to the system

____
____
____

10. What would you remove from the system

____
____
____

11. Overall how would you rate the system

❑ Excellent
❑ Very Good
❑ Good
❑ Poor

Date: _________
Initials _________