

Generating 3D Product Design Models in Real-Time using Hand Motion and Gesture

A thesis submitted to Brunel University in fulfilment of the requirements for the degree of Master of Philosophy

By

Stephan Orphanides

**School Of Engineering and Design
Brunel University**

May 2010

Acknowledgements

I give my sincere thanks to all who helped me in this interesting project, especially to my supervisor Dr. Jinsheng Kang. Thank you for giving me this opportunity. Thanks to Dr. Qin for the interesting discussions I had with you and for always being available and welcoming. Thanks to William Brown and Xiao Yi, for the collaborative research which I found fascinating, and for organising and helping out at the capture sessions. Finally I thank my entire family who supported me during the progress of this investigation.

Abstract

Three dimensional product design models are widely used in conceptual design and in the early stage of prototyping during the design processes. A product design specification often demands a substantial amount of 3D models to be constructed within a short period of time. Current methods begin with designers sketching product concepts in 2D using pencil and paper, which in turn are then translated into 3D models by a design individual with CAD expertise, using a 3D modelling software package such as Pro Engineer, Solid Works, Auto CAD etc. Several novel methods have been used to incorporate hand motion as a way of interacting with computers. There are three main types of technology available to capture motion data, capable of translating human motion into numeric data which can be read by a computer system. The first being, hand gesture glove-based systems such as “Cyberglove”, these systems are generally used to capture hand gesture and joint angle information. The second is full body motion capture systems, optical and non-optical-based, and finally vision based gesture recognition systems which capture full degree of - freedom (DOF) hand motion estimation. There has yet to be a method using any of the above mentioned input devices to rapidly produce 3D product design models in real time, using hand motion and gestures. In this research, a novel method is presented, using a motion capture system to capture hand gestures and motion in real time, to recreate 3D curves and surfaces, which can be translated into 3D product design models. The main aim of this research is to develop a hand motion and gesture-based rapid 3D product modelling method, allowing designers to interactively sketch out 3D concepts in real time using a virtual workspace.

A database of a number of hand signs was built for both architectural hand signs (preliminary study) and Product Design hand signs. A marker set model with a total of eight markers (five on the left hand and three on right hand/marker pen)

was designed and used in the capture of hand gestures with the use of an Optical Motion Capture System. A preliminary testing session was successfully completed to determine whether the Motion Capture system would be suitable for a real-time application, by effectively modelling a train station in an offline state using hand motion and gesture. An OpenGL software application was programmed using C++ and the Microsoft Foundation Classes which was used to communicate and pass information of captured motion from the EVaRT system to the user.

Abbreviations

Throughout this Thesis, several programs, hardware, and software's are referred to in abbreviated form. Please see the list below showing each abbreviation and its associated full meaning.

OpenGL - (Open Graphics Library), is a cross-platform API for writing applications that produce 2D and 3D computer graphics.

EVaRT - EVa Real-Time Software is an application provided by Motion Analysis which allows the user to set up, calibrate, capture motion in real-time, capture motion for post processing, edit and save data in the format of their choice.

MFC - Microsoft Foundation Classes is a library that wraps portions of the Windows API in C++ classes.

Mocap - Motion capture is a term used to describe the process of recording movement and translating that movement into a digital format.

SDK - A software development kit is typically a set of development tools that allows for the creation of software applications.

Direct X - Microsoft DirectX is a collection of application programming interfaces for handling tasks related to multimedia, especially game programming and video, on Microsoft platforms.

Table of Contents

Acknowledgements	2
Abstract & Abbreviations	3
1 Introduction	8
1.1 Research Motivation	10
1.2 Aims and Objectives	10
1.3 Preliminary Concepts	11
1.3.1 Challenge of real-time hand gesture modelling for product design	11
1.3.2 Motion Capture	12
2 Literature Review	15
2.1 Motion Synthesis	
2.1.1 Manual Synthesis	18
2.1.2 Physically-Based Synthesis	19
2.1.3 Data-Driven Synthesis	22
2.2 Motion Description	27
2.3 HCI (Human Computer Interaction)	29
2.4 Hand Gesture Recognition	34
2.4.1 Image Pre-Pre-processing	35
2.4.2 Tracking	37
2.4.3 Recognition	38
3 Framework of Hand Gesture and Motion Based Design Method	41
3.1 Design Needs	41
3.2 System Design	42

3.3 Advantages and Disadvantages	43
3.4 Identification of Key Feasibility Studies	44
3.5 Hand Gesture and Motion	45
3.5.1 Generic Applications	45
3.5.2 Specific Applications	45
3.6 Real-time Communication	46
3.6.1 Data Capture	46
3.6.2 Data Exchange	46
3.6.3 Design Interface	47
3.7 Overview of Proposed Research	47
 4 Feasibility Study 1 – Hand Motion and Gesture	 49
4.1 Hand Gesture and Sign Language for Architecture	50
4.2 Marker Placement	54
4.3 Motion capture system	57
4.3.1 Planning a session	57
4.3.2 Setup and Calibration	58
4.3.3 Hand motion capture and recognition	60
4.4 Motion Sketches and Initial Data Processing	65
4.5 3D Curve and Surface Generation, And Preliminary Results	68
 5 Feasibility Study 2 – Interactive Real-time Communication Application	 70
5.1 Design of Software Application	73
 6 Testing and Results	 78
 7 Conclusions and suggestions for Further work	 82
7.1 Summary	82
7.2 Final Thought and Discussion	83

8 References	84
Appendix A Hand gesture signs for product design motion capture	89
Appendix B Programming code for software development	90

1 Introduction

Products constitute one of the classic four P's of the marketing mix (Product, Price, Place, Promotion) and the most fundamental characteristics of a product is its exterior form factor and design. Product design is often mentioned as being the most important determinant of a new products performance and it's sales success. In society today, the aesthetic appearance of products is the most significant regardless of their function. When presented with a choice between two products identical in price and function, consumers will select the item which they consider to be more attractive and aesthetically pleasing. The form of a product contributes to its success in several ways, most commonly to gain consumer notice, when markets are cluttered with products intended for the same purpose. When designing products, designers make choices regarding their characteristics, some of these include; shape, scale, proportion, materials, colour, reflections and texture. Given the purpose of a product, its target market and its desired performance specification the design team set about creating a design form which will be successful. Production costs and manufacturing processes also determine the outcome of a products form, managers put pressure on designers to develop products which can be manufactured cheaply and adhere to quality control parameters. Design objectives and constraints are becoming ever more excessive, as companies go in search for something special which will set their product apart from the competition. This results in the design process becoming increasingly complex, and more time spent on conceptual design to ensure the correct product is designed and released to market.

A product's form, once developed, may evoke diverse physiological responses and feelings from consumers. These reactions are the result of subtle design refinements implemented by the designer. These can often be intentionally planned or often a natural effect which occurs rarely at the concept stage. These rare subtleties seldom occur when the designers mind is free at the drawing board in the middle of producing multiple 2D sketch-based conceptual drawings.

Unfortunately transferring these subtle refinements to the finished product as to be experienced by the user, is often a complete task of its own. Most 3D modelling packages although extremely powerful, are geared toward expert users with years of experience, with tools that are hard to learn and slow and meticulous to use. The main reason for this is that current methods were originally developed for Computer Aided Design (CAD) professionals where accurate mathematical control was an essential requirement. The fundamental problem when representing 2D sketches of curves and surfaces as a 3D entity, is that the majority of natural real world object have non-planer curves. The problem of finding the 3D space curve from a 2D sketch is mathematically undeterminable, as it has many possible solutions. Recent attempts to create simpler interfaces for 3D modelling have been developed, based on the human ability to quickly outline a global overview of an object. These approaches are frequently referred to as 3D Sketching. Their principle is to generalise the shape of the 3D model and apply details by means of different editing operations such as extrude and cut. Sketching is essentially an activity driven by a significant amount of curves. The main limitation of this method, stems from the ability of the 3D sketching system only being able to perform few operations based on it's analysis of the user defined strokes.

Improvements in this area could encourage stylists and designers to directly create the digital model, and continue to work on it making alterations to produce new products. In essence the goal would be to capture the prominence of sketch based curves.

All the above have been the focus of research in 3D modelling for years, but there is yet to be a system that tackles the problem successfully. The quest to seek the "ideal" form for a product remains to be a key goal for designers. Concept selection is a convergent process, thus the products form still remains a vital factor. There are many different methods and techniques that fall into the broad definition of 3D modelling CAD Sketching and HCI. The main selection of these methods will be analysed and discussed in the next section.

1.1 Research Motivation

The motivation of this project is one of significant concern to the product design industry, where designers use hand drawn sketches to realise the conceptual design of a product. Ultimately, the main problem with this method is, it forbids the designer to make changes which can later be easily translated to the final product. The design of a product must go through several stages of electronic composition and alterations before it reaches production, during which vital subtle design inspirations realised by the designer are either lost during the transition, or take a vast amount of time and money to implement, becoming unfeasible. The only way round this problem is for the designer to begin the whole conceptual design process again. This again uses more time and money delaying the products lead time from the design stage to consumer market. Hence the aim would be to enable the designer, animator or artist to have control over the electronic 3D model at the conceptual design stage.

1.2 Aims and Objectives

Aims

- Explore the feasibility of rapidly creating 3D product design models in real time, using hand motion and gestures.

Objectives

- Create a system to provide a natural 3D interactive design interface for two hand interaction and gesture in real-time.
- Use a motion capture system for motion recognition and allow for multi-users to work simultaneously for collaborative work.
- Investigate new methods that could lead to a new design process that can support virtual product design by hand signs and 3D sketches with actual 3D world references.
- Define a set of hand signs and gestures for the design application that can be further developed into a new design modelling language for a wide range industry applications

1.3 Preliminary concepts

1.3.1 Challenge of real-time hand gesture modelling for product design

Product Design is a creative process, which often requires a blend of the following professions to be a success; art, design and engineering. The imagination, inspiration and passion of artists and designers are crucial in creating innovative conceptual designs. In an ideal world whilst at the creative design stage with inspiration, the designer may want to quickly and effortlessly express design ideas in 3D without any impediments. This behaviour reflects common means of communication found amongst humans, used in society on a daily basis. Unfortunately 2D Sketches that represent 3D ideas and the operation of 3D CAD software are time consuming, and very tedious, which ultimately neglect inspiration and the perfect design model from being realised.

The main challenge of generating 3D models for product design using hand gesture and motion lies with HCI (human computer interaction). The human hand has a very complex structure, with 27 degrees of freedom: 4 in each finger, 5 in the thumb, and 6 DOF for the rotation and translation of the wrist, tracking and recording this complex motion has been a challenge for several years [1]. There are several methods used to capture human motion, all having their advantages and disadvantages making them suitable for different applications.

One method of obtaining accurate motion data is by using glove-based devices. This method is generally used to capture hand gestured motion along with joint angles. The main issues surrounding this technology are that glove based devices are often cumbersome and wired to the subjects hands, offering restrictions in movement. The second widely used method is vision-based gesture recognition, which uses stereo visual equipment to recognise static and temporal gestures. A common problem found with this method is that complex finely tuned recognition algorithms are needed which are computationally intensive as well as taking a long time to program. The final method known as full body motion capture will be discussed in the following section.

1.3.2 Motion Capture

Motion capture often abbreviated as 'mocap' is defined as "The creation of a 3D representation of a live performance." The first use of motion capture dates back to the 19th century with the motion studies of Eadweard Muybridge where basic photography was applied for medical and military purposes. The demand for realistic three-dimensional animation in computer graphics is increasing making motion capture a prominent feature in this field. Mocap is also widely used in military, entertainment (film, game), sports, and medical applications. Back in the 1980's when the first motion capture systems were being introduced, many considered them to be a fairly controversial tool, as the effort involved to 'clean up' the motion capture data equalled that of the time taken to use key framing, the alternative method for generating realistic motion. Software and hardware advancements since then have allowed motion capture to become a feasible tool for the capture of motion.

Current motion capture systems can be separated into two main groups, optical systems and non-optical systems. Furthermore, optical systems can be subdivided into systems which utilise passive markers, active markers, and no markers (marker-less).

The most common of the above, passive optical systems require the subject to wear a body suit with markers attached, coated with a retro-reflective material. Light is then reflected back to the cameras lens, the centroid of the marker is estimated by the 2D image captured. When two or more cameras see this marker a 3D fix can be obtained. Multiple cameras are used to update a computer of the markers exact position at up to 500 frames per second. System setups can have anywhere from 6 - 300 cameras at any one time. The more cameras present, the less likely marker swapping will occur as more cameras will be able to see a single marker at any given time. Having additional cameras also helps with the ability to capture a group of people or more than one subject in a larger capture volume.

Active optical systems triangulate the position of a marker by illuminating one LED very quickly. Instead of reflecting light back that is generated externally, the markers themselves are powered to emit their own light. The advantages of active optical systems are that identifying each marker as an individual makes it useful for real-time applications.

More recently research by Stanford, MIT, and Max Planck Institute, has seen the development of markerless systems which permit the user to not wear special equipment for tracking. These applications require the use of computer algorithms to break down the human body, fragment it into sections, and identify them for tracking.

Non-optical based systems can be subdivided into systems that rely upon inertia, mechanics and magnetism.

Inertial motion capture technology uses small inertial sensors and gyroscopes that transmit digital information such as joint angle measurement wirelessly, to a computer where the motion is viewed and recorded. As with optical based systems, the more gyroscopes, the higher the accuracy. The advantages of inertial motion capture systems are that they do not require the use of any cameras or external markers. This in turn puts little restriction on the size of the capture volume. Inertial based systems are popular amongst the film and game industries as manufacturers offer systems at base prices ranging from \$25,000 to \$80,000 USD.

Mechanical motion capture systems commonly referred to as exoskeleton systems track body joint angles directly. They use plastic or metal rods to create rigid structures which mimic the human skeleton. Movement from the performer conform potentiometers that articulate at the main joints of the body. Once again these systems offer occlusion free capture and an unlimited capture volume. In contrast to this some believe wearing bulky exoskeleton suits for mechanical motion capture restrict the user from natural motion. In the medical industry

where accurate natural motion is required to study and diagnose a patient's gait, this is not practical.

Magnetic systems use the magnetic flux of orthogonal coils on a transmitter and receiver to measure the position and rotation of the given motion. Each sensor can output up to 6 degrees of freedom, providing good results using less markers required with optical based systems. Unfortunately, the use of magnetism in these systems incurs interferences with near by electronic devices such as lighting monitors, computers and phones.

For our research purposes a motion capture system by Motion Analysis Corporation namely Eagle Digital Real-Time System will be used for the capture of motion.

2 Literature Review

In the following section, the area of research will be separated into four main sections. Motion Synthesis, Motion Description, Hand Gesture Recognition, HCI (Human Computer Interaction). Past and present research will be discussed and reviewed.

2.1 Motion Synthesis

Studies of human motion or a human-like character have been present in various areas over several years. Obtaining, analysing and generating human motion has been the main focus of researchers in the areas of biomechanics, robotics and computer graphics. The focal point of this review will be computer graphics and virtual environments.

The demand for realistic three-dimensional animation within a virtual environment is increasing. Today's constant technological advances are allowing virtual worlds to strongly correspond to a real life scene. Due to this, various indoor and outdoor scenes such as, houses, offices, aeroplanes, train stations, stadiums, and work environments can be replicated and used for training purposes, pre-construction simulation and safety testing.

Newly qualified professionals can perform multiple complex training operations in their given field, risk free. Without the need for expensive setup costs, virtual environments can be implemented to simulate a training exercise, once they have finished or made a mistake they are able to reset the system and prepare for a second try. This method of virtual training environment exercises are proving to be irreplaceable especially when there is great potential for mistake and more importantly, have an enormous cost.

More importantly, the availability of on-demand training environments makes it possible to perform unlimited amounts of exercises, in turn increasing the number of trainees and their expertise.

Over the past few years the increase in rendering and processing speeds, are allowing for more complex shapes and a higher number of these shapes to be incorporated into a virtual environment making them more detailed and additionally more realistic. Higher screen resolutions, improved lighting and shading all add to the realism of the scene making the environment more believable to the viewer.

Unfortunately, a realistic true to life scene does not always present the viewer with acceptance and believability. Picture a scene of a busy London street, or a shopping centre, without the presence of virtual characters the viewer will begin to dismiss the scene as being unconvincing.

Therefore, virtual characters and their motion present an unavoidable part of almost any virtual environment. These virtual characters can be humans, animals, and many other beings which all bring to life an environment and make the virtual world resemble the real world.

As mentioned earlier, faster rendering and processing times also help with the appearance of virtual characters as they do with other objects such as furniture, doors etc. It is therefore safe to assume that features of virtual humans such as hair, clothing and skin will all appear realistic. Unfortunately with virtual humans just having these characteristics is not sufficient. Virtual humans cannot satisfy the user's expectations by standing still in one position and looking realistic like pieces of furniture. They must breath, walk, act, and most importantly interact with their surroundings.

The most important part of a virtual human is the way they act. Their individual movement and reactions to the surrounding stimuli within the environment is what

makes them realistic and believable to the viewer. Consider an animated cartoon character such as “Top cat”, it is clear to note that nothing appears realistic about a cat talking to a human police officer. Nevertheless, humans recognise familiar reactions and movements which are exhibited by real life humans. Thus, when “Top cat” reacts in the same manner as a human, he is generally accepted regardless of his appearance as a cartoon character.

The demand for realistic three-dimensional animation in computer graphics is increasing. In response to this demand, many methods to create animation effectively have been proposed and realised. It is important when creating animation of human motion that it appears to be natural and generate human motion as automatically as possible so that the workload of animators is minimised. The motion of a human figure is usually modelled to have very high DOF (Degrees of Freedom). Due to this, if an animator were to generate such motion the workload would be quite heavy, and a high level of artistic skill would be required to attain realistic natural motion.

The motion of Virtual characters can be split into two main sections: how they move and how they act. The first refers to the spatial transition of the character for example moving from point A to B whilst avoiding obstacles such as furniture and or other virtual humans. The second refers to the execution of individual motions, for example reaching to open a door, striking a ball with a hand or foot, and taking a seat at a table. These two parts belong to path planning and motion synthesis respectively.

The area of motion synthesis can be broken down and divided into three categories: manual synthesis, physically-based synthesis, and data-driven synthesis.

2.1.1 Manual Synthesis

Manual synthesis is one of the earliest methods of motion synthesis, it is also the simplest, consisting of the character's degrees of freedom to be specified manually at given points in time, known as keyframes. This method is often referred to as keyframing. The quality of the motion is governed by the skill of the animator as they have full control over the animation. After several poses of the animation have been set by the animator keyframes in between can be computed through simple interpolation. However, interpolation rarely produces accurate results when the distance between two keyframes is extreme, therefore the animator must manually construct a considerable portion of the character poses which make up the motion. This is a very tedious and time consuming, especially when most film and animation is broadcast at a minimum of 24 frames per second. Also, to create these poses such that they appear realistic when played in real time involves a great deal of skill from the animator. There are several parameters which the animator must consider in order to produce a realistic scene, some of these include: location/orientation, joint angles, shape, material properties and lighting.

As the animator has full control over the scene and controls every detail of the character's movement, this method of manual synthesis is commonly found in the production of cartoons where realistic motion need not be developed as long as it is expressive to the viewer.

The alternative method to keyframing categorized under manual synthesis, is the design of algorithms that are capable of replicating sets of motion. The advantage of this method over keyframing is that it allows the animator to create entire motion at once, rather than one pose at a time, and these motions can be controlled interactively for online character animation such as computer games.

Perlin [2] and Perlin and Goldberg [3] confirmed that a variety of motions can be generated with simple and efficient algorithms which avoid computation intensive dynamics and constraint solvers commonly found in physically-based synthesis.

Nevertheless, these algorithms can be tricky to design, and more importantly whenever a new kind of motion is required for animation, a completely new algorithm must be developed.

2.1.2 Physically-Based Synthesis

Physically-based synthesis is the generation of realistic motion by consideration of dynamic conditions. As human motion in real life is governed by the laws of physics, physical simulation is a natural approach for animating motion. This strategy reduces the freedom of the human character by eliminating motions that cannot occur based on considerations of dynamic consistency. Hodgins et al. [4] proposed a human model containing information about mass and inertia, and generated dynamically consistent motion by applying proportional-derivative servos to compute joint torques based on the desired and actual value of each joint. Several dynamically correct athletic behaviours were accomplished: running, bicycling, and vaulting. Despite the fact users are limited to three types of motion, parameters such as speed and direction can freely be adjusted. Comparable methods were developed by Wooten and Hodgins [5] [6] to generate motions like tumbling and flipping. In addition, a second model was developed which was capable of adapting existing simulated behaviours to new characters, with different limb lengths, masses, or moments of inertia, for example men/women to children [7]. Basic actions such as balance, protective stepping when balance is disturbed, protective arm reactions when falling, multiple ways of rising upright after a fall, and several more vigorously dynamic motor skills were realised by Faloutsos et al [8]. Controller design remains a difficult method for creating motion, as finding joint torque values that yield specific motions is difficult, especially when a chosen controller is capable of producing a limited range of motion.

Laszlo et al. [9] applied limit cycle control to adjust the joint controller which creates robust walking gaits for a fully-dynamic 19 degree- of-freedom human model. The limit cycle control technique offers an automated way of adding

closed-loop control to a basic desired open-loop motion. The open-loop component of the control can be tailored in a variety of ways to produce stylistic variations and useful parameterizations of the motion without any loss of physical realism.

Faloutsos et al, [10] uses a support vector machine (SVM) capable of pre-learning conditions of different controllers for the composition of different actions thus providing a meta-controller that switches attributes to transition between several motions.

Komura et al. [11] [12] created a model that considers not only mass and inertia but also musculoskeletal structure. Naturally, the model was designed on the musculoskeletal configuration of humans. Due to this, such an approach can create a human model capable of realistic motion similar to that of a real life human. Unfortunately, the good results produced by this system are often hindered as the model is very costly to make and requires heavy computational power because of its many degrees of freedom. In order to overcome this problem, simplified human models have been proposed.

An alternative to constructing joint controllers is to use constrained optimisation which creates full dynamically correct motion from a small amount of predefined keyframe poses.

Popovic et al. [13] generated a model with three significant simplifications: the elbows and spine are abstracted away, the upper body is reduced to the centre of mass, and movement is abstracted from symmetric motion. Once this simplified model is obtained, spacetime optimisation and dynamic calculation is performed. The advantages of the simplified model can be noticed, as only the fundamental properties of the motion are considered making the model easier to handle because only crucial degrees of freedom are conceived.

Safanova et al. [14] show it is possible to reduce the degrees of freedom found in captured motion from 60 to less than 10 using Principle component analysis with

examples of forward, vertical, and turning jumps; with running and walking; and with several acrobatic flips. Although this approach can improve convergence and make individual poses appear more natural, the sequence of keyframes used to create motion after optimisation does not match the fidelity of captured motion.

A rapid and user friendly approach to creating complex realistic ballistic motion was proposed by Liu and Popovic [15]. A rough motion is defined by a small amount of keyframes which is computed via spline interpolation, once complete, an optimisation process is performed to generate the minimal deviation from the initial motion which enforces a small set of linear and angular momentum constraints. These trajectories are derived from Newton's laws, when the character is airborne and from biomechanically-inspired model of momentum transfer when the character is on the ground.

Some new methods consider dynamic conditions between the model and its surrounding environment. As the majority of human motion is performed in contact with the ground, ground reaction forces are the most significant external forces. Tak et al. [16] modifies the trajectory of the ZMP to convert dynamically inconsistent motion to consistent motion.

Fang and Pollard [17] consider a new optimisation method which avoids traditional approaches of constraining parameters involving joint torques. They consider constraints and objective functions that lead to linear time first derivatives which results in fast computation times. Their system is particularly useful for synthesizing highly dynamic motions.

Several approaches which also consider external reactions with the environment, all make it possible to easily handle aggressive external perturbations. Ko et al. [18] created motion of a character capable of carrying a heavy load from captured motion of humans walking normally using inverse dynamics.

Carrying a heavy load on the back of a character and receiving a sudden perturbation to the body, were considered by Oshita et al. [19] The balance maintenance system controls the angular acceleration of the character's joints so as to track the user-input motion. Environmental physical input is then applied to the character, the dynamic motion control computes the angular joint accelerations in order to produce dynamically changing motion in response to the physical input. Zordan et al. [20] considered a human model that combines dynamic simulation and human motion capture data. By controlling the simulation with motion capture data, the character retains subtle, expressive details from the data while adding responsiveness and interactivity from the dynamic simulation. The motion is modified with time-scaling, blending, and inverse kinematics to achieve specified tasks, such as being boxed by someone. Balance of the character is maintained through control torques computed according to feedback of the centre of mass.

2.1.3 Data-Driven Synthesis

Human motions can be captured using motion capture, a system that digitally records movement of a subject. A highly realistic and automatically dynamically correct motion is captured which can be used as raw material for various data-driven synthesis algorithms. Despite the fact that the original motion is automatically dynamically consistent, once the motion has been synthesised the motion becomes dynamically inconsistent. Nevertheless, for the application of computer graphics this is not always important as the motion must only appear natural.

One of the most common methods to edit motion is by using spacetime constraints. This method was first introduced by Witken et al. [21] and consists of generating motion by constraining a point of the body in a specified position at a given time. The original model was designed for a general articulated object, not just a human model. The original method also calculated external forces and joint torques so that the model is constrained at the specific position at the correct time.

Shortly after, Liu et al. [22] developed a way of efficiently solving spacetime constraints with the use of hierarchical calculation that was used by Rose et al. [23] to generate motion of a human figure. The motion transition generation used by Rose et al. is a combination of spacetime constraints and inverse kinematic constraints that generates seamless and dynamically plausible transitions between motion segments. The use of fast recursive dynamics formulation made it possible to use spacetime constraints on systems with many degrees of freedom, such as human figures. A new method of spacetime constraints was proposed by Gleicher [24]. The method excluded all dynamic calculation and the motion generated used captured motion for reference and constraints about the position and the time. The advantage found was that the spacetime constraints could be solved more easily due to the exclusion of dynamic calculation. The motion created was not necessarily dynamically correct, however, as mentioned earlier this is not vital in computer graphics. Gelichers's method was improved by Lee et al. [25] to a more efficient one using hierarchical calculation. A hierarchical curve fitting technique with a new inverse kinematics solver was used configure an articulated figure to meet the constraints in each frame. The result is an analytical method that greatly reduces the load of a numerical optimization to find the solutions for full degrees of freedom of a human-like articulated figure. A real-time system called a "pin – and-drag interface" was introduced by Yamane et al. [26-27]. This method is mainly based on efficient inverse kinematics calculations. Its basic function was to enable animators to generate a natural motion by dragging a link to an arbitrary position with any number of links pinned in the global frame, as well as other constraints such as desired joint angles and joint motion ranges.

Spacetime constraints rely on the indirect modification through constraints about body parts. There are other methods however which allow the direct modification of motion by controlling the parameters of motion. Human motion was decomposed in the frequency domain and the trajectory was modified for every frequency component by Bruderlin et al. [28] in their motion signal processing method. Several potentially useful operations were noticed: multiresolution filtering, waveshaping and adding smooth displacement maps. Witkin et al. [29]

introduced a variant of displacement mapping called motion warping, in which motion was warped not only in space but also in time.

Unuma et al. [30] proposed a method for modelling human figure motion with emotions. They combined cyclic motions by linearly combining the Fourier coefficients of each degree of freedom. This meant that transitions from a walk to a run could smoothly and realistically be performed. Additionally, more subtle human behaviours could be expressed, for example the “briskness” of a walk could be controlled. However transition from running to walking by this method is unnatural.

Amaya et al. [31] generated emotional motion from neutral motion, by using signal processing techniques to calculate certain emotional transformations. This is then applied to existing motion to produce the same motion but with an emotional quality such as being sad or angry. The difference was represented as a warp of two components: speed and spatial amplitude. These warps are then applied to different neutral motions, in order to generate similar emotional content.

Chi et al. [32] used movement observation science, specifically Laban Movement Analysis (LMA) to extract valuable parameters for the form and execution of qualitative aspects of movements. They believed, the consideration of whole body engagement would lead to naturalness for procedurally generated gestures.

Three tools were presented by Neff and Fiume [33] to adjust a motion’s succession, amplitude and extent, allowing the animator to quickly adjust the various expressive aspects of a motion.

Algorithms were introduced that not only adjusted kinematic properties, but also preserved physical correctness. Tak et al. [18, 34] use Spacetime sweeping to incorporate the kinematic and dynamic constraints in a scalable framework. They ensured that the body’s zero moment point remained inside the support polygon,

which ensures physical validity. Popovic and Witkin [13] created a solution to editing captured motion that take dynamics into consideration. A novel approach for mapping motion between characters with drastically different kinematic structures was conceived. This was realised by targeting the original motion onto a simplified model that maintained essential dynamic features.

Using blending methods is another way of generating human motion, but common problems found with this method is that unless the input motions are similar and are chosen carefully unrealistic motion is generated. An improved blending method was introduced by Kovar et al. [35] that can allow a variety of input motions compared to previous methods. A registration curve was introduced to automatically construct a data structure that encapsulates relationships involving the timing, local coordinate frame, and constraint states of an arbitrary number of input motions. This in turn, expands the amount of different motions that can be blended without the need for manual intervention. The matching and feasibility of input motion is first found by a dynamic time warping technique and then high quality blended motion is generated based on a matching process. The disadvantages to this method can be noticed when logically corresponding parts of motions have dissimilar poses. For example, A character reaching for a glass which in pose one is above the head and in pose two is placed on the floor. In cases like these the timewarp curve created is not as accurate as manual labelling. In order to rectify this problem, a dense set of sample motion is needed to create a smooth transition. This method also fails to enforce any physical constraints like balance.

Learning models and statistical models are also used to generate motion. A style machine based on the Hidden Markov Model was proposed by Brand et al. [36]. The algorithm can automatically segment the data, identify primitive motion cycles, learn transitions between primitives, and identify the stylistic DOFs that make primitives look quite different in different motion-capture sequences. Motion is generated in a broad range of styles by adjusting stylistic parameters. Grochow et al. [37] presented an inverse kinematic system based on a learned

model of human poses. Their system could produce likely poses, in real-time based on a set of given constraints. The parameters of the Scaled Gaussian Process Latent Variable Model are all learned automatically, no manual tuning is required for the learning component of the system.

Other researchers have developed methods that generate motion by combining small segments. Motion graphs by Kovar et al. [38] consider using motion capture data to automatically construct a directed graph called a motion graph. Consisting of both original motion and automatically generated transitions, motion can be generated by building walks on the graph. The framework was also applied to path synthesis. There are however a few drawbacks to the framework: the computational bottleneck in graph construction is locating candidate transitions and the transition thresholds must be specified by hand.

In summary this section of motion synthesis was completed in order to get a greater understanding of common motion synthesis practice currently available. With this in mind, full body motion is not much different to that of hand motion, both are compromised of a moving skeleton in a different joint configuration. Therefore many of the above processes used in full body motion synthesis could be adapted for our application of hand motion synthesis if needed.

2.2 Motion Description

Motion is essentially a series of poses or frames in sequence. Therefore, a description of these poses in a format capable of being read by a computer is needed. Virtual humans are a simplified version of real-life humans and can be defined as a set of bones or joints. Motion poses are therefore rotations and translation of these bones and joints. Several companies and organisations have created their own motion data formats. The most popular and widely used are:

- BVA (Bio Visions Animation data)
- BVH (Bio Visions Hierarchical data)
- ASF (Acclaim Skeleton Format)
- HTR (Hierarchical Translation Rotation)
- CSM (Character Studio's Motion data)

The BVA data format was one of the first to be introduced. It is by far the simplest of all the data formats and has no hierarchical skeleton structure. Each bone segment is represented individually there is no connection information. Due to this at each frame a global position of each bone is recorded. This format is fairly easy to use, but complications can arise.

Biovision released the BVH format shortly after to overcome the problems presented by the BVA format. The file structure is similar to the BVA format, however contains hierarchical information. With the hierarchical system in place, only local rotations of bone segments are needed to drive the motion. This hierarchical information complicates the motion, however makes it significantly easier to modify and combine different motions.

Acclaim introduced their ASF format, achieving the same criteria as the BVH format, but with a more complex file structure. The ASF data format consists of two separate files, one which portrays the skeleton information and another that contains the motion data. The advantage to using two separate files to describe the

complete motion is that during any one motion capture session there is likely to be only one subject (skeleton structure) but multiple motion files.

All of the above file formats, apart from Biovision's BVA format all describe the motion in a hierarchical way. All of them represent the character as a set of bones and joints. The advantages of using the hierarchical approach is that it makes it easier when describing the poses related with each frame. Most importantly, there is no need to specify translations for individual segments. The hierarchical description describes connections between joints and only one translation is made at the root of the skeleton. Due to this, less amount of information is required to be stored in the file, as only the position of one segment relative to its parent needs to be recorded.

2.3 HCI (Human - computer interaction)

Human-computer interaction (HCI) is defined as the interaction between people and computers. With HCI being a very broad topic, it involves the convergence of various subject titles, from mainstream design to computer science and more specifically behavioural sciences. Many more areas of study exist that concentrate on the interaction of humans and computers at the interface stage. HCI is generally segmented into two associated sections, software and hardware. A simple example of a HCI hardware found in every household that has a computer is the pc peripheral, a computer mouse. This device is designed to take direct user inputs by detecting two-dimensional motion relative to its supporting surface, and translating this into electronic computer commands which can be understood by the computer. HCI is not only associated with small scale commercial computers for office and home use, but also large-scale computerised industrial and educational systems such as aircrafts and power stations. One of the most important aspects of HCI is ensuring computer user satisfaction. Due to its multidisciplinary background HCI is also sometime referred to as (MMI) man-machine interaction or (CHI) computer-human interaction. HCI's key goal is to create methods and improve the needs between computers and their users.

The ongoing long term goal of HCI is to reduce the barrier between what a user wants to accomplish when using a computer and the computers understanding of the user's intended task. HCI has been a hot topic of research for decades, where new interfaces and interaction techniques and equipment are being developed constantly. Researchers tend to concentrate on the development of new models and theories of interaction, and the experimentation of new hardware devices.

As mentioned above HCI is a very large area of study which spans across many different disciplines, as far a field as ergonomics and anthropometrics. Due to this in the following section we will concentrate on past and present research which is directly associated with our proposed research. More specifically we are

interested in different hardware input devices and techniques surrounding hand gesture and motion.

A human being's most important physical connection to everything in life from communication to interaction is the human hands. We use our hands naturally unwarily for almost all tasks carried out in day to day life. With computers becoming one of the most useful and widely used systems in our lives, we are often left restricted when confronted with standard mediator devices such as keyboards, mice, and trackballs etc. Even though such devices have proven to be successful at producing accurate and precise communication with computers, they fail to offer a natural connection between the two. First time users must adjust to these devices and learn how to use them which add a secondary unnecessary link between the user and their accomplishable task. If we assess users who have not grown up around computers whether it be at home or work, they find the hardest part of learning IT skills is not understanding software application or operating system layout (which is generally self intuitive) but actually controlling the pointer of a mouse and entering data via the keyboard. Due to this, very little of our natural hand motion is transferred to the task itself.

In an attempt to overcome these issues researches and industry have strived to design, build and study several different concepts that allow computers to communicate directly with the users hands, leaving them free of common limitations found from conventional devices. The development of electronic glove based devices has been an important part of this growth. The widespread availability of glove based devices on the market has led to a vast amount of research projects which use these devices as communicatory tools for computer applications and computer-controlled devices. One of the most popular fields these devices are used in, is within virtual reality and 3D modelling, but have also shown up in fields such as medicine, sports and video games. The following information will review key aspects with regards to hand tracking technologies and applications.

Hand tracking devices surfaced as early as the 1970's with the work from researchers at the Massachusetts Institute of Technology. They began experimenting with a piece of hardware known as Polhemus, this device emits a pulsed magnetic field from a stationary field. Accompanying sensors are applied to the subject's hands which intercept this magnetic field and determine the hands position and orientation in 3D space. This initial study was designed to show the affects of a simple general-purpose input device based on the direct interpretation of hand motion. Even though this study seems simple and trivial by today's standards, the ability to allow a user to indicate graphical elements of interest using their hand as a direct input was an innovative concept which set a foundation for future research of this type.

Hand motion capture techniques are generally segmented into two different categories, these being position tracking and finger tracking. Hand position is specified by the location of the hand in 3D space along with the orientation of the palm. These processes are almost identical to the systems used for full body motion capture, but adapted slightly for hand motion. These were described earlier in this section therefore will not be covered detail here. Whilst finger tracking is determined by assessing the position and orientation of each finger at every joint. This is usually accomplished using glove based devices equipped with electromechanical sensors. Another very common method of capturing finger orientation and position (hand gesture recognition) is via vision-based hand gesture estimation with the use of one camera and algorithmic image processing. This will be discussed in further detail in the following section.

Optical tracking consists of, placing small markers on the hands, either flashing LEDs or small infrared reflective markers. Three or more cameras surround the capture area and pick out the markers in their field of view. The system then correlates the marker positions and uses different lens perspectives to calculate the coordinate of each marker in 3D space.

Magnetic tracking systems use multiple sources and sensors to report position and orientation information. Equipment from leading manufacturers such as Ascension Technologies and Polhemus can track points from three to 20 feet in distance and at 100 Hz. The advantages of magnetic systems is that they do not rely on line of site to determine marker placement, however have the disadvantage of being open to the problem of metallic objects nearby that will affect the magnetic field and give incorrect interpretations of marker information.

The third and final type of hand tracking system available is noticeably the least used. Acoustic tracking equipment emits high frequency sound to triangulate a source's position in the capture volume. These systems are precise within a few millimetres if the microphones are placed correctly. If multiple systems are used together they must be operated at different frequencies to avoid interference. Magnetic tracking systems also rely on line of site between the microphones and sources for accurate results.

In the following section, a brief overview of wearable HCI technologies that are used for finger and gesture tracking will be discussed.

A glove based gesture interface system by Piekarski et al. [39] referred to as “Tinmith-Hand” was introduced to offer two main types of interaction. These being, a menu based system by which each menu item is assigned to a different finger, and the second a manipulation of 3D objects in virtual space. The main application for the Tinmith hand is to offer outdoor augmented modelling of 3D architecture.

Several different hand devices have been developed which can give accurate results but are often criticised for being very expensive and bulky equipment making the usability of them unfeasible. The main concern raised by users is of long trailing cables and essential physical properties to the equipment which apply obstacles to the user when trying to complete a task which should be natural and intuitive to learn.

Due to this a wireless finger tracking concept was proposed by Foxlin et al. [40]. It makes use of an ultrasonic emitter worn on the index finger and a head mounted receiver which can track the position of the emitter in 3D space to an excellent 0.5mm accuracy and from a distance of 400mm.

An attempt to avoid placing markers or sensors on the hand all together was made by Rekimoto [41] with the wrist mounted device, namely “Gesture Wrist”. A wristband with capacitive sensors mounted to it is used to define finger shape. The differentiation between hand gestures is accomplished by the sensors monitoring the cross sectional shape of the wrist along with assessing bulges made by sinews under the skin. Unfortunately the Gesture Wrist can only recognise very few gestures such as a fist and pointing gesture. The main advantage of this system is its unobtrusive and versatile nature, as it can be easily mounted to any commercial wristwatch. A wireless body worn networking device is utilised to avoid the use of cables to the gesture device.

Work by Gandy et al. and Ukita et al.[42-43] respectively, both use infrared imaging technology to reduce the task of distinguishing hand and handheld objects apart from the background. Infrared lighting is mounted near the camera which is fitted with an infrared-pass filter, object closer to the head are easily separated from objects further away by light intensity decrease.

2.4 Computer Vision-based Gesture Recognition

The human hand which is a 20 DOF device is noted to be the most effective natural interaction tool for use with HCI. Currently the only technology which satisfies the advanced requirement of hand input for HCI is glove based devices. As discussed in the previous section the majority of these devices have their own drawbacks, some of which include not allowing the user to interact naturally with the computer controlled environment due to bulky restrictive equipment and long heavy cabling, not to mention being very costly. Some of these systems also require long setup and calibration times. Vision based alternatives have the prospect to supply the user with a more natural non-contact alternative. Therefore, there have been large amounts of research efforts to encourage the use of hands as direct input devices for HCI. This will be useful to professionals in fields such as CAD, 3D modelling and film.

The most challenging aspects of this research are extracting the 3D pose of the hand and fingers as accurately as glove-based devices can. In order for this to be successful correct estimation of kinematic parameters of the skeleton of the hand must be realised. Once these gestures of user commands have been extracted continuous articulated 3D motion signals must be extracted to drive a dynamic virtual interface which in some cases mirrors the complexity of the human hand itself.

Recent attempts have been made to recover the full kinematic arrangement of the human hand, much like glove based devices do. This method offers many problems as the hand has a large number of degrees of freedom, resulting in a large variety of shapes and poses with self-occlusions. If this method is accomplished it can offer great advantages, most importantly full DOF pose estimation is essential for advanced virtual environment application, where advanced control and intricate detail is required. Hand gesture estimation is very similar to full body human pose estimation. Therefore, many of the algorithms

found in hand tracking share similarities to those found in full body pose estimation.

In this section an overview of computer vision-based gesture recognition is given. The section is divided into three main sections: Image pre-processing, tracking, and gesture recognition. In certain systems discussed, these three defined groups are often merged but are still present. The abstraction of tracking and gestures can exist via two different methods. These are defined by systems that rely on the knowledge of the appearance of the hand as an image, and the other as a virtual model of the hand.

The majority of research on gesture recognition requires the detection of dynamic gestures relative to individual computer commands or to understand sign language. Research by Starner et al. [44] recognises American sign language with the use of a head mounted camera pointing in the direction of the hands. This has an advantage over systems that use static cameras that face the user as it does not have to accommodate for movement in body postures. The main idea behind this research was to use skin colour segmentation in order to extract key hand gesture information such as location, orientation, motion and shape. With the use of Hidden Markov Models they were able to recognise full sentences spelt out using American Sign Language even though the vocabulary database was limited to 40 words.

2.4.1 Image Pre-processing

Image pre-processing is a task by which individual video frames are prepared for further analysis. Noisy data is suppressed from the image, along with the extraction of important clues that could lead to identifying the hand as an individual object against the background. Pre-processing can also be referred to as feature extraction.

Different areas of pixels that correspond to different parts of the hand are extracted by a process known as colour segmentation or background subtraction. These areas are then studied to define the location and orientation of the hand. One of the obstacles found with feature extraction is that skin colour varies vastly from human to human and can appear different under certain lighting. Complex segmentation algorithms proposed by Zhu et al. and Dominguez et al. [45-46] attempt to solve these issues however still suffer from being computationally demanding as well as being affected by sudden lighting and illumination changes. Another common problem found with feature extraction is with background objects which can interfere with capture and assume the position of a hand if the shape and colour characteristics are similar. Most researched test new proposals on a static coloured background such as green or blue, or against a known layout background. This is fine for research tests, but in the real world this is unsatisfactory. Some researchers such as Oka et al. [47] have tried using infrared mounted cameras in the hope to enhance the cameras vision of the skin on the hand.

Moving objects in a captured video file can be noticed and defined by computing the inter frame deviations and optical flow. A system presented by Wong et al. [48] proposes a tracking based motion segmentation algorithm. It is capable of tracking moving objects against a moving background with the use of a hand held camera. This system suffers from the ability to determine which of several moving objects is a hand, as well as not being able to perceive a static hand pose.

Some researches use contour detection to extract key information of objects within an image. A stochastic algorithm namely Condensation algorithm proposed by Isard and Blake [49] is an alternative method to using trackers based on Kalman filters. This algorithm uses factored sampling to recognise contour information that represents the hands shape, this method is useful as it does not rely solely on skin colour and lighting. This type of contour detection will result in a large number of edges being detected from the image background and hand itself. Some type of post processing is therefore needed to make this system more robust.

O'Hagan et al. and Crowley et al. [50-51] introduce a method of tracking hands and fingertips in imagery by making comparisons between the initial frame against a template image of a hand or fingertip in real-time. The template image is translated over an area of interest on the image and correlated with each pixel. The pixel which results in the highest correlation is defined as the location of the target object. Problems which can occur with this method known as template matching, is that the system cannot deal with scaling or rotation of the tracked object. A quick fix for this problem can be to update the template regularly, but this runs the risk of the system tracking an interfering object. The location and number of search templates are dependent on the assurance in the tracking of features.

2.4.2 Tracking

Tracking is another major feature which contributes to the complete process of gesture recognition in HCI. A tracking feature is implemented to identify and keep track of hands from frame to frame. This is usually accomplished using two different methods, dependant on what type of feature extraction is employed. One method is to continuously track one major feature from frame to frame or by deducing the location of the tracked hand from the entire feature set.

A popular tracking system known as the Kalman filter is used by modelling the dynamic properties of the tracked hand. This system is probability based distribution demonstrating both the knowledge and ambiguity about the position of the hand. Research by Isard et al. [49] demonstrates certain floors with the Kalman filter one of which is, due to being probability distributed the state of the tracked object is simulated as Gaussian. Therefore, in scenarios with detailed backgrounds the Kalman filter struggles to perform accurately. This however is not the case with predefined backgrounds, as seen in work by [52] where acceptable results can be achieved.

The algorithm mentioned earlier proposed by Isard et al. [49] known as condensation was introduced to avoid the problems found with limiting assumption of normal distribution found with the Kalman filter. The Condensation algorithm work by modelling the probability distribution using a set of particles and then perform all the necessary calculations on this set of particles. This type of algorithm is also referred to as random sampling.

Other work by Gupta et al. [53-54] use random sampling methods which gives very good results even with cluttered backgrounds. A random sampling method known as Monte Carlo is used in conjunction with adaptive colour models by Perez et al. [55] to provide effective tracking of objects with a sudden dramatic change in shape. Mammen et al. [56] use a combination of the Condensation algorithm, colour segmentation and region growing for the successful tracking of two hands whilst avoiding common problems such as mutual occlusions.

MacCormick et al. [57] continued and developed on a method known as Partitioned Sampling that avoids the high costs found with particle filters that track more than one object. A hand drawing application is presented to show that the solution is to located the base of the object first and then establish the configuration of connected link in a hierarchical manner. The system first locates the palm and then determines the angles between the palm, thumb and index finger respectively. These angles are then used to distinguish between a few selected gestures which correspond to different drawing commands. They use a similar skin colour matching process found in [58]. In order to avoid the effects found with a cluttered background they use detailed motion models and background subtraction methods to achieve acceptable results.

2.4.3 Recognition

For hand recognition general well established algorithms from the area of pattern recognition are mostly used. Rigoll et al. [59] introduce a system that can recognise complex dynamic gestures using Hidden Markov Models (HMM). The

models are trained on a database of 24 isolated hand gestures which were performed by 14 different people. They achieved a recognition percentage rate of 92.9%. All gestures were recognised in real-time with high accuracy due to data reduction capabilities of the system. The only downfall of this system is it is unable to recognise gestures continuously which is vital for real-world applications. HMM's are one of the two main popular types of recognition methods. More research using Hidden Markov Models can be seen with Lee et al. [60] and Starner et al. [44].

Correlation is the other most popular method of recognition with hand gestures. This can be seen in research by Crowley et al. [51] where cross-correlation is used as a means of tracking pointing devices for a digital desk. Further research which also uses correlation methods can be found with Birk et al. [61] and Fillbrant et al. [62]

Neural Networks are the third recognition method however suffers from problems when modelling non-gestural patterns. This is realised in research by Lee et al. [60].

More recently research by Bray et al. [63] proposes a hand tracker based on 'Stochastic Meta-Descent' (SMD) for high dimensional spaces. This algorithm is based on a gradient descent approach that uses adaptive and parameter-specific step sizes. The system is improved and made more robust by integrating a deformable hand model with actual anthropometrical measurements based on linear blend skinning. In order to improve robustness of this system further it is suggested that multiple cameras could be used also adding pseudo joints to the model would improve the tracking.

Lin et al. propose a new representation for the nonlinear manifold of articulated motion, with a stochastic simplex algorithm that facilitates a very efficient search. They combine known methods of the Monte Carlo technique with a Nelder-mead simplex search when the gradient is not readily accessible. Their experiments

show that their algorithm is robust in tracking hand motions in cluttered backgrounds.

Research by Stenger et al. [64] proposes to solve the problems of recognising multiple patterns. Classifiers are arranged in a tree in order to recognise multiple object classes. Each different pattern class corresponds to the hand in a different pose, or set of poses. With the advantages of being easy to generate and being labelled with a known 3D pose, this allows them to be used in a model-based tracking framework.

Recent work by Zhou et al. [65] introduces the concept of eigen-dynamics and proposes an eigen dynamics analysis (EDA) method to which learns the dynamics of natural hand motion from labelled sets of motion captured data. Their experiments on both synthesized and real-world data demonstrate the robustness and effectiveness of their techniques.

In summery more recent work by Wang et al. [66] proves that a single camera can be used to capture articulated hand motion with the user wearing a simple Lycra glove imprinted with a specialised custom coloured pattern. This research provides evidence that this type of system can be used in a home or work environment for everyday use with relative restrictions. For our research purposes the use of a Motion capture system is suitable for our intended application.

3 Framework of Hand Gesture and Motion Based Design Method

This research proposes a novel method to generate 3D product design models in real-time using hand motion and gesture. In order for this to be successful we must outline the design needs of the proposed system, and determine how we will achieve our final goal. With this in mind we decided to look closely at the end users needs in this case the designer. In this section these goals are discussed and the proposed research is drafted.

3.1 Design Needs

The most important aspect of this research is to consider the needs of the end user, in this case the designer. The person using our proposed system must be able to efficiently and effortlessly complete their intended task. We propose a system that can be adapted to different expertise, study, and applications. Our system will be designed to be used either for generic 3D modelling applications such as 3D sketch based modelling or specific 3D modelling applications such as Architectural Modelling, Conceptual Design Modelling, and various Engineering Based Modelling etc.

Rapid conceptual design models are becoming widely more used in the daily lives of designers and engineers, as discussed earlier, it would be ideal of them to interact with computers in real-time enabling them to produce accurate electronic conceptual design models. This leads us to our second important goal, which is to allow for user interactive communication for direct response. To make this process as simple and natural as possible for the designer a motion capture system is used to offer hands free interactive recording of 3D hand gestures and motion. The following sections discuss how the above objectives can be achieved in more detail.

3.2 System Design

The design of our systems structure and framework is important as it allows us to understand where the user fits in amongst our complete proposed system. The user must have a clear understanding of where they belong in the chain, in order to use the system to it's full potential. Please see the flow chart below (Fig. 3.1) which gives a detailed overview of our proposed system framework and structure.

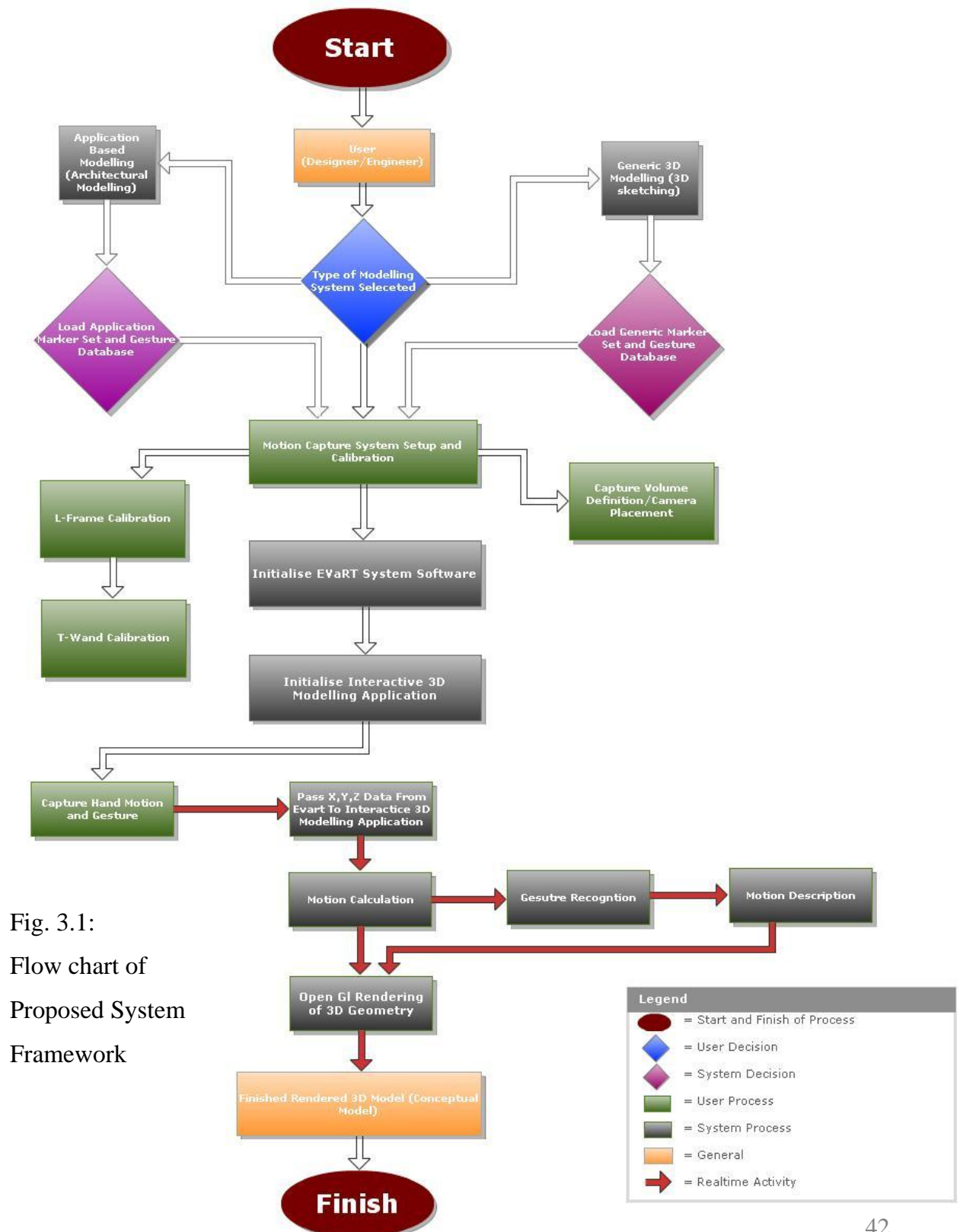


Fig. 3.1:
Flow chart of
Proposed System
Framework

3.3 Advantages and Disadvantages

Current methods of HCI, Hand Gesture Recognition and rapid 3D modelling all have their advantages and disadvantages as discussed in the literature review (section 2). Our initial goal was to try and improve on some of these disadvantages, and create a system that uses a hands free approach to rapid conceptualise 3D models in a 3D space. Before we decided on using an optical motion capture system as our input device for recognising 3D hand motion and gesture, we considered several readily available equipments such as the Nintendo Wii controller and web camera/image processing. Even though these devices have many advantages which include being simple and cheap to obtain as well as mass produced in a commercial environment, there were several aspects of their use or function which would either be discarded for our proposed system or take too much time and expertise to implement and extract the needed data.

For example the Nintendo Wii remote of which it's primary job is motion sensing, allowing the user to interact with and manipulate items on screen through gesture recognition and pointing through the use of an accelerometer and optical sensor technology. Whilst the above technology is useful for the gaming console it is paired with, for our application we would not need the accelerometer, and its gesture recognition capabilities are limited to simple single handed gestures. Furthermore the Wii remote can by no means differentiate between different static hand gesture signs. These are the main reasons why we decided against this use of input device.

Other Devices such as a simple camera/webcam setup were also discarded as they require complex knowledge and implementation of image processing algorithms, which not only take time to execute but also are unnecessary if a motion capture system is used which can easily provide us with the 3D location of X,Y,X coordinates in 3D space. This core data is the information behind all motion and gesture based systems.

3.4 Identification of Key Feasibility Studies

We must be realistic in our approach when considering our final goals and objectives in order to successfully realise our proposed system which will be capable of generating 3D product design models in real-time using hand motion and gesture. As this research proposal is part of an MPhil degree, it must be completed within one academic year. The type of proposed work will involve software development and hardware testing which is complex and with limited programming knowledge, will be very time consuming. It would be unrealistic to assume that we can finish our complete design system as shown in the flow chart in Fig. 3.1.

We therefore set about identifying and defining two key aspects of our overall system that if achieved, would provide sufficient evidence that with further time and work dedicated to programming and development a complete real-time interactive system can be realised.

Below is a breakdown of these two feasibility studies which must be completed in order for our proposed system to be feasible. Within each section further details of what needs to be achieved is given.

- Feasibility study 1 – Hand Motion and Gesture
 - ✓ Recognise Hand motion and Gesture to model 3D Geometry.
 - ✓ Ascertain whether motion capture system is suitable for real-time interaction.

- Feasibility Study 2 – Real-time Interactive Communication
 - ✓ Render 3D geometry using OpenGL in real-time.
 - ✓ Program interactive 3D application capable of communicating in real-time with the EVaRT system.

3.5 Hand Gesture and Motion

We decided in order to make our system as versatile and adaptable as possible to work directly with different areas or Design and Engineering, it must be capable of accepting different types of hand gestures and motion. These include static hand gestures and dynamic hand gestures as well as modelling via 3D motion sketches. We propose to cover these types of hand motion and gestures across the feasibility studies. This will be discussed further in Feasibility study 1 – Hand Motion and Gesture (Section 4).

3.5.1 Generic Applications

For generic modelling applications, 3D motion sketches are used to realise 3D geometry. This type of application is useful for quick free hand design concepts which do not belong to a specific type of industry. This approach has similar characteristics to that found in research by Masry et al [67]. Noticeable differences are that they use a tablet pc for 2D line entry which is transformed to custom 3D primitives. The acknowledged advantage of our proposed system is that no 2D -3D translation is needed, as we will have real world 3D point data.

3.5.2 Specific Applications

For Application specific modelling, a set of hand gestures must be defined and used to describe different product parts, components etc. This type of modelling is useful for specific types of engineering application as seen in our preliminary study – Feasibility study 1 (Section 4) : Architectural modelling. This type of system offers the user a more direct and quick approach to modelling of items which have been predefined, however have limitations of only being suited to one type of modelling exercise.

3.6 Real-time Communication

For our second feasibility study, (Feasibility Study 2 – Interactive Real-time Communication Application – Section 5) a direct communication path must be made between the motion capture system and the user, for real-time interactive feedback to be accomplished. In order for this to be achievable three main areas must be considered and researched, these being data capture, data exchange, design interface. These will be discussed briefly in the remainder of this section, 3.6.1 – 3.6.3 and in more detail in the relevant section 5.

3.6.1 Data Capture

All data capture will be done via the hardware provided by Motion Analysis corporation. All data will be processed and recorded in a track file (trc.) format which is encoded in ASCII format. For further examination and inspection for real-time reading of the 3D X,Y,X coordinate information, the track file can be opened in Microsoft excel in tabulated format.

3.6.2 Data Exchange

Data exchange will be primarily controlled in two ways, these being hardware based and software based. With regards to the hardware level all captured motion will be transferred from the mainframe computer which controls the motion capture hardware to a secondary slave computer which will run our interactive control application, via Ethernet connection. The system will also be designed to output all captured motion to a saved file for reference purposes after the capture session and design modelling is complete.

On a software based level, the data produced by EVaRT must be transferred internally to our dedicated software application. In order for this to occur a data pipeline must be implemented between the two. This is essential before any type of OpenGL Graphics or gesture recognition can begin. This will be done using the Motion Analysis software development kit (SDK).

3.6.3 Design Interface

The design of our interface software application is essential for the user to clearly see what they are modelling. For successful results this part must put the designer in full control of the system to do this as much of the interface design must be customisable to the users needs. This will be explained further in Feasibility study 2, section 5.1

3.7 Overview of Proposed Research

For our research intentions, the ready availability of an optical marker based motion capture system by Motion Analysis will be utilised for the capture of hand motion in electronic format. A set of hand signs and gestures will be defined, and classified into groups. These groups will be ordered by type of component and surface generation. The defined hand gestures will hold key information that describe different design parameters of the product being conceptualised. One hand will be used to determine what part of the product will be drawn and the other hand used to give further information about the product's shape, placement, dimensions, etc.

Initial testing will begin with a joint collaborative research project with Dr Xiao Yi, School of Civil Engineering and Architecture, Beijing Jiaotong University, 100044, China. This research will focus on the creation of architectural conceptual design in an offline state. The main purpose of this research is to establish whether the optical motion capture system used is suitable for our application, before software development begins for real time processing of hand gestures. Furthermore to ensure that hand gesture and motion can be realised.

Once complete, a new set of hand signs and gestures will be ascertained for use with conceptual product design. A different approach will be applied for product design models when compared with that used on the architectural design research.

A 3rd party software application will be designed and programmed capable of communicating in real-time with the Evart 5.04 system with the use of the Motion Analysis SDK. The program will be able to display OpenGL graphics and act as a 3D interactive interface for controlling our system.

4 Feasibility Study 1 – Hand Motion and Gesture

The main goal of the initial testing research project is to design a hand motion and gesture-based rapid 3D architectural modelling system, allowing designers and engineers to freely sketch out 3D conceptual design models in a 3D workspace. Initial testing shall be carried out in an offline state, where motion will be captured and cleaned up after which design gesture processing will be computed using a numerical computing environment such as MATLAB. 3D curve and surface generation will be carried out afterwards and transferred to a 3rd party 3D modelling software application such as Alias Studio. In turn this will enable us to ascertain whether or not the optical motion capture system used is suitable for future planned research of real-time hand gesture modelling for product design and if hand motion and gesture recognition can be realised.

The left hand will display hand signs and the right will perform motion sketches with the use of a pointing device, in this preliminary study a pencil is used. All information derived from the hand signs and motion sketches, will be processed offline to generate 3D models. The flow diagram illustrated in Fig. 4.1 represents the complete process for the initial study of 3D architectural design modelling.

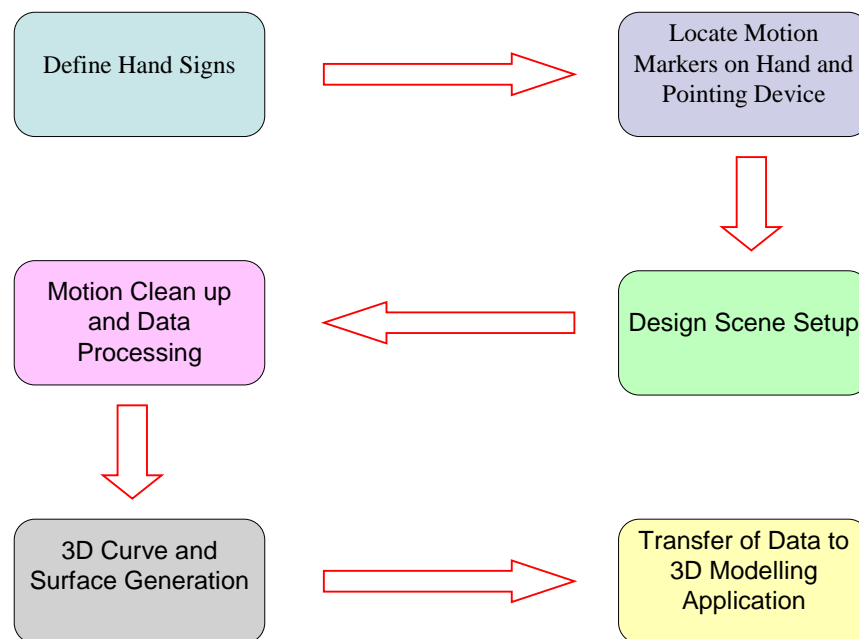


Fig. 4.1 3D Architectural Design Modelling Flow Diagram [68] – (Fig.1)

4.1 Hand Gesture and Sign Language for Architecture

Sign language is the visual transmission of sign patterns used amongst the hearing impaired as a means of communication. This alternative to acoustically conveyed sound patterns dates back to the second century BC. Different sign languages are developed amongst communities of deaf people, thus cultural and geographical differences will lead to the generation of distinct sign languages. Due to this in neighboring regions sign languages may be fairly similar, however the same word or phrase can still be expressed by different hand shapes dependent on the differences in sign languages. Sign languages are considered to be just as complex as any oral language, and have every linguistic constituent necessary to be classified as true languages. Examples of this can be seen when choosing just two letters from British and American Sign Languages and comparing the two. Fig 4.2 represents the differences between the letter B and M in the British Sign Language (BSL) and American Sign Language (ASL).

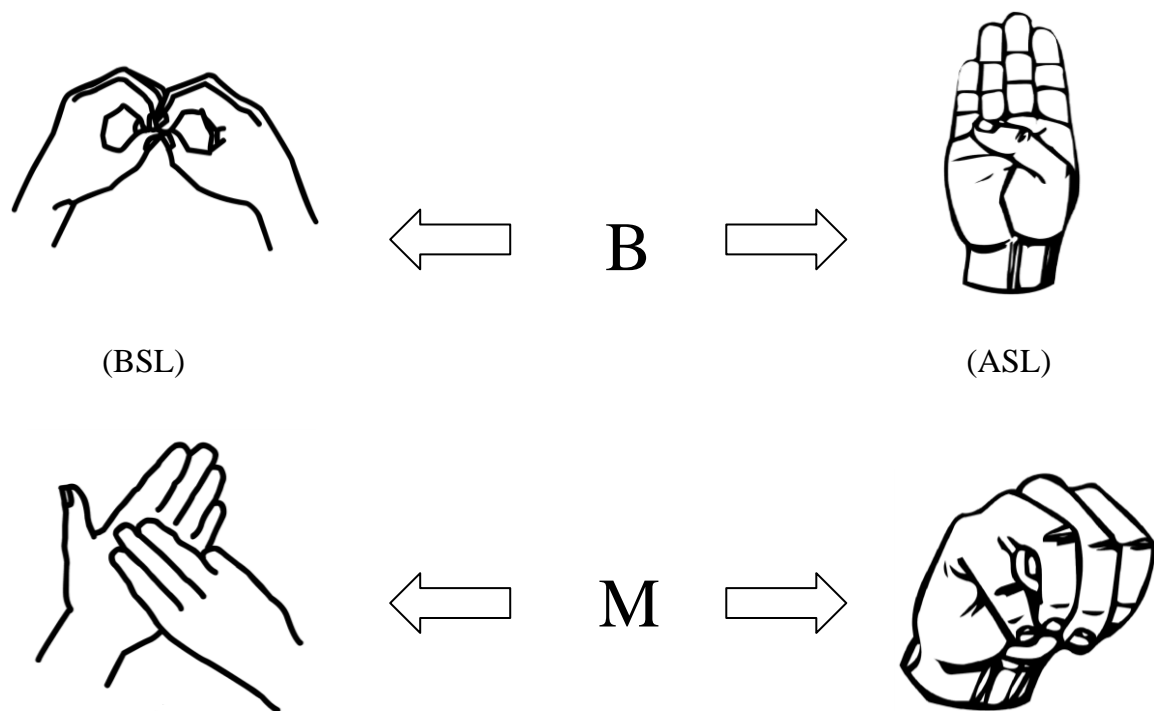


Fig. 4.2 Differentiation between alphabet letters B and M in the British Sign Language and American Sign Language. (Images from www.coloring-pages-book-for-kids-boys.com – Date visited 29/09/09)

It is important when designing our set of hand signs and gestures for the architectural system that a universal sign language is developed that can be easily understood and recognised by both architects and designers no matter what their nationality, culture and education. Concurrently it must also be unique and clear without any confusion, allowing for a fast and easy learning process for initial users. Sign languages can be divided into groups according to hand configuration, movement and place of articulation. More often, hand motions and spatial sequences are linked together to express complex concepts. Effectively, a designer's ideas can be expressed with a static gesture accompanied by movement and position.

In order to successfully design a set of feasible hand signs for architecture, an engineering structure such as a house must be considered and segmented into basic building units. These units commonly found in architectural structures are displayed in Fig. 4.3. A set of hand signs have been designed to match the architectural structure presented in Fig. 4.3. Our design of hand gestures is composed of the left hand and the right holding a Marker-Pen, symbolising different design information. The left had will describe what type of structure is to be designed/drawn by means of a given hand sign, whilst the Marker-Pen will provide the system with other vital information such as the unit's size, shape and relative location. When the above two communicatory features are combined 3D architectural models can be rapidly produced.

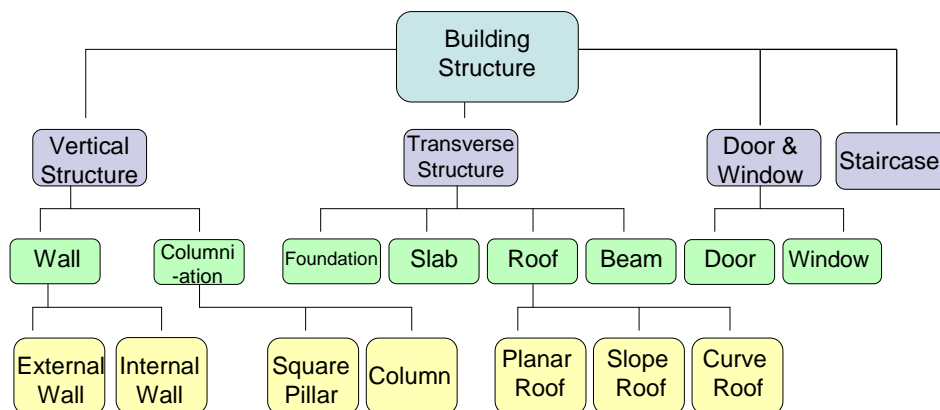


Fig. 4.3 Segmentation of Architectural Building Structure [68] – (Fig.3)

An architectural building has been segmented and divided into groups of components according to its structure. The four main components to the building structure are identified as vertical and traverse components, such as walls and beams. Doors/Windows and staircases are the third and fourth main components which make up the core of the engineering structure. A hierarchical system has been employed to break these components down further into subcategories, for example, walls can be defined as external or internal walls.

Based on the above classification we then set about defining a set of hand gestures that best described each individual building component. Four types of the left hand gestures have been designed corresponding to the four main types of components. The left hand gestures were designed as to be easily interpreted and recognised by architects and designers. Vertical structures are conveyed by vertical hand gestures. An example of this is shown with the gestures that represent a wall and columniation. When a palm and a fist are combined with the thumb up or down, four basic units can be gestured. Therefore, a vertical palm with the thumb up or down means an external or inside wall (Fig 4.4A and 4.4B), whilst a fist with the thumb up or down shows a square pillar or column respectively (Fig 4.4C and 4.4D).

Likewise, a set of horizontal hand gestures have been designed for the transverse components and the door and window components. For example, flat structures such as foundations, slabs (or floors) and beams are represented by a horizontal palm (Fig 4.4E), whilst a horizontal palm with fingers separated represents a roof (Figure 4.4F). A fist means a window (Figure 4.4G) and with the thumb up indicates a door (Figure 4.4H).

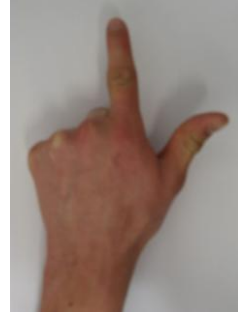
In addition to the above left and right hand gestures, we also introduced various editing gestures like 'delete' and 'finishing' which were designed differently from the normal modeling gestures.



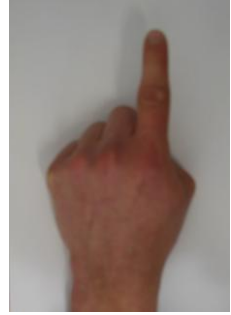
(A) External Wall



(B) Internal Wall



(C) Square Pillar



(D) Column



(E) Flat Structure



(F) Roof



(G) Window



(H) Door

Figure 4.4 A-H: Examples of the left hand gestures [68] – (Fig. 4)

4.2 Marker Placement

All hand gestures in this initial testing will be captured with an optical based motion capture system. Facial markers are used for this test as they have a very small diameter of 4mm. This will hopefully increase performance and reduce system noise. The right hand will not be marked. We have chosen a Marker-Pen as opposed to the hand to act as the input for design motion.

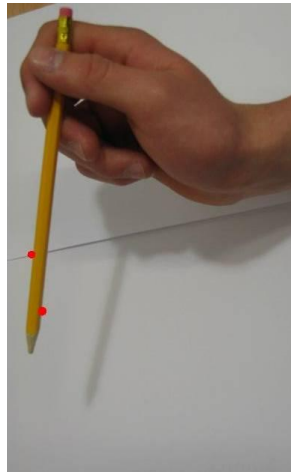
Our next task is to determine the amount of markers and their placement on the human hand and Marker-Pen. It is vital to the success of the capture process that this is completed correctly. Variations in the number of markers and their position will have a large affect on the systems performance. The optical motion capture system used during the capture process is very sensitive to background noise. This noise can cause occlusions between markers and a process known as marker switching takes place. This essentially will cause major problems with algorithm processing if the trajectory of one marker interchanges with that of another. Prior to this a large amount of post processing would need to be completed in order for gestures to be recognised.

We therefore set about designing a marker placement grid that incorporates the use of the supplied motion analysis skeleton template. The skeleton template is used to produce strong links between identified markers in the hope of producing a robust template, with the promise of avoiding the issues noted above with marker-swapping and marker occlusions.

Initially we began by placing just one marker at the tip of the pencil however after a preliminary test session we found it far too sensitive and the marker was quickly confused with adjacent markers on the left hand. A second attempt with two markers had a similar negative result. We finally settled on 3 markers in a triangle configuration. Although initial tests showed this is the most reliable pen marker

set out of the three we tried, only the original marker of the tip will be acknowledged for motion processing. The other two are solely required to increase the robustness of the template and eliminate large quantities of post processing.

Marker-pen configuration set two and three are presented in Fig. 4.5A and Fig. 4.5B respectively.



(A)

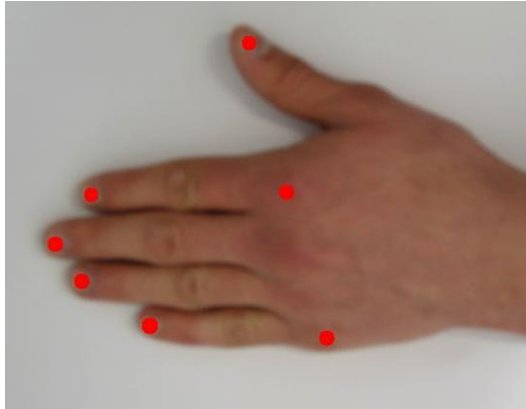


(B)

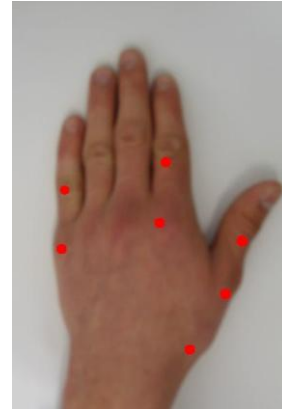
Figure 4.5 A &B: Marker-pen marker configurations [68]- (Fig.5)

With regards to the design of the left hand marker set, several combinations were tried and tested. Over time it became apparent that fewer markers yielded better results. Unfortunately there is a limit as to how many markers can be applied, as the system must be able to distinguish easily between several different hand gestures. Our goal was to strike a balance between having enough markers on the left hand to experience consistent hand gesture recognition whilst at the same time limit the amount of markers to avoid noise and occlusions. Just one extra marker on the left hand may increase the chance of occlusion or confusion, as well as complications in the data for generating 3D curves or surfaces. Ultimately a minimum of 8 markers were used on the left hand to represent all gestures. Initially markers were placed on the finger tips and knuckles as shown in Fig. 4.6A. However, with trial and error, certain gestures such as a closed fist and hand closed laid flat, became problematic. Markers at the finger tips began to switch as

they were too close to one another or became entirely hidden from the cameras field of view. Finally, the markers were moved from the finger tips to Proximal Interphalangeal joints and 1st joint of the index and little finger. Two more markers were added to the Distal Interphalangeal joint and 1st joint of the thumb. This final marker placement configuration is shown in Fig. 4.6B. This marker placement proved to be the most reliable and robust tested in order to generate all gestures.



(A)



(B)

Figure 4.6 A &B: Left hand marker configurations [68] – (Fig.5)

The gesture coordination of both the left hand and Marker-Pen have been designed to work in conjunction with one another. The designer/architect must first display a left hand gesture to begin a given modeling command, and then move the Marker-Pen with their right to describe appropriate geometric information

4.3 Motion Capture System

The hardware used to capture the hand motion of humans was the Motion Analysis Eagle Digital System. This is an optical motion capture system, consisting of seven Digital Cameras, the Eagle Hub, to which all cameras are connected and uplinks to a computer terminal. All the hardware components are controlled by EVaRT 5.04 Real Time software. It is within this software where all data is recorded, processed and displayed, and where post processing takes place.

The system is capable of capturing the most complex of motions with extreme accuracy, to the nearest 2mm and up to 200 frames per second. For our purposes a capture rate of 120 frames per second was adequate along with facial markers 4mm in diameter. As the Eagle system has real-time capabilities it is possible for the user to see capture results at the same time as the subject is performing the specific hand gestures and motion sketches.

The different stages of a complete motion capture session can be typically summarised as follows:

- Studio set-up for multi-camera capture
- Calibration of motion capture system
- Capture of motion
- Clean-up and post-processing of point cloud data.

A brief description of the general tasks required for a capture session will be described in the following sections.

4.3.1 Planning a session

Before a capture session can begin a suitable space must be found. A room with no external light is ideal but very hard to find, a suitable alternative was a room with specific black curtains that blocked out 100% of natural light. An open space with flat flooring and a matt finish surface with no furniture near by, in the cameras field of view is essential. In the case of capturing samples of full body

human motion, the space must be at least large enough for a subject to walk freely around. Fortunately for us, our subjects will be stationary and in the seated position, therefore a capture volume of approximately only 3m by 2m squared is required. This in turn, will allow for the closer placement of the seven cameras to the subjects hands, in the hope of capturing clean data.

4.3.2 Setup and Calibration

After familiarisation with the system it became apparent that larger volumes were more prone to occlusions and noise and hence gave less accurate results. After capturing data at several different locations, the most optimum capturing sessions, in which the data used for the results, was captured in a large indoor laboratory with specifically designed black motorised curtains which blocked out all natural light. A design scene within EVaRT was setup to accept a maximum capture volume of approximately $2\text{m} \times 3\text{m} \times 1.5\text{m}$. A small lab desk of dimension $90\text{cm} \times 60\text{cm} \times 75\text{cm}$ was placed in the centre of this capture volume along with a lab chair for the subject to sit on.

An L-frame calibration device was then placed at the centre of the desired capture volume. This frame has four markers, which are used by the EVaRT software to define the XYZ axes of the capture volume. Each camera sees the markers and registers their origins via 3D triangulation. If successfully calibrated the markers appear in the 3D display, and the cameras orientate themselves to their correct position in the 2D display panel.

A T-frame wand device, exactly 500mm wide, is then used to establish the camera linearization parameters. The wand device is held by an assistant in the capture volume and waved energetically across the XYZ planes. Red markers fill the 2D displays for each camera, based on how many repeated markers there are, basic calibration is then complete if it is a low number for each camera. A low deviation is required of less than 1 in order for calibration to be successfully accepted before real-time capturing can begin. If the deviation is above one, this often means there is a problem with an objects reflective surface in or around the

capture volume which is interfering with calibration results. During initial testing we found the legs of the desk were of black reflect paint and caused distinct interference. To overcome this we used black cloth similar to that used for the curtains in order to mask the legs. A second problem could arise where one camera is in the field of view of another adjacent camera. In this case the noisy area is either digitally masked or the camera must be moved to another position, in which the previous L-frame calibration procedure must be repeated.

A mixture of two different camera configuration setups suggested by motion analysis for 6 and 8 cameras was considered for our first attempt at motion capture. These configurations can be seen in Fig. 4.7A and B respectively.

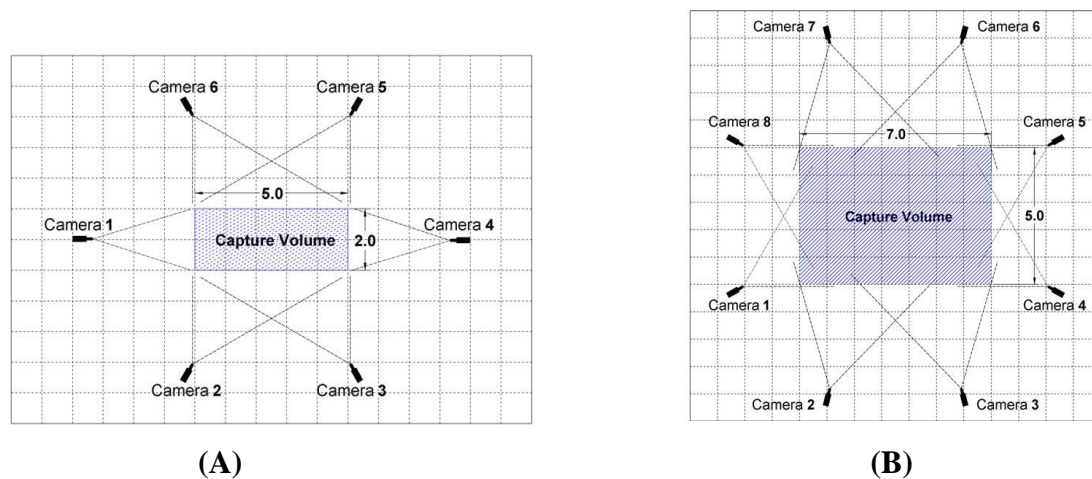


Fig 4.7 A&B: 7 and 8 camera configuration setup (Images from Motion Analysis Manual)

With both of the above setups, after adjusting the angles of each camera slightly to accommodate for the missing or extra camera, we found them to be unsuitable for our given task. The main reason being the markers on the subject's hands are mostly located on the top, and gestures performed with the subjects hands in an elevated position facing them, concealed the markers from the majority of the cameras. As the motion capture system works on a 3D triangulation process to acquire a markers location in 3D space, with so many cameras not being able to see the markers at any given time large gaps in data were missing midway through a hand gesture.

In the aid of finding a solution to these camera placement issues, we adjusted the camera tripods higher and placed more around the back of the subject as opposed to equally spread out circularly around the capture volume as before. This fixed all marker tracking issues, and we continued capturing with the camera configuration shown in Fig. 4.8.



Fig. 4.8 Final 7 camera configuration setup [68] – (Fig.6)

4.3.3 Hand motion capture and recognition

During our main motion capture sessions the positions of the markers on the left hand and on the Marker-Pen were captured and recorded via the software EVaRT 5.04, the motion capture rate was set to 120 frames per second (fps). Fig. 4.9 displays the marker template and marker identity of the left hand and marker-pen.

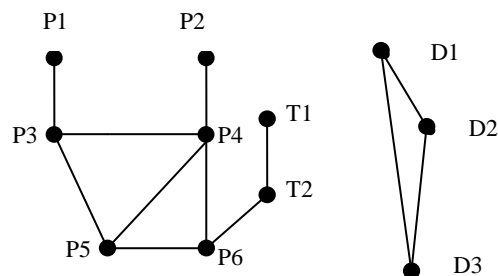


Fig. 4.9: Motion capture templates and identities [68] – (Fig.7)

Based on our proposed marker placement configurations presented above, two skeleton templates were designed and saved to the project file of our motion capture session. Templates for both the left hand and Marker-Pen were created by joining links between each marker, in a form which was regarded as the most rigid in order to create a robust skeleton template. The first frame of every capture session was considered the base pose and the user should return to this position at the beginning and end of every captured scene. This technique is used to help the template associate the raw markers with their corresponding identities and linkages. With this in mind, approximately three to four simple generic hand gestures like lifting the hand up and down were performed, in order to strengthen the template and its awareness of the marker configuration and location in the defined capture volume. With this complete, and the template successfully extended, the template was strong enough to begin a full session of motion capture. A screenshot of the system fully functional is shown in Fig. 4.10. The group of coloured markers in left hand corner represent the left hand and the three makers on the right are that of the marker-pen.

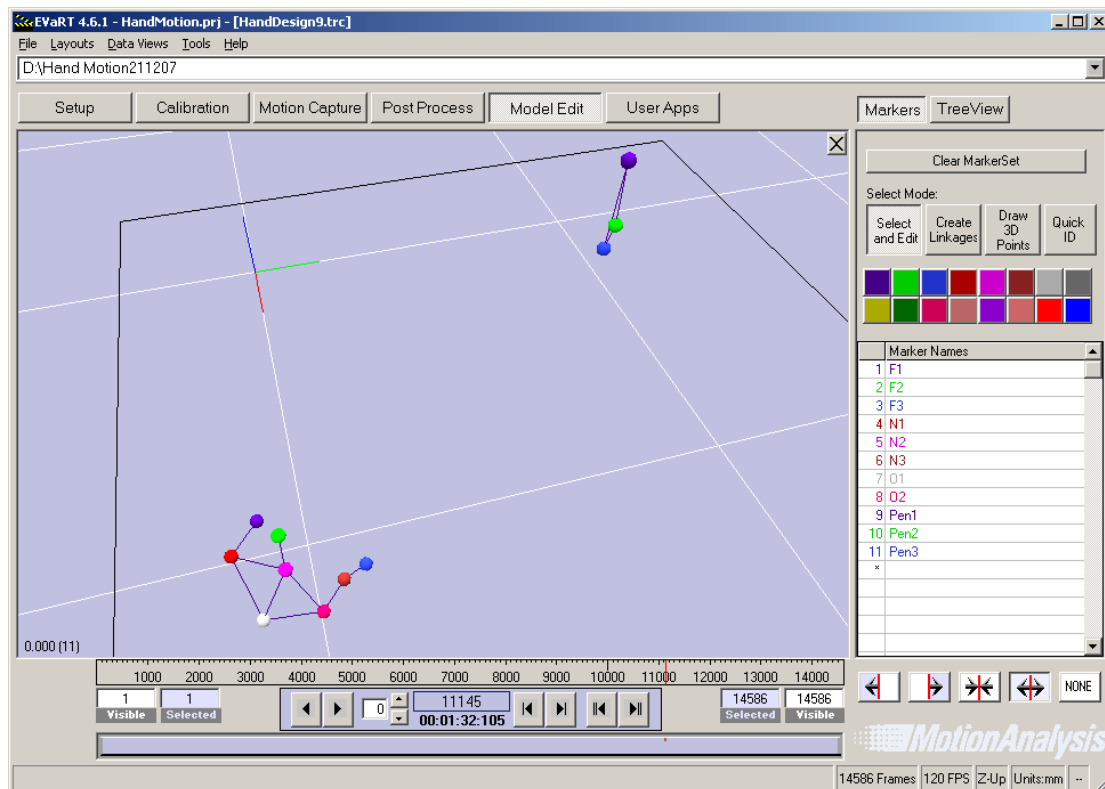


Fig. 4.10: Marker templates in a real-time motion capture scene. [68] – (Fig.8)

When each design task was completed, all 3D maker data in the form of x, y, and z coordinates were saved to a track file with extension trc./trb.. The Raw data was also saved for future research of real-time processing. Raw data enables EVaRT to rerun the scene in real-time as though motion capture was taking place at that time. Both trc. and trb track files are encoded in ASCII format which can be read by Microsoft Excel in a tabulated format. Processing this data obtained, would reveal hand sign commands and their associated design information. As the templates used were trained before starting each design task, the data collected need minimal if any post-processing editing and cleaning up.

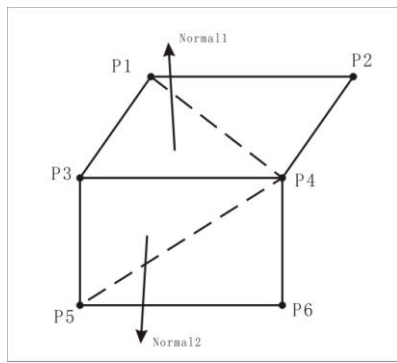
Hand signs performed by the user on the left hand can be differentiated from one another via the markers position in 3D space. A planar plane can be recognised relative to the position of three markers. Marker orientation information (normal) such as Normal 1, 2, and 3 shown in Fig. 4.11A and 4.11B, are used together with the location information of markers T1 and T2 shown in Fig 4.9.

Each normal and its associated direction and angle is used to calculate and recognise different hand signs.

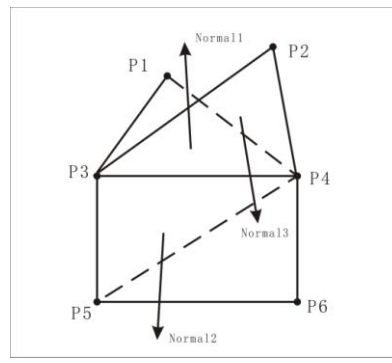
- N1 (normal 1) is the normal of plane 1 – P3/P4/P1.
- N2 (normal 2) is the normal of plane 2 – P5/P4/P3
- N3 (normal 3) is the normal of plane 3 – P3/P4/P2

The hand gestures shown in Fig. 4.11C, Normal 1 and Normal 3 are fairly similar, while with the pose presented Fig. 4.11D, they are quite different.

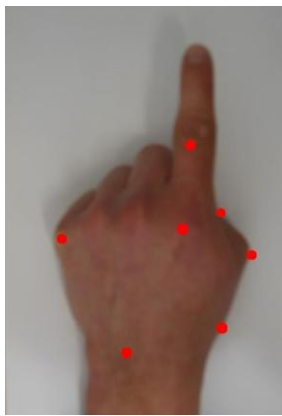
Based on the values of N1, N2, and N3, the angles between them are calculated. Angle θ is the included angle of Normal 1 and Normal 2 shown in Fig. 4.11E, and β is the included angle of Normal 2 and Normal 3 shown in Fig. 4.11E. For our system to rapidly identify vertical structures, ϕ is defined as the angle between N2 and a horizontal plane with a fixed normal (0, 0, 1). The angle between two vectors is calculated from their dot product. [68]



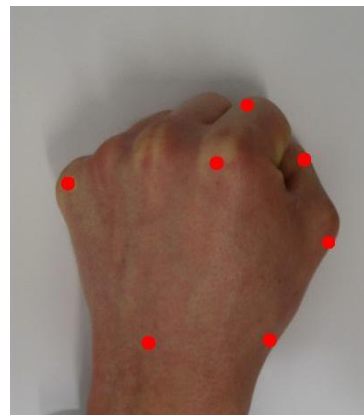
(A)



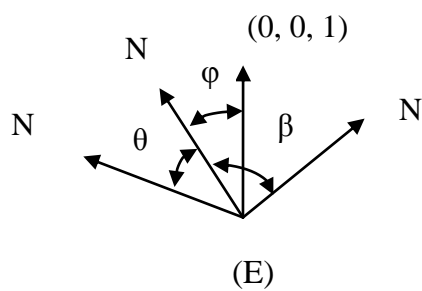
(B)



(C)



(D)



(E)

Fig. 4.11 A-E: Gesture recognition [68] – (Fig.9)

A rule-based recognition method is used to recognise hand signs. Below is a list of a few example rules.

(1) If angle $\phi > 45^\circ$, plane 2 is roughly a vertical plane, and therefore a vertical

structure is displayed.

(2) If angle $\varphi > 45^\circ$ and angle $\theta < 30^\circ$, N1 and N2 are nearly parallel, the sign is a wall. Under a wall structure, the distance δ between P4 and T1 is used to further classify wall structures. If the distance δ is bigger than a threshold, it means that the thumb is close to the palm, the sign means an external wall. Otherwise, it means an inside wall.

(3) If angle $\varphi > 45^\circ$ and angle $\theta > 30^\circ$, but angle $\beta < 30^\circ$, then the gestures show in Fig. 4.11A and 4.11B equate to a square pillar or a column. Furthermore, if the δ is bigger than a given threshold, it represents a square pillar, if not it corresponds to a column.

(4) If angle $\varphi < 45^\circ$, plane 2 is roughly a horizontal plane, this shows a transverse structure.

(5) If angle $\varphi < 45^\circ$ and angle $\theta < 30^\circ$ and angle δ is small, then the sign is a flat structure.

(6) If angle $\varphi < 45^\circ$ and angle $\theta < 30^\circ$, and the distance between P1 and P2 is bigger than a threshold, the gesture shown is a roof.

Although different thresholds were used in our rules, they were easily obtained. These thresholds were determined by measuring given data-sets against certain gestures. [68]

4.4 Motion Sketches and Initial Data Processing [68] – P682

The 3D position of marker D1 is used to specify motion sketches performed by the user. Once the system receives a new modeling command with the left hand, the right can move the marker pen to a start point and begin to draw a sectional curve. Based on type of sketch command drawn by the user, the 3D sketch data can be located and extracted from the recorded motion capture data. Once this initial sketch data has been identified and obtained a curve smoothing process is applied to the 3D curve data. This is completed by selecting points every 30 frames, in order to avoid subtle distortions caused by natural hand vibrations. The curve data is then segmented by finding key points along the curve where it changes path direction dramatically. Once this segmentation process is completed, the data is reduced significantly in size. The curve is represented as a 3D poly line in time sequence:

$$\{s_i\}, i=1, 2, \dots, N. S_i \text{ is a 3D key point.}$$

For 3D building models to be constructed from 2D sketches, it is crucial to evaluate the 3D sketches along their relevant axes. A general three-dimensional form can be defined by a 2D shape lofting along a path. The motion description of the Marker-Pen can be classified into two different categories: drawing a 2D shape on a flat surface and moving along a lofting line direction. An example of this can be seen when defining an external wall, the pen is used to draw the shape of the wall in the X-Y plane, afterwards moving the pen over the plane along the Z axis. The height at which the Marker Pen is displaced from the surface defines the height of the wall. This can be seen in Fig. 4.12A.

From previous processed data $\{S_i\}, i=1, 2, \dots, N$, the sketch will be cleaned up based on certain geometric constraints. For a sub-segment, $j=m, m+1, \dots, k$, we can determine the points on this segment move towards a similar direction, from this we are able to use the direction to tidy-up the line segment. The direction is determined using the following three processes:

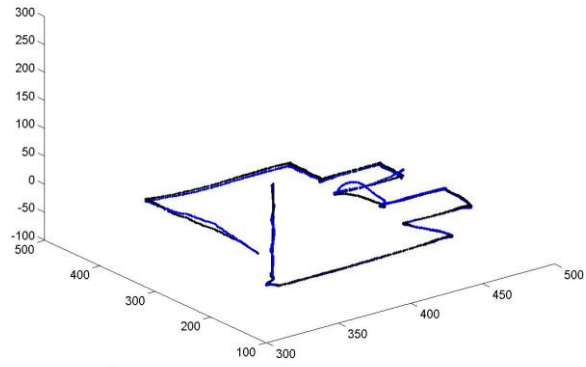
- Compute $\delta x = X_k - X_m, \delta y = Y_k - Y_m, \delta z = Z_k - Z_m$ and
 $\delta \max = \max(|\delta x|, |\delta y|, |\delta z|).$

- Determine the direction $(dx, dy, dz) = (\delta x / \delta \max, \delta y / \delta \max, \delta z / \delta \max)$ if any component is near zero, it must be assigned to zero.

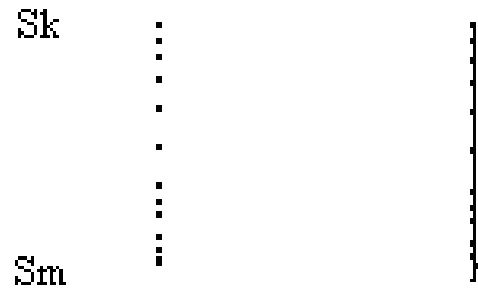
- If a directional component among dx , dy and dz is equal to zero, the segment will be projected to a corresponding plane through the first point S_m and if the segment is a straight line all points between the beginning and end point must be removed. Fig. 4.12B shows a vertical line segment, after the above two steps, dx and dy are equal to zero. This segment has therefore been projected twice. Firstly, for $dx = 0$, the corresponding projection plane is the YZ plane passing through the point S_m , i.e., the plane equation is $X=X(S_m)$. After the projection, all points have the same X components. Secondly, for $dy = 0$, the projection plane is likewise determined as $Y=Y(S_m)$. Based on this projection, all points have the same Y components. After the two projections, the segment becomes a straight line segment. Furthermore, the points between the two ends will be removed, this is shown in the right side of Fig. 4.12B. [68]

Fig 4.11A represents an external wall being drawn. The motion sketch begins with the maker at the tip of the Marker-pen moving in the X - Y plane and then a vertical line is made for extrusion. The blue line shows the original Marker-Pen trajectory whilst the dark line is the result of the clean-up process.

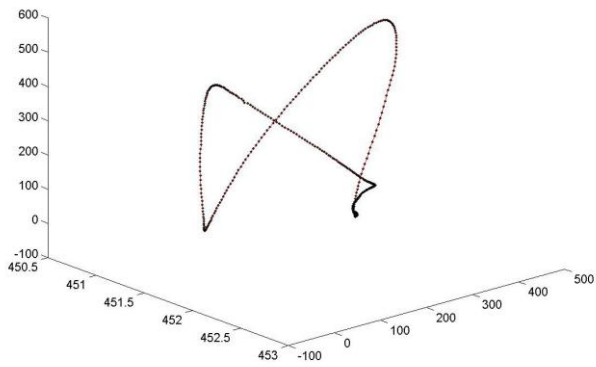
As mentioned previously, as well as motion sketches that geometrically define the basic building units of our construction, the Marker-Pen is also used to communicate simple editing gestures, such as the delete (identified with the letter x drawn in 3D space) and or a finish symbol that represents the end of a motion sketch (in this case drawn as the letter O). Fig 4.12C and Fig 4.12D show these two editing gestures respectively. [68]



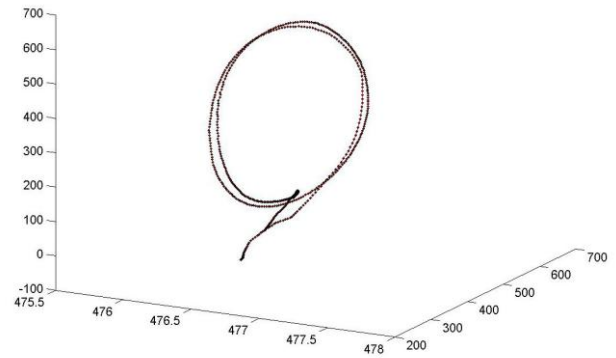
(A)



(B)



(C)



(D)

Figure 4.12 A-B: Motion sketch tidy-up and editing gestures [68] – (Fig.10)

4.5 3D Curve and Surface Generation and Preliminary Results

All of our motion sketch processing and hand sign recognition was completed in a prototype system produced using the numerical computing environment MATLAB. Once individual hand gesture recognition has been completed and its accompanying design sketch has been processed, 3D geometric modelling data is generated. This data is then channelled to a 3D CAD modelling software, in this case we chose Alias Studio for its useful OBJ interface mechanism allowing easy rendering and editing functions.

Our initial testing sessions concentrated on the rapid 3D modelling of a train station using our defined set of architectural hand gestures. This saw our volunteer design student draw the outside walls of the building using a large piece of sketch paper on the desk as drawing aid. With this complete, the remaining building structure units were added. Progressively the model train station was complete with the following components: ground, external wall, 1 door, 2 windows, 4 columns, a secondary level, and a freeform curved roof.

Each design student had 3 attempts to get used to the equipment and different drawing sequences before their test commenced. Each designer spent less than approximately 10 minutes to complete the given task of modeling a train station. Each individuals captured data was stored in Trc. And Trb files which were accessed via Microsoft Excel in tabulated form. This can be seen in Fig. 4.13. Below each marker heading is a list of the marker's position in XYZ coordinates for every frame, in this case 120 frames per second. With this data, a clean-up process was applied and a 3D surface model was produced. Fig 4.14 represents the 3D wireframe model created in Alias Studio and a rendered equivalent is shown in Fig 4.15.

Microsoft Excel - BuildingA1.trc

File Edit View Insert Format Tools Data Window Help

Type a question for help

100%

BuildingA1

	A	B	C	D	E	F	G	H	I	J	K
1	PathFileTy	4 (X/Y/Z)	C:\Documents and Settings\NAB2004\Desktop\HandMotion 140208\BuildingA1.trc								
2	DataRate	CameraRa	NumFrame	NumMarke	Units	OrigDataR	OrigDataS	OrigNumFrames			
3	120	120	15336	11 mm		120	1	15336			
4	Frame#	Time	F1	Y1	Z1	F2	Y2	Z2	F3	Y3	Z3
5			X1	Y1	Z1	X2	Y2	Z2	X3	Y3	Z3
6											
7											
8	1	0	323.3585	307.9332	463.6477	340.049	194.9404	471.4968	278.1729	157.6393	464.4401
9	2	0.008	323.3637	307.9304	463.6525	340.0525	194.9518	471.4851	278.1732	157.5937	464.4778
10	3	0.017	323.367	307.9289	463.6557	340.0551	194.9605	471.4845	278.1738	157.5653	464.504
11	4	0.025	323.3679	307.9284	463.6568	340.0574	194.9653	471.4843	278.1742	157.5569	464.5182
12	5	0.033	323.3673	307.9271	463.6563	340.0597	194.9666	471.4847	278.1717	157.5521	464.534
13	6	0.042	323.3666	307.9249	463.6552	340.0612	194.9657	471.4846	278.1633	157.5303	464.5667
14	7	0.05	323.3673	307.9232	463.6539	340.0604	194.9637	471.4834	278.1503	157.4867	464.616
15	8	0.058	323.3692	307.9238	463.6525	340.0573	194.9617	471.4814	278.1392	157.4393	464.6612
16	9	0.067	323.3712	307.9273	463.6511	340.0531	194.9606	471.4793	278.1365	157.4169	464.6756
17	10	0.075	323.3713	307.9333	463.6504	340.05	194.9614	471.4778	278.1446	157.436	464.6479
18	11	0.083	323.3692	307.9399	463.6501	340.0492	194.964	471.4769	278.1593	157.4874	464.5925
19	12	0.092	323.3684	307.9445	463.6498	340.0506	194.9668	471.4769	278.1735	157.5451	464.5373
20	13	0.1	323.365	307.9462	463.6493	340.0529	194.9682	471.4778	278.1822	157.5855	464.503
21	14	0.108	323.3658	307.9463	463.6487	340.0548	194.9681	471.4779	278.1848	157.599	464.4942
22	15	0.117	323.368	307.9461	463.6476	340.0554	194.9674	471.4801	278.183	157.591	464.5039
23	16	0.125	323.3702	307.9462	463.6453	340.055	194.967	471.4811	278.1792	157.5755	464.5203
24	17	0.133	323.3713	307.9462	463.643	340.0541	194.9681	471.4826	278.1731	157.561	464.5395

Ready

Sum=328306581.7

NUM

Fig. 4.13: Trc. File of captured motion[68] – (Fig.11)

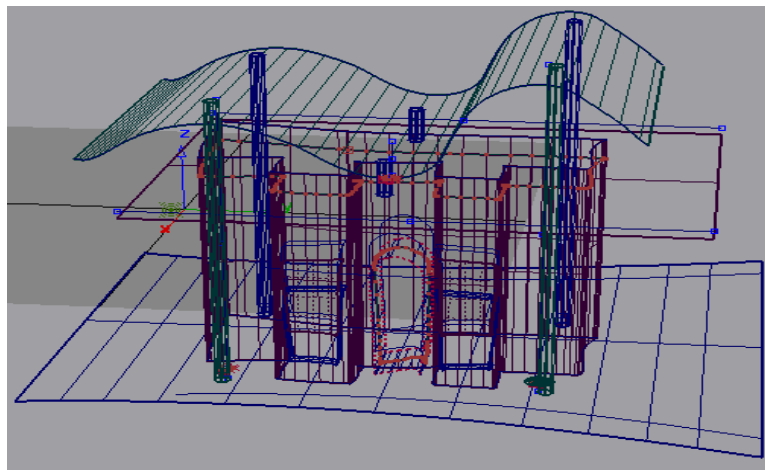


Fig. 4.14: Wireframe model of train station in Visual Studio[68] – (Fig12.a)

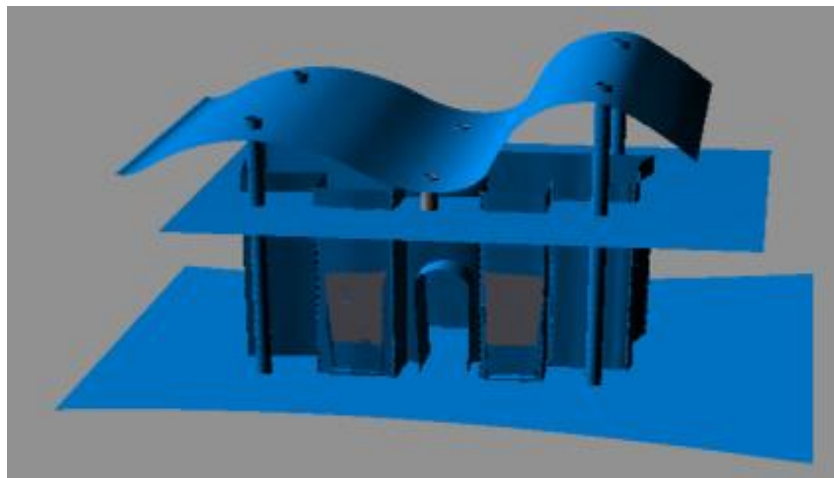
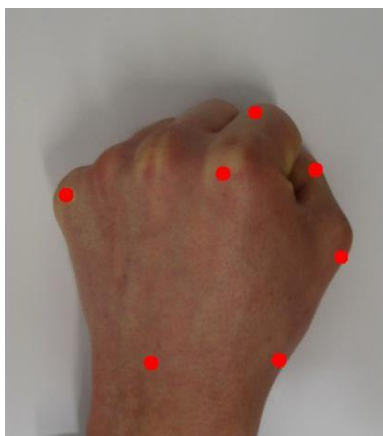


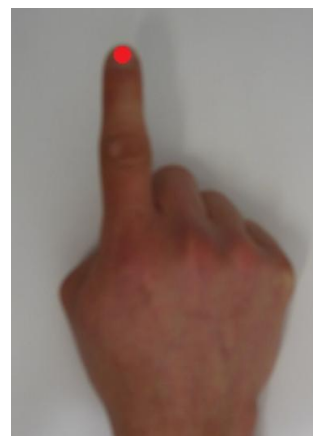
Fig. 4.15: Rendered model of train station in Visual Studio[68] – (Fig 12.b)

5 Feasibility Study 2 – Interactive Real-time Communication Application

In the attempt to keep things simple and effective with the transition from architectural hand signs to product design hand signs, we felt it suitable to maintain a similar approach. We used the same amount of markers on the left hand with the same configuration as before, in order to capture our new proposed hand gestures. To make the experimental interaction between the user and computer even more natural than with our preliminary research, we decided to replace the three markers on the marker pen in the right hand, with one marker on the index finger, see Fig. 5.1A – B. This will hopefully have the effect of leaving the user free of any input device, which can cause unnatural, restrictive behavior.



A



B

Fig. 5.1 A – B: Marker placement for product design capture.

As with our previous study, only one marker on the right hand is needed to capture the motion description. In certain cases when new designers would begin the motion capture session, the system would not successfully create a robust template to give successful occlusion free results. In these cases, two more markers were added to the right hand similarly to the marker-pen, with the intent to make the

system recognise and keep track of the main marker at the tip of the index finger. This configuration can be seen in Fig. 5.2.

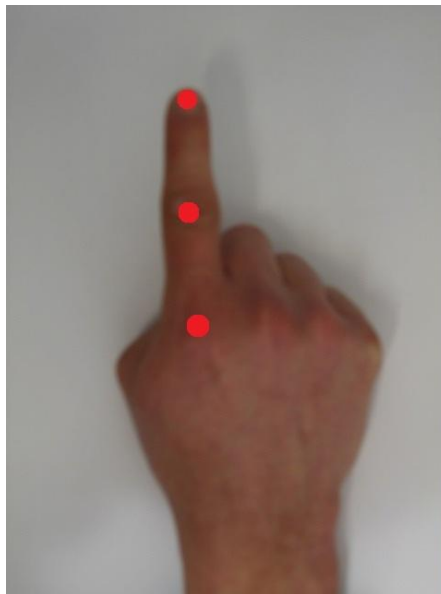


Fig. 5.2: Right hand marker set used if marker switching is present.

In the hope of creating a versatile real-time system that will let the user model and construct a product of their choice, we set about designing a set of hand gestures which can define a large range of products. In order to do this, instead of selecting a product and segmenting it into categorised components (as with architectural modelling) we took inspiration from common modelling commands found in CAD packages. This in turn should improve the flexibility of our system allowing the user to have full control over what they choose to model, offering no restrictions. Several hand sign gestures were designed and logged for use in the gesture database, which describe several 3D CAD modelling commands, such as primitive shapes, surfaces, extrude etc. See Appendix A, for a complete list of left hand gestures for product design and their corresponding functions. For ease of use and orderliness each hand sign is categorised into sections according to their function. Please see the flow diagram in Fig 5.3 representing the basic breakdown of this grouping by displaying a few per category.

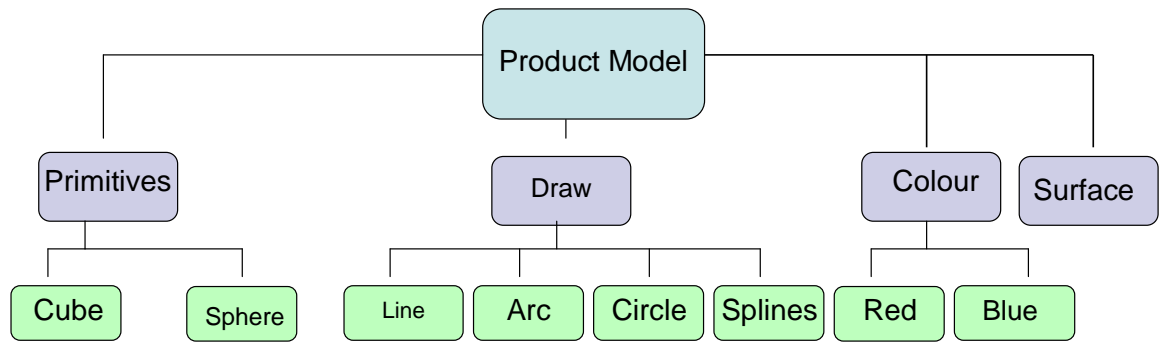


Fig. 5.3: Flow diagram showing basic Hand gesture categorisation for product design models.

5.1 Design of Software Application

Before work could begin on the development of a software program that would act as an interface and control system between the Motion Analysis Hardware/Software and the user, a wide range of programming languages and Application Programming Interfaces (API's) were researched and considered. In order to create a program that could be incorporated well with the software Motion Analysis provide with their system (EVaRT), we studied the Software Development Kit (SDK) provided by the company in order to ascertain which was the best programming language to use. With the SDK written in C++, and it being one of the most advanced, multipurpose and most popular programming languages in the world, we decided that all core programming would be completed in this language.

Furthermore, our proposed software would run only on a windows platform, we therefore decided to use the Microsoft Foundation Classes (MFC). The Microsoft Foundation Class Library is based around parts of the windows API. The classes in the MFC Library are written as object orientated in the C++ programming language. The MFC Library helps the programmer a vast amount of time by providing predefined code that has already been written. It also provides an overall framework for developing the application program.

There are MFC Library classes for all graphical user interface elements within a common windows application (windows, menus, frames, tool bars, status bars, etc). MFC helps with building interfaces, for handling events such as messages from other applications, for handling keyboard and mouse input, and for creating ActiveX controls.

As our software will act as a visual aid between the user and the motion capture system, it must be capable of displaying 3D graphics. In order for this to occur we must utilise a graphics API to display 2D and 3D computer graphics. After some research we realised there were only two mainstream graphics API's which would

be suitable for our purpose, these being OpenGL or DirectX. After much consideration we decided that OpenGL was the best API, as it ties well with the use of the Microsoft Foundation Classes. The basic operation of OpenGL is to accept primitives such as points, lines and polygons, and convert them into pixels. OpenGL is widely used in the gaming industry as well as CAD, virtual reality, scientific visualization, information visualization, and flight simulation.

To get a greater understanding of the program we intend to create and where it fits into the complete system and all software and hardware used please see Fig. 6.1 which represents a flow diagram for the software integration.

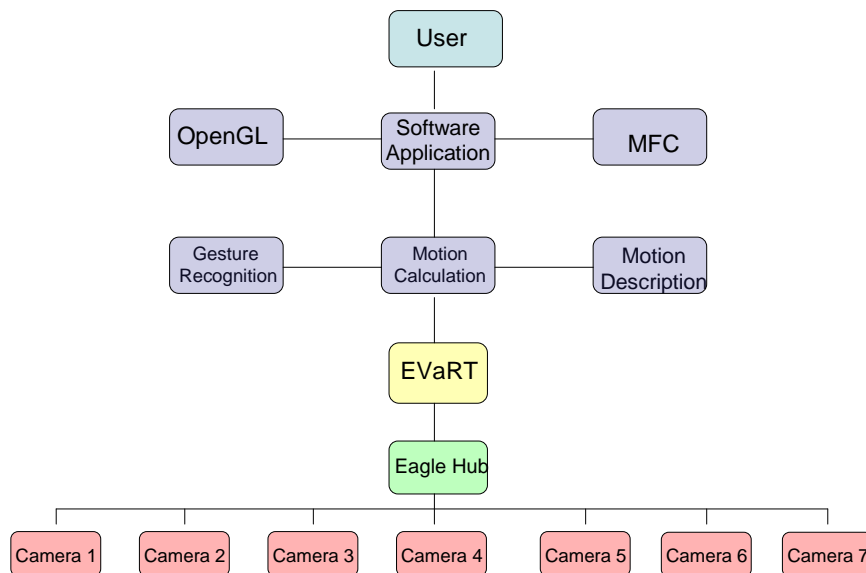


Fig. 6.1: Software integration

Before programming began we designed the layout of the interface window and how the main graphics would be displayed to the user. We developed an interface similar to that of most popular 3D CAD packages, as it allows the user to view the 3D scene from different perspectives via different viewports. We chose to split the main window into four viewports, each representing a different axis. Viewport

A represents the X axis; viewport B represents the Y axis, viewport C represents the Z axis, and viewport D represents the scene from a 3D perspective.

Fig. 6.2 shows the four viewports described above. For testing and display purposes, a 3D cube with different colour faces is shown in the centre of the 3D scene.

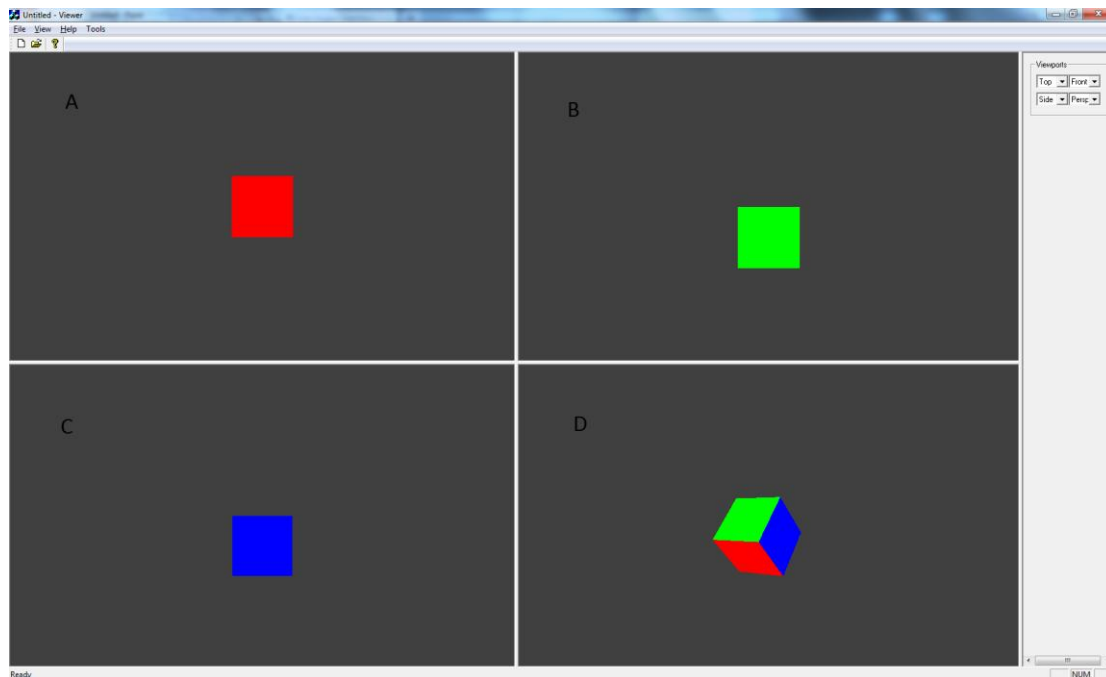
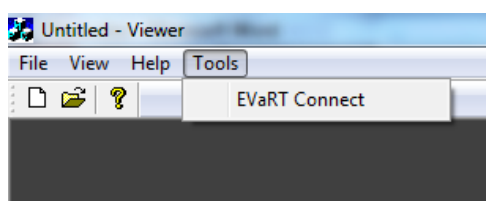


Fig. 6.2: Print screen of viewport layout and axis specification.

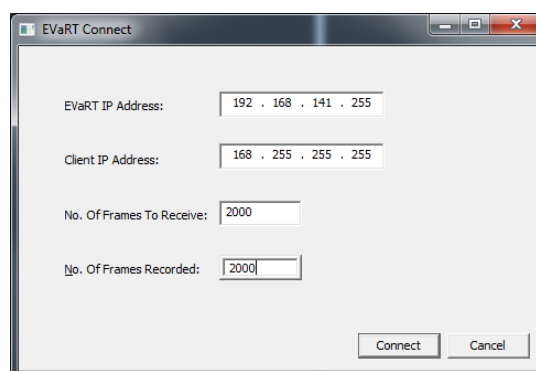
The main window of our software can be minimised and moved to anywhere on the screen. This feature was implemented in order to use our software on a dual screen setup, making configuration between the EVaRT system and our software easier. Each viewport can also be resized and defined according to the user's preference. This is controlled by the vertical drop down menu toolbar to the right of the screenshot displayed in Fig. 6.2. We designed all aspects of our software so the user and or assistant has full control over it's appearance in the hope of getting better results from the designer whilst at the modelling stage.

Before the software is able to receive information from the Motion Analysis equipment the user must make connection to the EVaRT system by accessing the

“EVaRT Connect” dialogue box. To do this they must access the “Tools” toolbar located on the top bar. Once selected the user is faced with a secondary window which asks for some information specific to making a connection. An EVaRT IP address and Host IP address is required. As our software can be run from a separate computer to that of where EVaRT is running, all information is past through the Ethernet connection. The host IP which is required is of the computer that is running our software and the EVaRT IP is of the computer where the EVaRT system is running. Further information about frame capture is required, the number of frames to receive from the system and the number of frames to record, both of which can be defined by the user dependant on their proposed capture session. Please see Fig. 6.3A and Fig. 6.3B. for the breakdown of this menu system and dialog box respectively.



(A)



B)

Fig. 6.3A – B: Screenshot of menu system and Evart Connect Dialog box respectively.

As with the main window of our program, the EVaRT Connect box was programmed to be modeless. A modeless window allows the user to continue working with the main application while the modeless window stays open or minimised. This is well suited to our application, as the user must make connection using this box, but also needs to continue working with the window behind, whilst modelling. Please see Appendix B for programming code of software development.

A more detailed flow diagram showing the internal working of our application, and how each file contributes to a process function is show below in Fig. 6.4.

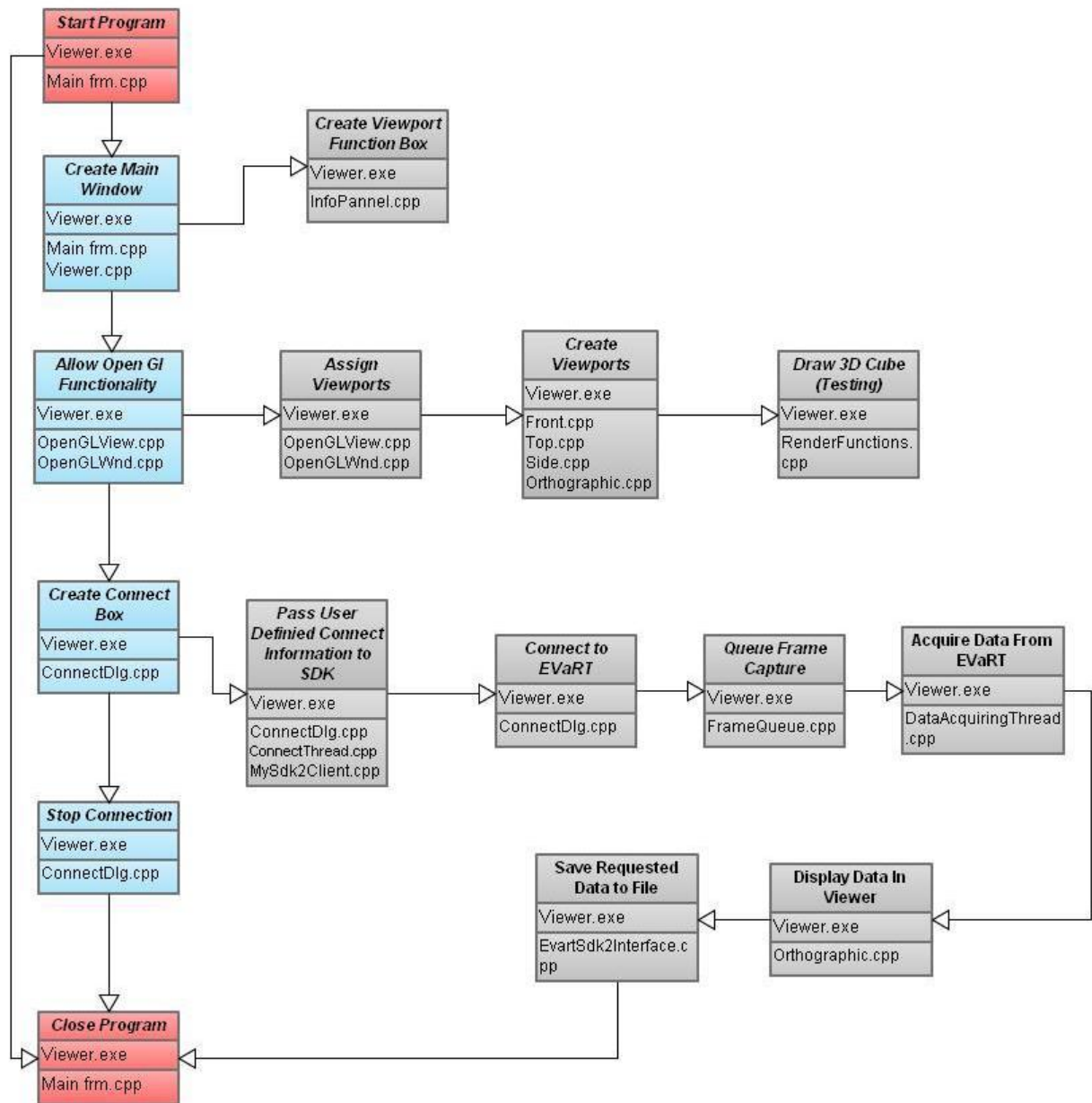


Fig. 6.4: Flow Diagram of internal functionality of Software Application

6 Testing and Results

Due to the limited time constraints placed on this research proposal, I was unable to complete the programming of our software application in the given time. With limited previous knowledge of programming, I experienced large amounts of program compile errors, and other programming related issues which would take longer to solve than initially expected. Just to learn the basics of C++ in order to understand the Microsoft Foundation Classes and the OpenGL API, took several months.

As time progressed I decided to spend most of my time concentrating on getting the software application to connect with the EVaRT system, in order to successfully prove that real-time communication was possible. With this in mind, I displayed an OpenGL scene in the top left hand viewport of the application which was able to display a user defined amount of markers that were present in the motion capture data. The markers trajectory is also shown in continuous form. Due to the limited amount of time I was unable to program an interface toolbar or toolbox that would allow the user to control and define the settings for which markers to display. This must all be completed within the Integrated Development Environment (IDE) and then compiled and run afterwards. It is here where the user must also define the client and host IP addresses and also the number of frames to display.

I tested this successfully with a demo, both with the motion capture system capturing motion in real time and also EVaRT playing back previously recorded data emulated in real time. Please see Fig. 7.1 – 7.3 shown on pages 80 and 81, for screen shots of this procedure in action, 2 markers on the left hand have been selected to show in our OpenGL viewport.

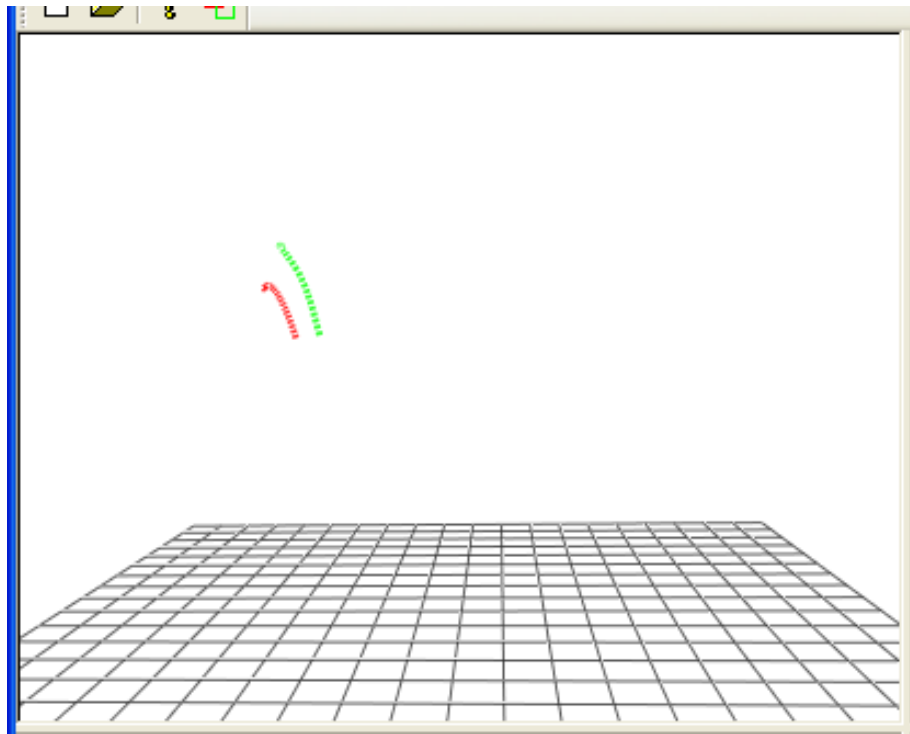


Fig. 7.1: Screenshot of OpenGL Viewport displaying marker 1 and 5 on the left hand.

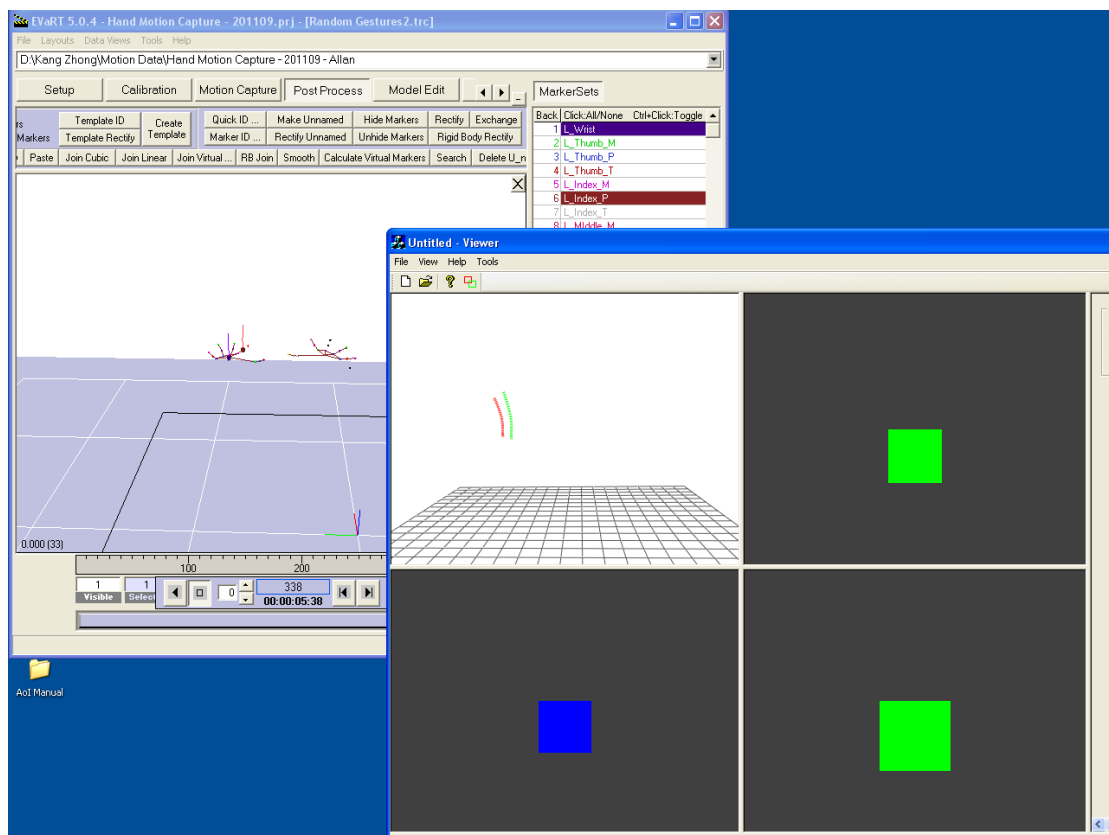


Fig. 7.2: Screenshot of EVaRT system and our application running together.

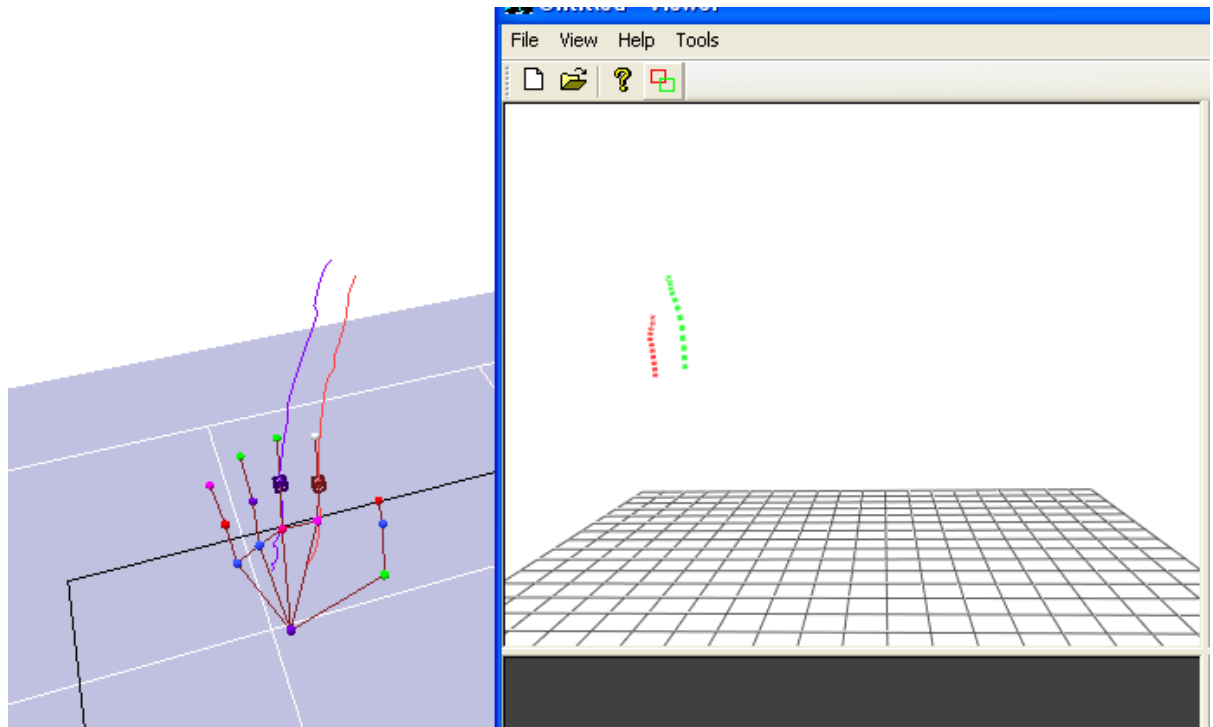


Fig. 7.3: Screenshot of EVaRT system and our application running together with two markers selected on the left hand.

7 Conclusion and Suggestions for Further Work

7.1 Summary

A comprehensive literature survey was conducted with regards to the analysis and synthesis of Human Motion, Motion Description, Human Computer Interaction and Hand Gesture Recognition. This led to the focus of generating 3D product design models in real-time using hand motion and gesture. A motion capture system was used to capture hand gestures and motion descriptions in an initial study to generate rapid architectural structures in an offline state. The data was post-processed in the form of a hierarchical skeleton structure, after which data extraction was implemented in order to recognise hand gestures. Once complete this information was transferred to a common 3D software application where a wireframe model and meshed equivalent of a train station was realised. A total of 8 design volunteers were used in this study. This investigation focussed on determining whether the motion capture system used would be suitable for our planned real-time application and whether hand motion and gesture recognition could be successfully realised. With successful results from our initial study, this led to development of an interactive system that would allow the user to model 3D product design models in real-time.

A new set of hand signs were developed for the real-time application of product design models. An OpenGL capable software application was programmed using C++ and the Microsoft Foundation Classes. Due to limited time, I was unable to complete the program, however I was able to successfully display 3D graphics from the motion capture system running in real-time. This proves that with further time dedicated to programming, real-time modelling of 3D product design concepts can be realised.

Further work could be carried out by, completing the software application and investigating further methods to which the system could be used with. A larger database of hand signs can be collated in order to make the system more versatile and easily transferable between different disciplines and areas of research.

7.2 Final Thought and Discussion

The following section will discuss and reflect on the research covered in this thesis, creating a subjective overview on the achievements made and continuation of possible future research.

With limited time available the main aim of this research was completed by implementing two key feasibility studies which covered two main features of the proposed system framework. In hindsight it would have been more beneficial to the overall achievement of this project, to have spent more time on feasibility study two and less on feasibility study one. This could have possibly lead to the realisation of using a complete set of hand gestures and motion to create some type of 3D geometry in real-time. A simple 3D primitive shape would have been sufficient to show possible modelling commands for completing full conceptual product design models. This would have ideally been better than what was achieved in feasibility study two, which was OpenGL graphics of a motion trajectory.

Motion systems are becoming more popular across different industries and on different platforms, with the latest release of a motion system known as Kinect by Microsoft. This computer peripheral is designed for the gaming industry and is intended to be used by users for free interaction with a computer device in order to control onscreen gaming commands. It is clear to see that most development in this area is being focused on hands free interaction in a home or work environment. With this in mind if further work was to continue on this research, as recommended earlier, it would be crucial to complete the real-time communication program in order to realise a complete 3D product design model. After this has been achieved it would be beneficial to focus on replacing the motion capture system used in this research with a more permanent hands free alternative that would be much more suited for use in homes or offices. This will allow for a hassle free practical approach for groups of professionals to share design ideas by creating 3D design models in real-time using 3D space.

8 References

- [1] E. George and S. Karan, "Handrix: animating the human hand," presented at the Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, San Diego, California, 2003.
- [2] K. Perlin, "Real Time Responsive Animation with Personality," *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, vol. vol.1, MARCH 1995 1995.
- [3] K. Perlin and A. Goldberg, "Improv: a system for scripting interactive actors in virtual worlds," presented at the Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, 1996.
- [4] K. H. Jessica, *et al.*, "Animating human athletics," presented at the Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, 1995.
- [5] W. L. Wooten and J. K. Hodgins, "Animation of Human Diving," *Computer Graphics Forum*, vol. 15, pp. 3-13, 1996.
- [6] W. L. Wooten, *et al.*, "Simulating leaping, tumbling, landing and balancing humans
Simulating leaping, tumbling, landing and balancing humans," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, 2000, pp. 656-662 vol.1.
- [7] K. H. Jessica and S. P. Nancy, "Adapting simulated behaviors for new characters," presented at the Proceedings of the 24th annual conference on Computer graphics and interactive techniques, 1997.
- [8] F. Petros, *et al.*, "The virtual stuntman: dynamic characters with a repertoire of autonomous motor skills " *Computers & Graphics* vol. 25, 2001.
- [9] L. Joseph, *et al.*, "Limit cycle control and its application to the animation of balancing and walking," presented at the Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, 1996.
- [10] F. Petros, *et al.*, "Composable controllers for physics-based character animation," presented at the Proceedings of the 28th annual conference on Computer graphics and interactive techniques, 2001.
- [11] T. Komura, *et al.*, "A Muscle-based Feed-forward Controller of the Human Body," vol. 16, ed, 1997, pp. C165-C176.
- [12] T. Komura, *et al.*, "Creating and retargetting motion by the musculoskeletal human body model," *The Visual Computer*, vol. 16, pp. 254-270, 2000.
- [13] P. Zoran and W. Andrew, "Physically based motion transformation," presented at the Proceedings of the 26th annual conference on Computer graphics and interactive techniques, 1999.
- [14] S. Alla, *et al.*, "Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces," *ACM Trans. Graph.*, vol. 23, pp. 514-521, 2004.
- [15] C. K. Liu and P. Zoran, "Synthesis of complex dynamic character motion from simple animations," presented at the Proceedings of the 29th annual conference on Computer graphics and interactive techniques, San Antonio, Texas, 2002.
- [16] S. Tak, *et al.*, "Motion Balance Filtering," vol. 19, ed, 2000, pp. 437-446.
- [17] C. F. Anthony and S. P. Nancy, "Efficient synthesis of physically valid human motion," *ACM Trans. Graph.*, vol. 22, pp. 417-426, 2003.

- [18] H. Ko and N. I. Bidler, "Animating human locomotion with inverse dynamics," *IEEE Computer Graphics and Application*, vol. 16, 1996.
- [19] M. Oshita and A. Makinouchi, "A Dynamic Motion Control Technique for Human-like Articulated Figures," *Computer Graphics Forum*, vol. 20, pp. 192-203, 2001.
- [20] Z. Victor Brian and K. H. Jessica, "Motion capture-driven simulations that hit and react," presented at the Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, San Antonio, Texas, 2002.
- [21] W. Andrew and K. Michael, "Spacetime constraints," presented at the Proceedings of the 15th annual conference on Computer graphics and interactive techniques, 1988.
- [22] L. Zicheng, *et al.*, "Hierarchical spacetime control," presented at the Proceedings of the 21st annual conference on Computer graphics and interactive techniques, 1994.
- [23] R. Charles, *et al.*, "Efficient generation of motion transitions using spacetime constraints," presented at the Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, 1996.
- [24] G. Michael, "Retargetting motion to new characters," presented at the Proceedings of the 25th annual conference on Computer graphics and interactive techniques, 1998.
- [25] L. Jehee and S. Sung Yong, "A hierarchical approach to interactive motion editing for human-like figures," presented at the Proceedings of the 26th annual conference on Computer graphics and interactive techniques, 1999.
- [26] K. Yamane, *et al.*, "Synergetic CG choreography through constraining and deconstraining at will
Synergetic CG choreography through constraining and deconstraining at will," in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, 2002, pp. 855-862 vol.1.
- [27] K. Yamane, *et al.*, "Natural motion animation through constraining and deconstraining at will
Natural motion animation through constraining and deconstraining at will," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 9, pp. 352-360, 2003.
- [28] B. Armin and W. Lance, "Motion signal processing," presented at the Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, 1995.
- [29] W. Andrew and P. Zoran, "Motion warping," presented at the Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, 1995.
- [30] U. Munetoshi, *et al.*, "Fourier principles for emotion-based human figure animation," presented at the Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, 1995.
- [31] A. kenji, *et al.*, "Emotion from motion," *Graphics Interface*, pp. 222-229, 1996.
- [32] C. Diane, *et al.*, "The EMOTE model for effort and shape," presented at the Proceedings of the 27th annual conference on Computer graphics and interactive techniques, 2000.

- [33] N. Michael and F. Eugene, "Aesthetic edits for character animation," presented at the Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, San Diego, California, 2003.
- [34] T. Seyoon, *et al.*, "Spacetime sweeping: an interactive dynamic constraints solver
Spacetime sweeping: an interactive dynamic constraints solver," in *Computer Animation, 2002. Proceedings of*, 2002, pp. 261-270.
- [35] K. Lucas and G. Michael, "Flexible automatic motion blending with registration curves," presented at the Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, San Diego, California, 2003.
- [36] B. Matthew and H. Aaron, "Style machines," presented at the Proceedings of the 27th annual conference on Computer graphics and interactive techniques, 2000.
- [37] G. Keith, *et al.*, "Style-based inverse kinematics," vol. 23, ed: ACM, 2004, pp. 522-531.
- [38] K. Lucas, *et al.*, "Motion graphs," presented at the Proceedings of the 29th annual conference on Computer graphics and interactive techniques, San Antonio, Texas, 2002.
- [39] W. Piekarski and B. H. Thomas, "The Tinmith system: demonstrating new techniques for mobile augmented reality modelling," presented at the Proceedings of the Third Australasian conference on User interfaces - Volume 7, Melbourne, Victoria, Australia, 2002.
- [40] E. Foxlin and M. Harrington, "WearTrack: A Self-Referenced Head and Hand Tracker for Wearable Computers and Portable VR," presented at the Proceedings of the 4th IEEE International Symposium on Wearable Computers, 2000.
- [41] J. Rekimoto, "GestureWrist and GesturePad: Unobtrusive Wearable Interaction Devices," presented at the Proceedings of the 5th IEEE International Symposium on Wearable Computers, 2001.
- [42] M. Gandy, *et al.*, "The Gesture Pendant: A Self-illuminating, Wearable, Infrared Computer Vision System for Home Automation Control and Medical Monitoring," presented at the Proceedings of the 4th IEEE International Symposium on Wearable Computers, 2000.
- [43] N. Ukita, *et al.*, "Wearable vision interfaces: towards wearable information playing in daily life," *In 1st CREST Workshop on Advanced Computing and Communicating Techniques for Wearable Information Playing*, pp. 47--56, 2002.
- [44] T. Starnier, *et al.*, "Real-Time American Sign Language Recognition Using Desk and Wearable Computer Based Video," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, pp. 1371-1375, 1998.
- [45] X. Zhu, *et al.*, "Segmenting Hands of Arbitrary Color," presented at the Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition 2000, 2000.
- [46] S. M. Dominguez, *et al.*, "Robust finger tracking for wearable computer interfacing," presented at the Proceedings of the 2001 workshop on Perceptive user interfaces, Orlando, Florida, 2001.
- [47] K. Oka, *et al.*, "Real-Time Tracking of Multiple Fingertips and Gesture Recognition for Augmented Desk Interface Systems," presented at the

- Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition, 2002.
- [48] K. Y. Wong and M. E. Spetsakis, "Motion Segmentation and Tracking," 2002.
 - [49] M. Isard and A. Blake, "Contour tracking by stochastic propagation of conditional density," ed, 1996, pp. 343-356.
 - [50] R. O'Hagan and A. Zelinsky, "Finger track — A robust and real-time gesture interface," ed, 1997, pp. 475-484.
 - [51] J. L. Crowley, *et al.*, "Finger Tracking as an Input Device for Augmented Reality," pp. 195 - 200, 1995.
 - [52] J. M. Rehg and T. Kanade, "DigitEyes: vision-based hand tracking for human-computer interaction," *Proceedings of 1994 IEEE Workshop on Motion of Nonrigid and Articulated Objects*, pp. 16-22, 1994.
 - [53] N. Gupta, *et al.*, "CONDENSATION-based Predictive EigenTracking," *Indian Conference on Computer Vision, Graphics and Image Processing*, 2002.
 - [54] N. Gupta, *et al.*, "Developing a gesture-based interface," *IETE Journal of Research: Special Issue on Visual media Processing*, vol. 48, pp. 237-244, 2002.
 - [55] P. Perez, *et al.*, "Color-Based Probabilistic Tracking," presented at the Proceedings of the 7th European Conference on Computer Vision-Part I, 2002.
 - [56] J. P. Mammen, *et al.*, "Simultaneous tracking of both hands by estimation of erroneous observations " *British Machine Vision Conference (BMVC)*, 2004.
 - [57] J. MacCormick and M. Isard, "Partitioned Sampling, Articulated Objects, and Interface-Quality Hand Tracking," presented at the Proceedings of the 6th European Conference on Computer Vision-Part II, 2000.
 - [58] J. MacCormick, *Stochastic Algorithms for Visual Tracking: Probabilistic Modelling and Stochastic Algorithms for Visual Localisation and Tracking*: Springer-Verlag New York, Inc., 2002.
 - [59] G. Rigoll, *et al.*, "High performance real-time gesture recognition using Hidden Markov Models," ed, 1998, pp. 69-80.
 - [60] H.-K. Lee and J. H. Kim, "An HMM-Based Threshold Model Approach for Gesture Recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, pp. 961-973, 1999.
 - [61] H. Birk, *et al.*, "Real-time recognition of hand alphabet gestures using principal component analysis," *10th Scandinavian Conference on Image Analysis*, 1997.
 - [62] H. Fillbrandt, *et al.*, "Extraction of 3D Hand Shape and Posture from Image Sequences for Sign Language Recognition," presented at the Proceedings of the IEEE International Workshop on Analysis and Modeling of Faces and Gestures, 2003.
 - [63] M. Bray, *et al.*, "3D Hand Tracking by Rapid Stochastic Gradient Descent Using a Skinning Model," *First European Conference on Visual Media Production (CVMP)*, 2004.
 - [64] B. Stenger, *et al.*, "Hand Pose Estimation Using Hierarchical Detection," ed, 2004, pp. 105-116.
 - [65] H. Zhou and T. S. Huang, "Tracking Articulated Hand Motion with Eigen Dynamics Analysis," presented at the Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2, 2003.
 - [66] R. Y. Wang, *et al.*, "Real-time hand-tracking with a color glove," *ACM Trans. Graph.*, vol. 28, pp. 1-8, 2009.

- [67] M. Masry, *et al.*, "A freehand sketching interface for progressive construction of 3D objects," presented at the ACM SIGGRAPH 2007 courses, San Diego, California, 2007.
- [68] X. Yi, *et al.*, "Generating 3D architectural models based on hand motion and gesture," *Comput. Ind.*, vol. 60, pp. 677-685, 2009.

Appendix A:

Hand gesture signs for product design motion capture



Surface



Cube



Spline



Sphere



Circle



Line



Red (colour)



Arc

Appendix B:

Programming code for software development

C3DModelView.h

```
#pragma once
#include "OpenGLView.h"
#include "DataAcquiringThread.h"

class C3DModelView : public COpenGLView
{
protected:
    C3DModelView(void); // protected constructor used by dynamic creation
    virtual ~C3DModelView(void);

    DECLARE_DYNCREATE(C3DModelView)

// Attributes
public:
    float m_zoom,
           m_xpos,
           m_ypos,
           m_xrot,
           m_yrot;
    int    m_lastMouseX,
           m_lastMouseY;

    fMarkerCoordinate markerCoordinates1[50];
    fMarkerCoordinate markerCoordinates2[50];
    bool bStart;

// Overrides
protected:

    //      Main OpenGL functions.
    virtual void DoOpenGLDraw();
    virtual void DoOpenGLResize(int nWidth, int nHeight);

    // Generated message map functions
protected:
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    afx_msg LRESULT OnUpdateCoordinates(WPARAM wParam, LPARAM lParam);

    DECLARE_MESSAGE_MAP()
};
```

ConnectDlg.h

```
#pragma once
#include "afxwin.h"
#include "Viewer.h"

// CConnectDlg dialog
class CConnectDlg : public CDialog
{
    DECLARE_DYNAMIC(CConnectDlg)

public:
    CConnectDlg(CWnd* pParent = NULL); // standard constructor
    virtual ~CConnectDlg();

// Dialog Data
    enum { IDD = IDD_CONNECT };

protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support

    DECLARE_MESSAGE_MAP()
public:

    CString m_NOFTR;
    CString m_NOFR;
```

```

        DWORD m_Eipaddress;
        DWORD m_Cipaddress;
        afx_msg void OnBnClickedOk();
        CStatic m_Status;
};

```

ConnectThread.h

```

#pragma once

#include "DataAcquiringThread.h"
#include "FrameQueue.h"

// CConnectThread

class CConnectThread : public CWinThread
{
    DECLARE_DYNCREATE(CConnectThread)

protected:
    CConnectThread();           // protected constructor used by dynamic creation
    virtual ~CConnectThread();

public:
    void SetConnectParams(sConnectParams* params);
    void SetFrameQueue(CFrameQueue* pFrameQueue);
    void SetDataAcquiringThread(CDataAcquiringThread* pDataAcquiringThread);

    virtual BOOL InitInstance();
    virtual int ExitInstance();
    virtual int Run();

protected:
    sConnectParams* m_pConnectParams;
    CFrameQueue* m_pFrameQueue;
    CDataAcquiringThread* m_pDataAcquiringThread;
    DECLARE_MESSAGE_MAP()
};

```

DataAcquiringThread.h

```

#pragma once

#include "FrameQueue.h"
#include "MainFrm.h"

typedef struct sConnectParams
{
    CString sServerIP;
    CString sClientIP;
    int iMaxFrames;
} sConnectParams;

typedef struct fMarkerCoordinate
{
    float xPos;
    float yPos;
    float zPos;
} fMarkerCoordinate;

// CDataAcquiringThread

class CDataAcquiringThread : public CWinThread
{
    DECLARE_DYNCREATE(CDataAcquiringThread)

protected:
    CDataAcquiringThread();           // protected constructor used by dynamic
creation
    virtual ~CDataAcquiringThread();

public:
    virtual BOOL InitInstance();
    virtual int ExitInstance();
    virtual int Run();

```

```

void SetConnectParamsReady();
void SetFrameHasArrivedEvent();
void SetDisconnectedEvent();
void SetMainFrame(CMainFrame* pFrame);

//=====
// for testing
//=====
void SetServerIP(CString sServerIP);
void SetClientIP(CString sClientIP);
void SetMaxFrames(int iMaxFrames);
//=====

protected:
    sConnectParams m_sConnParams;
    bool m_bConnectParamsReady;
    CFrameQueue m_cFrameQueue;
    CMainFrame* m_pFrame;

    HANDLE m_hFrameHasArrivedEvent;
    HANDLE m_hDisconnectedEvent;
    DECLARE_MESSAGE_MAP()
};

```

FrameQueue.h

```

#pragma once

#include "afxmt.h"
#include "EvaRT2.h"

class CDataAcquiringThread;

// CFrameQueue command target

class CFrameQueue : public CObject
{
public:
    CFrameQueue();
    virtual ~CFrameQueue();

    sFrameOfData* GetFrame();
    void AddFrame(sFrameOfData* ptrFrame);
    void SetDataAcquiringThread(CDataAcquiringThread* pDataAcquiringThread);

private:
    CPtrList m_FrameList;
    CCriticalSection m_csForFrameQueue;
    CDataAcquiringThread* m_pDataAcquiringThread;
};

```

MainFrm.h

```

// MainFrm.h : interface of the CMainFrame class
//
//=====

#ifndef __AFX_MAINFRM_H__B9860475_351C_4E83_92B5_E4437E60684C__INCLUDED_
#define __AFX_MAINFRM_H__B9860475_351C_4E83_92B5_E4437E60684C__INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define INFOBAR_SIZE 150 // constant for creating
                        // splitter windows

#include "Perspective.h"
#include "Front.h"
#include "Top.h"
#include "Side.h"
#include "InfoPannel.h"
#include "ConnectDlg.h"

class CMainFrame : public CFrameWnd

```

```

{

protected: // create from serialization only
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:
    BOOL            m_initSplitters;                // Have the splitters // been
initialized?
    CSplitterWnd m_mainSplitter,                    // Splitter windows
        m_viewportSplitter;

    HWND m_hwndTopLeft;
    HWND m_hwndTopRight;
    HWND m_hwndBottomLeft;
    HWND m_hwndBottomRight;

// Operations
public:
    CConnectDlg *Connect;

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // control bar embedded members
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;

// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnSize(UINT nType, int cx, int cy);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnToolsEvertconnect();
    afx_msg void OnToolsConnect();
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_MAINFRM_H__B9860475_351C_4E83_92B5_E4437E60684C__INCLUDED_)

```

OpenGLView.h

```
#pragma once
```

```

// The base class for all OpenGL-based views.
// Handles all the basic OpenGL init stuff for MFC.

class COpenGLView : public CView
{
protected:
    COpenGLView(void); // protected constructor used by dynamic creation
    ~COpenGLView(void);

```

```

        DECLARE_DYNCREATE(COpenGLView)

protected:
// Overrides
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual void OnDraw(CDC* pDC);

// Debug functions.
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Attributes
    HGLRC m_hRC; //Rendering Context
    CDC* m_pDC;  //Device Context

// Operations
    // OpenGL init stuff.
    BOOL SetupPixelFormat();
    BOOL InitOpenGL();

    // Main OpenGL functions.
    virtual void DoOpenGLDraw() {};
    virtual void DoOpenGLResize(int nWidth, int nHeight) {};

    // Each viewport uses its own context, so we need to make sure the correct
    // context is set whenever we make an OpenGL command.
    void SetContext() { wglMakeCurrent( m_pDC->GetSafeHdc(), m_hRC ); }
    void SwapGLBuffers() { SwapBuffers( m_pDC->GetSafeHdc() ); }

protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnDestroy();
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);

    DECLARE_MESSAGE_MAP()

};

```

OpenGLWnd.h

```

#if !defined(AFX_OPENGLWND_H__83310F64_5D53_44AE_9D08_E29D6A961D6F__INCLUDED_)
#define AFX_OPENGLWND_H__83310F64_5D53_44AE_9D08_E29D6A961D6F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// OpenGLWnd.h : header file
//

#include "RenderFunctions.h"
#include "ViewerDoc.h"

////////////////////////////////////

// COpenGLWnd view

// The base class for all our viewport views.
// Handles all the basic OpenGL init stuff for
// MFC. A similar method could be used for
// another API like Direct 3D.

class COpenGLWnd : public CView
{
protected:
    COpenGLWnd();           // protected constructor used by dynamic creation
    DECLARE_DYNCREATE(COpenGLWnd)

// Attributes
public:
    void (*m_RenderScene) ( CViewerDoc* doc );    // void function pointer to the
    rendering

```

```

        // function. Used to change to easily
        // change what a viewport displays.
protected:
    HGLRC m_hRC; //Rendering Context
    CDC* m_pDC;  //Device Context

// Operations
public:
    // OpenGL init stuff
    BOOL SetupPixelFormat();
    BOOL InitOpenGL();

    // A couple of functions to allow outside
    // forces to manipulate the class.
    void SetRenderFunc( void (*func) ( CViewerDoc* ) ) { m_RenderScene = func; }
    void RenderScene();
    // Each viewport uses its own context
    // so we need to make sure the correct
    // context is set whenever we make an
    // OpenGL command.
    void SetContext() { wglMakeCurrent( m_pDC->GetSafeHdc(), m_hRC ); }
    void SwapGLBuffers() { SwapBuffers( m_pDC->GetSafeHdc() ); }

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(COpenGLWnd)
protected:
    virtual void OnDraw(CDC* pDC);          // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}AFX_VIRTUAL

// Implementation
protected:
    virtual ~COpenGLWnd();

#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

    // Generated message map functions
protected:
    //{AFX_MSG(COpenGLWnd)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnDestroy();
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_OPENGLWND_H__83310F64_5D53_44AE_9D08_E29D6A961D6F__INCLUDED_)

```

Orthographic.h

```

#if !defined(AFX_ORTHOGRAPHIC_H__89B35520_EA7E_4AF2_9E2F_3E6CE6E3347B__INCLUDED_)
#define AFX_ORTHOGRAPHIC_H__89B35520_EA7E_4AF2_9E2F_3E6CE6E3347B__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Orthographic.h : header file
//

#include "OpenGLWnd.h"

////////////////////////////////////
// COrthographic view

```

```

class COrthographic : public COpenGLWnd
{
protected:
    COrthographic();           // protected constructor used by dynamic creation
    DECLARE_DYNCREATE(COrthographic)

// Attributes
public:
    int         m_lastMouseX,
               m_lastMouseY;
    float m_zoom,
           m_xpos,
           m_ypos;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(COrthographic)
    //}AFX_VIRTUAL

// Implementation
protected:
    virtual ~COrthographic();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
    //{AFX_MSG(COrthographic)
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_ORTHOGRAPHIC_H__89B35520_EA7E_4AF2_9E2F_3E6CE6E3347B__INCLUDED_)

```

C3DModelView.cpp

```

#include "StdAfx.h"
#include "Viewer.h"
#include "C3DModelView.h"

IMPLEMENT_DYNCREATE(C3DModelView, COpenGLView)

C3DModelView::C3DModelView(void)
{
    m_zoom = 0.0f;
    m_xpos = 0.0f;
    m_ypos = 60.0f;
    m_xrot = 0.0f;
    m_yrot = 0.0f;

    bStart = false;
}

C3DModelView::~C3DModelView(void)
{
}

BEGIN_MESSAGE_MAP(C3DModelView, COpenGLView)
    ON_WM_MOUSEMOVE()
    ON_MESSAGE(WM_MARKERCOORDINATE, &C3DModelView::OnUpdateCoordinates)
END_MESSAGE_MAP()

```

```

LRESULT C3DModelView::OnUpdateCoordinates(WPARAM wparam, LPARAM lparam)
{
    fMarkerCoordinate* fMarkers = reinterpret_cast<fMarkerCoordinate*> (lparam);

    float fNewX1, fNewY1, fNewZ1, fNewX2, fNewY2, fNewZ2;

    fNewX1 = fMarkers[0].xPos / 10.0f;
    fNewY1 = fMarkers[0].yPos / 10.0f;
    fNewZ1 = fMarkers[0].zPos / 10.0f;
    fNewX2 = fMarkers[1].xPos / 10.0f;
    fNewY2 = fMarkers[1].yPos / 10.0f;
    fNewZ2 = fMarkers[1].zPos / 10.0f;
    delete [] fMarkers;

    if(!bStart)
    {
        for(int i = 0; i < 50; i++)
        {
            markerCoordinates1[i].xPos = fNewX1;
            markerCoordinates1[i].yPos = fNewY1;
            markerCoordinates1[i].zPos = fNewZ1;

            markerCoordinates2[i].xPos = fNewX2;
            markerCoordinates2[i].yPos = fNewY2;
            markerCoordinates2[i].zPos = fNewZ2;
        }
        bStart = true;
    }
    else
    {
        for(int i = 0; i < 49; i++)
        {
            markerCoordinates1[i] = markerCoordinates1[i+1];
            markerCoordinates2[i] = markerCoordinates2[i+1];
        }

        markerCoordinates1[49].xPos = fNewX1;
        markerCoordinates1[49].yPos = fNewY1;
        markerCoordinates1[49].zPos = fNewZ1;

        markerCoordinates2[49].xPos = fNewX2;
        markerCoordinates2[49].yPos = fNewY2;
        markerCoordinates2[49].zPos = fNewZ2;
    }

    OnDraw(NULL);
    return 0;
}

void C3DModelView::DoOpenGLDraw()
{
    // Clear the buffers.
    glClearColor(1.0f, 1.0f, 1.0f, 0.75f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Look at the middle of the scene.
    glLoadIdentity();
    gluLookAt(-200, -60, 10, 0, 0, 0, 0, 1, 0);

    // Position the camera
    glTranslatef( m_xpos, -m_ypos, -m_zoom );

    // Rotate the camera
    glRotatef(-90.0f, 1.0f, 0.0f, 0.0f );
    glRotatef(-20.0f, 0.0f, 1.0f, 0.0f );
    glRotatef( m_xrot, 0.0f, 0.0f, 1.0f );
    glRotatef( m_yrot, 0.0f, 1.0f, 0.0f );

    // Set up some nice attributes for drawing the grid.
    glPushAttrib(GL_LINE_BIT | GL_ENABLE_BIT | GL_COLOR_BUFFER_BIT);
    glEnable(GL_LINE_SMOOTH);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glDisable(GL_LIGHTING);
    glHint(GL_LINE_SMOOTH_HINT, GL_NICEST);
}

```



```

        glLineWidth(1.0f);

        // Create the grid.
        glBegin(GL_LINES);
        for (int i = -100; i <= 100; i = i+10)
        {
            glColor4f(0.2f, 0.2f, 0.2f, 0.8f);
            glVertex3f((float)i, -100, 0);
            glVertex3f((float)i, 100, 0);
            glVertex3f(-100, (float)i, 0);
            glVertex3f(100, (float)i, 0);
        }
        glEnd();
        // End of Creating the grid

        glLineWidth(3.0f);
        if(bStart)
        {
            // Draw the lines.
            glBegin(GL_LINES);
            glColor4f(1.0f, 0.0f, 0.0f, 0.8f);
            for(int i = 0; i < 50; i++)
                glVertex3f(markerCoordinates1[i].xPos,
markerCoordinates1[i].yPos, markerCoordinates1[i].zPos);
            glEnd();

            glBegin(GL_LINES);
            glColor4f(0.0f, 1.0f, 0.0f, 0.8f);
            for(int i = 0; i < 50; i++)
                glVertex3f(markerCoordinates2[i].xPos,
markerCoordinates2[i].yPos, markerCoordinates2[i].zPos);
            glEnd();
        }

        glPopAttrib();
        glFlush();
    }

    // Change the perspective viewing volume to
    // reflect the new dimensions of the window.
    void C3DModelView::DoOpenGLResize(int nWidth, int nHeight)
    {
        // Create the viewport.
        glViewport(0, 0, nWidth, nHeight);

        // Load the identity projection matrix.
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();

        // Create a perspective viewport transformation.
        gluPerspective(45.0f, (float)nWidth / (float)nHeight, 0.1f, 1000.0f);

        // Go back to the modelview matrix.
        glMatrixMode(GL_MODELVIEW);

        glLoadIdentity();
        //gluLookAt(-12, 12, 12, 0, 0, 0, 0, 1, 0);
        gluLookAt(-200, -60, 10, 0, 0, 0, 0, 1, 0);
    }

    // Move the camera if control is being pressed and
    // the appropriate mouse button is being held down.
    void C3DModelView::OnMouseMove(UINT nFlags, CPoint point)
    {
        if( nFlags & MK_CONTROL )
        {
            if( nFlags & MK_LBUTTON )
            {
                // Left mouse button is being
                // pressed. Rotate the camera.
                if ( m_lastMouseX != -1 )
                {
                    m_xrot += point.x - m_lastMouseX;
                    m_yrot += point.y - m_lastMouseY;
                    // Redraw the viewport.
                    OnDraw( NULL );
                }
            }
        }
    }

```

```

        m_lastMouseX = point.x;
        m_lastMouseY = point.y;
    }
    else if ( nFlags & MK_MBUTTON )
    {
        // Middle mouse button is being
        // pressed. Zoom the camera.
        if ( m_lastMouseY != -1 )
        {
            m_zoom += point.y - m_lastMouseY;
            // Redraw the viewport.
            OnDraw( NULL );
        }
        m_lastMouseY = point.y;
    }
    else if ( nFlags & MK_RBUTTON )
    {
        // Right mouse button is being
        // pressed. Pan the camera.
        if ( m_lastMouseX != -1 )
        {
            m_xpos += (point.x - m_lastMouseX) * 0.15f;
            m_ypos += (point.y - m_lastMouseY) * 0.15f;
            // Redraw the viewport.
            OnDraw( NULL );
        }
        m_lastMouseX = point.x;
        m_lastMouseY = point.y;
    }
    else
    {
        m_lastMouseX = -1;
        m_lastMouseY = -1;
    }
}
else
{
    m_lastMouseX = -1;
    m_lastMouseY = -1;
}

COpenGLView::OnMouseMove(nFlags, point);
}

```

ConnectDlg.cpp

```

// ConnectDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Viewer.h"
#include "ConnectDlg.h"
#include <windows.h>
#include "DataAcquiringThread.h"

// CConnectDlg dialog

IMPLEMENT_DYNAMIC(CConnectDlg, CDialog)

CConnectDlg::CConnectDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CConnectDlg::IDD, pParent)
    , m_NOFTR(_T(""))
    , m_NOFR(_T(""))
    , m_Eipaddress(0)
    , m_Cipaddress(0)
{
}

CConnectDlg::~CConnectDlg()
{
}

void CConnectDlg::DoDataExchange(CDataExchange* pDX)

```

```

{
    CDialog::DoDataExchange(pDX);
    DDX_Text(pDX, IDC_EDT_NOFTR, m_NOFTR);
    DDX_Text(pDX, IDC_EDT_NOFR, m_NOFR);
    DDX_IPAddress(pDX, IDC_EDT_EIPADDRESS, m_Eipaddress);
    DDX_IPAddress(pDX, IDC_EDT_CIPADDRESS, m_Cipaddress);
    DDX_Control(pDX, IDC_NOT_STATUS, m_Status);
}

BEGIN_MESSAGE_MAP(CConnectDlg, CDialog)

    ON_BN_CLICKED(IDOK, &CConnectDlg::OnBnClickedOk)
END_MESSAGE_MAP()

// CConnectDlg message handlers
void CConnectDlg::OnBnClickedOk()
{
    //CString m_Eipaddress;
    //GetDlgItem( IDC_EDT_EIPADDRESS )->GetWindowText( m_Eipaddress);
    ///code not needed
    ///char *string = new char[m_Eipaddress.GetLength() +1];
    ///string = m_Eipaddress.GetBuffer(m_Eipaddress.GetLength() +1);
    ///m_Eipaddress.ReleaseBuffer();

    //CString m_Cipaddress;
    //GetDlgItem( IDC_EDT_CIPADDRESS )->GetWindowText( m_Cipaddress);
    ///code not needed
    ///char *string1 = new char[m_Cipaddress.GetLength() +1];
    ///string1 = m_Cipaddress.GetBuffer(m_Cipaddress.GetLength() +1);
    ///m_Cipaddress.ReleaseBuffer();

    //CString m_NOFTR;
    //GetDlgItem( IDC_EDT_NOFTR )->GetWindowText( m_NOFTR);
    ///code not needed
    ///char *string2 = new char[m_NOFTR.GetLength() +1];
    ///string2 = m_NOFTR.GetBuffer(m_NOFTR.GetLength() +1);
    ///m_NOFTR.ReleaseBuffer();
    //
    //{
    //MySdk2Client sdkClient;

    //
    /// Try to connect to the given EVaRT SDK2 server
    //if (sdkClient.Connect(m_Eipaddress, m_Cipaddress,atoi(m_NOFTR))
    //{
    //    m_Status.SetWindowText("Connected");
    //    // Just wait until we've received the required number of frames
    //    while (sdkClient.IsFinished() == false)
    //    {
    //        Sleep(1000);
    //    }
    //    sdkClient.Disconnect();
    //    m_Status.SetWindowText("Finished Recording Frames");
    //}
    //else
    //{
    //    m_Status.SetWindowText("Could not connect to EVaRT System");
    //}

    //
    //}

}

/* Example code I made to get code from IP control(Cstring) convert it to a (char)
and send it to the and edit box
-----

```

```

CString m_Eipaddress;
GetDlgItem( IDC_EDT_EIPADDRESS )->GetWindowText( m_Eipaddress);

char *string = new char[m_Eipaddress.GetLength() +1];
string = m_Eipaddress.GetBuffer(m_Eipaddress.GetLength() +1);
m_Eipaddress.ReleaseBuffer();

m_NOFTR = m_Eipaddress;
GetDlgItem( IDC_EDT_NOFTR )->SetWindowText( m_NOFTR );

/* Example code I made to test get/set IP address
-----

CString m_Eipaddress;
GetDlgItem( IDC_EDT_EIPADDRESS )->GetWindowText( m_Eipaddress);

CString m_Cipaddress = m_Eipaddress;
GetDlgItem( IDC_EDT_CIPADDRESS )->SetWindowText( m_Cipaddress );

/* Example code I made to test value variables
-----

UpdateData();
double NOFTR, NOFR;
NOFTR = atof(m_NOFTR);

NOFR = NOFTR * 2 ;

m_NOFR.Format("%.3f", NOFR);

UpdateData(FALSE);

/* Example code I made to test control variables
-----

float EvertIP, ClientIP, Nofr;
char StrEvertIP[50], StrClientIP[50], StrNofr[50];

m_EvertIP.GetWindowText(StrEvertIP, 50);
m_ClientIP.GetWindowText(StrClientIP, 50);

EvertIP = atof(StrEvertIP);
ClientIP = atof(StrClientIP);

Nofr = EvertIP * ClientIP;

sprintf(StrNofr, "%.3f", Nofr);

m_Nofr.SetWindowText(StrNofr);

-----

double Noftr, Nofr;
char StrNoftr[22], StrNofr[22];

m_Noftr.GetWindowText(StrNoftr, 22);
Noftr = atof(StrNoftr);
Nofr = Noftr * 4;

sprintf_s(StrNofr, "%.3f", Nofr);

m_Nofr.SetWindowText(StrNofr); */

```

ConnectThread.cpp

```
// ConnectThread.cpp : implementation file
//

#include "stdafx.h"
#include "ConnectThread.h"
#include "MySdk2Client.h"

// CConnectThread

IMPLEMENT_DYNCREATE(CConnectThread, CWinThread)

CConnectThread::CConnectThread()
{
    m_pConnectParams = NULL;
    m_pFrameQueue = NULL;
    m_pDataAcquiringThread = NULL;
}

CConnectThread::~CConnectThread()
{
}

BOOL CConnectThread::InitInstance()
{
    // TODO: perform and per-thread initialization here
    return TRUE;
}

int CConnectThread::ExitInstance()
{
    // TODO: perform any per-thread cleanup here
    return CWinThread::ExitInstance();
}

BEGIN_MESSAGE_MAP(CConnectThread, CWinThread)
END_MESSAGE_MAP()

// CConnectThread message handlers

int CConnectThread::Run()
{
    while(!m_pConnectParams || !m_pDataAcquiringThread || !m_pFrameQueue)
        Sleep(500);

    MySdk2Client sdk2Client;
    sdk2Client.SetFrameQueue(m_pFrameQueue);
    if(sdk2Client.Connect(m_pConnectParams->sServerIP, m_pConnectParams->sClientIP,
m_pConnectParams->iMaxFrames))
    {
        while(!sdk2Client.IsFinished())
            Sleep(1000);
        sdk2Client.Disconnect();
        MessageBox(NULL, "Finished receiving frames.", "Connect Thread",
MB_OK|MB_ICONINFORMATION);
    }
    else
    {
        MessageBox(NULL, "Failed to connect to the EVaRT system.", "Connect
Thread", MB_OK|MB_ICONEXCLAMATION);
        return 0;
    }

    m_pDataAcquiringThread->SetDisconnectedEvent();
    return 1;
}

void CConnectThread::SetConnectParams(sConnectParams* params)
{
    m_pConnectParams = params;
}

void CConnectThread::SetFrameQueue(CFrameQueue* pFrameQueue)
```

```

{
    m_pFrameQueue = pFrameQueue;
}

void CConnectThread::SetDataAcquiringThread(CDataAcquiringThread*
pDataAcquiringThread)
{
    m_pDataAcquiringThread = pDataAcquiringThread;
}

```

DataAcquiringThread.cpp

```

// DataAcquiringThread.cpp : implementation file
//

#include "stdafx.h"
#include "DataAcquiringThread.h"
#include "ConnectThread.h"

// CDataAcquiringThread

IMPLEMENT_DYNCREATE(CDataAcquiringThread, CWinThread)

CDataAcquiringThread::CDataAcquiringThread()
{
    m_hFrameHasArrivedEvent = NULL;
    m_hDisconnectedEvent = NULL;
    m_bConnectParamsReady = false;
    m_pFrame = NULL;
}

CDataAcquiringThread::~CDataAcquiringThread()
{
    if(m_hFrameHasArrivedEvent)
        ::CloseHandle(m_hFrameHasArrivedEvent);
    if(m_hDisconnectedEvent)
        ::CloseHandle(m_hDisconnectedEvent);
}

BOOL CDataAcquiringThread::InitInstance()
{
    // TODO: perform and per-thread initialization here
    m_hFrameHasArrivedEvent = ::CreateEvent(NULL, FALSE, FALSE, NULL);
    ::ResetEvent(m_hFrameHasArrivedEvent);

    m_hDisconnectedEvent = ::CreateEvent(NULL, FALSE, FALSE, NULL);
    ::ResetEvent(m_hDisconnectedEvent);

    m_cFrameQueue.SetDataAcquiringThread(this);

    return TRUE;
}

int CDataAcquiringThread::ExitInstance()
{
    // TODO: perform any per-thread cleanup here
    return CWinThread::ExitInstance();
}

BEGIN_MESSAGE_MAP(CDataAcquiringThread, CWinThread)
END_MESSAGE_MAP()

// CDataAcquiringThread message handlers

int CDataAcquiringThread::Run()
{
    while(!m_bConnectParamsReady)
        Sleep(500);
    CConnectThread* pConnectThread =
(CConnectThread*)::AfxBeginThread(RUNTIME_CLASS(CConnectThread));
    pConnectThread->SetConnectParams(&m_sConnParams);
    pConnectThread->SetFrameQueue(&m_cFrameQueue);
    pConnectThread->SetDataAcquiringThread(this);
}

```

```

bool bExit;
sFrameOfData* pFrameOfData;
bExit = false;
pFrameOfData = NULL;

//for testing
FILE* pMemoFile = fopen("FrameNumber.txt", "w");

while(!bExit)
{
    if(::WaitForSingleObject(m_hFrameHasArrivedEvent, 10) == WAIT_OBJECT_0)
    {
        // Add codes here ==>>
        pFrameOfData = m_cFrameQueue.GetFrame();
        fprintf(pMemoFile, "%d\n", pFrameOfData->iFrame);

        fMarkerCoordinate* fMarkers = new fMarkerCoordinate[2];
        fMarkers[0].xPos = pFrameOfData->BodyData[0].Markers[0][0];
        fMarkers[0].yPos = pFrameOfData->BodyData[0].Markers[0][1];
        fMarkers[0].zPos = pFrameOfData->BodyData[0].Markers[0][2];

        fMarkers[1].xPos = pFrameOfData->BodyData[0].Markers[5][0];
        fMarkers[1].yPos = pFrameOfData->BodyData[0].Markers[5][1];
        fMarkers[1].zPos = pFrameOfData->BodyData[0].Markers[5][2];

        if(m_pFrame->m_hwndTopLeft)
            PostMessage(m_pFrame->m_hwndTopLeft,
WM_MARKERCOORDINATE, NULL, reinterpret_cast<LPARAM> (fMarkers));

        /*if(m_pFrame->m_hwndTopRight)
            PostMessage(m_pFrame->m_hwndTopRight,
WM_MARKERCOORDINATE, NULL, reinterpret_cast<LPARAM> (fMarkers));*/
    }
    else
    {
        if(::WaitForSingleObject(m_hDisconnectedEvent, 1) ==
WAIT_OBJECT_0)
        {
            bExit = true;
        }
    }

    //for testing
    fclose(pMemoFile);
    //MessageBox(NULL, "The file has been closed", "Data Acquiring Thread",
    MB_OK|MB_ICONINFORMATION);

    return 1;
}

void CDataAcquiringThread::SetConnectParamsReady()
{
    m_bConnectParamsReady = true;
}

void CDataAcquiringThread::SetFrameHasArrivedEvent()
{
    if(!::SetEvent(m_hFrameHasArrivedEvent))
        MessageBox(NULL, "Failed to set the ArrivedEvent", "DataAcquiring
Thread", MB_OK|MB_ICONEXCLAMATION);
}

void CDataAcquiringThread::SetDisconnectedEvent()
{
    ::SetEvent(m_hDisconnectedEvent);
}

void CDataAcquiringThread::SetMainFrame(CMainFrame* pFrame)
{
    m_pFrame = pFrame;
}

//=====
void CDataAcquiringThread::SetServerIP(CString sServerIP)
{
    m_sConnParams.sServerIP = sServerIP;
}

```

```

void CDataAcquiringThread::SetClientIP(CString sClientIP)
{
    m_sConnParams.sClientIP = sClientIP;
}

void CDataAcquiringThread::SetMaxFrames(int iMaxFrames)
{
    m_sConnParams.iMaxFrames = iMaxFrames;
}
//=====

```

FrameQueue.cpp

```

// FrameQueue.cpp : implementation file
//

#include "stdafx.h"
#include "FrameQueue.h"
#include "DataAcquiringThread.h"

// CFrameQueue

CFrameQueue::CFrameQueue()
{
    m_pDataAcquiringThread = NULL;
}

CFrameQueue::~CFrameQueue()
{
}

// CFrameQueue member functions
sFrameOfData* CFrameQueue::GetFrame()
{
    sFrameOfData* ptrFrame;
    m_csForFrameQueue.Lock();
    if(!m_FrameList.IsEmpty())
        ptrFrame = (sFrameOfData*)m_FrameList.RemoveHead();
    else
        ptrFrame = NULL;
    m_csForFrameQueue.Unlock();
    return ptrFrame;
}

void CFrameQueue::AddFrame(sFrameOfData* ptrFrame)
{
    m_csForFrameQueue.Lock();
    m_FrameList.AddTail((void*)ptrFrame);
    m_csForFrameQueue.Unlock();
    m_pDataAcquiringThread->SetFrameHasArrivedEvent();
}

void CFrameQueue::SetDataAcquiringThread(CDataAcquiringThread* pDataAcquiringThread)
{
    m_pDataAcquiringThread = pDataAcquiringThread;
}

```

OpenGLView.cpp

```

#include "StdAfx.h"
#include "OpenGLView.h"

IMPLEMENT_DYNCREATE(COpenGLView, CView)

COpenGLView::COpenGLView(void)
{
}

COpenGLView::~COpenGLView(void)
{
}

BEGIN_MESSAGE_MAP(COpenGLView, CView)
    ON_WM_CREATE()
    ON_WM_DESTROY()

```



```

        ON_WM_ERASEBKGD()
        ON_WM_SIZE()
    END_MESSAGE_MAP()

    BOOL COpenGLView::SetupPixelFormat()
    {
        static PIXELFORMATDESCRIPTOR pfd =
        {
            sizeof(PIXELFORMATDESCRIPTOR),    // size of this pfd
            1,                                // version number
            PFD_DRAW_TO_WINDOW |              // support window
            PFD_SUPPORT_OPENGL |              // support OpenGL
            PFD_DOUBLEBUFFER,                  // double buffered
            PFD_TYPE_RGBA,                     // RGBA type
            24,                                // 24-bit color depth
            0, 0, 0, 0, 0, 0,                  // color bits ignored
            0,                                  // no alpha buffer
            0,                                  // shift bit ignored
            0,                                  // no accumulation buffer
            0, 0, 0, 0,                        // accumulation bits ignored
            16,                                // 16-bit z-buffer
            0,                                  // no stencil buffer
            0,                                  // no auxiliary buffer
            PFD_MAIN_PLANE,                    // main layer
            0,                                  // reserved
            0, 0, 0                            // layer masks ignored
        };

        int nPixelFormat = ::ChoosePixelFormat( m_pDC->GetSafeHdc(), &pfd );

        if ( nPixelFormat == 0 )
            return FALSE;

        return ::SetPixelFormat( m_pDC->GetSafeHdc(), nPixelFormat, &pfd );
    }

    BOOL COpenGLView::InitOpenGL()
    {
        // Get a DC for the Client Area
        m_pDC = new CClientDC(this);

        // Failure to Get DC
        if( m_pDC == NULL )
            return FALSE;

        if( !SetupPixelFormat() )
            return FALSE;

        // Create Rendering Context
        m_hRC = ::wglCreateContext( m_pDC->GetSafeHdc() );

        // Failure to Create Rendering Context
        if( m_hRC == NULL )
            return FALSE;

        // Make the RC Current
        if( !::wglMakeCurrent( m_pDC->GetSafeHdc(), m_hRC ) )
            return FALSE;

        // Usual OpenGL stuff
        glClearDepth(1.0f);
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_TEXTURE_2D);

        return TRUE;
    }

    // Initialize OpenGL when window is created.
    int COpenGLView::OnCreate(LPCREATESTRUCT lpCreateStruct)
    {
        if (CView::OnCreate(lpCreateStruct) == -1)
            return -1;

        if ( !InitOpenGL() )
        {
            MessageBox( "Error setting up OpenGL!", "Init Error!",
                MB_OK | MB_ICONERROR );
        }
    }

```

```

        return -1;
    }

    return 0;
}

// Set a few flags to make sure OpenGL only renders in its viewport.
BOOL COpenGLView::PreCreateWindow(CREATESTRUCT& cs)
{
    cs.lpszClass = ::AfxRegisterWndClass(CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS |
CS_OWNDC,
        ::LoadCursor(NULL, IDC_HAND), NULL, NULL);
    cs.style |= WS_CLIPSIBLINGS | WS_CLIPCHILDREN;

    return CView::PreCreateWindow(cs);
}

void COpenGLView::OnDraw(CDC* pDC)
{
    SetContext();
    DoOpenGLDraw();
    SwapGLBuffers();
}

void COpenGLView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);

    if ( 0 >= cx || 0 >= cy || nType == SIZE_MINIMIZED )
        return;

    SetContext();
    DoOpenGLResize(cx, cy);
}

// Shutdown this view when window is destroyed.
void COpenGLView::OnDestroy()
{
    CView::OnDestroy();

    wglMakeCurrent(NULL, NULL);
    wglDeleteContext(m_hRC);
    if(m_pDC)
    {
        delete m_pDC;
        m_pDC = NULL;
    }
}

// Override the errase background function to
// do nothing to prevent flashing.
BOOL COpenGLView::OnEraseBkgnd(CDC* pDC)
{
    return TRUE;
}

#ifdef _DEBUG
void COpenGLView::AssertValid() const
{
    CView::AssertValid();
}

void COpenGLView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}
#endif // _DEBUG

```

OpenGLWnd.cpp

```

#include "StdAfx.h"
#include "OpenGLView.h"

IMPLEMENT_DYNCREATE(COpenGLView, CView)

COpenGLView::COpenGLView(void)

```

```

{
}

COpenGLView::~COpenGLView(void)
{
}

BEGIN_MESSAGE_MAP(COpenGLView, CView)
    ON_WM_CREATE()
    ON_WM_DESTROY()
    ON_WM_ERASEBKGD()
    ON_WM_SIZE()
END_MESSAGE_MAP()

BOOL COpenGLView::SetupPixelFormat()
{
    static PIXELFORMATDESCRIPTOR pfd =
    {
        sizeof(PIXELFORMATDESCRIPTOR),    // size of this pfd
        1,                                // version number
        PFD_DRAW_TO_WINDOW |              // support window
        PFD_SUPPORT_OPENGL |              // support OpenGL
        PFD_DOUBLEBUFFER,                  // double buffered
        PFD_TYPE_RGBA,                     // RGBA type
        24,                                // 24-bit color depth
        0, 0, 0, 0, 0, 0,                  // color bits ignored
        0,                                  // no alpha buffer
        0,                                  // shift bit ignored
        0,                                  // no accumulation buffer
        0, 0, 0, 0,                        // accumulation bits ignored
        16,                                // 16-bit z-buffer
        0,                                  // no stencil buffer
        0,                                  // no auxiliary buffer
        PFD_MAIN_PLANE,                    // main layer
        0,                                  // reserved
        0, 0, 0                            // layer masks ignored
    };

    int nPixelFormat = ::ChoosePixelFormat( m_pDC->GetSafeHdc(), &pfd );

    if ( nPixelFormat == 0 )
        return FALSE;

    return ::SetPixelFormat( m_pDC->GetSafeHdc(), nPixelFormat, &pfd );
}

BOOL COpenGLView::InitOpenGL()
{
    // Get a DC for the Client Area
    m_pDC = new CClientDC(this);

    // Failure to Get DC
    if( m_pDC == NULL )
        return FALSE;

    if( !SetupPixelFormat() )
        return FALSE;

    // Create Rendering Context
    m_hRC = ::wglCreateContext( m_pDC->GetSafeHdc() );

    // Failure to Create Rendering Context
    if( m_hRC == NULL )
        return FALSE;

    // Make the RC Current
    if( !::wglMakeCurrent( m_pDC->GetSafeHdc(), m_hRC ) )
        return FALSE;

    // Usual OpenGL stuff
    glClearDepth(1.0f);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_TEXTURE_2D);

    return TRUE;
}

```

```

// Initialize OpenGL when window is created.
int COpenGLView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CView::OnCreate(lpCreateStruct) == -1)
        return -1;

    if ( !InitOpenGL() )
    {
        MessageBox( "Error setting up OpenGL!", "Init Error!",
            MB_OK | MB_ICONERROR );
        return -1;
    }

    return 0;
}

// Set a few flags to make sure OpenGL only renders in its viewport.
BOOL COpenGLView::PreCreateWindow(CREATESTRUCT& cs)
{
    cs.lpszClass = ::AfxRegisterWndClass(CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS |
CS_OWNDC,
        ::LoadCursor(NULL, IDC_HAND), NULL, NULL);
    cs.style |= WS_CLIPSIBLINGS | WS_CLIPCHILDREN;

    return CView::PreCreateWindow(cs);
}

void COpenGLView::OnDraw(CDC* pDC)
{
    SetContext();
    DoOpenGLDraw();
    SwapGLBuffers();
}

void COpenGLView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);

    if ( 0 >= cx || 0 >= cy || nType == SIZE_MINIMIZED )
        return;

    SetContext();
    DoOpenGLResize(cx, cy);
}

// Shutdown this view when window is destroyed.
void COpenGLView::OnDestroy()
{
    CView::OnDestroy();

    wglMakeCurrent(NULL, NULL);
    wglDeleteContext(m_hRC);
    if(m_pDC)
    {
        delete m_pDC;
        m_pDC = NULL;
    }
}

// Override the errase background function to
// do nothing to prevent flashing.
BOOL COpenGLView::OnEraseBkgnd(CDC* pDC)
{
    return TRUE;
}

#ifdef _DEBUG
void COpenGLView::AssertValid() const
{
    CView::AssertValid();
}

void COpenGLView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

```

```

#endif // _DEBUG

Orthographic.cpp
// Orthographic.cpp : implementation file
//

#include "stdafx.h"
#include "Viewer.h"
#include "Orthographic.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// COrthographic

IMPLEMENT_DYNCREATE(COrthographic, COpenGLWnd)

COrthographic::COrthographic()
{
    m_zoom = 5.0f;
    m_xpos = 0.0f;
    m_ypos = 0.0f;
}

COrthographic::~COrthographic()
{
}

BEGIN_MESSAGE_MAP(COrthographic, COpenGLWnd)
   //{{AFX_MSG_MAP(COrthographic)
    ON_WM_SIZE()
    ON_WM_MOUSEMOVE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// COrthographic diagnostics

#ifdef _DEBUG
void COrthographic::AssertValid() const
{
    CView::AssertValid();
}

void COrthographic::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}
#endif // _DEBUG

////////////////////////////////////
// COrthographic message handlers

void COrthographic::OnSize(UINT nType, int cx, int cy)
{
    COpenGLWnd::OnSize(nType, cx, cy);

    if ( 0 >= cx || 0 >= cy || nType == SIZE_MINIMIZED )
        return;

    // Change the orthographic viewing volume to
    // reflect the new dimensions of the window
    // and the zoom and position of the viewport.
    SetContext();
    glViewport( 0, 0, cx, cy );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glOrtho( (float)(cx)/(float)(cy)*m_zoom-m_xpos,
(float)(cx)/(float)(cy)*m_zoom-m_xpos,
        -m_zoom+m_ypos, m_zoom+m_ypos, -200.0f, 200.0f );
    glMatrixMode( GL_MODELVIEW );
}

```

```

}

void COrthographic::OnMouseMove(UINT nFlags, CPoint point)
{
    // Move the camera if control is being
    // pressed and the appropriate mouse

    // button is being held down.

    CRect cr;

    GetClientRect( &cr );
    if ( nFlags & MK_CONTROL )
    {
        if ( nFlags & MK_MBUTTON )
        {
            // Middle mouse button is being
            // pressed. Zoom the camera.
            if ( m_lastMouseY != -1 )
            {
                m_zoom += (point.y - m_lastMouseY) * 0.25f;
                // Apply the position changes to
                // the viewport.
                OnSize( SIZE_MAXIMIZED, cr.Width(), cr.Height() );
                OnDraw( NULL );
            }
            m_lastMouseY = point.y;
        }
        else if ( nFlags & MK_RBUTTON )
        {
            // Right mouse button is being
            // pressed. Pan the camera.
            if ( m_lastMouseX != -1 )
            {
                m_xpos += (point.x - m_lastMouseX) * 0.25f;
                m_ypos += (point.y - m_lastMouseY) * 0.25f;
                // Apply the position changes to
                // the viewport.
                OnSize( SIZE_MAXIMIZED, cr.Width(), cr.Height() );
                OnDraw( NULL );
            }
            m_lastMouseX = point.x;
            m_lastMouseY = point.y;
        }
        else
        {
            // No mouse button was pressed.
            // Mark the mouse flags to indicate
            // the camera did not move last
            // message.
            m_lastMouseX = -1;
            m_lastMouseY = -1;
        }
    }
    else
    {
        // Control was not pressed.
        // Mark the mouse flags to indicate
        // the camera did not move last
        // message.
        m_lastMouseX = -1;
        m_lastMouseY = -1;
    }
}

COpenGLWnd::OnMouseMove(nFlags, point);
}

```