3D FACE RECONSTRUCTION USING STEREO VISION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MEHMET DİKMEN

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2006

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Dr. İsmet Erkmen
Head of the Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Prof. Dr. Uğur Halıcı
Supervisor

Examining Committee in Charge:

Prof. Dr. Kemal Leblebicioğlu	(METU, EE)	_____
Prof. Dr. Uğur Halıcı	(METU, EE)	_____
Prof. Dr. Hayri Sever	(Başkent Univ, CENG)	_____
Asst. Prof. Dr. İlkay Ulusoy	(METU, EE)	_____
Asoc. Prof. Dr. Gözde Bozdağı Akar	(METU, EE)	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: Mehmet Dikmen

Signature :

ABSTRACT

3D FACE RECONSTRUCTION USING STEREO VISION

Dikmen, Mehmet

M.S., Department of Electrical and Electronics Engineering

Supervisor: Prof. Dr. Uğur Halıcı

September 2006, 80 pages

3D face modeling is currently a popular area in Computer Graphics and Computer Vision. Many techniques have been introduced for this purpose, such as using one or more cameras, 3D scanners, and many other systems of sophisticated hardware with related software. But the main goal is to find a good balance between visual reality and the cost of the system. In this thesis, reconstruction of a 3D human face from a pair of stereo cameras is studied. Unlike many other systems, facial feature points are obtained automatically from two photographs with the help of a dot pattern projected on the object's face. It is seen that using projection pattern also provided enough feature points to derive 3D face roughly. These points are then used to fit a generic face mesh for a more realistic model. To cover this 3D model, a single texture image is generated from the initial stereo photographs.

Keywords: 3D Face Reconstruction, stereo cameras, structured light

ÖZ

STEREO GÖRME İLE 3 BOYUTLU YÜZ GERİÇATIMI

Dikmen, Mehmet

Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Uğur Halıcı

Eylül 2006, 80 sayfa

3B yüz modelleme, günümüzde, bilgisayar grafikleri ve bilgisayarla görme alanlarının her ikisinde de popüler bir alan olarak karşımıza çıkmaktadır. Bu konuda; 1 veya daha fazla kamera, 3B tarayıcı ve karmaşık donanım ile beraberindeki yazılımları kullanan çok sayıda yöntem tanıtılmıştır. Fakat asıl amaç, görsel gerçeklikle sistemin maliyeti arasında iyi bir denge kurmaktır. Bu tezde, iki stereo kamera kullanarak 3B insan yüzü geriçatımına çalışılmıştır. Birçok sistemin aksine; yüzdeki nitelikler, iki fotoğraftan, yapısal ışık kullanarak otomatik olarak çıkarılmıştır. Ayrıca yapısal ışık kullanımının, 3B yüzü, kabaca elde etmeye yetecek yeterli yüz nitelik noktaları oluşturabildiği görülmüştür. Bu noktalar, daha sonra, daha gerçekçi bir model oluşturmak için genel bir 3B modelin değiştirilmesi amacıyla kullanılmıştır. Bu 3B modeli kaplamak için de başlangıçta çekilen fotoğraflardan elde edilen bir doku resmi kullanılmıştır.

Anahtar Kelimeler: 3B yüz geriçatımı, stereo kamera, yapısal ışık

ACKNOWLEDGEMENT

I would like to express my deepest gratitude and appreciation to my supervisor Prof. Dr. Uğur Halıcı who inspired, encouraged and supported me at all levels of this study.

I would like to thank Dr. İlkay Ulusoy, Tolga İnan and Soner Büyükatalay whose support and suggestions made great contributions to this work.

The greatest thanks go to my family members for their infinite support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

2D: Two dimensional

3D: Three dimensional

RGB: Red Green Blue

RBF: Radial Basis Function

ICP: Iterative Closest Point

CHAPTER 1

INTRODUCTION

Reconstruction of human faces is necessary in order to generate human face models looking like as real as possible. This process requires a translation from two dimensional (2D) spaces to three dimensional (3D) spaces. In this case, face photographs form the 2D space, and 3D space is a 3D face model which consists of a mesh generated by the 3D coordinates of its vertices and a texture image that wraps the mesh.

3D face modeling has been currently receiving a lot of attention among the Computer Vision and Computer Graphics communities and is a thriving research field that can yield to various applications such as virtual reality, animation, face recognition, etc… [1], [3], [5], [11], [15], [18]. In all these applications, the reconstructed face needs to be compact and accurate, especially around significant areas like nose, the mouth, the orbits, etc… [19]. This is why reconstruction of a human face is a much harder task than reconstruction of any other solid shape.

While making an original-like face model is a very hard task because of its complex structure, people did not give up. As the computer technology grows day by day, some people are getting used to believe that one day every behavior of a human will be represented and translated to a model or application by computers. In fact, when we talk about recognition for example, we are very successful about distinguishing

other people by just looking at their faces. We can also detect most of the emotions of a human because emotions are reflected to faces as expressions so that we are able to know if one is sad, happy, angry, etc…

Computer games can be shown as a good example which needs realistic human face models. People had to use their imaginations much more than they use today while playing computer games, 10-15 years ago. But now, the technology introduces a great virtual reality so that people don't need to use their imaginations much.

1.1 Face Reconstruction Techniques

Reconstruction of a 3D object requires, first of all, gathering the 3D information about that object. This process is called "data acquisition". It is the fundamental part of the reconstruction process and also has a very important role in computer vision applications. After an accurate data acquisition, a registration is needed to fit and deform a generic face model with that data to complete reconstruction. So dividing the whole process in two is necessary.

1.1.1 Range Data Acquisition

The determination of a 3D object can be obtained by using either a single image or multiple images. The 3D structure information is useful for object recognition, part

inspection and robot guidance tasks, etc. There are four major approaches to acquire the 3D information [17], [23]:

- Passive monocular approaches,
- Passive binocular approaches,
- Active monocular approaches,
- Active binocular approaches.

Generally speaking, in the first two methods the object features (e.g., points, lines or contours) used for the 3D object structure recovery are imbedded in the rather complicated scene and are not easy to extract from the scene. Quite often, additional information about the objects is required. For instance, object reflectance model or object surface of a regular pattern is assumed in some passive monocular methods, and the correspondence information between object features and image features is needed in most of the passive binocular methods. This additional information is either not available or hard to obtain [17].

In the active monocular methods using the "time of flight" technique, the depth determination is generally not very accurate if the returned beam is weak or noisy [21].

For the active binocular method, triangulation is used to compute the depth information, but they also suffer from the possible depth inaccuracy [21]. Besides, the triangulation technique determines the range data on a point-by-point basis, so it is a slow process; the object must remain still during the whole process.

To reduce the preprocessing time required for data acquisition, some active sensing methods apply the grid coding technique [10], [24]. In these methods a set of parallel stripes or a crossing grid pattern is projected onto the object to produce a spatial encoded image for analysis. For instance in [10], this technique is applied to derive the normal vector of each visible polyhedral face of an object and the depth parameter of the face equation based on the given dimensions of the grid on the code plate.

A more concrete classification for the existing range data acquisition techniques can be done as follows:

- **Stereo acquisition technique:** Two or more cameras that are positioned and calibrated are employed to acquire simultaneous snapshots of the subject [1], [12], [19], [20]. The depth information for each point can be computed from geometrical models and by solving a correspondence problem. This method has the lowest cost and highest ease of use.
- **Structural (or Structured) light technique:** This technique involves a light pattern that is projected on the face of the object, where the distortion of the pattern reveals the depth information [6], [7], [10]. For instance, assume that a stripe pattern is projected on the face. There will be some curved lines since the surface of the face is not flat at every point. These curvatures are used to determine the depth information of places where they lie. This setup is relatively fast, cheap, and allows a single standard camera to produce 3D and texture information.

- **Laser sensor:** This technique is typically more accurate, but also more expensive and slower to use [14], [15], [18]. The acquisition of a single 3D head scan can take more than 30 seconds which is a restricting factor for the deployment of laser based systems.

Actually, data acquisition is all about feature point determination. This is nearly the half of the whole process. But technically, this is the hardest and the most important phase, since human faces differ from each other by their individual features.

Generally, feature points are detected by two ways:

- Manual marking or,
- Automatically.

Correct localization of the feature points is crucial for reconstruction, so that manual marking methods suffer from the sensitivity of the user assistance [9], [16]. But if the main goal is different or does not vary much on this problem, it is quite useful. For instance in [16], the aim is to establish a dense correspondence between each faces' points in order to extract the eigenvectors of Point Distribution Models to learn the shape variations present in their training set for recognition. [9] is another example for manual feature point marking. They use a single image to estimate 3D shape and texture and do not use structured light so that manual marking shall be called a necessity.

Generally, if one would like to detect feature points automatically by the stereo vision technique, he/she must perform the calibration of cameras appropriately. In

[20], from a calibrated pair of stereo pair of face, a disparity map is computed with a correlation algorithm. The produced dense disparity map is actually a depth map. It is then used to calculate 3D points which form an organized 3D point cloud to become a face mesh. In this case, all pixels except the background within the face images may be thought as feature points. Background selection is also important; since it may lead many difficulties during the computation of depth map.

Another example for detecting feature points automatically is done with a frontal and a profile view of a person's face [1]. The algorithm starts by computing the 3D coordinates of automatically extracted facial feature points. The feature extraction starts by detecting the centers of the eyes and the mouth in the frontal view. Then it detects the eyes and the mouth by building two likelihood maps; the skin likelihood map and the mouth likelihood map. This procedure finds the locations of 12 feature points (4 for each eye and 4 for mouth). The nose tip is located from the profile view by detecting edges easily, since nose tip has the minimum Z distance to the origin in this view. From this point in both views the remaining feature locations are completed by finding the corresponding of the existing ones in other view. After calculating the 3D coordinates of these locations, they are used to deform a 3D generic face model to obtain 3D face model for that person.

1.1.2 Face Registration

Face registration can be declared as bringing the entire default 3D model vertices as close as possible to the corresponding 3D coordinates of the feature points calculated

from images. The reverse is also possible, i.e., bringing the calculated 3D points close to the default 3D model. This is referred especially when there are many calculated 3D points forming a 3D point cloud enough to represent the individual's face roughly. This matching requires rotating, translating and scaling of the one to be moved closer to the other.

With regard the use of 3D in registration, methods can be categorized into two separate groups:

1. **3D to 3D registration:** Matching is done between two 3D point clouds. One is a base mesh, and other is the 3D data generated in anyway after the data acquisition process.

   A technique that combines an Active Shape Model with Iterative Closest Point (ICP) [4] method is presented by Hutton et al. [16]. A dense model is built first by aligning the surfaces using a sparse set of hand-placed landmarks, then spline warping is used to make a dense correspondence with a base mesh. A 3D point distribution model is then built using all mesh vertices. The technique is used to fit the model to new faces.

   Point cloud is the most primitive 3D representation for faces, and it is difficult to work with. Achermann and Bunke employ Hausdorff distance for matching the point clouds [2]. They use a voxel discretization to speed up matching, but it causes some information loss.

   When the data are in point cloud representation, ICP is the most widely used registration technique [4]. The similarity of two point sets that is calculated at each iteration of the ICP algorithm is frequently used in point cloud-based face recognizers.

2. **3D to 2D registration:** These approaches exploit a 3D morphable model which is used to fit 2D data [8], [9]. In an analysis-by-synthesis loop the algorithm iteratively creates a 3D textured mesh, which is then rendered back to the image and the parameters of the model updated according to the differences. This approach easily separates texture and shape and is used in 3D shape assisted 2D recognition.

1.2 Work Summary

In this study, a combination of stereo and structural light techniques is used to obtain 3D information. In fact, structured light is only used to define the projection patterns. The reason why we used a pattern is to obtain more feature points from the subject's face. These points will not be limited to be located at the eyes, nose, mouth, ears, etc. We can also obtain many points located at the cheeks, forehead, etc.

The process consists of some basic steps that are explained below:

1. A pair of stereo photographs of the human face to be reconstructed is captured by again a pair of stereo cameras. These cameras are positioned in parallel to the floor having the same focus and capturing resolution. The individual is positioned between them. The distance between the subject and cameras are determined manually so that the face of the subject can be seen from the both cameras completely.

2. During the capturing process, a grid pattern containing various number of dots is projected onto the individual's face so that the captured photographs contain these dots to be processed and extracted as feature points later.
3. To obtain feature point locations from the dots in the photographs, some preprocessing is applied to each pair. Histogram equalization and sharpen filtering are applied to each photograph to help identifying the dots more easily.
4. An algorithm is developed to obtain both the dot-marked feature positions in each pair of photographs automatically and the correspondences between them. The output of this step is a table containing the columns of feature points found in one pair, their correspondences in the other pair, and the pixel differences in terms of x-coordinate values of the correspondent features.
5. A depth map is generated by the help of pixel difference values of the correspondent feature points.
6. 3D space coordinates of these feature points are calculated from the previously defined depth map which the depth values give the z-coordinate value in 3D space.
7. The ICP algorithm [4] is implemented to bring the data cloud containing the 3D coordinates of feature points to a generic face model as close as possible.
8. An interpolation is performed to deform this generic face model with our calculated 3D feature point values. A smoothing algorithm is also applied over the entire face mesh after the interpolation.
9. Texture map and texture image are formed from the photos and the 3D face mesh. Then texture image is fitted on to 3D face mesh.

1.3 **Organization**

The rest of this thesis is organized as follows: The next chapter describes the automatic extraction of the feature points from each photograph and the calculation of 3D coordinates of them altogether. The second chapter describes the registration of our 3D data to a generic face model. The third chapter describes the generic face mesh deformation using the feature points and also the smoothing surface process. The fourth chapter describes the texture extraction using initial images and the feature point coordinates. The fifth chapter describes the FaceBuilder application, in which programming language Java and Java3D library employed. Finally in the last chapter, conclusion and possible future works are explained.

CHAPTER 2

FEATURE POINT EXTRACTION

2.1 Introduction

This chapter consists of acquiring 3D information of a human face from a pair of captured stereo photographs. Here, we can categorize this section into three parts; which are preprocessing with stereo images, automatically detection of the feature points, and calculation of 3D coordinates of features. The flow of these steps is listed below:

1. Preprocessing step consists of applying some image processing techniques to captured images to make it easy to detect the dots projected on the individual's face. This process returns a pair of images in which only projected dots are visible.
2. After this preprocessing, the positions of the dots, which are referred as our features from now on, are detected by an algorithm developed in this thesis. While detecting each on one image, the correspondent dot on the other is also detected by the same algorithm. The positions of the correspondent feature points and the pixel differences between them are kept to be used in the next step.
3. Finally, recovered feature points and the knowledge of the difference between the correspondent features are then used to produce a depth map for

each pair of images. 3D values are calculated from the depth map easily, since it gives the z-coordinate values of features.

It will be necessary to mention a little about capturing the stereo images to be preprocessed in first step. An individual is placed in front of the stereo cameras at a distance between 1 and 2 meters. We used camera zooms to locate the face of the subject inside the viewing area of both cameras so that the distance can vary. Subject is also positioned between these cameras by looking perpendicular to the center of them. A dot-pattern is projected onto his face and images are captured by containing these patterns so that the preprocessing can be made with these dots.

2.2 Preprocessing

Figure 2.1: Captured stereo images

A pair of captured stereo images is shown in Figure 2.1 when a dot-pattern is projected onto an individual's face. The white-colored points projected onto the

individual's face are the feature points which we want to obtain in order to use in reconstructing the 3D model.

Obtaining these projected dots means that acquiring their coordinate values as (x, y). But this can not be simply done by just looking the intensity values. Practically it is possible since the RGB value of the white color is (255,255,255) or a little below of these values. But the illumination or shadowing makes it harder to apply such a threshold for the intensity since there will always be dark places containing our white dot. This is because the RGB value is highly decreased so that it becomes much smaller than the value of the white color, i.e., (255, 255, 255).

So we need some pre-processing to mark these dots by a single, uncomplicated programming. The main idea is to identify them by their pixel intensity so to make this marking easier; two image processing methods should be applied. These are sharpening and color depth reducing. In some cases, histogram equalization may be applied before sharpening but in this work it is not needed so it is left for future work.

2.2.1 Sharpen Filtering

The sharpen filter enhances edges using a simple algorithm. This is very fast to compute but can produce artificially over-sharp or over-noisy images if it is not used carefully.

The sharpen filter uses a simple 3x3 square matrix convolution to enhance edges. The matrix for this convolution is:

$$\begin{bmatrix} -0.5 & -0.5 & -0.5 \\ -0.5 & 5.0 & -0.5 \\ -0.5 & -0.5 & -0.5 \end{bmatrix} \quad \text{Eq. 2.1}$$

This is the matrix used in this thesis. There are many other types of matrices but the difference is only how they affect the strength of the sharpening. For example, if the positive value of the matrix element is increased, then resulting image is more enhanced than the original input image. Similarly decreasing the value of that element results with less enhance in the input image.

(a) Original image (b) Sharpened image

Figure 2.2: Sharpening example

Figure 2.2 shows the effect of this filter when the matrix given in Eq. 2.1 is applied to our individual's photograph.

The key point in choosing the convolution matrix is that the sum of the matrix elements has to be 1. Choosing a different value for the middle element of the matrix changes the brightness of the image. That means, higher values makes it brighter and lower values make it darker.

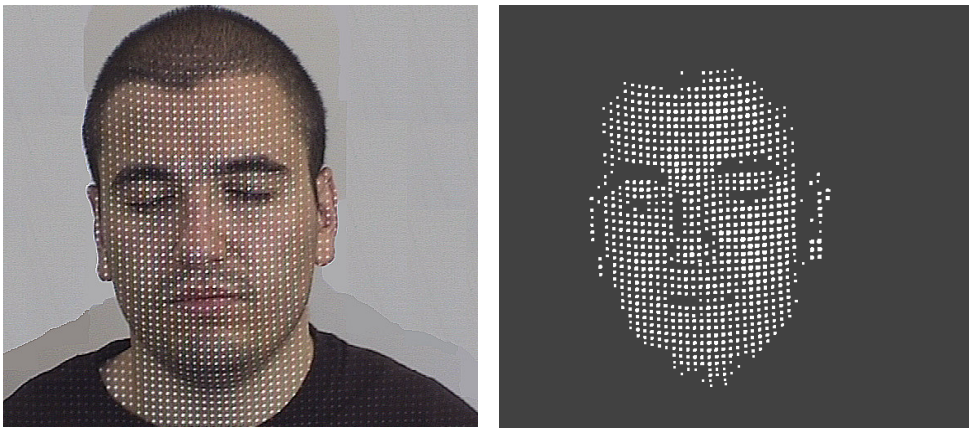
2.2.2 Thresholding

This process is all about setting the color of the projected dots as white and all other pixels as black. So that we will have white dots on a black background which makes it very easy to determine the x-y coordinate values of the features projected on each stereo pair.

Since sharpening helped a lot about making white and white-like pixels more visible, now detecting these dots are much easier. A simple procedure is followed which is described below:

- For each pixel intensity look for its RGB value, so that;
    - If this value is larger than the threshold, then mark this pixel as white.
    - Else, mark it as black.

Figure 2.3 shows the result of color depth decreasing applied on one of the sharpened stereo pair.



(a) Sharpened image (b) Thresholded image

Figure 2.3: Thresholding

2.2.3 Thinning

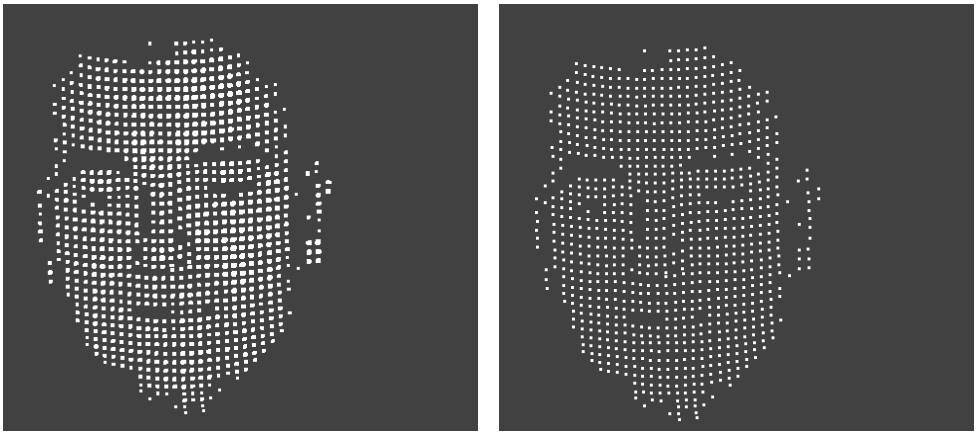
This process is needed because the projected dots are not actually one-pixel-points. Since each of them will represent a single feature point one must singularize all of these point groups.

The algorithm starts from upper-left corner and begins searching a white-colored pixel. When found, another searching starts within a window of which this point represents the upper-left corner of it. Window size is chosen experimentally and depends on the characteristic of the projected pattern. In this work window is chosen as a 5x5 square.

For any white-colored pixel found in that window, their coordinate values are added so that the average coordinate value can be colored as white and all other pixels

existed in window are colored as black. We can say that each window represents a white-colored pixel group and similarly each has only one white pixel intensity.

This process is repeated for each pixel found as white until the lower-right corner of the image is reached, i.e. there is no pixel remained unchecked.



(a) Before thinning (b) After thinning

Figure 2.4: Thinning

Figure 2.4 shows the result of the thinning operation when applied on a thresholded image.

2.3 Finding Correspondences

After the several steps explained above, now we have 2 stereo photographs each containing many white points representing our individual's feature data. We need to

obtain 3D point coordinates from the 2D ones which are now the white points on a black background.

The key part of calculating 3D coordinate values is the determination of the depth values which are actually the z-coordinate values. To achieve this, one must find the correspondences between the 2D points located at the stereo pairs so that the depth can be calculated with the help of disparity values between the correspondent points.

An algorithm is developed in this study for the correspondence problem. The algorithm starts with a pair of known points which are correspondent to each other. One for the left and one for the right photograph. These points are marked manually. Marking is performed by changing the color of the pixels to red which were white initially. To guarantee the result, two additional pairs of points are marked, too. These are for the left and right ear on each stereo image.

These marked points are our reference points for finding others and their correspondents. The algorithm runs as a turn-based-style i.e., switches searching operation from one stereo pair to another when necessary.

Searching starts from the marked point on the left image. Whenever a white point is found, it stores the pixel location (i.e., x-y coordinate values) and gives the turn to the right image. In the right image, searching continues from the last marked point which is at the start, the correspondent of the initial marked point on the left image.

To determine if the white point found on the right image is the correspondent of the one in the left or not, following procedure is applied.

1. For left image, find the minimum pixel distance between the last truly marked point and the one which we are now searching if it has a correspondent or not.
2. Store this distance.
3. For the right image, similarly, find the minimum pixel distance again between the nearest marked point and the one we are checking. The only important thing here is that the nearest marked point must be exactly the correspondent of the last marked one in the left. Otherwise, leave this white point and try for the next one.
4. Compare this distance with the one found in the left image.
    - If the difference between them is smaller than a threshold, then accept this point as a correspondent of the white point found in the left image (The threshold is determined by the pixel difference between any two neighbor points at x-coordinate located at the image projected onto the individual's face). Mark these points as red, store their positions and give turn to left image.
    - Otherwise, pass this point and try the next one until the y-coordinate value of the white point on the left image is exceeded by a threshold (in this work it is 2). The reason of choosing a small value for threshold is that, since the stereo photographs are taken in same vertical level, only the x-coordinate values of a feature differs in these photographs and y-coordinates remains same (i.e., assume a

point on the left image as (x, y), then the correspondent of this one is defined as (x+d, y) where d is called the disparity).

Repeat searching a white point on the left image until no white-colored point remains. Whenever a white point is found, again give turn to the right image and repeat all these steps to determine if the white point found on the right image is a correspondent of the one found on the left image or not.

After the whole process is finished, we have a 2D point set containing the information about the correspondent of each feature point.

You can also realize that there will be some points on one of the stereo images which have no correspondents. This situation results from the viewpoint of the stereo cameras, i.e., one camera may have a wider view range than the other one at some sides of the object to be viewed. For example, left camera can see more area of the left side of an object than the right one.

2.4 Depth Map and 3D Point Calculation

Now we have a pair of points for each feature point. So we can calculate the 3D coordinate values by the help of these stereo pairs.

For a pair of parallel, perspective cameras, the depth of a scene point corresponding to some image location (x1; y1) in the left image, and (x2; y2) in the right image is given by [12]:

$$Z = \frac{b * f}{x_2 - x_1} \qquad \text{Eq. 2.2}$$

where b is the baseline distance between the camera focal points, and f is the focal length of the camera.

While this formula is fair enough to calculate the depth, in this thesis, b and f are not used. We preferred a single calculation strongly based on only the disparity (i.e., x_2-x_1) to locate our 3D features as close as possible to our default 3D face model.

Of course, just the disparity is not enough to calculate depth value of the 3D point. For this problem, some kind of normalization is performed according to the default 3D model's depth bounds. The maximum depth value (i.e., min z-coordinate value) of the model is taken as a reference so that it can be matched with the feature having the minimum disparity. Why minimum comes from the depth depends on the disparity in reverse proportion. Other feature depths are calculated according to this one assumed as reference.

$$Z = \frac{d_{\min} * Mz_{\max}}{|x_2 - x_1|} \qquad \text{Eq. 2.3}$$

where d_{min} is the minimum disparity value, Mz_{max} is the z-coordinate value of the vertex of our default 3D model which has the maximum depth value (i.e., min z-value), and | . | denotes the absolute value.

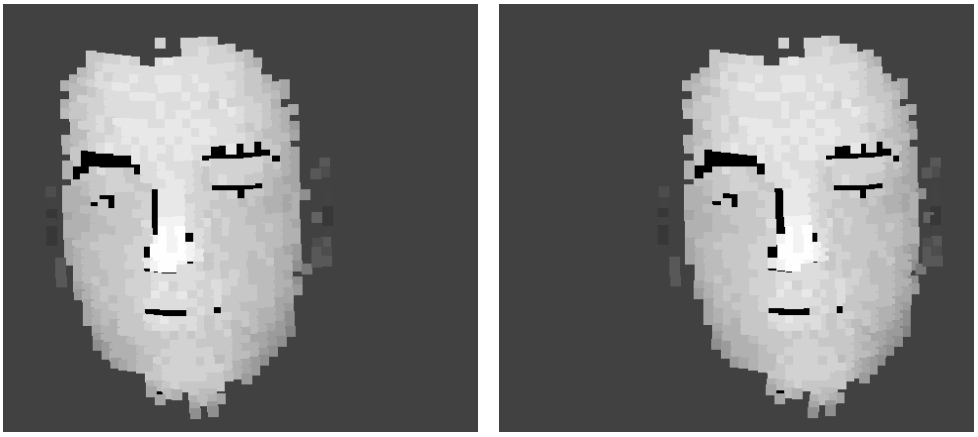


Figure 2.5: Depth maps of each stereo pair

Figure 2.5 shows the depth maps generated by Eq. 2.3. The depth information is shown in gray scale. Darker areas represent deeper points in 3D space, while lighter means closer.

The x and y coordinates can be calculated by the help of again the x and y coordinate values of the related white points on stereo images. In this work, we have used the default model's height (i.e., the difference of the y-coordinate values between the maximum and the minimum vertices) as a reference for normalization. So that the height of our recovered 3D point cloud will be same as the default model's height. But the width will be different which is more important since the face geometry mostly depends on the width.

The first step of the calculation is computing the average x and y values, since we have two points for each feature.

$$Ix = \frac{I_{left}(x) + I_{right}(x)}{2}$$
$$Iy = \frac{I_{left}(y) + I_{right}(y)}{2} \qquad \text{Eq. 2.4}$$

$I_{left}(x)$ and $I_{right}(x)$ are the x coordinate values of the point in the left and the right images respectively. Now, the height is fit to the model's height so that width can be found according to the relation between the 2D width and height of our features.

$$h = h_{model}$$
$$w = \frac{fx * h}{fy} \qquad \text{Eq. 2.5}$$

where h_{model} is the height of our default model. h and w are the height and the width of our 3D point cloud. Finally, fx and fy are the width and the height of the 2D feature points respectively. So, x and y coordinate values can be calculated as:

$$X = \frac{Ix - \min(Ix)}{fx} * w - (w/2)$$
$$Y = \frac{Iy - \min(Iy)}{fy} * h - (h/2) \qquad \text{Eq. 2.6}$$

Where min(.) denotes the minimum value of the given data array.

CHAPTER 3

RIGID 3D REGISTRATION

3.1 Introduction

In the previous chapter we found 3D coordinates of the features obtained from the projections. But these coordinates are not likely to match with our default (generic) 3D model. So we need a registration to bring our feature point cloud as close as possible to this generic 3D face mesh. It should be noted that the registration that we apply is rigid, that is we only rotate and translate the point cloud. The relative positions of the points with respect to each other are not changed.

The output of this step will be again a 3D point cloud of our features. But this time, the coordinates will be the exact positions of our features in 3D space since the 3D feature points will be in best alignment with our generic face mesh.

3.2 Iterative Closest Point Algorithm

For the registration, Besl's ICP algorithm [4] is implemented. The algorithm is commonly used for the following problem in computer vision: Given 3D data in a sensor coordinate system which describes a data shape that may correspond to a model shape, and given a model shape in a model coordinate system in a different geometric shape representation; estimate the optimal rotation and translation that

aligns or registers the model shape and the data shape by minimizing the distance between the shapes.

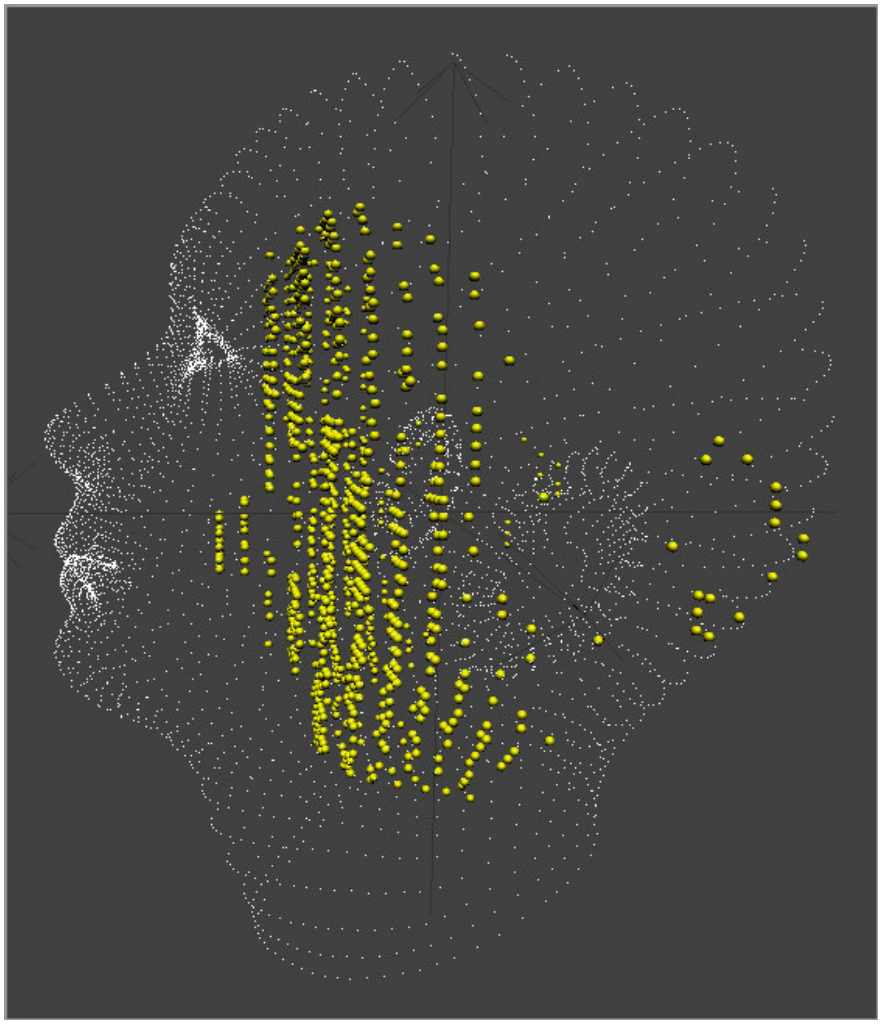


Figure 3.1: Before rigid 3D registration

Figure 3.1 shows our 3D point cloud (yellow points) and the 3D model (white points) before the registration is applied.

Assume that P represents our data shape consisting of 3D points and X represents model shape. We would like to move the data points to be in best alignment with the vertices of the model shape so that the operation can be called as point-to-point registration.

The iteration is initialized with setting $P_0 = P$, $\vec{q}_0 = [1\ 0\ 0\ 0\ 0\ 0\ 0]^t$, and iteration number $k = 0$. $\vec{q}_0$ is called the registration vector which consists of a unit quaternion vector $\vec{q}_R$, and a translation vector $\vec{q}_T$.

The unit quaternion is vector in the form $\vec{q}_R = [q_0\ q_1\ q_2\ q_3]^t$, where $q_0 \geq 0$, and $q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$. The translation vector is defined as $\vec{q}_T = [q_4\ q_5\ q_6]^t$.

The registration vector is defined relative to the initial data set P_0 so that the final registration represents the complete transformation. Steps 1, 2, 3 and 4 are applied until convergence within a tolerance τ.

1. Compute the closest points Y_k:

The distance d between an individual data point $\vec{p} \in P$ and the model shape X is denoted:

$$d(\vec{p}, X) = \min_{\vec{x} \in X} \|\vec{x} - \vec{p}\| \qquad \text{Eq. 3.1}$$

When the closest point computation (from $\vec{p}$ to X) is performed for each point in P, there will be a set of closest points $Y = \{ \vec{x} \mid \vec{x} \in X \}$.

2. Compute the registration $\vec{q}_k$:

The first thing that must to be done is calculating the "center of masses"; $\vec{\mu}_p$ for our point set P, and $\vec{\mu}_x$ for the default model X. Assume that N_p and N_x are the number of points in P and X respectively.

$$\vec{\mu}_p = \frac{1}{N_p}\sum_{i=1}^{N_p}\vec{p}_i \text{ and } \vec{\mu}_x = \frac{1}{N_x}\sum_{i=1}^{N_x}\vec{x}_i \qquad \text{Eq. 3.2}$$

The cross-covariance matrix $\sum_{px}$ of the sets P and X is calculated by the formula given below:

$$\sum_{px} = \frac{1}{N_p}\sum_{i=1}^{N_p}\left[(\vec{p}_i - \vec{\mu}_p)(\vec{x}_i - \vec{\mu}_x)^t\right] = \frac{1}{N_p}\sum_{i=1}^{N_p}\left[\vec{p}_i\vec{x}_i^t\right] - \vec{\mu}_p\vec{\mu}_x^t \qquad \text{Eq. 3.3}$$

The cyclic components of the anti-symmetric matrix $A_{ij} = (\sum_{px} - \sum_{px}^T)_{ij}$ are used to form the column vector $\Delta = [A_{23} \;\; A_{31} \;\; A_{12}]^T$. This vector is then used to form the symmetric 4x4 matrix $Q(\sum_{px})$:

$$Q(\sum_{px}) = \begin{bmatrix} tr(\sum_{px}) & \Delta^T \\ \Delta & \sum_{px} + \sum_{px}^T - tr(\sum_{px}) I_3 \end{bmatrix} \quad \text{Eq. 3.4}$$

Where I_3 is the 3x3 identity matrix and tr(.) denotes the trace (i.e., the sum of the diagonal elements of the given matrix). The unit eigenvector $\vec{q}_R = [q_0 \quad q_1 \quad q_2 \quad q_3]^t$ corresponding to the maximum eigenvalue of the matrix $Q(\sum_{px})$ is selected as the optimal rotation. The equation $(Q - \lambda I)\, x = 0$ is solved to find the eigenvalues λ and the eigenvectors x of the given matrix Q. The 3x3 rotation matrix generated by this unit eigenvector is given below:

$$R = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 + q_2^2 - q_1^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_0^2 + q_3^2 - q_1^2 - q_2^2 \end{bmatrix} \quad \text{Eq. 3.5}$$

The optimal translation vector is calculated as:

$$\vec{q}_T = \vec{\mu}_x - R(\vec{q}_R)\vec{\mu}_p \quad \text{Eq. 3.6}$$

The complete registration state vector becomes $\vec{q} = [\vec{q}_R \mid \vec{q}_T]^t$.

3. Apply the registration ($P_{k+1} = \vec{q}_k(P_k)$):

The registration state vector is applied to the all points $\vec{p}$ in our data set P. The optimal rotation is applied by a single matrix multiplication $P = R * P$.

The translation vector $\vec{q}_T$ is added on to the each point in our data point set *P*, such as:

$$P = \left[(\vec{p}_0 + \vec{q}_T) \quad (\vec{p}_1 + \vec{q}_T) \quad ... \quad (\vec{p}_{Np} + \vec{q}_T) \right] \qquad \text{Eq. 3.7}$$

4. Terminate the iteration when the change in mean-square error falls below a preset threshold $\tau > 0$ specifying the desired condition of the registration: $d_k - d_{k+1} < \tau$.

d_k denotes the mean-square error and is calculated by this way:

- For all point $\vec{p}$ in our data set *P*, find the distance between *p* and its closest point $\vec{x}$ in model shape *X*.
- Calculate the sum of the square of these distances.
- Find the square root of the sum.

$$d = \sqrt{\sum_{i=1}^{Np} \left| \vec{p}_i - \vec{x}_i \right|^2} \qquad \text{Eq. 3.8}$$

The iterative closes point algorithm always converges monotonically to a local minimum with respect to the mean-square distance objective function. The detailed proof of this theorem can be found in [4].

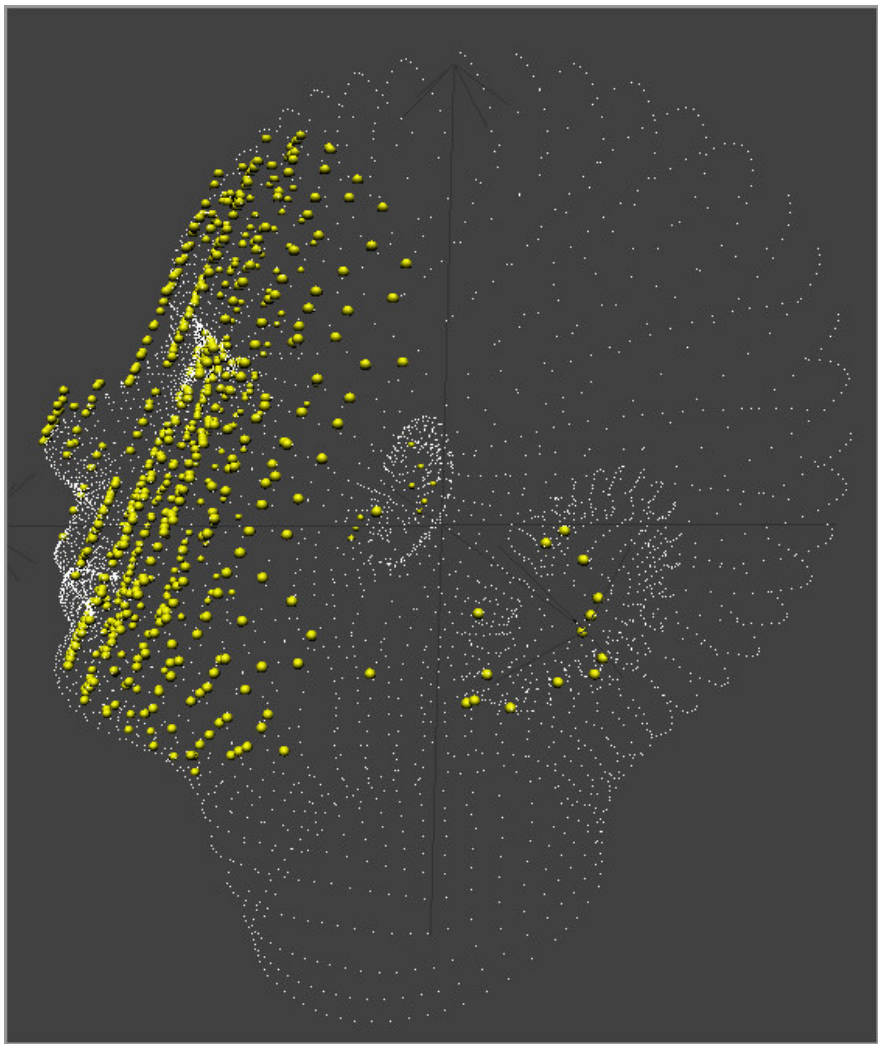


Figure 3.2: After rigid 3D registration

The final position of our 3D point cloud (yellow points) is shown in Figure 3.2. In this result, it is not surprising to have some of our 3D points match with the same vertex point of the 3D model. What we want is to match each 3D data point with only one vertex point. This necessity is required for the deformation step since each vertex point must converge to a single 3D data point because of the nature of the deformation.

So to complete the registration properly these multiple matchings must be eliminated to become a single matching. This operation is done by checking each 3D data point and its matched vertex.

Whenever a re-matched vertex point is found, i.e., already a matching of a 3D data point is checked earlier; the 3D coordinate values of these 3D data points are summed and the number of the summed points is stored. After the summation, the added points are eliminated from the original set so that the number of points is reduced as desired for the elimination.

When checking of all 3D data points is finished, simply all sums are divided to their corresponding "number of summed points" values (i.e., the average coordinate values are calculated for each sum) and the resultant 3D coordinate values are inserted as new 3D data points for the corresponding multi-matched vertices. This operation guarantees that each vertex will have only one 3D data point which is newly created so that the deformation can be performed without any conflicts.

CHAPTER 4

FACE MESH DEFORMATION

4.1 Introduction

At this point we finally obtained the real 3D coordinates of our features. These values are more realistic because they are also matched with the closest vertices of our default face model. We want to construct the 3D face model of the subject by deforming an initial 3D model with these calculated 3D coordinates obtained from the previous chapters. Now, what we have to do is to assign the new coordinates of these vertices of our model to these calculated 3D feature coordinate values, so that we can finally deform the generic 3D model by calculating the final coordinates of all other vertices for a smooth interpolation.

The main goal in that part is to perform a smooth interpolation while deforming our face mesh [22]. But soon, it is realized that there is no way to perform a deformation smooth enough with so many features. The details are discussed in later stages of the work.

For a given number of points, the displacement of the feature point *i* located on the face mesh is found by:

$$\mathbf{d}_i = \mathbf{X}_i - \mathbf{X}_i^0 \qquad \text{Eq. 4.1}$$

where $\mathbf{X}_i, \mathbf{X}_i^0$ are final and initial 3D coordinates of *i*'th feature point.

The general idea for deformation is that finding a function which produces displacement vectors for all vertices according to the known displacements. So the general function **f** can be written as:

$$\mathbf{d}_i = \mathbf{f}(\mathbf{X}_i) \quad i = 1...n \qquad \text{Eq. 4.2}$$

Where n is the number of vertices of the face mesh. There are several ways to construct such a function. In this work a weighted linear combination of radial basis functions are used.

4.2 Deformation using Radial Basis Functions

Radial Basis Functions will replace the function **f** in our main formula Eq. 4.2. so that it will be in the following form:

$$\mathbf{f}(\mathbf{X}) = \sum_i \mathbf{c}_i \mathbf{r}(\left\|\mathbf{X} - \mathbf{X}_i^0\right\|) \qquad \text{Eq. 4.3}$$

Where **r** stands for radial basis function (RBF), and **c** is the displacement coefficient for each feature point. First of all RBFs should be selected as a smooth, continuous, and decreasing function. Two different RBFs are used in this work:

Polynomial Functions: $r(x) = a + bx + cx^2 + ... + kx^n$

Exponential Functions: $r(x) = e^{-(a+bx+cx^2+...+kx^n)}$ Eq. 4.4

4.2.1 Gaussian Interpolation

The Gaussian function used for this work is the exponential function with only a single non-zero coefficient of $\mathbf{x^2}$. The Gaussian also has a zero mean (i.e, $\mu=0$).

The variable **x** is scaled so that RBF output at some vertex of 3D model will be close to zero. This vertex must be the matching of the nearest neighboring feature point of our 3D point cloud. This operation provides a different boundary for the RBF function output for each different **x**.

After deciding the RBF type, we need to calculate **c** vectors which are the coefficients of the RBF. Values of c vectors are ordered according to guarantee that the function **f** moves initial feature points to their known destination coordinates. The following equation shows the matrix representation of Eq. 4.3 for a number of N known feature points:

$$\begin{bmatrix} \mathbf{r}(\|\mathbf{X}_0^0 - \mathbf{X}_0^0\|) & \cdots & \mathbf{r}(\|\mathbf{X}_j^0 - \mathbf{X}_0^0\|) & \cdots & \mathbf{r}(\|\mathbf{X}_N^0 - \mathbf{X}_0^0\|) \\ \vdots & & \vdots & & \vdots \\ \mathbf{r}(\|\mathbf{X}_0^0 - \mathbf{X}_i^0\|) & \cdots & \mathbf{r}(\|\mathbf{X}_j^0 - \mathbf{X}_i^0\|) & \cdots & \mathbf{r}(\|\mathbf{X}_N^0 - \mathbf{X}_i^0\|) \\ \vdots & & \vdots & & \vdots \\ \mathbf{r}(\|\mathbf{X}_0^0 - \mathbf{X}_N^0\|) & \cdots & \mathbf{r}(\|\mathbf{X}_j^0 - \mathbf{X}_N^0\|) & \cdots & \mathbf{r}(\|\mathbf{X}_N^0 - \mathbf{X}_N^0\|) \end{bmatrix} \begin{bmatrix} \mathbf{c}_0 \\ \vdots \\ \mathbf{c}_i \\ \vdots \\ \mathbf{c}_N \end{bmatrix} = \begin{bmatrix} \mathbf{d}_0 \\ \vdots \\ \mathbf{d}_i \\ \vdots \\ \mathbf{d}_N \end{bmatrix} \quad \text{Eq. 4.5}$$

Simply

$$\mathbf{R} * \mathbf{C} = \mathbf{D} \quad \text{Eq. 4.6}$$

Where **R** is the RBF results of known feature point displacements, **C** is the unknown coefficient vectors written as a vector and **D** is the displacement vectors of feature points from their initial to final positions.

Since matrix **R** is an NxN square matrix and has an inverse, **C** can be calculated as follows:

$$\mathbf{C} = \mathbf{R}^{-1} * \mathbf{D} \quad \text{Eq. 4.7}$$

Now this function **f** can be applied to all vertices of the generic face mesh, and deform it to fit to our subject's face features.

4.2.2 Linear Interpolation

A linear function is tried after seeing the results of the Gaussian interpolation. The results are not as smooth as we expected, so that first, a linear function is constructed. When it is realized that the result can not be improved, to decrease the running time and complexity, some little changes are performed.

A linear function of $r(x) = a + bx$ is chosen from the polynomial function set. The coefficients are determined according to the output of the RBF. The coefficient **a** will be the radius value of the deformation area around **x**.

There are some steps similar to the ones in the exponential function part. Such as, the determination of radius **a** according to the output of RBF at nearest neighboring feature point will be close to zero. Similarly the coefficient **b** is chosen again making that output a desired value which is below one for the closest neighboring feature.

An extra process is done by finding all of the vertices of the mesh around each feature point **x** within a radius of **a**. Since the final positions of each feature **x** is known, the unit direction vector can be calculated for all such that:

$$\vec{\mathbf{d}}_i = \frac{\mathbf{X}_i - \mathbf{X}_i^0}{\left\| \mathbf{X}_i - \mathbf{X}_i^0 \right\|} \qquad \text{Eq. 4.8}$$

Now instead of applying the function **f** to all vertices of the generic face mesh, we can apply it only to vertices that lie in the boundaries of the features.

4.2.3 Results

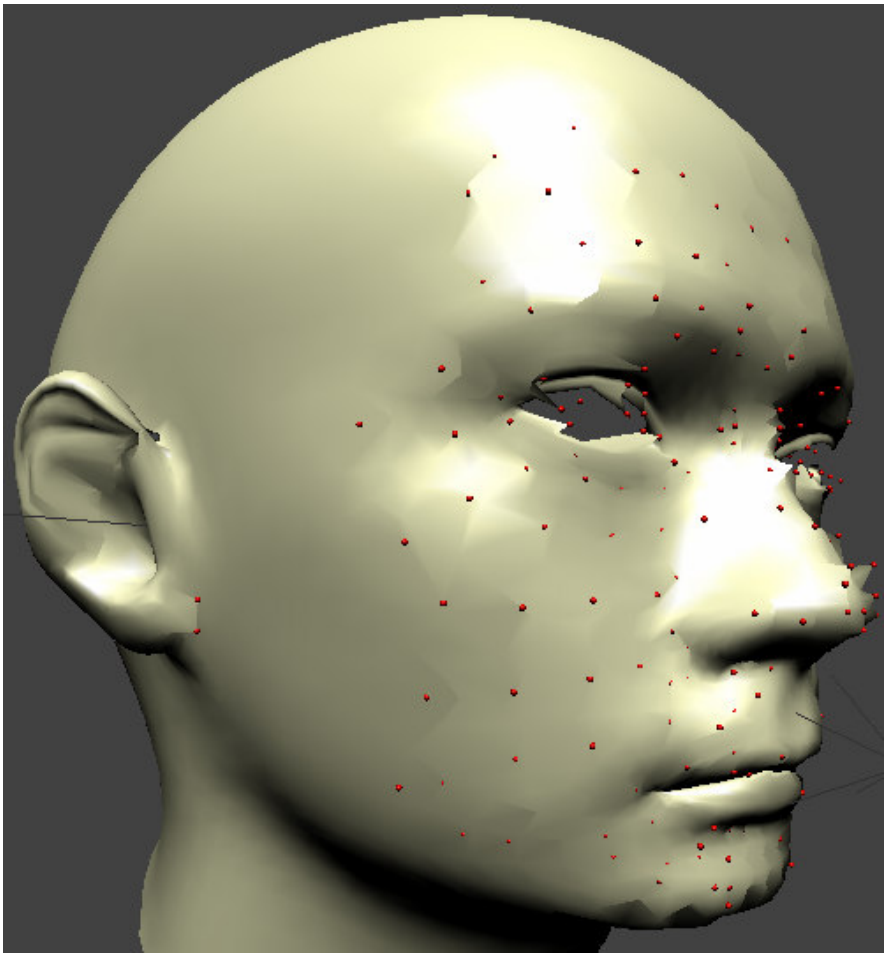


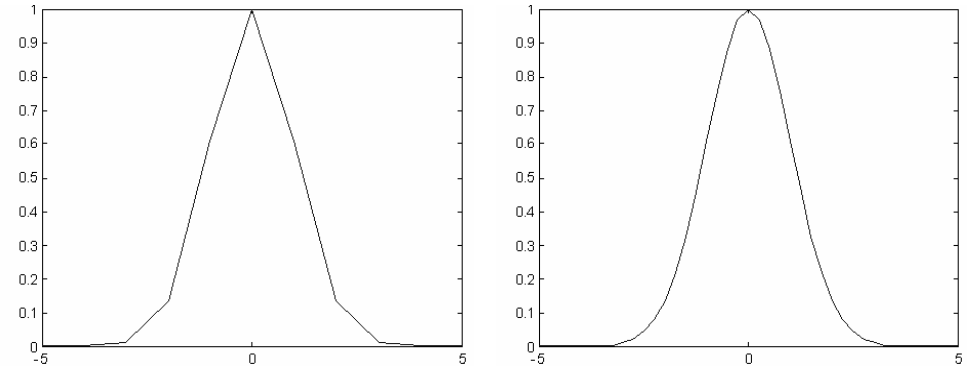
Figure 4.1: Deformed face mesh

Figure 4.1 shows the result of the deformation. The red colored points are the features which the face mesh is deformed according to their 3D coordinates.

It is clearly seen that the new mesh is too much noisy, having lots of holes and peaks, despite using a smooth interpolation. This result may be understandable with a linear

interpolation. But these noises are observed again when the Gaussian interpolation is applied. So that it can be explained only in that way:

To form a Gaussian, there must be enough points lying on the curve. Otherwise, it won't be smooth and will look like a sharp peak as it is in our result.



(a) Gaussian with 10 points (b) Gaussian with 40 points

Figure 4.2: Gaussians with different number of points

Figure 4.2 clearly shows that the number of points lying on the Gaussian curve directly affects its smoothness. In our case, we realized that there is not enough number of vertices lying on each feature's boundary like the example shown in Figure 4.2(a) so that the resultant Gaussian can not be sufficiently smooth. The reason why there is not enough number of vertices is that the neighboring features are very close to each other so that only few vertices lie between them.

4.3 Surface Smoothing

To overcome the problem explained above, a surface smoothing algorithm is implemented. The proposed algorithm [13] performs smoothing without changing the number of vertices on our face mesh.

The algorithm works as follows:

- For each vertex v_i on the mesh, calculate Δx_i as the weighted sum of the vectors from the vertex to its neighbors. In Eq. 4.9, N_i denotes the number of neighbors of v_i, and v_j denotes the j[th] neighbor of v_i.

$$\Delta x_i = \frac{\sum_{j=1}^{N_i} (v_j - v_i)}{N_i} \qquad \text{Eq. 4.9}$$

- Then, for each vertex v_i, add in λ times Δx_i (where λ is a number between 0 and 1).

$$v_i = v_i + \lambda \cdot \Delta x_i \qquad \text{Eq. 4.10}$$

- Repeat the algorithm until the mesh is sufficiently smooth.

This process produces Gaussian smoothing, which causes shrinkage of the mesh over time. The shrinkage is the only handicap but it does not affect the result directly since symmetry of the mesh does not change, it only gets smaller.

The shrinkage can be overcome by the help of a negative value of λ. This value is called μ and must be less than -λ. The calculation in Eq. 4.10 is performed one more time, but this time, μ is used instead of λ. This second step counteracts the shrinkage while still smoothing the mesh.

While this step prevents the mesh to shrink, it is not preferred due to the fact that an extra step will be added for each iteration which doubles the running time of the algorithm. In this thesis, we realized that this second step does not only counter attack the shrinkage, it also prevents the smoothing for our case. This requires again more iteration to obtain a smooth surface, so it is given up.

The final mesh is shown in Figure 4.3:

Figure 4.3: Smoothed face mesh

CHAPTER 5

TEXTURE EXTRACTION

5.1 Introduction

Up to now, we have captured a pair of stereo photographs of the subject's face. We have semi automatically marked the feature points which were projected onto the individual's face during the capturing. Then 3D coordinates of these feature points are calculated by the help of stereo information and these coordinates are used to deform the 3D generic face mesh. Finally a surface smoothing algorithm is implemented to get rid of the noises caused by the deformation. Now we will construct a texture image to cover our deformed face mesh.

The texture extraction problem can be defined as follows: Given a set of photographs, their viewing parameters and a fitted face mesh; compute texture color **T(p)** for each point **p** on the face mesh.

In this thesis we are going to find each texture color without the knowledge of viewing parameters. We have tried a different technique which collects the texture color with the help of the feature locations on our initial photographs. We are going to locate a boundary on both the texture map and initial photographs where the face lies inside and gather necessary information as explained later.

5.2 Forming the Texture Image

Before starting there is a point that needs to be clarified. Since, both of our photographs are captured from the frontal position, we do not expect to form a texture image which totally covers our deformed face mesh. There is not enough information about the profile of the face on frontal photographs. That kind of information can only be obtained by profile photographs.

Recall that our initial photographs contain a projected pattern. Our texture image must not contain these patterns so another pair of stereo photographs is also captured for this purpose.

We still need the initial captured stereo photographs that contain projections because these projections will help us to determine their new coordinates on the texture map.

A texture map is a 2D plane covering the entire 3D model containing the surfaces of it. Each surface is shown as polygons, preferably triangles in which each corner of the polygon represents a corner (vertex) of the corresponding surface of the 3D model.

Here, you can find the texture map of our 3D face mesh which is given in Figure 5.1:

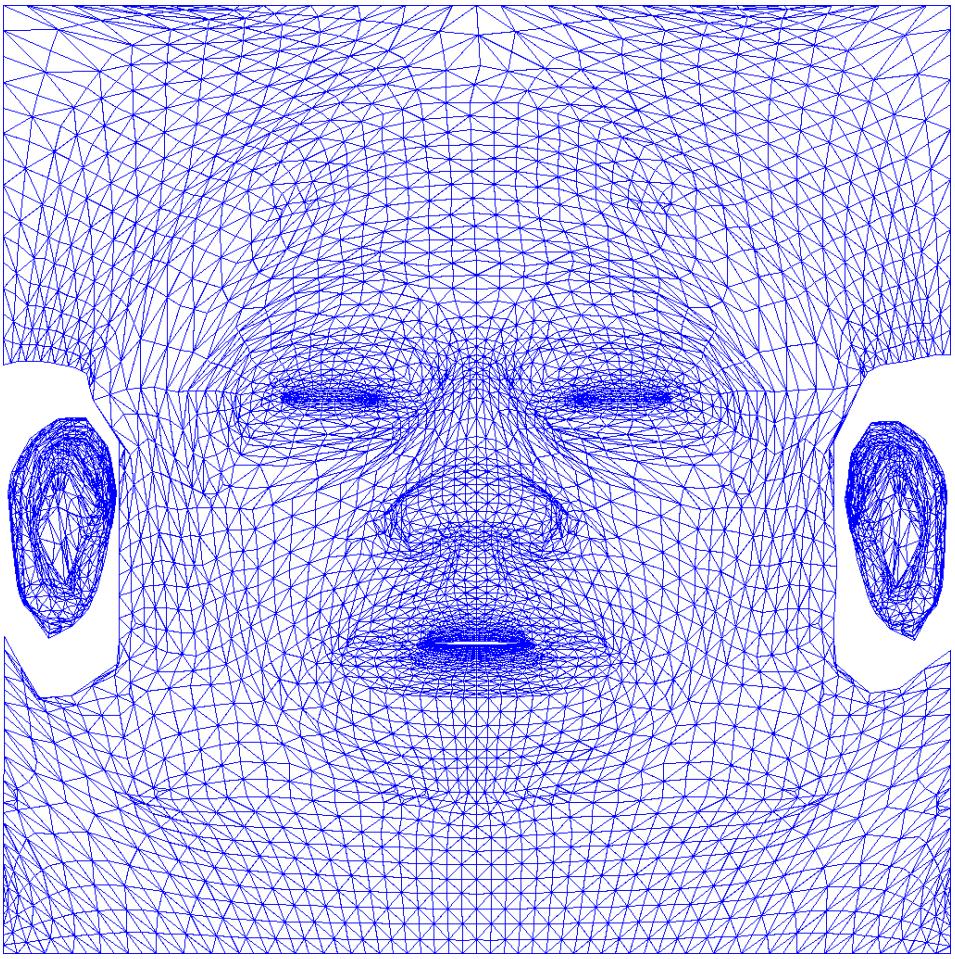


Figure 5.1: Texture map

The algorithm runs like a scan-line process. It is based on scanning a line from initial photographs and putting the scanned colors to the corresponding place on texture image. Of course the scanned line will need spreading or narrowing to be fit on the texture image.

For this purpose, a scanning boundary is selected on both initial photographs. This boundary is selected as a rectangle. Four projected points are found to represent the

boundary of that rectangle which are the points having the minimum x-value, maximum x-value, minimum y-value and the maximum y-values.

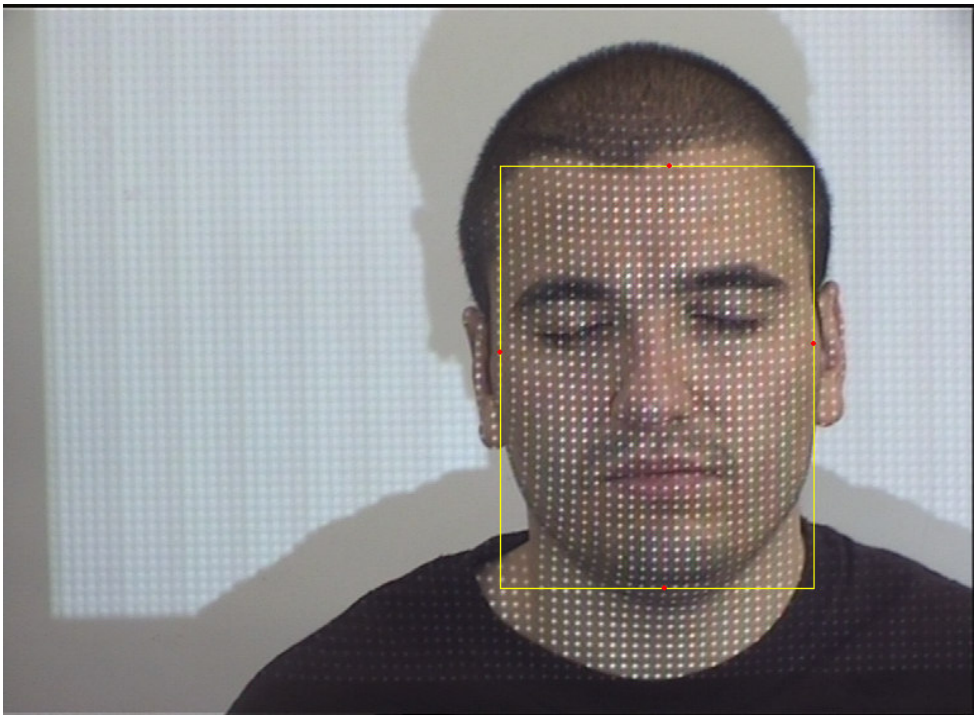


Figure 5.2: Scanning boundary

The scanning is performed inside the yellow box which is created by the help of the points having minimum and maximum of x and y values which are shown as red. There are some regions inside the box which doesn't lie on the face. To simplify the scanning process, a hexagon boundary is introduced. In the texture map, it looks like the frontal region of the face is located in a hexagonal area as shown in Figure 5.3. This is why we preferred that kind of boundary for the scanning process.

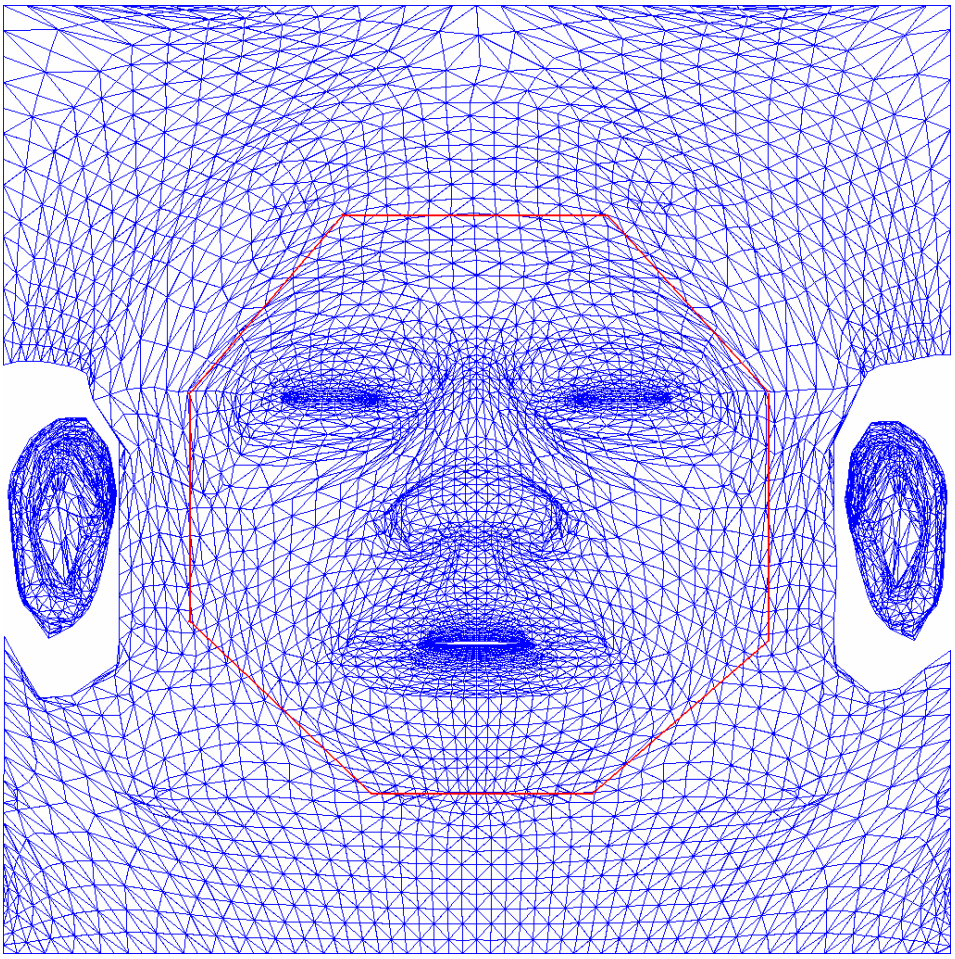


Figure 5.3: Frontal region of the face on texture map

So, the real boundary becomes something like in Figure 5.4 when scanning process starts:

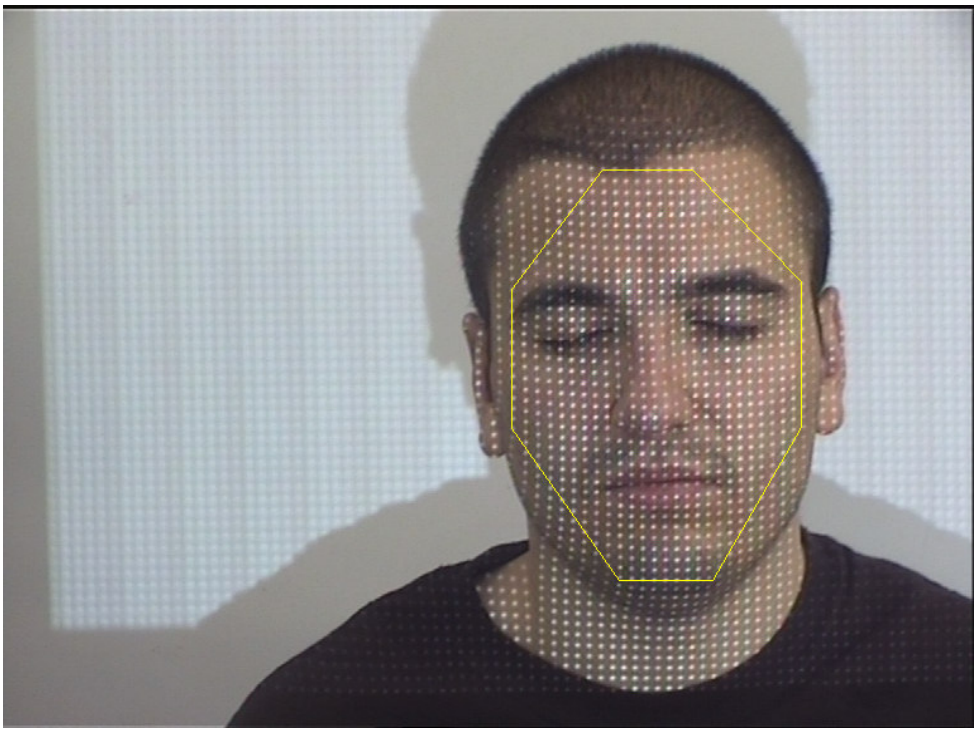


Figure 5.4: Real boundary

The projected points that are used to merge to form this polygon hopefully have correspondents on the texture image. We know the correspondent vertices that match with these points after the calculations in early chapters. We also know that every corner of a triangle on the texture map represents a vertex on 3D face model. So with that information we can easily define a similar boundary for texture just like the one on initial photographs.

There is just one thing left to perform this operation correctly. The length of a scanned line may not fit to its place on the texture image. Also we may need to put the scanned line more than one times to the corresponding place on texture. This problem comes from the size of the corresponding boundaries.

To overcome this, whenever a line is scanned on a photograph its length is calculated as the number of pixels on x-coordinate. Length of its place on texture is also calculated so that the line can be spread or narrowed according to the ratio between them. This operation is performed for every line on photograph to put them their correct places on texture.

There may be non-colored lines on the texture image when the whole process is finished. These lines are colored according to the line which lies up next to them. An appropriate resizing is performed if necessary and then the empty line is filled with that one.

Figure 5.5: Created texture image

Figure 5.5 shows the texture image created by the method which is explained above. This is the last part of the whole process. Now application can show the final result that is a deformed face mesh with the texture created from subject's photographs.

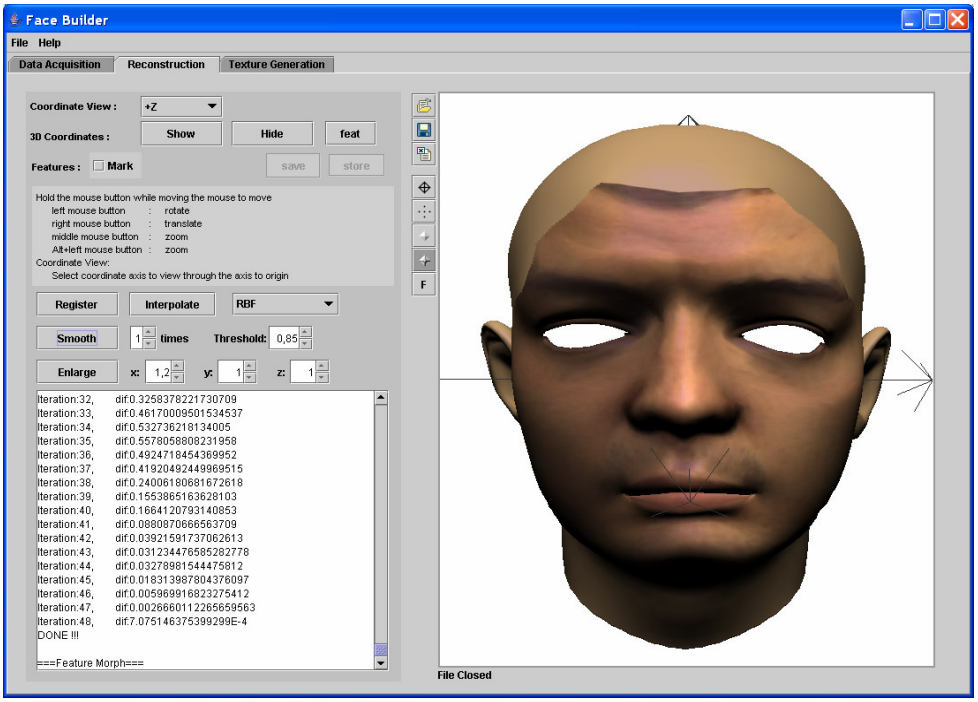


Figure 5.6: The complete model

CHAPTER 6

FACEBUILDER

6.1 Introduction

In the previous chapters, the details of the study are described. This chapter explains the FaceBuilder application developed for this thesis which is able to reconstruct 3D face models from a pair of captured stereo photographs.

The FaceBuilder application is compiled and developed with Standard Java Development Kit (JDK) version 1.4.2. In addition to this, standard Java core application employs Java3D and JFreeChart. Java3D is a 3 dimensional, high-level graphics extension of Java that is used to form 3D part of FaceBuilder. JFreeChart is a free library capable of drawing vast variety of plots on to the screen, and is used for process analysis and debugging. Standard JDK also includes Java2D that has been employed in this thesis for image processing and viewing purposes.

In this chapter; Java programming language, Java3D and the FaceBuilder application will be explained.

6.2 Java

First of all, Java is an object oriented programming language that needs an interpreter Java Runtime Environment (JRE) to run java programs. Platform independency makes Java very feasible for internet applications. Once the code is written and deployed, the application can run on any operating system.

Second, Java has a huge developer community that publishes free extensions like JFreeChart which is also used in this study for plotting purposes in explaining program execution. A lot of open source examples and free books are also available at internet on nearly all Java related subjects.

Finally, Java is a pure object-oriented language that makes it invaluable. The programs that are written in this manner are extensible and easy-to-understand. It also has a default exception handling mechanism that is easy-to-debug and shortens the development time.

In the Java programming language, all source code is first written in plain text files ending with the `.java` extension. Those source files are then compiled into `.class` files by the Java compiler (`javac`). A `.class` file does not contain code that is native to your processor; it instead contains *bytecodes,* the machine language of the Java Virtual Machine. The Java launcher tool (`java`) then runs your application with an instance of the Java Virtual Machine. Because the Java Virtual Machine is available on many different operating systems, the same `.class` files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or MacOS.

However, Java has some drawbacks such as the lack of simple binary types like unsigned char and the interpreter that could cause a performance decrease. Since excluding some simple data types is the choice of SUN Inc. developers, there is nothing to be done about it. But on the other hand, new versions of JRE use Just-In-Time Compilers that is able to convert simple binary operations to native platform machine code, and nearly no performance difference can be sensed compared to native codes.

6.3 Java2D

Java2D is a 2D graphics framework which is included in standard JRE. This library is capable of creating arbitrary shapes, texts and images. You can manipulate generated graphics with the help of Java2D API (Application Programming Interface). Affined Transformations that are rotation and translation, Composition that generates transparent graphics, Color Transformation that changes any color parameter, Filtering that applies any kind of convolution to graphics for blurring or sharpening purposes, Data Selection that chooses any arbitrary region of interest, are widely used tools in FaceBuilder.

Images can be loaded from and saved to JPEG, PNG and GIF formats which are most common ones. Images are easily resized by the Java2D API. User can define rendering quality and performance by setting rendering parameters.

The Java2D API involves:

- **Shapes** - standard forms such as rectangles and ellipses, as well as arbitrary shapes like Polygon.
- **Stroke** - vary the thickness of a line, produce arbitrary dot-dash sequences, select how corners and line ends are shaped, etc.
- **Painting** - fill the interiors of shapes, as well as outlines, not just with solid colors but with gradients of one color to another, with textured patterns of tiled images.
- **Transformations** - affine transformations (i.e. those that keep parallel lines parallel) can take a shape and translate it, rotate it, compress or expand it. Multiple transformations can be compounded together.
- **Compositing** - overlay shapes and images according to several optional rules that take advantage of the alpha transparency.
- **Clipping** - clip not just with rectangles but with arbitrary shapes.
- **Anti-aliasing** - can remove the stair-stepped jaggedness caused when shapes are rendered with the finite sized pixels.
- **Text** - vast array of text rendering capabilities that range from the basic *drawString()* method to create your own editor from scratch.
- **Color** - lots of tools to help render colors in a consistent manner regardless of the device on which they are displayed.
- **Images** - can use the same tools used with shapes, e.g. transformations, compositing, clipping etc. on images.
- **Image processing** - several standard image filters; edge enhancement, blurring, inversion, etc. are available.

- **Printing** - can print the graphics display to the local printer in manner faithful to what is seen on the monitor.

6.4 Java3D

Java3D is a Java extension developed by Sun Microsystems, Inc. as a joint collaboration with Silicon Graphics, Inc., Intel Corporation, Apple Computer, Inc. and it is free to use. In this study Java3D version 1.3 is employed. Actually this extension is not implemented in pure Java; it is a wrapper of two commonly used graphics libraries, OpenGL and DirectX. This study uses OpenGL core Java3D that is used on platforms like UNIX.

Java3D extension is designed as a high-level library, which is the main advantage of using it. Developers do not need to optimize its performance for different hardware platforms.

The scene graph programming model is a simple and flexible way for both representing and rendering complex 3D worlds. All information of the virtual world and the entire scene is included in this scene graph.

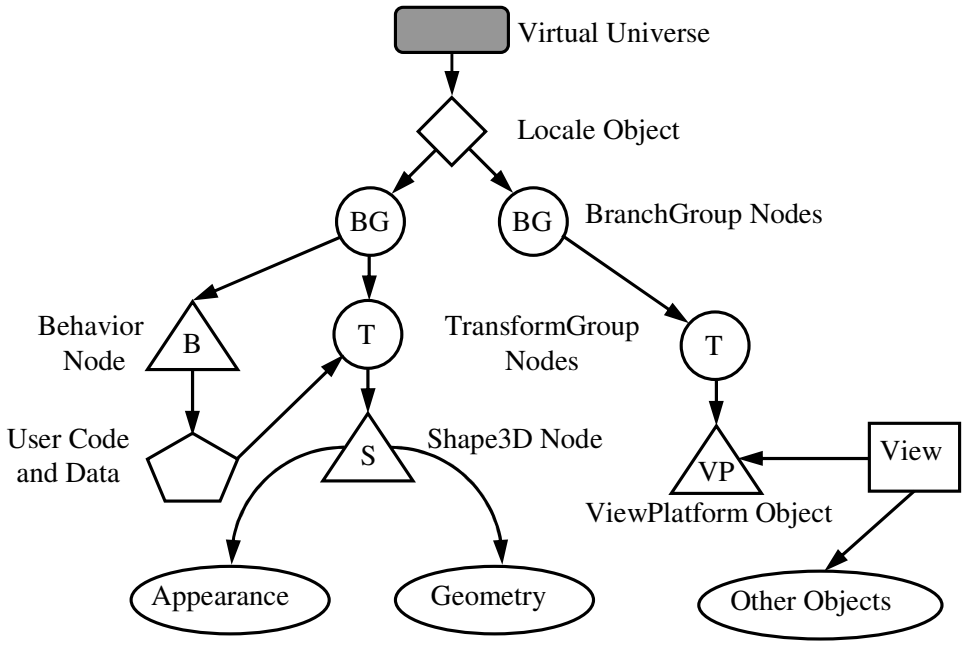


Figure 6.1: Java3D scene graph

Figure 6.1 illustrates a simple application's scene graph. A simple scene graph consists of superstructure components which are a VirtualUniverse, Locale objects, and scene graph branches, or subgraphs. Each subgraph is rooted by a BranchGroup node that is attached to the superstructure. Subgraphs hold data of 3D shapes and viewing properties.

A VirtualUniverse object is the root of a scene graph. User can create more than one VirtualUniverse in Java 3D which is not the case vast majority of applications. All Java 3D scene graphs must be connected to a Virtual Universe object to be displayed.

Below the VirtualUniverse object is a Locale object. The Locale object defines the origin, in high-resolution coordinates, of its attached subgraphs. A Virtual Universe may contain as many Locales as needed.

The scene graph branch itself starts with the BranchGroup nodes. Only BranchGroup objects can attach to Locales. A BranchGroup roots a subgraph, or branch graph, of the scene graph. In Figure 6.1, there are two subgraphs and, thus, two BranchGroup nodes. Attached to the left BranchGroup are two subtrees. One subtree consists of a user-extended Behavior leaf node which contains Java programming language code to manipulate the transform matrix associated with the object's geometry. The other subtree in this BranchGroup consists of a TransformGroup node that specifies the position (relative to the Locale), the orientation, and the scale of the geometric object in the virtual universe. A single child, a Shape3D node, refers to two component objects: a Geometry object and an Appearance object. The Geometry object describes the geometric shape of a 3D object and the appearance object describes the appearance of the geometry.

The right BranchGroup has a single subtree that consists of a TransformGroup node and a ViewPlatform leaf node. The TransformGroup specifies the position (relative to the Locale), the orientation, and the scale of the ViewPlatform. This transformed ViewPlatform object defines the end user's view within the virtual universe. Finally, the ViewPlatform is referenced by a View object that specifies all of the parameters needed to render the scene from the point of view of the ViewPlatform.

6.5 FaceBuilder

In this section, the face modeling process is explained in conjunction with our application named FaceBuilder. Face modeling can be divided into three sub-problems which are:

1. Range data acquisition, feature point and correspondence calculation handled in Data Acquisition module,
2. 3D registration and 3D model deformation handled in Reconstruction module and,
3. Texture generation handled in Texture Generation module.

A proper process must be performed in the same order that is data acquisition, reconstruction and texture generation.

6.5.1 Data Acquisition

Face Modeling starts with range data acquisition in which captured digital stereo photographs are loaded to FaceBuilder. FaceBuilder can use common image formats JPEG, GIF and PNG, which are standard Java2D file formats. In this study PNG image format is used.

Below, the data acquisition module is shown in Figure 6.2:

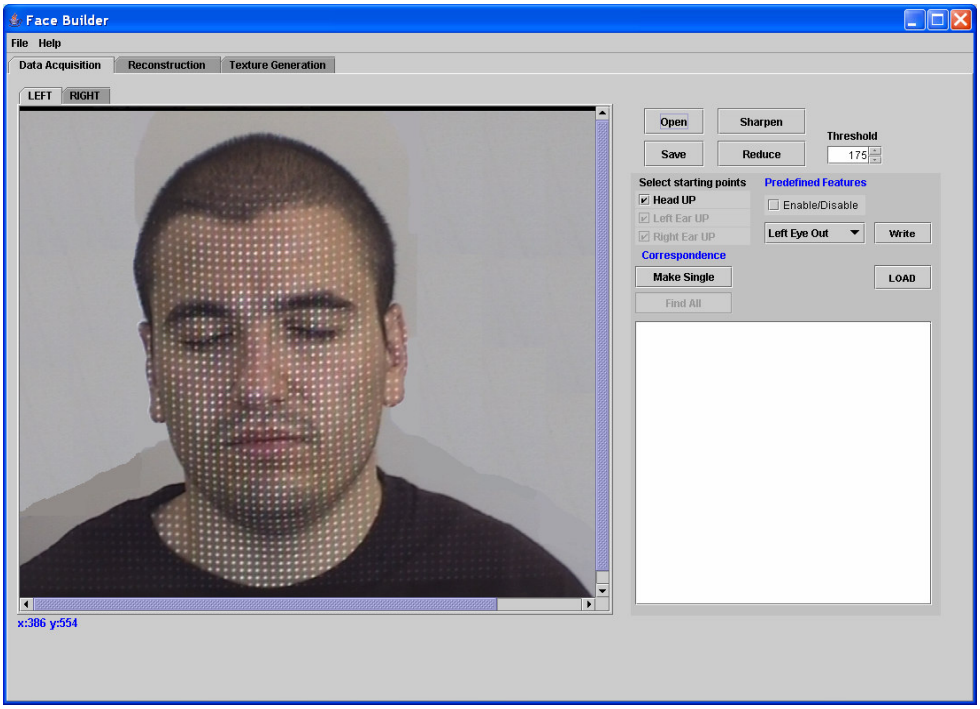


Figure 6.2: The user interface of the Data Acquisition Module

Quality of face images does not directly affect the results of the modeling process. But lower resolution images may cause problems. The resolution must be high enough to show all projected points clearly. Otherwise, it may cause information loss and mislead to wrong correspondence finding. In this study, images with a 768x576 resolution are used.

The resolution also varies on the number of points included in the projection pattern. The more points the pattern has, the more resolution is needed. For the resolution chosen in this work, an 85x64 dot-pattern is used.

Besides resolution, sharpness is another important factor that affects feature point marking process. Feature points can be found and marked more precisely on shaper images. For this purpose, user can sharpen any image by pressing the sharpen button located on the user interface. It is highly recommended to sharpen first so that the application can identify the projected white points easier as described in earlier chapters.

The image format may be important since for instance, a more compressed and a less quality JPG image may cause hard to gather correct information or lose it. PNG is preferred since the acquired pixel information is same with a raw image and it requires much less memory than a raw one. Although JPG and GIF images require lower memories, they are not preferred due to those risks of loosing correct information from the images. But the quality of the image is not that important for textures so, JPG and GIF image formats can be used to texture images.

In addition to these constraints, face images should have equivalent luminance. Flash should not be used because shining makes harder to identify projection patterns. It also affects the texture images. It is better not to have shadow on face for the same reason. Some of these imperfections can be eliminated by the help of histogram equalization of face images but for a better result, the photographs shall be taken in equivalent lightning conditions if possible.

All rules to achieve a good result are listed below:

- Stereo photographs should be captured approximately at the same vertical level.

- Resolution of images should not be smaller than 768x576. For PNG images, it shouldn't be higher than 1024x768 to save memory.
- Similar lighting conditions should be supplied for both left and right images. Photographs should be taken with plenty of light, which should not cause shining and shadows on the face.
- It will be better if subject has neither any hair on face, like bread or mustache, nor any instruments that occlude face parts, like glasses. These things may absorb the patterns over the face so they are not preferred.
- Subject must be placed as close as possible to the middle of the focal points of the stereo cameras.

In order to construct a face model; first, 2 stereo images captured with patterns should be loaded using the FaceBuilder application. To determine the projected points; sharpening, reducing, single making and marking of 3 starting points must be performed in this order. The starting points must be marked in the order of head, left ear, and right ear. When, for instance, a point is marked for head in left image, its correspondence must be marked in right image for next before marking another point on left.

Next step is to run the algorithm for finding the correspondences and the 3D coordinates of these projected points representing features. The algorithm needs these 3 pair of points which are correspondent to each other so that other points and their correspondences can be found.

The correspondence algorithm strongly depends on the pattern projected on to the subject's face. The distances between the points on pattern image defines the threshold value to determine if a point can be a correspondent of the one located in other image. Presently, the algorithm can not determine the threshold for different projection patterns having different number of points. It is left for future work since one may use stripes instead of points on pattern image.

At the same process, 3D coordinates of the points which have correspondences are also calculated. The calculation is done according to the coordinates of our default 3D face model so that the features are placed as close as possible to that model. Since height of the model is taken as a reference, only the width and depth can changed by deformation at later step.

An extra feature marking can be made to improve the speed in later steps. This option is located at the predefined feature section on the module. The number of these predefined features is 18 which are; inner and outer corners of eyes, upper and lower connections of ears to the face, left-right-up and down corners of mouth, and left-right-saddle and tip of nose. These points are selected for their ease of distinguish ability from all point of views. They are also marked on the default face model to be used in next module.

6.5.2 Reconstruction

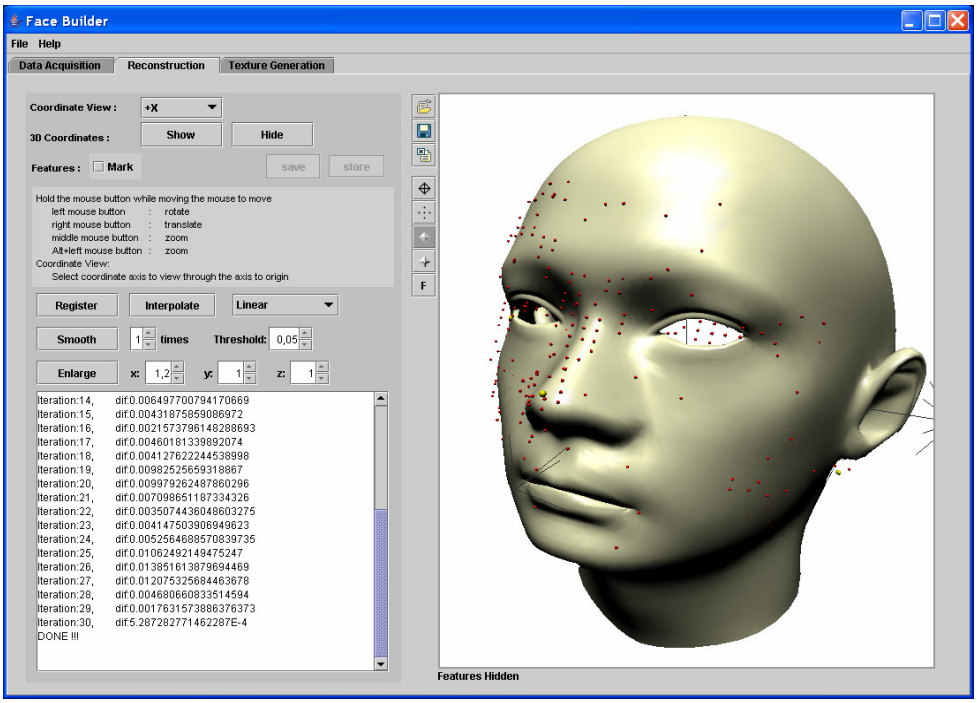


Figure 6.3: The user interface of the Reconstruction Module

The Reconstruction module is composed of a 3D scene viewer and a panel containing all options needed for morphing as shown in Figure 6.3.

The initial face mesh is loaded to the scene viewer when the application starts. User can also load any other mesh model. The FaceBuilder application accepts OBJ and VRML 3D file formats. In addition to 3D geometry definition, these formats also support texture loading. A default face mesh and a texture map with 5974 vertices are mainly used in this thesis. There is another face mesh available with 827 vertices, but it is thought that a more detailed mesh having more vertices should provide better results with so many features like we have.

On the left panel there are two sections separated by an information label between them. The upper part consists of a variation of viewing position of the face mesh and show-hide options for the 3D locations of our features calculated at previous module. All of these are displayed in 3D scene viewer. There is an extra option for the predefined features as shown in Figure 6.4 if they are marked or loaded in the data acquisition module. This option enables user to mark the correspondents of these pre-defined features on the face model. This option is added to improve both the performance and the registration speed which will be explained later. In order to continue the registration process, first these features must be shown to allow the application load their 3D coordinate values.

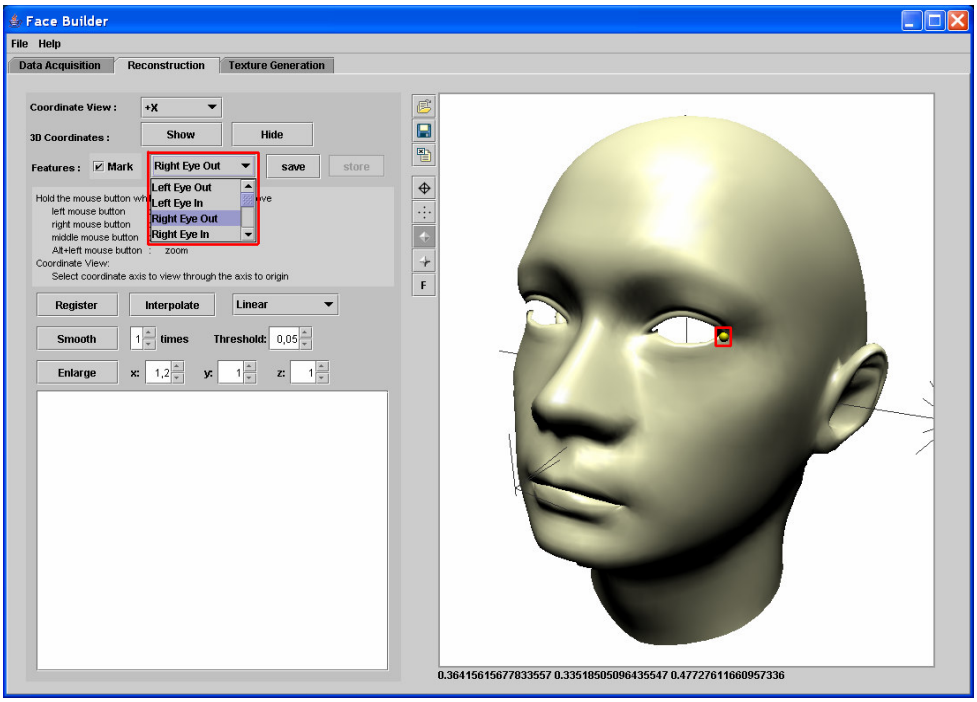


Figure 6.4: Adding additional points

All other steps are done on the lower part of that panel. First, the registration must be performed. This process works in two different ways. If pre-defined features are used, then these are registered with the ones marked on the face mesh as described above. After this, another registration is performed between the automatically found feature coordinates and the vertices of the face mesh. If pre-defined features are not used, then the registration is performed once just between the features and vertices of the face model.

In our experiments, it is realized that the registration requires lots of running time since the face mesh we use is a very detailed one, so we decided to use pre-defined features to improve the speed. Since there are much less pre-defined features than the ones found automatically, the registration of them requires less time. Also this provides an approximate registration which brings all features close to the model. The second registration is still needed because the features are not close enough to the model and also they are not matched with corresponding vertices which is required for the deformation process. Despite an extra registration, it requires very little running time when compared with the one which does not use pre-defined features. Using these additional features, the algorithm runs on 30 iterations while 95-100 iterations are needed when they are not used. These additional features are also used to deform the face mesh so that more satisfiable results are obtained.

Another improvement in speed is achieved by using a less detailed face mesh with 827 vertices. It still requires 80-85 iterations but the time in each is reduced dramatically. Unfortunately the deformation could not be applied to this mesh which will be explained later, so that another registration is again applied with the detailed

face mesh knowing that it will require again less time. Because first registration is initially made an approximate alignment again just like the one in which we used pre-defined features.

Second, the deformation must be performed by using a Radial Basis Function (RBF). User can change the type of the RBF's which are Linear and Gaussian functions. The result of the interpolation is not satisfiable because of the noises occurred on the face model at this step. So a smoothing process is needed to eliminate these noises. When a less detailed face mesh is used there are still noises that are not ignorable so a smoothing is necessary for this model, too.

Finally, the smoothing process must be performed. User can choose the threshold which defines the amount of the smoothing. This step greatly reduces the noise and provides a smooth surface but it results shrinkage on the face model. The algorithm automatically enlarges the model according to the amount of shrinkage which is determined by the boundaries of the features. When the less detailed face mesh is used, the amount of the shrinkage becomes so high that the face geometry is disrupted badly. So a second registration is applied with a detailed one as described above and the resultant mesh is used for the rest of the process.

On the same panel, user can also enlarge the model manually by defining the appropriate parameters.

6.5.3 Texture Generation

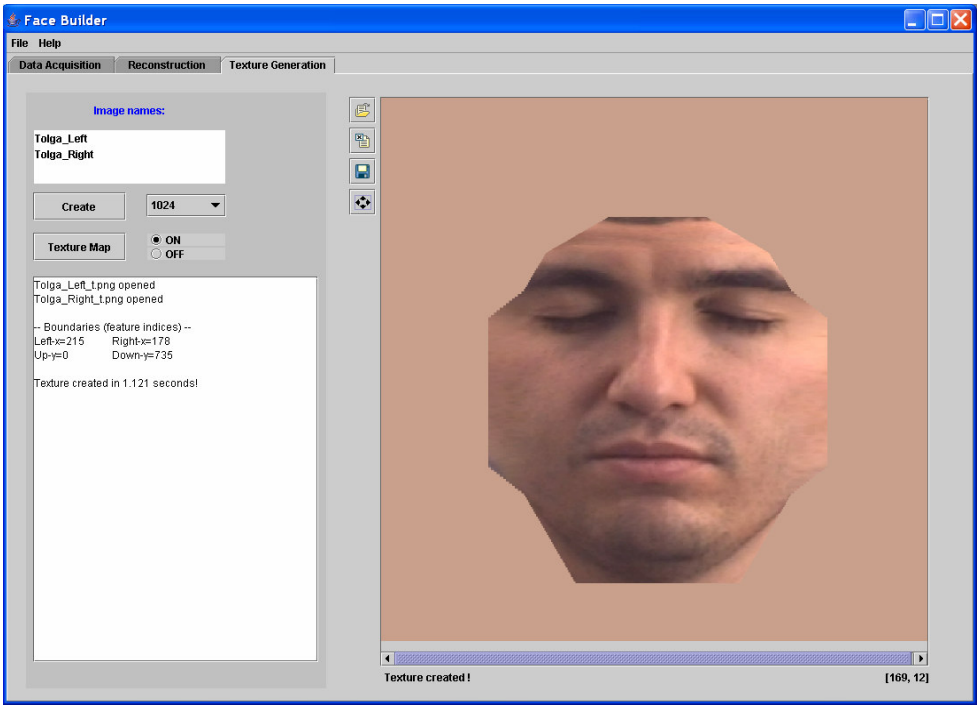


Figure 6.5: The user interface of the Texture Generation Module

Figure 6.5 shows the texture generation module which is developed to collect all the necessary information needed and then construct the texture image. Data used for texture generation is listed below:

- Stereo image files without projections
- Texture map
- Deformed face mesh

Since some of the information about the data listed above may have been changed in the previous steps, this module automatically collects the most recent versions of the listed data.

The main idea is gathering color values from the initial images and putting them to their appropriate locations on the texture image which is also transformed to fit the texture map.

The texture is thought to be a square image and the length of the edge is initially set as 1024 pixels. The application also allows user to change it.

CHAPTER 7

EXPERIMENTAL RESULTS

The results of the FaceBuilder application on different subjects are explained in this chapter. The application is executed with 2 images having dot patterns projected on them as shown in Figure 7.1. You can also see the patterns obtained from these images in the same figure.

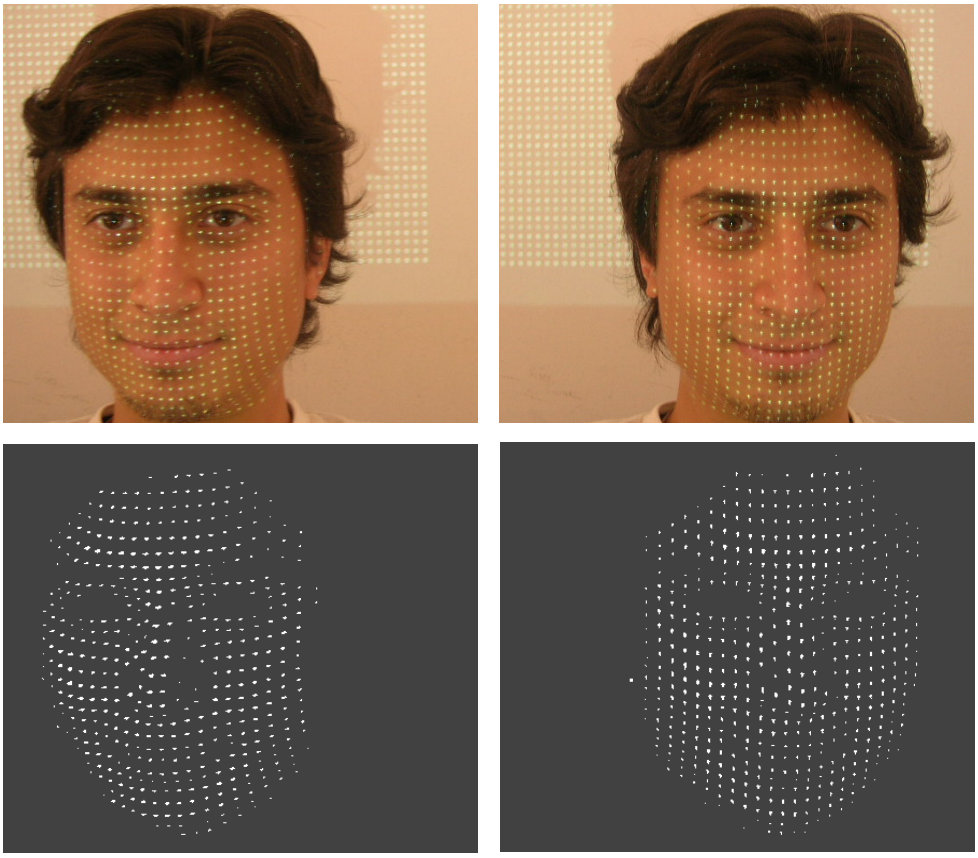
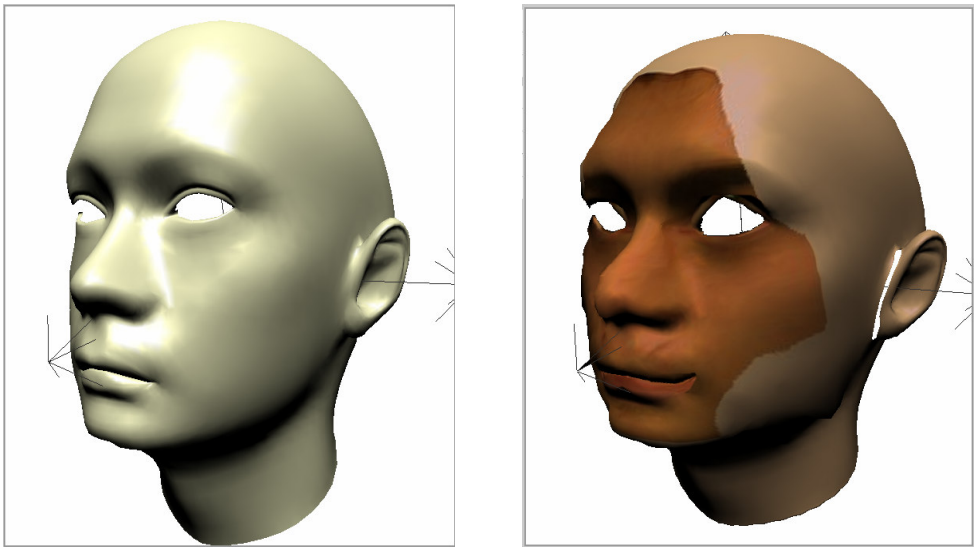


Figure 7.1: Face images with dot patterns

Figure 7.2 shows the result of the application when applied to the subject in Figure 7.1.



(a) initial model (b) reconstructed 3D model

Figure 7.2: 3D model of the subject in Figure 7.1

As shown in Figure 7.2, the eyes, mouth and the nose is the most alike parts of the face when compared with the face of the given subject. The eyes become round, the mouth is deformed to be seen like smiley and the nose tip become more wide. You can see the thickness of the face is also changed when you look carefully. Anyway, there are other examples which show the change on the thickness of the face better with different views of the subject. One of the best result showing the change on the thickness is showed in Figure 7.3

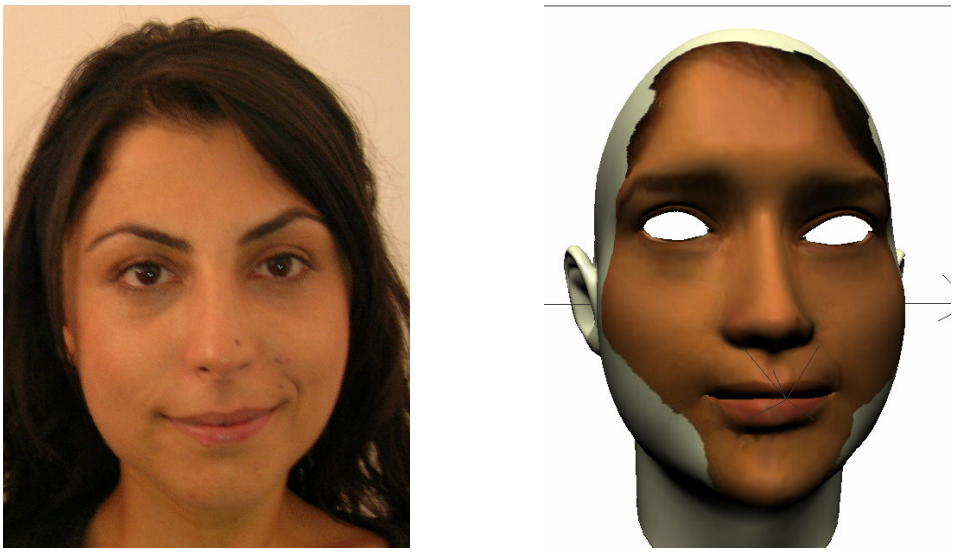


Figure 7.3: 3D model of the subject

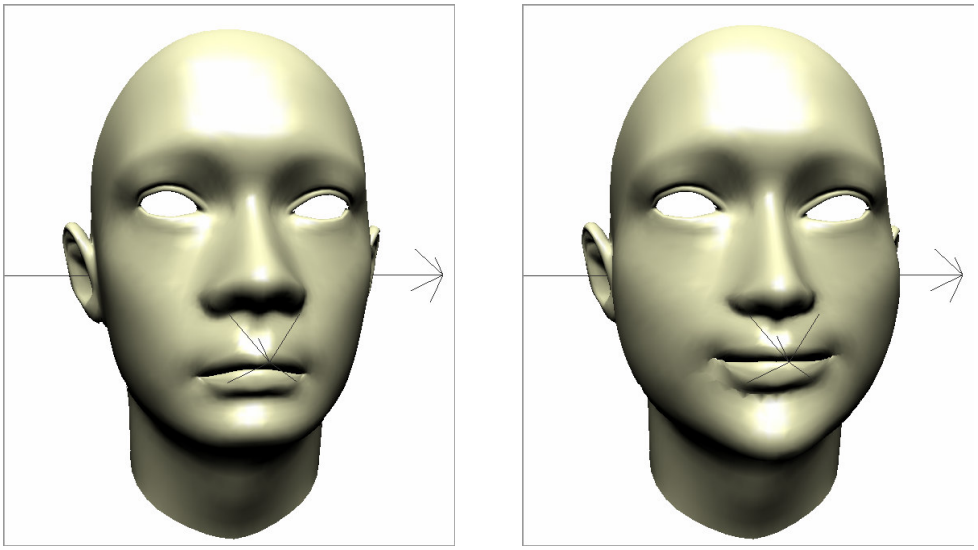
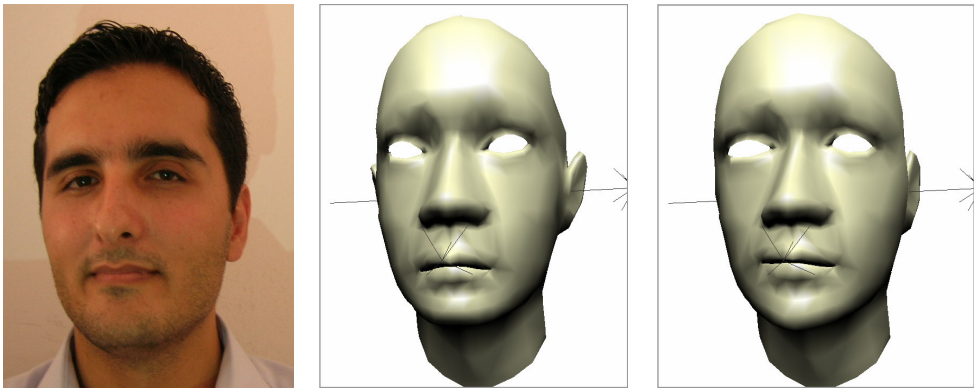


Figure 7.4: 3D deformation of the initial face mesh for the subject

Figure 7.3 clearly shows the strongest point of the application which is the identifying the thickness of the face. This change can be seen more clearly in Figure

7.4, which shows the result of the deformation of the same subject displayed in Figure 7.3.

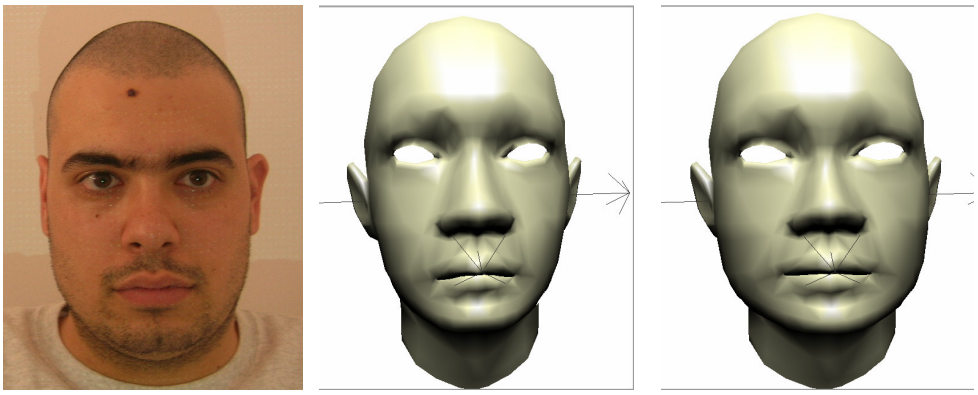


(a) face image (b) initial face mesh (c) deformed face mesh

Figure 7.5: The result of the deformation for a thin face

Figure 7.5 shows the results of another subject which has a thin face. Since the initial face mesh is already a thin one, the application didn't change the thickness much. It made small changes which are not realizable easily.

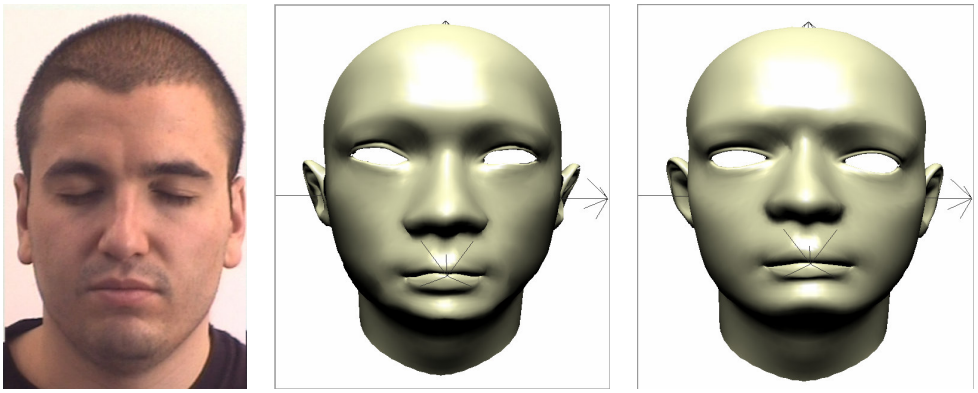
Here you can find another example in Figure 7.6. This time the thickness is only occurred on cheeks around the mouth. The 3D model has not a smiley pose like the pose which the initial image has.



(a) face image (b) initial face mesh (c) deformed face mesh

Figure 7.6: An example for the thickness occurred around the mouth

Remember that we have used pre-defined features to improve the performance. Using them on the deformation step also improved the result as shown in Figure 7.7. For the subject in (a), the results of the deformation are shown when these features are used (c) and not used (b).



(a) (b) (c)

Figure 7.7: Deformation with 18 pre-defined features

CHAPTER 8

CONCLUSION

3D computer graphics is a rapidly developing area as a result of the development of faster computers and graphic cards. Most of the computer games are now developed in 3D and the number of 3D animation films is raised nowadays. But this technology is usually used on these areas only. Ignoring the limited 3D environment development tools on some computer games, home users are not able to generate their own 3D environment. This study tries to generate 3D face models with a pair of constricted photographs.

There are very little products generating realistic 3D models and most of them are using very complicated methods and extremely expensive hardware like Pixar Animation Studios. This study presents a face modeling with a little help of extra constraints. A projector, a pair of stereo cameras and a computer is necessary to generate the face model of a subject.

In this thesis, 3D face model reconstruction by morphing an initial 3D model with a pair of photographs captured from stereo cameras is performed with some little problems. The initial photographs are captured while a pattern having an array of points was being projected on subject's face. The information needed to deform the initial model is gathered from this projection. A pair of photographs without

projection is also processed to form a single texture image covering the deformed 3D face model.

It can be said that the application works well in determining the thickness of the faces. The results obtained are almost satisfactory as shown in previous chapters on different subjects. However, some improvements are needed as explained in the next section.

8.1 Future Works

Many additions and improvements can be introduced to the proposed study. These additions can both improve the speed and the reality of the model to be reconstructed.

First of all, a different type of projection pattern can be used. Instead of a dot-pattern, a stripe pattern may be preferred. This pattern may contain parallel stripes having different thicknesses and colors so that data acquisition part can be rebuilt without so much effort.

The main drawback in this study lies on the data acquisition part. An improvement in 3D feature coordinate calculation can be performed with the currently used pattern. Most of the problem results from the multiple point elimination part referred as making single after color depth decreasing. This elimination introduces some errors on 3D coordinate values which are not perceivable. Anyway, this can also be

overcome by using a different pattern such as the one mentioned above which contains parallel stripes. While tracking the points on the stripes, only the left-most pixels can be used on a thick line. No elimination or normalization will be needed if this operation is performed on every line as choosing the left-most point of the line on the current row. Also, if enough number of points is obtained, then there may be no need for a default 3D model to fit the 3D point cloud which contains the calculated 3D coordinates of the projected dots. Instead, a new model can be built by triangulation of these 3D points of our point cloud.

The second biggest drawback in this thesis is the process used in the model deformation. In fact, the rigid 3D registration does not guarantee to provide a good matching between 3D data points and the 3D model vertices. So, 3D coordinates of some vertices of 3D model may be reassigned with wrong 3D points which cause a bad result. At this point, it is necessary to add an extra step which performs a correct matching between all 3D points and the vertices of the mesh.

Again at the deformation step, our interpolation algorithm produces holes and peaks on the surface of the face mesh so we used a smoothing algorithm to get rid of them. Smoothing overcomes this well but it also introduces small data loss and shrinkage. A more sophisticated interpolation can be performed, for instance building a new function depending on the feature neighborhood relations, so that there will be no need for an extra algorithm to perform smoothing.

The texture generation part needs unsurprisingly some improvement. Actually it will be better to use a different method for this work. But small improvements may yield

better textures. For example, more photographs can be used which shall be captured from profiles, back and top of the face. When the information gathered from these images are added as a weighted sum to the resultant image, it will surely provide a whole and better texture. Again, a pixel based coloring algorithm can be introduced by using the camera parameters to give better texture images.

These future works can improve the whole process greatly and there are surely new, more accurate methods waiting to be discovered for face model generation.

CHAPTER 9

REFERENCES

[1] A.-N. Ansari and M. Abdel-Mottaleb, "3D Face Modeling using two Orthogonal Views and a Generic Face Model", in ICME 03, Proceedings of International Conference on Multimedia and Expo, 3:289-92, 2003

[2] Achermann B., H. Bunke, "Classifying Range Images of Human Faces with Hausdorff Distance", in Proc. ICPR, pp. 809-813, 2000

[3] Atick J.J., Griffin P. A., and Redlich A. N., "Statistical Approach to Shape from Shading: Reconstruction of 3D Face Surfaces from Single 2D Images", Computation in Neurological Systems, vol. 7, no. 1, 1996

[4] Besl P., McKay N., "A Method for Registration of 3-D Shapes", IEEE Trans. PAMI, vol. 14, no. 2, pp. 239-256, 1992

[5] Besl P. J., "Geometric Modeling and Computer Vision", Proc. IEEE, vol. 76, no. 8, pp. 936-958, Aug. 1988

[6] Beumier C., Acheroy M., "Automatic 3D Face Authentication", Image and Vision Computing, vol. 18, no. 4, pp. 315-321, 2000

[7] Beumier C., M. Acheroy, “Automatic Face Authentication from 3D Surface”, In Proceedings of British Machine Vision Conference, 1998

[8] Blanz V. and Vetter T., “A Morphable Model for the Synthesis of 3D Faces”, Computer Graphics Proc. SIGGRAPH ’99, pp. 187-194, 1999

[9] Blanz V. and T. Vetter, “Face Recognition Based on Fitting a 3D Morphable Model”, IEEE Trans. PAMI, vol. 25, no. 9, pp. 1063-1074, 2003

[10] Chen Z., Ho S.-Y., Tseng D.-C., “Polyhedral Face Reconstruction and Modeling from a Single Image with Structured Light”, IEEE Trans. on Systems, Man, and Cybernetics, vol. 23, no. 3, May/June 1993

[11] Frederic Pighin, Jamie Hecker, Dani Lischinski, Richard Szeliski, David H. Salesin, “Synthesizing Realistic Facial Expressions from Photographs”, in Computer Graphics (Proc. SIGGRAPH'98), pp. 75-84, (1998)

[12] Fua P., “A Parallel Stereo Algorithm that Produces Dense Depth Maps and Preserves Image Features”, Machine Vision Applications, 6(1), 1993

[13] Gene H. Golub, Taubin G., Tong Zhang, "Optimal Surface Smoothing as Filter Design," Research Report, ECCV, March 1996

[14] Guenter B., Grimm C., Malvar H., and Wood D., “Making Faces”, Proc. SIGCRAPH, July 1998

[15] Hsu R. L. and Jain A. K., ”Face Modeling for Recognition”, Proc. ICIP, Greece, Oct. 7-10, 2001

[16] Hutton T. J., Buxton B. F., and Hammond P., “Automated Registration of 3D Faces Using Dense Surface Models”, in Proceedings of the British Machine Conference, pp. 439-448, 2003

[17] Jarvis R. A., “A Perspective on Range Finding Techniques for Computer Vision”, IEEE Trans. Pattern Anal. Machine Intell., vol. PAMI-5, pp. 122-139, Mar. 1983

[18] Lee Y., Terzopuolos D., Waters K., “Realistic Modeling for Facial Animation”, Proc. SIGGRAPH, pp. 55-61, LA, Aug. 1995

[19] Lengagne R., Fua P., Monga O., “3D Face Modeling from Stereo and Differential Constraints”, Proc ICAFGR, pp. 148-153, April 1998

[20] Lengagne R., Tarel J. P., Monga O., “From 2D Images to 3D Face Geometry”, in Proceedings of the 2nd International Conference on Automated Face and Gesture Recognition (FG’96), Killington, Vermoni, U.S.A., 1996

[21] Lowe D. G., “Fitting Parameterized Three-Dimensional Models to Images”, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 13, no. 5, pp. 441-450, 1991

[22] Noh J.Y., Fidaeo D. and Neumann U., “Animated deformations with radial basis functions”, Proc. ACM symposium on virtual reality software and technology, Seoul, Korea, 2000

[23] Wang Y. F. and Aggarwal J. K., “An Overview of Geometric Modeling Using Active Sensing”, IEEE Cont. Syst. Mag., vol. 3, no. 2, pp. 5-13, June 1988

[24] Waxman A. M. and Moigne J. Le, “Projected light grids for short range navigation of autonomous robots,” in Proc. 7th Int. Coni Pattern Recog., July 1984, pp. 203-206.