Interactive Exploration of Historic Information via Gesture Recognition

Sam Bailey

A thesis submitted for the degree of
Master of Science
at the University of East Anglia
September 2012

Interactive Exploration of Historic Information via Gesture Recognition

Sam Bailey

Abstract

Developers of interactive exhibits often struggle to find appropriate input devices that enable intuitive control, permitting the visitors to engage effectively with the content. Recently motion sensing input devices like the Microsoft Kinect or Panasonic D-Imager have become available enabling gesture based control of computer systems. These devices present an attractive input device for exhibits since the user can interact with their hands and they are not required to physically touch any part of the system. In this thesis we investigate techniques to enable the raw data coming from these types of devices to be used to control an interactive exhibit. Object recognition and tracking techniques are used to analyse the user's hand where movement and clicks are processed. To show the effectiveness of the techniques the gesture system is used to control an interactive system designed to inform the public about iconic buildings in the centre of Norwich, UK. We evaluate two methods of making selections in the test environment.

At the time of experimentation the technologies were relatively new to the image processing environment. As a result of the research presented in this thesis, the techniques and methods used have been detailed and published [3] at the VSMM (Virtual Systems and Multimedia 2012) conference with the intention of further forwarding the area.

Table of Contents

Chapter 1

Introduction

There has always been a need for interactive exhibitions which are discussed in Section 2, most of which are focused on providing a means in which to gain knowledge on a particular subject and or subjects. Usually, there are large quantities of information that are difficult to disseminate to the masses without a medium that is intriguing and informative.

Traditional historical exhibition techniques are recognised throughout the world. An exhibition is mostly, but not strictly, the term used for a collection of items on display. Archaeological heritage data is collectively growing and is being accumulated from various sources, some of which are highly detailed. It is of growing concern as to what information should be presented publicly, how the data is handled, how it is presented and to what audience.

Architectural information on discovered artifact's may be attained with conventional methods such as historical documentation, literary material and according to research by Kwee et al. [30], internet resources, who also split information into two categories, hard and soft. Firstly, hard information usually consists of the physical, tangible attributes of the artefact, its blueprints and design (if available). Secondly,

soft information gives details on the artefacts original intentions and narrative through time. Categorising information aids in interpretation and assimilation but it is not necessarily possible to divide into simple categories. Although these are valid information gathering techniques they do not offer an interactive or empirical experience, furthermore their presentation may not be tailored for public viewing. Secondly, even though internet resources may be readily available and easily acquired the historical validity may be lacking.

The ideas behind this project were to enable such interaction in which information on a specific subject could be presented, the subject being the twelve listed buildings found in and around Norwich City. Rich archives on these buildings had and is still being gathered and stored in a collective database owned by the Norwich Heart. The twelve listed heritage buildings span the Norman, medieval, Georgian, Victorian and modern eras.

Figure 1.1 shows all twelve listed buildings and consist of Norwich Castle, Norwich Cathedral, The Great Hospital, The Halls - St. Andrew's and Blackfriars, The Guildhall, Dragon Hall, The Assembly House, St James Mill, St Johns Roman Catholic Cathedral, Surrey House, City Hall and The Forum.

'HistOracle' became the project name, having a loose meaning towards an oracle of history.

1.1 Aims

Designing and creating an interactive and informative experience proved challenging as there are many technical and ergonomic aspects that must be considered. The

Figure 1.1: The Norwich twelve listed heritage buildings. From left to right: (1A) Norwich Castle, (2A) Norwich Cathedral, (3A) The Great Hospital, (4A) The Halls - St. Andrew's and Blackfriars, (1B) The Guildhall, (2B) Dragon Hall, (3B) The Assembly House, (4B) St James Mill, (1C) St Johns Roman Catholic Cathedral, (2C) Surrey House, (3C) City Hall, (4C) The Forum

aims initially discussed were in place to roughly guide the project and were a larger superset of aims that this MSc fell under. These aims consisted off;

1. Enable a more extensive exploration experience of selected listed buildings in the Norwich area.

2. The ability to view these buildings from different time periods using custom 3D model data.

3. Display contextual information, in the form of text, videos, images, audio files etc. about the currently viewed structure.

4. To actively retrieve information from a database enabling expandability to add new listed buildings.

5. Create a system that is compatible on multiple platforms with a multitude of technology such as touch-screens, wireless input, haptic feedback devices and a standard mouse and keyboard.

In addition to the aims previously explained it was originally decided that extra functionality was required. This included;

1. To render Norwich in real-time using pre-built models created from LIDAR.
2. Rendering shadows and using other processing techniques to create realistic images.
3. Enable free exploration of Norwich using an intuitive navigation system.

It was clear that certain aims were out of scope, such as enabling complete freedom to explore and producing a product that was compatible with many forms of input device. Enabling freedom of movement involves many techniques to be implemented as part of the overall system, in scope of the project there was not enough time to complete these. Also, it was decided that concentrating on a single input device (gesture input) was more suitable, although transferring control to other devices was not a large task.

1.2 Motivations

The field of virtual historical environments spans a multitude of theoretical and practical application. Virtual immersive technologies have been accessible for private and public uses for decades but in recent years (15 or so) newer technologies have created possibilities for real-time historical interactive virtual environments with a variety of input devices.

Currently there is a lot of emphasis on technologies such as the devices used in this project. Their uses span into a multitude of areas, offering new possibilities to individuals where physical touch is not possible or desired. Some of these areas include but are not restrained to; medical imaging (for situations like surgeries [38]), security [46], environment mapping (real time 3D environment generation) [52] and

virtual reality [27]. Technologies such as the Panasonic D-Imager and the Microsoft Kinect are relatively new. There is the potential for ever advancing gesture based applications which could aid people throughout their lives as well as entertain. The practical applications of devices such as these expand beyond simple entertainment purposes.

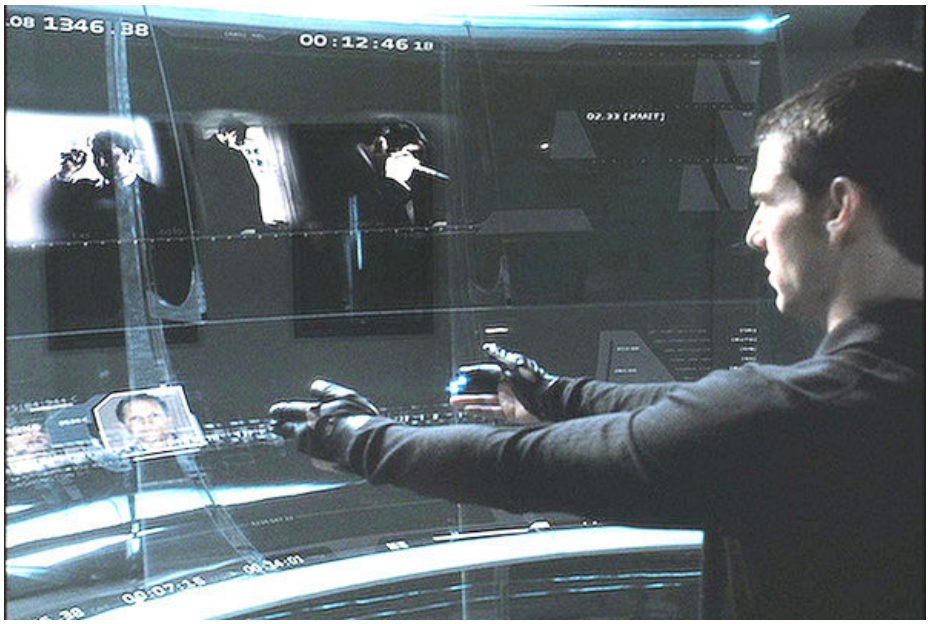

Figure 1.2: The futuristic interface used in the film Minority Report. Image taken from [1]

The line between fantasy science-fiction and reality is beginning to merge. The ability to accurately track and acknowledge the presence of a person and individual hand and fingers is something that Hollywood will manufacture (or fake) whereas science will produce. A prime example of the pre-conception of this technology comes from the film 'Minority Report', in which the protagonist interacts with a computer via a spatially static holographic interface, using autonomous gesture input (can be seen in Figure 1.2).

Chittaro et al.[10], Lepouras and Vassilakis [34], and Niederauer et al.[44] have presented exhibitions that succeeded in capturing the publics attention, showing that interactivity can play a large role in the success of an exhibition.

In the next Chapter (2), this work among other relevant work is discussed in context of presenting information on particular historical subjects in informative and interactive ways, including the use of contemporary technologies such as depth interpretation. In the rest of the thesis, subjects are discussed relating to both image processing (Chapter 3) and 3D rendering (Chapter 4). Some topics include depth analysis and hand tracking, which concentrate on the practicalities of finding and tracking an object in a scene using depth data, and the methods used to help eliminate errors. Also, 3D model organisation and loading which explains how an efficient loading method was used to load large amounts of data in relation to current graphics hardware, for displaying on screen interactively. Finally, the results (Chapter 5) from public testing are presented and discussed in regards of the success of the whole project, followed by a discussion and a conclusion of the project and the results gathered (Chapter 6) .

Chapter 2

Background

2.1 Navigation Techniques

We are exposed to navigational tasks regularly; for example we use our extremities to coordinate and navigate ourselves to complete particular tasks such as interacting with our surroundings. We perceive our environment through our senses with the freedom to view in any direction. Successful navigation in a virtual environment requires consideration of many contributory factors such as sensory orientation and interface expectations, which are discussed throughout this chapter. Furthermore, common problems associated with navigation faced by interface designers of virtual environments are; transitions between point A and B, the restrictions applied to transitional movement, viewport rotation and the input/output expectations. We learn by playing, interacting with objects, solving puzzles and also from observation, all of which can be supplied with augmented and virtual reality. Research by Mosaker [43] shows that tourists interacting with a cultural heritage system, want to learn through doing, being told, observing and asking. Engagement with the activity is paramount to a users experience. They want to feel the presence of time and enjoy the cultural involvements tied to the object/artefact. Without user navigation, virtual environments would not provide an interactive experience, rather place the user in an expensively rendered environment that could have been computer and presented

via other methods.

2.1.1 Viewpoint Control

Since the late 70's, there has been research into view or 'camera' control within a virtual environment that has proven that user and application requirements both play important roles in influencing application design. It is widely considered that the metaphor explanations of camera control by Ware et al.[61], give clarity to the three control types that are consistently used in graphics applications, 'eyeball-in-hand', 'scene-in-hand' and 'flying-vehicle-control'. Firstly, it is clear that a degree of understanding can be attained by each title. Assumptions can be formed as to how the camera system will respond to user input due to the physical representation of the metaphor. One can assume that while physically holding an object certain possibilities arise such as complete freedom of rotation and movement around the object, hence the 'eyeball' is 'in hand'. Continuing this logic, further deductions can be made in reference to the 'scene-in-hand' and 'flying-vehicle-control' in that both navigation techniques produce different results, flying vehicle control giving complete movement with the six degrees of freedom, enabling the user to traverse over the landscape according to a directional vector, usually calculated from the viewing angle. Importantly, their research has shown that the interpretation of these metaphors/descriptive control methods greatly affects user understanding and interaction, even before first use.

Movement in a 3D environment is measured within six degrees of freedom consisting of translation and rotation in each axis. The degree of control given to a user often dictates the success of a navigational system. For example, flight simulators

promote realistic control of an aeroplane; in this case the user would be able to perform movement in each axis (yaw, pitch and role) whereas a car simulator is restricted to yaw (left and right).

The metaphors created by Ware et al.were a significant contribution to the field of virtual 3D viewpoint control, providing a clear overview of how we interpret and analyse physical navigational tasks. Evaluation of these metaphors in real situations was undertaken by Bowman et al.[7] who substantiate Ware's research, suggesting that a successful virtual travel technique promotes a collection of seven component; speed, accuracy, spatial awareness, ease of learning, ease of use, information gathering and level of immersion. For example a video game requires speed, accuracy, ease of use and ease of learning for fast passed action and real-time events. On the other hand, a historical simulation is solely focused on the reconstruction of an environment where spatial awareness, information gathering and ease of use are paramount. Moreover, their research consisted of experimental navigation using head tracking hardware and the flying-vehicle metaphor. Restrictions were applied to the X and Y axis, showing that virtual reality navigation using head and limb tracking hardware was improved when the user was able to traverse the environment via pointing instead of looking in the desired direction.

However, navigation in 3D environments is not strictly contained to confined areas, which the previous control systems are tailored for. Environmental navigation in arbitrary virtual environments is advancing and becoming easier to navigate. The possibilities that arise from 'flying-vehicle-control' do not accommodate large scale navigation, furthermore if this type of control were to be used in an environment requiring large movements, errors would occur and accuracy would be lost. After studying various existing control metaphors, Mackinlay et al[35] and Fukatsu

et al.[22] developed techniques in which the user gained movement control over vast environments. Mackinlay realised this technique by placing a marker on the target geometry, the viewpoint then traverses towards the target until reached - similar to a more locally based technique by [5]. When the distance between viewpoint and target decreases, camera speed is also reduced, thus enabling both rapid and controlled movements towards a point of interest. Furthermore, this provides adaptive movement as control is relative to distance, giving the advantage of detailed, scalable movement over large and small distances. Although advantageous for traversing an environment efficiently there is little or no discussion of spatial awareness (defined as a necessity for a successful camera system by Bowman et al. [7]), a user could become disoriented due to rapid movement to new locations.

Fukatsu et al.[22] try to remedy this issue by concentrating on providing both global and local viewpoints simultaneously. Their research introduced the user to multiple viewpoints of the virtual environment. The local view being that of the first person, from the user's position and orientation and the second global 'bird's eye' view situated above, maintaining a constant distance from the users global viewpoint coordinates (defined along the positive Y and negative X axis). Experiments were undertaken involving tasks set in a tailored virtual environment such as a maze, where landmarks and points of interest were placed. Task completion time was recorded to analyse discrepancies between bird's eye view settings, such as orientation and proximity. Furthermore, the technique gives a unique overlay detailing positional information to the user, enabling a further understanding of their surrounds, ultimately improving accuracy while positioning objects and presenting a global understanding and perception of size, but does not offer a complete overview of the entire environment which is proposed by Stoakley et al.[54].

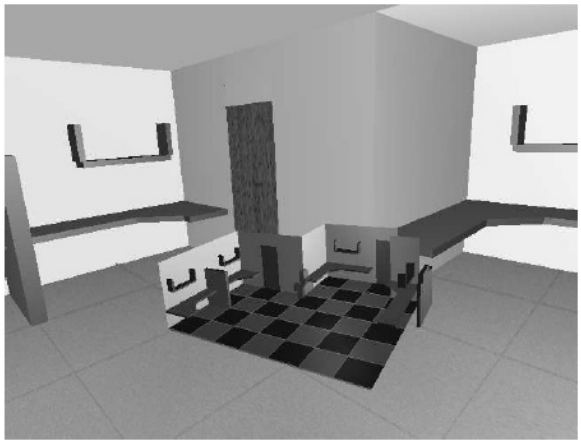

Figure 2.1: World In Miniature (WIM).

The world-in-miniature (WIM) was coined by Stoakley as a camera metaphor essentially providing a 3D overview of the geometry being explored, in this instance, presented via augmented reality on a separate hand-held tablet (Figure 2.1). An advantageous perspective was rendered, encouraging the benefits gained from map-like tasks. Moreover, this is also reflected in the 'through-the-lens' technique which, proposed by Stoev et al.[55], is an augmentation of both the WIM and bird's eye view. The user is presented with a 'magic lens', offering a secondary window that renders the scene from a different location and view orientation. In this example, an external hand-held pad is used to further augment the interaction process with a 'grab-and-drag' style of movement, where the pad can be dragged to any position. Two positional states also give behavioural control, the first translating the magic lens according to the users viewpoint and the second retaining a static position in all axes. Also, the pad offers a preview window in which the viewer is able to adjust their secondary view precisely without requiring the user to physically move to the location. Furthermore, these techniques are similar to the 'bird's eye' view previously discussed, each providing valuable positional information in different situations, the first offering intuitive cognitive recognition of orientation and position, the latter providing multiple, scalable points of view that enable easy object manipulation in a 3D environment.

In more recent years, the study of camera controls and their constraints has been carried out by Fitzmaurice et al.[19]. Their research collected valuable information

consisting of user ability, experience and tolerance in a commercial 3D modelling environment. Furthermore, the research contained a mix of techniques previously mentioned, applied in an examinable fashion where data was collected on various aspects of interaction. Close examination showed that user interaction with 3D software, both beginners and experienced users, can be considerably enhanced by a clear user interface. Moreover inexperienced users often fall into previously learned habits or become confused in 3D space, often transferring their own understanding of 2D applications to 3D tasks. Moreover, it was found that beginners were often unaware of particular tools available and compensated with other methods, often worsening the situation. User interaction is also researched and is discussed later in Section (2.3).

2.1.2 Virtual Exhibits

Camera metaphors previously discussed fare well in situations that involve or require direct user input. Due to the requirements of an interactive virtual environment complete control of the camera may not be possible, in these situations other traversal techniques could be implemented to provide the desired level of interaction. Environmental immersion can be conveyed in two common methods, the first being an interactive 3D environment containing informative object models, the second encapsulating a consecutive path of pre-rendered environmental images enabling 360 degree viewpoints at pre-defined stops.

Furthermore, the utilisation of a tour based systems has been explored by museums, the gradual progression through an information rich environment aids in the assimilation of data as Galyean [24] describes. This progressive restriction is a simple but effective analogy that provides the opportunity to explore a confined area (Figure 2.2) moreover, the area available for exploration progresses along a pre-defined path.

The system consists of a central anchor in which the viewpoint is attached to; any movement is dictated by a spring situated between the anchor and viewpoint thus restricting the user from distancing themselves from the central anchor. It was found that the 'river analogy' encouraged people to explore but also provided more control over intentional focus points in the environment.

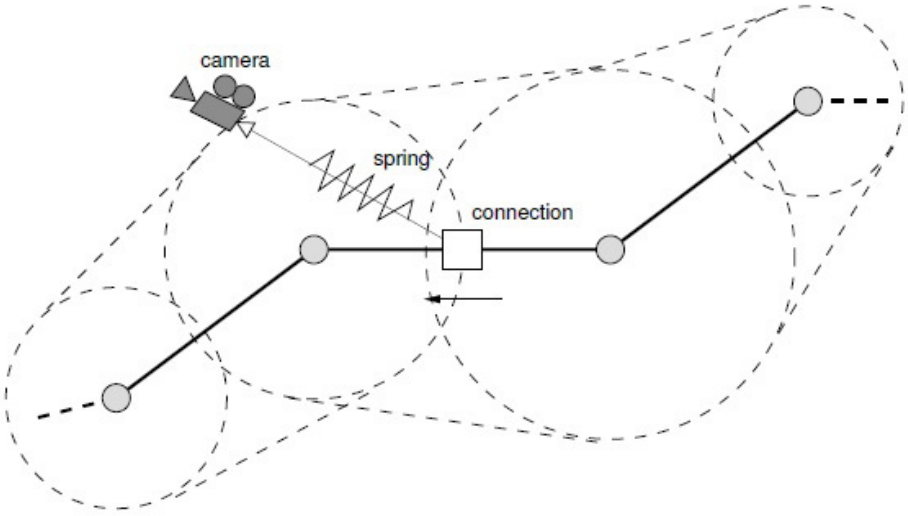

Figure 2.2: The 'river' analogy including a path of travel, spring constraint and maximum travel distance from each node.

Applying constraints to a user's experience can be limiting and sometimes frustrating due to the confinement within a designated area, on the other hand it offers the narrator or system designer to apply strict guidance to the user's experience. Laycock et al.[32] promote a system that combines efficient rendering of complex virtual environments such as detailed architecture and interaction using pre-rendered haptically aware movies. Due to the overhead required by haptic devices it is difficult to render high quality architecture and maintain an efficient runtime for the haptic device. Haptic feedback offers a higher level of user immersion and direct physical feedback that extends the user's understanding of the object in a virtual environment. However, achieving a high quality image while incorporating haptic feedback can be

problematic as a feedback device usually requires the application to be running at 1 kHz. Furthermore, this method of haptic preservation is realized by rendering a consecutive set of environment maps that depict a scene along a set path (often multiple branching paths, giving the impression of freedom). The maps are then rendered in realtime, removing the need to render the geometry directly and enables the processing power to be directed towards the haptic feedback device. Camera control is initialised using a spring system, assigning a virtual spring for both the X and Y axes separately much like the river analogy described in [24]. While the user interacts with the scene clicking in the direction they wish to move, this movement amount is calculated based on the magnitude of the spring force. While camera rotation is active, interaction with haptic objects is disabled to force a distinction between movement and object interaction, helping to stop confusion.

Although this analogy efficiently maintains the viewer within a specified path it does not dictate how or when the automatic path is generated. Intuitive high level exhibition design is a difficult topic in which Elmqvist et al.[17] and Chirrato et al.[10] both provide systems that attempt to remedy user compatibility with exhibit creation and automatic tour generation, respectively. The first consists of automated indoor and outdoor navigation of arbitrary environments calculated using voxels and path finding algorithms. The first being a voxel (volumetric element) representation of the environment in question using a recursive subdivision of space. This 3D voxel representation can then be used to deduce clear 'visibility sets' in which groups of voxels are stored together determined by their visibility and proximity to the points of interest. Furthermore, redundant voxel sets can be disregarded which in turn increases efficiency. All information is stored in a scene graph, depicting appropriate segment connections providing logical steps between spatial nodes. Elmqvist's research showed that tour generation computation times were not hindered by model

complexity (number of vertices) due to the combination of voxelisation and standard TSP (Travelling Salesman Problem) heuristics. On the other hand scenes containing high amounts of occlusion, such as the interior of a museum, produced fast computational times. Although an efficient path generation method is described, scene manipulation including artefact placement is not mentioned. Intuitive control over artefacts in a 3D virtual environment is paramount for general acceptance as virtual exhibition creation software.

Unlike Elmqvist, Chittaro et al.[10] created a system that enabled the curator to design a virtual tour from start (artefact and location selection) to finish (automated tour calculation) as shown in Figure 2.3. These efforts resulted in the ability to generate fully interactive tours from lesser experienced users, reducing the need for professional input.

Figure 2.3: A real-time rendering of a finalised tour with items placed in the virtual environment using the design system.

Curators are presented with a system based around familiarities gained from other software packages and general operating system use. Tour creation is based on Views of Interest (VOI), specific objects (usually artefacts) or manually designated views (such as murals) can be created which help to aid the curator through the design process and is used later to automatically construct the tour. Moreover, a different weighting is associated with each VOI, where a definition of how the object is treated is defined between a full field of view, front facing items, non occluding items

and visual size. The first describes whether the artefact is entirely included in the view; the second defines the front of the object that should be visible including a tolerance of 40 degrees. The third states if the object is able to be occluded by other artefacts in the scene. Finally, size defines the proportional amount of screen space the artefact covers. Moreover, this information is used to automatically generate the tour path from one VOI to another. The curator is able to link VOI's using a tailored interface that displays connecting nodes and their start, end and transitional variables. Unlike voxelisation segmentation, the automated tour generation renders each room to an image from an isometric top-down viewpoint where a pixel depicts a segment of the room, thus enabling a safe path to be calculated through the environment avoiding impassable objects, similar to the techniques proposed by Leronutti et al.[26].

On the opposite side of the spectrum, customised virtual exhibits are usually prefabricated to some degree as well as the environment in which the design is undertaken. The idea of using existing design and rendering software has been researched by Lepouras and Vassilakis [34], and Niederauer et al.[44] who delved into the realm of commercial game engines; CryEngine, Unigine and the Unreal Engine are all sophisticated examples. As specifically tailored hi-end systems require a large amount of development time, game engines offer an inexpensive graphical alternative. Thus the engine is a self contained rendering system normally consisting of many rendering techniques, user interface for custom development and often rules that can be applied to the objects in the virtual world, such as physics. For example, if the system required a high visual quality, most current game engine designs allow for the use of the most current techniques, enabling development time to be placed elsewhere.

2.2 Recognition

In the field of computer vision, body, face and limb recognition is at the forefront of sophisticated HCI techniques as it holds possibilities for further envelopment into arbitrary virtual environments. Furthermore, as well as obvious entertainment and gaming application it is an autonomous means of interaction with the physical world, and a potential method in which to aid in arbitrary social and everyday tasks. Real world applications is an area researched by many, Zhou and Hoang [70] implemented robust security surveillance tracking in which individuals could be recognised and tracked even at a distance.

2.2.1 Wireless Recognition

It is only logical to progress into the area of full body recognition (including limb and individual body part recognition), which in many situation is an amalgamation of techniques previously described that can satisfy more natural forms of spatial interaction. A conclusion can be made from naturalistic input such as those discussed, is that it is usually more intuitive to users because they are the controller. The notion of unencumbered control without restrictions has been explored by various scientists including Johan Warren [62]. A system intended for input via body recognition can either provide a small number of abstract actions which are easy to comprehend, or many actions which can be interpreted as literal translation. Higher detail facilitates the capacity for better translation, movement becomes nearer that of the users, natural and more literal which is nearer to a one-to-one relationship. More advanced recognition techniques and higher bandwidth allocation can foresee detailed expressive input that can react to an individual's personality and performance.

Previous research into the relatively new area of full body motion capture and

recognition has shown that current hardware is more than adequate for handling facial and body recognition in real-time. The aim of wireless gesture recognition is to enable completely free and unhindered application control using only hand and body gestures. A simplistic approach has been taken by Laakso and Laakso [31], who developed a cost-effective system aimed towards game control. The configuration uses a top down approach, approximating the user's position and mass by segmenting them from the ground, using a bounding volume placed around the user's mass aids in calculating the contour of the limbs. A vector is then composed using the least-squares method giving the vector's length and direction, indicating the fingers. From this, predefined gestures are interpreted as rudimentary actions such as move, jump, shoot etc. Although effective, taken out of a tightly controlled environment the system could be prone to errors.

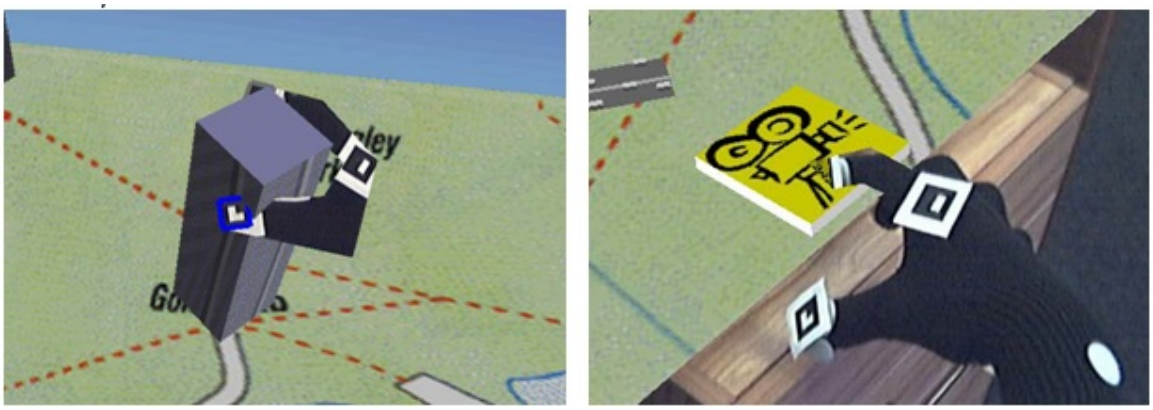

Figure 2.4: Augmented reality controlled via a visual goggles-and-glove approach where the thumb and forefinger are visually tracked while interacting with the environment. This approach is extremely restrictive due to the confined environment and hardware required.

Segmentation of individual body parts for recognition is one of many tasks that have to be overcome in a system that requires it. Firstly, it is usually more efficient to work with processed images where the background has been removed, this aids in defining movement amongst elements, as described by Maes et al.[36], Chu et al.[13]

and Carbini et al.[9]. Each technique differs in gesture processing but all are facilitated by background removal techniques.

Maes et al.[36] aimed to create fully autonomous interactive agents with specific behavioral patterns, such as a dog. The system aimed to provide an augmented reality without the need for intrusive wearable hardware such as a headset. Instead, the user is presented with a large screen acting as a 'magic mirror' in which all augmented information is displayed in conjunction to the user's actions. The user's 3D volume is used in conjunction with the ground plane to determine positional information, furthermore this is used for autonomous agents to navigate and interact. Intuitive control is given when interacting with virtual buttons; the user must make a "pointing gesture" over the required button which then selects the appropriate button. ALIVE (Artificial Life Interactive Video Environment) lends itself to multiple environments such as entertainment, teaching, training and digital assistants. Moreover, it was advantageous over a more cumbersome approach like goggles-and-gloves, experimented by [8], in which nodes placed on a table in a uniform square grid were used to visually calculate positional information. The same nodes are used on the users glove (seen in Figure 2.4), enabling interaction with augmented objects but limits the user to a first person view, where errors in accuracy and confusion can occur on orientation in the virtual environment.

Image processing was researched by Chu et al.[13] where compiled image information was gathered via four or more cameras, fitted with infra-red (IR) filters enabling low light level calibrations. Although, this decision implied that only a single user could be traced at any one time due to the difficulties encountered when segmenting IR information. Segmentation of streamed information is vital for classification, in order to aid in this process Chu et al. used back lighting to silhouette the users figure.

Visual information is then processed using four sets of scan lines, vertical, horizontal and at 45 and -45 degrees (Figure 2.5). Under the assumption that users interact with the system while standing, vertical scan lines are used in conjunction with the horizontal scan lines to calculate the user's volume (torso). Extremities are then processed when movement occurs, enabling detection of pointing direction according to volume mass collection. Both these techniques are used to translate body gestures into actions appearing to the user or in the virtual environment but neither handle multiple inputs.

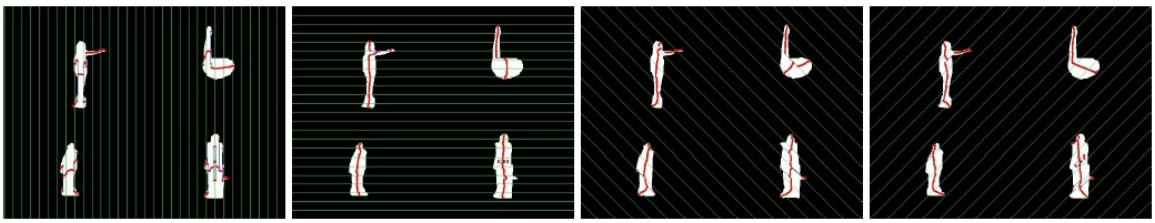

Figure 2.5: The scanlines used to detect the users volume. Four angles are used, vertical, horizontal, 45° and -45°.

The latter uses a more advanced method of facial and hand recognition, defined by Carbini et al.[9] in which the left and right hand are treated separately, the first for (main hand) is used for selection and the other hand for manipulation. The image is divided into 16 segments, discarding those that do not contain facial or extremity information or any sign of movement. Furthermore, as multiple users could be tracked at once the bounding volume of each unique user had to be determined. Boundaries of the individual's 'working sphere' is placed around the user with the head at the origin. This acts as an efficient detection area and also provides indication when two working areas intersect, also accommodating for body shape and eradicates detection errors when user proximity is too close, disregarding hand gestures in this area (Figure 2.6).

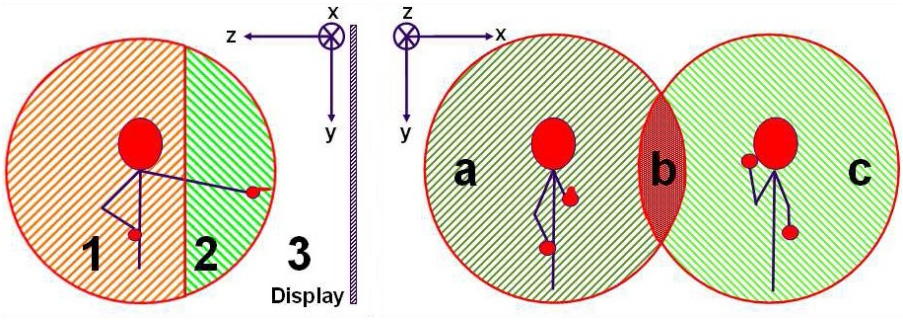

Figure 2.6: On the left, 1, 2 and 3 are the working boundary, recognition area and ignored area, respectively. An example of multiple working areas, a and c are individuals with unique working areas and b is the intersection at which gestures are ignored.

Correct sampling is paramount for hand gesture recognition, for example sign language, an alternate method of hand extraction is proposed by Quek [47] in which hand strokes are extracted from streaming video via edge detection, yielding motion vectors from this data. Information is smoothed to locally align and reduce noise from the result. Vector calculations are intrinsic for motion estimation with 2D data. This is also realised by Yang et al.[67], showing that trajectories can be estimated by a two frame comparison. Furthermore the system is engineered to recognise patterns in dynamic images without prior knowledge of the subject. Sign utterances (hand gestures) are recognised from hand location, shape and motion. Images are processed and segmented using a multi-scale segmentation method, regions between consecutive frames are matched to obtain two-view correspondence. Transformations are then computed using the two-frame correspondence enabling pixel level trajectories to be computed (Figure 2.7). At the final stage of analysis, a time delay neural network (TDNN) is used to recognise feature vectors in temporal space.

Recognition via feature vectors (not unlike those used by Quek [47]) has been proposed by William T. Freeman and Michal Roth ([21]) in which feature vectors are used to distinguish explicit hand gestures. The underlying process consists of two

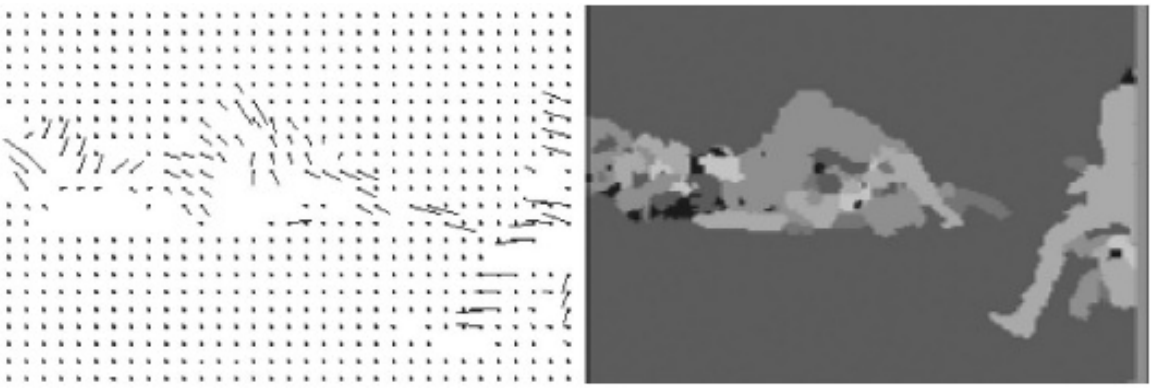

Figure 2.7: On the left are the final calculated vectors from the motion segmentation. On the right pixels of the same intensities and feature correspondences are grouped.

phases, training and practical application. During training the dominant orientations in the image are calculated using the feature vector histogram. This histogram is generated by analysing contrasting areas in the image, where heavily contrasting areas are considered. Figure 2.8 depicts an orientation histogram example. The image (a) produces the raw histogram (b) where a clearly defined spike represents the vector angle found in the image. Due to the computational capabilities of the hardware during experimentation these feature vectors were categorised into 36 bins. Therefore each feature was required to be aware of its neighbours, this was accomplished by blurring the histogram (c). The polar plot of the resulting histogram was then saved as the comparison, where the angle is considered to be the direction of intensity gradient plus 90°.

Although the proposed technique is generally robust against lighting discrepancies, it is fundamentally flawed in various respects. As the training process provides the option to configure the system to recognise static gestures, feature vectors do not offer much leniency between similar gestures. It is uncommon that the user will produce the exact pre-defined posture, small variations in hand orientation produce significantly different polar plot images but are usually considered to be identical by

the user. Furthermore, for the system to successfully attain gestures from real-time data the hand must dominate the screen which is impractical for any real-world usage.

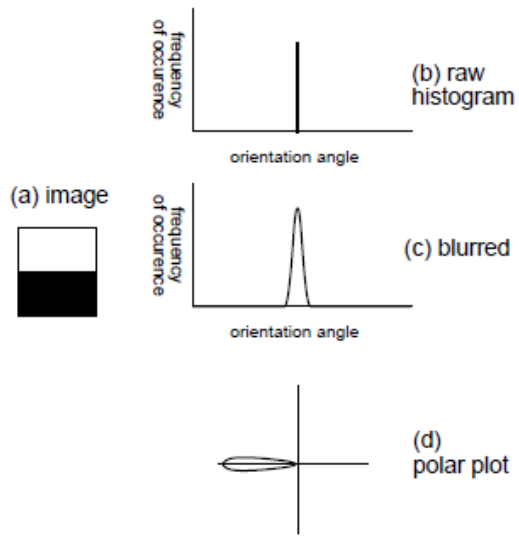

Figure 2.8: A simple example of the orientation histogram, starting with the original image which is processed into a raw histogram, blurred and then processed into a polar plot for gesture recognition (*a*, *b*, *c* and *d* respectively.

Enabling recognition via trajectories is also utilised by Kenji Oka et al. ([45], [49]) who propose a novel system which enables desktop (literal) interaction with detection of multiple fingertips. The technology used is infra-red, where advanced prediction techniques and hand gestures are analysed to interact with the work surface to perform specific tasks. They are quick to point out that their system has advantages over similar methods which use colour and/or background segmentation to successfully track a users hand, this is due to the use of an infra-red camera.

The method proposed initially searches for the user's hand via a search window of a fixed size, which including part of the wrist, indicating the user's orientation. This region is then analysed using normalized correlation with a circular template of appropriate size (this was reliable as the distance between the camera and the user's hand was consistent, although the size was varied depending on the size of the image initially found). The center of the user's hand is discovered by morphological erosion, the idea being that the center of the hand contains the largest volume and erosion

exposes this area.

Fingertip prediction was enabled by the use of vector trajectories, opening possibilities for object manipulation. Trajectories were predicted by analysing fingertip positions between frames, this enabled the calculation of velocity. The sum of the square of distances between detected and predicted fingertips is computed for all possible combinations, taking the result with the least sum as the most reasonable. This enabled the recognition of symbolic gestures, based on the Hidden Markov Model of multiple fingertips.

Figure 2.9: An example of three fingertips being tracked and interacting with the desk, augmented by the projector.

2.2.2 Recognition using 3D Data

Previous research discussed have directed their work towards facial, limb and body recognition using technologies that provide 2D positional information. This decision was not necessarily out of choice but due to hardware limitations at the time. When considering active motion capture for body and gesture recognition the progressive step is to acquire information on all three dimensions X, Y and Z. The third axis (Z) contains depth information of the scene, which can vastly increase the recognition accuracy when processing all given information. Recent technological advancements have enabled hardware that can seamlessly stream depth information (Z) alongside traditional video (X, Y) without the need for any spatially aware external devices such as headsets, VR gloves or accelerometer hardware.

Some of the first substantial research in this field was carried out Krumm et al.[29]

whos methodology included a visual recognition system for seamless home integration that could aid in general remote tasks such as pausing a video when the viewer leaves the viewing area. 'EasyLiving' uses two or three stereoscopic cameras to track up to three individuals via RGB and depth image data. Pixel disparities within a small range were calculated to create 'people shaped blobs', their location, (past and present) was stored for each instance. Using the knowledge of the cameras location in the room a individual can be tracked, evaluating positional information from all cameras.

Engineering a multi-person full -body recognition system using 3D/stereoscopic data to successfully compute and react to user presence is difficult when only considering a single input type such as colour histogram in streaming data. This was realised by Darrell et al.[15] who understood that exploiting multiple visual processing techniques would increase robustness, theoretically if one module were to fail the next could succeed. Processing visual information for segmentation consisted of first smoothing the image and removing disparities with a morphological closing operator, face pattern detection then pinpoints probable faces in the scene. Flesh hue regions are then classified, components are then connected using stereo image data and finally individual attributes are recorded for each new individual (Figure 2.10 (a), (b) and (c) respectively). Furthermore, effort was placed in storing information on tracked individuals, enabling short, medium and long-term tracking, the former processing individuals continuously in view and the latter relying on statistical appearance models which determines an individual's physical attributes. Darrel et al. concluded that range data, or depth data, was significantly important for person segmentation in arbitrarily complicated scenes, also that face pattern analysis was the most successful element for individualisation.

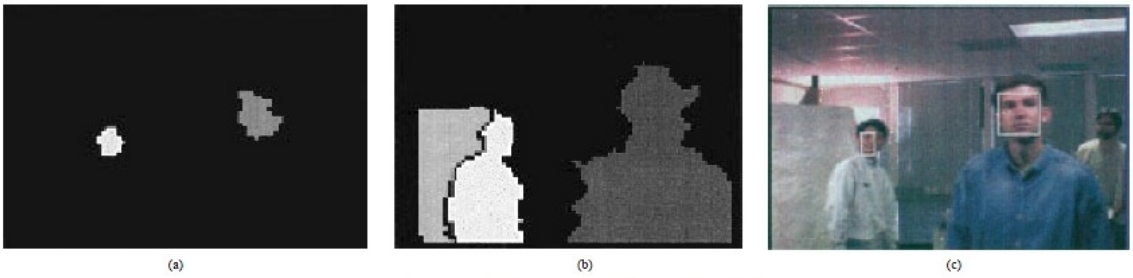

Figure 2.10: Face detection shown in (c) is accomplished with flesh hue region detection (a) and elemental composition using stereo data (b).

Contemporary research in the area of 3D data processing has proven beneficial in areas outside computer graphics, such as medical imaging, aiding the hearing impaired, a form of polygraph and sign language interpretation [42], which is notoriously hard to understand digitally due to the complexity of hand gestures and lack of boundaries between signs [66]. The benefits of acquiring and handling 3D for gesture recognition have been explored by Malassiotis et al.[37] whos research showed that this step towards complete wireless freedom is extremely important in the area of full body recognition and is enabled by techniques such as 3D image acquisition. The system designed does not rely on colour, illumination and camera configuration and other determinate environmental attributes as 3D data is less prone to interference and is sufficient to segment the user's arm and hands from the background. The image is segmented to help categorisation, assuming that the body is formed of clusters, the arm, forearm hands etc. Clusters are formed by analysing the depth data, measuring the Euclidian distance which forms small regions with respect to their neighbouring pixels. Clusters are then merged according to cluster variance, terminating when a certain number of clusters is reached. These then form extremity information which can be categorised by their shape, such as the elongated shape of the arm as seen in Figure 2.11.

Figure 2.11: Depth information displayed as colour (red to green), where red indicates a smaller depth value thus is closer to the source (camera). This information is processed to give segmentation of the hand and arm and forearm.

2.3 Interaction

Given the opportunity scientists and software developers would design the most intriguing multi-user systems possible, as technology advances; new possibilities that have never been realised become possible. Furthermore, technologies such as touch screens and 3D positional cameras (Zabulis et al.[68]) have been in the pipeline for many years but have only recently been realised and become commercially available such as the Microsoft Xbox Kinect.

2.3.1 Efficient Human-Computer Interaction

Interactions between human and computer (HCI, Human-Computers Interaction) greatly affect the outcome and understanding held by the user, a well designed interface will stay incognito, only becoming obvious when needed. On the other hand a poorly designed interface will hinder the applications performance and even prevent the user from completing desired tasks. Each unique system (in the context of non-automated commercial and non-commercial products) requires input in some form such as haptic devices, head tracking hardware, a stylus and traditional mouse and keyboard are all valid methods of input and provide advantages in different scenarios.

Careful consideration of interaction theory can translate to well developed HCI techniques, the methodologies proposed by Fitzmaurice at al.[19] (previously discussed), and Chittaro et al.[11] both realise that augmentation of the physical translates differently when confined in a virtual environment. The first methodology concentrates on the visual representation and interpretation of possible 3D actions that can be executed. 'Safe navigation' is an interesting theory in which a set of rules are offered and sometimes applied to the user. For example, the user can be restricted to a building environment with the use of collision detection. Research showed that best results were produced when interface buttons were within the user's visual focus and easily accessible. Moreover, important or commonly used buttons could be defined by their size, colour and transparency highlighting their importance. Knowing what the user requires means the navigation can be customised to the specifications. For example there are differences in navigation requirements, when viewing 'object-in-hand' the navigation would be restricted to zooming and rotation around the objects axis. On the other hand, 'free' navigation implies the user would be able to walk around the scene, meaning motion would be the primary navigational motivation.

Readily available inexpensive hardware, such as a mouse and keyboard, are not always suitable for system that requires multiple or advanced input techniques in order to achieve specific application requirements. Advanced hardware input devices, such as touch screens and head tracking hardware, offer a multitude of input techniques but are sometimes out of a projects budget. Zeleznik et al.[69] defined a multi-gesture technique that only requires the input of a single button input device, giving obvious advantages over other technologies. Traditional camera control systems use interfaces that require interaction with multiple button hardware devices or via on-screen controls mapped to particular control processes. Using gestures with 'UniCam', a single

button is needed to manipulate all camera control, leaving other buttons for application specific functions. The 'UniCam' enables the user to directly control camera operations via basic gesture interactions (Figure 2.12). Camera navigation includes translation, orbiting around a point of interest (scene in hand analogy), animating focus on an object's surface and zooming in on a region of the object. Screen space is divided into sections, each defining specific functions. For example, the screen edge constitutes rotation, when clicked on the user is able to rotate around the object. Navigation starts by recording the original input device position and determining the direction in which the user moves the cursor. Different directions trigger different actions, these actions include; vertical movements either zoom or translate horizontally, horizontal movement enters the 2D film-plane translation mode. Zeleznik's virtual environment interaction techniques do not cover unique ground but rather present an efficient method of interaction with previously developed concepts, more specifically it enhances the 'scene-in-hand' control type.

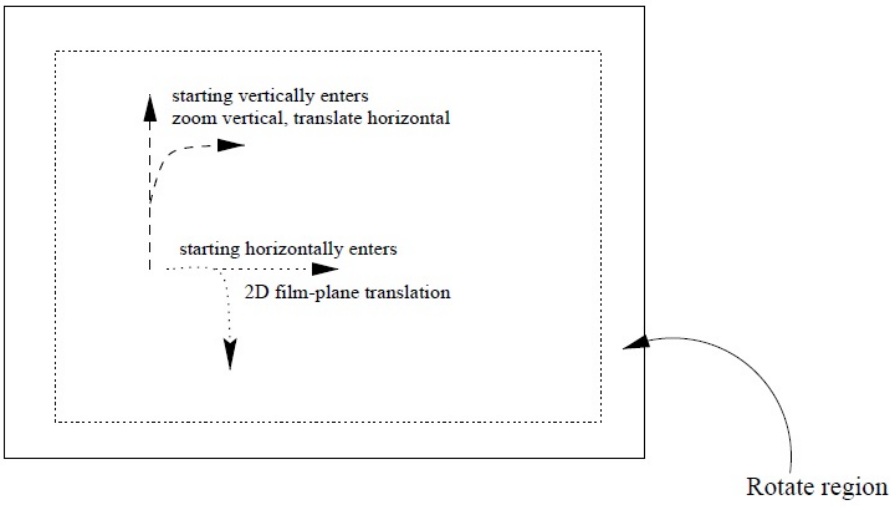

Figure 2.12: This diagram shows the initial gesture control entry conditions, where the start and end 'clicks' are evaluated in regards to their vectors, applying the appropriate action (translate, zoom, strafing and rotating) when detected.

Although well tailored towards single click input, more advanced input devices do not necessarily interface as efficiently. Moreover, this gestural recognition was further researched and substantiated by Mine et al.[41], who moved into the concept of full body gestural input. This area is a mixture of computer science, sensory study and kinaesthesia (the body's awareness of locomotion and position of extremities in space). Natural human instinct is to interact with object in a very physical manner; often gaining knowledge of an object's use by the objects physical constraints (e.g. a bottle top can only be unscrewed along a single axis). Working within 'arm's reach' provides clues towards the user's proprioception (perception of limb orientation and position), where framing user interaction and working within their working volume gives real-world tactile information which can improve the user's tactile experience, thus decreasing task completion time [41]. Certain mandatory and common tasks can become problematic when interfacing with a virtual environment purely by body gestures. Mine's research showed that certain user interface (UI) tasks, such as menu interaction, have to be executed in a different manner. Menu elements are usually required for general tasks but become lost when situated in world coordinates, especially when freedom of movement is given. On the other hand, elements that are consistently present, rendered in screen space coordinates, obscure the scene behind. It was found that enabling invocation to a 'pull-down' menu, with a fixed location situated just above the user's field of view, access was simple and kept other extremities to continue interaction with the scene behind. Widget control is also used to manipulate objects within the virtual environment, such as scale, movement and rotation. Research showed that the use of a physical widget tool over virtual widget representations situated on the selected object enabled a more tactile experience, fortifying accuracy during manipulation tasks and improving task performance. Representation of virtual movement controls play a vital role in accuracy and understanding, as shown by Chittaro et al.[12] and Fitzmaurice [19], who applied widget details to each

bounding box selection, the first applying visual aids to the geometry such as transparency, to visualise object placement. Both interaction and navigation are closely tied together, if one is clearly defined, the other usually follows suit.

Human computer interaction (HCI) spans a wide variety of areas, hardware devices and techniques; furthermore it is an integral aspect of computer science as it dictates many aspects of day-to-day and professional interaction. Such technologies that encapsulate the ability to track multiple users in a spectrum of situations offer themselves to installations such as virtual museum exhibitions. For example, when considering a traditional museum exhibit the aim is to provide as much information, in the form of media, text etc., within a confined area and to the masses. It is clear this implies that visitors were encouraged to enter and exit exhibits as they please, so information needed to be presented in an upfront and captivating manor. Zabulis et al.[68] describe a system that intuitively handles multiple users, presenting each with contextual information based on their location, length of visit and native language. The visual tracking system consists of eight cameras situated above the exhibition area connected to two computers, information from all cameras is synchronised over both computers (Figure 2.13). Visual information is processed into a 3D matrix consisting of voxels, as the

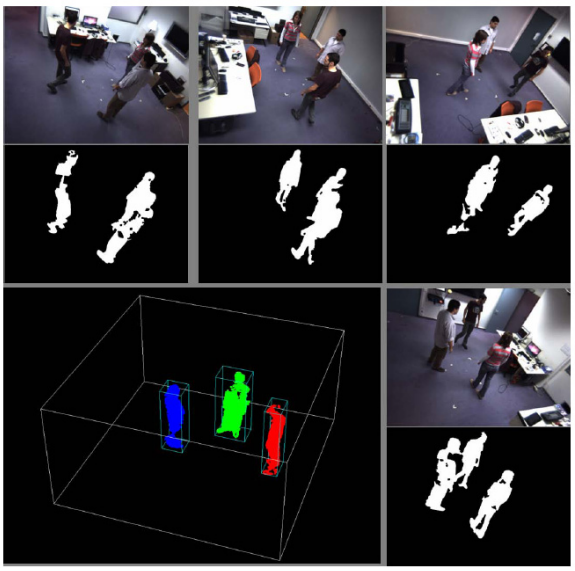

Figure 2.13: An example of three individuals (blue, green and red) being tracked using a single computer and information from four cameras. Each individual is separated from the background and represented in three dimensions.

voxels become occupied their state becomes 1.

A voxel is considered occupied when all images return an approximately similar result. Generating a voxelized reconstruction of the controlled environment aids in tracking multiple targets, certain scenarios require close examination such as multiple visitors entering the tracking area in close proximity and users standing near one another. As information is distributed on an individual level, each visitor is tracked as a unique blob, when a blob divides into two the secondary visitor becomes unique, inheriting the language settings of its parent. Finally, as each visitor enters the exhibition (through the appropriate language gate) they are presented with the artefact at its original size, displayed via two short throw rear projectors. The display is segmented into a 5 x 4 grid enabling 20 interaction slots, as the distance between visitor and screen decreases information dynamically changes according to the time spent in particular locations and the slot in which the visitor is standing. Furthermore, results were positive, resulting in visitors becoming fascinated with the technology used but eventually taking a deep interest in the exhibitions content. Moreover, the advantage of a system only requiring commercial hardware is its expandability and robustness, the number of computers required for efficient image processing is directly proportionate to the number of cameras used and the number of cameras required is determined by the estimated number of multiple users (3 cameras are adequate for tracking 4 individuals). Although the discussed methods are robust for a large amount of scenarios there are particular problems that cannot be addressed, such as users rotating in close proximity and individual limb/head tracking which is advantageous for a more detailed, immersive experience.

2.3.2 Hand and Body Gestures

Restrictive interaction has been shown to be detrimental to task performance, Barrie et al.[4] attempted to design a system in which virtual objects can be manipulated in an augmented reality but results showed that the use of low powered wireless sensors (Figure 2.14) and headset became cumbersome and impractical. The system was restricted by the headset connections and also assumes that the user is fixed is space, meaning only the top half of the body is used to capture.

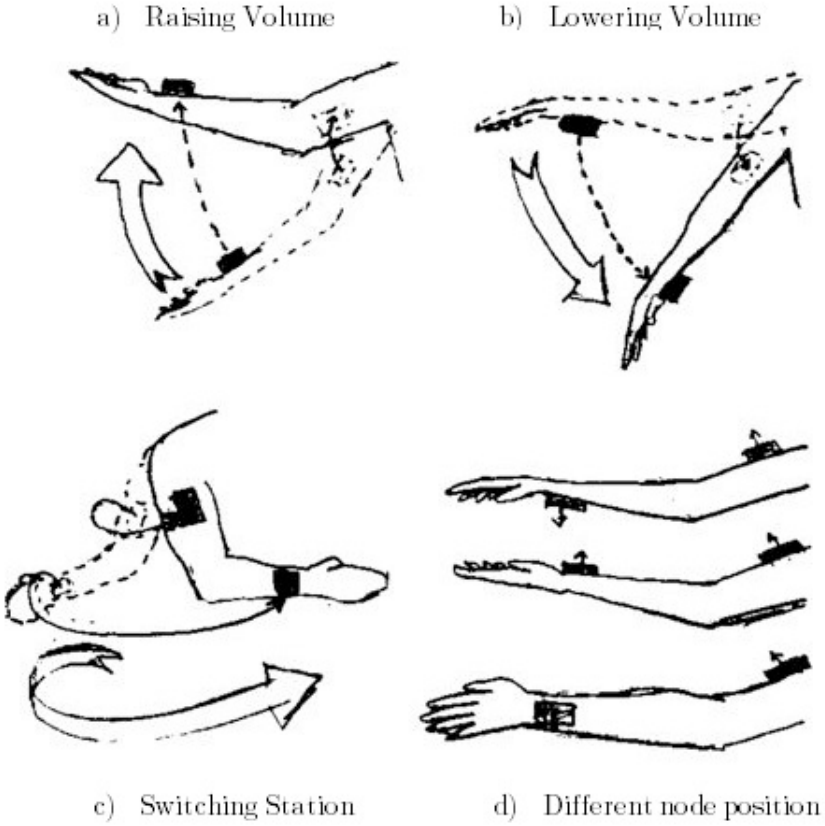

Figure 2.14: The diagram shows node placement on the arm and forearm. Gestures are tailored for virtual radio control giving generic controls over volume and stations.

With the advancement of depth devices such as the Microsoft Kinect and Panasonic D-Imager the availability of these devices and the depth data has become easily attainable. For such projects such as Barrie et al. have proposed, the possibility of

increased accuracy and functionality would be vast. Frati and Prattichizzo [20] have seen the potential of combining both traditional physical tracking technologies with the Microsoft Kinect and proposed a technique which allows the user to control an avatar, where items can be manipulated. To attain this, they use the depth information from the Kinect to aid in haptic accuracy. This is accomplished by firstly processing the depth data from 11bit to 8bit information. The closest object to the device (which is assumed is a hand) is computed, then the bounding box calculated in which OpenCV is used to establish the fingertips. Haptic hardware was then attached to the user's hand, providing tactile information of the interaction between the user's avatar and virtual objects (seen in Figure 2.15).

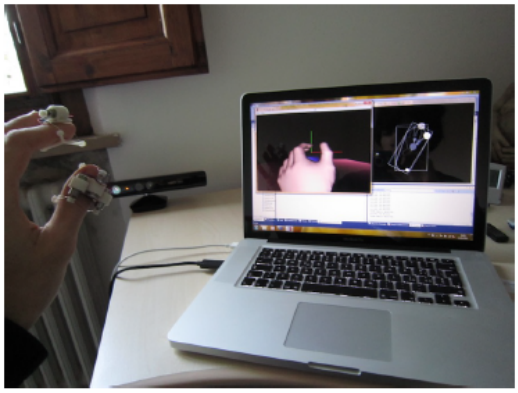

Figure 2.15: The diagram the Kinect based haptic system where the Kinect is placed in front of the user. The screen shows both the tracking algorithm results (right) and virtual environment and avatar (left).

A more contemporary and natural mode of interaction without the requirements (physical hardware attached to the user) of the previous interactive techniques is proposed by Boulos et al. [6] who describe the use of multiple technologies to enable full control of 3D map software, such as Google Maps, Bing Maps 3D and NASA World Wind using only gestures. The project is highly dependent on specific SDK (software development kit) environments from both Microsoft and other companies. The use of OpenNI and FAAST Middleware enabled environmental exploration including navigation, rotating, zooming, panning and tilting. Because of the dependencies found in this project, adaptation to hardware that is not developed by Prime Sense was

essentially impossible, limiting the software to a niche set of devices.

Moving towards hand segmentation, Matthew Tang [59] proposes a technique in which the hand can be recognised using the Microsoft Kinect via skin estimation and balancing, also using the Microsoft Kinect SDK. Using a combination of colour and depth filtering (e.g. the low resolution depth data is used to segment skin colours in both the foreground (hand) and background (neck)) accuracy was found from experimentation between the two, meaning that lower light levels produced poor skin tone results but could be rectified a certain degree with the depth information. The centre of mass (essentially the centre of the hand) was calculated with Equation 2.3.1. Where N is the number of pixels in the image, $\mathrm{p}(x, y)$ is the probability that the pixel at (x, y) is part of the hand. Then the centre of mass (xc, yc) is simply calculated as such:

$$x_c = \frac{1}{N} \sum_{(x,y)} xp(x, y); y_c = \frac{1}{N} \sum_{(x,y)} yp(x, y) \tag{2.3.1}$$

Image data was then filtered by angle offset of the hands centre using Equation 2.3.2, producing information on the hand and converted into a histogram of 80 bins.

$$a(x, y) = tan^{-1}(\frac{x - x_c}{y - y_c}) \tag{2.3.2}$$

Finally, the result could be evaluated with SURF features. These features were then used for gesture recognition, which identified sequences of poses (e.g. hand open, hand closed would classify as a 'grasp'. Vice versa would classify as a 'dropping' gesture). The hardware used for these projects is discussed in more detail in the next Section, along with their commercial realisation.

2.4 Commercial Environment

Deployment of full body tracking technologies in a commercial environment opens new possibilities for human computer interaction, especially games development. It is well known that the games development sector of computer graphics is one of the largest facilitators of new input and graphical techniques. Furthermore, the casual gaming market has aimed to provide completely wireless and interactive experiences with affordable hardware. One of the first mass-produced commercially successful wireless devices is the Nintendo Wii remote.

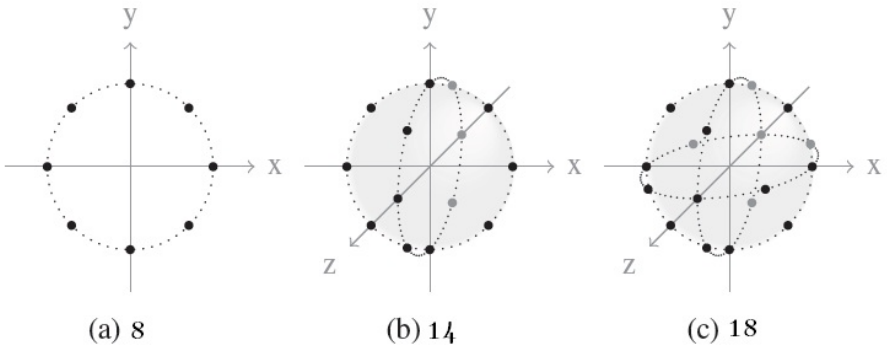

Figure 2.16: Cluster set numbers, where clusters are distributed evenly over each axis in a circular formation. This aids in simplifying positional information. It was found that (b) produced the best results as the user did not often move the Wii controller directly behind themselves.

The controller is primarily used for Nintendo Wii game interaction but offers development and experimentation in other areas outside of the gaming environment. Motion and rotation can be detected by accelerometers in the Wiimote (Nintendo Wii Controller) which can be translated into gestural input. Furthermore the Wiimote offers spatial awareness, gathering information on the controller's position, orientation and movement in 3D space. Moreover, the system designed by Schlömer et al.[50] first quantities the accelerometer data (which is provided as a vector) using a k-mean algorithm. Two filters are used to simplify input data, an idle state and a directional equivalence filter. A Hidden Markov Model is then used to aid in gesture recognition

as it allows for variance in input, then a Bayes-classifier is used for the remaining component. Information is quantified for readability via cluster sets, each cluster is placed in circular form, consisting of eight cluster points distributed equally around the circle (Figure 2.16). Points are orthogonal axis aligned and restricted over the three axis, (X, Y and Z) creating a quantified sphere of detection nodes.

Microsoft Xbox Kinect (code name Project Natal), consists of a 8-bit VGA camera and 11-bit 640x480 monochrome depth sensor for accurate facial and body recognition (Figure 2.17). Currently, hardware at this commercial level is only available from Microsoft, the device offers formidable gesture and facial recognition with up to six individuals tracked at once, two of which 48 individual joint positions are recognised [28, 63]. Limor Fried (founder of Adafruit Industries) stated that the Kinect would take "thousands of dollars, multiple Ph.D.s and dozens of months," to develop and "You can just buy this at any game store for $150" [65]. Capability of the device also implies that it could be tailored for other purposes such as aiding in education, helping rehabilitation or people with disabilities, and other commercial functions.

2.5 Interactive Rendering

There are many factors to consider when rendering historical 3D data and multimedia. Visual representations of existing or past historical data, such as architecture, is an experience that transcends the visual interpretation and combines many external factors which hold meaning when considering the structure's social and environmental background. Maintaining these non tangible values is often difficult in a virtual reconstruction. However, the more accurate the reconstruction the easier it is to convey the structure's narrative thus providing a deeper and more authentic impression. On the other hand, it can be misleading to the viewer if the visual quality is aimed

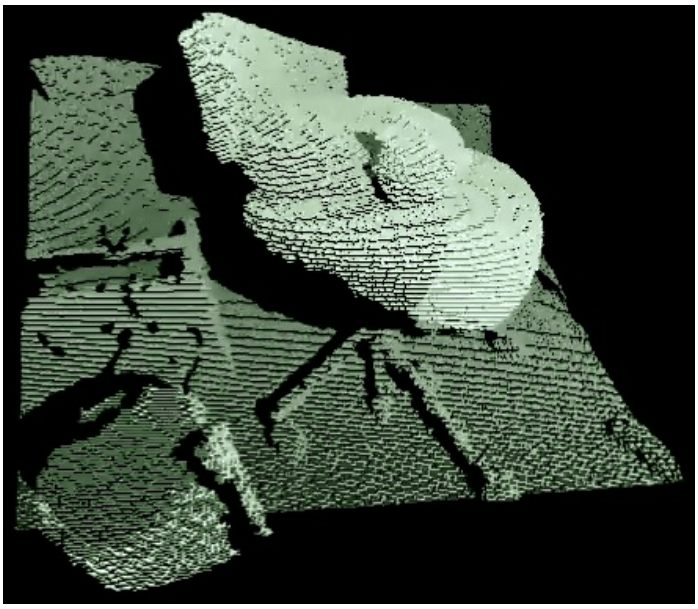

Figure 2.17: Depth information from the Kinect displayed in real-time [2]. The subject includes a chair with a sleeping cat, surrounded by various objects. Each pixel can be seen returning the correct depth information with the chair presented in the foreground.

towards photorealism. Photorealism may give the viewer the impression that the visualisation is exactly how the historical site looked at the time viewed [71].

It is considered that a frame rate of 60 (Hz or frames per second) can justifiably be called interactive. Values below start to become apparent to the viewer, on the other hand anything above is wasted due to our inability to process visual information at higher rates [14]. Virtual historical reconstructions depend heavily on the validity and accuracy of the historical data. There are various methods of data collection, especially when considering 3D data, which offer real-world accuracy in terms of geometric detail. However, some methods offer highly detailed results that are applicable for offline rendering but unsuitable to be used in a real-time environment. Specialised techniques have been developed especially for architectural documentation purposes such as laser scanning (for the exterior and interiors of architecture)

and LIDAR (Light Detection And Ranging) which is used to construct geological information over large areas. Furthermore, the information produced by laser scanning technologies is extremely high resolution, resulting in extremely detailed 3D models. Both Ashley et al.[48] and Stumpfel et al.[56] reconstruct degraded historical sites in 3D form, such as the Parthenon, Greece, with laser technology. Importantly, they both recognise the potential in 3D documentation as the reconstruction of segments could be reformed to create a complete understanding of the original structures.

Diagram 2.18 is an example of laser scanned geometry from The Odeion of Gortyna [48]. A novel method of constructing archaeological sites is discussed by Guidi et al.[25] who demonstrated how multiple technologies such as radar sensors, TOF laser scanning, triangulation range cameras and close range photogrammerty) can be amalgamated to aid in the reconstruction process. His findings showed that the combination yielded efficient results, providing more complex geometry in areas of interest, such as statues and relief modelling, and simple geometry for larger structures such as walls, columns and arches. On the other hand, 3D data can be created via more conventional means such as hand modelling (CAD), using architectural blue-prints, images and other tangible resources, as shown by Sundstedt et al [57]. Furthermore this technique lends itself to real-time rendering as conventional modelling methods give control over the complexity and consistency of the 3D data.

An important area in historical visualisation is what should be displayed and when. This applies in two respects, firstly it is common that the acquisition of data is undertaken after important structural elements have been removed, are missing or have been destroyed thus data needs to be manually constructed. Secondly, if a visualisation application is to accommodate large numbers of 3D data, measures need to be taken to accomplish interactive rates, which is discussed in Section 2.6.

Figure 2.18: A partial 3D reconstruction of The Odeion of Gortyna using laser scanning technologies.

Temporal and structural uncertainty is a prominent problem in historical data. Time is notoriously difficult to handle when visualising structures as their integrity and construction may have changed radically without record or clear reason in times such as war. Handling time based data sets is important due to the user only being able to view the architecture/object from a single point in time. These data sets can be overlapping, where uncertain structural additions are present in a particular time period. Thus enabled the user to decide the time period viewed and for content to be changed dynamically. Uncertainty also contributes to the way in which data sets are rendered; ncertain or incomplete data sets such as structural data can be shown to be uncertain using visual cues. Methods to display uncertainty have been researched by Sundstedt et al.[71] who experimented with rendering particular parts of the geometry with different degrees of transparency according to the certainty of an artefact. For example if an artefact had a certainty probability of 0.2% the items transparency could be rendered with this figure. Furthermore, other visual cues such as blurring, depth of field, pseudo colours, contour lines, blinking materials, wireframe

and sketch like qualities can be used as visual indications. Geometrical techniques can be employed such as rising and sinking, representing the building's construction and demolition according to the time period visualised.

2.6 Efficiency

Efficiency in interactive real-time systems is of up most importance when considering frame rates and user interaction. In a virtual environment the complexity of the scene directly relates to visual performance. Furthermore when rendering hundreds even thousands of buildings for a cityscape, equally processing billions of triangles in an intricately constructed and highly detailed model, forethought must be placed in techniques that aim towards discarding geometry from the rendering pipeline when required. There are many techniques that are applicable for historical visualisation systems which accomplish both occlusion and culling. Sections 2.6.1 describe some of these techniques and their validity.

2.6.1 Visibility Culling Techniques

There are particular aims that occlusion and culling techniques try to satisfy, but all attempt to shorten render time and increase efficiency. To briefly summarise, the fundamentals are to exclude any primitives from the render pipeline if a visibility check returns false. In the context of view frustum culling, if any item is outside of the frustum bounding box it is considered not to be visible to the viewer.

Occlusion Culling

Peter Wonka et al. [64] concentrate on visibility pre-processing, in detail they have produced a visual culling method which offers real practical application in the context of large, open scenes. In essence they aim to solve situations in which large areas of

occlusion can be culled due to overlapping, or occluder fusion (the occlusion that occurs from the summation of occluding objects). Peter Wonka et al. noticed that the inaccuracies that occur with occluder fusion could be remedied with the use of pre-processed sampled areas and the process of 'shrinking' occluding objects to enable sampling visibility from the interior of the sampled area, in which the graphics hardware plays a large role.

Efficiency is gained by using previously developed technologies that are in essence purposed for different results. Speed is gained by using the graphics hardware capabilities, shadow mapping is a common computational task that is aimed towards generating areas of shadow in real-time (discussed in section 2.7.1), the technicalities behind generating such shadows involve the z-buffer. Furthermore the technique is making use of data that is already present during the rendering phase of computation. A two-pass technique is used to successfully cull objects.

To generate the sample point, the orthographic projection of the scene is rendered into the z-buffer which creates the cull map. The visibility for each cell can then be determined by comparing the z-value in the cull map to the z-value of the bounding boxes of each object in the scene grid at that cell.

The general idea behind occlusion culling in urban areas is to reduce the number of items that are rendered before the rasterization phase. Laura Downs et al. [16] took a similar approach to [64], but rather than generating cell occlusion maps, occlusion is accomplished by generating an occlusion horizon in which object could be removed from rendering. The technique is based on a projected horizontal view and assumes that most occluding objects can be described by a height field. The cull horizon is

generated by traversing a plane from eye to the horizon extent (front-to-back). Objects that are not obscured are added to the rasterization list, then to the occlusion horizon, offering an extremely fast method of quickly creating occluder fusion and removing all objects occluded from the user's perspective.

Although efficient, there are requirements that stated each object to be considered for occlusion was evaluated by its CVP (convex vertical prism), meaning another step was required if these were not explicitly provided. Furthermore, a very present limitation to this technique is the inability for vertical accent. The algorithm is defined within the natural limitation of a human, for example an urban drive through.

2.7 Scene Realism

Realism is relative to the scope of the project and the individual's interpretation of visual information. The stage at which the realism of a virtual environment is decided is at design, this decision greatly affects the methods used to construct, gather and render 3D information. As previously discussed, added realism often means complicated arbitrary geometry which in terms of real-time performance is more computationally expensive, as discovered by Todt et al.[60] who generated models specifically for static offline rendering, similar to static transitional techniques described in Section 2.1.2, which were too complex for a real-time environment.

Furthermore, a scene is considered to be aimed towards realism when visual representations of artefacts, architecture and environments contain real world elements. There are various integral visual cues that can make a scene more realistic, correct lighting and shadows give indications towards a more authentic representation. These techniques are discussed in the next section.

2.7.1 Shadows

The realism of a scene can be greatly increased by the addition of shadows. As shadows are present where light is, modelling a realistic scene without them can produce flat images. Furthermore, shadows can aid in spatial awareness as they produce another layer of detail, giving the viewer information on distances between objects, for example a bouncing ball in relation to the ground and the lighting source. The most typical method of generating shadows in real-time is using shadow maps as they are more suited to hardware processing. A major advantage of shadow mapping is that it places the bulk of the rendering process into techniques that are firmly set in graphics hardware architecture, texture mapping and depth buffering. Shadow maps are adequately suited for real-time applications due to no extra geometry calculations.

It is generally considered that shadow map techniques are accepted as one of the most efficient and robust means in which to produce shadows in a real-time environment. This is partially because shadow map techniques are directly supported by most graphics hardware (created via depth buffering and texture mapping). Furthermore advanced shadowing techniques can be produced on devices that are not necessarily at the fore-front of their area. The general implementation of shadow mapping is as follows; $p_z \leq shadowmap\,(p_x, p_y)$. Where p defines a point in the light's image space. Then the texture is mapped to the scene according to the viewpoint image plane.

$$\frac{p_r}{p_q} \leq texture2D\left(\frac{p_s}{p_q}, \frac{p_t}{p_q}\right) \tag{2.7.1}$$

The advantage of shadow and depth maps is that it is faster to calculate and process data from multiple light sources. This involved multiple passes, these different passes are described in detail by [51]. Segal et al. describe three passes, hidden surface removal, shadow testing and final rendering. Thus for multiple lights, multiple passes are required, unless differed shading is implemented. Furthermore, the

performance of shadow mapping does not depend on the geometry in the scene and its complexity, meaning highly detailed models do not hinder the shadow process [39].

Aliasing problems are introduced when creating shadows with shadow maps. ASM (Adaptive Shadow Maps) developed by Fernando et al.[18] aim to rectify aliasing issues by adaptive subdivision, hierarchically categorising shadow map segments in accordance to the detail needed, computed from a set of viewer perspectives. Furthermore the technique is view dependent, providing possibilities for both internal and external lighting conditions such as landscapes and cityscapes. This research shows that ASMs are versatile, conforming to the amount of memory assigned and producing crisper un-aliased shadows. Figure 2.19 shows the comparison between conventional shadow maps (left) at 2,046 x 2,046 against Adaptive Shadow Maps (right), effectively at 65,536 x 65,536, both using 18mb of system memory.

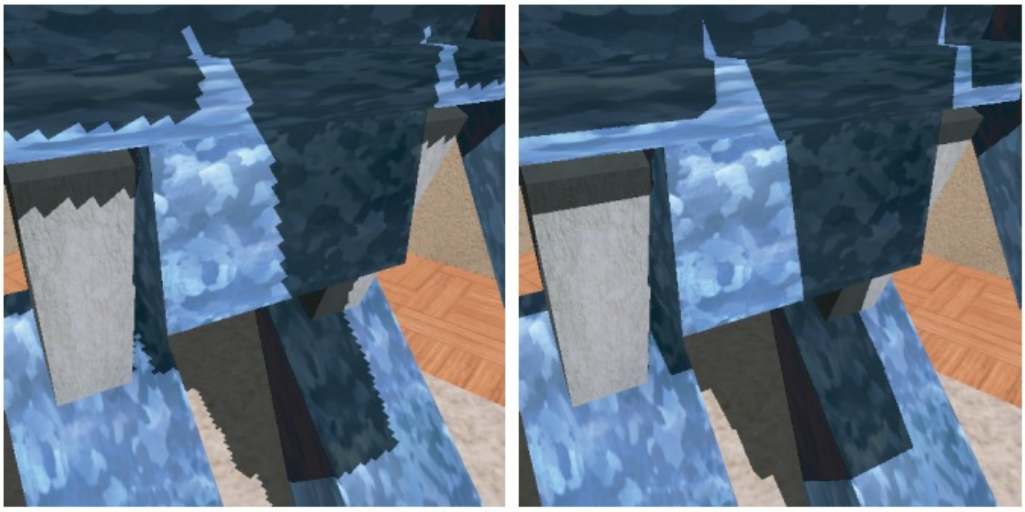

Figure 2.19: A comparison between conventional shadow maps at 2,046 x 2,046 (right), and Adaptive Shadow Maps at 65,536 x 65,536 (right). Clear aliasing issues are present on the left, where ASMs sacrifice a small amount of performance to aid aliasing issues.

Although Fernando et al.[18] have created high quality perspective dependant

driven shadow maps the final result showed ASMs to be more computationally expensive which is unsatisfactory when considering real-time performance requirements. A similar, more efficient parametrization of shadow maps was proposed by Stamminger and Drettakis [53]. Aliasing occurs when the user's viewpoint is zoomed and the shadow map boundary is visible at texture pixel level. The size of the pixels on the occluders surface are at:

$$d = d_s \frac{r_s}{r_i} \frac{cos\beta}{cos\alpha} \tag{2.7.2}$$

Aliasing occurs when the $d_s r_s / r_i$ becomes too large. Perspective shadow maps transform scene and light information into post-perspective space using the camera matrix which is then used to generate the shadow map. Moreover, point lights are generally inverted, creating an orthogonal view and enabling aliasing issues to occur in the distance where fewer pixels are present and detail is not needed.

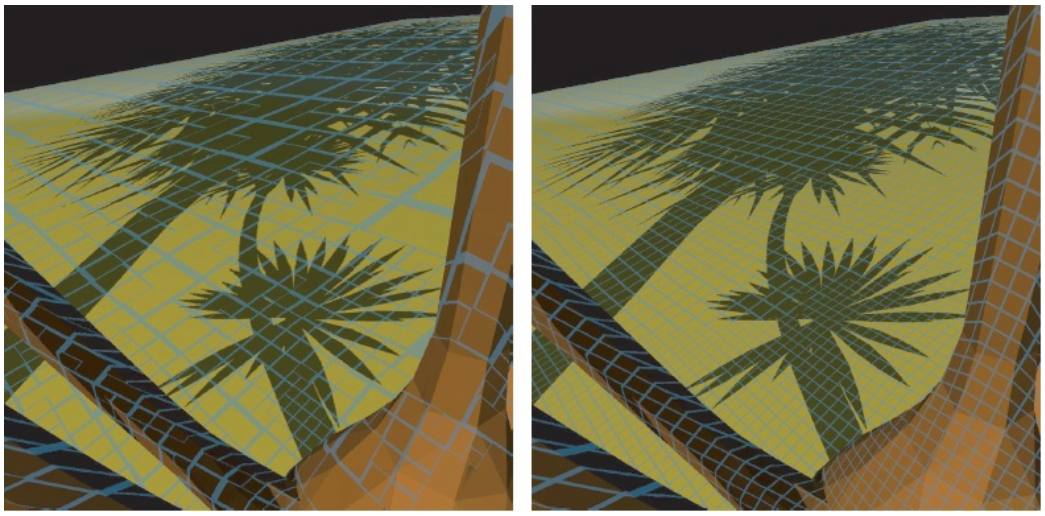

Figure 2.20: A comparison between ASM and RMSM shadow maps. Each square of the blue grid represents 32x32 texels, finer areas are present closer to the viewpoint coordinates resulting in more accurate shadows where needed.

Dynamic scenes are commonplace in computer graphics. Although ASM's improve on traditional shadow maps they cannot handle interactive rates when processing moving objects due to the expensive means in which edge-detection is calculated. Lefohn et al.[33] introduced RMSMs (Resolution Matched Shadow Maps) which improved directly upon ASM's by generating all shadow information in a single step rather than iterating through each page request. Furthermore, the majority of the processing is carried out on the GPU which is optimised for tasks such as geometry and shadow calculations. The algorithm consists of three major steps, firstly all geometry is rendered including shadow information (s, t, z_1), shadow map coordinates and depth information, respectively. Depth information is then transformed into light space. Secondly, a set of shadow pages is created using sort, scan and gather, removing all non-unique elements for compaction. Finally, the quadtree of shadow maps is generated by rendering both quadtree page tables and memory textures. This is then used to lookup shadow information, gathering the highest resolution page that is of equal or higher quality than the request page. Thus, this technique can be used for static scenes such as architecture by recycling previously rendered quadtree information but at a cost of using more physical memory. Figure 2.20 shows both ASM (left) and RMSM (right) shadow mapping.

The addition of shadows aids in the understanding of the scene, especially when a virtual environment consists of many individual items. It provides depth, offering extra positional information which ultimately aids with scene realism and spatial awareness. In the next Chapter, image processing is discussed, describing the techniques used throughout the project including the experimentation and outcomes.

Chapter 3

Image Processing

This Chapter covers the implementation of all technical aspects of the project including image processing algorithms such as hand recognition, hand tracking, gesture recognition and error handling. Also, particular decisions are discussed that were made during implementation in favour of successful results. Later, model handling, real-time rendering and the rendering pipeline are discussed.

When implementing image processing in conjunction with real-time rendering techniques, performance and efficiency are a high priority. Image processing and real-time rendering can be computationally intensive especially in situations where both are required simultaneously, which is becoming a common occurrence in areas such as contemporary computer games, an example being those that use the Microsoft Kinect, such as Kinect Sports [40].

During implementation all aspects of the project evolved according to constant user feedback. As expected, the largest changes were seen in context of all image processing techniques, usually resulting in more robust and efficient algorithms. It was also obvious that there were many methods of accomplishing the same result, which were investigated before implementation. Furthermore, it was also found that there were the possibilities of improvement in many areas; this was due to the field of

image, depth and body tracking consistently evolving under both the academic and public spotlight.

Firgure 3.1 shows the image data flow, including high level processing such as depth thresholding and gesture and finger analysis.

3.1 Hardware Analysis

In this instance two devices were available that provided 3D depth information, the Panasonic D-Imager and the Microsoft Kinect. As efficiency is important, reproducing code for each output device was impractical, so acquiring and working with raw data from the two devices involved studying their image output specifications at a low level, thus enabling an understanding of each devices output data structure.

As both devices varied in image depth size, the code was written to handle different input data types but was also efficient in recycling as many algorithms and as much code as possible. Advantages were gained from adaptive code for both devices as results were consistent, furthermore pointing towards expandability to other devices.

	Panasonic D-Imager	Microsoft Kinect
Image Size (W x H)	160 x 120	640 x 480
Range (meters)	1.2m - 9m approx.	1.2m - 3.5m approx.
Horizontal Field of View	60°	57°
Vertical Field of View	44°	43°
Framerate (Hz)	30	30
VGA Output Type	8-bit grey scale	8-bit colour
Depth Output Type	11-bit	16-bit
Depth Technology	Infrared	Laser

Table 3.1: Panasonic D-Image and Microsoft Kinect hardware comparison table.

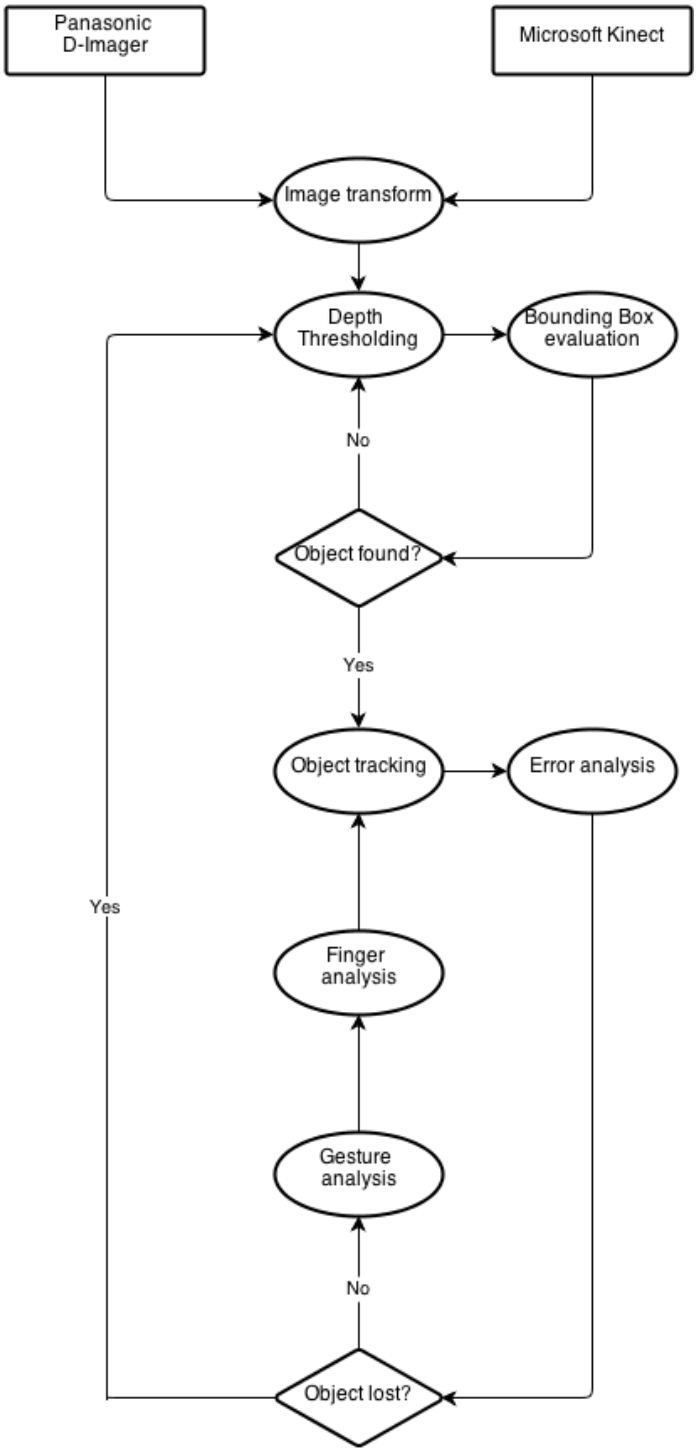

Figure 3.1: This Figure provides a high level overview of how the image processing is initialised and processed, showing the data flow from either device and the object tracking, finger and gesture analysis.

Both devices aim to provide image data that is purposed for use in varying situations and both use different technologies. The D-Imager acquires depth by measuring the time between emission (via infra-red LEDs) and reception (infra-red lens), also known as the time-of-flight (TOF). This time is then analysed, representing the distance between camera and subject. The technology lends itself to real-time applications as the entire scene is processed at once. However, there are major issues involving the device's LED signal strength and false positives. For example, artificial illumination emits relatively low levels of infra-red light compared to natural light, which contains far more (approximately 1 watt and 50 watts per square meter, respectively); therefore is easily interfered, disrupting depth values and returning incorrect readings.

Microsoft opts to use a different method of spatial analysis. By design the Kinect uses an infra-red laser projector, a technology that provides robust readings and overall superior result over the D-Imager. The 'Time coding' technique makes use of the projected pseudo-random infrared spots by using two depth images, one infra-red view of the target spots and the other is the hard coded virtual pattern projected into the environment. Furthermore, the physical distance between laser projector and infra-red CMOS sensor differ, so each image (real and virtual) are specially engineered to be offset; with this information stereo triangulation is used to calculate each depth point according to the disparity in the images.

This technique offers clear advantages. Importantly it is not susceptible to environmental effects such as natural infra-red sources due to the laser projector producing a high intensity dot field which yields clear points of interest. On the other hand there are potential issues with pseudo random points; information is attained from a pre-defined number of infra-red laser points rather than infra-red waves as an entire

image, theoretically meaning fine data could be lost as image data at a pixel level is created with depth smoothing (the interpreted values between each point).

Initial examination of the technical specifications showed that there was a large discrepancy in image resolution and output format. Three conclusions were made from these observations; firstly the Kinect provides a larger image to work with, thus offering more detail at greater distances. Secondly, the D-Imager only provides an 8-bit grey scale image, eliminating the possibility of colour processing techniques. Finally both RGB and depth data is structured differently where the D-Imager produces 11-bits and the Kinect produces 16-bits, therefore when working with data streams of different bit values processing must conform to both.

3.1.1 Image Acquisition

The first phase of developing an image processing library that was not tied to specific hardware was creating a basic foundation on which any device could be used. This involved experimentation with different capturing methods. As the Kinect was not officially supported for software development in a Windows environment at the time, exploration into third party libraries was required. On the other hand, the D-Imager was supplied with a simple means in which to access raw image and depth information thus requiring no other intermediary platform. Project requirements meant access to raw information from both cameras was vital, therefore a third-party library was required to ascertain flexibility with the Kinect.

After researching potential libraries two were chosen for testing, OpenNI and

OpenKinect, the latter being open source. Firstly, OpenNI was founded by the designer of the Microsoft Kinect PrimeSense. Core functionality is based on the combination of individual nodes, each node offering access to the cameras data streams (colour camera, depth information and sound). After experimentation it was found that even though more advanced techniques are included such as tracking and full skeletal recognition, only PrimeSense devices were usable. In conclusion, functionality was varied but flexibility was minimal meaning separate algorithms would have had to be devised for both devices, thus being impractical.

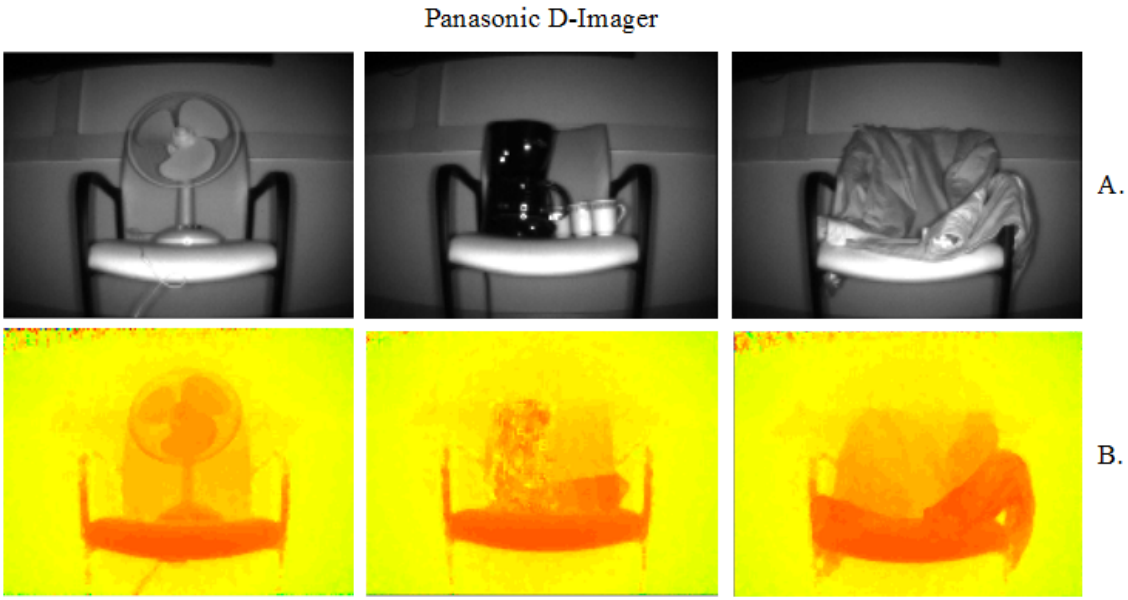

Figure 3.2: Tests were carried out on different material types approximately one meter in distance to analyse how both devices reacted. This diagram shows both RGB (A) and depth images (B) from the Panasonic D-Imager. Various materials are present including plastic, glass, ceramics and leather (from left to right).

Secondly, OpenKinect is an open source library that provides basic interaction with the Kinect. The core functionality is restricted to accessing data streams, but it enables access to the raw data values which was essential, therefore making it the better option. Below are the initial images taken from both devices. This was a

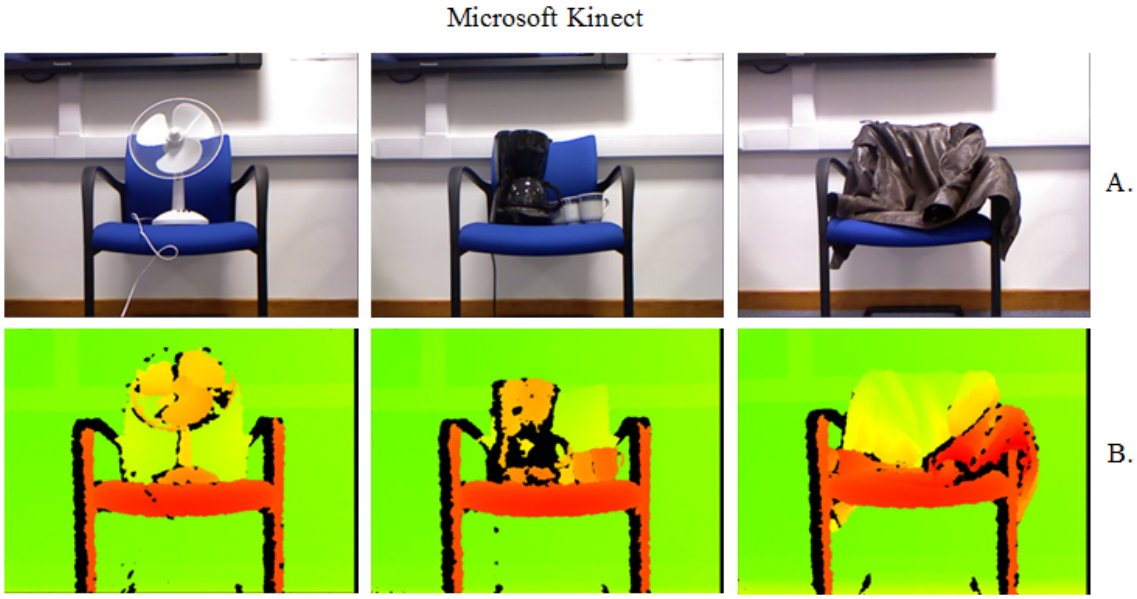

Figure 3.3: This diagram shows both RGB (A) and depth images (B) from the Microsoft Kinect. Like the previous diagram, various materials are present including plastic, glass, ceramics and leather (from left to right).

preliminary test to analyse image quality and to become familiar with the limitations of both.

The screenshots in Figure 3.2 and 3.3 show both RGB (image set A) and depth images (image set B) from both devices; the depth information has been processed into a colour image (16bit to 8bit conversion). At first, an RGB scale conversion was used to standardise image data over both devices. Both devices were tested under the same conditions with the intention of highlighting each device's limitations with different materials (plastic, metal, glass, porcelain and leather). It is clear that cloth (right) do not pose a problem to either device. Depth values returned were mostly consistent, even minor details such as fabric folds are detected.

The examples in Figure 3.4 shows the clarity of depth images from both devices, where the colour grading is based on distance from closest to farthest (red, yellow, green and blue). The scene consists of three chairs, each approximately 1, 2 and 3 meters from the devices (1, 2 and 3 seen in B and D). Finer details that are present in the Kinect depth representation cannot be seen in the D-Imager, such as the shelf found on the back wall. It was also clear that negative values, or values that do not and should not hold any bearing on the final depth image are negated completely by the Kinect, where as the D-Imager still interprets these values as valid distances, these errors can be seen in 3.5.

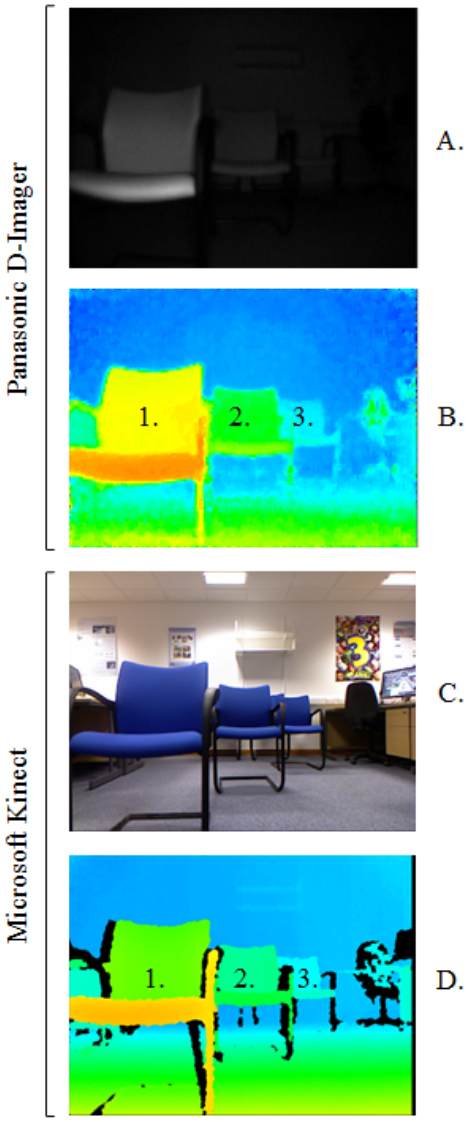

Figure 3.4: Example of depth clarity, where the colour gradients represent distance.

On the other hand, major discrepancies became apparent between the two technologies when placing reflective and metallic objects in front of both devices. As previously discussed, each technology uses infra-red but in very different ways; reflective objects such as windows, watches, chrome and porcelain are often found in environments in which devices like these are used, therefore it is paramount that signal quality is not material dependant. Material surface becomes

an issue with the Panasonic D-Imager, as glass is both reflective and transparent. Issues occur due to information being reflected and passing through, this is then incorrectly interpreted by the D-Imager which produces random depth values. This becomes even more apparent when the device is situated in a open naturally lit area, Figure 3.5 shows the random values being produced (the multi-coloured scattering) which greatly hinders the consistency of the depth information, almost rendering it unusable without heavy error handling.

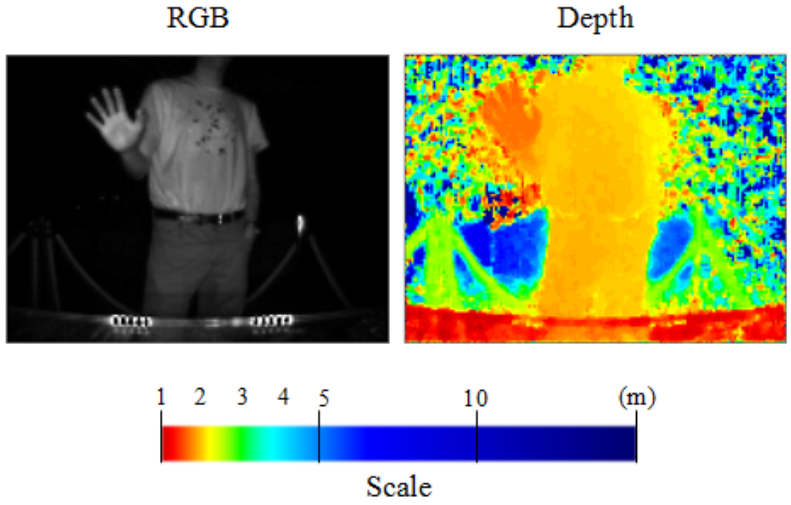

Figure 3.5: An example of the Panasonic D-Imager situated in a large structure mostly constructed with glass and metal. Results are greatly affected by sunlight and reflections from multiple surfaces.

3.1.2 Data Handling

As previously stated, the initial stages of implementation showed that a generic interface was required to interact with both cameras. This intermediary platform consisted of an accessor class based around the device context. Both device contexts were constructed in a similar manner, consisting of standard output calls that in turn filled locally stored variables with new data, if data was available. For efficiency a pointer was returned, enabling the option of direct access or data duplication (which was

more common) for image analysis. Data from each device was supplied with the same number of bits but a different range; both devices supplied a 16 bit stream. Eeach device produced different maximum depth values, the D-Imager ranging between 0 to 1500 and the Kinect between 0 and 2048. Retrieving and storing these data values required a storage type which was an array of unsigned short which holds values between 0 and 65,535.

Image processing is a large area in computer science and has been developed by numerous companies such as Intel, Microsoft and Apple. More specifically, Intel has developed a fully functional open source image processing library in collaboration with WillowGarage named OpenCV. In the context of depth processing, OpenCV does not offer any direct functionality for use with device's such as Kinect, rather it offers general functionality that can aid in processing.

OpenCV was chosen due to its more flexible nature. For example, placing data into OpenCV matrix container's (Mat) did not require anything more than ensuring the containers data pointer was assigned to the devices data values. This technique meant data allocation into each RGB matrix container was extremely fast, but on the other hand it did not work with 16-bit depth values as more processing was required. Thus all function computations had access to both raw data and OpenCV matrix conversions, opening major processing functionality to all image data gathered.

3.1.3 Program Cycle

For clarity the system cycle passes through two states, session in and out. While implementing functionality for both hand searching and tracking it became clear that state-based code segmentation would aid in application definition. The following

algorithm is a high level overview of image processing from start to finish.

Program cycle

```
1: if New data available then
2:   Update local variables with new image data.
3:   Perform depth thresholding.
4:   Convert processed depth map into grey scale image (16bit to 8bit).
5:   if Currently Not Tracking then
6:     Perform hand search on processed depth map image.
7:     if Hand found then
8:       Assign hand rectangle size and position.
9:       Change current session to tracking.
10:    end if
11:  else if Currently Tracking then
12:    Ensure bounding rectangle is inside the depth map image size.
13:    Perform error checking.
14:    if Errors found then
15:      Accumulate error time.
16:      if Error Time ≥ Error Threshold then
17:        Clear memory.
18:        Change current session to idle.
19:      end if
20:    else
21:      Perform mean-shift on found object.
22:      Perform gesture recognition using depth map.
23:      Perform click recognition.
24:      if Click type = Grasp Select then
25:        if Click = Left Click then
26:          Simulate Left Click.
27:        end if
28:      else if Click Type = Hover Select then
29:        if Button hover detected then
30:          Accumulate hover time.
31:          if Hover Time ≥ Hover Threshold then
32:            Reset variables.
33:            Simulate Left Click.
34:          end if
35:        end if
36:      end if
37:    end if
38:  end if
39: end if
```

This logic proved inexpensive to process and furthermore it enabled a quick-in,

quick-out approach, meaning individuals could interact with the system for a short period without the need of image and hand re-calibrations for the next user. They could also deliberately remove themselves from tracking by dropping the hand in use. This was deemed advantageous while watching media as direct interaction is not required.

3.2 Depth Image Processing

The following section covers all areas involved with image processing, including the libraries and techniques used to create an easily modifiable image processing foundation and then continues to explain core algorithms, aimed towards a convenient and expandable image processing library.

Sections are ordered in chronological order; the image processing pipeline has a natural sequence from start to finish which are defined by prerequisites. These include depth and adaptive thresholding, image accumulation, object recognition, hand analysis and hand tracking. To successfully track an object such as a hand, the image first needs to be filtered, removing any unwanted background artefacts. Accepting an object of certain dimensions involved evaluating its temporal size, if it meets certain criteria it is then tracked. When a hand is found it can then be further evaluated in regards to individual fingers for gesture recognition.

3.2.1 Depth Thresholding

Thresholding is a technique that filters a set of data, often discarding values above or below a given threshold and converting an image into binary format. A maximum threshold will return all values that are below the value provided where as a minimum threshold will return all values greater than the provided value. Multiple thresholding involved mixing both the maximum and minimum methods e.g. only returning values that fall between the minimum and maximum.

Early research showed that colour space was not available, this meant that background removal, which relied heavily on colour, could not be used. Therefore an

inexpensive alternative was produced. The same principle can be applied to depth information; thresholding depth data produces consistent results, creating data that is easy to work with.

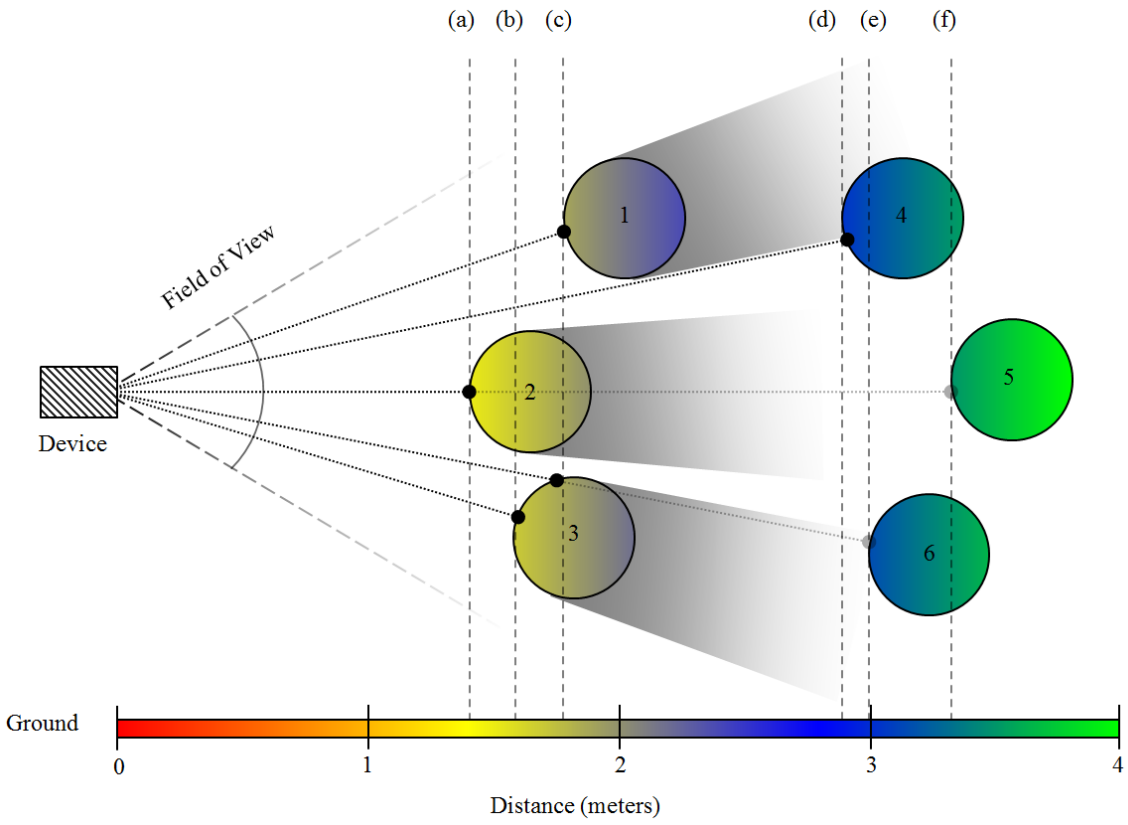

Figure 3.6: In this example situation six objects (spheres) are in front of the camera and within the device's field of view. Each object is at a different distance from the camera, 2 and 5 being the closest and furthest away, respectively. Distance is represented as colour, through red to green.

In Figure 3.6, a to f represent distances between the camera and objects, 1 to 6. Each dotted line defines the distance between the device and the closest part of the object. This differs according to each device and its distance from the object. In this case, if depth thresholding was performed on an arbitrary static range (e.g. between 1 and 2 meters) a slice of the environment would be produced, revealing sphere 2 and 3 in full and a small portion of sphere 1. Governing the distance in which pixels

are accepted enables fine control over image segmentation, providing consistent data for further processing. Furthermore computational complexity is very low meaning a large amount of processing time is saved for more complicated tasks.

Although efficient, there were disadvantages that hindered functionality when considering a context in which many objects fight to be the closest to the device. In a busy scene, the closest object to the device is segmented, working on a first come first served basis, but as to what and where the object is located is not known at this time, thus more information was needed to successfully attain and track the object.

3.2.2 Adaptive Thresholding

As previously explained attaining a statically defined image segment based on depth values was relatively trivial but did not offer much expandable functionality. Therefore an adaptive (or dynamic) threshold was considered to improve depth segmentation.

During experimentation it became clear that unintentional movement away and toward the device was commonplace. In terms of depth analysis, certain issues arose; hand tracking initialisation (explained in Section 3.3.2) was engineered to provide appropriate upper and lower thresholds but did not update in accordance to the user's position. Thus turning away (rotating) and returning or stepping backwards and forwards etc. created special conditions that often invalidated the initial threshold values. The following algorithm aids in consistent tracking.

Depth tracking algorithm

1: **for** $i = 0$ to $History - 1$ **do**
2: DepthHistory[History-i-1] = DepthHistory[History-(i+1)-i]
3: **if** DepthHistory[i] > 0 and DepthHistory[i+1] > 0 **then**

```
4:       Deviation ← DepthHistory[i] = DepthHistory[i+1]
5:    end if
6: end for
```

The algorithm passes values from the front to the back of an array, keeping a history of previous depth values. The average depth deviation is used later for further examination, it is calculated at this point in the processing pipeline to save CPU time;

$$D = \sum_{i=0}^{h-1} mv^{i} - mv^{i-1} \tag{3.2.1}$$

Where (h) defines the number of values and (mv) hold the minimum distance values which are constantly updated each processing loop. The resulting signed $(+/-)$ value (D) indicates the objects movement and relative speed (measured in arbitrary units) towards or away from the camera.

3.2.3 Image Accumulation

The byproducts of thresholding as described in Sections 3.2.1 and 3.2.2 are a lack of spatial awareness and the tendency to analyse objects that are not necessarily of interest. Therefore, this issue was remedied using depth accumulation at a pixel level. The aim was to ignore parts of the environment that were static, or near static. Therefore, all objects that were analysed and found to be the closest to the device were stored in image format for later access if certain conditions were met (such as their physical presence over time). Positional information of objects was essential for assessing image data; this was stored via binary images which could be analysed as a whole to determine if pixels were static over a period of time.

Figure 3.7 shows how depth accumulation works at different distance segmentations (Depth Slice) to produce the binary image, which are then analysed for static object's (Binary Accumulation). In this example only four depth slices have been

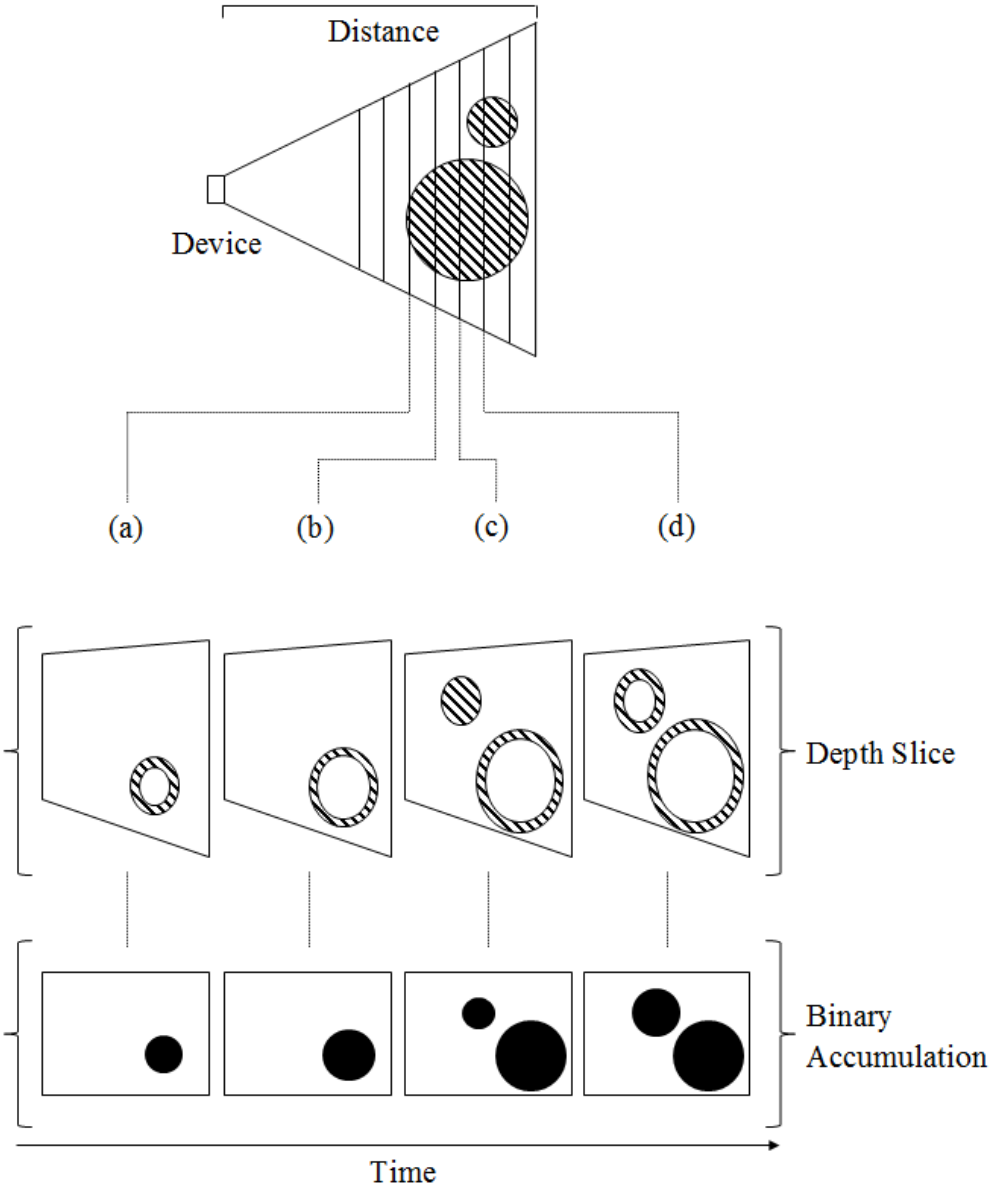

Figure 3.7: In this example situation two objects (spheres) are in front of the camera and within the device's field of view. Each object is at a different distance from the camera, slices a to d are represented by their depth slice, the binary accumulation shows areas in which objects are fixed and recorded over time.

taken for consideration. Furthermore, static object recognition is time dependant; the longer an item is stationary the greater the chance it is added to the binary accumulation image. A pixel satisfies conditions and is added to the binary static map when each pixel in all history images are present and aligned.

Slices a to d mark the progressions in the minimum depth associated with each image processing loop, as object's are recognised as static their associated binary map pixels mark locations in which to ignore. When new image data is available (e.g. stored binary accumulation information is accessed) the depth values which are ignored do not flag as the closest object in the scene, enabling scene evaluation to move on to the next closest object or in this example, the smaller sphere.

$$DE(x,y) = \begin{cases} 1, & \text{if } \sum_{i=0}^{D_{rows}} \sum_{j=0}^{D_{cols}} \sum_{k=0}^{H} H(i,j)^k = 255 \\ 0, & \text{otherwise} \end{cases} \tag{3.2.2}$$

Where each binary image (H) is evaluated (x,y) for either a positive or negative value (positive being black (0) and negative being white (255)). The resulting evaluated binary image contains pixels that represent static areas; these pixels have been determined by positive pixels in each depth history image.

The aim of post processing depth information is to attain a clear picture of the object(s) of interest. When this phase is completed the resulting information can be further analysed for particular hand and finger positions. To accomplish this, object recognition is used to recognise and track specific areas which is explained in detail in the next section.

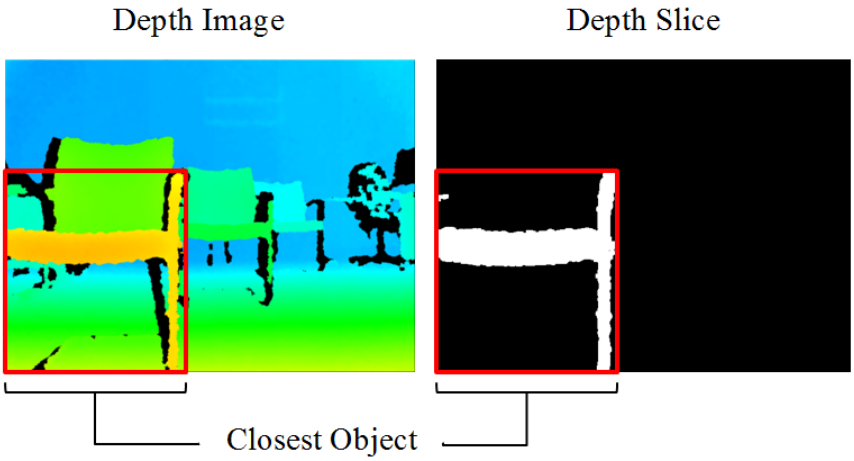

Figure 3.8: As a result of depth slicing the closest object in this scene is initially considered for tracking. The highlighted area to the right depicts the segmented area.

3.3 Object Recognition

Object recognition is an extremely important phase in successfully finding and tracking a hand. Common image recognition techniques can be computationally expensive and sometimes inaccurate; some examples include feature and template-based matching. The basic premise to template matching is acquiring a convolution mask (template) that best resembles the object(s) to be located; it is usually performed on pre-processed images where thresholding or edge detection has taken place. The result is found from the sum of the coefficients, but is an intensive task as it requires a convolution kernel to process each pixel. When completed the positions of the best fit values are retained.

Alternate techniques become a possibility due to the extra depth information; certain issues are circumvented such as complicated backdrops interfering in detection (in reference to the Kinect only), reducing the need for complete feature recognition over an entire image (found in template matching). Depth information can also

be treated as a grey scale map, depending on how the data is handled this can become advantageous, combining the possibilities found with depth information and traditional image processing techniques.

3.3.1 Techniques for Hand Recognition

During implementation many iterations of hand recognition were attempted. Attempts aided in refining a technique that was contextually appropriate, producing the most concise results while not being overly intensive. Initial attempts included frame differencing and depth analysis, both of which used bounding box analysis but analysed image data differently. Initial attempts were aimed to easily activate tracking, creating a means in which to identify the user's controlling hand.

Firstly frame differencing was tested, results seen in Figure 3.9) and show the discrepancies between temporal images. As video data is streaming, immediate changes between frames were calculated with $frame_n \wedge frame_{n-1}$. When discrepancies were prominent enough to analyse, due to a wave, the bounding box was accepted for tracking (bounding box analysis is explained in the next section). Frame differencing supplied regularly consistent results but was greatly affected by a variety of common situations, such as lighting changes, environmental irregularities and the distance between camera and user (it is more difficult to analyse frame discrepancies (such as a wave) when the user is at a greater distance).

Secondly, depth analysis was a progression towards intuitive tracking initialisation as issues became prevalent with image differentiation; the nature of frame differencing required persistent movement over a short amount of time (e.g. a wave), this in turn presented an object that could be tracked. A consequence of motion detection

Figure 3.9: To the left the raw depth image (grey scale), to the right is the frame differencing result of a wave, seen as a pink highlight.

is inaccuracy of the original object shape, therefore while an object may be found its defined bounding box tended to inaccurately describe the underlying object, which caused issues further down the processing pipeline.

The next approach tested took advantage of the discrepancies in temporal depth images. The idea behind this type of depth analysis was treating the provided depth information as a rectangular pressure pad, calculating a synthesised depth evaluation map based on depth values over time. Therefore actions such as a press (in this context a press means a sudden change in depth usually towards the device) could be seen and used for tracking initialisation, thus eliminating underlying issues such as inaccuracies in positional information described previously.

Through testing it was found that initialising system tracking via blanket depth analysis required a significant change in depth over all. Furthermore, a balance was needed between the physical movement required to enable tracking and ease of use; this balance could not be equal due to the unpredictability of the environment. For example, objects rapidly moving towards the camera could be falsely interpreted, because of this accepted depth values had to be relatively high creating situations that

were difficult to analyse.

3.3.2 Object Recognition via Bounding Box Analysis

At this point in the processing pipeline the depth image had been filtered to best segment the hand in focus, or in this case the closest object. Therefore a large segment in the recognition pipeline has been fulfilled; creating meaning from the filtered data was the last step in recognising a potential hand. A few viable methods were available to accomplish this; one of which was previously mentioned, template matching could have been applied to the resulting depth slice but due to the clear results that depth slicing produced it was deemed unnecessary. An alternative technique was developed that took advantage of the pre-processing that had already taken place.

Gaining more information on sliced object's is accomplished with a combination of locating image contours and bounding box techniques. The next stage of bounding box analysis is finding the object's contours using an OpenCV implementation of S. Suzuki border following algorithm [58]. As a result the bounding box can be calculated, producing the tightest possible vertically aligned object area (bounding box) and its coordinates relative to the overall image.

Initial attempts at hand tracking resulted in using the bounding box information directly, tracking the closest object that satisfied a specified area defined by a proportion of the entire depth image. This was the most rudimentary method in which to track the object of interest but contained many disadvantages. For example, consistent depth thresholding coupled with definition of the object's minimal area would provide a size and position, but experiments showed that translation was imprecise

and tracking would occur on unwanted object's. Thus there was a need for control over how, what and when an object was tracked. A few techniques were added between initial object recognition and object tracking to reduce error.

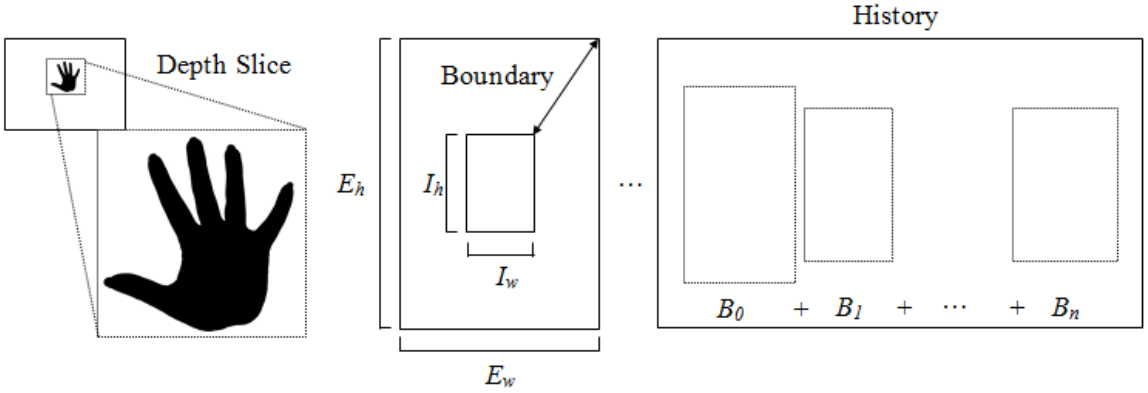

Figure 3.10: This figure depicts the acceptance of an object for tracking. To the left, the minimal bounding box is attained which in turn is saved in the sample history. After n samples, the history is evaluated in context of the bounding box sizes. If the object is consistently available (as the user presents their hand) and is within the dynamic 'accepted' parameters, the object if followed.

To the left in Figure 3.10 the initial depth slice has been processed and the object's minimal bounding box found. Boundary analysis is performed with the intent of discarding object's that do not fall in between specified parameters (minimum and maximum external (Ew, Eh) and internal (Iw, Ih) boundary). These parameters were based on assumptions gathered from observation; a typical hand can range significantly in size and shape, also its orientation is (normally) naturally upright if the subject is asked to hold their palm outward. These limitations aided in determining how data was treated in that the minimum bounding box of a hand would be rectangular. This resulted in increased consistency over what was tracked. To further increase robustness of recognition, an object's bounding box was stored so its history could be considered over time (to the right in Figure 3.10).

The summed difference is divided by the total number of stored bounding boxes and evaluated in terms of average deviation in size and position, thus creating a means in which to check for an object's movement and size over time. Motivations behind periodic bounding box analysis were to enable position relative input (explained in Section 3.3.4). Therefore, the user was able to present their main hand for tracking in a natural and comfy position opposed to instantaneous tracking which caused issues, catching the user unaware.

3.3.3 Hand Tracking

There were two tracking techniques that held preference, both having advantages and disadvantages over one another. The first was using pre-existing data as the foundation for positional tracking information. Data accumulated at this phase in the processing pipeline consisted of the object's position relative to the original depth/RGB image and the minimal bounding box. Hypothetically, this information was adequate for rudimentary object tracking as it is consistently updated every new frame. However, limitations arose during initial experimentation. A resistance to environmental unknowns is paramount when dealing with varied surroundings. A common situation such as a large number of individuals in view could degrade tracking consistency; when users are engaged in interaction there will inevitably be distractions or obstructions. Therefore the proposed tracking method would be severely hindered and susceptible to interference, such as unidentified objects breaking the line of sight between device and tracked object.

For want of a more robust technique, the second method utilised the OpenCV implementation of the mean-shift algorithm (first proposed by K. Fukunaga and F.

Hostetler [23]) in conjunction with various systematic responses to interference enabling a large number of obstructions to be ignored, which meant when tracking was lost it could continue (within a set search period) if the object returned to the same approximate position. This was accomplished by timing the period in which the tracked object was 'lost', when a certain period of time had passed the system would reset and look for a new object to track. If the object returned into the approximate area it was last tracked, tracking continued as normal.

3.3.4 Movement Interpretation

Perhaps one of the most important factors in the user's experience is the way in which their movements are interpreted from hand to screen. Unpredictable movement in two dimensional space can be detrimental to the user's understanding, thus a robust technique was introduced to enable consistent control. First inspection showed that the movement produced by the OpenCV implementation of the mean-shift algorithm required buffering in some form to aid accuracy. Direct movement produced by the algorithm was sporadic, giving a very raw interpretation of pixel movement.

From the beginning it was intended that the tracking system could be used by both left and right handed individuals as excluding one or the other would be impractical. It was also apparent that a hand could vary in shape and size dramatically or even in extreme cases the user could be impaired, with limited extremity movement or irregularly shaped hands. Therefore the ergonomics of system interaction needed to be natural, adjusting itself depending on the user's hand size. In this case, an assumption was made that was relatively consistent; user hand size often correlated to their height and thus to their arm length (although this is not a given). Arm length and handedness directly effects mobility and control therefore, counter-measures were

taken to enable person independent operation. The pivotal extents (PE) in Figure 3.11 are an example of the approximate maximum movement attainable by an individual, its shape is dictated by the ball and socket composition of the arm and shoulder.

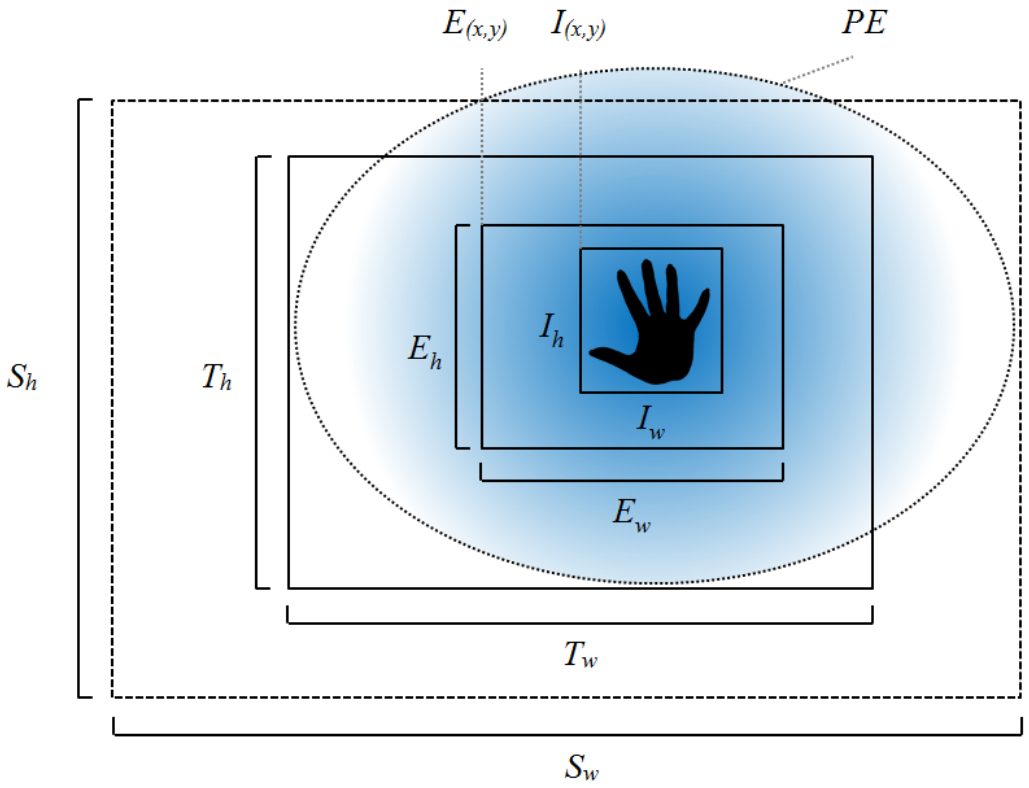

Figure 3.11: This figure depicts all width and height information used to generate accurate movement coordinates according to the hand position. Where $S_{w,h}$ is the screen width and height, $T_{w,h}$ is the image size produced by the device and $E/I_{w,h}$ are the external and internal bounding boxes used for relative positional calculations, determined by the user's hand size. The blue area (PE) indices the approximate area of movement produced by an individual (note the oblong shape).

The combination of development, experimentation and testing showed a clear need for interpreted movement. To elucidate, using mapping between the total image size (defined in Section 3.3.2) and bounding box presented issues when attempting to reach the farthest areas of the screen (PE). This was confirmed via testing with a

few individuals who all showed signs of strain when attempting to reach the opposed sides of their handedness (e.g. a right handed individual reaching to the extreme left and vice-versa). These notions lead to the incorporation of limitations that made it easier to move the cursor within the natural limitations of each user. Figure 3.11 describes the method in which movement was translated. Where $S(w, h)$ is the arbitrary screen size (not to scale), $T(w, h)$ is the image size from the device and $E(w, h)$, $I(w, h)$ are the external and internal bounding boxes, respectively. Each positional anchor is found at the top left corner ($E(x, y)$, $I(x, y)$).

Using the object's bounding box, an external volume is created which represents screen coordinates. This scaled volume is elongated in width and height, multiplied by a percentage of the original with an increased width which is also multiplied by the difference in aspect ratios between screen (S) and image size (T). Movement is then calculated according to screen size, image size and external and internal bounding boxes with Equation 3.3.1.

$$N_{(x,y)} = \frac{(I_{(x,y)} - E_{(x,y)})S_{(x,y)}}{E_{(x,y)} - I_{(x,y)}} \tag{3.3.1}$$

Furthermore, adapting the external volume according to the internal bounding box position enabled the user to relax their posture while being tracked without losing mobility to the screen extents. This was accomplished with a simple box check.

Significant improvements were found when transferring all movement to vector based calculations. To further smooth control, two vectors were used to increment positional data $P^{'} = P_{x,y} + (T_{x,y} - P_{x,y})S$. Where P is the current position and T is the target position. Furthermore it aided in removing erroneous samples from the original movement data.

3.4 Error Checking

When handling streaming data there are usually inconsistencies due to the imprecise nature of image data. Error checking and inevitably error correction are an important part of image processing as they aid in rectifying errors that may slow or even stop application functionality. Errors encountered illustrated these facts and dealing with them was a common occurrence. Furthermore, experimentation with image processing techniques showed that certain variables (such as threshold parameters) could not be clearly defined; fine tuning proved invaluable. To elucidate, as stated previously there are inconsistencies within image data which often required a trial-and-error approach to attain the desired result.

More prominent errors needed attention. These errors consisted of: tracking of incorrect object (or areas), drifting onto undesired areas of the body and inclusion of the wrist during finger detection. Experimentation with the latter is explained in Section 3.5.2.

Error handling algorithm

Require: ProcessedImage, HandRect, MinDepthValues
Ensure: CheckRectWithScreen(HandRect, ProcessedImage)

```
HandTemp ← HandTemp(HandRect)
ImageFill ← 0
Width ← HandRect Width
Height ← HandRect Height
ImageSize ← Width*Height
for y = 0 to Width do
    for x = 0 to Height do
        PixelIndex ← y + x * Width
        Pixel ← ProcessedImage[PixelIndex]
        if Pixel ≠ 0 then
            ImageFill=ImageFill+1
        end if
    end for
```

```
14: end for
15: if ImageFill ≥ImageSize-(ImageSize/5) then
16:    return 1
17: else if Image Fill ≤ImageSize/5 then
18:    return 1
19: else if MinDepthValues[0] ≤ 0 then
20:    return 1
21: else
22:    return 0
23: end if
```

As a concise silhouette of the object had been produced, error checking was made efficient by concentrating on the presence of volume specific to the object's bounding box. As the previous algorithm indicates, dynamic error analysis was based on the positive values found in the cropped image (which is produced by extracting parts of the image that fall inside the bounding box). Each positive value (anything above 0) indicates physical presence, this is then tallied. This final tallied value is a clear indication of the image volume. Simple checks can be performed and the result returned, in order to continue or nullify tracking.

CheckRectWithScreen processes the object's bounding rectangle, rectifying its position if its extent exceeds the total size of the image. Also, *HandTemp* temporarily stores the cropped image to be processed.

3.5 Hand Analysis

Now that a foundation had been created and direct access to both depth and RGB pixels were available, deeper image processing techniques could be designed and experimented with to further increase robustness in hand and object tracking. In this section an overview of hand extraction and segmentation is given with details on the methods behind hand volume calculations, wrist removal and finger analysis. As this

process was important the aims were clear;

- To calculate the structure of the hand using point detection,
- To calculate the hand volume based on circumference and,
- To extract finger tips

There are important reasons as to why particular processing decisions were made. To successfully extract information for hand gesture recognition it is of vital importance that the hand is clearly defined. Therefore, processing has been split into stages to maximise efficiency.

3.5.1 Hand Volume

Upon examination of the structure of a hand, it became clear that to interpret simple gestures, measures had to be taken to understand the movements and actions from a large spectrum of hand shapes and postures. This meant that using a hard coded approach towards hand analysis (such as pre-recorded gesture definitions) would produce inconsistent results from one user to another. Therefore the first step towards hand analysis involved gathering basic information that could serve towards gesture responses.

Calculating the hand volume provided an overview of the hand's current posture. As the final product was to recognise gestures this information became invaluable. There were a few methods in which the hand volume could be calculated one of which had already been used previously during error checking (calculating the number of positive values in the image) but this method was not tailored for more complicated strategies involving positional and distance variables. Therefore an alternate approach was required.

To produce anything meaningful the outline structure of the filtered image (usually a hand) was required. This provided a basis in which to work; using OpenCV's bounding polygon functionality which approximates image contours, returning an array of points according to the largest contour found (seen in Figure 3.13 $P0$ to Pn).

Initial experimentations using the bounding polygon points were based on the angles of each point relative to the next and previous points (seen in Figure 3.12). The resulting angles were then evaluated using on-left calculations indicating line placement (left or right of) relative to the surrounding points. In practice there were too many erroneous values produced due to the analysis of all points; which created problems further on.

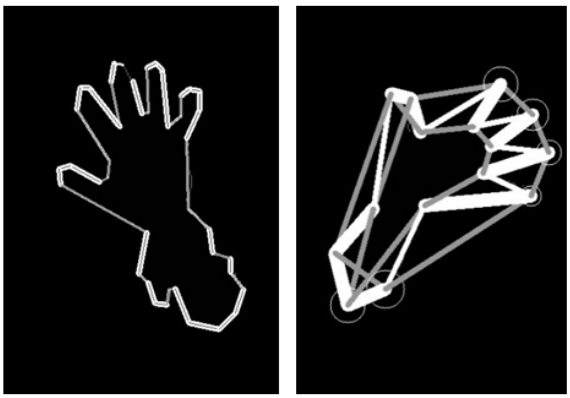

Figure 3.12: These results are from the initial attempts at finger analysis, where each fingertip is circled if found.

Moving towards a more practical method, analysing the circumference of the hand yielded satisfactory results. Keeping the principle that the volume represented a relatively robust indication of the hand's posture the circumference of the hand enabled measurements to be made against all points in the contour array. Figure 3.13 is an example of the approximated contour (represented by the blue lines connecting each point in the array) and both inner and outer radii ($R1$ and $R2$, respectively) which are used for later processing. The radius is measured from the centre of mass to the farthest point in the contour array (in this case the thumb produces the largest distance). Examination into the possibilities offered with the current data revealed

techniques by which to categorise hand postures.

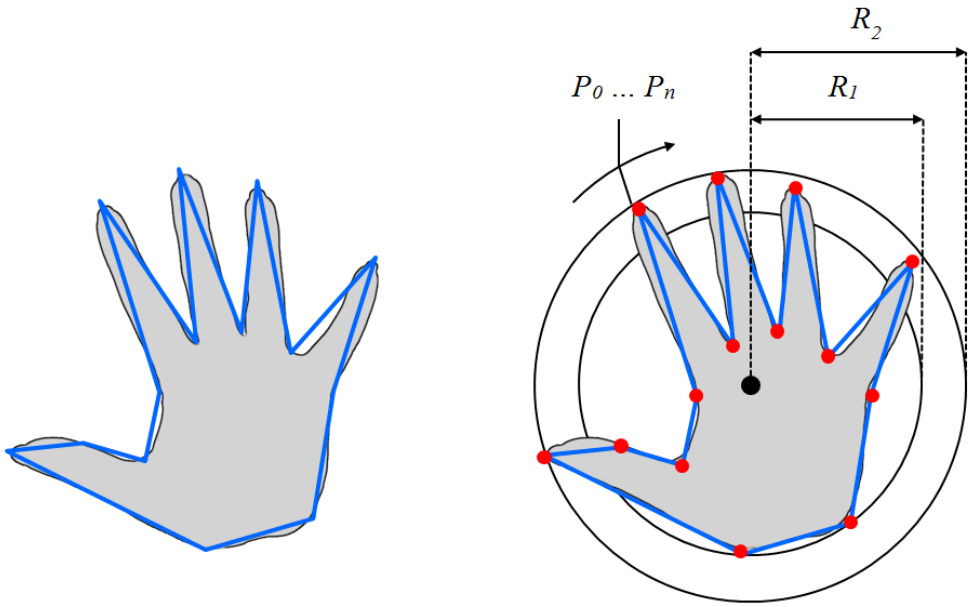

Figure 3.13: This figure depicts the method in which extended fingertips are detected. The circumference of the hand is found from the array of points generated from image analysis. Where P_0 to P_n are the bounding volume coordinates and R_1 and R_2 represent the inner and outer radius which are used for initial point evaluation.

3.5.2 Finger Analysis

To accomplish gesture recognition, finger processing was required to further segment the hand. To successfully interpret hand postures finger movement did not necessarily need to be tracked on an individual basis but an indication as to the finger state (extended or recoiled) was required (explained in detail in Section 3.5.5). Furthermore there are constants in the characteristics of a hand that can be taken advantage of; the use of fingertips is commonplace, touch and dexterity is attained from the end of the finger thus making it an important area.

With the combination of both hand circumference and the approximated (and simplified) hand outline the finger extents can be located, revealing the potential for

gesture recognition.Figure 3.14 shows five accepted points which lie outside of the inner circumference. These points represent the finger tips ($F0$ to $F3$), the thumb (T), the wrist (W) and the centre of the hand (C). This was an important discovery as it clearly produced the layout of the hand and more importantly, extracted fingertip positions. The following algorithm snippet describes the process taken to detect fingertips;

Fingertip algorithm

1: InnerCircumference $\leftarrow Radius - (Radius/4.5)$
2: **for** $i = 0$ to $NumberofPoints$ **do**
3: DistX $\leftarrow$ CentreX - Point[i]X
4: DistY $\leftarrow$ CentreY - Point[i]Y
5: Distance $\leftarrow \sqrt{(DistX * DistX) + (DistY * DistY)}$
6: **if** Distance $>$ InnerCircumference **then**
7: FingerCount $\leftarrow FingerCount + 1$
8: **end if**
9: **end for**

The function analyses each point in the array, calculating the distance between the centre ($Center(X, Y)$) and current point ($Point(i)$). This is compared against the new circumference ($InnerCircumference$) which is a percentage of the original (just under half (4.5)). If a finger is found the tally is increased ($FingerCount$), after all points have been processed the resulting tally indicates fingertips found.

Using multiple circumferences gave advantages while processing different hand types. As the size and shape of the potential hand is unknown, processing the hand against circumferences attained from initial recognition meant a balance between system adaptation and processing ease.

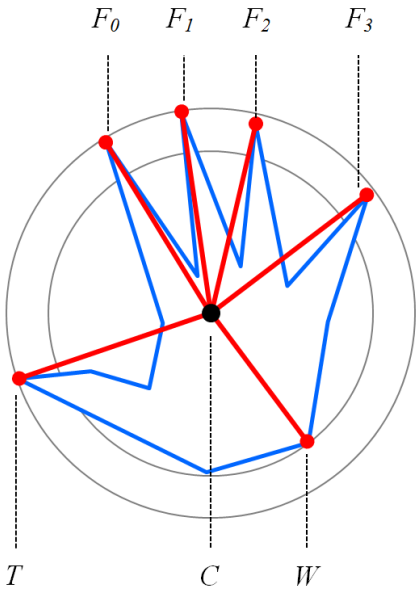

Figure 3.14: After evaluation of each point in the contour array, the furthest most points are considered to be 'fingertips'. Where F_0 through F_3 represent the four fingers and T, C and W represent the thumb, hand centre and wrist. Although robust, results did vary; on occasions it was found that not all fingers were detected, as well as the wrist not being defined.

3.5.3 Adaptive Volume Response

Upon initial implementation measurements were taken to identify fingertips from the moving image. As previously described this was enabled by calculating the outer circumferences followed by the inner circumference. Different factors influenced how the inner circumference was treated, predominantly the finger arch positions and finger length. To elucidate, the length of an individual's fingers can vary dramatically, so to detect all fingers successfully a balance was found between inner and outer circumference, the former being a percentage of the later. Figure 3.15 is an example of a typical hand with a combination of extended (thumb, index and middle) and semi-recoiled (little and ring) fingers, resulting in large discrepancies between outer circumference and shortest finger.

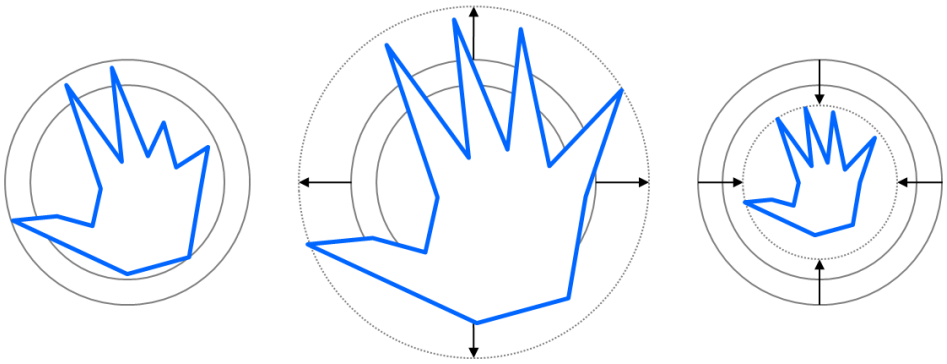

Figure 3.15: This diagram gives examples of likely hand postures. From left to right; a hand with partially retracted fingers, this could be in any combination from the index finger to the thumb. The centre shows the arm extending out towards the depth device, resulting in a larger physical presence. To the right is the opposed situation where the arm is retracting towards the user. Each situation requires adaptive volume response to accommodate the changes.

The initial method implemented was based on static readings taken after the hand was found; all gesture response was based on these readings. While tracking a hand movement in all axes is possible thus there was the potential to retract or extend the arm from the body. This created a physically smaller or larger hand (respectively) according to the perspective and distance of the camera. For example Figure 3.15 shows conditions that break the static circumference; the middle extending past the circumference and the second being contained within, in both cases fingertips cannot be extracted. Therefore there was a need to continually update the hand circumference. Figures 3.15 also shows dynamic volume response represented by the dotted circles; each of which are a better basis in which to analyse fingertips. A definitive decision was made to use adaptive volume analysis which is explained in the next section. Figure 3.16 shows a real-world implementation example taken in real-time, where A, B and C are examples of different scenarios.

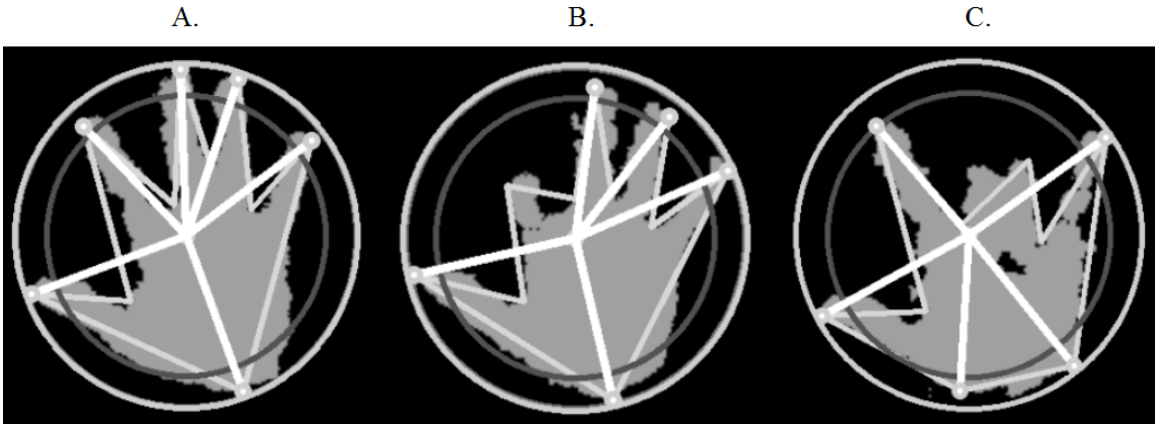

Figure 3.16: These figures were taken from the real-time output. Where (A) shows all fingers extended, the middle (B) shows the index finger retracted and the right (C) shows the middle and ring finger retracted.

3.5.4 Wrist Removal

Throughout the hand segmentation process the desired result was to attain a clearly defined binary interpretation of the hand (or object) to be processed. Furthermore, areas of the extremity that are unwanted such as the forearm and wrist need to be removed if still present before the final stages of segmentation (finger and area analysis), therefore steps need to be taken to threshold the area of interest.

Figure 3.17 shows the final thresholding applied to the depth map that attempts to remove the wrist area (W) to leave the rest of the hand clear for processing (H). The area in front of the hand (E) is considered to be viable space for fingers to move into if the hand was in a more relaxed position; this area is still processed as part of the hand but is ideally the 'excess' area.

3.5.5 Gesture Recognition

The next logical stage in autonomous gesture recognition is correctly interpreting the posture of the tracked hand. The relationship between the individual's intentions and

Figure 3.17: This figure shows the dissection of the hand to the wrist where E is empty space for parts of the hand to move in to, H is the hand and W is the wrist.

system response need to be intertwined, as it is paramount that the user's intentions are correctly responded to.

It became clear that correctly intercepting complicated hand postures (gestures) was a difficult task due to the wide variation in hand types and system understanding. For example, if a user were asked to follow instructions that described various controlling gestures (such as click, drag, scroll and zoom) their understanding and physical action is likely to differ to the next person. This was taken into consideration; as a result it was decided to limit interaction functionality to favour ease-of-use although potential functionality could be (and for experimental purposes, was) extended.

As a result of continuous testing and adaptation, various gestures based on the techniques implemented showed that clicking (simulating the left and right mouse button), scrolling and zooming were possible.

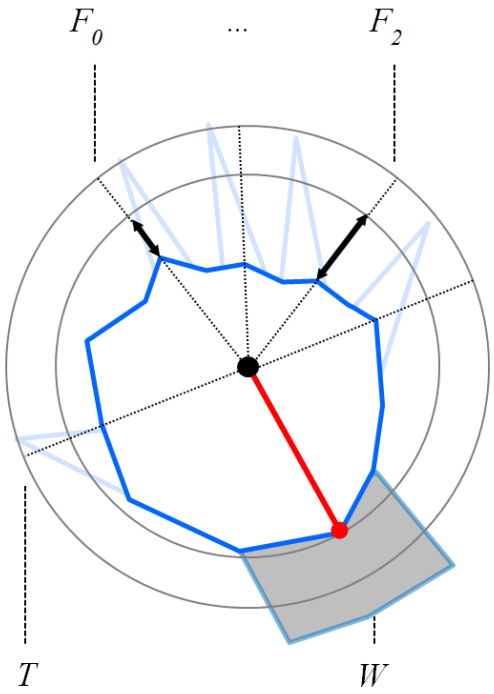

Figure 3.18: In reference to the Figure (3.17) which shows the detection of all fingertips, gesture recognition is accomplished by accepting or rejecting points that are inside or outside a pre-calculated circumference. In this case, all fingertips are retracted - representing a 'grasp' like action.

Click gesture; the physical action required to click or select an item changed through experimentation. Like all gestures, many iterations were created using a combination of variables, such as the number of recoiled fingers and timing controls. Figure 3.18 is a sample of the system interpretation when all fingers are recoiled and the hand is clenched. All fingers and the thumb are contained within the inner circumference and the wrist removed (W) with the technique proposed previously, indicating a grasp which can be translated as a click.

Zoom gesture; as the distance between the device and the centre of the hand is consistently updated there is opportunity to use this information for zooming in and

out. Two thresholds are all that is required to register zoom commands, if the threshold is passed zooming occurs.

Scroll gesture; as the hand is tracked gestures can be intercepted while the object is in motion, if a gesture is recognised (such as clicking) it could be translated differently if movement occurs. Therefore scrolling up down, left and right is attainable; while the hand is clenched movement is directly translated into the equivalent of a scrolling action (simulating a two-directional scroll wheel or ball). The amount of movement was governed by the distance between the initial grasp and new position; two approaches were feasible using this technique. The first uses the movement values to directly manipulate selected objects; the second uniformly scrolls the screen according to the vector direction between the two positions.

3.5.6 Alternative Selection Techniques

During development, experimentation and public testing it was decided to test an alternate method of selection. This decision was made to help improve system flexibility and provide more comparable methods for testing. The first technique introduced was termed the 'physical' selection technique, this involved the user performing a grab which was then interpreted as a click. The second and new technique introduced was termed the 'hover' selection technique.

To elucidate, the user controls the movement of the cursor by direct interaction, as they move over interactive user interface elements (buttons) they are presented with a timer. A blue circle slowly completes as the selection is made, when the circle is complete a selection is made. Figure 3.19 shows a selection being made on the

information button.

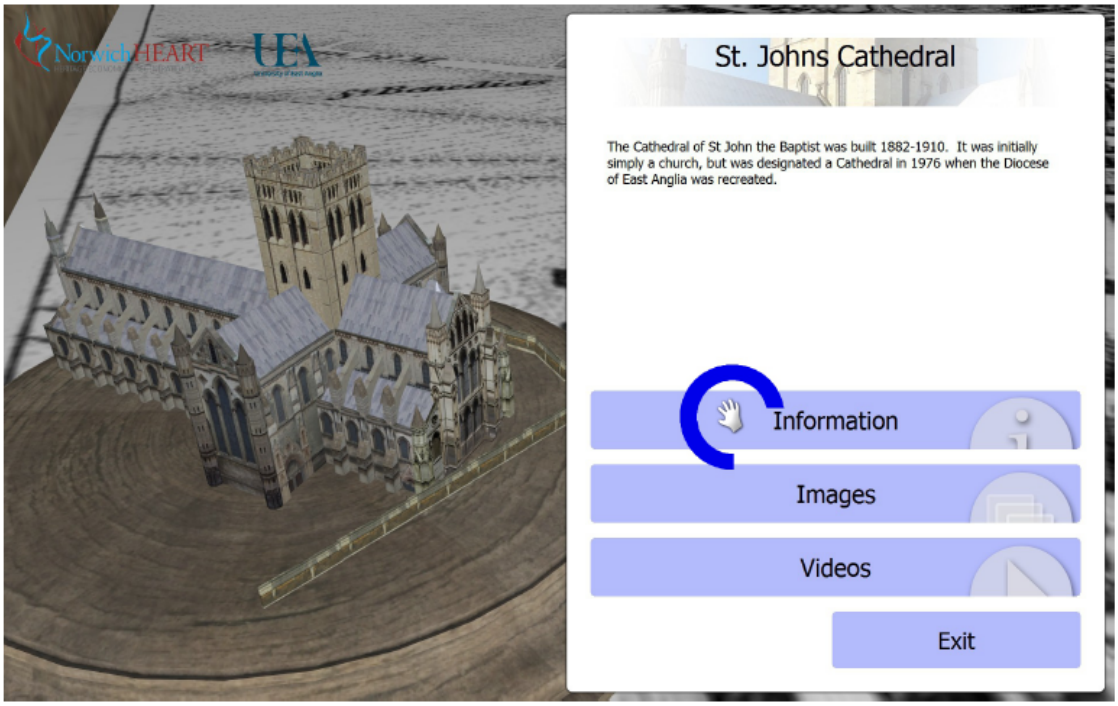

Figure 3.19: The Figure shows a user interacting with a button on the user interface. The blue circle indicates the time passed while on the item. When the circle is complete, a selection is made.

This chapter presents the process which was used to find, track and interpret simple gestures from a hand. It also presents the algorithms used to remove errors that occur. In addition, evidence is presented reinforcing the reasons why certain approaches were taken, such as radii being used instead of angles for finger detection. The next chapter details the visible side of the gesture system. This includes all rendering, efficiency and GUI (Graphical User Interface) aspects, all of which are extremely important to the user's experience.

Chapter 4

Rendering

From the beginning the final product was always intended to be presented in real-time. Alternative methods were available that could have reduced implementation complexity but would have been more restricted in terms of functionality compared to a real-time 3D environment. One possible alternative could be to present information without real-time data, keeping all interaction within a 2D environment. All information divulged would be in media form, such as extensive pre-rendered videos, images and text (even original documents etc.) produced to illustrate the structures and their heritage. Although presentation methods such as these would be quicker to produce there would be close similarities to that of a web-based interface, ultimately becoming detrimental to the interactive experience.

There are advantages and disadvantages to handling visual data at runtime; unlike a purely two dimensional interface an extra degree of fidelity is attained, providing a more immersive experience. Furthermore handling visual data in real-time enables a highly tailored interactive environment, objects can be manipulated to suit the current context, buildings can be hidden entirely and objects can be changed or swapped according to the user's actions. On the other hand, there are underlying issues that

can become problematic if disregarded. Usually real-time rendering requires a significantly larger amount of processing over other techniques. As visual information is consistently updated performance can become an issue if rendering is coupled with other intensive tasks, such as image processing. Also, there can be issues with model compatibility; there are distinct differences between real-time and offline rendering, the first requiring less complicated 3D models (a smaller number of polygons) to enable efficient rendering of high frame rates (frames per second). Whereas offline rendering can be produced with extremely complex models and textures as the final product usually consists of static images or pre-rendered video.

4.1 Model Handling

An intrinsic element of creating an adaptive model loader was choosing a commonly used 3D file format that could be used to export all required model details, such as position information and multitextured surfaces. Thus exploration into different file types was required as restrictions were present; all models required exportation into a common format, even from different 3D modelling packages which in this context consisted of Autodesk 3Ds Max and Maya. These requirements narrowed the selection to Autodesk Interchange File (.fbx), 3D Object File (.obj) and Legacy 3D Studio (.3ds), all of which satisfyed model storage requirements. Analysis of these three file types made evident that both .fbx and .3ds formats were not appropriate for the project. As all model data was to be read manually into the application it was chosen to use the .obj format as some content was already in this format. Therefore the first step was to understand how information was stored.

It was decided that a contemporary graphics library was to be used for screen rasterisation, OpenGL 4.0+ was a choice that offered most flexibility but had particular specifications in terms of model structure. It was required that all models were

exported in terms of triangles as previous versions utilised quads.

Upon the project outset there were clear ideas of what would be displayed in terms of 3D data, it was intended to render (or have the ability to render) many buildings simultaneously with the option to concentrate on a particular item. This presented challenges; one of the largest influential hurdles was 3D content. Content had been generated from numerous sources, ranging from simplified mock-ups to finely modelled architecture with extensive interiors, often in different states of completion. Therefore a need was created for an efficient but robust rendering engine able to read and display model data from a variety of sources.

4.1.1 Model Structure

With clearly defined goals it was possible to start analysing the data structure found in models, this was important as a greater understanding of the model format later aided in better model representation, which in turn enabled greater efficiency in the rendering of highly detailed models. The general overview of a 3D model consists of three sets of variables that make up a single triangle; its vertex positions, texture coordinates and face normals. Figure 4.1 denotes the construction of an object, on the left is the cube in its entirety, in the middle is a detailed view of the defining elements of a single triangle face. A triangle is constructed with the following components; three vertex positions ($P0, P1, P2$) which form the geometry and physical characteristics such as scale (usually relative to the environment), position and orientation. The second holds a texture coordinate (Tc) for each position (seen on the right), relating to a point on an image. Finally, each vertex uses a normal vector (N), which is used in lighting calculations, collision detection and culling.

There is a clearly defined structure in which the model data is formed that can be

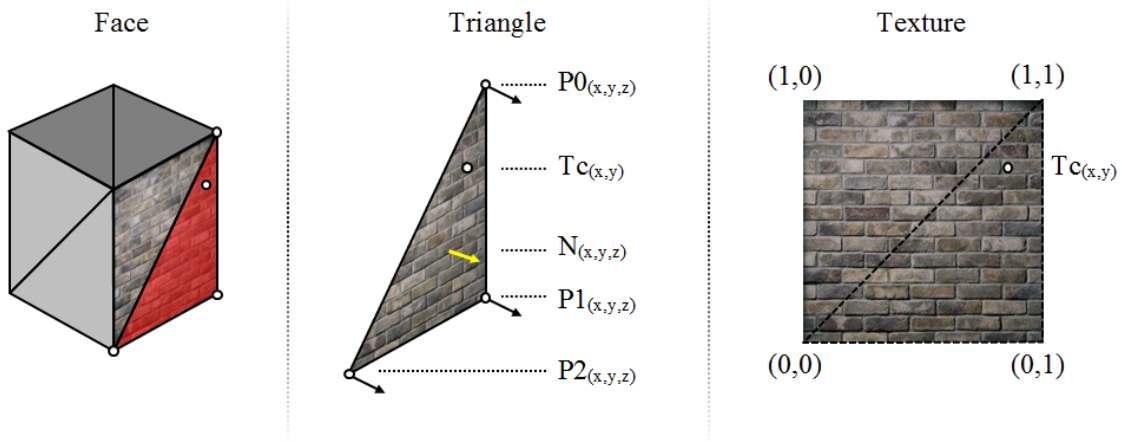

Figure 4.1: To the left the entire 3D object can be seen, the highlighted red area draws attention to a single triangle in the mesh. The individual triangle (centre) is constructed of three vertex positions ($P0$ to $P2$) complete with vertex normal, a face normal (N) and texture coordinates (Tc). To the right the texture map in which Tc directly maps onto the model's surface.

interpreted in multiple ways. Models are split into three levels of detail, the first and highest level describing the model or models as separate or joined entities (which is closely tied to performance, explained in Section 4.1.3), the second defines the geometry with faces or indices, a term commonly used in 3D model data handling. The third and final level details all required variables for each triangle, in this context a face consists of a single triangle. A model can contain a small number of unique values that are shared by multiple faces, vertex sharing is an efficient means in which to save on space and avoid duplicated values, a technique that proved to be invaluable when handling large models, which is explained in more detail in the next section.

In Figure 4.2 each case shows how single and multiple meshes are constructed in one model file. From left to right each mesh contains a different number of individual meshes and objects. Objects can be joined into the same mesh or be separate, therefore when reproducing model data in real-time storing each mesh in reference

to its textures was found to be advantageous. This was due to the smaller number of texture calls, essentially equalling the number of unique textures rather than the number of unique meshes (of which could be many more).

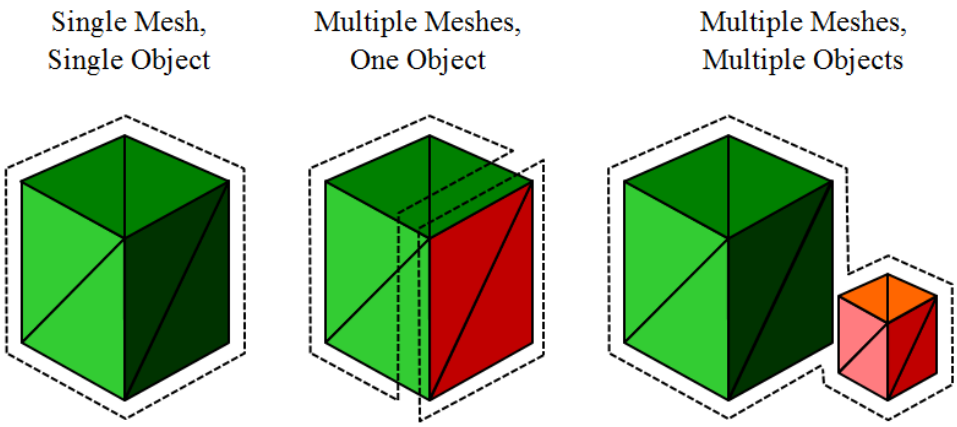

Figure 4.2: A diagram depicting three combinations of mesh possibilities. From left to right, the single mesh defined with a single object, multiple meshes contained as part of the same object (the individual meshes can also be considered by their applied material) and separate multiple meshes contained in separate objects.

4.1.2 Vertex Organisation

As previously stated there is a particular and clear layout that the .obj format follows. Each mesh in the model has a set of unique vertex positions, texture coordinates and vertex normals (v, vn and vt, respectively). These values are usually at floating point accuracy as double precision is usually unnecessary, for example v -10.000000 0.000000 10.000000 (x, y, z). Each face defines a combination of these values, often re-using entries if vertices, normals or texture coordinates are shared. For example (f 1/1/1 2/2/1 3/3/1) creates the first face in a cube, where the order is defined as 1^{st} vertex position, 1^{st} texture coordinate, 1^{st} vertex normal, 2^{nd} vertex position, 2^{nd} texture coordinate, 2^{nd} vertex normal etc., the number is the index to the corresponding value at that line. Note that both faces share the same vertex normal; a simple object like

a cube is a good example of how face information can be interpreted at a low level.

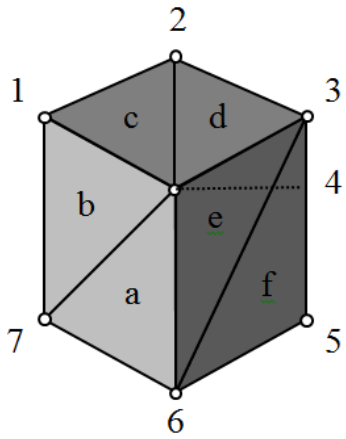

Figure 4.3: Face data is organised into vertices, vertex normals and texture coordinates. Geometry is constructed from an arbitrary number of values where values are shared between triangles. Here 1 to 7 represent the vertices shared that form triangles a to f

.

To elucidate, issues can arise when dealing with models that contain a large number of vertices. Each face in a model is unique but usually shares combinations of vertex positions, normals and texture coordinates. If this is not accounted for model data can become bloated. Figure 4.3 exemplifies a simple cube where a to f indicate each individual triangle that makes the geometry, 1 to 7 represent the vertices that the triangles share. In this example there are a further six triangles obscured from view and another vertex situated diagonally down from the fourth that constructed the rest of the cube. During implementation model loading methods evolved considerably, two notable techniques were tested which directly affected real-time performance. Both used similar principles but one suffered from slower loading times whereas the other demonstrated slower real-time performance. The two techniques were essentially identical, core processing algorithms were shared but a small but profound difference in face-to-vertex interpretation meant great differences were seen

in the total amount of memory used. Due to the nature of the file format it was noted that geometry could be constructed directly in a small number of steps;

- Raw data is stored temporarily in that of the same order found in the file.
- Face data is then constructed using the previously read position, normal and texture data according to the faces defined.
- Data is then stored on the graphics card and removed from memory.

This first technique tended to produce accurate results in comparison to the original model in context of smoothing groups, vertex positions and textures, furthermore models were loaded quickly as no further processing was required. The high level algorithm below describes the processing techniques used; model loading was broken into three distinct sections which run in order of the physical layout found in the files, material loading, vertex data acquisition and face construction. The following algorithm uses the various model loading algorithms (including material and vertex loading) to load model data.

Model loading algorithm

```
1: Create new ModelData
2: if Model file exists then
3:    CurrentMaterialID ← 0
4:    CurrentMeshID ← 0
5:    while End of file not reached do
6:       if Character = 'm' then
7:          Load materials
8:          for i = 0 to NumberofMaterials do
9:             Create new Mesh Data
10:            ModelData Mesh[i] MeshID ← CurrentMeshID
11:            ModelData Mesh[i] MatID ← CurrentMaterialID
12:            Increase CurrentMeshID
13:         end for
14:      end if
15:      if Character = 'v' then
16:         if Second character = 'n' then
17:            Create temporary normal
```

```
18:            Read and store normal values
19:          end if
20:          if Second character = 't' then
21:            Create temporary texture coordinate
22:            Read and store texture coordinate values
23:          else
24:            Create temporary vertex
25:            Read and store vertex values
26:          end if
27:        end if
28:        if Character = 'u' then
29:          if Second character = 's' then
30:            Search existing materials
31:            if Material already exists then
32:              Assign mesh ID to found MaterialID
33:            else
34:              Assign mesh ID to CurrentMaterialID
35:            end if
36:          end if
37:        end if
38:        if Character = 'f' then
39:          Increase face count
40:          Read face
41:        end if
42:      end while
43:      Store model in array
44:    end if
```

An important note is that all model data is created dynamically on the system heap, meaning pointers to information can be passed between functions without the need of copying complete segments of data; this proved advantageous due to the large number of model's and their size in memory. Material loading is the first process to occur, mesh data is then formed according to the number of materials. As each material is unique, geometry is organised according to material, thus multiple meshes in the model can be assigned with the same material ID, reducing the number of texture calls to the graphics card. Details as follows;

Material loading algorithm

```
Require: ModelLocation, ModelPointer
Separate file name from file path
Search for material file
if Material file exists then
    Create new material pointer
    while End of file not reached do
        if Characters = 'new' then
            Scan material name
            Create new material
        end if
        if Characters = 'K' then
            if Second character = 'a' then
                Read and store ambient values
            end if
            ...
            Repeat for 'd' diffuse and 's' specular values
            ...
        end if
        if Characters ← 'm' then
            if Second character = 'd' then
                Load diffuse texture
            end if
            ...
            Repeat for 's' specular and 'u' normal map textures
            ...
        end if
    end while
    Store the new material in array
end if
```

This reduced algorithm shows the steps taken to load all relevant material data, lighting properties (ambient, diffuse and specular) and associated textures (diffuse, specular and normal maps). If any of the three textures are not found the corresponding texture is not flagged meaning it is not used in the rendering process. The final stage is constructing face data with information previously acquired. As previously stated there are two techniques that significantly affect real-time performance. This first algorithm describes face data construction using direct interpretation, creating new geometry for each new triangle, even if data is shared.

Vertex/Face loading algorithm

```
1: FaceCount ← 0
2: PointCount ← 0
3: for i = 0 to number of meshes do
4:    for i = 0 to facecount * 3 do
5:       Current index ← total face count + i
6:    end for
7:    Total face count + = current facecount*3
8: end for
9: for i = 0 to facecount do
10:   for j = 0 to 3 do
11:      Data[Point count] = Vertices[Face[i] Vertex[j]-1] x
12:      Data[Point count+1] = Vertices[Face[i] Vertex[j]-1] y
13:      Data[Point count+2] = Vertices[Face[i] Vertex[j]-1] z
14:      PointCount + 3
15:      Data[Point count] = TextureCoords[Face[i]Texture[j]-1] x
16:      Data[Point count+1] = TextureCoords[Face[i]Texture[j]-1] y
17:      Data[Point count+2] = TextureCoords[Face[i]Texture[j]-1] z
18:      PointCount + 3
19:      Data[Point count] = Normals[Face[i] Normal[j]-1] x
20:      Data[Point count+1] = Normals[Face[i] Normal[j]-1] y
21:      Data[Point count+2] = Normals[Face[i] Normal[j]-1] z
22:      PointCount + 3
23:   end for
24: end for
```

The resulting data is stored in an interleaved array which contains all geometry, texture and normal information side by side. It is generally considered that storing data on the graphics hardware in different formats results in discrepancies in performance. This example shows data alignment as; $VVVTT(T)NNN$, which describes a single face or one triangle (the encapsulated texture coordinate (T) was always assigned to null as all textures were 2D, the option of 3D textures was not used but it is possible). This decision was justified by experimentation; another method attempted was containing all vertex, texture and normal information in individual arrays stored separately on the graphics hardware, it was found that more calls to swap data were required to render geometry, thus combining these arrays only required a single call to

graphics memory. Assessing performance while using unorganised and duplicate data showed that performance decreased significantly when larger models were loaded, this was due to geometry bloating. As many large models were expected to be rendered simultaneously a more efficient method of vertex organisation was developed.

Face organisation algorithm

```
1: FaceVTN0, FaceVTN1, FaceVTN2 ← NULL
2: FaceStructure ← NULL
3: for i = 0 to 3 do
4:    Search for lack of texture coordinates '//' in i
5:    if Texture coordinate not found then
6:       Insert single texture coordinate
7:    end if
8: end for
9: Read face values into temporary faces
10: Search previous indices for duplicate value
11: if Current face index 0 is unique then
12:    Store new Vertex (Vertices[FaceVTN0 Vert -1])
13:    if No texture coordinates exist then
14:       Store Empty Texture Coordinate
15:    else
16:       Store new Texture Coordinate (TextureCoords[FaceVTN0 Tex -1])
17:    end if
18:    Store new Normal (Normals[FaceVTN0 Norm -1])
19:    FaceStructure 0 ← Model TriCount
20:    Increase Model TriCount
21:    Mesh Index ← Face 0
22: else
23:    Face vertex index ← found index
24: end if
25: if Current face index 1 is unique then
26:    ...
27: else
28:    ...
29: end if
30: if Current face index 2 is unique then
31:    ...
32: else
33:    ...
34: end if
```

35: Save new face

This simple but noticeably effective sorting algorithm increases real-time performance as all duplicated values are removed; indexing becomes streamlined as values are cross referenced if found to already exist. Each index is saved using a map, when a new face is found in the model file its indices are checked against previously processed values, if the value is found it is re-used, otherwise the index is new. A brief comparison between the two techniques clearly showed the differences between one another. Table 4.1 shows the results between various models including their final float count, load time and average running speed (frames per second). Models ranged in complexity, thus ranged greatly in the number of vertices they contained.

without Index Sorting			with Index Sorting		
Float Count	Load Time (s)	FPS	Float Count	Load Time (s)	FPS
658,457	0.57496	558	298,153	1.36578	598
1,087,047	0.89519	289	447,372	2.78557	372
1,122,903	0.57496	350	574,029	1.36578	410
3,564,297	1.62563	255	1,233,000	4.54779	345
8,547,725	2.95487	186	5,487,563	8.14458	302

Table 4.1: Load time and performance comparison with and without index sorting.

Examination of the resulting data shows that there is a clear increase in performance after indices have been recycled, although greater performance resulted in longer loading times; in this context increased performance was favourable. Static models could be loaded once at the beginning of the system life-cycle, entities were not required on demand thus no loading was performed during run-time.

4.1.3 Rendering Efficiency

Challenges were presented when intentions for visual fidelity increased; as discussed in Section 4.1.1, there are many aspects to model loading and vertex organisation that affect visual performance. For example, there are processes that take longer to

complete under OpenGL than others, these include vertex buffer object (VBO) and texture switching (the process of switching the currently used objects). Although, these processes are required to correctly render 3D data. Therefore keeping these calls to OpenGL to a minimum was essential.

Furthermore as a large number of complex model's were to be loaded and rendered simultaneously, other techniques could be employed to further aid performance. As previously discussed in Section 4.1.1 there are efficient methods in which to store and reference model data. As each model could hypothetically contain thousands of meshes but only a few textures, model data was stored in context of texture count, which meant that the number of texture calls (texture switching) is reduced dramatically (refer to Figure 4.2).

Another commonly used technique that has shown to drastically improve performance when dealing with a large number of renderable geometry is view culling. To elucidate, in the context of real-time rendering of large urban scenes, during interaction the user is viewing a narrow portion of the scene, meaning a large percentage of objects not currently in view could be excluded from the rendering pipeline. This process is called frustum culling.

There are two types of frustum check that were required to successfully remove items from the rendering pipeline. These included a point check - requiring a single point (x, y, z) and a bounding box check, requiring two points, the maximum and minimum bounding box extents. During application load all items that were to be considered for rendering were provided with a bounding box which could be used for the frustum check. The following algorithms are the simple calculations needed, both

are based on matrices as the frustum is calculated into a 6x4 matrix (representing the six sides of the frustum) for efficiency. Before any calculations can be performed against the view frustum it needs to be updated in case of user interaction. View culling algorithm were as follows;

Single point culling algorithm

```
1: Frustum, CullPoint CullBoxMax, CullBoxMin
2: for p = 0 to 6 do
3:    if (Frustum[p][0] * CullPoint-x) + (Frustum[p][1] * CullPoint-y)
      + (Frustum[p][2] * CullPoint-z) + Frustum[p][3] ≤ 0) then
4:       return false
5:    end if
6: end for
7: return true
```

Where the point ($CullPoint$) is compared against all sides of the view frustum ($Frustum$). The next algorithm checks a bounding box against the frustum;

Bounding box culling algorithm

```
1: Frustum, CullBoxMax, CullBoxMin
2: for p = 0 to 6 do
3:    if Frustum[p][0] * (CullBoxMin-x) + Frustum[p][1] * (CullBoxMin-y)
      + Frustum[p][2] * (CullBoxMin-z) + Frustum[p][3] > 0 then
4:       Continue
5:    end if
6:    if Frustum[p][0] * (CullBoxMax-x) + Frustum[p][1] * (CullBoxMin-y)
      + Frustum[p][2] * (CullBoxMin-z) + Frustum[p][3] > 0 then
7:       Continue
8:    end if
9:    ...
10:   Process all combinations, returning true if any are satisfied.
11:   ...
12:   return true
13: end for
14: return false
```

When calculating the bounding box of an object, the maximum and minimum extents are calculated against each side of the frustum. Generally the computational

intensity required to check an item is less than actually rendering it, therefore CPU time can be saved, resulting in better performance and no visual discrepancies.

4.1.4 Building Description Language

Scene management became a necessity as the amount of associated information grew with each 3D building representation. This meant information had to be linked by a single entity, which held all relevant information. To create the rich user experience that contained both media and information a custom building description file was created (BD) or Building Description. Other than providing convenience, items could be easily updated and edited externally without the need for internal code exchange.

The BD format was designed to cater for the custom requirements of HistOracle. As each building contained rich media, three URLs define the local access points for images, video and information. The structure is as follows:

n Building Name (used as the model identifier)

m Model data location relative to the application directory

n Video link

p Picture link

i Information link

h Header image for GUI

d Short description for the initial window

c Camera path

o Starting opacity of building (for dynamic building swaps), between 0 and 1

+ Indicates the end of a description

The following is an example used to define Norwich Cathedral. If multiple 3D model locations are defined, they are treated separately but are grouped together

under the current building name. Also the opacity of each building can be set as not all buildings were present in context of the map they are on, therefore these buildings are available to view but only fade into view when selected. An example of this is the Forum.

```
n Norwich Cathedral
m Resources/Buildings/Cathedral.obj
m Resources/Buildings/Cathedralbase.obj
i http://historacle.webapp3.uea.ac.uk/cathedral/profile.php
p http://historacle.webapp3.uea.ac.uk/cathedral/imageGallery.php
v http://historacle.webapp3.uea.ac.uk/cathedral/videoGallery.php
h Resources/Header Images/cathedral.png
c Resources/Camera Paths/cathedral.ASE
o 1.0
d Norwich Cathedral was founded by the first Bishop of Norwich, Herbert
  de Losinga. He acquired and cleared land in the centre of the city for
  the new Cathedral and adjacent Priory complex. Norwich Cathedral
  is the most complete Norman Cathedral in the UK and boasts a wealth
  of Romanesque features with later Gothic additions to create one of
  the most atmospheric sacred spaces in Europe.
+
```

The next section describes the design and implementation of the GUI (Graphical User Interface) used for the project, including the choices made to present rich media.

4.2 Graphical User Interface (GUI)

The following details the ideas and beliefs behind the graphical user interface designed for the case study, the details of which are found in Section 1.1. The graphical user interface is an integral point of contact between user and application, there are many aspects of user interaction which can profoundly affect the way in which each user understands what he or she is doing and what is expected of them. Certain questions were needed to be answered before and during implementation to enable a balance between application compatibility and ease of use. Therefore, the first steps towards interface design involved a clear definition of targets and what was to be expected in terms of complexity and depth of information.

From the beginning of the project (informally known as HistOracle) it was clear that not all requirements were going to be known from the outset; although there were general targets that were expected to evolve. These targets consisted of;

- Produce a legible interface that could be understood no matter the individuals native language.
- Attain a visually pleasing layout in keeping with the context of the project.
- Clearly show where the user is in navigation, offering the ability to return to previous windows.
- Always be able to return or 'reset' the system back to a home screen.
- Enable a 'quick-in-quick-out' approach to all users.
- Uncomplicated selection options.
- Keep buttons and interface elements easily selectable (in reference to size and shape).

4.2.1 Design

Various design decisions were made throughout development but from the outset there were targets which were considered to be important. These included; a consistent theme that would run throughout all user interface elements including graphics and button shape, also it was decided that keeping the design and structure simple would improve legibility; offering clear, informative buttons and interactive items. Furthermore, it was agreed that all information presented would be overlaid on the real-time environment (such as the building models themselves), which was also a technique by which to inform the user as to their currently focused object. This separation between overview and detailed view was made by presenting the user the specific object they selected in close proximity.

Clear control flow is integral to a smooth transition between various windows of information. In the context of HistOracle, simple functionality produced the most concise results, building four interface levels which resulted in an uncomplicated visual experience. These layers (or windows) consisted of;

1. Introduction screen, containing instructions on how to use the system. The user cannot continue onto the next stage without initialising tracking and clicking start.

2. Building selection screen, all buildings are separated between the left and the right of the screen with an overview of all buildings.

3. Building media selection, after a building has been selected the user is presented with further options (information, pictures and video).

4. Information window, depending on user interaction a specific window containing information, pictures or video is the last layer in the user interface.

At any point the user can return to the previous window (except the building selection, as a timer automatically returns the system to the first layer) or exit back to the building selection. Figure 4.4 is the control flow seen when interacting with the system, important processes to note are the interface loop found at stage 6 and 7; this is formed due to the rotary nature created when exploring all information on a particular building. Also, the database at stage 4 that provides all building information and media which is structured around an SQL database. Furthermore in terms of future expandability there are many possibilities due to the web based presentation used. As all major information uses a nested web browser and an SQL database back-end, content could be seamlessly added and/or edited without the need for major changes to internal program code.

4.2.2 Construction Context

Like the media requirements there were choices made as to how the GUI was created. Due to time restrictions it was apparent that creating a UI from scratch was impractical; although feasible there were third party options that provided a satisfactory means by which to construct what was required. Libraries such as wxWidgets, FLTK (Fast Light Tool Kit), GLGooey, Crazy Eddie's GUI System and Qt, the latter two offering the capabilities of creating completely custom GUI layouts. Qt also contained media and web modules making it the obvious choice.

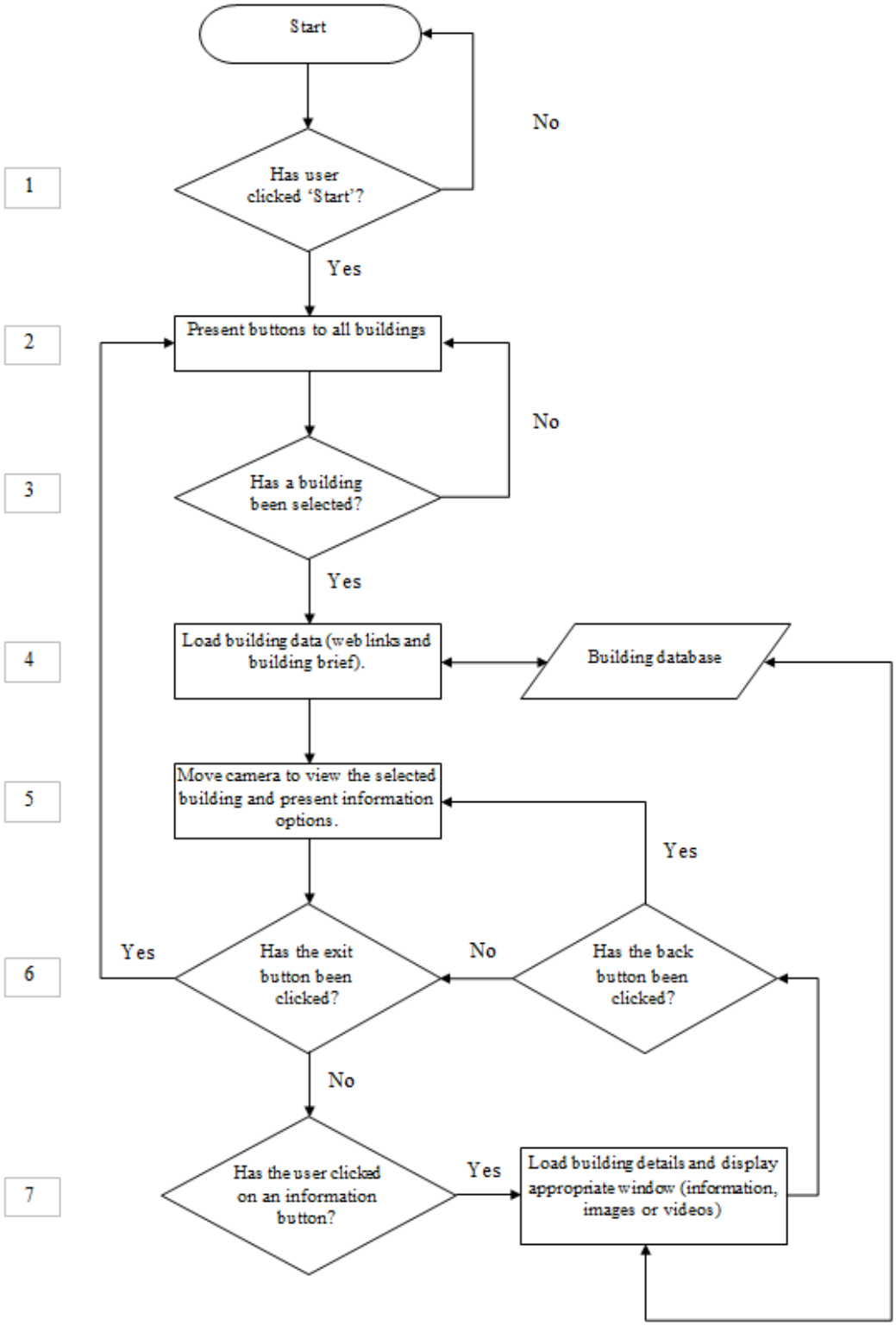

Figure 4.4: User interface control flow. This diagram is a high level overview of the user interface logic. The application is designed to continually run until closed, thus there is no internal end to the flow logic.

4.2.3 Layout

The graphic user interface was created with certain considerations in mind, including ease of access, element positions and handedness. The layout was intentionally designed for both left and right handed user's although a preference had to be taken to one side when designing specific windows. Button and user interface element placement greatly affects the way in which the user experiences and uses each part of the interactive system. Poor placement can lead to confusion, for example an interactive user interface element situated at the top of the environment (relative to the screen size) is harder to select in the context of physical control and gesturing. To elucidate, the relationship between element position and a gesture controlled interface is closely intertwined. Gesture control is greatly aided by an interface that works along side the advantages and disadvantages of the control system. For example, physical navigation (such as using a hand) does not usually attain the accuracy found by traditional hardware devices such as a mouse; generally there is no requirement for greater accuracy as navigational tasks often only require button selection. Therefore when creating the interface it was tailored towards these philosophies, easy navigation in a gesture environment.

Initial designs showed that a simple approach to overall user interface design yielded the clearest results. As their is an overview of all interactive buildings, each had to be easily accessible. This proved a difficult task due to clustered buildings which were physically close together. Figure 4.5 shows the first attempt, presenting all buildings at once to the user. As previously stated, the close proximity of buildings and their physical size meant they were difficult to interact with, and when indicating their whereabouts with a dialogue showed that interactive items became messy and

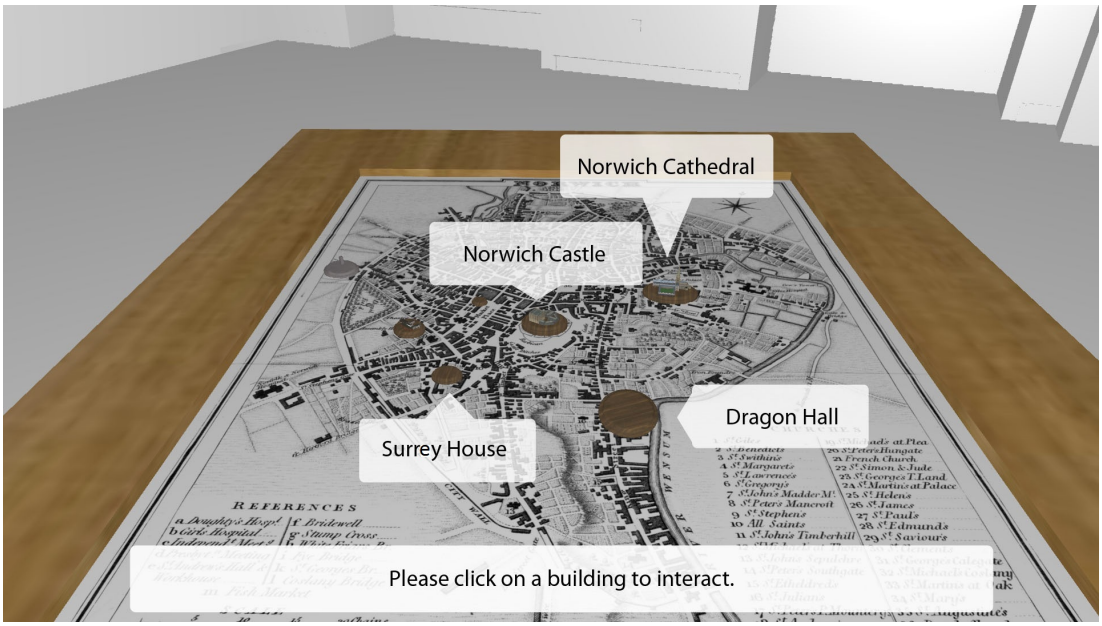

Figure 4.5: In this design each building available for inspection has a floating selectable dialogue, titled with the name of the building. After a user selects a dialogue, the next layer (videos, images and information) of options can be shown.

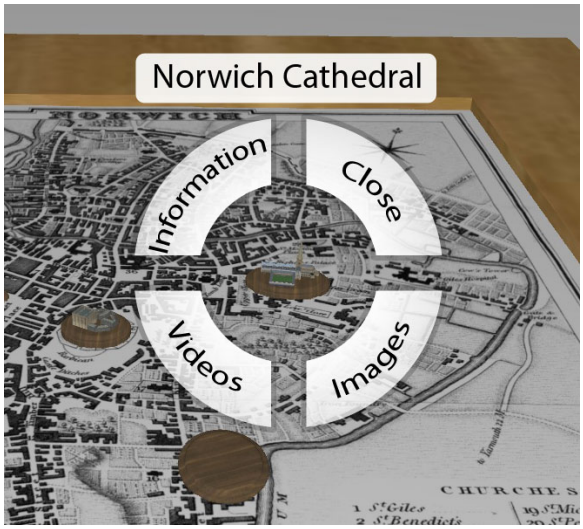

Figure 4.6: A radial menu would be presented to the user after interaction. All available options are displayed in each segment of the menu, providing buttons in close proximity to the user's cursor.

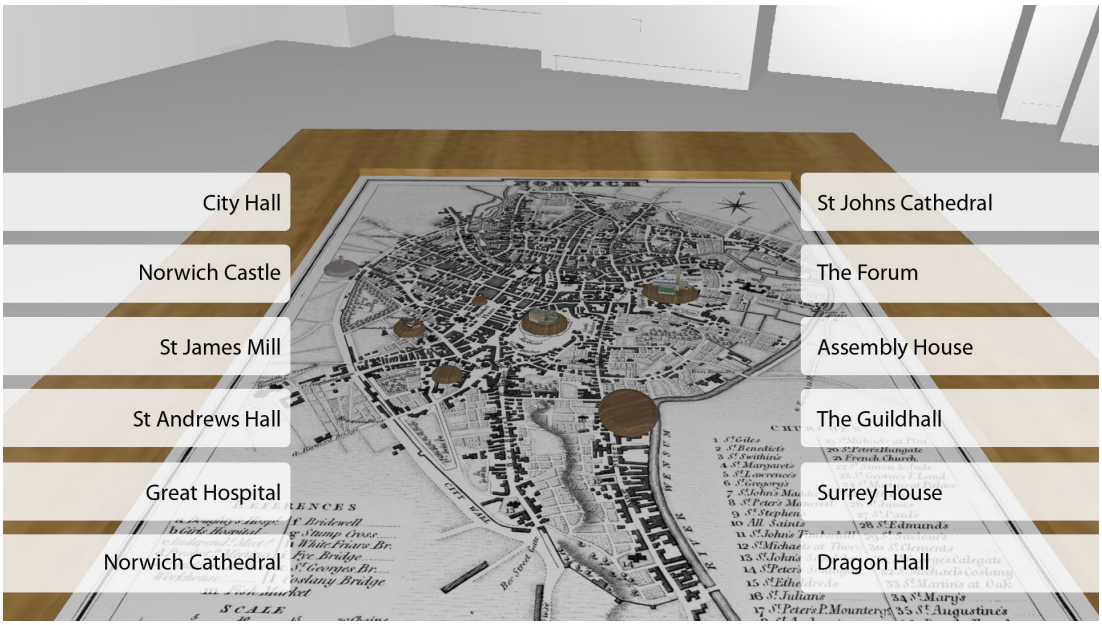

Figure 4.7: Interface elements were repositioned and placed on both the left and right of the screen, which meant items were more easily accessible when considering gesture control.

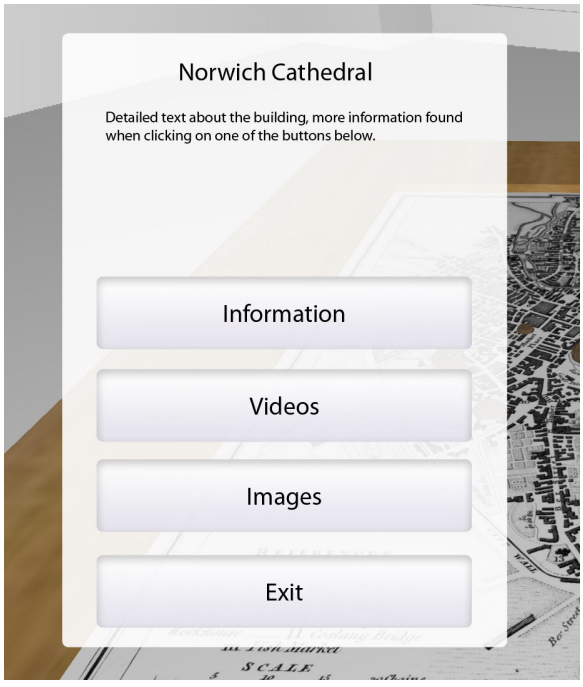

Figure 4.8: A more simple design approach provided a clearer and cleaner interface in which text was not rotated or unreadable. Unlike that found in Figure 4.6.

disorganised. For clarity, interface elements were redesigned, into more consistent positions on both the left and right of the screen as seen in Figure 4.7. When a building is selected it was thought that creating radial menus would provide ease of access to all available data in close proximity. Figure 4.6 is an early attempt at designing a feasible layout in which the user would be presented with the radial menu after clicking on a building. In keeping with the design found in Figure 4.7, when a building is selected instead of a radial menu a more simplified interface was chosen to keep consistency in the user experience. Figure 4.8 were the initial ideas in how to keep a standardised layout throughout, this being displayed when a user selects a building.

In reference to list 4.2.1, Figure 4.9 shows the second layer of the user interface. Highlighted areas 1 and 2 show all interactive buttons which are connected to their associated buildings. In the background the real-time environment shows both the 18^{th} century map and appropriate surroundings (such as candle sticks, chairs and barrels of the period) and all twelve available buildings. Areas A through L show the concentration of all the available buildings; in keeping with the time period, buildings that were not present during the 18^{th} century are initially hidden from view but are still selectable from the building menu. When the building is selected it is brought into view in its correct position. Not only does this keep the map context correct, it creates space between buildings that are otherwise closely situated.

After interacting with a building button the user is flown down towards the building selected and presented with the 3D real-time model. The layout consists of two separate areas, Figure 4.10 shows interface layer three (from list 4.2.1). It was decided that all information displayed (including videos and pictures) were to be oriented to

Figure 4.9: This image was taken from the final design of the user interface. The white areas represent the different regions of the initial UI, where 1 contains the university and charity logos. Areas 2 and 3 contain the buttons representing all buildings. Also, the areas circled A to L highlight all the buildings and their physical placement on the 19^{th} Century map of Norwich city.

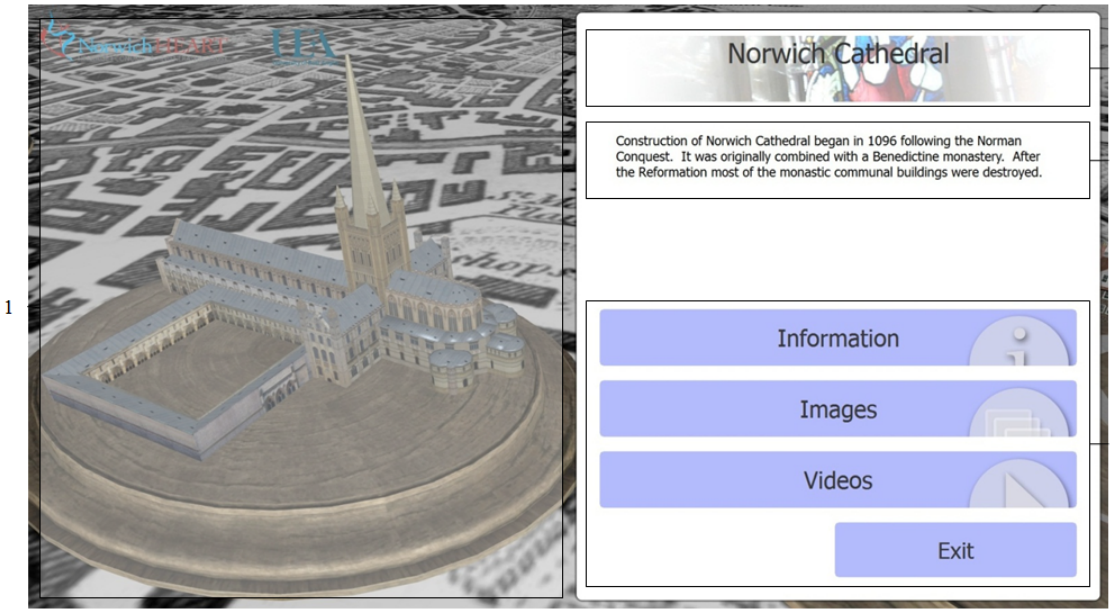

Figure 4.10: This image shows the window design found for each building where 1 contains all real-time rendered images (in this case a cathedral) and 2 to 4 brief information about the buildings and buttons to extra content.

the right as the real-time background was positioned to the left. This ran throughout all extended information windows (layers 3 and 4). Also, the diagram shows the format in which information and interactive items were formed, favouring buttons to the bottom of the window due to the limitations found with gesture based interaction. More specifically, areas 2 and 3 display general information about the currently selected building, the building name and related background image and blurb. Area 4 contains all available buttons to each media window.

4.2.4 Media Playback

Opposed to the ergonomic requirements there were very prominent problems that needed to be addressed. As the application was intended to present real-time graphics and media content such as extra information, images and video, a distinct need for the ability to display all media types was present. This factor had a large bearing on how the GUI was developed and what was used to attain an integrated environment that satisfied all requirements.

Various libraries were available that could handle different media types (such as .jpg, .png, .avi, .wmv, .mp4 etc), as the media data base grew it was not necessarily known what format each item would be, therefore the libraries available were considered in terms of their compatibility. There were four options available that filled criteria, GStreamer and FFmpeg are open source libraries which both boast the capability to render multiple video file types. Qt also plays the native media types supported by the operating system, furthermore it provides a platform in which to construct complete interfaces. The final option was QuickTime, which did not support all media types and was also closely intertwined with Apple.

After experimentation it was found that the obvious choice for media playback was Qt due to the extra functionality which was appropriate for the project; after the initial development stages it was considered that media handling would take place entirely inside a web based environment. This choice proved vital and is explained in detail in Section 4.2.1.

The next and final phase was to amalgamate both projects (gesture recognition via depth image processing and real-time rendering) in to a practical application that could be used as an informative tool to the public. As the targets for both image processing and rendering were similar, project aims were consistent, providing a significant bearing and leading both towards compatibility with one another, hence the choices made when designing the GUI (which was tailored towards the case study).

Given the type of 3D data that was provided, an efficient and cohesive method of loading and rendering large virtual models has been discussed, including the optimisation of such models for contemporary graphics hardware. Vitally, these optimisations aid in freeing the computer to perform other time consuming tasks such as image processing. As the GUI is important, being the entity that the user interacts with, different types and layouts have been discussed. Important factors have been considered, such as visibility of GUI elements, their position on the screen and their appearance. Most importantly, their ease of access has been taken as paramount in context of the gesture based system it will be used with.

The next chapter concentrates on the results collected during public testing, and reflects on the public's impressions.

Chapter 5

Results

To evaluate the performance of the techniques presented we asked members of the public to try the interactive system described in the previous section. The system was set up in the Forum building in Norwich, which is a public building housing a number of business and services including the library. Due to the significant amount of reflective surfaces in the forum only the Microsoft Kinect could be used as the reflections caused problems for the Panasonic D-Imager (seen in Figure 3.5, Section 3.1.1). Public testing consisted of two private presentations of the system, as well as a public display which lasted for a week. Both periods included a set up consisting of a single computer, a large screen (42 inch), a Microsoft Kinect and importantly, feedback sheets. The participants were asked to fill out a questionnaire which reflected their experience, these results can be seen in Tables 5.1 and 5.2. The first included both selection methods, the latter only exhibited the grasping technique. The user's were asked to rate the quality of interaction on a scale from V. Poor (very poor) to V. Good (very good). Table 5.1 shows the results from initial public testing where only the grasping selection technique was available.

After this, both selection strategies were tested, the results are shown in Table 5.1. The table illustrates the user's opinions of the two systems. Rows 1-4 detail

general opinion of the system whilst the two types of selection methods are compared in rows 6-10.

	Questions	*V. Poor*	*Poor*	*Fair*	*Good*	*V. Good*
1	The Clarity of the instructions.			1	13	19
2	Ease of starting the system with your hand.			11	13	9
3	Movement and control of the cursor		3	7	14	9
4	Ability to select buttons	1	3	6	12	11
5	Ease of accessing the information		2	2	17	12
6	Quality of information			4	10	19
7	The layout of the information				14	19

Table 5.1: This table shows the results gained from a week of consistent testing by the public. Individuals were asked to answer general questions that covered the experience as a whole. These results focus purely on the tactile grasping technique.

	Questions	*V. Poor*	*Poor*	*Fair*	*Good*	*V. Good*
1	Ease of starting the system with your hand.			3	15	3
2	Speed of navigation			7	11	3
3	Accuracy of navigation		2	3	12	4
4	Layout of information			1	6	14
A	Ease of use		1	3	14	1
B	Ease of button selection		1	2	9	7
C	Button selection time			6	12	1
D	Ease of use	1	2	4	8	8
E	Ease of button selection		1	7	9	2
F	Button selection response time	2		5	9	3

Table 5.2: This table were the results gained from testing both techniques simultaneously on the same individual. It also includes more specific questions aimed at both types of selection techniques (hover selection, A - C and grasp selection, D - F).

During user exposure it became clear that opinions were divided, some individuals favoured the physical selection process (grasp selection) over the hover approach simply because they were under full control as to when they wanted to select an item. On the other hand, some user's found the hover selection technique favourable due to the consistency of the automated approach. It can be seen that the hover

selection method produced fewer errors than the grasping selection method but was not necessarily the more intuitive option. Figure 5.1 show the initial grasp selection impressions.

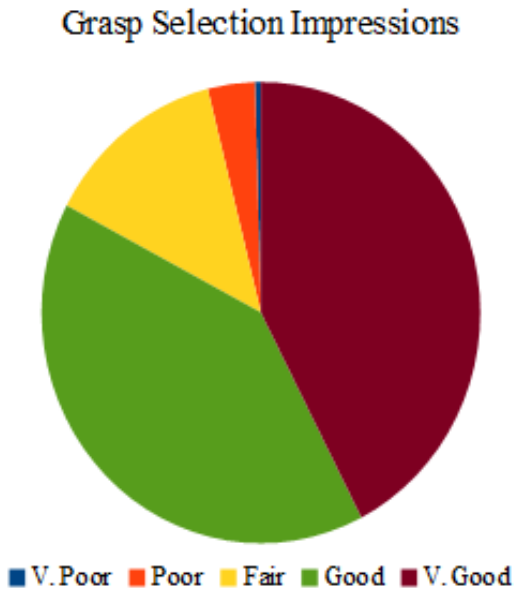

Figure 5.1: This chart gives an idea of how the initial testing was judged by the general public. This initial testing concentrated on the grasping selection technique.

It is worth noting that this initial set of results reflects the opinions of individuals that had not been exposed to any other selection technique, such as the hover selection. Moreover, if these individuals had been exposed to other methods, their opinions may have differed.

Figure 5.2 clearly shows the divided opinions of the general public, where the grasp technique votes are spread more evenly between fair and very good. These results help to indicate possible areas of improvement, for example the hover technique indicated that individuals with a lesser understanding of the system (usually but not restricted to older individuals) would prefer a semi-automatic navigational process

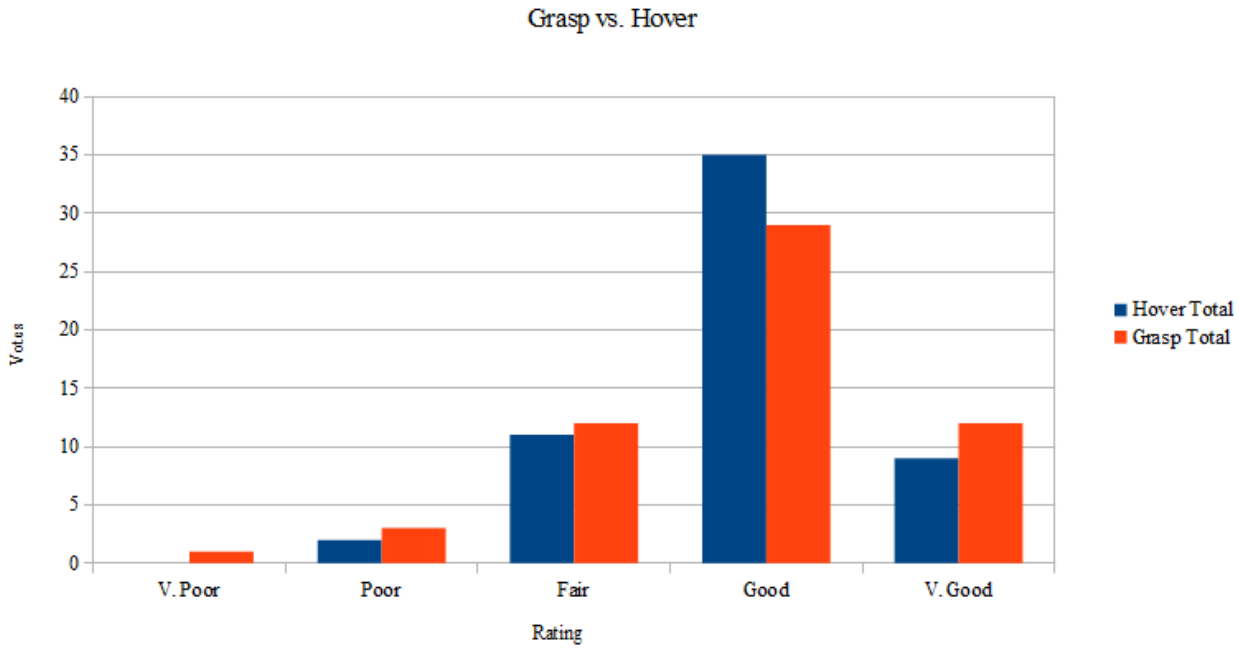

Figure 5.2: This graph shows the direct comparisons between both the hover and grasp technique (blue and red, respectively).

in which control is taken from them, to favour basic selection methods. Results also highlighted that users who persisted and took time to learn the grasping technique gained more satisfaction from having complete control over what and when items were selected.

5.1 Target Summary

Although the project targets were ambitious, they certainly were not impossible as most were satisfied. Certain targets were prioritised over others, there were original intentions to provide complete (or near to) freedom of movement around the virtual environment which was deemed unnecessary after development had started. Target 1 and 3 stated that the subject area (Norwich) was to be rendered entirely. Detailed information was available in the form of LIDAR data (Light Detection And Ranging) discussed in section 2.5, was being used to efficiently re-produce Norwich city in its

entirety. When considering the scope of the project, the overhead involved in reproducing Norwich completely and providing the user complete freedom was too large. Preliminary testing showed that even though rendering large sections of the city was possible, drastic countermeasures would have had to be taken to efficiently render all data and keep an interactive frame rate. Furthermore there were concerns over full interactivity. Giving the user complete access to roam either in the proximity of the building or entire city would give a more interactive experience but on the other hand would cause situations in which the user cannot return (such as getting lost or stuck). Another overbearing factor would be the complexity of control required to navigate using devices other than a traditional mouse or keyboard or a controller, especially when tailoring human-computer interaction to gesture based devices.

There was also secondary information such as the contextual data for each building (6^{th} item) which proved to be invaluable and was able to provide temporal details for each structure (videos showing the physical changes over time, 5^{th} item). Rendering each building (and potentially the surrounding area) from all possible time periods meant loading and processing very large quantities of data, thus containing all contextual information in other forms removed the need for real-time temporal data for each building, this resulted in more realistic requirements, which were more attainable in the projects duration. Also, this meant that data requirements were smaller due to the large volume of models that represented each historical record.

Chapter 6

Conclusion

It is important to stress that the area of gesture recognition and HCI (Human Computer Interaction) that does not require physical interaction is in its infancy in regards to the current capabilities of commercial and academic achievement. One of the most important aims driving this thesis was to create a proof of concept in that an entirely encapsulated gesture driven system could be produced with; currently available hardware (2010) and multiple depth devices, potentially any depth device that provides access to raw data.

We have presented multiple image processing techniques that take advantage of depth information acquired from depth devices such as the Panasonic D-Imager and the Microsoft Kinect, irrespective of the device. These image processing techniques involve the analysis of the entire scene, focusing on areas that are closest to the devices physically and discarding objects that do not fit a particular profile. A combination of dynamic depth and bounding box analysis resulted in an intuitive system. The success of the system has been measured with public testing. The virtual exhibit highlighted rich information on the listed 12 buildings of Norwich, where the gesture driven interaction provided an intuitive method of accessing these archives.

Also, we have presented two techniques that use the image processing library designed and that are compatible with multiple depth devices which are capable of interpreting gesture based commands. We have tested the system at public events and the results have been presented. Two types of selection methods were developed, one more active grasping action and the second involving hovering over click-able menu items. It was observed that there was a clear divide between both techniques, although not all participants understood and mastered the physical selection technique, meaning some individuals favoured the alternative method.

Given the scope of the project, some intended functionality was deemed too large to successfully implement. This was due to ambitious longer term targets, such as providing full navigational control via gestures and supporting multiple hardware devices. On the other hand, those targets that were met have been tested with members of the public and proven to be satisfactory.

6.1 Future Work

Searching for and tracking a single hand was made relatively efficient and robust with the techniques presented in this thesis. As a hand could be discovered within a short period of time, the system could be extended to accurately track more than one hand which would open up a multitude of more advanced gestures. Furthermore, tracking multiple hands would create the possibility to grant multiple users access to content simultaneously. Alternatively, it could give a single individual advanced control of the content presented, including direct manipulation in three axes. As a result of the research presented in this thesis, the techniques and methods used have been detailed and published [3] at the VSMM (Virtual Systems and Multimedia 2012) conference with the intention of further forwarding the area.

Bibliography

[1] *Minority report*, Website, December 2002, `http://stuq.nl/media/minority-report-ui.jpg`.

[2] *Diy 3d scanning*, Website, November 2010, `http://www.youtube.com/watch?v=Xd2HzafqxT8&feature=related`.

[3] Sam Bailey, Chris Powell, Stephen D. Laycock, and Andy M. Day, *Interactive exploration of historic information via gesture recognition*, VSMM, IEEE, 2012, pp. 211–218.

[4] P. Barrie, A. Komninos, and O. Mandrychenko, *A pervasive gesture-driven augmented reality prototype using wireless sensor body area networks*, Proceedings of the 6th International Conference on Mobile Technology, Application & Systems, ACM, 2009, pp. 1–4.

[5] Fabio Bettio, Enrico Gobbetti, Fabio Marton, Alex Tinti, Emilio Merella, and Roberto Combet, *A point-based system for local and remote exploration of dense 3D scanned models*, The 10th International Symposium on Virtual Reality, Archaeology and Cultural Heritage, October 2009, pp. 25–32.

[6] Kamel Boulos, Maged N, Bryan J Blanchard, Cory Walker, Julio Montero, Aalap Tripathy, and Ricardo Gutierrez-Osuna, *Web GIS in practice X: a microsoft kinect natural user interface for google earth navigation*, July 26 2011.

[7] D. A. Bowman, D. Koller, and L. F. Hodges, *Travel in immersive virtual environments: An evaluation of viewpoint motion control techniques*, IEEE Proceedings of VRAIS'97 (1997), no. 7, 45–52.

[8] V. Buchmann, S. Violich, M. Billinghurst, and A. Cockburn, *FingARtips: gesture based direct manipulation in Augmented Reality*, Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia, ACM, 2004, p. 221.

[9] S. Carbini, J.E. Viallet, O. Bernier, and B. Bascle, *Tracking body parts of multiple people for multi-person multimodal interface*, Computer Vision in Human-Computer Interaction (2005), 16–25.

[10] L. Chittaro, L. Ieronutti, and R. Ranon, *VEX-CMS: A tool to design virtual exhibitions and walkthroughs that integrates automatic camera control capabilities*, Smart Graphics, Springer, 2010, pp. 103–114.

[11] Luca Chittaro, Lucio Ieronutti, and Roberto Ranon, *Navigating 3D virtual environments by following embodied agents: a proposal and its informal evaluation on a virtual museum application*, Psychnology **2** (2004), no. 1, 24–42. MR J.psychnology.2.1.24

[12] Luca Chittaro, Roberto Ranon, and Lucio Ieronutti, *3D object arrangement for novice users: the effectiveness of combining a first-person and a map view*, VRST (Stephen N. Spencer, Yoshifumi Kitamura, Haruo Takemura, Kiyoshi Kiyokawa, Benjamin Lok, and Daniel Thalmann, eds.), ACM, 2009, pp. 171–178.

[13] C.W. Chu and R. Nevatia, *Real time body pose tracking in an immersive training environment*, Human–Computer Interaction (2007), 146–156.

[14] D.G. Curry, G.L. Martinsen, and D.G. Hopper, *Capability of the human visual system*, Proceedings of SPIE, vol. 5080, 2003, pp. 58–69.

[15] T. Darrell, G. Gordon, M. Harville, and J. Woodfill, *Integrated person tracking using stereo, color, and pattern detection*, International Journal of Computer Vision **37** (2000), no. 2, 175–185.

[16] Laura Downs, Tomas Möller, and Carlo H. Séquin, *Occlusion horizons for driving through urban scenery*, SI3D (John F. Hughes and Carlo H. Séquin, eds.), ACM, 2001, pp. 121–124.

[17] N. Elmqvist, M.E. Tudoreanu, and P. Tsigas, *Tour generation for exploration of 3D virtual environments*, Proceedings of the 2007 ACM symposium on Virtual reality software and technology, ACM, 2007, pp. 207–210.

[18] R. Fernando, S. Fernandez, K. Bala, and D.P. Greenberg, *Adaptive shadow maps*, Proceedings of the 28th annual conference on Computer graphics and interactive techniques, ACM, 2001, pp. 387–390.

[19] G. Fitzmaurice, J. Matejka, I. Mordatch, A. Khan, and G. Kurtenbach, *Safe 3d navigation*, Proceedings of the 2008 symposium on Interactive 3D graphics and games, ACM, 2008, pp. 7–15.

[20] Valentino Frati and Domenico Prattichizzo, *Using kinect for hand tracking and rendering in wearable haptics*, World Haptics (Lynette A. Jones, Matthias Harders, and Yasuyoshi Yokokohji, eds.), IEEE Computer Society, 2011, pp. 317–321.

[21] William T. Freeman and Michal Roth, *Orientation histograms for hand gesture recognition*, Tech. report, Mitsubishi Electric Research Laboratories, December 1994, IEEE Intl. Wkshp. on Automatic Face and Gesture Recognition, Zurich, June, 1995,.

[22] S. Fukatsu, Y. Kitamura, T. Masaki, and F. Kishino, *Intuitive control of bird's eye overview images for navigation in an enormous virtual environment*, Proceedings of the ACM symposium on Virtual reality software and technology, ACM, 1998, p. 76.

[23] K. Fukunaga and L. D. Hostetler, *The estimation of the gradient of a density function, with applications in pattern recognition*, IEEE Trans. Information Theory **21** (1975), no. 1, 32–40.

[24] Tinsley A. Galyean, *Guided navigation of virtual environments*, SI3D (Michael Zyda, ed.), ACM, 1995, pp. 103–104, 210.

[25] Gabriele Guidi, Fabio Remondino, Michele Russo, Fabio Menna, and Alessandro Rizzi, *3D modeling of large and complex site using multi-sensor integration and multi-resolution data*, VAST (Michael Ashley, Sorin Hermon, Alberto Proença, and Karina Rodriguez-Echavarria, eds.), Eurographics Association, 2008, pp. 85–92.

[26] L. Ieronutti, R. Ranon, and L. Chittaro, *Automatic derivation of electronic maps from X3D/VRML worlds*, Proceedings of the ninth international conference on 3D Web technology, ACM, 2004, pp. 61–70.

[27] Robert Kooima, *Kinect virtual reality with 3d tv*, Website, February 2011, `http://kinect.dashhacks.com/kinect-hacks/2011/02/26/kinect-virtual-reality-3d-tv`.

[28] Oliver Kreylos, *Xbox kinect / hack culture*, Website, November 2010, `http://www.contagiousmagazine.com/2010/11/following_the_release_of_the.php`.

[29] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale, and S. Shafer, *Multi-camera multi-person tracking for easyliving*, Visual Surveillance, 2000. Proceedings. Third IEEE International Workshop on, IEEE, 2000, pp. 3–10.

[30] V. Kwee, A.D. Radford, and D.C. Bruton, *Educative visuals-digital delivery of architectural information for (potential) heritage buildings*, International Symposium on Virtual Reality, Archaeology and Cultural Heritage VAST (7th: 2006: Nicosia, Cyprus), 2006.

[31] S. Laakso and M. Laakso, *Design of a body-driven multiplayer game system*, Computers in Entertainment (CIE) **4** (2006), no. 4, 7.

[32] R.G. Laycock, S.D. Laycock, and A.M. Day, *Haptic navigation and exploration of high quality pre-rendered environments*, CIPA/VAST/EG/EuroMed 2006: 37th

CIPA International Workshop dedicated on e-Documentation and Standarisation in Cultural Heritage, Eurographics Association, 2006, p. 17.

[33] A.E. Lefohn, S. Sengupta, and J.D. Owens, *Resolution-matched shadow maps*, ACM Transactions on Graphics (TOG) **26** (2007), no. 4, 20.

[34] George Lepouras and Costas Vassilakis, *Virtual museums for all: employing game technology for edutainment*, Virtual Reality **8** (2004), no. 2.

[35] J. Mackinlay, S. Card, and G. Robertson, *Rapid controlled movement through a virtual 3D workspace*, 1990, pp. 171–176.

[36] P. Maes, T. Darrell, B. Blumberg, and A. Pentland, *The ALIVE system: Wireless, full-body interaction with autonomous agents*, Multimedia Systems **5** (1997), no. 2, 105–112.

[37] S. Malassiotis, N. Aifanti, and MG Strintzis, *A gesture recognition system using 3D data*, 3D Data Processing Visualization and Transmission, 2002. Proceedings. First International Symposium on, IEEE, 2002, pp. 190–193.

[38] Jamie Tremaine Matt Strickland and Greg Brigley, *Kinect surgery assistance*, Website, March 2011, `http://sunnybrook.ca/media/item.asp?i=616`.

[39] T. McReynolds and D. Blythe, *Programming with openGL: Advanced rendering*, SIGGRAPH'97 Course Notes (1997), 51–57.

[40] Microsoft, *Kinect games, kinect sports*, Website, November 2001, `http://marketplace.xbox.com/en-US/Product/Kinect-Sports/66acd000-77fe-1000-9115-d8024d5308c9`.

[41] M.R. Mine, F.P. Brooks Jr, and C.H. Sequin, *Moving objects in space: exploiting proprioception in virtual-environment interaction*, Proceedings of the 24th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., 1997, pp. 19–26.

[42] S. Mitra and T. Acharya, *Gesture recognition: A survey*, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews **37** (2007), no. 3, 311–324.

[43] L. Mosaker, *Visualising historical knowledge using virtual reality technology*, Digital Creativity **12** (2001), no. 1, 15–25.

[44] C. Niederauer, M. Houston, M. Agrawala, and G. Humphreys, *Non-invasive interactive visualization of dynamic architectural environments*, ACM Transactions on Graphics **22** (2003), no. 3, 700.

[45] K. Oka, Y. Sato, and H. Koike, *Real-time tracking of multiple fingertips and gesture recognition for augmented desk interface systems*, FG, 2002, pp. 411–416.

[46] Panasonic, *Panasonic d-imager security examples*, Website, March 2011, `http://pewa.panasonic.com/components/built-in-sensors/3d-image-sensors/d-imager/`.

[47] F. Quek, *Toward a vision-based hand gesture interface*, Virtual reality software & technology: proceedings of the VRST'94 Conference, 23-26 August 1994, Singapore, World Scientific Pub Co Inc, 1994, p. 17.

[48] Fabio Remondino, Stefano Girardi, Lorenzo Gonzo, and Alessandro Rizzi, *Multi-resolution modeling of complex and detailed cultural heritage*, VAST (Michael Ashley, Sorin Hermon, Alberto Proença, and Karina Rodriguez-Echavarria, eds.), Eurographics Association, 2008, pp. 1–8.

[49] Y. Sato, Y. Kobayashi, and H. Koike, *Fast tracking of hands and fingertips in infrared images for augmented desk interface*, FG, 2000, pp. 462–467.

[50] Thomas Schlomer, Benjamin Poppinga, Niels Henze, and Susanne Boll, *Gesture recognition with a wii controller*, Tangible and Embedded Interaction (Albrecht Schmidt, Hans Gellersen, Elise van den Hoven, Ali Mazalek, Paul Holleis, and Nicolas Villar, eds.), ACM, 2008, pp. 11–14.

[51] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli, *Fast shadows and lighting effects using texture mapping*, Computer Graphics (SIGGRAPH '92 Proceedings) **26** (1992), no. 2, 249–252.

[52] David Molyneaux Shahram Izadi, Otmar Hilliges, *Kinect surface reconstruction (kinectfusion)*, Website, October 2011, `http://research.microsoft.com/en-us/projects/surfacerecon/`.

[53] M. Stamminger and G. Drettakis, *Perspective shadow maps*, ACM Transactions on Graphics (TOG) **21** (2002), no. 3, 557–562.

[54] R. Stoakley, M.J. Conway, and R. Pausch, *Virtual reality on a WIM: interactive worlds in miniature*, Proceedings of the SIGCHI conference on Human factors in computing systems, ACM Press/Addison-Wesley Publishing Co., 1995, pp. 265–272.

[55] S.L. Stoev, D. Schmalstieg, and W. Straßer, *Two-handed through-the-lens-techniques for navigation in virtual environments*, Immersive Projection Technology and Virtual Environments 2001: proceedings of the Eurographics Workshop in Stuttgart, Germany, May 16-18, 2001, Springer Verlag Wien, 2001, p. 51.

[56] J. Stumpfel, C. Tchou, N. Yun, P. Martinez, T. Hawkins, A. Jones, B. Emerson, and P. Debevec, *Digital Reunification of the Parthenon and its Sculptures*, 4th International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage, Brighton, UK, 2003.

[57] V. Sundstedt, A. Chalmers, and P. Martinez, *High fidelity reconstruction of the ancient Egyptian temple of Kalabsha*, Proceedings of the 3rd international conference on Computer graphics, virtual reality, visualisation and interaction in Africa, ACM, 2004, pp. 107–113.

[58] S. Suzuki and K. Abe, *Topological structural analysis of digitized binary images by border following*, Computer Vision, Graphics and Image Processing **30** (1985), 32–46.

[59] M. Tang, *Recognizing hand gestures with microsofts kinect*, Web Site: http://www. stanford. edu/class/ee368/Project_11/Reports/Tang_Hand_Gesture_Recognition. pdf (2011).

[60] S. Todt, C. Rezk-Salama, T. Horz, A. Pritzkau, and A. Kolb, *An Interactive Exploration of the Virtual Stronghold Dillenburg*, Proc. Eurographics Cultural Heritage (2007), 213–218.

[61] C. Ware and S. Osbourne, *Exploration and virtual camera control in virtual three dimensional environments*, Proceedings of SIGGRAPH Symposium on Interactive 3D Graphics 1990 (1990), 175–183.

[62] J. Warren, *Unencumbered full body interaction in video games*, Unpublished Master's Thesis, Parsons School of Design (2003).

[63] Wikipedia, *Kinect*, Website, November 2010, `http://en.wikipedia.org/wiki/Kinect`.

[64] P. Wonka, M. Wimmer, and D. Schmalstieg, *Visibility preprocessing with occluder fusion for urban walkthroughs*, Rendering Techniques 2000 (2000), 71–82.

[65] Jenna Wortham, *With kinect controller, hackers take liberties*, New York Times, November 2010, `http://www.nytimes.com/2010/11/22/technology/22hack.html?_r=2`.

[66] Y. Wu and T. Huang, *Vision-based gesture recognition: A review*, Gesture-Based Communication in Human-Computer Interaction (1999), 103–115.

[67] M.H. Yang, N. Ahuja, and M. Tabb, *Extraction of 2d motion trajectories and its application to hand gesture recognition*, Pattern Analysis and Machine Intelligence, IEEE Transactions on **24** (2002), no. 8, 1061–1074.

[68] Xenophon Zabulis, Dimitris Grammenos, Thomas Sarmis, Konstantinos Tzevanidis, and Antonis A. Argyros, *Exploration of large-scale museum artifacts through*

non-instrumented, location-based, multi-user interaction, VAST (Alessandro Artusi, Morwena Joly, Geneviève Lucet, Denis Pitzalis, and Alejandro Ribés, eds.), Eurographics Association, 2010, pp. 155–162.

[69] R. Zeleznik and A. Forsberg, *UniCam2D gestural camera controls for 3D environments*, Proceedings of the 1999 symposium on Interactive 3D graphics, ACM, 1999, pp. 169–173.

[70] J. Zhou and J. Hoang, *Real time robust human detection and tracking system*, Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on, IEEE, 2005, p. 149.

[71] T. Zuk, S. Carpendale, and W. D. Glanzman, *Visualizing temporal uncertainty in 3D virtual reconstructions*, The 6th International Symposium on Virtual Reality, Archaeology and Cultural Heritage (Pisa, Italy) (Mark Mudge, Nick Ryan, and Roberto Scopigno, eds.), Eurographics Association, 2005, pp. 99–106.