3D FACE RECONSTRUCTION USING STEREO IMAGES AND STRUCTURED LIGHT

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

BY

AHMET OĞUZ ÖZTÜRK

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

NOVEMBER 2007

Approval of the Thesis

"3D FACE RECONSTRUCTION USING STEREO VISION"

Submitted by **AHMET OĞUZ ÖZTÜRK** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen

Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İsmet Erkmen

Head of Department, **Electrical and Electronics Eng.** _____

Prof. Dr. Uğur Halıcı

Supervisor, **Electrical and Electronics Eng.** _____

Examining Committee Members

Prof. Dr. Kemal Leblebicioğlu
Electrical and Electronics Eng., METU _____

Prof. Dr. Uğur Halıcı
Electrical and Electronics Eng., METU _____

Assist. Prof. Didem GÖKÇAY
Department of Medical Informatics, METU _____

Assist. Prof. İlkay Ulusoy
Electrical and Electronics Eng., METU _____

Mehmet Dikmen (M.S.)
Computer Eng., Başkent Uni. _____

Date: 28.11.2007

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 : Ahmet Oğuz ÖZTÜRK

Signature :

ABSTRACT

3D FACE RECONSTRUCTION USING STEREO IMAGES AND STRUCTURED LIGHT

Öztürk, Ahmet Oğuz

M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Uğur Halıcı

November 2007, 93 pages

Nowadays, 3D modelling of objects from multiple images is a topic that has gained great recognition and is widely used in various fields. Recently, lots of progress has been made in identification of people using 3D face models, which are usually reconstructed from multiple face images. In this thesis, a system including stereo cameras and structured light is built for the purpose of 3D modelling. The system outputs are 3D shapes of the face and also the texture information registered to this shape. Although the system in this thesis is developed for face reconstruction, it is not specific to faces. Using the same methodology proposed in this study 3D reconstruction of any object can be achieved.

Keywords: 3D reconstruction, stereo camera, structured light.

ÖZ

STEREO GÖRÜNTÜ VE YAPISAL IŞIK KAYNAĞI KULLANARAK YÜZÜN 3 BOYUTLU GERİ ÇATIMI

Öztürk, Ahmet Oğuz

Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Uğur Halıcı

Kasım 2007, 93 sayfa

Birden fazla fotoğraf kullanılarak cisimlerin 3 boyutlu modellemesinin yapılması günümüzde gittikçe daha da ilgi çeken ve birçok alanda kullanılan bir konu olarak karşımıza çıkmaktadır. Son dönemde çoğunlukla birden fazla fotoğraf kullanılarak geri çatımı yapılan 3B yüz modelleri kullanılarak, kimlik tespiti alanında ilerleme kaydedilmiştir. Bu tezde, stereo kameralar ve yapısal ışık kullanılarak insan yüzünün 3B geri çatımını yapan bir sistem yapılmaya çalışılmıştır. Sistemin sonuçları 3B yüz şekli ve bu şekille ilişkilendirilmiş olan dokudur. Bu tezdeki sistem, yüzün geri çatımı için geliştirilmiş olmasına rağmen yüze özel değildir. Yüze uygulanan işlemlerin aynısı uygulandığında herhangi bir cismin 3B görüntüsünü çıkarabilecek durumdadır.

Anahtar Kelimeler: 3B geriçatım, stereo kamera, yapısal ışık.

ACKNOWLEDGEMENT

I would like to express my deepest gratitude and appreciation to my supervisor Prof. Dr. Uğur Halıcı, my co-supervisor Assist. Prof. İlkay Ulusoy who inspired, encouraged and supported me at all levels of this study that is prepared in METU Computer Vision & Intelligent Systems Research Lab. This thesis study is made as a part of Tübitak Project 105E128.

I would like to thank to Erdem Akagündüz who helped me in all technical topics and encouraged me after each small improvement in my study, to Ömer Eskizara for his help in user interface development and camera calibration, to Nesli Erdoğmuş for her help in post processing phases, to Osman Selçuk Şentürk who believed in me most and supported me during the whole study, to Berkan Solmaz, Erkan Şentürk, Emre Ün, and Özden Akçay for their help in preparing face database and making face scanning experiments, and to Serkan Ünal and my old friend Saygın Savran for their help in translation.

The greatest thanks go to my family members and all my friends whose names were not mentioned above, for their infinite support.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

3D modeling of objects using images is a topic that has gained great recognition and is widely used in various fields. In an attempt to find the cheapest, quickest and most accurate one, different scan methods have been used. But the goal of achieving a comprehensive perception of the world through computer vision is still far from being attained. The main reason is that the way the human brain works is not known completely, and computer vision community is still dealing with low-level problems related to vision, like color and texture perception, and intermediate-level problems, like motion detection, depth and shape acquisition and object recognition. Another reason why it is hard to emulate the human perception is that humans take advantage of active perception capabilities to optimize perceptive tasks. For example, we can converge or diverge our eyes, move our head, change our point of view, etc. [1]. The use of these capabilities to optimize perception tasks is another important research field in computer vision known as active vision [2].

While object scanning has been widely used in robotics and industrial design fields, 3D reconstruction and recognition of human face has made a great impact on various fields, especially in the biometrics and security fields. To construct a 3D model of a real face, the 3D shape information obtained by reconstruction of human face is covered with

texture, and the produced models could also be used in computer games and animations [14]. Face information is usually kept in a 3D point cloud structure, which is a set of 3D point coordinates to form the 3D shape of the scanned object. Face scanning is more complex than the scanning of most of the other objects due to the fact that the protruding regions on the face, such as the nose and the eyes, cause occlusions.

The main methods used for face scanning are laser sensor, multiple image acquisition technique and structured light technique, which are described below:

Laser sensor: The technique is based on computing the distance to a surface by measuring the round-trip time of a light beam, which is emitted from a laser source. Reflected laser is detected by a sensor, and the depth information is calculated by using the speed of laser and the difference of the times the beam is emitted and sensed. This is the most accurate but the most expensive and the slowest scan technique, which is common for scanning static objects, usually in industry where accuracy is very important [6]. The acquisition of a single 3D head scan can take more than 30 seconds. As it is hard to keep a human face fully stable during 30 seconds, the laser scanning is not the most appropriate method for 3D face reconstruction. In addition to this, most of the laser types are harmful for the eyes, which makes it impractical for face scanning purpose.

Multiple image acquisition technique: This technique is based on taking images of an object from multiple cameras and calculating the coordinates of each point in 3D space using the stereo analysis, which is based on the disparity between the corresponding points in the image pairs. Cameras should be calibrated and positioned before scanning. Stereo acquisition technique is a special case where only two parallel cameras are used. This technique has the lowest cost and the highest ease of use, but cannot be applicable

to every field since finding matching points in each image is not an easy task, especially for the areas having no feature points. Also the determination of corresponding image points between the stereo image views is a crucially important step in stereo image analysis. Also perspective effects, occlusions, photometric variation and inherent ambiguities make disparity estimation very difficult [3]. Occlusion is the most important problem among these difficulties. This problem can be overcome by using multiple cameras. But this time, finding corresponding points between multiple cameras becomes a more difficult task.

Structural (or Structured) light technique: This technique is based on imaging the object while projecting a structured light pattern on the object, and then the depth information is calculated by using the distortion of the light pattern on the surface of the scanned object. In this system, a single light beam is projected on the object and a cross section scan from the top to the bottom of the object is taken. In order to obtain the 3D point cloud of the object, all the cross sections of the object are found either by rotating both the light and camera around the object, or by rotating the object around itself, and combining the obtained cross section scans [7]. This technique can be said as the average of the previous two methods in terms of expense, ease in use and reconstruction speed.

The lately mentioned two techniques, stereo acquisition technique and structured light technique, are the most commonly used methods in face scanning. In fact the method proposed in this study uses stereo cameras and structured light together and combines the most appropriate properties of these two methods [8].

In this thesis study, structured light method is used to determine the homologous points between stereo images, for disparity calculation. Homologous points are the points in a stereo image couple that have the same coordinates in real world. A structured light system is based on the projection of a single pattern or a set of patterns onto the

measuring scene, which is imaged by a single camera or a set of cameras. The patterns are specially designed [9] and vaious methods are used for constructing different structured light patterns. In this thesis study, time multiplexing method is used where a set of patterns is successively projected onto the measuring surface [10], [11]. During this projection period, the object to be scanned shall be stationary.

1.2 The Scope of the Thesis

In this thesis, a combination of stereo acquisition and structured light techniques are used. Stereo cameras capture images of the scanning object while structured light is emitted from the projector. Structured light is used to match the homologous points between two images that enable stereo analysis. Some post-processing techniques are applied on the final 3D point cloud, and 3D model of the face with color texture is obtained in ".obj" file format as an end product. This ".obj" file can be used in various visualization tools such as Polyworks Imview, Object Viewer, etc., in order to visualize the final model.

The block diagram of 3D Scanner System is given in Figure 1.1.

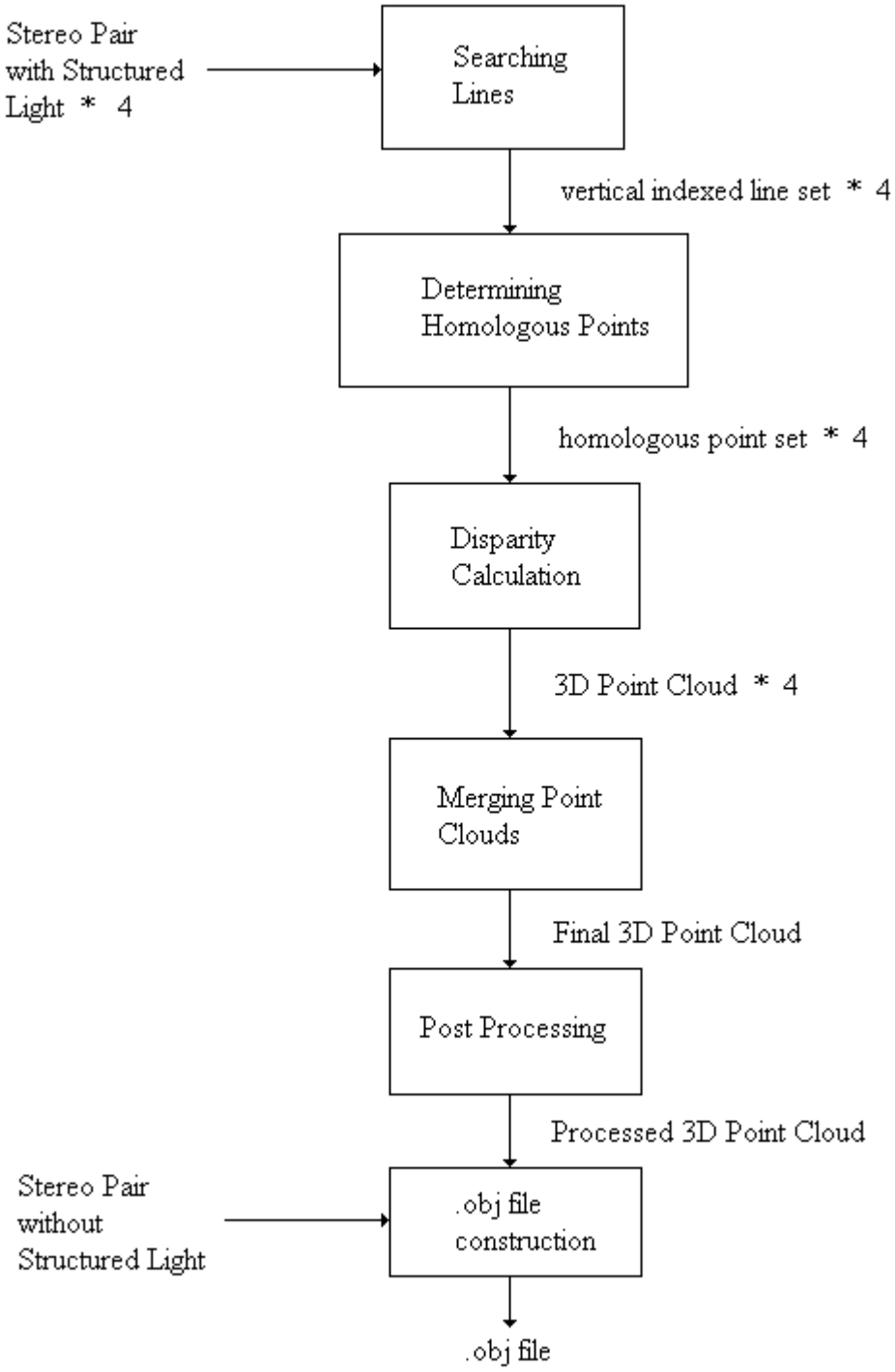

Figure 1.1: Block Diagram of 3D Scanner System

The process consists of following steps:

1. Cameras are calibrated and placed in parallel to each other. Projector is setup such that a sliding, black and white lined texture is projected. Cameras are placed on the projector such that the structured light is projected from the middle of the cameras, and the object to be scanned is placed in a position that is in the sight of view of the cameras, and is at a distance where the texture is projected on the object optimally. This usually corresponds to a distance of 40 cm. from the projector/camera system.
2. Sliding texture is projected on the object and a pair of images is captured from both cameras for each texture, which changes its pattern with one-second intervals. Then images without projection are taken from both cameras simultaneously.
3. Stereo image pairs taken with texture projection are analysed one by one. Analysis phase can be summarized as follows:
    i) The images obtained with texture projection are converted to grayscale format. The peak points of the light intensity are detected on the face.
    ii) Detected points are combined from top to bottom to construct a line and stored with a specific format.
    iii) After applying the same procedure to both of the images, depth information of each point is calculated by using the disparity of the points on the line structures.
4. 3D point clouds found as a result of the analysis of each stereo image pair are merged to get a final 3D point cloud.
5. Post-processing such as hole filling, linearizing the point coordinates, spike removal, smoothing is applied on 3D point cloud to obtain the final shape.
6. 3D point cloud is converted into ".obj" file format, where texture information is also included.

3D Scanning consists of; capturing the images, storing these images in bitmap format, analyzing the stereo image couples, and obtaining 3D point cloud. This part is implemented in Microsoft Visual C++ 6.0 including a user interface. This part of the software is named as 3DFaceScanner program. The design and the functionalities of 3DFaceScanner are given in Appendix-A in details and user manual of the program is given in Appendix-B.

Scanning devices typically produce a dense set of surface points, where each point samples a 3D position and possible additional attributes such as normal information, color, or material properties. Depending on the specific acquisition method, a number of scanning artifacts can occur like holes, noise and spikes [12]. These artifacts shall be removed or reduced in post processing phase to obtain a better 3D model [13]. In this thesis, post processing includes filling holes, spike removal, smoothing the surface and converting the final 3D point cloud into ".obj" file format. ".obj" file format is a widely used format for 3D models. Properties of ".obj" file format is explained in Appendix-C. Post processing part is implemented by using MATLAB. Actually MATLAB code is automatically generated by 3DFaceScanner program with the person's name ('person's name'.m). Person's name is the field to be filled in 3DFaceScanner user interface.

1.3 The Organization of the Thesis

The organization of the thesis is as follows:

- Chapter 2 gives background information about scanner methodologies.
- Chapter 3 gives the necessary information related to image capturing.
- Chapter 4 tells the procedure about finding the lines on the face.
- Chapter 5 gives information about how 3D point cloud is obtained.
- Post-processing methods are explained in Chapter 6.
- Chapter 7 includes the experimental results of the study.
- Chapter 8 includes conclusion.

- Appendix A gives information about definition of source code.
- Appendix B is the user's manual for the 3D scanner user interface.
- Appendix C gives information about ".obj" file format.

CHAPTER 2

BACKGROUND

2.1 Introduction

There are several techniques for reconstructing 3D images as explained in introduction part. The technique used in this study is a special case of multiple image acquisition technique by using structured light to find homologous points between the images. This chapter gives background information about multiple image acquisition techniques and different usage methods of structured light.

2.2 Techniques for 3D Modelling by Using Multiple Image Acquisition

The techniques for developing 3D models from multiple images can be divided into four [24]:

2.2.1 Binocular Stereo with parallel cameras

Stereo correspondence between stereo images is the technique applied in this thesis study. This technique is based on capturing two stereo images and developing 3D model by using disparities between homologous points in stereo images. The technique can be divided into two steps that are: rectification of stereo images and stereo matching. A 2D

disparity map is then generated for 3D reconstruction [15], [16]. Then 3D model can be obtained and depth values of each point in disparity map can be achieved by disparity calculation.

2.2.2 Binocular Stereo using Triangulation

Triangulation is the method to find the 3D coordinates of a point in real world by using multiple images captured from calibrated cameras where cameras do not need to parallel to each other. Three main steps are involved in binocular stereo technique. The first step is to calibrate the cameras; the second step is to find the correspondence between the stereo-pair images and the last step is to calculate the 3D coordinates of the corresponding points in the images by triangulation technique [17].

2.2.3 Photometric Stereo

Photometric stereo gives the ability to estimate local surface orientation by using several images of the same surface taken from the same viewpoint but under illumination from different directions. The theory was developed by R.J. Woodham [18]. For Lambertian surfaces, a surface normal can be determined if the considered surface point is illuminated from three or more light sources. Photometric stereo technique allows reconstruction of a 2.5-D model. The reconstruction accuracy depends on the quality of the generation of the surface normal and the transformation from the surface normal to the depth map [19].

2.2.4 Orthogonal Views

This technique is based on developing a 3D point cloud by using two images, one captured from the front and one captured from the side of the scanned face. 3D coordinates of the face points, visible in both images, are found by using x and y-coordinates of the front view, and z-coordinates from the side view. Changes in illumination between the front and side views make it difficult to automatically find corresponding pixels in both images. Research has been focusing on extracting similar features such as the eyes, eyebrows, lips, nose and mouth, which can be extracted using computer vision techniques. These features can then be mapped to a 3D generic face model to reconstruct a 3D face [22].

2.3 Stereo Imaging and Disparity Calculation with Parallel Cameras

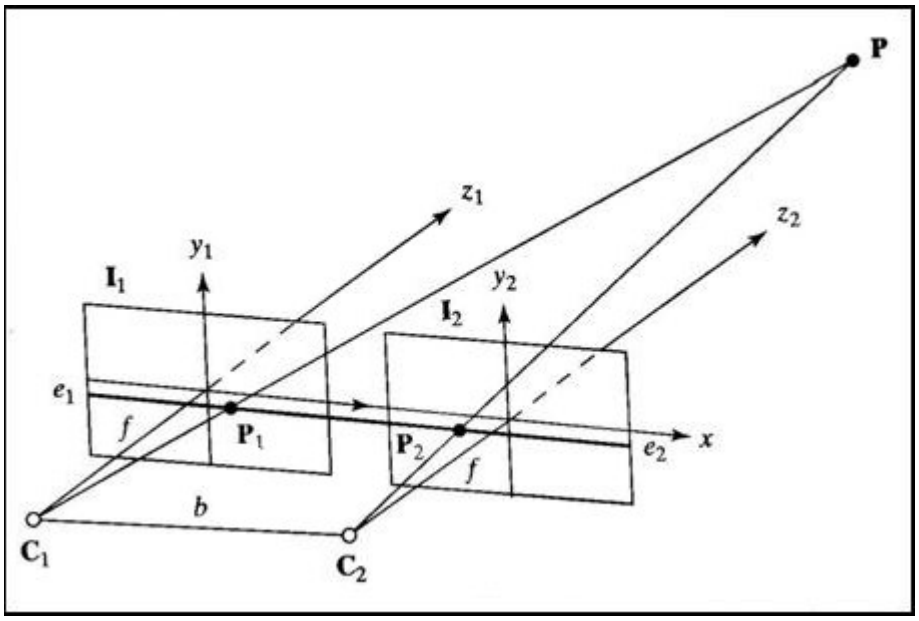

Figure 2.1: Epipolar geometry for parallel cameras

For parallel stereo pinhole cameras, a condition as in Figure 2.1 occurs. In the figure; the focal length of the cameras (f), the camera center points (C1 and C2) in the image planes (I1 and I2), the image points (of three-dimensional point P) P1 and P2 and baseline distance between cameras (b) are known. According to the figure, point P has the same y-coordinates and different x-coordinates in the image plane of both cameras [26]. This property will be used during determination of homologous points and disparity calculations. Triangulation is not needed since the cameras are parallel. Depth values can be found by using disparity calculations.

Let P1 = (x1, y1) and P2 = (x2, y2) be homologous points. In parallel stereo cameras y1 is equal to y2 and disparity is determined as:

$$d = x_1 - x_2 \qquad \text{(Eq. 2.1)}$$

Depth z at the image coordinate (x, y) can be calculated from given disparity d for that coordinate, the cameras' focal length f and the the distance between the cameras b as follows [4]:

$$z(x, y) = bf / d(x, y) \qquad \text{(Eq. 2.2)}$$

2.4 Stereo Matching with Structured Light

A coded structured light system is based on the projection of a single pattern or a set of patterns onto the surface, which is imaged by a single camera or a set of cameras. The patterns are specially designed so that codewords are assigned to a set of pixels [9]. Projecting structured light pattern techniques differ in how each point is identified in the pattern.

Structured light techniques can be divided as follows:

2.4.1 Time Multiplexing Strategy

Time multiplexing is one of the most commonly used strategies that is based on temporal coding where, a set of patterns is successively projected onto the scanned surface. Also this is the technique used in this thesis study. High accuracy can be achieved in measurements by using time multiplexing strategy and successive light patterns. This is due to two factors: first, since multiple patterns are projected, the codeword basis tends to be small and therefore a small set of primitives is used, being easily distinguishable among each other; second, a dense 3D point cloud can be achieved, since homologous point number increases with increasing structured light pattern number [25]. If the objective is to obtain high accuracy and dealing with static scenes, time-multiplexed structured light is the most suitable, but the performance decreases as the scanned object moves even slightly while capturing the images, since the capturing process should be repeated for each pattern [9]. So face shall be kept as static as possible to obtain the most accurate result.

2.4.2 Spatial Neighbourhood

The techniques in this group tend to concentrate the entire coding scheme in a unique pattern. Codification consists of identifying a set of points of the pattern with the information contained in a small neighborhood around each one of them. However, the decoding stage becomes more difficult since the spatial neighborhood cannot always be recovered and 3D errors can arise [23]. This technique is applicable to both static and dynamic scenes.

2.4.3 Direct Codification

This technique is based on creating a pattern so that every pixel can be labeled by the information represented on it. Thus, the entire codeword for a given point is contained in a unique pixel. In order to achieve this, it is necessary to use either a large range of color values or introduce periodicity [20]. In theory, a high resolution of 3D information can be obtained. However, the sensitivity to noise is very high because the "distance" between codewords is small. Moreover, the imaged colors depend not only on the projected colors, but also on the intrinsic color of the measuring surface. This means, in most cases, that one or more reference images must be taken. Therefore, these techniques are not typically suitable for dynamic scenes. Direct codification is usually constrained to neutral color objects or pale objects [21].

CHAPTER 3

IMAGE ACQUISITION

3.1 Introduction

The first step in this study is to capture the image of the face from both cameras simultaneously. Sliding texture is projected on the object continuously with one-second intervals and four pairs of images (one pair for each projected texture) are taken from both cameras with one-second intervals, then images without texture projection are captured from both cameras simultaneously which sums up to a total of ten images. This chapter can be analyzed in three parts, which are setup configuration, structure of the projected texture and image capturing.

3.2 Setup Configuration

The equipments needed to setup the configuration are two computers, a projector and two identical cameras. The cameras are connected to the computer that includes 3DFaceScanner user interface. Machanism of the system can be seen in Figure 3.1. Cameras are placed on the upper sides of the projector to avoid occlusion of light. These cameras are positioned in parallel to the floor having the same focus and capturing resolution. The distance between the cameras is set to 7.5 cm and the distance between cameras and the human face is set to 40 cm. This distance is optimized emprically. Other

computer is connected to projector in order to provide texture to the projector. With these conditions, configuration becomes ready to scan a human face.

Figure 3.1: Mechanism of the 3D Face Scanner system

3.3 Structure Of The Projected Texture

In this study, a structured light pattern, which is composed of vertical white lines on a black background, is selected. When this type of light is used, 3D point cloud, which is very dense vertically but sparse horizontally, is obtained. This is due to the fact that the depth values are calculated only for the points where light is projected. To obtain a denser point cloud, the structured light is slided four times horizontally with one-second intervals, and in each slide, images are captured from both cameras.

Structured light patterns projected on the object are given in Figure 3.2. Each pattern includes two white pixels and six black pixels in horizontal. First four are the slided structured lights. In each slide, each white line slides two pixels to the right, and after four slides, white lines turn to their initial positions. The last one includes no pattern and is used for capturing images without structured light.

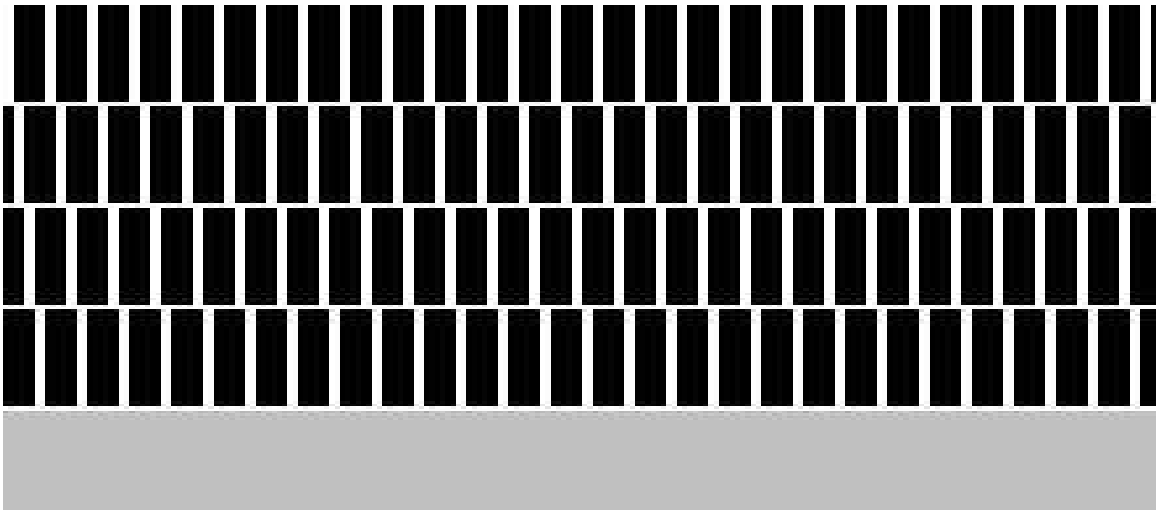

Figure 3.2: Projected structured light samples

3.4 Taking Image

The following process is to capture images by using the configured setup. Although structured light is used, it is in not necessary to place the setup in a dark area. Four pairs of images of the face were taken from both cameras with one-second intervals, while sliding texture was projected. A pair of stereo images is shown in Figure 3.3. Then the following image pairs were captured while texture was not projected. The purpose of capturing these last images is to take the image of the face without structured light texture in order to use them while obtaining the tissue texture of the real face.

With the ten images taken, that is four stereo pairs with structured light, one pair without structured light, the image acquisition part of the project is completed. The software developed in this thesis, which consists of analyzing bitmap files, reconstruction and post-processing, handles the rest of the study.

Figure 3.3a: Image taken from the left camera

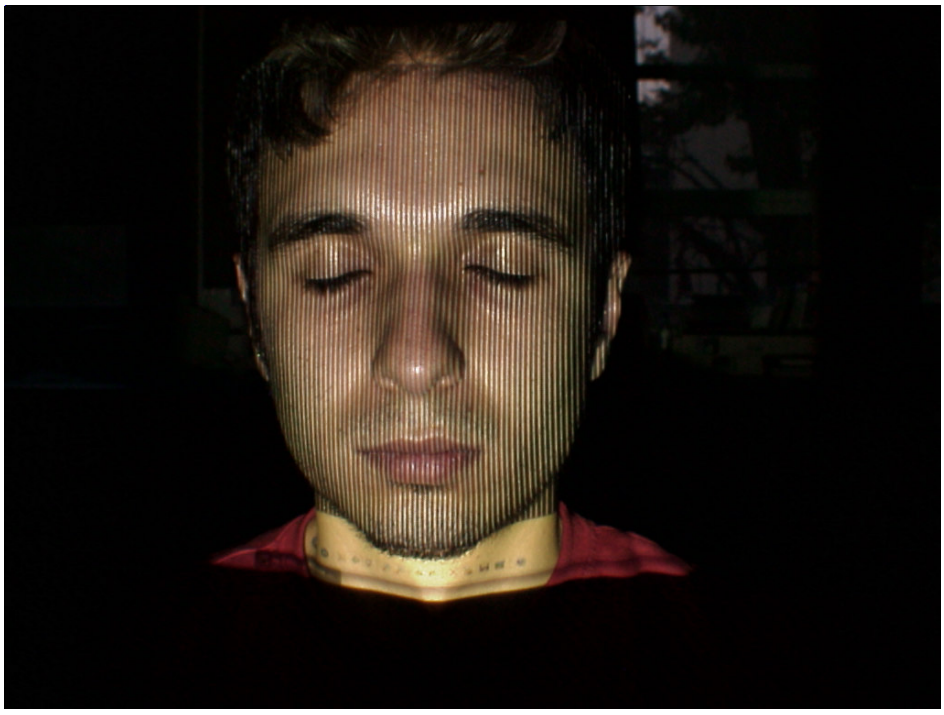

Figure 3.3b: Image taken from the right camera

CHAPTER 4

FINDING THE LINES ON THE FACE

4.1 Introduction

In the previous chapter, ten bitmap files were obtained. These were the images of the face, four taken from left, four taken from right with texture projected on the face and in addition stereo images taken without texture reflection.

In this chapter, the stereo bitmap files taken with texture reflection will be studied. The output of this chapter will be two structures, one for left image and one for right image separately, each of which stores the coordinates of the lines on the images. This process will be done for all stereo pairs. Bitmap files will be analyzed and line coordinates will be found separately, then the coordinate information will be combined to find the depth information, in the next chapter.

The procedure for finding lines on faces has two phases:
In the first phase, point locations will be found; in the second phase, points will be joined to construct the lines.

Whole chapter gives information about analyzing one pair of bitmap files. In the operation phase, the analysis will be repeated for four pairs, so the analysis will be done four times.

4.2 Finding Point Locations

The process in this part of the study is to find the highlighted points on the images and to store these points in a definite structure.

The original images before starting the process were as in Figure 3.3. Firstly, these images were turned into grayscale format as shown in Figure 4.1.

Then, the brightest points of the white lined texture were found on the images based on intensity thresholding. When the intensity values are read on a fixed horizontal line on the image, a sinusoidal like wave is obtained which is given in Figure 4.2.

The steps of the procedure to find the brightest points on horizontal lines on the bitmap is explained as follows:

1) The intensity values of the points on a horizontal line on the image is read from left to right and the coordinates of the points where intensity value is started to decrease are stored.
2) The same procedure as in Step1 is applied while reading from right to left this time.
3) The common coordinates found in Step1 and Step2 are the local maximum points for the intensity values. But in this point set, there may be some points, which have maximum intensity values locally, but do not have intensity values large enough to be a bright line point, so are not the desired points. To eliminate these points, a filtering was applied according to average intensity values.

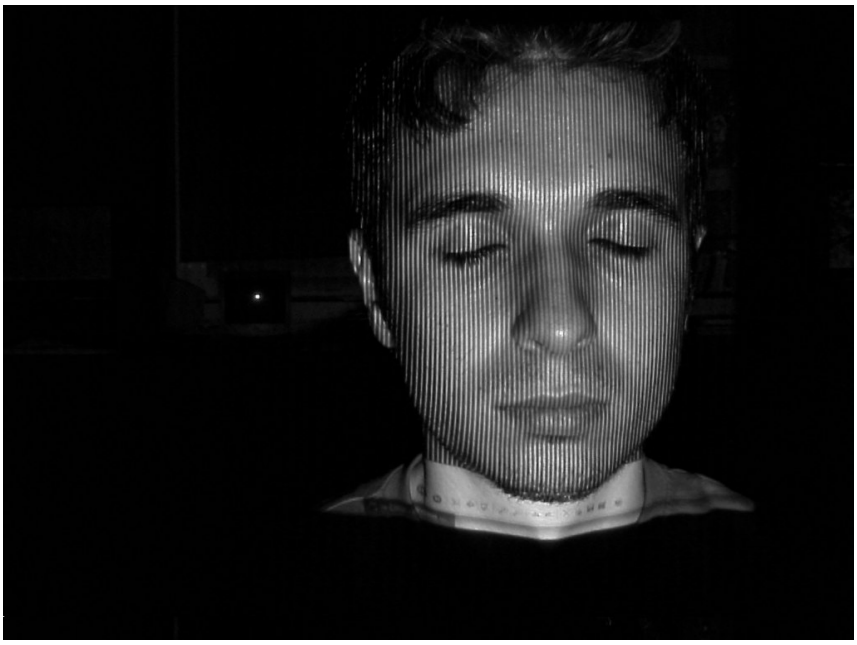

Figure 4.1a: Image taken from left camera in grayscale format

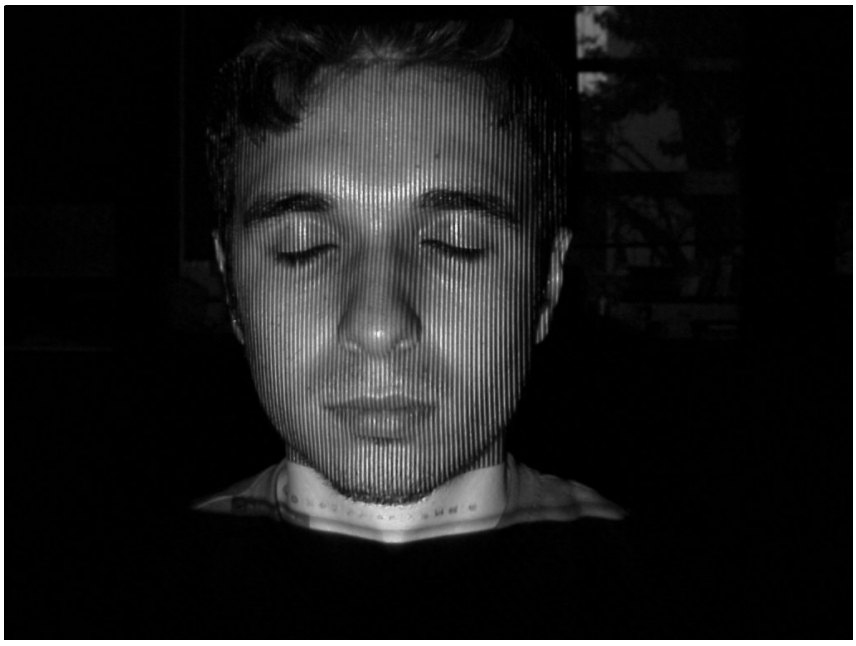

Figure 4.1b: Image taken from right camera in grayscale format

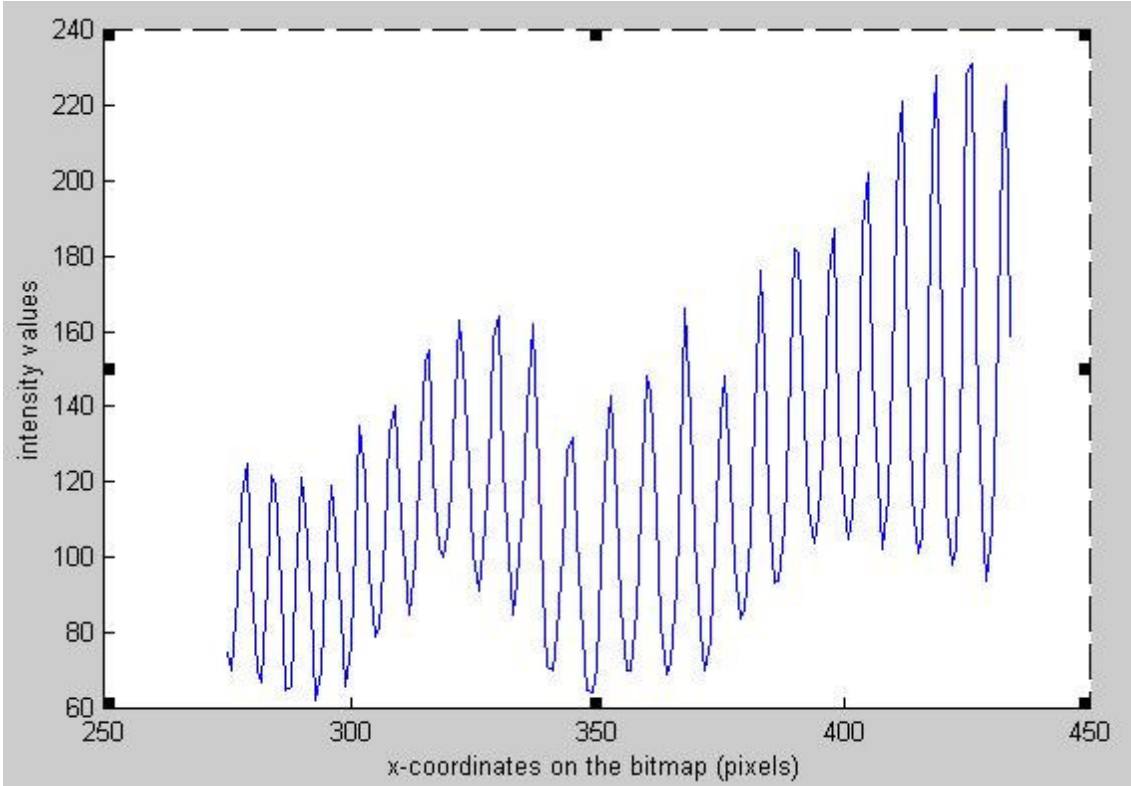

Figure 4.2: Intensity vs x-coordinate plot for the image seen in Figure 4.1b on horizontal line y = 390

After this process, images as in Figure 4.3 were obtained. These figures show the brightest points of the white lines, which were projected on the face. As seen in the figure, brightest lines on each horizontal line forms regular vertical lines on the surface of the object to be scanned.

Figure 4.3a: Image captured from left after “detect peak points algorithm” is applied

Figure 4.3b: Image captured from left after “detect peak points algorithm” is applied

4.3 Joining Points to Construct Lines

After point locations are determined in the first phase, these points will be combined to construct indexed vertical lines in the second phase. We can categorize this second phase into four steps: (1) determining the initial point, (2) initial line search starting from the initial point, (3) searching missing parts, and (4) finding the lines at both edges, which are not found during initial line search.

4.3.1 Determining Initial Point

Initial point is a point that is marked manually on the images corresponding to the same point on the face, so that in both bitmap files it is known that they correspond to the same 3D point thus they are corresponding matching points. This point is used as a starting point for line searching algorithm. For this purpose, inside the white lined texture to be projected on the face, a red point is placed. This red point is projected on the forehead area. After images are captured by both cameras, 3DFaceScanner user interface requests the user to set the initial points manually in both images by marking the red point on the forehead.

Defining an initial point helps overcome three main difficulties:

1) In this study, cameras are assumed to be ideally parallel to the floor and to each other, but because of several conditions (e.g. rougness on the floor), the y coordinate of the images taken from both cameras may not be exactly equal. When an initial point is set in both images, these points shall have the same y coordinates if the cameras are ideally parallel to the floor and to each other. But if there occured a deviation in the position of the cameras, by using the difference in the y coordinates of the initial points, the images could be registered.

2) Since the initial point is placed on the face, it could be a good starting point to start the line search. In both of the bitmap files, line search algorithm begins with the marked initial point.

3) One of the most important and the hardest part of 3D reconstruction topic is stereo matching. Setting an initial point provides great easiness for this process. Since the initial point is a point which is known as a homologous point in both images, after lines on the face are detected, comparison of the lines for disparity calculation is started with the initial point pair. The line on which the initial point takes place is called initial line and it has index 0. As we go left the index is decreased and as we go right the index is increased.

4.3.2 Initial Line Search

This is the first step of the line searching procedure. The aim of the initial line search is to find the continuous lines starting from the initial point in the forehead.

A simple procedure for the continuous line search is given below:

i) Starting from the given point (x, y), search towards top: if there is a point located at (x-1, y-1) or (x, y-1) or (x+1, y-1), repeat part (i) with the point found.

ii) Starting from the given point (x, y), search towards bottom: if there is a point located at (x-1, y+1) or (x, y+1) or (x+1, y+1), repeat part (ii) with the point found.

After the initial line is found, search will continue towards right and left side of the forehead. But when the lines on the face towards one side finish, search shall not continue.

The procedure for this operation is summarized below:

i) Using the “initial point”, find the “initial line”
ii) Set the “search point” to “initial point”
iii) Starting from the “search point” search to right to find another bright point and set “search point” to this newly found point
iv) Store the distance between two points
v) Find the vertical continuous line including the “search point”
vi) Go back to step (iii) and repeat the procedure between step (iii) and step (v), 5 times to calculate an average distance between lines on the object.
vii) Continue point and vertical line search, until the distance between two points are larger than twice the average distance we calculated.

After this part, continue the same procedure towards the left side of the forehead to complete the initial line search and continuous lines, which fall on the face towards the right and left sides of the initial point shall be found. After initial line search, the images given in Figure 4.4 were obtained.

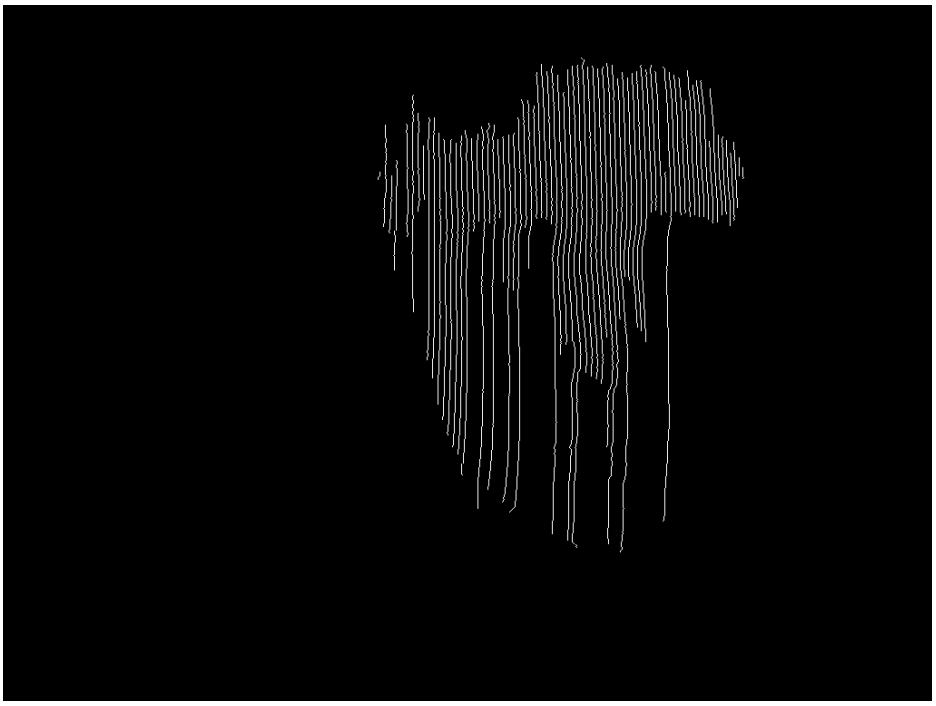

Figure 4.4a: Image taken from left after “initial line search” is applied

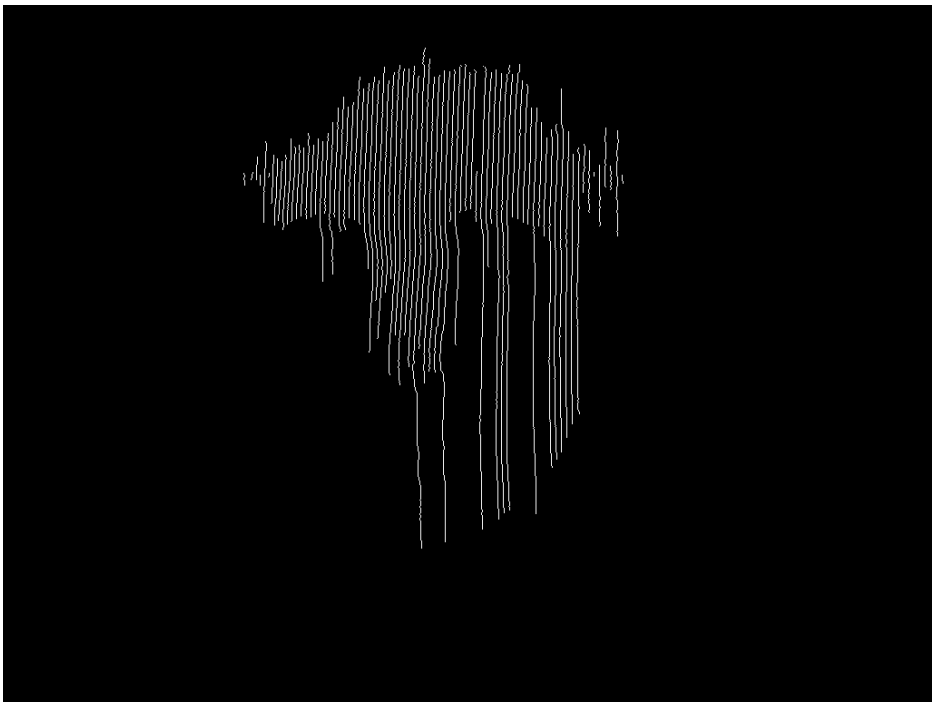

Figure 4.4b: Image taken from right after “initial line search” is applied

4.3.3 Searching Missing Parts

The success in finding the lines on the face is in close relation with the quality of the images and clarity of the bright points. But no matter how good quality the image is, because of several reasons (e.g. occlusions, rougness of the face, shadows), bright point may form a continuous line. In other words, when the white lined texture is projected on the face, some lines could not be followed from top to bottom continuously. This situation mostly appears for the lines which passes through nose. When the lines which fall upon nose are studied, line may not be followed between nose and lip during line search, because of the curl of nose. There are two reasons for this; one reason is that the line is broken on the tip of the nose because of the curl, and the second reason is that because of the shadow of the nose, bright points between nose and lip may not be identified clearly.

During the matching process, even if there are spaces between the upper and lower parts of a vertical line, these parts should be stored together and which line in the lower part of the face belongs to which line on the upper part of the face should be identified. Because, after finding the line coordinates, the aim is to match the corresponding lines in stereo images in order to calculate the depth information.

In initial line search, continuous lines are searched starting from the initial point, which is placed in forehead. When it is assumed that the quality of the images are not perfect and all bright points may not form continuous lines all over the face, the line structure obtained shall include most of the lines, but as in Figure 4.4, these lines may be cut especially in nose level. The aim of this part is to merge the corresponding line segment in the lower part of the face and the upper part of the face.

Initially, the longest continous line is determined. This line is considered as the reference line that will be used for matching the lines in the lower part and the upper part of the

face, since the longest line has points both in the lower and upper part and it is surely known that these points are belonging to the same line. Then line search will be made starting from the lower coordinates of the longest line towards the right and the left sides, and if the found lines have the properties that they can be lower parts of the lines at the upper part, the lines in the lower parts are merged with the upper ones. When the same procedure is applied to all lines, each bright point on the face is stored as a part of a line structure.

This procedure can be summarized as follows:

i) Find the longest line after initial line search. Set this line as the "current line".

ii) **If** there is a line in the right side of the current line, set this line as the "right line".

iii) **If** absolute (line length of current line - line length of right line) > threshold (this means that right line is not complete, some extra search shall be done on the right line),

Then staring from lowest point of the current line, search a point in the right side, in a margin of average line distance.

If a point can be found in the right side, start to construct the line to both up and down and store this line as the lower part of the right line.

Else, go up on the current line until a point in the right side is found or lowest point of the right line is reached.

If a point is found, construct line and store as the lower part of the right line.

Else (this means that, lower part of right line does not exist or cannot be found in the bitmap file), skip this search. Set the right line as the "current line" and turn to part (i).

Else (this means that right line is long and complete enough), skip this search. Set the right line as the "current line" and turn to part (ii).

iv) Repeat part (iii) until the lines in the right side are finished.

v) When the lines in the right side are finished, repeat all procedure starting from (ii), for the lines in the left side.

The images after “search missing parts” procedure applied are shown in Figure 4.5.

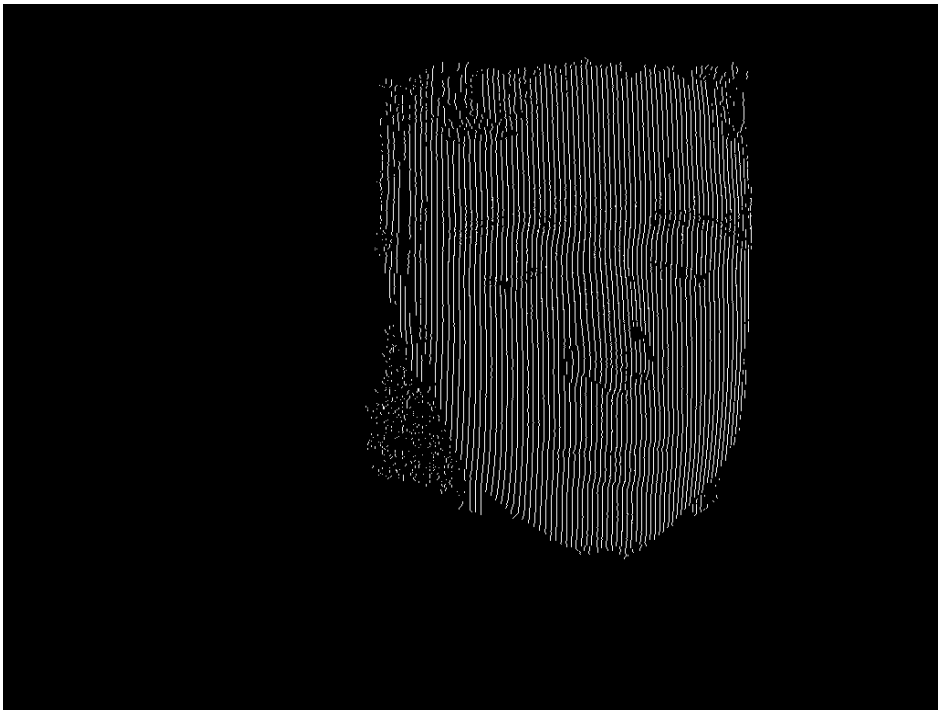

Figure 4.5a: Image taken from left after “search missing parts” is applied

Figure 4.5b: Image taken from right after “search missing parts” is applied

4.3.4 Finding Lines At The Edges

When “Searching Missing Parts” procedure is applied, we end up a problem at the edge of the face as shown in Figure 4.5. The reason is that in initial line search, search starts from forehead and continues towards right and left sides. But when hair at the edges of the face starts, no more lines can be found and initial line search finishes without finding further lines at both sides of the face. The aim of this section is to find the lines at the edges, which were not found in “Initial Line Search” and “Search Missing Part” sections.

In this section, initially, the right most line among the lines found previously is determined. From the middle point of this line, point search is applied to the right side in a margin of average line distance. If a point can be found, line construction is applied towards up and bottom, then the same procedure is applied starting from the middle

point of the new found line. If a point cannot be found, point search continues from each point on the right most line until a point is found. When all points are tried and nothing is found, it means that the lines in the right side are finished. Then the procedure is repeated to left side starting from the middle point of the left most line.

As a result of this two-sided search, the final images obtained are as in Figure 4.6.

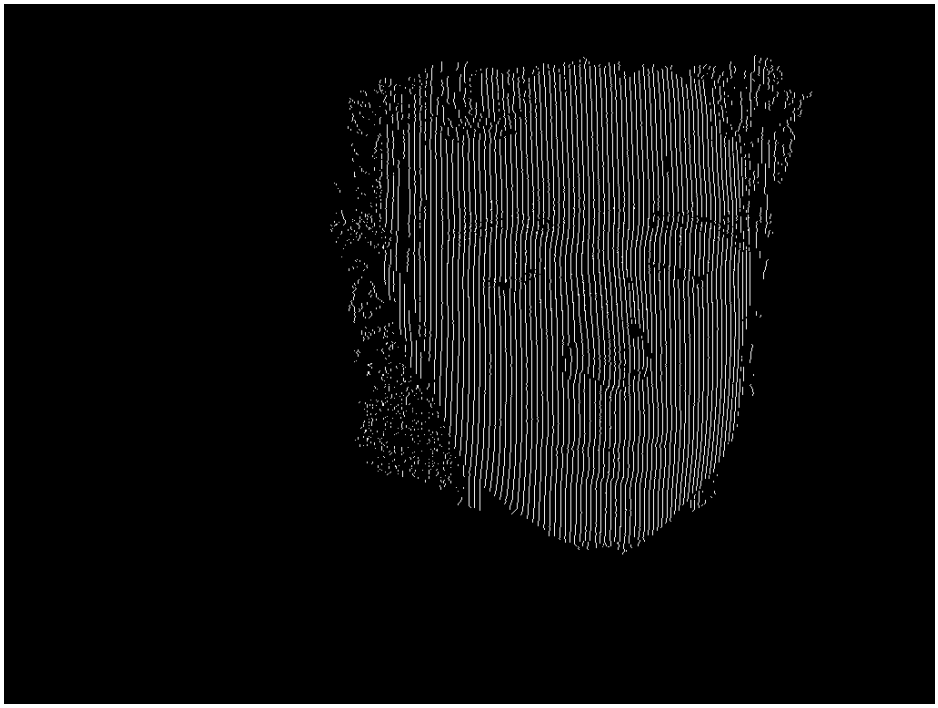

Figure 4.6a: Image taken from left “find lines at the edges” is applied

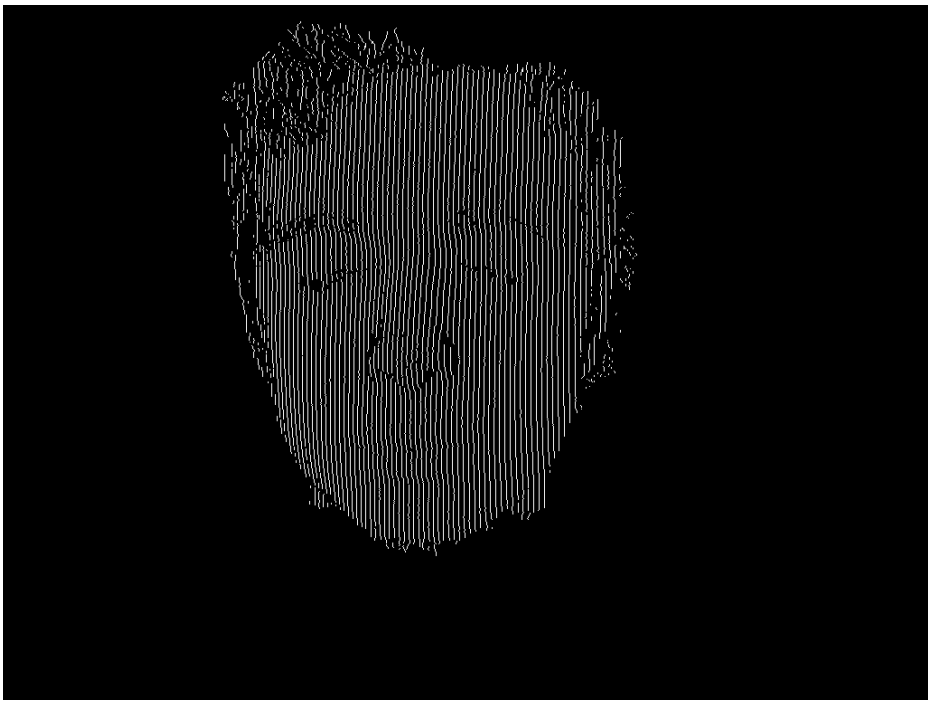

Figure 4.6b: Image taken from right “find lines at the edges” is applied

CHAPTER 5

OBTAINING 3D POINT CLOUD

5.1 Introduction

In the previous chapter how to find the vertical indexed lines corresponding to the brightest points on the structured light projected face images is explained.

In this chapter, first homologous points are determined and by using the coordinates of these points, the disparity is calculated which is in turn used to find out the 3D coordinate of the points in real world. After applying the procedure on four couple of stereo images, four coordinate sets will be obtained. Then 3D coordinate sets will be merged to obtain the final 3D point coordinate set.

The procedure obtaining 3D point cloud explained in this chapter has three phases:
(1) determining homologous points, (2) 3D point coordinate analysis for each of the four pair of images, (3) obtaining a final 3D point cloud from merged coordinates.

5.2 Determining Homologous Points

The reason to find the homologous points is to calculate the depth values of these points by using the deviation of their x coordinates. This is called disparity calculation as explained in the next section.

Determining homologous points i.e. stereo matching, is a very important problem. In this thesis, stereo matching problem is solved by using structured light and manually setting initial points in each stereo image pair. Initial point pairs are the only points that are known to be homologue and they are the starting points for disparity calculation.

After finding the lines on the surface, which was projected by the structured light, the following results can be achived:

Let the images in Figure 5.1 be a couple of stereo images captured. Let $line_i$ and $line_j$ be the vertical lines indexed with i and j, with i>0 and j<0, and let $line_0$ be the initial line. Since the cameras are parallel to the floor, the points in both images that are on $line_i$ and have the same y-coordinates are the homologous points. In the same manner the points in both image files that are on $line_j$ and have the same y-coordinates are the homologous points. By using this property, when all the indexed lines in a stereo image couple are analysed, a set of homologous points can be obtained.

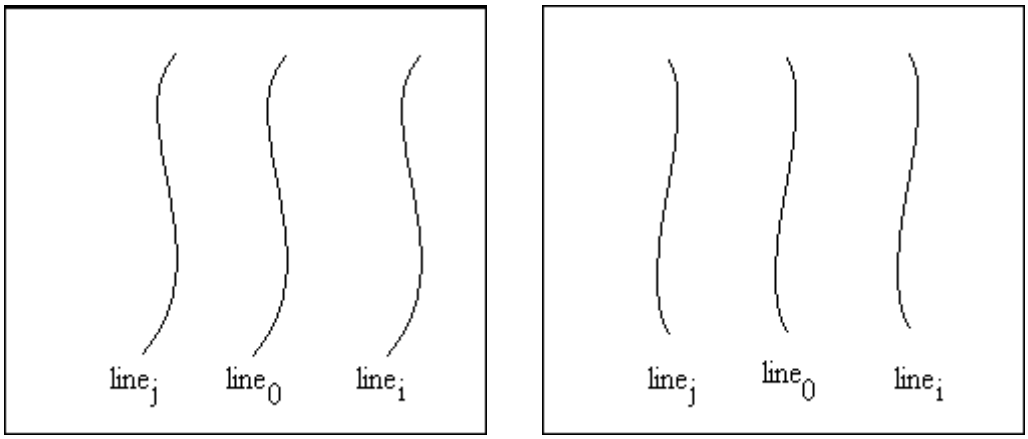

Figure 5.1: An example for stereo image couple

5.3 Disparity Calculation and Finding Depth

To obtain the coordinate of a point on the world coordinate system, three coordinate values are needed. x and y-coordinates of each point is known from images. The third coordinate, which is the depth value, will be found by using disparity calculation.

On the initial line, starting from top to bottom, disparity calculation given in Eq.2.2 is made for all homologous points and depth values are stored. If a point in one image does not have a homologous point on the other image, disparity calculation is not done and depth value for this coordinate is stored as invalid.

When disparity calculation is made for all the lines in the right and left side, three matrices storing x, y and z values are obtained. This process is repeated for all four bitmap pairs to obtain twelve (4*3) coordinate matrices.

5.4 Collecting Matrices

In this part, twelve coordinate matrices will be merged horizontally to obtain three coordinate matrices for x, y and z-coordinates. This part can be divided into two sections that are determining the final matrix size and merging the matrices.

5.4.1 Determining Final Matrix Size

The sizes of coordinate matrices obtained by analyzing each stereo image pair are different. Because of that, row and column numbers of the final matrix shall be determined before starting to merge.

If it assumed that matrix sizes of 3D point clouds for each image pair is row (i) and column (i), where i denote the image pair number, final column size will be:

Final_Column_Size = column (1) + column (2) + column (3) + column (4) (Eq. 5.1)

But how to determine final row size is a question to be solved. During the analysis of the stereo image pairs, the lines on the surfaces where structured light is projected focally can be identified easily. But during analysing some of the image pairs, some lines at the top and bottom of the object are accidentally processed. To prevent this problem, during determining final matrix row size, common coordinates of 3D point clouds of each image pairs are considered. As a result of this, a dense matrix including common coordinates of all four image pairs are obtained.

Then the final row size will be:

Final_Row_Size = min (row (1), row (2), row (3), row (4)) (Eq. 5.2)

5.4.2 Merging The Matrices

Twelve matrices to be merged have the point coordinates of the points in 3D point cloud. This section describes merging these matrices vertically and obtaining one set of x, y and z matrices. One set of coordinate matrices is as below:

x-coordinate matrix:

$$\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,Columnsize} \\ x_{2,1} & & & \cdots \\ \cdots & & & \cdots \\ x_{Rowsize,1} & x_{Rowsize,2} & \cdots & x_{Rowsize,Columnsize} \end{bmatrix}$$

y-coordinate matrix:

$$\begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,Columnsize} \\ y_{2,1} & & & \cdots \\ \cdots & & & \cdots \\ y_{Rowsize,1} & y_{Rowsize,2} & \cdots & y_{Rowsize,Columnsize} \end{bmatrix}$$

z-coordinate matrix:

$$\begin{bmatrix} z_{1,1} & z_{1,2} & \cdots & z_{1,Columnsize} \\ z_{2,1} & & & \cdots \\ \cdots & & & \cdots \\ z_{Rowsize,1} & z_{Rowsize,2} & \cdots & z_{Rowsize,Columnsize} \end{bmatrix}$$

In the matrices given above, Rowsize is the length of the longest vertical line and Columnsize is the number of vertical lines in 3D point cloud of one image couple. In the matrices, columns having the same index corresponds to coordinates of a vertical indexed line. Rows having the same index corresponds to coordinates of the points on the indexed lines on the same horizontal line.

Let's name x-coordinate matrix of the first point cloud as X (1). Then there become twelve matrices named, X (i), Y (i), Z (i), where i = 1, 2, 3, 4. Let's name the final coordinate matrices as Xfinal, Yfinal and Zfinal. Let's C (i) be the column indexes of the matrices X (i), Y (i) and Z (i) and C_f be the column index of final matrix.

Then the merging procedure of the matrices is as below:

1) Set C (1…4) and C_f as zero. Initially all matrix column indexes are set as zero.

2) Find out which matrix among X (i) in the column C (i), i = 1, 2, 3, 4, is smallest. Store the index i as ‘smallest’.
3) Copy the column to the final matrix.
    a. Copy Xfinal (C_f) = X (smallest) (C (smallest))
    b. Copy Yfinal (C_f) = Y (smallest) (C (smallest))
    c. Copy Zfinal (C_f) = Z (smallest) (C (smallest))
4) Increment C_f
5) Increment C (smallest)
6) If C_f is smaller than Final_Row_Size go back to step (2)

When this procedure is finished, three merged coordinate matrices (Xfinal, Yfinal and Zfinal) are obtained. This is the actual 3D scanner output including some invalid points (for the coordinates that disparity cannot be calculated) and some noisy points (spikes). An example of 3D scanner output in this study can be seen in Figure 5.2.

After merging process is applied, three matrices, Xfinal, Yfinal and Zfinal, have same column size and row size. Property of these matrices is that the points in the neighbouring indexes in the matrices are the neighbouring points in real world. This property will provide us easiness during converting matrices into “.obj” file format.

The next chapter will describe the post processing part of the 3D scanner, which converts the output of 3D scanner into a better format.

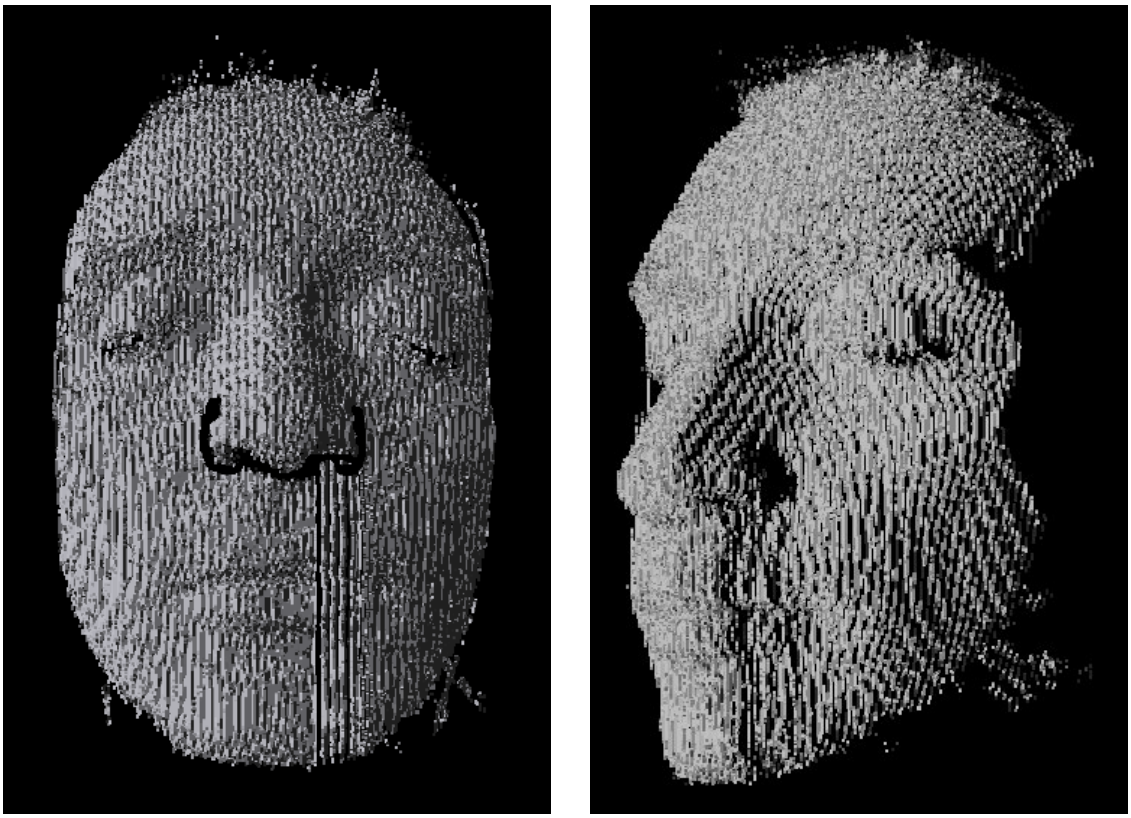

Figure 5.2: Output of 3D scanner

CHAPTER 6

POSTPROCESSING

6.1 Introduction

In the previous chapter, a 3D scanner output was obtained in 3D point cloud format. But this 3D point cloud may have some invalid and noisy depth values, so some post processing shall be applied on scanner output to get a better model. All phases of the post processing will be applied only on the Z matrix, which stores the depth values of 3D point cloud.

The procedure explained in this chapter has four phases:
In the first phase invalid coordinates in 3D point cloud will be filled; in the second phase spikes will be removed, in the third phase noisy depth values will be smoothed; in the fourth phase the final model will be converted to ".obj" file with covering the surface with the texture of the object.

6.2 Filling Holes

There are two reasons to have invalid depth values in a 3D point cloud:

1. While merging 3D point clouds, for the sake of simplicity, point clouds are formed in rectangular structure and for completing this rectangular structure,

despite there is no data on neither of the analyzed images for the coordinates outside the borders of the scanned object, these coordinates are stored and their depth values are set as invalid.

2. In some of the coordinates inside the borders of the object, data could be missing due to shadow or occlusion. So, the depth values for these coordinates are set as invalid also.

Since first type of invalid points do not belong to the object, their invalid depth values can be neglected and they will be eliminated during generation of ".obj" file and they will not exist in the final file. But latter type of invalid points should be filled since they belong to the object, and leaving them as invalid will cause holes on the body of the object on the final shape.

To fill the holes due to the second type of invalid points, firstly the points of these types should be identified. For this purpose, all points in 3D point cloud are searched for the points with invalid depth value and then their adjacent points are analyzed. If for each point, at least four of neighbouring depth coordinates out of a total of eight neighbouring coordinates are valid, it is understood that this point should be filled with a depth value computed as the average depth value of its valid adjacent points. After each point is searched in this manner, a 3D point cloud without any holes on its body will be obtained.

Figure 6.1a shows the 3D shape of the scanned face before filling invalid values on 3D point cloud and Figure 6.1b includes the output of the same scan after filling invalid values process is completed.

Figure 6.1a: before filling invalid values

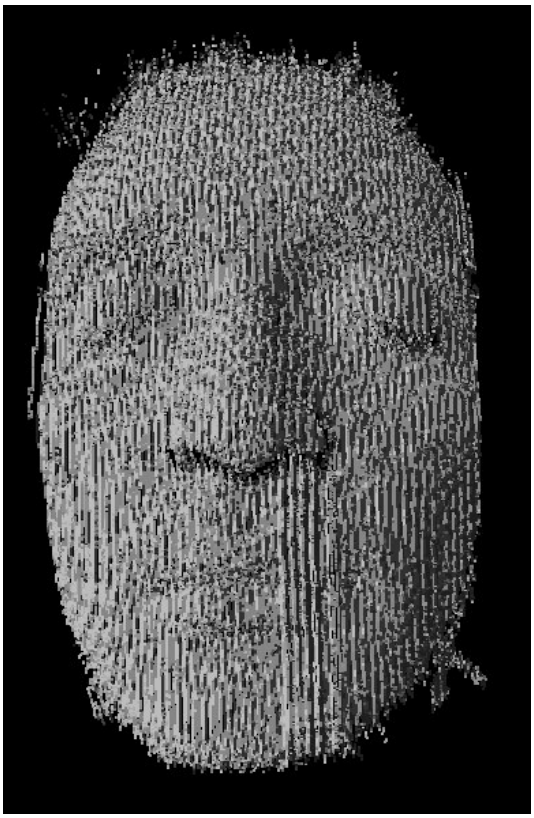

Figure 6.1b: after filling invalid values

6.3 Spike Removal

Spikes are abnormal depth values inside a smooth area in 3D point clouds. During analysing of stereo image pairs, because of some reasons some spikes may occur undesirably. The main reson for spike formation is mismatch of homologous points between stereo images. Around some protruding areas, like nose, there may appear some errors during tracking the lines vertically. As a result of this, disparity is calculated wrong and spikes occur. But most of the spike formations happen at the background which does not effect the original object model.

In this study spike coordinates are removed by using median filtering on Z (depth) matrix. By using a median filtering with window size [20 20], spike removal is

completed. The output of 3D point cloud before spike removal is as in Figure 6.2a and after spike removal is as in Figure 6.2b.

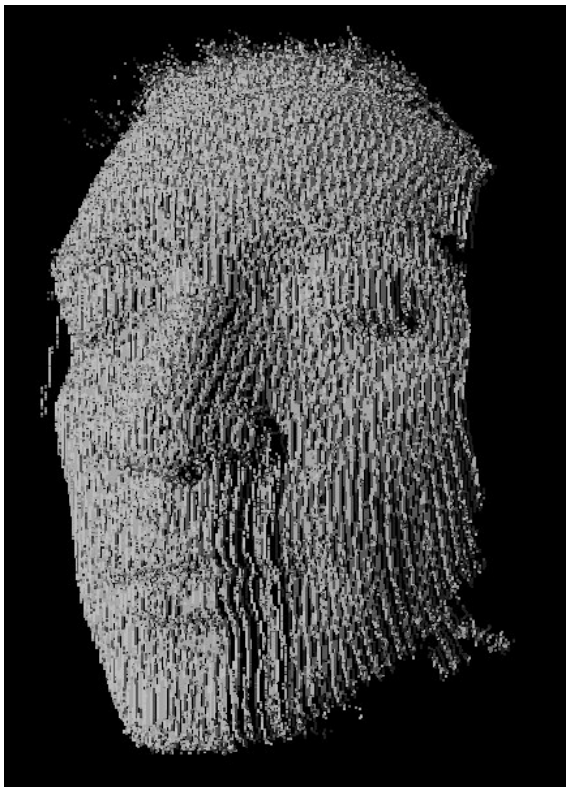

Figure 6.2a: before spike removal

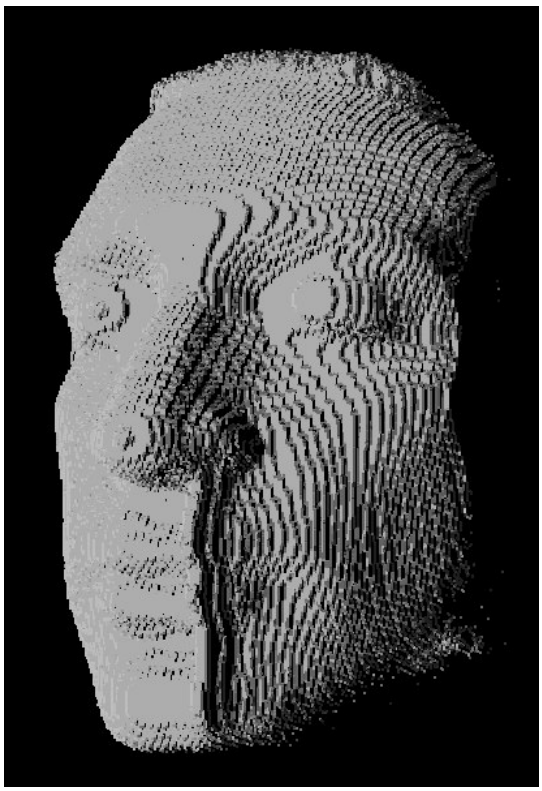

Figure 6.2b: after spike removal

6.4 Smoothing Surface

In this study, image analyses and disparity calculation is very dependent to outside effects, like light, shadow, and occlusion. Because of these effects, some errors may occur while finding the highlighted points on the object, which causes deviations during disparity calculation between the homologous points in stereo image couples. These deviations result in protruding surfaces.

In this study gaussian filtering is applied on Z (depth) matrix to smooth the protruding surfaces. The filtering matrix used is given below in Eq. 6.1 and Eq. 6.2.

$$g = \begin{bmatrix} 0.05 & 0.25 & 0.4 & 0.25 & 0.05 \end{bmatrix} \quad \text{(Eq. 6.1)}$$

$$g^T \times g = \begin{bmatrix} 0.0025 & 0.0125 & 0.02 & 0.0125 & 0.0025 \\ 0.0125 & 0.0625 & 0.1 & 0.0625 & 0.0125 \\ 0.02 & 0.1 & 0.16 & 0.1 & 0.02 \\ 0.0125 & 0.0625 & 0.1 & 0.0625 & 0.0125 \\ 0.0025 & 0.0125 & 0.02 & 0.0125 & 0.0025 \end{bmatrix} \quad \text{(Eq. 6.2)}$$

The output of 3D point cloud before smoothing the surface is as in Figure 6.3a. The output of 3D point cloud after applying gaussian filtering is as in Figure 6.3b.

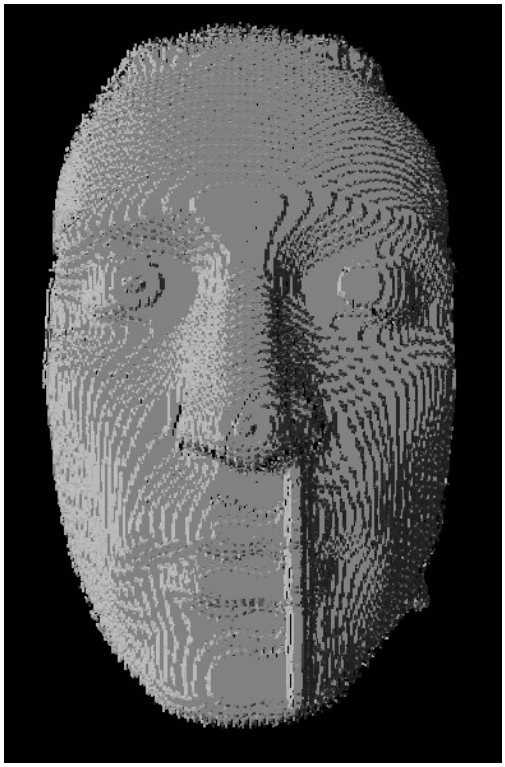

Figure 6.3a: before smoothing

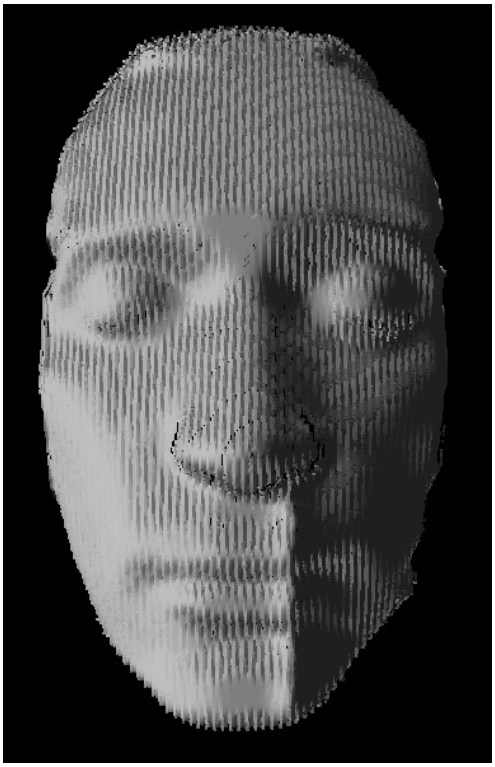

Figure 6.3b: after smoothing

6.5 Converting to ".obj" File

The last part of the study is forming a textured ".obj" file, which is composed of post-processed 3D point cloud and the image, which was captured without projecting structured light. The required inputs for the program that forms the ".obj" file are post-processed 3D point cloud with x, y and z-coordinates and the bitmap file of the object to be scanned.

Obj file format includes four different values. These are (1) 3D coordinates of the vertices, (2) normal values of the vertices, (3) texture coordinates that will be covered over the shape and (4) order of the connection of the vertices.

3D coordinates of the vertices are directly x, y and z-coordinates of 3D point cloud that is post-processed. The coordinates having invalid depth values are the points, which are assumed to remain out of the boundaries of the object to be scanned, so these points will not take place in the ".obj" file. For this reason during forming the vertices not all but only the point coordinates that have valid depth values are stored as vertices.

Normal values are calculated one by one for each vertex. Normal value of the plane of a vertex is found by using the coordinates of the vertex and its eight neighbouring vertices, and this value is stored in ".obj" file in the same format with the vertices written in the first part.

Texture coordinates are the 2D coordinates of the texture on the bitmap that will be covered on the constructed mesh. x and y-coordinates on the bitmap files have the same coordinates with the x and y-coordinates of the 3D point cloud. Because of this, in the same order with the vertices, x and y-coordinates of the vertices on the bitmap file are written to ".obj" file with normalized values to the range [0, 1]. Figure 6.4 shows the 3D models before and after covering texture on 3D point cloud.

Lastly written values are the order of the connection of the vertices. Assuming that the 3D point cloud already stores the point coordinates in a mesh structure, vertices to be connected will be the neighbouring vertices. If each mesh is marked with the vertex at its up-left corner, V(i,j) mesh will be composed of the vertices V(i,j), V(i+1,j), V(i+1, j+1) and V(i, j+1) in order.

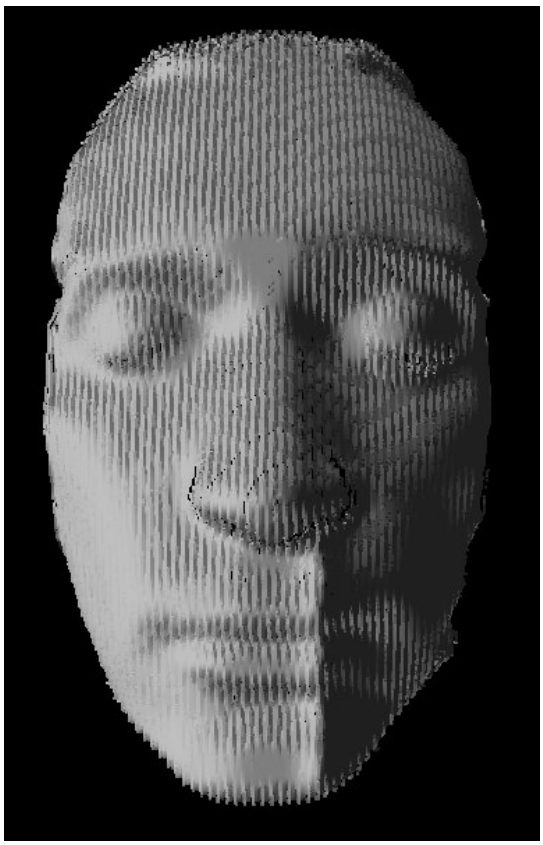

Figure 6.4a: before covering texture

Figure 6.4b: after covering texture

CHAPTER 7

EXPERIMENTAL RESULTS

7.1 Introduction

This chapter shows the results obtained from 3DFaceScanner and post-processing phases. All the examples include one image captured without texture projection, one image with texture projection and the final model from several views. First four examples are experimental results of human faces; the last example is the experimental result of a stationary object. The fourth human face example includes comparison with the Enterface 3D scanner.

7.2 Results

Results seen in this chapter are the chosen results from a plenty of experiments. In the images, while usually smooth parts of the face like forehead and cheek result in good results, protruding areas, especially nose, causes problems in some models. There are two reasons of this problem. First one is, analyzing the lines around nose is so difficult and causes some depth calculation mistakes in this area. Second one is, the area just under the nose cannot be seen clearly in the frontal images.

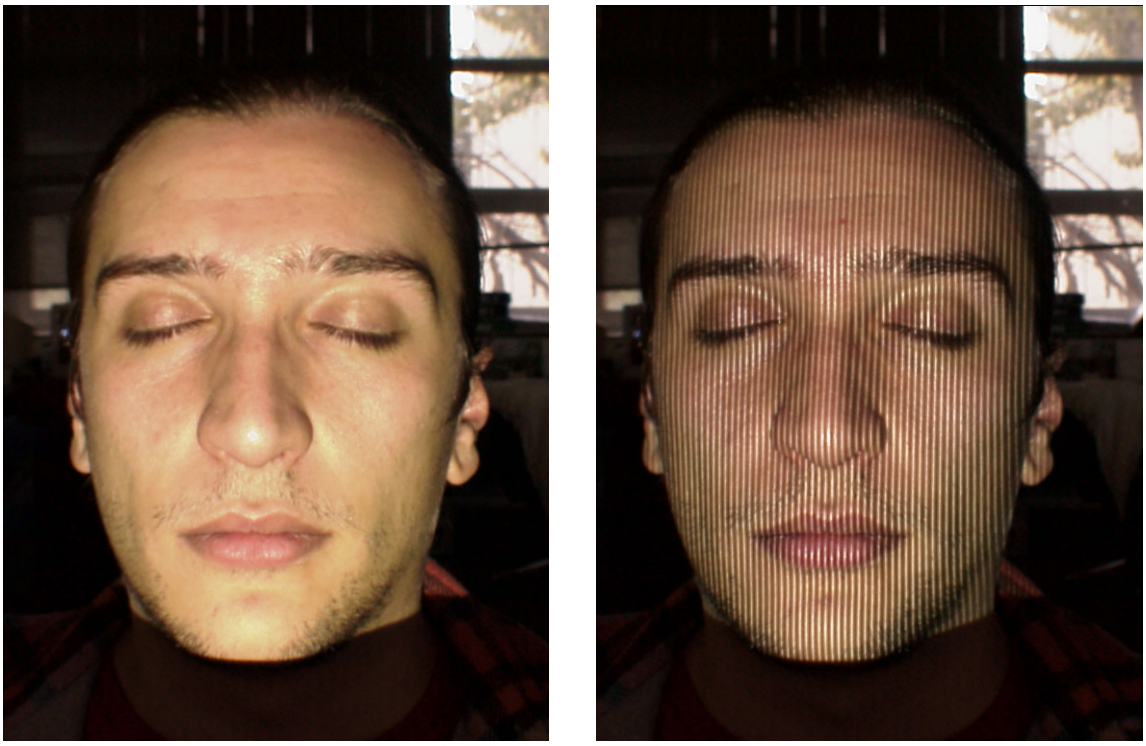

Figure 7.1: Images captured without/with structured light reflection

Images captured with and without structured light projection can be seen in Figure 7.1. In the first example, forehead of the person is open and there seems no problem to prevent the initial line search. Image is exactly from the front and the problems that can happen because of occlusion in right and left side of nose is minimized. Vertical lines on the face can be easily determined. Shodows in the nose area is minimized and with all these properties explained above, images are very suitable for this study. Images of model viewed from different views can be seen in Figure 7.2.

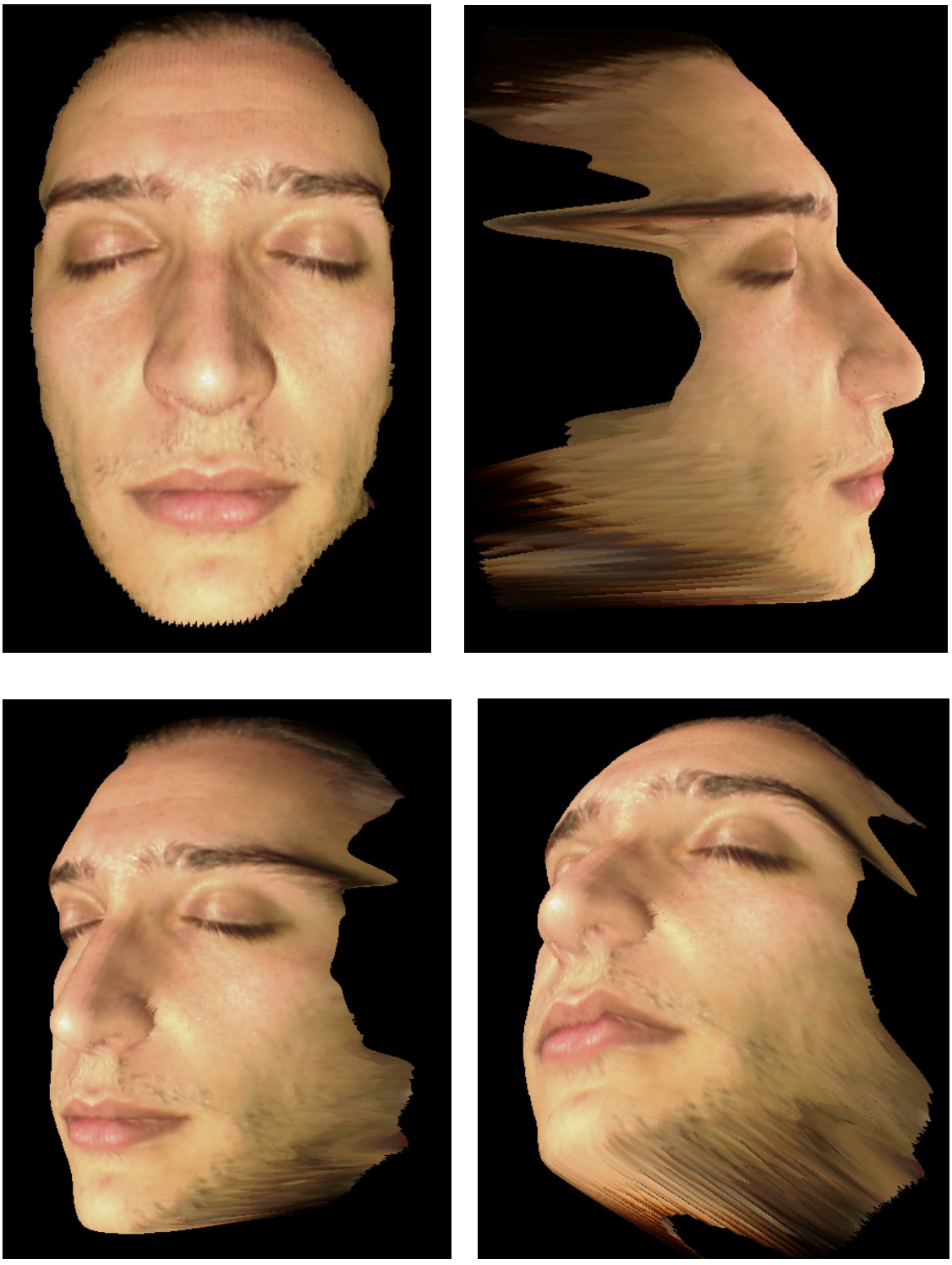

Figure 7.2: Images of 3D model from different views

In the second example, the images seen in Figure 7.3 are captured not exactly from the front but from a little bit right side. Because of that occlusions occur in the left side of the 3D model. The results of this experiment can be seen in Figure 7.4. From the image captured with structured light it can be seen that the vertical lines passing throught the lips and nose cannot be determined very well. Because of that there is some corruption on the skin texture around nose and lips. But except these two regions, the shape of the face has been constructed well. Especially if the image captured from the right side of the ".obj" file is considered, depth values of the nose and the forehead seems to be calculated very close to real shape.

Figure 7.3: Images captured without/with structured light reflection

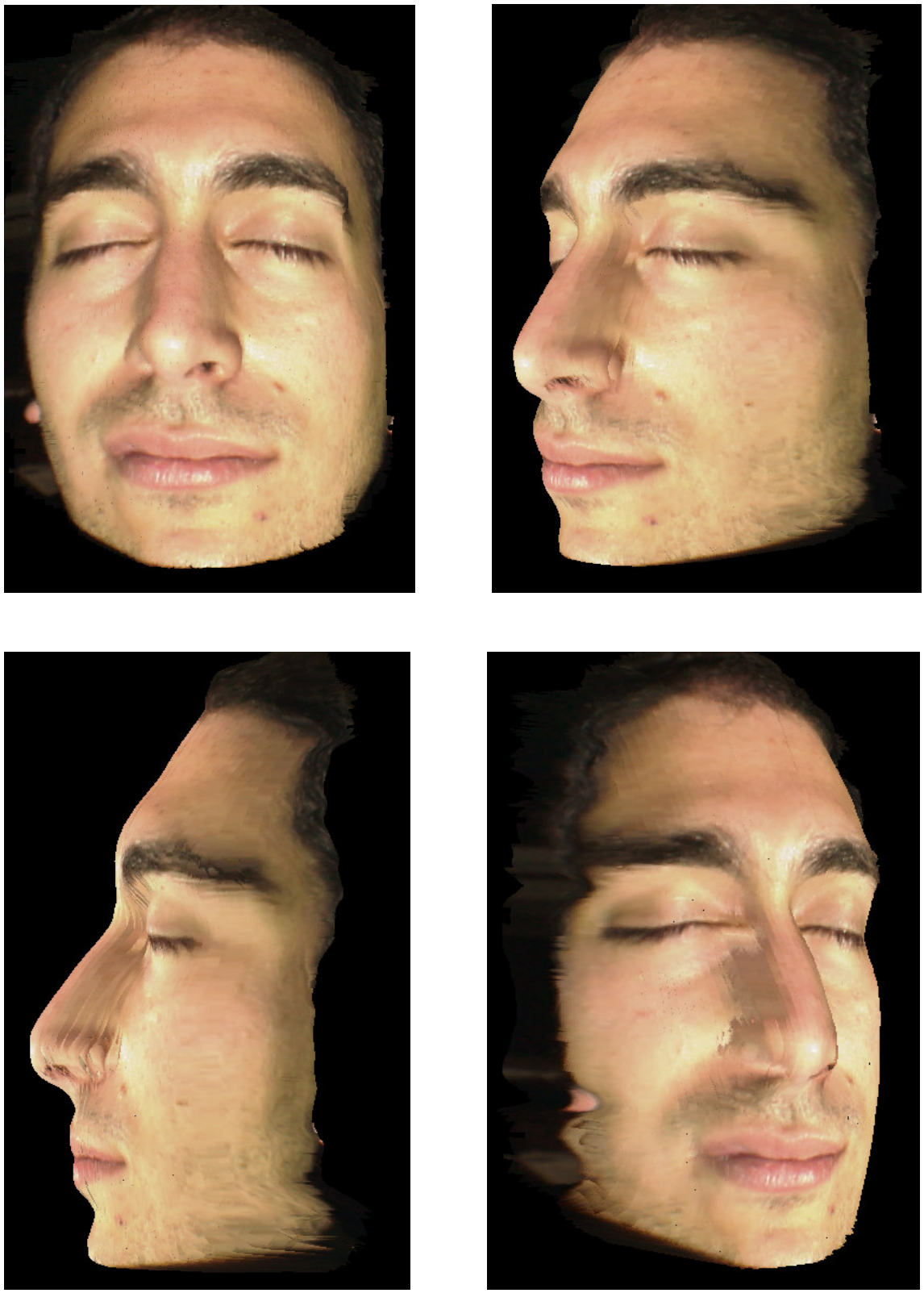

Figure 7.4: Images of 3D model from different views

Captured images for the third example is given in Figure 7.5 and images of 3D model from different views are given in Figure 7.6. By looking at the results it can be seen that person's head have moved slightly during capturing the images. Especially if the area around the nose in the image captured from right side of 3D model is analysed, the movement of the head can be sensed. As a result of this example, the importance of stability during capturing the images can be understood.

Figure 7.5: Images captured without/with structured light reflection

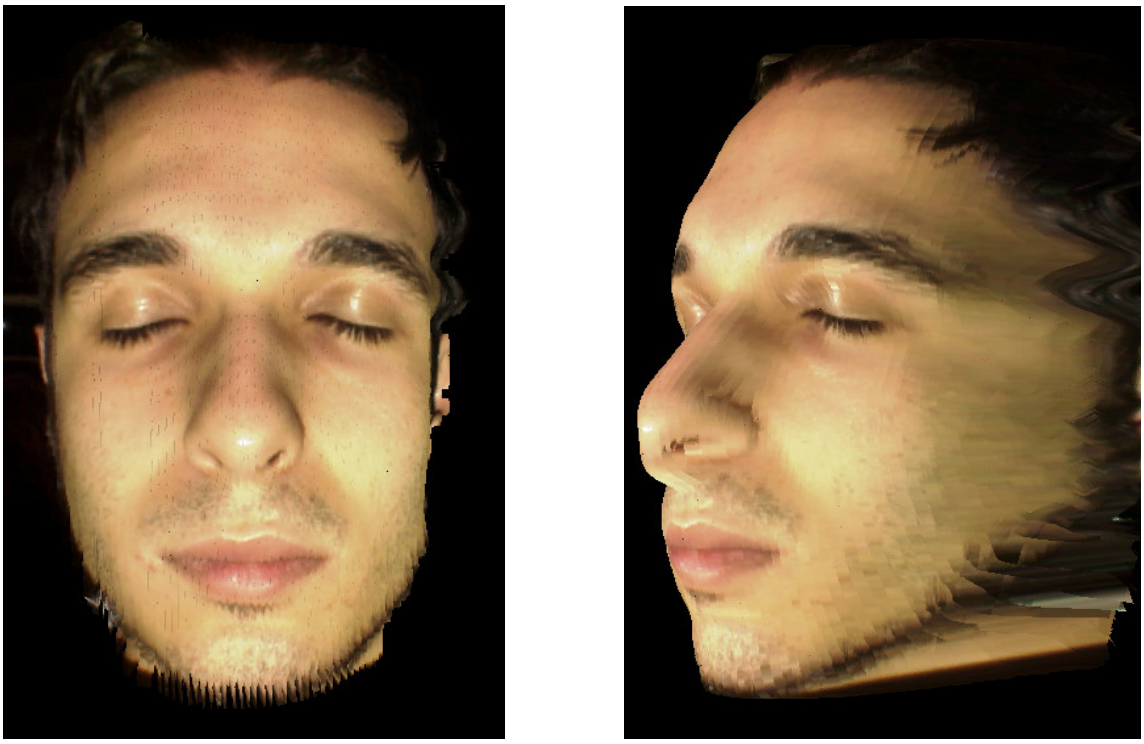

Figure 7.6: Images of 3D model from different views

Captured images for the fourth example are given in Figure 7.7. Since images are captured from below, region around the node is very successful in the 3D model output. The comparison of the thesis study and Enterface Scanner results can be seen in Figure 7.8, Figure 7.9, Figure 7.10 and Figure 7.11.

Figure 7.7: Images captured without/with structured light reflection

Figure 7.8a: 3DFaceScanner front

Figure 7.8b: Enterface front

Figure 7.9a: 3DFaceScanner profile

Figure 7.9b: Enterface profile

Figure 7.10a: 3DFaceScanner side

Figure 7.10b: Enterface side

Figure 7.11a: 3DFaceScanner below

Figure 7.11b: Enterface below

The last example is a colorful ball. This example is provided to show that: (1) this scanner is not speacial to face scanning; (2) this scanner gives better results in motionless objects. Images captured with and without structured light projection can be seen in Figure 7.12 and images of the ".obj" file captured from different views can be seen in Figure 7.13.

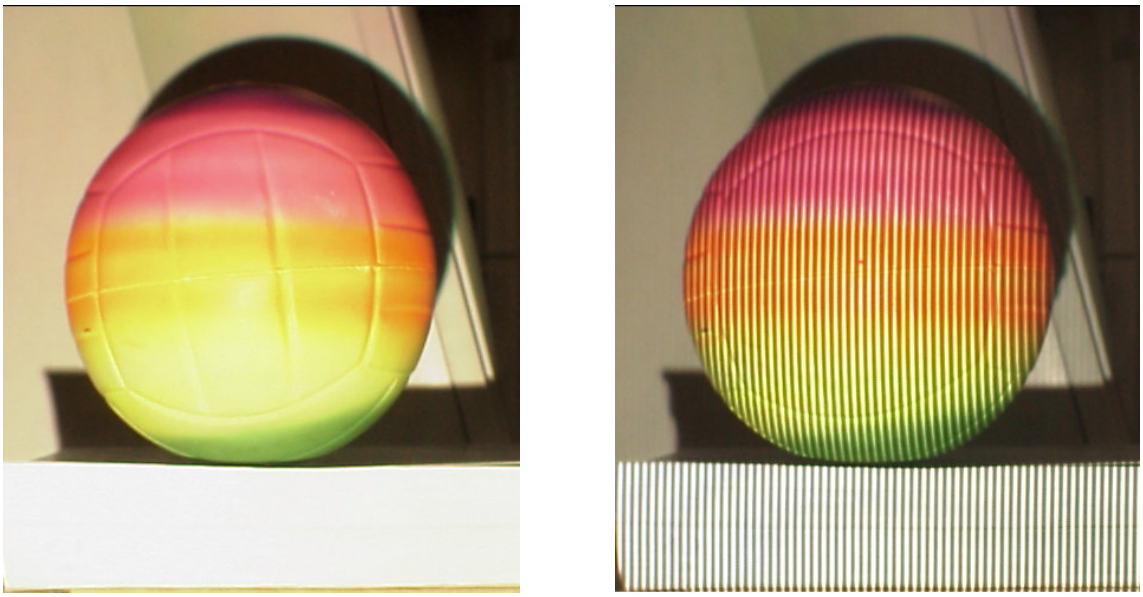

Figure 7.12: Images captured without/with structured light reflection

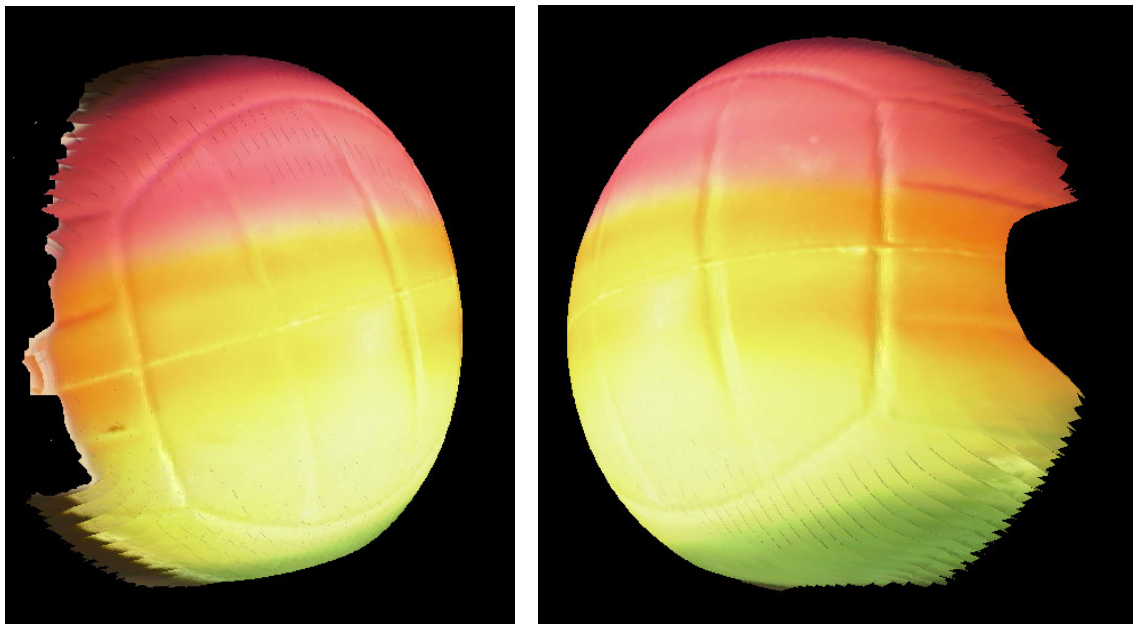

Figure 7.13: Images of obj output from different views

CHAPTER 8

CONCLUSION

8.1 Work Done

In this thesis, stereo cameras and a projector system have been constructed for 3D object modelling purpose where homologous points of stereo images have been detected with the help of structured light reflected on the object. Then by triangulation, the depths of these points are found. Finally 3D model of the object is constructed. Stereo cameras are placed just above the projector, parallel to each other and to the ground. The distance between them is fixed to be 8 cm. Distance between the object to be scanned and the cameras should be between 42 and 48 cm approximately based on our experiences. The image acquisition duration requires five seconds: four images taken in one-second intervals as structured light slides in one direction and last image pair is taken without reflecting the structured light on the object. During this operation, scanned object should be kept as motionless as possible. Although structured light is used, the system works properly at daylight.

Tasks carried out after image-capturing process can be classified in two groups. First one is the composition of 3D point cloud by processing the images; and second one is the completion of missing points and enhancing the 3D shape information by post-processing of this 3D point cloud. Composition of the 3D point cloud takes approximately one minute whereas the post-processing of point cloud and generating the

obj file takes between eight to eleven minutes when executed with MATLAB code in Intel (R) Pentium 4 CPU 2.4 GHz processor, without optimizing the source code.

Points worth to mention are that the 3D scan system in this thesis is not specific to the face and that the 3D model of the object is covered with the texture of the scanned object. Although application of the process generally to human face has been described throughout the examples and narration, satisfactory results have been obtained with other objects as they can be seen in Figure 7.13.

8.2 Difficulties Encountered and Future Works

Major difficulty faced in this study is the mixing of structured light lines to each other on protruding areas like nose. Since cameras face the scanned area with a certain angle, in some occasions, the vertical line passing on the nose may combine with the vertical line passing on left or right of the nose under the nose and this situation creates problem in all line alignments. In such a case, this problem disturbs all surface structure and results in high distortion on 3D model of the object. Currently, only white light is provided as structured light via projector in the thesis. The reason for that is the inability of the cameras to differentiate colors on the face clearly especially on the daylight. If a more powerful and a sharper light source as laser instead of projector is used, mixing of adjacent lines can be avoided even on jagged areas.

Another method for transmitting structured light is projecting infrared light. Current method is susceptible to daylight. So, it is important for image quality to make shootings in a dimly lighted environment as much as possible. Although, in a system set up with infrared transmitter and infrared cameras, there will be no day light interference and image quality of a dark room can be achieved even on outdoors. It must be bared in mind that for this method along with infrared cameras a regular camera should be used for capturing the texture to cover surface of the object.

Another difficulty faced on the thesis, is the lack of feedback between light source and cameras, which in some occasions results in loss of synchronization between them. In this case, elements that make up our scanning method disappear, thus there will be no point to carry on the analysis. It requires restarting whole scanning procedure. One of the reasons for restarting is the inability of the cameras to capture the image of the object simultaneously. Image capturing moments of the cameras may differ approximately 0.3 seconds. This condition is the movement of the structured light to the other slide in 0.3 seconds time interval, which is the time, passes between shooting of two different cameras. Increasing the time interval between slides to reduce the possibility of this condition will increase the shooting duration, and longer duration would cause loss of image quality since the object to be scanned may move. So, instead of reducing time interval between slides, time difference between the capturing from two cameras should be reduced or a messaging system should be implemented between the computer, which controls the projector, and the computer, which control the cameras.

Initial point is very crucial since it is the point that 3D point cloud is originated from and that provides the connection between two images. This point is reflected as a red dot on the object to be scanned. But, since the cameras may not detect the colored light emitted from the projector and reflected to the face accurately, finding this point automatically is a difficult and risky task. Because of the importance of this point and knowing its location exactly, this point is set in each of the images manually, which causes loss of time and divergence from automation. Instead of this manual setting, if initial point can be detected automatically, it will not only provide ease of use but also automation of the method will be increased.

Currently system is made up of projector and stereo cameras, which are connected to two different computers; therefore it is not mobile and requires a setup. One of the most important future works which will add significant value to this thesis is to assemble this system in a box as an embedded system and to operate it mobile. For constructing such a system and making its dimensions smaller, instead of a projector, which is a multi-

purpose big device, any light source, which can reflect the desired structured light consecutively, is required.

Another inadequate aspect of the current system is the limited distance between the object to be scanned and cameras and projector. This limitation aroused from the fact that there is a certain focus setting for the projector and the object should be on the range of this setting where the light reflects properly. If a system, which can change the focus setting of the projector according to the distance of the object to be scanned, thus can reflect structured light to the object on any distant properly, would be developed, this system can be used as a robot-eye and can identify any object on any distance.

In this study, one-pixel deviations during analyzing stereo image couples result in errors in depth calculations. These errors slightly distort the final 3D model. But if subpixel resolution is used during the analysis of images, the amount of error can be reduced or removed.

During covering the surface of the object with texture, one of the images captured without structured light projection (in this thesis image captured from right camera is used, but left one can also be used) is used. Because of this, only textures that can be captured from that camera can be covered. Instead, if the images captured from both cameras are fused and texture is covered with the fused image, more realistic results can be achieved during texture coverage.

Also during covering the surface, the image captured without structured light projection is used as is. But this image includes the shadows that are result of the ambient light. This results in bright and dark regions on the surface, which are actually shall not exist on the surface. To prevent this, some processes shall be applied on the image to remove shadows (e.g. albedo [5]). By this process, surface of the 3D model can be covered with its real texture.

In this thesis, stereo camera number is restricted with two and cameras are adjusted to capture the images from the frontal view. Because of this, in the final 3D model, only the depth values of the points of the object that can be seen from the front could be achieved. But instead of this system, if a multiple camera system that can view 360 degree of the object can be developed, and the images captured from all these cameras can be merged, complete 3D point cloud of the object can be achieved.

REFERENCES

[1] Carles Matabosch Geron`es, “Hand-Held Scanner For Large Surface Registration”, PhD Thesis, pp. 1-3, Girona April 2007.

[2] Aloimonos, Y., I. Weiss, A. Bandopadhay, “Active vision”, International Journal of Computer Vision 1, pp. 333–356, 1987.

[3] Imir Thomo, Sotiris Malasiotis, Michael G. Strintzis, “Optimized Block Based Disparity Estimation in Stereo Systems Using a Maximum-Flow Approach”, Image Processing 2001 on vol. 2, pp. 1-7, 2001.

[4] Shin ’ichi Satoht, Yuichi Ideharat, Hiroshi Mot, Takashi Hamada, “Subject Region Segmentation in Disparity Maps For Image Retrieval”, Image Processing 2001 on vol. 2, pp. 725-727, 2001.

[5] Maria Petrou, Svetlana Barsky, “Shadows and Highlights Detection in 4-Source Colour Photometric Stereo”, School of Electronics, Computers and Mathematics, Image Processing 2001 on vol. 2, University of Surrey, Guildford, GU2 7XH, UK, pp. 1-6, 2001.

[6] Rein-Lien Hsu, Jain, A.K, “Face Modeling For Recognition”, Image Processing 2001 on vol. 2, pp. 1-4, 2001.

[7] Beumier, C; Acheroy, M, “Automatic 3D face authentication”, Vision Computing on vol. 18, no. 4, pp. 315-321, 2000.

[8] Mehmet Dikmen, “3D Face Reconstruction Using Stereo Vision”, pp. 1-15. September 2006.

[9] J. Salvi, J. Pages, J. Batlle, "Pattern codification strategies in structured light systems," Pattern Recognition, vol. 37, Issue 4, pp. 827-849, April 2004.

[10] Jordi Pages, Joaquirn Salvi, Rafael Garcia, Carles Matabosch, “Overview of coded light projection techniques for automatic 3D profiling”, ICRA '03. IEEE International Conference on vol. 1, pp. 3-5, 2003.

[11] Chadi Albitar, Pierre Graebling, Christophe Doignon, “Robust Structured Light Coding for 3D Reconstruction”, Image Processing 2001 on vol. 2, pp. 1-7, 2001.

[12] T. Weyrich, M. Pauly, R. Keiser, S. Heinzle, S. Scandella, M. Gross, “Post-processing of Scanned 3D Surface Data”, Image Processing 2001 on vol. 1, pp. 1-6, 2001.

[13] Coloma Ballester, M. Bertalmio, V. Caselles, Guillermo Sapiro, Joan Verdera, “Filling-In by Joint Interpolation of Vector Fields and Gray Levels”, IEEE Transactions On Image Processing, vol. 10, no. 8, pp. 1200-1204, August 2001.

[14] Won-Sook Lee, Prem Kalra, Nadia Magnenat Thalmann, “Model Based Face Reconstruction For Animation”, MIRALab, CUI, University of Geneva, Geneva, Switzerland, pp. 1-4, 1999.

[15] O.D.Faugeras, Q.T.Luong, “The fundamental matrix: theory, algorithms, and stability analysis”, pp. 143–75, 1996.

[16] Stan Birchfield, Carlo Tomasi, “Depth discontinuities by Pixel-to-Pixel stereo”, pp. 1073–1080, New Delhi, India, 1998.

[17] Mingli Song, Zhao Dong, Christian Theobalt, Huiqiong Wang, Zicheng Liu, Hans-Peter Seidel, “A Generic Framework for Efficient 2D and 3D Facial Expression Analogy”, Image Processing 2004, pp. 3-9, 2004.

[18] R.J. Woodham, “Photometic method for determining surface orientation from multiple images”, vol. 19, pp. 139–144, 1980.

[19] C.Y. Chen, R. Klette, C.F. Chen, “Improved fusion of photometric stereo and shape from contours”, pp. 103–108, New Zealand, 2001.

[20] Asla M. Sá, Paulo Cezar Carvalho, Luiz Velho, “Recovering Registered Geometry and High Dynamic Range Texture with Coded Structured Light”, Image Processing 2001, pp. 1-2, 2001.

[21] Li Zhang, Brian Curless, Steven M. Seitz, “Rapid Shape Acquisition Using Color Structured Light and Multi-pass Dynamic Programming”, 3D Data Processing Visualization and Transmission, pp. 1-3, 2002.

[22] Marc Proesmans, Luc Van Gool. Reading, “a method for extracting dynamic 3D with texture”, Virtual Reality Software and Technology Conference, pp. 95–102, Lausanne, Switzerland, 1997.

[23] Jordi Page, Joaquim Salvi, Carles Matabosch, “Implementation Of a Robust Coded Structured Light Technique For Dynamic 3D Measurements”, Image Processing 2003 on vol. 2, pp. 1-3, 2003.

[24] Mark Chan, Chia-Yen Chen, Gareth Barton, Patrice Delmas, Georgy Gimel’farb, Philippe Leclercq, Thomas Fischer, “A Strategy for 3D Face Analysis and Synthesis”, Department of Computer Science, University of Auckland, pp. 384-386, 2003.

[25] Mark Young, Erik Beeson, “Viewpoint-Coded Structured Light”, Computer Vision and Pattern Recognition, 2007, CVPR '07, IEEE Conference, pp. 1-2, 2007.

[26] http://egweb.mines.edu/faculty/tvincent/Welding/fundamentals_of_stereo_computer.htm

APPENDIX A

DEFINITION OF SOURCE CODE

A.1 Main Classes

BitmapPoint.cpp
BitmapLine.cpp
Bitmap.cpp
CameraView.cpp

A.2 User Interface Methods

3DfaceScanner.cpp

A.3 Common Functionalities

linearization.cpp
bitmap_functions.cpp

A.4 Definitions of Main Classes

A.4.1 BitmapPoint

A.3.1.1 Definition:

BitmapPoint is a single point in a bitmap file.

A.3.1.2 Attributes:

int x: x-coordinate of the bitmap point
int y: y-coordinate of the bitmap point

A.3.1.2 Methods:

BitmapPoint(int x_point, int y_point)

definition : constructor of BitmapPoint class. Sets x and y coordinates of the point to the input parameters.

parameters:
x **:** x-coordinate of the point
y **:** y-coordinate of the point

returns: None

BitmapPoint()

definition **:** default constructor of BitmapPoint class. Sets x and y coordinates values to 0.

parameters **:** None

returns **:** None

bool findNearestPointToRight(unsigned char ** edarray, int search_offset, int &distance, int &x_position, int height, int width)

definition: finds the nearest white point in the right of the current point according to double dimension input point array and sets the point to the coordinates of the point found. If the point cannot be found, coordinates of the point do not change.

parameters:
edarray **:** the array of points which are black or white
search_offset **:** the offset to start the search. i.e. start to search from x + search_offset coordinate
distance **:** the difference between the initial point and the final point
x_position **:** the x value of the final point
width and height **:** the width and height properties of the bitmap file

returns:
TRUE - if point is found
FALSE - elsewhere

bool findNearestPointToLeft(unsigned char ** edarray, int search_offset, int &distance, int &x_position, int height, int width)

definition **:** finds the nearest white point in the left of the current point according to double dimension input point array and sets the point to the coordinates of the point found. If the point cannot be found, coordinates of the point do not change.

parameters:
edarray **:** the array of points which are black or white
search_offset : the offset to start the search. i.e. start to search from x - search_offset coordinate
distance **:** the difference between the initial point and the final point
x_position **:** the x value of the final point
width and height: the width and height properties of the bitmap file

returns:
TRUE - if point is found
FALSE - elsewhere

bool findNearestPointToRightBetween(unsigned char ** edarray, int search_offset_min, int search_offset_max, int &distance, int &x_position, int height, int width)

definition **:** finds the nearest white point in the right of the current point according to double dimension input point array and sets the point to the coordinates of the point found. If the point cannot be found, coordinates of the point do not change. Search is done between specific input margins.

parameters:
edarray **:** the array of points which are black or white
search_offset_min **:** the offset to start the search
search_offset_max **:** the offset to finish the search
distance **:** the difference between the initial point and the final point
x_position **:** the x value of the final point
width and height **:** the width and height properties of the bitmap file

returns:
TRUE - if point is found
FALSE - elsewhere

bool findNearestPointToLeftBetween(unsigned char ** edarray, int search_offset_min, int search_offset_max, int &distance, int &x_position, int height, int width)

definition **:** finds the nearest white point in the left of the current point according to double dimension input point array and sets the point to the coordinates of the point found. If the point cannot be found, coordinates of the point do not change. Search is done between specific input margins.

parameters:
edarray **:** the array of points which are black or white

search_offset_min **:** the offset to start the search
search_offset_max **:** the of set to finish the search
distance **:** the difference between the initial point and the final point
x_position **:** the x value of the final point
width and height **:** the width and height properties of the bitmap file

returns:
TRUE - if point is found
FALSE – elsewhere

bool isSame(BitmapPoint Point2)

definition **:** compares the coordinates of the input point with its own coordinates.

parameters:
Point2 **:** BitmapPoint to compare

returns:
TRUE - if Point2 has same coordinates with this point
FALSE - elsewhere

A.3.2 BitmapLine

A.3.2.1 Definition:

BitmapLine is the line structure class, which consists of BitmapPoints. This class only shows vertical lines in a bitmap file. For one each bitmap file, lines are stored in linked list form.

A.3.2.2 Attributes:

BitmapPoint initialPoint: This is the first point, which is found as a result of line search.
int *x_position **:** stores the x coordinate value for each corresponding y coordinate. Initially, all indexes are set to –1.
bool *bFilled **:** shows if the x coordinate value for each corresponding y coordinate is set or not.
int y_position_up **:** stores the smallest y coordinate of the BitmapLine
int y_position_down **:** stores the largest y coordinate of the BitmapLine
int line_length **:** shows the length of the line. Calculated as y_position_down - y_position_up
BitmapLine *right **:** stores the address of the line at the right.
BitmapLine *left **:** stores the address of the line at the right.

static int map3Dindex: stores the point index number during calculating and storing depth values.
static int map3DlineNumber: stores the number of the total lines.

A.3.2.3 Methods

BitmapLine(int height)

definition **:** Constructor of BitmapLine class.

parameters:
height **:** height of the bitmap

returns **:** None

bool constructLineUp(unsigned char ** edarray, BitmapPoint initialPoint, int &y_position, int height, int width)

definition **:** Fills the x_position array from the initial point to top until the search of the line is finished.

parameters:
edarray **:** the array of points which are black or white
initialPoint **:** the point where line search starts
y_position **:** the y coordinate where search finishes

returns:
FALSE - if error occurs
TRUE - elsewhere

bool constructLineDown(unsigned char ** edarray, BitmapPoint initialPoint, int &y_position, int height, int width)

definition **:** Fills the x_position array from the initial point to bottom until the search of the line is finished.

parameters:
edarray **:** the array of points which are black or white
initialPoint **:** the point where line search starts
y_position **:** the y coordinate where search finishes

returns:
FALSE - if error occurs
TRUE – elsewhere

bool constructLine(unsigned char ** edarray, BitmapPoint initialPoint, int &y_positionUp, int &y_positionDown, int height, int width)

definition **:** Fills the x_position array from the initial point to top and bottom until the search of the line is finished. This method calls constructLineUp and constructLineDown methods successively.

parameters:
edarray **:** the array of points which are black or white
initialPoint **:** the point where line search starts
y_positionUp **:** the y coordinate where search finishes at the top
y_positionDown **:** the y coordinate where search finishes at the bottom

returns:
FALSE - if error occurs
TRUE – elsewhere

static void construct3DMapForOneLine(BitmapLine *lineRight, BitmapLine *lineLeft, int *x, int *y, double * xval, double * yval, double *zval, int uppest, int downest)

definition **:** Fills x, y, xval, yval and zval arrays according to two lines matching in different bitmap files. Finds the depth information for points on the lines according to disparity.

parameters:
lineRight **:** BitmapLine pointer showing the line in the right bitmap
lineLeft **:** BitmapLine pointer showing the line in the left bitmap
x **:** coordinate array showing the x coordinates of the points in the right bitmap file
y **:** coordinate array showing the y coordinates of the points in the right bitmap file
xval **:** coordinate array showing the x coordinates of the points arranged according to the disparity between bitmaps
yval **:** coordinate array showing the y coordinates of the points arranged according to the disparity between bitmaps
zval **:** disparity array showing the disparities of the points
uppest **:** coordinate of the uppest point which is common in both right and left bitmap
downest **:** coordinate of the downest point which is common in both right and left bitmap

returns **:** None

A.3.3 Bitmap

A.3.3.1 Definition:

Bitmap is the class, which stores all the properties of a bitmap file.

A.3.3.2 Attributes:

char *bitmapName : string which stores the bitmap name.
char *dHeader : array which stores the header data of the bitmap.
char h18, h19, h20, h21, h22, h23, h24, h25: height values of bitmap file
ifstream iStream : file stream that points to input bitmap file
int width, height : width and height values of the bitmap
unsigned char **garray : array to store gray scale values of the pixels
unsigned char **edarray : array to store the peak point detection coordinates
char **dr : array to store R values of the bitmap.
char **db : array to store G values of the bitmap.
char **dg : array to store B values of the bitmap.
BitmapPoint initialPoint : point to start the searching algorithm. Set manually just before the search in both bitmaps
BitmapLine *initialLine : first found line in the right of initial point.
BitmapLine *currentLine : the line which is worked on
BitmapLine *leftestLine : the line found in the most left
BitmapLine *rightestLine : the line found in the most right.
BitmapLine *uppestLine : the line which has the smallest y value
BitmapLine *downestLine : the line which is the biggest y value
int nofLinesToRight : number of lines in the right of the initial point
int nofLinesToLeft : number of lines in the left of the initial point
int nofLines : total number of lines
int min_y : the value of y coordinate of the point which the uppest line point found on the face
int max_y : the value of y coordinate of the point which the downest line point found on the face
int averageDistanceRight : average distance between successive lines in the right side of the initial point
int averageDistanceLeft : average distance between successive lines in the left side of the initial point

A.3.3.3 Methods:

Bitmap(char *inputBitmapName)

definition : Constructor for Bitmap class. Initializes bitmap attributes and fills the rgb arrays according to input bitmap name by invoking readBitmap method.

parameters:
inputBitmapName **:** the name of the bitmap file

returns **:** none

Bitmap()

definition **:** Default constructor for Bitmap class. Initializes all bitmap attributes, sets width and height to default 1024 and 768 and sets default header.

parameters **:** none

returns **:** none

void readBitmap()

definition **:** reads the bitmap file and fills the rgb arrays.

parameters **:** none

returns **:** none

void grayScale()

definition **:** converts the bitmap into grayscale format by filling grayscale array.

parameters **:** none

returns **:** none

void detectPeaksHorizontal()

definition **:** performs "finding peak points" algorithm on grayscale array horizontally. Its purpose is to find the line points on the face.

parameters **:** none

returns **:** none

void makeGrayBitmap(char *grayBitmapName)

definition **:** Creates a bitmap file with the grayscale information.

parameters:
grayBitmapName **:** name of the bitmap file to be created

returns **:** none

void makeEdgeBitmap(char *edgeBitmapName)

definition **:** creates a bitmap file with the edge detection information.

parameters:
edgeBitmapName **:** name of the bitmap file to be created

returns **:** none

void makeLinesBitmap(char *lineBitmapName)

definition **:** creates a bitmap file with the lines detected.

parameters:
lineBitmapName **:** name of the bitmap file to be created

returns **:** none

void setInitialPoint(int x_pos, int y_pos)

definition **:** sets the initial point which is the starting point to search for the lines.

parameters:
x **:** x-coordinate of the initial point
y **:** y-coordinate of the initial point

returns **:** none

void searchInitialLinesToRight()

definition **:** performs initial line search search for the lines to the right starting from the initial point. Vertical search continues until the line is cut. Horizontal search continues during line is found within a distance near the average line distance.

parameters **:** none

returns **:** none

void searchInitialLinesToLeft()

definition **:** performs initial line search for the lines to the left starting from the initial point. Vertical search continues until the line is cut. Horizontal search continues during line is found within a distance near the average line distance.

parameters **:** none

returns **:** none

void searchMissingPartsToRight()

definition **:** performs searching the missing parts of the lines in the right side of initial point. This method does not find or add new lines, only finds the upper and lower parts of the lines found in the initial search.

parameters **:** none

returns **:** none

void searchMissingPartsToLeft()

definition **:** performs searching the missing parts of the lines in the left side of initial point. This method does not find or add new lines, only finds the upper and lower parts of the lines found in the initial search.

parameters **:** none

returns **:** none

void searchMoreLinesToRight()

definition **:** performs searching the missing lines in the right side of initial point. This method finds new lines at the right and left sides of the face, which are not found in the initial search.

parameters **:** none

returns **:** none

void searchMoreLinesToLeft()

definition **:** performs searching the missing lines in the left side of initial point. This method finds new lines at the sides of the face, which are not found in the initial search.

parameters **:** none

returns **:** none

void searchLines()

definition **:** performs vertical line search in the bitmap file. This method calls all the line search methods successively.

parameters **:** none

returns **:** none

void deallocateMemory()

definition **:** deallocates the memory that is allocated in the initialization phase for the Bitmap class.

parameters **:** none

returns **:** none

void makeColorBitmap(char *fileName)

definition **:** creates a bitmap file with dr, dg and db color arrays.

parameters:
fileName **:** name of the bitmap file to be created

returns **:** none

void fillRGBBuffers(const unsigned char * buffer)

definition **:** Fills the RGB color buffers by using the input buffer.

parameters:
buffer **:** buffer having the RGB values in order.

returns **:** none

void slideBitmap(int yOffset)

definition **:** slides the bitmap in upwards with the pixel number given. The upper part slides from the bottom of the image. This method is used to equalize the y coordinates of two bitmaps when there is difference in y coordinates between the initial points of the bitmaps.

parameters:
yOffset **:** the amount of pixels that bitmap file will be slided.

returns **:** None

static void construct3DMap(Bitmap bmpRight, Bitmap bmpLeft, int *x, int *y, double * xval, double * yval, double *zval, int uppest, int downest)

definition **:** constructs 3D map of the given two bitmaps

parameters:
bmpRight **:** bitmap of the image that is taken from right side
bmpLeft **:** bitmap of the image that is taken from left side
x **:** coordinate array showing the x coordinates of the points in the right bitmap file
y **:** coordinate array showing the y coordinates of the points in the right bitmap file
xval **:** coordinate array showing the x coordinates of the points arranged according to the disparity between bitmaps
yval **:** coordinate array showing the y coordinates of the points arranged according to the disparity between bitmaps

zval **:** disparity array showing the disparities of the points
uppest **:** coordinate of the uppest point which is common in both right and left bitmap

downest **:** coordinate of the downest point which is common in both right and left bitmap

returns **:** none

A.3.4. CameraView

A.3.4.1 Definition:

This class is used as an interface between the cameras and the software. Initializes the cameras, capture images and sends them to Bitmap class.

A.3.4.2 Attributes

int iCameraNumber **:** number of cameras connected to firewire card.
C1394Camera theCamera **:** camera object that shows the camera properties
unsigned long w, h **:** width and height values of the bitmap file.
unsigned char **bitmap **:** buffer array which stores the rgb values of the image.
int m_pBitmapLength **:** total bitmap length created after capturing the image. Changes by the resolution of the image.

A.3.4.3 Methods:

CameraView()

definition **:** this is the default constructor for CameraView class. Initializes the class attributes.

parameters **:** none

returns **:** none

bool initializeCameras()

definition **:** Finds the camera number connected to firewire card and initializes bitmap number according to this number.

parameters **:** none

returns:
FALSE - if no camera connected or firewire card is used by another program
TRUE – elsewhere

bool captureImage(int cameraNumber)

definition **:** This method performs capturing the image from each camera and fills the bitmap array according to the images.

parameters:

cameraNumber: This is the number of cameras found during performing initializeCameras method.

returns:
FALSE - if error occurred during capturing the image
TRUE – elsewhere

A.4 Definitions of User Interface Methods

A.4.1 Definition:

User interface methods supply the interface between the buttons; edit boxes and the bitmap algorithms.

A.3.4.2 Attributes

CPoint lPoint[5], rPoint[5] : initial point coordinates for bitmap files.
bool bPointSelected : set to true when initial point is selected.
bool bPointPointed : set to true when initial point is pointed.
int pictureNumber : number of the picture shown on the screen.
int yOffset : y-coordinate difference between the right and left bitmaps. It is determined after initial points are set.
int nontexturedBitmap : number of the picture which is captured without texture reflection.
char right[5][100], left[5][100]: name of the bitmap files in character array form.
CString rFilename[5], lFilename[5]: name of the bitmap files in CString form.

A.3.4.3 Methods:

bool OnInitDialog()

definition : operates on the initialization phase of user interface, initializes the attributes of CPointCloudDlg class

parameters : none

returns : false if initialization fails, true elsewhere.

void DrawPoint (CPaintDC *dc, CPoint pLoc, int r,int g,int b)

definition **:** draws a point on the screen where initial point is marked

parameters:
dc **:** dc object to paint
pLoc **:** point location to draw
r, g, b **:** RGB values of the color to draw the point

returns **:** none

void OnOK()

definition **:** operates when "Take Photo" button is pressed take 5 images from each camera with 1 second time difference between capturing the images view the first image on the user interface screen

parameters **:** none

returns **:** none

void OpenFile (CString m_sFilename1)

definition **:** opens the bitmap file with the given name and views it on the user interface

parameters **:**
m_sFilename1: name of the bitmap file that will be shown on the screen.

returns **:** none

void OnScanFromFile()

definition **:** operates when "Scan From File" button is pressed. This method makes the same analyse with Take Photo button. Only difference is image is not captured in this method, instead images in the folder is analysed.

parameters **:** none

returns **:** none

void OnSetInitialPoint()

definition **:** operates when "Set Initial Point" button is pressed sets the initial point for the given image and views the next image when all initial points are set, start analyse.

parameters **:** none

returns **:** none

void OnStartScan()

definition **:** operates when start scan button is pressed. This method starts analysis of bitmap files.

parameters **:** none

returns **:** none

void OnRestart()

definition **:** operates when restart button is pressed. It restarts the whole operation from the beginning.

parameters **:** none

returns **:** none

void OnExit()

definition **:** operates when exit button is pressed. it exits from the program.

parameters **:** none

returns **:** none

void analyse()

definition **:** this is the main method. analyse the bitmap files and create an output file in the required format

- find the nontextured bitmap
- analyse the 5 bitmap pairs
- collect the output matrices

- linearize the final matrices
- create the output file

parameters **:** none

returns **:** none

void analyseBitmaps (BitmapPoint firstPoint, BitmapPoint secondPoint, char * firstPicture, char * secondPicture, double ***xMatrix, double ***yMatrix, double ***zMatrix, int &matrixRow, int &matrixColumn, int ***xCoor, int ***yCoor, int iPictureNumber)

definition **:** analyse the input bitmaps and creates three matrices holding x, y and z coordinates of the textured points on the face

parameters **:**
firstPoint **:** initial point of the right bitmap file
secondPoint **:** initial point of the left bitmap file
firstPicture **:** name of the right bitmap file
secondPicture: name of the left bitmap file
xMatrix **:** output parameter that will be filled with x coordinates
yMatrix **:** output parameter that will be filled with y coordinates
zMatrix **:** output parameter that will be filled with z coordinates
matrixRow **:** output parameter that will be filled with row value of the output matrices
matrixColumn: output parameter that will be filled with column value of the output matrices
xCoor **:** stores the x-coordinates of the points before disparity calculation
yCoor **:** stores the y-coordinates of the points before disparity calculation
iPictureNumber: count of the input bitmap files from a total of 5

returns **:** none

A.4 Definitions of Common Functionalities

A.4.1 linearization

A.4.1.1 Definition:

Common functions are used for linearization of 3D point cloud values.

A.4.1.2 Fuınctions:

void linearizePointCoordinates(double **X, double **Y, double **Z, int &row, &int column)

definition **:** linearize the 3D point cloud and fills the gaps.

parameters:
X **:** x-coordinates of 3D point cloud
Y **:** y-coordinates of 3D point cloud
Z **:** z-coordinates of 3D point cloud
row, column **:** row and column number of bitmap file

returns **:** none

void makeOutputFile(char *outputFileName, int row, int column, double **X, double **Y, double **Z)

definition : **creates an output file and fills with the input matrices in the required format**

parameters:
outputFileName : **name of the output file to be created**
row, column : **row and column numbers of input matrices**
X, Y, Z : **input matrices which store x, y and z coordinates**

returns : **none**

void makeOutputRGBFile(char *outputFileName, int row, int column, int **XCoorFinal, int **YcoorFinal)

definition : creates an output file and fills with the x and y-coordinates before disparity calculation.

parameters:
outputFileName : **name of the output file to be created**
row, column : **row and column numbers of input matrices**
xCoorFinal : **input matrices which store x-coordinates before disparity calculation**
yCoorFinal : **input matrices which store y-coordinates before disparity calculation**

returns : **none**

void findAverage(double ** Matrix, int i, int j)

definition **:** calculates the average point of neighbouring points in a matrix

parameters:
Matrix **:** input matrix
i **:** column coordinate of the input point
j **:** row coordinate of the input point

returns **:** none

void generateIsSpace(double ** Z, bool ** isSpace, int column, int row)

definition **:** fills isSpace boolean array according to depth information of input bitmap file. isSpace is filled with true if the coordinate is a space and shall be filled later, with false if the coordinate has a normal depth and no more operation is needed.

parameters:
Z **:** depth information of input bitmap file
isSpace **:** output boolean array that shows if the point is empty or not
column **:** column number of depth information
row **:** row number of depth information

returns **:** none

void initialLinearization(double ** Z, int column, int row)

definition **:** fills the empty coordinates which have four neighbors which are valid, by calculating the average of their neighbor points.

parameters:
Z **:** input matrix
column **:** column number of Matrix
row **:** row number of Matrix

returns **:** none

A.4.2 bitmap_functions

A.4.2.1 Definition:

This file performs the common functionalities used in other classes.

A.4.2.2 Functions:

void makeBitmap(unsigned char ** garray, int width, int height, char * dHeader, char * output_name)

definition **:** creates the bitmap file in gray scale form with the given name.

parameters:
garray **:** the array of points to construct the bitmap file
width and height **:** the width and height properties of the bitmap file
dHeader **:** the header information of the bitmap file
output_name **:** name of the bitmap file to be created

returns **:** none

void makeRGBBitmap(unsigned char ** dr, unsigned char ** dg, unsigned char ** db, int width, int height, char * dHeader, char * output_name)

definition **:** creates the bitmap file in renkli form.

parameters:
dr **:** the array of red points to construct the bitmap file
dg **:** the array of green points to construct the bitmap file
db **:** the array of blue points to construct the bitmap file
width and height **:** the width and height properties of the bitmap file
dHeader **:** the header information of the bitmap file
output_name **:** the name of the bitmap file to be created

returns **:** none

void detectPeakPointsHorizontally(unsigned char ** garray, unsigned char ** array, int width, int height)

definition **:** fills the array by operating peak point algorithm on gray scale array horizontally.

parameters:
garray **:** array of input points
array **:** array of output points
width **:** the width property of the bitmap file
height **:** the height property of the bitmap file

returns **:** none

void makeBitmapForLines(BitmapLine* lines, int lineNumberToRight, int lineNumberToLeft, int width, int height, char * dHeader, char * output_name)

definition **:** creates the bitmap file with the lines detected.

parameters:
lines **:** initial line detected in the bitmap file
lineNumberToRight **:** number of the lines in the right of the initial line
lineNumberToLeft **:** number of the lines in the left of the initial line
width and height **:** the width and height properties of the bitmap file
dHeader **:** the header information of the bitmap file
output_name **:** the name of the bitmap file

returns **:** none

void collectMatrices(double ***xMatrix, double ***yMatrix, double ***zMatrix, int ***xCoor, int ***yCoor, int *matrixRow, int *matrixColumn, double ***xMatrixFinal, double ***yMatrixFinal, double ***zMatrixFinal, int ***xCoorFinal, int ***yCoorFinal, int &matrixRowFinal, int &matrixColumnFinal, int nontexturedBitmap)

definition **:** collect 5 matrices and obtain a final matrix for each of x, y and z matrices.

parameters:
xMatrix **:** input double dimension matrix array containing x coordinates
yMatrix **:** input double dimension matrix array containing y coordinates
zMatrix **:** input double dimension matrix array containing z coordinates
xCoor **:** input double dimension matrix array containing x coordinates before disparity calculation
yCoor **:** input double dimension matrix array containing y coordinates before disparity calculation
matrixRow **:** input integer array containing matrix row values
matrixColumn **:** input integer array containing matrix column values
xMatrixFinal **:** output matrix storing final x coordinates
yMatrixFinal **:** output matrix storing final y coordinates
zMatrixFinal **:** output matrix storing final z coordinates
xCoorFinal **:** output matrix storing final x coors before disparity calculation
yCoorFinal **:** output matrix storing final y coors before disparity calculation
matrixRowFinal **:** row number of final output matrices
matrixColumnFinal **:** column number of final output matrices
nontexturedBitmap **:** number of image which is taken without texture reflection

APPENDIX B

USER MANUAL

The steps to be followed in order to run the 3D Scanner system and observe the results, is as follows:

1) Put the setup on top of the projector.

2) Connect the graphics card output of the computer to the input of the projector in order to project the texture. Open texture.ppt file from the computer, which is connected to projector, then press F5.

3) Connect the cameras to the firewire card of the computer on which the 3DFaceScanner program is installed.

4) Run the 3DFaceScanner.exe program and verify that the screen displayed is as the screenshot shown in Figure B.1.

5) Position the person or object to be scanned at a distance where texture falls on optimally (30-40 cm away from the cameras).

6) After entering the name of the object of person into the “Name” field of the program, hit the “Take Photo” button.

7) On the first obtained image, find the initial point, which is displayed by the texture as a red point, and mark the point using the left mouse button. The point can be changed using the left mouse button until the “Set Initial Point” button is pressed.

8) After hitting the “Set Initial Point” button, the second image is displayed. Repeat the same procedure for all the pictures. For the stereo image pair, which is captured without texture reflection, press “Set Initial Point” button without setting a point.

9) When initial points are set for all 10 pictures, “Start Scan” button will be enabled. Press “Start Scan” to start analysing the bitmaps.

10) There is time difference between capturing the image from each camera, so sometimes synchronization problems may occur. (for example for two bitmap files captured at the same time, one of the images may be seen as textured and the other non-textured.) To overcome this problem, “Restart” button can be pressed any time problem is seen, to restart the whole process.

11) If it is wanted to re-analyse the image pairs captured before and stored in the folder, name of the person is written to “Person’s Name” field, then “Scan From File” button is pressed. In this situation, pre-captured images are shown on the screen one by one and after setting initial points, analyse process is started when “Start Scan” button is pressed.

12) When the scanning process is finished, “Exit” button is enabled. Program exits when “Exit” button is pressed.

13) The next phase is to post process the output data for obtaining a better obj file. During 3D Scan process, a folder with the person’s name is created and required

files are placed in this folder including a matlab file with the person's name. By executing this file, obj file is created in 8-12 min. period. This file can be viewed by any obj file viewer. (e.g. Polyworks Imview)

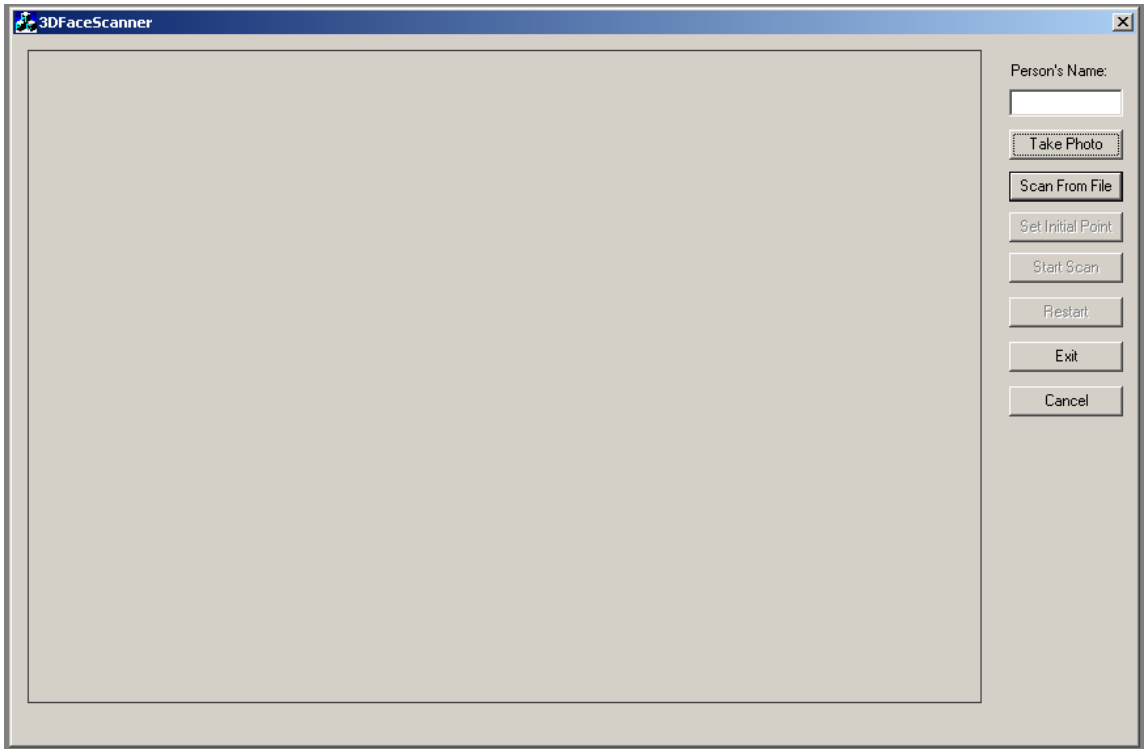

Figure B.1: Screenshot of 3D Face Scanner program

APPENDIX C

GENERAL INFORMATION ABOUT ".OBJ" FILE FORMAT

The OBJ file format is a simple data-format that represents 3D geometry. This file format includes four types of data: the position of each vertex, the texture coordinate associated with a vertex, the normal at each vertex, and the meshes that make each polygon. It is a universally accepted format.

In a typical ".obj" file structure,
3D vertex coordinates are written on a single line, which starts with the letter 'v'. Each line starting with 'v' increments the vertex index and these indexes are used while forming the meshes. These vertexes are the elements of 3D point cloud shape data constructed for the scanned object.

2D texture corner coordinates are written after 'vt'. Each 'vt' increments the texture corner index and these indexes are used while covering each mesh. The texture coordinates are obtained from images captured from the scanned object.

Normal value of each vertex is written after 'vn'. Each 'vn' increments the normal index and these indexes are used while constructing the surface.

The vertex, texture corner and normal value indexes are written after 'f' in order to obtain a face. These meshes will come together to form the total 3D model. If a mesh is made up of four vertexes, each quartet of numbers specifies a geometric vertex, texture

vertex, and vertex normal. The reference numbers must be in order and must be separated by slashes (/).

A simple mesh example is as below:

```
v       x-coor(1)       y-coor(1)       z-coor(1)
v       x-coor(2)       y-coor(2)       z-coor(2)
v       x-coor(3)       y-coor(3)       z-coor(3)
v       x-coor(4)       y-coor(4)       z-coor(4)

vt      x-texture(1)    y-texture(1)
vt      x-texture(2)    y-texture(2)
vt      x-texture(3)    y-texture(3)
vt      x-texture(4)    y-texture(4)

vn      x-normal(1)     y-normal(1)     z-normal(1)
vn      x-normal(2)     y-normal(2)     z-normal(2)
vn      x-normal(3)     y-normal(3)     z-normal(3)
vn      x-normal(4)     y-normal(4)     z-normal(4)

f       1/1/1   2/2/2   3/3/3   4/4/4
```

After all the points in the 3D point cloud are stored as vertexes and meshes are formed with neighbouring vertexes, meshes can be covered according to the texture corners of the image file which is in bitmap file format in our case.