2D/3D IMAGING SIMULATOR

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

BY

NESLİ BOZKURT

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

NOVEMBER 2008

Approval of the thesis

"**2D/3D IMAGING SIMULATOR"**

submitted by **Nesli Bozkurt** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering** by,

Prof. Dr. Canan Özgen _____
Dean, **Graduate School of Natural and Applied Sciences**

Prof. Dr. İsmet Erkmen _____
Department Chair, **Electrical and Electronics Engineering Department**

Prof. Dr. Uğur Halıcı _____
Supervisor, **Electrical and Electronics Engineering Department**

Examining Committee Members:

Prof. Dr. Kemal Leblebicioğlu _____
Department of Electrical and Electronics Engineering, METU

Prof. Dr. Uğur Halıcı _____
Department of Electrical and Electronics Engineering, METU

Assist. Prof. Dr. İlkay Ulusoy _____
Department of Electrical and Electronics Engineering, METU

Assoc. Prof. Dr. Veysi İşler _____
Department of Computer Engineering, METU

Ahmet Oğuz Öztürk _____
TÜBİTAK, Uzay Teknolojileri Araştırma Enstitüsü

Date: _____

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: Nesli Bozkurt

Signature:

ABSTRACT

2D/3D IMAGING SIMULATOR

Bozkurt, Nesli

M.S., Department of Electrical and Electronics Engineering

Supervisor: Prof. Dr. Uğur Halıcı

November 2008, 84 pages

3D modeling of real objects has an increasing importance in numerous areas. Although many methods and solutions are already proposed for 3D data acquisition, research continuing in this area is still intense. However, a crucial drawback about 3D data extraction algorithms is their testing and validation difficulty. Additionally, obtaining calibrated 2D and 3D imaging systems is troublesome due to their high effort demand for calibration and high cost. In this thesis, a 2D/3D Imaging Simulator is proposed in order to ease development and testing of 3D data interpretations of different methods and also to generate synthetic images for miscellaneous use. Furthermore, an example application on FRGC database is explained in detail.

Keywords: 2D/3D Imaging, Simulation, Face Model Preprocessing, FRGC, Scanner

ÖZ

2B/3B GÖRÜNTÜLEME SİMÜLATORÜ
Bozkurt, Nesli
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Uğur Halıcı

Kasım 2008, 84 sayfa

Gerçek cisimlerin 3B modellenmesi çeşitli alanlarda artan bir öneme sahiptir. 3B veri elde edilmesi konusunda halihazırda birçok yöntem ve çözüm önerilmiş olmasına rağmen, bu alandaki çalışmalar halen yoğun bir şekilde sürmektedir. Ancak, testleri ve geçerlilik denetimleri, 3B veri çıkarım algoritmaları ile ilgili önemli bir dezavantaj teşkil etmektedir. Ayrıca, kalibre edilmiş 2B ve 3B görüntüleme sistemlerinin edinimi, kalibrasyon için yüksek çaba ihtiyacından ve yüksek masrafından dolayı sıkıntılıdır. Bu tezde, farklı yöntemlerle 3B veri analizlerinin geliştirilmesini, test edilmesini kolaylaştırma ve bunun yanında muhtelif kullanım amaçlı yapay veri üretme amacıyla bir 2B/3B Görüntüleme Simülatörü önerilmektedir. Ek olarak, FRGC veri tabanı üzerinde bir örnek uygulama da detaylı olarak anlatılmıştır.

Anahtar Kelimeler: 2B/3B Görüntüleme, Simülasyon, Yüz Modelleri Ön-işlemesi, FRGC, Tarayıcı

ACKNOWLEDGMENTS

I would like to express my utmost gratitude and respect to my supervisor Prof. Dr. Uğur Halıcı who made invaluable contributions to this thesis and also to Asst. Prof. İlkay Ulusoy who guided me in my studies as my research project manager. Without their extensive vision, positive approach and efficacious communication, this work would not be as it is.

I am grateful to Erdem Akagündüz, who always provided me with insightful comments and invaluable guidance and whom I will always be delighted to work with.

I would like to express my gratitude and my love to my best friend, my peaceful shelter and my husband, Ufuk Erdogmus for his endless encouragement and soothing love.

Finally, I also would like to thank to my family for their everlasting love and support.

This thesis is partially written within the context of the METU BAP Project, entitled "3D Face Scanner" conducted in METU Computer Vision and Intelligent Systems Research Laboratory.

To My Family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Today, computer graphics technology is capable of rendering very realistic images of the 3D models. In both hardware and software cases, powerful graphics tools are available. 3D models are used for different purposes such as; documentation of fragile and unique objects like historical artifacts [1] or masterpieces, entertainment, web commerce and also in dental and orthopedics.

At this point, a strong need for easy and low-cost acquisition of 3D models is developed. At the very beginning, even before the existence of 3D modeling packages, everything was digitized by hand or sketched on graph paper and the numbers were entered by using any text editor.

After the 3D modeling packages came out, the workload was a bit decreased but it was still a very time consuming process and the achievable level of detail and realism was not enough. Therefore, many different approaches for “sampling” the world were born and this research area is still expanding rapidly [2].

In order to create 3D models of the world, today’s one of the best solutions is 3D non-contact scanners. 3D scanner is a device to analyze an object to mainly collect data on its shape. The purpose of this device is to create a point cloud of geometric

samples on the surface of the subject. Using these points, the shape of the object can be reconstructed. Like cameras, 3D scanners also have a cone-like field of view and they can not collect the depth information for the obscured surfaces. Eventually, a "depth-image" is produced by the 3D scanner which includes the distance to a surface at each point of the picture.

Many kinds of non-contact scanners exists which can be classified as active and passive. Active scanners function by emitting some kind of radiation or light and detect its reflection to interpret a distance value. Whereas, passive scanners take advantage of the visual cues that already exist in the scene.

In both cases, interpreting and processing the obtained data which can be done by using dozens of algorithms, is crucial. However, it is not an easy task to test and verify those algorithms used. Multiple scan models of different kinds are needed for verification. For instance, a laser scan data may be needed as the ground truth when a stereo system or the post-processing results of the same system is to be tested. Apart from high cost of the scanning systems, it also needs a remarkable effort to merge imaging systems and calibrate them accordingly. Additionally, acquiring these two data simultaneously is not very straightforward. Usually, the images are taken consecutively and most of the time, the unstable scene and human imperfection prevent the two images to be identical.

As in many situations dealing with similar issues, the best option is using computer simulations. Simulators are tools that imitate real systems and can be utilized for analysis of the system for research, development and testing of new algorithms or sub-systems. Simulation is the process of designing a model of a real or imagined system and conducting experiments with that model. It allows the analysis of a system's capabilities, capacities, and behaviors without requiring the construction of or experimentation with the real system [3].

In our case, simulator is being used referring to a "virtual environment" that allows users to manipulate the objects within this environment and collect the needed data.

The functioning and the use of the 2D/3D Imaging Simulator will be explained in detail in the following chapters, but before that, brief information about the applicable algorithms and methods will be presented.

1.1 Computer Vision

The advance in computer graphics, starting from early 60's, let 3D modeling into both academic and industrial world as well as making it an important part of the everyday life. With the technologic improvements in computer graphics hardware, we have entered a new era, where the acquisition of 3D data is massively demanded.

Possible applications of the 3D data are [29]:

- industrial inspection for 3-D objects (quality control, deformation analysis, food inspection, printed web defect analysis)
- 3-D sensing (three dimensional measurement of objects), 3-D growth monitoring
- Z-keying
- novel view synthesis, image-based rendering, virtual environments
- autonomous vehicles, robotics
- medical, biomedical and bioengineering (stereoendoscopy, stereoradiographs, automatic creation of three dimensional model of a human face or dental structure from stereo images)
- scanning electron microscope
- surveillance (motion tracking and object tracking to measure paths)
- transport (traffic scene analysis)
- digital photogrammetry, remote sensing (generating Digital Elevation Models, surveying, cartography)
- 3-D database for urban and town planning
- stereolithography, stereosculpting (automatic acquisition of digital 3-D information used in CAD-CAM systems. This information can be fed into computer controlled milling machines for rapid solid modeling)

- asset monitoring and management
- 3-D model creation for e-commence or on-line shopping

Today, for the computers, it is not enough only to see what is in front of them. Machines also would like to know how far the object that they see is. Thus, apart from the x and y coordinates, the z coordinate - depth - is also to be calculated. In the area of computer vision, researchers have been working for many years to provide these capabilities for machines. The initial work was targeted towards robotics and automation, e.g. allowing a robot to navigate through an unknown environment. In recent years the focus has shifted to visualization and communication, resulting in much more interaction with the computer graphics community [2].

For this purpose, numerous methods are proposed. Here, three of them will be mentioned as they are applicable to the constructed simulator:

1.1.1 Shape from Shading

The method that is known as “shape from shading” method in the literature aims to resolve the problem to derive the 3D shape information from shading in a single two-dimensional image of the object. It was first formulated by Horn [4] in 1975. In the first years, researchers focused on the computational part of the problem. Nevertheless, due to the poor quality of the results, questions about existence and the uniqueness of the solutions as well as the ones related to the convergence of numerical schemes for computing the solutions became central in the last decade of the 20th century [5]. Still known to be an ill-posed problem, the method is highly popular due to its minimal data requirement.

The gradual variation of the shade in an image reveals very important information about the 3D shape of the surface. In order to understand the shape from shading

problem, firstly it is important to study how the image intensities are formed. Yet again, finding a unique solution is difficult and many assumptions have to be made.

Assuming that a surface is composed of continuous patches, the measured brightness on the image E(x,y) will be determined by three main factors: Material properties of the patch, the direction of illumination and the patch orientation, or surface normal in other words. Now, let's have a closer look at how the image brightness changes:

Considering the geometry in Figure 1, the amount of light received by the sensor depends on the incident light intensity, the incidence angle, the emittance angle and the phase angle. If we assume that the incident ray intensity is I_1 and the unit area illuminated is dS, then the light falling to this unit area is calculated as: I1 cos(i) dS

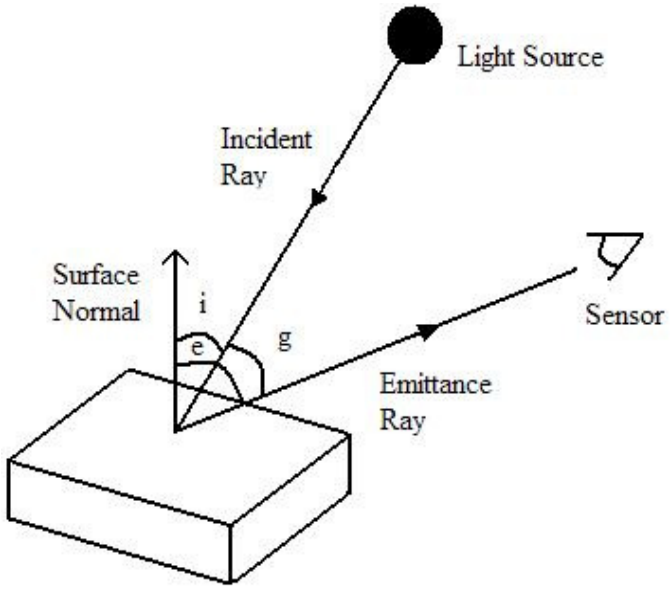

Figure 1 Lighting geometry and incidence (i), emittance (e) and phase angles

Reflectance of a surface can be defined as the proportion of the reflected radiation intensity to the incident radiation intensity. The amount of light that is percepted by the sensor can be formulated as: I_2 cos(e) dS dw where I_2 is the emitted ray intensity and dw is the solid angle subtended by the sensor area to the unit surface. So, reflectivity function is I_2/I_1.

The relation between the image brightness and the reflectivity map can be simplified to $E(x_1, x_2) = R(n(x_1, x_2))$ where x_1 and x_2 are the coordinates of a point **P** in the image, R reflectivity map and E is the image brightness[5]. Here, R is a function of surface normal at (x_1,x_2), in other words, the surface orientation. The surface orientation is characterized by the components of the surface gradient in x- and y-directions.

$$p = \frac{\delta z}{\delta x} \quad q = \frac{\delta z}{\delta y} \quad n = (-p,-q,1)^T \Longrightarrow \quad E(x, y) = R(p, q) \qquad \textbf{Equation 1.1}$$

The Equation 1.1 describes the mapping between x,y coordinate space of the image and the p,q gradient-space of the surface. However, additional information is needed to uniquely solve the partial differential equation derived from this relation. This additional constraint is the assumption of surface smoothness. In this way, arbitrariness for the neighboring gradient assumptions is avoided.

Starting from this point, the relation between gradient and image coordinates can be expressed in various ways for different conditions, such as orthographic SFS with a far light source in [6], [7], perspective SFS with a far light source in [8], [9], [10] or perspective SFS with a point light source at the optical center in [11], [12]. In most cases, taking the surface as Lambertian is sufficient where reflectance is the cosine of the incidence angle.

By assuming a smooth surface, the differential equations for the gradient values at neighboring points can be solved for a specific direction. This way, strips of 3D models are obtained.

However, in addition to the smoothness assumption for the surface, in order to obtain the whole surface, an initial point (x_0,y_0,z_0,p_0,q_0) with known coordinates and gradients for each strip is needed. Such set of points can be obtained around a

neighborhood isolated (singular) maximum at $R(p_0,q_0)=E(x_0,y_0)$ or occluding boundaries. [13]

1.1.2 Stereo Reconstruction

Stereo reconstruction method, based on the same principle as the human visual system, is one of the earliest solutions proposed for 3D reconstruction. The problem of reconstructing scene geometry from calibrated stereo image pairs has a 30-year history in computer vision. However, as pointed out in [14], biological evolution solved this problem million years ago, and humans have studied the concept of stereo vision since at least the time of Leonardo ad Vinci[15].

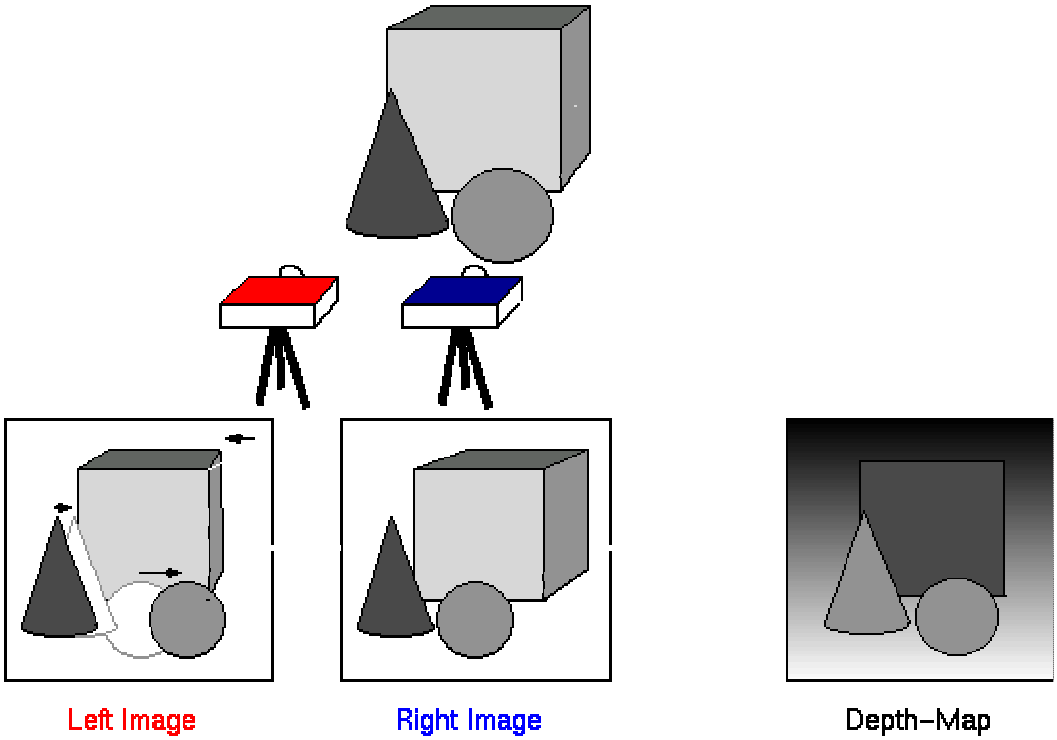

Figure 2 Stereo imaging geometry and the depth map [30]

Contrary to shape from shading method, stereo reconstruction functions with at least two images of the same scene from different angles. Stereo image pairs contain the

depth information about the scene. In each image, the same point in the scene is projected on different pixels due to the distance in between the point and the sensors. Using these relations, the 3D coordinates of the point can be extracted. The simplest case for stereo imaging is using two cameras with parallel viewing directions and separated by a distance ***d***. In order to simplify the system further, the viewing directions can be perpendicular to the line connecting the camera sensors.

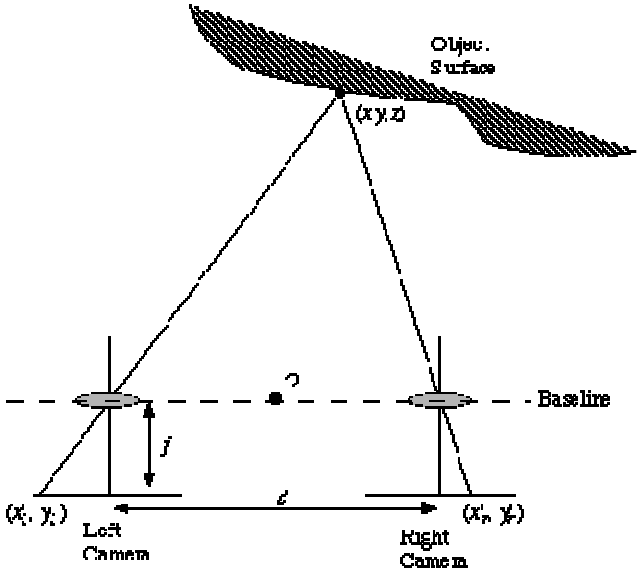

Figure 3 A simplified stereo imaging system[31]

If we consider a point (x,y,z) in the scene in 3D world coordinates, f as the focal length of both cameras and take (x_l,y_l) and (x_r,y_r) as its projection on left and right image planes respectively, following equations on geometric relations of these points are obtained:

$$\frac{x_l}{f}=\frac{x+d/2}{z} \qquad \frac{x_r}{f}=\frac{x-d/2}{z} \qquad \frac{y_l}{f}=\frac{y_r}{f}=\frac{y}{z} \qquad \textbf{Equation 1.2}$$

By solving these equations for (x,y,z), the 3D coordinate of the point can be calculated as:

$$x = \frac{d(x_l + x_r)}{2(x_l - x_r)} \qquad y = \frac{d(y_l + y_r)}{2(x_l - x_r)} \qquad z = \frac{d \cdot f}{x_l - x_r}$$ **Equation 1.3**

Disparity is defined as the quantity "x_l-x_r" which appears in these equations. It can be analyzed from the equations that the depth – z coordinate of the point is inversely proportional to disparity. The disparity of a fix point in the scene increases with the distance between the cameras. This leads to the following trade-off: If more accurate disparity maps are to be obtained, the distance between the cameras should be increased. Disparity can only be measured in pixel differences, hence far away objects that yield to small disparity cannot be distinguished in stereo pairs. By increasing the distance between the lenses, this problem can be overcome. However, as camera separation becomes larger, correlating the stereo images will be more difficult. The correlation problem is named as "*stereo correspondence problem*". To gain a better understanding of the problem let's look at the following Figure 4:

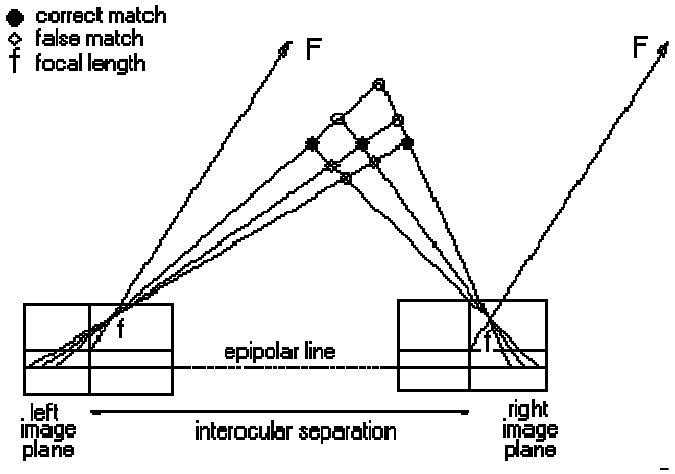

Figure 4 The stereo correspondence problem[32]

Each camera views 3 dots and the 9 dots represent all the possible matches that could be made, the black dots are correct matches whereas the rest are incorrect[27].

If we summarize the stereo approach with three distinct and widely accepted:

1. Detection and recording of the feature points in each image,
2. Matching of the corresponding feature points across the stereo pair,
3. Combination of left and right images using disparity equation and extraction of depth information,

then one can point out that the hardest part of this method is the step 2 that of identifying corresponding points in the two images.

To simplify this step, epipolar geometry is exploited. If we approximate the cameras by the pinhole camera model, geometric relations between a point in the scene and camera positions constraint the point's projections onto the stereo images. '*Epipolar Plane*' is the plane defined by the object point in the scene and the positions of the two cameras. This plane intersects the two image planes with the '*Epipolar Line*'s. The geometry of the system restricts that a point in one of the images must lie somewhere on the corresponding epipolar line in the other image. So, the matching problem of the feature points is somewhat simplified. Epipolar geometry is not valid only for parallel viewing angles, but also non-parallel ones.

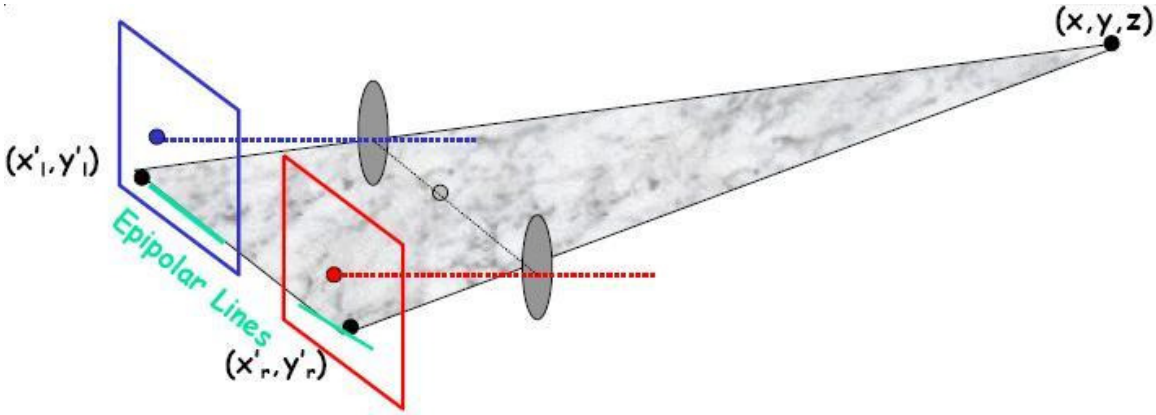

Figure 5 Epipolar plane and epipolar lines [13]

Stereo matching has been studied over decades in computer vision. The proposed approaches for interest point extraction from the images can be broadly classified into feature-based and correlation-based approaches. For both approaches it is important for the feature points to be well-defined and stable across the view.

In correlation-based algorithms, small image windows of a fixed size are used for matching, and the correspondence is decided by using a measure of the correlation of these windows within each image. Some important correlation-based approaches were proposed by Marr and Poggio[16], Grimson[17], Pollard, Mayhem and Frisby[18], Gimmel'Farb[19], Baker and Binford[20] and Ohta and Kanade[21]. Typically, correlation-based algorithms yield to dense depth maps but they are susceptible to noise and surface foreshortening in the stereo images. On the other hand, feature-based methods use a set of features to find correspondence in two images. These algorithms are more robust against noise but give sparse depth maps and interpolation is necessary. Numerous successful correlation-based approaches were proposed such as, Okutomi and Kanade[22], Cox et al.[23], Koch[24] and Falkenhagen[25,26].

Using *"Structured Light"* on the object and marking the pattern projected on the object as the feature points in both images greatly simplifies the stereo correspondence problem. This special application of structured light in a stereo vision system will be explained in detail in the "1.1.3 Structured Light" section.

1.1.3 Structured Light

Binocular stereo can produce a dense depth map of a scene, but the computed depth values may not be very accurate. Structured light on the other hand produces more accurate depth values and is considered as one of the most frequently used techniques in computer vision. It is the process of projecting a known pattern of pixels (often grids or horizontal bars) on to a scene [27].

As the light pattern is distorted by the surface shape, its projection onto the image plane carries more information on 3D model of the surface.

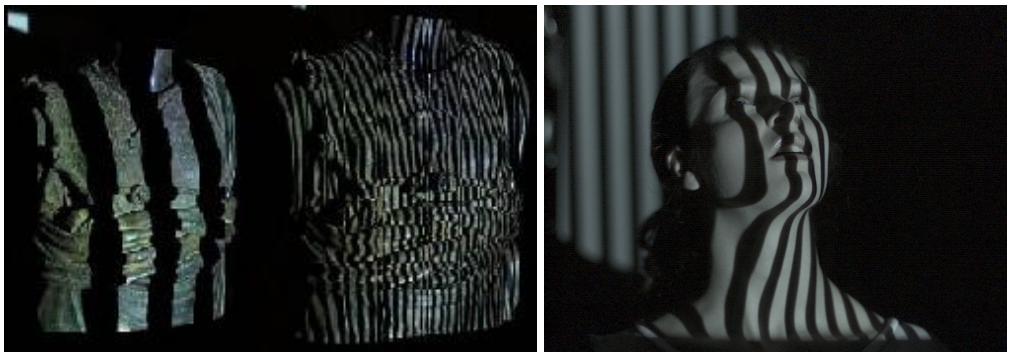

Figure 6 Different textures are projected onto the scene [34]

In order to get dense depth information, the projected texture has to be moved in the scene. This way whole surface can be scanned. Another technique is to project multiple stripes at once onto the scene. Structured light method requires good lighting control and for this reason, it can give satisfactory results only at indoor environments. For distinguishing between the stripes Coded Light Approach is used in which the stripes are coded either with different brightness or different colors. Hence, there is no correspondence problem.

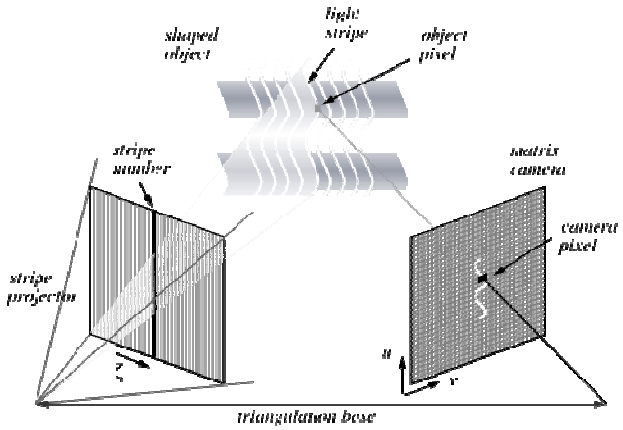

Figure 7 Structured light system and the deformation of a single stripe [35]

Actually, structured light system is an active stereo system where one of the cameras is replaced with a projector or laser unit. The triangulation concept that is used in stereo vision is also applicable in this system. The idea is that once the perspective projection matrix of the camera and the equations of the planes containing the sheets

of light relative to a global coordinate frame are computed from calibration, the triangulation for computing the 3-D coordinates of object points simply involves finding the intersection of a ray from the camera and a plane from the sheet of light of the projector or laser[33].

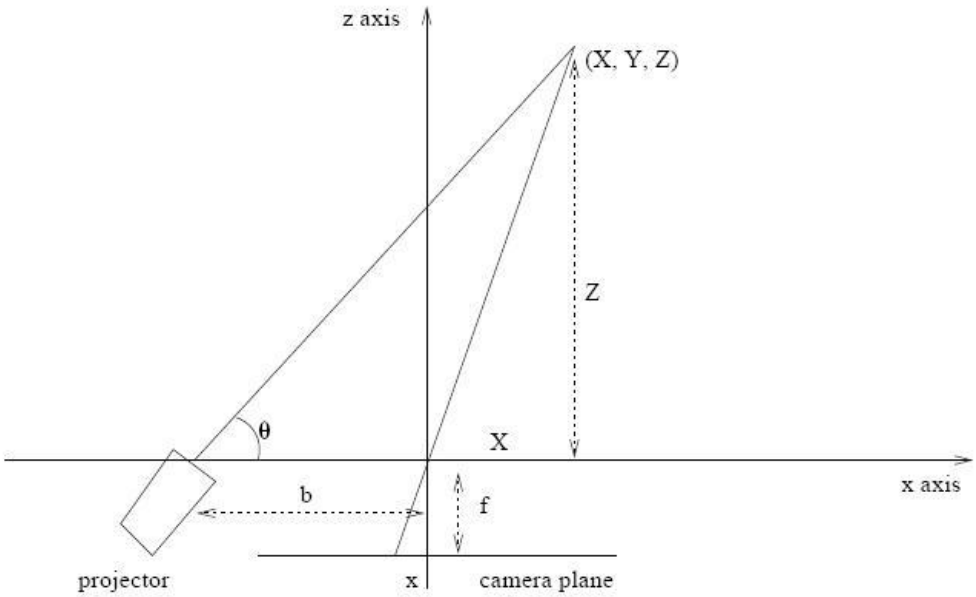

Figure 8 Structured light system geometry

From the given geometry given in Figure 8, (X,Y,Z) can be found as:

$$(X,Y,Z) = \frac{b}{f\cot\Theta - x}(x, y, f) \qquad \textbf{Equation 1.4}$$

In order to explain a more general approach, let's say that the calibrated camera has the perspective projection matrix C = $[q_{ij}]$. In the image captured by the camera, for any point (u,v) on a stripe of light following equations can be obtained using the calibration matrix:

$$(q_{11} - uq_{31})X + (q_{12} - uq_{32})X + (q_{13} - uq_{33})Z + (q_{14} - uq_{34}) = 0 \quad \textbf{Equation 1.5}$$

$$(q_{21} - uq_{31})X + (q_{22} - uq_{32})X + (q_{23} - uq_{33})Z + (q_{24} - uq_{34}) = 0 \quad \textbf{Equation 1.6}$$

In the same global coordinate system, the equation of the light stripe can also be written as in Equation 1.7.

$$aX + bY + cZ + d = 0 \quad \textbf{Equation 1.7}$$

We have the three unknowns as the 3D coordinates of the point and we can transform the equations to the matrix multiplication representation given in Equation 1.8.

$$\begin{bmatrix} q_{11} - uq_{31} & q_{12} - uq_{32} & q_{13} - uq_{33} \\ q_{21} - uq_{31} & q_{22} - uq_{32} & q_{23} - uq_{33} \\ a & b & c \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} uq_{34} - q_{14} \\ uq_{34} - q_{24} \\ -d \end{bmatrix} \quad \textbf{Equation 1.8}$$

By solving this system, X,Y,Z coordinates can be calculated.

The stripe detection, or in other words detection of the edge between illuminated and nom-illuminated areas is the crucial point for robustness of the method. Using the obtained equation system in Equation 1.8, points along the edges can be reconstructed and a dense range image can be computed to produce a realistic model of the surface.

In literature, there are also studies on using both stereo vision and structured light at the same time. One of the reasons of doing this is to recover the geometry of a non-rigid scene continuously in time[36]. Hence, the reconstruction of the human face while speaking is possible using stereo cameras for the capture. Structured light can

give quite accurate depth information but it is restrictive about the illumination and rigidity. On the other hand, stereo vision can also be used for video images where the scene is captured with two cameras. The geometry of the scene is first calculated by using structured light and afterwards, stereo system is used to track the object in 3D scene. Since the geometry is known at time t, the next image at time t+dt can be estimated. This way, the correspondences in image colors are easier to predict using local matching process and the depth map of the scene at t+dt is obtained. Finally, after all scenes are reconstructed, the video is transformed to have a third dimension.

Another reason for using stereo vision and structured light simultaneously is to avoid obstructions. Hence, the 3D coordinates for the points that are obscure to one of the cameras, can be calculated using the other one.

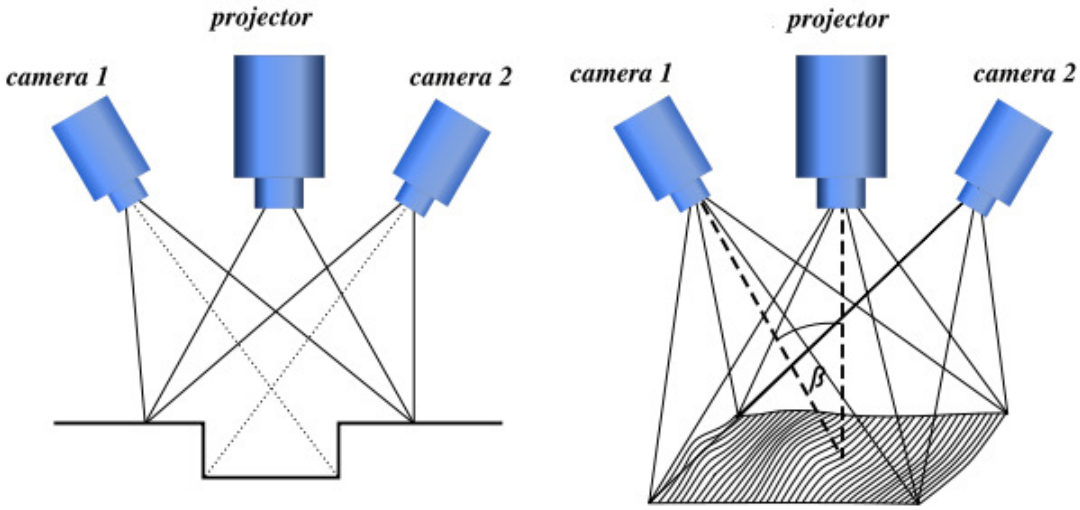

Figure 9 Fringe pattern recording system with 2 cameras [37]

As more cameras inserted in the system, more calibration is required. Calibrating two cameras and a projector is not a really easy task. Additionally, even the small errors in the obtained calibration matrices can result in serious miscalculations about the surface shape and position. In order to validate the algorithm used in imaging exactly, these defects should be eliminated.

1.2 2.5D Recognition

For decades, a lot of effort has been devoted to recognition - mostly face recognition, using images, namely 2D information. Many 2D recognition systems can achieve good results, however still the performances are highly dependent on the environmental effects. The difficulties arise with different illumination conditions and pose and facial expression variations.

Figure 10 Examples for illumination, pose and gesture variations [44]

Due to the mentioned problems, alternative approaches are being evaluated. One of these is the face recognition systems that use three-dimensional depth information and are becoming more and more important everyday[38, 39]. Obviously, unless the texture information is utilized, illumination drawback is eliminated. Still, negative effects of the pose and facial expression variations to the recognition system are to be handled.

However, to analyze these variations properly, 3D scans of the faces with either pose or facial expression variations. If these two parameters change simultaneously, their contributions to the extracted features to be matched cannot be easily disassociated. However, in real systems, elimination of such system and subject defects is almost impossible.

Numerous methods have been applied and many more are still being developed for 3d model based face recognition. These approaches can be simply exemplified as ICP based shape matching methods [38, 40] and 3d morphable model based methods [41, 42, 43].

CHAPTER 2

2D/3D Imaging Simulator

2D/3D Imaging Simulator is a tool that enables the user to create an environment by loading 3D models of numerous objects in any desired position and rotation. The user can also set the camera position and rotation which is used to take the picture or the 3D scan of the scene in front of it. Moreover, lights and texture projections can be added according to the user's inputs on various specifications.

The reason of the need for such a tool will be explained in the 'Motivation' section. Afterwards, in the 'Related Work' section, few other examples of this kind of tools will be presented. Detailed information about the attributions and the abilities of the tool will be given in the 'Features' section. Finally, in the 'Technical Specifications' section the technical properties and limitations will be reported.

2.1 Motivation

As mentioned before, the construction and maintenance of an imaging system can be very laborious and costly. The calibration of the cameras and projectors to be use is laborious and troublesome. Additionally, even if we assume that we have the perfect

system for imaging, we cannot control the position, rotation and the expression of the objects and this makes it hard to verify our results.

Figure 11 The stereo imaging system in METU VISION Research Laboratory

Moreover, even if we assume that the objects are also completely controllable, the need for the ground truth, which is the real 3D model in most cases, is impossible to acquire. Hence, in real system applications, results from another dependable source are used for verification and testing.

To overcome all this issues, 2D/3D Imaging Simulator can be utilized. As described previously, this tool imitates real systems and produces the corresponding 2D or 3D data according to the user's choices. Using the simulator, one can take multiple images of the same object from different angles and distances. This way the variations for illumination and expression can be eliminated.

The models reconstructed by using the methods that are explained in the previous chapter can be easily compared to the original model, and the success of the implementation can be measured correctly.

Additionally, in order to test the robustness of a 3D object recognition algorithm against rotation variations, multiple scans of a set of models can be taken from different angles and used in the recognition process.

Synthetic data acquisition is also extremely helpful to enrich a database as a matter of divergence and variety.

2.2 Related Work

Today, very few programs which are developed to simulate an imaging environment can be found, at least on the internet. There maybe some other tools that function similarly, however the only reachable ones will be mentioned briefly in the following subsections.

The first example is a virtual scanner developed in the Computer Science Department at the Johns Hopkins University, as a part of a 1.55 million dollar project called “Digital Hammurabi: High-Resolution 3D Imaging of Cuneiform Tablets”, to aid the researchers hardware and software development [45]. The second example is a virtual vision laboratory created by the Computer Science Department at the Columbia University, to provide some laboratory exercise for the students [46]. Finally, a realistic simulated robot for vision algorithms is designed by Claus C. Aranha, Schubert R. Carvalho and Luiz M.G. Goncalves, for ability to work on work on several aspects of cognition, attention, sensorimotor development, computer vision, and other research fields, without the immediate need of a real robot [47].

2.2.1 Virtual Scanner

As a part of the “Digital Hammurabi: High-Resolution 3D Imaging of Cuneiform Tablets” project conducted at the Johns Hopkins University, Computer Science

Department, a virtual 3D surface scanner is developed that can take a 3D model and "scans" it in one pass, leaving, much as a real scanner would, holes in the data. This scan provides the means to calculate the next best view and angle for subsequent passes of the scanner in order to fill in the holes.

The graphical user interface for setting the parameters of the virtual scanner is shown in the figure 12.

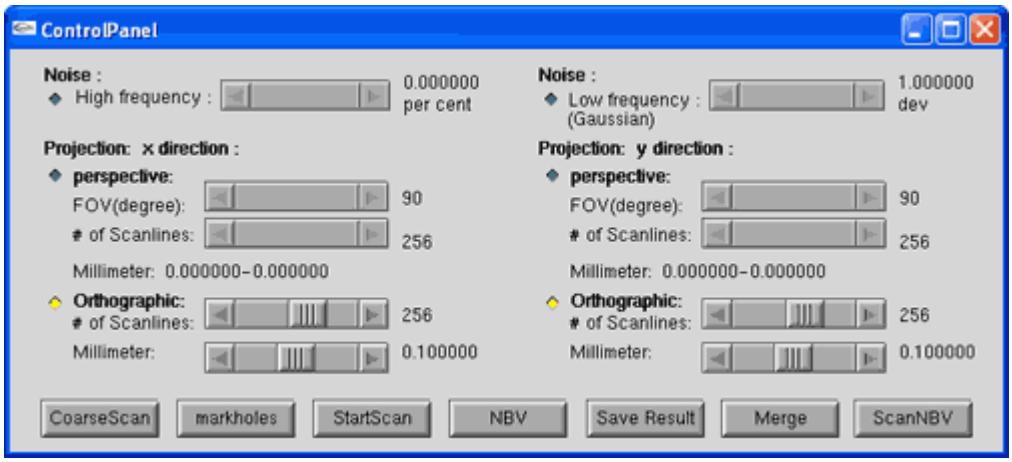

Figure 12 The user interface for setting the parameters of the virtual scanner [45]

This virtual scanner can model the noise consisting of holes that can be encountered in a real 3D scanner result. So that, as the subsequent scans are taken, the decision is made for the best view and angle so that the holes can be eliminated.

2.2.2 Virtual Vision Laboratory

Virtual Vision Laboratory [48] (VVL) is used create a realistic robotic vision lab that is accessible to students. The approach is to model a robotics lab that can be manipulated and moved as in a real work cell. All movement in the workspace is shown with animation, along with a simulated image from a robot mounted camera.

Apart from the robotic movements, stereo vision is realized for environment perception.

VVL is an educational tool to assist in teaching various aspects of robotics/computer vision. It was developed as a joint project between Columbia University, Lehigh University, Georgia Institute of Technology and funded by NSF Combined Research and Curriculuum Development Program, Award Number 9315517.

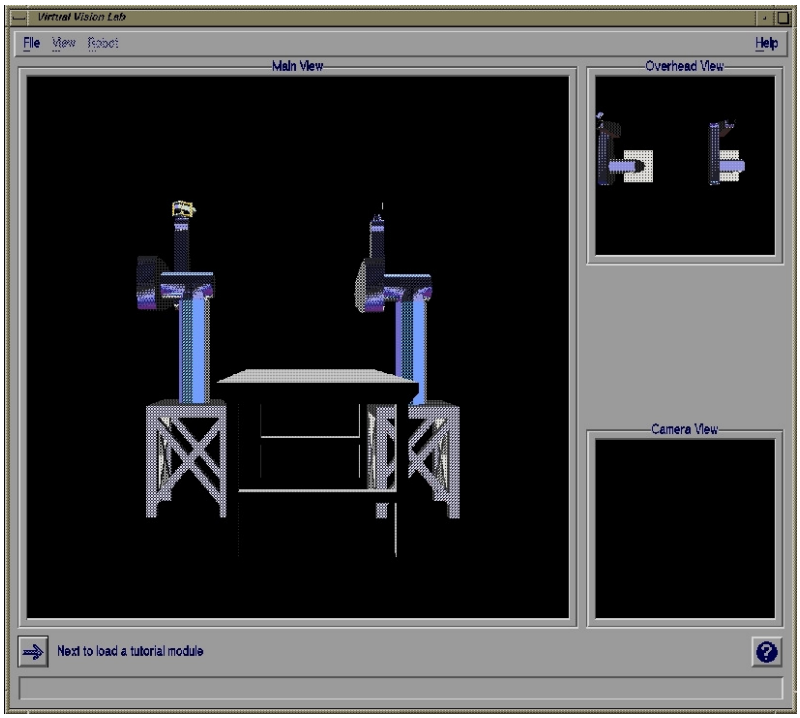

Figure 13 The tutorial window of the Virtual Vision Laboratory tool

The environment contains two Puma 500 six degree-of-freedom robots as well as objects that the robots manipulate. One of the robots holds a camera, while the other holds a gripper. The output of the camera is used so that the robot holding the gripper can locate, pick up and move objects. Real algorithms are used to interpret the camera output.

The tool has ‘Help’, ‘Tutorial’ and ‘Camera Calibration’ modules. ‘Help’ module gives information about the functionality and the features of the tool whereas the

'Tutorial' module lets the students follow the steps of an implemented experiment. Finally, in the 'Camera Calibration' module, the correspondence between the points in the acquired 2D images and the actual 3D locations. So that, eventually the robot can be instructed with the gripper to pick up an object in the real 3D world.

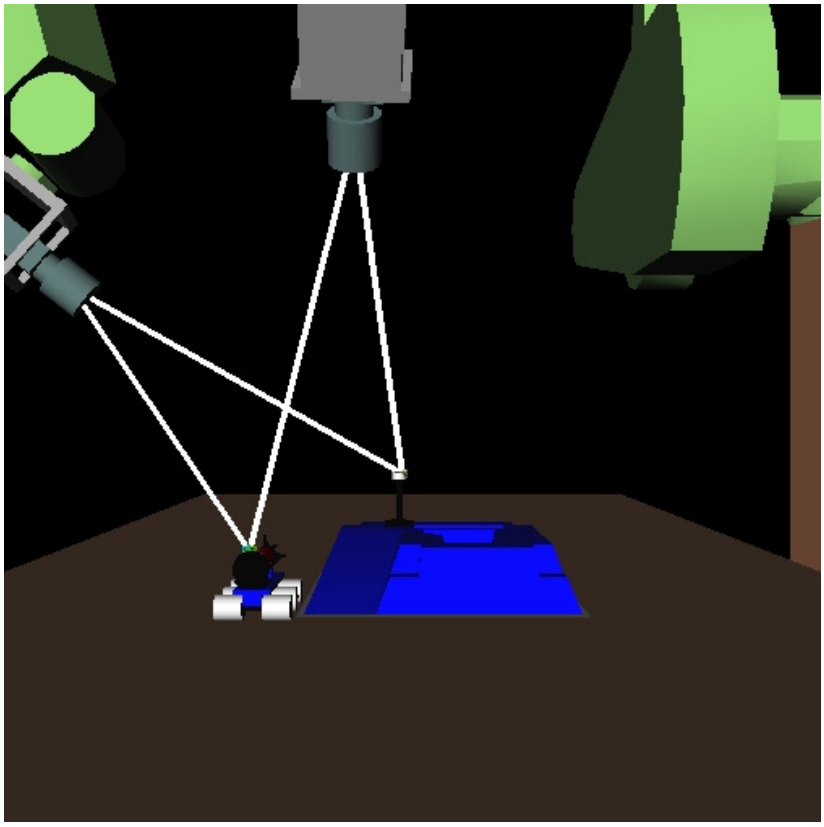

Figure 14 The two virtual cameras used for stereo vision

Due to the complexity, fragileness, and expense of actual robotic equipment, there are usually no hands-on resources available to students that would allow them to test topics they learn in class. However, this tool gives them to opportunity to visualize a realistic experiment and hence grasp the geometry behind.

2.2.3 Cabrio: A Realistic Simulated Robot

In [47], concepts and algorithms for control of visual motor commands and realistic

simulation of basic abilities as visual servoing and perception for robotics vision simulation are introduced. A humanoid robot, called Cambio, is simulated. This way, the usefulness of a simulated platform as an inexpensive alternative for testing and developing computer vision algorithms in real-time robotics applications is proven.

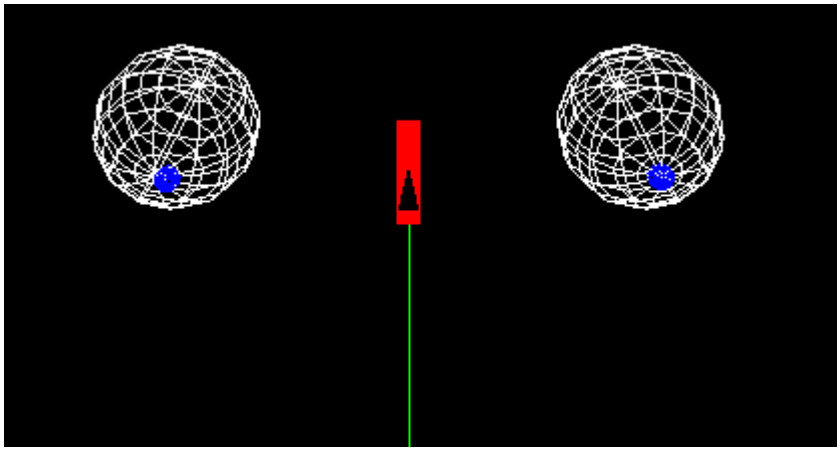

Figure 15 Stereo “eyes” of the robot which are able to verge to a place in the world

Similar to the Virtual Vision Laboratory, a dynamic simulation of a stereo head and its image processing capability and of two integrated arms for object manipulation is provided.

Figure 16 Robot in front of several different shaped and positioned cubes and the views from the left and the right eyes

Parallel to the driving motivation of this thesis, the main goal of Cambio is to allow researchers in Computer and Robotics Vision to perform experiments at reduced costs, running different tests with different robotic setups, using different testing algorithms, and in different situations putting the robot to its limits without the fear of breaking it up.

2.3 METU VISION 2D/3D Imaging Simulator

2D/3D Imaging Simulator developed in this thesis is a powerful tool for simulating an imaging environment and generating output data. Generally speaking, using this simulator is actually creating a world that consists of single/multiple objects, lights, a camera and a texture projector.

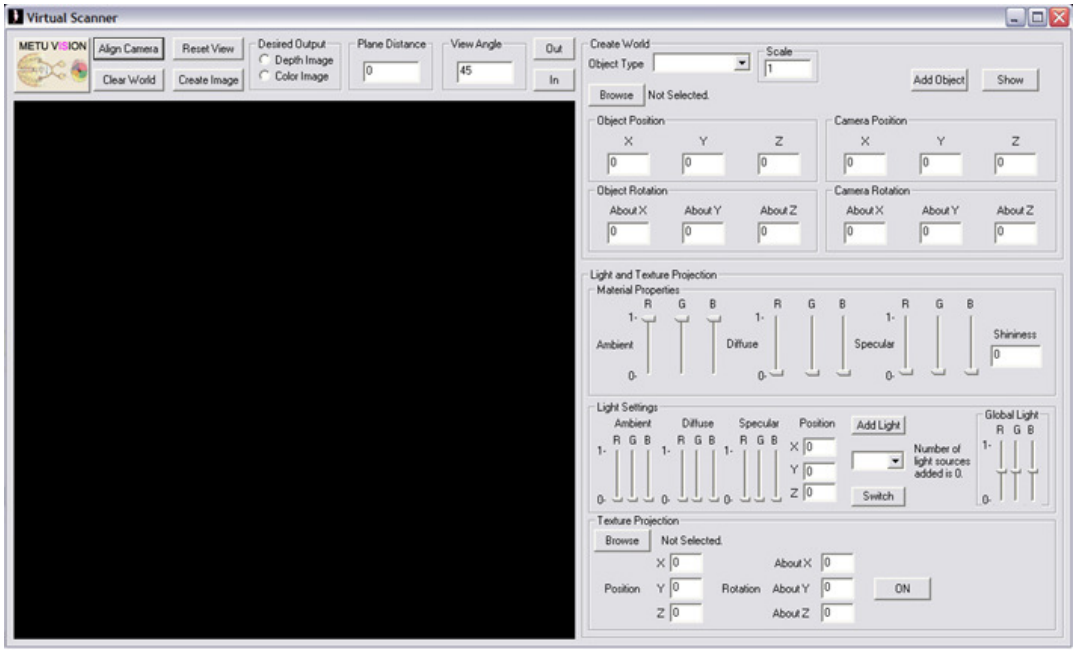

Figure 17 2D/3D Imaging Simulator graphical user interface

The tool has a simple and easy-to-use graphical user interface that allows the user to see the world that he created from different views via the view screen. Through this interface, the user can see and change the settings for the virtual environment.

The detailed information and explanations about all parts of the tool will be given in the following subsections. Also, the user's manual can be found in Appendix A.

2.3.1 Object Settings

The objects are the main components of the world that is created in 2D/3D Imaging Simulator. The user can load as many objects as desired provided that the object data have the compatible format with the programmed parser. The data format is very simple and yet adequate for satisfying results, and can be illustrated as the following:

v x_1 y_1 z_1

v x_2 y_2 z_2

...

f i_{11} i_{12} i_{13}

f i_{21} i_{22} i_{23}

...

where the vectors at the beginning, indicated with 'v's are the vertices that constitute the object itself. On the other hand, the vectors that are indicated with 'f's are actually the indices that tell us which vertices the faces are composed of. This data is enough to define the surfaces that form an object. The files should be named with the extension ".obj" after they are created.

For each object to be loaded, the user should enter the position (x, y, z) and the rotation around these axes. The corresponding fields in the GUI are necessary to fill in and their initial values are all zero.

After the .obj files are ready, they can be chosen to be loaded into the simulator by browsing to the necessary folder and selecting the file. Next, the user should select the object's type from the corresponding drop-list. The objects can be in two different types:

- With a matched bitmap image as their texture
- Without such a texture image

For objects with a matched texture image, the image file should be with the same name and in the same folder with the object file, and in this case, the object will be displayed with this image textured over its surface. On the other hand, the objects without a texture image will be displayed as gray objects with modifiable surface properties. This feature can be accessed in "material properties" section of the GUI. But, before switching to that part, the "matched texture image" concept should be emphasized:

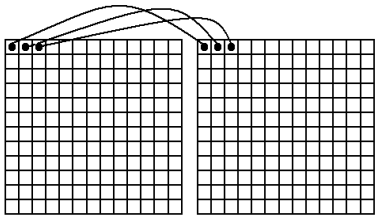

Figure 18 Matched image illustration

If we think that the objects are represented in 2.5D data, the surface is defined as a regular mesh with depth values at each entry, namely as a depth image. Its matched texture image is the (R, G, B) image, having the same size with the depth image and storing the color data at each vertex.

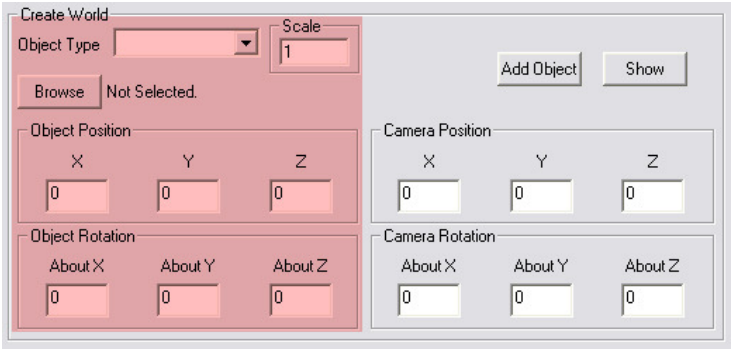

Figure 19 Object settings are highlighted in the GUI

After the required values are entered, “Add” button adds the object into the virtual world at the stated location, with the stated rotation and scale. This procedure can be repeated multiple times, before hitting the “Show” button which finalizes the world and displays it in the “View Screen”. After this step, the material properties can be adjusted unless the object loaded is with a matched texture file. In this case, user has the ability to change the surface reaction parameters to the light. Those parameters can be listed as:

- Diffuse (R, G, B)
- Ambient (R, G, B)
- Specular (R, G, B)
- Shininess

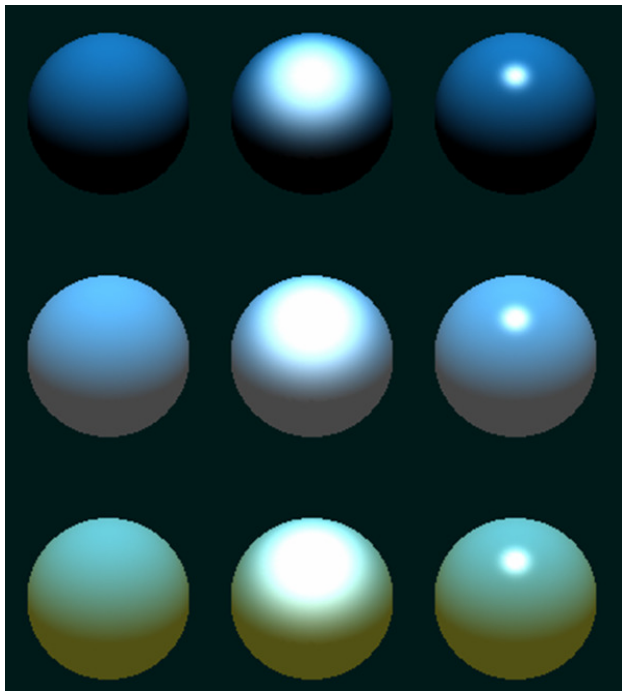

Figure 20 Objects with different material properties, illuminated by identical light sources

A material can have separate diffuse, ambient and specular colors. These (R, G, B) values define the material’s response to diffuse, ambient and specular light, respectively. The values are fixed to common face values for the objects with matched texture image.

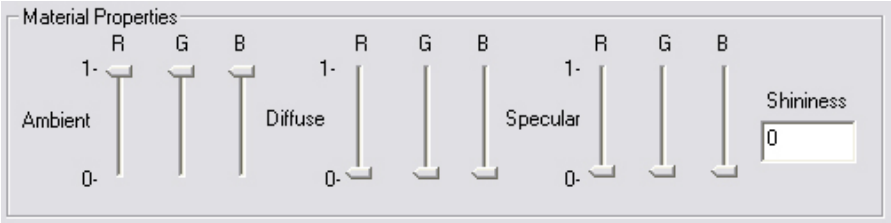

Figure 21 Material Properties section in the GUI

Before finalizing the world, one more thing to complete is to determine the camera properties which will be explained in the next section.

2.3.2 Camera Settings

Within the tool, camera actually refers to a point in space where the image or scan is taken. To reset the world and start over, "Clear World" button erases all objects loaded. For this reason, camera settings are rather simple. Similar to objects, camera has position and rotation parameters which determine the camera location and direction. The camera is designed as a pinhole camera, with a viewing angle and a focal length, which is referred as "image plane distance".

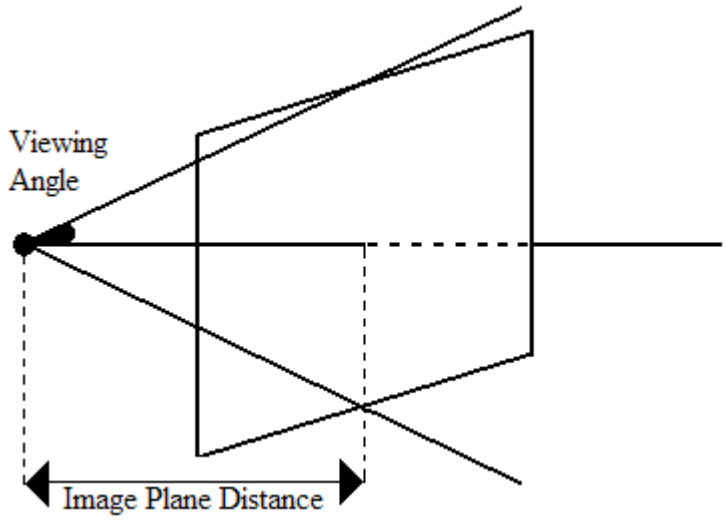

Figure 22 Camera design and parameters

Those two properties can be adjusted using the corresponding fields in the GUI.

In the “View Screen” the view can be changed in many ways and the output data, either image or scan, can be taken as if the world were photographed in that direction. The “camera” aspect in the tool is for more precise data acquisition. With the “Align Camera” button, the view is aligned to the position and direction of the camera, so that the user would know the exact setup.

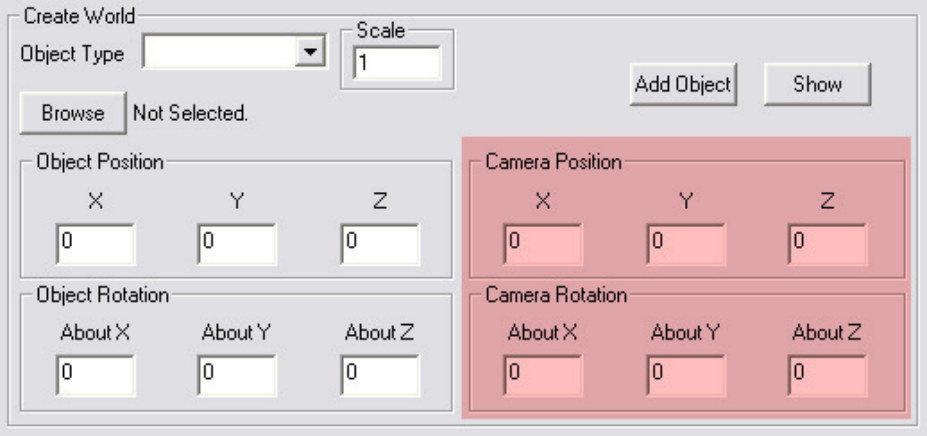

Figure 23 Camera settings are highlighted in the GUI

“Reset View” button, on the other hand, resets the view to the initial configuration, which is positioned at the origin, and directed towards “+z”.

2.3.3 Lighting Settings

When an environment is created and simulated in the 2D/3D Imaging Simulator tool, the world is initially illuminated by a global ambient light by default. This property makes it possible to see the objects when they are added, without any adjustments required.

The global light is ambient, which means it reflects with equal intensity in all directions. This implies that it does not reveal any information about the shape of the objects, since the reflection does not depend on the surface orientation. The intensity and the color of this light can be changed via the user interface.

Apart from the global light, eight other light sources can be added to the environment. These lights are not directional but point sources, meaning that they emit light on all directions. Hence the only spatial property to enter is the position of the light source.

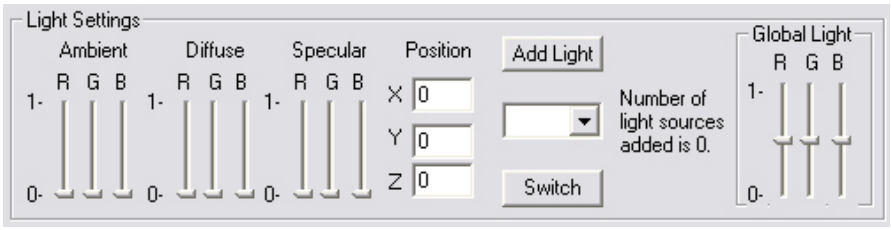

Figure 24 Light Properties section in the GUI

Similar to the material properties, the light properties can also be stated as “ambient”, “diffuse” and “specular” R, G, B values. Unlike the ambient light, the diffuse light reflection from the surface is related to the angle between the surface normal and the incidence angle. On the other hand, specular lighting is reflected more in the manner of a mirror where most of the light bounces off in a particular direction defined by the surface shape. Specular lighting is what produces the shiny highlights and helps to distinguish between flat, dull surfaces such as plaster and shiny surfaces like polished plastics and metals.

Those settings can be configured before adding the light as well as after adding all or some other lights and selecting the desired light source from the drop-list. This selection also enables the user to switch it on/off.

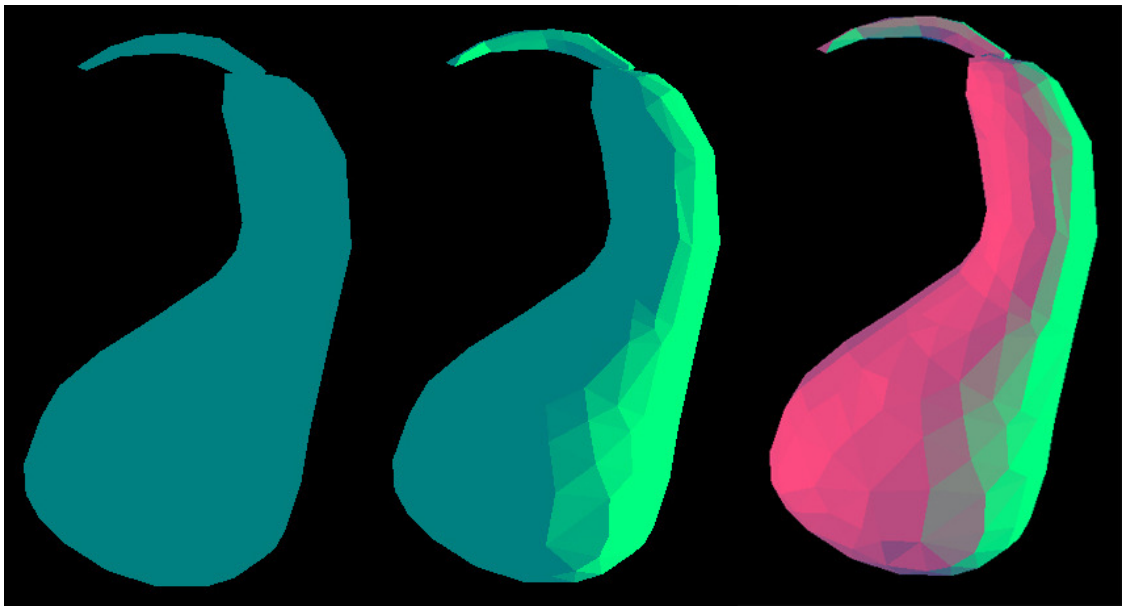

Figure 25 The first object is just with ambient light, second one is with the ambient light and a diffuse light positioned to its right and third one is with ambient light and two diffuse lights with different colors and positions

2.3.4 Texture Projection Settings

In the 2d/3D Imaging simulator, there is also the projection simulation, which is designed to realize structured light experiments to extract the shape information. Just like the camera, projector also has a position and a direction.

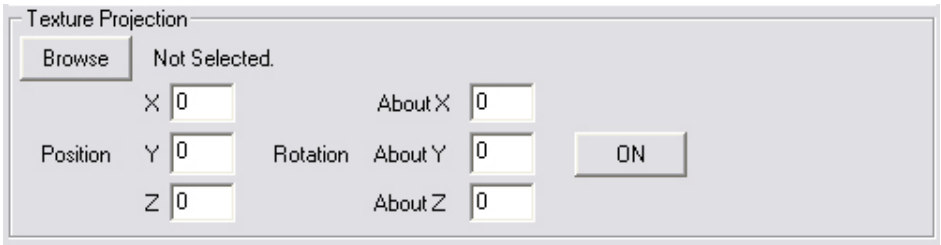

Figure 26 Texture Projection Properties section in the GUI

The texture to be projected on to the object is assigned by browsing the folders and selecting the image file. The important requirements for image is to be square, its size to be a power of two and to have an at least one-pixel-width frame of color (150,150,150), stated in (R,G,B) format. These limitations are imposed by the OpenGL's texturing tools.

The texture projection can be switched on and refreshed after changing the position and rotation values by using the "ON" button in the user interface, in the Texture Projection section.

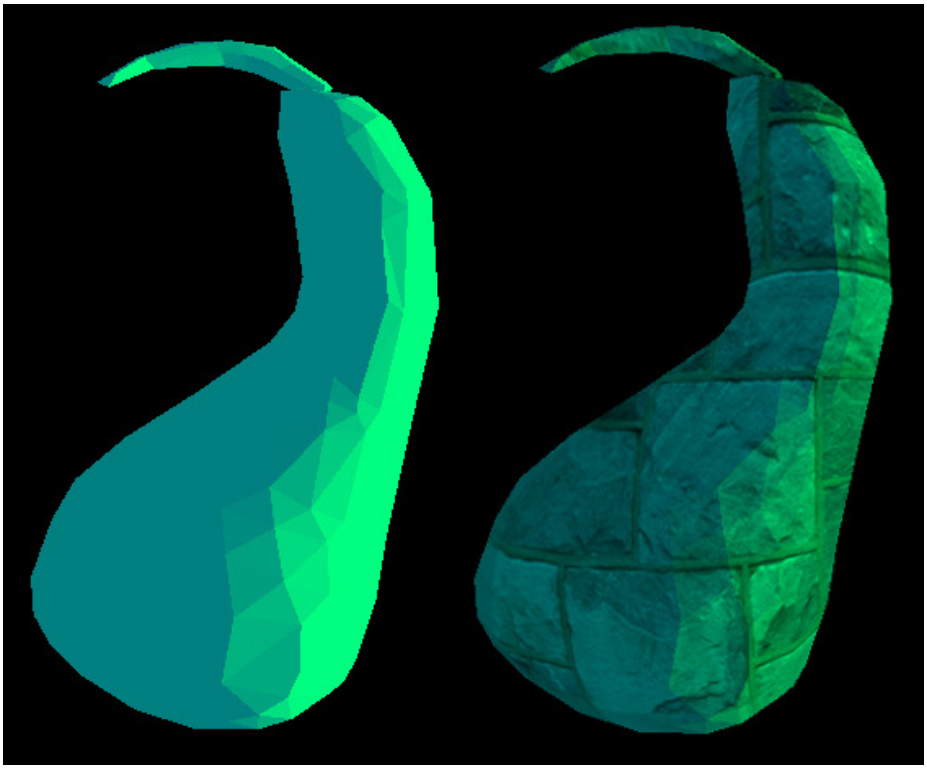

Figure 27 Objects without and with the texture projection

2.3.5 Outputs

The aim of this tool is to generate synthetic data for testing and/or application procedures. The output date can be in two types:

- Depth data: Scanner output
- Color date: Camera output

Figure 28 Output settings are highlighted in the GUI

After pressing the "Create Image" button, the tool generates the image data which is taken from the camera position, towards camera direction, in chosen type.

2.3.6 View Screen

The "View Screen" is the main part of the GUI, where the world created is displayed and the looking angles and positions are manipulated. The coordinate system used in the virtual environment is as shown in the following:

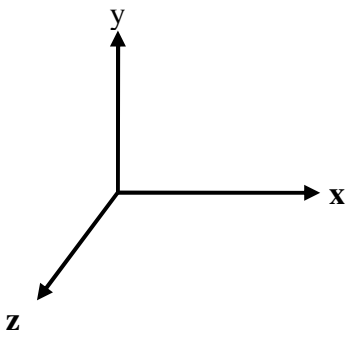

Figure 29 The coordinate system used in the View Screen

All the inputs for positions, rotations and directions are taken into account according to this configuration.

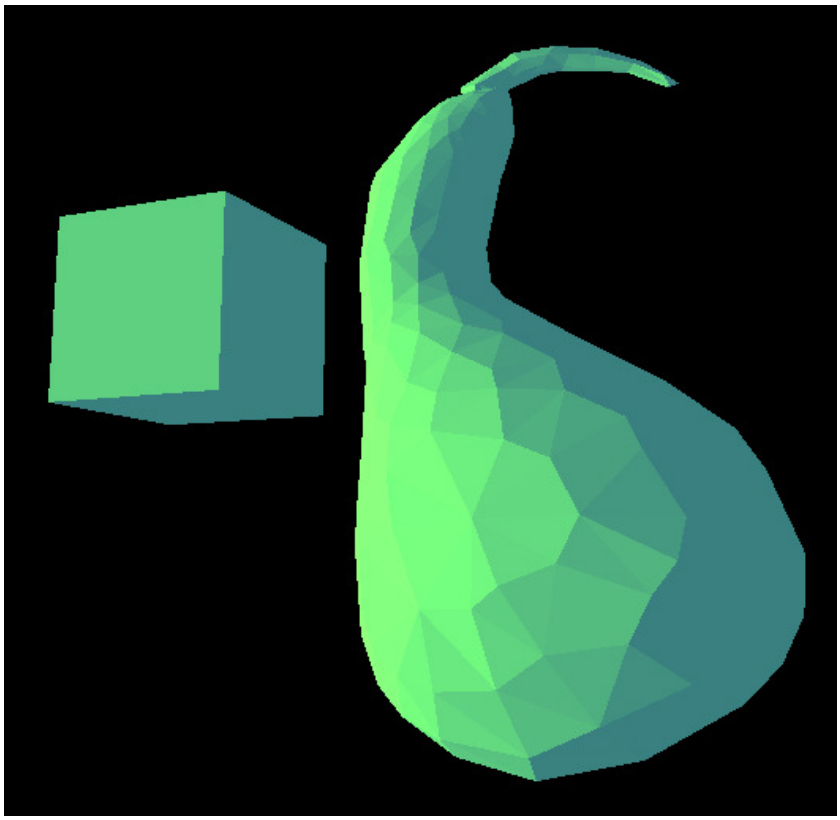

Figure 30 An example snapshot from the "View Screen" in the GUI

In/Out buttons help user to zoom into and zoom out from the scene, whereas holding left-button down on the screen allows user to rotate the objects.

2.4 Technical Details

In this section, technical details on the development environment, the coding structure and the work flow of the 2D/3D Imaging Simulator tool will be explained.

In computer program and software product development, the development environment is the set of processes and programming tools used to create the program or software product. During the development of this tool, its object-oriented software is implemented in C++, using Microsoft Visual Studio 6.0.

For the 3D visualization Open Graphics Library (OpenGL) is utilized. OpenGL is a standard specification defining a cross-language cross-platform application programming interface for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in computer-aided design, virtual reality, scientific visualization, information visualization, and flight simulation. OpenGL is managed by the non-profit technology consortium, the Khronos Group, Inc. [55]

Finally, the graphical user interface is designed by using the Microsoft Foundation Class Library (MFC), which is a library that wraps portions of the Windows API in C++ classes. The classes are defined for many of the handle-managed Windows objects and also for predefined windows and common controls. [56] Those objects and controls are organized into the tool's user interface for better convenience.

2.4.1 The Coding Structure

As mentioned before, the tool software is developed in object-oriented structure with C++. Except for some auxiliary code fractions, all parts are implemented as classes. Those classes can be listed and explained as:

- **CVirtualScannerDlg:** This class is generated automatically in order to manipulate the objects and controls placed in the graphical user interface. All the methods of this class are the message handlers for MFC and are implemented according to the functions to be installed. It acts as an interface between the GUI and the main control class that is explained below.

- **OpenGLControl :** This is the main class that administrates and realizes all the main calculations and the applications. It creates and displays the virtual

world according to the user's settings. It also creates the output images, either color or depth.

- **TexProj:** This small class only enables the texture projection. It is invoked from the OpenGLControl class with the user inputs. It is originally implemented by Steve Butrimas, as a sample to show how a projected texture technique can be used to produce a light map. [58]

- **Matrix4x4:** This is another small auxiliary class for some matrix calculations such as: taking inverse, taking transpose and the basic mathematical operations, written by Kevin Harris.

- **CVirtualScannerApp:** CVirtualScannerApp is another automatically generated class for the GUI application, in which no additions or modification is made.

- **CFolderDialog:** In MFC, the browse operation is only done for files, hence this class is added to the workspace to enable folder navigation. It is implemented by Mihai Filimon and can be found as an open source code. [57]

Apart from the classes, a small source code for bitmap operations is used. It is written by John Burkardt in order to employ some routines to read a BMP file and extract and return the graphics information (RGB pixel arrays), or to write internal RGB pixel arrays into a properly formatted BMP file. [59]

The detailed information about the main methods and their role in the whole picture can be found in Appendix B. In the next section, the work flow for the 2D/3D Imaging Simulator tool will be explained to give some idea about the way of operations. Additionally, the class-to-class and user-to-class interfaces within the software will be given.

2.4.2 The Work Flow

Since the activation and the execution of the tool are triggered by the user, firstly the user-class interface will be explained.

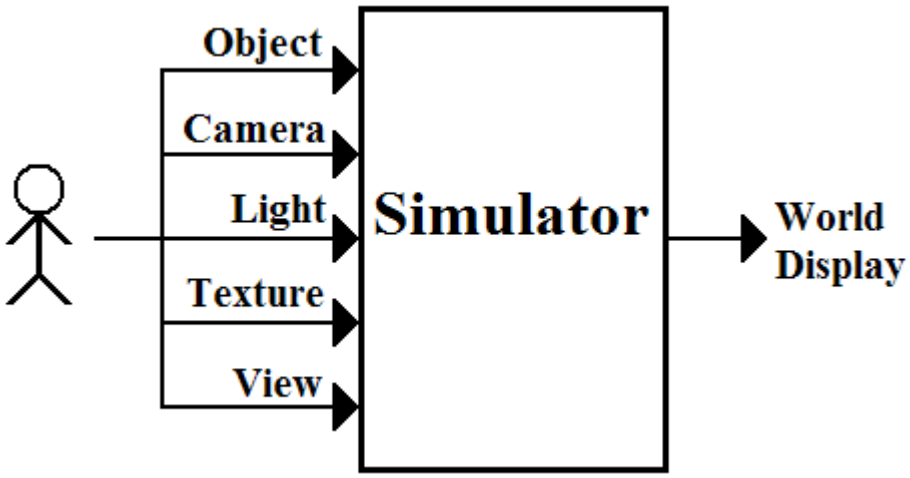

Figure 31 User inputs to the simulator

The tool does not display anything until the user enters the object and the camera settings and hits the “Show” button. However before this step, the user has to select objects to be loaded and specify their positions and rotations. Each time the “Add” button is pressed, the .obj files are read by the program and all the vertex and color information are stored in the memory with an appropriate data structure. After that, “Show” button switches on the display and by using the OpenGL functions the triangles composed of the vertices are drawn in the view screen.

The display functions in a loop by switching the buffers each time the loop is completed. This enables to refresh the image on the view screen and hence the user can change light, texture and viewing angle settings even after the world is finalized and displayed.

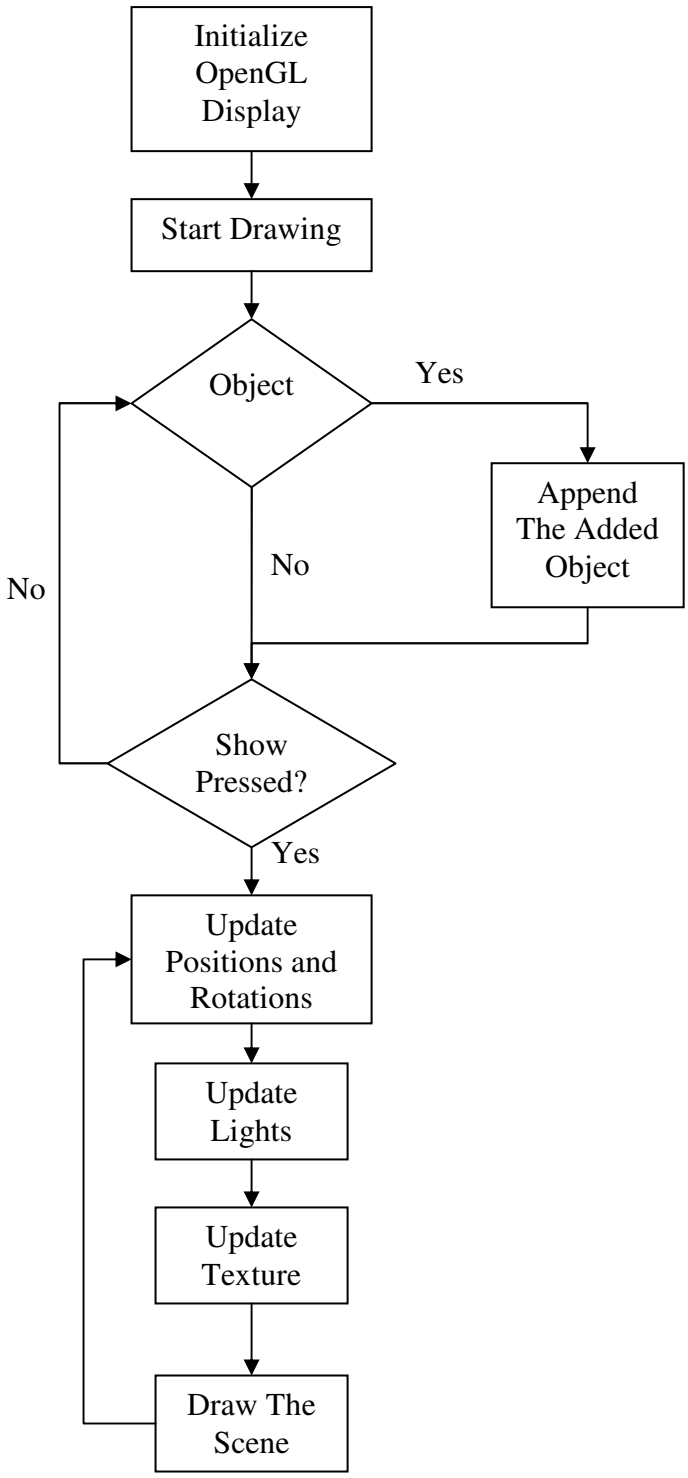

Figure 32 Work flow for the simulator display

For creating the output images, again OpenGL brings in a big convenience with the built-in depth and color buffers. For the color information, the displayed pixels'

green, red and blue components are read and written into a bitmap image. Similarly, for the depth information, the displayed pixels' depth values are read and written into a text file. The depth information is also recorded into a grayscale bitmap image for visualization.

CHAPTER 3

Example Application

In order to check the usability and the performance of the METU VISION 2D/3D Imaging Simulator, an example application has been conducted on FRGC database and the 3D data acquisition method which utilizes structured light and was mentioned in part 1.1.3.

In the following subsections, firstly detailed information will be given on the post-processing of the scan data, which is a crucial process in 3D data related research. Afterwards, in the consequent sections, the example application will be explained and results will be given.

FRGC Database

In both post-processing and structured light method application parts, FRGC database has been used. FRGC, Face Recognition Grand Challenge, was conducted to fulfill the promise of the new techniques in automatic face recognition, which were boosted by advances in computer vision techniques, computer design, sensor design, and interest in fielding face recognition systems, from May 2004 to March 2006. The design of the FRGC starts from performance in FRVT 2002, establishes a

performance goal that is an order of magnitude greater, and then designs a data corpus and challenge problem that supports meeting the FRGC performance goal [49].

The FRGC was designed to challenge researchers and advance the face recognition technologies [49]. FRGC ver1.0, which has been used in this thesis, was a small challenge problem designed to introduce researchers to the FRGC challenge problem protocol, procedures, and data formats. It includes 1126 images, consisting of 366 training images, 152 controlled gallery images, and 608 uncontrolled probe images.

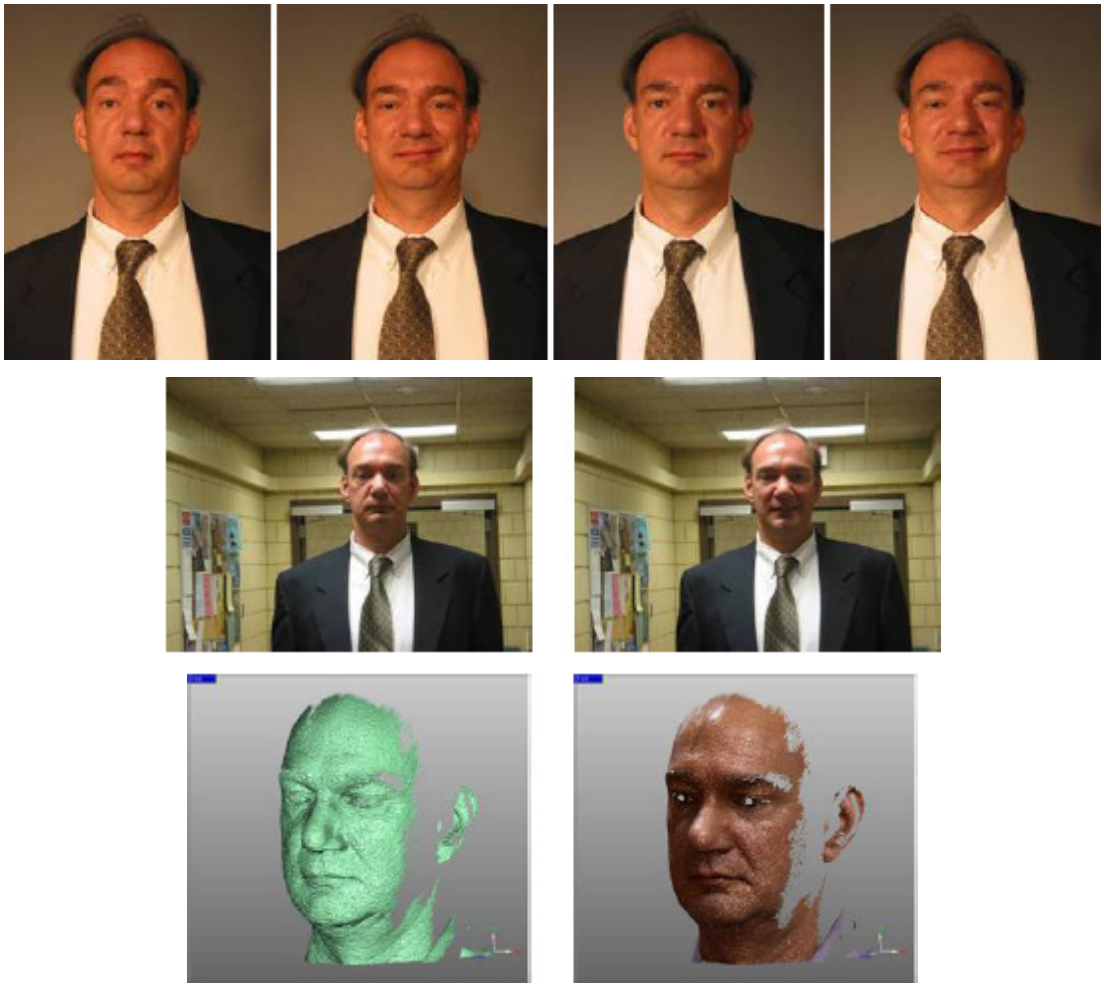

Figure 33 Images from one subject session. (a) Four controlled stills, (b) two uncontrolled stills, and (c) 3D shape channel and texture channel pasted on 3D shape channel [49].

The FRGC database was built at the University of Notre Dame. A “subject session” is the set of all images of a person taken each time a person’s biometric data is collected. For the FRGC data for a subject session consists of four controlled still images, two uncontrolled still images, and one three-dimensional image. Figure 24 shows a set of images for one subject session [49].

The controlled images were taken in a studio setting, are full frontal facial images taken under two lighting conditions (two or three studio lights) and with two facial expressions; smiling and neutral. The uncontrolled images were taken in varying illumination conditions; e.g., hallways, atria, or outdoors. Each set of uncontrolled images contains two expressions, smiling and neutral. The 3D images were taken under controlled illumination conditions and they consist of both range and texture channels [49].

The 3D data from FRGC v1.0 database is supplied in .abs form. In this file, first row and column numbers are given in two lines and the data format is stated in the next line. Finally, flag, x-, y- and z- values are written in four consecutive lines, respectively. Flag value specifies the validity of the data, “0” value implies that the pixel is invalid and all entries belonging to that pixel is “-999999.000000”.

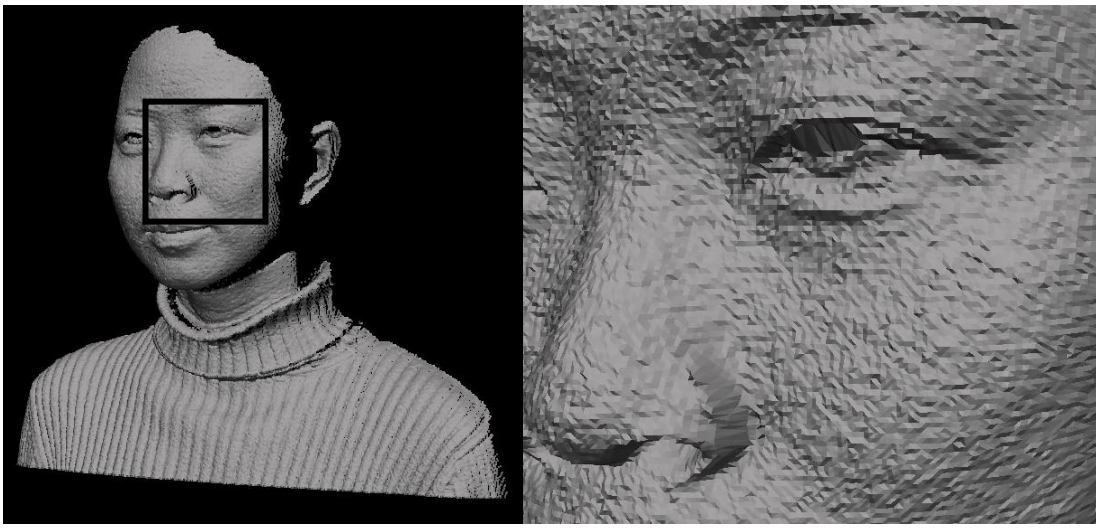

Figure 34 Original scan data and a detail

Along with the coordinates of the pixels also their colors are provided with a registered bitmap file.

The original data files are distorted with spikes and holes. The surfaces are also noisy due to the imaging system imperfectness. The models usually include shoulders, neck and the head as shown in Figure 25.

Preprocessing

The goal of the preprocessing is to obtain the optimum, the most convenient point cloud for 3D face application software which possibly serves for face recognition, morphing, modeling, animating etc. For this reason, it is important that while correcting the erroneous points, the new coordinates should be as close as possible to the real positions, the features should be preserved. Additionally, extraction of the face from the whole image yields to easier data storage and manipulation.

Face Region Extraction

Before all other corrections, the first step of preprocessing is to segment the facial region from the overall point cloud. Face region extraction is good for obtaining more manageable data sizes and hence helps the denoising part to be faster than otherwise. Additionally, in our case, face region is assumed to have the most valuable data within the model, since shoulders and neck can vary according to the clothes that are worn.

For face segmentation, the assumption that face is closer to the scanner than other scanned parts of the body is used. Firstly, a threshold is applied and facial and non-facial regions are separated from each other. Afterwards, similar to [50], by labeling the connected components, the facial component is located as the one which is positioned in the middle of the scan and has the largest connected area .

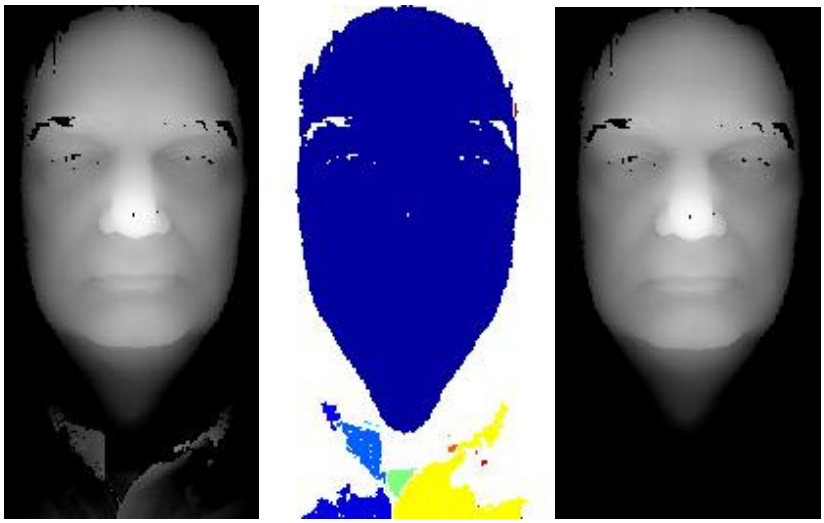

Figure 35 The facial region extraction

Spike Removal

First of all, the spikes should be detected. In order to decide whether a pixel value indicates a spike or not, the neighbor pixels are taken into account. The decision mechanism is rather simple. If the pixel value differs widely from more than the half of its valid neighbors in 5x5 kernels, it is treated as a spike and its value is changed to the mean of these neighboring pixels.

Here is a pseudo code for the process is given below:

For each pixel in the depth image

Calculate the difference between this pixel and each neighboring pixel in its 5x5 neighborhood

If the number of pixels with higher difference than the threshold is higher than the half of the number of the valid pixels in the 5x5 kernel

Set the value of the pixel to the mean of the neighboring pixels with high difference

End

End

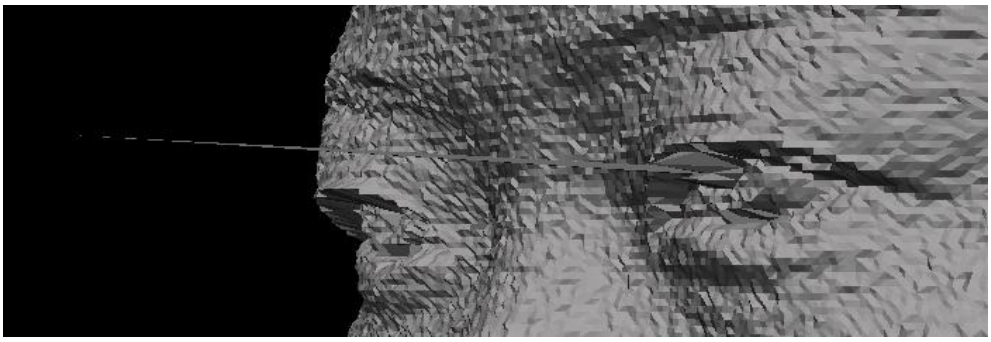

Figure 36 A spike from an original scan data

Hole Filling

Holes in a scan data are composed of invalid pixels. Every model in the database has holes due to the scattering in the facial surface, especially around pupils.

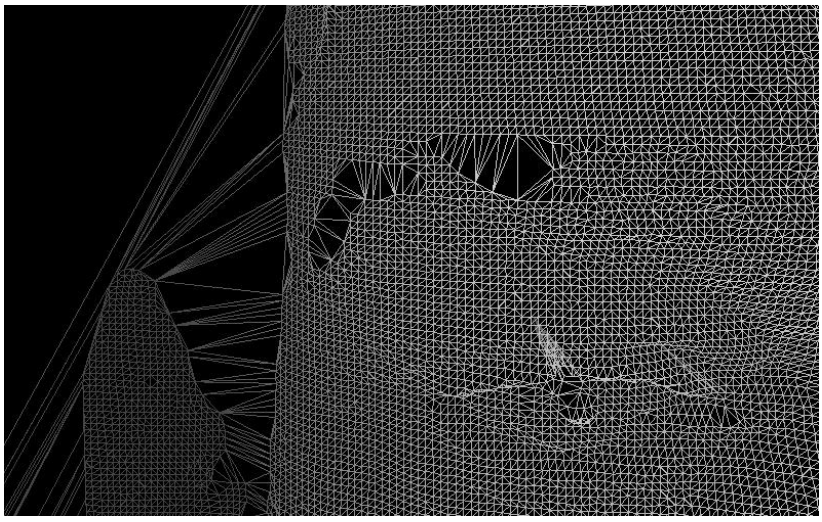

Figure 37 Holes can be seen around the eyebrows and the pupil in the example.

The invalid pixels within the face can also be treated as spikes in the negative direction. Hence, the same algorithm for the spikes works for the small holes as well.

However, some holes can be too large that they can not be fixed, since 5x5 kernels do not supply enough number of valid entries.

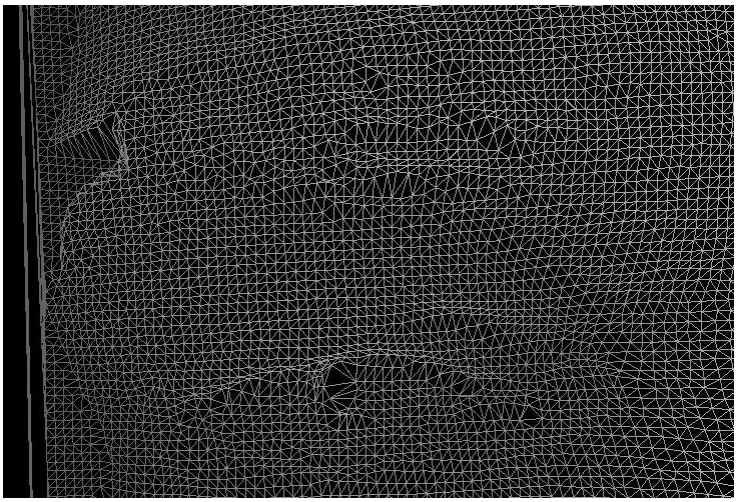

Figure 38 Holes around the eyebrows and the pupil in the example are filled.

In this case, face symmetry with respect to the vertical line passing through the point at the tip of the nose is utilized to assign depth values to the invalid pixels. After symmetry is applied, if some small holes still remain, the hole-filling algorithm is repeated once more. After the hole-filling is completed, the surface is continuous, yet noisy.

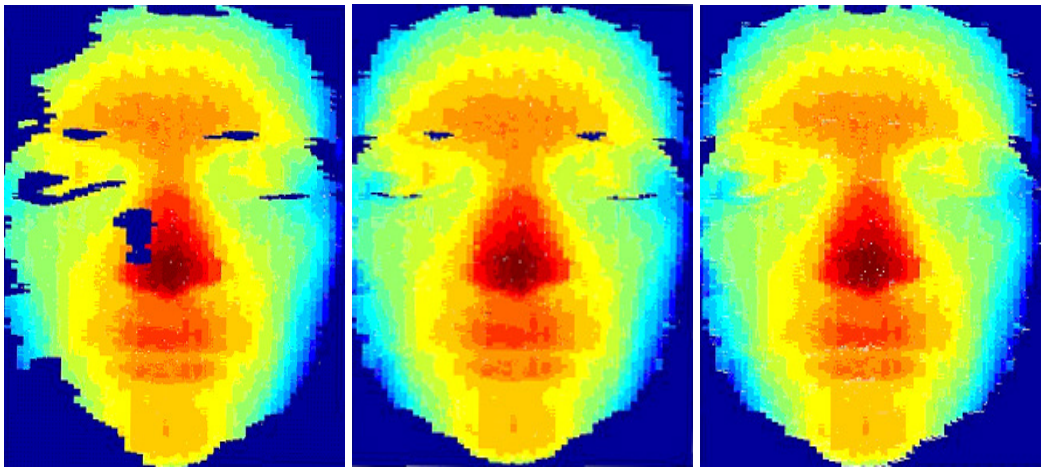

Figure 39 Results for the utilization of face symmetry and re-application of the hole filling method

In order to have a smooth surface which is in better accordance with actual face model, the noise should be filtered out.

Smoothing

In the final step, the input is the model of the face with the holes and spikes removed. After this point, the original depth data which is corrupted by noise is supposed to be processed in order to estimate the denoised point cloud. However, while computing the noise-free models, the main concern should be to preserve the features in the surface so that the shape effectiveness is improved for further applications.

Among numerous methods, *Bilateral Filtering* is proven to be a powerful yet simple, non-iterative scheme for edge-preserving smoothing in 2D. Additionally, this image processing technique has also been successfully extended to 3D by making small modifications in the application.

Bilateral filter is distinguished from Gaussian blur by its additional “range” term. In this case, not only the distances between positions matter, but also the variation of the intensities are also taken into account in order to keep the features from derogation. Consequently, the edges where high intensity differences occur are successfully preserved.

In 2D, the intensity values are a function of position values, whereas in 3D the position in fact, is the signal itself. Hence, the modification of bilateral filter to be applied to 3D data is not very straightforward. Like the most image processing algorithms that are extended to surfaces, normal information at each point of the surface can be used to form an intensity space like in images.

Several such extensions have also been proposed for 3D application. The bilateral filter is extended to polygonal meshes without connectivity in [51], to mostly connected meshes, using an iterative approach in [52]. Additionally, in [53] the bilateral filter for images was modified to use a type of first-order predictor, then to generalize this extension to meshes.

In our case, since the points are regularly located on a grid, instead of dealing with neighboring surfaces, square kernels of adjustable size are applied in which the range weights are calculated by using normal values at that points.

The proposed bilateral filter for 3D is as follows:

$$P_e = BF[P] = \frac{1}{W_p} \sum_{S \in V} G_{\sigma_s}(\|P - S\|)\, F_{\sigma_r}(|N_p - N_s|)\, S \qquad \textbf{Equation 3.1}$$

where P_e is the point to be estimated and S's are the points in the voxel V, which are going to contribute to the calculation of denoised position of P. $G\sigma_s$ is the spatial weight function, which is taken as a one-dimensional Gaussian filter. Hence, the weight decreases with the spatial distance to P. $F\sigma_r$ is the range weight function, which is again a one dimensional Gaussian function that decreases the influence of pixels S with a range value, N_s, much different from N_p. This means, if a point in V, is very distant to the point to be denoised the drastic decline in $G\sigma_s$ will attenuate the weight of that point. Similarly, the difference between the normal value of a point in V and the point to be denoised is high; the falloff in $F\sigma_r$ will make that point less effective on the estimation. Without the range weight function, this filter simply applies a Gaussian blur. The range function is what provides us with the feature preserving behavior. Finally, W_p is the normalization factor calculated by adding all weights for each point:

$$W_p = \sum_{S \in V} G_{\sigma_s}(\|P - S\|)\, F_{\sigma_r}(|N_p - N_s|) \qquad \textbf{Equation 3.2}$$

Two important parameters in the estimation are σ_s and σ_r, which mainly specify their corresponding weights. σ_s and σ_r are the standard deviations in space and intensity (range in 3D case) difference, respectively. In the calculations, σ_s is taken

proportional to the model size, in other words it is taken to be 2% of the largest distance that is between the points on each end of the diagonal. On the other hand, σ_r is taken proportional to normal variations throughout the point cloud. That is, it is calculated by taking the average of the values in the gradient of the normal image.
To be able to utilize the first order information, normal values at each point should be calculated. Occasionally, this task is not very simple, for instance, when there is insufficient connectivity among the polygons or when the samples are consist of point clouds without any other information. However, in our case, since the point cloud is given a regular mesh, normal calculations are trivial.

With the help of the following example, the effect of the bilateral filter will be understood and observed more easily. Firstly, a synthetic surface with an edge is generated as the test subject.

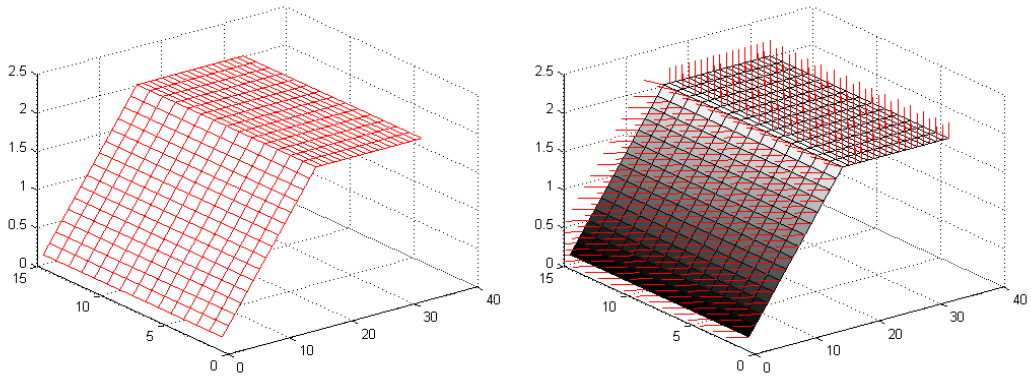

Figure 40 Synthetically generated surface with an edge

Afterwards, a random noise is added to this surface to simulate the noisy scan data. The noise range is taken to be 5% of the total range variation within the surface which is approximately the same proportion when compared to 3D scan data.

In the figure below, the noisy surface and the normals at each point is shown. As you can notice, the major change in the normals is observed around the edge. This characteristic is what constitutes the essence of bilateral filtering.

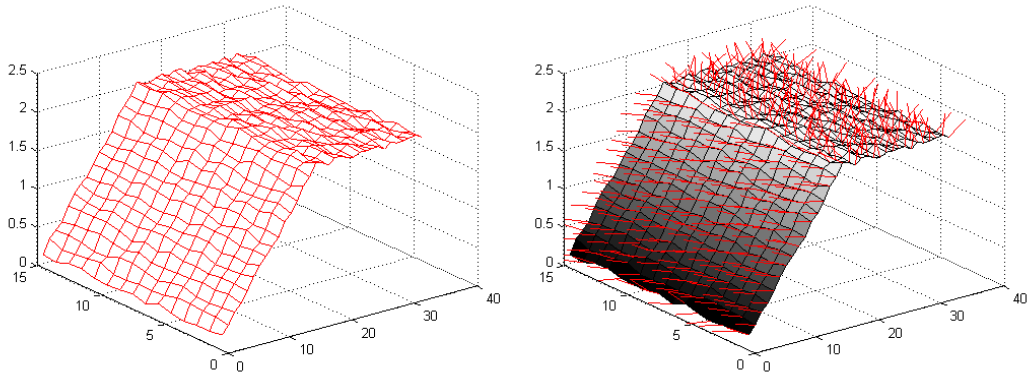

Figure 41 Artificially created noisy surface

Firstly, the noisy surface is smoothed by the bilateral filter. It can be observed in the figure below that the points that are close to the edge are not affected by the steep downfall, contrary to Gaussian filter which is also applied to the noisy surface and shown in Figure 35.

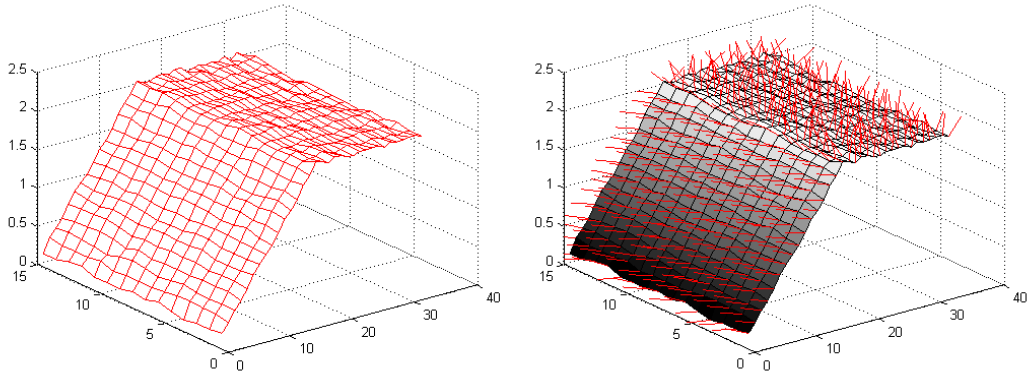

Figure 42 The denoised surface after applying the bilateral filter once

Since the surface normals are getting closer to their perfect estimates, they are aligned and the differences between normals of the points on the same surface reduce and between the normals of the points on different surfaces increase. Hence, consecutive applications of this filter yields to better estimation. The result after the second and third applications of the filter is shown in the next figure.

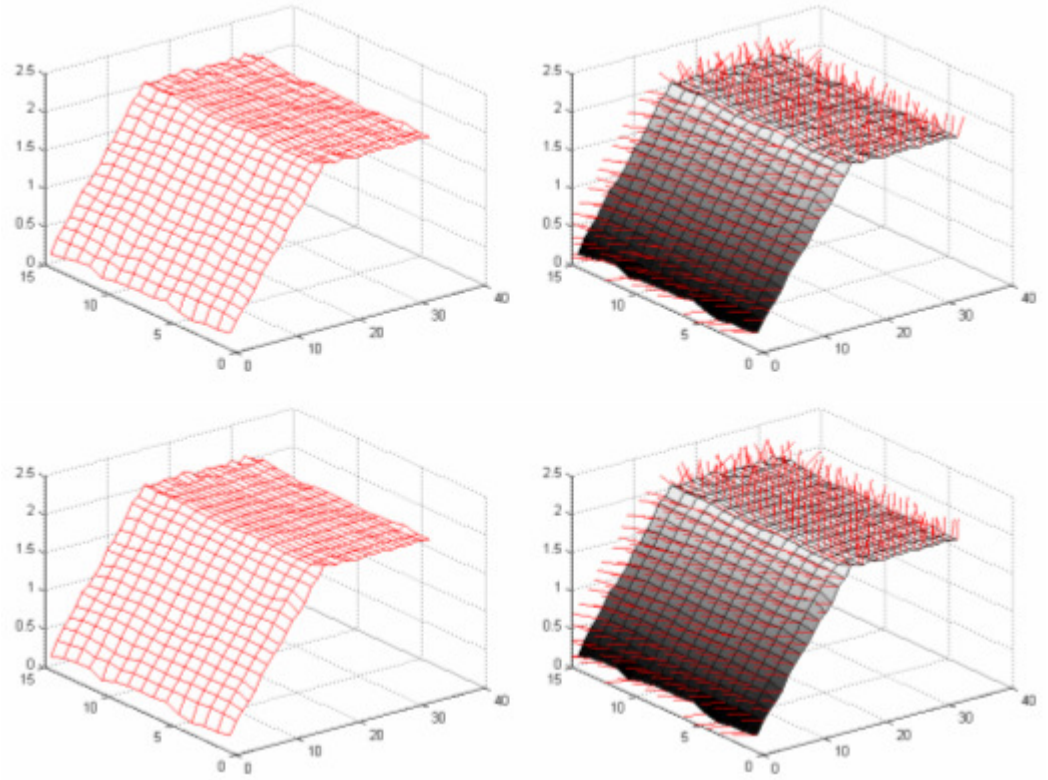

Figure 43 The denoised surface after applying the bilateral filter two and three times

On the other hand, applying the Gaussian filter only with the same standard deviation to the noisy surface yields to smoother surfaces but graded corners. Successive application of this filter does not fit for the purpose of corner conservation.

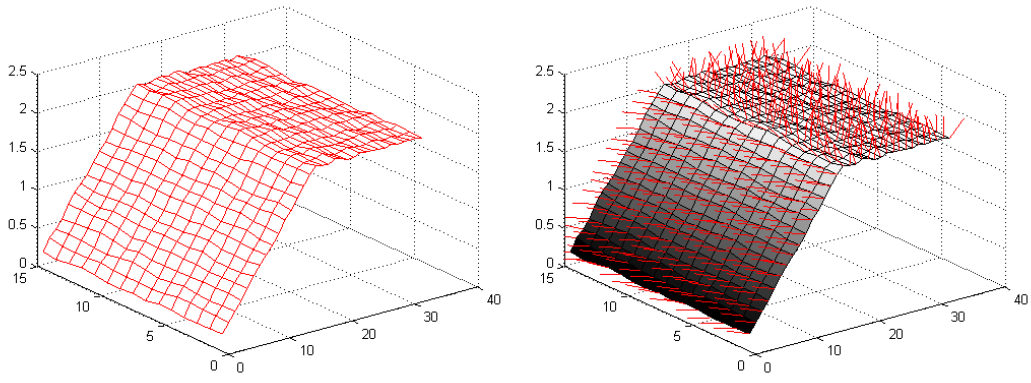

Figure 44 The denoised surface after applying the Gaussian first time

Although the first application of the Gaussian filter does not seem to affect the cornerness too much, it is obviously needed to be reapplied to reach the same level of surface smoothness. Unfortunately, at this point, the advantage to preserve the shape sharpness is lost.

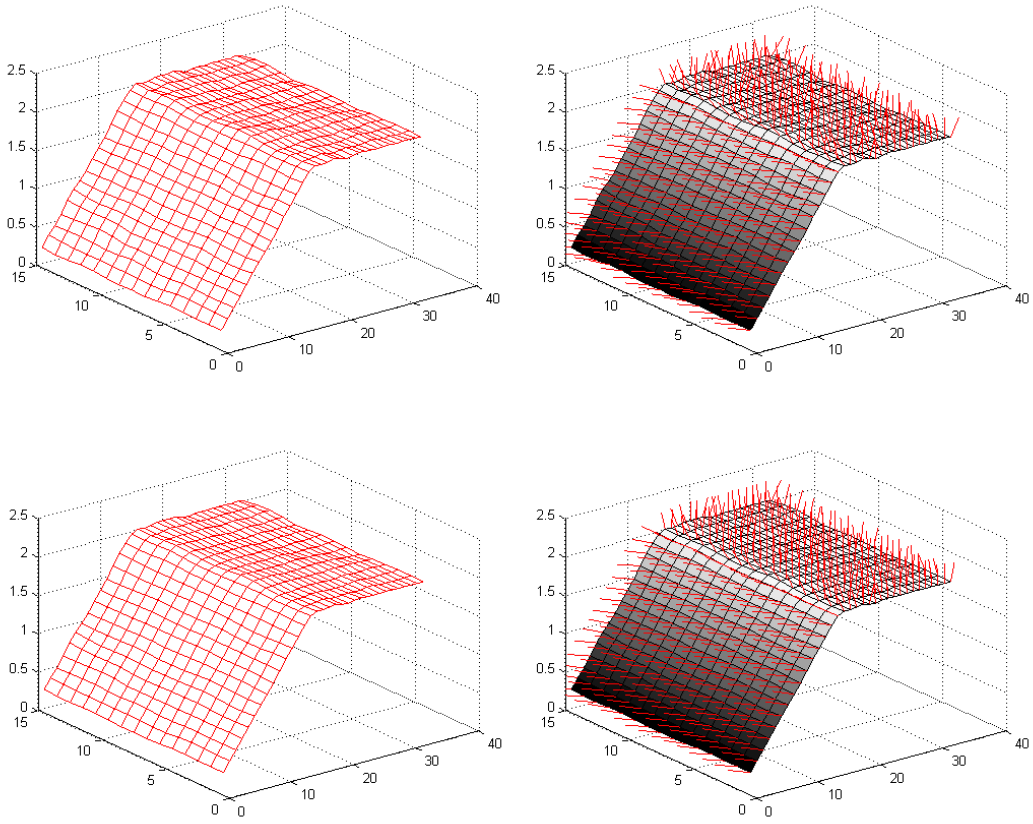

Figure 45 The denoised surface after applying the Gaussian filter two and three times

The Gaussian filter may seem to smooth the surface much more effectively than the bilateral filter, but this disadvantage of the edge-preserving smoothing can be countered by adjusting the weights to be more appropriate for the noise characteristic in the models.

The difference between two filtering effects in edge-preserving sense can be seen more clearly in the following figure.

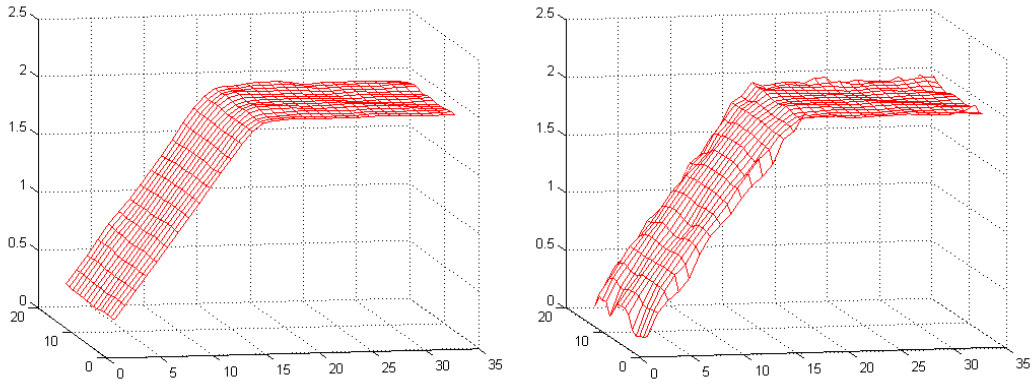

Figure 46 Surfaces denoised with Gaussian and Bilateral filters, respectively.

The results of the bilateral filtering on the FRGC images are very satisfactory. The noisy scan data, and the same data after smoothing is given below for better comparison. The edge details around the eye is well preserved whereas the facial surface is cleared off the noise.

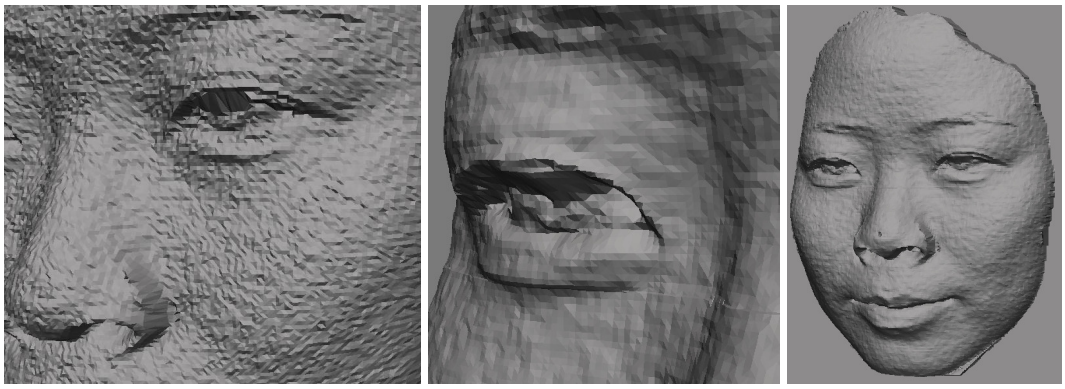

Figure 47 The result of the bilateral filtering on the FRGC data

The preprocessing on the depth data is completed after the denoising. As a result, smooth, hole and spike free face model with the edges preserved, is obtained. This model will be used to test the structured light method implemented by Ahmet Oğuz Öztürk as his master's thesis in November, 2007 [54].

Application of Stereo Images and Structured Light Method

In this section, an example application of the 2D/3D Imaging Tool will be explained in detail. Briefly, a method for 3D data acquisition that combines stereo acquisition and structured light technique will be tested.

As mentioned in the previous chapters, multiple cameras are used for capturing stereo images which can be used to build a depth map using the disparities between the interest points. Those points are not easy to detect correctly in the stereo image set, hence structured light projection is utilized.

In the specific method that is applied here, the structured light is a projected texture onto the face that is composed of horizontal lines. After the images are obtained via our simulator, the tool, which is implemented by Ahmet Oğuz Öztürk as a part his master's thesis is used to derive the 3D shape.

Stereo Image Acquisition with a Projected Texture

To start with a 3D model is obtained from the FRGC database to work with. This model is chosen to be the one that has been given as an example in the preprocessing part.

As discussed before, using the 2D/3D Imaging Simulator, the 3D objects can be freely positioned and moved around in the virtual world, as well as the camera and the projector. For this application, the object is preferred to change position with respect to the camera for the sake of convenience. The camera is kept at the origin whereas the projector is placed according to the object. The steps of the stereo image acquisition with a projected texture are as follows:

First of all, the 3D model that is obtained from the FRGC database and preprocessed as explained previously is turned into an obj. file and made compatible to load into the simulator. After the preprocessing, the corresponding .obj file is created with the name “04202d356_oa.obj”.

In order to obtain the stereo images, firstly the 3D model is added into the virtual world in the 2D/3D Imaging Simulator at the point (-0.5, 0, -5). The camera position is set as the origin and finally the projector is placed at (-0.5, 0, 0) with a direction towards (-0.5, 0, -5). This way the image that is simulated as taken form the right camera can be obtained by selecting the “Color Image” option and hitting the “Create Image” button. The same process is repeated by placing the object at (0.5, 0, -5) and the projector positioned at (0.5, 0, 0) with a direction towards (0.5, 0, -5). Consequently, the same scene is simulated to be shot by the left camera.

The textures to be projected onto the face are created according to the needs of the simulator that were mentioned previously. The horizontal one-pixel-thick white lines on a gray (150,150,150) background are positioned with three pixel distance in between. In four texture images, the lines are shifting one pixel towards right to increase the resolution of the scanner.

Additionally, in all texture images a point is marked with red in order to be set as the initial point while using the scanner tool. This point is adjusted to be on the forehead, where the transverse width has its maximum value.

The process of obtaining right and left images is repeated with four different textures and without a texture. After creating each image, the bitmap file is renamed as “<name>Right<n>” where <name> is the name of the subject, and <n> is the image set number. Finally, all images are moved in to a folder named “Outputs” and this folder is copied into the folder where the scanner executable is.

The resultant images and the setup of the simulator can be seen in the following figure and the table, respectively.

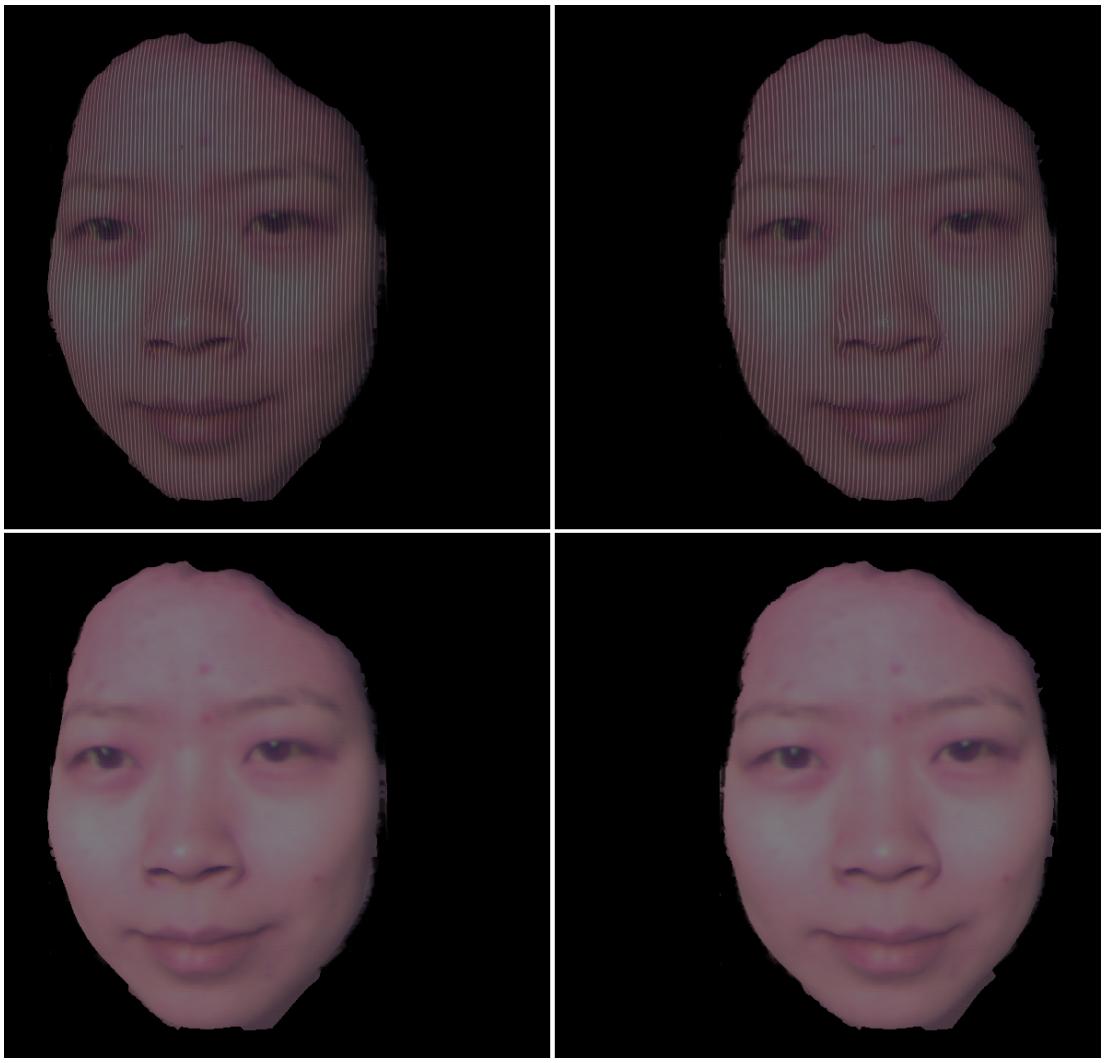

Figure 48 Right and left images with/without texture projection

Values	X (R/L)	Y (R/L)	Z (R/L)
Camera Position	0.0	0.0	0.0
Object Position	-0.5/0.5	0.0/0.0	-5.0/-5.0
Projector Position	-0.5/0.5	0.0/0.0	0.0/0.0
Projector Direction	-0.5/0.5	0.0/0.0	-5.0/-5.0

Table 1 The setup of the simulator

3D Shape Derivation

After the “Outputs” folder is ready with 10 images from right and left cameras, the “3D Face Scanner” [54] tool is started by executing the “3DFaceScanner.exe” file. The graphical user interface of the tool is simple and friendly. The tool is actually designed to capture images in real-time and create the 3D model successively. Additionally, the user can also load the previously captured image to rescan the object later. This feature of the tool is utilized within this section.

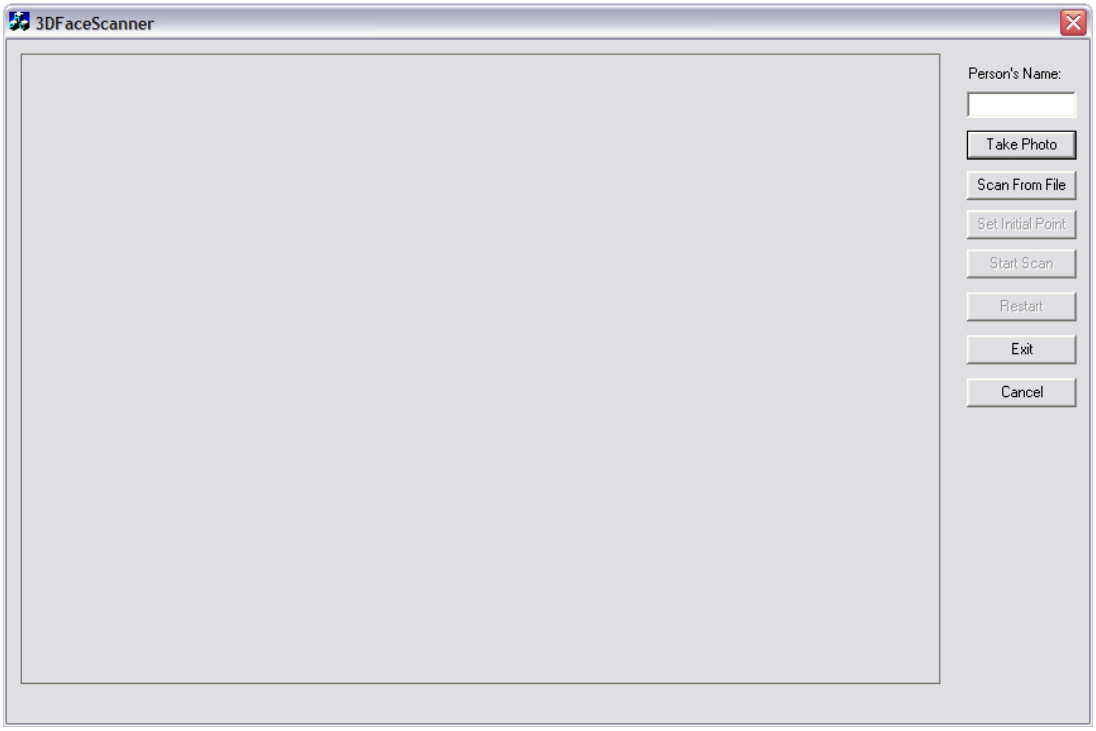

Figure 49 Graphical user interface of the 3D Face Scanner

The “Person’s Name” field is filled with the name chosen for the subject while saving the images created by the 2D/3D Imaging Simulator tool. After pressing the “Scan From file” button, the 10 images in the “Outputs” folder will be displayed in order and the user is expected to mark the initial points in each image with a texture projection. After marking the initial point by left-clicking on the image, “Set Initial Point” button is pressed and the next picture is displayed.

After all images are displayed, “Start Scan” button is enabled and by clicking on it, the scan process can be started. The scanning takes approximately 10 seconds and the “Exit” button is enabled afterwards.

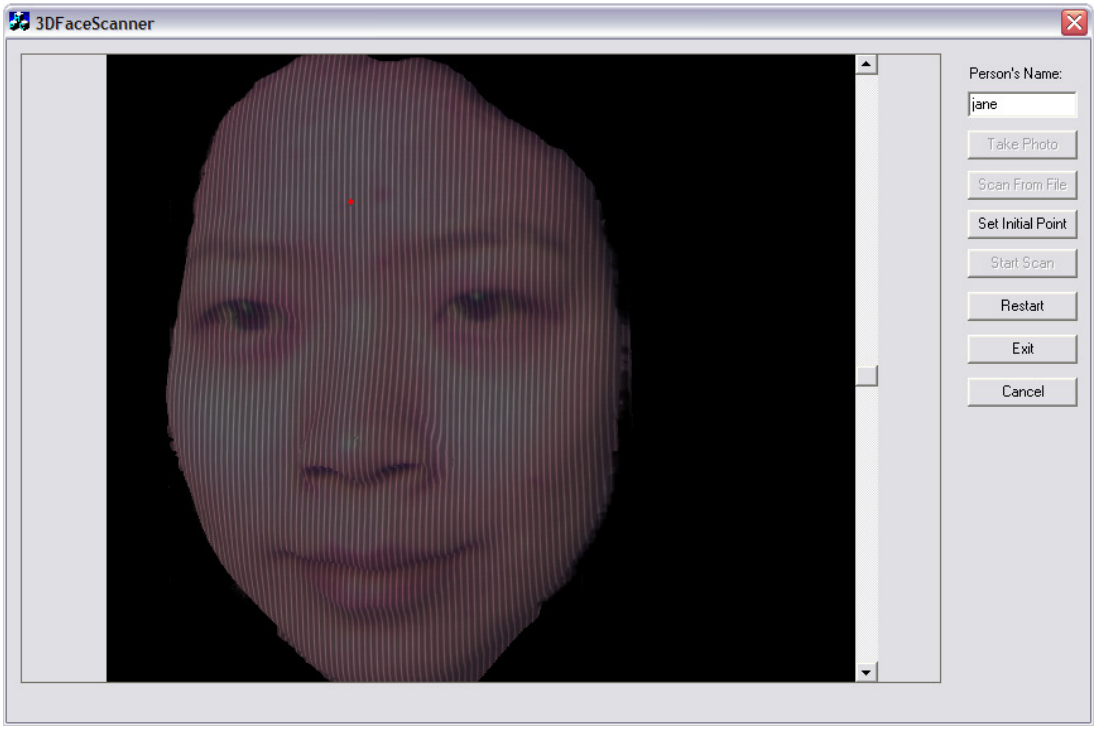

Figure 50 Loading images from file and setting the initial points

After exiting the 3D Face Scanner, a new folder with the name of the subject is created with shape data saved inside. However, these files are just the intermediate products of the process. The 3D data is to be post-processed and the corresponding .obj file is to be created. For this purpose, the “<name>.m” file is executed and finally after this execution the .obj file is obtained.

In the “Outputs” folder, new images are observed to be created in which processing steps of the 3D Face Scanner can easily be seen. As mentioned before, the line information in the picture of the face is exploited to create a sparse but regular depth map.

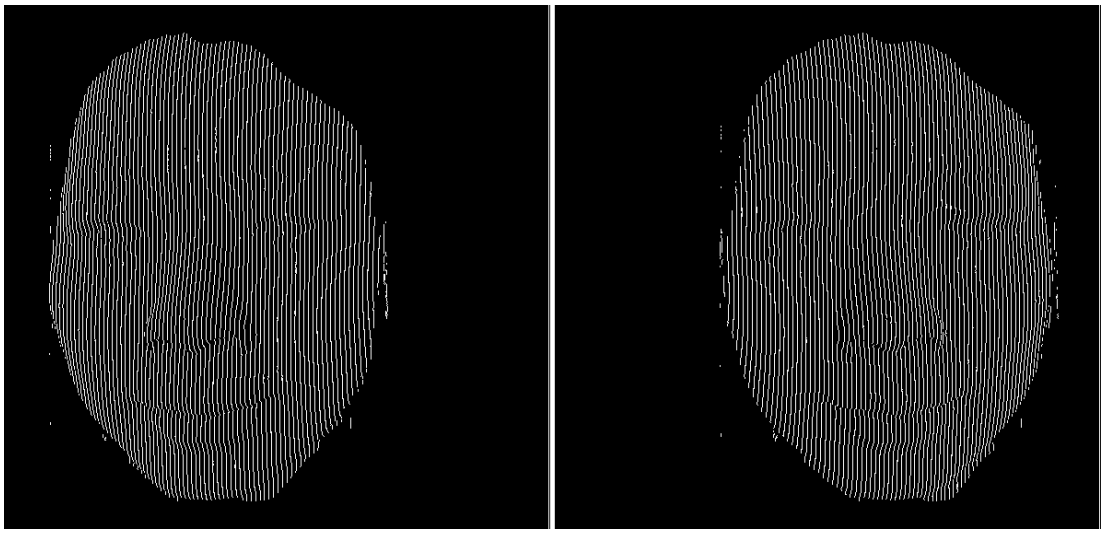

Figure 51 Right and left image samples after the lines are extracted

The output of the 3D Face Scanner can be viewed using numerous obj-viewers. In the next section, the 3D model obtained is given. However, visual observation is not enough to measure the success of a scanner. Hence, a comparison between the resultant and the actual models should be made.

Comparison of the Result with the Actual Shape

The .m file that is executed after the scanning is completed creates the .obj file as the last product. This program reads the text files created by the scanner and uses them to generate the .obj file where the 3D coordinates of the vertices are recorded along with the normal vector and texture coordinate information.

Before writing the 3D coordinates, the Matlab program a median and a Gaussian filtering in order to smooth the facial surface. This gives a better look to the 3D model, however as in most filtering cases, it can also lead to important data loss and/or corruption.

Below you can find the images of the 3D model obtained together with the original model itself:

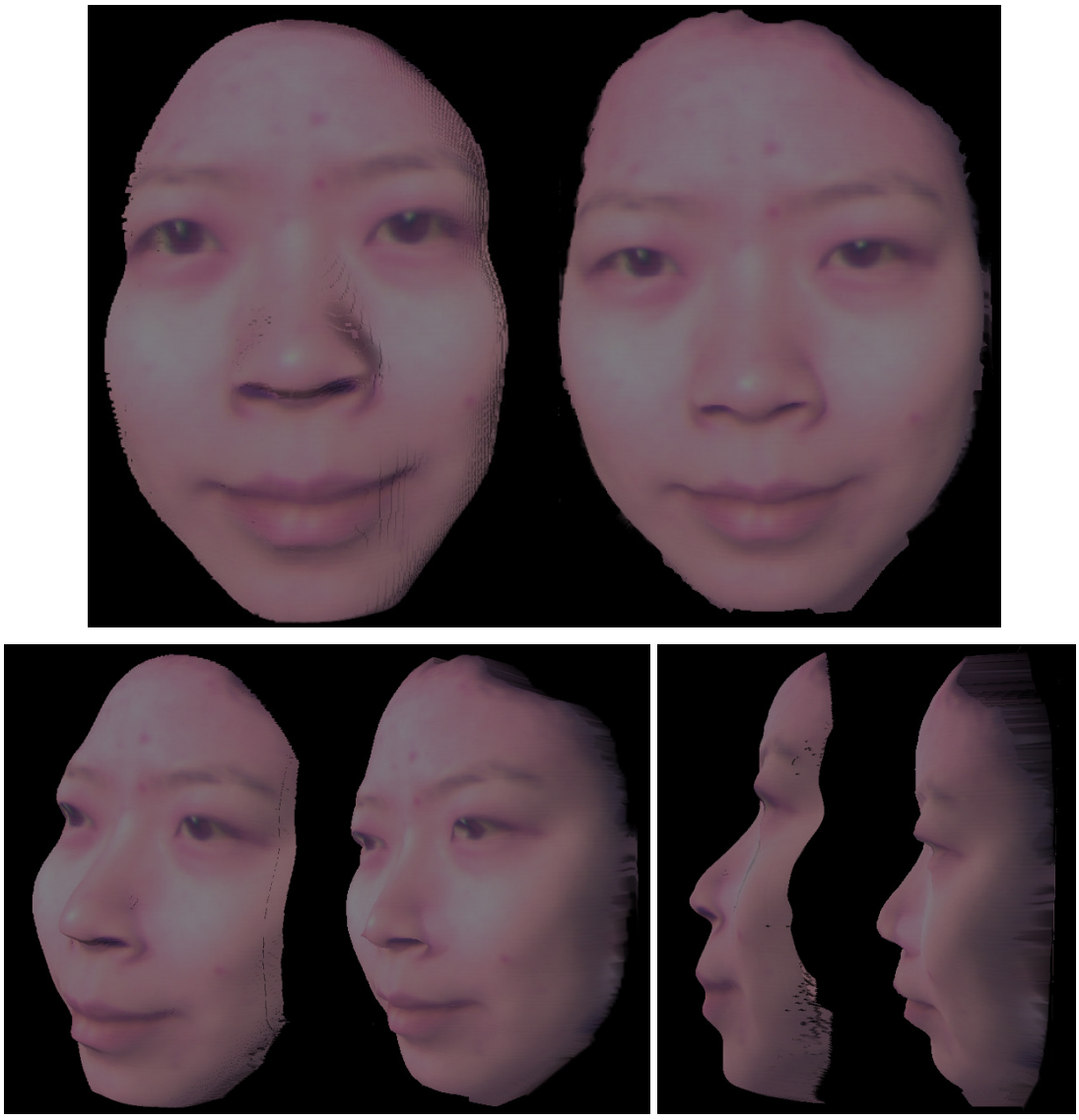

Figure 52 The 3D model obtained (left) is compared with the actual model (right)

The generated 3D model has a different scale from the original model; hence for the comparison of the two a preliminary processing should be made. This normalization can be done in numerous ways and in this work one of the simplest methods is chosen to be applied, which is to normalize Z vertices between 0 an 1. Thereby, a little more accurate evaluation can be done.

The resultant depth maps are shown in the following figure. This pair exposes how the generated model is lacking detailed shape information to view. Moreover, the

scale discrepancy in the images should be noted. Hence, an interpolation has to be applied after the normalization.

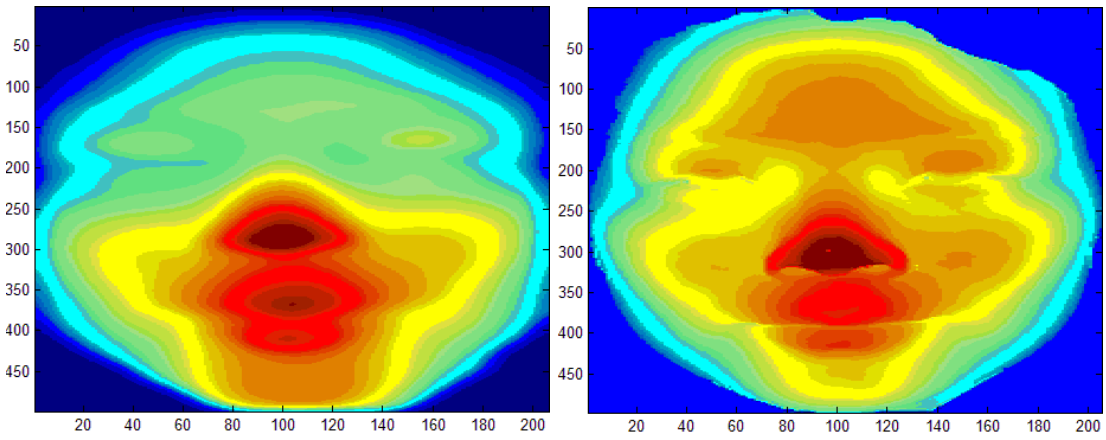

Figure 53 The normalized depth image obtained (left) is compared with the actual interpolated model (right)

Certainly, for the evaluation the images need to be registered to each other. Image registration is the process of transforming the different sets of data into one coordinate system. Registration is necessary in order to be able to compare or integrate the data obtained from different measurements.

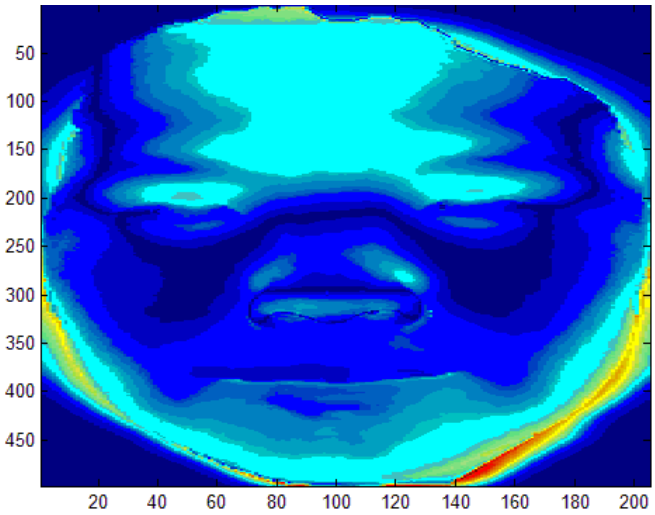

Figure 54 The diff image between the two depth images

Since in this experiment, both scans are focused on the face and the faces are similarly oriented, the two 3D models are assumed to be registered. For better comparison, a registration algorithm should be used. However, this process is out of the scope of this thesis.

As given in the previous figure, the error reaches its maximum at the eyes and forehead area and its minimum around the cheeks. The erroneous data that surrounds the whole face stems from both the lack of registration and the weakness of the structured light method around the edges.

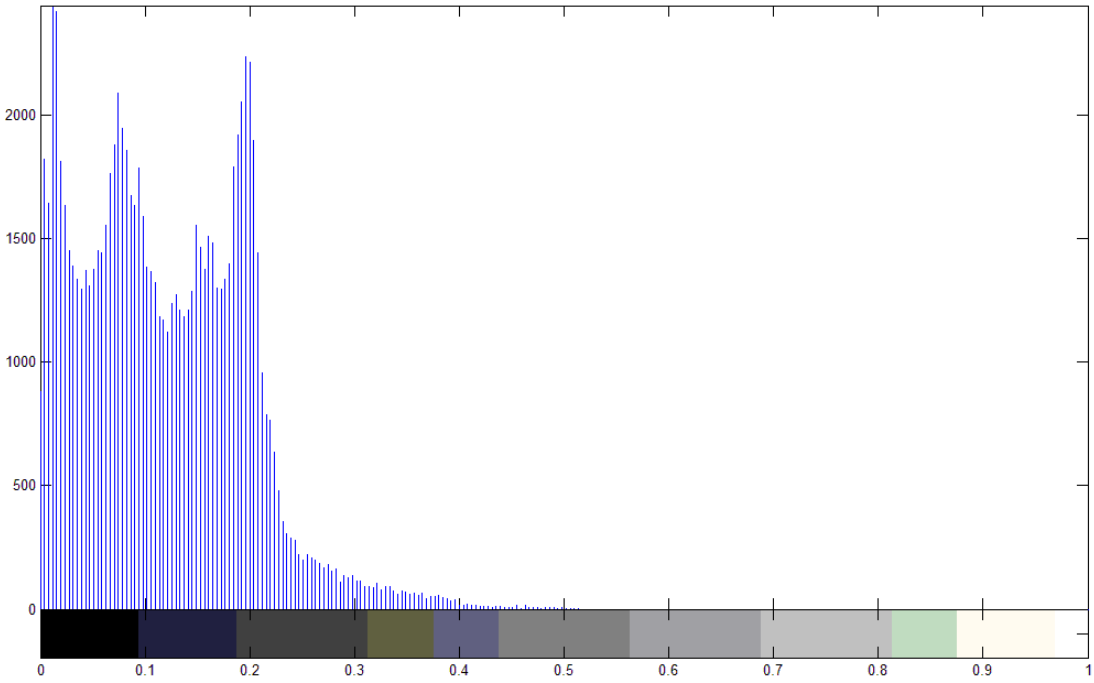

Figure 55 The error histogram for the compared depth images

Since the depth values are normalized, the error histogram should be discussed with the percentage values. The mean of the error values are 0.1119 which is around 11% for the best case, in which the actual pixel value is 1. The absolute values of the differences between the depth images are accumulated around 0, 0.1 and 0.2.

As a result, 2D/3D Imaging Simulator enables the user to make a thorough result analysis for various algorithms and applications. In the next chapter, these advantages will be highlighted briefly along with the main points that are mentioned throughout the thesis.

CHAPTER 4

CONCLUSION

In this thesis, 2D/3D Imaging Simulator tool is presented. Today's complex and versatile imaging systems are capable of acquiring high quality color and shape data. However, these are expensive systems which also demands arduous efforts for calibration, operation and maintenance. Taking all those disadvantages into account, the simulator presented here offers a great convenience.

Algorithm development process requires easily accessed and controlled data to be tested, corrected and verified. 2D/3D Imaging Simulator allows the users to generate their own data with known setup parameters and environment conditions. Consequently, the algorithms developed and implemented can be correctly tested on the images synthesized.

Additionally in the thesis, the usability of the developed tool is proven with an example application on 3D face reconstruction using stereo images and structured light. Images of the 3D models obtained from the FRGC database with a structured light projected on them are generated after the preprocessing. Those images are used in the "3D Face Scanner Tool" by Ahmet Oğuz Öztürk and 3D models are calculated using the disparity between the textures on the faces. It is shown that contrary to the

real systems; a detailed analysis can be enabled using the 2D/3D Imaging Simulator, since the actual model is available as the ground truth.

In the following subsections, the difficulties encountered and the plans for the future work will be explained briefly.

4.1 Encountered Difficulties

There are several major difficulties faced in this study, such as texture projection, depth measurement and memory consumption optimization in the development stage of the tool and hole filling and feature-preserving smoothing in the application part.

The difficulty in the texture projection arose from the misjudgment on interpreting how it can be simulated in OpenGL. Initial ideas on the topic was relating the texture projection to shadow casting in OpenGL, which makes sense if the structured light is considered as structured "shadow" on the surface. However, this is a very cumbersome and an ineffective method, contrary to the "Light Mapping" method used in this thesis. Lighting can be easily simulated by creating a texture map that mimics the light pattern and by applying it as a texture to the lit surface.

Similarly, in the depth measurement, initial approach was to apply ray tracing to the scene. In computer graphics, ray tracing is a technique for generating an image by tracing the path of light through pixels in an image plane. However, this technique has a major drawback: its performance. Since no performance enhancement algorithms were inserted, the scanning process would take hours. This problem was overcome by utilizing the OpenGL's depth buffer. Using the depth buffer accelerated the depth image generation drastically.

About hole filling and feature-preserving smoothing issues, the solutions applied are explained in detail in Chapter 3.

4.2 Future Work

Even though the tool is capable of expanding user's test and analysis adequacy, it can stillbe accepted as its initial phase. Certainly, many additions and expansions can be applied in order to increase the usability of the tool. Some of the properties that would be a useful addition to the tool are explained in the following.

One of the ideas for additional features is to enable the tool to conduct automated runs. In other words, the tool will automatically parse a script written or generated by the user, and it will function according to the commands typed in it. This simple modification would allow the user to generate the outputs according to the required setups without having to operate the tool while it is running. Thinking that the most time consuming part of the tool operation is the wait-time for the images to be generated, and with this ability, this idle time can be saved.

Another efficacious property for the 2D/3D Imaging Simulator would be the realistic scanner and camera models defined within the tool. Firstly, modeling the scanner noise would give the user the opportunity to observe how an actual scan would be like and hence to develop and test 3D noise removal algorithms based on the applied noise model. Secondly, similar to adding scanner imperfections, predefined and user-defined camera models can be added to the system. Namely, intrinsic and extrinsic parameters for the cameras can be defined and used in image acquisition and hence more realistic images can be generated that would help the user for instance, to test a camera calibration application. In conclusion, this realistic models for the scanner and the camera yields to an expanded observation and analysis opportunity.

Lastly, one of the main missing features of the tool is the shadow casting in the virtual world. In real world, the objects occlude light in different ways and cast shadows on themselves or other objects. In OpenGL, shadows are not directly supported. However, there are many standard algorithms for rendering shadows that can be easily applied to the tool. For example, exploiting the stencil buffer in shadow

calculations has become fairly popular among computer graphics researchers due to its convenience and flexibility. Shadow rendering property would help to display much more realistic scenes as well as acquiring more accurate images.

4.3 Notes

In this last section, small notes about the virtual instruments designed in the METU VISION 2D/3D Imaging Simulator Tool will be given. The purpose of this section is to warn the users about the instruments functioning ideally. Certainly, the ideal behaviors of the camera, the scanner and the lights mutilate the resultant outputs. Those effects will be discussed in detail in the following.

Firstly, as mentioned before, the camera implemented into the simulator, is a pinhole camera which is far away from modeling a real camera correctly. As shown in Figure 22, the only parameters of this camera are the view angle and the focal length. As a matter of fact, this idealistic behavior removes all the distortions inserted into the images by the real cameras. This yields to both positive and negative characteristics. For the advantages, firstly, a perfect testing environment is created that discards all the side effects of a real camera and this leaves the user only with the imperfectness of his testing material. Secondly, it gives the user a ground truth image to compare his image rectification results. Image rectification is a transformation process used to project multiple images onto a common image surface. It is used to correct a distorted image into a standard coordinate system [61]. However, for the disadvantages, if we have a system that works on actual images and takes the actual camera model into account, this ideal assumption prevents the user from obtaining the simulated results correctly. Ideal camera model inserts a limitation into our simulating system that the outputs are only reliable with methods that have the same assumptions. Previously, in the future work section, design of realistic camera models is suggested to be a useful addition into the tool.

On the other hand, for the scanners, the situation is less complicated. The main deviation of the scanners from the ideal is the noise on the 3D model obtained. This noise can be modeled in various ways; however in our system this has not been realized. Similar to the camera, the scanning process in the simulator is also ideal. Again, as an advantage, since the scanner noise is eliminated, the user is left with the defects he created. Additionally, obtaining the perfect scan output enables the algorithms for noise removal from the actual 3D models to be verified easily. Again, the user should be aware that the simulator works correctly only if the simulated system adopts the same assumptions.

Lastly, the results of ideal light behavior are similar but more influential since it is related to the created virtual world itself. In the METU VISION 2D/3D Imaging Simulator, the lights are modeled as directional sources, that emits light ideally in a specific direction and the light neither reflects back or nor scatters. In computer graphics, there are many techniques to produce very high degree of photorealism, such as ray tracing. However, the computational cost of those methods is high. The necessity of such an addition can be arguable, but of course, it would be a huge enhancement for more realistic scene simulation. As repeated several times before, the user should take the ideal light characteristic into account before applying his model in this simulator.

REFERENCES

[1] British Archaeology, *The Stonehenge Laser Show*, November, 2003.

[2] Marc Pollefeys and Luc Van Gool, *From Images to 3D Models*, Communications of the ACM, Vol. 45, Issue 7, 2002, pp. 50-55.

[3] Roger D. Smith, *Simulation Article*, 4th ed. New York: Encyclopedia of Computer Science, 2000.

[4] B. K. P. Horn, *Obtaining shape from shading information,* Mit Press Series Of Artificial Intelligence Series, Shape From Shading, 1989, pp.123-171.

[5] E. Prados, O. Faugeras, *Shape from Shading: A Well-Posed Problem?*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2, 2005, pp.870-877.

[6] P. Dupuis and J. Oliensis, *An Optimal Control Formulation and Related Numerical Methods For a Problem in Shape Reconstruction*, Vol. 4, Number 2, The Annals of Applied Probability, 1994, pp. 287-346.

[7] E. Rouy and A.Tourin, *A Viscosity Solutions Approach to Shape-from-Shading*, Vol. 29, Issue 3, SIAM J. Numerical Analysis, 1992, pp. 867-884.

[8] F. Courteille, A. Crouzil, J. D. Durou and P.Gurdjos, *Towards Shape from Shading under Realistic Photographic Conditions*, Vol. 2, Proceedings of the 17th International Conference on Pattern Recognition, 2004, pp. 277-280.

[9] E. Parados and O. Faugeras, *Perspective Shape from Shading and Viscosity Solutions*, Vol. 2, ICCV, 2003, pp. 826-831.

[10] A. Tankus, N. Sochen and Y. Yesherun, *Reconstruction of Medical Images by Perspective Shape-from-Shading*, ICPR, 2004.

[11] E. Parados and O. Faugeras, *Unifying Approaches and Removing Unrealistic Assumptions in Shape from Shading: Mathematics Can Help*, ECCV, 2004.

[12] E. Parados and O. Faugeras, *A Mathematical and Algorithmic Study of the Lambertian SFS Problem for Orthographic and Pinhole Cameras*, Technical Report RR-5005, INRIA, 2003.

[13] METU, EEE, Robot Vision Lecture Notes, Shape from Shading
http://www.eee.metu.edu.tr/~alatan/Courses/rv08.pdf
Last Access: September, 2008

[14] P. Belhumeur, *A Bayesian Approach to Binocular Stereopsis*, Vol. 19, Number 3, International Journal of Computer Vision, 1996, pp.237-260.

[15] O. Williams, M. Isard and J. MacCormick, *Estimating Disparity and Occlusion in Stereo Video Sequences*, Vol. 2, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005, pp. 250-257.

[16] D. Marr and T. Poggio, *A Computational Theory of Human Stereo Vision*, Vol. 204, Number 1156, Proceedings of the Royal Society of London Series B-Biological Sciences, 1979, pp. 301-328.

[17] W. Grimson, *From Images to Surfaces: A Computational Study of the Human Early Visual System*, MIT Press, Cambridge, Massachusetts, 1981.

[18] S. Pollard, J. Mayhew and J. Frisby, *PMF: A Stereo Correspondence Algorithm Using a Disparity Gradient Limit*, Vol. 14, Number 4, Perception, 1985, pp. 449-470.

[19] S. Gimel'farb, *Symmetrical Approach to the Problem of Automatic Stereoscopic Measurements in Photogrammetry*, Vol. 15, Cybernetics, Consultant Bureau, N.Y., 1979, pp. 235-247.

[20] H. Baker and T. Binford, *Depth from Edge and Intensity Based Stereo*, International Joint Conference on Artificial Intelligence, Vancouver, Canada, 1981, pp. 631-636.

[21] Y. Ohta and T. Kanade, *Stereo by Intra- and Inter-scanline Search Using Dynamic Programming*, Vol.7, Number 2, IEEE Trans. On Pattern Analysis and Machine Intelligence, 1985, pp. 139-154.

[22] M. Okutomi and T. Kanade, *A Locally Adaptive Window for Signal Processing*, Vol. 7, International Journal of Computer Vision, 1992, pp. 143-162.

[23] I. Cox, S. Hingorani and S. Rao, *A Maximum Likelihood Stereo Algorithm*, Vol. 63, Number 63, Computer Vision and Image Understanding, 1996.

[24] R. Koch, *Automatische Oberflachenmodellierung Starrer Dreidimensionaler Objekte aus Stereoskopischen Rundum-Ansichten*, PhD Thesis, University of Hannover, Germany, 1996.

[25] L. Falkenhagen, *Depth Estimation from Stereoscopic Image Pairs Assuming Piecewise Continuous Surfaces*, European Workshop on Combined Real and Synthetic Image Processing for Broadcast and Video Productions, Hamburg, Germany, 1994.

[26] L. Falkenhagen, *Hierarchical Block-Based Disparity Estimation Considering Neighborhood Constraints*, Proc. International Workshop on SNHC and 3D Imaging, Rhodes, Greece, 1997.

[27] Robert B. Fisher, CVonline: The Evolving, Distributed, Non-Proprietary, On-Line Compendium of Computer Vision
http://homepages.inff.ed.ac.uk/rbf/CVOnline/LOCAL_COPIES/MARBLE/medium/stereo/corresp.htm
Last Access: September, 2008

[28] Wikipedia, Structured Light
http://en.wikipedia.org/wiki/Structured_light
Last Access: September, 2008

[29] CSIRO Mathematical and Information Sciences
http://www.cmis.csiro.au/IAP/recentprojects/stereoEE02.htm
Last Access: September, 2008

[30] Robert B. Fisher, CVonline: The Evolving, Distributed, Non-Proprietary, On-Line Compendium of Computer Vision
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/HENKEL/research/stereo/principle.html
Last Access: September, 2008

[31] Robert B. Fisher, CVonline: The Evolving, Distributed, Non-Proprietary, On-Line Compendium of Computer Vision
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/HENKEL/research/stereo/principle.html
Last Access: September, 2008

[32] Robert B. Fisher, CVonline: The Evolving, Distributed, Non-Proprietary, On-Line Compendium of Computer Vision
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/medium/stereo/corresp.htm
Last Access: September, 2008

[33] The University of Western Australia, Computer Science and Software Engineering, Computer Vision Lecture Notes
http://undergraduate.csse.uwa.edu.au/units/233.412/Lectures/stereo.pdf
Last Access: September, 2008

[34] C. Rocchini, P. Cignoni, C. Montani, P. Pingi and R. Scopigno, *A Low Cost 3D Scanner Based on Structured Light*, Vol. 20, Issue 3, Eurographics Conf., Interlaken, Switzerland, 2002, pp. 299-308.

[35] Wikipedia, Structured Light 3D Scanners
http://en.wikipedia.org/wiki/Structured_Light_3D_Scanner
Last Access: September, 2008

[36] L. Zagorchev, A. Goshtasby, *Recovering Non-Rigid 3-D Structures from Stereo and Structured Light*, Wright State University, Intelligent System Laboratory, 2003.

[37] Wikipedia, Structured Light 3D Scanners
http://upload.wikimedia.org/wikipedia/commons/a/aa/3-proj2cam.svg
Last Access: September, 2008

[38] B. Ben Amor, K. Ouji, M. Ardabilian, and L Chen, *3D Face Recognition by ICP-Based Shape Matching*, ICMI, 2005.

[39] Xiaoguang Lu, Dirk Colbry, Anil K. Jain, *Three-Dimensional Model Based Face Recognition*, Vol. 1, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004, pp. 362-366.

[40] Xiaoguang Lu, Dirk Colbry, Anil K. Jain, *Matching 2.5D Scans for Face Recognition*, International Conference on Biometric Authentication, 2004.

[41] J. Huang, B. Heisele, V. Blanz, *Component-based Face Recognition with 3D Morphable Models*, Proc. of the 4th International Conference on Audio- and Video-Based Biometric Person Authentication, Guildford, UK, 2003, pp. 27-34.

[42] V. Blanz, T. Vetter, *Face Recognition Based on Fitting a 3D Morphable Model*, Vol. 25, Number 9, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2003, pp. 1063-1074.

[43] B. Moghaddam, J.H. Lee, H. Pfister, R. Machiraju, *Model-Based 3D Face Capture with Shape-from-Silhouettes*, Proc. of the IEEE International Workshop on Analysis and Modeling of Faces and Gestures, Nice, France, 2003, pp. 20-27.

[44] Dr. Dirk Joel Luchini Colbry, Publications
http://www.public.asu.edu/~dcolbry/pubs.html
Last Access: September, 2008

[45] Digital Hammurabi Project, Virtual Scanner
http://www.jhu.edu/digitalhammurabi/version2/news/virtual_scanner.html
Last Access: September, 2008

[46] The Virtual Vision Lab: Instructional Lab Modules for Machine Vision
http://www1.cs.columbia.edu/~allen/VVL/contents.html
Last Access: September, 2008

[47] C. C. Aranha, S. R. Carvalho, L. M. G. Goncalves, *Cambio: A Realistic Simulated Robot for Vision Algorithms*, Vol. 11, Number 11, Journal of WSCG, Plzen, Czech Republic, 2003.

[48] T. Jones, *The Virtual Vision Laboratory*, Master's Thesis, Department of Computer Science, Columbia University, 1995.

[49] P. J. Phillips, P. J. Flynn, T. Scruggs, K. W. Bowyer, J. Chang, K. Hoffman, Marques J., J. Min, and W. Worek, *Overview of the Face Recognition Grand Challenge*, Proc. Computer Vision and Pattern Recognition, San Diego, 2005, pp. 947–954.

[50] K. Chang, K. Bowyer, and P. Flynn, *Multiple Nose Region Matching for 3D Face Recognition under Varying Facial Expression*, Vol. 28, Number 10, IEEE Trans. on Pattern Analysis and Machine Intelligence, 2006, pp. 1695-1700.

[51] T. R. Jones, F. Durand, and M. Desbrun, *Non-Iterative, Feature-Preserving Mesh Smoothing*, ACM Transactions on Graphics, 2003, pp. 943-949.

[52] S. Fleishman, I. Drori, and D. Cohen-Or, *Bilateral Mesh Denoising*, ACM Transactions Graphics - Proceedings of ACM SIGGRAPH, 2003, pp, 950-953.

[53] P. Choudhury and J. Tumblin, “The Tri-lateral Filter for High Contrast Images and Meshes”, Proceedings of the Eurographics Symposium on Rendering, 2003, 186-196.

[54] A. O. Öztürk, *3D Face Reconstruction Using Stereo Images and Structured Light,* Master's Thesis, Department of Electrical and Electronics Engineering, Middle East Technical University, 2007.

[55] Wikipedia, OpenGL
http://en.wikipedia.org/wiki/OpenGL
Last Access: September, 2008

[56] Wikipedia, Microsoft Foundation Class Library
http://en.wikipedia.org/wiki/Microsoft_Foundation_Class_Library
Last Access: September, 2008

[57] CFileDialog Class That Only Displays Folders
http://www.codeguru.com/cpp/w-d/dislog/dialogforselectingfolders/article.php/c1883
Last Access: September, 2008

[58] Steve Butrimas, Texture Projection
http://www.codesampler.com/usersrc/usersrc_6.htm#oglu_projtexture
Last Access: September,2008

[59] John Burkardt, Microsoft BMP Files - Read and Write Utilities
http://www.csit.fsu.edu/~burkardt/cpp_src/bmp_io/bmp_io.html
Last Access: September, 2008

[60] Wikipedia, Image Registration
http://en.wikipedia.org/wiki/Image_registration
Last Access: September, 2008

[61] Wikipedia, Image Rectification
http://en.wikipedia.org/wiki/Image_rectification
Last Access: September, 2008

APPENDIX A

THE USER'S MANUAL

1. Starting Up
    a. Make sure that compatible .obj files are ready.
    b. Make sure that texture images matched with the .obj files are ready. (if needed)
    c. Make sure that texture images to be projected are ready in the compatible format. (if needed)
    d. Run the "VirtualScanner.exe" file.
2. Creating the Virtual World
    a. Browse folders for the .obj file to be loaded.
    b. Choose the 3D object type: with or without texture.
    c. Enter scale.
    d. Enter object position and rotation.
    e. Press the "Add Object" button.
    f. Repeat a-e for more objects.
    g. Enter camera position and rotation.
    h. Press the "Show" button.
    i. Press the "Reset World" button to reset the world.
3. Adjusting the Material Properties (for objects without texture)
    a. Adjust the ambient properties.
    b. Adjust the diffuse properties.
    c. Adjust the specular properties.
    d. Enter the "Shininess" value.
4. Adjusting the Light Properties
    a. Adjust the global ambient light.
    b. Enter the light position.
    c. Adjust the ambient, diffuse and specular light properties.
    d. Press the "Add Light" button.

    e. Repeat b-d for more lights.
    f. To change a light's settings, select the light from the drop list.
    g. Adjust the light's settings.
    h. Press "Switch" to turn the light on and off.
5. Adjusting the Texture Projection Settings
    a. Browse folders for the texture image to be projected.
    b. Enter the projection position and rotation.
    c. Press the "ON" button.
    d. Repeat a-c to change the projection.
6. Outputs
    a. Select the output type: color or depth image
    b. Enter the plane distance.
    c. Enter the view angle.
    d. Press the "Create Image" button.
7. View Control
    a. Use the "In"/"Out" buttons to zoom in and out, respectively.
    b. Use the "Align Camera" button to align the view to the camera's position and direction.
    c. Use the "Reset View" button to align the view to the origin and towards the -z direction.

APPENDIX B

SOURCE CODE DEFINITIONS

(In Alpabetical Order)

1. Bmp_io

bool bmp_24_write (char *file_out_name, unsigned long int width, long int height, unsigned char *rarray, unsigned char *garray, unsigned char *barray)

"bmp_24_write" writes the header and data for a BMP file using three colors. Inputs are the name of the output file, the X dimension of the image, the Y dimension of the image and the pointers to the red, green and blue color arrays. Output is true if an error occurs.

2. folder_dialog

CFolderDialog(CString* pPath)

This class is used to navigate into a folder without being obliged to choose a file. The input argument represents string where selected folder is going to be saved.

3. matrix4x4

OpenGL matrices are column major and can be quite confusing to access when stored in the typical, one-dimensional array often used by the API. In this class, there are some shorthand conversion macros, which convert a row/column combination into an array index.

Matrix4x4f invertMatrix(const Matrix4x4f &matIn)
This method takes the inverse of a matrix and returns the result.

void setMatrix(float *matrix)
This method is used to set a private attribute "float m_matrix[16]" equal to the input matrix.

4. OpenGLControl

void AdjustLights()
This method is called every time the OpenGL scene is redrawn. It manages the changes and modifications to the light settings in the virtual world by using OpenGL's light related methods.

void Create(CRect rect, CWnd *parent)
This method is called only at the beginning of the execution to create the OpenGL display in the graphical user interface of the 2D/3D Imaging Tool. It handles some general initializations such as the display size and color resolution. It invokes the InitGL() function for further initialization.

void CreateImage(CString ObjectPath)
"CreateImage" method is invoked to prepare the outputs of the tool as depth or color images. It reads the depth and color buffers created as the OpenGL library attributes and writes them into png or bmp images, respectively.

void DrawGLScene()

This is the main method that handles the OpenGL display screen. It sets the material properties for the objects without texture. It triggers the texture projection and invokes "AdjustLights" method for light manipulations. It realizes the translations, rotations and zoom in/out actions that are initiated by the user and the display of the virtual world after the "Show" button is pressed.

void InitGL()

This method makes main OpenGL initializations and settings for the view scene.

void OnPaint()

"OnPaint" method is the base method that is called repetitively and invokes the "DrawGLScene" function. In case an object is added to the scene, it calls the "SetupWorld" for required procedure.

OpenGLControl()

This is the class constructor in which the class attributes are initialized.

void project(float fpx, float fpy, float fpz, float fdx, float fdy, float fdz)

"project" method realizes the texture projection by calling the loadLightMapTexture method of the "projtex" class.

void SetupWorld(VERTEX Pos, VERTEX Ang, char *File)

"SetupWorld" appends the newly added objects to the rest of the vertices in the virtual world by reading the compatible .obj files and it organizes the storage of the corresponding color, texture and position data.

void triangleNormal(VERTEX a, VERTEX b, VERTEX c, float *normal)

This is an auxiliary method to calculate the normal of a surface created by the given three vertices as the inputs.

~OpenGLControl()

The class destructor that deletes the display buffer and the "TexProj" class instance created.

5. projtex

void loadLightMapTexture(const char *name)

This method sets up texture generation mode and the corresponding planes and builds the mipmaps to be used in texture projection.

void loadTexture(const char *name, int *textsize)

"loadTexture" method mounts the matched texture image of the object on its 3D model.

void textureProjection(Matrix4x4f &mv)

Here, the transformations on the texture matrix for the light map are realized using the basic equation: M = Bias * Scale * ModelView for light map * Inverse ModelView.

6. StdAfx

This is a source file that includes just the standard includes. VirtualScanner.pch will be the pre-compiled header and stdafx.obj will contain the pre-compiled type information.

7. VirtualScanner

bool InitInstance()

Standard initialization function for the VirtualScannerDlg.

8. VirtualScannerDlg

void CVirtualScannerDlg(CWnd* pParent)

It is the class constructor where the AFX data initializations are made.

void DoDataExchange(CDataExchange* pDX)

This method is called by the framework to exchange and validate dialog data that is entered or read by the user.

void OnAddLight()

It handles the GUI operations for the tool when a light is added and sets the related attributes of the OpenGLControl class.

void OnAddObject()

It reads the object settings entered by the user and transmits the data obtained to the OpenGLControl class.

void OnAlighCamera()

It sets the looking angle and the looking position variables in the OpenGLControl class equal to the camera settings and hence realizes the camera alignment.

void OnBrowseObject()

It opens the dialog box for file browsing for 3D object.

void OnBrowseTexture()

It opens the dialog box for file browsing for the texture image to be projected.

void OnChangeObjShininess()

It updates the shininess information for the object every time this value is reentered by the user.

void OnClearWorld()

All variables and attributes for the OpenGLControl class are reset as id the tool is restarted.

void OnColor()

This method indicates that the output is going to be the color image.

void OnCreateImage()

"CreateImage" function of the OpenGLControl class is invoked with required arguments and attribute settings.

void OnDepth()

This method indicates that the output is going to be the depth image.

bool OnInitDialog()

The GUI components are initialized to their starting states and the "Create" function of the OpenGLControl class is invoked to start the OpenGL display in the GUI.

void OnLButtonDown()

This method controls the mouse button activity on the display screen for view angle and position modifications using the mouse.

void OnLButtonUp()

This method controls the mouse button activity on the display screen for view angle and position modifications using the mouse.

void OnMouseMove()

This method controls the mouse button activity on the display screen for view angle and position modifications using the mouse.

void OnPaint()

This is the method that is called repetitively for GUI visualization.

void OnProjectTexture()

It invokes the "project" function of the OpenGLControl class.

void OnResetView()

It sets the viewing position and angle parameters of the OpenGLControl class to zero to reset them to the origin.

void OnSelchangeLightList()

"OnSelchangeLightList" method controls the selection of the added lights in the virtual world from the corresponding drop list and updates the slider controls in the GUI according to the light selected.

void OnShowWorld()

This method finalizes the setup of the virtual world by finally inserting the camera settings and switches the virtual world display on.

void OnSwitchLight()

It toggles the selected light on and off.

void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)

This method manages the slider controls in the GUI.

void OnZoomIn()

It is called when the "In" button is pressed by the user and it moves the looking position "0.1" towards the looking direction.

void OnZoomOut()

It is called when the “Out” button is pressed by the user and it moves the looking position “0.1” towards opposite of the looking direction.