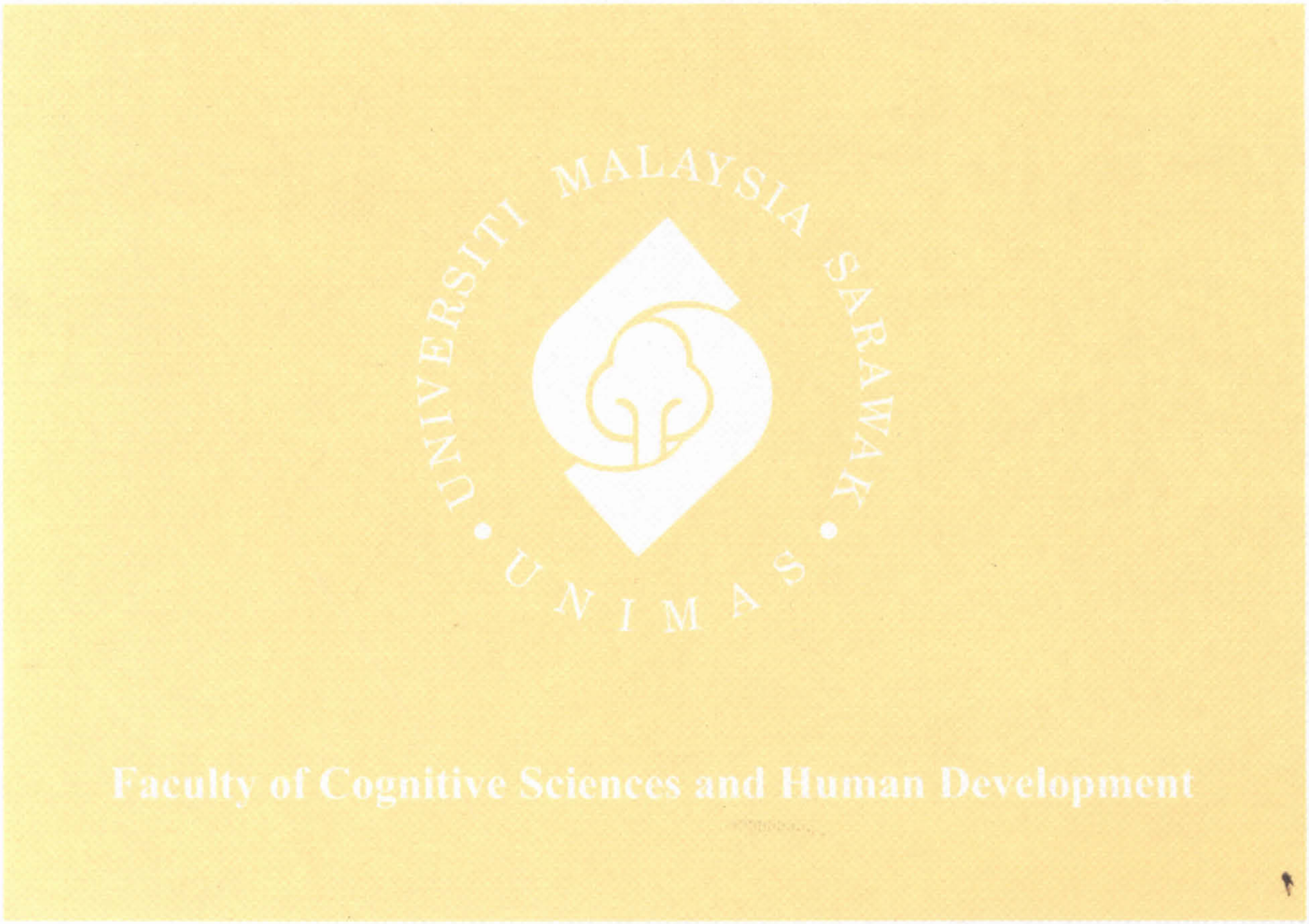

AUGMENTED REALITY APPLICATION BUILDER: INTEGRATION OF FINGER TRACKING FORINTERACTION IN AUGMENTED REALITY SYSTEM

Allen Choong Chieng Hoon

Master of Science
2010

AUGMENTED REALITY (AR) APPLICATION BUILDER: INTEGRATION OF FINGER TRACKING FOR INTERACTION IN AR SYSTEM

ALLEN CHOONG CHIENG HOON

A thesis submitted in fulfilment of the requirements for the Degree of Master of Science (Cognitive Science)

Faculty of Cognitive Sciences and Human Development
UNIVERSITI MALAYSIA SARAWAK
2009

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Ng Giap Weng for his helpful advice, guidance and comments.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

AUGMENTED REALITY (AR) APPLICATION BUILDER: INTEGRATION OF FINGER TRACKING FOR INTERACTION IN AR SYSTEM

Augmented Reality (AR) is a technology that generates virtual information to the real world to the users perception and interaction to perform the tasks. This research studied and developed a computer library, AR Application Builder, which was integrated with the finger tracking feature for the interaction with the virtual objects in the AR system. The AR Application Builder allows users to build AR applications. Hence, the users are able to use their bare finger to interact with the virtual objects. The development of the AR Application Builder involved video capturing, image processing, vision-based AR registration, virtual objects generation, and finger tracking. Video capturing was implemented by using DirectShow. An image library was built to perform image processing functions: greyscale transformation and thresholding. ARToolKit libraries were integrated to implement vision-based registration. OpenGL and GLUT were used to draw the 3D computer graphics as the augmented objects. Finger tracking involved skin colour segmentation and the template matching. The template matching function was implemented by using OpenCV. Furthermore, an evaluation was conducted to test the system functionality of finger tracking for interaction with the virtual object in AR system. Thus, an AR video player was built for the evaluation. The result of the evaluation showed that the participants were able to use bare finger to interact with the virtual objects of the AR application. This allows the participants to interact with the virtual objects naturally without any other device.

ABSTRAK

PEMBINA APLIKASI AUGMENTED REALITY (AR): INTEGRASI PENJEJAKAN JARI UNTUK INTERAKSI DALAM SISTEM AR

Augmented Reality (AR) adalah satu teknologi yang menghasilkan maklumat maya dalam dunia sebenar untuk menambah tanggapan dan interaksi penguna semasa melaksanakan tugas. Penyelidikan ini belajar dan membina sebuah perpustakaan komputer, Pembina Aplikasi AR (AR Application Builder), yang disepadukan dengan penjejakan jari untuk interaksi dengan benda-benda maya dalam sistem AR. Pembina Aplikasi AR membenarkan pengguna membina pelbagai aplikasi AR. Jadi, pengguna berupaya menggunakan jari mereka untuk interaksi dengan benda-benda maya. Pembangunan Pembina Aplikasi AR terlibat penangkapan video, pemprosesan imej, pendaftaran AR berdasarkan visi, generasi benda-benda maya, dan penjejakan jari. Penangkapan video telah dilaksana dengan menggunakan DirectShow. Satu perpustakaan imej telah dibina untuk menjalankan fungsi-fungsi pemprosesan imej: transformasi skala kelabu dan "thresholding". Perpustakaan ARToolKit disepadukan untuk melaksanakan pendaftaran berdasarkan visi. OpenGL dan GLUT digunakan untuk melukis grafik komputer 3D sebagai benda tambahan. Penjejakan jari terlibat segmentasi warna kulit dan "template matching". Fungsi "template matching" telah dilaksanakan dengan menggunakan OpenCV. Selain itu, satu penilaian telah dijalankan untuk menguji fungsi penjejakan jari dan interaksi dengan objek maya dalam sistem AR. Oleh itu, satu pemain video AR telah dibina untuk penilaian. Hasil penilaian itu menunjukkan bahawa peserta-peserta berupaya menggunakan jari untuk berinteraksi dengan objek maya implikasi AR tersebut. Peserta-perserta boleh berinteraksi dengan object maya secara semula jadi tanpa peranti lain.

CHAPTER 1
INTRODUCTION

1.0 Overview

Augmented Reality (AR) Application Builder is a computer library integrated with the finger tracking for interaction with virtual objects in the AR system. The users are able to develop AR applications by using the library. The library allows users to render the AR environment with 2D and 3D virtual objects in the AR environment. The library requires a video capture device so that the users can capture real-time environment and render the AR environment. When they are rendering their AR environment, they can use a mouse device or a finger to interact with the virtual objects.

Besides developing the computer library, this research also studied on computer

vision techniques for AR registration. AR registration is a method to enable the virtual objects align properly in the real environment so that users can perceive the virtual objects in the correct position and orientation. Then the system will generate 3D computer graphics in the real environment.

AR also emphasises the interaction between the users and virtual objects to perform real-world tasks. The bare finger as an input device will produce a natural interaction for the users. This research implemented the finger tracking for the interaction with virtual objects in the AR system. This provides an AR system that allows the user to interact with the virtual objects with bare finger.

1.1 Background

AR is a deviation of Virtual Reality (VR) (Azuma, 1997). AR is a technoloogy that allows users to see, hear, feel, and smell the virtual objects, which are integrated in the real world (Bonsor, 2001). AR technology generates 2D and 3D computer graphics which are accurately integrated into the real environment (Malik, 2002). AR technology is implemented on wearable computers (Starner, Mann, Rhodes, Levine, Healey, Kirsch, Picard, & Pentland, 1997). Wearable and mobile computers, such as wristwatch, laptop, palmtop, tablet PC, head-mounted display (HMD), and head-up display (HUD) can be worn by the users. Currently, AR is used in medicine, repair and maintenance, labelling, robotics, military, entertainment, education, games, and other fields (Azuma, 1997; Billinghurst,

2002; Thomas, 2003). Besides that, AR will be one of the technologies which emerges with the phone technology in future (Crago, 2008).

The biggest problem for AR is the registration problem (Azuma, 1997; Zhou, Wang, Yan, & Xu, 2000). Computer vision and image processing techniques are used in AR to solve the registration problem (Azuma, 1997; Klinker, 1999). Computer vision is one of the Artificial Intelligence (AI) fields. The goal of computer vision is to analyse and interpret images (Schiewagen, 2001). Besides that, computer vision also simulates human being's visual perception (Meer, Stewart, & Tyler, 2000). Perception is the conscious experience when our senses receive stimuli from the environment (Goldstein, 2005). Moreover, perception is the gateway to other cognitions such as memory, attention, reasoning, problem solving, and decision making (Goldstein, 2005). Therefore, a camera is needed to simulate our eyes to receive the information from the real environment.

Moreover, using techniques of computer vision and image processing, the finger can be used as an input device for the interaction in the AR environment (Hardenberg, 2001). For the interaction in the AR environment, wired gloves are one of the devices mostly used (Azuma, 1997). Furthermore, using the finger as an input device can create natural interaction. Natural interaction is the interaction that does not use any devices (Hardenberg, 2001).

ARToolKit is an open-source software library. It is frequently used in developing

Augmented Reality and Mixed Reality applications (Haller, Hartmann, Luckeneder, & Zauner, 2002). ARToolKit uses computer vision techniques to detect a fiduciary marker. Camera orientation and position are computed based on the detected fiduciary marker. ARToolKit provide interaction with the virtual object by using another fiduciary marker on a handheld paddle (Gordon, Billinghurst, Bell, Woodfill, Kowalik et al., 2002). However, ARToolKit does not provide natural interaction such as the bare finger.

1.2 Problem Statement

AR enhances the users' senses by augmenting a virtual layer on the real world especially visual senses (Aaltonen & Lehikoinen, 2006). Interaction is also an important aspect in AR (Azuma, 1997). The virtual objects will not only enhance the visual perception of the users, but also assist the users to accomplish the real-world tasks.

Interaction with the pure virtual information in AR is difficult (Azuma, Baillot, Behringer, Feiner, Jullier, & MacIntyre, 2001). The AR prototypes in various previous research normally used the keyboard or mouse for the interaction. Moreover, some of the AR systems used gesture recognition or tracking 6DOF (degrees of freedom) pointers. The open source ARToolKit library allows the users to use paddle to interact with the virtual object. A marker is needed on the paddle so that the paddle can be tracked (Irawati, Green, Billinghurst, Duenser, &

Ko, 2006). However, bare finger interaction with the virtual object is difficult to be implemented due to the intangibility of the virtual objects. Hence, this research has implemented several computer vision algorithms to make the bare finger interact with the virtual object.

Integration of finger tracking in AR system requires finger detection and recognition. However, the conditions of the environment will affect the accuracy of the finger detection, especially background colour and the lighting of the environment. These factors will affect the finger detection for the interaction.

Furthermore, AR system performs registration, virtual objects generation, and interaction in real-time. If any of these methods is computationally complicated and requires a lot of time for the execution, system latency will be occurred. Hence, integration of the finger tracking in AR system not only requires an optimal algorithm for registration, but also an algorithm for the finger tracking which will not affect the system latency.

1.3 Research Objective

The general objective of this research is to design and develop Augmented Reality (AR) Application Builder, which allows users to build their own AR application and the integration of bare finger interaction with the virtual objects.

The specific objects are as follows:

1. To develop AR Application Builder which allows the users to create the AR application.
2. To implement the computer vision algorithms: colour segmentation and template matching, to detect the bare finger.
3. To integrate the finger tracking with the AR applications, so that the finger can be used as an input device to interact with the virtual objects in the real environment.

1.4 Conceptual Framework

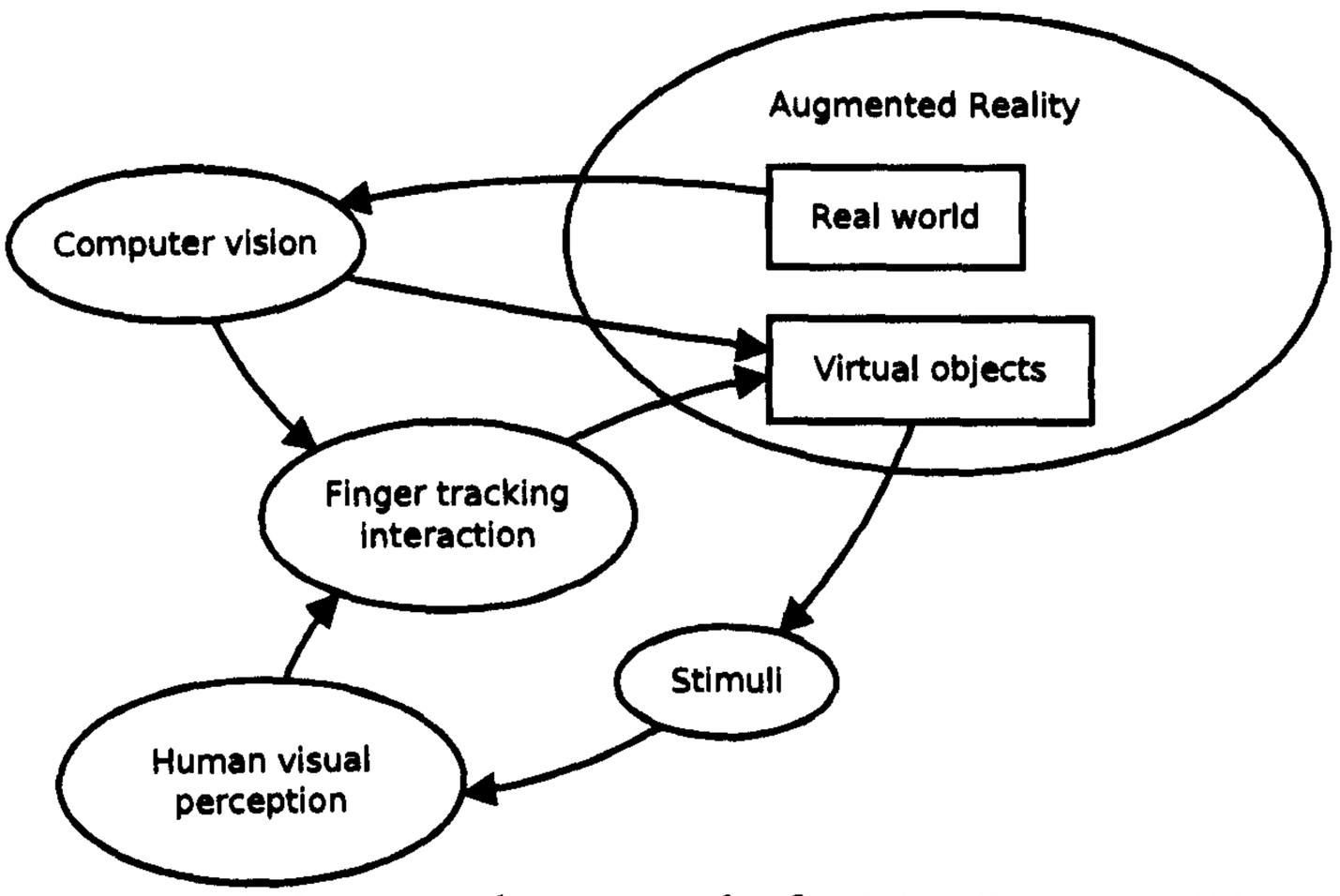

Figure 1.1: Conceptual Framework of AR Application Builder.

Figure 1.1 shows the conceptual framework of this research. AR combines the real world with the virtual objects to enhance the users' senses especially visual senses (Aaltonen & Lehikoinen, 2006). However, human visual peception is very sensitive to the error of the registration (Simon & Berger, 2000). Computer vision

algorithms which simulates human visual perception is used to understand the real world information. Consequently, the registration of the real world and virtual objects can be performed to generate correct scene to the users. The virtual objects becomes stimuli for the visual perception of the users. The users can use bare finger to interact with the augmented objects. In order to understand the interaction with bare finger, computer vision algorithms are used for finger tracking to allows the natural interaction happened in AR system.

1.5 Scope

This study focuses on the development of a computer library, AR Application Builder, which allows users to interact the virtual objects with the bare finger in an AR environment. The computer vision algorithms were implemented for the finger tracking. Furthermore, the interaction of the virtual objects were also implemented.

1.6 Significance of the Study

The key contribution of this research is the integration of the finger tracking in the AR system for interaction. There were several studies on the interactions of the AR system. Finger tracking as an input device in AR was done in previous research (Dorfmüller-Ulhaas & Schmalstieg, 2002). However, this research implemented the bare finger as an input device for the interaction in AR.

1.7 Limitations

There are several limitations in this research. The bare finger tracking did not provide any depth information. Consequently, the finger cannot interact with the virtual objects with depth information.

1.8 Definition of Terms

Several terms are used in this research. The literal definition of terms used in this research are defined as follows:

- Augmented Reality (AR) – AR is a technology that enhances the users' senses by augmenting a virtual layer on the real world (Aaltonen & Lehikoinen, 2006).
- Registration – The real and virtual worlds coexist and the objects are aligned correctly (Azuma, 1997).
- Virtual object – The computer generated objects which are superimposed on top of the video of the real world in AR system (Buchmann, Violich, Billinghurst, & Cockburn, 2004).
- Finger tracking – The finger of the user can be tracked by the system and interaction can be performed (Crowley, Berard, & Coutaz, 1995).
- Barehanded interaction – An interaction using bare hand without using any device and wires (Hardenberg & Bérard, 2001).

For the research purpose, the operational definition of terms are defined as

follows:

- Augmented Reality (AR) – Generation of virtual objects in the real environment to enhance the meaning of the real world to the users.
- Registration – A process to align the virtual object and the real environment properly in order to generate a correct scene in AR.
- Virtual object – The computer generated graphics especially 3D computer graphics which uses three-dimensional representation of geometric information in AR.
- Finger tracking – Detection of the finger so that the finger can be used as an input device in the AR system.
- Barehanded interaction – Finger as a part of the hand to interact with the virtual objects in the AR system without using any other device.

1.9 Summary

Augmented Reality (AR) Application Builder is a computer library integrated with finger tracking for the interaction with virtual objects in the AR system. The users are able to develop AR applications by using the library. The users can interact with the virtual objects by using finger as an input device. AR is a technology which augments a virtual layer on the real world especially visual senses for the users. The virtual objects are used to assist users to accomplish real-world tasks. Thus, interaction is needed between the users and the virtual objects. Interaction with the pure information in AR is difficult. This causes bare finger

interaction with the virtual object difficult to be implemented. Therefore, the objective of this research is to design and develop AR Application Builder which is integrated with the bare finger interaction with the virtual objects, and the users are allowed to use the library to build their own AR applications. The scope of this research focuses on the development of the computer library. Computer vision algorithms for finger tracking is implemented. The key contribution of this research is the integration of the finger tracking in the AR system for interaction. However, the limitations of this research is the bare finger tracking does not contain any depth information.

CHAPTER 2
LITERATURE REVIEW

2.0 Overview

This chapter discusses Augmented Reality (AR). In this research, vision-based registration for AR was used. Therefore, image processing techniques were used for vision-based registration. The image processing techniques were greyscale transformation, thresholding, labelling, and edge detection. Furthermore, template matching was used to identify the fiduciary marker. The generation of the virtual objects is also discussed. Interaction is implemented in this research. The finger is used as a natural input device in the AR system. The implementation of the finger tracking is discussed in this chapter.

2.1 Augmented Reality (AR)

Augmented Reality (AR) is a deviation of Virtual Reality (VR) (Azuma, 1997). VR is a technology that allows the user to be immersed in the virtual environment (Milgram & Kishino, 1994). The virtual environment is a simulation of the real environment using computer graphics. The simulation of the real environment enables the users to sense with the five sensory systems: visual, auditory, tactile, smell, and taste (Burdea & Coiffet, 2003). Besides sensing of the virtual world, VR system also allows the users to interact with the virtual environment.

However, there is a difference between AR and VR. AR enhances the users' senses especially visual senses by augmenting a virtual layer on the real world (Aaltonen & Lehikoinen, 2006). AR technology generates virtual objects in the real environment. Consequently, the users view is rendered in the real environment instead of virtual environment. When a person is rendering in a VR system, he or she cannot perceive any real objects since he or she is immersed inside a virtual environment generated by the VR system. Contrarily, when a person is rendering in an AR system, he or she can perceive the real objects in the real environment. Furthermore, the AR system will generate the virtual objects in the real environment. As a result, the user can perceive the real objects together with the virtual objects within the real environment. In conclusion, the AR system enhances the real world with virtual objects which exists together in the same space as the real world (Azuma, Baillot, Behringer, Feiner, Jullier, & MacIntyre, 2001).

Besides adding virtual objects, AR system also has the potential to remove the real objects by covering it using virtual background (Azuma, 1997). The ability of removing the real object from the real world is "diminished reality", which is a subset of AR (Azuma et al., 2001). The ability of diminished reality can be used to remove some information of the world. For example, a building can be removed so that the constructor can plan for the new building.

The objective of AR is to add meaning to the objects of the real environment in order to enrich the users' experiences towards the objects (EDUCAUSE Learning Initiative, 2005). AR is able to enhance education, especially the curriculum. This is because it can improve the users' experiences towards the objects by adding the meaning towards the real environment (EDUCAUSE Learning Initiative, 2005). Due to the ability of AR to add extra information, AR can be used in many fields, such as medical, aviation, training, annotation, entertainment and gaming, manufacturing and repair, and academy (Azuma, 1997; Bonsor, 2001; EDUCAUSE Learning Initiative, 2005). According to Bimber, Encarnação, annd Schmalstieg (2003), AR is able to be applied in digital storytelling. Moreover, the digital storytelling is developed in an interactive system. AR system can also be used in video conferencing (Kato, Billinghurst, Morinaga, Tachibana, 2001; Billinghurst & Kato, 2002; Billinghurst, Cheok, Kato, & Prince, 2002). The users can see each other by using fiduciary markers. The image of the users are displayed as virtual objects in the real environment. Furthermore, AR is able to be implemented in entertainment field: for example, ARQuake, a first-person outdoor

shooting game (Pierkarski & Thomas, 2002). Besides that, AR is also used to develop an AR racing game (Oda, Lister, White, & Feiner, 2008) and a AR real-time strategy game (Phillips & Pierkarski, 2005). Because the users can play the games in an outdoor environment, the implementation of AR allows users to experience games in a new and exciting way (Avery, Thomas, Velikovsky, & Piekarski, 2005). According to Cooper, Keatly, Dahlquist, Mann, Slay et al. (2004), AR can be applied in an indoor game such as Chinese checkers.

In order to implement the AR, AR system needs to incorporate with a wearable computer (Barfield & Caudell, 2001). When the user is walking in an environment, the wearable computer, which has the visual detection can capture the information of the real environment. Using the wearable computer, the computer will generate the extra information to the real environment. As a result, the user can view the computer generated information within the real environment through the wearable computer.

Generally, AR uses two types of HMD (head-mounted display). They are optical-see through HMD and video-see through HMD (Azuma, 1997; Bonsor, 2001; Ansar & Daniilidis, 2001). Optical-see through HMD allows users to see the real environment through the HMD. The users can see the real environment directly without any other media. However, video-see through HMD will display the real environment in the video form for the users in real-time. Video-see through HMD has a disadvantage that there is a delay in adjusting the image when the users

move their heads (Bonsor, 2001). This is because the real environment is captured through a camera, then the information of the real environment is sent to the video. Though AR system is running in real-time, there is a delay to send the information of the real environment to the video for the users. As a result, moving the head will affect the performance of the real-time video. In this research, video-see through method is used. This is because the application which is developed can use any video device. During the development, a web camera is used for the AR rendering.

Registration problem is the biggest problem in AR (Azuma, 1997; Zhou, Wang, Yan, & Xu, 2000). An objective of AR is to provide the users an impression that the virtual objects are part of the real environment (Liu, Storring, Moeslund, Madsen, & Granum, 2003). Registration is to join the virtual object and the real environment properly in order to generate a correct scene. As a result, users can perceive the virtual object which existed naturally in the real environment. Registration is very important because the human visual system is very sensitive to the error of the registration (Simon & Berger, 2000). Therefore, several methods are introduced to solve the registration problem so that the users can perceive the virtual object as it is in the real environment.

Knowledge-based 3D registration is one of the techniques to solve the registration problem. Using knowledge-based 3D registration, a head tracker is used to track the user's head position and orientation (Zhou, Wang, Yan, & Xu, 2000). By using

the trackers, the user's head position and orientation can be used to calculate the relation between the virtual object. Consequently, the registration can be solved. However, the tracking devices will cause the system latency and lack of accuracy (Zhou, Wang, Yan, & Xu, 2000).

Image processing based 3D registration is another technique to solve the registration problem. Using image processing, the information of the image from the real environment is extracted. The objects of the real environment are identified and then the position and orientation of the user's head are registered using the information of the processed image. The advantage of using image processing for registration is that the whole system is flexible. Nevertheless, image processing uses vast computation which will cause latency of the system (Zhou, Wang, Yan, & Xu, 2000).

Besides that, vision-based registration is another technique to solve the registration problem (Koller, Klinker, Rose, Breen, Whitaker et al., 1997). A fiduciary marker is placed on the real objects in the real environment. The fiduciary marker is recognised using computer vision algorithm. Computer vision is a method to understand the environment from visual input (Grimson & Mundy, 1994). Computer vision algorithm is able to solve the registration problem (Azuma, 1997; Kato, Billinghurst, Poupyrev, Immato, & Tachibana, 2000). Computer vision is a challenging method in solving the registration problem because it tracks the position and orientation of the user's head using the fiduciary

marker that is captured from the real environment (Lourakis & Argyros, 2004). Using the computer vision algorithm, the position and orientation of the user's head is calculated for the registration. Vision-based registration using fiducial mark can provide excellent performance (Zhou, Wang, Yan, & Xu, 2000).

Vision-based registration is focused on this research. Computer vision algorithms are based on the visual perception of the human beings. The stimuli of the world are received by visual sensory system. Then the human perceives and understands the information received. Similarly, computer vision algorithms allow the computer to understand the three-dimensional relationship of the virtual objects and the real environment based on a two-dimensional fiduciary marker.

2.2 *Vision-Based Registration*

ARToolKit is a prominent software toolkit for developing AR applications (Azuma et al., 2001). It is a software library which uses computer vision methods to compute the camera position and orientation (Woods, Mason, & Billinghurst, 2003). ARToolKit is written in C programming language (Slay, Thomas, & Vernik, 2002; Hamsphire, Seicher, Grasset, & Billinghurst, 2006; Ong, Chong, & Nee, 2006). ARToolKit uses vision-based registration method to solve the registration problem. Hence, fiduciary markers are needed in order to implement vision-based registration in an AR system.

To display virtual object correctly in the real environment, the relationships

between the camera and the fiducial mark is required. Therefore, system calibration is very important for accurate registration (Azuma, 1997; Kato & Billinghurst, 1999; Azuma et al., 2001). Calibration of the camera can be achieved using fiduciary marker (Kato & Billinghurst, 1999).

A square marker can be used as a fiducial mark in AR (Kato & Billinghurst, 1999; Kato, Billinghurst, Poupyrev, Immato, & Tachibana, 2000; Kanbara & Yokoya, 2002). The square marker is used to analyse the relationship between the marker coordinate and camera coordinate. Hence, the camera position and orientation can be computed. Camera coordinate indicates the position and orientation of the user's head if the HMD is equipped. The examples of 2D square markers are as follows:

Figure 2.1: Examples of 2D square markers used in ARToolKit.

The above example patterns are taken from ARToolKit. ARToolKit provides a utility to create new patterns. The markers are saved as 16x16 pixels greyscale image data (Mooser, You, Neumann, 2006). The size of the pattern image data is small so that template matching during real-time detection will not reduce the AR performance.

The image of the real environment is captured by using a camera. The image of the real environment is digitised and processed. Firstly, thresholding method is applied towards the input image. Thresholding is a method to convert the image into black and white according to a threshold value (Russ, 2007). Secondly, outline contour of the square marker is extracted. As a result, four line segments are acquired (Kato & Billinghurst, 1999). The intersections of the four lines can be used to compute the four vertices of the square marker.

The pattern within the square marker is compared with the patterns which are already stored in the system using template matching method. When the pattern is recognised, the four vertices of the detected marker are used to calculate the transformation matrix (Kato & Billinghurst, 1999). As a result, the transformation matrix from the marker coordinates to the camera coordinates can be found.

Generally, in an AR system, each marker handles a virtual object. Therefore, when a marker is partially blocked, the virtual object cannot be displayed on the screen. This is because the square marker cannot be detected and the four vertices cannot be calculated. When the AR system is used in a collaborative environment on a table with multiple markers, the markers were blocked by other collaborative users. As a result, the virtual objects cannot be displayed properly. However, the problem can be solved if the position of every fiducial marks is estimated (Kato, Billinghurst, Poupyrev, Immato, & Tachibana, 2000). Therefore, the multiple fiduciary markers with the fixed positions are required. However, during

collaborative environment, the virtual objects need to be picked or moved, since the position of the markers are fixed, the markers no longer can be picked up. Consequently, another fiduciary marker is used as a tool to pick up the virtual objects on the multiple fiduciary markers (Kato, Billinghurst, Poupyrev, Immato, & Tachibana, 2000). The fiduciary marker is placed on handheld paddle which allows the users to use it to interact with the virtual objects (Gordon, Billinghurst, Bell, Woodfill, Kowalik et al., 2002).

ARToolKit uses confidence factor in order to identify the fiduciary markers (Coquillart & Göbel, 2004; Fiala, 2005). When two or more markers match a pattern, the marker with the highest confidence factor will be identified. Furthermore, the confidence factor threshold value must be set in ARToolKit (Fiala, 2005). As a result, false detection for the marker will occur.

On the other hand, ARTag is another vision-based AR system. ARTag also uses 2D markers (Fiala, 2004). The fiduciary marker detection technique of ARTag is different from ARToolKit. ARToolKit stores the pattern in a file, but ARTag does not need any pattern file (Fiala, 2005). However, ARTag can identify 2002 markers. Besides that, ARToolKit stores the pattern file in greyscale (Mooser, You, Neumann, 2006). Contrarily, the markers of ARTag stores the marker data as black and white. The examples of ARTag markers are as follows:

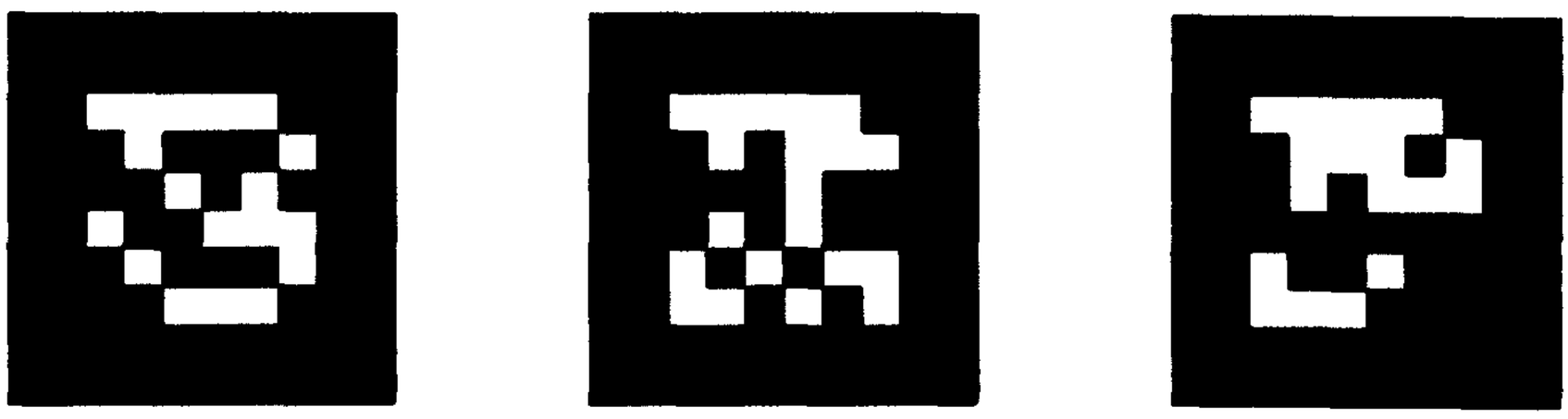

Figure 2.2: Example of markers of ARTag.

The images above are the example of ARTag markers. The sub-image within the marker is a 6x6 grid of black and white cells. Because of the characteristic of the marker, there are 2002 different types of markers available.

ARTag does not use confidence factor to identify the marker. Moreover, ARTag uses edge-based unique feature detection (Fiala, 2005). As a result, there is a very low false detection of the marker (Fiala, 2004). Besides that, if there are similar patterns in the ARToolKit system, there will be confusion when the marker is detected. This is because the detected marker might match both patterns in the system. However, using the edge-based detection method, ARTag does not have any marker confusion (Fiala, 2004).

Both ARToolKit and ARTag use vision-based registration method in AR system. Vision-based registration method requires computer vision algorithm. In order to accomplish computer vision algorithm, some image processing techniques are required (Ahn, Lehr, & Turner, 2005).

2.3 Image Processing

Image processing and computer vision are related (Ahn, Lehr, & Turner, 2005). This is because image processing techniques can be used in computer vision algorithm. Image processing is a set of algorithms to process the image in order to enhance the quality, reconstruct the image, and acquire the information of the image (Britannica Concise, 2007). There are several techniques in image processing. These techniques will be used in this research in order to solve the problem of image recognition.

Greyscale transformation is to change the colour of the images into greyscale. This is a basic process for thresholding and edge detection. Thresholding is one of the segmentation techniques that divides the images into several parts. Besides that, labelling is a technique to find the connected components of an image. Edge detection is another technique of segmentation that finds the contour of the images (Acharya & Ray, 2005).

2.3.1 Greyscale Transformation

The real environment is captured by a camera in an AR system. Thus, the colour image is captured from the real environment. Then, the colour image is being processed for the vision-based registration in AR system. In order to perform the computer vision algorithm, greyscale transformation of the image is required (Cho, Park, & Neumann, 1997).

Greyscale transformation is an image processing method that transforms the colour image into monochromatic image. The pixel components of an image consist of red, green, and blue (RGB). To transform the pixels into greyscale, red, green, and blue of every pixel should be extracted according to the proportion as 30:59:11 (Sanchez & Canton, 2003; Shrivastava, 2005; Meegoda, Juliano, & Banerjee, 2006; Parekh, 2006). Eventually, 30% of the red, 59% of the green, and 11% of the blue are combined into one value. Therefore, the formula to transform the colour image into greyscale is shown as below:

$$greyscale = \text{red} \times \frac{30}{100} + \text{green} \times \frac{59}{100} + \text{blue} \times \frac{11}{100}$$

Figure 2.3: Formula of greyscale transformation.

Since the colour image contains three colour components in every pixel, the colour image requires more memory storage of the computer. However, when the colour image is reduced to greyscale image, each pixel of the image only stores a value. When the greyscale image is stored in 8-bit unsigned integer, the range of the greyscale values are between 0 and 255 (Hornberg & NetLibrary Inc, 2007). As a result, greyscale image requires less memory storage.

Besides that, the colour image is transformed into greyscale image because the colour information is not needed in other image processing. Colour information of the image is used when the objective of the computer vision is colour related, such as colour segmentation, skin segmentation, high-level object recognition.

However, since vision-based registration AR system detects the existence of the fiduciary marker, greyscale image is preferred.

Furthermore, AR requires real-time rendering, the latency of the system will reduce the rendering performance. When the colour image is used, more memory is required. As a result, more time is needed to manipulate the image for detection. Thus, this will affect the rendering performance. Contrarily, when the greyscale image is used for AR rendering, the rendering performance in real-time will improve.

2.3.2 Thresholding

After the colour image is greyscale transformed, image segmentation is applied towards the greyscale image (Cho, Park, & Neumann, 1997). Image segmentation is one of the image processing techniques. The objective of image segmentation is to divide the image into meaningful parts (Goshtasby, 2005). So that, the divided parts can be extracted and studied. In order to implement image segmentation, thesholding is one of the methods used. Thresholding is one of the computer vision algorithm to implement the vision-based registration (Kato & Billinghurst, 1999).

After thresholding is applied towards an image, the result image will become binary image. The binary image is an image consisting of black and white with binary value. Therefore, the greyscale information of the image is removed.

However, a threshold value is needed so that the the greyscale value can be divided into black or white.

The algorithm to perform thresholding towards a greyscale image is as follows:

1. Define a threshold value, T,
2. Search through all the pixels of the image,
3. If a pixel value is greater than T, then the pixel will be changed to black (or white),
4. Else, the pixels will be changed to white (or black).

The greyscale value which is greater than the threshold value will be converted into black. Contrarily, the greyscale value which is lesser than the threshold value will be converted into white. The result of thresholded image is as follows:

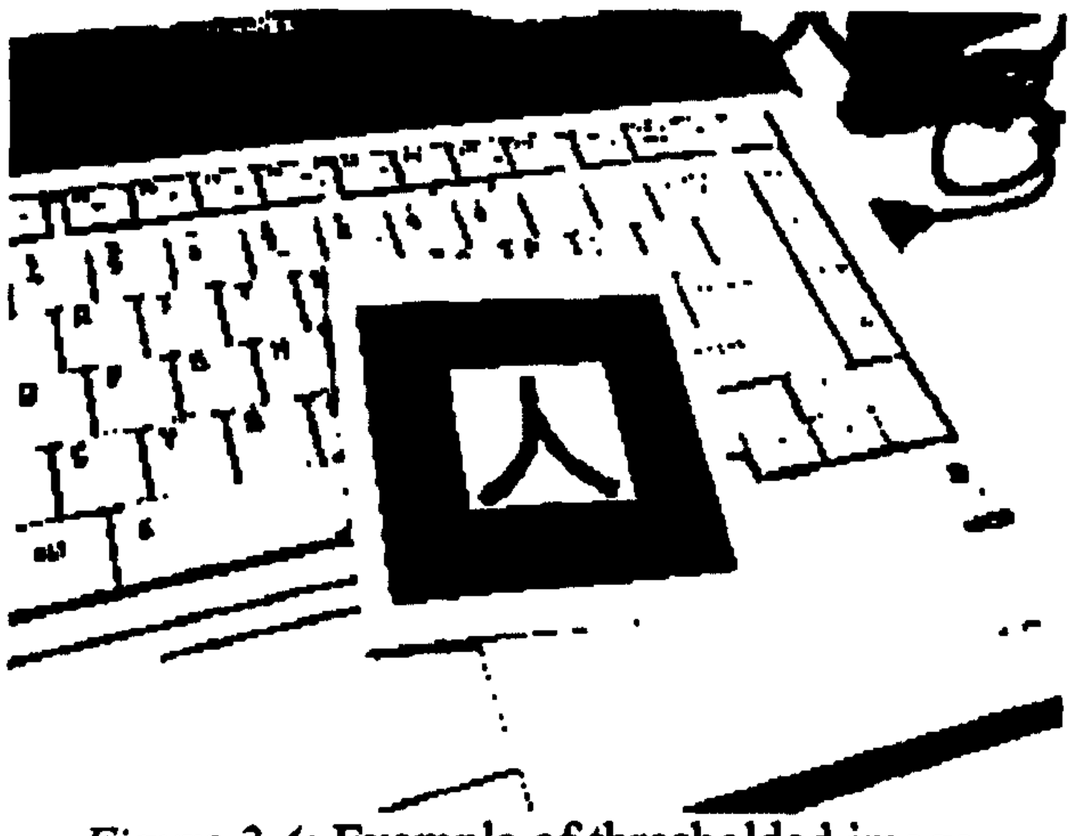

Figure 2.4: Example of thresholded image.

Figure 2.4 shows the result of thresholding. The image is turned into black and white with the binary value, 0 and 1. Furthermore, the image above shows the

existence of a fiduciary marker. It is easily perceived with human eyes. However, further image processing is needed in order to extract the information of the fiduciary marker.

The default threshold value of the ARToolKit is 100 (Barakonyi, Fahmy, & Schmalstieg, 2004). Besides that, different theshold value will result different image. The resulting image might have too many black regions or too many white regions. Therefore, an adaptive thresholding algorithm is developed for ARToolKit (Pintaric, 2003).

Adaptive thresholding algorithm will adjust the threshold value in the real-time. Furthermore, when the marker is intervened by occlusions which blocks the lighting, the adaptive thresholding algorithm is able to modify the threshold value. However, adaptive thresholding algorithm targets only one marker instead of multiple markers.

Further processing methods are applied to the binary image so that the registration of the AR system can be implemented.

2.3.3 Labelling

After the thresholding is implemented towards the image, a binary image is acquired. Then, labelling method is implemented towards the binary image. Labelling is a basic task of computer vision (Chaudary & Aggarwal, 1991).

Labelling is a technique which is used in AR system (Kawano, Ban, & Uehara, 2003).

The objective of labelling is to find the connected regions of the image (Rekimoto & Ayatsuka, 2000). As a result, the square marker can be differentiated from the environment. Moreover, the pattern within the fiduciary marker can be separated from the square marker. Therefore, the pattern within the square marker can be extracted for template matching. The pattern within the square marker is used to identify the fiduciary marker, so that the related virtual object can be generated on the identified marker.

The main task of connected component labelling algorithm is to locate the connected components of the binary image and assign a unique label to every connected component (Mehta & Sahni, 2005). In order to implement connected component labelling, every pixel of the binary image is accessed. Since the binary image is used, the pixels of the image which has the value 1, is assigned with a unique label (Pitas, 2000). So that the current accessed pixel will be labelled as the connected regions. The 8-neighbourhood of the accessed pixel is checked for the connectedness of the other labelled regions. Otherwise, a new unique label is assigned to the pixel. Consequently, the pixel will form a new region.

Labelling technique is a computer vision algorithm. It is commonly used in augmented reality (Kawano, Ban, & Uehara, 2003; Mehta & Sahni, 2005;

Kerdvibulvech & Saito, 2008). After the connected component labelling is implemented, the contour of the square marker needs to be extracted.

2.3.4 Edge Detection

Edge detection is applied to the binary image with the connected component labelling (Zhang, Fronz, & Navab, 2002; Wagner & Schmalstieg, 2007). Using edge detection, the contour of the square marker can be extracted. As a result, the four lines and four vertices of the square marker can be calculated. The four vertices are important for calculating the transformation matrix of the camera. Eventually, the registration of the AR system is completed.

The result of edge detection is as follow:

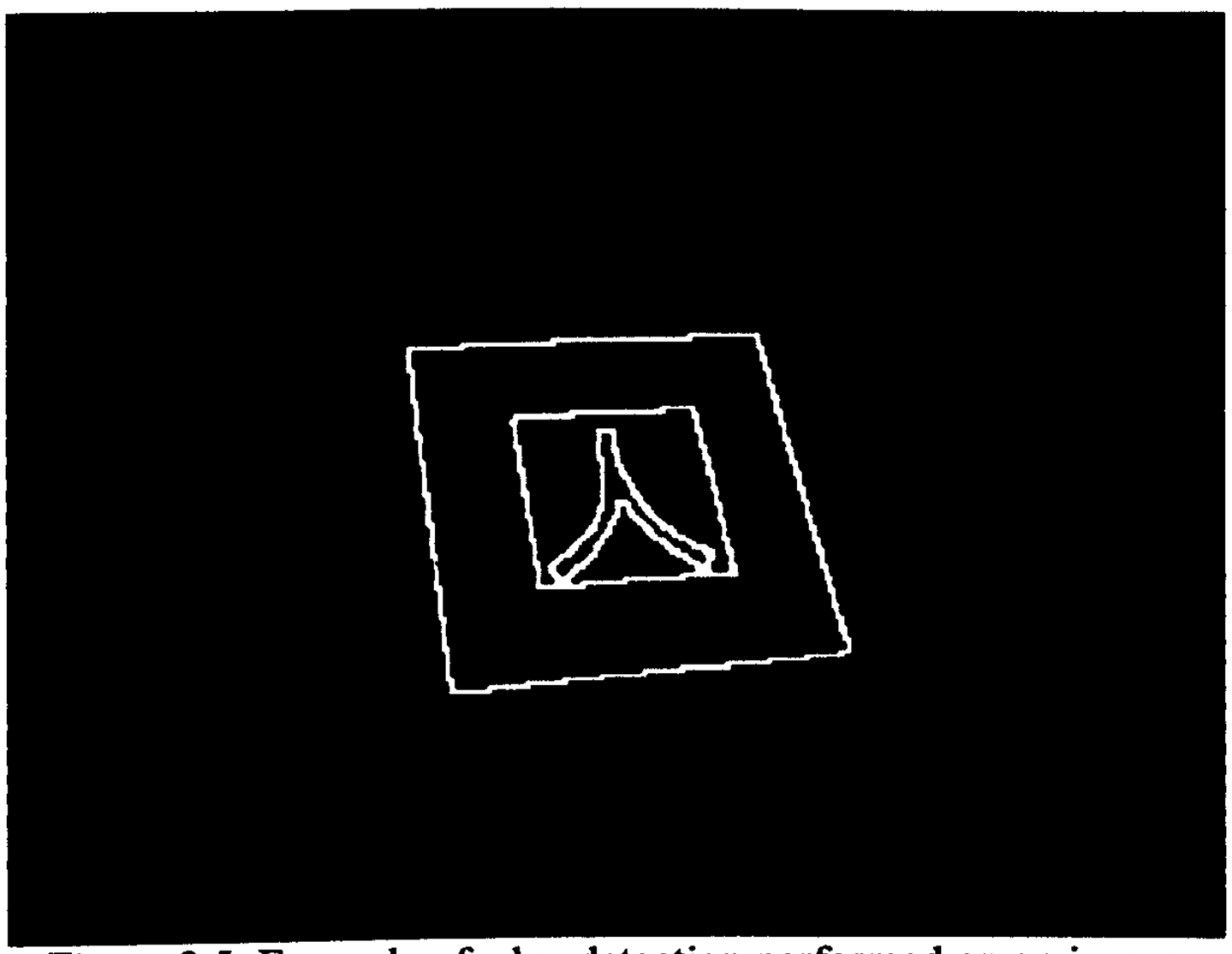

Figure 2.5: **Example of edge detection performed on an image.**

Figure 2.5 shows that the contour of the marker is extracted. As a result, four lines of the square marker can be known. The intersections of the four lines consists of four vertices which are useful for AR registration.

There are several types of edge detection methods. The commonly used edge detection methods are Robert operator, Sobel operator, Prewitt operator, Canny operator, and Krisch operator (Acharya & Ray, 2005). Sobel operator and Prewitt operator are frequently used in image analysis (Woods, 2006). Furthermore, Sobel operator is used in AR system for edge detection (Kawano, Ban, & Uehara, 2003).

Sobel operator is a 3x3 gradient operator (Acharya & Ray, 2005). Sobel operator uses two types of convolution masks. The convolution masks are as follows:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \text{ and } \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

The above convolution masks are horizontal convolution mask and vertical convolution mask. Since Sobel operator is a gradient operator, the gradient components need to be calculated. The formulae of the calculation of gradient component are as follows:

$$G_x=[f(i-1,j-1)+2f(i-1,j)+f(i-1,j+1)] -[f(i+1,j-1)+2f(i+1,j)+f(i+1,j+1)] \text{ and}$$

$$G_y=[f(i-1,j-1)+2f(i,j-1)+f(i+1,j-1)]$$
$$-[f(i-1,j+1)+2f(i,j+1)+f(i+1,j+1)]$$

The above formulae are the calculations for the gradient components. The first formula is the horizontal gradient component, and the second formula is the vertical gradient component. Next, the gradient magnitude can be calculated using the formula as follow:

$$G[f(x,y)]=\sqrt{G_x^2+G_y^2}$$

Finally, a threshold value is used to determine the result of the image. When the gradient magnitude is greater than the threshold value, the current pixel can be assigned by 1. If the gradient magnitude is lesser than the threshold value, the current pixel can be assigned by 2. As a result, the contour of the image can be extracted.

Eventually, the four vertices can be calculated using the four lines of the square marker. The transformation matrix is computed and the 3D object is generated according to the identified pattern.

2.4 Template Matching

Vision-based registration AR system uses a fiduciary marker to generate the virtual object in the real environment. The fiduciary marker is designed with a

square border and a unique pattern. In order to generate correct virtual object, the pattern within the square border needs to be identified. Therefore, template matching is a technique to identify the pattern within the square border (Zhang, Fronz, & Navab, 2002; Wagner & Schmalstieg, 2007). For instance, ARToolKit uses the traditional template matching algorithm to compare the detected fiduciary marker with the stored patterns (Mooser, You, Neumann, 2006).

Template matching is a common technique in computer vision for feature-based tracking and object recognition (Thayananthan, Navaratnam, Torr, & Cipolla, 2004). Template matching technique measures the similarity between the template and the image. The algorithm of template matching is to find the data of the image which is best matched with the data of the template (Fredriksson, 2001; Fredriksson, Mäkinen, & Navarro, 2007).

Correlation is a template matching technique that finds the similarity between the template and the image (Hardenberg, 2001). Sum of squared differences (SSD) is calculated in correlation template matching. The formula of correlation template matching is shown as follow:

$SSD(x, y)=\sum_{u,v}(S(u,v)-I(x+u, y+v)^2)$, where u x v is the size of the template (S), and I is the image.

When SSD is small, there is a high correlation between the template and the

compared image. When SSD equals to zero, then the image is identical to the template. However, correlation template matching is lighting sensitive (Hardenberg, 2001).

To solve the lighting problem, normalised cross-correlation (NCC) is used. NCC divides the correlation value by the overall luminosity (Hardenberg, 2001). The formula is as follows:

$$NCC(x,y)=\frac{\sum_{u,v} S(u,v)\cdot I(x+u,y+v)}{\sqrt{\sum_{u,v} S^2(u,v)\cdot \sum_{u,v} I^2(x+u,y+v)}}$$

By using template matching, the object can be detected according to the stored template.

Template matching algorithm is used to search a pattern in an image. Thus, every pixel of the image needs to be accessed (Russ, 2007). However, since AR system involves in real-time rendering, it is impossible to access every pixel of the image in real-time. This is because accessing every pixel of the image will cause the system latency. In order to improve the AR rendering performance, a square marker is used. This is because, after the square marker is detected, the pattern within the border is extracted for template matching. Therefore, template matching algorithm does not need to search the pattern from the whole image. As a result, the pattern within the square border can be identified in real-time and the

virtual object can be generated on top of the fiduciary marker.

2.5 Virtual Objects Generation

The main purpose of AR is to create the 3D virtual objects which are realistic so that the augmented objects will integrate into the visual perception of the perceiver (Azuma, 1997). In order to achieve the purpose, shading plays an important role in computer graphics. This is because, lighting, shading, and texture will make the virtual objects become realistic. Besides that, lighting condition and surface reflection needs to be calculated in real-time during the the AR rendering. Therefore, the calculation of the lighting will affect the performance of the AR rendering. Furthermore, a mirror can be used in order to calculate the light source correctly (Kanbara & Yokoya, 2002). Thus, computer generated lighting can be reflected correctly and produce the shading which is compatible in the real environment. As a result, the virtual objects becomes more realistic.

Texture mapping is a component of computer graphics which will improve the visual perception in AR rendering. Texture mapping is a method that maps a 2D image on 3D objects so that the 3D objects will be displayed with specific texture. As a result, the surface of the 3D objects will be perceived as rough instead of completely smooth.

In addition, animations of the computer generated objects are also important in

this research. This is because some of the 3D objects should be displayed with movements, especially the animals, human beings, game characters, etc. Therefore, this research also studies how to implement the animation of the 3D objects during the AR rendering.

2.5.1 Shading

In this research, the shading of computer generated objects was studied. This is because, shading will provide depth perception for the perceiver (Hubona, Shirah, & Jennings, 2004). There are two types of shading: flat shading and vertex shading (Simpson, 2002). Flat shading appears when the lighting is reflected across the whole surface of the polygon (Simpson, 2002). In contrast, the vertex shading appears when the colour is assigned differently to every vertex of the polygon (Simpson, 2002). The basic vertex shading is Gouraud shading. Phong shading is another vertex shading which is more complex than Gouraud shading. Phong shading also assigns colour to every vertex of the polygon. Moreover, Phong shading also averages the colour of the pixels based on the neighbour pixels. As a result, Phong shading produces smoother shading than Gouraud shading (Simpson, 2002). However, Phong shading is rarely used because it requires more rendering time than Gouraud shading (Simpson, 2002).

2.5.2 Texturing

Texture mapping is another element of computer graphics. It is used to make the

virtual objects become realistic (Haerberli & Segal, 1993; Astle & Hawkins, 2004). If there is no texture mapping, the object will be drawn with the material colour without any texture on the surface. As a result, the object will be displayed with extreme smoothness of surface (Heckbert, 1986). When the virtual object is wrapped with the texture using texture mapping, the object can be drawn with small scale geometric differences (McMillian, 1997). Using texture mapping, a simple object can be drawn with roughness without modifying the shape of the object. Texture mapping can be used to create the appearance of substructure and detail over a 3D object (Finney & ebrary Inc, 2004). Therefore, in this research, texture mapping is studied to implement into the AR during rendering. When the user renders the AR environment with the texture mapping, a simple object will be drawn with detailed texture, and this will cause the object to appear more realistic.

The texture maps consist of arrays of data. Every element of the arrays of data is a "texel". There are three types of texture mappings. They are one-dimensional texture mapping, two-dimensional texture mapping, and three-dimensional texture mapping. Normally, two-dimensional texture mapping is applied towards the 3D virtual object. Therefore, a 2D image is used as a texture. This is because two-dimensional texture has width and height properties.

The textures are mapped to the polygon which is described by vertices. Thus, each vertex of the polygon is assigned with a texture coordinate. The texture coordinate is called a U,V coordinate (Adams, 2004). As a result, the programmers can

deliberately define any part of the texture to map on the 3D object.

Furthermore, there are some restrictions of using textures. The texture is limited with the size of power of two, such as 2, 4, 8, 16, 32, etc (Adams, 2004). Moreover, the width and the height of the texture are same. For instance, 128x128 or 256x256. Because of the limitation of the video cards, normally the texture with 256x256 size is used (Adams, 2004).

Because of this, this research targets on 256x256 2D texture mapping. The textures are used to map on the 3D object in order to draw the substructure of the object. As a result, the computer generated objects in the real environment will produce an integrated perception for the AR system users.

2.5.3 Animations

Animations of the 3D objects are used in AR system (Starner, Mann, Rhodes, Levine, Healey et al., 1997; Reiners, Stricker, Klinker, & Müller, 1998; Haller, 2002; Geiger, Oppermann, & Reimann, 2003; Gillet, Sanner, Stoffler, Goodsell, & Olson, 2004; Barakonyi & Schmalstieg, 2006). This is certainly the case in an interactive AR system. Examples for their use can be found in, machinery maintenance on a laser printer (Starner et al., 1997), educational field for biological processes (Gillet et al., 2004), animated agent who gives instructions (Barakonyi & Schmalstieg, 2006), animated demonstrations for repairing vehicles (Reiner et al., 1998), and animated characters as toys for children (Haller, 2002).

This is because, animations provide a series of steps which can be followed by the users. As a result, users can learn new skills from the demonstrations or instructions.

Animation is a series of successive frames and each frame is a still image. To create the illusion of motion animation techniques are used. This process works by producing successive still images (Williams, Kent, Holzman, & Williams, 1993). This research studies on how to implement the animation of the virtual objects during AR rendering. AR rendering with the animated object will enhance the AR experience. This is because users will perceive the virtual objects not only in different perspectives, but also different behaviours of the virtual objects.

Animation is one of the important components in this research. This is because some computer generated objects require animations, this is especially true of animals and human beings. In addition, animation is the essential part to make the objects become alive (Mealing, 1998). Without animations, these objects become static, thus reducing the realism of the generated objects. In order to simulate the animal's behaviour, animation is the key component for the simulation. The animal's behaviour simulation will improve the quality of the AR rendering. So that, AR rendering with the virtual animals will provide a realistic impression on the user. However, some objects do not require animation, such as furniture and buildings.

In order to perform the animations of the virtual objects, animation data is stored in the key frame (Kiss, 2002). The key frame data of the virtual objects contain two parts. They are timing and transformation. Timing will determine the speed of the frame which will be drawn during the animation. Yet, the transformation data contains the translation, rotation, and scale transform of the objects (Chen, 2003; Zhang & Liang, 2007).

Since animation data is saved as key frames, interpolation is required so that the frames between the key frames can be drawn. Interpolation is a calculation to acquire a series of frames between two key frames (Cunningham, 2007). Therefore, when two key frames are given, the frames between them need to be calculated and draw, the animations will be performed smoothly.

2.5.4 Graphic API (Application Programming Interface)

In order to implement the 3D rendering, graphic API is concerned. OpenGL, Direct3D, and Java 3D are modern graphic APIs (Shirley, Ashikhmin, Gleicher, Marschner, Reinhard, Sung et al., 2005). Java 3D is a 3D graphic API for Java platform. Since this research is developing the system on Windows platform, Java 3D is not used to render the 3D objects. Therefore, OpenGL and Direct3D are preferred in this research. Besides that, both APIs have the libraries that uses C++ programming language (Shirley et al., 2005).

Direct3D is part of DirectX suite of APIs which is developed by Microsoft

(Thorn, 2005). It is a graphic API which simplifies the drawing process. It is able to load 3D objects, implement the texture mapping, and build animations of the 3D objects. It is normally used in game development to display the 3D computer graphics of the games. Since it is developed by Microsoft, Direct3D is targeting Windows platform.

On the other hand, OpenGL is a high performance graphic system developed by Silicon Graphics (Kilgard, 1993). It allows programmers to render high quality 2D and 3D objects. OpenGL can be used to render computer graphics primitives such as points, lines, polygons, bitmaps, and images. Through these primitives, higher quality objects can be rendered. OpenGL is important in this research since the 3D objects of this research are rendered using OpenGL.

OpenGL is used in AR system (Reiner, Stricker, Klinker, & Müller, 1998; Wagner & Schmalstieg, 2005). Furthermore, ARToolKit also uses OpenGL (Geiger, Oppermann, & Reimann, 2003). Therefore, OpenGL was chosen to represent the computer generated 3D objects in this research.

2.6 Interaction

AR technology improves users perception and the interaction with the real world (Azuma, 1997). AR technology not only emphasises on the generation of the virtual objects, but also the interaction with the virtual objects. Some of the previous research studied on how users will interact with the AR applications

(Azuma, Baillot, Behringer, Feiner, Jullier, & MacIntyre, 2001). According to Azuma et al. (2001), most of the previous researchers focused on the registration of the virtual object in the real environment, but interaction of the system was less focused.

Interaction with the completely virtual information is difficult (Azuma et al., 2001). There is no tactile perception towards the virtual object. The users cannot feel the virtual object through the tactile perception, but only through the visual perception. Thus visual feedback is crucial in the interaction. Furthermore, audio feedback can also be used in the interaction, so that the interaction becomes more complete.

Though the virtual objects do not produce tactile feedback, a glove with tactile feedback can be used (Azuma, 1997). The glove with tactile feedback can simulate the real forces in the environment for the users, so that the users can feel the objects during the interaction.

Glove-based user interaction technique was applied in the AR technology (Thomas & Piekarski, 2002). A wired glove is worn by the users as an input device. Then, the users can interact with the virtual objects in the AR environment. Glove-based interaction allows the users to manipulate the AR information in real-time. Besides wired glove, special dots on the hand can also be used for interaction (Mackay, 1998). The special dots are used to assist the

detection of the hand position.

Similarly to the concept of adding special dots to the hand, Dorfmüller-Ulhaas and Schmalstieg (2001) implemented marked glove for the interaction in the AR system. Computer vision algorithms are applied to the markers of the glove to detect the finger. The real location of the finger is translated to the virtual location. Consequently, interaction with the virtual object can be performed by using the marked glove.

2.6.1 Bare Finger Interaction

There was little research on the bare hand interaction in AR (Buchmann, Violich, Billinghurst, & Cockburn, 2004). Hands are our essential medium for interaction. Thus, AR system should allow us to use our bare hands for interaction. Bare hand interaction is an interaction does not equip the glove as the input device. Bare hand interaction produce a natural way of interaction (Buckmann, Violich, Billinghurst, & Cockburn, 2004).

The interaction of the fingertips in AR system was studied in the previous research (Buckmann, Violich, Billinghurst, & Cockburn, 2004). The implementation of bare finger required computer vision algorithm and image processing methods (Letessier & Bérard, 2004). The techniques involved in bare hand and bare finger detections are colour segmentation, edge detection, connected component labelling, and image differencing (Hardenberg & Bérard,

2001). Image differencing is used for motion detection.

2.6.2 Colour Segmentation

Colour segmentation is frequently used to find the position of the objects (Comaniciu & Meer, 1997). A colour consists of red, green, and blue components. Colour segmentation method relies on the colour image instead of greyscale image. The colour components are important in colour segmentation. The object within the range of colour components can be extracted from the whole image.

The colour segmentation can be implemented in the finger tracking. This is because human skin has its own range of colour.

The effectiveness of the colour segmentation depends on the targeted colour range. Besides the RGB (red, green, and blue) colour component, there is also HSV (hue, saturation, and value) colour model (Foley, 1995). RGB model can be converted to HSV colour model by using an algorithm as follows:

Hue is measured as the angle of 360°. The hue value is based on the RGB component. The hue of the red colour is 0°, green colour is 120°, and blue colour is 240°. On the other hand, the saturation is a ratio ranging from 0 to 1. Saturation is the purity of the colour. The value is also measured in the range from 0 to 1. When the value is 0, the colour will become black. When the value is close to 1, the saturation will affect the the colour. If the saturation is 0 and value is 1, the

```
void RgbToHsv(double r,double g,double b,double *h,double
*s,double *v) {
  double maximum = max(r,g,b);
  double minimum = min(r,g,b);
  *v = max; //Value
  *s = (maximum != 0) ? ((maximum - minimum)/maximum) : 0;
  if(*s == 0)
    *h = 0; //Cannot defined
  else {
    double d = maximum - minimum;
    if(r == maximum)
      *h = (g-b)/d;
    else if(g == maximum)
      *h = 2 + (b-r) / d;
    else if(b == maximum)
      *h = 4 + (r-g) /d;

    //Convert hue to degree
    *h *= 60
    if(*h < 0)
      *h += 360
  }
}
```

Figure 2.6: Algorithm of converting RGB to HSV.

colour will be white. The HSV colour model is popular in the work on skin colour segmentation (Zarit, Super, Quek, 1999).

Colour segmentation can be applied towards the skin colour for the finger tracking. According to Kovač, Peer, and Solina (2003), a formula can be used to describe the skin colour. The formula is as follows:

$$\begin{aligned}&\text{red} > 95 \textit{ and } \text{green} > 40 \textit{ and } \text{blue} > 20 \textit{ and}\\ &\textit{max}(\text{red}, \text{green}, \text{blue}) - \textit{min}(\text{red}, \text{green}, \text{blue}) > 15 \textit{ and}\\ &|\text{red} - \text{green}| > 15 \textit{ and } \text{red} > \text{green} \textit{ and } \text{red} > \text{blue}\end{aligned}$$

The formula uses the RGB colour model. The formula above is applied towards the pixel colour of the image so that the skin can be differentiated from the other

object. When a pixel colour of the image fulfils the condition of the formula, the pixel is considered as skin. Consequently, the skin region can be extracted for the use of finger tracking. In the AR system, the algorithm for skin colour segmentation needs to be performed in real-time.

2.6.3 Finger Tracking

After the skin colour segmentation is implemented, fingers are normally targeted for the interaction. Subsequently, finger tracking is implemented as an input device. The implementation of the finger tracking involves the template matching (Koike, Sato, & Kobayashi, 2001; Oka, Sato, & Koike, 2002). The normalised cross-correlation (NCC) of the template matching for the finger tracking was used. As a result, lighting problem towards the detected finger can be reduced.

To accurately detect the finger especially the fingertips, a template of a fingertip is used (Kjeldsen, Pinhanez, Pingali, Hartman, Levas, Podlaseck et al., 2002). When one of the fingertips is tracked, the fingertip can be used as a pointer device in the AR system. The fingertip as a pointer device allows interaction such as clicking, touching, and pointing. These interactions can normally be done by a mouse device. However, for the mobility of the AR system, mouse device is not a suitable input device. Furthermore, mouse device does not produce natural way of interaction. Hence, finger is used to replace the mouse device in this research.

2.7 Summary

This research studies on the Augmented Reality (AR) technology. The main problem of the AR is the registration problem (Azuma, 1997). There are three methods to solve the registration problem (Zhou, Wang, Yan, & Xu, 2000). They are knowledge-based 3D registration, image processing based 3D registration, and vision-based registration. Vision-based registration does not produce system latency like the other two methods. Vision-based registration is focused in this research. Computer vision algorithms simulate the visual perception of the humans to understand the world information. Therefore, computer vision algorithms are used in vision-based registration in order to understand the three-dimensional relationship between the real world and virtual world. ARToolKit is an obvious software toolkit for developing the AR applications. It uses the vision-based registration. A fiduciary marker is needed for the vision-based registration. Hence, ARToolKit uses square markers as the fiduciary markers for the registration. Image processing methods are used to accomplish the computer algorithm for registration. The greyscale transformation modify the image from colour to greyscale. Thresholding is implemented to convert the image to binary format. Labelling is used to find the connected components of the image. Then, edge detection is applied to find the contour of the fiduciary marker. Edge detection finds the contour by calculating the gradients of the image. Robert operator, Sobel operator, or Prewitt operator is used to extract the contour of the image. The vertices of the square marker are used to calculate the transformation

matrix so that the 3D object can be generated correctly. The pattern of the fiduciary marker is recognised by using template matching method. The generation of the realistic virtual objects is main purpose of AR. Generation of the virtual objects involves shading, texturing, and also animations. Besides that, the interaction with the virtual object is implemented in this research. Bare finger interaction is focused to produce natural interaction. Therefore, colour segmentation towards the skin of the hand and finger is implemented. Furthermore, finger tracking is emphasised by using template matching for the fingertips. As a result, the finger can be used as a pointer device to interact with the virtual objects.

CHAPTER 3
METHDOLOGY

3.0 Overview

This chapter discusses about the research design. A combination of exploratory model and reuse model is used as the research design. There are six stages of the research design: 1) system specification, 2) collection of the objects, 3) objects construction and modification, 4) system construction and modification, 5) testing, and 6) implementation. Finally, an evaluation on the system functionality was performed.

3.1 Research Design

This research involves development of a computer library, AR Application

Builder. The computer library was applied on an AR system which allows finger for the interaction. The system development process in this research is a system development life cycle (SDLC) with the combination of reuse model (Erdil, Finn, Keating, Meattle, Park, & Yoon, 2003) and exploratory model (Center of Technology in Government, 1998).

Reuse model refers to the usage of existing program components and modifying them so that they are able to be used in the new system (Erdil et al., 2003). Reuse model is suitable for object-oriented programming. C++ language was chosen for the system development of this research. C++ is an object-oriented programming language. Besides that, ARToolKit which was written in C language can be used with C++. ARToolKit was reused in this research in order to perform registration of the virtual objects and the real world. ARToolKit contains the computer vision algorithms to detect and recognise the fiduciary markers for registration.

Exploratory model was used when the requirements of the system were difficult to be identified at the beginning of the project (Center of Technology in Government, 1998). Exploratory model can be used in the research of Artificial Intelligence (AI). Exploratory model is able to be applied in this research because this research involved computer vision, since computer vision is a subfield of AI. Besides that, finger tracking for natural interaction in AR system required computer vision algorithms to detect and recognise the finger. However, ARToolKit did not provide feature of finger tracking for interaction. This research

uses exploratory model to develop the finger tracking algorithm based on computer vision. Skin segmentation and template matching uses normalised cross-correlation (NCC) was implemented to detect and recognise the fingertip of the users. The combination of reuse model and exploratory model which includes the cognitive approach in SDLC constitutes an SDLC-Cognitive-Intuitive approach in this research. The computer vision simulates the visual perception of human. This allows the system to understand the real world as human perceiving the stimuli of the real world. Furthermore, the natural interaction with bare finger of the users towards the virtual objects in the AR system is a response of human towards the stimuli of the world.

There are six stages in the research design: 1) system specification, 2) collection of the objects, 3) objects construction or modification, 4) system construction or modification, 5) testing, and 6) implementation. System development process in this research is shown as follows:

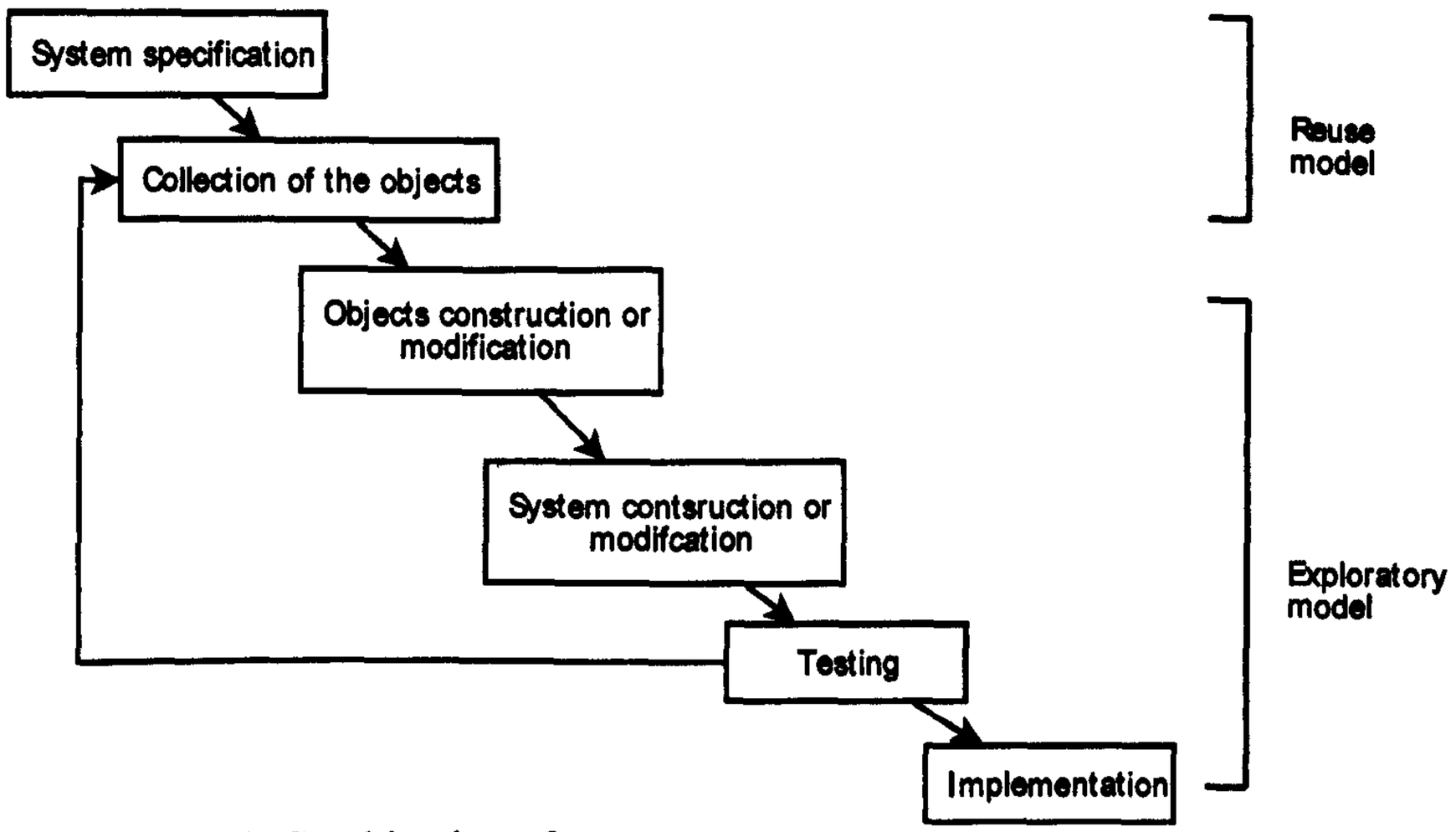

Figure 3.1: Combination of reuse model and exploratory model for system development process.

Figure 3.1 shows the stages of this research. Firstly, the system specification is outlined. Secondly, according to the specification, the objects or the libraries of software modules are collected. Then, the collected objects are modified and other objects are built. Next, the system is constructed by using these objects. After the system is constructed, it is tested by the developer. After the stage of testing, the stages from the objects collection are repeated. The repetition is ended when the desired outcome is produced. Lastly, the system is implemented.

3.1.1 System Specifications

In the first stage, system architecture of the research was designed as follows:

Figure 3.2 shows that the video input is retrieved through the DirectShow with the video capture device. The DirectShow library can also load the video files. Then

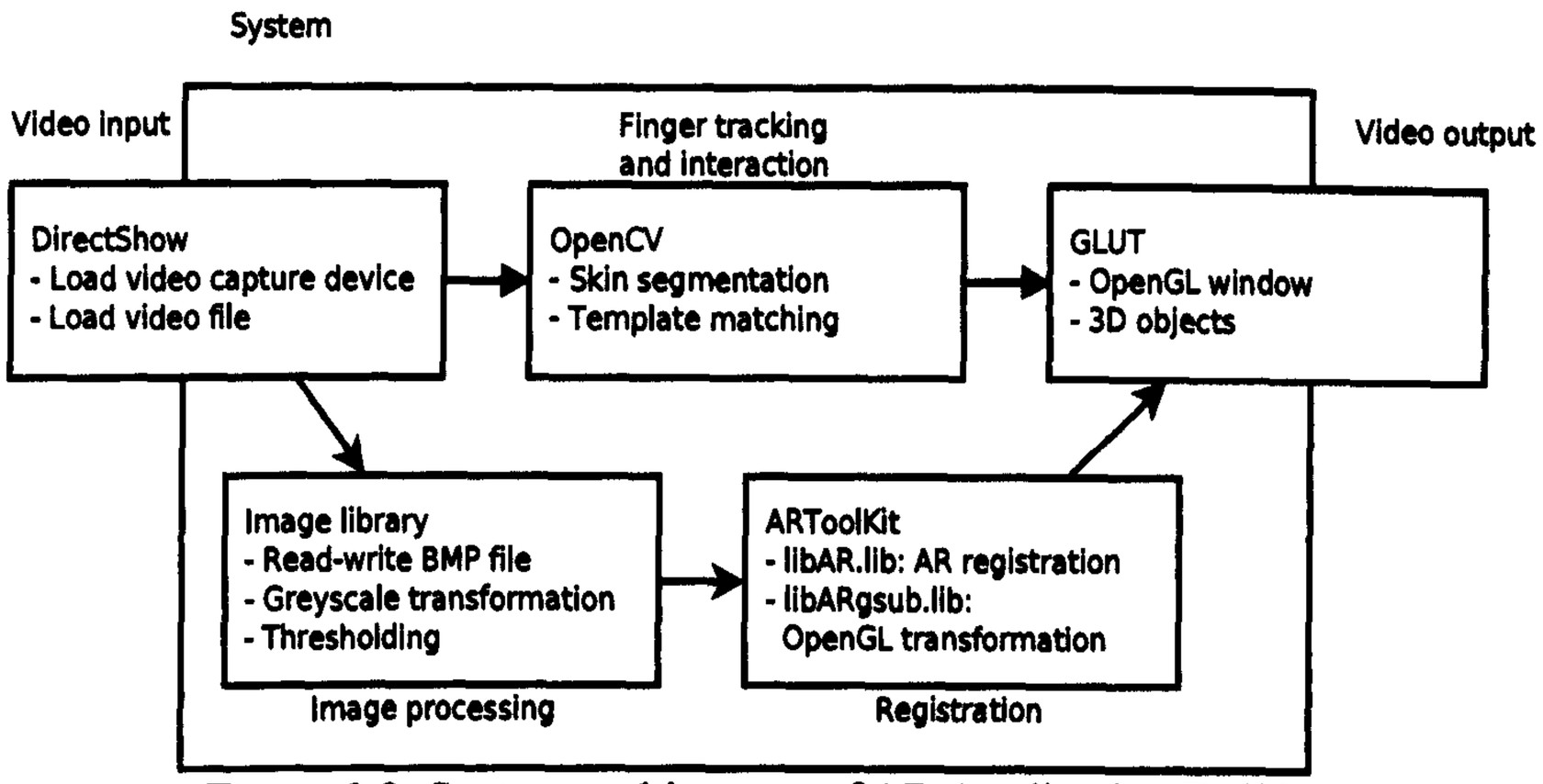

Figure 3.2: System architecture of AR Application Builder.

the video stream from DirectShow is processed by using the image library. Greyscale transformation and thresholding were used to process the video stream. Next, the processed video stream was used for AR registration. ARToolKit libraries were used in this research for the AR registration. The video and virtual objects were displayed through the OpenGL using the GLUT. OpenGL was used because it is able to draw the 3D computer graphics as the augmented objects on the video stream. Besides that, finger tracking was performed by using OpenCV library. Finger detection was performed on the input video stream. Then the finger information was used for interaction with the virtual objects in the real environment. OpenCV contains computer vision algorithms that are required in this research, especially template matching with sum of squared differences and normalised cross-correlation.

Technology requirements of AR Application Builder were outlined as follows:

- Visual Studio C++ Express 2008

- Windows SDK for Windows Server 2008 and .NET Framework 3.5
- DirectX SDK June 2008 or above
- OpenGL Utility Toolkit (GLUT)
- ARToolKit 2.72
- OpenCV 1.1 pre 1

Since this research is targeted for the Windows platform, Microsoft Visual C++ 2008 Express was used as an IDE (Integrated Development Environment). It is a free IDE and contains Microsoft C/C++ Compiler version 9.0. In order to develop Windows application, Windows SDK 6.1 (Windows SDK for Windows Server 2008 and .NET Framework 3.5, 2008) was used. It contains the Windows libraries and header files for developing Windows application. DirectShow was used to enumerate the video capture device. GLUT was used to create OpenGL compatible window to represent 3D computer graphics during the rendering of the AR environment. ARToolKit which contains vision-based registration algorithm was used to draw virtual objects on top of the fiduciary marker. OpenCV was used to detect and recognise finger for finger tracking interaction.

Next, functional requirements of AR Application Builder were outlined as follows:

- Video capture device enumeration
- Vision-based registration for AR
- Image processing

- Finger tracking
- 3D computer graphics generation
- Interactions

Finally, hardware requirements of AR Application Builder were also outlined. The following list is the hardware requirements of AR Application Builder:

- Graphic card which supports OpenGL
- A video capture device such as a web camera is needed to capture the image of the real environment.

3.1.2 Collection of the Objects

At the stage of collection of the objects, relevant libraries were collected for the development of the system. The libraries which were essential in this research were DirectShow, GLUT (OpenGL Utility Toolkit), ARToolKit, and OpenCV.

DirectShow is one of the components of DirectX. DirectShow is an application programming interface (API) that can perform multimedia playback and capturing. DirectShow is available in Windows SDK 6.1. In this research, DirectShow was used for video capturing and open the video files.

OpenGL (Open Graphics Library) header and library files are available in Windows SDK 6.1. GLUT was used as the utility toolkit to create the windows

that is compatible with OpenGL. Besides that, GLUT provides function to create some primitive 3D objects like cube, cone, sphere, and teapot.

ARToolKit is an open source AR library under GPL license. This library was used to implement the vision-based registration. Hence, square markers are needed for the registration. Two ARToolKit libraries were used: libAR.lib and libARgsub.lib. libAR.lib is used for AR registration and calculation of the transformation matrix. libARgsub.lib is used to initialise the OpenGL setting so that it compatible to the AR registration.

OpenCV contains several computer vision algorithms written in C programming language. OpenCV was used in this research to implement the computer vision algorithms such as edge detection, image segmentation, connected components labelling, and template matching. These algorithms were used for finger tracking and the interaction of the virtual object in AR environment.

3.1.3 Construction and Modification of the Objects

At this stage, several objects were constructed in this research for the needs of the system. Besides that, some collected libraries were modified.

A DirectShow wrapper library was constructed. The library wrapped the DirectShow functions into a class (object) so that it is compatible to the C++ programming language. Furthermore, enumeration of the video capture device

was implemented in the class to load a video capture device such as a web camera or a digital video camera. Not only the enumeration of the video capture device, the class is also able to load the video files such as AVI and MPEG.

An image library which can read write BMP file format was constructed. The algorithm of reading and writing BMP file format was based on Bernard (2003) and Sarath (2007). The image library was built with image processing functions: greyscale transformation and thresholding. Greyscale transformation is essential in computer vision because it reduces the information of the image since the colour information is not used in AR registration. Thresholding is performed after the greyscale information to make the image becomes segmented. The segmented image provides the information of the fiduciary marker. Consequently, the fiduciary marker can be extracted by using ARToolKit libraries.

The ARToolKit libraries were modified so that they are compatible to the system of this research. This is because ARToolKit uses DSVL (DSVideoLib) library to load the video capturing. However, the DSVL was replaced by DirectShow wrapper library in this research, therefore modifications on the libAR.lib were needed.

Furthermore, a computer graphics library was also built. The computer graphics library contains the algorithm for shading, texturing, and animations. These algorithms were used to generate the virtual objects as the augmented objects in

the AR system.

3.1.4 System Construction and Modification

At the stage of system construction or modification, the libraries which were collected, modified, and constructed were used to develop the AR Application Builder. The flow of the system is illustrated as follows:

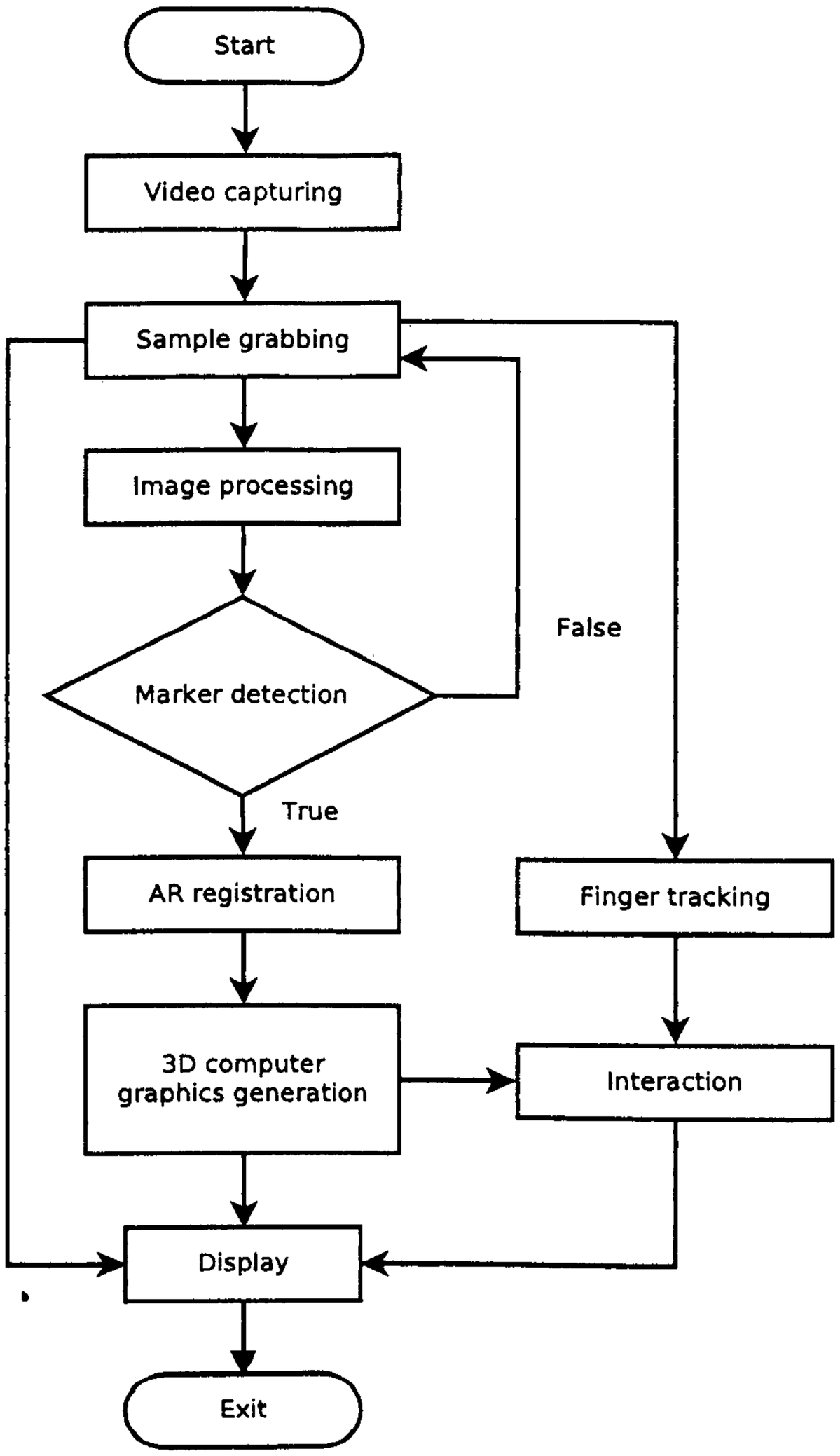

Figure 3.3: Flowchart of the implementation of AR Application Builder.

Figure 3.3 illustrates that the video is captured, and the video sample is grabbed. Then, image processing is performed. Next, fiduciary marker is detected. If the marker is detected, the AR registration is performed and 3D computer graphics are generated. These were displayed as the video output. At the same time, the finger is detected for the finger tracking. Then the interaction with the computer

generated objects is performed.

3.1.5 Testing

After the system was constructed, the system was also tested. The debugging was carried out to fix the bugs of the system. Besides that, the functionalities of the constructed libraries were tested to accomplish the development of the AR Application Builder.

3.1.6 Implementation

At the implementation stage, AR Application Builder, a computer library was built by integrating all the functions and libraries of this research study. Besides that, the AR Application Builder was also used for developing some AR applications for education, tourism, gaming, and entertainment.

3.2 Evaluation

After AR Application Builder was built, the system functionality evaluation was carried out. According to previous research studied by Avery, Thomas, Velikovsky, and Piekarski (2005), 44 participants were used to evaluate the user satisfaction and learnability towards an AR system. Besides that, a research studied by Kerdvibulvech and Saito (2008) used 15 participants to collect the qualitative feedback towards an AR system. Evaluation method of this research

was based on qualitative method to ensure that AR Application Builder provided the required functionalities towards the users.

30 students from the Faculty of Cognitive Sciences and Human Development (FCSHD) of Universiti Malaysia Sarawak (UNIMAS) were selected as the participants. They were selected in this research because of their computer proficiency. The participants were given consent forms before the evaluation. The consent form is attached in Appendix 1. The evaluation was conducted on 8 September, 2008, at the VR Lab in FCSHD, UNIMAS.

An AR application was built by using AR Application Builder for the evaluation. The AR application generates a video player with the play, pause, and stop buttons on the square marker. The users are able to use their finger to "click" the play, pause, and stop buttons and view a video in the AR environment.

The participants were given the tasks for as follows:

1. Use a web camera to detect the marker.
2. Use the finger to click play button to start playing the video in AR system.
3. Click pause button to pause the video.
4. Click play button to resume the video.
5. Click stop button to stop the video.
6. Click play button to play the video from the beginning.
7. Watch the video completely.

The functionality of the finger tracking for interaction was evaluated. The participants were given a form of evaluation for the system functionality. The form of evaluation are attached in Appendix 2.

3.3 Summary

System development process of this research is a combination of reuse model and exploratory model. Exploratory model on AI field which uses computer vision algorithms involves cognitive theory such as visual perception. As a result, the combination of reuse model and exploratory model constitutes SDLC-Cognitive-Intuitive approach in this research. There were six steps in system development process in this research: 1) system specification, 2) collection of the objects, 3) objects construction or modification, 4) system construction or modification, 5) testing, and 6) implementation. The system was targeted on Windows platform. C++ programming language was used. Several libraries were collected. They were DirectShow, GLUT (OpenGL Utility Toolkit), ARToolKit, and OpenCV. DirectShow was used for video capturing. GLUT was used with OpenGL to draw the 3D computer graphics. ARToolKit was used for vision-based registration. OpenCV was used for finger tracking and interaction. Besides that, ARToolKit libraries were modified for the system compatibility. An image library was built for reading and writing the BMP file. The image library contains some image processing functions such as greyscale transformation and thresholding. Besides that, another library which contains the algorithms of shading, texturing, and

animations was also built. Then, the system architecture was designed. Next, the system was tested with the functionalities. Finally, AR Application Builder, a computer library was built by integrating all functions and libraries together.

Furthermore, the evaluation was carried out to test the system functionality of the finger tracking for interaction with the virtual object in an AR application. The application was built by using AR Application Builder. The participants were given consent forms. They were given the tasks for using the finger to interact with the virtual object in AR environment. The results of the evaluation were collected for further analysis.

CHAPTER 4
DESIGN AND DEVELOPMENT

4.0 Overview

This chapter discusses how the AR Application Builder was developed in this research project. Video capturing by using DirectShow is discussed. Development of the image library which contains image processing functions is explained. Next, the generation of the virtual objects by using OpenGL and GLUT is also discussed. Finally, the implementation of finger tracking by using OpenCV is discussed.

4.1 Video Capturing with DirectShow

A video capturing device such as web camera could be used as a capture device

for this research. DirectShow was needed in order to acquire the input data from a web camera.

Filters are the basis of DirectShow that performs some operations towards the multimedia stream (Sink, 2001). Filters are connected with each other through the pins. There are many types of filters. A source filter is used to read a source stream, then the stream data will go from the output pin of the source filter to the input pin of the render filter. The source stream can be a video file, an audio file, a video stream from a capture device, or a video stream from the internet. Next, some operations can be performed towards the input stream. The input stream can be split into audio data and video data, converted to another format, compressed, decompressed, or finally rendered. These operations depend on the filters that are used. The render filter can be used to display the stream data as a video or audio depends on the source stream. The images below shows the conceptual diagram of filters and pins in DirectShow.

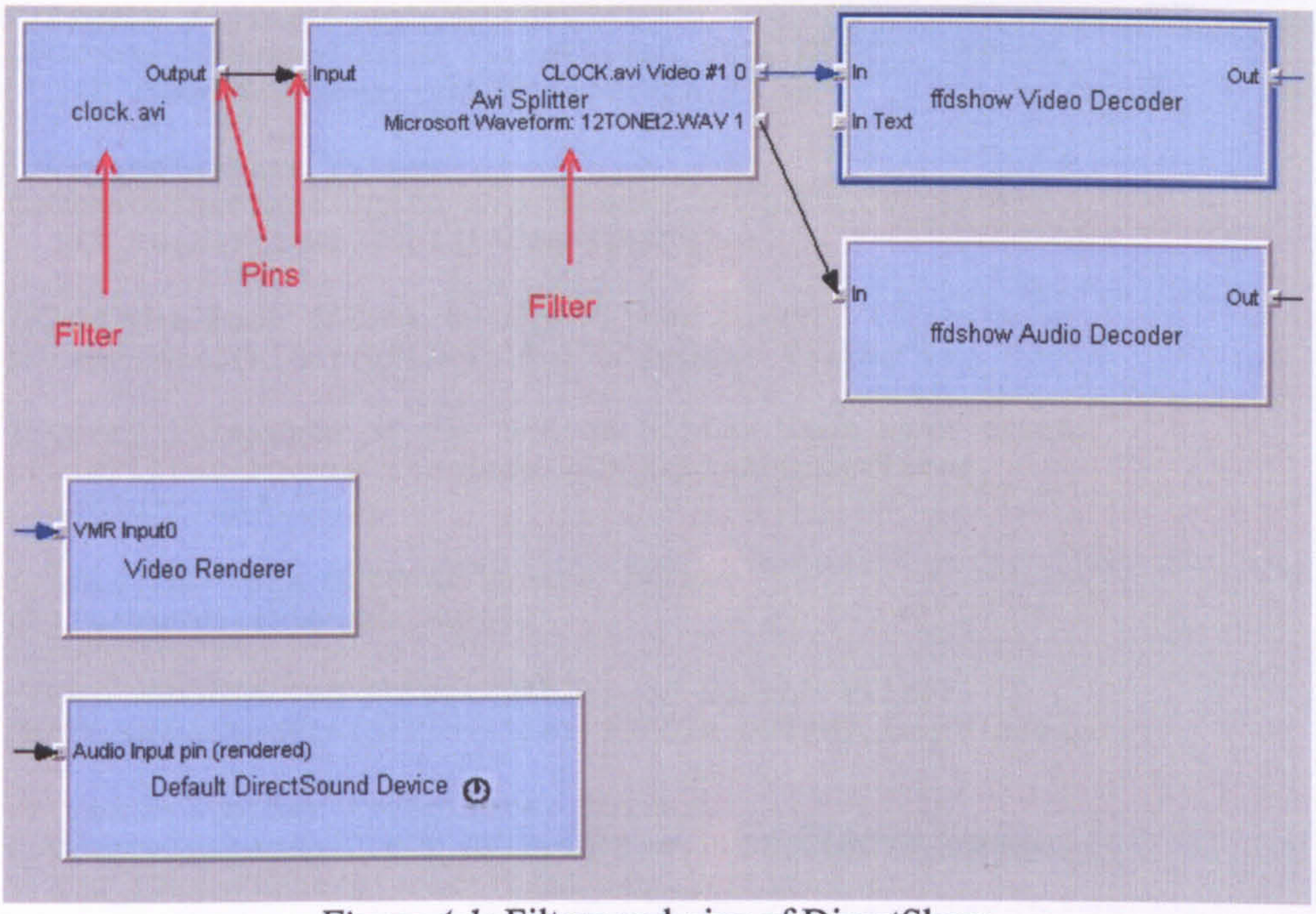

Figure 4.1: Filters and pins of DirectShow.

Figure 4.1 shows that a "clock.avi" file is a source stream. Then the data of the source stream flows from an output pin of the source filter to the input pin of a filter called "Avi Splitter". Next, since the stream is split into video and audio, the video stream flows to "ffdshow Video Decoder" filter while the audio stream flows to "ffdshow Audio Decoder" filter. Finally, both streams are decoded and continue flow to "Video Renderer" filter for the video stream and "Default DirectSound Device" filter for the audio stream. Using the "Video Renderer", the video of the "clock.avi" is displayed on the screen. Likewise, the "Default DirectSound Device" filter is used to play the sound of the "clock.avi" file. The algorithm of DirectShow is shown as follows:

```
//Create Filter Graph Manager
CoCreateInstance(CLSID_FilterGraph,NULL,CLSCTX_INPROC,
  IID_IGraphBuilder,(void**)&pGraph);

//Create a base filter
CoCreateInstance(CLSID_AsyncReader,NULL,CLSCTX_INPROC,
  IID_IBaseFilter,(void**)&pFileFilter);

//Add the base filter to the Filter Graph
pGraph->AddFilter(pFileFilter,L"Source Filter");

//Query interface of the Source Filter from base filter
pFileFilter->QueryInterface(IID_IFileSourceFilter,
  (void**)&pFile);

//Load the file through Source Filter
pFile->Load(filename,NULL);

//Get unconnected output pin(s) of Source Filter
GetUnconnectedPin(pFileFilter,PINDIR_OUTPUT,&outFilePin);

//Create another AviSplitter filter
CoCreateInstance(CLSID_AviSplitter,NULL,CLSCTX_INPROC,
  IID_IBaseFilter,(void**)&pAviSplitFilter);

//Add AviSplitter filter to the Filter Graph
pGraph->AddFilter(pAviSplitFilter,L"AVI Splitter");


//Get unconnected input pin of AviSplitter Filter
GetUnconnectedPin(pAviSplitFilter,PINDIR_INPUT,&inSplitPin);

//Connect the output pin of Source filter with the input pin of
//AviSplitter filter
pGraph->Connect(outFilePin,inSplitPin);

//Get the output pins of AviSplitter filter
GetPin(pAviSplitFilter,PINDIR_OUTPUT,0,&outSplitPin1);
GetPin(pAviSplitFilter,PINDIR_OUTPUT,1,&outSplitPin2);

//Render both pins
pGraph->Render(outSplitPin1);
pGraph->Render(outSplitPin2);
```

Figure 4.2: Snippet of creating filters using DirectShow.

The coding above shows that the filters are added and the pins are connected for the video streaming. In this research, the capture device such as web camera was enumerated and it was used as the source stream. The real environment was captured as the video stream for AR registration. Enumeration of the video

capture device was implemented. The coding of the enumeration of the video capture device is shown as follows:

```
//Create Capture Graph Builder 2, required for capture device
CoCreateInstance(CLSID_CaptureGraphBuilder2,NULL,CLSCTX_INPROC,
  IID_ICaptureGraphBuilder2,(void**)&pCapture);

//Create Filter Graph
CoCreateInstance(CLSID_FilterGraph,NULL,CLSCTX_INPROC,
  IID_IGraphBuilder,(void**)&pGraph);

//Set Filter Graph for Capture Graph Builder 2
pCapture->SetFiltergraph(pGraph);

//Enumerate a filter for the capture device
FindCaptureDevice(&pSrcFilter);

//Add the capture device to the filter graph
pGraph->AddFilter(pSrcFilter,L"Video Capture");

//Continue with adding other filters, and connect the filters
//through the pins. Sample Grabber filter and Null Renderer
//can be added. Finally, render the pins.
```

Figure 4.3: Snippet of video capture device enumeration in DirectShow.

The coding above shows the algorithm of video capture device enumeration. The video capture device was used as the source filter in this research for the AR system. The frames of the video stream were grabbed for the image processing and AR registration.

4.1.1 Sample Grabbing

By using DirectShow, a video file or video capture device can be played. The image data of the video stream needs be grabbed for the image processing and AR registration, so that the 3D objects can be drawn on the video. DirectShow

provides a filter, Sample Grabber, which can be used to grab the image of the video stream. The Sample Grabber filter is added after source filter to grab the image data. The image below shows the conceptual diagram of the Sample Grabber in DirectShow:

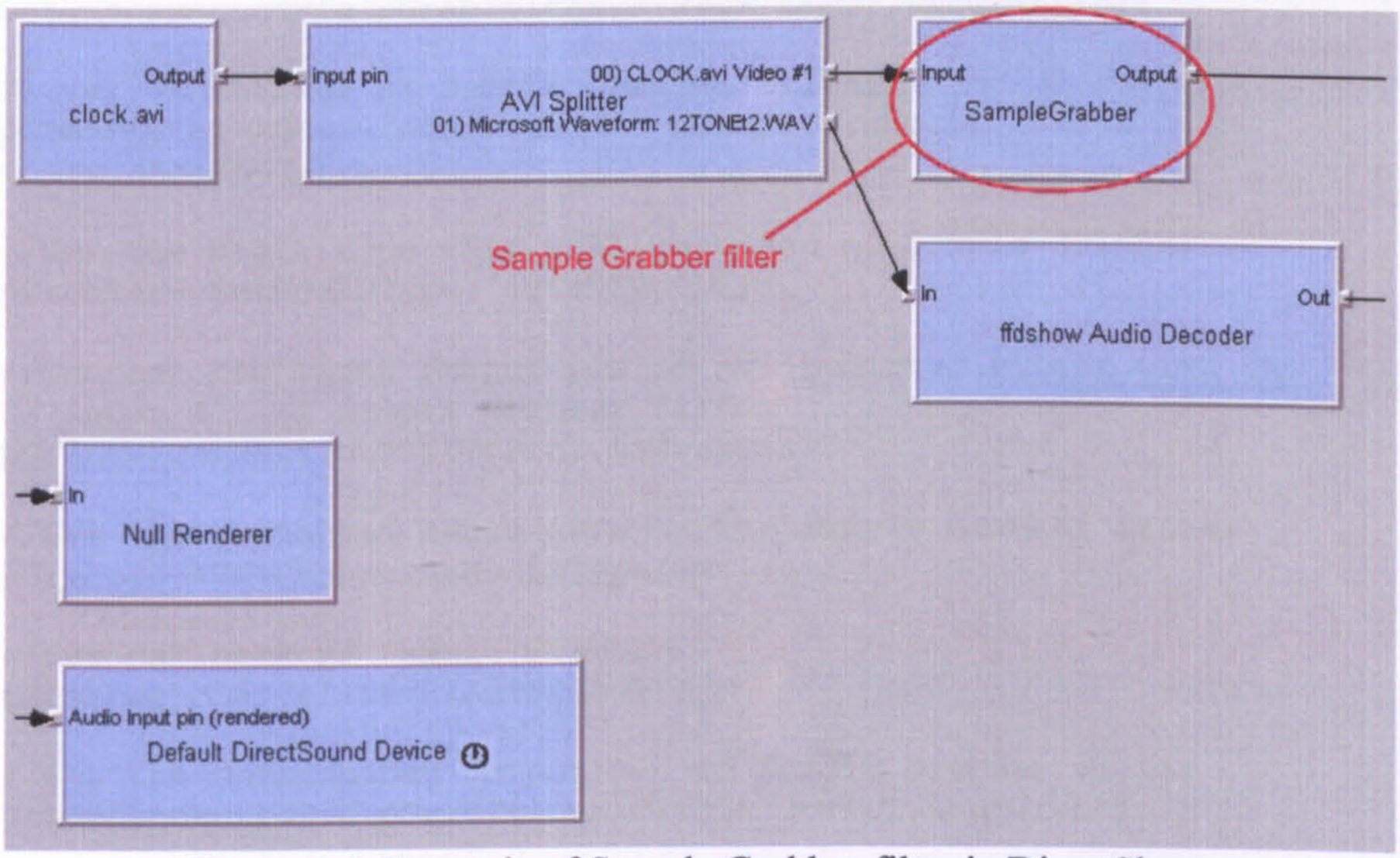

Figure 4.4: Example of Sample Grabber filter in DirectShow.

Figure 4.4 shows that a "SampleGrabber" filter is added after the "AVI Splitter". Consequently, the image of the video stream can be grabbed for image processing and vision-based AR registration. A snippet of coding of Sample Grabber is shown as below:

```
//Create the Filter Graph, insert the Source Filter, Avi Splitter
// Filter, and connect them

//Create the Sample Grabber
CoCreateInstance(CLSID_SampleGrabber,NULL,CLSCTX_INPROC,
  IID_IBaseFilter,(void**)&pGrabFilter);

//Add the Sample Grabber to the Filter Graph
pGraph->AddFilter(pGrabFilter,L"SampleGrabber");

//Get the unconnected input pin of Sample Grabber Filter
GetUnconnectedPin(pGrabFilter,PINDIR_INPUT,&inGrabPin);

//Query interface of Sample Grabber filter
pGrabFilter->QueryInterface(IID_ISampleGrabber,
  (void**)&pGrabber);

//Set the media type that will received by Sample Grabber
pGrabber->SetMediaType( &grabType );

//Connect the video output pin of Avi Splitter Filter with the
// input pin of Sample Grabber filter
pGraph->Connect(outSplitPin1,inGrabPin);

//Get the connected media type of the Sample Grabber filter
pGrabber->GetConnectedMediaType(&mt);

//Set callback of Sample Grabber
pGrabber->SetCallback(&grabCB,1);

//Get the unconnected output pin of Sample Grabber filter
GetUnconnectedPin(pGrabFilter,PINDIR_OUTPUT,&outGrabPin);

//Create Null Renderer
CoCreateInstance(CLSID_NullRenderer,NULL,CLSCTX_INPROC,
  IID_IBaseFilter,(void**)&pNullFilter);

//Add Null Renderer to the Filter Graph
pGraph->AddFilter(pNullFilter,L"Null Renderer");

//Render the output pin of the Sample Grabber
pGraph->Render(outGrabPin);

//Render the audio stream from Avi Splitter filter
pGraph->Render(outSplitPin1);
```

Figure 4.5: Snippet of creating Sample Grabber in DirectShow.

The coding above shows the example of using Sample Grabber to connect with the source filter. Consequently, the image data of the video stream can be grabbed for further processing.

4.2 Image Processing

An image library was built in this research so that greyscale transformation and thresholding can be performed. Besides that, reading and writing BMP file functions were also included in the image library.

An image file contains 2-dimensional data. Each data of the image is a pixel which consists of red, green, and blue. Image processing is applied to the image data in real-time detection.

Greyscale transformation function was written in the image library. The coding of the greyscale transformation is shown as follows:

```
void PictureGreyscale(unsigned char** pData,int width,int height)
{
   float temp;
   int i=0;
   while(i<width*height) {
      temp=0.299f * *(*pData + i*3) + 0.587f *
         *(*pData + i*3 +1) +
         0.114f * *(*pData + i*3 +2); //With the sequence RGB
      *(*pData+i*3) = *(*pData+i*3+1) = *(*pData+i*3+2) =
         (int)temp;
      i++;
   }
}
```

Figure 4.6: Snippet of greyscale transformation function.

The coding above shows that the first parameter is a pointer to the image data, followed by width and height. The formula to convert the colour image into greyscale is performed to the image data.

Next, thresholding function was also written in the library. The algorithm is shown as follows:

```
void PictureThresholding(unsigned char** pData,int width,int
  height,int threshold) {
  int i;
  for(i=0;i<width*height;i++) {
     *(*pData + i*3) = *(*pData+i*3) > threshold ? 255 : 0;
     *(*pData +i*3 +1) = *(*pData+i*3+1) > threshold ? 255 : 0;
     *(*pData +i*3 +2) = *(*pData+i*3+2) > threshold ? 255 : 0;
  }
}
```

Figure 4.7: Snippet of thresholding function.

The coding above shows the function thresholding. The parameters of the function are pointer to the image data, width, height, and a threshold value. Based on the threshold value, the image is converted into black and white.

These functions are applied towards the image grabbed from the video stream. The processed images are then used for AR registration.

4.3 AR Registration

AR requires real-time detection and complex calculation for the registration of the real environment and 3D world. The ARToolKit was used in this research to perform the registration. libAR.lib and libARgsub.lib from ARToolKit were used in this research.

arDetectMarker() is a function from libAR.lib. This function was used for the

marker detection. A modification was implemented so that the function is compatible to the system. The modification is shown as follows:

```
//Original
int arDetectMarker( ARUint8 *dataPtr, int thresh,
                    ARMarkerInfo **marker_info,
                    int *marker_num );

//Modified
int arDetectMarker( ARUint8 *dataPtr,int w,int h, int thresh,
                    ARMarkerInfo **marker_info,
                    int *marker_num );
```

Figure 4.8: The modification of arDetectMarker() for the compatibility of the system.

Two parameters were added to arDetectMarker(). They were width and height of the image. The parameters were needed in this research because this research did not use libARvideo.lib for loading captured video. Instead, DirectShow library was used for the video capturing. As a result, modification was done to accomplish the task of marker detection.

After the marker is detected, a transformation matrix is calculated. The algorithm of marker detection is shown as follows:

```
ARMarkerInfo *marker_info;
int marker_num;
arDetectMarker((unsigned char*)img.pData,img.width,img.height,
  100,&marker_info,&marker_num);

int i,j,k;
for(i=0;i<g_objnum;i++) {
  k=-1;
  for(j=0;j<marker_num;j++) {
    if(g_obj[i].id==marker_info[j].id) {
      if(k == -1) k=j;
      else if(marker_info[k].cf < marker_info[j].cf) k = j;
    }
  }
  if(k==-1) {
    g_obj[i].visible=0;
    continue;
  }
  if(g_obj[i].visible==0){
    arGetTransMat(&marker_info[k],g_obj[i].marker_center,
      g_obj[i].marker_width,
      g_obj[i].trans);
    }
  else {
    arGetTransMatCont(&marker_info[k],g_obj[i].trans,
      g_obj[i].marker_center,
      g_obj[i].marker_width,g_obj[i].trans);
  }
  g_obj[i].visible=1;
}
```

Figure 4.9: Snippet of marker detection.

The algorithm above shows that when a marker is detected, arGetTransMat() and arGetTransMatCont() are called. These functions are used to calculate the camera position and orientation in matrix. The ARToolKit matrix format is used. Thus, the matrix needs to be converted to OpenGL matrix format. The coding below shows the conversion of the matrix format:

```
double gl_para[16];
for(i=0;i<g_objnum;i++) {
  if(g_obj[i].visible==0) continue;
  argConvGlpara(g_obj[i].trans,gl_para);

  //Draw object here
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  argDraw3dCamera( 0, 0 );

  glMatrixMode(GL_MODELVIEW);
  glLoadMatrixd(gl_para);

  if(g_obj[i].id==0) {
    //Draw the object using OpenGL
    ...
  }
}
```

Figure 4.10: Snippet of converting AR matrix to OpenGL matrix.

The coding above shows that argConvGlpara() is called. This function is used to convert the ARToolKit matrix format to OpenGL matrix format. argConvGlpara() is one of the functions of libARgsub.lib. After the conversion, glLoadMatrxd() function is called to load the transformation matrix in OpenGL. As a result, the 3D computer graphics can be drawn in a correct view for the users as an AR object.

4.4 Virtual Objects Generation with OpenGL

Besides the image library, a computer graphic library was also built in this research. OpenGL is a 3D graphic API. It was used to represent 2D and 3D objects in this research. GLUT is used to perform system level operation, such as creating the window which has a compatible device context for OpenGL. Besides that, it also allows the integration of the keyboard and mouse input in the OpenGL. Primitives 3D objects such as cube, sphere, and teapot can be drawn using the functions of GLUT.

OpenGL was used to draw 3D objects in this research. A 3D object was described in 3D vertices. The coordinate of every vertex needed to be known. In OpenGL, glVertex*() function can describe the coordinate of the vertices. However, these vertices did not represent any colour. The material colour of the object needed to be described. There were two ways to see the colour of the 3D objects. It depended on the existence of light. If the lighting was enabled, glMaterial*() should be used to describe the material colour. On the other hand, if the lighting was disabled, glColor*() should be used. Both methods produced the result differently. When lighting was enabled and glMaterial*() was used, the 3D objects will be drawn with shadow. On the other hand, if the lighting was disabled, the objects will be rendered with glColor*(), and the shadow will not be drawn. The lighting was preferred in this research project because the 3D objects will be appeared with the shading. This makes the 3D objects have better appearance and provide better visual perception for the users. With the help of shading, it will

provide depth perception for the perceiver (Hubona, Shirah, & Jennings, 2004).

4.4.1 Shading

There are two types of shading, flat shading (or surface shading) and vertex shading. OpenGL does not provide calculation for the shading. The calculation of the shading needs to be implemented manually. Flat shading is easier than vertex shading. Flat shading required fewer statements in the algorithm than vertex shading. Besides that, vertex shading produced better view, because vertex shading enhances the depth perception and the smoothness of the 3D object.

The normal, or the perpendicular vector of the surface, was needed for the flat shading calculation. The coordinates of three vertices are required in order to calculate the normal. For instance, the vectors were v1, v2, and v3. To apply the normal to the OpenGL, glNormal*() function is used. The coding below shows a snippet of glNormal*().

```
glBegin(GL_POLYGON);
glNormal(n1,n2,n3);
glVertex(v1,v2,v3);
glVertex(v4,v5,v6);
glVertex(v7,v8,v9);
glEnd();
```

Figure 4.11: Statements with glNormal*() in OpenGL for the lighting.

The function glNormal*() is used within glBegin() and glEnd(). Any vertex after the glNormal*() is affected by the normal value. The snippet above shows an

example of flat shading. Three vertices which create a polygon use same normal vector. The lighting will be reflected perpendicularly to the face of the polygon. Flat shading algorithm is shown as follows:

```
i = 0;
while(i < numberOfVertex) {
  glBegin(GL_POLYGON);
  glNormal3fv(normal);
  glVertex3fv(vertex1);
  glVertex3fv(vertex2);
  glVertex3fv(vertex3);
  glEnd();
  i++;
}
```

Figure 4.12: Flat shading algorithm in OpenGL.

When a list of vertices that describe a 3D object, the function is able to draw the object with the flat shading.

Vertex shading is more complex than flat shading. Vertex shading is the normal lighting that reflects perpendicularly from the vertex. The normal vector of each vertex needed to be calculated. All the normal vectors of the faces that connected to the vertex needed to be acquired in order to calculate the normal vector of the vertex. By summing up all the normal vectors, and normalise it, the result will be the normal vector of the vertex. If a list of vertices which describes a 3D object was given, the algorithm is shown as follows:

```
nFace = Calculate the number of face
For each face, calculate the faceNormal
Initialise all vertexNormal to (0,0,0)
Initialise currFace=0
For each vertex
  if vertex is in the current face
    vertexNormal[i] = vertexNormal[i] + faceNormal[currFace]
  else
    increase currFace by 1
For each vertexNormal
  Normalise vertexNormal
```

Figure 4.13: Vertex shading algorithm for OpenGL.

When each vertex normal is calculated, the list of vertex normals is used in OpenGL by using glNormal*() as the algorithm of flat shading. The computer generated object will be shown with the shading, which was smoother than flat shading. The vertex shading enhances the smoothness of the virtual object. The snippet using the glNormal*() in vertex shading is shown as below:

```
glBegin(GL_POLYGON);
glNormal3f(n1,n2,n3);
glVertex3f(v1,v2,v3);
glNormal3f(n4,n5,n6);
glVertex3f(v4,v5,v6);
glNormal3f(n7,n8,n9);
glVertex3f(v7,v8,v9);
glEnd();
```

Figure 4.14: OpenGL statements of vertex shading.

The coding above shows that every vertex is defined after different normal, the vertices do not share the same normal. Every vertex is displayed with different shading. The lighting is reflected with different angles according to each normal.

4.4.2 Texturing

In 3D computer graphics, texture mapping is an essential component for texturing. Texture mapping in OpenGL only supports the width and height, which are power of two (2, 4, 8, 16, ...) (Miller, 2000). The texture is 2D and it will wrap the 3D object according to the coordinate described by glTexCoord*().

To implement texture mapping, an image is used. Width, height, and the image data in RGB (red, green, and blue) were acquired. This data was stored in memory. glBindTexture() function was called in order to bind the texture to the 3D object. The implementation of texture mapping in OpenGL is shown as below:

```
GLuint textureId = glGenTextures()
glBindTexture() to bind the textureId to a target object
glTexImage2D() to set the texture image properties
glTexParameteri() to set the texture parameter
glEnable(GL_TEXTURE_2D)
For each vertex
  glTexCoord2f(tx,ty) where tx and ty is the texture coordinate
  glVertex3f(vx,vy,vz) where vx, vy, and vz are the coordinate
    of the vertex
glDisable(GL_TEXTURE_2D)
glDeleteTextures() to delete the texture using textureId
```

Figure 4.15: Pseudo code of texture mapping in OpenGL.

A 2D image is used as a texture in order to map to the 3D object. The texture mapping enhances the visual perception towards the 3D objects because the surface was drawn with roughness instead of entire smoothness. Therefore, the prerequisite to implement the 2D texture mapping was to load a 2D image data into the memory, then the OpenGL can use the 2D image data to map onto the 3D object.

In this research project, the video stream of the real environment was displayed through the GLUT window and OpenGL as the output. As a result, the 3D objects can be integrated with the video stream as the augmented objects.

4.4.3 Animations

Animation is another important component in 3D computer graphics. Animation is a series of images which makes the viewers feel that the 3D objects are moving. In 3D computer graphics, the key frame of the animation is required and the key frame data is stored in a file. In this research project, there were two types of key frame data. They were transformation data and vertex data.

The transformation data describes the position of the 3D object. The animation makes the 3D object change its position without changing the shape. Generally, there are three types of transformation: translation, rotation, and scale transform. These types of transformation do not change the shape of the 3D object in an animation, but the position of the object relatively to the world.

All the transformation type is included in the coding above. Translation, rotation, and scale transform exist together in a frame. On the other hand, the vertex data is different from transformation data. A vertex does not have any shape. Rotation and scale transform do not affect a vertex, only the 3D coordinate affects the vertex. Therefore, the vertex data describes the 3D coordinate of every vertex of the object. In animation, different 3D coordinates of every vertex are stored for

```
currFrame = 0; //Initialise
while(currFrame < maxFrame) {
  glTranslatef(tx,ty,tz); //where tx, ty, tz are the translation
    //parameters of currFrame
  glRotatef(deg,rx,ry,rz); //where deg is the angle, rx, ry, rz
    //are the rotation parameters of currFrame
  glScalef(sx,sy,sz); //where sx, sy, sz are the scale parameters
    //of currFrame

  //Draw the object in currFrame
  ...

  currFrame++;
}
```

Figure 4.16: Snippet of transformation animation in OpenGL.

every key frame. Every key frame displays the vertices in different coordinates. As a result, the shape of the 3D object was changed gradually. The following is the algorithm to perform the animation of vertices.

```
currFrame = 0; //Initialise
while(currFrame < maxFrame) {
  //Draw the object in currFrame
  glBegin(GL_POLYGON);
  glVertex3f(x1,y1,z1);
  glVertex3f(x2,y2,z2);
  glVertex3f(x3,y3,z3);
    //x*,y*,z* are the coordinates of the vertices in
    // current frame
  glEnd();

  currFrame++;
}
```

Figure 4.17: Snippet of vertex animation in OpenGL.

The algorithm above shows that the vertices are defined from frame to frame. The key frame data stores the different coordinates of the vertices which described the shape of the object. Consequently, the coordinates of the vertices are changed from frame to frame and animation occurs.

However, interpolation is required in order to make the animation work more smoothly. Interpolation is to estimate an intermediate value of two variables when one of the variable corresponds to several discrete values of the other variable. In this research project, interpolation was used to calculate the frames between two given key frames of an animation. Interpolation is a calculation of the coordinate of the vertices between two key frames, or the transformation of the 3D object between two key frames. The calculation of interpolation for coordinate of the vertices and transformation of the 3D object were same. The calculation of the interpolation is shown as below:

$$output_i = inputFrame1_i + (inputFrame2_i - inputFrame1_i) \times \frac{currUnit}{maxUnit}$$

Figure 4.18: Formula of interpolation for animation.

When a list of vertices or transformation value is given, the above formula can be used. *maxUnit* is a constant which can be decided by the programmer. The *currUnit* will be increased from frame to frame in order to make the 3D object animated smoothly. The snippet of coding below shows how to use the interpolation to perform animation:

```
void CalcInterpolate(float *v1,float *v2,int size,
                     int currUnit,int maxUnit,
                     float *vOut) {
  int i;
  for(i=0;i<size;i++) {
    vOut[i]=v1[i] + ((v2[i]-v1[i])*currUnit/maxUnit);
  }
}

...
currUnit = 0;
while(currUnit < maxUnit) {
  //Calculate interpolation
  float *vInter; //A list of vertices for interpolation
  CalcInterpolate(&v[frame1],&v[frame2],size,currUnit,maxUnit,
    vInter);

  //The result of interpolation vertices is stored in vInter
  ...

  //Draw the vertices using vInter.
  ...

  currUnit++;
}
...
```

Figure 4.19: Snippet of interpolation for animation.

The coding above shows that CalcInterpolate() function uses the formula to calculate the interpolation coordinates of the vertices. Consequently, the calculated vertices are drawn. Then, the increment of the *currUnit* is implemented. The algorithm is performed until *currUnit* was equal to *maxUnit*, which was defined by the programmer. This is the interpolation between two key frames. The animation was also performed in this research to produce animated 3D virtual objects for the users.

4.5 Finger Tracking

Finger tracking was implemented in this research for the interaction with the

virtual objects. The implementation of the finger tracking began with the skin colour segmentation. The formula for the skin colour according to Kovač, Peer, and Solina (2003) was used. The coding of skin segmentation is shown as below:

```
for(int i=0;i<width * height * 3;i+=3) {
  if(imageData[i] > 95 && //red
     imageData[i+1] > 40 && //green
     imageData[i+2] > 20 && //blue
     max(imageData[i],imageData[i+1],imageData[i+2]) -
       min(imageData[i],imageData[i+1],imageData[i+2]) > 15 &&
     abs(imageData[i] - imageData[i+1]) > 15 &&
     imageData[i] > imageData[i+1] &&
     imageData[i] > imageData[i+2] )

     skin[i] = 1;
  else
     skin[i] = 0;
}
```

Figure 4.20: Snippet of colour segmentation based on Kovač, Peer, and Solina (2003).

The snippet above shows that every pixel of the image is traversed. If the colour components of the current pixel fulfilled the skin colour conditions, then an array is assigned with true. Thus, the array stores the skin region of the image. After the colour segmentation towards the skin of the fingers, the position of the finger region is known.

A template of a fingertip was used in this research. The following image is the template of the fingertip in this research:

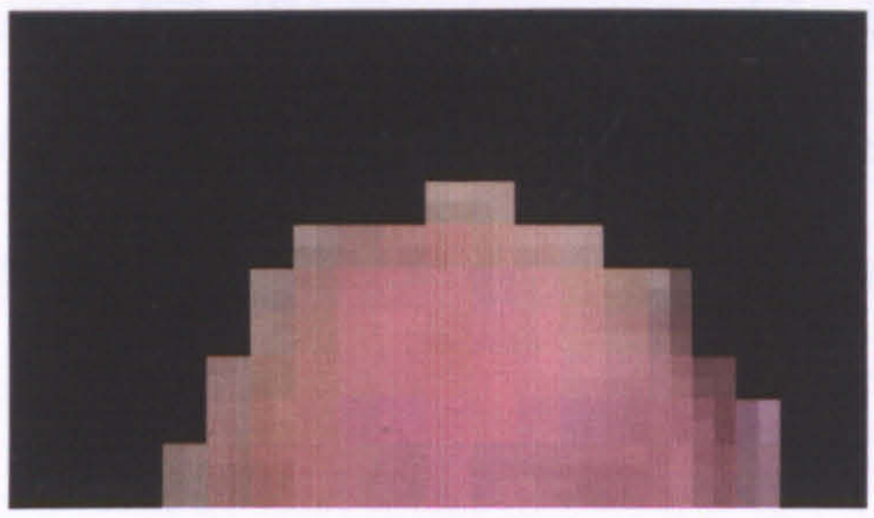

Figure 4.21: Template of fingertip used in finger tracking

Figure 4.21 shows the fingertip after the skin colour segmentation. The image was saved as the template for template matching.

OpenCV contains function for template matching. The function of template matching in OpenCV is cvMatchTemplate(). This function provides different matching methods including sum squared of differences (SSD) and normalised cross-correlation (NCC). The template matching with NCC method was used to match the fingertip of the user. After the template matching, the fingertip of the users was tracked. The two-dimensional information of fingertip was used for the interaction. As a result, the interaction was able to be performed with the finger tracking.

4.6 Summary

Video capturing was used in this research to capture the video of the real environment. DirectShow was used for video capturing. After the video stream was captured, the greyscale transformation and thresholding were applied to the captured video. Thus, an image library was built. The processed video stream was

used for AR registration. ARToolKit libraries were used for the registration. Then, the virtual objects were drawn by using OpenGL with GLUT. Finally, OpenCV was used for finger tracking. The 2D coordinate of the tracked fingertip was used for the interaction with the virtual objects.

CHAPTER 5
FINDINGS AND DISCUSSIONS

5.0 Overview

This chapter discusses the findings of the research after the development of the AR Application Builder. Video capturing which uses DirectShow is discussed. Image processing functions of the image library is also discussed. The generation of the virtual object using OpenGL is discussed. Furthermore, the AR registration which uses ARToolKit libraries is discussed. Some of the applications built in this research is shown. Interaction by using mouse clicks and finger tracking is also discussed. Finally, the outcome of the evaluation on the finger tracking for the interaction with the virtual object in an AR system is discussed.

5.1 Video Capturing

DirectShow was used in this research. A class was written to wrap the DirectShow functions. The main functions of the class was to load a capture device such as web camera. A screenshot of using the DirectShow to capture image of the real environment is shown as below:

Figure 5.1: Screenshot of video capturing using DirectShow.

Figure 5.1 is a screenshot of an image captured from capture device from the real environment. The DirectShow library was able to load the capture device and capture the image from the real environment in real-time. However, when there were more than one capture device, the library will enumerate the first device according to the setting of the computer system. Therefore, the other capture devices might need to be disabled so that the target capture device can be enumerated.

In addition, the library also had the ability to load video files. The image below shows an example of loading a video file and displaying the video through OpenGL:

Figure 5.2: Screenshot of playing a video file using DirectShow and display in OpenGL as texture mapping.

Figure 5.2 shows that the image data was grabbed from the video stream by using DirectShow. Then the image data from the buffer was used as texture and mapped to the OpenGL.

Besides that, the video file was able to be loaded in the AR rendering using the library. As a result, the virtual object on the fiduciary marker was a video played in real-time.

5.2 Image Processing

The image library which was built in this research was also tested. Several functions were written in the image library. Greyscale transform based on Parekh (2006) was one of the basic function that converts the colour image into greyscale image. The following image is an example of implementation of greyscale transform:

Figure 5.3: Greyscale transformation of the coloured image.

Figure 5.3 shows the difference of the colour image and greyscale image. There colour components were reduced to a greyscale value for each pixel, lesser memory storage was required to store the greyscale image.

Besides that, the thresholding function was also written in the library. The following image shows the result of thresholding:

Figure 5.4: Result image after thresholding is applied.

Figure 5.4 shows a result image of thresholding using threshold value 100. Thresholding was a basic method in image processing and computer vision.

5.3 Virtual Object Generation

OpenGL was used for building computer graphic library in this research. The library was tested for the functionality of shading. The following is a screenshot of flat shading:

Figure 5.5: Flat shading of a 3D object in OpenGL.

Figure 5.5 shows the result flat shading. The lighting reflects perpendicularly from the surface so that the surface of the 3D object appeared roughly. On the other hand, the vertex shading produced the following output:

Figure 5.6: Vertex shading of a 3D object in OpenGL.

Figure 5.6 shows that vertex shading makes the surface of the 3D object appear smooth because the lighting reflected perpendicularly from every vertex. Though

vertex shading was preferred in this research project, both algorithm were written in the library.

The texturing functionality of the library was also tested. The following screenshot shows an example of a 3D model with the texture mapping:

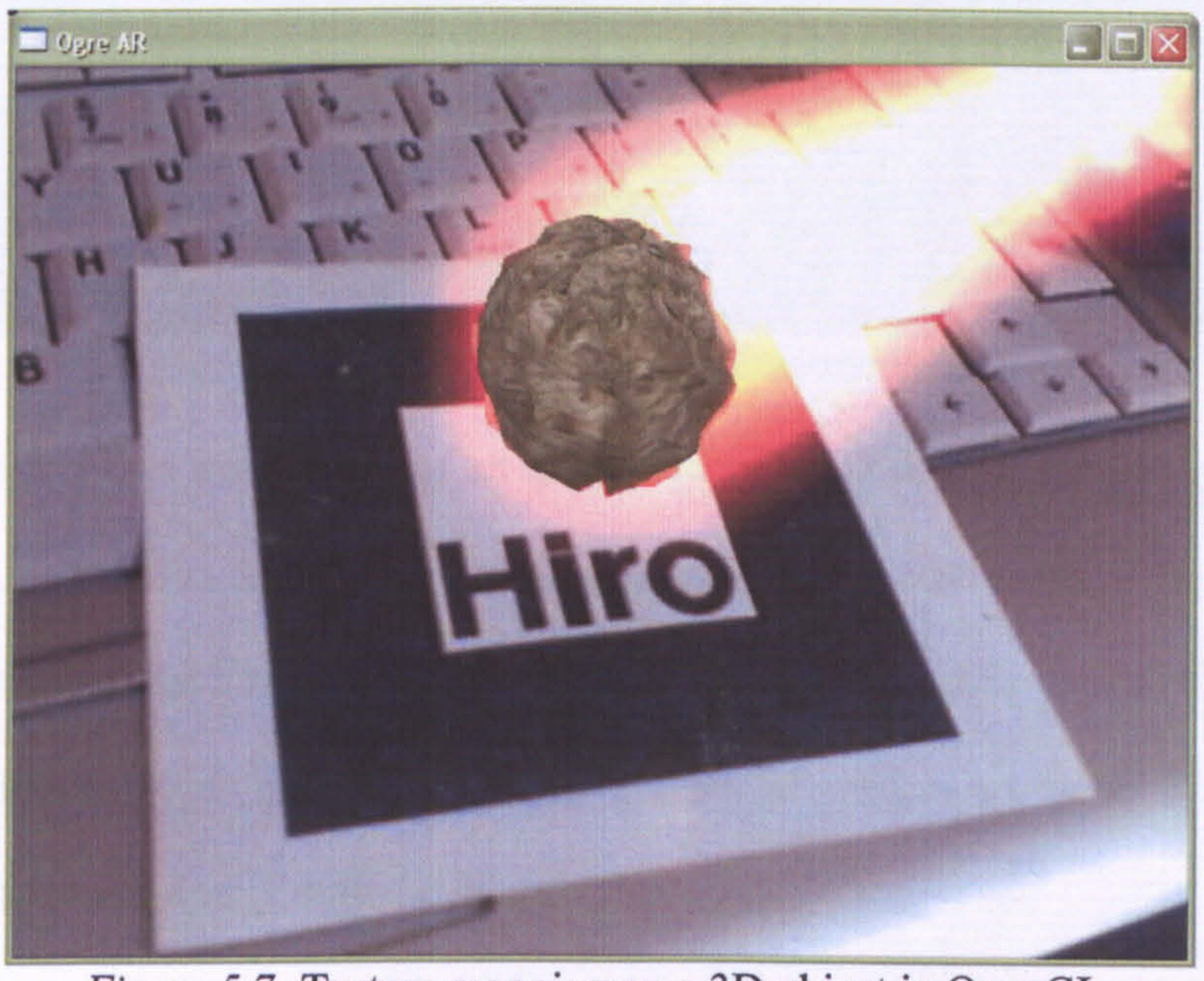

Figure 5.7: Texture mapping on a 3D object in OpenGL.

The figure above shows an example of texture mapping. The surface of the meteor is wrapped by a texture to produce a rough surface. The texture mapping was implemented in the AR system.

Moreover, the algorithm of the animations was also implemented in the library. The following image shows an example of animation:

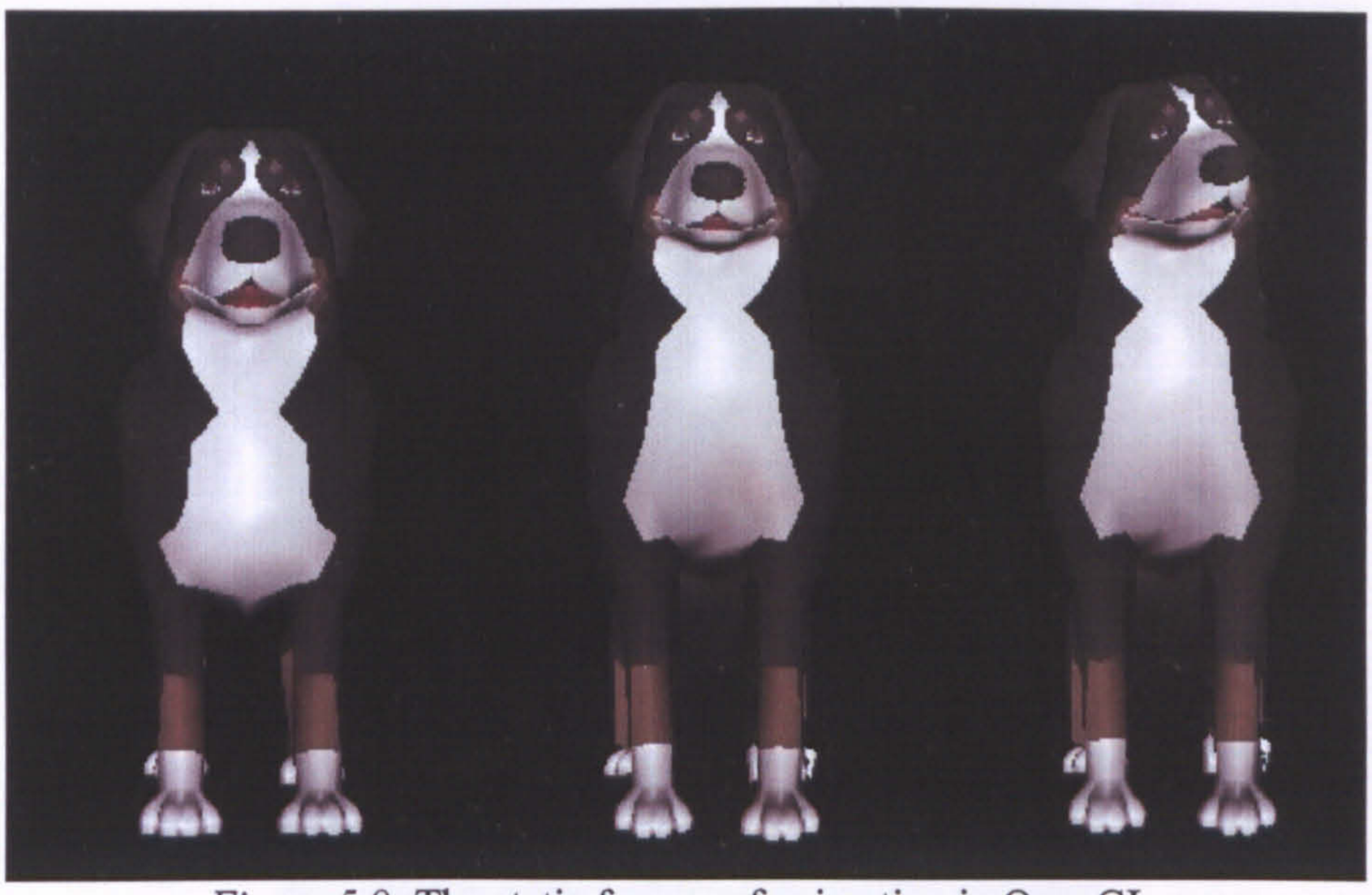

Figure 5.8: The static frames of animation in OpenGL.

Figure 5.8 shows a sequence of an animation of a 3D dog. The numbers of the vertices are same in the three key frames. However, the coordinates of the vertices are different in the three key frames. The shape of the dog was modified without destroying the whole object because the number of vertices are same. The object was animated smoothly with the interpolation.

5.4 AR Registration

The registration of the AR object in the real environment was implemented by using ARToolKit libraries. Several applications were built by using the AR Application Builder with the ARToolKit libraries for the registration. The following screenshot is one of the applications in this research:

Figure 5.9: Screenshots of playing video in AR system.

The screenshots above shows a sequence of a video in an AR application. The DirectShow library was able to be applied in AR system not only to enumerate the capture device, but also to play the video file in the AR environment.

Furthermore, an AR game with the virtual aeroplane was also built by AR Application Builder. The screenshots are shown as below:

Figure 5.10: Screenshots of 3D aeroplane as an AR game.

Figure 5.10 shows a sequence of screenshots from an AR game of this research. A 3D model of an aeroplane was loaded by the applciation. During the AR rendering, the 3D aeroplane flew around the fiduciary marker. Besides that, the texture mapping was able to be applied to the aeroplane.

The AR application of tourism was also built in this research. The following figure shows an example of 3D object in the tourism application:

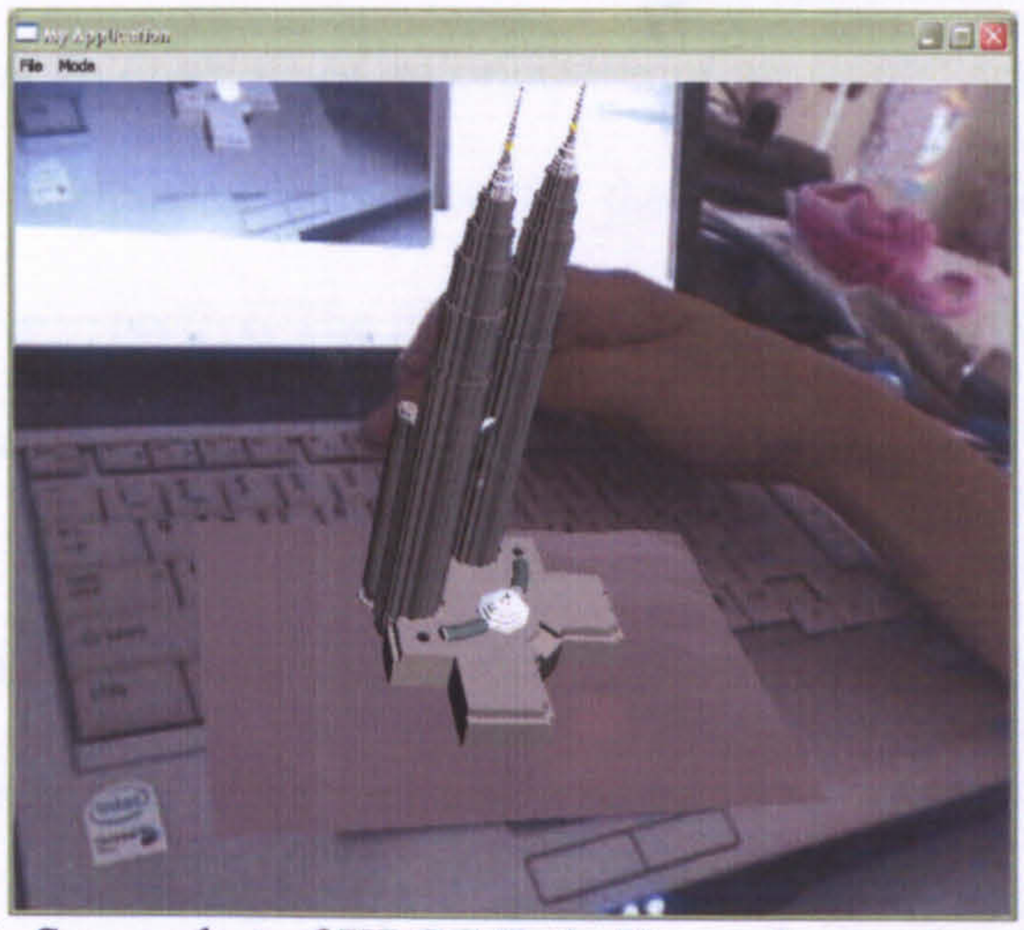

Figure 5.11: Screenshot of KLCC Twin Tower for tourism application.

The screenshot above shows a KLCC Twin Tower of Malaysia. It is a model for the tourism application in this research. Besides the tourism application,

educational applications were also built. The following figure is a screenshot of an example of an educational application:

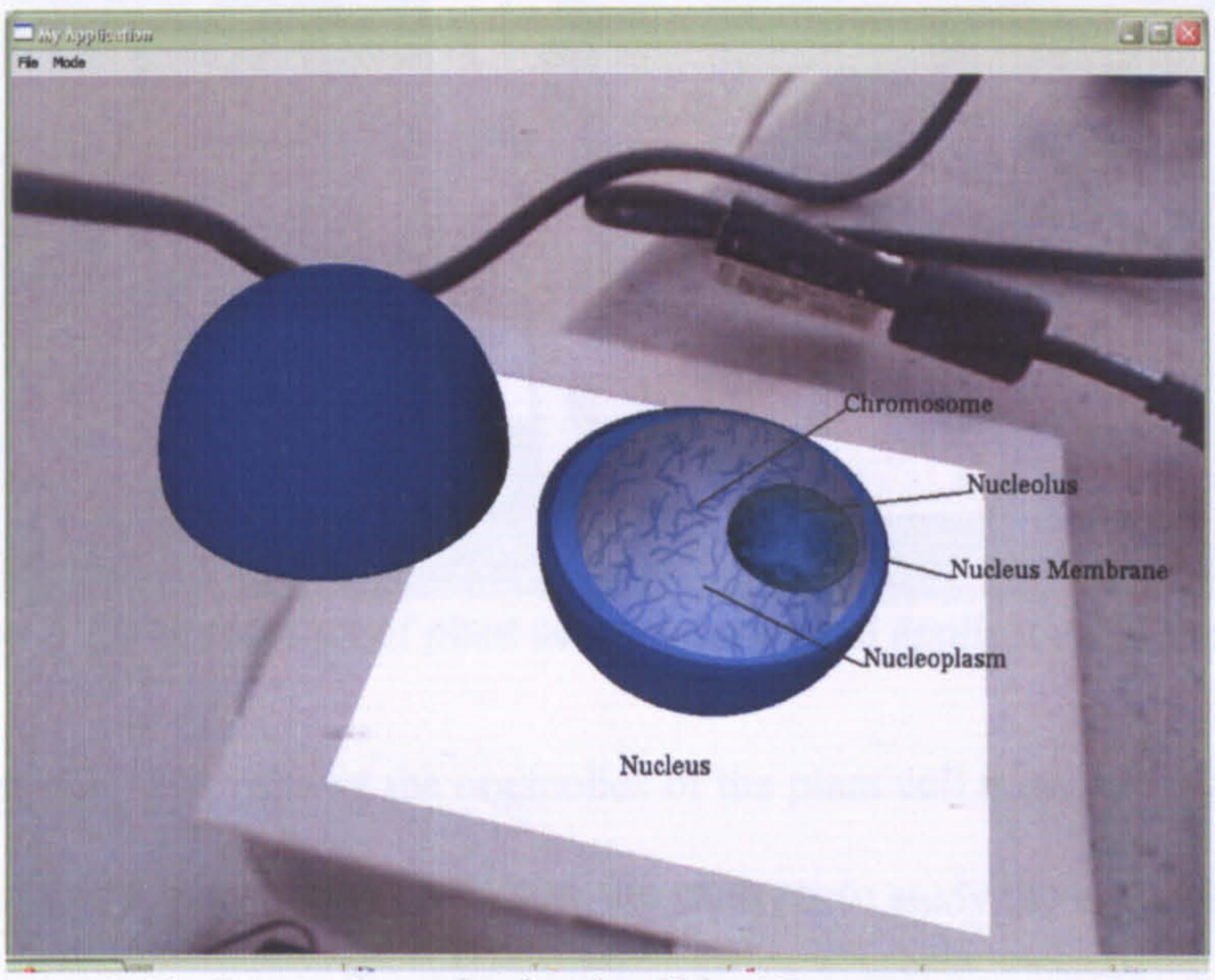

Figure 5.12: Screenshot of animal cell in AR system as an education application for biology.

Figure 5.12 shows an example of an organelle of an animal cell, nucleus, in the AR system. This application was used for the students to study the biological objects in the 3D perspective. The purpose of this application was to enhance the students' understanding towards the biological objects.

Besides the animal cell, there was another educational application related to the plant cell. The screenshot of the application is shown as below:

Figure 5.13: Screenshot of plant cell in educational application in biology.

The screenshot above shows the organelles of the plant cell in an AR system. The purpose of this application was to allow the students to study the organelles of the plant cell and how they looks like.

AR applications in education provide better learning environment to the users. The three-dimensional virtual objects produce more information to the students. The students are able to understand the learning objects better than traditional learning instruments such as text books. The students can perceive the computer generated virtual objects as the real objects because they are seamlessly integrated in the real environment. The learning with the AR application allows the students to construct new knowledge based on what they perceived in AR system. The students can use the constructed knowledge in future. For example, the learning objects of biology allows the students to understand the structure of the animal

cells and the plant cells. The students are able to compare the organelles of two types of cells. The students can learn more through the insights of the AR learning method.

5.5 Interaction

The interaction of the virtual object was also tested in this research. The screenshots below show an example of interaction with the virtual objects by using mouse:

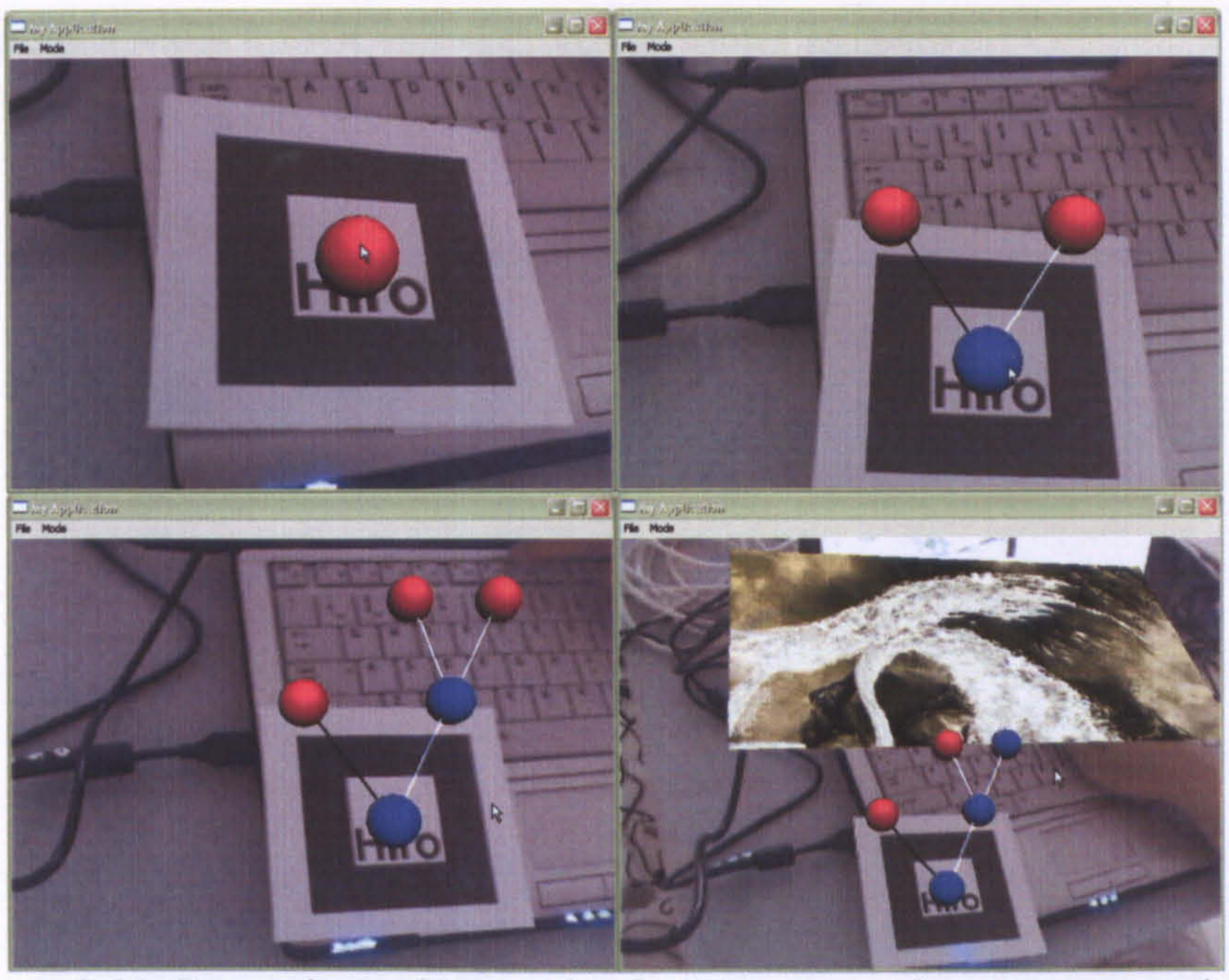

Figure 5.14: Screenshots of interaction in AR system using mouse as an input device.

Figure 5.14 shows a sequence of interactions with the virtual object in an AR system. The mouse was used to click the red sphere. When the red sphere was

clicked, it turned into blue colour and expanded two more red spheres. Contrarily, if the blue sphere was clicked, it turned into red colour again and the children spheres were collapsed. When one of the spheres of the final level was clicked, a video was displayed.

5.5.1 Finger Tracking

The finger tracking for the interaction was tested in this research. The finger tracking allowed the fingers to manipulate the virtual object in real-time. This produced a nature input towards the AR system. The screenshots for the finger interaction towards the AR virtual object are shown as below:

Figure 5.15: Screenshots of finger tracking as an input device for interaction in AR system.

Figure 5.15 shows that the teapot is able to be resized by the bare finger. The

system firstly detected the skin colour of the fingers in order to determine the region of the finger. After the finger region was determined, the 2D position of the finger was used to compare with the position of the green dots relatively. The green dots were used to determine the size of the teapot. Hence, by using the bare fingers, the dots were moved away. As a result, the size of the teapot was modified.

Using finger to interact with the virtual objects is a natural response of the users. The users perceive the virtual objects as they are the real objects since they are registered correctly in the real environment. The users can feel that the objects are on top of the fiduciary marker. Therefore, the users will touch the virtual objects by using finger. The bare finger tracking is important as it reduces the unnatural behaviours such as using the wired glove to touch the virtual objects in the real environment.

5.6 Evaluation

An application was built for the evaluation purpose. The application was an AR video player with three buttons. The three buttons were play, pause, and stop. The users were able to use their finger to touch the button and trigger the function of the button. The screenshots of the application is shown as below:

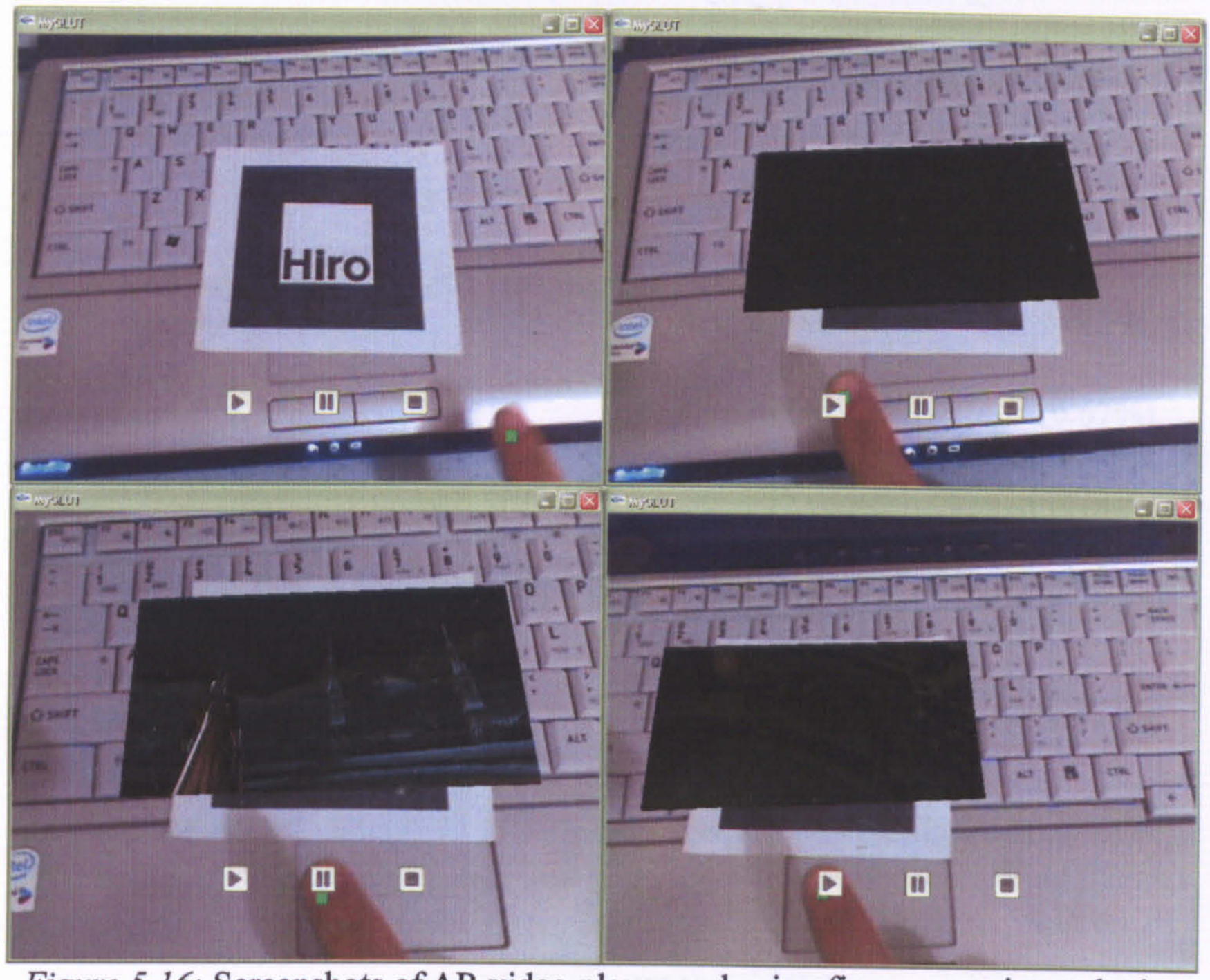

Figure 5.16: Screenshots of AR video player and using finger as an input device to operate the virtual player.

Figure 5.16 shows that when a marker is detected, three buttons are appeared. A green dot is also appeared on the finger. The green dot is used as a cursor for the user. Thus, the users are able to know where is the cursor and try to move their finger as the cursor to the target position of the screen.

An evaluation was conducted to test the system functionality of the finger tracking for interaction with the virtual object in AR video player. The result was collected. The following table shows the result of the evaluation:

Table 5.1
The result of the evaluation on the system functionality of the finger tracking for interaction with the virtual object in AR system.

No.	Tasks	Responses	
		Success	Fail
1	Use a web camera to detect the marker to show the buttons	30 (100%)	0 (0%)
2	Use the finger to click play button to start playing the video in AR	28 (93.33%)	2 (6.67%)
3	Click pause button to pause the video	27 (90%)	3 (10%)
4	Click play button to resume the video	28 (93.33%)	2 (6.67%)
5	Click stop button to stop the video	28 (93.33%)	2 (6.67%)
6	Click play button to play the video from the beginning	28 (93.33%)	2 (6.67%)
7	Watch the video completely	30 (100%)	0 (0%)

Table 5.1 shows that 100% of the participants were able to use the web camera to detect the marker. None of the participants failed to do so. 93.33% of the participants were able to use their finger to click the play button to start playing the video in AR. 6.67% of the participants failed to do so. There were 90% of the participants were able to pause the video, and 10% failed to pause the video. 93.33% of the participants were able to resume, stop, and play the video again from the beginning, 6.67% of the participants failed to do so. Finally, all of the participants were able to watch the video completely.

The table indicates that the detection of the marker was successfully recognised.

This is because the detection of the marker and AR registration implemented by using ARToolKit. Furthermore, most of the users were able to click or touch the buttons. However, the rate of successfulness for clicking the "pause" button was less than other tasks. The reason was the position of the "pause" button which allocated between the other buttons. The participants might unintentionally click the other buttons before click the "pause" button. Finally, all the participants were able to watch the video completely. This indicates that the integration of the video in the AR application was a success.

The participants were able to use their finger as an input device for natural interaction to interact with the virtual objects in the AR system. Computer vision algorithms for skin segmentation was able to differentiate the finger from the environment. Furthermore, the template matching which uses normalised cross-correlation was able to track the fingertip of the participants. As a result, the participants were able to use their fingertip as a pointer device in the AR system. The fingertip replaced the functionality of the mouse to interact with the virtual objects in the real environment. The participants could respond to the virtual objects as the stimuli from the world visual perception towards the computer generated objects. As a conclusion, the functionality of the AR Application Builder AR was successfully implemented which allows the users to interact with the virtual objects in the AR system using bare finger as an input device for the interaction.

5.7 Summary

The DirectShow library was able to enumerate the capture device and also play the video files. The image library was able to perform greyscale transformation and thresholding towards the video stream. Then, the processed video stream was used for vision-based registration. The vision-based registration algorithms accomplished by using ARToolKit libraries. The computer graphic library which was built in this research was able to perform shading, texture mapping, and animations. These functions and libraries were used to build the AR Application Builder. Several applications were built by using AR Application Builder in this research. They were video in AR, tourism application, AR game, and AR application in biology subject. Besides that, interaction by using a mouse was also tested. Then, the finger tracking was implemented to interact with the virtual object. Finally, an evaluation was conducted to evaluate the system functionality of the finger tracking for interaction with the virtual object in AR. An application was built for evaluation. The application was AR video player. The result of the evaluation indicates that the integration of the finger tracking for interaction with the virtual objects in AR system was successfully implemented in this research. The participants were able to interact with the virtual objects naturally without using the devices such as mouse or wired gloves.

CHAPTER 6
CONCLUSION

6.0 Overview

This chapter is a conclusion of this research. The contributions of this research are discussed. Recommendations for the future study are also discussed. Finally, the research is concluded with a summary.

6.1 Contributions of the Study

The major contribution of this research is the comprehensive literature reviews on the Augmented Reality (AR) technology. The AR technology involves the vision-based registration to generate the 3D computer graphics. Besides that, image processing methods are also discussed. The 3D computer graphics techniques are

also explained. Furthermore, finger tracking is also applied in this research. The implementation of the algorithms also provide technical contribution towards the future research.

AR Application Builder also contributes in cognitive science in the sense of visual perception. AR requires computer vision to simulate human visual perception to understand the real world (Meer, Stewart, & Tyler, 2000). Based on the 2D fiduciary marker, the 3D virtual world can be constructed with computer vision algorithms. The users can sense the stimuli generated by the AR system and perceive with visual perception. The purpose of AR is to generate realistic virtual objects (Azuma, 1997), this allows the users to perceive the virtual objects as they are real objects. Consequently, the users will interact with the virtual objects. Since the virtual objects are integrated in the real world, natural interaction is important. The evaluation of this research indicates that AR Application Builder which provides finger tracking feature allows the users to interact with the virtual objects without using other device such as wired gloves.

This research developed a library, AR Application Builder. The source code of the library is attached in Appendix 4. The users are able to use the library to create their own AR applications. The most prominent feature of AR Application Builder is the integration of the finger tracking. Furthermore, the finger can be used to interact with the virtual objects in the AR system. This can enhance the AR experience of the users because the users not only perceive the virtual objects in

the real environment, but also interact with the virtual objects with their bare finger. The methods of finger tracking in this research can be used in future study to produce a natural interactive AR environment.

Besides that, this research can contribute in educational field such as biology, chemistry, and mathematics. In biology, DNA molecules, cells, tissues and organs can be modelled. Then the 3D models are used in AR so that the students can interact with them. The system of this research is used to create an application about animal and plant cells as shown in Chapter 5. The organelles of the animal and plant cells are displayed in AR environment for the students to study what the organelles look like. Besides that, the students will understand better than 2D perspective in the book because the organelles are represented in 3D perspective.

In chemistry, atoms, molecules, particles, and the structure of metals are able to be modelled. Besides that, the physical reactions of the chemical object can be shown in animation. For example, the animation can show the reaction of the water molecules when the water is boiled.

In mathematics, the solid objects such as pyramid, tetrahedron, cuboid, sphere and cone in AR allow the students to perceive the object at different angles. Therefore, AR in biology, chemistry and mathematics helps the students to understand the objects in different perspective and augment interest to learn the subject.

Besides the educational field, the system can also be used in developing AR in

games and toys. The virtual toys such as aeroplane, chess, furniture, jigsaw puzzle, battle field models and cars can be modelled as shown in Chapter 5. Sound and animation can also be included with the 3D objects. The virtual toys can be enjoyed by the children because they can interact with the toys.

AR is of the technologies which emerges with the phone technology (Crago, 2008). The smartphone can run the software like a desktop computer, the applications built by AR Application Builder are able to be ported into a phone device in future.

The applications which were developed in this research project had been participated in several conferences and exhibitions. The participated conferences and exhibitions are attached in the Appendix 3.

6.2 Recommendations for Future Study

This research implemented the finger tracking for interaction with 2-dimensional information. There was no depth information of the finger. The depth information of the finger tracking can be integrated in the future research so that the users can interact with the virtual objects with depth information. As a result, the users were able to move the object further from the camera or nearer to the camera.

Furthermore, finger gesture recognition can be implemented in the future study. The finger gesture can be used to interact with the virtual objects in the AR

system. The gesture recognition can produce different commands towards the AR system.

Besides that, voice recognition can also be integrated as another natural way of interaction. Voice recognition can be used to express the commands to the AR system. Simple commands were useful in the AR system as a supplementary interactions.

Usability of the AR system is also important. Usability evaluation can be conducted in the future study to improve the finger tracking for interaction of this research. As a result, the users can use the system with high satisfaction and ease.

6.3 Summary of the Research

This research developed a computer library, AR Application Builder. AR Application Builder allows the users to develop the AR applications. Moreover, finger tracking for interaction with the virtual objects was also implemented in this research. As a result, the user can use their bare finger to interact with the virtual objects in the AR system.

The literature review of this research includes the Augmented Reality (AR) technology, vision-based AR registration, image processing, template matching, virtual objects generation, and interaction with the finger tracking.

The methodology of this research uses a combination of reuse model and exploratory model for system development process to develop the AR Application Builder. The system uses DirectShow to load the video capture device, ARToolKit for vision-based registration, OpenGL and GLUT to draw the virtual objects, OpenCV for finger tracking and interaction. Furthermore, an image library for image processing was written, and a computer graphic library for shading, texture mapping, and animations was also written. AR Application Builder was built by integrating these functions and libraries together.

Furthermore, evaluation was conducted to test the system functionality of finger tracking for virtual objects in AR system. The evaluation result shows that the finger tracking for interaction was successfully implemented. The skin colour segmentation and template matching with normalised cross-correlation method applied for finger tracking. As a result, the users can use their finger to interact with the virtual objects in AR system.

Several applications were built in this research using AR Application Builder. They were AR toys, AR puzzle, AR chess, AR aeroplane, AR mobile pet, and AR film. Besides that, there were also applications involved biological field and tourism. Finally, the AR Application Builder can be improved and implemented with future studies.

REFERENCES

Aaltonen, A., & Lehikoinen, J. (2006). Exploring augmented reality visualizations. In *proceedings of the working conference on advanced visual interfaces*, 453-456. USA: ACM New York.

Acharya, T., & Ray, A.K. (2005). *Image processing: Principles and applications*. New Jersey: John Wiley & Sons, Inc.

Adams, J. (2004). *Programming role playing games with DirectX*. UK: Thomson/Course Technology.

Ahn, I., Lehr, M., & Turner, P. (2005). *Image processing on the GPU*. University of Pennsylvania. Retrieved March 4, 2007, from http://www.cis.upenn.edu/~suvenkat/700/projects/alt.pdf

Ansar, A. & Daniilidis, K. (2001). Linear augmented reality registration. In W. Skarbek (Ed.), *Computer Analysis of Images and Patterns: 9th International Conference, CAIP 2001, Warsaw, Poland, September 5-7, 2001: Proceedings*. New York: Springer.

Avery, B., Thomas, B.H., Velikovsky, J., & Piekarski, W. (2005). Outdoor augmented reality gaming on five dollars a day. In *proceedings of the Sixth Australasian Conference on User Interface, 40*, 79-88. Australia: Australian Computer Society, Inc.

Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Jullier, S., & MacIntyre, B. (2001). Recent advances in augmented reality. *Computer Graphics and Applications, IEEE, 21*(6), 34-47.

Azuma, R.T. (1997). A survey of augmented reality. *Teleoperators and virtual environments, 6*(4), 355-385.

Barakonyi, I., & Schmalstieg, D. (2006). Ubiquitous animated agents for augmented reality. *IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR 2006)*, 145-154.

Barakonyi, I., Fahmy, T., & Schmalstieg, D. (2004). Remote collaboration using Augmented Reality Videoconferencing. In *proceedings of Graphics Interface*

2004, 89-96. Canada.

Barfield, W. & Caudell, T. (2001). Basic concepts in wearable computers and augmented reality. In W. Barfield & T. Caudell (Eds.), *Fundamentals of wearable computers and augmented reality* (pp. 3-26). Mahwah, New Jersey: Lawrence Erlbaum Associates.

Bernard, M. (2003). How to load a bitmap. In *GameDev.net*. Retrieved April 2, 2007, from http://www.gamedev.net/reference/articles/article1966.asp

Billinghurst, B. (2002). Augmented reality in education. *New Horizons for Learning*. Retrieved March 1, 2007, from http://www.newhorizons.org/strategies/technology/billinghurst.htm

Billinghurst, M., & Kato, H. (2002). Collaborative augmented reality. *Communications of the ACM*, 45(7), 64-70.

Billinghurst, M., Cheok, A., Kato, H., & Prince, S. (2002). Real world teleconferencing. *IEEE Computer Graphics and Applications, 22*(6), 11-22.

Billinghurst, M., Kato, H., & Poupyrev, I. (2001). The MagicBook: Moving seamlessly between reality and virtuality. *Computer and Graphics*, 745-753.

Bimber, O., Encarnação, L.M., & Schmalstieg, D. (2003). The virtual showcase as a new platform for augmented reality digital storytelling. In *proceedings of the workshop on virtual environments 2003*, 87-95. USA: ACM New York.

Bonsor, K. (2001). How augmented reality will work. *HowStuffWorks*. Retrieved January 26, 2007, from http://www.howstuffworks.com/augmented-reality.htm/printable

Britannica Concise. (2007). Image processing. In *Encyclopedia Britannica*. Retrieved March 5, 2007, from http://concise.britannica.com/ebc/article-9367827/image-processing

Buchmann, V., Violich, S., Billinghurst, M., & Cockburn, A. (2004). FingARtips: gesture based direct manipulation in Augmented Reality. In *proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia, 212-221*. USA: ACM New York.

Burdea, G.C. & Coiffet, P. (2003). *Virtual reality technology.* Wiley-IEEE.

Center of Technology in Government. (1998). *A survey of system development process models*. University at Albeny/SUNY. Retrieved March 6, 2007, from http://www.ctg.albany.edu/publications/reports/survey_of_sysdev/survey_of_sysdev.pdf

Chaudary, V., & Aggarwal, J.K. (1991). On the complexity of parallel image component labeling. In C.L. Wu, H.D. Schwertman, & K. So (Eds.), *Proceedings of the 1991 International Conference on Parallel Processing, August 12-16, 1991: Algorithms & Applications* (pp. 183-187). CRC Press.

Chen, J.X. (2003). *Guide to graphics software tools*. London: Springer.

Cho, Y., Park, J., & Neumann, U. (1997). Fast color fiducial detection and dynamic workspace extension in video see-through self-tracking augmented reality. In *proceedings of the Fifth Pacific Conference on Computer Graphics and Applications*, 168-177. Los Alamitos, CA: IEEE Computer Society Press.

Comaniciu, D. & Meer, P. (1997). Robust analysis of feature spaces: Color image segmentation. In *proceedings of IEEE Conference on Computer Vision and Pattern Recognition, San Juan, Puerto Rico*, 750-755.

Cooper, N., Keatley, A., Dahlquist, M., Mann, S., Slay, H., Zucco, J. et al. (2004). Augmented Reality Chinese Checker. In *proceedings of the 2004 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, 117-126. USA: ACM New York.

Coquillart, S., & Göbel, M. (2004). *Authoring of mixed reality applications including multi-marker calibration for mobile devices*. Eurographics Symposium on Virtual Environments.

Crago, A. (2008). Top 5 Emerging Phone Technologies. In *HowStuffWorks*. Retrieved March 30, 2008, from http://communication.howstuffworks.com/emerging-phone-technologies.htm/printable

Crowley, J., Berard, F., & Coutaz, J. (1995). Finger tracking as an input device for augmented reality. *International Workshop on Gesture and Face Recognition*, Zurich.

Cunningham, S. (2007). *Computer craphics: Programming in OpenGL for visual communication.* Prentice Hall.

Dorfmüller-Ulhaas, K. & Schmalstieg, D. (2001). Finger tracking for interaction in augmented environments. In *proceedings of the 2nd ACM/IEEE International Symposium on Augmented Reality (ISAR'01).*

EDUCAUSE Learning Initiative. (2005). 7 things you should know about augmented reality. Retrieved March 3, 2007, from http://www.educause.edu/ir/library/pdf/ELI7007.pdf

Erdil, K., Finn, E., Keating, K., Meattle, J., Park, S., & Yoon, D. (2003). *Software maintenance as part of the software life cycle.* Tufts University. Retrieved April 20, 2007, from http://hepguru.com/maintenance/Final_121603_v6.pdf

Fiala, M. (2005). ARTag Rev2 fiducial marker system: Vision based tracking for AR. *Workshop of Industrial Augmented Reality.*

Finney, K.C., & ebrary Inc. (2004). *3D game programming all in one.* UK: Thomson/Course Technology.

Foley, J.D. (1995). *Computer graphics: Principles and practice, Volume 385.* Addison-Wesley.

Fredriksson, K. (2001). *Rotation invariant template matching.* University of Helsinki, Finland. Retrieved March 5, 2007, from http://ethesis.helsinki.fi/julkaisut/mat/tieto/vk/fredriksson/rotation.pdf

Fredriksson, K., Mäkinen, V., & Navarro, G. Rotation and lighting invariant template matching. *Information and Computation, 205*(7), 1096-1113. Elsevier.

Geiger, C., Oppermann, L., & Reimann, C. (2003). 3D-registered interaction-surface in augmented reality space. *IEEE International Augmented Reality Toolkit Workshop*, 5-13.

Gillet, A., Sanner, M., Stoffler, D., Goodsell, D., & Olson, A. (2004). Augmented reality with tangible auto-fabricated models for molecular biology applications. In *proceedings of the Conference on Visualization*, 235-241. Los Alamitos, CA: IEEE Computer Society.

Goldstein, E.B. (2005). *Cognitive psychology: Connecting mind, research and everyday experience*. New York: Thomson Wadsworth.

Gordon, G., Billinghurst, M., Bell. M., Woodfill, J., Kowalik, B., Erendi, A. et al. (2002). The use of dense stereo range data in augmented reality. In *proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR 2002)*.

Goshtasby, A.A. (2005). *2-D and 3-D Image Registration*. Wiley-IEEE.

Grimson, W.E.L., & Mundy, J.L. (1994). *Computer vision applications. Communications of the ACM, 37*(3), 44-51. USA: ACM New York.

Haerberli, P. & Segal, M. (1993). Texture mapping as a fundamental drawing primitive. In *proceedings of the Fourth Eurographics Workshop on Rendering*, 259-266.

Haller, M. (2002). Student projects using ARToolKit. *IEEE International Augmented Reality Toolkit Workshop*, 2.

Haller, M., Hartmann, W., Luckeneder, T., & Zauner, J. (2002). Combining ARToolKit with scene graph librarires. In *proceedings of the First IEEE International Augmented Reality Toolkit Workshop*.

Hamsphire, A., Seicher, H., Grasset, R., & Billinghurst, M. (2006). Augmented reality authoring: Generic context from programmer to designer. In *proceedings of the 20th Conference of the Computer-Human Interaction Special Interest Group (CHISIG) of Australia*, 409-412. New York: ACM.

Hardenberg, C. (2001). *Fingertracking and handposture recognition for real-time human-computer interaction*. University Berlin. Retrieved March 6, 2007, from http://iihm.imag.fr/publs/2001/Hardenberg01_FingerTracking.pdf

Hardenberg, C. & Bérard, F. (2001). Bare-hand human-computer interaction. In *proceedings of the 2001 Workshop on Perceptive User Interfaces*, 1-8. USA: ACM New York.

Heckbert, P. (1986). Survey of texture mapping. *IEEE Computer Graphics and Application, 6*(11), 56-67.

Hornberg, A., & NetLibrary Inc. (2007). *Handbook of Machine Vision*. Wiley-VCH.

Hubona, G.S., Shirah, G.W., & Jennings, D.K. (2004). The effects of cast shadows and stereopsis on performing computer-generated spatial tasks. *Systems, Man and Cybernetics, Part A, IEEE Transactions, 34*(4), 483-493.

Irawati, S., Green, S., Billinghurst, M., Duenser, A., & Ko, H. (2006). "Move the couch where?": Developing an augmented reality multimodal interface. In *proceedings of the 2006 Fifth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'06)*, 183-186. Washington, USA: IEEE Computer Society.

Kanbara, M., & Yokoya, N. (2002). Geometric and photometric registration for real-time augmented reality. In *IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'02)*, 279. Los Alamitos, CA: IEEE Computer Society

Kato, H., & Billinghurst, M. (1999). Marker tracking and HMD calibration for a video-based augmented reality conference system. In *proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality, 99*, 85-94. IEEE Computer Society.

Kato, H., Billinghurst, M., Morinaga, K., & Tachibana, K. (2001). The effect of spatial cues in augmented reality conferencing. In *proceedings of the 9th International Conference on Human-Computer Interaction (HCI International 2001)*. Lawrence Erlbaum Associates.

Kato, H., Billinghurst, M., Poupyrev, I., Immato, K., & Tachibana. (2000). Virtual object manipulation on a table-top AR environment. In *proceedings of the International Symposium on Augmented Reality (ISAR 2000)*, 111-119. New York: ACM.

Kawano, T., Ban, Y., & Uehara, K. (2003). A coded visual marker for video tracking system based on structured image analysis. In *proceedings of the 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality*. Washington, DC: IEEE Computer Society.

Kerdvibulvech, C., & Saito, H. (2008). Guitarist fingertip tracking by integrating a Bayesian classifier into particle filters. *Advances in Human-Computer Interaction, 2008*. New York: Hindawi Publishing Corp.

Kilgard, M.J. (1996). *The OpenGL Utility Toolkit (GLUT) programming interface API version 3. Silicon Graphics, Inc*. Retrieved October 14, 2008, from http://users.informatik.uni-halle.de/~schenzel/ws02/opengl/spec3.pdf

Kiss, S. (2002). *Computer animation for articulated 3D characters*. University of Twente, Technical Report TR-CTIT-02-45, ISSN 1381-3625.

Kjeldsen, R., Pinhanez, C., Pingali, G., Hartman, J., Levas, T., Podlaseck, M. et al. (2002). Interacting with steerable projected displays. In *proceedings of Fifth IEEE International Conference on Automatic Face and Gesture Recognition*, 402-407. IEEE Computer Society.

Klinker, G. (1999). Augmented reality: A problem in need of many computer vision-based solutions. In *NATO Advanced Research Workshop at the 8 International Conference on the Computer Analysis of Images and Patterns (CAIP-99)*. Springer.

Koike, H., Sato, Y., & Kobayashi, Y. (2001). Integrating paper and digital information on EnhancedDesk: A method for realtime finger tracking on an augmented desk system. *ACM Transactions on Computer-Human Interaction, 8*(4), 307-322. New York: ACM.

Koller, D., Klinker, G., Rose, E., Breen, D., Whitaker, R., & Tuceryan, M. (1997). Real-time vision-based camera tracking for augmented reality applications. In *proceedings of the ACM Symposium on Virtual Reality Software and Technology*, 87-94. USA: ACM New York.

Kovač, J., Peer, P., & Solina, F. (2003). Human skin colour clustering for face detection. In *proceedings of IEEE Region 8 Computer as Tool (EUROCON'02), 2*, 144-148. Washington, DC: IEEE Computer Society.

Letessier, J. & Bérard, F. (2004). Visual tracking of bare fingers for interactive surfaces. In *ACM Symposium on User Interface Software and Technology (UIST)*, 119-122. New York: ACM.

Liu, Y., Storring, M., Moeslund, T.B., Madsen, C.B., & Granum, E. (2003). Computer vision based head tracking from re-configurable 2D markers for AR. In *proceedings of the Second IEEE and ACM International Symposium on Mixed and Augmented Reality*, 264-267. Washington, DC: IEEE Computer Society.

Lourakis, M.I.A., & Argyros, A.A. (2004). Vision-based camera motion recovery for augmented reality. In *Computer Graphics International Conference (CGI'04)*. Washington, DC: IEEE Computer Society.

Mackay, W.E. (1998). Augmented reality: Linking real and virtual worlds: A new paradigm for interacting with computers. In *proceedings of the Working Conference on Advanced Visual Interfaces*, 13-21. ACM New York.

Malik, S. (2002). *Robust registration of virtual objects for real-time augmented reality.* Carleton University. Retrieved February 1, 2007, from http://www.cv.iit.nrc.ca/~gerhard/PubSummary2/malik_thesis_final.pdf

McMillan, L. (1997). *An image-based approach to three-dimensional computer graphics*. University of North Carolina at Chapel Hill: USA. Retrieved October 19, 2008, from ftp://ftp.cs.unc.edu/pub/technical-reports/97-013.pdf

Mealing, S. (1998). *The art and science of computer animation*. Intellect Books.

Meegoda, J.N., Juliano, T.M., & Banerjee, A. (2006). Framework for automatic condition assessment of culverts. *Transportation Research Record, 1948*(1), 26-34. Transportation Research Board of the National Academies.

Meer, P., Stewart, C.V., & Tyler, D.E. (2000). Robust computer vision: An interdisciplinary challenge. *Computer Vision and Image Understanding, 78*(1), 1-7.

Mehta, D.P., & Sahni, S. (2005). *Handbook of data structures and applications*. USA: CRC Press.

Milgram, P. & Kishino, F. (1994). A taxonomy of mixed reality visual displays. *IECE Transactions on Information and Systems E Series D, 77*, 1321-1321. Institute of Electronics, Information & Communication Engineers.

Miller, N. (2000). OpenGL Texture Mapping: An Introduction. In *GameDev.net*. Retrieved October 13, 2008, from http://www.gamedev.net/reference/articles/article947.asp

Mooser, J., You, S., & Neumann, U. (2006). TriCodes: A barcode-like fiducial design for augmented reality media. In *Multimedia and Expo, 2006 IEEE*

International Conference, 1301-1304.

Oda, O., Lister, L.J., White, S., & Feiner, S. (2008). Developing an augmented reality racing game. In *proceedings of the 2nd International Conference on Intelligent Technologies for Interactive Entertainment*, Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (ICST), Belgium.

Oka, K., Sato, Y., & Koike, H. (2002). Real-time fingertip tracking and gesture recognition. *IEEE Computer Graphics and Applications, 22*(6), 64-71. IEEE Computer Society.

Ong, S.K., Chong, J.W.S., & Nee, A.Y.C. (2006). Methodologies for immersive robot programming in an augmented reality environment. In *proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia*, 237-244.

Parekh, R. (2006). *Principles of Multimedia.* New Dehli: Tata McGraw-Hill.

Phillips, K., & Piekarski, W. (2005). Possession techniques for interaction in real-time strategy augmented reality games. In *proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*. ACM New York.

Piekarski, W., & Thomas, B. (2002). ARQuake: The outdoor augmented reality gaming system. *Communications of the ACM, 45*(1), 36-38. USA: ACM New York.

Pintaric, T. (2003). An adaptive thresholding algorithm for the Augmented Reality Toolkit. In *proceedings of the Second IEEE International Augmented Reality Toolkit Workshop (ART03)*. IEEE Computer Society.

Pitas, I. (2000). *Digital image processing algorithms and applications*. Canada: Wiley-IEEE.

Reiners, D., Stricker, D., Klinker, G., & Müller, S. (1998). Augmented reality for construction tasks: Doorlock assembly. In *proceedings of the International Workshop on Augmented Reality (IWAR'98): Placing artificial objects in real scenes*, 31-46. Natick, MA, USA: A. K. Peters, Ltd.

Rekimoto, J., & Ayatsuka, Y. (2000). Cybercode: Designing augmented reality environments with visual tags. In *proceedings of DARE 2000 on designing augmented reality environments*, 1-10.

Russ, J.C. (2007). *The image processing handbook*. USA: CRC Press.

Sanchez, J., & Canton, M.P. (2003). *The PC Graphics Handbook*. USA: CRC Press.

Sarath. (2007). How to save bitmap to file? Retrieved April 9, 2007, from http://sarathc.wordpress.com/2007/03/14/how-to-save-bitmap-to-file/

Schierwagen, A. (2001). Vision as computation, or: Does a computer vision system really assign meaning to images? M. Matthies, H. Malchow, & J. Kriz (eds.). *Integrative systems approaches to natural and social sciences* (pp. 579-787). Berlin, Germany: Springer-Verlag.

Shirley, P., Ashikhmin, M., Gleicher, M., Marschner, S., Reinhard, R., Sung, K., et al. (2005). *Fundamentals of Computer Graphics*. A K Peters, Ltd.

Shrivastava, P. (2005). *Implementation and analysis of eidochromatic transfrom for color image compression*. Texas Tech University. Retreived September 16, 2008, from http://etd.lib.ttu.edu/theses/available/etd-03302005-162035/unrestricted/ETD_submit_0405.pdf

Simon, G. & Berger, M.O. (2000). Registration with a moving zoom lens camera for augmented reality applications. In D. Vernon (Ed.), *Computer Vision, ECCV 2000: 6th European Conference on Computer Vision, Dublin, Ireland, June 26-July 1, 2000: Proceedings* (pp. 578-594). Germany: Springer-Verlag.

Simpson, J. (2002). Game engine anatomy 101. In *ExtremeTech*. Retreived October 19, 2008, from http://www.extremetech.com/article2/0,2845,594,00.asp

Sink, K. (2001). *DirectX 8 and Visual Basic Development*. USA: Sams Publishing.

Slay, H., Thomas, B., & Vernik, R. (2002). Tangible user interaction using augmented reality. In *proceedings of the Third Australian Conference on User Interfaces*, *7*, 13-20.

Starner, T., Mann, S., Rhodes, B., Levine, J., Healey, J., Kirsch, D., Picard, R.W., & Pentland, A. (1997). Augmented reality through wearable computing. *Presence, 6*(4), 386-398.

Thayananthan, A., Navaratnam, R., Torr, P.H.S., & Cipolla, R. (2004). Likelihood models for template matching using the PDF projection theorem. In *proceedings of British Machine Vision Conference (BMVC 2004)*. British Machine Vision Association.

Thomas, B.H. (2003). Challenges of making outdoor augmented reality games playable. In *2nd CREST Workshop on Advanced Computing and Communicating Techniques for Wearable Information Playing in Nara, Japan.*

Thomas, B.H. & Piekarski, W. (2002). Glove based user interaction techniques for augmented reality in an outdoor environment. *Virtual Reality, 6*(3), 167-180.

Thorn, A. (2005). *DirectX 9 graphics: The definitive guide to Direct3D*. USA: Wordware Publishing, Inc.

Wagner, D., & Schmalstieg, D. (2007). ARToolKitPlus for pose tracking on mobile devices. In *proceedings of 12th Computer Vision Winter Workshop (CVWW'07)*, 6-8.

Williams, R.H., Kent, A., Holzman, A.G., & Williams, J.G. (1993). *Encyclopedia of computer science and technology: Volume 29*. CRC Press.

Woods, E., Mason, P., & Billinghurst, M. (2003). MagicMouse: An inexpensive 6-degree-of-freedom mouse. In *proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques (GRAPHITE 2003)*. New York: ACM.

Woods, J.W. (2006). *Multidimensional signal, image, and video processing and coding*. USA: Academic Press.

Zarit, B.D., Super, B.J., & Quek, F.K.H. (1999). Comparison of five color models in skin pixel classification. In *proceedings of the International Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, 50*. USA: IEEE Computer Society.

Zhang, H., & Liang, D. (2007). *Computer graphics using Java 2D and 3D*. USA: Prentice Hall.

Zhang, X., Fronz, S., & Navab, N. (2002). Visual marker detection and decoding in AR systems: A comparative study. In *proceedings of IEEE International Symposium on Mixed and Augmented Reality (ISMAR'02)*, 79-106. Washington, DC, USA: IEEE Computer Society.

Zhou, Y., Wang, Y., Yan, D., & Xu, T. (2000). Vision-based registration using 3-D fiducial for augmented reality. In T. Tan, Y. Shi., & W. Gao, *Advances in Multimodal Interfaces – ICMI 2000: Third International Conference, Beijing, China, October 14-16, 2000: Proceedings*. Germany: Springer.

APPENDICES

Appendix 1: Consent Form

Research Participant's Consent Form
Universiti Malaysia Sarawak (UNIMAS)
Faculty of Cognitive Sciences and Human Development
Research on "AR Application Builder:
Integration of Finger Tracking for Interaction in AR System"

I hereby consent to participate as a subject in a research entitled "AR Application Builder: Integration of Finger Tracking for Interaction in AR System" conducted by Allen Choong Chieng Hoon from Faculty of Cognitive Science and Human Development, UNIMAS.

I will be asked to perform specific tasks. I am allowed to say whatever comes to mind while I work. I can verbalize or talk incomplete sentences or 'making sense'. The research study will take about 30 minutes, however the duration of task is at my preference.

I have been informed that data collected will be kept in strict confidence. I am free to ask questions at any time without penalty. As I am working on the tasks, I will not be provided help or answer questions. Even though the experimenter is not able to answer most of my questions, I can ask them anyway. The experimenter will note down my questions and answer any questions that I still have after I have finished the entire task. This is because the experimenter wants to create the most realistic situation possible.

I understand that my participation is completely voluntary, and that I am free to withdraw from the research at any time I choose without penalty. I understand that this research is not expected to involve risks of harm any greater than those ordinarily encountered in daily life. I also understand that it is not possible to identify all potential risks in any procedure, but that all reasonable safeguards have been taken to minimise the potential risks. To the best of experimenter's knowledge, there are no physical or psychological risks associated with the procedures in this study

"I have read the above description of the study and of my rights as a subject. I agreed to participate in this study."

Signature of subject: _____ Date: _____

Appendix 2: Evaluation Form

This test case aims to test how well you can use your finger to control the application.

No.	Tasks	Success (√) / Fail (x)	Comments/Remarks
1	Use a web camera to detect the marker to show the buttons		
2	Use the finger to click play button to start playing the video in AR		
3	Click pause button to pause the video		
4	Click play button to resume the video		
5	Click stop button to stop the video		
6	Click play button to play the video from the beginning		
7	Watch the video completely		

Appendix 3: List of Participated Conferences and Exhibitions of this Research Project

- Ng, G. W. Amalia, M., Allen Choong, C.H. & Angeline Lee, L. S. (2007). *Augmented Toys Technology (ATTech) in Helping Traumatized Children During Disasters*. 1st International Malaysian Education Technology Convention (IMETC 2007), Universiti Technology Malaysia, Johor.
- Ng, G.W. & Allen, C.C.H. (2008). *Augmented Reality: The Potential of Future Educational Technology.* Joint Colloquium on "Cognitive and Computational Methods", Universiti Malaysia Sarawak.
- **Inaugural Unimas R & D Exhibition 2007, CAIS UNIMAS, 24-26 January 2007**

  4th Place: Augmented Reality in Education
- **Malaysian Technology Exhibition (MTE) 2007, PWTC Kuala Lumpur, 29-31 March 2007**

  Won Bronze Medal Award

  Magic Board-Games: A New Way of Games Entertainment
- International Exposition of Research and Invention of Institutions of Higher Learning 2007 (PECIPTA 2007), Kuala Lumpur Convention Centre (KLCC), 10-12 August 2007
- Exhibition Augmented Reality in Education during visitor, Dr. Lai from MIMOS Berhad to University Malaysia Sarawak, Malaysia.
- Exhibition Augmented Reality in Education for Signing Ceremony &

Memorandum (MoU) Documents Exhanges, Palace of The Golden Horses Hotel, Seri Kembangan Selangor, Mines, KL.

- **MSC Malaysia Mobile Interactive Content Competition 2007, Hotel Crowne Plaza Princess, KL, 03 August 2007.**

  Short listed the top 6 to pitch for the competition: Augmented Toys Technology (ATTech).

- Exhibition Augmented Reality in Education during visitors from Fakultas Keguruan dan Ilmu Pendidikan (FKIP), Universitas Tangjung Pura, Pontianak, Kalimantan, INDONESIA, to Faculty of Cognitive Sciences and Human Development, University Malaysia Sarawak, Malaysia, 27 August 2007.

- **MSC Malaysia APICTA 2007**

  Won Best Top 3 Research Project for the Best of Education & Training category.

- **Maxis Mobile Content Challenge 2007**

  Participate

- **Inaugural Unimas R & D Exhibition 2008, CAIS UNIMAS, 05-07 March 2008**

  3rd Place: Augmented Reality (AR) Application Builder: An Interactive Real Time 3D Environment.

- **MSC Malaysia APICTA 2008**

15 October 2008, KLPAC

Winner for Best Tourism and Hospitality Category

Project Title: An Augmented Reality System for Recognising Text on Street Signs for Tourism

Winner for Best Tertiary Student Project – Software/ Hardware Category

Project Title: Finger Tracking and Bare-Hand Posture as an Input Device using Augmented Reality

Winner for Best Tertiary Student Project – Creative Multimedia Category

Project Title: An Interactive Augmented Reality Games: 3D Aeroplane Games (3DCoolPlane)

- **International APICTA 2008**

  12-15 November 2008, Jakarta, Indonesia

  Merit Award for Best Tourism and Hospitality Category

  Project Title: An Augmented Reality System for Recognising Text on Street Signs for Tourism

Appendix 4: Source Code

```
////////////DirectShow
class CMyShow
{
public:
  CMyShow();
  ~CMyShow();

  //This function is to load the capture device such as webcam
  int LoadCaptureDevice();

  //This function is to load the .avi file
  int LoadAvi(LPCWSTR filename);

  //This function is to run the stream
  void Run();
  //This function is to set window
  void SetWindow(HWND hwnd,int ON_OFF);

  CSampleGrabberCB grabCB;
  IMediaControl *mediaControl;
  IGraphBuilder *pGraph;

  IMediaSeeking *mediaSeek;

  //For the reference time, we need only their address
  REFERENCE_TIME curr; //Current time
  void Stop();
  void Pause();
  void Play();

  PLAYSTATE state; //Status

  void Destroy(); //Destroy all the filter, including filter
graph and grabber.

private:
  void InitMediaSeek();
};

CMyShow::CMyShow() {}
CMyShow::~CMyShow() {}

int CMyShow::LoadCaptureDevice()
{
  HRESULT hr;

  //CaptureGraph
  ICaptureGraphBuilder2 *pCapture;

hr=CoCreateInstance(CLSID_CaptureGraphBuilder2,NULL,CLSCTX_INPROC
```

```
,IID_ICaptureGraphBuilder2,
      (void**)&pCapture);
  if(FAILED(hr)) printf("Fail pCapture\n");

  //FilterGraph

hr=CoCreateInstance(CLSID_FilterGraph,NULL,CLSCTX_INPROC,IID_IGra
phBuilder,(void**)&pGraph);
  if(FAILED(hr)) printf("Fail pGraph\n");

  //Attach FilterGraph to the CaptureGraph, pCapture connect
pGraph
  hr=pCapture->SetFiltergraph(pGraph);
  if(FAILED(hr)) printf("Fail SetFiltergraph\n");

  //Source Filter
  IBaseFilter *pSrcFilter=NULL;
  hr=FindCaptureDevice(&pSrcFilter);
  if(FAILED(hr))
  {
    printf("Fail FindCaptureDevice\n");
    return 0; //To inform that, if return 0, that means fail to
find capture device
  }

  //Add pSrcFilter to pGraph, pSrcFilter connect pGraph
  hr=pGraph->AddFilter(pSrcFilter,L"Video Capture");
  if(FAILED(hr)) printf("Fail AddFilter Video Capture\n");

  //Connect StreamConfig with Source Filter through
ICaptureGraphBuilder2::FindInterface
  IAMStreamConfig *pStreamCfg;
    hr = pCapture->FindInterface(&PIN_CATEGORY_CAPTURE,
        &MEDIATYPE_Interleaved, pSrcFilter,
        IID_IAMStreamConfig, (void **)&pStreamCfg);

    if(hr != NOERROR)
        hr = pCapture->FindInterface(&PIN_CATEGORY_CAPTURE,
            &MEDIATYPE_Video, pSrcFilter,
            IID_IAMStreamConfig, (void **)&pStreamCfg);

  //For the property, because it is for IPin, therefore must use
StreamConfig to
  ISpecifyPropertyPages *pProp;
  CAUUID cauuid;
  hr=pStreamCfg->QueryInterface(IID_ISpecifyPropertyPages,
(void**)&pProp);
  if(hr==S_OK)
  {
    hr=pProp->GetPages(&cauuid);
    hr = OleCreatePropertyFrame(NULL, //This should be parent
window
        30,30,NULL,1,
        (IUnknown**)&pStreamCfg,cauuid.cElems,
        (GUID*)cauuid.pElems,0,0,NULL);
```

```
  }
  CoTaskMemFree(cauuid.pElems);
  pProp->Release();

  IPin *outSource;

hr=GetUnconnectedPin(pSrcFilter,PINDIR_OUTPUT,&outSource);//pSrcF
ilter is Video Capture
  if(FAILED(hr)) printf("Fail GetUnconnectedPin for
outSource\n");

  //BaseFilter
  IBaseFilter *pGrabFilter; //Use IBaseFilter because want to
enumerates Pins

hr=CoCreateInstance(CLSID_SampleGrabber,NULL,CLSCTX_INPROC,IID_IB
aseFilter,
    (void**)&pGrabFilter); //Use SampleGrabber
  if(FAILED(hr)) printf("Fail pGrabber\n");

  //Add pGrabFilter to pGraph, pGrabFilter connect pGraph, where
pGrabFilter is for pGrabber
  hr=pGraph->AddFilter(pGrabFilter,L"Sample Grabber");
  if(FAILED(hr)) printf("Fail AddFilter Sample Grabber\n");

  //pGrabFilter query interface
  ISampleGrabber *pGrabber;
  hr=pGrabFilter->QueryInterface(IID_ISampleGrabber,
(void**)&pGrabber);
  if(FAILED(hr)) printf("Fail pGrabber\n");

  //The media type that pGrabber will be connected
  AM_MEDIA_TYPE setType;
  ZeroMemory(&setType, sizeof(AM_MEDIA_TYPE));
  setType.majortype = MEDIATYPE_Video;
  setType.subtype = MEDIASUBTYPE_RGB24;
  hr = pGrabber->SetMediaType(&setType);
  if(FAILED(hr)) printf("Fail SetMediaType\n");

  IPin *inGrab;

hr=GetUnconnectedPin(pGrabFilter,PINDIR_INPUT,&inGrab);//pGrabFil
ter is SampleGrabber Filter
  if(FAILED(hr)) printf("Fail GetUnconnectedPin for inGrab\n");
  //Connect 2 pins
  hr=pGraph->Connect(outSource,inGrab);
  if(FAILED(hr)) printf("Fail Connect 2 pins\n");

  AM_MEDIA_TYPE mt;
  hr=pGrabber->GetConnectedMediaType(&mt);
  if(FAILED(hr)) printf("Fail ConnectedMediaType\n");
  // Examine the format block.
  VIDEOINFOHEADER *vih;
  if ((mt.formattype == FORMAT_VideoInfo) &&
     (mt.cbFormat >= sizeof(VIDEOINFOHEADER)) &&
```

```
    (mt.pbFormat != NULL) )
{
    vih = (VIDEOINFOHEADER*)mt.pbFormat;

    grabCB.Width=vih->bmiHeader.biWidth;
    grabCB.Height=vih->bmiHeader.biHeight;
}
else
{
    // Wrong format. Free the format block and return an error.
    FreeMediaType(mt);
    return 0;
}

  // Free the format block when you are done:
  FreeMediaType(mt);

  //Need to render the pGrabFilter
  IPin *outGrab;
  hr=GetUnconnectedPin(pGrabFilter,PINDIR_OUTPUT,&outGrab);

  //Using Null Renderer, so that will not popup a window
  //Because if popup a window, need to use IVideoWindow to turn
off the window
  IBaseFilter *pNullRender;

CoCreateInstance(CLSID_NullRenderer,NULL,CLSCTX_INPROC,IID_IBaseF
ilter,
     (void**)&pNullRender);
  pGraph->AddFilter(pNullRender,L"Null Renderer");

  pGraph->Render(outGrab);

  //Setup pGrabber
  hr=pGrabber->SetOneShot(FALSE);
  if(FAILED(hr)) printf("Fail SetOneShot\n");
  hr=pGrabber->SetBufferSamples(FALSE);
  if(FAILED(hr)) printf("Fail SetBufferSamples\n");
  //Callback function
  pGrabber->SetCallback(&grabCB,1);

  //Release pSrcFilter
  pSrcFilter->Release();
  pGrabFilter->Release();
  pNullRender->Release();

  this->InitMediaSeek();

  return 1;
}


void CMyShow::Run()
{
  pGraph->QueryInterface(IID_IMediaControl,
```

```
(void**)&mediaControl);
  mediaControl->Run();

  this->state=play;
}

int CMyShow::LoadAvi(LPCWSTR filename)
{
  HRESULT hr;

  //FilterGraph

hr=CoCreateInstance(CLSID_FilterGraph,NULL,CLSCTX_INPROC,IID_IGra
phBuilder,(void**)&pGraph);
  if(FAILED(hr)) printf("Fail pGraph\n");

  IBaseFilter *pFileFilter;

hr=CoCreateInstance(CLSID_AsyncReader,NULL,CLSCTX_INPROC,IID_IBas
eFilter,
     (void**)&pFileFilter);
  if(FAILED(hr)) printf("Fail pFileFilter\n");

  hr=pGraph->AddFilter(pFileFilter,L"Source Filter");
  if(FAILED(hr)) printf("Fail pGraph->AddFilter\n");

  IFileSourceFilter *pFile;
  hr=pFileFilter->QueryInterface(IID_IFileSourceFilter,
(void**)&pFile);
  if(FAILED(hr)) printf("Fail pFileFilter->QueryInterface\n");

  hr=pFile->Load(filename,NULL);
  if(FAILED(hr))
  {
    printf("Fail pFile->Load\n");
    return 0; //Fail to load the file
  }

  IPin *outFilePin;
  hr=GetUnconnectedPin(pFileFilter,PINDIR_OUTPUT,&outFilePin);
  if(FAILED(hr)) printf("Fail GetUnconnectedPin outFilePin\n");

  IBaseFilter *pAviSplitFilter;

hr=CoCreateInstance(CLSID_AviSplitter,NULL,CLSCTX_INPROC,IID_IBas
eFilter,
     (void**)&pAviSplitFilter);
  if(FAILED(hr))printf("Fail pAviSplitFilter\n");

  hr=pGraph->AddFilter(pAviSplitFilter,L"AVI Splitter");
  if(FAILED(hr)) printf("Fail AddFilter pAviSplitFilter\n");

  IPin *inSplitPin;
  hr=GetUnconnectedPin(pAviSplitFilter,PINDIR_INPUT,&inSplitPin);
  if(FAILED(hr)) printf("Fail inSplitPin\n");
```

```
hr=pGraph->Connect(outFilePin,inSplitPin);
if(FAILED(hr)) printf("Fail connected\n");

IPin *outSplitPin1;
hr=GetPin(pAviSplitFilter,PINDIR_OUTPUT,0,&outSplitPin1);
if(FAILED(hr)) printf("Fail outSplitPin1\n");

IPin *outSplitPin2;
hr=GetPin(pAviSplitFilter,PINDIR_OUTPUT,1,&outSplitPin2);
if(FAILED(hr)) printf("Fail outSplitPin2\n");

IBaseFilter *pGrabFilter;

hr=CoCreateInstance(CLSID_SampleGrabber,NULL,CLSCTX_INPROC,IID_IB
aseFilter,
     (void**)&pGrabFilter);
  if(FAILED(hr)) printf("Fail pGrabFilter\n");

  hr=pGraph->AddFilter(pGrabFilter,L"SampleGrabber");
  if(FAILED(hr)) printf("Fail AddFilter pGrabFilter\n");

  IPin *inGrabPin;
  hr=GetUnconnectedPin(pGrabFilter,PINDIR_INPUT,&inGrabPin);
  if(FAILED(hr)) printf("Fail inGrabPin\n");

  ISampleGrabber *pGrabber;
  hr=pGrabFilter->QueryInterface(IID_ISampleGrabber,
(void**)&pGrabber);
  if(FAILED(hr)) printf("Fail pGrabFilter QueryInterface\n");

    AM_MEDIA_TYPE grabType;
ZeroMemory(&grabType, sizeof(AM_MEDIA_TYPE));
grabType.majortype = MEDIATYPE_Video;
grabType.subtype = MEDIASUBTYPE_RGB24;

  hr = pGrabber->SetMediaType( &grabType );
if(FAILED(hr)) printf("Fail pGrabber->SetMediaType\n");

bool invert=false; //Used twice
hr=pGraph->Connect(outSplitPin1,inGrabPin);
if(FAILED(hr))
{
   hr=pGraph->Connect(outSplitPin2,inGrabPin);
   if(FAILED(hr)) printf("Fail connect both pins\n");

   invert=true;
}
   //}

IPin *outGrabPin;
hr=GetUnconnectedPin(pGrabFilter,PINDIR_OUTPUT,&outGrabPin);
if(FAILED(hr)) printf("Fail outGrabPin\n");

AM_MEDIA_TYPE mt;
```

```
hr=pGrabber->GetConnectedMediaType(&mt);
if(FAILED(hr)) printf("Fail ConnectedMediaType\n");

VIDEOINFOHEADER *vih;
if ((mt.formattype == FORMAT_VideoInfo) &&
   (mt.cbFormat >= sizeof(VIDEOINFOHEADER)) &&
   (mt.pbFormat != NULL) )
{
   vih = (VIDEOINFOHEADER*)mt.pbFormat;

   grabCB.Width=vih->bmiHeader.biWidth;
   grabCB.Height=vih->bmiHeader.biHeight;
}
else
{
   // Wrong format. Free the format block and return an error.
   FreeMediaType(mt);
   return 0;
}

// Free the format block when you are done:
FreeMediaType(mt);

hr=pGrabber->SetOneShot(FALSE);
if(FAILED(hr)) printf("Fail SetOneShot\n");
hr=pGrabber->SetBufferSamples(FALSE);
if(FAILED(hr)) printf("Fail SetBufferSamples\n");
hr=pGrabber->SetCallback(&grabCB,1);
if(FAILED(hr)) printf("Fail pGrabber->SetCallback\n");//*/

hr=pGraph->Render(outGrabPin);
if(FAILED(hr)) printf("Fail pGraph->Render\n");

if(invert==false)
{
   hr=pGraph->Render(outSplitPin2);
   if(FAILED(hr)) printf("Fail pGraph->Render(outSplitPin2)\n");
}
else
{
   hr=pGraph->Render(outSplitPin1);
   if(FAILED(hr)) printf("Fail pGraph->Render(outSplitPin1)\n");
}

pFileFilter->Release();
pAviSplitFilter->Release();
pGrabFilter->Release();
pNullFilter->Release();

this->InitMediaSeek();

   return 1;

}
```

```
//To set window, you must have video renderer, therefore, there
should be no null renderer
void CMyShow::SetWindow(HWND hwnd,int ON_OFF)
{
  HRESULT hr;

  IVideoWindow *videoWindow;
  hr=pGraph->QueryInterface(IID_IVideoWindow,
(LPVOID*)&videoWindow);
  if(FAILED(hr)) printf("Fail QueryInterface(IID_IVideoWindow,
(void**)&videoWindow)\n");

  //This part is to turn of the video
  if(ON_OFF==0)
  {
    videoWindow->put_AutoShow(OAFALSE);
    return;
  }

  hr=videoWindow->put_Owner((OAHWND)hwnd);
  if(FAILED(hr)) printf("Fail videoWindow-
>put_Owner((OAHWND)hwnd)\n");

  hr=videoWindow->put_WindowStyle(WS_CHILD|WS_CLIPCHILDREN);
  if(FAILED(hr)) printf("Fail put_WindowStyle\n");

  if(videoWindow)
  {
    RECT rc;
    GetClientRect(hwnd,&rc);
    videoWindow->SetWindowPosition(0,0,rc.right,rc.bottom);
  }

  hr=videoWindow->put_Visible(OATRUE);
  if(FAILED(hr)) printf("Fail put_Visible\n");
}

void CMyShow::InitMediaSeek()
{
  HRESULT hr=pGraph->QueryInterface(IID_IMediaSeeking,
(void**)&mediaSeek);
  if(FAILED(hr)) printf("Fail MediaSeeking\n");
}

void CMyShow::Stop()
{
  REFERENCE_TIME start=0;
  this->mediaSeek-
>SetPositions(&start,AM_SEEKING_AbsolutePositioning,
    NULL,AM_SEEKING_NoPositioning);
    //So that, after stoping, the position of the media will go
to "0"

  //To avoid curr time mixed with pause, set curr to 0
  this->curr=0;
```

```
  //And need to stop running the media
  this->mediaControl->Stop();

  this->state=stop;
}

void CMyShow::Pause()
{
  this->mediaControl->Pause();
  this->mediaSeek->GetPositions(&this->curr,NULL);
  //Since it is pausing, therefore need to GetPositions()
  // of the current time.

  this->state=pause;
}

void CMyShow::Play()
{
  //Check the current time first
  if(this->curr==NULL)
    this->curr=0;

  this->mediaSeek->SetPositions(&this-
>curr,AM_SEEKING_AbsolutePositioning,
    NULL,AM_SEEKING_NoPositioning);
  this->mediaControl->Run();

  this->state=play;
}

void CMyShow::Destroy()
{
  this->mediaControl->Stop();
  this->state=stop;

  //From DirectShow documentation
  IEnumFilters *pEnumFilter=NULL;
  HRESULT hr=this->pGraph->EnumFilters(&pEnumFilter);
  if(SUCCEEDED(hr))
  {
    IBaseFilter *pFilter=NULL;
    while(S_OK==pEnumFilter->Next(1,&pFilter,NULL))
    {
      pGraph->RemoveFilter(pFilter);
      pEnumFilter->Reset();
      pFilter->Release();
    }
    pEnumFilter->Release();
  }

  this->pGraph->Release();
  this->mediaControl->Release();
  this->mediaSeek->Release();
```

```
  delete[] this->grabCB.buffer;
  this->grabCB.buffer=NULL;
  this->grabCB.Release();
}

////////////////Image library
void PictureTranspose(unsigned char** pData,int* width,int*
height) {
  int w=*width;
  int h=*height;
  BYTE* newData=new BYTE[w*h*3];

  int i,j;
  for(j=0;j<h;j++) {
    for(i=0;i<w;i++) {
      memcpy(&newData[ (i*h + j) * 3],*pData + ((j*w+i) * 3),3);
    }
  }
  delete[] *pData;
  *pData=newData;
  *width=h;
  *height=w;
}

void PictureGreyscale(unsigned char** pData,int width,int height)
{
  //Coding from
http://www.ambuehler.ethz.ch/CDstore/www6/Posters/701/poster701.h
tml
  float temp;
  int i=0;
  while(i<width*height) {
    temp=0.299f * *(*pData + i*3) + 0.587f * *(*pData + i*3 +1) +
0.114f * *(*pData + i*3 +2); //With the sequence RGB
    *(*pData+i*3) = *(*pData+i*3+1) = *(*pData+i*3+2) =
(int)temp;
    i++;
  }
}

void PictureThresholding(unsigned char** pData,int width,int
height,int threshold) {
  int i;
  for(i=0;i<width*height;i++) {
    *(*pData + i*3) = *(*pData+i*3) > threshold ? 255 : 0;
    *(*pData +i*3 +1) = *(*pData+i*3+1) > threshold ? 255 : 0;
    *(*pData +i*3 +2) = *(*pData+i*3+2) > threshold ? 255 : 0;
  }
}

int LoadBmp(const char* filename,unsigned char** pData,int*
width,int *height) {
  cout<<"Start loading bitmap... "<<filename;

  ifstream file;
```

```
file.open(filename,ios::binary);

if(!file.is_open()) {
   cout<<"Fail open file."<<endl;
   return 0;
}

//Part 1: Get BITMAPFILEHEADER
BITMAPFILEHEADER bmfh;
file.read((char*)&bmfh,sizeof(BITMAPFILEHEADER));

//Check header
if(bmfh.bfType!=19778) { //"19778" is the magic number
   cout<<"Not a bitmap file."<<endl;
   return 0;
}

//Part 2: Get BITMAPINFOHEADER
BITMAPINFOHEADER bmih;
file.read((char*)&bmih,sizeof(BITMAPINFOHEADER));

int w = *width = (int)bmih.biWidth;
int h = *height = (int)bmih.biHeight;
int bitcount=bmih.biBitCount;

if(bitcount<8) {
   cout<<"Error bitmap file."<<endl;
   return 0;
}

  //Part 3: Get the RGBQUAD, if the bitcount is 8.
  //Get the number of colour
  int nColour=1<<bmih.biBitCount;
  RGBQUAD *pColour;
  if(bitcount==8) {
     pColour=new RGBQUAD[nColour];
     file.read((char*)pColour,nColour*sizeof(RGBQUAD));
  }

  //Part 4: Get the image data.
  //Calculate the padding, which the bitmap file, if the width
with rgb cannot be divided by 4,
  // the data will be expanded with extra byte until dividable by
4.
  int padding=CalcWordPadding(w*(float)bitcount/8);
  //Calculate the pixel data size;
  int nDataSize=(int)((w*((float)bitcount/8)+padding) * h);

  BYTE* pTemp=new BYTE[nDataSize];
  file.read((char*)pTemp,nDataSize);

  //Close the file, since reading finish
  file.close();

  //Part 5: Convert the data into RGB pixel format
```

```
  int nImageData = w*abs(h)*3; //"3" because of RGB
    //And afraid that height might be negative
  *pData=new BYTE[nImageData];
  if(bitcount==24)
    ConvertBmp24(*pData,w,h,pTemp,nDataSize);
  else if(bitcount==8)
    ConvertBmp8(*pData,w,h,pTemp,nDataSize,pColour);

  //Clean up
  delete[] pTemp;
  pTemp=NULL;

  if(bitcount==8)
    delete[] pColour;

  cout<<" end"<<endl;

  return 1;
}

int SaveBmp(const char* filename,unsigned char* pData,int
width,int height) {
  cout<<"Start saving bitmap file... "<<filename;

  int padding=CalcWordPadding(width*3);
  int padwidth=width*3 + padding;

  BITMAPINFOHEADER bmih={0};
  bmih.biSize=sizeof(BITMAPINFOHEADER);
  bmih.biBitCount=24;
  bmih.biCompression=BI_RGB;
  bmih.biHeight=height;
  bmih.biWidth=width;
  bmih.biPlanes=1;
  bmih.biSizeImage= padwidth*height;

  BITMAPFILEHEADER bmfh={0};
  bmfh.bfType=19778;
  bmfh.bfOffBits=sizeof(BITMAPINFOHEADER) +
sizeof(BITMAPFILEHEADER);
  bmfh.bfSize=bmfh.bfOffBits + bmih.biSizeImage;

  int i,j;
  ofstream file;
  file.open(filename,ios::binary);
  if(!file.is_open()) {
    cout<<"Fail open file!"<<endl;
    return 0;
  }
  else {
    file.write((char*)&bmfh,sizeof(BITMAPFILEHEADER));
    file.write((char*)&bmih,sizeof(BITMAPINFOHEADER));

    i=0;
    while(i<width*height*3) {
```

```
      file<<pData[i+2]<<pData[i+1]<<pData[i];
      i+=3;
      if(i%(width*3)==0) {
         for(j=0;j<padding;j++) {
            file<<(char)0x00;
         }
      }
   }//*/
}
file.close();

  cout<<" end"<<endl;
  return 1;
}

//////////////////OpenGL draw object
void CalcInterpolate(float *v1,float *v2,int size,int
currFrame,int maxFrame,float *vOut) {
  //Description: This function calculate the interpolation
between v1 and v2
  // according to the currFrame of the maxFrame
  int i;
  for(i=0;i<size;i++) {
     vOut[i]=v1[i] + ((v2[i]-v1[i])*currFrame/maxFrame);
  }
}


void GlPolygon(float *v,int *vi,int size) {
  //Description:
  //v is the vertex array in 1D
  //vi is the vertex index array in 1D, which termintated if the
value is -1,

  float normal[3];
  //Calculate 1st normal;
  VectorNormal2(&v[vi[0]*3],&v[vi[1]*3],&v[vi[2]*3],normal);

  //If the light is disabled
  float colour[4];
  if(glIsEnabled(GL_LIGHTING)==GL_FALSE)
    glGetFloatv(GL_CURRENT_COLOR,colour);//*/

  glBegin(GL_POLYGON);
  int i;
  for(i=0;i<size;i++) {
    if(vi[i]>=0) {
      if(glIsEnabled(GL_LIGHTING)==GL_FALSE) {
        glColor4fv(colour);
      }
      glNormal3fv(normal);
      glVertex3fv(&v[vi[i]*3]);
    }
    else if(vi[i]==-1) {
      glEnd();
```

```
        if(i<size-1) {
          glBegin(GL_POLYGON);
          //For the next normal, if it is not the end


VectorNormal2(&v[vi[i+1]*3],&v[vi[i+2]*3],&v[vi[i+3]*3],normal);
        }
      }
    }
}

void GlPolygonVertexShade(float *v,int *vi,int size) {
  //Description:
  //v is the vertex array in 1D
  //vi is the vertex index array in 1D, which termintated if the
value is -1,

  /* For the vertex shading, each vertex has its own normal. And,
if there is shared vertex
  for several face, the normal will be added up. Therefore, the
algorithm is to check through
  all the vertices, if the vertices connected to 2 or more faces,
then, need to calculate the normal of the
  each face.
  */

  //Note: Remember, when access every vertex, remember to
MULTIPLY BY 3

  //Calculate number of face
  int nFace=0;
  int i;
  for(i=0;i<size;i++) {
    if(vi[i]<0) nFace++;
  }

  //For each face, calculate its normal
  float *normal=new float[nFace*3]; //Because of x,y,z

VectorNormal2(&v[vi[0]*3],&v[vi[1]*3],&v[vi[2]*3],&normal[0]); //
First normal
  int j=1;
  for(i=0;i<size;i++) {
    if(vi[i]==-1 && i<size-1) {


VectorNormal2(&v[vi[i+1]*3],&v[vi[i+2]*3],&v[vi[i+3]*3],&normal[j
*3]);
      j++; //Next face
    }
  }

  //Calculate number of vertices
  int nVertex=0;
  for(i=0;i<size;i++) {
```

```
    if(vi[i]>nVertex)
      nVertex=vi[i];
  }
  nVertex++; //Include 0

  //Calculate normal for each vertex
  float *normal2=new float[nVertex*3]; //Include 0
  memset(normal2,0,nVertex*3*sizeof(float)); //Set every normal
to 0
  j=0; //For face
  for(i=0;i<size;i++) {
    if(vi[i]>=0) {
      VectorAdd(&normal2[vi[i]*3], //So that, the normal2 and v
are in same index
        &normal[j*3], //The normal of the face
        &normal2[vi[i]*3]); //Output
    }
    else if(vi[i]==-1) {
      j++;
    }
  }//*/
  delete[] normal;
  normal=NULL;

  //Normalise every normal
  for(i=0;i<nVertex;i++) {
    VectorNormalise(&normal2[i*3],&normal2[i*3]);
  }//*/

  //Draw all the info
  float colour[4];
  glGetFloatv(GL_CURRENT_COLOR,colour);

  glBegin(GL_POLYGON);
  for(i=0;i<size;i++) {
    if(vi[i]>=0) {
      if(glIsEnabled(GL_LIGHTING)==GL_FALSE) glColor4fv(colour);
      glNormal3fv(&normal2[vi[i]*3]);
      glVertex3fv(&v[vi[i]*3]);
    }
    else if(vi[i]==-1) {
      glEnd();
      if(i<size-1) {
        glBegin(GL_POLYGON);
      }
    }
  }
  glEnd();//*/

  delete[] normal2;
  normal2=NULL;
}

void GlPolygonVertexNormal(float *v,int *vi,int size,float *n,int
*ni) {
```

```
//Note: The size of normal should be same as size of vertex.

//Draw all the info
float colour[4];
glGetFloatv(GL_CURRENT_COLOR,colour);

glBegin(GL_POLYGON);
int i;
for(i=0;i<size;i++) {
   if(vi[i]>=0) {
      if(glIsEnabled(GL_LIGHTING)==GL_FALSE) glColor4fv(colour);
      glNormal3fv(&n[ni[i]*3]);
      glVertex3fv(&v[vi[i]*3]);
   }
   else if(vi[i]==-1) {
      glEnd();
      if(i<size-1) {
         glBegin(GL_POLYGON);
      }
   }
}
glEnd();
}

void GlPolygonTexture(float* v,int* vi,int size,float* t,int*
ti,float ratioU,float ratioV) {

   //Calculate number of face
   int nFace=0;
   int i;
   for(i=0;i<size;i++) {
      if(vi[i]<0) nFace++;
   }

   //For each face, calculate its normal
   float *normal=new float[nFace*3]; //Because of x,y,z

VectorNormal2(&v[vi[0]*3],&v[vi[1]*3],&v[vi[2]*3],&normal[0]); //
First normal
   int j=1;
   for(i=0;i<size;i++) {
      if(vi[i]==-1 && i<size-1) {


VectorNormal2(&v[vi[i+1]*3],&v[vi[i+2]*3],&v[vi[i+3]*3],&normal[j
*3]);
         j++; //Next face
      }
   }


   //Calculate number of vertices
   int nVertex=0;
   for(i=0;i<size;i++) {
      if(vi[i]>nVertex)
         nVertex=vi[i];
```

```
  }
  nVertex++; //Include 0

  //Calculate normal for each vertex
  float *normal2=new float[nVertex*3]; //Include 0
  memset(normal2,0,nVertex*3*sizeof(float)); //Set every normal
to 0
  j=0; //For face
  for(i=0;i<size;i++) {
    if(vi[i]>=0) {
      VectorAdd(&normal2[vi[i]*3], //So that, the normal2 and v
are in same index
        &normal[j*3], //The normal of the face
        &normal2[vi[i]*3]); //Output
    }
    else if(vi[i]==-1) {
      j++;
    }
}//*/
delete[] normal;
normal=NULL;

//Normalise every normal
for(i=0;i<nVertex;i++) {
  VectorNormalise(&normal2[i*3],&normal2[i*3]);
}//*/

//Draw all the info
float colour[4];
glGetFloatv(GL_CURRENT_COLOR,colour);

float texcoord[2];

glBegin(GL_POLYGON);
for(i=0;i<size;i++) {
  if(vi[i]>=0) {
    texcoord[0]=t[ ti[i]*2 ] * ratioU; //This is because, the
texture might be used with glTexSubImage2D()
    texcoord[1]=t[ (ti[i]*2+1) ] * ratioV;

    glTexCoord2fv(texcoord);
    glNormal3fv(&normal2[vi[i]*3]);
    glVertex3fv(&v[vi[i]*3]);
  }
  else if(vi[i]==-1) {
    glEnd();
    if(i<size-1) {
      glBegin(GL_POLYGON);
    }
  }
}
glEnd();

delete[] normal2;
normal2=NULL;
```

```
}

/////////OpenGL animation

class CTransform {
public:
  CTransform();
  CTransform(const CTransform& transform);
  ~CTransform();

  void Reset();

  void GlTransformAll();

  float translation[3];
  float rotation[4];
  float scale[3];
  float scaleOrientation[4];

  float* pPosKey; //PositionInterpolator key
  int nPosKeySize;

  float* pPosKeyValue; //PositionInterpolator keyValue
  int nPosKeyValueSize;

  float* pOriKey; //OrientationInterpolator key
  int nOriKeySize;

  float* pOriKeyValue;
  int nOriKeyValueSize;

  float* pScaKey; //PositionInterpolator for set_scale event
  int nScaKey;

  float* pScaKeyValue;
  int nScaKeyValue;

  float* pScaOriKey; //OrientationInterpolator key for
set_scaleOrientation event
  int nScaOriKey;

  float* pScaOriKeyValue;
  int nScaOriKeyValue;

  //To adjust the speed
  int nUnitMax; //Unit is the "unit" frame between the frame
  int nTimeMax; //Time is the calculation for the whole
animation.

private:
  int m_posFrame1;
  int m_posFrame2;
  int m_posCurrInter;

  int m_oriFrame1;
```

```
  int m_oriFrame2;
  int m_oriCurrInter;

  int m_currTime;

  int m_nUnitStep;
  int m_nTimeStep;
};


void CTransform::GlTransformAll() {
  int row;
  float* pInter;

  //Position
  if(m_posCurrInter>nUnitMax) {
    m_posCurrInter=0;
    m_posFrame1++;
    m_posFrame2++;
  }
  if(m_posFrame1==nPosKeySize-1) {
    m_posFrame1=0;
    m_posFrame2=1;
  }

  //Orientation
  if(m_oriCurrInter>nUnitMax) {
    m_oriCurrInter=0;
    m_oriFrame1++;
    m_oriFrame2++;
  }
  if(m_oriFrame1==nOriKeySize-1) {
    m_oriFrame1=0;
    m_oriFrame2=1;
  }

  //Position
  if(nPosKeySize>0) {
    row=nPosKeyValueSize/nPosKeySize;
    pInter=new float[3];
    CalcInterpolate(&pPosKeyValue[m_posFrame1*row],
      &pPosKeyValue[m_posFrame2*row],
      3, //Because only need 3
      m_posCurrInter,
      nUnitMax,
      pInter);
    glTranslatef(pInter[0],pInter[1],pInter[2]);
    delete[] pInter;
    m_posCurrInter+=m_nUnitStep;
  }
  else
    glTranslatef(translation[0],translation[1],translation[2]);

  //Orientation
  if(nOriKeySize>0) {
```

```
    row=nOriKeyValueSize/nOriKeySize;
    pInter=new float[4];
    CalcInterpolate(&pOriKeyValue[m_oriFrame1*row],
      &pOriKeyValue[m_oriFrame2*row],
      4,
      m_oriCurrInter,
      nUnitMax,
      pInter);

    //Calculate the degree
    float deg=Rad2Deg(pInter[3]);
    glRotatef(deg,pInter[0],pInter[1],pInter[2]);
    delete[] pInter;
    m_oriCurrInter+=m_nUnitStep;
  }
  else
    glRotatef(rotation[0],rotation[1],rotation[2],rotation[3]);


glRotatef(scaleOrientation[0],scaleOrientation[1],scaleOrientatio
n[2],scaleOrientation[3]);
  glScalef(scale[0],scale[1],scale[2]);
  glRotatef(-
scaleOrientation[0],scaleOrientation[1],scaleOrientation[2],scale
Orientation[3]);
}

//////////////OpenGL texture
class CTexture {
public:
  CTexture();
  char szFile[256];
  void TextureInit();
  void TextureDestroy();

  unsigned int texId[1]; //For OpenGL
};

CTexture::CTexture() {
  this->Reset();
  texId[0]=-1;//Initialise
  memset(szFile,0,256);
}

void CTexture::TextureInit() {
  LoadPicture(szFile);
  Resize(256,256);
  glGenTextures(1,texId);

  glPixelStorei(GL_UNPACK_ALIGNMENT,1);
  glBindTexture(GL_TEXTURE_2D,texId[0]);


glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,256,256,0,GL_RGB,GL_UNSIGNED_
BYTE,this->pData);
```

```
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
}

void CTexture::TextureDestroy() {
  glDeleteTextures(1,texId);
  this->Reset();
}

//////////Finger tracking
void skinSegmentation(unsigned char* imageData,int width,int
height) {
  for(int i=0;i<width * height *3;i+=3) {
    int r,g,b;
    r = imageData[i];
    g = imageData[i+1];
    b = imageData[i+2];

    int max = r;
    if(g>max)
      max = g;
    if(b>max)
      max = b;

    int min = r;
    if(g<min)
      min = g;
    if(b<min)
      min = b;

    if(r > 95 && g > 40 && b > 20 &&
      (max - min > 15) &&
      (abs(r - g) > 15) &&
      r > g && r > b)
    {
    }
    else {
      imageData[i] = imageData[i+1] = imageData[i+2] = 0;
    }
  }
}

void fingerTracking() {
  g_show->GrabBitmap();
  if(!g_show->cbBuffer) return;

  //To IplImage
  g_img = cvCreateImage(cvSize(g_show->width,g_show-
>height),IPL_DEPTH_8U,3);
  g_img->origin=1;
  memcpy(g_img->imageData,g_show->pData,g_show->cbBuffer);


  //For skin segmentation
```

```
  IplImage* finger = cvCloneImage(g_img);
  cvCvtColor(finger,finger,CV_BGR2RGB);
  skinSegmentation((unsigned char*)finger->imageData,finger-
>width,finger->height);
  cvCvtColor(finger,finger,CV_RGB2BGR);


  IplImage* result = cvCreateImage(
    cvSize(finger->width - g_finger->width + 1,finger->height -
g_finger->height + 1),
    IPL_DEPTH_32F,1);
  result->origin = 1;

  //Template matching
  cvMatchTemplate(finger,g_finger,result,CV_TM_CCORR_NORMED);
  double minval,maxval;
  CvPoint minloc,maxloc;
  cvMinMaxLoc(result,&minval,&maxval,&minloc,&maxloc);

  if(maxval > 0.6) {
    cvCircle(result,maxloc,1,cvScalar(0,255,0),5);
    maxloc.x += (float)g_finger->width/2;
    maxloc.y += (float)g_finger->height/2;

    cvCircle(g_img,maxloc,1,cvScalar(0,255,0),5);
  }

  cvReleaseImage(&finger);
  cvReleaseImage(&result);
}
```