Department of Computer Science and Engineering

The Chinese University of Hong Kong

2006-2007 Final Year Project Report (2nd Term)

A Generic Real-Time Facial Expression Modelling System

Group: LYU0603

Supervised by

Prof. Michael R. Lyu

Prepared by:

Cheung Ka Shun (05521661)

Table of Contents

Chapter 1: Introduction

1.1 Background Information

Image and Video based face coordinate recognition system is a system which tries to recognize human face coordinates such as eyes and nose using a gallery or a video (the video can be live recording if the computation speed is high enough) instead of using any mechanical apparatus such as Infrared sensor. In the other words, it is a computation-driven technology for automatically identifying a human face position from an image or a video. In contrast, infrared sensor based application can only recognize the position of human limbs and head (commonly applied in game industry) but cannot precisely recognize the facial expressions due to colour missing. The major mechanism behind image and video based face coordinate recognition system is trying to compare between the face database and the collected picture from image source (a frame in video perspective) so as to figure out the matched pattern between them.

There exists several kinds of application for image and video based face coordinate recognition system. Police force used face recognition technology to search for potential criminals and terrorists who attended the event. Another typical application is an accessibility tool which helps the disabled person. The application recognizes the eye-ball movement such that disabled person can control the computer by the movement of eyeballs or head.

1.2 Project Motivation

Nowadays, the utilization of the face recognition technology becomes more common and widely used in different area of information technology and entertainment industry. The applied level also spread from business level to user-level. The major reason is that the processing power of the electronic devices is much more improved than few years ago, and the cost of such devices reduced. End-user requires more interesting and attractive effect than before. The following are two examples which applied face recognition system.

Canon PowerShot SD900 includes face detect system feature which can automatically focus (AF) on human face. It facilitates user on focusing.

Fig.1.1 Canon PowerShot SD900

Logitech QuickCam Pro5000 supports face detection and cartoon simulation.

Fig.1.2 Logitech QuickCam Pro 5000 and its video effect

Video source (e.g. from webcam) was commonly used in net-meeting software such as Windows Live Messenger. This kind of applications is able to browse the real time video capture source from who is net-meeting with. To enrich the functionality and entertainment of such kind of applications, we decided to make a more interesting and innovative video output can be presented to the other side.

High computation power computer system has become more available to end-user. The resolution, frame rate and video quality (image with less noise) supported by webcam have improved in recent years. As the limitation caused by the hardware equipment has reduced, we would like to build an application which is able to detect human facial expression and to model the detected expressions into a 3-D model.

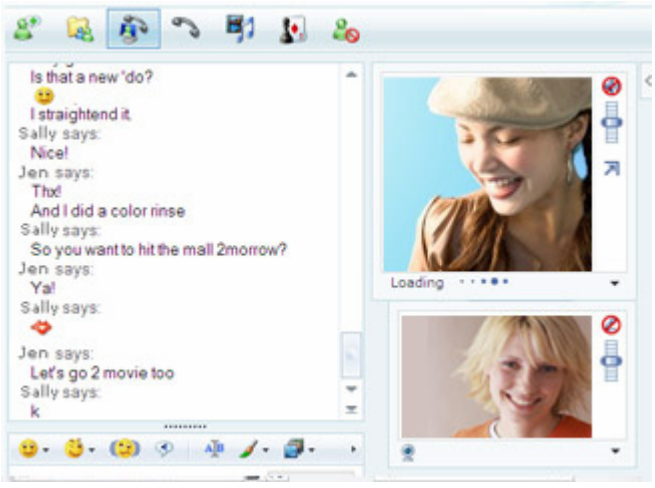

Fig.1.3 Using net-meeting features in Windows Live Messenger

Windows Live Messenger has become more popular than previous year, and there are many ICQ users migrating to Windows Live Messenger. The major reason is that Windows Live Messenger provides many interesting emotion pictures in its chatting environment. Windows Live Messenger also allows user to import self-defined emotion pictures so as to enrich the chatting environment. It is obviously that users are interested in some luxuriant features on top of basic functionality given by the software.

We think that the net-meeting can be further enhanced, like the market selling point in Windows Live Messenger chatting environment. Our enhancement is to provide a more interesting video output in net-meeting.

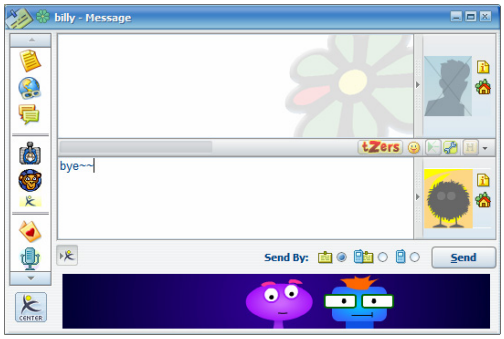

Fig.1.4. ICQ chatting environment (only supports pre-defined emotion pictures)

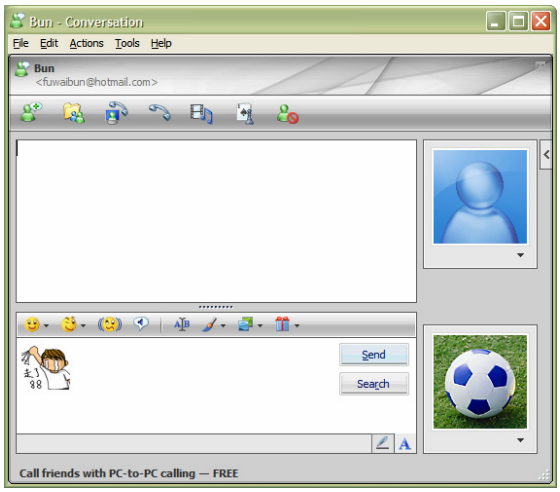

Fig.1.5. Windows Live Messenger chatting environment which supports self-defined emotion pictures.

As we have successfully built a virtual camera for embedding our facial expression modelling system into net-meeting engine in the beginning of semester two, we want to further utilize and explore the potential in the face recognition technology to construct another application.

In game industry, most games are developed under 3D graphics environment. In some 3D games, the player is role-playing a character but with a pre-defined human face. We would like to build an application that allows users to use their face in game character.

There exist some 3D games tried to capture the human face as face texture in 3D character but the result is not attractive as the developer ignored the face organs position in order to fit on the skeleton of the 3D model. With the help of facial detection technology in our system, we are willing to extract the human face in automatic manner. The face extraction can work precisely with minimal help of human fine adjustment.

1.3 Project Objective

The primary goal of Generic Real-Time Facial Expression Modelling System (GRTFEMS) is to discover the potential that a face detection system can perform. The first part of our system aimed to enrich and enhance the functionality of video source from a web-cam or video clip so that end-user can finally enjoy an interesting and dynamically generating video effect on net-meeting applications or computer games. The second part of our system is targeted on facilitating game developer to extract human face from game player and consequently use such human face as a texture in 3D model.

Due to the fact that this solution is implemented in software based approach, users are not required to pay extra cost on specific equipment. In other words, the application is able to work on different kinds of hardware as to achieve our generic goal.

GRTFEMS is composed of two parts. Virtual camera is the first part which can be embedded into current net-meeting software. When the user is net-meeting with his or her friends, he or she can use a virtual character to replace and display to the other side instead of displaying the realistic environment.

The second part of GRTFEMS is a 3D face model generator. The generator can extract a human face and approximate face shape to create a 3D model. The result model is saved in a standardized format such that game developer can make use of it as a character model.

1.4 Project Scope

This part describes our project scope. It defines what the project is going to be developed and to which extent. It is useful for scheduling our tasks so that we can work in a more efficient way.

This project is divided into three major parts. The first part is human facial expressions detection. It detects several varieties of facial expressions such as eyes blinking, mouth opening and head movement.

The second part of the project is facial expressions modelling. After collecting the facial information and analyzing the information in part one, we can model a 2-D or 3-D character which representing the actual movement of the human.

The last part is 3D face model generation. The face model generated should extract user face as a texture and approximate the shape of the face by the user. Other 3D environment such as gaming environment should be able to adopt our 3D model.

1.5 Project Equipment

In our project, there are two dedicated PCs for us to develop the application. The major configurations of our PCs and peripheral are listed as following:

CPU	Intel P4 3.0Ghz Processor
Motherboard	ASUS P5LD2
Memory Size	1.0GB
Display card	NVIDIA GeForce 6600
Video Capture Device	(i) Logitech Webcam Pro4000 (ii) Panasonic NV-GS180 DV Camera (2.3M Pixel, 3CCD, FireWire compatible)

We test our system on both video capture devices. Panasonic DV Camera usually has a better performance than Logitech Webcam Pro4000.

On the other hand, we use KIOSK as our testing environment as the lighting condition is balanced.

Fig.1.6. KIOSK, which has a balanced lighting environment

Chapter 2: Development Environment

2.1 Development Platform and Tools

This part introduces the programming platform we used to develop GRTFEMS. They are important because it also affect the running platform of our finalized system.

There is an important tool we have used in our project, which is GraphEdit. It is a visual tool for building DirectShow filter graph (a graph which comprised of DirectShow filters and connections). With the help of GraphEdit, we are able to experiment with filter graph we built in a convenient way because it provides some fundamental features like start or pause and graph visualization.

2.1.1 Microsoft Windows

Our system is developed under Microsoft Windows XP Professional Edition. Microsoft DirectX is our major programming APIs and it is only available in Windows environment. Microsoft Windows is the most widely used operating system, it can help us to achieve that our product is capable to be ran on most computers.

2.1.2 Microsoft Visual C++

Microsoft Visual C++ is an Integrated Development Environment (IDE) for programming language engineered by Microsoft. It provides many powerful tools and features for program development such as keywords colouring and IntelliSense (a form for automated completion feature). The debugger embedded is easy-to-use and powerful.

As programming language C++ is more compatible with Microsoft DirectShow and Direct3D than Visual Basic because Visual Basic can only access a small subset of DirectShow API. Therefore, the IDE we chosen must support C++. It is the reason why we chose Microsoft Visual C++ as our programming platform.

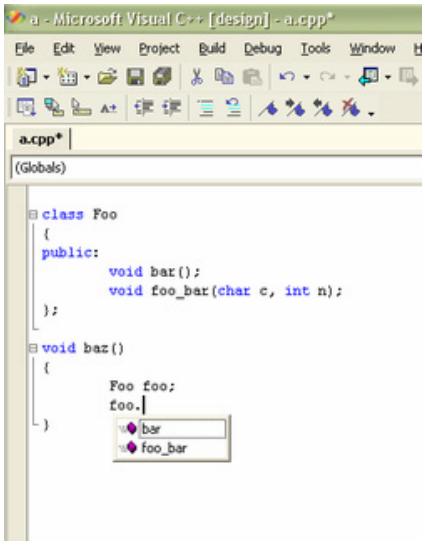

Fig.2.1 The above figure is an illustration of IntelliSense

2.1.3 GraphEdit

GraphEdit can visually build and run Microsoft DirectShow filters graph without any programming involved. It facilitates us in testing the filters connections and debugging. All filters registered are enumerated by GraphEdit, we can add the filters we want to the filter graph and connect them by simply drag and drop. GraphEdit also allow us to play, pause and stop the filter graph through the interface it provides. It eases the process of building a new application for graph building and testing.

Figure 2.2 shows a filter graph displayed visually. The filter closer to the left is upstream.

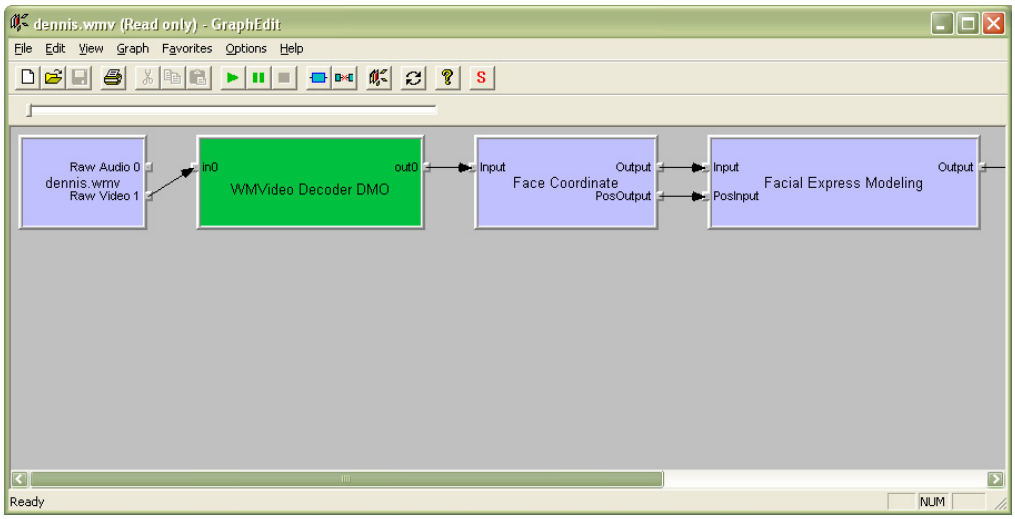

Fig.2.2 User interface of GraphEdit

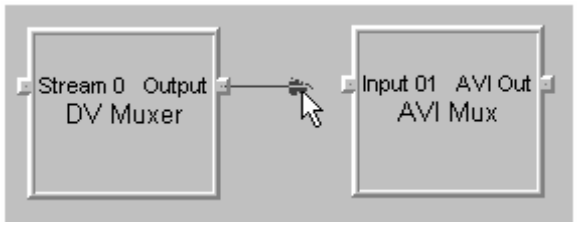

Fig.2.3 To connect the filters, we only have to drag an arrow which represents a stream between two filters

After we built a filter graph by GraphEdit, we can start to play the streaming of the filter graph by simply clicking on the play button. In figure 2.4 shows playing a movie clip (in mpg format) using filter graph.

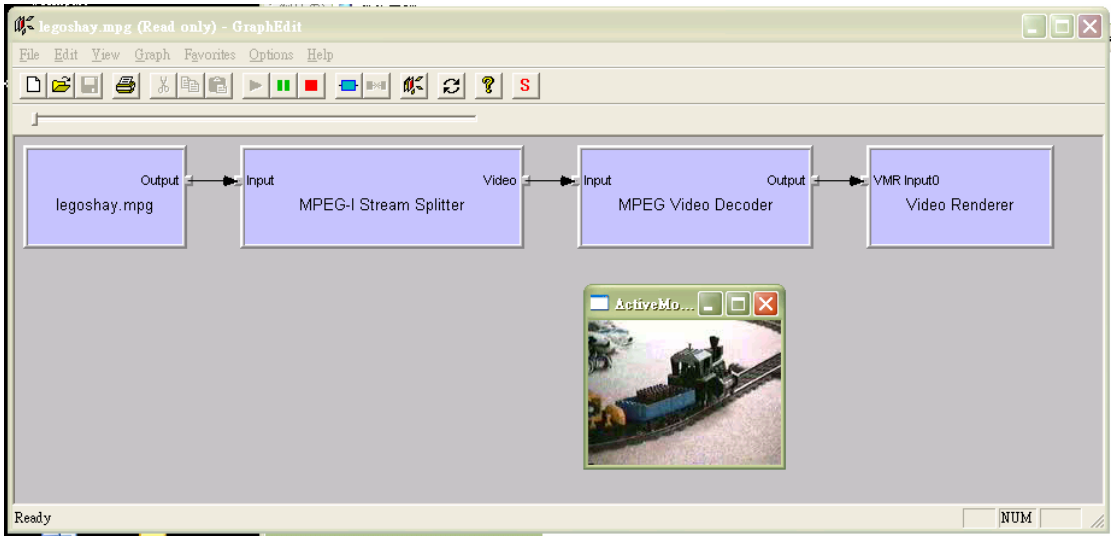

Fig.2.4 Playing a movie clip (.mpg) using GraphEdit*

Actually, GraphEdit is invoking the functions Run, Pause and Stop in IMediaControl interface to perform Playing, Pausing and Stopping of the streaming.

2.2 Programming API

In our project, we adopted C++ as our programming language, Microsoft Visual C++ as our programming environment with Microsoft DirectShow and Microsoft Direct3D as the programming API. Due to the fact that they are all developed by Microsoft Corporation, there is no compatibility problem between them

The following sections will further introduce the features of the programming platforms and state the reasons why we adopted them in our system. It also shows our study and learning on them.

2.2.1 Microsoft DirectShow

Microsoft DirectShow is an architecture which is applied for streaming media, see figure 2.5. It provides a mechanism for video source capture and then playback as a multimedia streams. It supports wide variety of media formats. Different video capture devices such as webcam and DV camera have their own video output format. With the help of DirectShow filters, we can use different video source in our project without modifying the code as to adapt the type compatibility. For example, it can support Audio-Video Interleaved (AVI) and Windows Media Video (WMV).

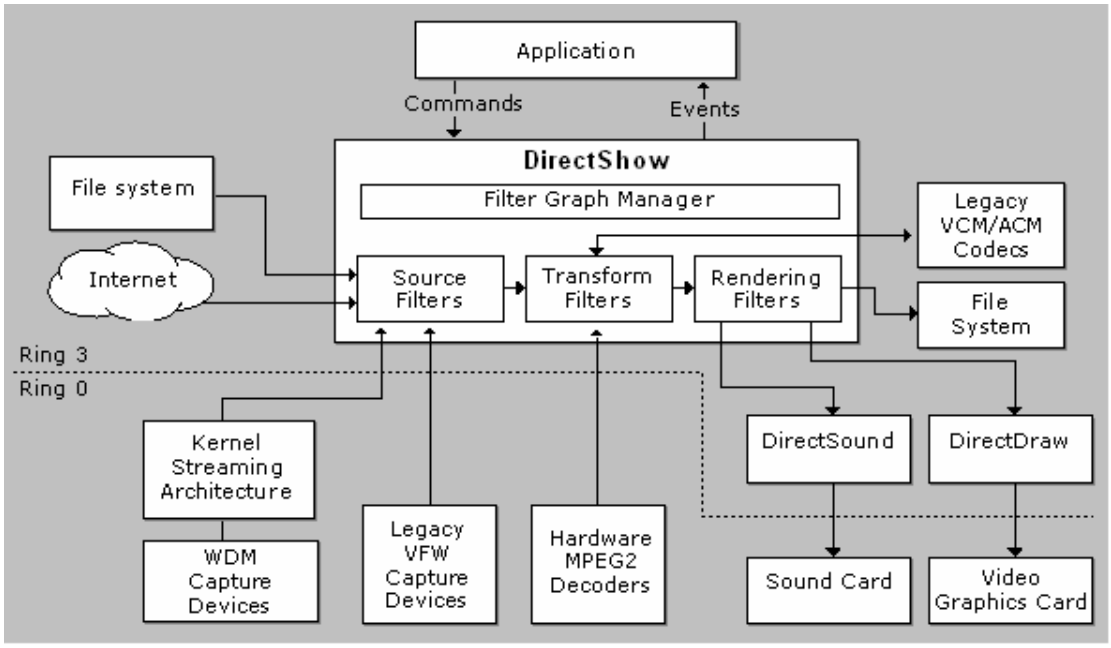

Fig.2.5. Architecture of DirectShow

DirectShow allows us to access underlying stream control architecture for applications that require custom solutions. It simplifies the task of creating digital media applications by isolating applications from the complexities of data transports, hardware differences, and synchronization. The good modularizing model helps developer has a better management on the system and separation of concerns.

Since Microsoft DirectShow uses modular architecture, the data streaming processing is done by COM (Component Object Model) object, the COM responsible for data streaming is called filter. There are some standard filters already provided by DirectShow such as AVI Splitter.

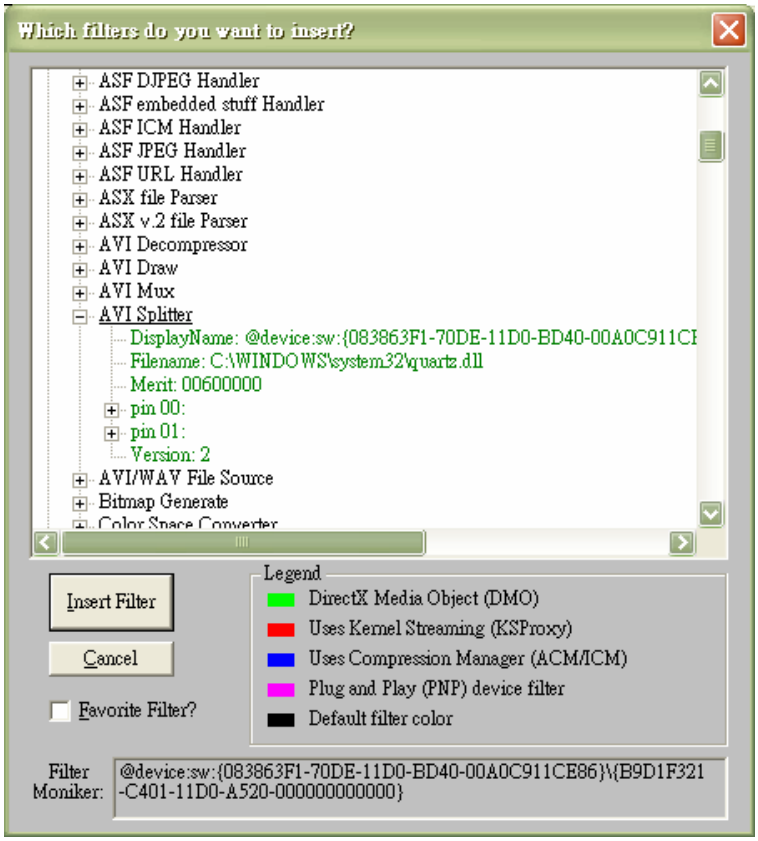

Fig.2.6. There are many standard DirectShow filters provided for streaming processing

DirectShow filter represents a stage in date processing. Each filter has a number of input and output pins which is an interface to connect with other filter. The design of such connection mechanism make the filters can be connected in different ways for different tasks to build a filter graph, as shown in figure 2.7. Developers can add custom effects or other filters at any stage in the graph, then render the file, URL or camera.

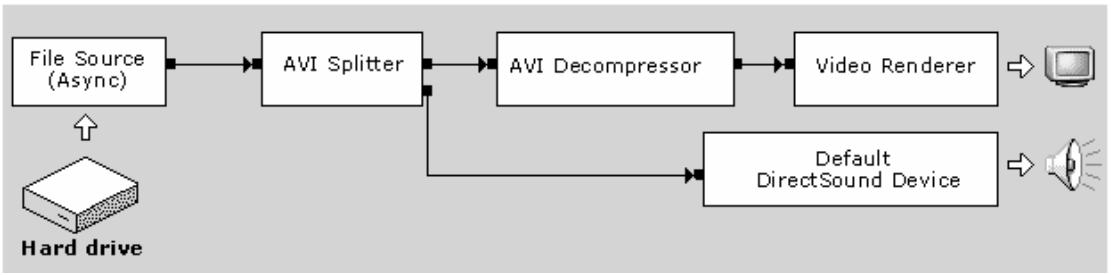

Fig.2.7 An example for DirectShow filter graph

In a DirectShow application, it is required that the application has to manage the data flow by itself. There is a COM object, Filter Graph Manager which is a high-level API for controlling data stream. The relationship between the application, filter graph manager and filter graph are illustrated in figure 2.8.

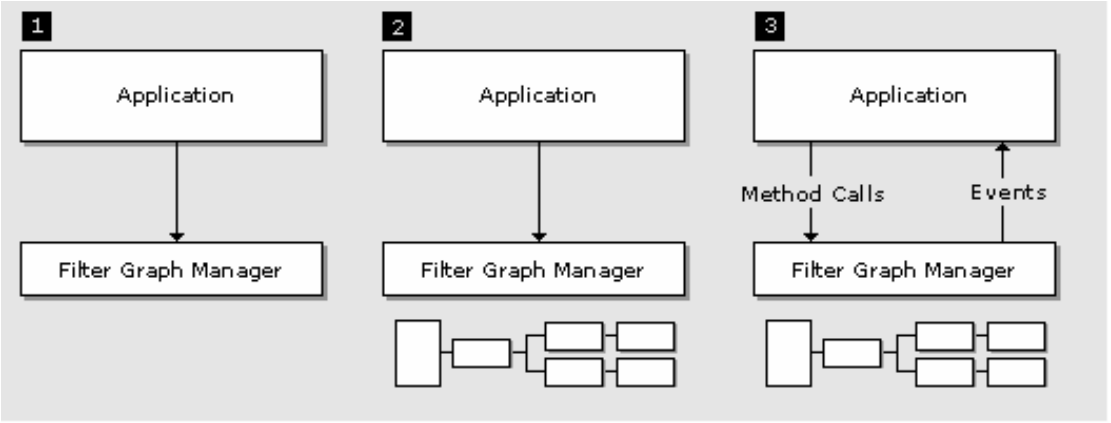

Fig.2.8 Relationship of application, filter graph manager and filter graph

There are totally five types of DirectShow filters. They are source filter, transform filter, renderer filter, splitter filter and mux filter.

The filters we implemented are source filter and transform filter. A source filter introduces data into the graph. The data might come from a file, a network, a camera, or anywhere else. Each source filter handles a different type of data source. A transform filter takes an input stream, processes the data, and creates an output stream. Encoders and decoders are examples of transform filters.

Our project is trying to build a transform filter in facial expression modelling and 3D face modelling. A source filter was built in net-meeting software embedment. Transform filter is an intermediate component in a video stream. It takes an input stream from the upstream filter, and then processes the data. After that, the filter will create an output dream to the downstream filter.

In Generic Real-Time Facial Expression Modelling System, the filter we built which is responsible for facial expression analysis is named as Facial Express Modelling filter. The functionality of this filter is to analyze the human facial expression in a frame of the video stream. After the facial expression result is obtained, it will initialize a Direct3D device which is responsible for drawing appropriate picture representing human expressions.

Another filter we built for embedding our 3D model result from Facial Expression Modelling filter into net-meeting application is Virtual Camera filter. Virtual Camera filter is a source filter that builds an inner filter graph

contains Facial Express Modelling filter and redirect the output to this source filter.

In the 3D face model generation part, we built a transform filter, Face Generator filter which derivates from Facial Expression Modelling filter. This filter is trying to extract the face from a video frame and prepare a texture which will be used in face rendering.

Due to the fact that DirectShow is an excellent video streaming processing API, and it provides many useful functional APIs for video processing, we adopted it in our project.

2.2.2 Microsoft Direct3D

Microsoft Direct3D is a part of DirectX API and only be available in Microsoft Windows platform. Direct3D provides an API to developers so that they can render three dimensional graphics using graphical acceleration device if it is available.

Microsoft Direct3D and Microsoft DirectShow are also under Microsoft DirectX software development kit. The programming techniques required are similar. One of the examples is that, the DirectX developer should have basic knowledge on COM object. If we are familiar with using and writing a COM object, it will be beneficial on programming with both APIs.

In Direct3D, we must be familiar with two systems which are featured in Direct3D in order to draw or model object in the Direct3D world. They are 3-D coordinate system and texture coordinate system. We will have more details later on.

2.2.3 OpenGL

OpenGL (Open Graphics Library) is a cross-platform API for developing 2D and 3D graphics application. Its graphics rendering API provides a high-quality colour images composed of geometric and image, and is making use of hardware acceleration to generate real-time 3D effects possible.

OpenGL produces similar operation to Direct3D like drawing different primitive and mapping texture to the primitive. It tries to hide the different capabilities among different hardware platforms by requiring the hardware company support the full OpenGL feature set.

OpenGL is in fact a low-level programming which requires the programmer stating exactly each step to render a scene. It has no high-level command to describe complex object. It also has no commands for performing windowing events like mouse click or key state.

Chapter 3: Implementation Overview

Our project implementation can be divided into two phases. The first phase is to analysis the human facial expression and modelling it with a cartoon character. The goal is to use the cartoon character to represent the user facial expression in net-meeting like MSN. The second phase is to generate a real human face model in 3D space. The face model data would be finally standardised for other client application to use.

Both phases have the same input data that is a live video stream and need to compute the position of the human face on the video in a real time manner. It is done by a Face Coordinate Filter which will be explained in the next chapter. Since the output from this filter is a critical data in doing the analysis, an assumption is made here such that the face tracking is usually working accurately in our system.

Facial Expression Analysis and Modelling

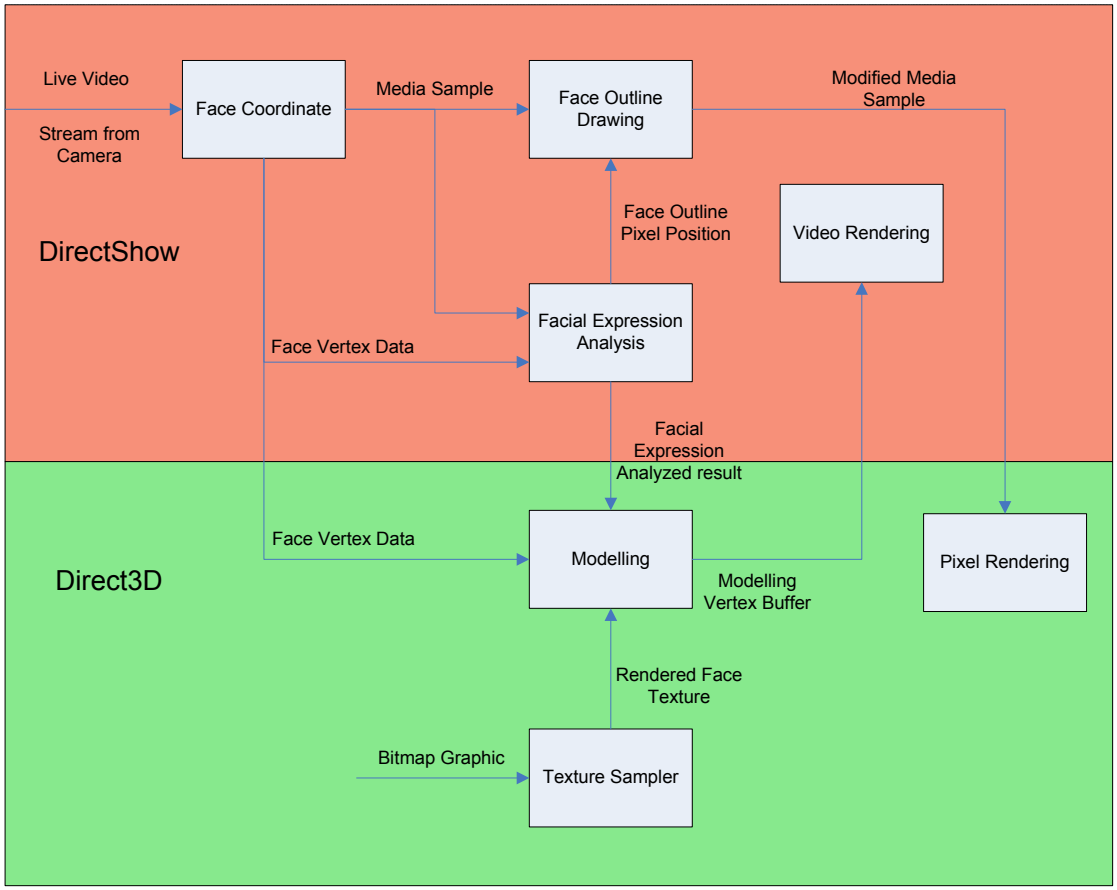

Fig.3.1 Overview of modules in facial expression analysis system

There are number of modules implemented for identifying the facial expression and perform modelling on the facial expression. "Face Outline Drawing" would modify the video frame to show the detected face area for user reference. "Facial Expression Analysis" is a key component in this system. It analysis the facial expression based on the result from the "Face Coordinate". Those analyzed result would be used by "Modelling" to decide which expression should be generated and select the right cartoon texture which is prepared by "Texture Sampler".

3D Face Generator

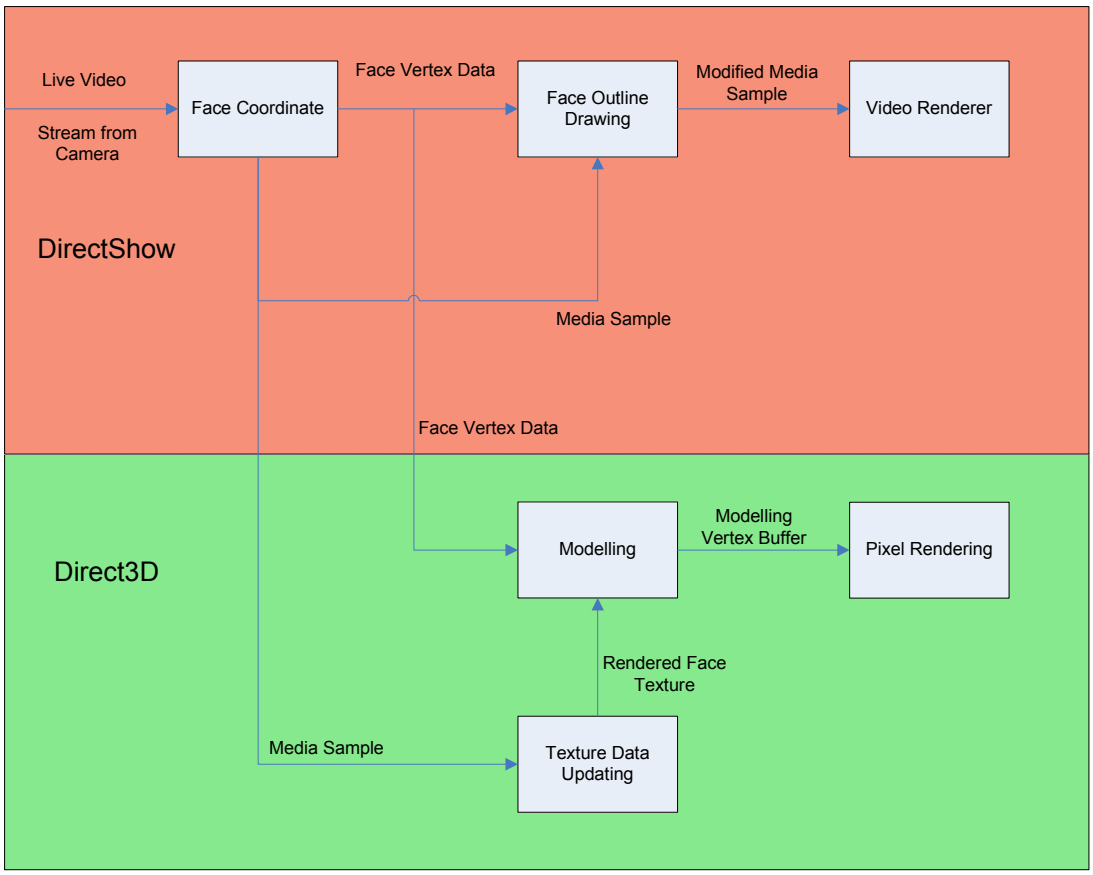

Fig.3.2 Overview of modules in face texture generator system

The above system is used for generating a human face texture dynamically. The DirectShow part is just the same as the facial expression analysis system but without the "Facial Expression Analysis". "Texture Data Updating" prepares a texture to store the whole video frame. It updates the texture frame by frame. This video frame texture would be used by "Modelling" to generate the desired face texture with the Effect Face Mesh which is defined specifically.

This system would be embedded into the 3D Face Generator to supply the human face texture. It copies the data into the common buffer so that

"FaceLab" can access the human face texture. "FaceLab" is responsible for approximating a human face by performing a statistical shape analysis. The generated face model would be rendered by OpenGL.

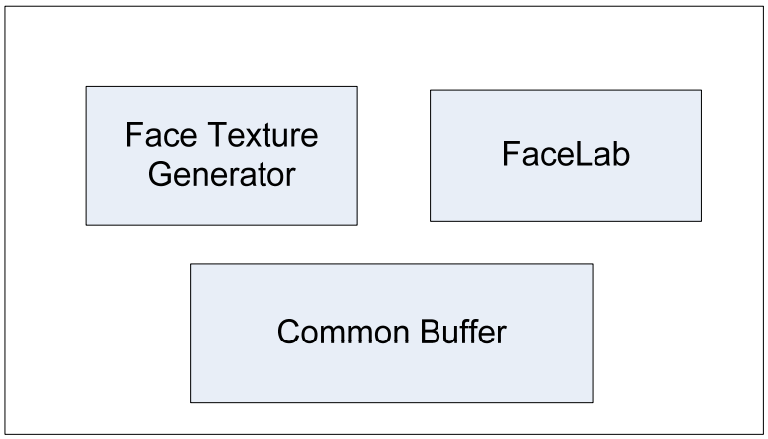

Fig.3.3 Overview of modules of 3D face generator

Finally, we would demo the resulted face model in a Direct3D application in which some simple face animation would be presented. This application also converts the specific face model data into a standardised file format.

Chapter 4: Face Coordinate Filter

FaceCoordinate filter is provided by VIEWLab, which is authorized to be adopted in our project. In our project, we have used FaceCoordinate filter as our programming interface. We had studied on FaceCoordinate filter so that we can utilize the output streamed out from this filter.

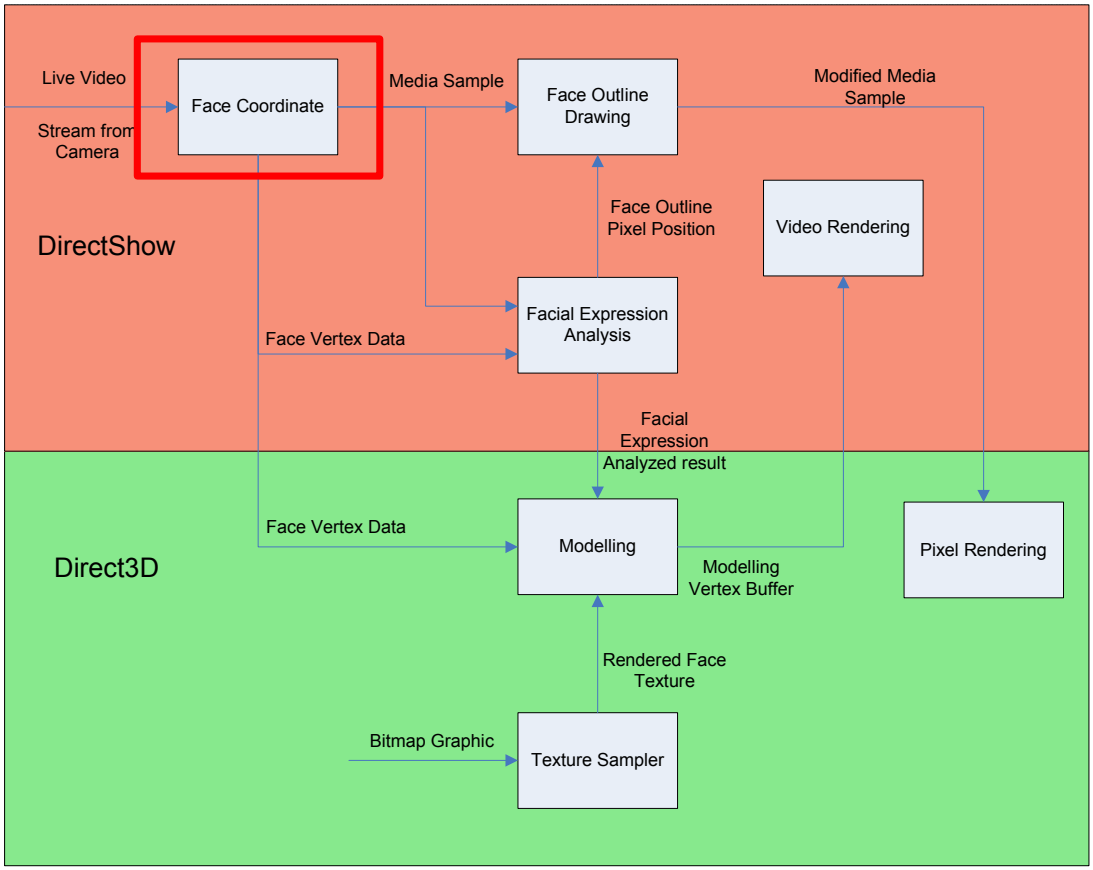

Fig.4.1 The relationship between Face Coordinate filter with our system

The major function of FaceCoordinate filter is to trace human face and then provide the coordinates of the face as an output. In this perspective, a face is being segmented with 100 vertices, which is shown in figure 4.2. A number (0-99) is given to each vertex which is an index for retrieving the particular vertex.

The detection of the face is done by using Active Appearance Models (AAMs) which can match statistical models of appearance to new images.

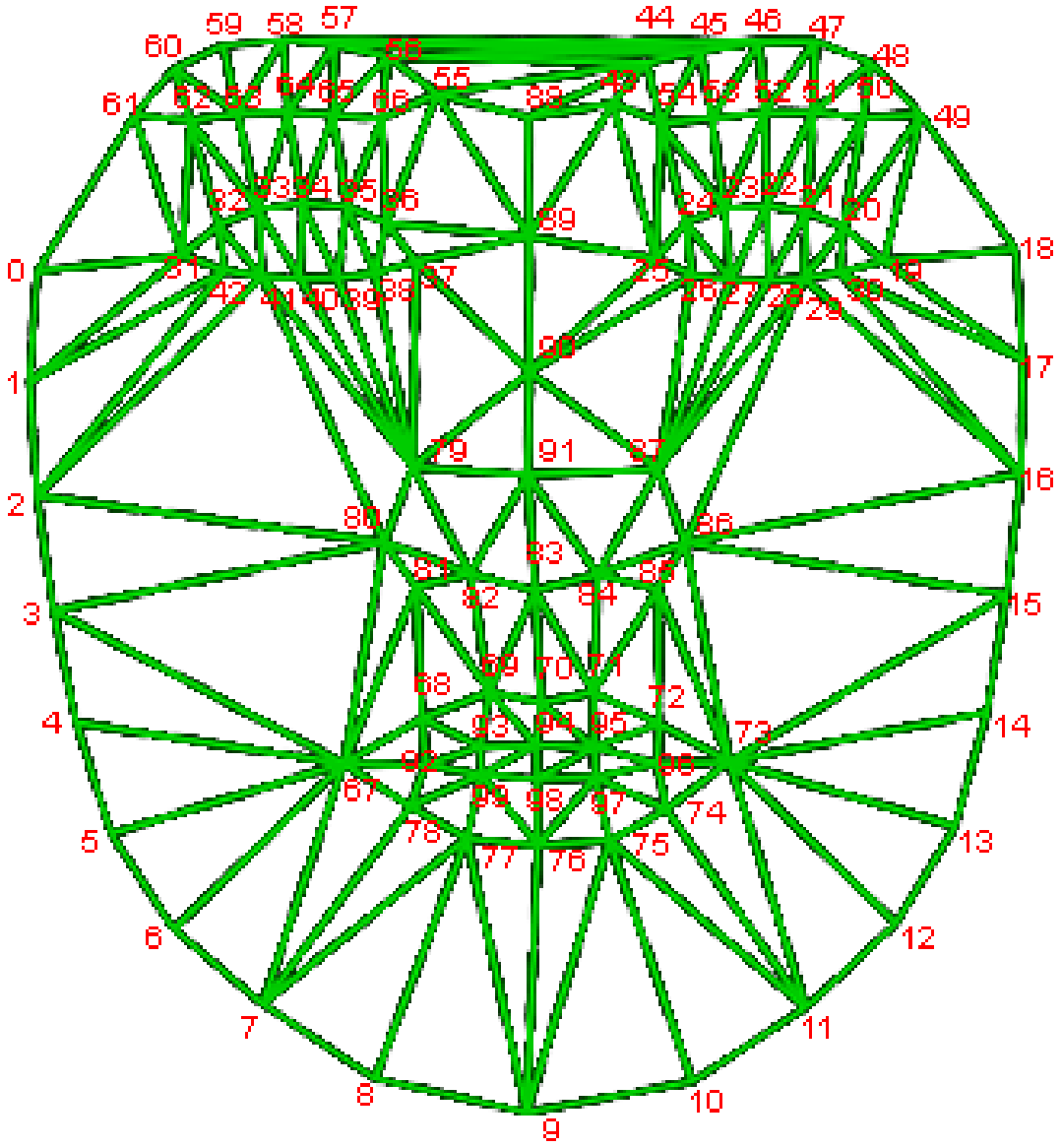

Fig.4.2. Human face mesh which was divided into 100 points and labelled with 0-99

Chapter 5: Face Outline Drawing

5.1 Introduction

In order to make a good prediction of the facial expression from the video source, we need to ensure the FaceCoordinate filter is working as accurate as possible. In order words, we want the face mesh is as close as possible to the human face area. Since our project concentration area is on the facial expression modelling, this module is not going to change the coding or algorithm presented in the FaceCoordinate filter. In this module, what we want to do is to add some indications in the video stream so that the user can manually adjust the face mesh fitting.

5.2 Basic concepts

In Figure 5.1, we can see that how it would look like when the face mesh is drawing above the human face. However, it is not easy to observe the changes in the face, especially both eyes which are small comparable to the whole face mesh. In addition, the vision effect is no good for the end user.

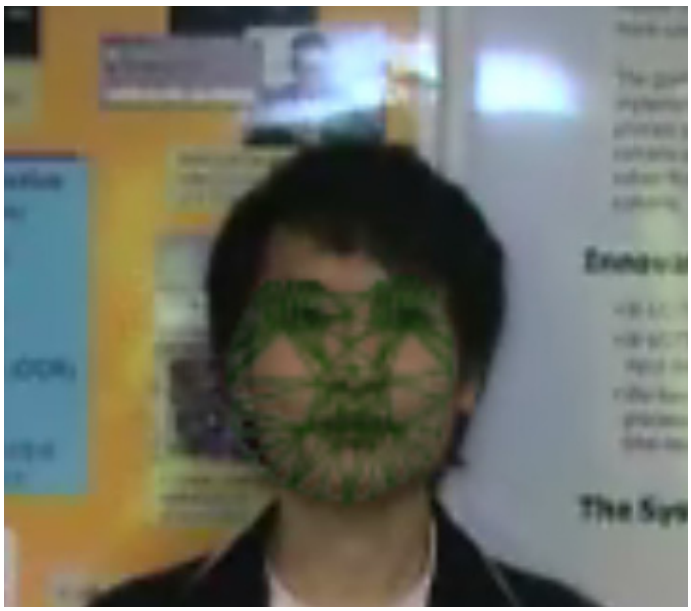

Fig.5.1 The face mesh fitting on the human face (eye opened).

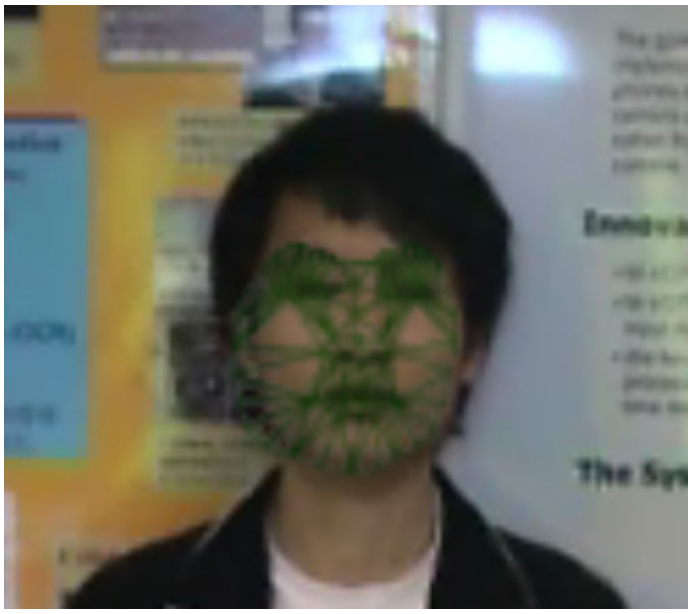

Fig.5.2 The face mesh fitting on the human face (eye closed).

We decide to make it simple to represent the face mesh by showing only the face mesh vertices and the result is as follow.

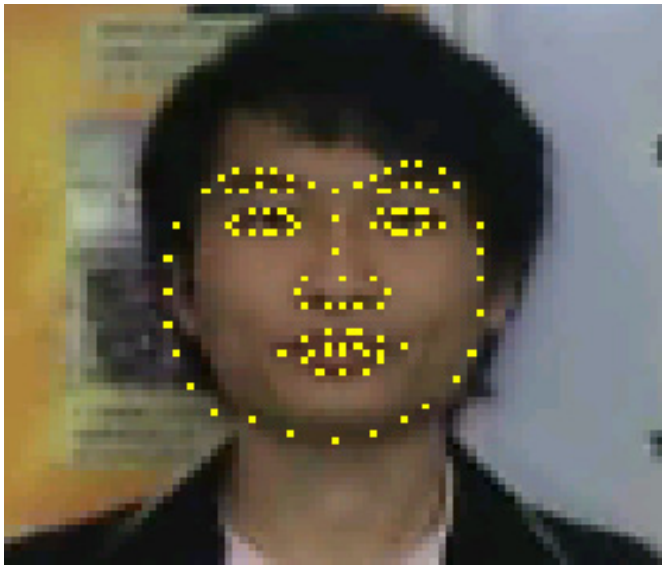

Fig.5.3 Shows all the face mesh vertices on the face.

5.3 Implementation

The first step in this module is to calibrate those face mesh coordinates to pixel coordinate. According to the implementation of the Face Coordinates Filter, the face detect system will provide a hundred points on the face area representing the different face components such as eye, mouth, nose, so and so forth. Those face mesh coordinates is represented in a 2D coordinate system with floating point number. It is, in fact, a 200 data array with the first 100 entries of x-coordinate followed by another 100 entries of y-coordinate. Performing calibration on those face mesh coordinates and re-structure the data format so that we can find out which pixel we would like to mark in the next step. The new x and y coordinate of a point after calibration is given by:

$$\forall iP(i,x) = \lfloor F(i) + 0.5 \rfloor$$
$$\forall iP(i,y) = \lfloor F(i+100) + 0.5 \rfloor \quad \text{where} \quad 0 \le i \le 100$$

We are then getting the frame buffer memory, which are simply arrays of bytes, from the media sample.

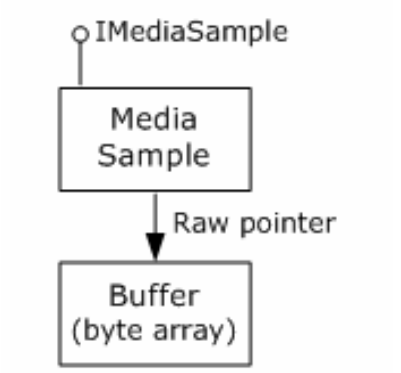

Fig.5.6 Retrieves the pointer to the buffer's memory

According to the DirectX SDK, the bytes placed in the frame buffer is in bottom-up orientation. It means that the buffer start with the bottom row of pixels, followed by the next row up, and so forth. The face mesh coordinate, however, is referring the top-left corner as (0, 0). We need to reverse count the buffer to find out the calibrated face mesh vertices and then alter the pixel value.

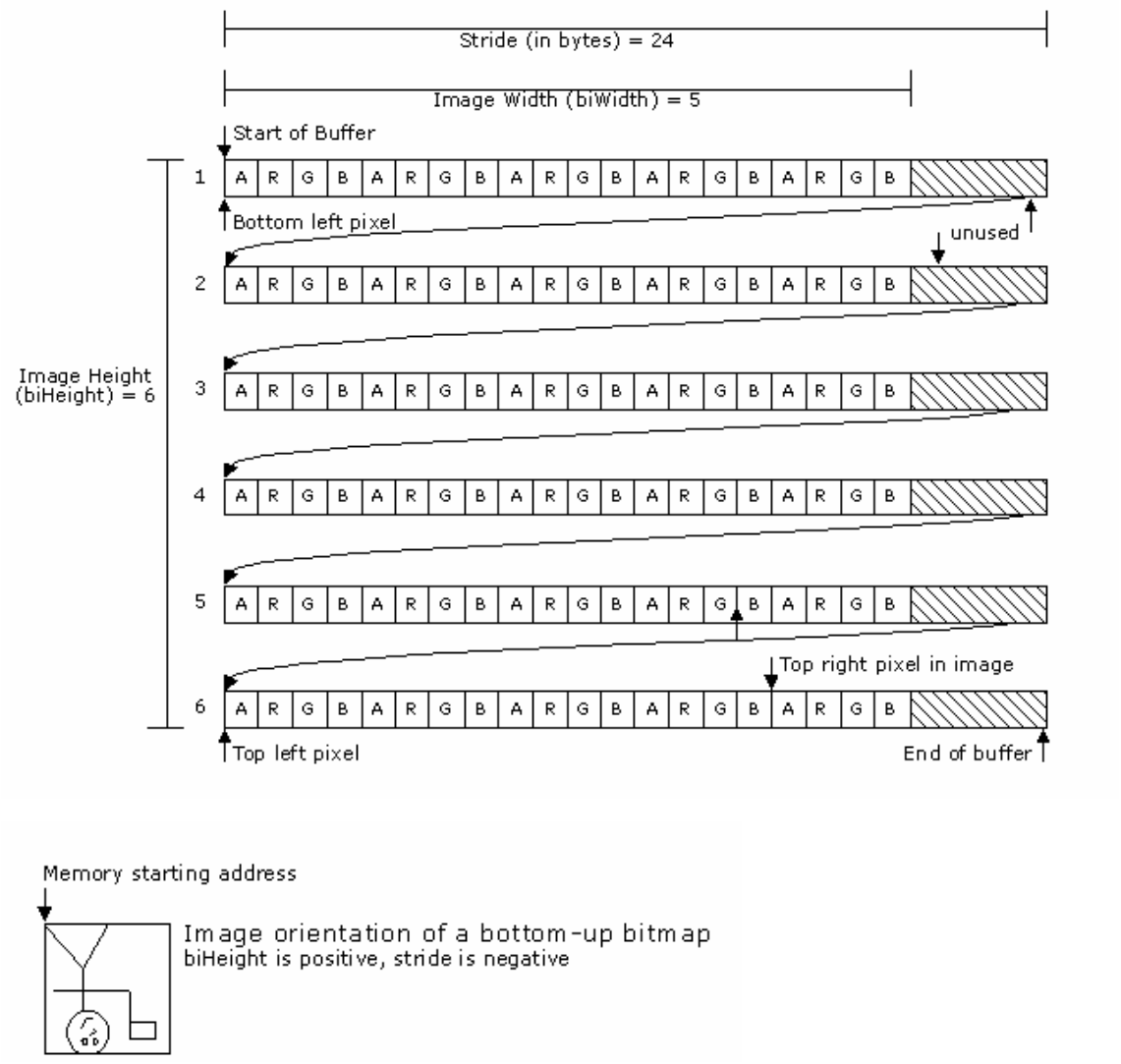

Fig.5.7 Physical layouts of bytes in a 5 x 6 bottom-up ARGB buffer

In our system, we choose yellow to mark out the point in the face. The new colour value of the particular pixel is given by:

$$\begin{aligned} P(i,R) &= 255 \\ P(i,G) &= 255 \qquad \forall i \in FaceOutlineVertex \\ P(i,B) &= 0 \end{aligned}$$

5.4 Conclusion

After the iteration of altering the pixel, we will get the result in Figure 5.7. Note that this step should be done after the analysis of the facial expression module. It is because this step is directly changing the video frame pixel value which is the important raw data in the analysis process.

Chapter 6: Facial Expression Analysis

6.1 Introduction

This is one of the major components in our system. The other one is the Modelling module which will be shown in details later on. The critical area in this module is how to distinguish different facial expressions from the video stream source. Since we already have the face coordinate value, the nature way is simply by comparing each value of the coordinates from two consecutive video frames and then predicting the movement of the face components. It is, however, not quite work in most of the time.

6.2 Detect by coordinate system

6.2.1 Introduction

Our first goal is to detect the eye blinking and mouth opening by using the coordinates retrieved from the FaceCoordinate filter. We tried to look at the difference between a pair of vertex such as (33, 41) and (34, 40). In the measurement, we need to record a reference distance so as to determine the opening or closing of the face component. The reference distance is recorded at the beginning of the video. The user can also re-initiate the reference distance by clicking the button on the property page of the filter. This step is important because this system can be used to adapt for different people.

A movie is recorded to analyze the variation of the coordinate. The movie includes the motion of eye blinking and mouth opening. We have collected the value of the face mesh coordinate from consecutive video frames. As we want to statistically analyze the coordinate variations, we have plotted graphs as follow. In the graphs, the x-axis is the frame number, which can be imagined as a timeline of the movie. The opening or closing of the face components is highlighted by a square frame.

As shown in figure 6.1, we used three pairs of vertex (33, 41), (34, 40), (35, 39) to compare their distance with the following formula:

$$\Phi = | V_1(y) - V_2(y) |$$

where V(y) is the y-coordinate of the face mesh vertex

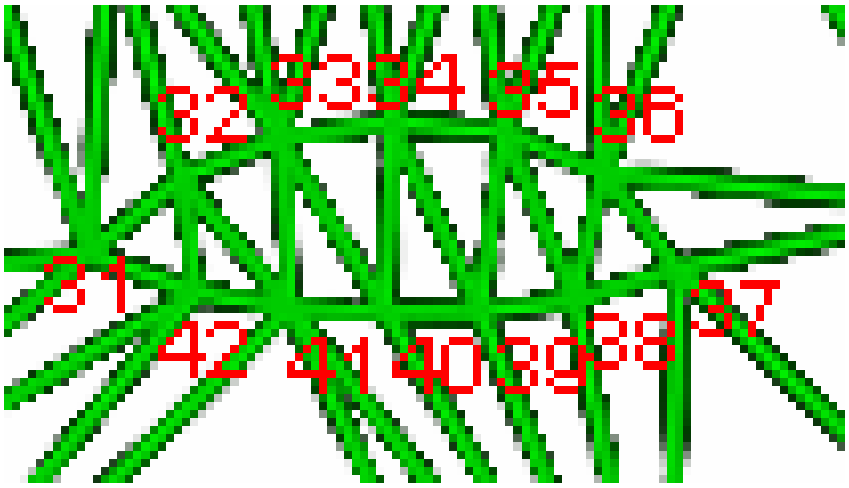

Fig.6.1 Part of face mesh showing left eye

The statistical result is presented in the following figures.

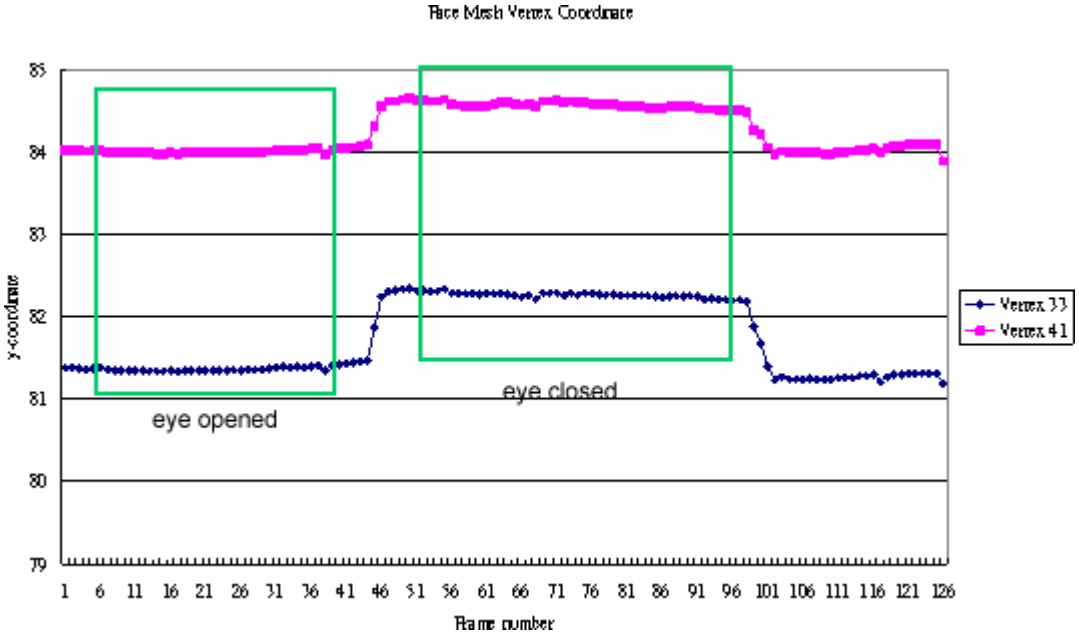

Fig.6.2a The relationships between coordinates of vertex 33 and vertex 41 and timeline (frame number)

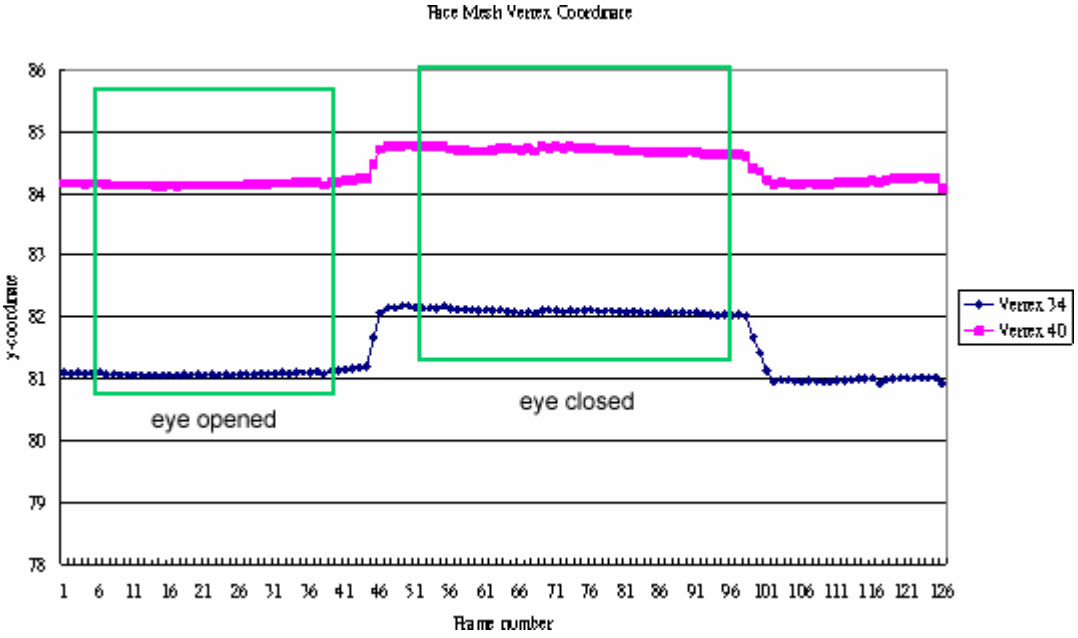

Fig.6.2b The relationships between coordinates of vertex 34 and vertex 40 and timeline (frame number)

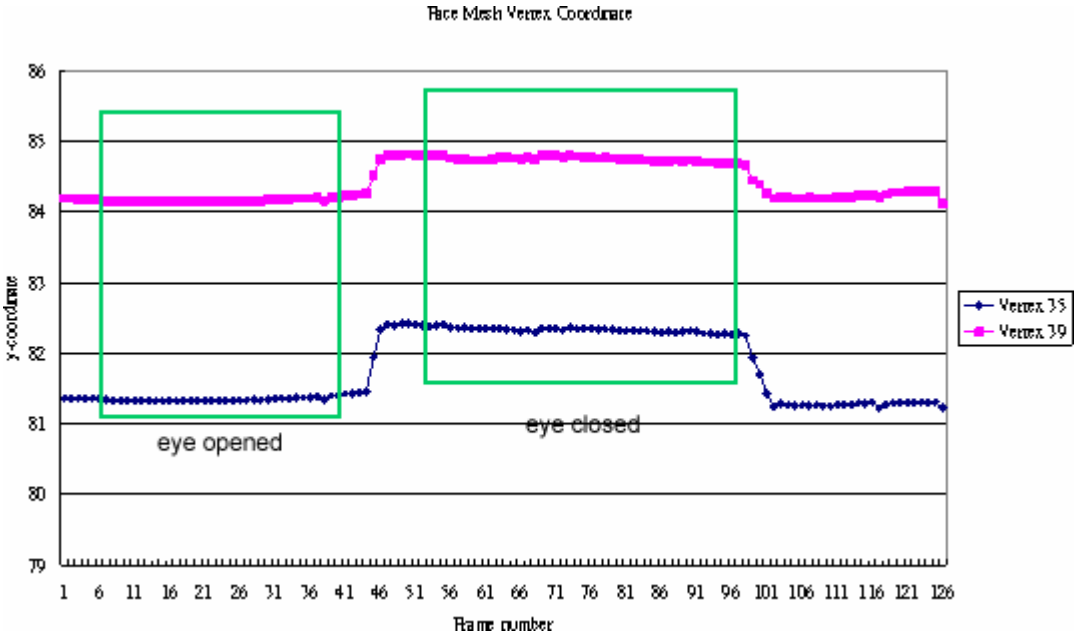

Fig.6.2c The relationships between coordinates of vertex 35 and vertex 39 and timeline (frame number)

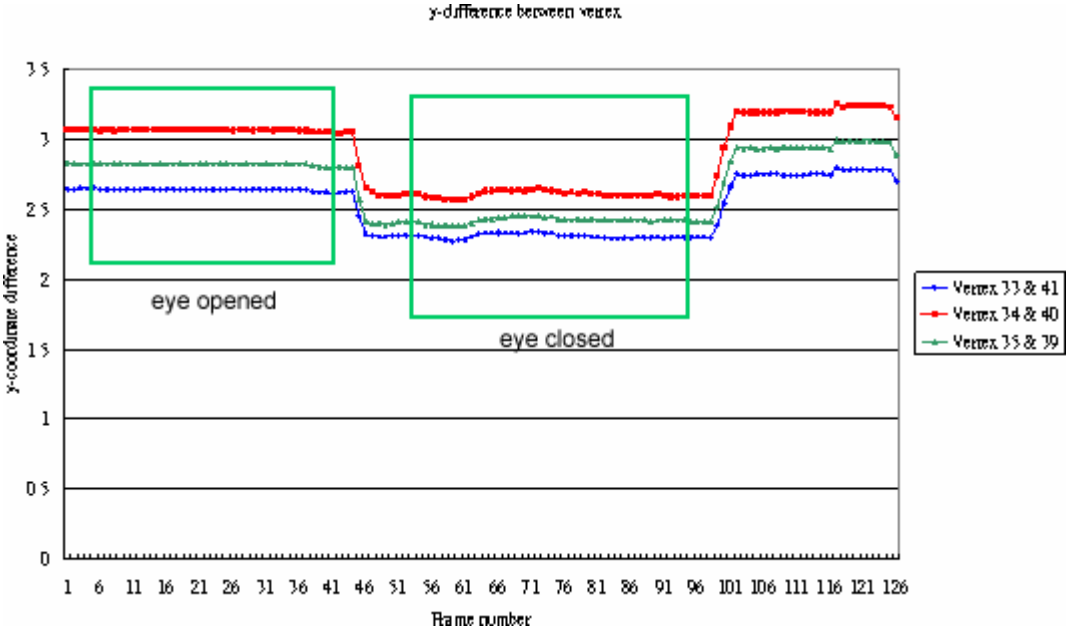

Fig.6.2d The change of y-coordinate difference of 3 vertex pairs when use is opening mouth

Next, we will show the statistical result of the different calculated pairs for mouth opening or closing detection. As shown in figure 6.3, we used totally 6 pairs of vertex to compare their difference. The outer mouth vertex pairs are (69, 77), (70, 76), (71, 75), while the inner mouth vertex pairs are (93, 99), (94, 98), (95, 97).

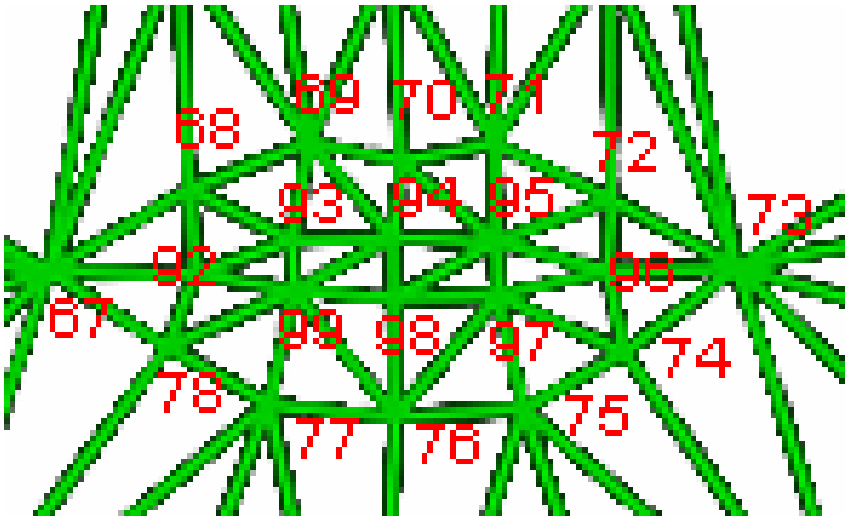

Fig.6.3 Part of face mesh showing the lips

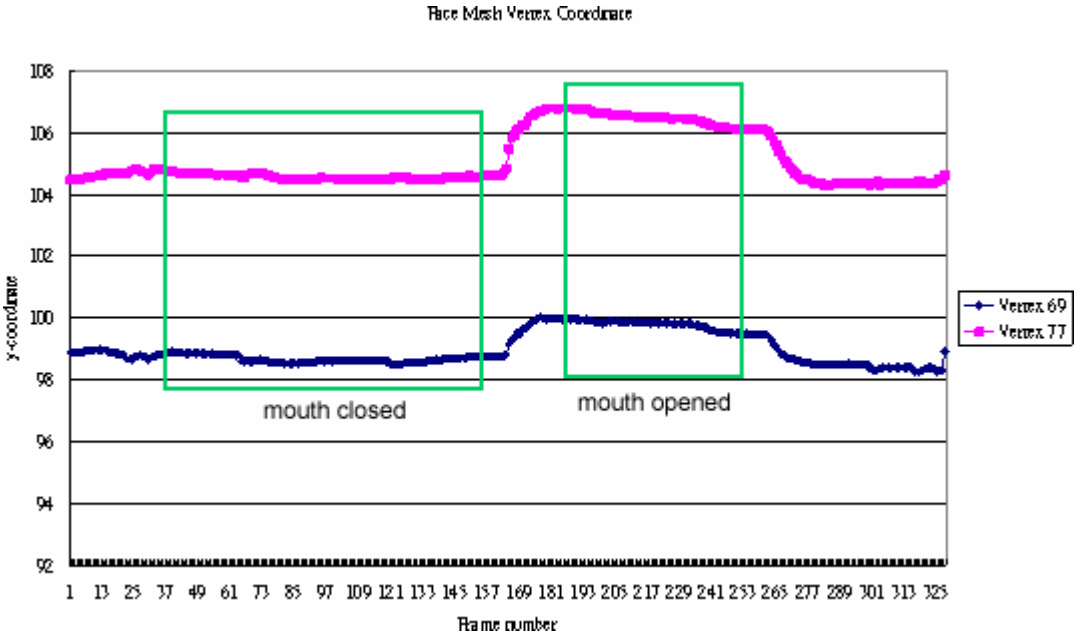

Fig.6.4a The relationships between coordinates of vertex 69 and vertex 77 and timeline (frame number)

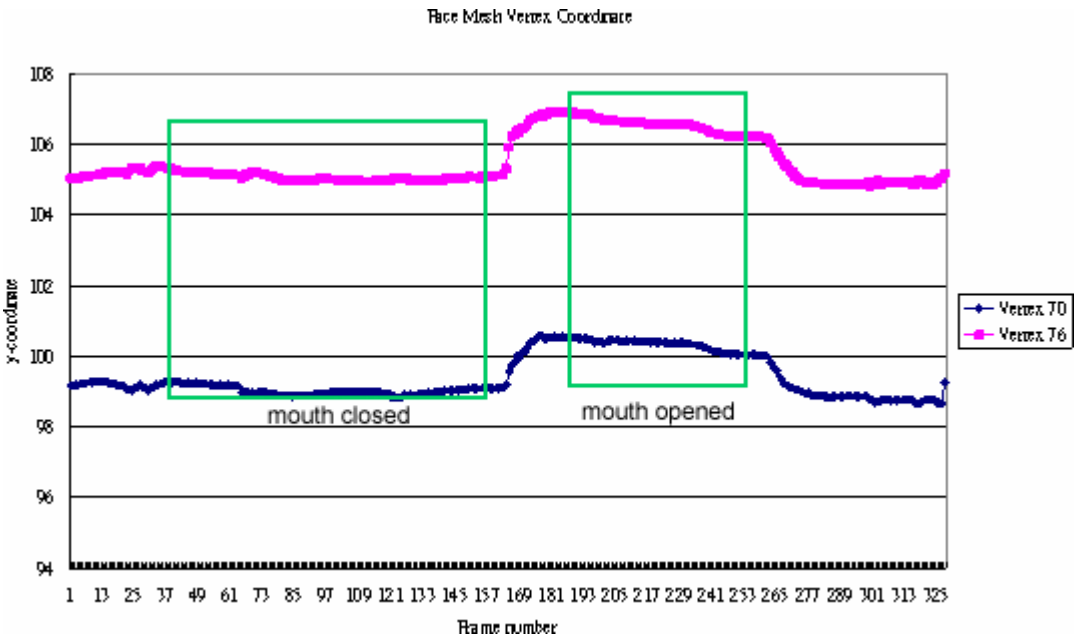

Fig.6.4b The relationships between coordinates of vertex 70 and vertex 76 and timeline (frame number)

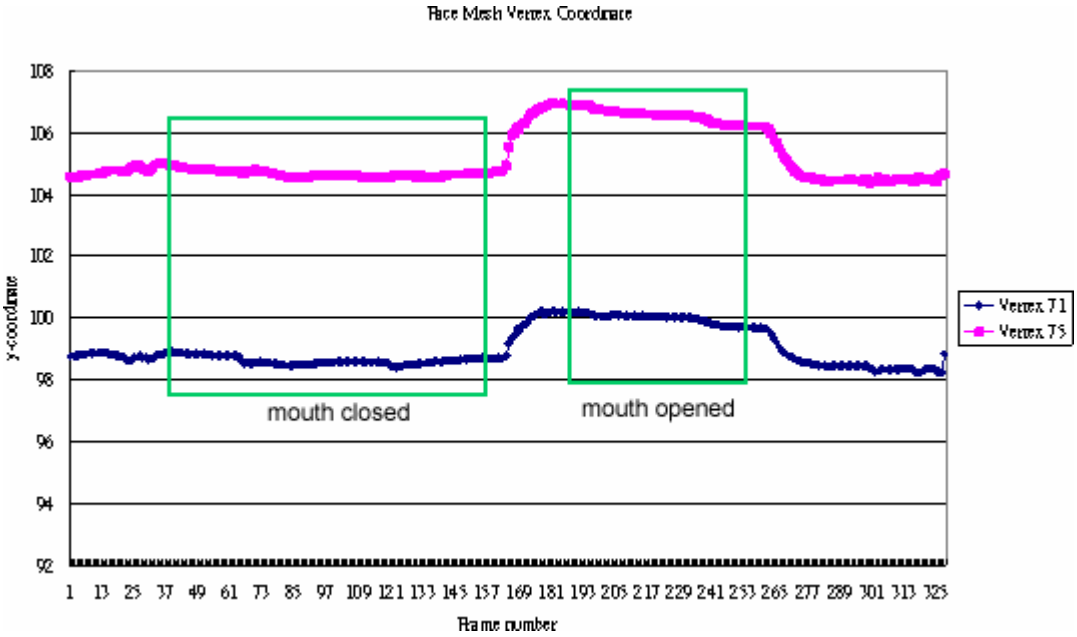

Fig.6.4c The relationships between coordinates of vertex 71 and vertex 75 and timeline (frame number)

Fig.6.4d The change of y-coordinate difference of 3 vertex pairs when use is opening mouth

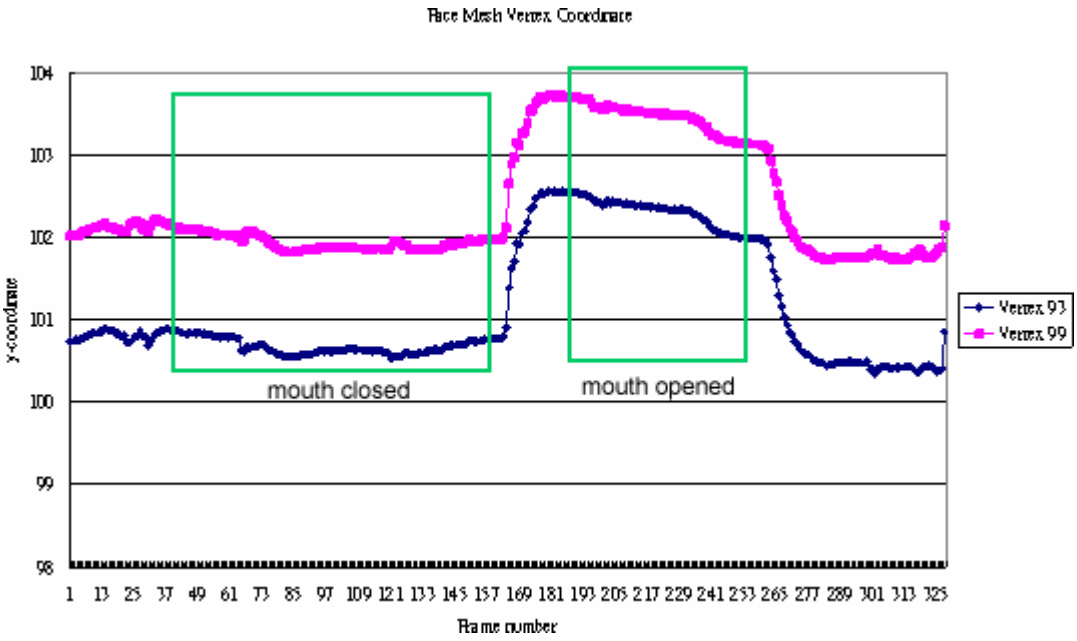

Fig.6.5a The relationships between coordinates of vertex 93 and vertex 99 and timeline (frame number)

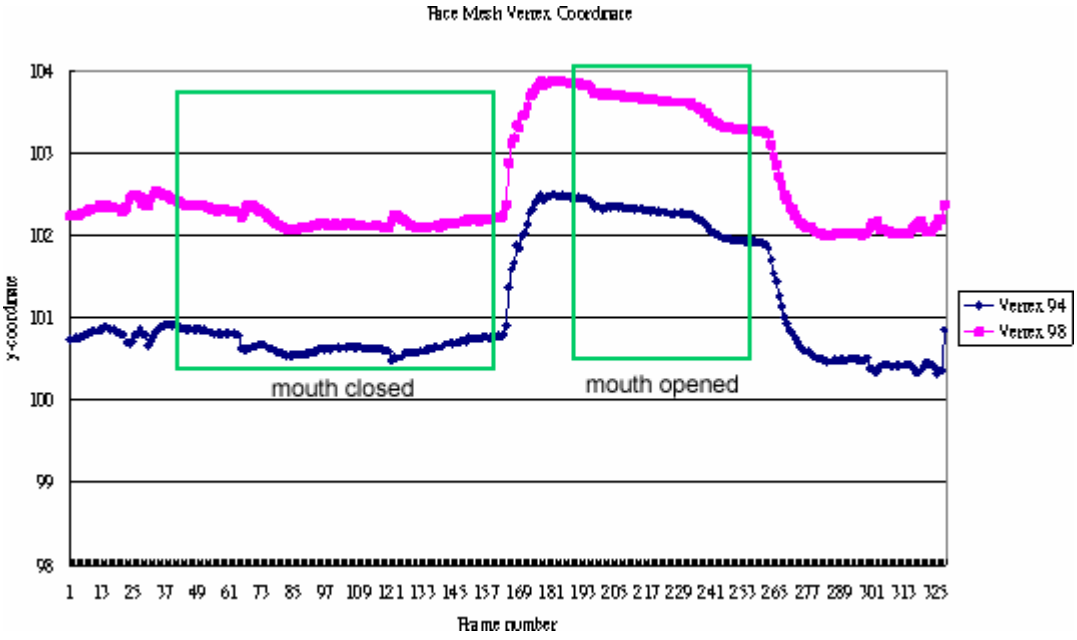

Fig.6.5b The relationships between coordinates of vertex 94 and vertex 98 and timeline (frame number)

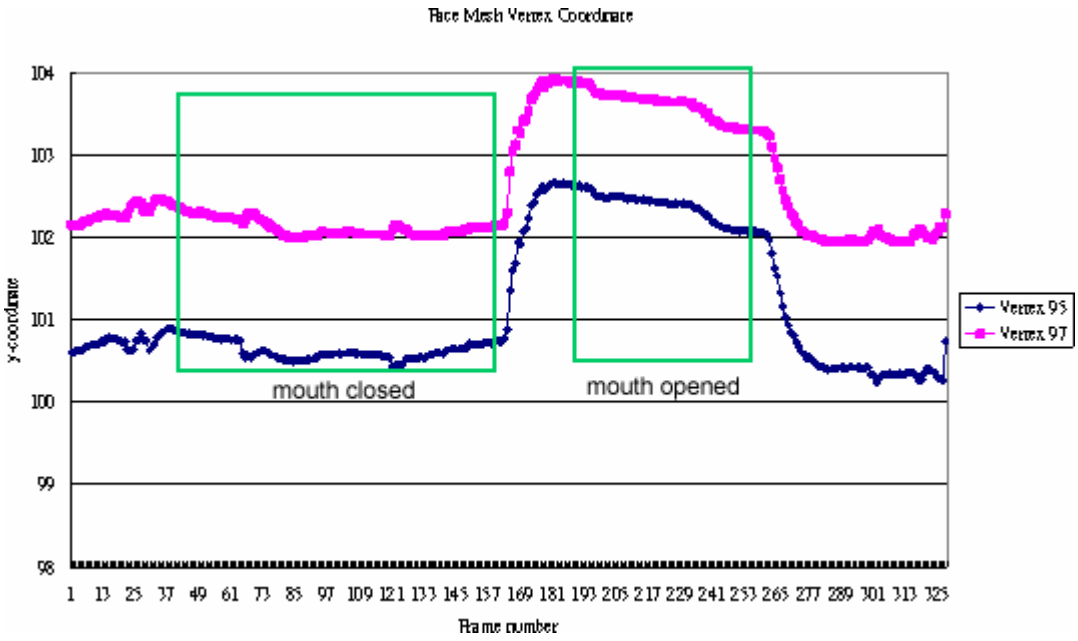

Fig.6.5c The relationships between coordinates of vertex 95 and vertex 97 and timeline (frame number)

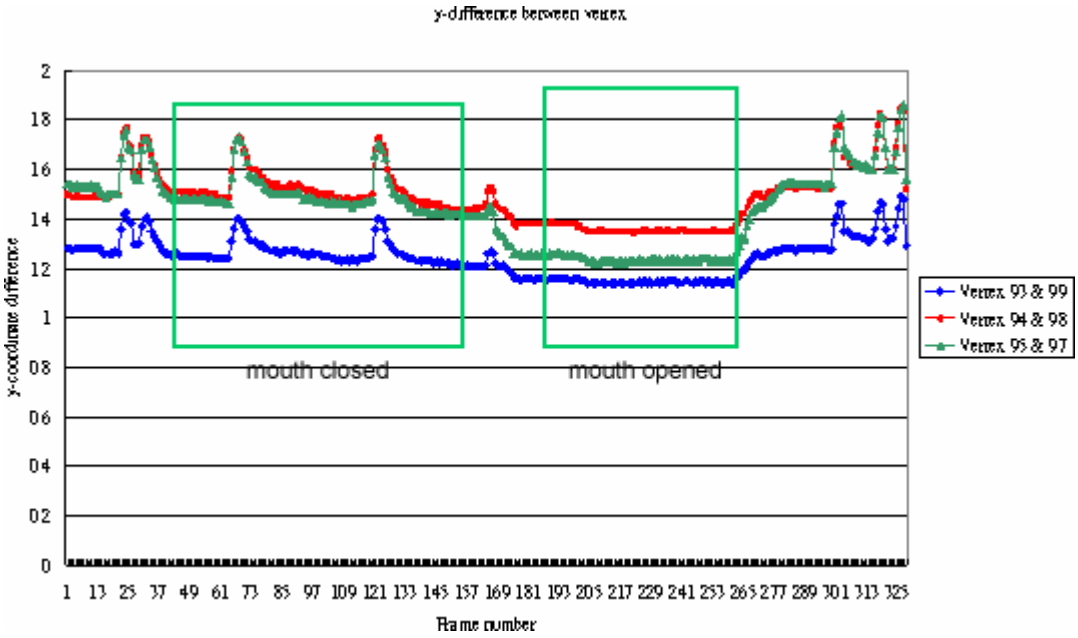

Fig.6.5d The change of y-coordinate difference of 3 vertex pairs when use is opening mouth

6.2.2 Interpretation

We can see that the average difference before opening mouth and closing mouth, or opening eye or closing eye, is just within 1 unit. This range is too small to distinguish the opening or closing of the face components.

There are many factors that can affect the difference obtained. If the user gets closer to the camera, the difference of the face components is increased because the human face is being magnified (nature of the camera). While the user gets far away from the camera, the difference of the face components is decreased because the human face is being minified. We have considered a solution that we would like to compare the ratio of y-coordinate difference to other y-coordinate difference of face component such as the y difference of the whole face, which has replaced using absolute value.

Another factor is that, if the user tries to move his or her head quickly, the mesh will fluctuate and the difference measured will not be precise. The reason of fluctuation is due to the fact that the frame rate is not high enough to allow Face Coordinate filter to detect the face.

6.2.3 Conclusion

We would like to conclude that, by using the coordinate system to determine whether the face components is opening or closing will result a high error rate. Nevertheless, it is a simple way to have a rough prediction of the movement and we will use it as a reference data.

Here, we propose three different approaches to identify the facial expression. Before showing the details, we first clarify the facial expression we are going to implement. The 100 face mesh coordinates we got from the FaceCoordinate filter is concentrated on the eye, nose, mouth and the face outline. Our study follows this as well and is going to determine the variation of the mouth, eye and the whole face.

To begin with, we segment the human face into 172 pieces of irregular triangle based on the 100 face mesh coordinates.

6.3 Competitive area ratio

6.3.1 Introduction

We have observed that when user opens his mouth, the triangles under the mouth are getting smaller. At the same time, the triangles marked as red in the figure 6.6 is getting larger. We want to compare these two sets of area to see the ratio between them. However, the triangles are irregular and we only have the coordinates of three vertices in each triangle. To calculate to area of the triangle, we have used the following approach.

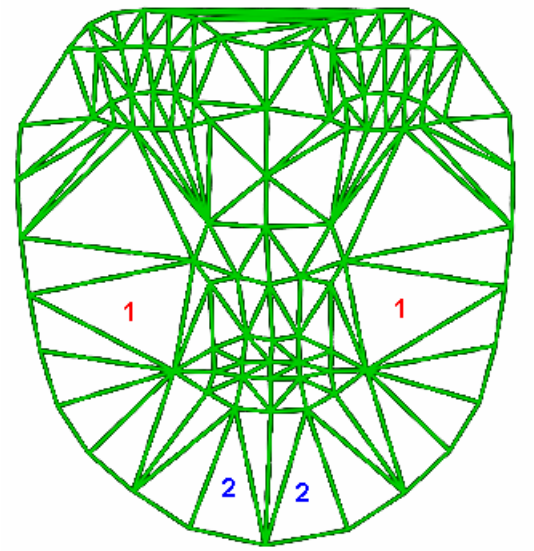

Fig.6.6 Face mesh marked by 1 and 2, which indicates triangle gong to be compared.

6.3.2 Interpretation

Consider an irregular triangle with vertices of arbitrary coordinate $(x_1, y_1), (x_2, y_2), (x_3, y_3)$

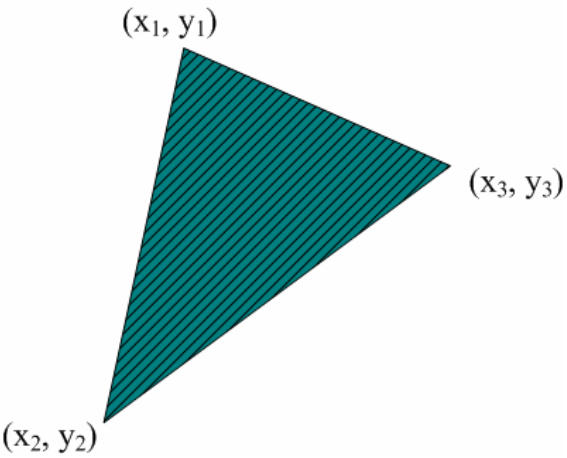

Fig.6.7 A triangle example showing how area is being calculated.

By applying the Cramer's Rule, we get

$$A = (\frac{1}{2})\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$ where A is the area of the triangle

The determinant of the matrix is

$$D = x_1 y_1 + x_2 y_3 + x_3 y_1 - x_1 y_3 - x_2 y_1 - x_3 y_2$$

It can be factored to:

$$D = (x_1 - x_3)(y_2 - y_3) - (x_2 - x_3)(y_1 - y_3)$$

So we can finally get the area of the triangle by

$$\frac{1}{2} D = \frac{1}{2}(x_1 - x_3)(y_2 - y_3) - (x_2 - x_3)(y_1 - y_3)$$

By comparing the ratio of triangles 1 and blue triangles 2 indicated in figure 6.6, we can know the mouth is opening when the ratio is larger than threshold.

After collecting the coordinate's information we can do the area calculation using the above formula. After that, we will record relationship of the area ratio between triangles marked by 1 and triangles marked by 2, with the frame number.

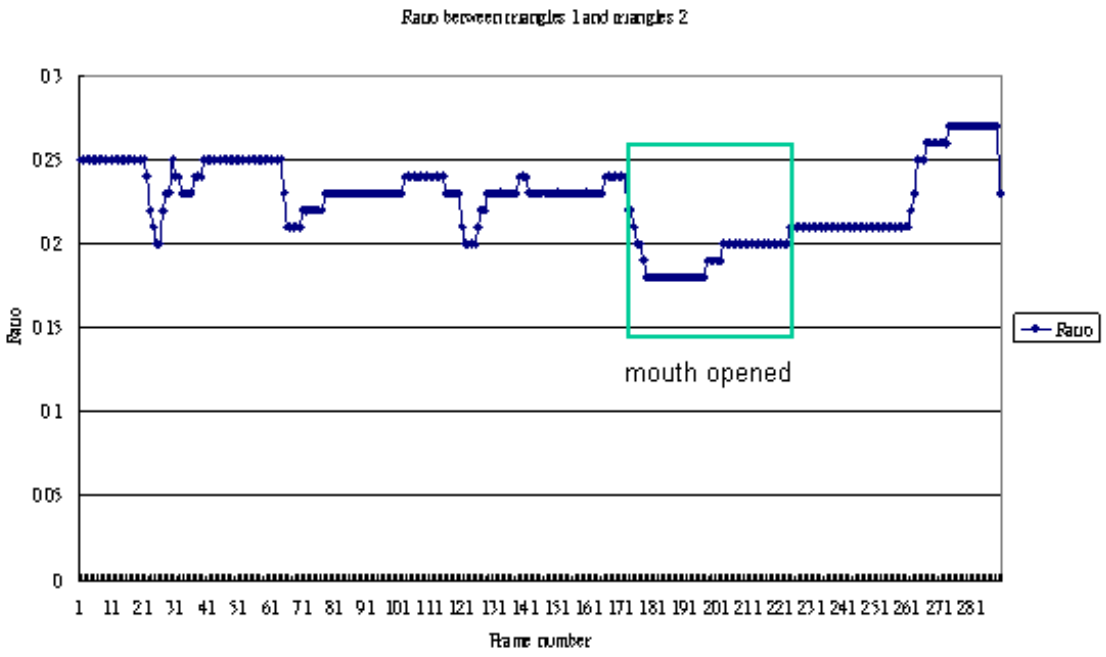

Fig.6.8 The ratio analysis between triangles

The threshold is statically obtained after sampling few movie clips. In figure 6.8, it is obvious that the ratio drops under 0.2 of the mouth is opening.

6.3.3 Conclusion

Competitive area ratio so far is an acceptable algorithm to find out the facial expressions. However, we must work out experiments as much as possible to obtain the changes of the human faces.

Also, we realize that different people would have different face area ratio. We try to make this system becomes more generic. That is, allow most people to play with this system. We should do initialization before user tries using this system. The initialization mainly record normalize original ratio before doing any facial expressions, and record the new ratio if user does different kinds of facial expressions.

6.4 Horizontal eye-balance

We want to measure whether the head is inclined. An example of head inclination is shown in figure 6.9.

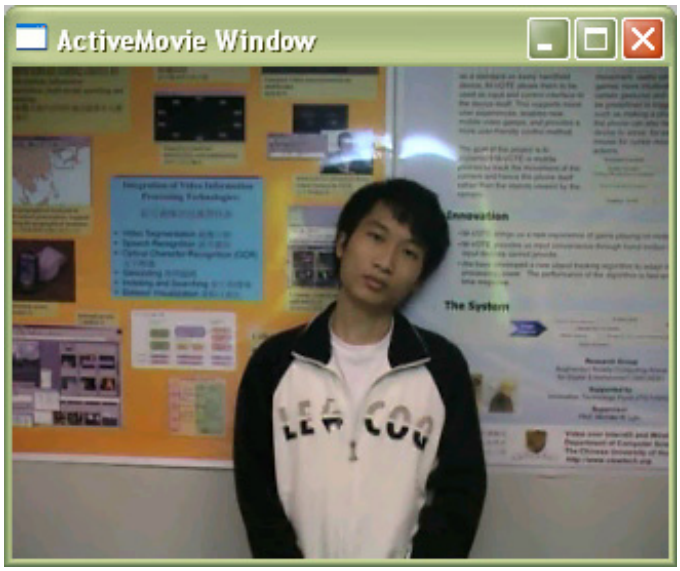

Fig.6.9 User's head is inclined

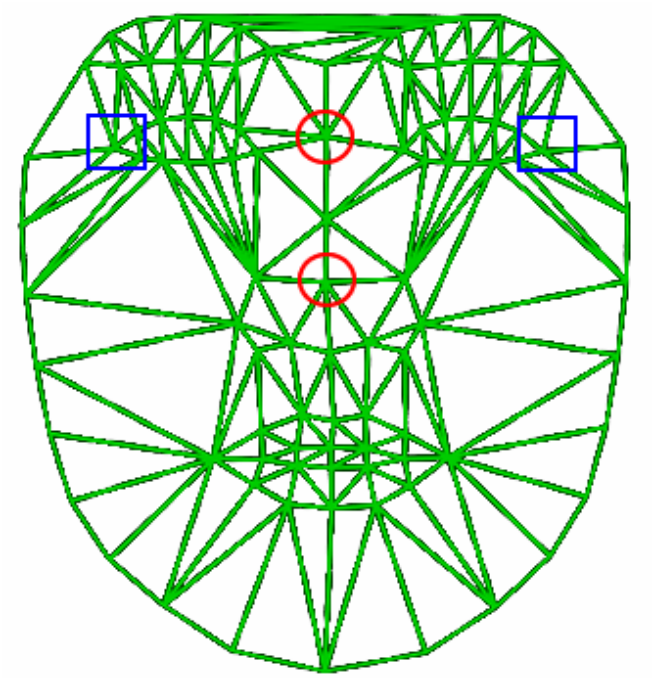

Fig.6.10 Face mesh with circles and squares to indicate the referenced vertices. Vertices surrounded by Circle: (89, 91), Vertices surrounded by Square (19, 31).

The first approach we used is using the angle between the line connected by the vertices in circles shown in figure 6.11 and the horizontal axis.

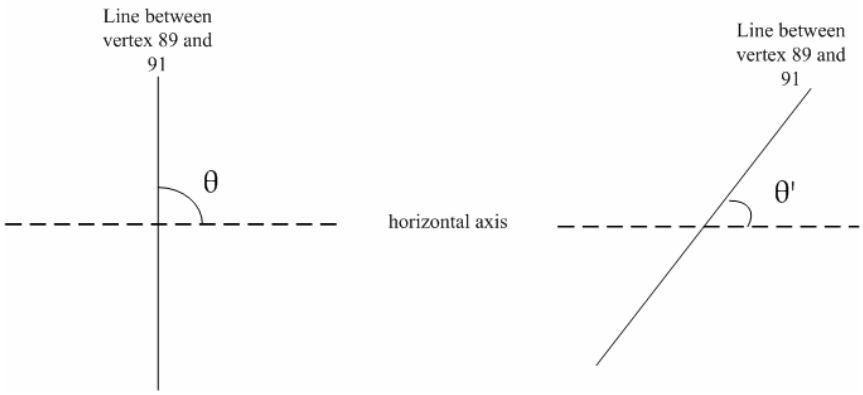

Fig.6.11. Calculation the angle using vertex (89, 91)

By comparing the angel measured, we can determine whether the head is inclined. However, this approach is not accurate if the user tries to turn his head to the left or to the right. Figure 6.12 shows the sampled line. Thus, the accuracy is poor by using this approach.

Fig.6.12 An example of inclination of the line without head inclination (head turning)

To avoid the above problem, we used another two referenced vertices (bounded by square in figure 6.13) to measure the angle between the line and the horizontal axis.

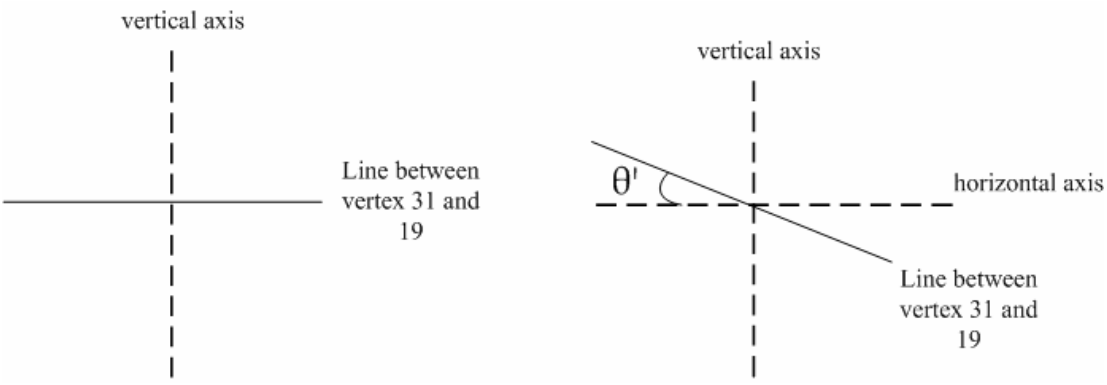

Fig.6.13 Calculation the angle using vertex (19, 31)

We use the following formula to calculate the angle,

$$\theta' = a\tan(\frac{|y_1 - y_2|}{|x_1 - x_2|})$$

If $y_1 < y_2$, it indicates head is inclined to right

Else if $y_1 > y_2$, it indicates the head is inclined to left.

6.5 Nearest-colour convergence

6.5.1 Introduction

In the following figure, we want to find out whether the eye is closed or not. To start with, we consider the red area of the eye and make an assumption that the iris is nearly black in colour.

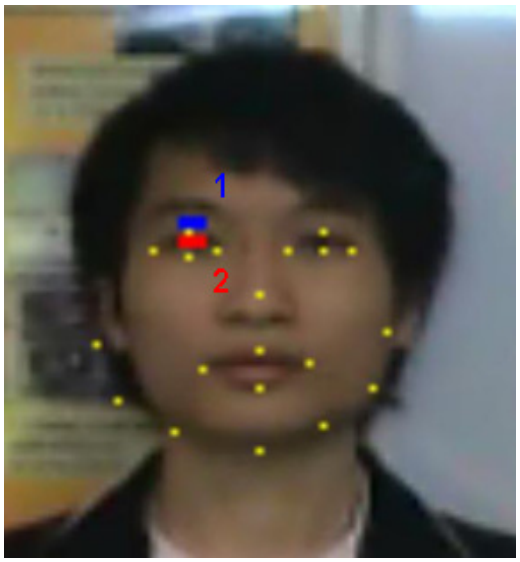

Fig.6.14 The sampling area of collection colour

6.5.2 Interpretation

As the video source is stored in RGB format, we treat the pixel colour into three different channel and store the value of each channel within the red area (indicated by 2). Then, we compute the average value in each channel so as to minimize the variation of the colour value. The average value is given by:

$$r_1 = \frac{\sum_{i=1}^{k}(i,R)}{k}$$

$$g_1 = \frac{\sum_{i=1}^{k}(i,G)}{k}$$

$$b_1 = \frac{\sum_{i=1}^{k}(i,B)}{k}$$

We can think of this result as if producing a new pixel with colour value (r_1, g_1, b_1). Besides, we would have a threshold colour value which indicates the colour should be contained in an opened eye. We, then, map the colour value into a 3-D coordinate system in such a way that r = x-coordinate, g = y-coordinate, b = z-coordinate. Using this coordinate geometry, we can find out the difference between the new pixel colour and the threshold colour by finding their distance in the 3-D space. The difference is given by:

$$diff. = \sqrt{(r_1 - r_\Phi)^2 + (g_1 - g_\Phi)^2 + (b_1 - b_\Phi)^2}$$

and the difference would be accepted if it has the following property:

$$0 \le diff \le \Phi$$

6.5.3 Conclusion

However, light intensity is different under different environment. Besides, different face may have different colour iris. To achieve a better accuracy, we have modified the threshold to become a colour sampling from the blue area (indicated by 1). This area is taken for analysis as they would have the nearest colour when the eye is closed. The new threshold value can be computed directly by applying those equations in the above and the other equation would be held.

6.6 Conclusion on face analysis algorithm

With the above three different approaches, our system can now detect three kinds of facial expression. To detect each facial expression, we may use more than one approach to maximize the accuracy.

There exists many factors affect the result obtained from the face analysis algorithm. Thus, we apply different algorithm as to compensate the weakness of the other algorithm.

We still try to think about whether some other new algorithm can detect a more interest facial expression or optimize the accuracy of our system. To achieve this, we need to do many statistic researches so as to find out the pattern of the data records.

Chapter 7: Texture Sampler

7.1 Introduction

In the rest of our system, these two modules are implemented so as to reflect the facial expression result generated from the previous module. We do this by making use of the Direct3D texture map capability.

7.2 Basic concepts

7.2.1 Texture Mapping

Texture behaves like bitmap graphic which uses two-dimensional array to store the colour value. In texture perspective, each colour value is called a textual element (Texel). Similar to a pixel in bitmap graphic, each Texel has its own address as a position indicator. There are two components, which can be imagined as a column number (u) and a row number (v).

Texture coordinates are used in texture space. The coordinates are representing the point relatively to the point (0, 0) in the texture. Before a texture is applied to a primitive in 3-D environment, the Texel addresses must be translated into screen coordinates or pixel locations.

Microsoft Direct3D uses inverse mapping to map the texels directly to the screen space. It is done by finding corresponding Texel position in each screen space pixel. The colour around the Texel position is being sampled to the screen space.

A generic addressing scheme is applied to Texel address as to provide a uniform address range for all textures. Under this addressing scheme, the vertex resides in the range, 0.0 to 1.0, inclusively. In our project, we normalize the coordinates in bitmap to this addressing scheme by dividing the x and y position of the pixel by the width and height of the bitmap graphic respectively.

In figure 7.1 we can see that with the same texture address (0.5, 1.0), we sampling different coordinate in different texture. That is, (2, 4) in texture 1 while (3, 6) in texture 2.

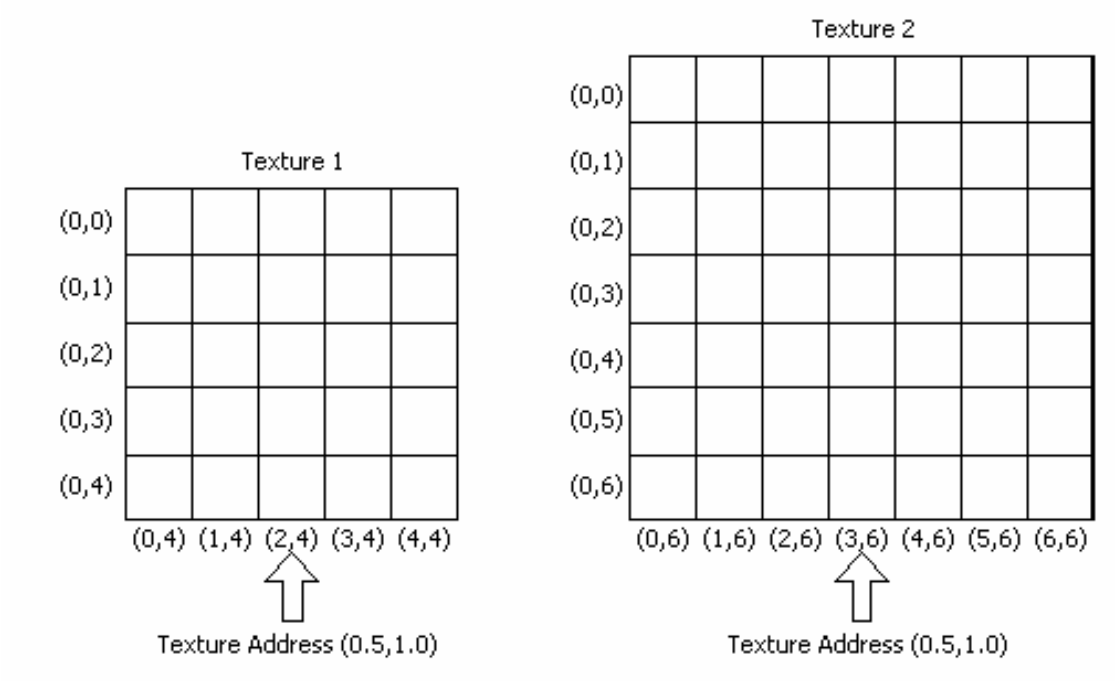

Fig.7.1. Above picture shows different textures address are pointed by same texture address

When the texture is going to map onto the primitive surface, it uses the addresses of four corners in a pixel to map with the 3-D primitive in object space. The pixel will be distorted since due the shape of primitive and the viewing angle.

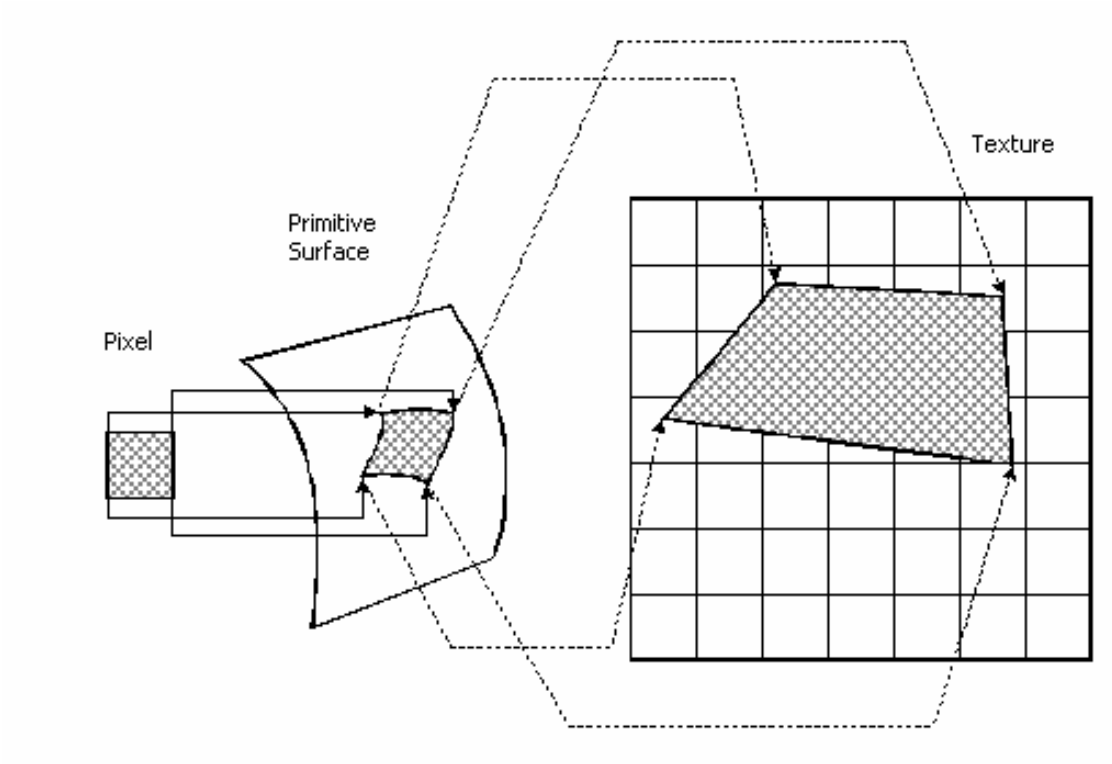

Fig.7.2. The distortion of a pixel when it was pasted on a primitive surface

The computation of pixel colour is done by collecting texels colour in the area which the pixel maps. There are a few texture filtering methods which determine how Direct3D use the texture to create a colour for a pixel.

The texture is being magnified or minified when texture filtering is being performed. There could be two possible results. That are, more than one pixel is being mapped into one Texel, or one Texel is being mapped into one pixel. However, these effects will make the image become blurry or aliased. To avoid such situation, blending must be done on Texel colours before we apply it to pixel. We are using the Direct3D default option, nearest-point sampling to simplify the filter operation.

7.2.2 3-D primitive

Texture only provides a kind of material on the surface. It likes a colour panel where we can choose a colour to be displayed. But how the shapes of object to be draw on the screen depending on what primitive is used.

We often refer 3-D primitives as a polygon in which there are at least three vertices. A triangle is the most common primitives and is used in our system.

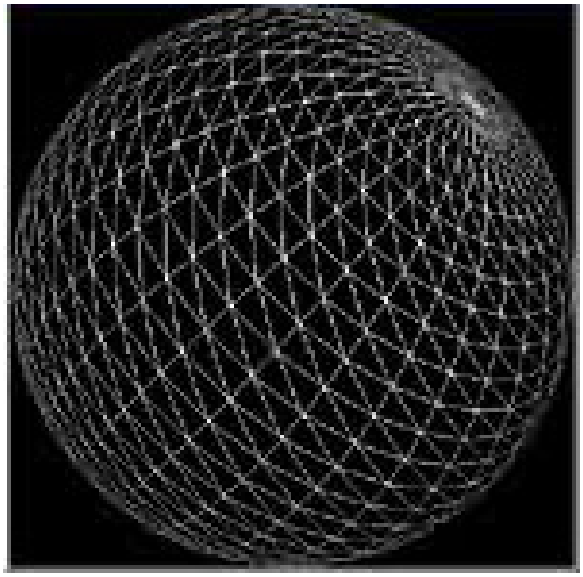

Fig.7.3 A wire frame sphere created by triangle unit

7.2.3 Vertex buffer and index buffer

Vertex buffer objects enable applications to directly access the memory allocated for vertex data which is used to represent the coordinate we want to draw on the scene. For example if we want to draw a triangle, we need to provide the 3 vertices coordinate to the Direct3D device.

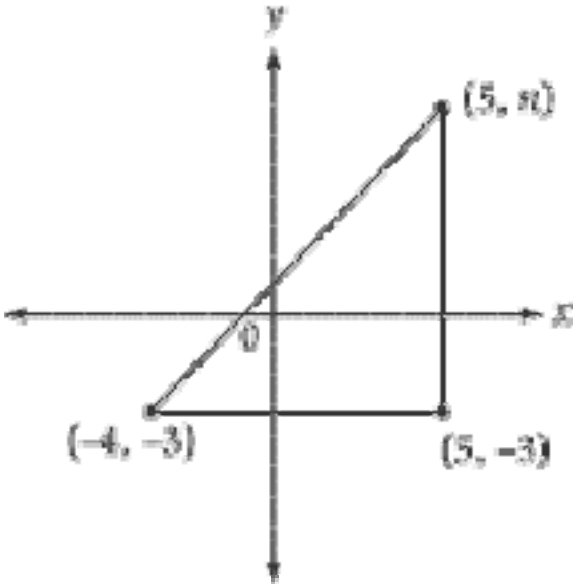

Fig.7.4 drawing a triangle using three vertices

The face mesh contains different size of triangle and each vertex is used to represent more than one triangle. There are many common vertices in the mesh.

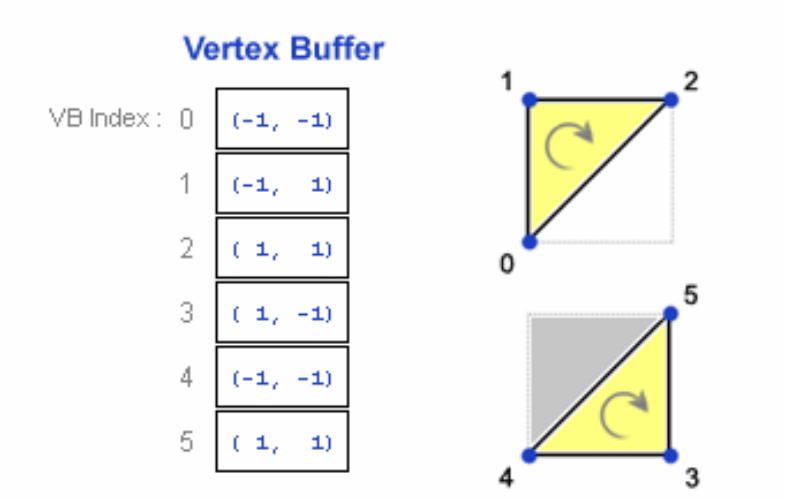

Fig.7.5 using vertex buffer to represent a square

Therefore, if we represent each triangle vertices separately, there would be a few hundred of data in the vertex buffer. In order to avoid duplicate data, we introduce the index buffer to compress the vertex buffer. Another important advantage it takes is to allow the display adapter to store those vertices in a vertex cache. The display adapter can fetch those recently used vertex data from the cache which enhance the performance greatly. Here is an example,

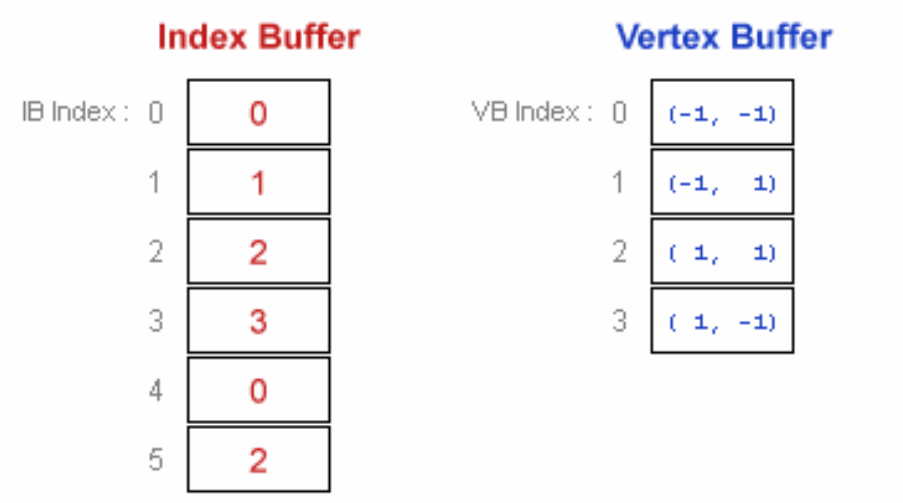

Fig.7.6 using index buffer to represent the square

7.3. Implementation

7.3.1 Pre-production of texture

Our approach is using a predefined image to associate with one facial expression and a set of images would be called a character model. A pre-production process is done in this purpose. We collect a few image of the same character and manually modify them to represent the different facial expression and their combination. There are currently two character models to be used in the system.

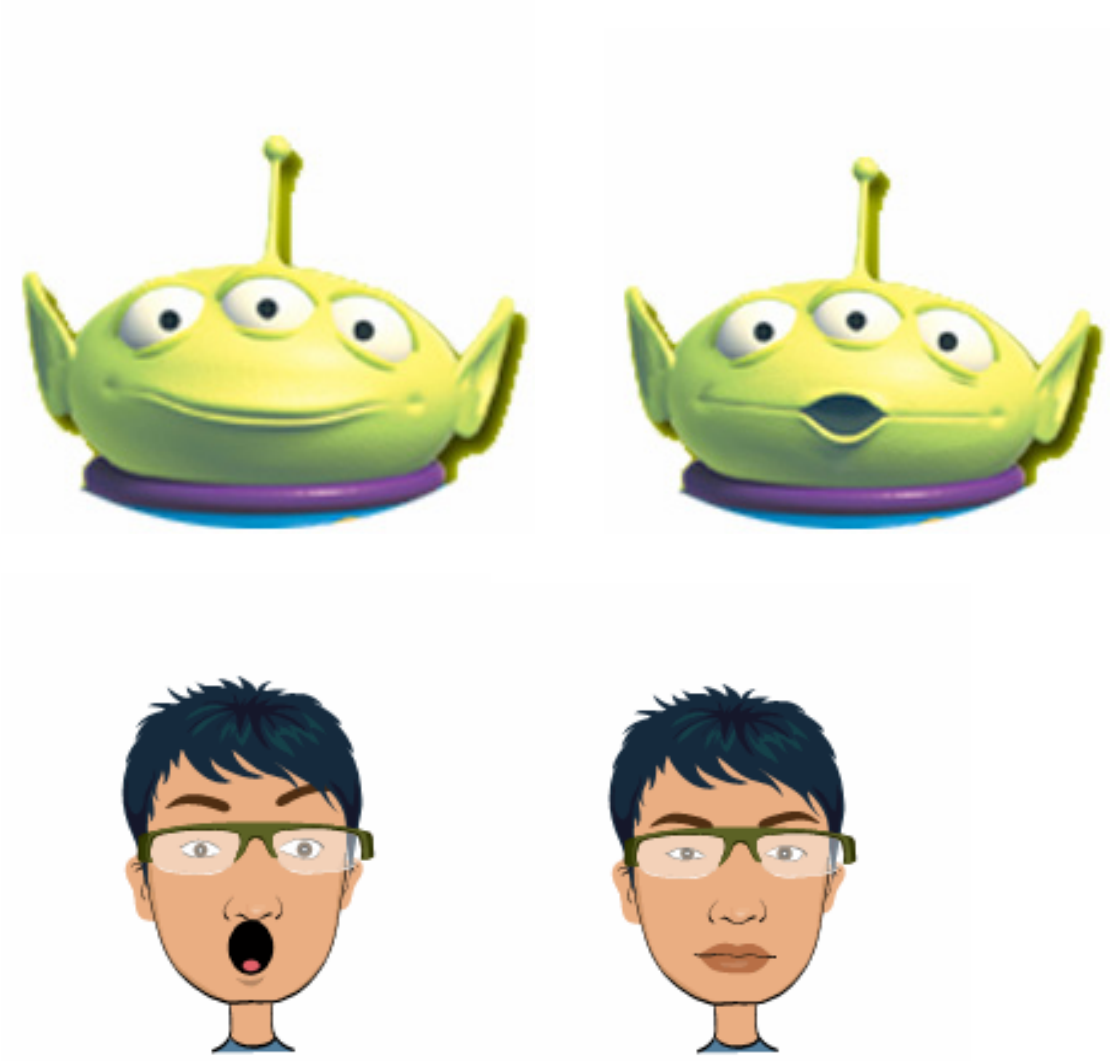

Fig.7.7 Some examples of facial expression in the two character models

7.3.2 Loading the texture

The next stage is to create the texture from those file-based images. For each image, we need to create a corresponding texture map. In order to have a smooth running when switching the character model during execution of the application, this step should be done before the application starting to run.

7.3.3 Mapping object coordinate to Texel coordinates

After this initialization, we will have our own texture resource database and is ready to be used in the system. The next step is to define a modelling vertex data to map the face mesh coordinates. The modelling vertex data would contain 100 modelling vertices in order to have a 1 to 1 mapping of the face mesh coordinates. The modelling vertex data will be further used by the next module to render the character. In each vertex, we need to have two set of coordinates, one is object coordinate used to draw geometric shapes in the 3D scene, and the other one is texel coordinates used to supply our own colour, i.e., the loaded texture in this time. Each texel has a unique address in the texture and we have manually mapped each modelling vertex to a specified texel.

7.3.4 Prepare the index buffer

We then reduce the size of buffer to be loaded into the display adapter by specifying an index buffer which has the same number of entry with the number of triangle in the mesh.

Each entry would contain three numbers and each number represents the corresponding index number in the vertex buffer. The index buffer can be thought as a list of triangle.

7.4 Conclusion

After the mapping, we would now have those 100 modelling vertices filled with Texel value and the index buffer filled with the list of triangle vertex.

Chapter 8: Facial Expression Modelling

8.1 Introduction

Up to this module, everything we need to render the face object is already here. With respect to the facial expression analyzed result, we set the corresponding texture that the Microsoft Direct3D device will be used to render.

8.2 Basic concepts

We render our object in the 3-D scene but what we finally see on the screen is only a projected 2-D rectangle. This is so called the viewports. To specify the viewports, we need a viewport's camera and define the x, y, z axis direction. Our system is using the right-handed Cartesian Coordinates, that is, the positive z-axis is pointing out of the screen.

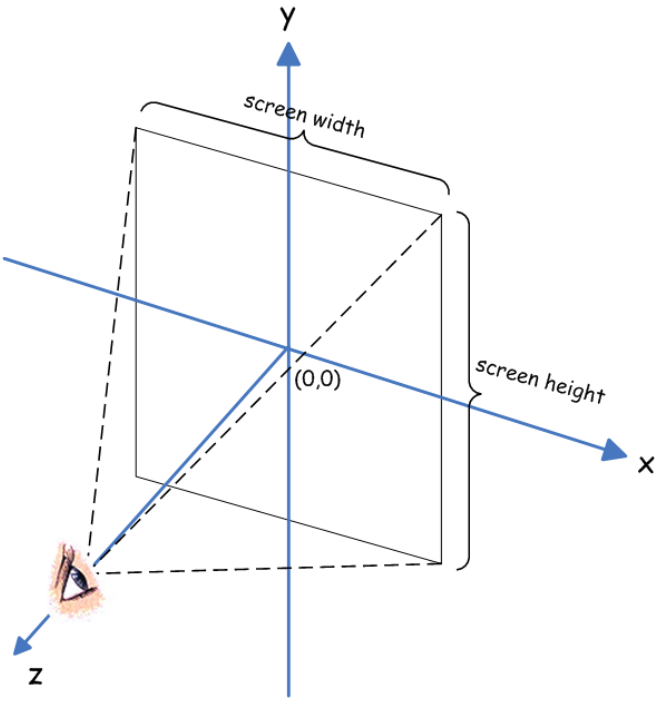

Fig.8.1 a simple viewport's example

Relative to the viewport's camera, we define a viewing frustum which is a 3-D volume in a scene. This volume affects how our object will be projected on the scene. This is done by specifying the fov (field of view) and by the distances of the front and back clipping planes.

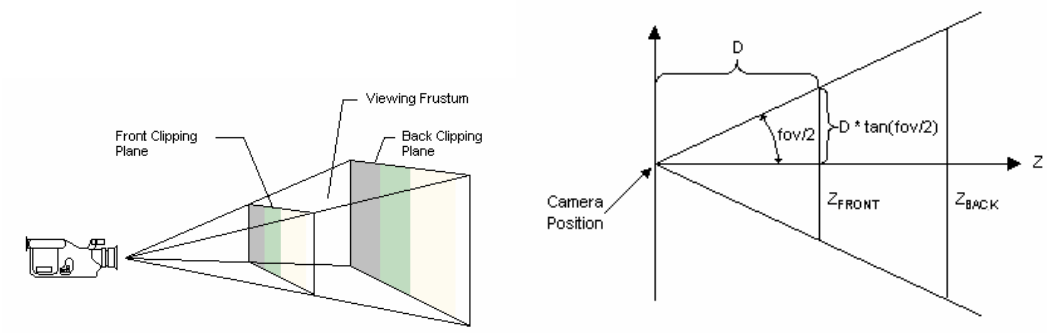

Fig.8.2 Define a viewing frustum using the fov and distance between the front and back clipping planes.

8.3 Implementation

To reflect the real-time movement of the face, we first need to update the object coordinates in the modelling vertex data from the face mesh coordinates. According to our own view port, the central position on the output screen is (0, 0). This configuration is needed in order to dynamically change the viewport dimension without shifting the origin. The face mesh coordinate, however, is referring the lower left corner as (0, 0).

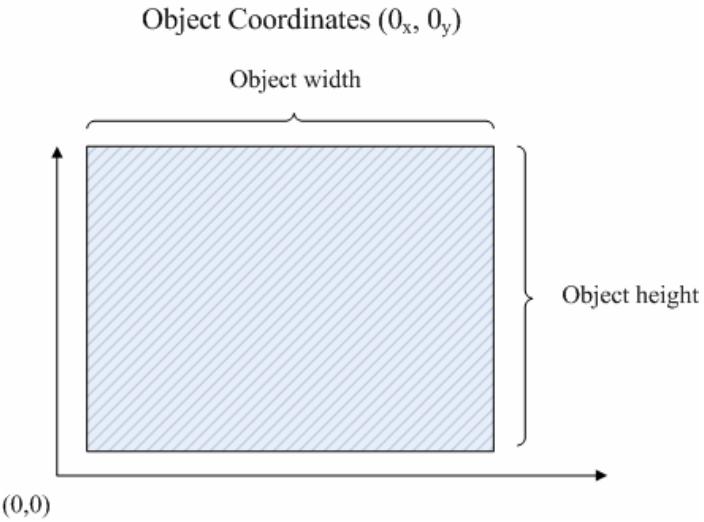

Fig.8.3 Object coordinates geometry

We need to shift the face mesh coordinates into the right position so that the whole rendered object will be seen in our screen. We can, then, fill up the vertex buffer which is our stream source with this complete modelling vertex data. The scene is finally ready to be rendered. We can then use the Direct3D device to render the index buffer.

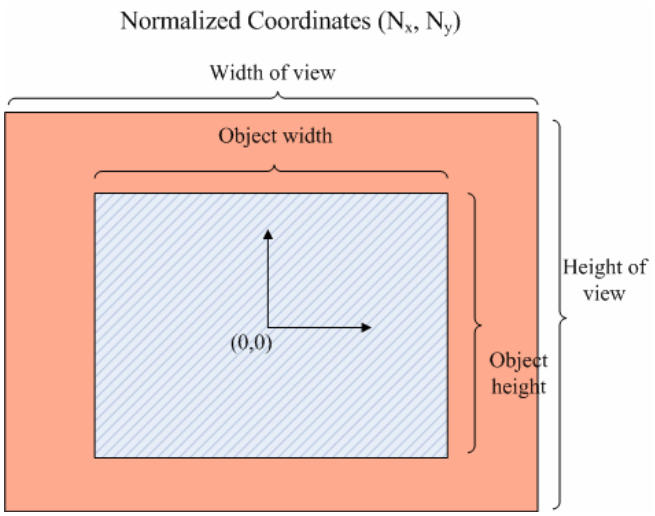

Fig.8.4 Normalized coordinates geometry.

8.4 Conclusion

After those shifting and rendering processes, we should able to see the output presented on two different screens. The ActiveMovie Window shows the modified video stream output. User can see himself/herself with adding the indications and check out whether the face mesh is fitted above their face. Once the mesh can detect the face, the character model will be shown on the Display View screen.

Fig.8.5 Normalized coordinates geometry.

Figure 8.5 An ActiveMovie Window showing the user (left) and the Display View showing the character model (right).

The facial expression image is now rendered under the modelling vertex data, therefore even in the same facial expression, the texture image and the rendered target would have different performance. Here is an example.

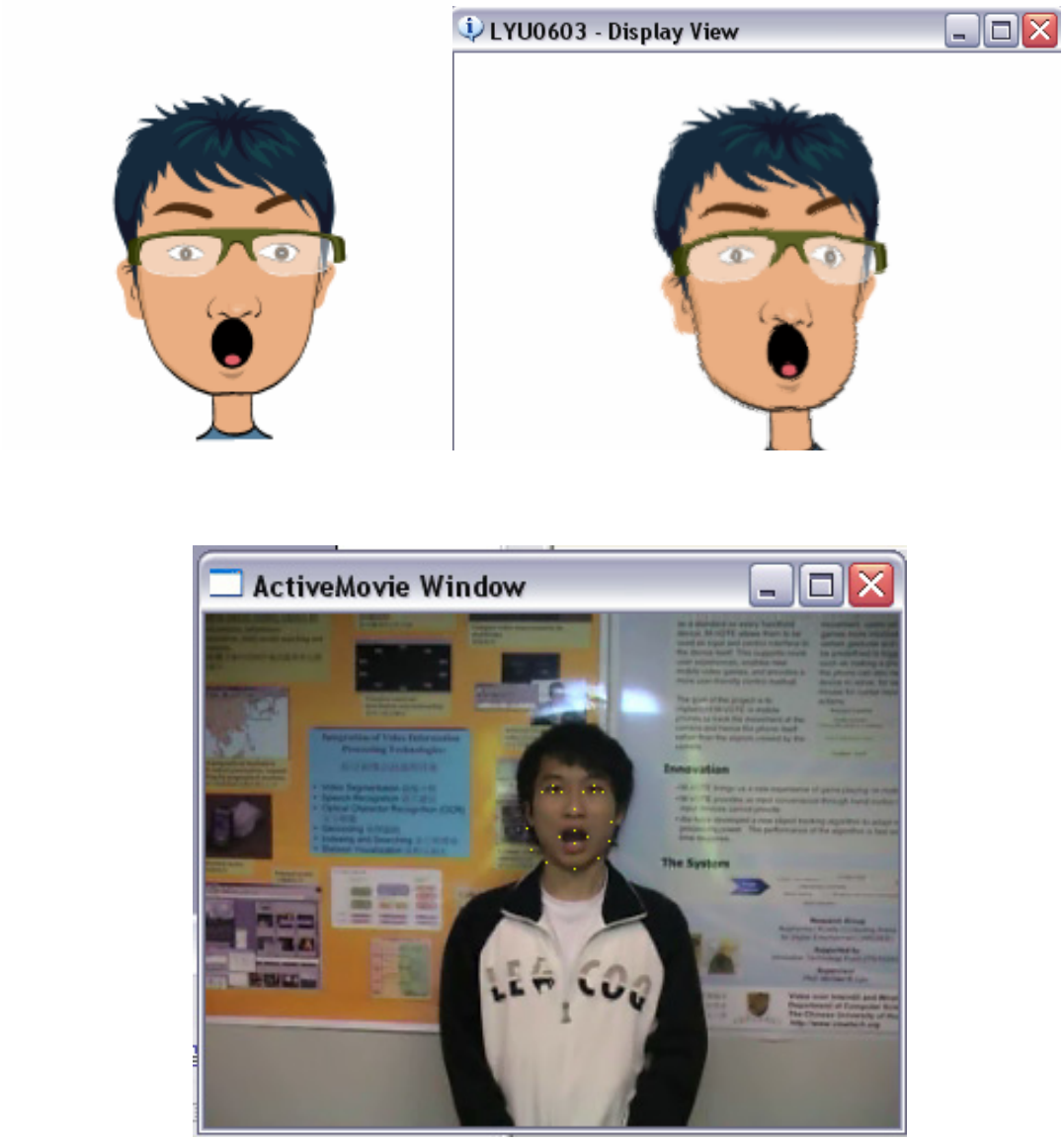

Fig.8.6 The original image file (top-left), the rendered output (top-right), and the current facial expression on the human face (bottom).

Chapter 9: Virtual Camera

9.1 Introduction

Virtual Camera is a DirectShow source filter we built which allows user to adapt our facial expression modelling filter into other existing applications such as MSN messenger. A source filter introduces data into the filter graph. The data can come from a file, webcam, network and etc. Virtual camera is simulating an ordinary webcam that it provides video data source to the computer system. The nature of the virtual camera is similar to a normal webcam that operating system can recognize it as a standard capturing device. The objective of building virtual webcam is to display our modelled cartoon character in camera output so that the one who is playing video conversion with the user is able to see the cartoon character only instead of the original video captured by webcam. In figure 9.1b, we can see that the Virtual Camera can be chosen in the MSN messenger "Audio and video setup" interface. Before the virtual camera can work properly, the computer should have a physical webcam installed in the computer and the virtual camera executable should have registered in the windows registry. Since virtual camera filter is a windows COM object, we have to register it into windows registry by regsvr32.exe.

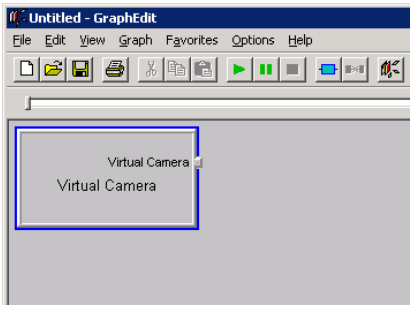

Fig. 9.1a Virtual Camera filter in GraphEdit

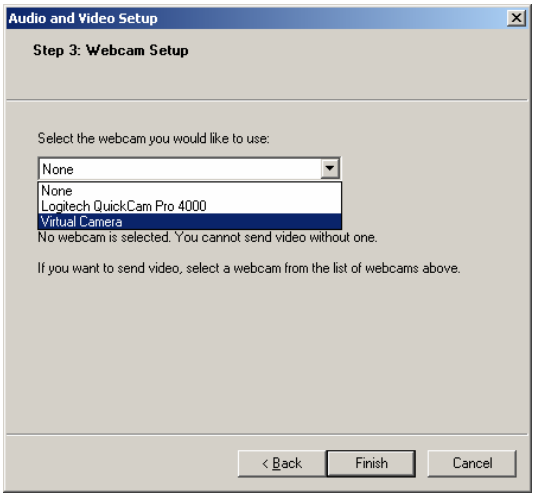

Fig9.1b Virtual camera can be detected in MSN Messenger

Besides cartoon output, a face detection outline window will pop up automatically when virtual camera is running. The face detection outline window is used to display the original video stream captured by the video capture device with the face mesh drawn by Direct3D. The face mesh draws the outline of the Face Coordinate filter detected, it facilitates user to check whether his or her face is being recognized and analyzed correctly. The face mesh is illustrating the face detection result. Once the user found that the face mesh is not fit onto his or her face well, he or she can tune the environment factors such as lighting and camera position as to achieve a better face recognition results. The face detection usually performs well in a balanced light condition, that is, the light source is distributed evenly in the

environment.

In figure 9.2 we can wee that user can use a cartoon character to display as its video conversion output and there is a window displays the face mesh detected.

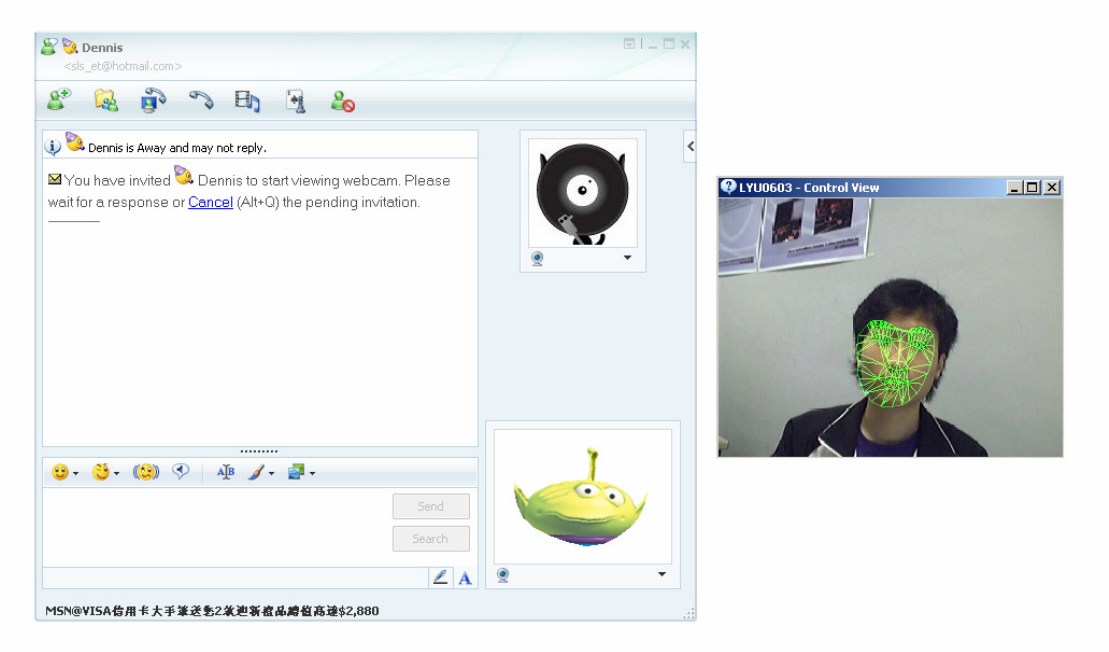

Fig9.2. Actual environment of using virtual webcam in MSN Messenger

9.2 Implementation

9.2.1 Inner Filter Graph

Virtual camera is implementing a class named as VCam which inherits from standard DirectShow class CSource. We have implemented and overridden the function inherited from CSource as to reach its functionality. The CSource class is a base class for implementing a source filter. A filter derived from CSource contains one or more output pins derived from the CSourceStream class. In our virtual camera source filter, we only possess of one output pin and the pin is programmed through CSourceStream derivation. The output pin creates a worker thread that pushes media samples downstream. We named our derived CSource class as CVCam and derived CSourceStream class as CVCamStream..

Once we create an output pin, we have to override three functions in CSourceStream. GetMediaType() validates the media types requested by the input pin which going to connect this output pin can be supported. The DecideBufferSize() method returns the pin's buffer requirements. The most important method we have to override is FillBuffer() method which fills a media sample buffer with data. The media sample buffer is the pool being passed to the downstream. We have built an inner filter graph to run the output we want and have copied the result drawn into this frame buffer such that the output pin in Virtual Camera source filter is providing the rendered data in another filter graph.

The architecture of virtual camera is that it has built and is running another inner filter graph which connects the required filter together and renders the output. We used IGraphBuilder interface which enables an application to build a filter graph. IGraphBuilder inherits from the IFilterGraph interface, it contains of functions in IFilterGraph which are fundamental in graph building. The graph built by the virtual camera first enumerates a physical existing capture device such as webcam to be a source filter in the filter graph. When virtual camera enumerates the input devices, it also enumerates itself. To prevent the recursive filter graph building, we have constraint the filter only accept the capture device with the name is not as same as Virtual Camera. After picking a device as input source, other filters which required in facial expression and modelling are inserted into the filter graph subsequently, they are Face Coordinate filter and Facial Express Modeling filter. We also have to add two additional filters, Sample Grabber and Null Renderer. Sample Grabber is responsible for providing an interface to virtual camera to access the video buffer going to be rendered. Due to the fact that a complete graph does not allow the absence of a terminate filter, a Null Renderer is added to be a terminal in a filter graph and it does nothing on rendering.

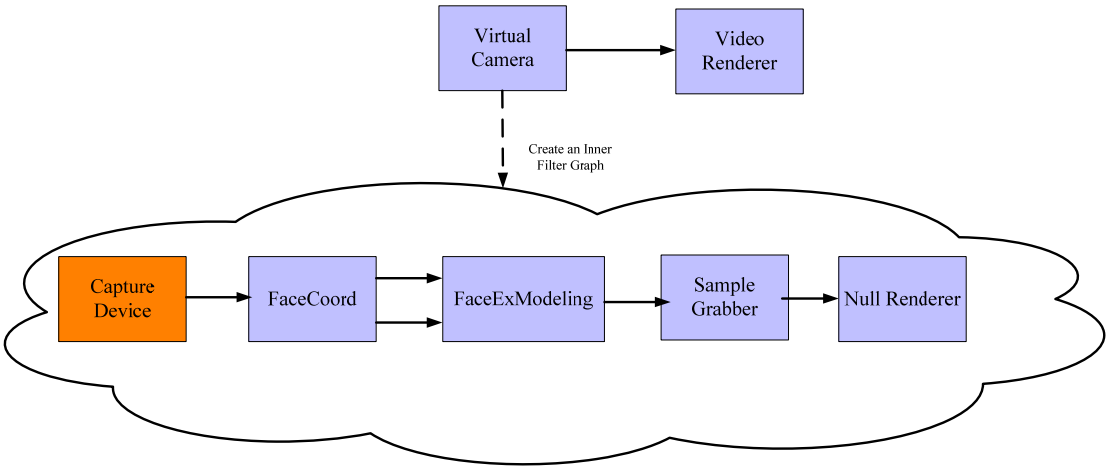

Fig.9.3.The inner graph built by virtual camera

After adding the filters required, we have to connect the filters together through their pins. The output pin in the source filter is connected to the input pin in Face Coordinate filters. The output pin and posOutput pin in Face Coordinate filter are connected to input pin and posInput pin in Facial Expression Modeling filter respectively. The normal output pin in Face Coordinate filter is for passing captured frame buffer and the posOutput pin is for passing face vertex position after face detection. The Sample Grabber and Null Renderer are connected one by one as described in figure 9.3.

The filter graph will not start playing itself after the filter graph is built. Therefore, we have to get the control of this filter graph by querying the IMediaControl interface which is used to start or stop the graph. With the help of IMediaControl, we can run the graph and the source filter will start capturing and passing the frame buffer to the downstream. The downstream filter will work out on the buffer and pass them downstream again until it reaches the Null Renderer.

9.2.2 Facial Expression Modelling filter modification

Before modification of the video stream, the output from the output pin of the Facial Expression Modeling filter is as same as the video source captured by the video capture device. To make the output to be the cartoon character we have modelled, we have modified Facial Expression Modeling Filter. We initialized a Direct3D device to draw the cartoon on a texture according to the facial expression detected. In other words, the Direct3D device draws an opened mouth cartoon when such motion is detected. Since the texture is varied frame by frame, we have to create a dynamically texture to be rendered. After the cartoon character is being rendered by Direct3D, we copied the pixel rendered into the buffer in Facial Expression Filter. The filter then pass such buffer downstream, that is, the buffer with the cartoon character is passed to the sample grabber.

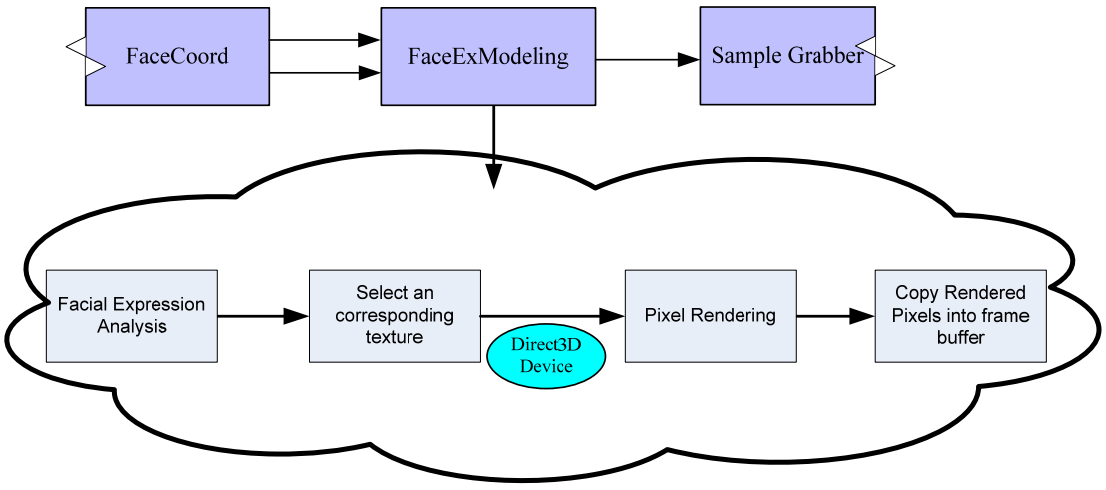

Fig.9.4.The mechanism behinds cartoon rendering to video output

On the other hand, we would like to display the detected face mesh to the user with pop-up window style. The window is used to display the original video frame captured by the video capture source device, with appending the face mesh drawing on the frame. We used Direct3D device to implement the face mesh drawing over the original captured buffer. First, the original video buffer is copied as a Direct3D texture. Then, we draw the face mesh to position with referring to the face position analyzed which is passed from Face Coordinate filter so that user can see whether the face coordinates detected is acceptable. The face mesh and the original video source are combined into a single dynamic texture which is going to be rendered on the pop up window.

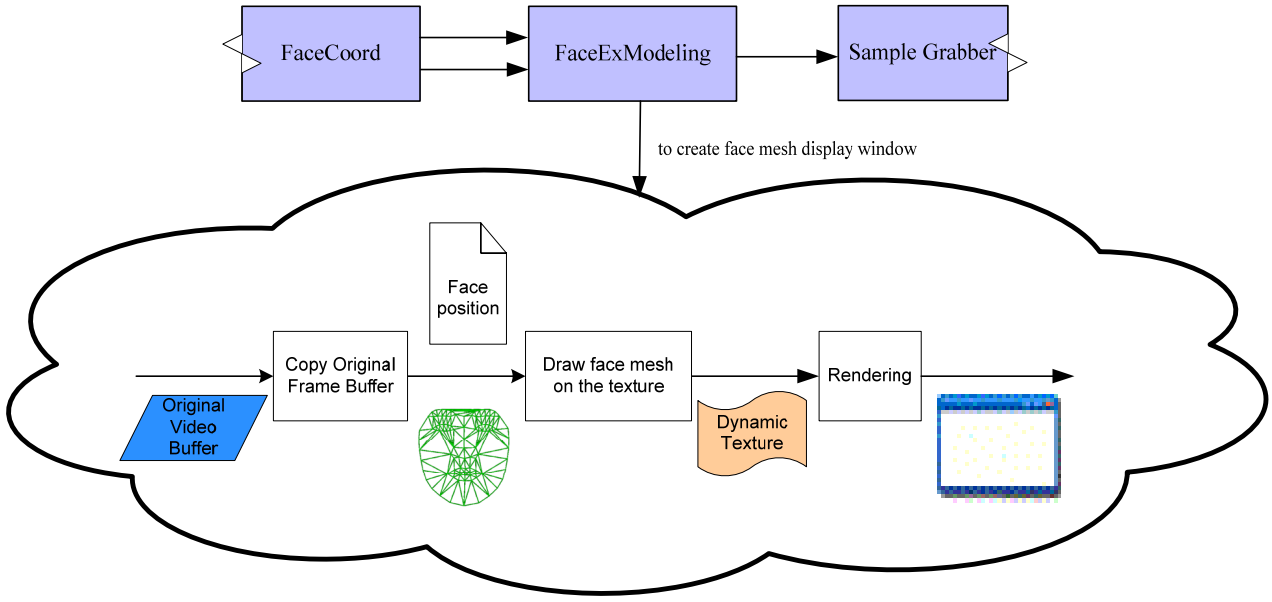

Fig.9.5.The mechanism behinds creating face mesh display window

9.2.3 Sample Grabber

Within the inner filter graph, Sample grabber plays an important role in our virtual camera as it provides an interface for our output filter (Virtual Camera filter) to copy the buffer from the inner filter graph. The sample grabber exposes a member method **GetCurrentBuffer**(long *pBufferSize, long *pBuffer) which retrieves a copy of the buffer associated with the most recent media sample. The buffer passing through the Sample Grabber filter is being copied to the memory pointed by pBuffer pointer and limited by the value pointed by pBufferSize.

We have added sample grabber in the inner filter graph built so it has the accessibility to grab the frame buffer in the filter graph before passing to the Null Renderer. In other words, virtual camera is copying the video output in the inner filter to be its video source input through sample grabber. Virtual Camera source filter has implemented an output pin which has overridden FillBuffer(). This function is to fill the frame buffer with data such that output pin could pass the media sample downstream. This routine is invoked by system in each frame so that the video frame buffer can keep updated in every frame. As a result, the latest buffer grabbed from Sample Grabber in inner filter graph can be refreshed into the buffer in source filter accordance to the frame rate in the inner filter graph.

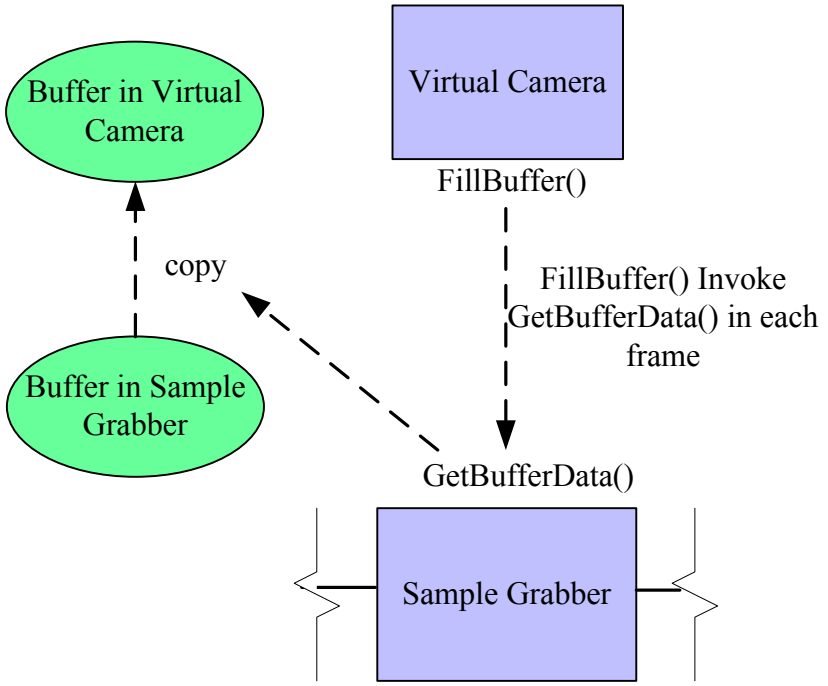

Fig.9.7.Relationship between virtual camera and sample grabber

Figure 9.7 illustrates the relationship between the Virtual Camera and Sample Grabber in inner filter graph. The Sample Grabber is a filter helps copying the frame buffer from inner filter graph to virtual camera.

9.2.4 Miscellaneous

Virtual camera filter registered the inner filter graph it built into Running Object Table (ROT). ROT is a globally accessible look-up table that keeps track of running objects. Objects are registered in the ROT by moniker.

After registration of filter graph in ROT, we can view that filter graph by GraphEdit. In the GraphEdit "File" menu, click "Connect to Remote Graph"... In the "Connect to Graph" dialog box, select the process id of the filter would like to be viewed and click OK. GraphEdit loads the filter graph and displays it. It facilitates us in checking whether the Virtual Camera has built a correct inner filter graph. The graphical view of the filter graph helps us in debugger since it can check the status and properties in filter graph.

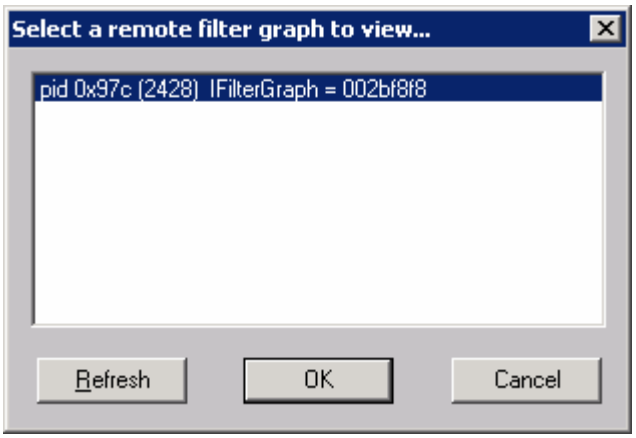

Fig.9.8.The interface to connect remote filter graph in GraphEdit

Nevertheless, registration in ROT is only done in debug project but not in release project because we should not disclose our filter graph to the user as to reduce the infrastructure of the application.

Chapter 10: 3D Face Generator

10.1 Introduction

This module develops a system to approximate the human face and shape. It comprised of two parts, the first part aims to approximate the human face shape by using PCA and rendering it in OpenGL, the second part is to capture the human face as a texture image data. Both the face shape data and the texture image data would be saved for further usage which will be described in the subsequent module.

10.2 FaceLab

The first part is adopted from the face analysis project FaceLab of Zhu Jian Ke, CUHK CSE Ph.D. Student. We have studied how the analysis done and implemented. We would have a brief description of the analysis.

10.2.1 Face analysis

Approximating the face shape is a conventional regression problem and can be decomposed into training and building parts. After the model is trained, it will not be modified again during fitting phase. The whole training phase is made of three steps.

Fig.10.1.Procedure of building the shape model

10.2.2 Data Acquisition

Data gathering is an important step to construct the face model. In 2D face modelling, it usually indicates landmark point in a set of still image. Our facial expression modelling is using this approach and 100 landmarks are sufficient to represent a 2D face image. In 3D dimension, thousands of landmarks are demanded to describe the complex structure of human face. It is therefore difficult to obtain such a great amount of data accurately by hand. Nowadays, 3D face data can be acquired either by equipment such as 3D scanner or computer vision algorithm. Two common 3D data acquisition method is compared below in terms of speed, accuracy, and other characteristics.

	Type	Accuracy	Speed	Registration	Cost	Data
Factorization	Algorithm	Poor	Fast	Auto	Low	Sparse
3D Scanner	Equipment	High	Slow	ICP	High	Dense

ICP – Iterative closet point algorithm

Fig.10.2.Summary of 3D data acquisition method

10.2.3 Data Registration

The next step is to perform the 3D data registration which is going to

normalize the 3D data into same scale with correspondences. It can apply factorization easily to register a sparse data set because the correspondences remain unchanged. However, thousands of landmarks are needed to approximate a good face shape result in a dense data set which is a hard to register. To obtain the correspondences, the most accurate way is to compute 3D optical flow but such algorithm is not available in the public. In addition, those commercial 3D scanners, 3D registration and surface reconstruction software are computed with specified hardware. After comparing those methodologies, it is decided to use software which can generate the human face to import the data directly. 18 human faces are used in the analysis and each has a set of 752 3D vertex data which is large enough to describe the shape of face.

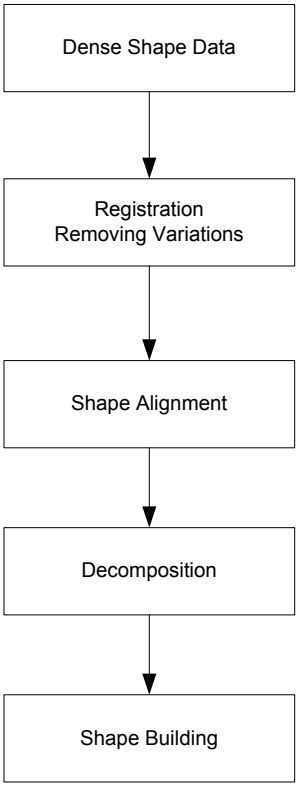

Fig.10.3.Steps to build a shape model

10.2.4 Shape Model Building

It is a main concern to maintain the shape of the object in a statistical shape analysis. A shape here is defined as a geometry data by removing the translational, rotational and scaling components. Since the data is imported from the software, the faces would have a uniform shape representation. The difference between the objects now is the size only. The object containing N vertex data is represented as a matrix below.

$$S = \begin{bmatrix} X & Y & Z \end{bmatrix}$$

The size of the matrix is 3n where each row represents a point of the object. The set of P shapes will form a point cloud in 3N-dimensional space which is a huge domain. It is undesirable to adjust each vertex to build the 3D model, so it is better to change the representation of the object in a manageable form. A conventional principle component analysis (PCA) is performed.

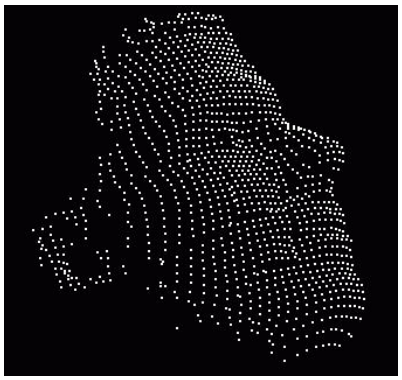

Fig.10.4.A point cloud of a human face

PCA is a technique for simplifying a multidimensional data set into a low dimension. It performs an orthogonal linear transform that transform the data to a new coordinate system such that the new axes would point directions of maximum variation of the point cloud.

In this implementation, the covariance method is used. It first computes the

empirical mean which represent the mean shape along each dimension. It is simply built from the mean of each corresponding point.

$$\overline{S} = \frac{1}{P}\sum_{i=1}^{P} S_i$$

It then calculates the covariance matrix from the objects and itself. The covariance matrix is

$$C = \frac{1}{P}\sum_{i=1}^{P}\left(S_i - \overline{S}\right)\left(S_i - \overline{S}\right)^T$$

The axes of the point cloud are collected from the eigenvectors of the covariance matrix. It is done by computing the matrix of eigenvectors which diagonalizes the covariance matrix. Each column matrix V_i represents one eigenvector.

$$V^{-1}CV = D$$

where D is the eigenvalue matrix of C. The eigenvalue represents the distribution of the objects data's energy among each of the eigenvectors. Finally, the resulted shape model is represented as

$$S = \overline{S} + V_i m_i$$

where m_i are the shape parameters. These parameters give the weights of the new set of axes. Adjusting the value of the shape parameters would generate a new face model which can be computed by

$$m_i = V_i^T (S - \overline{S})$$

Actually, there is unnecessary to keep all the axes of the shape model to create an acceptable face model. A subset of the eigenvector can be selected as the basic vectors. The goal is to determine the percentage of the total variance to be included. Since the eigenvalue represents the variation of the

corresponding axis, the columns of the eigenvector matrix and eigenvalue matrix is sorted in order of decreasing eigenvalue. The first seven is used in the system and achieve a 99% of the total variance.

The resulted data set is outputted as the 3D face mesh data which will be rendered on the screen by using OpenGL.

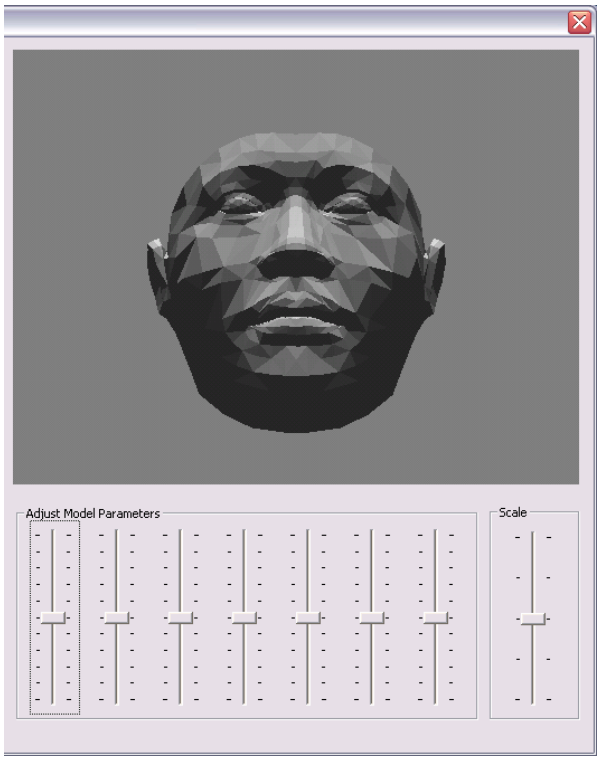

Fig.10.5.A mean mesh is rendered on the screen

The initial parameter is set to generate the mean shape. It can be done by setting all parameters to zero. Seven slider controls are provided to adjust the parameter and a scale slider control is used to adjust the scale of the parameter. The model would be changed immediately to reflect the changes of the slider.

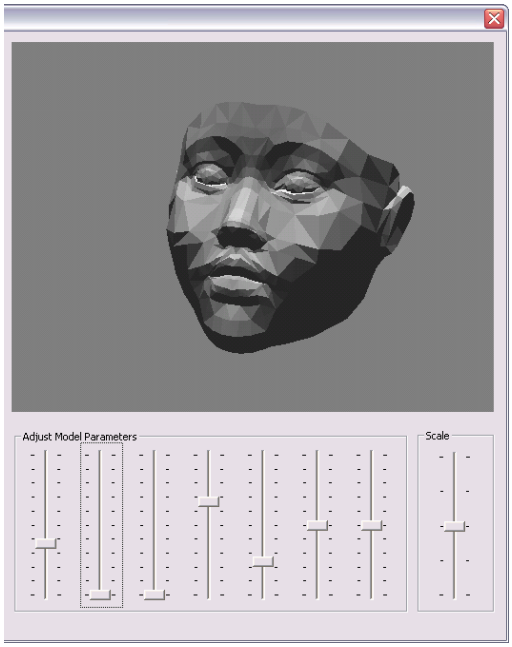

Fig.10.6a.A mesh with normal scale

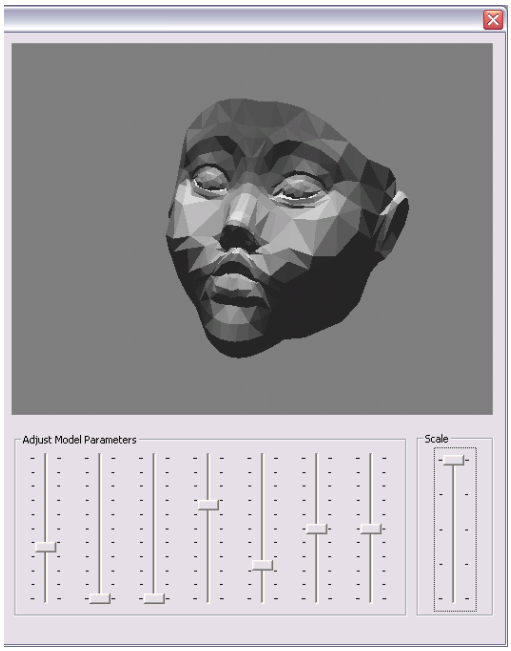

Fig.10.6b.A same mesh with large scale

The human face used in this shape analysis is mainly from adult male resulting in the face shape is tending to a male.

The face model acts as a skeleton of the face. In the next section, we are going to prepare the skin which is a face texture generated from a live video stream.

10.3 Face Texture Generator

We had to build a face texture which is extracted from a video frame of live capturing. The face texture will be used on a human face model rendering. In other words, the face texture retrieved will be mapped to a 3D face model. To maximize the resolution of the texture image that will directly affect the look and feel of the face model, we need to find a suitable way to get as much face data as possible. In our implementation, we had raised three approaches to achieve this goal. They are largest area triangle aggregation, human-defined triangles aggregation and single photo on effect face.

Both largest area triangle aggregation and human-defined triangle aggregation methods will ask the user to capture three photos. The photos are taken in the view of front face, left face and right face. The front face photo is that the user was looking to the front. The left face photo is that the user was turning his head to right so that his or her left face is exposed more in the camera. The right face is similar to that in left face but just the head turning direction. In each photo, user face should be well detected by the Face Coordinate before taking it. Face outline is drawn on the screen output so that user can see whether the face mesh is fit on their face well.

Fig10.7a.Front face photo

Fig.10.7b.Left face photo

Fig.10.7c.Right face photo

10.3.1 Largest area triangle aggregation

In this face texture generation method, we asked user to capture three photos before texture generation. When user invoked the grab function, the Face Generator filter will record down the face coordinates detected and frame buffer at that moment.

There are total 167 triangles on the face mesh. We wanted to create a new texture which is comprised of triangles from three photos captured. Within the same triangle (the area of face), such triangle being selected is based on the area of the triangles as a face mesh triangle detected with a larger area should have a better resolution. A larger triangle contains more number of pixels in it such that the data representing that face portion is in more details.

For each photo taken, there is a set of 100 face coordinates which is detected by Face Coordinates filter. We then select the triangles in face mesh one by one. As each vertex on the face mesh has a unique number to identify them, so each triangle has 3 vertex and they were stored in a file.

In calculation of area of triangles, we have used the same manner that used in competitive area ratio module. By comparing the area of three triangles, we adopt a triangle from a photo which has the largest area in its face mesh.

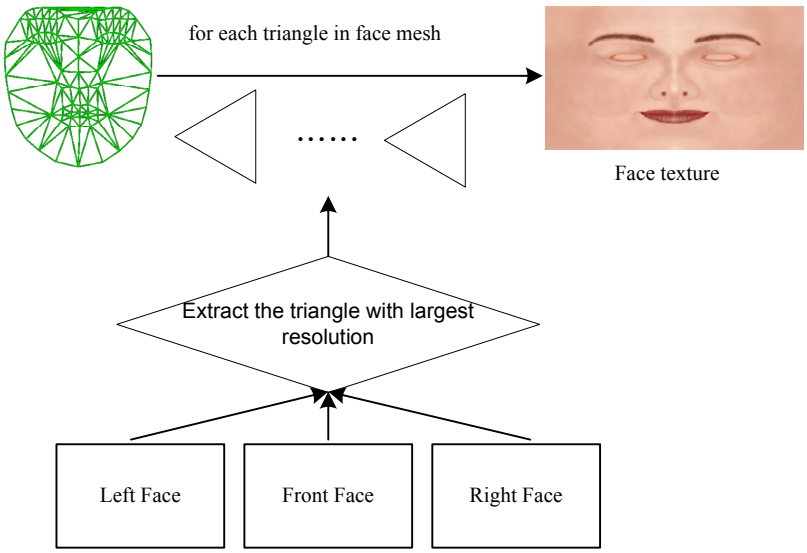

Fig.10.8.Flow of face texture generation

After selecting appropriate face triangle from three photos, a preview face texture is generated. We have done a number of experiments on this face aggregation approach. The results of the experiments may contain some mistakes. The face coordinates sometimes cannot be recognized precisely. Therefore, the face might be fragmented as shown in the following example.

Figure 10.7a, b and c are the front face, left face and right face respectively. Figure10.10 is the face texture result of the face aggregation. In this result, we can see some area of the right eye (on the left hand side of the picture) is duplicated. The reason of this phenomenon is that the triangles comprises the eye is extracted from different photos. The aggregated mouth is fragmented which is not acceptable in face texture. On the other hand, the skin colour on the face looks strange because the brightness on three photos is not the same.

Since we found that the result obtained from this approach is not acceptable, we have abandoned using this approach in face texture generation. The reason of failure of this approach is that we have not considered the face detection defection brought by the Face Coordinate filter

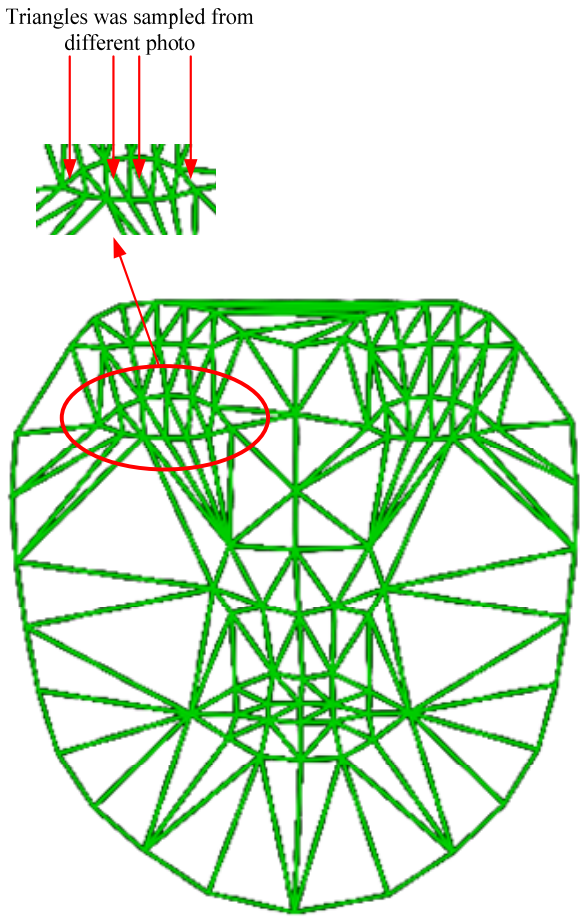

Fig.10.9.The result analysis by a mesh

Fig.10.10.The result of largest area triangle aggregation

10.3.2 Human-defined triangles aggregation

We have modified the triangle selection method into human-defined way such that we judge which photos should contribute a better resolution in a particular triangle. The triangles comprise an organ should be selected from a single picture to eliminate that possibility of fragmentation.

The human-defined selection is illustrated in figure 10.11. The triangles coloured with red uses the pixels on “right face” photo. The triangles coloured with purple adopt the pixels from “left face” photo.

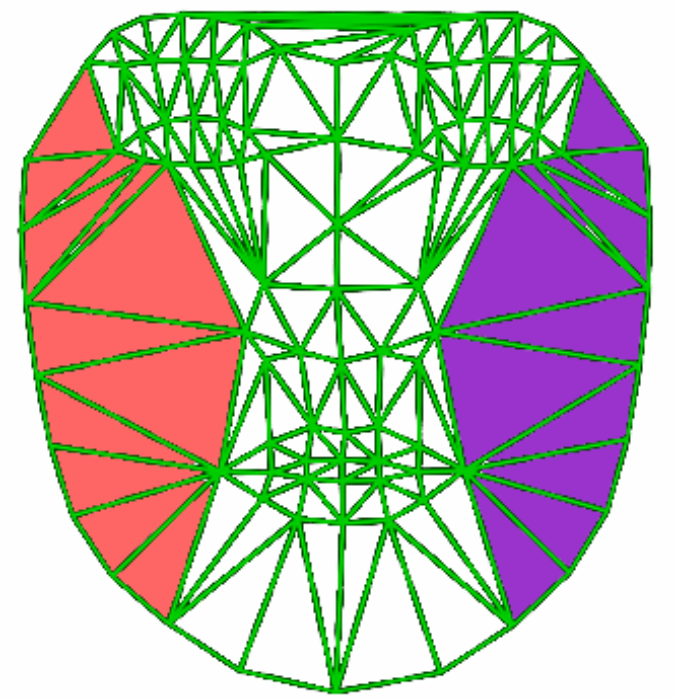

Fig.10.11.It shows the partition using the same sampling photos

In addition, we have redefined our face mesh as shown in figure 10.12. In the mesh, we can see that the vertex on the face mesh are adjusted into new position. This fine tune can increase the number of pixels representing a triangle because the area of the triangles is increased. Consequently, the resolution of the side faces can be increased. We named this newly defined face mesh as Effect Face Mesh.

We can imagine that a human face is painted and rolled up on a paper, and the human face will be painted on the paper which should look very similar to our Effect Face Mesh.

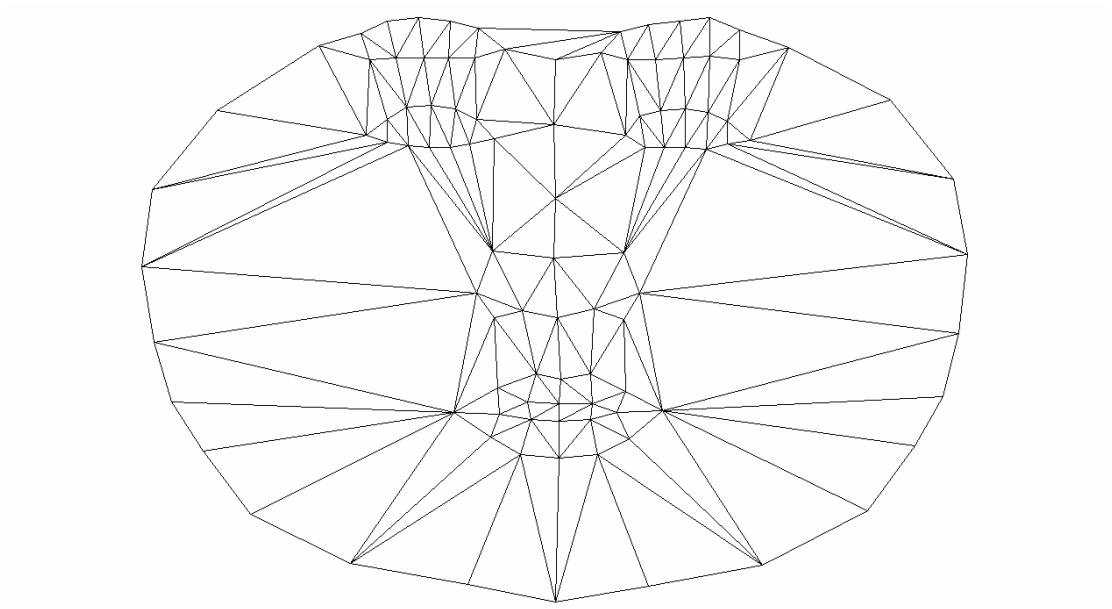

Fig.10.12.Effect face mesh

Figure 10.13 demonstrated the result of this aggregation approach. The result looks much better than the result generated by largest area triangle aggregation. We can see that the eye and mouth fragmentation is eliminated by using this method because we divided the face into three portions instead of picking each fragment from different pictures. The face looks more natural as the adjacent triangles were sampled in the same photos.

Nevertheless, we can see that the aggregated face looks like divided into three parts. It is because the brightness in three photos is different. Although we used the same camera in taking photo, the brightness has a slightly different in different time. In our example, the front face photo, the brightness is slightly darker than the side faces photos. We tried to fine tune the brightness on the photos in programming manner as to maintain a similar brightness value. However, we cannot achieve a satisfactory result as a slight different of brightness between three photos can cause a partition effect already.

Fig.10.13.Result of human-defined triangles aggregation

As we cannot generate a satisfactory texture by using three photos, we decided to use a single photo but with an Effect Face Mesh as mentioned. We found that the result is satisfactory. Figure 10.14 is the result of using a front photo only as source of pixel sampling. There is no more fragmentation in the new result.

Fig.10.14.Result of single photo on effect face

10.3.3 Implementation

The video is comprised of frames. We had to implement an interface which allows user to grab a still image from a live video. In our FaceLab application, it builds an inner filter graph for manipulating Face Coordinate and Face Generator filters. To capture a video frame, we have to ask the filter in the filter graph to save the buffer into file system. Although the Face Generator is instantiated by FaceLab, it still cannot directly control the filter because filter is created in IBaseFilter object. This problem was solved by creating an additional interface in Face Generator filter which allows querying from FaceLab.

Once FaceLab called the frame grabbing interface in Face Generator filter, the filter will write the frame buffer into a file in bitmap format. We save the buffer in bitmap format because bitmap is a file type without compression and similar to our frame buffer except the file header.

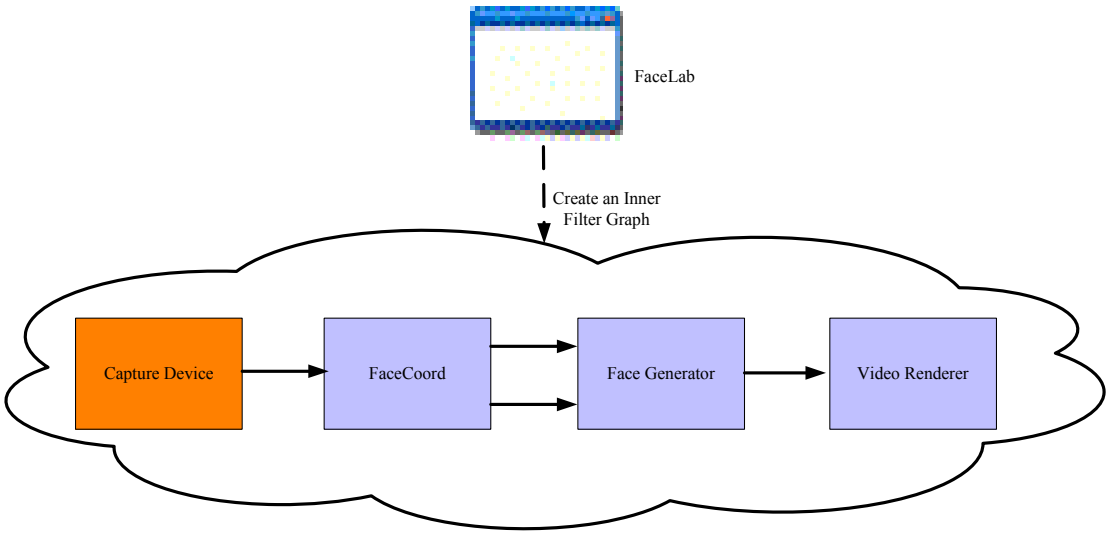

Fig.10.15.The inner filter graph in Face Generator

10.3.4 Face Generator Filter

The implementation of face texture generation is worked out within our Face Generator Filter. Face Generator filter is a DirectShow transform filter which is developed based on Facial Expression Modelling filter so it can be regarded as a derivative of Facial Expression Modelling filter.

In filter graph perspective, Face Generator is downstream filter of Face Coordinate filter. After Face Coordinate filter analyzed the human face, it passes the video frame buffer and detected face coordinates to Face Generator filter. Face Generator filter makes use of the Direct3D to draw the effected face texture. It generates a dynamic texture to hold the frame buffer data and updates the texture coordinate according to the detected face coordinate. The resulted face can be rendered with the geometry defined in the Effect Face Mesh on the screen. All these procedures would be performed frame by frame to show the effect immediately.

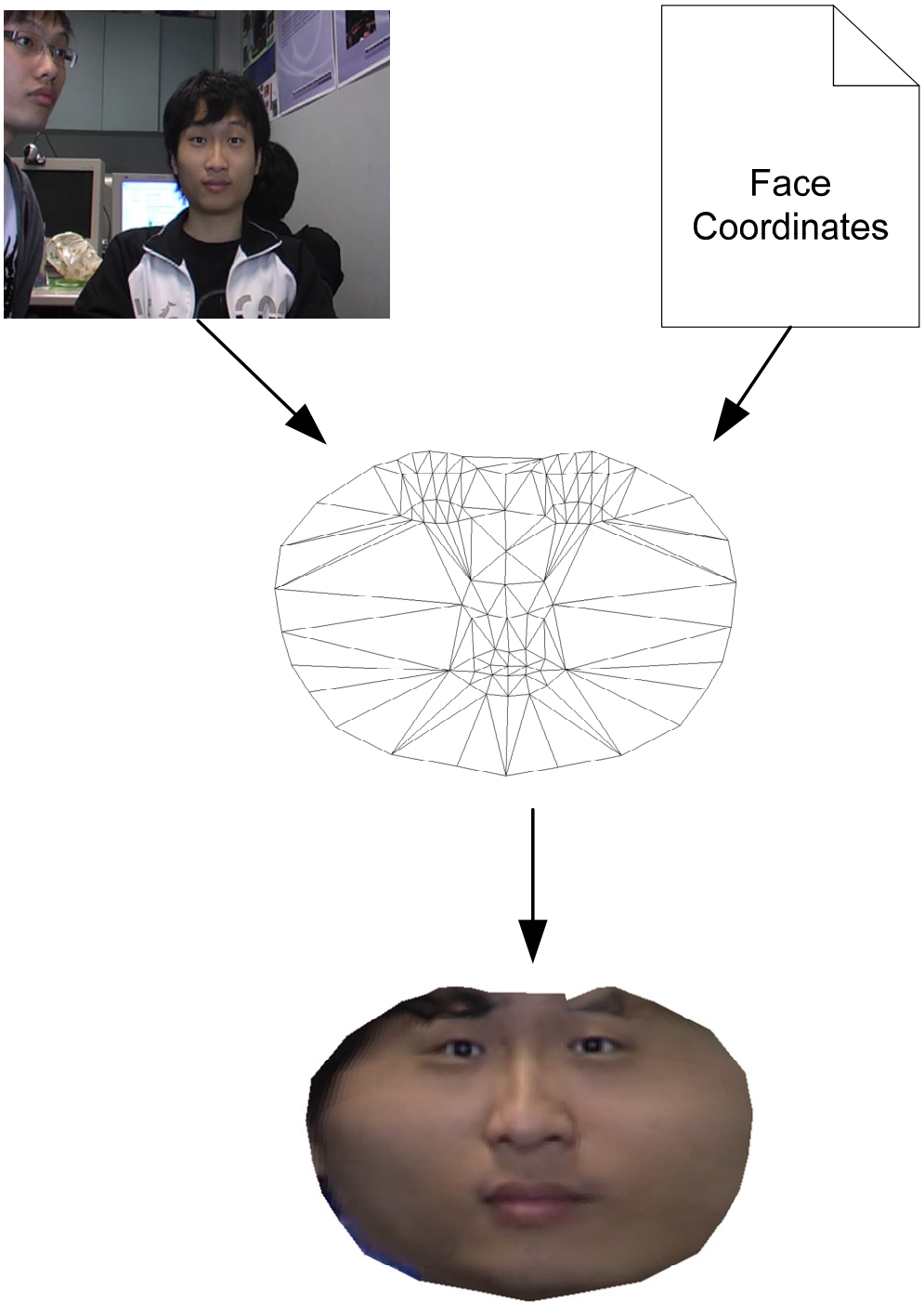

Fig.10.16.Illustration of the face aggregation

10.3.5 Dynamic Texture Generation

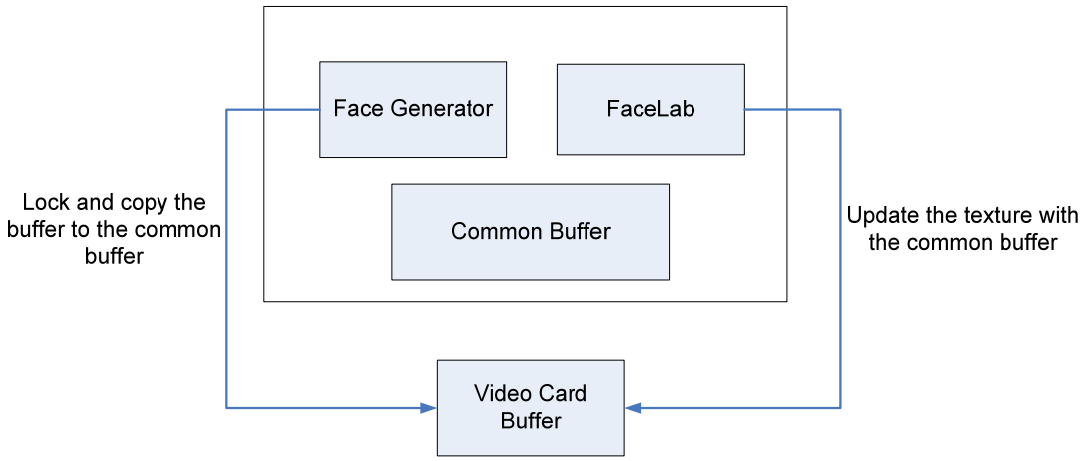

Fig.10.17.Generating a dynamic texture

The skin is now prepared and ready to be used. We need to get back the rendered data from the video display card and pass to the FaceLab. Before we can copy the buffer, we need to lock the video display buffer so that the display card would not alter the content of the buffer until we release the buffer. Face Generator exposes a routine, GetRenderData() which returns the pointer of the common buffer pointing to the Effect Face Texture pixels rendered by Direct3D. The FaceLab can only access the common buffer whenever the Face Generator is not updating the buffer. Since the FaceLab is implemented by OpenGL, it has its own texture buffer in the video display card. When the common buffer content is changed, we need to update the texture buffer to reflect the changes immediately. The result is shown here.

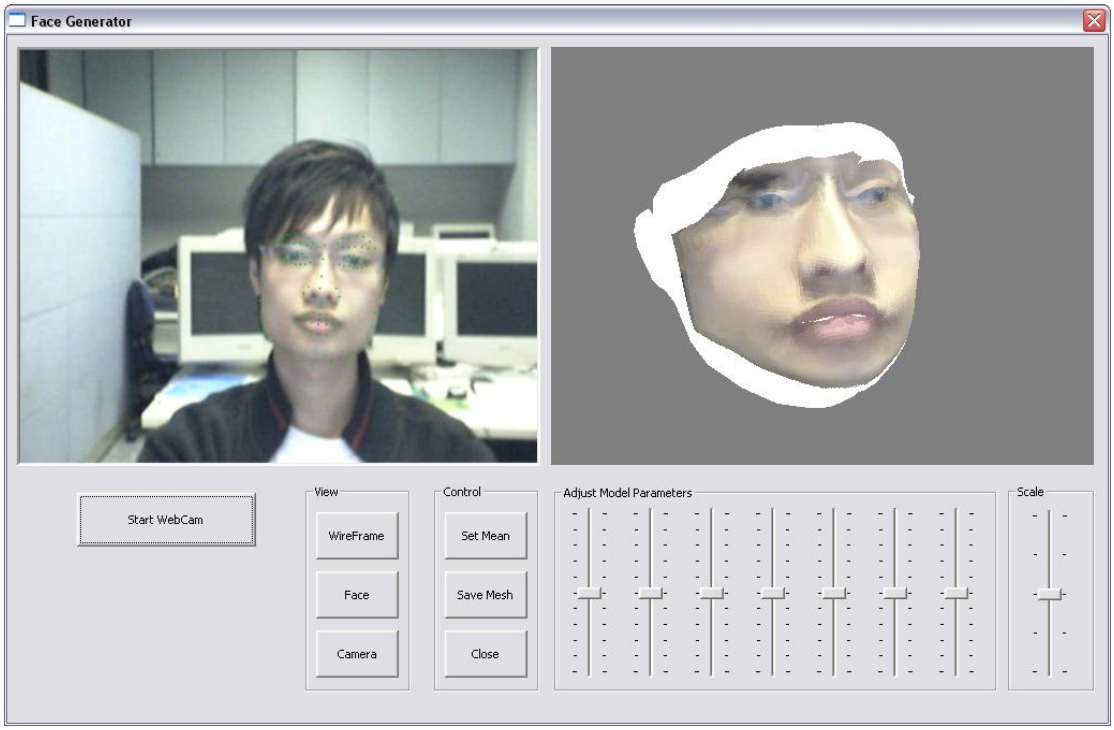

Fig.10.18.The face texture is real-time updated and mapped on the face model

The current view mode is the Camera mode. Other two view modes, WireFrame and Face, are also provided for user to view the changes on the face more easily.

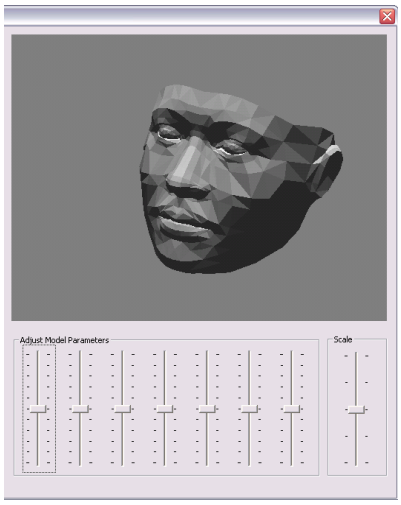

Fig.10.19a. The face view of the model

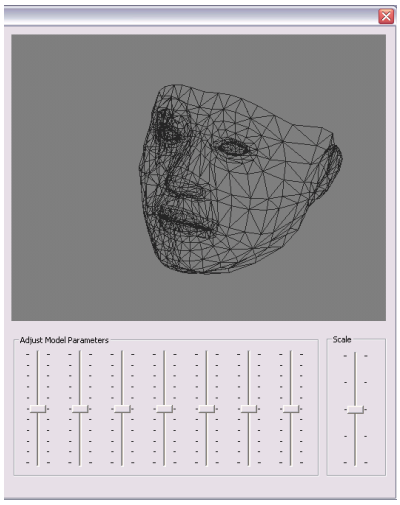

Fig.10.19b.The wire frame of the model

Once you have satisfied the shape of the face and fitted your face on the face model, you can save the face mesh data and the texture image by pressing the Save Mesh

button. The format of the mesh data is defined specifically for our system to use only while the texture image is simply saved in bmp format. In the next chapter, the mesh data would be standardized for the public to use.

Chapter 11: Face Viewer

11.1 Introduction

This module describes the building of three-dimensional appearance model from the saved mesh data and texture image in the pervious module. On top of that, we animate the model to make it more realistic and interesting, and demonstrate the capability of using the model in a game or computer animation. We standardize the mesh data into an industry standard file format that can be made free with any interested body.

We do this by choosing the Direct3D as the platform to build up the model because it is commonly used in the game industry. The mesh data including the vertex geometry and the face triangle data are loaded and saved in the Direct3D mesh. The texture image and the standardized texture geometry are next loaded into program to provide material of the mesh. We can now render the mesh as follows.

Fig.11.1.A mesh is rendered by the Direct3D

11.2 Select the face mesh and texture

We also show the pervious created mesh and texture image. You can simply double click the listed item located at the right hand side of the screen to change the mesh or face image.

Fig.11.2 Select different face mesh and texture to form different combination

11.3 Generate simple animation

In order to make it more realistic and interesting, some facial animations are added to the model. We only provide three animations but it is enough to show our purpose. All the facial expression animation would appear repeatedly.

11.3.1 Looking at the mouse cursor

To begin with, the model eyes is continuously tracked and projected to the mouse cursor. By using the feature point on the face mesh, we can easily find out the eye position. Whenever the mouse cursor is moved, we do update the face geometry in such a way that the two eyes will form a triangle planar with the mouse cursor. It looks like the model is keep looking at the mouse cursor. This also provides an easy way to exam the model within a constricted view angle. Here is some example.

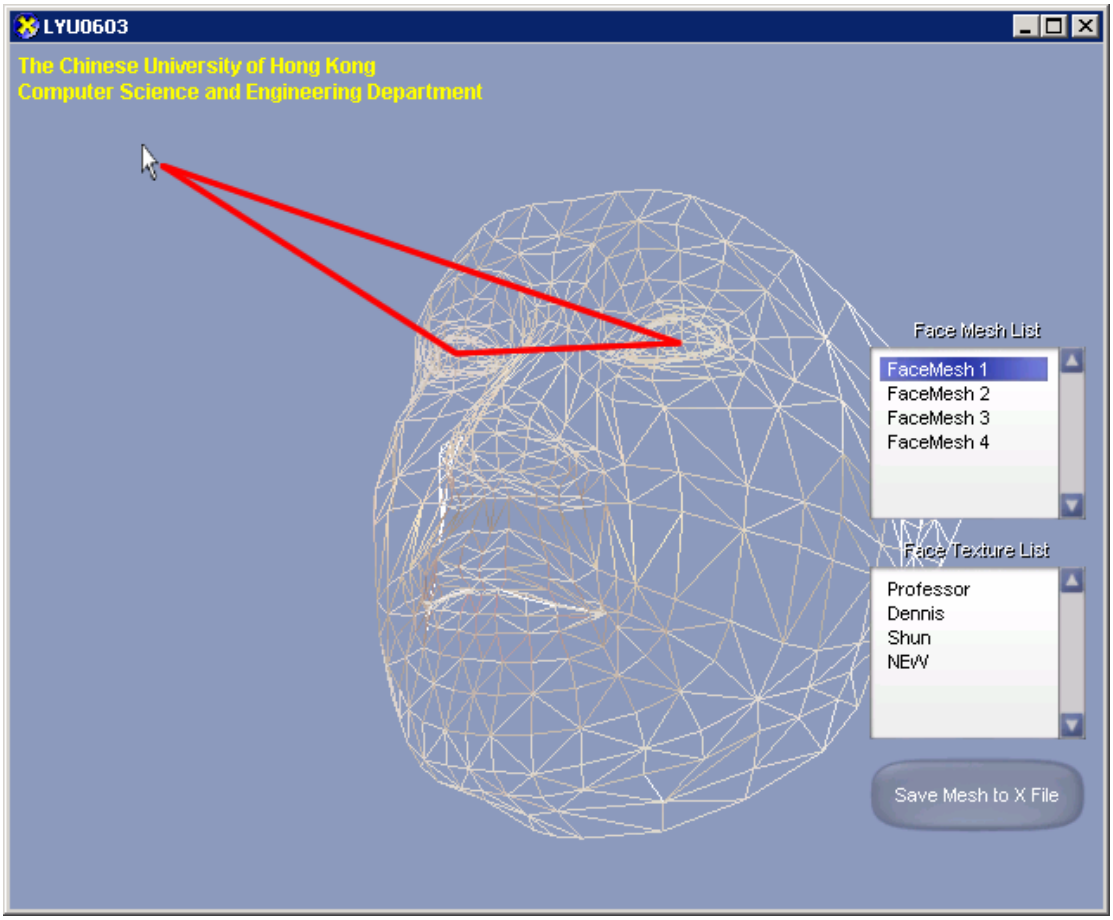

Fig.11.3.A triangle planar is maintained with the eyes and the mouse cursor

Fig.11.4.Moving the mouse cursor can change the view angle of the face model

11.3.2 Eye blinking

The next animation is the eye blinking which is one of the natural movements

on a human face. It is done by adjusting the vertex geometry on the eyelids. We move those eyelids vertex geometry up and down above the eye.

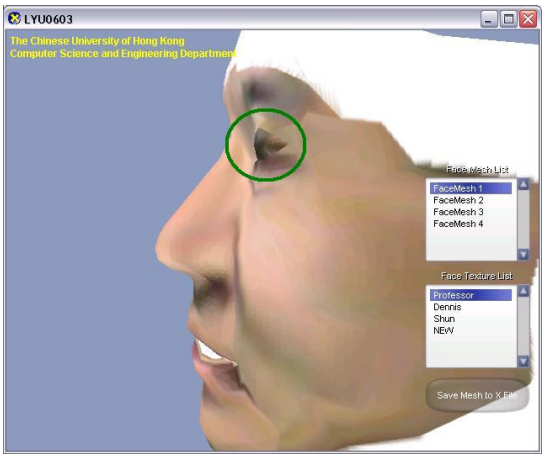

Fig.11.5.The eyebrow is coming out *Fig.11.6.the eyebrow need to move backward*

The eyebrow is also needed to move forward and backward to have a good appearance.

Fig.11.7.The eyes of face model is closed

11.3.3 Smiling

We move on to create a smile animation which is a litter bit complicated since there is a lot of muscle needed to be modified. The cheeks are being pushed

up and wide while the chin is being pulled down. It also has to change the shape of the lip to match up the smile expression.

Fig.11.8.Face model without smile

Fig.11.9face model with smile

11.4 Convert into standard file format

User can save the mesh in .x file format by pressing the Save Mesh to X File button. The system would save the current selected face mesh data into a .z file. The Microsoft DirectX .x file format is not specific to any application. It uses templates that don't depend on how the file is used. This allows .x file format to be used by any other application.

Chapter 12: Difficulties

Before we start working on this project, we both do not have much knowledge in image processing and 3D modelling. We had spent much time on finding the resource and studying related materials.

Our project is built by different SDKs includes DirectShow, Direct3D and OpenGL. However, the documentation on them may not be well written and there is not much programming examples can be found on internet. Therefore, we sometimes have to work out by trial-and-error as to find out the functionality of a method and the meaning of a parameter.

On the other hand, when we think about the algorithms or methods to deal with problems, they usually cannot work perfectly because the factors affecting the result are dynamic and very difficult to control a high accuracy. Therefore, it is hard to make our system becomes a generic application. A solution to a problem may only be suitable on a certain environment (e.g. light condition, different human face). As a result, we have to consider many different testing environments.

Our development is based on Face Coordinate filter. The number of vertex analyzed by this filter is limited. The accuracy of face detection by Face Coordinate filter can significantly affect the result of our facial expression modelling.

Chapter 13: Future Development

Although this project is completed, there is still room for improvement. In this chapter, we are going to discuss on how our project can be future developed in order to maximize the usability of our system.

In virtual camera, the video frame of the cartoon character was sent frame by frame to the other side through the net-meeting software. Nevertheless, the bandwidth may be limited in some circumstances which degrade the performance of our system. To overcome the problem of frame lagging, we can use signal passing instead of frame passing. Signal passing is a way that our system only passes the facial expression detected in form of number to the other side connected. Once other side received the number, it interprets it as facial expression and generate an appropriate model. However, the approach requires other side to install our system as to generate the 3D character.

Chapter 14: Conclusion

To conclude, we have successfully built a facial expression modelling system which can be used in net-meeting application. We also built a face generator which helps game developer to allow user make use of his or face to be the character in a 3D environment.

Facial Expression Modelling system can achieve its goal. The functionality of a webcam can be enriched with our project. It exerted the potential of face detection system with using normal end-user equipment. Face Generator eases 3D face extraction and modelling from webcam can be done in more easily way with the help of face detection.

Throughout this project, we have acquired much knowledge on image processing and 3D graphics and different toolkits like DirectShow, Direct3D and OpenGL. It trained us to be a fast-learner and self-motivated as we have to study the new knowledge by ourselves.

On the other hand, we learned how to think about a problem in more details because many problems cannot be easily solved by an intuitive idea. It wastes much time if we try to implement a wrong algorithm. A good evaluation on the design is important before implementation

Chapter 15: Contribution of Work

This section describes my contribution of work in addition to the knowledge that I have learnt in this project.

Preparation

Starting from the last summer, we have been preparing this project. Our original project is called "Tele-Boxing – Eye-Toy like network game on TeleStation", so the first thing we need to do is to study the video processing. It is recommended to use Microsoft DirectShow as the development platform since this is prevalent video processing technology. However, I and my partner have only a limited knowledge about image processing and even know nothing about the DirectShow. We spent several weeks on studying all those materials.

We then move on to think about how to detect the hand movement in a video stream. But we found out there is no good way to do it. The problem is that hand movements like a punch would require a volume study about the environment. Besides, boxing requires a fast movement detection of the hand. It is not feasible to detect such fast movement in a video stream.

Semester One

After we have considered all the characteristics of our project, we decide to change our topic to "A Generic Real-Time Facial Expression Modelling System" which is also a video analysis project. What we have learnt in the summer can still be applied in, but we have to start over our project.

We adopted to use a face detection filter in DirectShow so that we can focus on the facial expression analysis. Since our analysis relies heavily on the face detection, we first need to implement another filter used to read the detected face coordinates. Based on the nature of the detected face coordinate, we started to develop some algorithms to identify the facial expression. The implementation part of the algorithm is responsible by our partner. I modify the video frame to show the detected face area on the screen.

My main duty in this semester is to model the facial expression to show our analyzed result. I choose to do the modelling on Direct3D which is the same collection of the Microsoft DirectX. I think it would be more compatible to DirectShow application. However, I have no knowledge about Direct3D before this project. It takes me a lot of time to study it. Moreover, I manually modify the cartoon character to show different facial expression and their combination.

Semester Two

We started to build a virtual camera which embeds our Facial Expression Modelling System so that people can use a cartoon character to represent themselves on net-meeting. My partner works on developing the virtual camera. I start to find a way to replace the video frame buffer with our modeling cartoon character. This raises a problem on how to access the video card memory which stores the rendered cartoon character. I finally find a way to tackle the problem.

We then shift to create a 3D human face from the video stream. This comprises of the face model and the face texture. The face model is created by

a statistical face analysis which is adopted from other party. I study the mechanism of such analysis while my partner starts to integrate the face detection and the face model together.

The next thing is to retrieve the human face and mapped to the face model. My partner finds a suitable way to extract the human face from the video stream to maximize the resolution of the face. I responsible for mapping the face texture on the face model. The face model is built from OpenGL. I, therefore, have to spend some time to learn the OpenGL in order to find out the way on mapping the texture. Meanwhile, I have created a Direct3D application to show the resulted human face model. The application would show some simple animation with the face model and let the user to save the face model for other client application to use.

Conclusion

Doing this project really challenges me because I need to handle different kinds of platform and mix them together. Those platforms are really new to me. It takes a large proportion of time in doing this project. However, this would be a very valuable experience to learn the knowledge and develop the system. My interest in 3D modeling has increased a lot. We also encounter many problems in the project. It would be very useful in future job on learning those problem solving skills. But a more important thing I have learnt in this project: without my partner, I definitely cannot get through this job. I would like to take this as an opportunity for me to thanks for everything with my partner, Dennis.

Chapter 16: Acknowledgement

We would like to express our heartfelt thanks to our project supervisor, Professor Michael R. Lyu. Professor Lyu has given us many useful advices and supervision on the project. He is considerate of our workload and has reminded us the importance of good scheduling. He has provided us enough resources to complete this project.

Secondly, we would like to thank for our project manager, Mr. Edward Yau, who has given us many innovative ideas and suggestions in our project.

Thirdly, we would also like to thank for research staff, Un Tsz Lung, who helped us on solving the problem in implementation

Last but not least, we must thank for the author of Face Coordinate filter and FaceLab, Zhu Jian Ke.

Chapter 17: Reference

[1] Jianke Zhu, Steven C.H. Hoi, Edward Yau and Michael R. Lyu, "Automatic 3D Face Modeling Using 2D Active Appearance Models". Proceedings of the 13th Pacific Conference on Computer Graphics and Applications, Macau, China, October 12-14, 2005

[2] Jianke Zhu, Steven C.H. Hoi, Edward Yau and Michael R. Lyu, "Real-Time Non-Rigid Shape Recovery via Active Appearance Models for Augmented Reality". Proceedings 9th European Conference on Computer Vision (ECCV2006), Graz, Austria, May 7 - 13, 2006.

[3] Yuwen Wu, Hong Liu, and Hongbin Zha, "Modeling facial expression space for recognition". Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on 2-6 Aug. 2005 Page(s):1968 – 1973.

[4] Hongcheng Wang; and Ahuja, N., "Facial expression decomposition". Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on 13-16 Oct. 2003 Page(s):958 - 965 vol.2.

[5] Yeasin M., Bullot B. and Sharma R., "Recognition of facial expressions and measurement of levels of interest from video". Multimedia, IEEE Transactions on Volume 8, Issue 3, June 2006 Page(s):500 – 508.

[6] Aleksic, P.S. and Katsaggelos, A.K., "Automatic facial expression recognition using facial animation parameters and multistream HMMs". Information Forensics and Security, IEEE Transactions on Volume 1, Issue 1, March 2006 Page(s):3 – 11.

[7] Chan-Su Lee and Elgammal, A., "Nonlinear Shape and Appearance Models for Facial Expression Analysis and Synthesis". Pattern Recognition, 2006. ICPR 2006. 18th International Conference on Volume 1, 20-24 Aug. 2006 Page(s):497 – 502.

[8] Chandrasiri, N.P., Naemura, T. and Harashima, H., "Interactive analysis and synthesis of facial expressions based on personal facial expression

space". Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on 17-19 May 2004 Page(s):105 – 110.

[9] Yabuki, N., Matsuda, Y., Fukui, Y. and Miki, S., "Region detection using color similarity". Circuits and Systems, 1999. ISCAS '99. Proceedings of the 1999 IEEE International Symposium on Volume 4, 30 May-2 June 1999 Page(s):98 - 101 vol.4.

[10] Uysal, M. and Yarman-Vural, F.T., "A fast color quantization algorithm using a set of one dimensional color intervals". Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on Volume 1, 4-7 Oct. 1998 Page(s):191 - 195 vol.1.

[11] Chew Keong Tan and Ghanbari, M., "Using non-linear diffusion and motion information for video segmentation". Image Processing. 2002. Proceedings. 2002 International Conference on Volume 2, 22-25 Sept. 2002 Page(s): II-769 - II-772 vol.2.

[12] Mojsilovic, A. and Jianying Hu, "A method for color content matching of images". Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on Volume 2, 30 July-2 Aug. 2000 Page(s):649 - 652 vol.2.

[13] Jian Cheng, Drue, S., Hartmann, G. and Thiem, J., "Efficient detection and extraction of color objects from complex scenes". Pattern Recognition, 2000. Proceedings. 15th International Conference on Volume 1, 3-7 Sept. 2000 Page(s):668 - 671 vol.1.

[14] Unsang Park and Anil K. Jain, "3D Face Reconstruction from Stereo Video". Computer and Robot Vision, 2006. The 3rd Canadian Conference on 07-09 June 2006 Page(s):41 – 41.

[15] Yuankui Hu, Ying Zheng and Zengfu Wang, "Reconstruction of 3D face from a single 2D image for face recognition". Visual Surveillance and Performance Evaluation of Tracking and Surveillance, 2005. 2nd Joint IEEE International Workshop on 15-16 Oct. 2005 Page(s):217 – 222.

[16] Ansari, A.-N. and Abdel-Mottaleb, M., “3D face modeling using two views and a generic face model with application to 3D face recognition”. Proceedings. IEEE Conference on Advanced Video and Signal Based Surveillance, 2003. 21-22 July 2003 Page(s):37 – 44.

[17] KyoungHo Choi and Jenq-Neng Hwang, “A real-time system for automatic creation of 3D face models from a video sequence”. Acoustics, Speech, and Signal Processing, 2002. Proceedings. (ICASSP '02). IEEE International Conference on Volume 2, 2002 Page(s):2121 – 2124.

[18] Rozinaj, G. and Mistral, F.-L., “Facial features detection for 3D face modeling”. Industrial Technology, 2003 IEEE International Conference on Volume 2, 10-12 Dec. 2003 Page(s):951 - 954 Vol.2.

[19] Rudomin, I., Bojorquez, A. and Cuevas, H., “Statistical generation of 3D facial animation models”. Shape Modeling International, 2002. Proceedings 17-22 May 2002 Page(s):219 – 226.

[20] Sako, H. and Smith, A.V.W., “Real-time facial expression recognition based on features' positions and dimensions”. Pattern Recognition, 1996, Proceedings of the 13th International Conference on Volume 3, 25-29 Aug. 1996 Page(s):643 - 648 vol.3.

[21] Kanade, T., Cohn, J.F. and Yingli Tian, “Comprehensive database for facial expression analysis”. Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on 28-30 March 2000 Page(s):46 – 53.

[22] Youngsuk Shin, Shin Sooyong Lee, Chansup Chung and Yillbyung Lee, “Facial expression recognition based on two-dimensional structure of emotion”. Signal Processing Proceedings, 2000. WCCC-ICSP 2000. 5th International Conference on Volume 2, 21-25 Aug. 2000 Page(s):1372 - 1379 vol.2.

[23] Gokturk, S.B., Bouguet, J.-Y., Tomasi, C. and Girod, B., “Model-based face tracking for view-independent facial expression recognition”. Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on 20-21 May 2002 Page(s):272 – 278.

[24] Abboud, B. and Davoine, F., “Bilinear factorisation for facial expression analysis and synthesis”, Vision, Image and Signal Processing, IEE Proceedings-Volume 152, Issue 3, 3 June 2005 Page(s):327 – 333.

[25] Ma, L., Xiao, Y., Khorasani, K.and Ward, R.K., “A new facial expression recognition technique using 2D DCT and k-means algorithm”. Image Processing, 2004. ICIP '04. 2004 International Conference on Volume 2, 24-27 Oct. 2004 Page(s):1269 - 1272 Vol.2.

[26] Xiaoxu Zhou, Xiangsheng Huang, Bin Xu and Yangsheng Wang, “Real-time facial expression recognition based on boosted embedded hidden Markov model”. Image and Graphics, 2004. Proceedings. Third International Conference on 18-20 Dec. 2004 Page(s):290 – 293.

[27] Zhan Yong-zhao, Ye Jing-fu, Niu De-jiao and Cao Peng, “Facial expression recognition based on Gabor wavelet transformation and elastic templates matching”. Image and Graphics, 2004. Proceedings. Third International Conference on 18-20 Dec. 2004 Page(s):254 – 257.

[28] Sung Uk Jung, Do Hyoung Kim, Kwang Ho An and Myung Jin Chung, “Efficient rectangle feature extraction for real-time facial expression recognition based on AdaBoost”. Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on 2-6 Aug. 2005 Page(s):1941 – 1946.

[29] Jaewon Sung, Sangjae Lee and Daijin Kim, “A Real-Time Facial Expression Recognition using the STAAM”. Pattern Recognition, 2006. ICPR 2006. 18th International Conference on Volume 1, 20-24 Aug. 2006 Page(s):275 – 278.

[30] Abboud, B. and Davoine, F., “Appearance factorization based facial expression recognition and synthesis”. Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on Volume 4, 23-26 Aug. 2004 Page(s):163 - 166 Vol.4.

[31] Ansari, A.-N. and Abdel-Mottaleb, M., “3D face modeling using two orthogonal views and a generic face model”. Multimedia and Expo, 2003.

ICME '03. Proceedings. 2003 International Conference on Volume 3, 6-9 July 2003 Page(s):III - 289-92 vol.3

[32] Xiangyang Zhao and Limin Du, “Feature points adjusting based realistic 3D face synthesizing”. Intelligent Multimedia, Video and Speech Processing, 2004. Proceedings of 2004 International Symposium on 20-22 Oct. 2004 Page(s):254 – 257

[33] Chaumont, M. and Beaumesnil, B., “Robust and real-time 3D-face model extraction”. Image Processing, 2005. ICIP 2005. IEEE International Conference on Volume 3, 11-14 Sept. 2005 Page(s):III - 461-4

[34] Chen, Q. and Medioni, G., “Building human face models from two images”. Multimedia Signal Processing, 1998 IEEE Second Workshop on 7-9 Dec. 1998 Page(s):117 – 122

[35] In Kyu Park, Hui Zhang, Vezhnevets, V. and Heui-Keun Choh, “Image-based photorealistic 3-D face modeling”. Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on 17-19 May 2004 Page(s):49 – 54

[36] Buyukatalay, S., Halicl, U., Akagunduz, E., Ulusoy, I. and Leblebicioglu, K., “3D Face Modeling using Multiple Images”. Signal Processing and Communications Applications, 2006 IEEE 14th 17-19 April 2006 Page(s):1 – 4

[37] Zhong Xue, Li, S.Z. and Eam Khwang Teoh, “Facial feature extraction and image warping using PCA based statistic model”. Image Processing, 2001. Proceedings. 2001 International Conference on Volume 2, 7-10 Oct. 2001 Page(s):689 - 692 vol.2