Goldsmiths
UNIVERSITY OF LONDON

# solid sight

a project by

Kester Sheridan

MSc Computer Games

# Abstract

Stereopsis (from stereo- meaning "solid", and opsis meaning view or sight) is the process in visual perception leading to the sensation of depth from the two slightly different projections of the world onto the retinas of the two eyes, hence is where the name of the project comes from, Solid Sight. Computer vision has for many years been an active field of research but has taken on added significance in recent years in terms of the video games industry with the imminent release of Sony's Move motion-sensing game controller system for the PlayStation 3 console and Microsoft's Kinect controller-less system for the Xbox 360 console. Both utilise some of the latest developments in computer research and hardware. The Microsoft's system particularly ushers in new era as it dispenses with the need for the player to hold a remote or controller entirely instead using the latest developments in camera technology and AI to detect a person motions. In the course of this report we shall look further into the technology behind these two new pieces of hardware and try and put them into some form of context in the history of computer interfaces as well the new design considerations for such devices. We shall explore some of the latest research into motion capture in terms of full body, head and hands. Ultimately the report will document the implementation of a simple motion capture system that captures the movement of the head and hands of the user.

# Table of Contents

# 1. Introduction

## 1.1 Computer Interface


**Figure 1.1:** Nintendo Wii Games Console with Wii Remote (Wiimote) and Nunchuk controls

With the launch of the Nintendo Wii game console back in 2006, few could have imagined the impact this console with its innovative use of motion control would have on the games market, least of all their competitors Sony and Microsoft. Four years on and Microsoft and Sony are readying themselves to launch their own offerings into the arena of motion controller interfaces.

**Figure 1.2:** Sony PlayStation Move (right) & Supplementary Navigation Controller (left)

**Figure 1.3:** PlayStation Move Controller with Lit Sphere

Sony's offering is entitled Move, a motion-sensing game controller platform for the PlayStation 3 (PS3) video game console which is due for launch in the UK on 17th September 2010 (see figure 1.2). The technology comprises of a handheld motion controller wand and a supplementary navigation controller. The PlayStation Move features an orb at the head which can glow in any of a full range of colours using RGB light-emitting diodes (LEDs) with the colour dynamically selected to distinguish it from the environment (figure 1.3). The coloured light serves as an active marker, the position of which can be tracked along the image plane by the PlayStation Eye camera. The uniform spherical shape and known size of the light also allows the system to simply determine the controller's distance from the PlayStation Eye through the light's image size, thus enabling the controller's position to be tracked in three dimensions with high precision and

accuracy. A pair of inertial sensors inside the controller, a three-axis linear accelerometer and a three-axis angular rate sensor, are used to track rotation as well as overall motion. An internal magnetometer is also used for calibrating the controller's orientation against the Earth's magnetic field to help correct against cumulative error (drift) by the inertial sensors. The inertial sensors can be used for dead reckoning in cases which the camera tracking is insufficient, such as when the controller is obscured behind the player's back.


**Figure 1.4:** Microsoft Kinect (formally Project Natal)

Whilst Microsoft's Kinect (originally code named Project Natal) is based around a web-cam style add-on peripheral for the Xbox 360 games console and is due for launch two months later in November 2010 (see figure 1.4). Kinect is more radical than the other two controllers by Sony and Nintendo as it dispenses entirely with the need for hand-held controllers or remotes altogether instead relying on a series of cameras and sensors to detect the position of players' movements, meaning that controlling a racing car can be achieved by simply holding your hands in a ten-to-two position and turning an imaginary steering wheel left and right. The device features an "RGB camera, depth sensor and multi-array microphone running proprietary software" which provides full-body 3D motion capture, facial recognition, and voice recognition capabilities. The depth sensor consists of an infrared projector combined with a monochrome CMOS sensor, and allows the Kinect sensor to see in 3D under any ambient light conditions. The sensing range of the depth sensor is adjustable, with the Kinect software capable of automatically calibrating the sensor based on gameplay and the player's physical environment, such as the presence of furniture. It is capable of identifying and tracking body parts to within a 4-centimetre cube in space, every 10 milliseconds, using just 10 to 15 per cent of the Xbox's computing resources. Kinect's body-recognition algorithms were trained to recognise body parts by running through millions of images, both human and computer-generated. While the training focused on poses that might be used in games, the system was also taught to recognise more mundane postures and gestures. This gives developers access to a palette of 20 skeletal joints, tracked in 3D, with which to create poses and gestures that the system will respond to[1].

Sony and Microsoft with their new control systems will hope to tap into the lucrative market developed by the Nintendo Wii which widened the demographic for games, attracting the emerging casual market which neither the PlayStation 3 nor Xbox 360 have yet managed to do with their more conventional and perhaps less accessible and less intuitive multiple-button gamepads. This can be seen as part of a broader trend towards computers that understand and interact with humans in real space.

[1] New Scientist article, "Innovation: Microsoft's Kinect isn't just for games" by MacGregor Campbell, 21st June 2010, Web link: http://www.newscientist.com/article/dn19065-innovation-microsofts-kinect-isnt-just-for-games.html


**Figure 1.5:** ICU - Interactive Communication Unit

Sony Europe has been working with Atracsys, a small firm in Lausanne, Switzerland, that specialises in optical tracking to develop a system closer to Microsoft's Kinect, the Interactive Communication Unit (ICU). The ICU system (see figure 1.5) detects the position of specific points on the arms, legs and head to within 10 cubic centimetres and facial expressions using a pattern-matching algorithm to determine people's emotions, sex and approximate age from their appearance. It will initial be launched into the world of advertising allowing for interactive shop windows and billboards and then ultimately with the idea of launching a consumer system later when people have come familiar with the concept[2]. Jamie Shotton of Microsoft Research UK in Cambridge believes that the technology behind the Kinect system will have a great many practical applications outside of the games industry, such as hands-free access to patient files for surgeons, smoother presentation software, intelligent home-security system that can differentiate between family friends and intruders, or remote monitoring systems to help care for elderly people which can detect the patients well-being[3].

### 1.1.1 Motion Control Design

It would seem that such systems will only become more prevalent in the future and may ultimately replace the standard interaction interfaces of keyboard and mouse which have been the norm since the advent of popular computing in 1980s. The vocabulary of interaction: Cut and paste. Save, Windows, the desktop metaphor, that has been developed over the past 40 years can no longer be applied to such devices as these are no longer the same human-computer interaction paradigms that were designed at the beginning of modern computing in the 1960s and 1970s. In a gestural interface, a task as simple as clicking a button, takes on a whole new level of complexity as how do you represent a click as a gesture which will be universally accepted. A button press remains the same no matter how it is done, but a "swipe" can be performed in a variety of ways. Due to the worldwide success of Apple's iPhone and iPod Touch, the 'pinch to

[2] New Scientist article, "Sony demos game controller to track motion and emotion" by Colin Barras, 5th November 2009, Web link: http://www.newscientist.com/article/dn18115-sony-demos-game-controller-to-track-motion-and-emotion.html

[3] New Scientist article, "Innovation: Microsoft's Kinect isn't just for games" by MacGregor Campbell, 21st June 2010, Web link: http://www.newscientist.com/article/dn19065-innovation-microsofts-kinect-isnt-just-for-games.html

shrink' an image is one of the few universally accepted gestures. Its success has led Apple to file a patent for the gesture (US patent number 7479949)[4] and the company currently does not allow any application on its 'apps store' to use it other than its own[5]. Gestural interfaces are in their comparative infancy so such a universally recognized gesture is rare so making the design of motion control schemes trickier to design and develop compared to traditional button-oriented control schemes as the vocabulary does not yet exist.

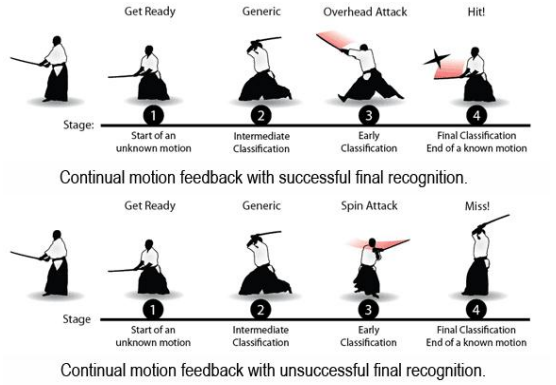


**Figure 1.6:** Motion Recognition Classification

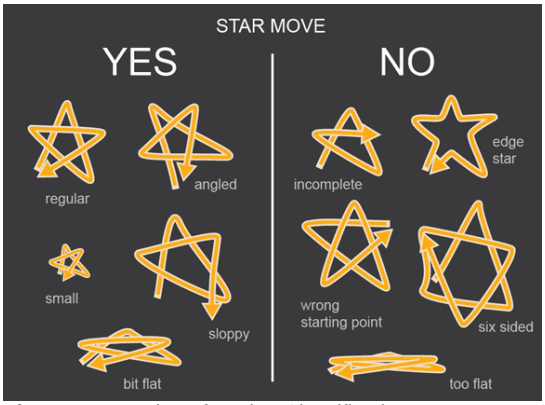


**Figure 1.7:** Design of Motion Classification


**Figure 1.8:** Namco's We Cheer 2 (2010) for the Nintendo Wii takes a "time window" approach to motion segmentation

In terms of the design of gesture interfaces, motion can be classified into three categories.

- **Motion Tracking** provides a 1:1 representation of how a move is being performed, giving the position and orientation of a point, or points, over time (e.g., the trajectory in space of your hand, the tip of a motion-sensitive controller, or joint positions of a 3D skeleton etc.). But tracking doesn't know what the move may signify, so it's like having the ability to see characters on the written page without being able to read them. 1:1 tracking has no lag because it does not categorize and relies on the game simulation to interpret the motion

[4] The Inquirer article "Apple wins multi-touch patent ", by Sylvie Barak, 27th January 2009, Web link: http://www.theinquirer.net/inquirer/news/1050655/apple-wins-multi-touch-patent

[5] Apple Insider article, "Apple rejected iPad app for using pinch to expand gesture" by Neil Hughes, 7th April 2010, Web link:http://www.appleinsider.com/articles/10/04/07/apple_rejected_ipad_app_for_using_pinch_to_expand_gesture.html

input.

- **Motion Recognition** tells the system what type of move is being performed, outputting a high-level "semantic" summary (e.g., "the player is doing an underarm throw") of a large amount of "syntactic" low-level motion sensor data (e.g., numerical time-series data from a single controller, or joint positions on a 3D skeleton etc.). It tries to replicates the human ability to quickly categorize human and animal movement regardless of the huge variety of gaits, body sizes and viewing angles that there can be and therefore it is computational more expensive than motion tracking. As motion occurs over time there is also likely to be a time lag as motion recognition segments continuous motion into discrete categories. This can lead to the player in a game sensing a distinct delay between starting a move and seeing the corresponding on-screen animation.[6] Early feedback is improved if the player's moves are designed so that they diverge from each other as soon as possible (see figure 1.6 & 1.7) or by asking the player to segment their own moves usually by holding down a button on the controller or by allowing the game to segment the motion by asking the player to perform a motion within a time window (e.g., "Ready, Go!") that naturally arises during gameplay and is communicated to the player (see figure 1.8).

- **A Hybrid approach** consists of using a combination of motion recognition and tracking. A useful trick is that games can "cheat" a partial 1:1 visual feedback to the player without using motion tracking. This is sometimes helpful if you want to avoid a full-scale physics simulation of the interaction between a moving arm and the game world, or to avoid requiring players to move in an overly precise and controlled manner. Using motion tracking to give the player precise control for example over a sword, whilst using motion recognition to recognize what the player does with the sword, over and above its physical impact, in order to give qualitative feedback to the player, such as rewards for appropriate or hard-to-perform moves, or for inputs to NPC AI to react to what the player is doing.

Unique to motion capture systems, one must also take into account that people's physical abilities are very different. Left-handed people can generate wildly different motion sensor data compared to right-handed people performing the same move. Different body shapes, especially limb length, are also a source of data variability. Since games are entertainment experiences requiring players to give physical performances, it is important to consider the needs of players with different abilities, so that the games can be enjoyed by as many people as possible. Consequently systems must be robust enough to cope with the variance in the population of game players with motion recognizers with very high "capacity" so that, if required, wildly different data can be recognized as a valid example of the very same move and with the in-built ability to "tune" a motion recognizer to a particular body type "under the hood" without distorting the definition of a valid motion.[7]

[6] Gamasutra article "Taking Games Beyond Whack and Tilt", by Anupam Chakravorty, Rob Kay, Stuart Reynolds, Ian Wright, 27th July 2010, Web link: http://www.gamasutra.com/view/feature/5935/taking_games_beyond_whack_and_tilt.php?page=3

[7] Gamasutra article "Taking Games Beyond Whack and Tilt", by Anupam Chakravorty, Rob Kay, Stuart Reynolds, Ian Wright, 27th July 2010, Web link: http://www.gamasutra.com/view/feature/5935/taking_games_beyond_whack_and_tilt.php?page=4

## 1.2 Brief History of Peripheral Interface Controllers

**Figure 1.9:** Doug Engelbart's Prototype mouse (1968)

**Figure 1.10:** Apple Macintosh Plus Mouse, (1986)

**Figure 1.11:** Alan Kotok, Steve Russell and the author J.M. Graetz play a round of Spacewar! at the Computer Museum in Boston

The origins of the mouse and keyboard, the de-facto standard of the modern computer interface can be traced back to 1960's. In 1963, Douglas Engelbart with the assistance of his colleague Bill English at the Stanford Research Institute invented the first mouse prototype (see figure 1.9), which was later demonstrated to greater effect in what would be later christened 'the mother of all demos'[8] on 9th December, 1968 at the Fall Joint Computer Conference (FJCC) at the Convention Center in San Francisco. The demo featured the first computer mouse the public had ever seen, as well as introducing interactive text, video conferencing, teleconferencing, email, hypertext and a collaborative real-time editor. The first comercially available integrated mouse was introduced in 1970 by the German company Telefunken as a part of their SIG-100 console for the Telefunken TR440 computer. However the mouse remained relatively obscure until the appearance of the Apple Macintosh in 1984 (see figure 1.10). Engelbart never received any royalties for his invention, as his patent ran out before it became widely used in personal computers.

The history of computer game interface design runs pretty much in parallel with that of history of computer interface design which is no surprise as that almost since the advent of computers, people have wanted to write video games for them. In 1962 an MIT student, Steve Russell programmed 'Spacewar!', the very first software-based computer game. Written on the DEC PDP-1 mainframe computer, this space-age dogfight was played on a $30,000 vector monitor using the computer's control panel switches to navigate two warring spaceships as they orbit a deadly sun (see figure 1.11).

**Figure 1.12:** Magnavox Odyssey console (1972)

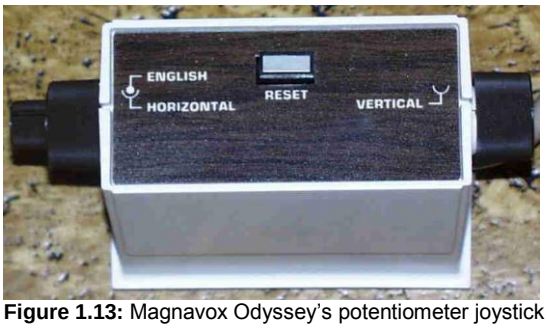


**Figure 1.13:** Magnavox Odyssey's potentiometer joystick

In 1972 the world's first commercially available home video game console, the Magnavox Odyssey (see figure 1.12) was released. It was based on the prototype, the Brown Box designed

[8] Page 42, "Insanely Great: The Life and Times of Macintosh, the Computer That Changed Everything" by Steven Levy, January 1994, Publisher: Viking, ISBN-13: 978-0670852444

by Ralph H. Baer in 1967. It boasts the first video game joysticks, two analogue potentiometers to measure position (see figure 1.13), as well as a rifle light gun, the Shooting Gallery (see figure 1.14) which is regarded as the first commercial light gun ever created for any video game console. However, sales of the console were poor due to poor marketing by Magnavox retail stores with many consumers believing that the console would only work on Magnavox televisions.


**Figure 1.14:** The World's First Commercial Light Gun, The Shooting Gallery (1972)


**Figure 1.15:** Atari 2600 four-switch "wood veneer" version, dating from 1980-1982 with CX40 joystick and Computer port view of the Atari standard connector (insert right)

Just as the Apple Macintosh helped to popularise the mouse as an interface, the introduction of Atari 2600, video game console (see figure 1.15) in October 1977 helped to popularise the joystick as form of interface. The console is also credited with popularizing the use of microprocessor-based hardware and cartridges containing game code, instead of having non-microprocessor dedicated hardware with all games built in. The Atari standard joystick, developed for the Atari 2600 was a digital joystick, with a single "fire" button, and connected via a DE-9 connector, the electrical specifications of which were for many years were seen as the standard digital joystick specification (see figure 1.15). Joysticks were commonly used as controllers in first and second generation game consoles, but then gave way to the familiar game pad (or sometimes called joypad or control pad) with the Nintendo Entertainment System and Sega Master System in 1985 and 1986 (see figure 1.16), though joysticks especially arcade-style ones were and are still popular after-market add-ons for any console. Today, the analogue sticks (or thumbsticks, due to them being controlled by one's thumbs) have become standard on the modern (seventh generation) gamepad having the ability to indicate the stick's displacement from its neutral position. This means that the software does not have to keep track of the position or estimate the speed at which the controls are moved. These devices usually use a magnetic flux detector to determine the position of the stick.


**Figure 1.16:** The Evolution of the Gamepad controller

The early video games consoles particularly the Atari 2600 proved to be hot-bed for the innovative forms of control. Many of the modern interface systems of the seventh generation of consoles can directly be traced back to the first and second generation of consoles. For example in 1982, Atari released the "Joyboard," a simple balance board controller for the Atari 2600 which utilised the four directional latches of the standard joystick (see figure 1.17). Leaning in a certain direction engaged these latches. The board was not a commercial success and only one game was ever released for the Joyboard, Mogul Maniac, a first-person slalom skiing game. Due to similarities in design and function, the Wii Balance Board, a peripheral launched in 2007 for Nintendo's Wii video game console, has been compared to the Joyboard. The Wii Balance Board (see figure 1.18) also allows users to control games with their feet, but uses more advanced technology in the form of pressure sensors under each foot. The modern steering wheel controller of the Xbox 360 (see figure 1.20) or PlayStation 3 may also be traced back to such controllers as the Atari CX20 Driving controller (see figure 1.19) for the Atari 2600 console. The controller may lack the force feedback of the modern steering wheel but allowed for 360° movement when playing the game, Indy 500 of which it was designed to be used with.

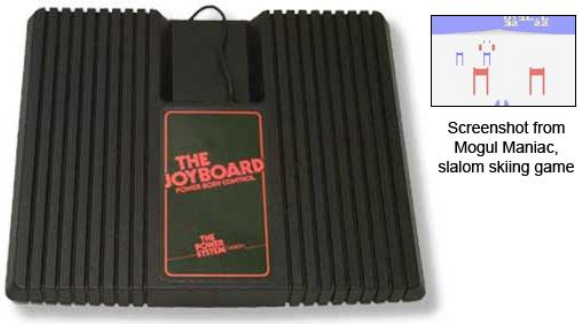


**Figure 1.17:** The Joyboard for the Atari 2600 (1982)


**Figure 1.18:** Nintendo Wii Fit Balance Board (2007)

**Figure 1.19:** Atari CX20 Driving Controller (circa 1980)

**Figure 1.20:** Steering Wheel for Xbox 360 (2009)

The origins of gesture-based interface can also be traced back to the 1980's console era but this time to the third generation era of consoles. The Power Glove (see figure 1.21) was a peripheral interface controller for the Nintendo Entertainment System (NES) released in 1989. It was the first controller to attempt to recreate human hand movements in real time on a television or computer screen. The glove consists of the traditional NES controller buttons on the forearm as well as a program button and buttons labelled 0-9, allowing for the user to input commands such as changing the firing rate. Carbon-based ink flex sensors around the knuckle regions detect the bend of the fingers by causing the carbon to compress so lowering the electrical resistance. There are two ultrasonic speakers (transmitters) in the glove and three ultrasonic microphones (receivers) placed around the TV monitor. The ultrasonic speakers take turns in transmitting a short burst (a few pulses) of 40 kHz sound and the system measures the time it takes for the sound to reach the microphones. A triangulation calculation is then performed to determine the X, Y, Z location of each of the two speakers, which specifies the yaw and roll of the hand. The system cannot measure the pitch of the hand, since the hand can pitch without moving the location of the two ultrasonic speakers. The Power Glove is a modification of the VPL Dataglove design so allowing it to be used with slower hardware and sold at a more affordable price. Whereas the Dataglove can detect yaw, pitch and roll, uses fibre optic sensors to detect finger flexure and has a resolution of 256 positions (8 bits) per four fingers (the pinky finger is not measured to save money, for it usually follows the movement of the ring finger), the Power Glove can only detect roll, and uses sensors coated with conductive ink yielding a resolution of four positions (2 bits) per four fingers[9]. This allows the Power Glove to store all the finger flexure information in a single byte. The glove itself was not a critical or commercial success due in part to the compromises in design and only two games were ever produced for it, Super Glove Ball and Bad Street Brawler. In 2002, the P5 Glove based on similar technology (see figure 1.22) was released for the PC allowing the glove to be used instead of a mouse, however very few games were ever written with 3D support for the glove and ultimately it suffered a similar fate as the Power Glove had.

[9] Mattel PowerGlove FAQ version 0.3, Web link: http://mellottsvrpage.com/glove.htm

**Figure 1.21:** Nintendo Power Glove (1989)

**Figure 1.22:** Essential Reality P5 Glove (2002)

The history of video games consoles is littered with a great many 'wild and wonderful' designs for peripheral interface controllers particular Nintendo who themselves or through licensing deals have produced a great many controller accessories, including DK Bongos controller for playing the rhythm video game, Donkey Konga for the Nintendo GameCube (see figure 1.23). Although other companies have also produced their own fair share of more bizarre controllers such as Sega with the Fishing controller for their Dreamcast console (see figure 1.24).

**Figure 1.23:** Namco's DK Bongos controller (2004)

**Figure 1.24:** Sega Fishing Controller for Dreamcast (circa 1999)

Even with the advent of the Microsoft's Kinect controller-less system, there will still be a market for controllers for consoles either for one off games or genre-specific games. The industry trend is towards a deepening of the medium of interactive entertainment by increasing a player's ability to naturally engage and participate in a virtual world; so meaning the modes of interaction are becoming identical to the modes we use in the real world. Opening a door by reaching forward and grasping its handle is simply a more engaging and natural experience compared to pressing a small button. Many of these controllers can be seen as part of this trend for natural engagement as the controller looks like the thing it represents, a simulacrum (see figures 1.25 and 1.26). Further many of these controllers such as steering wheels controllers also provide vibration feedback so adding to immersive experience and so are likely to continue to co-exist with such devices as Microsoft's Kinect which given its sophistication can still never give such sensory feedback.


**Figure 1.25:** Guitar Hero: World Tour (2008) with controllers representing drums, microphone and guitar

**Figure 1.26:** DJ Hero Turntable controller (2009)

## 1.3 Objectives

The aim of this project is to try and construct a lightweight motion capture system which in turn could be used as a controller-less interface system similar to Microsoft's Kinect to control a character in a game.

For this aim, the system should at least be able to:

- Use motion tracking for the player's head and hands providing a 1:1 representation of their position in real-time which can be mapped on to a 3D game character
- Be able to configure the camera settings to allow the system to work under different lighting conditions
- Work without any special setup in terms of the surroundings such as using a special chroma-key background
- Not require the user to mark the position of their joints via the interface before the system will work
- Be lightweight enough to allow additional game processing to run in parallel

## 1.4 Report Structure

Chapter 2 highlights the latest scientific research which has developed around the subject in the form of papers which have been presented at relevant conferences and published in journals. Chapter 3 gives an overview of the design consideration taken for the project. This is followed by the implementation details in chapter 4. Chapter 5 gives an overall evaluation of the system. We also reflect on the approach we have taken and its limitations in Chapter 6.

# 2. Review of the state of the Art

## 2.1 Motion Capture Overview

Motion capture, motion tracking, or mocap are terms used to describe the process of recording movement and translating that movement onto a digital model. It has a great many applications such as in military, computer animation, sports, medical applications and the field of robotics. For this very reason there has been a great deal of research in recent years surrounding this subject and is only likely to continue to increase as new forms of camera hardware such Time-Of-Flight cameras (see section 2.2) and other technologies are introduced that extend the scope of possible research.

Motion capture originally started as a photogrammetric analysis tool in biomechanics research in the 1970s and 1980s. The methods of extracting motion capture data can be divided into two broad categories; Optical systems and Non-optical systems

### 2.1.1 Optical systems

Optical systems utilize data captured from image sensors to triangulate the 3D position of a subject between one or more cameras calibrated to provide overlapping projections. Data acquisition is traditionally implemented using special markers attached to an actor (see figure 2.1); however, more recent systems are able to generate accurate data by tracking surface features identified dynamically for each particular subject. Tracking a large number of performers or expanding the capture area is accomplished by the addition of more cameras. These systems produce data with 3 degrees of freedom for each marker, and rotational information must be inferred from the relative orientation of three or more markers; for instance shoulder, elbow and wrist markers providing the angle of the elbow[10].

- **Passive Optical** systems use markers coated with a retro-reflective material to reflect light back that is generated near the cameras lens. An object with markers attached at known positions is used to calibrate the cameras and obtain their positions and the lens distortion of each camera is measured. Providing two calibrated cameras can see a marker, a 3 dimensional fix can be obtained (see figure 2.2). Typically a system will consist of around 6 to 24 cameras. Software is used to reduce the problems of markers swapping since all markers appear identical.

- **Active Marker** systems triangulate positions by illuminating one LED at a time very quickly or multiple LEDs with software to identify them by their relative positions. Rather than reflecting light back that is generated externally, the markers themselves are powered to emit their own light. This allows the system to better deal with problems when markers are occluded by props than the passive system.

- **Time Modulated Active Marker** systems are a refinement on Active marker systems by strobing one marker on at a time, or tracking multiple markers over time and modulating the amplitude or pulse width to provide marker ID. 12 megapixel spatial resolution

[10] http://en.wikipedia.org/wiki/Motion_capture

modulated systems show more subtle movements than 4 megapixel optical systems by having both higher spatial and temporal resolution and eliminate marker swapping problems.

- **Semi-Passive Imperceptible Marker** systems use specially built multi-LED IR projectors to optically encode the space. Instead of retro-reflective or active light emitting diode (LED) markers, the system uses photosensitive marker tags to decode the optical signals. By attaching tags with photo sensors to scene points, the tags can compute not only their own locations of each point, but also their own orientation, incident illumination, and reflectance.

- **Markerless** systems do not require subjects to wear special equipment for tracking. Special computer algorithms are designed to allow the system to analyse multiple streams of optical input and identify human forms, breaking them down into constituent parts for tracking.

**Figure 2.1:** Actor and Horse wearing markers as part of MotionAnalysis's Passive Optical Motion Capture System

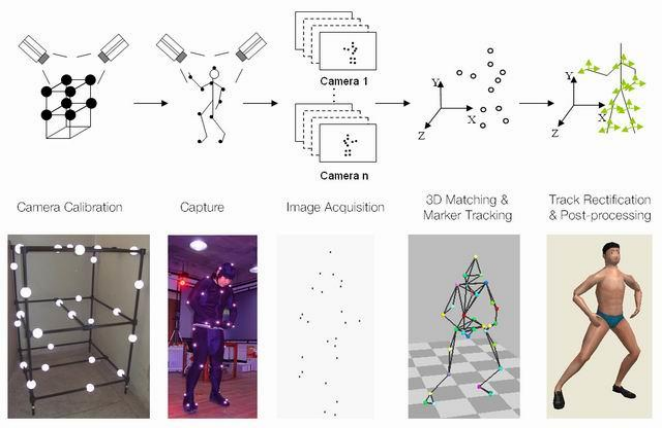


**Figure 2.2:** Stages in Motion Capture using a Passive Optical System

## 2.1.2 Non-Optical Systems

- **Inertial Systems** are based on miniature inertial sensors (usually gyroscopes to measure rotational rates) which transmit wirelessly to a computer whose software combines all of the sensors' data to build a biomechanical model in real time with the full six degrees of freedom. However these systems generally have a lower positional accuracy and positional drift which can compound over time.

- **Mechanical Motion Capture Systems** directly track body joint angles and are often referred to as exo-skeleton motion capture systems, due to the way the sensors are attached to the body measuring the performer's relative motion in real time. They do not suffer from problems of occlusion and are relatively low in price which has made them popular with independent Game development companies. Typically, they are rigid structures of jointed, straight metal or plastic rods linked together with potentiometers that articulate at the joints of the body (see figure 2.3)

- **Magnetic Systems** calculate position and orientation by the relative magnetic flux of three orthogonal coils on both the transmitter and each receiver. The relative intensity of the voltage or current of the three coils allows these systems to calculate both range and orientation to six degrees of freedom using two-thirds the number of markers required in optical systems. The markers are not occluded by non-metallic objects but are susceptible to magnetic and electrical interference from metal objects in the environment. The wiring from the sensors also tends to limit the range of movements available (see figure 2.4).

**Figure 2.3:** Inition Gypsy6 Mechanical Motion Capture Suit

**Figure 2.4:** Magnetic Motion Capture Suit

The remainder of this chapter looks at some of the research which has been made towards markerless optical motion capture which is most relevant to my own field of study and which has been presented at a variety of different conferences including CHI (ACM Conference on Human Factors in Computing Systems)[11], International Conference on Multimodal Interfaces[12], Conference on Human Factors in Computing Systems[13], and at workshops such as Dynamic 3D Imaging Workshop in Conjunction with DAGM[14] and in journals such as “Graphical Models”[15] and “Lecture Notes in Computer Science”[16].

## 2.2 Time-Of-Flight Cameras

Imaging 3-D measurement with useful distance resolution has mainly been realized so far with triangulation systems, either passive triangulation (stereo vision) or active triangulation (e.g., projected fringe methods). These triangulation systems have to deal with shadowing effects and ambiguity problems (projected fringe). Moreover, stereo vision cannot be used to measure a contrast-less scene. This is because the basic principle of stereo vision is the extraction of

[11] See website http://www.chi2010.org/
[12] See website http://icmi2009.acm.org/
[13] See website http://portal.acm.org/toc.cfm?id=1518701
[14] See website http://www.zess.uni-siegen.de/pmd-home/dyn3d/workshops/2009/
[15] See website www.elsevier.com/locate/gmod
[16] See website http://www.springer.com/computer/lncs?SGWID=0-164-0-0-0

characteristic contrast-related features within the observed scene and the comparison of their position within the two images. Also, extracting the 3-D information from the measured data requires an enormous time-consuming computational effort. High depth resolution can only be achieved with a relatively large triangulation base and, hence, large camera systems.

During the last few years, the first all-solid-state 3D-cameras based on the time-of-flight principle became available on the market. A time-of-flight camera (TOF camera) is a camera system that creates distance data with help of the time-of-flight (TOF) principle which is different from time-of-flight mass spectrometry. The principle is similar to that of LIDAR scanners with the advantage that whole scene is captured at the same time.

Time-of-flight cameras are relatively new devices, as the semiconductor processes have only recently become fast enough for such devices. The systems cover ranges of a few meters up to about 60 m. The distance resolution is about 1 cm. The lateral resolution of time-of-flight cameras is generally low compared to standard video cameras, at 320 × 240 pixels or less. The biggest advantage of the cameras may be that they provide up to 100 images per second.

The SwissRanger SR-3000 camera (see figure 2.4) is based on a phase-measuring time-of-flight (TOF) principle. A light source emits a near-infrared (NIR) wave front that is intensity-modulated with a few tens of MHz. The light is reflected by the scene and imaged by an optical lens onto the dedicated 3D-sensor. Depending on the distance of the target, the captured imaged is delayed in phase compared to the originally emitted light wave. Measuring the phase delay, the distances of the complete scene can be determined. The result of the acquisition is a depth map of the scene.


**Figure 2.5:** Messa SwissRanger SR3000 Time-of-Flight Camera

In its most simple form, a light pulse is transmitted by a sender unit and the target distance is measured by determining the turn-around time the pulse needs to travel from the sender to the target and back to the receiver. With knowledge of the speed of light the distance can then easily be calculated. However, the receiver needs to measure with pico-second-accuracy the delay between start and stop, if millimetre-precision is required (6,6ps for 1mm). To realize such system solutions with discrete components, as is done in today's TOF rangefinders, each component in the signal chain must have a very high system bandwidth.

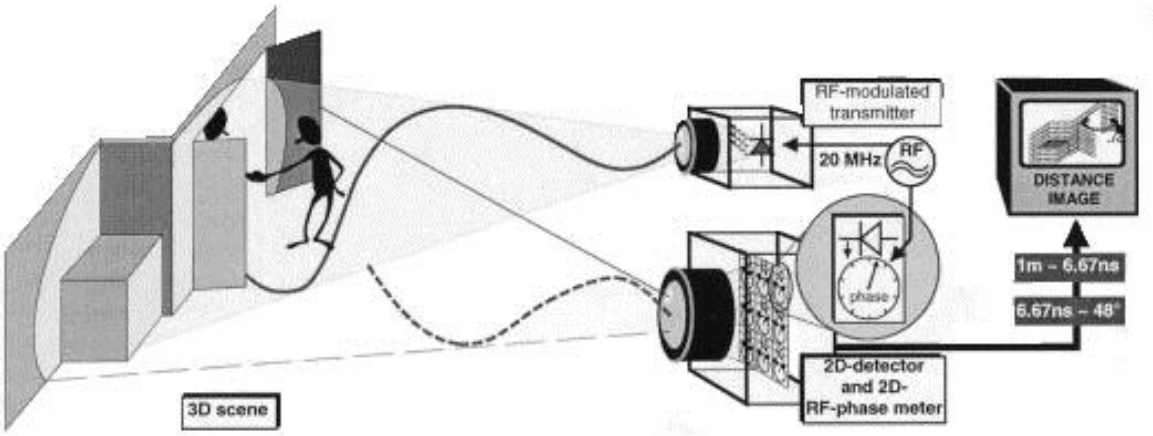


**Figure 2.6:** Principle of a non-scanning 3-D TOF-camera

Photonic Mixer Devices, the so-called PMD sensors, are based on time-of-flight principle. This new camera technology realizes three dimensional imaging without complex electronics and without scanning with a solid state imager similar to a CMOS device. This imager detects both the intensity and the distance in each PMD pixel or voxel respectively.

## 2.3 Research into Full-body Pose Estimation

The goal of recreating human postures while minimally encumbering the end user with sensor attachments (Markerless Optical Motion Capture) is an active research field. One of the main problems encountered is the uncontrolled degrees of freedom, like the swivel angle of the arms, which can lead over time to important differences between the end user and the virtual human model.

An interesting prior work is the one of Wren et al. at the M.I.T MediaLab [Wren]. In this work the authors present a real-time system for the 3D tracking of the upper human body in front of a virtual reality device. No performance evaluation of their system is given however. Moreover the possible gestures are restricted to a set of predefined movements learnt previously. This approximation, reduce the searchable space of human motions by learning-from-example.

The real-time system presented in the paper by Boulic et al. at the Virtual Reality Lab, Switzerland in 2006 is not restricted to a set of predefined movements as in the previous example [Boulic]. For recovering body movements, the user is located in an interactive space that consists of a workbench with two projection screens (see figure 2.7). This space is captured using two standard cameras. The stereo pair is used to capture the motions of certain parts of user's body determined by the particular body posture recovery algorithm. The background wall is covered with chroma-key material for a real-time response and there is only one person in the space with only the skin of their hands and face visible. The system can use both analytic and numeric Inverse Kinematics algorithms to determine the body posture given the end-effectors of the hands and head for comparing relative performance in real-time. Analytical methods find all possible solutions as a function of the lengths of the mechanism, its starting posture and the rotation constraints. Their advantages are their low computational cost and good accuracy compared to other methods and their drawbacks are that they can only be used for low DoF mechanisms and that they are not feasible when the system is ill-posed.

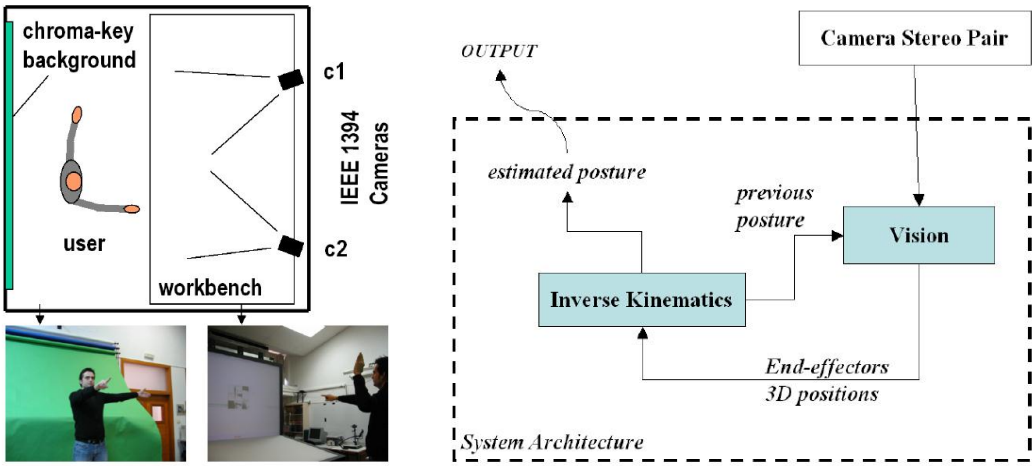


**Figure 2.7:** Vision System Layout and General System Architecture of Boulic et al Motion Capture system

The results of this skin-color detection are skin-color blobs, which are the inputs of the 2D-tracking algorithm. The 3D-tracking algorithm uses a Kalman filter to estimate the end-effector 3D position from the image measurements obtained by the 2D-tracking algorithm so smoothing the estimations between consecutive frames and reducing the positional jitter of the end-effectors that may cause wobbles on the full posture estimation. The 3D position candidates of each end-effector are calculated by triangulating all the possible combinations of 2D measurements from the two images of the stereo pair. Then for each end-effector, the candidate nearest to the position predicted by the estimation filter is selected.

The Inverse Kinematics algorithms require the 3D position of the wrist joints. To locate the wrists from the hands positions, the system uses the 2D ellipses found by the 2D-tracking algorithm, the 3D hand positions from the 3D-tracking algorithm, and the previous 3D positions of elbows estimated by the IK algorithm by searching in the image for the intersection between a 2D line, defined by the back projection of the corresponding elbow and 3D hand positions, and the 2D ellipse. From the 2D wrist positions is to possible to calculate their 3D coordinates by triangulation (see figure 2.8).

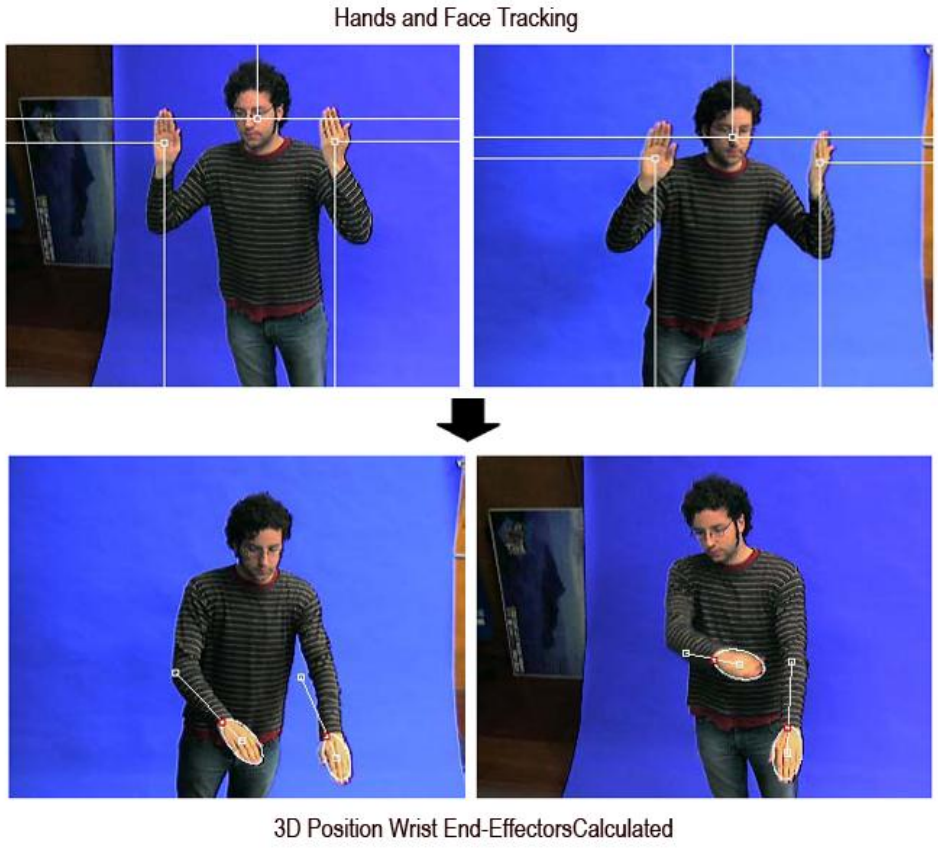


**Figure 2.8:** Hand and Face Tracking with resultant Wrist end-effectors calculated [Boulic]

The user's centre of mass as indicated in figure 2.9 is calculated using the following equation (see equation 2.1 part 1). This algorithm computes the image moments up to order 1 of the user's binary silhouette where $I(x,y)$ is the pixel value at the $(x,y)$ location (1 if the pixel belongs to the binary silhouette or 0 if not). These values are used to find the centre of gravity of the human shape in the image (see equation 2.1 part 2). Both centres of gravity are triangulated to obtain an approximation of the user's 3D centre of gravity.

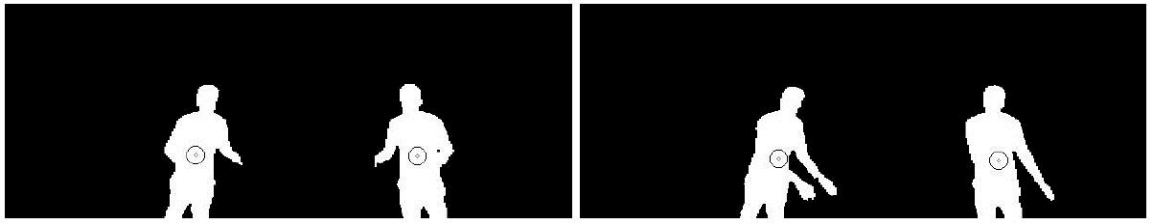

**Figure 2.9:** Centres of Gravity: the weight distribution on both feet is either equal (left) or privileges one foot (right)

$$m_{ij} = \sum_x \sum_y x^i y^j I(x,y), \quad \text{part 1}$$

$$c_x = \frac{m_{10}}{m_{00}}, c_y = \frac{m_{01}}{m_{00}}. \quad \text{part 2}$$

**Equation 2.1:** Two part algorithm for calculating centre of gravity

The system developed by Boulic et al uses a simplified body model which is defined at the calibration stage that occurs in the first frame. This stage consists in obtaining the initial positions for all the joints of the body model and in computing the length of the segments. In order to compute the lengths, the user adopts the posture of Fig. 2.10. Then, the positions for the shoulders and the elbows are manually selected in the first frame and the other joints are automatically located. These initial segment lengths remain constant for the rest of the session. The wrists end effectors are controlled by the IK algorithm. Some joint constraints are added to prevent unnatural posture to appear; in particular the shoulder and elbow joint limits are critical to prevent self-collision or the fully flexed singular posture to occur. The key motivation for modelling the simplified leg and trunk is to associate an approximate mass distribution so that the IK algorithm can simultaneously ensure the balance of the body by controlling the position of its centre of mass[17].

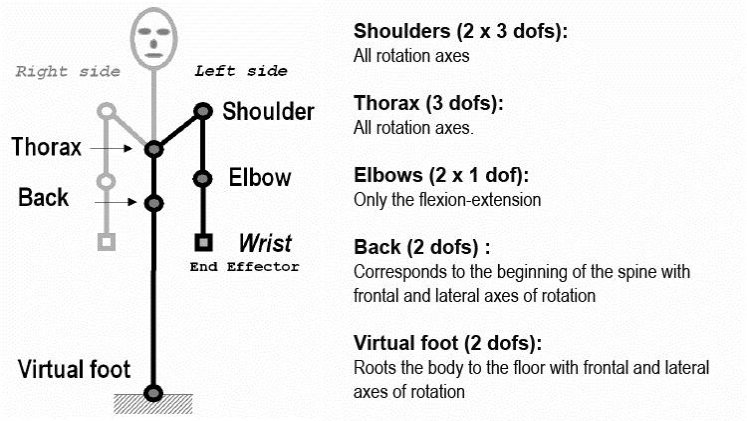


**Figure 2.10:** Simplified body model exploited by the numeric Inverse Kinematics. The kinematic chain involved in controlling the left wrist end effector is highlighted in black. The leg and trunk joints are also used by the right wrist

This system is extended further to the reconstruction of full-body motion in real time by the same team, Boulic et al in their 2008 paper [Unzueta] by using an analytic-iterative Inverse Kinematics (IK) method, called Sequential IK (SIK). This method reconstructs 3D human full-body movements in real time. The input data for the reconstruction are just the positions of wrists, ankles, head and pelvis. As before no pre-recorded full-body pose database is needed.

In SIK poses are reconstructed sequentially, by re-adjusting first the spine, then the clavicles, and finally each of the upper and lower limbs using simple analytic-iterative IK algorithms. The analytic-iterative reconstruction method of full spines and analytic methods for the clavicles, arms and legs, achieves very fast and visually acceptable results. This gives the computer sufficient time to carry out additional processes in real time, such as computer vision algorithms for markerless motion capture, artificial intelligence algorithms for action recognition or for reconstructing the motion of multiple participants simultaneously. No pre-recorded motion database is necessary, thereby avoiding the need for extra memory. First the orientation of the root joint is estimated from the known positions. The estimate the pelvis orientation is calculated as follows:

- **Y** direction defined by the vector that goes from the pelvis to the head which is

[17] Page 12, “Evaluation of On-line Analytic and Numeric Inverse Kinematics Approaches Driven by Partial Vision Input”, Ronan Boulic, Javier Varona, Luis Unzueta, Manuel Peinado, Angel Suescun, and Francisco Perales

subsequently normalized

- X direction defined by a weighted average of three vectors, the first one from the right wrist to the left one, the second one from the right ankle to the left one, both projected and normalized in the plane whose normal is the Y direction, and the third one, the X axis of the previous frame's pose. These weights (called, respectively, $w_1$, $w_2$ and $w_3$) control the dependency level of the torso's axial rotation with respect to the positions of the upper and lower end-effectors.

- **Z** calculated as the cross-product of X and Y

The configuration of the spine is found using a hybrid IK method that combines this estimated orientation with the positions of the root and head markers. Then the orientations of clavicles are determined with the positions of their corresponding known end-effector positions and the already positioned spine. Finally each of the limbs is situated according to their known end-effector positions with an analytic IK method. Complex biomechanical rotation limits are modelled from only a few known anatomical data to constrain the joint orientations and prevent elbows from penetrating the torso in order to obtain visually plausible human poses. Equation 2.2 shows the general procedure of this approach.

1: **procedure** SIK ($pos_{pelvis}, pos_{head}, pos_{wrists}, pos_{ankles}$)
2: Estimate $rot_{pelvis}$ from $pos_{pelvis}$, $pos_{head}$, $pos_{wrists}$ and $pos_{ankles}$
3: Reconstruct the spine from $rot_{pelvis}$, $pos_{pelvis}$ and $pos_{head}$
4: Reconstruct the legs from $pos_{ankles}$
5: Reconstruct the clavicles from $pos_{wrists}$
6: Reconstruct the arms from $pos_{wrists}$
7: **end procedure**

**Equation 2.2:** Sequential Inverse Kinematics Algorithm

## 2.4 Research into Head Pose Estimation

Methods for head pose estimation can be classified into two main categories in the area of estimation with monocular vision: model-based and face property-based. Model-based use 3D model of the face and typically recover the face pose by first establishing 2-3D features correspondences and then solving for the face pose using the conventional pose estimation techniques.

Property-based approaches are the simplest solution and equally the least accurate, Rae and Ritter in their paper build a system with the assumption that there is a unique causal-effect relationship between 3D face pose and certain properties of the facial image [Ritter]. Their goal is to determine the relationship from a large number of training images with known 3D face poses. They use neural networks to construct a mapping between 2D face images and 3D face poses, but, it cannot work well for previously unseen persons and backgrounds. Other proposed solutions use Eigenspace for face detection and head pose estimation [Darrell].

Model-based methods usually start with feature detection, followed by matching 2D/3D corresponding features and determining face pose using the matched features. Among all facial

features, the most commonly used are the eyes, nose and mouth, as is the case in the solution proposed in the paper by Horprasert, Yacoob and Davis [Yacoob].

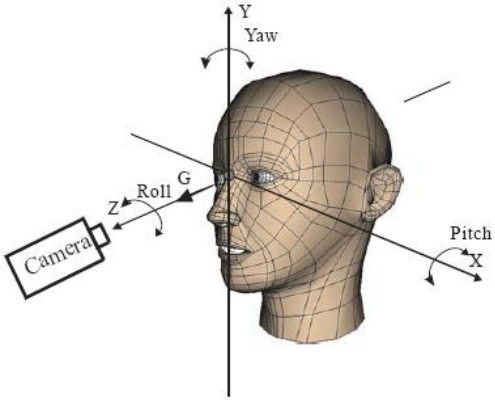


**Figure 2.11:** The definition of three head rotation angles: roll, yaw, pitch and camera position

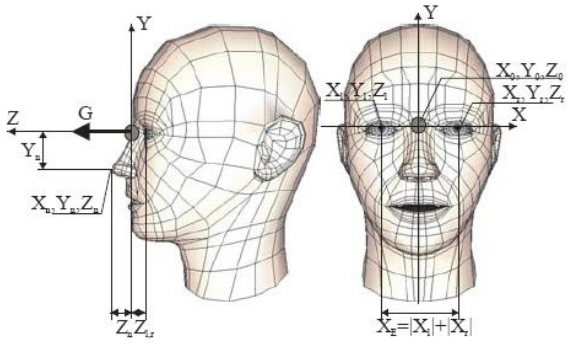


**Figure 2.12**: Diagram illustrating the location of characteristics points and centre of rotation

In the solution proposed by Dervinis, a model-based method for estimation 3D head position is used [Dervinis]. A head model, where 3D rotation centre is located between eyes is created (see figure 2.12). An algorithm for characteristic point of face detection, which characterized head 3D rotation using three angles ϕ, Θ, ψ is discussed. Another algorithm of which characterises the head by ***Z*** and ***Y*** coordinate of the nose tip, the depth of the eyes $Z_l$, $Z_r$ ($Z_l = Z_r$), and the distance between the eyes $X_E$ is also presented.

## 2.5 Research into Hand Gesture Estimation

An off-the-shelf system that can follow delicate hand movements in three dimensions to manipulate virtual objects remains beyond reach of current systems. The problem with systems such as Microsoft's Kinect for the Xbox is that they do not focus on the detailed movement of hands, limiting the degree to which players can manipulate virtual objects. Arm movements can be captured but more subtle pinches or twists of the wrists may be missed. Capturing detail of hands requires optical marker motion-capture systems that utilise sensor-studded data gloves in which flexible sensors detect joint movements. Really accurate gloves cost up to $20,000 and are unwieldy for the user to wear.

**Figure 2.13:** Wang's Gesture System using single camera (left), multi-coloured glove, and desktop application (right)

The system presented in the paper by Wang and Popovic [Wang] uses a consumer hand-tracking technique requiring only inexpensive, commodity components that could bring gesture-based computing to the masses. The Gesture-based system uses a single camera to track a hand wearing an ordinary cloth glove that is imprinted with a custom pattern (see figure 2.13). The key

to the system is the pattern of the gloves, each of which is comprised of 20 patches of 10 different colours - the maximum number a typical webcam can effectively distinguish between (see figure 2.14). The patches are arranged to maintain the best possible separation of colours. For example, the fingertips and the palm, which would frequently collide in natural hand gestures, are coloured differently. The upshot is that when a webcam is used to track a glove-clad hand, the system can identify each finger's location and distinguish between the front and the back of the hand (see figure 2.15).

**Figure 2.14:** Glove design consisting of large colour patches accounts for camera limitations, self-occlusion, and algorithm performance. The length of glove is 24cm


**Figure 2.15:** The palm down and palm up poses map to similar images for a bare hand. These same poses map to very different images for a gloved hand

Once the system has calculated the position of the hand, it searches a database containing 100,000 images of gloved hands in a variety of positions (see figure 2.16). Any more images than this would slows the computer down, and if you have fewer then there is sufficient data to adequately represent all the positions the hand can be in. Once it finds a match it displays it on screen. The process is repeated several times per second, enabling the system to recreate gestures in real time.

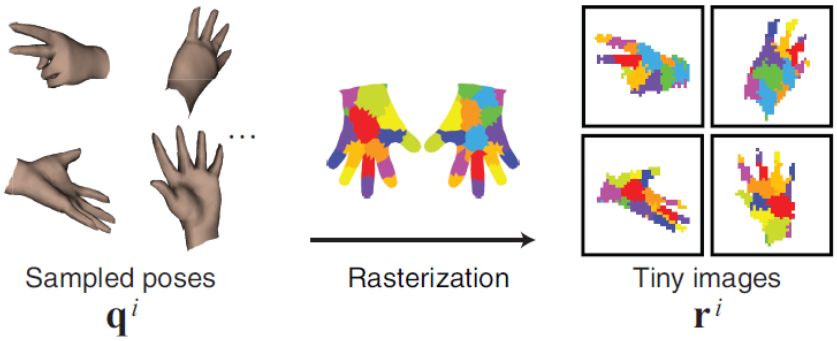


**Figure 2.16:** Pose Database consisting of large set of hand configurations $\{\mathbf{q}^i\}$, indexing each entry by a tiny (40 x 40) rasterized image of the pose $\mathbf{r}^i$

Wang has already shown that the system can correctly replicate most of the letters of the American Sign Language alphabet, although those that require rapid motion (J and Z) or involve the thumb (E, M, N, S and T) have yet to be perfected[18].

Although the gloves cost only about dollar to make, the current system is unlikely to have widespread appeal as wearing a glove is an inconvenience. In the paper by Alaska, Romero and Kragic, a system is demonstrated that shows that markerless motion capture may be possible without the encumberment of wearing gloves [Alaska]. Their system also uses a webcam and a database of hand positions to recreate an on-screen version, but attempts to pick out a bare hand in a stream of video from a webcam by detecting flesh colours. If you reach down and pick up a ball, the program will aim to find a matching image in its database of the positions the hand adopts as it reaches down and picks up a spherical object.


**Figure 2.17:** Data representation using Histogram of Orientated Gradients (HOG)

Identifying a hand using skin colour is far more difficult than picking out a multi-coloured glove. Even once a hand is detected; it is a massive challenge to accurately identify its position - especially if it is holding something as the object occludes parts of the hand so preventing the computer from knowing what the hidden bit is doing.

Once a bare hand is detected, it is a massive challenge to accurately identify its position; therefore a reference database containing images of hands picking up 33 different objects, such as a ball or a cylinder was created. A webcam was set up, which captured 10 frames per second, to test their system's capabilities by filming people grasping a cup, a ball or a pair of pliers. The database had images of a hand picking up a ball, but nothing for a cup or pliers. The system successfully created a virtual representation of a hand grabbing a ball, and came as close as it could to a hand holding a cup by displaying a hand grasping a cylinder. However the system failed to determine hands when people held the pliers.

These results nonetheless show that the system can not only reconstruct the gestures of empty hands, but can also generalise when dealing with some unknown objects. The shape of the pliers, and the grasp used to pick them up were too different from anything in the database for the system to find a match, but by expanding the reference database it should be possible to overcome this.

[18] New Scientist article “Thumbs up for gesture-based computing”, by Shanta Barley, 9th June 2010, Web link: http://www.newscientist.com/article/mg20627635.100-thumbs-up-for-gesturebased-computing.html?full=true

To make identification faster, an algorithm to rule out unlikely hand positions based on previous estimates of hand pose has been incorporated into the system. For example, if the last hand position was a hand stretched out with splayed fingers, the algorithm rules out database images of hands that are clenched into a fist. While this helps the system operate in real time, it creates problems of its own: if the hand moves very fast, it can indeed "jump" from being splayed out to being clenched. In this situation, the set-up struggles because that algorithm will rule out the correct pose. The system has already attracted the interest from makers of prosthetics, who want to improve their understanding about how people grip objects.[19]

[19] http://www.newscientist.com/article/mg20627635.100-thumbs-up-for-gesturebased-computing.html?full=true

# 3. Overall System Design

## 3.1 Hardware Peripheral Design

The Time-Of-Flight (TOF) cameras are much more accurate at determining depth through their hardware than conventional cameras are through software however there is not currently available a TOF camera aimed at the consumer market. There are TOF cameras available for industry on the market such as those produced by Messa (see figure 2.5) however these typically retail for around $3000 and so are beyond the normal consumer budget. In 2009, Israeli developer 3DV Systems were due to release the ZCam, a TOF webcam designed to be used as a game controller using gesture recognition to interpret hand and body gestures for controlling the interface and would retail for around $69.99 in the US[20]. However before the product could be released, the company agreed to sell its assets to Microsoft. It was originally speculated that ZCam technology would be used for Microsoft's Kinect, however it has later emerged that Microsoft is using technology licensed from PrimeSense[21] so it would seem ZCam acquisition was more strategic so meaning that the Kinect device would be the first device of its type aimed at the consumer market. It was therefore decided to use two PlayStation Eye cameras (see figure 3.2).


**Figure 3.1:** Screenshot from EyePet (2009), a game that utilises the Sony PlayStation 3 Eye Camera

This camera was developed by Sony for the PlayStation 3 games console and was launched in 2007. It is the successor to the EyeToy (see figure 3.3) for the PlayStation 2 and was launched with The Eye of Judgment augmented reality role-playing card game in the United States on October 23, 2007. This device has been designed to be used in various real-time image

[20] http://venturebeat.com/2008/07/10/3dv-fleshes-out-gesture-based-gaming-plans-hires-north-american-chief/
[21] "Kinect: The company behind the tech explains how it works" by Mike Schramm Jun 19th 2010, Joystiq Beta ( http://www.joystiq.com/2010/06/19/kinect-how-it-works-from-the-company-behind-the-tech/)

processing and object tracking applications such as The Eye Pet (see Figure 3.1) so seems the ideal choice for the needs of the project due to its high capture frame rate.

It is capable of capturing standard video with frame rates of 60 hertz at a 640×480 pixel resolution, and 120 hertz at 320×240 pixels, which is “four times the resolution” and “two times the frame-rate” of its predecessor, the EyeToy and has an unusually high frame capture rate for a device of this kind. Sony states that the PlayStation Eye can produce “reasonable quality video” under the illumination provided by a television set[22]. The camera features a two-setting adjustable fixed focus zoom lens; a 56-degree field of view similar to that of the EyeToy, for close-up framing in chat applications, or a 75-degree field of view for long shot framing in interactive physical gaming applications.

Although originally designed for the Playstation3 games console, third-party drivers have been developed for it to allow it to run with a PC running Windows or Linux, and are available at Code Laboratories website[23].

**Figure 3.2:** Sony’s Playstation3 Eye Camera

**Figure 3.3:** Sony’s Playstation2 EyeToy

The PlayStation Eye utilises the USB 2 protocol to connect the camera to the host controller. While USB 1.0 was introduced in the early 1996 with transfer speed of around 1.5MB/s, this was superseded by the release of USB 2.0 in April 2000 and the standardized USB-IF at the end of 2001 due to demands by manufacturers for increases in bandwidth requirements. This new protocol was now able to transfer data at theoretical maximum of 60MB/s. In practice this number is lower due to some inherent USB communication overheads and it is more like two thirds of that number, so somewhere around 40MB/s. The USB 2.0 provides four data transfer types:

- Isochronous transfers - guaranteed transfer speed (high priority), but not reliable delivery, mainly used by audio and video streaming devices.
- Interrupt transfers - guarantees delivery latency time, but provides small data packet sizes, mainly used by keyboards, pointing devices and game controllers
- Bulk transfers - guarantees data delivery by using all available data bandwidth and large data transfers, but does not guarantee latency or transfer speed, mainly used by data storage devices

[22] http://www.crunchgear.com/2007/05/01/the-playstation-eye-creation-story-as-told-by-dr-richard-marks/
[23] http://codelaboratories.com/

- Control transfers - very short data transfers (often a few bytes) with guaranteed delivery, mainly used to send device control packets and receive the device status information

The PS3Eye uses bulk transfers to send its image data to the USB host although isochronous transfers would seem to be the most suitable for the camera type devices. These kinds of transfers do not provide any guaranteed latency or speed. As the camera was designed to run on PS3 console for which Sony has a full control of both hardware and software, this choice of protocol is understandable especially as the camera's data stream requires high transfer bandwidth that is more suitable for bulk transfers (uses all available USB bandwidth).

However when using this camera on your PC, this choice of transfer can possibly lead to some problems especially if the camera is shared with some other devices that reside on the same USB bus. The PS3Eye is strictly a high-speed USB 2.0 device, meaning that if you have USB 1.0 port on your computer, the camera simply will not be able to transfer images to the machine. The camera provides the highest quality image sensor data (uncompressed raw data).

For example the 640x480 24bit RGB image captured at 30fps:

- Required Bandwidth = 640*480*30*3 = 27648000 bytes/s = 26.36 MB/s - Which is about 66% of total usable USB 2.0 bandwidth

The 640x480 24bit RGB image captured at 60fps results in:

- Required Bandwidth = 640*480*60*3 = 55296000 bytes/s = 52.73 MB/s - Which exceeds the total usable USB 2.0 bandwidth

Because of all this, most manufacturers of webcams resort to all kinds of tricks when it comes to webcam data transfers. Many use specialized chips on the camera that will compress the image before sending it out to the USB host (using MJPEG for example). Because the image is often compressed by factors of 5, 10 or more (in order to achieve USB 1.0/1.1 compliance), this kind of compression is lossy by nature thus resulting in various unavoidable image artefacts. As a side effect, your computer needs to decompress the received data in order to do display and further processing, all resulting in higher CPU usage than it should be. Some manufacturers claim that their cameras can capture at 60 fps, but in fact what they do is a temporal interpolation in software on the computer end to simulate this high capture frame rate. Some manufacturers simply instruct the camera to capture at say 320x240 pixels and then do spatial interpolation in software resulting in the 640x480 image size. Yet, some manufacturers might use both methods listed above at the same time. The third-party PC drivers which have been developed by Code Laboratories configure the camera to send the raw data directly from the image sensor, turning off any data processing that might exist in between[24].

The API which has been developed exposes the functionality to camera's variables and also manipulates the resultant image in the software. The table on the following page gives details of the variables that are configurable:

[24] http://codelaboratories.com/research/view/ps3eye-not-your-typical-webcam

| Configurable Variables | Possible Values |
|---|---|
| **Colour Mode** | 8 bit Grayscale / 32 bit per pixel RGBA image |
| **Camera Resolution** | 320 x 240 (QVGA) / 640 x 480 (VGA) |
| **Frame Rate (dependant on resolution)** | QVGA - 15, 30, 60, 75, 100, 125<br>VGA - 15, 30, 40, 50, 60, 75 |
| **Automatic Gain** | True / False |
| **Manual Gain** | Between 0 – 79 |
| **Automatic Exposure** | True / False |
| **Manual Exposure** | Between 0 – 511 |
| **Automatic White Balance** | True / False |
| **Manual White Balance: Red** | Between 0 – 255 |
| **Manual White Balance: Green** | Between 0 – 255 |
| **Manual White Balance: Blue** | Between 0 – 255 |
| **Flip Video Horizontally** | True / False |
| **Flip Video Vertically** | True / False |
| **Translate Video by Offset in X** | Between -500 and 500 |
| **Translate Video by Offset in Y** | Between -500 and 500 |
| **Rotate Video Image around centre point** | Between -500 and 500 |
| **Scale Video to simulate digital zoom** | Between -500 and 500 |
| **Lens Correction** | 3 Values Between -500 and 500 |
| **Lens Brightness** | Between -500 and 500 |

**Table 3.1:** Table showing configurable variables through API for third-party drivers for PS3 Eye

## 3.2 Binocular Disparity

Binocular disparity refers to the difference in image location of an object seen by the left and right eyes, resulting from the eyes' horizontal separation. The brain uses binocular disparity to extract

depth information from the two-dimensional retinal images in stereopsis. Human eyes are horizontally separated by about 50-75 mm (inter-pupillary distance) depending on each individual. Thus, each eye has a slightly different view of the world. In computer vision, binocular disparity refers to the difference in coordinates of similar features within two stereo images. The disparity of features between two stereo images is usually computed as a shift to the left of an image feature when viewed in the right image.

There seems to be very little documentation as to what is the optimal distance between two cameras in a stereo-rig should be. The Bumblebee2 Stereo camera produced by Point Grey has gap of 120mm between cameras (see figure 3.4). Figure 3.5 shows the set-up for the cameras in this system.

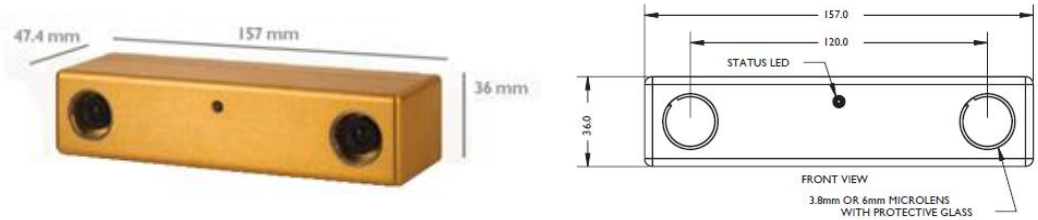


**Figure 3.4:** Bumblebee2 Stereo camera with a Binocular disparity of 120mm between lenses

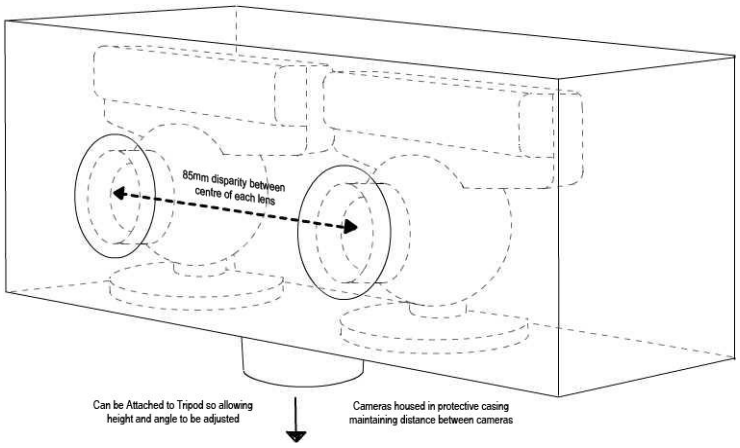


**Figure 3.5:** Stereo Camera set-up for the system

Computers accomplish this task by finding correspondences between points that are seen by one imager and the same points as seen by the other imager. With such correspondences and a known baseline separation between cameras, we can compute the 3D location of the points. Although the search for corresponding points can be computationally expensive, we can use our knowledge of the geometry of the system to narrow down the search space as much as possible. In practice, stereo imaging involves four steps when using two cameras.

1. **Undistortion** is the process of using camera calibration to mathematically correct for the main deviations from the simple pinhole model when using lenses as in practice no lens is perfect especially those generally found in webcams. In manufacturing, it is much easier to make a “spherical” lens than to make a more mathematically ideal “parabolic” lens. It is also difficult to mechanically align the lens and imager exactly. There many are kinds of distortions that occur but typically two main lens distortions are:

- **Radial distortions** arise as a result of the shape of lens. The lenses of real cameras often noticeably distort the location of pixels near the edges of the imager. This bulging phenomenon is the source of the "barrel" or "fish-eye" effect. A typical inexpensive lens is, in effect, stronger than it ought to be as you get further from the centre. Barrel distortion is particularly noticeable in cheap web cameras. The distortion is 0 at the (optical) centre of the imager and increases as we move toward the periphery. In practice, this distortion is small and can be characterized by the first few terms of a Taylor series expansion around **r** = 0. For cheap web cameras, we generally use the first two such terms; the first of which is conventionally called $k_1$ and the second $k_2$. For highly distorted cameras such as fish-eye lenses we can use a third radial distortion term $k_3$. In general, the radial location of a point on the imager will be rescaled according to the following equations:

$$x_{\text{corrected}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{\text{corrected}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

  where, $(x, y)$ is the original location (on the imager) of the distorted point and $(x_{corrected}, y_{corrected})$ is the new location as a result of the correction.

- **Tangential distortions** arise from the assembly process of the camera as a whole. This distortion is due to manufacturing defects resulting from the lens not being exactly parallel to the imaging plane. It is minimally characterized by two additional parameters, $p_1$ and $p_2$, such that:

$$x_{\text{corrected}} = x + [2p_1 y + p_2(r^2 + 2x^2)]$$
$$y_{\text{corrected}} = y + [p_1(r^2 + 2y^2) + 2p_2 x]$$

  Using these five distortion coefficients $k_1$, $k_2$, $p_1$, $p_2$ and $k_3$, it is now possible to undistort the images. More practical details of how the OpenCV library uses these distortion coefficients in one distortion vector, 5-by-1 matrix is given in the next chapter.

2. **Rectification** is the process of mathematically adjusting for the angles and distances between cameras. The outputs of this step are images that are row-aligned meaning the two image planes are coplanar and that the image rows are exactly aligned (in the same direction and having the same y-coordinates) and rectified.

3. **Correspondence** is the process of trying to find the same features in the left and right camera views. The output of this step is a disparity map, where the disparities are the differences in x-coordinates on the image planes of the same feature viewed in the left and right cameras: $x^l - x^r$.

4. **Reprojection** is the step by which if we know the geometric arrangement of the cameras, then we can turn the disparity map into distances by triangulation. The output is a depth map.

Figure 3.6 shows idealised model of two cameras whose image planes are exactly coplanar with each other, with exactly parallel optical axes (for each camera the optical axis is the ray

from the centre of projection *O* through the principal point *c* and is also known as the principal ray) that are a known distance apart, and with equal focal lengths $f_l = f_r$. The model assumes

- The principal points, $c_x^{left}$ and $c_x^{right}$ have been calibrated to have the same pixel coordinates in their respective left and right images. A principal point is where the principal ray intersects the imaging plane. This intersection depends on the optical axis of the lens.

- The images are row-aligned and that every pixel row of one camera aligns exactly with the corresponding row in the other camera. This camera arrangement is called frontal parallel.

- It possible to find a point ***P*** in the physical world in the left and the right image views at $p_l$ and $p_r$, which will have the respective horizontal coordinates $x_l$ and $x_r$.

In this idealised model, taking $x_l$ and $x_r$ to be the horizontal positions of the points in the left and right imager (respectively) it is possible to show that the depth is inversely proportional to the disparity between these views, where the disparity is defined simply by $d = x_l - x_r$. The depth ***Z*** can be derived by using similar triangles.

$$\frac{T-(x^l-x^r)}{Z-f}=\frac{T}{Z} \quad \Rightarrow \quad Z=\frac{fT}{x^l-x^r}$$

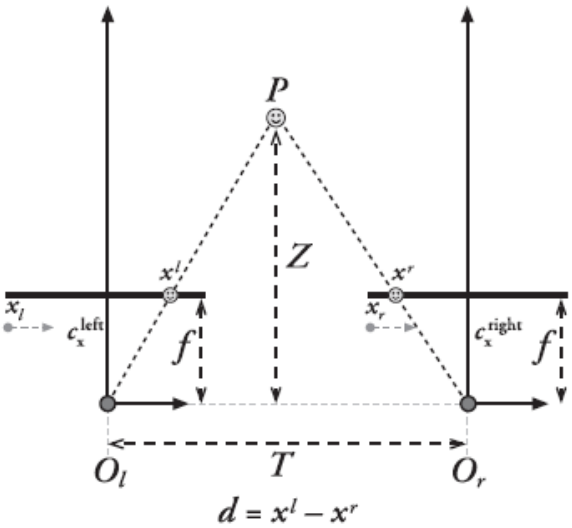


**Figure 3.6:** With a perfectly undistorted, aligned stereo rig and known correspondence, the depth *Z* can be found by similar triangles; the principal rays of the imagers begin at the centres of projection $O_l$ and $O_r$ and extend through the principal points of the two image planes at $c_l$ and $c_r$

Since depth is inversely proportional to disparity, there is obviously a nonlinear relationship between these two terms. When disparity is near 0, small disparity differences make for large depth differences. When disparity is large, small disparity differences do not change the depth by much. The consequence is that stereo vision systems have high depth resolution only for objects relatively near the camera (see figure 3.7). However real-world cameras are not idealised and require stereo calibration and rectification to map them into a geometry that resembles this ideal

arrangement.

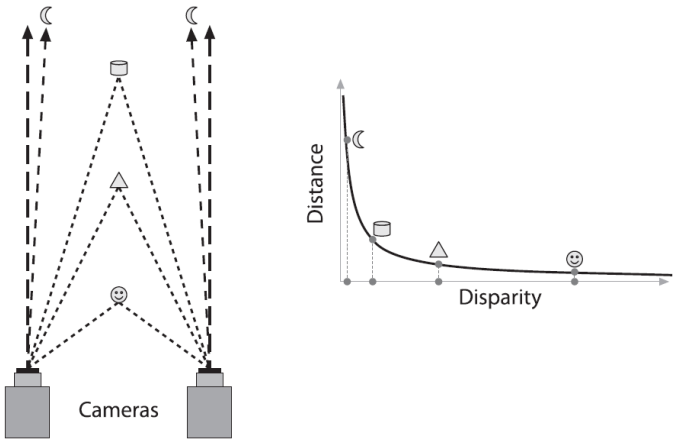


**Figure 3.7:** Depth and disparity are inversely related, so fine-grain depth measurement is restricted to nearby objects

## 3.3 System Architecture

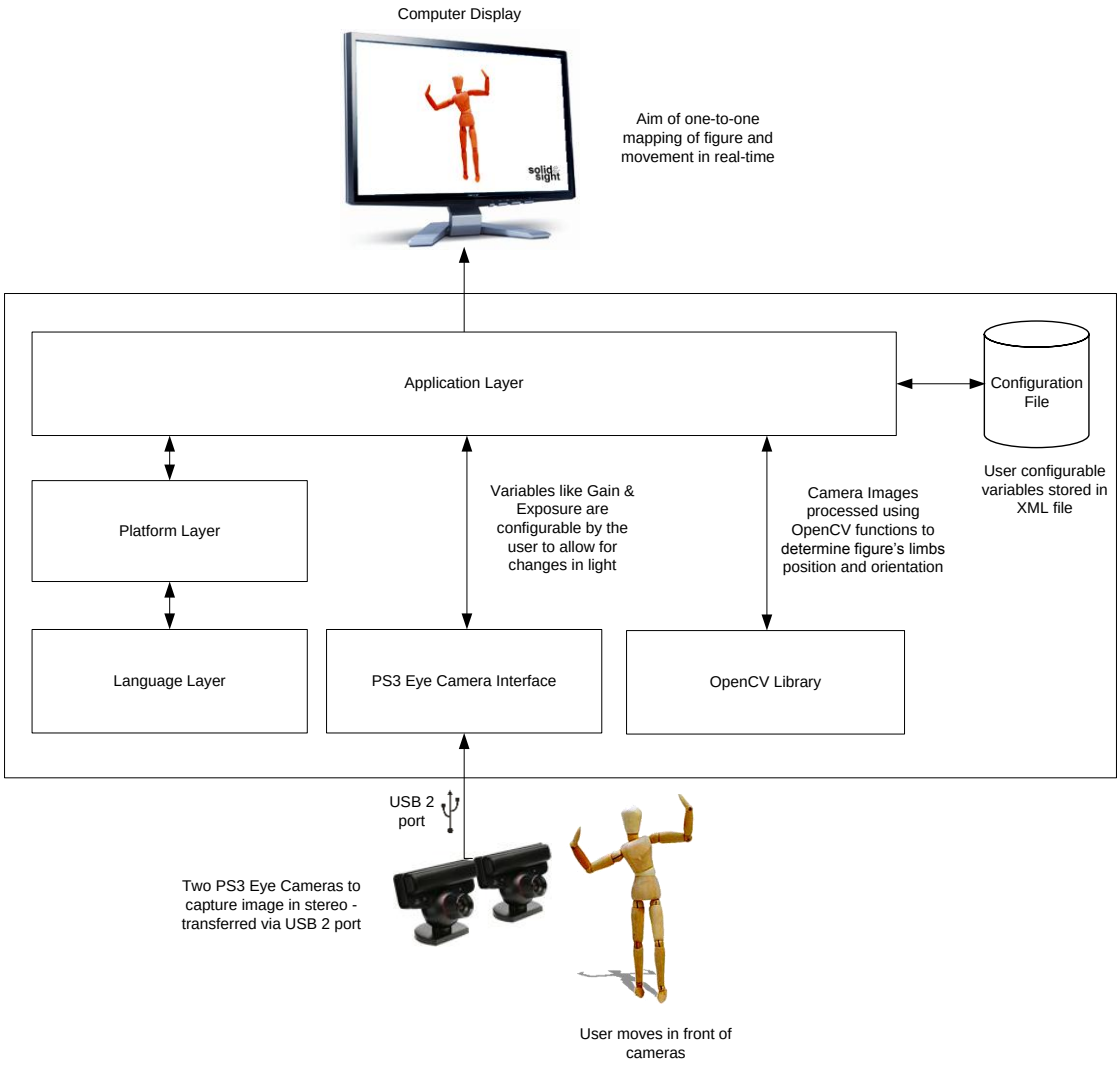


**Figure 3.8:** Level 0 Architecture

The system was developed using C++ and DirectX 9.0c. DirectX 9 was chosen over the later version of DirectX 10 due to more familiarity of this particular version of the SDK and also because the support for 3d models in the `X file` format has been withdrawn in the later version in favour of the `SDKMesh` file format.

The system utilises the OpenCV library. OpenCV is a computer vision library originally developed by Intel as part of their initiative to advance CPU-intensive applications. It is free for use under the open source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. The library is mainly written in C, which makes it portable to some specific platforms. However, since version 2.0, OpenCV includes both its traditional C interface as well as a new C++ interface that seeks to reduce common programming errors when using OpenCV in C. Much of the new developments and algorithms in OpenCV are in the C++ interface. There were several goals for OpenCV at the outset:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.

- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.

- Advance vision-based commercial applications by making portable, performance optimized code available for free—with a license that did not require commercial applications to be open or free themselves.

Using OpenCV is advantageous as it gives you a valuable head start in developing your own vision application instead of having to reinvent the basic functions from scratch so allowing you to build on top of what came before so enabling ambitious projects to be developed.

# 4. Implementation

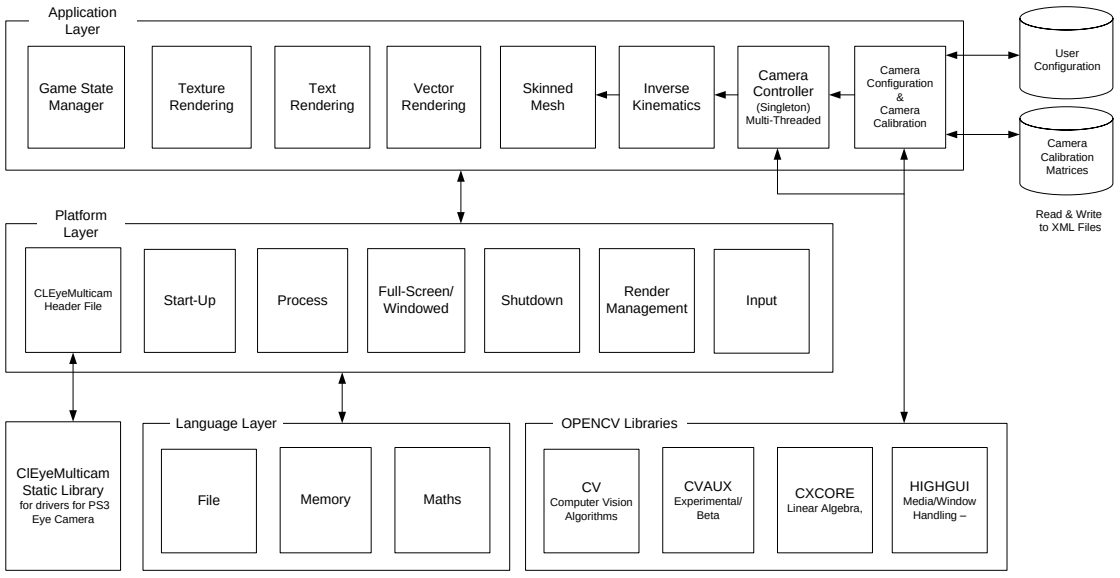


**Figure 4.1:** Level 1 System Architecture

## 4.1 Performance

As any form of image processing is notorious for being CPU-intensive and the fact that the aim was to try and achieve this in real-time, it was decided early on to try and maximise the performance of the code. One possible way to achieve this was suspend the normal way Windows applications behave. Normally Windows' applications wait for the Windows OS to send them application messages and only then do they do any processing when they receive these messages. This is required so that other applications can get processor time and Windows runs smoothly. However by changing the message loop in the `WinMain` function to no longer wait for the Windows messages it is possible to make the application run as fast as possible by hogging the CPU time. Consequently it is important to add a pause into this loop when the application is minimised or loses focus to allow normal Windows messages to resume so that other applications can run properly.

The biggest performance enhancement that can be made is to make the application multi-threaded, particularly with the image processing running in separate parallel threads to the rendering to the screen. Threads can simplify program design and implementation and also improve performance, but thread usage requires care to ensure that shared resources are protected against simultaneous modification and that threads run only when appropriate. This is particular pertinent within this system as there will be a large number of threads relying on the image data produced by the cameras. Therefore the code needs to be made thread-safe, i.e. several threads can execute the code simultaneously without any undesirable results. The core problem is that there is a critical code region such that, once a thread starts to execute the critical region, no other thread can be allowed to enter until the first thread exits from the code region. This critical code region problem can be considered a type of race condition because the first

thread “races” to complete the critical region before any other thread starts to execute the same critical code region. Thus, we need to synchronize thread execution in order to ensure that only one thread at a time executes the critical region.

### 4.1.1 Multi-threading using Boost Libraries

Originally it had been decided to use the thread library within the Boost set of libraries. It provides an object-oriented wrapper around the C-style `pthread` allowing for a more flexible solution. The `boost::thread` class represents a thread of execution in the same way that the `std::fstream` represents a file. Each thread has its own stack for automatic storage class objects. `Boost::thread::join` can wait for a thread to complete.

Mutexes are the main synchronization tools in the `Boost::thread` library. Mutexes have two main forms; simple mutex and recursive mutex. The simple mutex can be locked only once. If the same thread tries to lock such a mutex twice, it deadlocks. With a recursive mutex, a single thread may lock the same mutex several times and must unlock the mutex the same number of times. A thread may lock a mutex in three ways; with blocking, returning immediately on failure and wait until timeout. Consequently there are six kinds of mutexes: `boost::mutex`, `boost::try_mutex`, `boost::timed_mutex`, `boost::recursive_mutex`, `boost::recursive_try_mutex`, `boost::recursive_timed_mutex`.

To use the Boost thread library it is necessary to compile the libraries locally on the machine unlike some of other functionality which is available within the Boost libraries which are available in the pre-compiled headers. This is very involved process as the compilation is not very intuitive. However critically it would seem that the dynamic link-library (or dll) generated is not compatible with the static library which has been written for the P3 Eye Camera. Therefore it was decided to abandon this library in favour of the multi-threading available within the standard Windows libraries.

### 4.1.2 Multi-threading using Windows

Windows provides the `CRITICAL_SECTION` object as a simple “lock” mechanism for implementing and enforcing the critical code region concept. `CRTITICAL_SECTION` (CS) objects are initialized and deleted but do not have handles and are not shared with other processes. They have the advantage of not being kernel objects and are maintained in user space. This almost always provides performance improvements compared to using a Windows `mutex` kernel object with similar functionality, especially in Windows 2000 and later[25]. Threads enter and leave a CS, and only one thread at a time can be in a specific CS.

`EnterCriticalSection` blocks a thread if another thread is in the section, and multiple threads can wait simultaneously on the same CS. One waiting thread unblocks when another thread executes `LeaveCriticalSection`; you cannot predict which waiting thread will unblock. A thread owns the CS once it returns from `EnterCriticalSection`, and `LeaveCriticalSection` relinquishes ownership. By using an `_finally` block in each method which calls the `LeaveCritcalSection` means that uses CS will not be blocked as this part of code is always executed even if an error is thrown as there is no timeout for `EnterCriticalSection`.

[25] Page 280, Chapter 8, "Windows System Programming (Addison-Wesley Microsoft Technology)", by Johnson M. Hart, Publisher: Addison Wesley; 4th edition (16 Feb 2010)

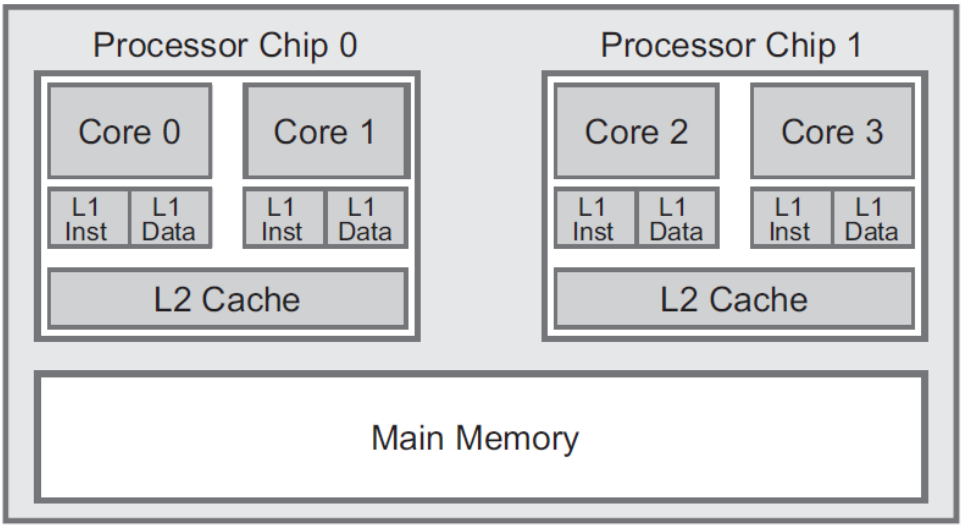


**Figure 4.2:** Memory Sub-system Architecture of a typical multiprocessor computer (two dual-core chips)

An optimizing compiler might leave the value of a variable in a register rather than storing it back in the variable. An attempt to solve this problem by resetting compiler optimization switches would impact performance throughout the code. The correct solution is to use the ANSI C `volatile` storage qualifier, which ensures that the variable will be stored in memory after modification and will always be fetched from memory before use. The `volatile` qualifier informs the compiler that the variable can change value at any time. However, the `volatile` qualifier can negatively affect performance, so it should only be used when required.

Using the `volatile` keyword only assures that the new data value will be in the L1 cache (see figure 4.2); there is no assurance that the new value will be visible to threads running on other cores. A memory barrier, however, assures that the value is moved to main memory and assures cache coherence. Moving data between main memory, processor cores, and caches can require hundreds of cycles, whereas a pipelined processor can access register values in less than a cycle[26].

### 4.1.3 Resolution

From the outset, it was the intention for the application to be able to run in both windowed mode and full-screen mode (i.e. the application covers the entire desktop, hiding all other running applications). Games generally run in full-screen mode for a fully immersive experience, however this requires a separate monitor or a remote debugger for debug purposes so it is advantageous for the application to be able to run in both. To enable full-screen mode in DirectX the `Windowed` property of the `D3DPRESENT_PARAMETERS` struct is set to false and for windowed mode it is set to true. The back buffer dimensions, `BackBufferWidth` and `BackBufferHeight` of the `D3DPRESENT_PARAMETERS` need to be set to match the client area dimension to avoid the image being stretched or loosing quality. When switching between full-screen mode and windowed mode such resources as the `IDirect3DDevice9`, `ID3DXSprite`, `ID3DXFont` and `ID3DXEffect` need to be reset generally be calling their `OnLostDevice()` and

[26] Page 264, Chapter 8, "Windows System Programming (Addison-Wesley Microsoft Technology)", by Johnson M. Hart, Publisher: Addison Wesley; 4th edition (16 Feb 2010)

`OnResetDevice()` functions. For this reason in the code, all of these resources are declared using the `extern` keyword specifying that they have external linkeage (i.e. can be used by the other files in the project) and are static in duration, allocated when the program begins and deallocated when the program ends. They are declared in the `Winapp` class which contains the main windows message loop and of which the `GameDemo` class is derived, the module containing the `WinMain` method (required to run the code as an executable). In the `GameDemo` class these resources such as `ID3DXFont` are allocated and deallocated. When in full-screen mode, the code must be able to handle the different aspect ratios (typically 4:3 for standard SVGA or 16:9 for wide screen) as well as handle being resized in windowed mode. This is achieved by first specifying a minuim windowed size by trapping the windows message `WM_GETMINMAXINFO` and setting the `ptMinTrackSize` values for `x` and `y`. Secondly, the design of the layout must take into account the minium size of the window so the information will be displayed correctly. By setting the flag of `D3DXSPIRITE_OBJECTSPACE` of `ID3DXSprite` will mean the sprite's coordinates are relative to its own local coordinate system as opposed to screen space (i.e. in units of pixels with positive y-axis going down and the positive x-axis going right) so that when the window is resized in window-mode or in full-screen mode, the images will be resized and their positions altered relative to their own coordinate space.

## 4.2 Menu System

As highlighted in section 3.1, the PS3 Eye camera third party drivers allow for the camera to be configured via code through the API (`CLEyeMulticam.h`). As the computer vision is notoriously susceptible to changes in lighting conditions, it was deemed to be important to allow the end-user to be able to configure each of the two cameras and any of the various other properties within the OpenCV setup which were important. Figure 4.2 shows the camera properties that can be altered via the menu system. These in turn are saved to an XML file stored locally so that when the application is started again these settings can be loaded up and applied to the specific camera.

Originally it was the intention to use modal dialog boxes for this configuration similar to way many windows applications such as Microsoft Word do when the user can configure properties. I investigated using the Microsoft Foundation Classes (or MFC), a library that wraps a portion of the Windows API in C++ classes, to create modal dialog boxes however I discovered that the MFC did not seemed work with the DirectX components I was using and caused the compiler to throw a number of errors. I subsequently tried circumventing the MFC, using the Windows API directly which is slightly less intuitive and more involved to use. However I realised that as I wanted the application to be able to run in full-screen mode, this would lead to problems when showing modal dialog boxes although it is possible to allow DirectX to cater for these by setting `IDirect3DDevice9::SetDialogBoxMode()` however this has certain restrictions. Therefore I decided to create my own menu system and custom controls using DirectX. The look and feel of the menu system was inspired by the menu system of the game GTA 4 (see figure 4.3).

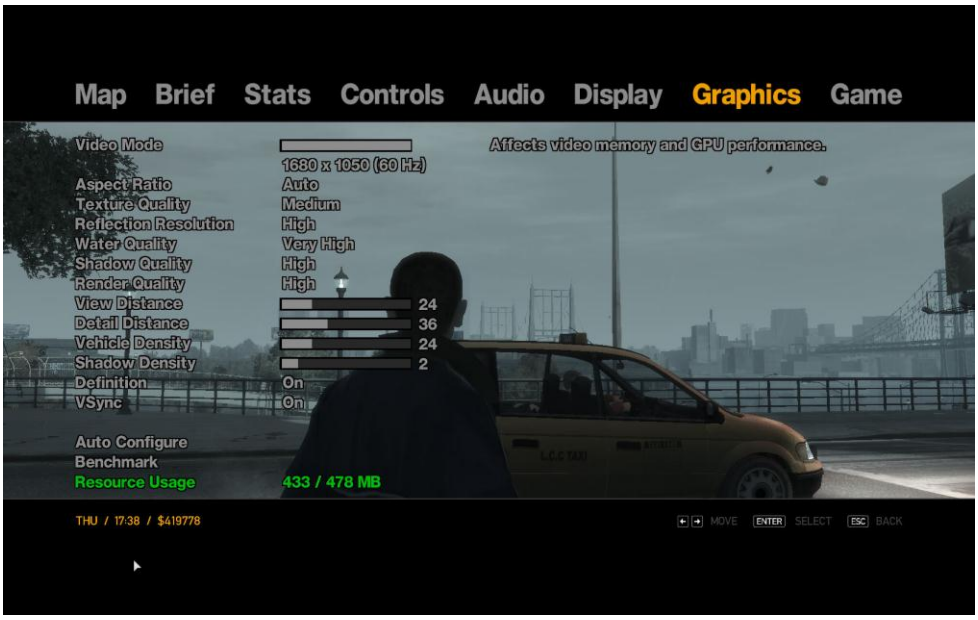


**Figure 4.3:** Screenshot from the video game GTA 4, illustrating the menu layout


**Figure 4.4:** Screenshot from menu system which allows user to configure cameras

The OpenCV library uses the image file format, `IplImage` which it inherits from the Intel Image Processing Library. This file format is not supported by DirectX and does not have any direct conversion. The DirectX function `D3DXLoadSurfaceFromMemory` allows for an easier conversion to display the image to a `LPDIRECT3DSURFACE9` surface texture of an `LPDIRECT3DTEXTURE9` texture without any time or process consuming conversion.

## 4.3 Skinned Mesh

This type of structure is created from several bones linked together in a hierarchical fashion. A bone usually has a parent bone and zero or more child bones. Any transformations applied to a parent bone also affect its children (and their children, and so on). After you have built or loaded a hierarchical bone structure, you need to apply a mesh to it so that each vertex in the mesh is linked to one or more bones.

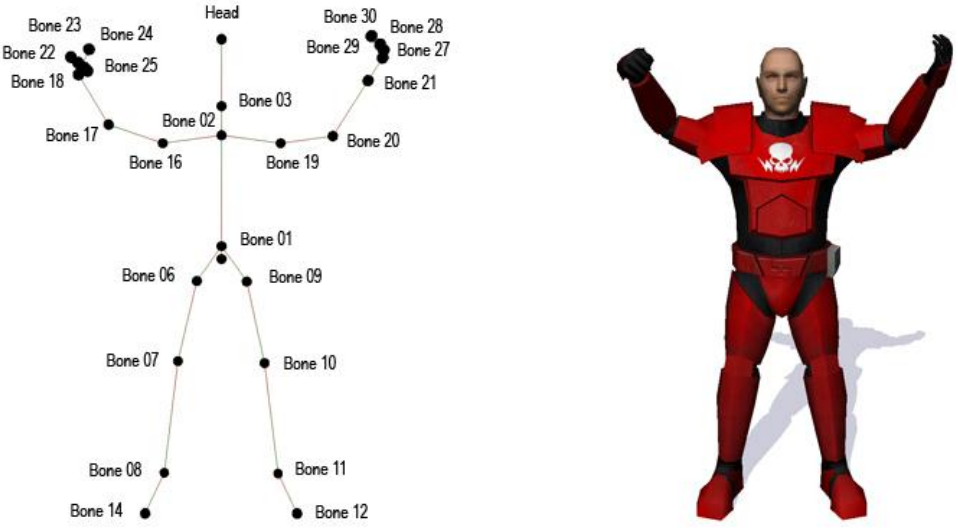


**Figure 4.5:** Skeleton with named bones and corresponding Skinned Mesh

When rendering we start from the top of the hierarchy and recursively travel down through the hierarchy. Each frame has an associated matrix, so you can see that by altering the rotation of the left arm frame all the upper and lower arm frames are also rotated. In skinning, frames are called bones and there tends to be just one mesh controlled by these bones.

When the model is positioned in the 3D world the first frame (known as the root) is transformed to the correct position. Each part of the model is positioned based on the frame matrix translating and rotating it from the frame above. By maintaining this structure we can animate parts of the model simply by altering one frame matrix. The frames can be named in a 3D package and then these names can be used in the code to find the correct frames to animate. The model is positioned based on the frame matrix translating and rotating it from the frame above.

In Direct3D, bones are described using the `D3DXFRAME` structure, and it is with this structure that the complete bone hierarchy will be built.

```
struct D3DXFRAME {
      LPSTR Name; //Name of bone
      D3DXMATRIX TransformationMatrix; //Local bone pos, rot & scale
      LPD3DXMESHCONTAINER pMeshContainer; //Mesh connected to bone
      D3DXFRAME* pFrameSibling; //Sibling bone pointer
      D3DXFRAME* pFrameFirstChild; //First child bone
};
```

The `D3DXFRAME` structure contains `Name` which can be used to find specific bone in the hierarchy. Each bone also has a transformation matrix describing its position, orientation, and scale in relation to its parent bone. This matrix describes transformations in local space only.

The main method for loading animation data, skin info and hierarchy is `D3DXLoadMeshHierarchyFromX`. Before calling this function it is necessary to create `ID3DXAllocateHierarchy` derived class to handle the creation of memory for our frame and mesh data. `ID3DXAllocateHierarchy` is an interface class that defines the functions that must implement. It is necessary to implement the following functions:

1. `CreateFrame` - requests allocation of a frame object
2. `CreateMeshContainer` - requests allocation of a mesh container object
3. `DestroyFrame` - deallocation of frame object
4. `DestroyMeshContainer` - deallocation of mesh container object

These will be called during the internal processing of the x file carried out by the `D3DXLoadMeshHierarchyFromX` function.

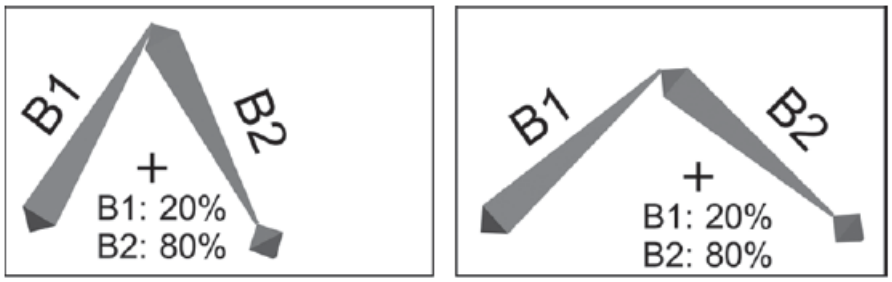


**Figure 4.6:** An example of how a vertex (the cross) is affected by two bones (B1 and B2) with the weights 20% and 80%, respectively. Notice how the vertex follows B2 more than B1 due to the weights.

A vertex can be linked (influenced) by one or more bones in the bone hierarchy. The amount a bone influences a vertex is determined by a weight value as shown in Figure 4.6. In more mathematical terms, the transformation matrix applied to a vertex is defined as follows:

$$M_{Tot} = (w_0 M_0 + w_1 M_1 \ldots + W_n M_n)$$

This formula multiplies the bone weight ($w_x$) with the bone transformation matrix ($M_x$) for all influencing bones and sums up the result ($M_{Tot}$). The resulting matrix is then used to transform the vertex. In DirectX, the information about which bones influence which vertices, as well as their respective weights, etc., is stored and controlled with the ID3DXSkinInfo interface.

Rendering a skinned character can either be done by using the CPU (called software skinning) or directly with the GPU (graphics processing unit, i.e., the graphics card) as the mesh is being rendered called hardware skinning. The system uses hardware skinning using a vertex shader to do the skinning calculations as the mesh gets skinned on-the-fly in the GPU rather than pre-processed each frame as is the case with software skinning. This makes hardware skinning also considerably faster.

## 4.4 Inverse Kinematics

The principle of an inverse kinematics solution to a motion problem is that, given a position in world space and the end point of a linked chain (end-effector), determine the rotations that have to be applied to the linked chain in order that the end effector reaches the goal. Most of the study of this problem stems from robotics. Each joint can be rotated in one or more directions. A single joint that rotates in the heading, pitch and bank is described as having three degrees of freedom (3 DoF). For an object to be completely free moving, it must have six degrees of freedom (6 DoF).

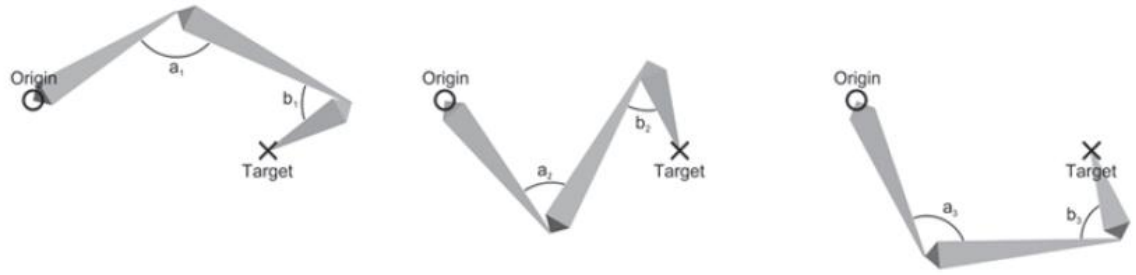


**Figure 4.6:** 2D Example of Inverse Kinematics with 3 possible solutions

The solutions to IK problems can be divided into two categories; analytical and numerical. Analytical solutions are generally the preferred way for simple IK chain with few links as they have an equation that can be solved directly. Numerical solutions attempt to find approximate (but somewhat accurate) solutions to the problem. The approximations are usually done by either iterating over the result, which finally converges toward the solution, or dividing the problem into smaller, more manageable chunks and solving those separately. Numerical IK solutions also tend to be more expensive compared to their analytical counterpart.

Two of most popular numerical methods for solving IK problems are Cyclic Coordinate Descent (CCD) and the Jacobian matrix. Cyclic coordinate decent simplifies the problem by looking at each link separately. CCD starts at the leaf node and moves up the chain, trying to minimize the error between the end point and the goal. However this approach can require a fair amount of passes over the IK chain before the result is acceptable and sometimes creating unnatural-looking solutions; for example, since the first link to be considered is the leaf (e.g., the wrist or ankle), it will first try to minimize the error on this link, which might result in really twisted hands or feet.

### 4.4.1 Single Chain Inverse-Kinematics

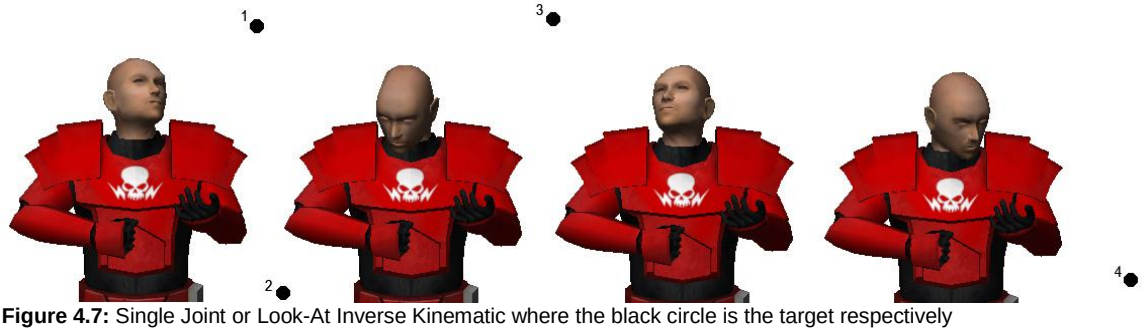


**Figure 4.7:** Single Joint or Look-At Inverse Kinematic where the black circle is the target respectively

The rotation of the head of the model is restricted to view cone of 120° to allow for more

naturalistic movement. The head is allowed to turn to look-at an imaginary point anywhere in this cone. The constructor of the `InverseKinematics` class takes a pointer to the skinned mesh. The constructor finds the head bone and does the necessary initializations for the IK class. The `ApplyLookAtIK()` function takes a Look-At target (in world space) and a max angle defining the view cone (FoV) of the character.

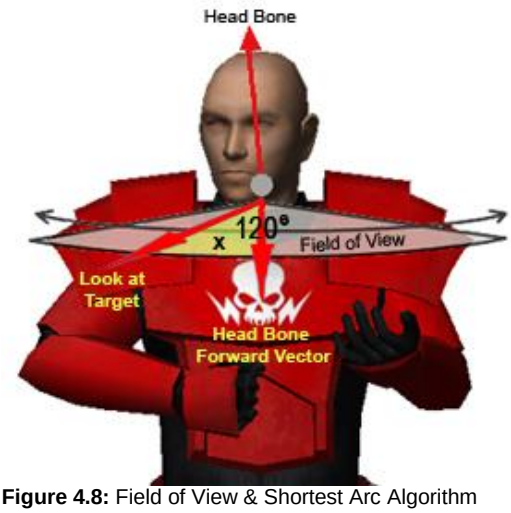


**Figure 4.8:** Field of View & Shortest Arc Algorithm for calculating Look-At Inverse Kinematics

The `ApplyLookAtIK()` function uses the shortest arc algorithm [Melax], to calculate the angle to rotate the head bone so that it faces the Look-At target. First the head bone is located and then the transformation is removed from the combined transformation matrix by setting elements 41, 42, 43, and 44 in the matrix to 0, 0, 0, and 1 respectively. The inverse of the resulting matrix is calculated so allowing for the head forward vector (in the local head bone space) to be calculated (see Figure 4.8). The forward vector of the head bone is calculated when the character is in the reference pose and the character is facing in the negative Z direction. The normalized vector to the target in bone space is calculated (since the head forward vector is in bone space). The cross product of these two vectors (head forward and target vector) is used as the rotation axis. Then the angle is calculated and capped to the max rotation angle and used to create the new rotation matrix (this is the matrix that will turn the head to face the target). Finally the combined transformation matrix is updated with the new rotation matrix and any child bones of the head bone are updated as well using the `SkinnedMesh::UpdateMatrices()` function.

### 4.4.2 Two-joint Inverse-Kinematics

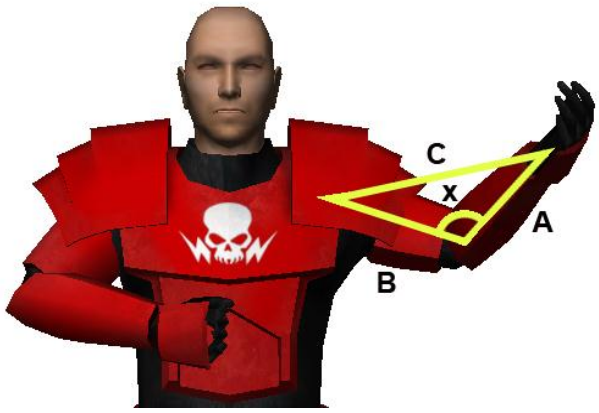


**Figure 4.9:** Two-joint Inverse Kinematic showing Law of Cosines Algorithm with X being the rotation angle

In the figure above, ***C*** is known because it is the length from the shoulder to the IK target. ***A*** and ***B*** are also known because they are simply the length of the upper and lower arm. To solve the angle **x**, the following formulae is used:

$$x = a\cos\left(\frac{A^2 + B^2 - C^2}{2AB}\right)$$

First the elbow is bent to the angle that gives the correct "length" for ***C***. Then the shoulder is rotated using the same method as the head turning previously.

## 4.5 Stereo Calibration

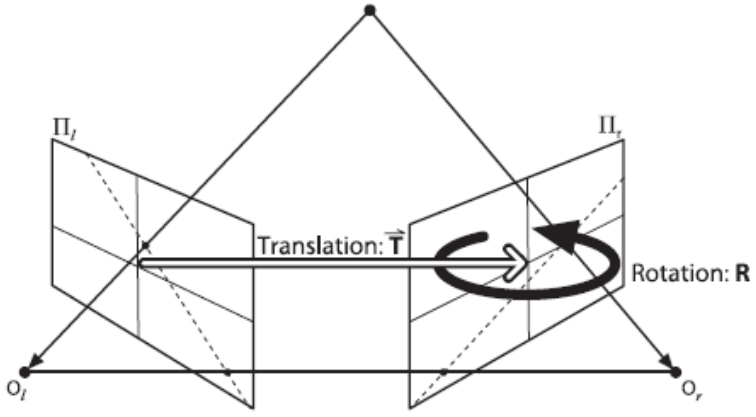


**Figure 4.10:** The essential geometry of stereo imaging is captured by the essential matrix `E`, which contains all of the information about the translation `T` and the rotation `R`, which describe the location of the second camera relative to the first in global coordinates

Stereo calibration depends on finding the rotation matrix **R** and translation vector **T** between the two cameras, as depicted in Figure 4.10. Both **R** and **T** are calculated by the OpenCV function `cvStereoCalibrate()`. The function calculates a rotation matrix and translation vector that relate to both the right and left cameras. OpenCV use multiple views of a planar object, a chessboard to calibrate the cameras. Using a pattern of alternating black and white squares, ensures that there is no bias toward one side or the other in measurement. Given a person holding a chessboard, or any other scene with a chessboard and a reasonably uncluttered background, the OpenCV function `cvFindChessboardCorners()` is used to locate the corners of the chessboard. Given many joint views of chessboard corners, `cvStereoCalibrate()` uses `cvCalibrateCamera2()` to solve for rotation and translation parameters of the chessboard views for each camera separately to any given 3D point ***P*** in object coordinates in the camera coordinates $P_l = R_l P + T_l$ and $P_r = R_r P + T_r$ for the left and right cameras, respectively. It is also evident from Figure 4.10 that the two views of *P* (from the two cameras) are related by $P_l = R^T(P_r - T)$, where *R* and *T* are, respectively, the rotation matrix and translation vector between the cameras.

$$R = R_r(R_l)^T$$

$$T = T_r - RT_l$$

It is then possible to plug these left and right rotation and translation solutions into the above equations, to solve the rotation and translation parameters between the two cameras. Once the system has the stereo calibration terms, we can pre-compute left and right rectification lookup maps for the left and right camera views using separate calls to `cvInitUndistortRectifyMap()`[27].

Stereo correspondence, matching a 3D point in the two different camera views can be computed only over the visual areas in which the views of the two cameras overlap. For better results it is necessary that the cameras are arranged to be as nearly frontal parallel as possible. OpenCV implements a fast and effective block-matching stereo algorithm, `cvFindStereoCorrespondenceBM()`.The block-matching parameters and the internal scratch buffers are kept in a data structure named `CvStereoBMState`. It is possible to configure the parameters of the data structure via the menu interface of the system (see figure 4.11).

[27] Page 428, "Learning OpenCV: Computer Vision with the OpenCV Library", by Gary Bradski and Adrian Kaehler, Publisher: O'Reilly Media; 1st edition (24 Sep 2008), ISBN-13: 978-0596516130

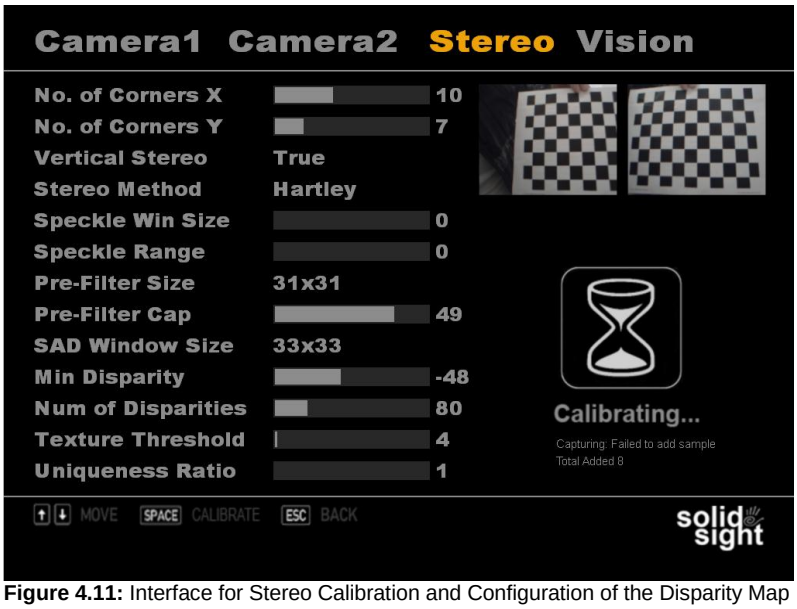


**Figure 4.11:** Interface for Stereo Calibration and Configuration of the Disparity Map

## 4.6 Body Movement Segmentation

To track the movement of the hands and head of the user, it is necessary to try and segment the figure from the background with a reasonable level of accuracy to avoid false positives. This movement of the user can then be analysed and the end-effectors of the head and hands can be mapped onto the skinned model.

The release of OpenCV 2.0 released on the 30th September 2009 includes an implementation of the algorithm developed by Farhad Dadgostar and Abdolhossein Sarrafzadeh in their paper of 2006 [Dadgostar]. The paper proposes a skin detection algorithm based on adaptive Hue thresholding. The skin classifier is based on the Hue histogram of skin pixels, and adapts itself to the colour of the skin of the person in the video sequence. As the paper describes, a static skin detector using Hue thresholding was developed which was named Global Skin Detector or GSD. However, the GSD can detect the actual skin pixels with a reasonable rate of accuracy, but it also falsely detects some non-skin pixels. In addition, it cannot distinguish those objects that have the similar Hue factor like skin colour (e.g., wood). In some situations even the amount of falsely detected pixels is more than the actual skin pixels, which makes it to be impractical for real-world applications. Therefore refinements were made to the algorithm; training the global skin detector using a set of training data and specifying the thresholds of skin colour in Hue colour space, detecting the in-motion pixels of the image, using a motion detection algorithm, and filtering the detected pixels using the Global Skin Detector so that the output of this step, are those pixels that have a higher probability of belonging to the skin regions of the image, and then recalculating the thresholds of the Hue Factor histogram so that they cover 90% of the area of the histogram.

However when it came to using this Adaptive Skin Threshold algorithm which has been implemented in OpenCV, there still seemed to be the same problem as was documented in the paper with the static skin detector, Global Skin Detector with the amount of falsely detected pixels being more than the actual skin pixels. This was the case when trying to use this Adaptive Skin Threshold algorithm in the system in an environment where either wooden desks or a yellow screen were present. These areas responded more strongly to the algorithm than the actual skin

of the person using the system so making it impractical to use. It was therefore decided to use a different technique to segment the head and hands.

Hue Saturation Lightness (HSL), and Hue Saturation Value (HSV) or often called HSB (B for brightness) are the two most common cylindrical-coordinate representations of points in an RGB colour model, which rearrange the geometry of RGB in an attempt to be more perceptually relevant than the Cartesian representation. They were developed in the 1970s for computer graphics applications, and are used for colour pickers, in colour-modification tools in image editing software, and less commonly for image analysis and computer vision[28].

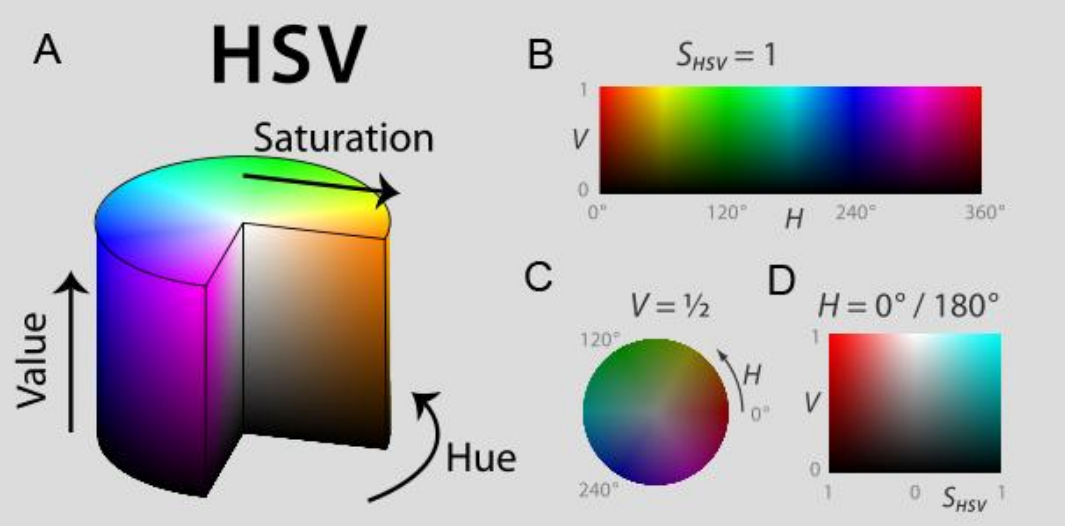


**Figure 4.12:** HSV showing cut-away 3D model (Left Diagram A), Two-dimensional plot showing the model's three parameters at once, holding the other constant: cylindrical shells (Top Right Diagram B) of constant saturation, in this case the outside edge of the cylinder; horizontal cross-section (Bottom Middle Diagram C) of constant HSV value, in this case the slices halfway down the cylinder; and rectangular vertical cross-section (Bottom Right Diagram D) of constant hue, in this case of hues 0° red and its complement 180° cyan.

OpenCV provides a function `cvCvtColor` that allows for easy colour conversion from standard RGB to HSV. It is then possible to output only a limited range of HSV values using the function `cvInRangeS` in which you specify the lower and upper range of values. Through experimentation it is possible to find a range of HSV values that seem to be more robust than the Adaptive Skin Threshold algorithm as they seem to be able to handle the inclusion of wood and other colours in the environment. This range of values was only tested for Caucasian skin although it would be possible to add configuration into the interface so that if need be a user would be able to configure the HSV range of values for their particular type of skin if it was found that the current HSV values did not work correctly. A person is better at finding the optimal values than a computer.

A median smoothing filter is then applied to this image to remove any noise or camera artefacts and to reduce the resolution of the image. OpenCV has a function `cvSmooth` which offers five different smoothing operations; `CV_BLUR`, `CV_BLUR_NO_SCALE`, `CV_MEDIAN`, `CV_GAUSSIAN` and `CV_BILATERAL`. The median filter is selected (`CV_MEDIAN`) as it is faster than Gaussian filter and replaces each pixel by the median pixel value in a squared neighbourhood around the centre pixel. The OpenCV function, `cvFindContours` is then applied to the resulting image using the parameters `CV_RETR_EXTERNAL` and `CV_CHAIN_APPROX_SIMPLE`. This has the

[28] Wikipedia Article "HSL and HSV", Web link: http://en.wikipedia.org/wiki/HSL_and_HSV

effect of simplifying the image further so meaning any holes in any of the shapes in the image are removed and only the external outer contours of any region are kept. At this stage, a number of contour regions will have been stored in memory; it is assumed that the contours for face and hands will be the largest in area in the entire image. By comparing the area of each region using the `cvContourArea` function in OpenCV, it is possible to find the three largest contour areas. For each of the three largest contours, the `cvMinAreaRect2` function is used to find the minimal-area bounding rectangle for the contour and the centre of this box can be used as the position of the end-effector. Figure 4.13 illustrates this procedure for the hand, comparing the results from using the Adaptive Skin Threshold algorithm and the procedure that has just been documented.

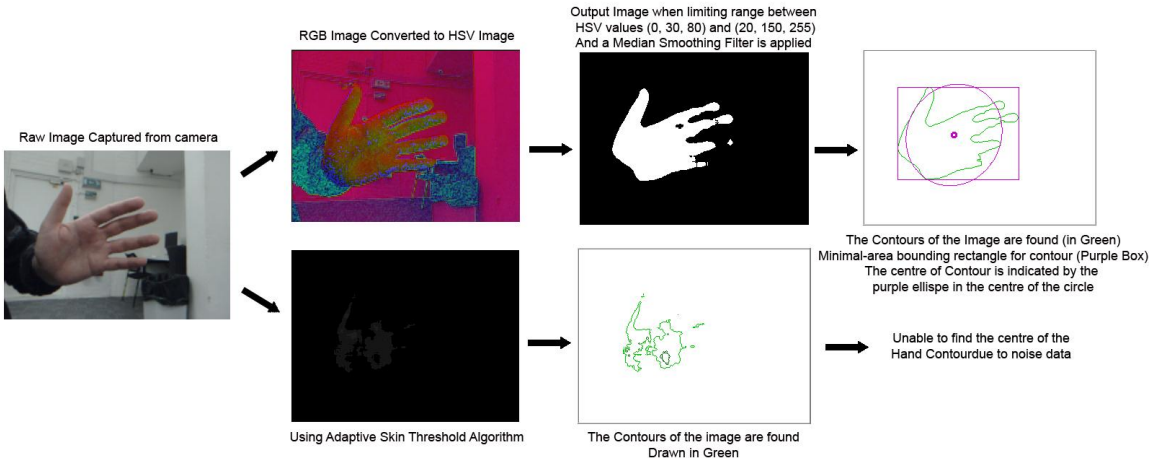


**Figure 4.13:** Procedure for determining the position of the end-effector of the hand and a comparison of the results of using the Adaptive Skin Threshold algorithm

Given the three end-effectors that have been calculated, it is necessary to determine which one is the head end-effector and by a process of elimination the two others must be the hand end-effectors. This is achieved by using the Haar classifier, which builds a boosted rejection cascade. OpenCV implements a version of the face-detection technique first developed by Paul Viola and Michael Jones commonly known as the Viola-Jones detector and extended by Rainer Lienhart and Jochen Maydt to use diagonal features. OpenCV refers to this detector as the “Haar classifier” because it uses Haar features or, more precisely, Haar-like wavelets that consist of adding and subtracting rectangular image regions (see figure 4.13) before thresholding the result. OpenCV ships with a set of pre-trained object-recognition files, but it is also possible to train and store new object models for the detector. A number of pre-trained objects already come with OpenCV including haarcascade_frontalface_alt2.xml which works best for frontal face detections. Assuming the cameras are placed above the television set, this would be the most appropriate classifier for the system as the player will be looking directly at the television screen so will have their face to the camera.

The Haar classifier that is included in OpenCV is a supervised classifier meaning the system will present histogram- and size-equalized image patches to the classifier, which are then labelled as containing (or not containing) the object of interest, which for this classifier is most commonly a face. The Viola-Jones detector uses a form of AdaBoost but organizes it as a rejection cascade of nodes, where each node is a multi-tree AdaBoosted classifier designed to have high (99.9%) detection rate (low false negatives, or missed faces) at the cost of a low (near 50%) rejection rate (high false positives, or “non-faces” wrongly classified). For each node, a “not in class” result at any stage of the cascade terminates the computation, and the algorithm then declares that no

face exists at that location. Thus, true class detection is declared only if the computation makes it through the entire cascade. For instances where the true class is rare (e.g., a face in a picture), rejection cascades can greatly reduce total computation because most of the regions being searched for a face terminate quickly in a non-class decision[29].

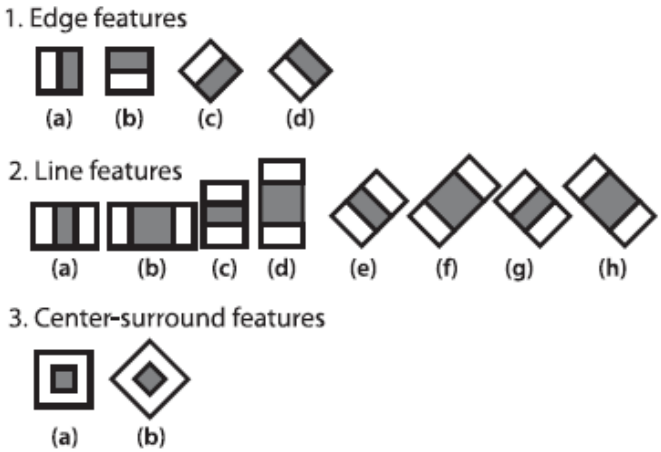


**Figure 4.14:** Haar-like features from the OpenCV source distribution (the rectangular and rotated regions are easily calculated from the integral image): in this diagrammatic representation of the wavelets, the light region is interpreted as “add that area” and the dark region as “subtract that area”

Checking only three regions of interest within the captured image for a face greatly speeds up the process. Once the face has been found this end-effector can be labelled as such and even if next time the face-detector fails to find a face it can be assumed that the region closed to the area where the face was previously detected is the face and can be labelled as such. It can further be assumed that out of the other two remaining end-effectors the one closest to the left-hand side of the capture image is the left hand and right hand to the right-hand side.

[29] Page 507, "Learning OpenCV: Computer Vision with the OpenCV Library", by Gary Bradski and Adrian Kaehler, Publisher: O'Reilly Media; 1st edition (24 Sep 2008), ISBN-13: 978-0596516130

# 5. Project Evaluation – Results

The system under controlled conditions where the background is fairly uniform in colour and the lighting conditions are fairly even gives reasonably good results. The hands of the model follow the user's movement reasonably accurately (see Appendix A).

There has not been sufficient time, to test the system in non-ideal conditions including the background and lighting although the system was designed to be configurable to allow for changes in lighting conditions and also the PS3 Eye cameras are designed to be able to work under lower lighting conditions than normal webcams.

It would also be necessary to test the system for users other than those with Caucasian skin to see if the tracking is still accurate.

# 6. Reflective evaluation

## 6.1 Summary and Conclusion

I had originally intended the system to be able to track the full body movement of the user including head and leg movement, however this proved to beyond the scope of the system due to time constraints. The initial 'learning curve' of developing the base system took too long, especially using C++ rather than a more rapid development language such as C#. However the latest version of the OpenCV library utilises C++ interfaces and so makes it harder to write wrappers for other languages. This meant that using another language other than C++ was not practical if I wanted to use the latest version of OpenCV 2.1.

The development of the system has given me a useful insight into the problems and challenges that are involved in developing a computer vision system.

## 6.2 Self-evaluation

I had originally intended to document the development process via an online blog. Although I started this blog, it ultimately proved to be too time-consuming and was subsequently not updated during the later stages of the development process although the initial research stage was documented and proved to be a useful reference when writing this report.

On reflection I spent too long researching how to implement such things as the full-screen mode which was ultimately not important to the overall system. The project would have also benefited from an initial design as I wouldn't have wasted time on such things as researching Windows Forms and the MFC (Microsoft Foundation Class Library) which never made it into the finished version.

## 6.3 Future Scope

- To implement head tracking
- Full-body segmentation
- Ability to track leg movements
- Face recognition so allowing for different models for different users

# 7. Bibliography

## Books

"Supercade – a visual history of the videogame age 1971 – 1984", by van Burnham, Publisher: MIT Press; 1st edition (10 Oct 2001), ISBN-13: 978-0262024921

"Real-Time Character Animation with Visual C++", by Nik Lever, Publisher: Focal Press; Pap/Cdr edition (17 Dec 2001), ISBN-13: 978-0240516646

"Computer Vision: Algorithms and Applications" (Texts in Computer Science), by Richard Szeliski, Publisher: Springer; 1 edition (1 Aug 2010), ISBN-13: 978-1848829343

"Game Programming Gems 6" (Game Programming Gems Series), Edited by Michael Dickheiser, Publisher: Charles River Media; 1st edition (March 7, 2006)

"Game Programming Gems 7" (Game Programming Gems Series), Edited by Scott Jacobs, Publisher: Charles River Media; 1st edition (January 22, 2008)

"Character Animation With Direct3D", by Carl Granberg, Publisher: Delmar; 1st edition (26 Mar 2009), ISBN-13: 978-1584505709

"Game Coding Complete, 3rd Edition", by Mike McShaffry, Publisher: Delmar; 3rd edition (9 April 2009) ISBN-13: 978-1584506805

"Introduction to 3D Game Programming with DirectX 9.0c: A Shader Approach" (Wordware Game and Graphics Library), by Frank D. Luna, Publisher: Wordware Publishing Inc. (1 Jun 2006), ISBN-13: 978-1598220162

"Learning OpenCV: Computer Vision with the OpenCV Library", by Gary Bradski and Adrian Kaehler, Publisher: O'Reilly Media; 1st edition (24 Sep 2008), ISBN-13: 978-0596516130

"Windows System Programming (Addison-Wesley Microsoft Technology)", by Johnson M. Hart, Publisher: Addison Wesley; 4th edition (16 Feb 2010), ISBN-13: 978-0321657749

"Designing Gestural Interfaces: Touchscreens and Interactive Devices", by Dan Saffer, Publisher: O'Reilly Media; 1st edition (26 Nov 2008), ISBN-13: 978-0596518394

"Remediation: Understanding New Media", by Jay David Bolter and Richard Grusin, Publisher: MIT Press; 1st edition (1 Mar 2000), ISBN-13: 978-0262522793

## Papers

**Stan Melax** "The Shortest Arc Quaternion", Games Programming Gems, Edited by Mark A. DeLoura, Publisher: Charles River Media; 1st edition (2000) [Melax]

**Ronan Boulic, Javier Varona, Luis Unzueta, Manuel Peinado, Angel Suescun, and Francisco Perales**, "Evaluation of On-line Analytic and Numeric Inverse Kinematics Approaches Driven by Partial Vision Input" [Boulic]

**Luis Unzueta, Manuel Peinado, Ronan Boulic, Ángel Suescun**, “Full-body performance animation with Sequential Inverse Kinematics”, Graphical Models Journal 70 (2008) 87–104, Published: Elsevier [Unzueta]

**R. Lange and P. Seitz**, “Solid-State Time-of-Flight Range Camera”, IEEE J. Quantum Electronics, Vol. 37 (3), 390-397, March 2001 [Lange]

**Dr.-Ing. Thorsten Ringbeck, Head of BU Systems, Dipl.-Ing. Bianca Hagebeuker, Product Marketing,** “A 3D Time-of-Fight Camera for Object Detection”, presented at “Optical 3-D Measurement Techniques” 09-12.07.2007 ETH Zürich, Plenary Session 1: Range Imaging I [Ringbeck]

**D. Dervinis, Department of Electronics,** Šiauliai University "Head Orientation Estimation using Characteristic Points of Face", ELECTRONICS AND ELECTRICAL ENGINEERING: MEDICINE TECHNOLOGY, 2006. No. 8 ISSN 1392 – 1215(72) [Dervinis]

**Rae R., Ritter H.J.**,“Recognition of human head orientation based on artificial neural networks”, IEEE Transactions on Neural Networks – 1998. – No. 9 – P. 257 – 265. [Ritter]

**Darrell T., Moghaddam B., Pentland A.P.**, "Active face tracking and pose estimation in an interactive room", IEEE Computer Society Conference on Computer Vision and Pattern Recognition. – 1996. [Darrell]

**Gee A., Cipolla R.**, "Fast visual tracking by temporal consensus", Image and Vision Computing. – 1996. – No. 14. – P. 105 - 114. [Gee]

**Badler, N. I., Hollick, M. J., Granieri, J. P.** (1993) "Real-Time Control of a Virtual Human Using Minimal Sensors, In Presence Teleoperators and Virtual Environments", Volume 2(1), pp. 82-86 [Badler]

**Horprasert A.T., Yacoob Y., Davis L.S.** "Computing 3D head orientation from a monocular image sequence", Proceedings of SPIE 25th, AIPR Workshop: Emerging Applications of Computer Vision. – 1996. – No. 2962. – P. 244 – 252 [Yacoob]

**Javier Varona, Jose M. Buades, Francisco J. Perales** "Hands and face tracking for VR applications", Computers & Graphics 29 (2005) 179–187 Elsevier [Varona]

**Robert Y. Wang, Jovan Popović**, "Real-Time Hand-Tracking with a Color Glove", Proc. SIGGRAPH 2009), 28(3) [Wang]

**Wren,C.R., Clarkson,B.P., Pentland, A.P.** (2000) “Understanding purposeful human motion”, Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition, pp.:378-383 [Wren]

**Javier Romero, Hedvig Kjellström, Danica Kragic** "Hands in Action: Real-Time 3D Reconstruction of Hands in Interaction with Objects", Robotics and Automation (ICRA), 2010 IEEE International Conference, pp.: 458-463 [Romero]

**Farhad Dadgostar, Abdolhossein Sarrafzadeh** "An adaptive real-time skin detector based on Hue thresholding: A comparison on two motion tracking methods", Pattern Recognition Letters 27 (2006) 1342–1352 [Dadgostar]

## Websites

BBC News – “Virtual human' Milo comes out to play at TED in Oxford”, by Jonathan Fildes, Technology reporter, BBC News, Oxford 13th July 2010, Web link: http://www.bbc.co.uk/news/10623423

Gamasutra feature article, “Taking Games Beyond Whack and Tilt”, by Anupam Chakravorty, Rob Kay, Stuart Reynolds, Ian Wright, 27th July 2010, Web link: http://www.gamasutra.com/view/feature/5935/taking_games_beyond_whack_and_tilt.php

Gamasutra feature article "A History of Gaming Platforms: Atari 2600 Video Computer System/VCS" by Matt Barton, Bill Loguidice, 28th February 2008, Web link http://www.gamasutra.com/view/feature/3551/a_history_of_gaming_platforms_.php

New Scientist article, “Innovation: Microsoft's Kinect isn't just for games”, by MacGregor Campbell, 21st June 2010, Web link: http://www.newscientist.com/article/dn19065-innovation-microsofts-kinect-isnt-just-for-games.html

Joystiq Beta Article "Kinect: The company behind the tech explains how it works", by Mike Schramm, 19th Jun 2010, Web link: http://www.joystiq.com/2010/06/19/kinect-how-it-works-from-the-company-behind-the-tech/

# Appendix A - **Results**

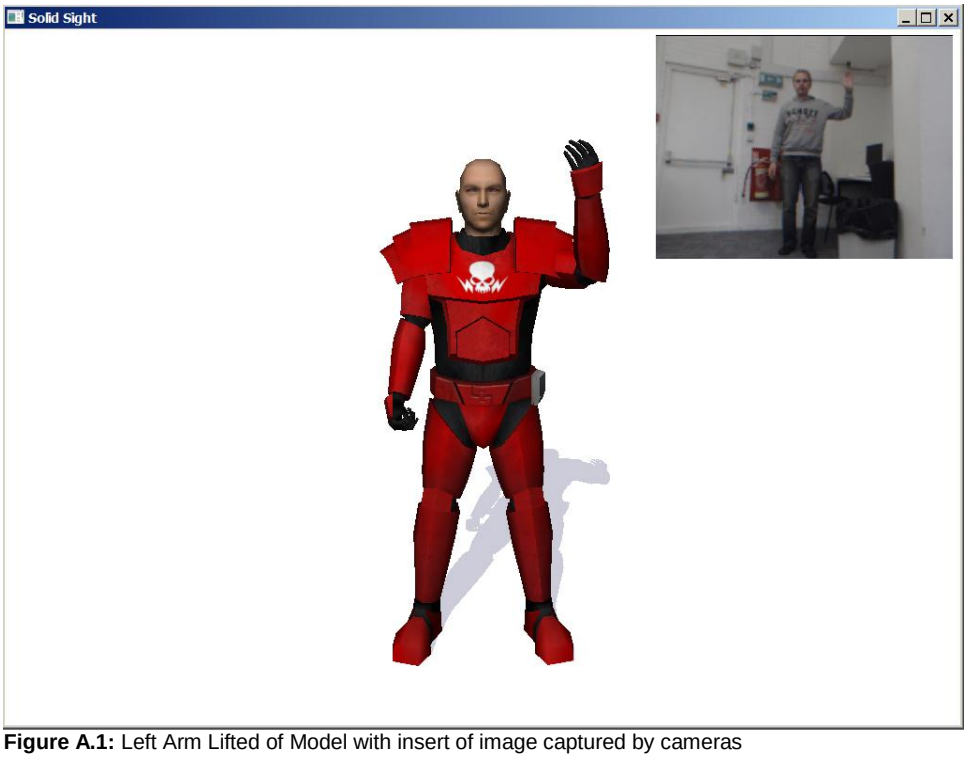


**Figure A.1:** Left Arm Lifted of Model with insert of image captured by cameras

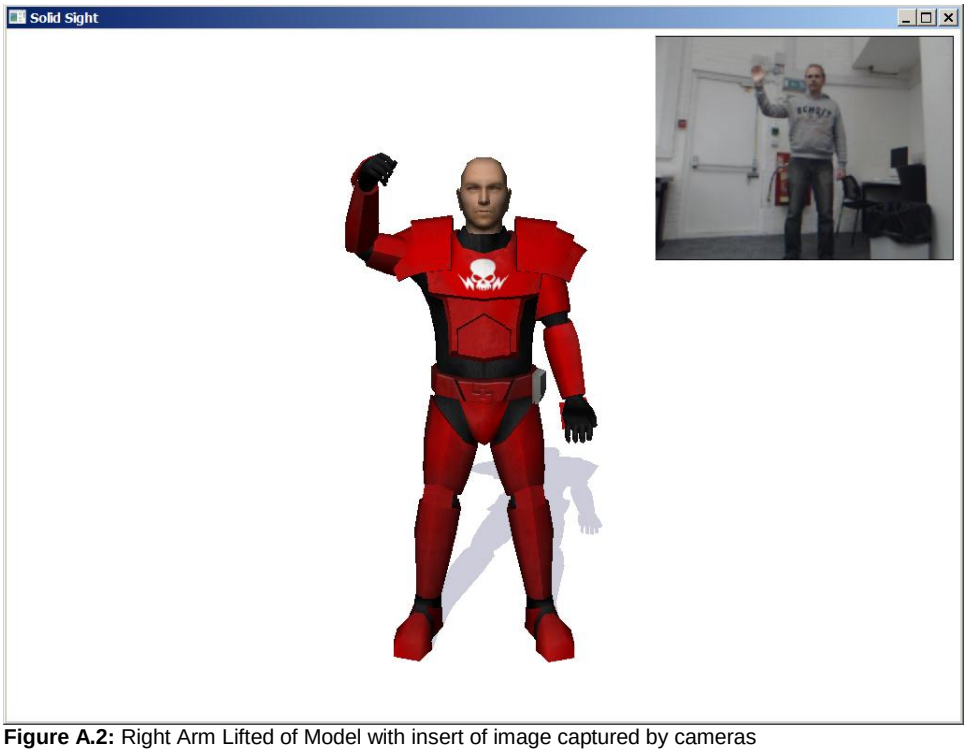


**Figure A.2:** Right Arm Lifted of Model with insert of image captured by cameras

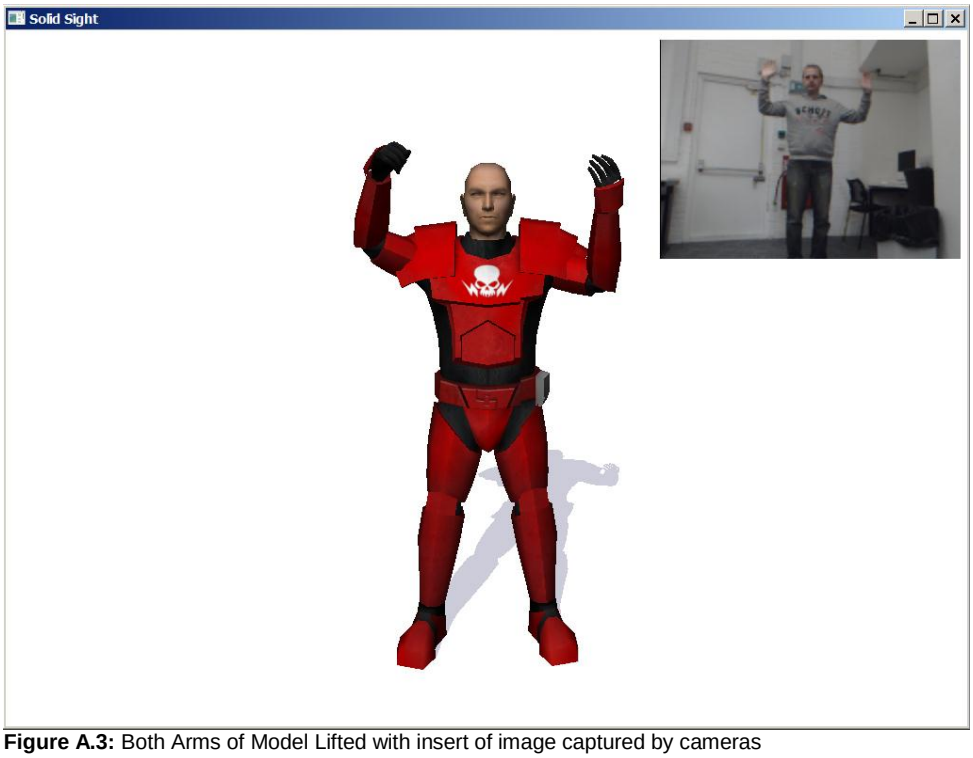


**Figure A.3:** Both Arms of Model Lifted with insert of image captured by cameras