An investigation into the use of a webcam for use as an HID for desktop virtual reality applications.

Oliver Smith

M.Sc. Dissertation

May 2009

Northumbria University
School of Computing, Engineering and Information Sciences

ABSTRACT

Currently many examples of using face tracking as an HID can be seen online especially Youtube (Harrison 2008) but as yet no clear explanation or evaluation of these systems have been published, so this project will concentrate on recreating this type of system and evaluating it with respect to realism, usability, accuracy and robustness. A system was built using FTLib (Huang 2008*b*) and OpenGL (OpenGL 2009), using OpenCV(Intel Corporation 2009) for webcam interfacing. The system was then tested both experimentally using videos and for usability and realism using real user testing. It was concluded that the chosen face tracking algorithm would be suitable for use in a desktop virtual reality system with some tweaking.

DECLARATION

I certify that the work contained in this report is the sole work of the author except where indicated. All material that has been taken from other sources has been clearly acknowledged. Quotations from other sources have been clearly marked, using quotation marks or a block quote.

Signature:

Date:

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Preface

The main purpose of this dissertation is to choose and evaluate a suitable face tracking algorithm for use in a desktop pseudo virtual reality system. The idea of this project was initially conceived after seeing examples of head tracking using a Nintendo Wii remote (Lee 2007) however the use of a wii controller is very specific and not suitable for wider use. The use of a more generic device such as a webcam would enable the system to be used across a much wider range of applications. Until now much has been written on the methods of head tracking but few academic publications have been written on the application of this to desktop virtual reality applications. There has been some research into the application of general body motion within games (Wang et al. 2006) which concludes that it provides a new more immersive dimension to the experience and suggests further study encompassing the distance between webcam and body would be interesting. This project will investigate current methods used for face tracking, methods of collecting video from a webcam and how to use the output of these methods to manipulate a 3d scene to create a 3d effect by tracking head movement and altering the screen display accordingly. Currently many examples of this can be seen online especially Youtube (Harrison 2008) but as yet no clear explanation or evaluation of these systems has been published, so this project will concentrate on recreating this type of system and evaluating it with respect to realism, usability, accuracy and robustness.

1.2 Objectives and Report Outline

Preliminary work (see §A) identified the following objectives which are reported in this report:

- To review and determine appropriate technologies for tracking faces in video - This study takes the form of a literature survey (see chapter 2), refining the requirements for the face tracking system, surveying different algorithms and libraries and then deciding that a library developed at Carnegie Mellon University(Huang 2008*b*) was the most appropriate due to its simplicity and apparent past success.
- To investigate different methods of developing simple 3d interactive environments

including how to interface tracked movements with these environments - As with the selection of the face tracking system this investigation takes the form of a literature survey (see chapter 3), refining the requirements for the 3D display system, surveying different rendering methods and then deciding that OpenGL was the most appropriate despite its complexity due to its reliability and it being a well established and mature system.

- To develop a piece of software which uses face tracking to create a perception of a 3D display on a 2D screen - A system was built that uses the chosen face tracking library to create a parallax effect using OpenGL, the design and build documented in chapter 4 and testing in chapters 5 & 6).
- To evaluate different methods of testing desktop virtual reality software - This review forms part of the User testing documentation (chapter 6) and discusses the pros and cons of user testing and associated factors which can influence the effectiveness of such a strategy.
- To evaluate using user trials the software with respect to (chapter 6):
  - Realism
  - Usability
  - Accuracy
  - Robustness

1.3 Results and Conclusions

Analysis of the built system showed that the chosen face tracking is a suitable HID for a desktop VR system but does require further work on the reliability of the face tracking aspect. Testing results show that the accuracy of the tracking library is sometimes less than perfect (§5.2.2) and would benefit from further work, however despite this users rated the system to be effective in producing a more convincing parallax effect than that gained using a mouse as an HID.

2. REVIEW OF FACE TRACKING TECHNIQUES

2.1 Overview

One of the main technical challenges in this project is choosing a suitable method which tracks human faces effectively. Many demonstrations of face tracking have been seen recently in popular Internet culture using a variety of technologies including IR tracking with a Nintendo Wii Remote (Lee 2007) and more computationally complex methods like OpenCV (Harrison 2008), however no current evaluation of the best technique to use in a desktop virtual reality system has taken place. Also this is a fast developing field, new tracking algorithms are constantly emerging and every day more made possible by the increase in computing power available to even a modest home computer. Methods of face tracking using webcams are more attractive then using propriety hardware such as the Nintendo Wii Remote as they use a piece of generic hardware which is well supported by all major computer platforms and do not require such complex installation setup routines as using propriety hardware.

Before analyzing any methods it is important to note the difference between face recognition and face tracking. Face recognition refers to systems that detect the location of a face with an image whereas face tracking refers to the tracking of a face in a video stream.

In this chapter criteria for the selection of a face tracking library (§2.2) are defined and then various methods of tracking faces are examined highlighting the relative merits of each (§2.3). After some consideration the most suitable system was found to be a simple face tracking library developed at Carnegie Mellon University by Huang (2008*b*).

2.2 Criteria

The face tracking solution used in this project must conform to the following criteria.

- **Availability** - A large number of face recognition and tracking algorithms have been published, however many only exist in the public domain as sets of mathematical

functions which would require work to implement. Some solutions include source code, which would greatly ease the implementation of this part of the system.

- **Simplicity of Implementation** - this partly ties in with availability in that even a complex algorithm can be simple to implement if a library or source code is available, in this project unless significant advantages are found to an algorithms pre-coded solutions will be preferable.

- **Performance** - When tracking a face in a video stream it is important that the system can keep up with the video otherwise any 3d effect gained will be delayed and the effect destroyed. Most published methods contain benchmark data but with the rate of increase in computer power as fast as it is then an algorithms that was unfeasible when published my now be easily used on a modest machine. Performance can also be considered in three other areas:

  - **Accuracy** - Face detection and tracking can be implemented in many different ways, depending on the method used the accuracy varies. While accuracy can apply to how specifically the face can be defined it can also take into account if the algorithm is affected by other body parts or other parts of the image. When classifying errors in statistical decision processes results which are false positives/negatives are referred to as Type I or II errors, i.e where no faces are detected when there are faces, this is a Type I error, and detecting a face where there are no faces is a false negative error known as a Type II error (Kimball 1957).
  - **Reliability** - in the context of this project reliability refers to the rate of success the algorithm has, i.e. how many frames it is successfully able to detect a face in. As the face tracking will take place on a video stream it is important that the algorithm is able to maintain an adequate frame rate of detection. As with accuracy what is adequate remains unknown but ideally it will be as high as possible without compromising performance.
  - **Partial Face Detection** - Any webcam can only have a limited field of view thus on some occasions the face being tracked will be partially outside the field of view, when the user gets close to the camera this effect happens more easily. Therefore the chosen method of face tracking must be able to track partial faces for optimal usability. The camera which will be used in this project is the Logitech QuickCam®Fusion^{TM}which has a field of view of 78 °(Hardware Zone 2005) this is wider than the average webcam but it will be beneficial in this application as it allows a full face to be accommodated to a proximity of $\approx 30cm$.

$$\frac{Height}{Width} \equiv \frac{1+\sqrt{5}}{2} \tag{2.1}$$

Fig. 2.1: The Golden Ratio

2.3 Face Detection Methods

Now criteria, on which to assess possible algorithms for suitability for use in this project, have been established the suitability of a range of face tracking methods will be considered. Both face detection and face tracking methods will be considered together as face detection on a frame by frame basis can produce the same results. Also it should be noted that methods aimed at face detection could potentially be modified to search in the area where the face was detected in the previous frame thus improving performance and accuracy.

2.3.1 Edge Based Algorithms

The most primitive way to detect a face in an image is to look for edges within images, this has been the most long running method of detection, the first example being Sakai, Nagao & Kanade (1972) who presented a method of analysing line drawings derived from photographs, while no use in this project it defined the area of face detection research which remained somewhat stagnant until the 1990s, when computer power had increased to a level where complex photographic and video analysis became possible on a modest computer. More recently edge tracking is used for more advanced facial feature technologies such as Govindaraju (1996) who presented a technique which identifies each edge of the face discreetly and then attempts to fit edges to a face model using the golden ratio for an ideal face:

Govindaraju's (1996) method is significantly more complex than previous edge detection methods as it involves multiple steps involving techniques outside edge detection. The main stages in the process are:

1. **Detection** of edges within the image using the Marr-Hildreth edge operator which uses second order differentiation to find edges within the image, Gaussian smoothing is also used to remove noise (Hutton & Dowling 2005).

2. **Thinning** to remove small and potentially insignificant edges using a standard algorithm (Hjelmas & Low 2001).

3. **Spur removal** to remove all small branches from the main edges, these could have been caused by lines in the face, such as wrinkles, or background.

4. **Filtering** to remove non face components.

5. **Corner Detection** allows each edge of the face to be identified, so that each side and the hairline can be independently detected.

6. Lastly all edges are used to form possible face candidates each of which is evaluated against a cost function derived from the golden ratio (Equation 2.1) to determine which are faces.

For this application a reasonably framerate of correctly tracked faces is desirable however when this algorithm was benchmarked from a set of 60 self generated images, with 'complex' backgrounds, containing 90 faces there was a 76% success rate and an average of 2 false positives per image, while the success rate would be within acceptable boundaries the falso positives would make the system very unreliable if the view was to alter sparadically rarther than a smooth tracking motion which renders algorithms such as this unsuitable for this application. Edge detection can however be employed as part of a more complex system.

2.3.2 Colour Based Algorithms

Another low level method of tracking faces is to search for areas of skin colour within a frame of video, though a simple method this can be surprisingly effective. Cai & Goshtasby (1999) describe a simple implementation of face detection using a searching algorithm based on colour histograms for common skin tones. Fortunately skin tones fit into a very small area of the visible spectrum (Yang & Robertson 2000) and very few other items are the same colour making skin detection via colour quite robust, however some more work is then needed to determine that the skin is a face. Most implementations of colour based face tracking use it as one of many tests in a neural network for example Lin (2007).

2.3.3 Neural Network Based Algorithms

A neural network is a collection of nodes which form a network to perform many simple processes sequentially and simultaneously on the input data and provide an output, with an obvious analogy to the human nervous system (Haykin 1999). They are commonly used in image processing and are prevalent in current face tracking methods.

Neural networks have become the basis for many recent face detection algorithms including Lin (2007) who describes an efficient face detection scheme which can detect more than one face in colour images or video with complex backgrounds and variable illumination. The process is divided into two stages:

1. The first searches for a face by separating the image into low resolution triangles and searching them for colours found in faces.

2. The second stage implements a neural network which considers size, illumination, pose, expression. The network uses data gained from training using large data sets (1500 images) to assess the probability

This algorithm overcomes the issue of complex backgrounds by removing all parts of the image that are not of skins tones in the first stage. this allows the system achieve quite high performance in comparison to other methods, using simple background processed using a Pentium 4 3.0GHz system location of one face took 62.5ms which is fast enough to maintain a fast frame rate with video.

With regard to availability this method is unpractical as the full detail of the detection method is not described in enough detail to allow implementation rendering the system impractical for use here.

2.3.4 Other Algorithms

While all methods of tracking faces in video mentioned above have many different published variations there are some unique examples such as using the artifacts of MPEG video compression to determine the position of faces the only example of this currently published is Wang & Chang (1997) they describe a novel method which takes the inverse quantized discrete cosine transform (DCT) coefficients (Ahmed, Natarajan & Rao 1974) of MPEG video as the input, and outputs the locations of the detected face regions. As the DCT is the function which is used to compress MPEG video using coefficients from it mean that this algorithm can proceed very fast as uncompression of each frame is not required. The works in three stages, first analysing chrominance, then shape, and finally frequency information. While this method is designed to be focused on speed rather than accuracy in this case Wang & Chang (1997) has assumed a lack of accuracy to mean that only a rectangular bounding box is used for the face, this would not be a problem to if this algorithm was to be used. For testing of this system it was found that on a Sun SPARC 5 workstation that when a test stream of video containing multiple different faces sourced from CNN News was used the average time for detection was 32.6ms even detecting multiple faces. This would be fast enough for implementation in this project, however it would need to work to limit it to the detection of one face. This method is also limited to an MPEG or JPEG stream which limits the live input sources which could be used.

2.4 Face Detection Libraries

Up to now all of the potential face tracking systems discussed have been purely theoretical algorithms with no publically available implementation for this project it would simplify matters greatly if a system could be found which satisfied the criteria and was already

in a state where it was available to implement immediately without complex coding to implement an algorithm, therefore from now on only actual face tracking libraries will be considered.

2.4.1 Carnegie Mellon Face Tracking Library

This is a simple face tracking and detection library developed by Huang (2008*b*) based on the algorithm described in Huang & Chen (2000), it aims aims to be robust and accurate face tracking system running in real-time. Before face tracking can happen human input is needed to define a rectangular area in which the face is enclosed a face detection algorithm is then used to detect this and then in subsequent frames of the video a face tracking method returns co-ordinates of the rectangle. The algorithm works by analysing colour distribution between the face and background regions and assigning a deformable oval to areas of skin tones in face shape. A particularly beneficial feature of this library is it allows the user to specify the number of faces to track as it is vital in this application only one face is tracked. The library is available as a dynamic link library (dll) which would mean that the whole system would need to be written in C++ otherwise a wrapper would need to be implemented to enable the use with Java and other programming languages. The underlying algorithm has been shown to be sufficiently accurate (Huang & Chen 2000) achieving between 78-90% accuracy on live video streak with normal office background even when tracking heads which are partly obscured, this could be useful in this applicaiton as if the face being tracked was to move very close to the screen or towards the edge of the webcam then only a partial face would be visiable in the image available for tracking.

2.4.2 OpenCV

While the library discussed in §2.4.1 is specifically aimed at face tracking OpenCV is a general low level image processing library which included object tracking capabilities. The algorithm which provides this functionality in OpenCV is known as Camshift (Hewitt 2007) it is a colour based method which works in four stages:

1. To create a colour histogram showing the distribution of colours within the face being tracked.
2. A face probability is calculated for each pixel in the current video frame.
3. The rectangle containing the face is then shifted to the area with highest probability of a face.
4. Finally the size and angle of the rectangle is calculated.

This approach is very similar to that of §2.4.1 and while no official benchmarks have been published the reliability of such tracking features can be indicated by the use of this algorithm from its use in US Military related research (Turcsanski 2007).

OpenCV also includes face detection functionality which is able to, after training, detect faces based on a complex neural network based approach, the speed of this has again not been verified. There is however some colloquial evidence to suggest that it is prohibitively slow for real time face tracking (Podolsky & Frolov 2008). Before object tracking can take place training is required, the training can be carried out in advance to create a haar cascade. This consists of a file containing a decision tree ordered by complexity. A Haar Cascade (Papageorgiou, Oren & Poggio 1998) works in a very different way to compare images. First the image is split into larger sections and the intensity of the pixels summed, this reduces the computational power required when compared to analysisng every individual pixels. The image is then compared to the image sections from the training, this is achieved by summing the sections and a distribution of high sums across image sections indications a good match. Each frame/image is passed through if the face is very clear then the location will be returned quickly, if it is more complex i.e. if the background is very cluttered then it will take longer as many more steps through the tree will be required. this requires a large number of images both negative and positive. Two haar cascades for detection of frontal faces included with OpenCV which would ease the application of this method if it was deemed worthy. The OpenCV API is written in C but wrappers are available for Java (Cousot & Stanley 2007)and .NET (Huang 2008*a*) if they were needed.

2.4.3 Seeing Machines FaceAPI

A recent commercial face tracking solution is FaceAPI produced by Seeing Machines (Seeing Machines 2008) very little documentation about its internal workings are available, nor any real benchmarks the only way currently to assess its accuracy is by downloading the demo provided, this appears to detect a face on a complex background within several seconds and then is able to track it perfectly including facial features even in bad light conditions.

2.4.4 Torch3vision

This is a face detection library based on the Touch3 computer machine learning library (Marcel & Rodriguez 2007), it works on a very similar basis to OpenCV requiring training then using edge detection and colour detection where possible to determine a face position. However this library does not come with pretrained example for face tracking so would require extensive training with real faces which would be time consuming and

unnecessary when easier solutions are available. Also as with many experimental libraries documentation is sparse and performance not measured.

2.5 Conclusion

After examining these solutions for face tracking it seems most sensible to choose a pre-witten library. OpenCV, The Carnegie Mellon Face Tracking Library and FaceAPI would provide all the features needed for this project. Even when benchmarking data is provided it cannot easily be compared as the systems are tested in such a diverse range of environments with no common features, i.e. the algorithm seen in §2.3.4 is tested on a Sun SPARC machine and a good performance is obtained but testing of FaceAPI has occurred on computers at least 10 years newer in which time computing power has increased hugely so even algorithms which appear to be unsuitable because they perform too slowly may now perform fast enough.

The Carnegie Mellon Face Tracking Library can be deemed to be most suitable for this project as it provides a simple and robust way to track faces which does not require training other than initial definition of the head location, this allows for very simple tracking of different skin tones and allows for different lighting conditions to some extent as it is recalibrated every time it is used. While OpenCV also provides all the features needed it seems unnecessarily complex for this application. FaceAPI has too much ambiguity surrounding it for it to be considered as a viable option.

3. REVIEW OF 3D DISPLAY TECHNIQUES

3.1 Overview

Now that the method of tracking faces in video has been considered the next choice is to decide on the system which will be used to render the 3d scene which will be controlled by the webcam movements. Firstly the criteria for the system are defined A selection of commonly used methods has been reviewed (§3.2) and their merits highlighted after which Java3D and OpenGL were selected as appropriate solutions depending on the language used (§3.5). Other solutions such as Direct3D were disregarded as being too complex.

3.2 Criteria

The 3D rendering system used in this project must conform to the following criteria:

- The system must render in real time as the image is to be modified constantly depending on head position. Some systems which use techniques such as ray tracing can be very slow taking up to several seconds for each frame unless very powerful computer systems are used.
- The chosen system must be able to be interfaced easily with the face tracking library DLL thus it must be able to interface in C++, however this will be considered more in the next chapter. If the system was written in Java systems such as JNI (Sun Microsystems Inc. 2003) could be used to provide compatibility.
- The chosen rendering system does not need to be complex, the aim of this system is not to display hugely complex 3D scenes but display a simple 3D shape such as a cube and allow the perspective to be controlled in real time. Also the simpler the system the more likely it is to be fast and efficient.

3.3 3D Rendering Techniques

There are many different techniques which can be used for rendering 3D graphics on a computer display so here only the three most common techniques will be considered.

However one feature is common to all techniques that after a 3d scene is defined by its x,y,z coordinates or any other method then it must be projected to create a 2d image for display onto the computers display. This is usually achieved by using a simple geometrical perspective projection (see figure 3.1). This relates directly to how the 3d effect will be created in this system, i.e. by moving the position of the focal point.

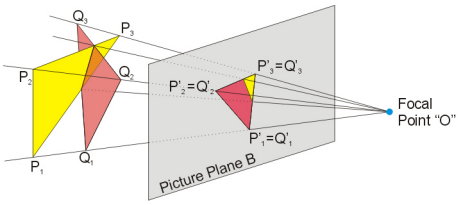


Fig. 3.1: Principle of geometrical perspective projection (Wikimedia 2007)

3.3.1 Rasterisation

Rasterisation is the process used by all current graphics cards to render 3d graphics, it is based on the idea that it is too slow for real time graphics to render scenes on a pixel by pixel basis thus the scene must be divided into larger sections. These segments are referred to as primitives, in a 3d scene they commonly consist of polygons which are used to build larger more complex shapes (Foley, Dam, Feiner & Hughes 1990). When the image is rendered projection is achieved by looping through each privative and drawing only the pixels required to display the primitives in contrast to a pixel by pixel approach where the whole image would be drawn regardless of if any objects were present in that part of the scene. This approach makes rasterisation much more computationally efficient, although recent optimisations to ray tracing algorithm may soon meantsn't the only viable option (Shrout 2008). A major system which uses rasterisation as it's main method of rendering is OpenGL which will be considered later (see §3.4.1)

3.3.2 Ray Tracing

Ray tracing is a pixel by pixel based rendered system which works in a more physically realistic way by considering the path of light from specified light sources and tracing rays, their reflections and shadows (as shown in figure 3.2). Due to the computational complexity ray tracing is not normally considered suitable for real time use, it finds more applications in realistic 3D rendering of films and still images (Glassner 1989). However with the advent of more powerful computers and increasingly efficient algorithms real time implementations of ray tracing have been developed such as OpenRT (OpenRT 2008) a version of OpenGL which uses ray tracing, this is however very much still in development

and is too immature to use in this system and also provides a great deal more complexity than is required.

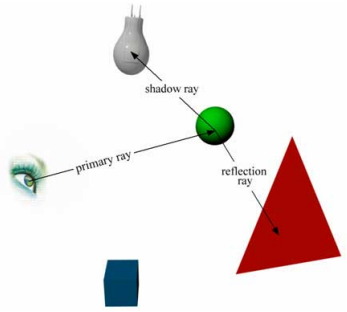


Fig. 3.2: Diagram of Ray Tracing (Takahashi 2009)

3.3.3 Ray Casting

Ray casting (Roth 1982) is similar to ray tracing however the effects of light reflecting off objects is not considered so it is not able to render shadows or reflections accurately without being used in combination with other methods to allow adding of shadows and reflections via texture maps. Development in this area has lagged behind that of ray tracing and rasterisation as they both provide simpler methods to render complex scenes with varying degrees of realism and speed. Ray casting will not be considered for use in this project as, although it meets speed requirements there are few readily available mature libraries.

3.3.4 Radiosity

Radiosity is a rendering algorithm developed for use in engineering heat transfer simulations (Goral, Torrance, Greenberg & Battaile 1983). It works in a similar way to ray tracing but is more specific about which rays it traces, choosing only rays that are reflected by objects after leaving the light source. The reflections are calculates on a property known as the view factor for each surface i.e how well each surface can see another. The algorithm is iterated until the scene is rendered. One major disadvantage of this method is that the rendering time increases quadratically with increasing surfaces unless texture mapping is used, however if radiosity based algorithms were used in this project this would not be a problem as the aim is not to produce a complex scene.

3.4 3D Rendering Libraries

Having examined the specific rendering algorithms to meet the needs of this project a rasterisation based library would seem most sensible as it is the method used by all common

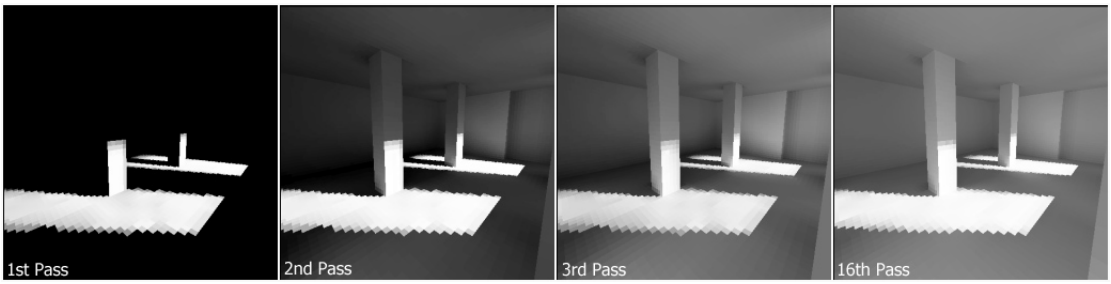


Fig. 3.3: The progressive rendering of a scene using radiosity (Elias 2007)

GPUs rather than other methods which would require CPU use to carry out the rendering. Also will be shown there are a great deal more options for simply implementing a rasterisation based system. On the windows platform there are two main low level graphics rendering libraries, OpenGL and Direct3D both of which have their relative merits.

3.4.1 OpenGL

OpenGL (OpenGL 2009) is a standard cross platform API used for creating 2D and 3D images. The system exists to hide the complexities of 3D accelerators from programmers and also to allow compatibility of code on different hardware. Because OpenGL is a low level API it requires exact specification of the rendered scene which introduces a degree of complexity, however as here only simple graphics are needed this will not be an issue, a simple 3d scene generated with OpenGL can be seen in figure 3.4 and the source code in §B.1. In its native state OpenGL requires the use of C/C++ which would be convenient as the chosen face tracking library also requires this, however if another programming language was to be used such as java there are simple methods for implementing the same functionality using systems such as JOGL (Davison 2007) (see §3.4.1).

JOGL

If java was used to code this system then OpenGL on its own is not compatible with java method calls therefore an additional library is needed one of which is JOGL (Sun Microsystems 2008) this sits between the java virtual machine and allows java to call all the functions of OpenGL and also integrate them with the Swing and AWT GUI libraries.

3.4.2 Java3D

In this project there is no specific requirement for low level control of the 3D environment so it would be perfectly acceptable to use a system which does not provide such fine grained control over the rendering of the scene. Java3d uses OpenGL or Direct3D but provides complete encapsulation and provides no direct method calls unlike JOGL this

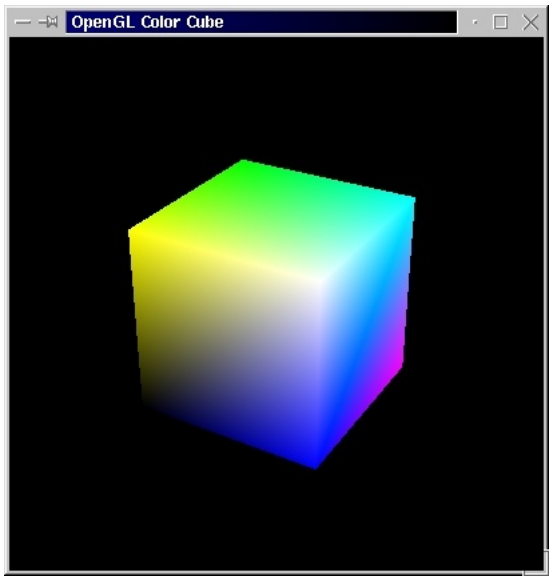


Fig. 3.4: A simple cube rendered using minimal OpenGL (Ma 2007)

has the advantage of simplifying code over the standard JOGL implementation. As a comparison Java3D based code to display the same cube as in figure 3.4 can be seen in §B.2. If the samples of code are examined then it quickly becomes apparent that Java3D is much simpler to produce simple 3D scenes quickly as individual coordinates do not have to be specified.

3.4.3 Direct3D

Direct3D is a proprietary 3D API which forms part of the Microsoft DirectX Multimedia API (Microsoft Corporation 2009*b*). It is similarly low level to OpenGL however the amount of code required to produce a single shape can be much larger than OpenGL involving around 200 lines of code just to draw a triangle in comparison to OpenGL and Java3D which only require around 100 lines of code to generate a spinning cube. Due to this it will not be considered further for use in this project.

3.5 Conclusion

After examining the possible options for 3D rendering it seems most sensible to choose the simplest system from a programming point of view therefore if the system is written in Java then Java3D will be used and if the system is coded in C++ then OpenGL will be used using similar code to the example discussed.

4. DESIGN AND IMPLEMENTATION

In this project, as it is an experimental development project, it is difficult to set a specific set of requirements from the outset as would be required by a more traditional development process (see §4.2.1). After a discussion of the basic requirements relevant development life cycles will be discussed during which an agile development process is highlighted as the most suitable. The testing ad experimental evaluation process for each phase is described in chapter 5.

4.1 Basic Requirements

By it's very nature this system has three main requirements:

- The system must be able to grab frames from a webcam and feed them to the 3D scene rendering system in a suitable format.
- The system must be able to track a face in the supplied frames at an adequate rate to provide a smooth output
- The system must display a three dimensional scene on the screen which responds to the movements of the users head to create realistic effect.

More specific requirements, such the the required rate of frame with face position data is unknown at this point and will need to be determined experimentally. This will require a more flexible development lifecycle to be followed, such as an agile development process (see §4.2.2) rather than more traditional approaches (see §4.2.1) which require fixed requirements before embarking on design and development.

4.2 Development Lifecyles

To ensure that the software produced in this project is produced as efficiently as possible a software development lifecycle (SDLC) will be employed to ensure the project is completed on time, meeting requirements and of the highest quality possible. However as software and systems development projects differ greatly in their requirements there are many development lifecycles which could be used (Bennett, McRobb & Farmer 2006).

4.2.1 Traditional Waterfall Models

Traditionally software development has followed a sequence of steps defined by a model known as the waterfall model (see fig 4.1).

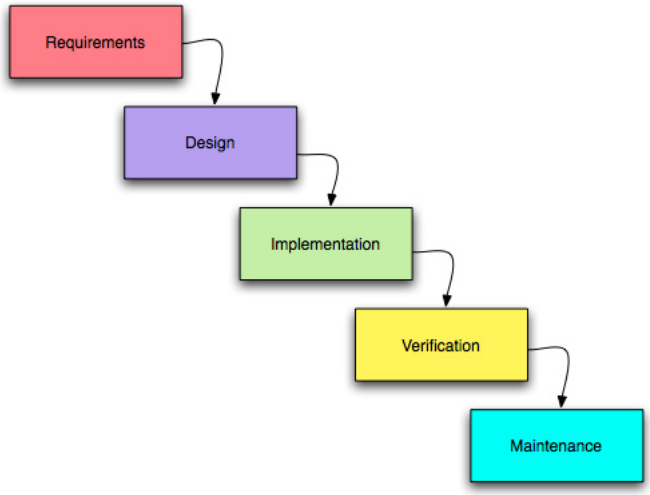


Fig. 4.1: The waterfall model (Vo 2007)

This model relies on each stage being completed before moving to the next stage of development, in most cases this is unrealistic, especially in this project where only basic requirements are set out initially and the specification is to built up throughout the development. This type of lifecycle is too restrictive and this project would benefit from a lifecycle which incorporates more flexibility of completion of the stages.

4.2.2 Iterative Methods

An iterative SDLC (see fig. 4.2) contains the same stages as the conventional waterfall model however it allows for continuous evaluation of the product and then modifying the design and code to reflect this. This satisfies the requirements of this project in allowing

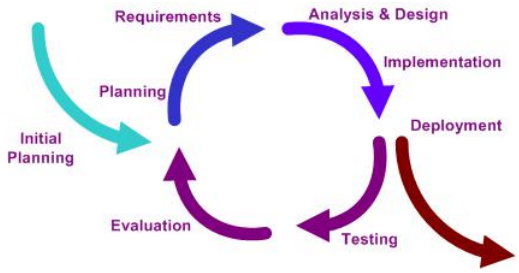


Fig. 4.2: An iterative SDLC (Vo 2007)

flexibility and changing requirements based on experimental design, however there is a more specific version of this which matches the needs of this project more closely.

Agile Development

Agile development lifecycles follow a similar basis to an iterative cycle but are more driven by a fixed time box and releasing a working release with greater regularity to encourage this. In comparison to waterfall lifecycles, where the emphasis is on producing a finished project on time, agile development releases the minimum useful functionality as soon as possible enable more through testing to take place as functionality is added (Bennett et al. 2006). This project would benefit from having many versions produced with small developments in each and thus the rest of the design will be document these steps following an agile type SDLC.

The use of an agile SDLC still has flaws. Due to the end structure of the program not being rigidly defined from the outset care will need to be taken to ensure conciseness of code and that it does not become unwieldy which could impact on performance and cause future modifications to be more complex.

4.3 Planned Phases

Despite the development and design being determined during development a brief plan of what features will be introduced in each iteration will help to prevent the development becoming an exercise in careless 'cowboy programming' (Bennett et al. 2006). These phases are not a finished plan, merely a framework on which to base continuing planning.

4.3.1 Phase 1

Basic requirements have already been defined in §4.1 therefore as a basis for further development an initial version of the software will be developed which is able to meet the first requirement:

> *"The system must be able to grab frames from a webcam and feed them to the 3D scene rendering system in a suitable format."*

Therefore phase 1 will be used to explore the best method to interface with webcams and then feed frames from them to the face tracking library. As all libraries currently specified interface well with C++ so this will be the language of choice unless specific requirements change during development.

Within Microsoft Windows the platform SDK the Directshow API (Microsoft Corporation 2008) provides functionality to access a webcam, however after viewing sample code (see §B.3) for webcam interfacing it may be more simple to use a more high level prewritten library such as that provided by OpenCV. OpenCV allows frames to be captured from a

webcam in several lines of code (Bradski & Kaehler 2008) rather than around 500 used in the Playcap (see §B.3) example, therefore OpenCV will be used for webcam interfacing.

Within OpenCV webcam input is handled by the `CvCapture.cvCaptureFromCAM()` method which initializes the camera and then frames can be captured one at a time using the `cvQueryFrame()` method which takes the capture object previously initialised as a parameter. The frames are then stored as an `IplImage` variable type. An IPL image is a format developed by Intel, within this application it is the object type used for captured frames and within each instance of `IplImage` every parameter of the image is stored including: size, colour depth, colour model, a pointer the the image data stored in BGR format (Bradski & Kaehler 2008, p42). In phase 2 any further conversion required of this image will be considered.

The resolution and actual framerate of the camera are fixed by the camera API which is hidden from this software, the resolution being 320x240 and the framerate 30fps.

4.3.2 Phase 2

The next specified basic requirement states:

> *"The system must be able to track a face in the supplied frames at an adequate rate to provide a smooth output"*

Therefore phase 2 will be used to develop the integration with the face tracking library and integrate any image conversion which is needed. OpenCV stores frames in IPL format (Bradski & Kaehler 2008), the face tracking library requires raw RGB data, however OpenCV provides simple access to raw BGR image data and a method `cvCvtColor()` to allow conversion of colourspace. A broad overview of the program structure is given in figure 4.3. This basic structure will be implemented in this phase to facilitate face tracking. Before any tracking can happen the library is initialised using the `FtInitialization()` method which allocates memory and takes the image dimensions as a parameter, in this case 640x480. All future interactions with the library require the use of the method defined in figure 4.3 as `ReadAFrame()` this method was developed in phase 1 and uses OpenCV to capture a frame from the webcam. The first stage of the face tracking is to provide the library with the initial face position, to allow this to be carried out simply a rectangle will be displayed on a live view of the webcam video stream and once the user has placed their head inside this rectangle a key press, captured with the `cvWaitKey()` method (Bradski & Kaehler 2008), will initiate the face tracking training `FTrack.FtTraining()`. The image which is passed the the face tracking library will need to be converted as it is required to be in RGB format, however OpenCV provides simple access to raw BGR image data and a method to flip the red and blue channels. Once training is complete then the `FtTrackNextFrame()` will be used in a loop to track the face and update the face position

```
{
    /*define a face tracking object*/
    CfaceTrack   FTrack;

    /* set the image size to 640x480 */
    FTrack.FtInitialization (640, 480);

    /* read a frame from image sequence (implementd by the user) */
    unsigned char ImgData = ReadAFrame();

    /* get face position, only needed at the very first frame,
    GetFacePosition() is implemented by the user */
    RECT FaceRect = GetFacePosition ();

    /* train classifier using current frame */
    FTrack.FtTraining (FaceRect, ImgData);
    BOOL bSuccess =  TRUE;

    while (bSuccess && !IsSequenceEmpty()){
         /* read a new frame */
         ImgData = ReadAFrame();
         /* update the parameters */
         bSuccess = FTrack.FtTrackNextFrame (FaceRect, ImgData);
         if (bSuccess){
                /* TODO: user can add the any other functions
                here with the given face location*/
         }
    }
}
```

Fig. 4.3: An example program segment for tracking faces (Huang 2008*b*)

rectangle accordingly. The output from this stage will be the face position as two separate variables which can be either printed to a file, the terminal or, when developed, the 3d display system.

4.3.3 Phase 3

The final specified basic requirement states:

> *"The system must display a three dimensional scene on the screen which responds to the movements of the users head to create realistic effect."*

To ease implementation this requirement will be broken into two parts, the implementation of the 3d system and then in phase 4 it will be linked with the face tracking system. As determined in §3.5 OpenGL will be used to render the 3D output and to simplify the programming of this GLUT (Kilgard 2009). This phase will also include the development of a test scene using simple lighting and built-in GLUT shapes such as `glutSolidSphere()` or `glutSolidCube()`. OpenGL includes a function to rotate a scene by a specified number of degrees using the `glRotatef()` operation integrated into this class will be a calculation to convert the face position into the difference in position of the face between the current frame and the previous frame and subsequently to calculate the degrees of rotation needed per pixel of face motion. To facilitate this a movement of half the field of view of the camera (240 pixels in the y dimension and 320 in the x dimension) corresponds to a rotation equal to half the angle of the field of view (39 °).

4.3.4 Phase 4

The aim of phase 4 is to combine all of the three requirements which have been satisfied in the previous phases into a finished working piece of software. This will involve integrating the output of the face tracking with the 3D display system, assessing through testing, if any smoothing of the face tracking data is needed. As an addition to this phase the design of the 3d scene may be altered to make it suitable for quantitative user tests. The movement of the scene will be achieved by altering the position of the OpenGL scene using the inbuilt `glRotatef()` method to rotate around a specified vector by a specific number of degrees, the number of degrees to rotated the scene per pixel of movement will need to be determined.

4.4 Implementation

As the development is following an agile development lifecycle as each stage is developed it is tested and the results of this testing used to influence and modify the planned actions in

the next phase, therefore each phase will now be documented discussing how the planned functionality was implemented, testing which took place and any modifications which resulted.

4.4.1 Phase 1

This phase implemented the basic frame capture using using OpenCV as planned, the face tracking library was also included at this stage in preparation for phase 2. Currently the program structure consists of 1 class which handles all the capture functionality in the main method (see figure 4.4).

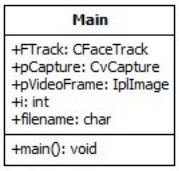


Fig. 4.4: Phase 1 Class Diagram

4.4.2 Phase 2

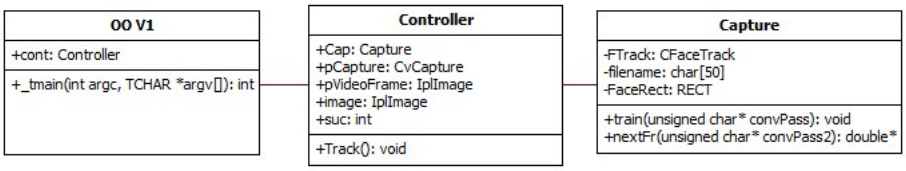


Fig. 4.5: Phase 2 Class Diagram

Now that the video capture functionality has been established the face tracking functionality can be implemented. This was implemented with few problems as described in figure 4.3 to produce a system which draws a rectangle on top of a live video stream from the camera and in response to a key press face tracking is started and the output is written to a file, for testing purposes. The GUI functionality was achieved using the HighGUI system included with OpenCV(Bradski & Kaehler 2008, p90) and the triggering of the face tracking process as planned. The structure of the program has been revised in this phase to allow greater encapsulation of functionality to ensure more robust and easily modifiable software is produced. The software now consists of three classes, one handling face tracking, one the video capture and one driver class containing the `main` to create the initial objects. (see figure 4.5) while the structure is very linear it will be developed in phase 4 to become more robust when the controller will control both the display and face tracking classes.

4.4.3 Phase 3

Phase 3 is not dependent on any previous stage to function therefore initially it was coded as a separate project for ease of testing.

As this phase is not integrated with the face tracking solution it does not actually feature dynamic movement, the variables which will take the position from the face tracking library were created and set manually to different values as required there for this phase will only feature one class (see figure 4.6). The phase was implemented much as planned in §4.3.3 and uses the inbuilt OpenGL method `glutDisplayFunc()` to get the x and y coordinates and then transform them into an angle of rotation each time the scene is redrawn. Redrawing happens in an endless loop started by calling the `glutMainLoop()` method the coordinates are static variables so this will allow them to be set directly from the controller class in phase 4. Currently the scene being rendered uses uniform lighting and a 3d teapot drawn using the glutWireTeapot() method, this will be changed in the next phase but the main problem to overcome in this phase is to allow the 3d scene to be rotated based on external coordinates.

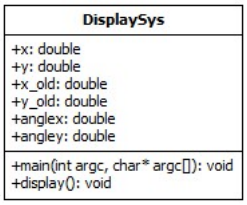


Fig. 4.6: Phase 3 Class Diagram

This phase was planned to include a calculation of the number of degrees the scene should move per pixel of face movement as the field of view of the webcam is 78 ° this means that the scene should rotate 0.12 ° per pixel.

4.4.4 Phase 4

Now the the core functionality has been developed the only remaining task is to link the face tracking and 3D display system, which are currently independent as results of phase 2 and 3.

During the implementation of this phase there were two main stages firstly integrating the 3D system from phase two so that it was no longer a standalone system and allowing it to run concurrently with the face tracking system. Secondly to provide a means for the two parts of the system to pass the face position from the face detection library to the 3D system.

To allow the 3D system to run alongside the face tracking system the two systems will

need to run in separate threads, this is due to OpenGL being based on an endless loop into which face tracking cannot easily be accommodated. To facilitate the creation of new threads the `AfxBeginThread()` method (Microsoft Corporation 2009*a*) from the MFC (Microsoft Foundation Class Library) was used to create a worker thread which contains the face tracking system, this captures the frames from the webcam directly and displays the output on the screen as well as handling training and all aspects of tracking. The face position coordinates are passed directly from the tracking thread to the display object via static variables, they are required to be static as they are used in the display() method which has a predefined header by OpenGL and is the method called every time the scene is redrawn so every time that the scene refreshes within the OpenGL loop it will have the most recent face position from the face tracking system. Initially there were problems passing this data between threads as it was unclear how to pass data between static and non static variables and methods, however this was overcome by declaring the coordinates as external variables in the controller class (see figure 4.7).

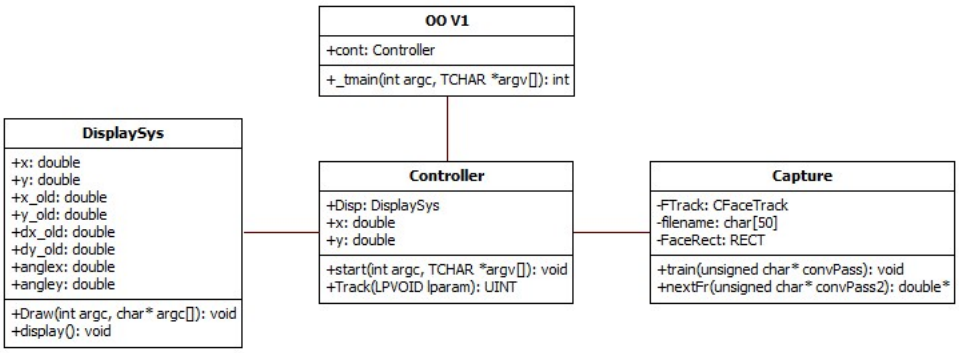


Fig. 4.7: Phase 4 Class Diagram

Currently the scene being rendered is still a teapot as in phase 3, this however can be changed just by modifying the display() method and will be discussed in chapter 6 as part of the development of a user test.

During development of this phase it was noted that there was a problem with the way rotations were being handled by the 3D system, if a movement was made around the y axis then the position of the x and z axis had also been translated accordingly this had not been noted previously as the initial 3d scene used too smaller rotations for the problem to be noticeable. A solution to this problem was implemented as part of this phase, it involved manipulating the matrices defining the viewport and model position so in effect the camera is moving to compensate for the rotation of the axis not being intentionally rotated.

5. EXPERIMENTAL EVALUATION

During each phase of the development process the system was tested, in particular the functionality which had been added during the specific phase. This chapter details a brief testing plan for each phase then the implementation (§5.1), results and evaluation of this process (§5.2).

5.1 Planned Testing

5.1.1 Phase 1

The requirement which was to be met in this phase was:

> *"The system must be able to grab frames from a webcam and feed them to the 3D scene rendering system in a suitable format."*

To test that the system can grab frames from the webcam the system will be coded to save each frame captured as a file which can then be examined to check that it is in line with what was in front of the webcam at the time, OpenCV provides integrated functionality to do this.

A measurement of the framerate of capture will be used to determine if this phase has been completed successfully this will be measured by integrating code which records the time taken to capture 90 frames of video and save them to disk, this will allow confirmation of the framerate and also, by visual checks to assure that the frames recorded are correct.

All frames that are imported into OpenCV will be in an appropriate format as OpenCV stores image data as raw BGR (Blue Green Red) data as part of its IPL Image format, this can be quickly converted to RGB as required by the face tracking library (Bradski & Kaehler 2008, p42).

5.1.2 Phase 2

Testing if the requirement for phase 2 is somewhat more difficult than phase 1 as it states:

> *"The system must be able to track a face in the supplied frames at an adequate rate to provide a smooth output"*

This raises the question of what is adequate, however this cannot be tested until phase 4, as the minimum framerate can only be known when the complete system is tested and relies to an extent on how much the users brain can fill in missing information. Therefore in this phase the testing shall concentrate on the accuracy of the tracking library. Testing will make use of OpenCV's functionality to read video files (Bradski & Kaehler 2008, p18) and producing a series of six short test videos with varying backgrounds, lighting and movement. For each video the face position will be manually determined for each frame and then plotted on the same graph as the output of the face tracking library which will be recorded to a file. If this is thought to be satisfactory then the development can proceed to phase 4.

The question of weather the system supplies frames at an adequate rate has already been addressed in §5.1.1 where it was concluded that this required user qualitative user testing.

5.1.3 Phase 3

The specified requirement for this phase was:

> *"The system must display a three dimensional scene on the screen which responds to the movements of the users head to create realistic effect."*

This requirement will only be part satisfied by this phase as it only involves the production of a standalone 3D display system based on OpenGL, the functionality to allowing testing the realism of the effect will only be achieved in phase 4, and tested mainly in the user evaluation detailed in chapter 6.

At this stage for testing purposes the movement can be generated artificially by manually changing the variables which will later take the output of the face detection. This will ensure that the rotation looks to be in the correct position for the assumed position of the face. As OpenGL is a well established 3d rendering system, for the purposes of this project it can be assumed that the rotation will behave as specified by the `glRotatef()` method documentation (OpenGL 2006).

In addition to generate a smooth output the system must redraw the 3d scene with sufficient regularity to allow smooth movement, this will be tested by adding code to automatically change the position of the scene, while this is occurring the refresh rate will be measured to assess if a smooth output is possible.

5.1.4 Phase 4

Now all functionality is in place the remaining requirements to be tested are (the sections in bold are the untested sections):

> *"The system must display a three dimensional scene on the screen which* ***responds to the movements of the users head to create a realistic effect.****"*
>
> *"The system must be able to track a face in the supplied frames at* ***an adequate rate to provide a smooth output****"*

To determine if the effect produced by the system create a realistic effect requires user evaluation rather than quantitative testing.

Problems determining what is an adequate framerate have already been identified in §5.1.2 however now the 3D system has been integrated with the face tracking so similar measurements will be performed again to measure the framerate being achieved by the software except rather than measuring the face tracking speed, this time the speed the 3D scene is refreshed will be measured. Testing of if this framerate is adequate will form part of the user testing (Chapter 6).

In addition to satisfying these requirements it is essential to test the basic functionality of the program, i.e. that a face is being tracked and its position is being passed to the 3D system so the system will again be tested using one of the videos used in phase 2, except this time incorporating the position logging into the 3D thread to show that the data is being passed correctly between threads.

5.2 Implementation and Results

5.2.1 Phase 1

At this stage only primitive testing can take place as the system is taking no user input merely capturing images from webcam. To test that images where being captured at a suitable speed extra code was added to allow OpenCV to save each frame as a JPEG image, the speed of the capture was tested quantatively by integrating a timer and measuring the time taken to capture and save 90 frames. Consistently the program completed this test

Test	Time / s	Framerate / fs^{-1}
1	3.51	25.6
2	3.20	28.1
3	3.74	24.1
4	3.90	23.1
5	3.45	26.1
6	3.76	23.9
Mean	3.59	25.0

Tab. 5.1: Results of Phase 1 Testing

in under 3.9 seconds and a mean time of 3.59s which means the minimum framerate is 23 frames per second and the mean 25 frames per second (see table 5.1) which should be

perfectly adequate for a smooth output considering films are produced with a framerate of 24fps. Also the conversion of a raw RGB data into a JPEG image will add a processing overhead not present in the final program meaning that the actual framerate is likely to be higher although the framerate will always be limited by the cameras theoretical maximum frame rate of 30 fps at 640x480.

5.2.2 Phase 2

Testing in this phase is required to ensure that the face tracking system is producing a sensible which corresponds to the face movements this was carried out by preparing 6 short videos 1s in duration and using the program to produce a file showing the face position in each frame. For each frame of video the face position was then mapped manually using Photoshop to show the pixel coordinates. The results were then plotted along with the manually determined position of the face.

To enable each frame to be analysed as an image and to ensure that the frames manually analysed corresponded exactly to the frames fed to the system. The test video generation was carried out in four steps:

1. A video was captured for each of the movements shown in table 5.2 in wmv format as this is the default used by the webcam software at 640 x 480 resolution, the same as is used by then system.
2. The useful sections of the video were captured as bitmap images using GOM Player (Gretech Corp. 2009).
3. The bitmap images were analysed manually using Adobe Photoshop to find the position of the center of the face rectangle.
4. The bitmap images were then converted back to a video in avi format using EasyBMPtoAVI (Macklin 2006). AVI format was used as that is the most easily read format by OpenCV (Carroll 2009).

After this the software was modified slightly so that instead of capturing frames from webcam it used the cvCreateFileCapture() method to load an avi file and capture and analyse each frame as if it were live video, the training phase being automatically carried out on the first frame of video. Code was also included to log the face position of every frame to a text file which was then processed using Microsoft Excel.

Initially from a qualitative point of view the results from this have have an inverse correlation in the y axis yet have a good correlation in the x axis (see figure 5.2), upon further investigation it was found that OpenCV reads in video frames upside down (Bradski & Kaehler 2008) causing this effect and that the `cvFlip()` method can be used to easily correct this.

Once the issue of the image being flipped had been resolved further analysis of the face tracking accuracy took place using 6 videos each lasting approximately one second containing a variety of movements, lighting and backgrounds. This initially took the form of plotting the face position of each frame against time for the theoretical and manual positions for comparison. However this is only of limited use, as the 3D system will work by considered dx and dy so plots of manual and automated dx and dy were also produced for each video, plots for video one have been included as an example (see figure 5.2 & 5.3).

The Spearman's Rank Correlation between the manually measured face position and the position determined by the software was calculated, along with a T-Test (p). A T-Test determines if the significance is great enough to reject a null hypothesis, i.e. to support the idea that the face is being tracked and the correlation between face positions is not just a coincidence (see Table 5.2). In this test a correlation of $0.5 < 1$ indicates a positive correlation and the greater the number the stronger the correlation. While this is useful as a measure of correlation this could occur coincidentally so a test is needed to show that this is not the case. A T-Test is designed to do this and shows the probability of the correlation occurring randomly, conventionally any correlation with a probability of less than 0.05 is deemed significant enough to support a hypothesis (Mooney & Swift 1999, p18).

As can be seen in table 5.2 all test videos have a correlation greater than 0.75 and only in one case it is less than 0.9 also for each of these results the confidence level is well below the required 0.05 figure, showing that all of the test situations the face was tracked successfully with a strong positive correlation between detected face position and manually determined face position. However on detailed examination there were some cases when the face tracking output did not correlate well with the actually face position:

- Video 1 contains a simple movement of the head from the central initial position to the left while moving closer to the camera and then back to the center with strong bright natural light from the side. On initial examination both dimensions of tracking seem to show the same general movement however not the same absolute face position, sometimes varying by up to 30% difference between the manually measured position and the position determined by FTlib, it should be noted that the area where the greatest difference occurs between the automatically tracked position and the manually tracked position is where the face is moving most rapidly in 3 dimensions at the point where it has become greatly increased in size, this may cause FTlib to produce inaccurate results as it is not able to search for a wide enough range of different face sizes.
- Video 2 features rapid face motion from the face detection system seems slow to respond to this producing much lower dx values than required (see graph for video 2 x) in practice this may not affect the realism of the 3d effect, this will be seen in phase 4.

- Video 3 contains rapid movement in the x dimension with the face at a distance further from the camera than is ideal, this seems to have lead to the motion of the face almost not being tracked at all as on visual inspection of the plots there is little close correlation between the actual position and the tracked position, however this result only occurred when a cluttered background was used combined with a small fast moving face which is a situation that should not occur if the system is used correctly.

- Video 4 contains a period of 8 frames where the face moves off the edge of the frame, in the x dimension the system seems to cope with this well, assuming the position is constant at the edge of the screen however in the y dimension most correlation is lost after that point. This is a case where despite the absolute face tracking being inaccurate the dy values still have a good correlation after this (see graph of dy video 4) and therefore in practice this may not influence the overall effect as badly as it initially appears.

- Video 5 is comprised of one frame repeated many times to form a stationary face test. This test produced interesting results because while there is a strong significant correlation in both dimensions however on visual examination of the plots the over the first $\tilde{5}$ frames the detected position of the face moved by approximately 70 pixels in the x direction and almost 25 in the y direction before becoming constant, this suggests that the initial position of the face specified by the rectangle displayed on screen is does not agree with what a face is compared to what the face detection library assumes, no documentation is available about the library where this is specified therefore before the final phase tests will be carried out as to weather the neck or hair should be part of the face aligned in the defined rectangle.

- Video 6 is unique in that it contains two faces, however this does not seem to have affected the correlation of results (this video is based on video 2) and a strong correlation can still be seen.

Up to now only the data from the absolute face position has been taken into account, while this and the dx/dy data both essentially show the same movements there are some important features shown by the new data. On the whole the correlation is still very strong with all but one (video 6) of the video having correlations between 0.7 and 0.9, in video 6 where two faces are present the tracking in the vertical dimension seems to have been severely impacted by the addition of a second moving face, The more important feature to note in the correlations for the relative face position is the greatly reduced significance when compared to the absolute position, the difference varying as much as a factor of 10^{17} in video one. This difference can be attributed to the much increased significance of noise on the relative position, as in this case the noise becomes of the same order of amplitude as the position itself compared to the noise is at least two orders of magnitude less than

$$x_t = 0.0383x_t + 0.242x_{t-1} + 0.061x_{t-2} + 0.006x_{t-3}.... \quad (5.1)$$

Fig. 5.1: Gaussian Smoothing Series

the absolute face position.

To combat the influence of noise on the overall effect the possibility of smoothing the face position data was explored. Different levels of moving average and Gaussian smoothing were tested to find the effect on the correlation of the relative face position (shown in table 5.3). First a simple moving average was calculated over 2,3 and 4 frames and the correlation measured against the manually tracked face for video 1. Moving averages up to a three frame range increase the correlation and significance, however for ranges greater than 3 frames the correlation starts to decrease again this is due to the overbearing effect of past frames which can be dramatically different if the face is fast moving. Another disadvantage of a moving average is that for each frame the range is increased by causes the smoothed output to lag behind the actual position which would cause the system to appear slow responding. A solution to smoothing the data without placing so much emphasis on far distant results is the use of Gaussian smoothing (Fisher, Perkins, Walker & Wolfart. 2003) using series of varying length of the form shown in figure 5.1. As with the moving average the influence of the smoothing on the correlation was tested and it was found that using a 3 frame Gaussian smoothing technique gave a slightly worse correlation than the comparable moving average and 4 frame Gaussian smoothing gave exactly the same correlation and significance as a 4 frame moving average. Based on this a 4 frame Gaussian smoothing system will be built into the system to try to improve the smoothness of the overall output, a Gaussian smoothing algorithm was chosen over a moving average as a Gaussian allows for sharp movements to be allowed and not reduced as much as a moving average algorithm would.

5.2.3 Phase 3

Phase 3 has only basic 3d rendering and it was tested on a rudimentary level by feeding different position coordinates manually to the system and the 3d scene changed position accordingly.

To test that that scene will move smoothly code was included to automatically rotate the scene one degree for every screen refresh using the already constructed display() method which is called when the scene is to be redrawn. This was implemented by using the `timeGetTime()` (Microsoft Corporation 2009*c*) method to record the start and end times of each iteration of the rendering loop. This produced a constant result of 0.003s which means the scene is refreshing at approximately 333.3fs^{-1} this is well in excess of the framerate of the webcam so will not restrict the speed of display of up to date 3D scenes.

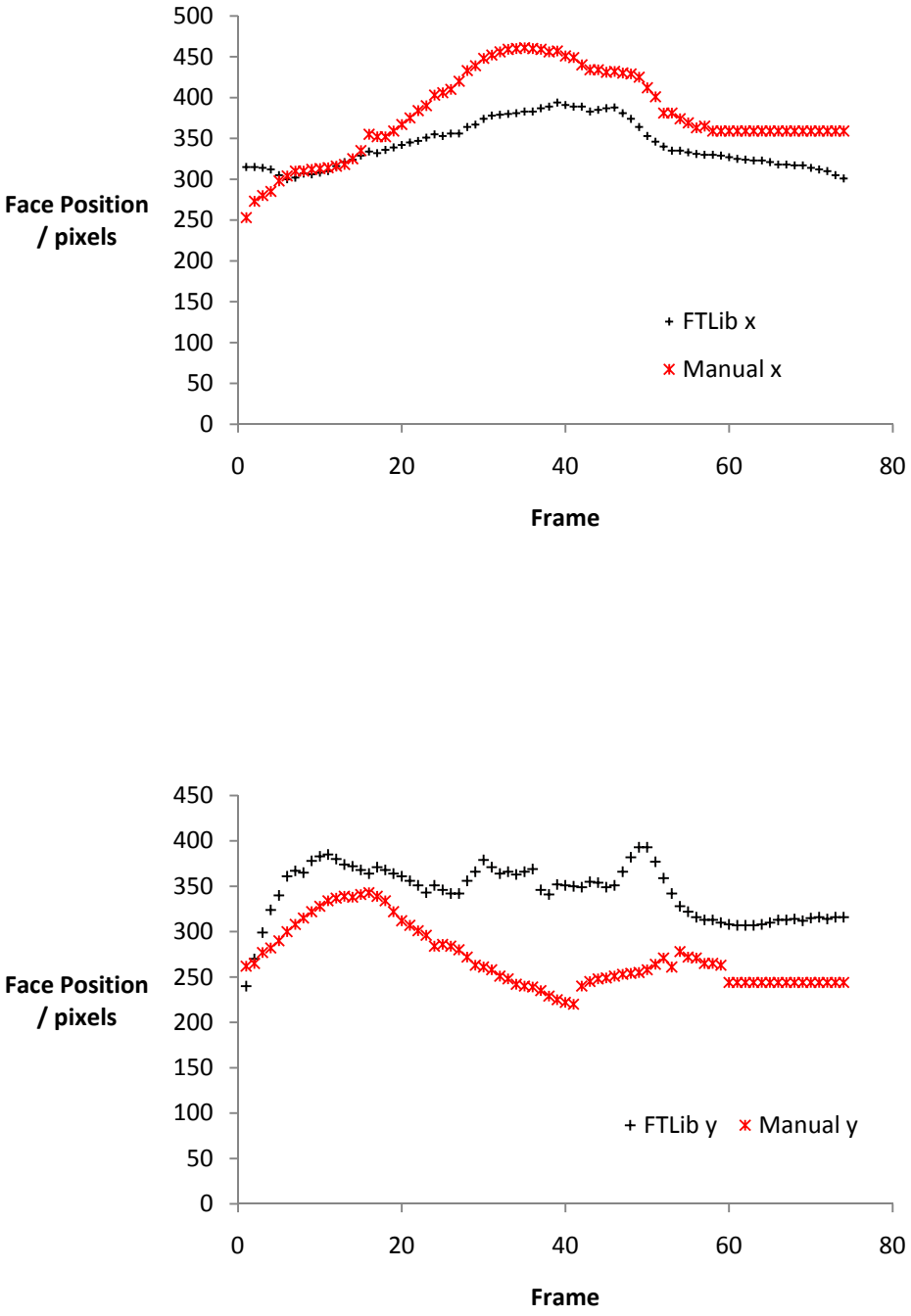


Fig. 5.2: Plot of the face position in video 1

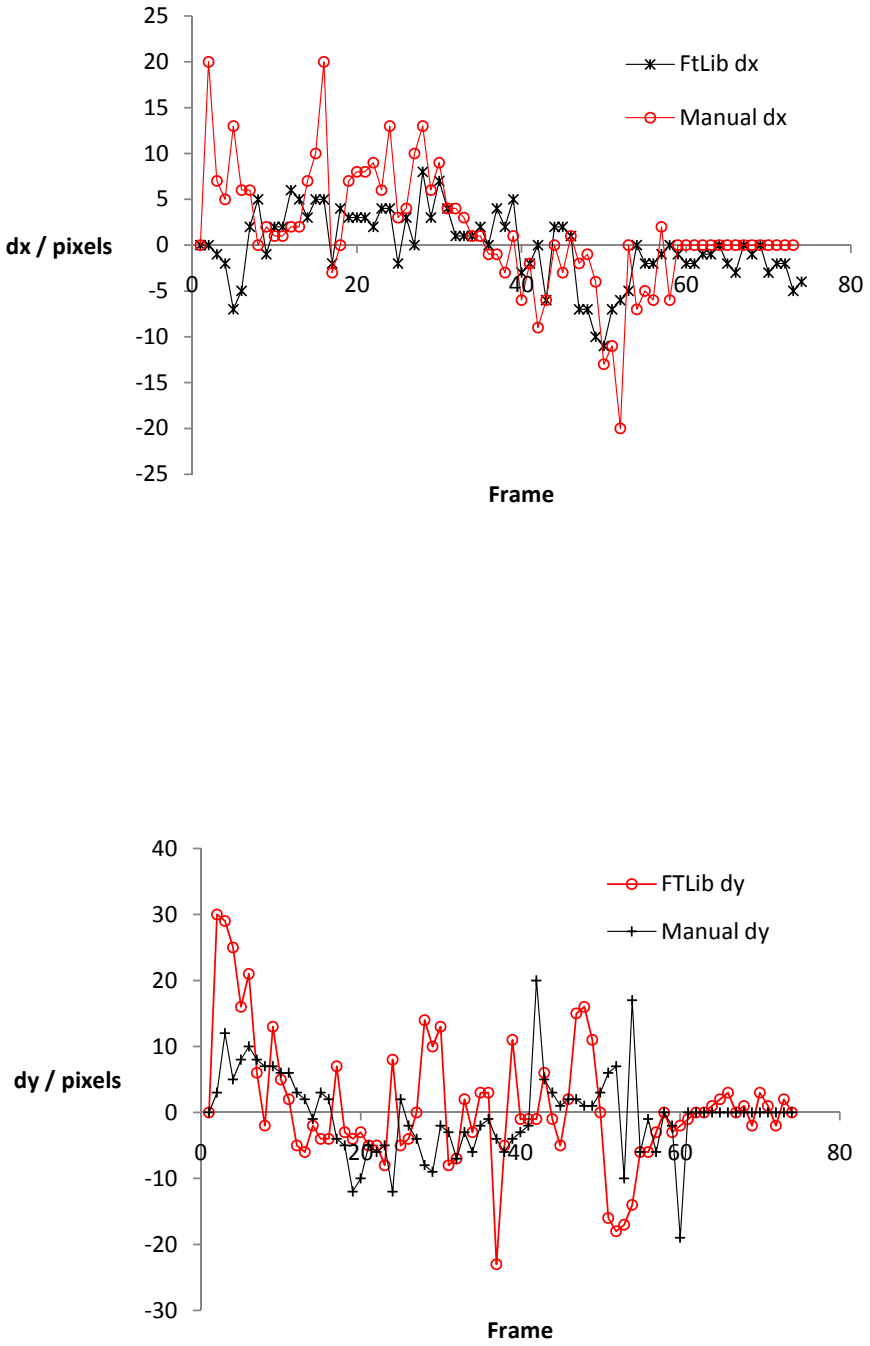


Fig. 5.3: Plots of the relative face position in video 1

Video	Description	Dimension	$\rho_{x,y}$ (p)
1	Simple movement to the left while moving closer to the camera and then back to the center. Strong bright natural light from the side. Plain background	x y dx dy	$0.994\ (2.03 \times 10^{-13})$ $0.982\ (1.82 \times 10^{-18})$ $0.986\ (1.58 \times 10^{-02})$ $0.956\ (2.92 \times 10^{-01})$
2.	Fast moving face in x direction approximately constant y position Uniform ambient artificial light Plain background	x y dx dy	$0.950\ (6.66 \times 10^{-04})$ $0.979\ (1.20 \times 10^{-09})$ $0.986\ (1.58 \times 10^{-02})$ $0.956\ (2.92 \times 10^{-01})$
3.	Fast moving face in x direction, approximately constant y position Smaller face than face rectangle Cluttered background	x y dx dy	$0.945\ (2.39 \times 10^{-01})$ $0.998\ (1.75 \times 10^{-11})$ $0.742\ (6.76 \times 10^{-01})$ $0.936\ (7.57 \times 10^{-01})$
4.	Slow moving face, curving from center to bottom right of the frame Face leaves video for 10 frames Plain Background	x y dx dy	$0.953\ (8.25 \times 10^{-02})$ $0.752\ (3.42 \times 10^{-12})$ $0.843\ (5.93 \times 10^{-01})$ $0.949\ (9.14 \times 10^{-03})$
5.	Perfectly still face Plain background	x y dx dy	$1.000\ (5.66 \times 10^{-08})$ $0.969\ (5.34 \times 10^{-09})$ $0.979\ (8.78 \times 10^{-02})$ $0.924\ (1.42 \times 10^{-02})$
6.	Two Moving Faces Based on Video 2 with extra face added with limited x motion Plain background	x y dx dy	$0.909\ (8.16 \times 10^{-02})$ $0.989\ (2.98 \times 10^{-04})$ $0.792\ (3.84 \times 10^{-01})$ $0.206\ (9.51 \times 10^{-01})$

Tab. 5.2: Descriptions of movements shown in test videos and testing results summary

Smoothing Method	Correlation (Significance)
no smoothing	0.986 (0.0409)
2 frame moving average	0.987 (0.0377)
3 frame moving average	0.988 (0.0359)
4 frame moving average	0.990 (0.0378)
2 frame Gaussian smoothing	0.986 (0.0545)
3 frame Gaussian smoothing	0.986 (0.0499)
4 frame Gaussian smoothing	0.990 (0.0378)

Tab. 5.3: Results of testing different smoothing methods on video 1 x relative position

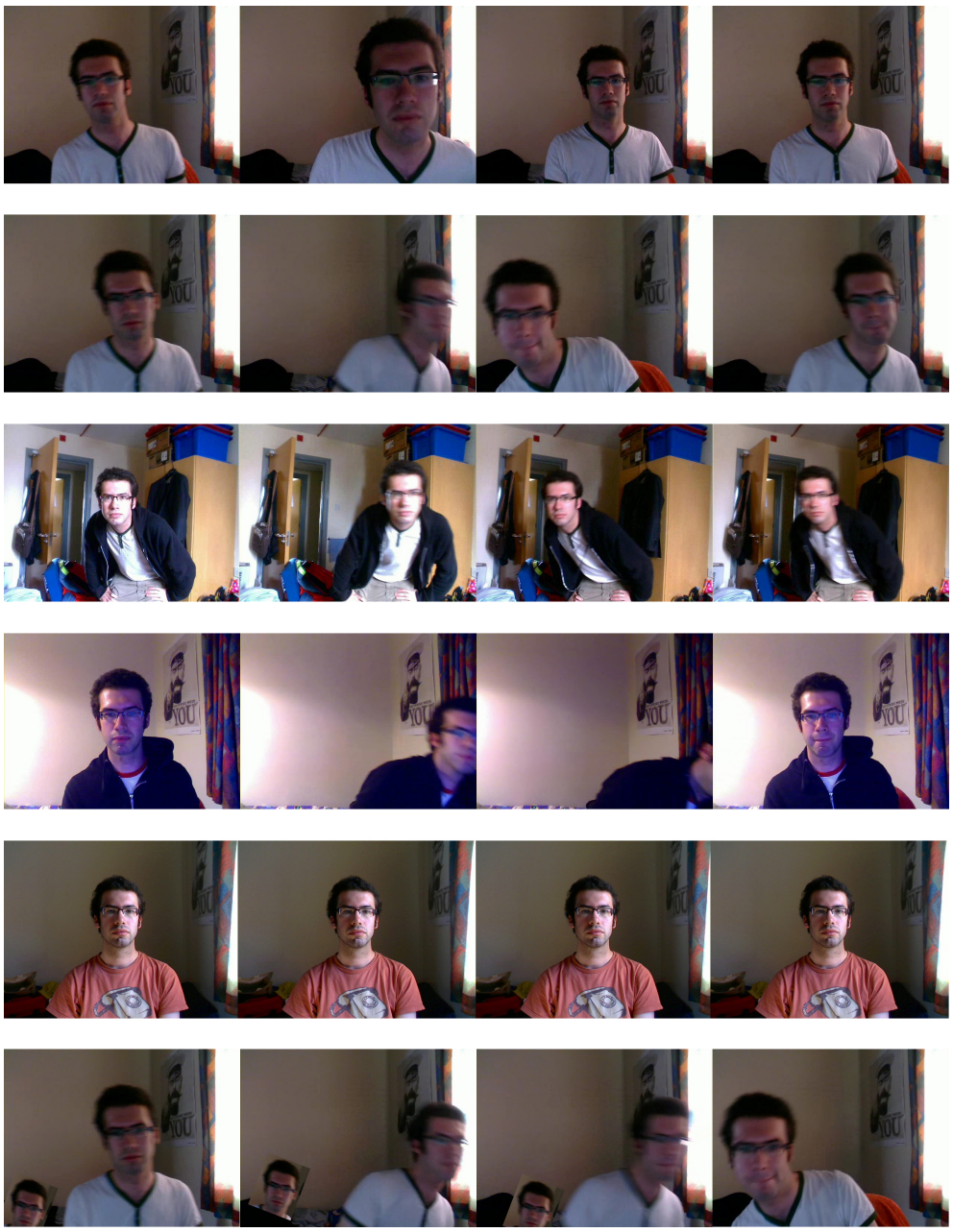


Fig. 5.4: Summaries of test videos 1-6 (Top to Bottom)

5.2.4 Phase 4

As discussed previously the majority of testing of the complete software will take place in the next chapter, however to test the new key functionality introduced in this phase a test to ensure that the face position is being passed accurately between the face library was required. This was carried out by moving the functionality used in phase 2 (see section §5.2.2) from the capture class to the DisplaySys class thus thus every time the display system refreshed it would print the coordinates of the face position to a file. The results of this compared with the results of phase two and in all cases proved identical.

Unplanned testing in this phase also highlighted issues with the rotation direction (see §4.4.4), which were resolved as part of the development.

6. USER EVALUATION

Now it has been established that the developed system performs to its specification, all that remains is to address weather the system gives the user a perception of what they are seeing being in 3D with an apparent parallax effect. To test this assessment by real users is required. In this chapter the scope of the testing will be established (§6.1) methods of testing this type of software will be discussed (§6.2), a testing plan outlined (§6.3) and implemented and a discussion of the results of this follows (§6.4).

6.1 Scope of User Testing

After experimentally evaluating the software the software requirement (see §4.1) still remaining to be fully satisfied is:

> "The system must display a three dimensional scene on the screen which responds to the movements of the users head to create realistic effect."

The only way this can be tested is to allow real users to test the software according to some kind of standardised methodology and compare their experiences. However before this can be undertaken certain questions need to be addressed:

- What task will users be asked to complete?
- How it be distinguished if a user is experiencing a three dimensionsional effect or are just able to manipulate a moving picture with face movement?
- What feedback methods are available for measuring how realistic the users find the generated effects.

From these questions arise two main areas of testing:

1. How well the user can interact with the system.
2. How much the user perceives a 3d effect.

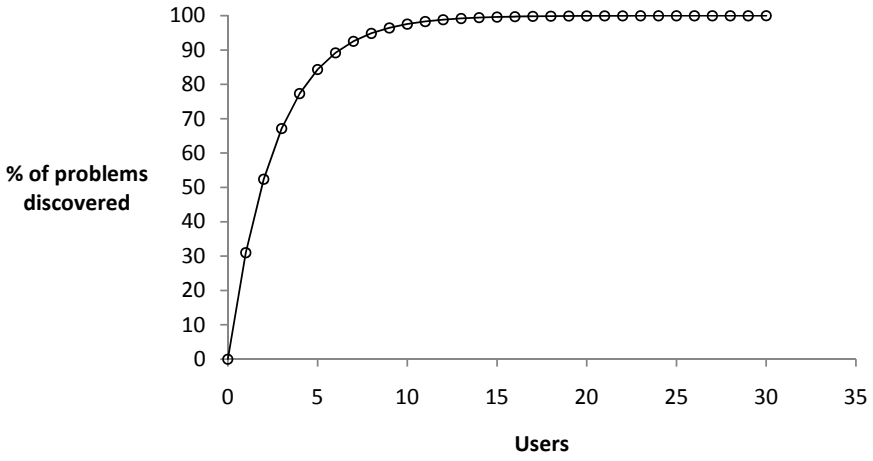
To allow the a user to test the system a 3D scene must be used that allows the user to judge quickly and simply if there is a good parallax effect and also how easy it is to interact with this. To test how good the level of interaction is a relatively simple process, a simple task such as aligning spheres will be used and the time taken until the user feels they are aligned will be recorded this will allow a quantitative test of how well users can manipulate the display and how well one half of the requirement is satisfied. To explore possible ways of extracting the most useful qualitative data from users during testing a brief literature survey will be carried out exploring different user testing techniques used in similar work.

6.2 Review of Qualitative User Testing Techniques

When performing user testing on software care is needed to obtain unbiased, reliable results. Greenberg & Buxton (2008) cites some common problems common associated with user testing in the HCI field:

- Often when developers are very familiar with their software and are looking to satisfy the requirements user tests are subconsciously developed to allow the requirements to be met most easily, not to show up software defects easily. Avoiding this in the project will require care as a test is being devised to show specific function is usable and therefore care must be taken when devising this test to allow for it to show weaknesses just as easily as good functionality.
- Sometimes when researchers are looking to demonstrate the specific benefits of their software any questions asked are targeted to test this specifically and therefore miss other issues which would have been identified with broader questions. This will be less of an issue in this project than in more general software usability testing, here user testing is being used to test one specific objective and other feedback in not required.
- Replication is mentioned as an area that is often lacking, this could also be a feature of this project, as time and resources are limited it will not be possible to conduct any kind of large scale testing and numbers are likely to be less than 10, however this should be sufficient to test if the software satisfies this requirement and therefore provide an answer to the overall research question.

While Greenberg & Buxton (2008) argues that larger groups of users produce better results other experts have claimed that the most efficient way of finding usability problems is using small groups, of less than five (Nielsen & Landauer 1993). Nielsen & Landauer (1993) also attempted to quantify this relationship producing a formula (see figure 6.1), which contradicts previous thoughts that groups of five users are most effective and suggests that



$$U = 1 - (1 - p)^n \tag{6.1}$$

Fig. 6.1: Equation and plot describing the optimum number of users compared to the number of usability problems uncovered; U is the proportion of problems uncovered, p the probability of a user uncovering a problem and n the number of users (Nielsen & Landauer 1993)

a number of users greater than ten will detect 97% of problems rather than 84% detected by five users. Nielsen (2000) also found that after a survey of users a probability of 31% could be used generally of a person finding a problem and therefore a generalised version of the function can be plotted to support the idea in this project of a group of between five and ten users being suitable.

Another issue highlighted is that the sample of users must be representative of the users using the system (Nielsen 2003). In this project this issue is very important as so far the face tracking library has not been tested using anything but a pale skin tone, ideally user testing will involve as greater mix of different faces as possible, although due to the initial training stage of the face tracking library in use (see §4.3.2) this should not be an issue as the system is always tailored to the current user.

From this brief survey these points can be concluded and will need to be considered in the planning of the user testing:

- The group of users should be at least five, if Nielsen & Landauer's (1993) relationship is taken to be true then this seems to provide the best ratio of results to time and effort spent, even if this theory is questioned common sense says that more users must allow more to be discovered in a survey.
- Attention must be paid when designing the task and questions put to the user so the task is not designed to favor one outcome.

6.3 Testing Plan

Now that some guidance has been obtained regarding the design of user testing strategies (see §6.2) the plan for the user testing will be outlined.

The user testing will be broken into two parts; firstly using a scene consisting of three spheres arranged in a triangle in the xz plane (see figure 6.2) and one in the yz plane, the user will be asked to align each pair of spheres by moving their head to either side of the screen, this process will be timed and observed in a constant environment. This test is designed to judge how well the system responds to movement of the head an adjusts the position of the 3D scene accordingly.

The 3D scene (figure 6.2) has been designed to be as simple as possible for the user to use, the objects are at different distances so any parallax effect will be immediately obvious testing both vertical and horizontal motion, the angles have been specified to be within the cameras angle of view, it is likely that the sphere above the origin will be difficult to access but it has been left in place as it could highlight a possible area of weakness in the system which would not otherwise be detected. All users taking part in this exercise will be asked to align the sphere at the front of the scene with each of the four spheres in the xy plane in a clockwise direction and the time taken recorded to allow a quantitative measure of how easily the user found it to interact with the system and use a webcam as an HID in a pseudo virtual reality system. To allow a comparison to be made against an existing HID device the system used in phase 3 will be modified to allow capture of mouse input, and then same test performed by each user, this can be handled with integrated methods in OpenGL. Less repetition will be used for mouse testing as it is an already well established system with which users are familiar.

The second part of the test will form the more qualitative part of the testing, asking users to rate the improvment of the percieved parallax effect achieved when using face tracking over mouse as an HID.

6.4 Implementation and Results

The basic tests were implemented as described (see §6.3) with only slight modification:

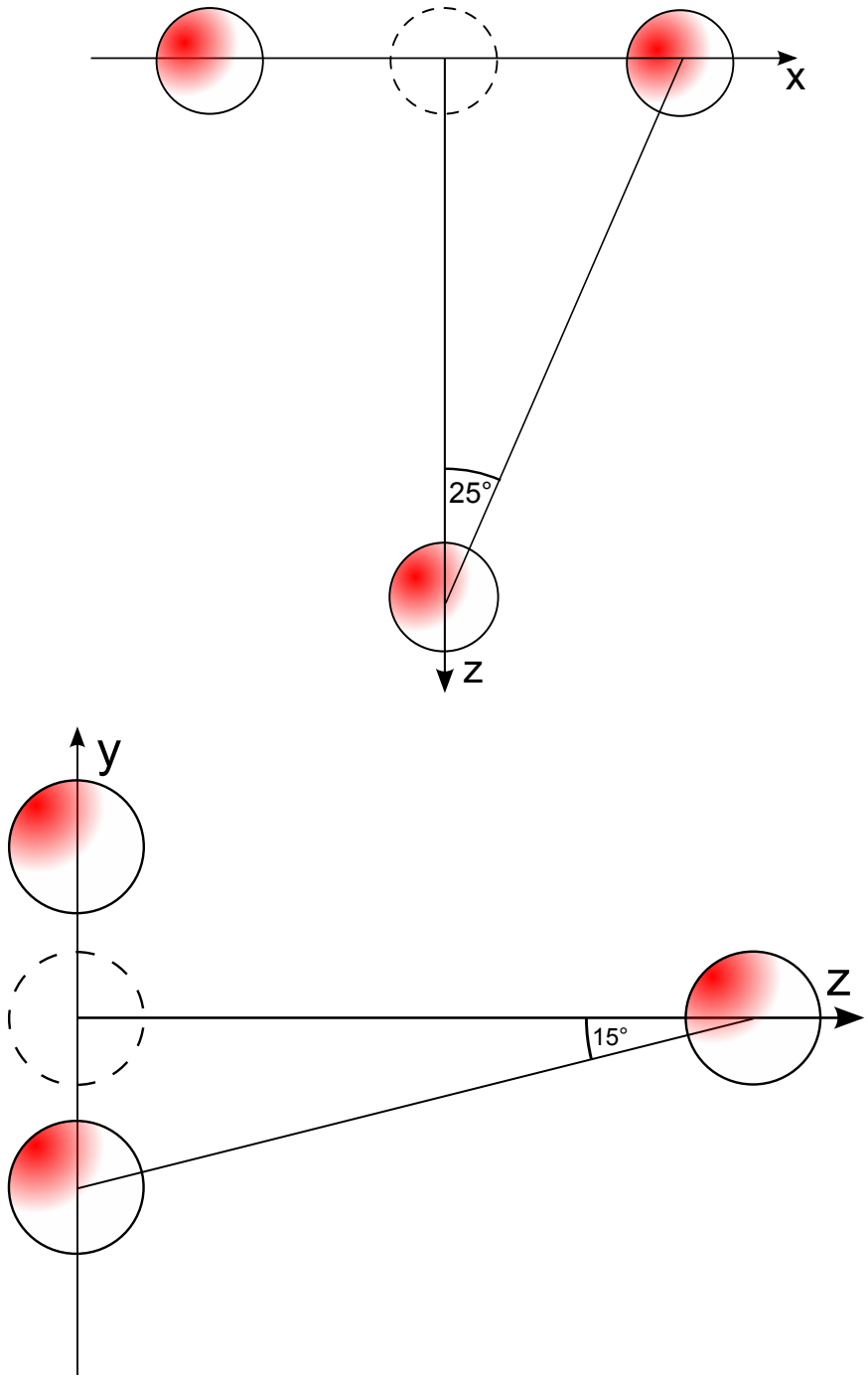


Fig. 6.2: The planned 3D test scene

- Instead of spheres teapots shapes were used as they allow the effects of the lighting to be shown more.
- The angles shown in figure 6.2 were modified after initial testing of the new scene before it was used for the user testing, the angles shown were too big to cope with sensible head movement so they were slightly decreased, rather than being 25 ° and 15 ° they became 21 ° and 11 ° (A screen shot can be seen in fig 6.3).

Fig. 6.3: Screenshot of the 3D test scene

A group of nine randomly chosen volunteers was assembled and each placed in front of the system and ask to align the front teapot with the rear four in a clockwise direction starting with the top one, using first the mouse then movement of their face, this process was timed and observed for each user, after this they were each asked to rate the realism of the parallax effect on a scale of 1-10, if a user felt that they could not align them due to the orientation of the 3D scene then a null result was recorded (an example of this is shown in figure 6.4). The results of this process are shown in table 6.1.

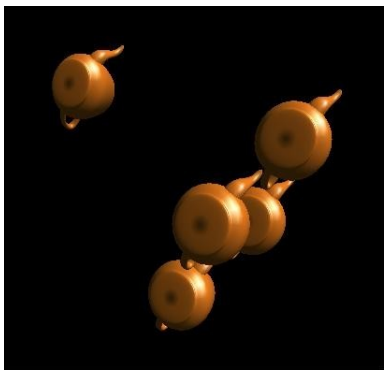


Fig. 6.4: Screen shot of anomalous 3D movement

	Time / s						
User	Mouse	Attempt 1	Attempt 2	Attempt 3	Attempt 4	Mean	Rating
1	4.8	-	24.6	-	29.2	26.9	6.0
2	5.1	20.1	25.9	22.4	-	22.8	7.0
3	3.7	-	23.1	26.3	-	24.7	4.0
4	7.0	19.5	21	30.2	-	23.6	8.0
5	6.3	24.5	25.7	21.8	24.9	24.2	7.5
6	4.6	26.2	23.1	25.9	31.0	26.6	5.0
7	5.4	-	-	21.6	28.4	25.0	9.0
8	3.9	-	19.2	25.2	-	22.2	5.0
9	4.7	23.4	26.7	-	-	25.1	4.0

Tab. 6.1: Results of User Evaluation

6.5 Discussion and Recommendations

Now all user testing is complete the results (shown in table 6.1) can be used to consider the initial research aim, to assess if a face tracking system is suitable as an HID for an desktop pseudo VR system.

On first examination of the results the most important factor is the high frequency of null results, where no time was recorded due to the 3D scene being in a state where the user was not able to complete the test. 33% of the tests produced this result which appeared from observatiopn of the testing process to caused by incorrect alignment of the face in the initial rectangle and also when the ambient light level was much stronger on one side of the face, this would lead to the webcam attempting to automatically correct the exposure and thus changing the colour balance of the image for which the tracking library was trained causing the tracking system to become confused and exhibit irratic movement. This lead to a second series of attempts by each user in a more controlled lighting area, however this only lead to a reduction to 20% of attempts recording a null result, this supports the suggestion that the problem can partly be attributed to bad lighting and partly to other factors such as misalignment of the face, in independent testing during development it was noted as a potential problem.

Now the anomalous results have been accounted for analysis can continue using the remaining results. All results gained from input with the face tracking system are slower by a factor of between 3 and 9, this was to be expected as this type of input is a very new system to most users, also as discussed in §5.2.2 the face tracking library used provides far from perfect results and thus during use of the system the 3D scene has short periods where movement goes unnoticed or is greatly exaggerated. Where the user was unable to rotate the scene to complete the test after half completion this was often cited as the cause.

The fact that users were able to complete the test 66% of the time under bad lighting conditions seems to be a testiment to a face tracking system being suitable for use in a

desktop virtual reality system, this project only serves as an experimental study of the possiblity and with further refinment could provide a much more useful system. Areas identified here for specific improvement are:

- The system which specifies the initial position of the face, the user is required to place their face inside a rectangle, however the system seems to be very sensitive to the alignment of the face within this, the issue could be avoided by using a system which detects the position of the face from the outset without user input, OpenCV can provide this functionality, using face detection via a Haar cascade (see §2.4.2), for still images and therefore could be used in this system in the training stage however a robust demonstration of this has not been produced. Other systems not explored such as Seeing Machines FaceAPI (see §2.4.3) automatically detect initial face position and then provides fast robust tracking.
- Perhaps most important of all issues is the tendency of the system to sporadically rotate the scene at such a rate that it impossible to correct this by head movement, attempts were made when coding the system to counteract this using smoothing, however this is clearly ineffective and in any future system would need further analysis into the causes of this.
- During the time the tests were taking place it was also noticed that the system crashes after running for around 90s, due to an OpenCV running out of memory, this did not interfere with any of the users tests but if the system was to become part of a larger product then this issue would need to be addressed.

7. CONCLUSION AND RECOMMENDATIONS

The original aim of this project was:

> "To research current face detection algorithms and implement, and evaluate a face tracking algorithm in a desktop virtual reality system."

In this section conclusions will be drawn from each stage of the process and critically evaluated with respect to the original aim and show that the face tracking algorithm used is suitable for use in a desktop reality system with certain modifications, then suggesting areas where further work may be beneficial (§7.2).

7.1 Conclusions

7.1.1 Initial Research

After the specification of the system had been established different methods of tracking faces were evaluated for their suitability within this system. OpenCV, The Carnegie Mellon Face Tracking Library and FaceAPI would have been able to provide all the features needed for this project however it was decided that The Carnegie Mellon Face Tracking Library (FTLib) was the most suitable as it seemed the simplest to interface with other systems due to having a minimal API. While it did provide all the required functions its simplicity lead to problems in the final product as it would be beneficial if there were more possibilities to tweak the way the face was detected. FTLib performs tracking based on colour and shape, this was a factor in its choice as it the simplest system was perceived to be the most reliable however in practice this was not the case, a more complex system such as OpenCV would have allowed more control of how the face was track rather than solely relying on colour. OpenCV can provide two different methods of face tracking one based on a library of images it is trained against and one which uses optical flow to track a predefined area of the image, both of these methods are less vulnerable to the problems diagnosed in chapter 6 as in particular the optical flow based methods (Bradski & Kaehler 2008, p322) are based on patches of light and dark which would not be affected by changes in apparent colour of the face due to lighting changes combined with the automatic exposure system of the webcam. Attempting to cope with variable light sources is a better solution than trying

to control the lighting as a part of a potentially larger program this system could easily be deployed where the developer is not able to influence the lighting of a user. OpenCV is not the only other option for coping with more complicated situations. FaceAPI (discussed in §2.4.3) seems, from personal experience of the demo provided (Seeing Machines 2008), is able to cope with almost any light source and also automatically acquires the initial position of the face, however it is not as widely documented as other options considered so further research into weather it was a viable system would be required before it could considered suitable for use in a system like this although online examples (Tarlier 2009) point towards FaceAPI as viable solution. Even after discovering some of the shortcomings with prewritten libraries this can still be seen as a better solution than writing a custom face tracking library.

Once the face tracking system had been chosen this left the 3D display system to be determined, the main options being DirectShow and OpenGL (discussed in chapter 3) OpenGL was determined to be the most suitable based on the simplicity of the code over that needed to make DirectShow display a rotating cube, however both systems would provide ample functionality to display almost any 3D scene desired so OpenGL was chosen for its simplicity. This proved to be a successful decision as there were no problems found with the 3D aspects of this system during testing (see chapter 6 and 5), the complications which occurred regarding the rotation of the axis would occur the same way in DirectX or OpenGL as they are both state machines and both would handle multiple rotations of different axis in the same way. Due to the reliability of OpenGL and the fact it is an established system this area of the project did not interfere with the more important process of evaluating the face tracking library and its suitability for the use in this type of system as any reliable 3D display system could have been used and still gained the same result. The comparisons which took place in chapter 6 between the mouse and the webcam as HIDs for the system show that the 3D environment has not influenced the conclusions drawn about the validity of the face tracking algorithm in a desktop virtual reality system as the system works flawlessly under mouse control.

7.1.2 Design

Up to this point the language the system was to be produced in was undecided, however as the API for the face tracking library was only available as a dynamic link library (`.dll`) it was deemed an unnecessary complication to use anything other than this for the project especially when OpenGL is easily interfacable with this.

Before the design process started (§4.2) it was concluded that an agile SDLC would be most appropriate as this would allow the program to be built up in stages testing each before progressing to the next, it also allowed the development to be started with only a limited amount of planning allowing decisions such as the choice of webcam interfacing method to be decided experimentally. This model proved successful throughout the project

and allows for phase 2 and 3 to overlap as phase 3 was the independent development of a 3D display system. This type of SDLC would be recommended for future projects as it allowed the possibility to experiment which is vital to a project of this nature and would not be possible if a strict and more traditional waterfall cycle (see figure 4.1) was used.

The only other major design choice not discussed is the issue of interfacing the webcam, to allow proper testing of the of the face tracking system it was determined early on this should be as simple as possible so a number of different APIs were considered OpenCV and DirectShow (§4.3.1) being the most realistic options as they provide good interfaces with C++ and are well documented, after looking into DirectShow it was deemed too complex as even simple tasks such as capturing one frame required tens of lines of code, during preliminary research (Appendix A) it was noted that some similar projects use OpenCV to handle all functionality in this type of system so this was then explored as an option, OpenCV was found to be a much simpler way to interface with a webcam which requires under ten lines of code to grab frames from a webcam, therefore OpenCV was chosen as the best system for this project. After testing in chapter 5, and every subsequent stage by proxy, it presented no problems to the functionality of the system and would be strongly recommended for any similar projects requiring webcam input. In §6.4 it was noted that OpenCV crashes after around 90s so this would indicate there is a problem with the way OpenCV has been implemented which would need to be resolved before this system could be used in any more practical context.

7.1.3 Experimental Testing

After the software development had taken place at each phase testing was carried out to ensure the planned functionality was in place and so it could be shown if the original aim was successful.

Phase 1

Phase 1 (§5.2.1) testing involved testing that the framerate provided by the OpenCV based capture system was was sufficient to provide a smooth output, it was concluded that this was the case as the system achieved a mean framerate of 25fps which is 85% of the theoretical framerate provided to OpenCV by the camera this is also the same framerate used in production video which apprears smooth.

Phase 2

Phase 2 (§5.2.2) was one of the most important phases in evaluating this face tracking algorithm, it involved testing the accuracy of the face tracking system experimentally before any visualisation system was developed to ensure it provided an accurate track of

the movement of the face, an aspect vital to showing that the tracking algorithm was suitable for this application. A series of test videos were made in varying conditions and with varying speeds of movement and the position of the face manually tracked and compared with the recorded output from FTLib. From this stage it was concluded that for simple movement the library coped well however when the face left the image for a time and returned (see figures 5.4 and 5.3) or move particularly fast the tracking system was unable to keep pace. Also tested was the influence that the tracking had on the framerate, this was found to be unaffected over the testing carried out phase 1. This lead me to the conclusion that it provided suitable tracking accuracy to continue with in the project however that it is an area which benefit from development in future work but the current face tracking filled the majority of the initial requirements identified in chapter 2 and the library was able to provide a high correlation between the manual and tracked face position.

The methods of testing used in this phase are not without their flaws, testing a face tracking system is a difficult task as there are an infinite number of different situations which could be presented to the camera and only six were covered here however within the time allowed it provided a suitable insight into the problems that are present with this face tracking library which would likely cover the majority of the major problems found in a larger sample. Ideally the library would be tested with many thousands of videos, or even a system developed for comparison with an alternative automated method of measurement of face position.

Phase 3

Phase 3 yielded few new discoveries, however it confirmed that OpenGL was a suitable system for the display of the 3D scene by successfully displaying the designed scene and moving as specified (further discussion of the choice of 3D environment can be found in section 7.1.2).

Phase 4

Experimental testing of the final phase was very limited as user testing was required to help evaluate the software against the initial aim (discussed in §7.1.4). The experimental testing of this phase consisted only of ensuring the multi threaded system implemented since phase 3 to allow graphics and face track to occur concurrently work correctly without influencing the function of any part previously tested. After testing with the same videos as used in phase 2 it was concluded that multi threading was an appropriate solution to the problem of concurrency, this could have been achieved by incorporating the video capture and tracking into the OpenGL display loop however it was concluded that this caused the system to become unnessacerily complex.

Implementing this type of multi threaded system ensures robustness of the software, if all functionality was incorporated into one class combining the OpenGL display system and the face tracking it would become more difficult to isolate the cause of such issues issues, also the way this system has been developed would allow simple replacement of the face tracking library with another for comparative tests.

7.1.4 User Testing

As the original aim stated that the face tracking algorithm was to be evaluated for use in a desktop virtual reality system this can only be truly tested with real users and virtual reality is dependent on human perception. Testing took the form of asking users to carry out a specific task and surveying them on the effectiveness of the parallax effect over a traditional HID, the mouse. This test allows the realism and usability to be assessed. It was concluded that the face tracking system is a suitable HID for a desktop VR system but does require further work on the reliability of the face tracking aspect (as discussed in §7.1.1 and 6.5).

The user testing involved nine users, this is slightly less than the optimal (according to 6.1) however according to Nielsen & Landauer's (1993) work this provides 96% efficeny of detecting usability errors, as an initial study of the usuability and realism this seems to be sufficient to draw suitable conclusions as between 15 and 25 users would be requried to increase the effiency of testing to over 99%, and therefore the conclusions drawn should be considered reliable. In furture work it may be benifical to increase the number of users in the test if testing a more complex system involving more than one face tracking library where the issues of usability could be more subtle.The user testing involved nine users, this is slightly less than the optimal (accordng to 6.1) however according to Nielsen & Landauer's (1993) work this provides 96% efficeny of detecting usability errors, as an initial study of the usuability and realism this seems to be sufficient to draw suitable conclusions as between 15 and 25 users would be requried to increase the effiency of testing to over 99%, and therefore the conclusions drawn should be considered reliable. In furture work it may be benifical to increase the number of users in the test if testing a more complex system involving more than one face tracking library where the issues of usability could be more subtle.

7.2 Recommendations for Future Work

- At the most basic level future work would be required on resolving issues identified such as the crashes of the capture system.
- Although the face tracking algorithm used was concluded to be suitable to the intended purpose other better systems may exist and further comparative work should

be carried to determine the most appropriate solution.

- The experimental study of the accuracy of the face tracking library performed here in §5.2.2 encompasses only a small section of what could be done to test an algorithm, as already identified more similar tests could be performed with real users and more videos in differing environments but more interesting questions can be raised such as the influence of noise on the perceived parallax effect or what influence cumulative error in the location of the face has upon the system.

- While in this project the application of face tracking has been considered as an HID for a desktop virtual reality system it would only take minimal work to adapt the system to work in other applications:

  - **Games** - Many computer games currently require the user to manipulate the first person camera in tandem with the weapon aim and behavior of the character or model using the mouse. If face tracking was to be employed as an HID to control the camera it could create a much more immersive environment. This is already an area of active research by Wang, Xiong, Xu, Wang, Zhang, Dai & Zhang (2006).
  - **Computer Mouse** - Earlier in this study face tracking was compared to the use of a mouse, with further refinement the system could be used to provide an alternative HID for the mouse pointer, this is already an active area of development for Tu, Tao & Huang (2007) who have developed an implementation of this.

APPENDIX

A. TERMS OF REFERENCE

Project Title An investigation into the use of a webcam for use as an HID for desktop virtual reality applications.

Supervisor: Joe Faith

A.1 Background

Recently a lot of attention has been drawn to the applications of head tracking for desktop virtual reality applications using Nintendo Wii controllers (Lee 2007), since then head tracking has become popular with many other hobbyists working on the topic, however the use of a wii controller is very specific and not suitable for wider use. The use of a more generic device such as a webcam would enable the system to be used across a much wider range of applications. Until now much has been written on the methods of head tracking but few academic publications have been written on the application of this to desktop virtual reality applications. There has been some research into the application of general body motion within games (Wang et al. 2006) which concludes that it provides a new more immersive dimension to the experience and suggests further study encompassing the distance between webcam and body would be interesting.

This project will investigate current methods used for face tracking, methods of collecting video from a webcam and how to use the output of these methods to manipulate a 3d scene to create a 3d effect by tracking head movement and altering the screen display accordingly. Currently many examples of this can be seen online especially on Youtube (Harrison 2008) but as yet no clear explanation or evaluation of these systems has been published, so this project will concentrate on recreating this type of system and evaluating it with respect to realism, usability, accuracy and robustness. Broadly this project can be divided into two components, the face detection component and a system to render a 3d scene from different angles:

A.1.1 Face Tracking

The main technical challenge to any system which tracks movement via webcam is detecting the head without detecting other parts of the body, printed faces or multiple faces in the same image and also performing this analysis at a suitable rate for real time

use (Hjelmas & Low 2001). Broadly speaking there are two commonly used methods of detecting faces, although often a mixture of the two is used (Yang & Ahuja 2001):

- Detecting there is an area of skin then confirming it is a face by looking at the shape and possibly other features such as eyes or mouth.
- Alternatively algorithms exist which look for the specific features of a face directly then looking for the relative positions to confirm the position of a face.

Other techniques are sometimes employed such as the use of MPEG video compression artifacts to detect moving areas of the image (Wang & Chang 1997) and also using layering techniques to detect areas that have moved then detecting facial features but neither has been widely developed into a practical solution (Lee, Kim & Park 1996). Most of the face detection techniques currently published are based on still images and just optimising the techniques to allow the position of the face to be recomputed for each frame of video to allow tracking, with the advent of fast home computers this is a viable solution but even then it is still advantageous to have low processing overheads to allow more processing power to concentrate on the actual visuals displayed to the user.

The most commonly used face detection library is OpenCV (Intel Corporation 2009) it is this that forms the basis of many hobbyist projects for example (2008) and thus will be considered for possible use in this project as it is able to easily interface with C/C++, C# (Huang 2008*a*), Java and Processing languages (Cousot & Stanley 2007). This library has been proved to be reliable in many areas both related and unrelated to face recognition, including its use in US Military related research (Turcsanski 2007).

A.1.2 3D Environments

Many systems exist which enable 3d environments to be programmed relatively simply such as Java 3D, Direct3D and OpenGL /JOGL, which of these is used will partly depend on the language chosen, Java 3D provides a Java API based on top of Direct3D or OpenGL to easily create 3D graphics at a high level, more low level graphics functionality is provided by Direct3D and OpenGL which will probably not be required during this project. The 3d effects referred to previously will be created by passing a x,y,z coordinate from the face tracking component to the 3D component which will move the camera in the scene accordingly.

A.1.3 Webcam Interfacing

Before any face tracking can take place the video containing the face to be tracked must be interfaced with the tracking system. If Java is used then this can be accomplished in a few different ways:

- Using the Java Media Framework (JMF) (Sun Microsystems Inc. 2008) this enables a webcam to be easily interfaced with the java environment.
- Using the TWAIN interface this can be used to control the camera and grab frames however it is much slower than JMF and probably not suitable for real-time use (Davison 2007).
- Using OpenCV the webcam is interfaced with directly thus extra integration is not required.

The best method will be determined as part of the project.

A.1.4 Testing

Previously I identified that a feature of this project would be to evaluate the realism, usability, accuracy and robustness of the system produced. As very few systems have been documented there is no precedent about how face tracking systems could be tested when used for human computer interaction, previously where face tracking was used to control the mouse along with more complex gesture recognition (Tu et al. 2007) usability was measured by comparing with traditional input methods, part of this project will include determining the best methods to use for testing this system.

A.2 Aim

To research current face detection algorithms and implement, and evaluate a face tracking algorithm in a desktop virtual reality system.

A.2.1 Objectives

- To review determine appropriate technologies for tracking faces in video
  - Identify pros and cons
  - Identify best system for use in this project
- To investigate different methods of developing simple 3d interactive environments including how to interface tracked movements with these environments
  - Identify pros and cons
  - Identify best system for use in this project
- To develop a piece of software which uses face tracking to create a perception of a 3D display on a 2D screen.

  - Determine Requirements
  - Design
  - Build
  - Test
- To evaluate different methods of testing desktop virtual reality software
  - Identify pros and cons
  - Identify best methods for use in this project
- To evaluate using user trials the software with respect to:
  - Realism
  - Usability
  - Accuracy
  - Robustness
- Report the findings of the other objectives in a dissertation.

A.3 Dissertation Outline

1. Introduction
2. Review of face tracking techniques
3. Review of 3D creation techniques
4. Discussion of the software requirements
5. Software Design Documentation
6. Discussion of development
7. Review of Evaluation Techniques
8. Evaluation
9. Discussion of future enhancements
10. Conclusion

A.4 Relationship to course

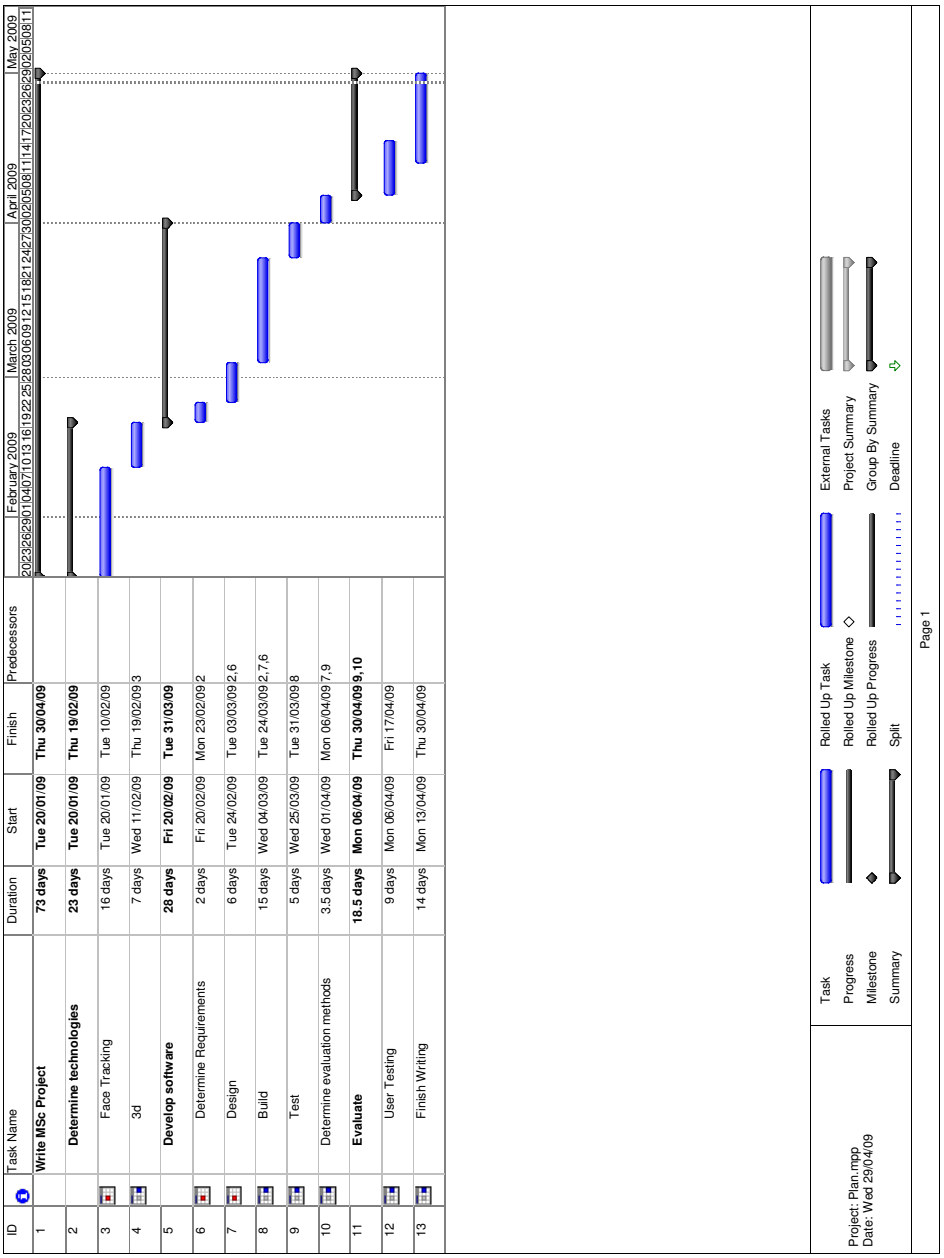
This project relates strongly to the course MSc. Computing and IT because it incorporates significant programming and technical content and considers a solely computing based subject areas. This will use material covered in the modules:

- CM0720: Systems Analysis & Design with UML
- CM0718: Programme Design & Implementation

A.5 Resources / Constraints

This project will require the use of a webcam and standard desktop/laptop computer all of which I have available to me. If the system is to be compared to (2007) then the use of a wii would be required which is not currently available to me.

A.6 Schedule of Activities



B. EXAMPLE CODE

B.1 A Simple Program to Display a Cube Using OpenGL

Code provided by Kilgard (2008)

```
/* Copyright (c) Mark J. Kilgard, 1997. */

/* This program is freely distributable without licensing fees
   and is provided without guarantee or warrantee expressed or
   implied. This program is -not- in the public domain. */

/* This program was requested by Patrick Earl; hopefully someone else
   will write the equivalent Direct3D immediate mode program. */

#include <GL/glut.h>

GLfloat light_diffuse[] = {1.0, 0.0, 0.0, 1.0};  /* Red diffuse light. */
GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};  /* Infinite light location. */
GLfloat n[6][3] = {  /* Normals for the 6 faces of a cube. */
  {-1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {1.0, 0.0, 0.0},
  {0.0, -1.0, 0.0}, {0.0, 0.0, 1.0}, {0.0, 0.0, -1.0} };
GLint faces[6][4] = {  /* Vertex indices for the 6 faces of a cube. */
  {0, 1, 2, 3}, {3, 2, 6, 7}, {7, 6, 5, 4},
  {4, 5, 1, 0}, {5, 6, 2, 1}, {7, 4, 0, 3} };
GLfloat v[8][3];  /* Will be filled in with X,Y,Z vertexes. */

void
drawBox(void)
{
  int i;

  for (i = 0; i < 6; i++) {
    glBegin(GL_QUADS);
    glNormal3fv(&n[i][0]);
    glVertex3fv(&v[faces[i][0]][0]);
    glVertex3fv(&v[faces[i][1]][0]);
    glVertex3fv(&v[faces[i][2]][0]);
    glVertex3fv(&v[faces[i][3]][0]);
    glEnd();
  }
}

void
display(void)
{
```

```
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  drawBox();
  glutSwapBuffers();
}

void
init(void)
{
  /* Setup cube vertex data. */
  v[0][0] = v[1][0] = v[2][0] = v[3][0] = -1;
  v[4][0] = v[5][0] = v[6][0] = v[7][0] = 1;
  v[0][1] = v[1][1] = v[4][1] = v[5][1] = -1;
  v[2][1] = v[3][1] = v[6][1] = v[7][1] = 1;
  v[0][2] = v[3][2] = v[4][2] = v[7][2] = 1;
  v[1][2] = v[2][2] = v[5][2] = v[6][2] = -1;

  /* Enable a single OpenGL light. */
  glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);
  glEnable(GL_LIGHT0);
  glEnable(GL_LIGHTING);

  /* Use depth buffering for hidden surface elimination. */
  glEnable(GL_DEPTH_TEST);

  /* Setup the view of the cube. */
  glMatrixMode(GL_PROJECTION);
  gluPerspective( /* field of view in degree */ 40.0,
    /* aspect ratio */ 1.0,
    /* Z near */ 1.0, /* Z far */ 10.0);
  glMatrixMode(GL_MODELVIEW);
  gluLookAt(0.0, 0.0, 5.0,  /* eye is at (0,0,5) */
    0.0, 0.0, 0.0,      /* center is at (0,0,0) */
    0.0, 1.0, 0.);      /* up is in positive Y direction */

  /* Adjust cube position to be asthetic angle. */
  glTranslatef(0.0, 0.0, -1.0);
  glRotatef(60, 1.0, 0.0, 0.0);
  glRotatef(-20, 0.0, 0.0, 1.0);
}

int
main(int argc, char **argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutCreateWindow("red 3D lighted cube");
  glutDisplayFunc(display);
  init();
  glutMainLoop();
  return 0;             /* ANSI C requires main to return int. */
}
```

B.2 A Simple Program to Display a Cube Using Java3D

Code provided by Sun Microsystems Inc. (2002).

```
/*
 * @(#)HelloUniverse1.java 1.55 02/10/21 13:43:36
 *
 * Copyright (c) 1996-2002 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *  - Redistributions of source code must retain the above copyright notice,
 * this list of conditions and the following disclaimer.
 *  - Redistribution in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * Neither the name of Sun Microsystems, Inc. or the names of contributors may
 * be used to endorse or promote products derived from this software without
 * specific prior written permission.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY
 * OF SUCH DAMAGES.
 *
 * You acknowledge that Software is not designed,licensed or intended for use in
 * the design, construction, operation or maintenance of any nuclear facility.
 */

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import javax.media.j3d.Alpha;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.RotationInterpolator;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.vecmath.Point3d;

import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.universe.SimpleUniverse;

public class HelloUniverse1 extends Applet {
```

```
private SimpleUniverse u = null;

public BranchGroup createSceneGraph() {
  // Create the root of the branch graph
  BranchGroup objRoot = new BranchGroup();

  // Create the TransformGroup node and initialize it to the
  // identity. Enable the TRANSFORM_WRITE capability so that
  // our behavior code can modify it at run time. Add it to
  // the root of the subgraph.
  TransformGroup objTrans = new TransformGroup();
  objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
  objRoot.addChild(objTrans);

  // Create a simple Shape3D node; add it to the scene graph.
  objTrans.addChild(new ColorCube(0.4));

  // Create a new Behavior object that will perform the
  // desired operation on the specified transform and add
  // it into the scene graph.
  Transform3D yAxis = new Transform3D();
  Alpha rotationAlpha = new Alpha(-1, 4000);

  RotationInterpolator rotator = new RotationInterpolator(rotationAlpha,
      objTrans, yAxis, 0.0f, (float) Math.PI * 2.0f);
  BoundingSphere bounds = new BoundingSphere(new Point3d(0.0, 0.0, 0.0),
      100.0);
  rotator.setSchedulingBounds(bounds);
  objRoot.addChild(rotator);

  // Have Java 3D perform optimizations on this scene graph.
  objRoot.compile();

  return objRoot;
}

public HelloUniverse1() {
}

public void init() {
  setLayout(new BorderLayout());
  GraphicsConfiguration config = SimpleUniverse
      .getPreferredConfiguration();

  Canvas3D c = new Canvas3D(config);
  add("Center", c);

  // Create a simple scene and attach it to the virtual universe
  BranchGroup scene = createSceneGraph();
  u = new SimpleUniverse(c);

  // This will move the ViewPlatform back a bit so the
  // objects in the scene can be viewed.
  u.getViewingPlatform().setNominalViewingTransform();
```

```
    u.addBranchGraph(scene);
  }

  public void destroy() {
    u.cleanup();
  }

  //
  // The following allows HelloUniverse to be run as an application
  // as well as an applet
  //
  public static void main(String[] args) {
    new MainFrame(new HelloUniverse1(), 256, 256);
  }
}
```

B.3 An Example of Webcam Interfacing Using Microsoft DirectShow

Code sourced from samples in Windows Vista Platform SDK (Microsoft Corporation 2008).

```
// THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF
// ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO
// THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
// PARTICULAR PURPOSE.
//
// Copyright (c) Microsoft Corporation. All rights reserved.

//
    ------------------------------------------------------------------------------

// File: PlayCap.cpp
//
// Desc: DirectShow sample code - a very basic application using video
    capture
//       devices.  It creates a window and uses the first available
    capture
//       device to render and preview video capture data.
//
    ------------------------------------------------------------------------------


#define _WIN32_WINNT 0x0500

#include <windows.h>
#include <dshow.h>
#include <stdio.h>
#include <strsafe.h>

#include "PlayCap.h"

// An application can advertise the existence of its filter graph
```

```
// by registering the graph with a global Running Object Table (ROT).
// The GraphEdit application can detect and remotely view the running
// filter graph, allowing you to 'spy' on the graph with GraphEdit.
//
// To enable registration in this sample, define REGISTER_FILTERGRAPH.
//
#define REGISTER_FILTERGRAPH


//
// Global data
//
HWND ghApp=0;
DWORD g_dwGraphRegister=0;

IVideoWindow  * g_pVW = NULL;
IMediaControl * g_pMC = NULL;
IMediaEventEx * g_pME = NULL;
IGraphBuilder * g_pGraph = NULL;
ICaptureGraphBuilder2 * g_pCapture = NULL;
PLAYSTATE g_psCurrent = Stopped;


HRESULT CaptureVideo()
{
    HRESULT hr;
    IBaseFilter *pSrcFilter=NULL;

    // Get DirectShow interfaces
    hr = GetInterfaces();
    if (FAILED(hr))
    {
        Msg(TEXT("Failed to get video interfaces! hr=0x%x"), hr);
        return hr;
    }

    // Attach the filter graph to the capture graph
    hr = g_pCapture->SetFiltergraph(g_pGraph);
    if (FAILED(hr))
    {
        Msg(TEXT("Failed to set capture filter graph! hr=0x%x"), hr);
        return hr;
    }

    // Use the system device enumerator and class enumerator to find
    // a video capture/preview device, such as a desktop USB video
       camera.
    hr = FindCaptureDevice(&pSrcFilter);
    if (FAILED(hr))
    {
        // Don't display a message because FindCaptureDevice will
           handle it
        return hr;
    }
```

```
    // Add Capture filter to our graph.
    hr = g_pGraph->AddFilter(pSrcFilter, L"Video Capture");
    if (FAILED(hr))
    {
        Msg(TEXT("Couldn't add the capture filter to the graph! hr=0x%
            x\r\n\r\n")
            TEXT("If you have a working video capture device, please
                make sure\r\n")
            TEXT("that it is connected and is not being used by another
                application.\r\n\r\n")
            TEXT("The sample will now close."), hr);
        pSrcFilter->Release();
        return hr;
    }

    // Render the preview pin on the video capture filter
    // Use this instead of g_pGraph->RenderFile
    hr = g_pCapture->RenderStream (&PIN_CATEGORY_PREVIEW, &
        MEDIATYPE_Video,
                                   pSrcFilter, NULL, NULL);
    if (FAILED(hr))
    {
        Msg(TEXT("Couldn't render the video capture stream. hr=0x%x\r\
            n")
            TEXT("The capture device may already be in use by another
                application.\r\n\r\n")
            TEXT("The sample will now close."), hr);
        pSrcFilter->Release();
        return hr;
    }

    // Now that the filter has been added to the graph and we have
    // rendered its stream, we can release this reference to the filter
        .
    pSrcFilter->Release();

    // Set video window style and position
    hr = SetupVideoWindow();
    if (FAILED(hr))
    {
        Msg(TEXT("Couldn't initialize video window! hr=0x%x"), hr);
        return hr;
    }

#ifdef REGISTER_FILTERGRAPH
    // Add our graph to the running object table, which will allow
    // the GraphEdit application to "spy" on our graph
    hr = AddGraphToRot(g_pGraph, &g_dwGraphRegister);
    if (FAILED(hr))
    {
        Msg(TEXT("Failed to register filter graph with ROT! hr=0x%x"),
            hr);
        g_dwGraphRegister = 0;
    }
#endif
```

```
    // Start previewing video data
    hr = g_pMC->Run();
    if (FAILED(hr))
    {
        Msg(TEXT("Couldn't run the graph!  hr=0x%x"), hr);
        return hr;
    }

    // Remember current state
    g_psCurrent = Running;

    return S_OK;
}


HRESULT FindCaptureDevice(IBaseFilter ** ppSrcFilter)
{
    HRESULT hr = S_OK;
    IBaseFilter * pSrc = NULL;
    IMoniker* pMoniker =NULL;
    ICreateDevEnum *pDevEnum =NULL;
    IEnumMoniker *pClassEnum = NULL;

    if (!ppSrcFilter)
        {
        return E_POINTER;
        }

    // Create the system device enumerator
    hr = CoCreateInstance (CLSID_SystemDeviceEnum, NULL, CLSCTX_INPROC,
                           IID_ICreateDevEnum, (void **) &pDevEnum);
    if (FAILED(hr))
    {
        Msg(TEXT("Couldn't create system enumerator!  hr=0x%x"), hr);
    }

    // Create an enumerator for the video capture devices

        if (SUCCEEDED(hr))
        {
            hr = pDevEnum->CreateClassEnumerator (
                CLSID_VideoInputDeviceCategory, &pClassEnum, 0);
                if (FAILED(hr))
                {
                        Msg(TEXT("Couldn't create class enumerator!  hr
                            =0x%x"), hr);
            }
        }

        if (SUCCEEDED(hr))
        {
                // If there are no enumerators for the requested type,
                    then
```

```
                // CreateClassEnumerator will succeed, but pClassEnum
                    will be NULL.
                if (pClassEnum == NULL)
                {
                        MessageBox(ghApp,TEXT("No video capture device
                            was detected.\r\n\r\n")
                                TEXT("This sample requires a video
                                    capture device, such as a USB
                                    WebCam,\r\n")
                                TEXT("to be installed and working
                                    properly.  The sample will now
                                    close."),
                                TEXT("No Video Capture Hardware"),
                                    MB_OK | MB_ICONINFORMATION);
                        hr = E_FAIL;
                }
        }

    // Use the first video capture device on the device list.
    // Note that if the Next() call succeeds but there are no monikers,
    // it will return S_FALSE (which is not a failure).  Therefore, we
    // check that the return code is S_OK instead of using SUCCEEDED()
        macro.

        if (SUCCEEDED(hr))
        {
                hr = pClassEnum->Next (1, &pMoniker, NULL);
                if (hr == S_FALSE)
                {
                Msg(TEXT("Unable to access video capture device!"));
                        hr = E_FAIL;
                }
        }

        if (SUCCEEDED(hr))
    {
        // Bind Moniker to a filter object
        hr = pMoniker->BindToObject(0,0,IID_IBaseFilter, (void**)&pSrc)
            ;
        if (FAILED(hr))
        {
            Msg(TEXT("Couldn't bind moniker to filter object!  hr=0x%x"
                ), hr);
        }
    }

    // Copy the found filter pointer to the output parameter.
        if (SUCCEEDED(hr))
        {
            *ppSrcFilter = pSrc;
                (*ppSrcFilter)->AddRef();
        }

        SAFE_RELEASE(pSrc);
    SAFE_RELEASE(pMoniker);
```

```
    SAFE_RELEASE(pDevEnum);
    SAFE_RELEASE(pClassEnum);

    return hr;
}


HRESULT GetInterfaces(void)
{
    HRESULT hr;

    // Create the filter graph
    hr = CoCreateInstance (CLSID_FilterGraph, NULL, CLSCTX_INPROC,
                           IID_IGraphBuilder, (void **) &g_pGraph);
    if (FAILED(hr))
        return hr;

    // Create the capture graph builder
    hr = CoCreateInstance (CLSID_CaptureGraphBuilder2 , NULL,
    CLSCTX_INPROC,
                           IID_ICaptureGraphBuilder2, (void **) &
                               g_pCapture);
    if (FAILED(hr))
        return hr;

    // Obtain interfaces for media control and Video Window
    hr = g_pGraph->QueryInterface(IID_IMediaControl,(LPVOID *) &g_pMC);
    if (FAILED(hr))
        return hr;

    hr = g_pGraph->QueryInterface(IID_IVideoWindow, (LPVOID *) &g_pVW);
    if (FAILED(hr))
        return hr;

    hr = g_pGraph->QueryInterface(IID_IMediaEvent, (LPVOID *) &g_pME);
    if (FAILED(hr))
        return hr;

    // Set the window handle used to process graph events
    hr = g_pME->SetNotifyWindow((OAHWND)ghApp, WM_GRAPHNOTIFY, 0);

    return hr;
}


void CloseInterfaces(void)
{
    // Stop previewing data
    if (g_pMC)
        g_pMC->StopWhenReady();

    g_psCurrent = Stopped;

    // Stop receiving events
    if (g_pME)
```

```
        g_pME->SetNotifyWindow(NULL, WM_GRAPHNOTIFY, 0);

    // Relinquish ownership (IMPORTANT!) of the video window.
    // Failing to call put_Owner can lead to assert failures within
    // the video renderer, as it still assumes that it has a valid
    // parent window.
    if(g_pVW)
    {
        g_pVW->put_Visible(OAFALSE);
        g_pVW->put_Owner(NULL);
    }

#ifdef REGISTER_FILTERGRAPH
    // Remove filter graph from the running object table
    if (g_dwGraphRegister)
        RemoveGraphFromRot(g_dwGraphRegister);
#endif

    // Release DirectShow interfaces
    SAFE_RELEASE(g_pMC);
    SAFE_RELEASE(g_pME);
    SAFE_RELEASE(g_pVW);
    SAFE_RELEASE(g_pGraph);
    SAFE_RELEASE(g_pCapture);
}


HRESULT SetupVideoWindow(void)
{
    HRESULT hr;

    // Set the video window to be a child of the main window
    hr = g_pVW->put_Owner((OAHWND)ghApp);
    if (FAILED(hr))
        return hr;

    // Set video window style
    hr = g_pVW->put_WindowStyle(WS_CHILD | WS_CLIPCHILDREN);
    if (FAILED(hr))
        return hr;

    // Use helper function to position video window in client rect
    // of main application window
    ResizeVideoWindow();

    // Make the video window visible, now that it is properly
    positioned
    hr = g_pVW->put_Visible(OATRUE);
    if (FAILED(hr))
        return hr;

    return hr;
}
```

```
void ResizeVideoWindow(void)
{
    // Resize the video preview window to match owner window size
    if (g_pVW)
    {
        RECT rc;

        // Make the preview video fill our window
        GetClientRect(ghApp, &rc);
        g_pVW->SetWindowPosition(0, 0, rc.right, rc.bottom);
    }
}


HRESULT ChangePreviewState(int nShow)
{
    HRESULT hr=S_OK;

    // If the media control interface isn't ready, don't call it
    if (!g_pMC)
        return S_OK;

    if (nShow)
    {
        if (g_psCurrent != Running)
        {
            // Start previewing video data
            hr = g_pMC->Run();
            g_psCurrent = Running;
        }
    }
    else
    {
        // Stop previewing video data
        hr = g_pMC->StopWhenReady();
        g_psCurrent = Stopped;
    }

    return hr;
}


#ifdef REGISTER_FILTERGRAPH

HRESULT AddGraphToRot(IUnknown *pUnkGraph, DWORD *pdwRegister)
{
    IMoniker * pMoniker;
    IRunningObjectTable *pROT;
    WCHAR wsz[128];
    HRESULT hr;

    if (!pUnkGraph || !pdwRegister)
        return E_POINTER;

    if (FAILED(GetRunningObjectTable(0, &pROT)))
```

```
        return E_FAIL;

    hr = StringCchPrintfW(wsz, NUMELMS(wsz), L"FilterGraph %08x pid %08
        x\0", (DWORD_PTR)pUnkGraph,
              GetCurrentProcessId());

    hr = CreateItemMoniker(L"!", wsz, &pMoniker);
    if (SUCCEEDED(hr))
    {
        // Use the ROTFLAGS_REGISTRATIONKEEPSALIVE to ensure a strong
            reference
        // to the object.  Using this flag will cause the object to
            remain
        // registered until it is explicitly revoked with the Revoke()
            method.
        //
        // Not using this flag means that if GraphEdit remotely
            connects
        // to this graph and then GraphEdit exits, this object
            registration
        // will be deleted, causing future attempts by GraphEdit to
            fail until
        // this application is restarted or until the graph is
            registered again.
        hr = pROT->Register(ROTFLAGS_REGISTRATIONKEEPSALIVE, pUnkGraph,
                            pMoniker, pdwRegister);
        pMoniker->Release();
    }

    pROT->Release();
    return hr;
}


// Removes a filter graph from the Running Object Table
void RemoveGraphFromRot(DWORD pdwRegister)
{
    IRunningObjectTable *pROT;

    if (SUCCEEDED(GetRunningObjectTable(0, &pROT)))
    {
        pROT->Revoke(pdwRegister);
        pROT->Release();
    }
}

#endif


void Msg(TCHAR *szFormat, ...)
{
    TCHAR szBuffer[1024];  // Large buffer for long filenames or URLs
    const size_t NUMCHARS = sizeof(szBuffer) / sizeof(szBuffer[0]);
    const int LASTCHAR = NUMCHARS - 1;
```

```
    // Format the input string
    va_list pArgs;
    va_start(pArgs, szFormat);

    // Use a bounded buffer size to prevent buffer overruns.  Limit
        count to
    // character size minus one to allow for a NULL terminating
        character.
    (void)StringCchVPrintf(szBuffer, NUMCHARS - 1, szFormat, pArgs);
    va_end(pArgs);

    // Ensure that the formatted string is NULL-terminated
    szBuffer[LASTCHAR] = TEXT('\0');

    MessageBox(NULL, szBuffer, TEXT("PlayCap Message"), MB_OK |
        MB_ICONERROR);
}


HRESULT HandleGraphEvent(void)
{
    LONG evCode;
        LONG_PTR evParam1, evParam2;
    HRESULT hr=S_OK;

    if (!g_pME)
        return E_POINTER;

    while(SUCCEEDED(g_pME->GetEvent(&evCode, &evParam1, &evParam2, 0)))
    {
        //
        // Free event parameters to prevent memory leaks associated
            with
        // event parameter data.  While this application is not
            interested
        // in the received events, applications should always process
            them.
        //
        hr = g_pME->FreeEventParams(evCode, evParam1, evParam2);

        // Insert event processing code here, if desired
    }

    return hr;
}


LRESULT CALLBACK WndMainProc (HWND hwnd, UINT message, WPARAM wParam,
    LPARAM lParam)
{
    switch (message)
    {
        case WM_GRAPHNOTIFY:
            HandleGraphEvent();
            break;
```

```
        case WM_SIZE:
            ResizeVideoWindow();
            break;

        case WM_WINDOWPOSCHANGED:
            ChangePreviewState(!(IsIconic(hwnd)));
            break;

        case WM_CLOSE:
            // Hide the main window while the graph is destroyed
            ShowWindow(ghApp, SW_HIDE);
            CloseInterfaces();  // Stop capturing and release
                interfaces
            break;

        case WM_DESTROY:
            PostQuitMessage(0);
            return 0;
    }

    // Pass this message to the video window for notification of system
        changes
    if (g_pVW)
        g_pVW->NotifyOwnerMessage((LONG_PTR) hwnd, message, wParam,
            lParam);

    return DefWindowProc (hwnd , message, wParam, lParam);
}


int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hInstP, LPSTR
    lpCmdLine, int nCmdShow)
{
    MSG msg={0};
    WNDCLASS wc;

    // Initialize COM
    if(FAILED(CoInitializeEx(NULL, COINIT_APARTMENTTHREADED)))
    {
        Msg(TEXT("CoInitialize Failed!\r\n"));
        exit(1);
    }

    // Register the window class
    ZeroMemory(&wc, sizeof wc);
    wc.lpfnWndProc   = WndMainProc;
    wc.hInstance     = hInstance;
    wc.lpszClassName = CLASSNAME;
    wc.lpszMenuName  = NULL;
    wc.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
    wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
    wc.hIcon         = LoadIcon(hInstance, MAKEINTRESOURCE(
        IDI_VIDPREVIEW));
    if(!RegisterClass(&wc))
```

```
    {
        Msg(TEXT("RegisterClass Failed! Error=0x%x\r\n"), GetLastError
            ());
        CoUninitialize();
        exit(1);
    }

    // Create the main window.  The WS_CLIPCHILDREN style is required.
    ghApp = CreateWindow(CLASSNAME, APPLICATIONNAME,
                         WS_OVERLAPPEDWINDOW | WS_CAPTION |
                             WS_CLIPCHILDREN,
                         CW_USEDEFAULT, CW_USEDEFAULT,
                         DEFAULT_VIDEO_WIDTH, DEFAULT_VIDEO_HEIGHT,
                         0, 0, hInstance, 0);

    if(ghApp)
    {
        HRESULT hr;

        // Create DirectShow graph and start capturing video
        hr = CaptureVideo();
        if (FAILED (hr))
        {
            CloseInterfaces();
            DestroyWindow(ghApp);
        }
        else
        {
            // Don't display the main window until the DirectShow
            // preview graph has been created.  Once video data is
            // being received and processed, the window will appear
            // and immediately have useful video data to display.
            // Otherwise, it will be black until video data arrives.
            ShowWindow(ghApp, nCmdShow);
        }

        // Main message loop
        while(GetMessage(&msg,NULL,0,0))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    // Release COM
    CoUninitialize();

    return (int) msg.wParam;
}
```

BIBLIOGRAPHY

Ahmed, N., Natarajan, T. & Rao, K. R. (1974), 'Discrete cosine transform', *IEEE Trans. Computers* pp. 90–93.

Bennett, S., McRobb, S. & Farmer, R. (2006), *Object Oriented Systems Analysis and Design*, McGraw Hill.

Bradski, D. G. R. & Kaehler, A. (2008), *Learning OpenCV*, O'Reilly Media, Inc.

Cai, J. & Goshtasby, A. (1999), 'Detecting human faces in color images', *IMAGE AND VISION COMPUTING* **18**(1), 63–75.

Carroll, J. (2009), 'Opencvwiki: Video codecs', Available at: *http://opencv.willowgarage.com/wiki/VideoCodecs* (Accessed: 10th Apr. 2009).

Cousot, S. & Stanley, D. E. (2007), 'Opencv - processing and java library', Available at: *http://ubaa.net/shared/processing/opencv/*.

Davison, A. (2007), *Pro Java 6: Game Development with Java: 3D and JOGL*, Apress.

Elias, H. (2007), 'Radiosity progress.png', Available at: *http://en.wikipedia.org/wiki/File:Radiosity Progress.png* (Accessed: 26th May. 2009).

Fisher, R., Perkins, S., Walker, A. & Wolfart., E. (2003), 'Gaussian smoothing', Available at: *http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm* (Accessed: 22nd Apr. 2009).

Foley, J., Dam, A. V., Feiner, S. K. & Hughes, J. F. (1990), *Computer Graphics: Principles and Practice*, Addison Wesley.

Glassner, A. S. (1989), *An Introduction to Ray Tracing*, Morgan Kaufmann.

Goral, C., Torrance, K. E., Greenberg, D. P. & Battaile, B. (1983), 'Modeling the interaction of light between diffuse surfaces', *Computer Graphics* **18(3)**, 213–222.

Govindaraju, V. (1996), 'Locating human faces in photographs', *Int. J. Comput. Vision* **19**(2), 129–146.

Greenberg, S. & Buxton, B. (2008), Usability evaluation considered harmful (some of the time), *in* 'CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems', ACM, New York, NY, USA, pp. 111–120.

Gretech Corp. (2009), 'Gom software', Available at: *http://www.gomlab.com/eng/* (Accessed: 22nd Mar. 2009).

Hardware Zone (2005), 'Logitech quickcam fusion delivers most sophisticated technologies', Available at: *http://www.hardwarezone.com/news/view.php?id=2950&cid=7* (Accessed: 22nd Jan. 2009).

Harrison, C. (2008), 'Head tracking with a generic webcam', Available at: *http://uk.youtube.com/watch?v=Q-nrmxNKt84* (Accessed: 2nd Nov. 2008).

Haykin, S. (1999), *Neural Networks: A Comprehensive Foundation*, Prentice Hall.

Hewitt, R. (2007), 'Seeing with opencv', *Servo* **3**, 36–40.

Hjelmas, E. & Low, B. (2001), 'Face detection: A survey', *Computer Vision and Image Understanding* **83**, 236–274.

Huang, C. (2008*a*), 'Emgu cv', Available at: *http://www.emgu.com/wiki/index.php/* (Accessed: 22nd Jan. 2009).

Huang, F. J. (2008*b*), 'Carnegie mellon face tracking library', Available at: *http://amp.ece.cmu.edu/projects/FaceTracking/* (Accessed: 22nd Nov. 2008).

Huang, F. J. & Chen, T. (2000), 'Tracking of multiple faces for human-computer interfaces and virtual environments', *IEEE International Conference on Multimedia and Expo* **3**, 1563–1566.

Hutton, J. & Dowling, B. (2005), 'Computer vision demonstration website', Available at: *http://users.ecs.soton.ac.uk/msn/book/new_demo/laplacian/* (Accessed: 22nd Jan. 2009).

Intel Corporation (2009), 'Sourceforge.net: Open computer vision library', Available at: *http://sourceforge.net/projects/opencvlibrary/* (Accessed: 22nd Jan. 2009).

Kilgard, M. (2009), 'Glut - the opengl utility toolkit', Available at: *http://www.opengl.org/resources/libraries/glut/* (Accessed: 1st Mar. 2009).

Kilgard, M. J. (2008), 'Cube', Available at: *http://euro-com.blogspot.com/2008/09/cube.html* (Accessed: 22nd Jan. 2009).

Kimball, A. W. (1957), 'Errors of the third kind in statistical consulting', *Journal of the American Statistical Association* **52**, 133–142.

Lee, C. H., Kim, J. S. & Park, K. H. (1996), 'Automatic human face location in a complex background using motion and color information', *Pattern Recognition* **29**(11), 1877 – 1889. Human face location;Motion detection;Face recognition;.

Lee, J. C. (2007), 'Head tracking for desktop vr displays using the wii remote', Available at: *http://www.cs.cmu.edu/johnny/projects/wii/* (Accessed: 2nd Nov. 2008).

Lin, C. (2007), 'Face detection in complicated backgrounds and different illumination conditions by using ycbcr color space and neural network', *Pattern Recogn. Lett.* **28**(16), 2190–2200.

Ma, T. (2007), 'Opengl color cube', Available at: *http://www.terrence.com/opengl/ccube/ccube.html* (Accessed: 26th Feb. 2009).

Macklin, P. (2006), 'Easybmptoavi movie creator', Available at: *http://easybmptoavi.sourceforge.net/* (Accessed: 10th Apr. 2009).

Marcel, S. & Rodriguez, Y. (2007), 'Torch3vision - the vision package of touch3', Available at: *http://torch3vision.idiap.ch/* (Accessed: 22nd Jan. 2009).

Microsoft Corporation (2008), 'Directshow', Available at: *http://msdn.microsoft.com/en-gb/library/dd375454(VS.85).aspx* (Accessed: 25nd Feb. 2009).

Microsoft Corporation (2009*a*), 'Afxbeginthread (mfc)', Available at: *http://msdn.microsoft.com/en-us/library/s3w9x78e(VS.80).aspx* (Accessed: 23rd Apr. 2009).

Microsoft Corporation (2009*b*), 'Directx: Advanced graphics on windows', Available at: *http://msdn.microsoft.com/en-gb/directx/default.aspx* (Accessed: 22nd Feb. 2009).

Microsoft Corporation (2009*c*), 'timegettime - msdn library', Available at: *http://msdn.microsoft.com/en-us/library/ms713418.aspx* (Accessed: 3rd Mar. 2009).

Mooney, D. & Swift, R. (1999), *A course in mathematical modeling*, Cambridge University Press.

Nielsen, J. (2000), 'Why you only need to test with 5 users', Available at: *http://www.useit.com/alertbox/20000319.html* (Accessed: 22nd Apr. 2009).

Nielsen, J. (2003), 'Recruiting test participants for usability studies', Available at: *http://www.useit.com/alertbox/20030120.html* (Accessed: 26th Apr. 2009).

Nielsen, J. & Landauer, T. K. (1993), A mathematical model of the finding of usability problems, *in* 'CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems', ACM, New York, NY, USA, pp. 206–213.

OpenGL (2006), 'glrotatef', Available at: *http://www.opengl.org/sdk/docs/man/xhtml/glRotate.xml* (Accessed: 22th Apr. 2009).

OpenGL (2009), 'Opengl - the industry's foundation for high performance graphics', Available at: *http://www.opengl.org/* (Accessed: 22nd Jan. 2009).

OpenRT (2008), 'Openrt - real-time ray-tracing project', Available at: *http://www.openrt.de/* (Accessed: 22nd Feb. 2009).

Papageorgiou, C., Oren, M. & Poggio, T. (1998), A general framework for object detection, *in* 'SIXTH INTERNATIONAL CONFERENCE ON COMPUTER VISION', IEEE Comp Soc, NAROSA PUBLISHING HOUSE, 22 DARYAGANJ, DELHI MEDICAL ASSOCIATION RD, NEW DELHI 110 002, INDIA, pp. 555–562. 6th International Conference on Computer Vision, BOMBAY, INDIA, JAN 04-07, 1998.

Podolsky, A. & Frolov, V. (2008), 'Face tracking', Available at: *http://www.cs.bgu.ac.il/ orlovm/teaching/saya/reports/saya-tracking-report.pdf* (Accessed: 22nd Jan. 2009).

Roth, S. D. (1982), 'Ray Casting for Modeling Solids', *Computer Graphics and Image Processing* **18**(2), 109–144.

Sakai, T., Nagao, M. & Kanade, T. (1972), Computer analysis and classification of photographs of human faces, *in* 'Proc. First USA-JAPAN Computer Conference', pp. 55–62.

Seeing Machines (2008), 'faceapi - the real-time face tracking toolkit for developers and oems', Available at: *http://www.seeingmachines.com/pdfs/brochures/faceAPI-techspecs.pdf* (Accessed: 22nd Jan. 2009).

Shrout, R. (2008), 'Nvidia comments on ray tracing and rasterization debate', Available at: *http://www.pcper.com/article.php?aid=530* (Accessed: 22nd Jan. 2009).

Sun Microsystems (2008), 'Jogl api project', Available at: *https://jogl.dev.java.net/* (Accessed: 22nd Jan. 2009).

Sun Microsystems Inc. (2002), 'Hello universe 1', Available at: *http://www.java2s.com/Code/Java/3D/HelloUniverse1.htm* (Accessed: 22nd Feb. 2009).

Sun Microsystems Inc. (2003), 'Java se desktop technologies - java native interface', Available at: *http://java.sun.com/j2se/1.4.2/docs/guide/jni/* (Accessed: 22nd Jan. 2009).

Sun Microsystems Inc. (2008), 'Java se desktop technologies - java media framework api (jmf)', Available at: *http://java.sun.com/javase/technologies/desktop/media/jmf/* (Accessed: 22nd Jan. 2009).

Takahashi, D. (2009), 'Caustic graphics to create graphics chips with novel ray-tracing technology', Available at: *http://venturebeat.com/wp-content/uploads/2009/03/ray-tracing.jpg* (Accessed: 26th May. 2009).

Tarlier, F. (2009), 'Faceapi with flight simulator 10 (fsx)', Available at: *http://www.vimeo.com/2880313* (Accessed: 26th Feb. 2009).

Tu, J., Tao, H. & Huang, T. (2007), 'Face as mouse through visual face tracking', *Comput. Vis. Image Underst.* **108**(1-2), 35–40.

Turcsanski, A. (2007), 'Computer vision: Opencv support for darpa autonomous vehicle', Available at: *http://www.cs.utah.edu/turcsans/DUC/* (Accessed: 22nd Jan. 2009).

Vo, T. H. (2007), 'Software development process', Available at: *http://cnx.org/content/m14619/latest/* (Accessed: 26th Feb. 2009).

Wang, H. & Chang, S.-F. (1997), 'A highly efficient system for automatic face region detection in mpeg video', *IEEE Transactions on Circuits and Systems for Video Technology* **7**(4), 615–628.

Wang, S., Xiong, X., Xu, Y., Wang, C., Zhang, W., Dai, X. & Zhang, D. (2006), Face-tracking as an augmented input in video games: enhancing presence, role-playing and control, *in* 'Proceedings of the SIGCHI conference on Human Factors in computing systems', ACM.

Wikimedia (2007), 'Perspective projection', Available at: *http://upload.wikimedia.org/wikipedia/commons/thum Projection Principle.jpg/350px-Perspective Projection Principle.jpg* (Accessed: 22nd Jan. 2009).

Yang, L. & Robertson, M. (2000), Multiple-face tracking system for general region-of-interest video coding, Vol. 1, pp. 347 – 350.

Yang, M. H. & Ahuja, N. (2001), *Face Detection and Gesture Recognition for Human-Computer Interaction*, Kluwer Academic Publishers.