T.C.
BAHÇEŞEHİR ÜNİVERSİTESİ

DEVELOPING AN INTERACTIVE VISUALIZATION TOOL FOR HISTORICAL BUILDINGS ON TOUCH SCREENS

Master of Science Thesis

EMRE BEGEN

İSTANBUL,2013

T.C.

BAHÇEŞEHİR ÜNİVERSİTESİ

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

COMPUTER ENGINEERING

DEVELOPING AN INTERACTIVE VISUALIZATION TOOL FOR HISTORICAL BUILDINGS ON TOUCH SCREENS

Master of Science Thesis

EMRE BEGEN

Supervisor: Asst. Prof. Dr. Övgü ÖZTÜRK

İSTANBUL,2013

T.C.
BAHÇEŞEHİR ÜNİVERSİTESİ

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
COMPUTER ENGINEERING

Title of the Master's Thesis : Developing an interactive visualization tool for historical buildings on touch screens

Name/Last Name of the Student : Emre BEGEN

Date of Thesis Defense : 03/09/2013

The thesis has been approved by the Graduate School of Natural and Applied Sciences.

Asst. Prof. F. Tunç BOZBURA

Acting Director

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

Examining Committee Members:

Asst. Prof. Dr. Övgü ÖZTÜRK :

Asst. Prof. Dr. Devrim ÜNAY :

Asst. Prof. Dr. Egemen ÖZDEN:

ACKNOWLEDGMENTS

I would like to thank all people who have helped and inspired me during my study.

Especially, I offer my sincerest gratitude to my supervisor, Asst. Prof. Dr. Övgü Öztürk, who has supported me, thought-out my thesis with his experience and knowledge. It would be impossible to complete this study without his encouragement, motivation and guidance.

I would like to show my gratitude to my friends, Olcay Bahtiyar and Yusuf Gökhan Yavuz for their professional insight. Without their support, this thesis would not be constructed.

Also I would like to thank to Katsushi Ikeuchi for him researchs about the computer vision and 3d model segmentation. My research would not have been possible without him helps.

Finally, I would like to thank to my fiancée, Sinem Şimşek, for her everlasting love, endless support and encouragement in every part of my life.

Emre BEGEN

3 September, 2013

ÖZET

TARİHİ YAPILARI DOKUNMATİK EKRANLARDA GÖRSELLEŞTİREBİLMEK İÇİN İNTERAKTİF BİR ARAÇ GELİŞTİRMEK

Emre BEGEN

Bilgisayar Mühendisliği

Tez Yöneticisi: Yard. Doç. Dr. Övgü ÖZTÜRK

Eylül 2013, 56 Sayfa

Bu çalışmanın amacı tarihi binaların ve eserlerin 3 boyutlu olarak görselleştirilmesi ve oluşturulan görsellerin çoklu dokunmatik ekranlarda insan bilgisayar etkileşimi araçları ile kontorolü ve incelenmesi kolay bir yapıya getirmektir.

Öncelikle görselleştirme için gerekli olan dataları elde etmek adına bir lazer tarayıcı ile tarama işlemi yapılmıştır. Tarama işlemi sonrası elde edilen veriler bir yazılım aracılığı ile görselleştirme için uygun dosya formatına dönüştürülmüştür. Oluşturulan dosya içerisinde sakladığımız verileri okumak ve bu okuduğumuz verilerin 3 boyutlu olarak görselleştirilebilmesini sağlayan bir program geliştirilmiştir. Geliştirilen bu program aynı zamanda insan bilgisayar etkileşimini sağlayabilmek için el hareketlerini tanıma sistemini de içermektedir. Son aşamada ise oluşturulan 3 boyutlu tarihi eser modelinin renklendirilmesi için gerçek modelin renkli fotoğrafları çekilmiştir. Çekilen bu fotoğraflar bir yazılım aracılığı ile görselleştirme için gerekli olan bilgileri barındıran dosyaya renk kodları olarak eklenmiştir.

Bunlara ek olarak, bu tez daha önce yapılmış 3 boyutlu görselleme araçlarını, insan bilgisayar etkileşimi ve yöntemleri hakkında kısaca bilgi ve bu çalışmanın tarihi eserleri 3 boyutlu olarak nasıl görselleşebileceğini de özetler.

Anahtar Kelimeler: 3 Boyutlu Görselleme,El Hareketlerini Tanıma,El Yazısı Tanıma

ABSTRACT

DEVELOPING AN INTERACTIVE VISUALIZATION TOOL FOR HISTORICAL BUILDINGS ON TOUCH SCREENS

Emre BEGEN

Computer Engineering

Supervisor: Asst. Prof. Dr. Övgü ÖZTÜRK

September 2013, 56 Pages

The objective of this thesis is to visualize of the historical buildings and historical monuments in the form of 3 Dimensinonal (3D). Also controlling and analyzing of this 3D models in the multi touch screen with a human computer interaction tool is a objective of this thesis.

Firstly historical buildings was scanned by a laser scanner for visualizing buildings and collecting datas. After the scanning process, collected datas was transform to an appropriate file format by a software for visualize. A program was developed for reading datas and 3 D visualize by using these datas. In addition that this program concirn a hand motion recognition system for showing to human computer interaction. Lastly the coloured photoghraphs was taken of the real historical model for colouring the 3D model. And these photoghraphs was added in the form of color code by an software for coloured visualization.

In addition to all of these, this thesis abstract the previous work of 3D visualization, hand motion recognition and shortly explain of these. Also it gives some information about how can we visualize the historical objects by this work.

Keywords: 3D Visualization. Hand Motion Recognition, Hand Written Recognition

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

3D	:	3 Dimensional
OPENGL	:	Open Graphic Library
FTIR	:	Frustrated Total Internal Refection
DI	:	Diffused Illumination
SAW	:	SURFACE WAVE TOUCH SURFACES
CH	:	Cultural Heritage
API	:	Application Programming Interface
PLY	:	Polygon File Format
Kb	:	Kilobyte
Mb	:	Megabyte
CPU	:	Central Processing Unit
FPS	:	Frame Rate

1 INTRODUCTION

The objective of this thesis is to visualize of the historical buildings and historical monuments in the form of 3 Dimensinonal (3D). Also controlling and analyzing of this 3D models in the multi touch screen with a human computer interaction tool is a objective of this thesis.

Firstly historical buildings was scanned by a laser scanner for visualizing buildings and collecting datas. After the scanning process, collected datas was transform to an appropriate file format by a software for visualize. A program was developed for reading datas and 3 D visualize by using these datas. In addition that this program concirn a hand motion recognition system for showing to human computer interaction. Lastly the coloured photoghraphs was taken of the real historical model for colouring the 3D model. And these photoghraphs was added in the form of color code by an software for coloured visualization.

In this thesis we used C#.NET development language for make a visualization application and SharpGL C#.NET opengl library for rendering process. We did translation,rotation scaling properties in our application. Also we can control the tranformation properties with a multi touch screen.

We worked worked some files with different size. In the beginning we worked small files and we success until we work huge datas. We had so much trouble with huge datas. Our system was not work the program, if it did then the program was working so slowly. We try to find a solution for this problem.

We have some success result in the experimental results. It has also our survey about the program which we made for develop a 3D visualization of historical objects.

2 RELATED WORK

There are several relevant thread of related work. Some of these multi touch technology, Open Graphic Library(OpenGL) and hand motion recognition.

We begin to explain some technical aspects some multi touch technology. Secondly we give some important knowledge about OpenGL and its usage areas and finally we try to describe hand motion recognition and its some methodology.

2.1 MULTI TOUCH TECHNOLOGY

Multi-touch displays are interactive graphics devices that combine camera and tactiletechnologies for direct on-screen manipulation. Multi-touch technology is not entirelynew, since the 1970s it has been available in different forms.

Modern computer interaction consists of a monitor, keyboard and a mouse. Limited by the operating system, it allows us only to control a single pointer on the screen. With multi-touch, multiple input pointers are introduced to the system which all can be controlled independently. Depending on the size of the display multi-touch allows multiple persons to interact with the same display at the same time.

In 1997 Matsushita et al. presented the HoloWall [Nobuyuki Matsushita and Jun Rekimoto 1997]. The HoloWall is a finger, hand, body and object sensitive wall. Multiple users can interact with the surface on the front side of the wall. The wall is made out of glass with a rear projection material attached. Behind the wall a digital projector is placed for display. On the same side as the digital projector a camera and an infrared illuminator is placed. When a user or object touches the wall, it reects infrared light which is captured by the camera.

In 2007 Microsoft presented their version of a multi-touch table, MS Surface [Microsoft Corporation. 2007].The table looks like a coffee table with an interactive surface. The technique used in the table is similar to the HoloWall. A diffuser is attached to the screen material. The table is illuminated with infrared light from the back. When a user touches the table, it will reect infrared light which is captured by the cameras inside the table. Because of the use of multiple cameras, the

input resolution is high enough to detect objects. Microsoft has demonstrated how fiducial markers can be used to tag objects and allow specific interaction with the table.

A few months after the press release of MS Surface, Microsoft Research Cambridge [Shahram Izadi, Steve Hodges, Alex Butler, Alban Rrustemi, and Bill Buxton 2007] demonstrated a multi-touch system on a notebook LCD panel. The construction of ThinSight consists of an array of retro-reflective optosensors which are placed behind the LCD panel. Each sensor contains an infrared light emitter and an infrared light detector. When an object touches the LCD panel, it will reect infrared light which the detector can notice. Because this technique relies on reecting infrared light, the system allows interaction with fi ducial markers.

2.1.1 TOUCH TECHNOLOGIES

Before describing Frustrated Total Internal Refection(FTIR) and Diffused IlluminationDI there are a number of alternative technologies that can be used to construct multitouch surfaces:

- Resistance Based Touch Surfaces
- Capacitance Based Touch Surfaces
- Surface Wave Touch Surfaces(SAW)

2.1.1.1 RESISTANCE BASED TOUCH SURFACES

Resistance based touch surfaces generally consist of two conductive layers which are coated with substances such as indium tin oxide [Wikipedia 2008] These layers are separated by an insulating layer, usually made of tiny silicon dots (see figure 2.1 according [Tyco Electronics. Elo TouchSystems. 2008]). The front of the panel is typically made of a exible hard coated outer membrane while the back panel is often a glass substrate. When users touch the display, the conductive layers are connected, establishing an electric current that is measured once horizontally and vertically by the controller in order to determine the exact position of a touch. Such touch surfaces have the advantage of low power consumption and are used in mobile devices such as the

Nintendo DS [Nintendo Co. 2008], PDAs and digital cameras and can be operated both with fingers or a stylus.

However, resistance based surfaces provide a low clarity interactive surface (about 75%-85%) and additional screen protection cannot be applied without impacting on their functionality.

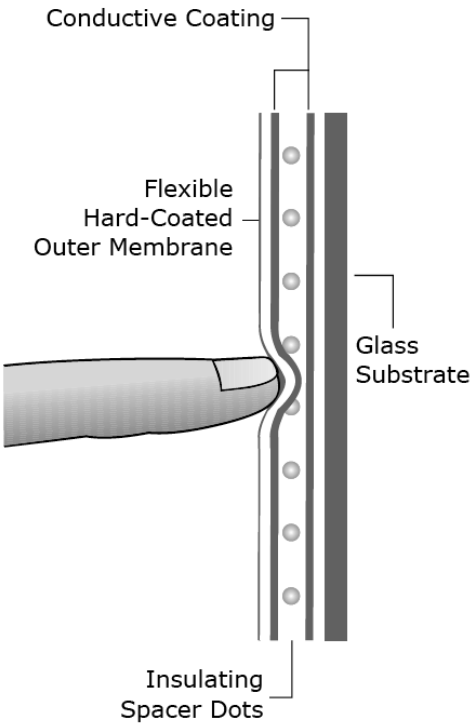

Figure 2.1 Schematic construction of a touch screen based on resistive technology according [http://www.elotouch.com/.]

2.1.1.2 CAPASITANCE BASED TOUCH SURFACES

In general capacitance based (multi-) touch surfaces can be subdivided into two classes:

- Surface Capacitance
- Projected Capacitance

Both techniques were primarily developed for single touch interaction. One advantage of capacitive touch surfaces in comparison to other technologies is their high clarity; making capacitive touch surfaces very suitable for use in many kinds of touch displays beyond simple touch pads. Capacitive touch screens can also be operated by any conductive device and are hence not limited to finger based interaction.

However, capacitive touch panels are relatively expensive to produce although they exhibit high durability and reliability.

2.1.1.3 SURFACE WAVE TOUCH SURFACES(SAW)

Systems that use surface wave technology are similar to those that use infrared grid technology. Transmitting and receiving piezoelectric transducers, for both the X- and Yaxes, are mounted on a faceplate and ultra-sonic waves on a glass surface are created and directed by reectors. By processing these to electronic signals and observing the changes when the faceplate is touched, it is possible to calculate the position of that interaction. Most SAW systems can support dual-touch.

2.1.1.4 OPTICAL BASED TOUCH SURFACES

Both optical and camera based-approaches share the same concept of processing and filtering captured images on patterns. As already discussed a number of systems are based on infrared illumination and as a result can suffer interference from ambient light in the environment. Due to their simple configuration optical approaches have the potential to be very robust.

2.2 OPEN GRAPHICS LIBRARY (OPENGL)

OpenGL provides a basic or core library of functions for specifying graphics primitives, attributes, geometric transformations, viewing transformations and many other operations. To be hardware

independent, the core library does not provide interactive input and output routines; however these routines are available in auxiliary libraries.

2.2.1 OPENGL GRAPHIC PRIMITIVES AND ATTRIBUTES

A computer scene contains various components which can be called graphics output primitives. These geometric primitives include points, straight line segments, circles, conic sections, and polygons. All geometric primitives are positioned in a reference frame called the world coordinate system. Furthermore, each geometric primitive has associated with it various attributes such as size and color. In general, there are two methods used for dealing with attributes. First an OpenGL function can be called with attribute values as arguments. Secondly, and more typically, OpenGL attributes are treated as state variables. All OpenGL primitives are displayed with the current state values until these attribute setting are changed.

Table 2.1: A summary OpenGL Libraries(OpenGL ARB Guide 2005)

Library	Functionality	Prefix
Core	Specify graphics primitive, attributes, geometric transformations, viewing transformations and many other operations	"gl" for function, "GL " for symbolic constant, "GL" for built-in data type
Utility	Set up viewing and projection matrices, draw curves, spheres, or tessellations, display quadrics and B-splines, process the surface rendering operations or some other task	"glu" ("Graphics Library Utilities")
Utility Toolkit	Contains a set of functions to interact with windowing system	"glut" OpenGL Utility Toolkit

2.2.2 COLOR

Color is a basic attribute for all primitives, and this can be represented by a red-green- blue triple (R, G, B). OpenGL provides two modes: RGB and RGBA . The difference is that RGBA has a fourth parameter, α, for color blending of which simulation of transparency, or translucent effects is an important application (OpenGL ARB Guide 2005). The most often used color routines are

glColor(colorComponent)

and

glClearColor(colorComponent)

2.2.3 POLYGONS

For this surface implementation, a filled area is an important graphics component. Although in principle any filled area shape is possible, we will only be concerned about convex polygons as provided by OpenGL. Just as a curved line can be approximated with a set of line segments, a curved surface can be described by a set of polygonal facets, sometimes referred to as a polygon mesh. This is how we will implement surfaces (Hui Zhao 2006). A polygon is a plane segment determined by a set of vertices, and the order in which they are to be connected by line segments. In OpenGL, a polygon must be convex. We also distinguish the back face from the front face of a polygon by imagining an object enclosed by a set of polygonal surfaces. The side of a polygon facing into an object is called the back face, the outward side is the front face. Color and other attributes can be set for the back face and the front face separately (OpenGL ARB Guide 2005).

Mathematically, a plane in R^3 is represented as

$$Ax + By + Cz + D = 0$$

A vector space is formed through multiplying the vector (A,B,C) by constants. The normal vector of the polygon contained by the plane represented by the above equation belongs to this vector space. This normal vector is perpendicular to the polygon, and points in a direction from the back face of the polygon to its front face.

Polygon vertices must be specified in a COUNTERCLOCKWISE order as we view the polygon from "outside" or view its front face. The following example demonstrates how to produce the polygon shown in Figure 2.4 in OpenGL

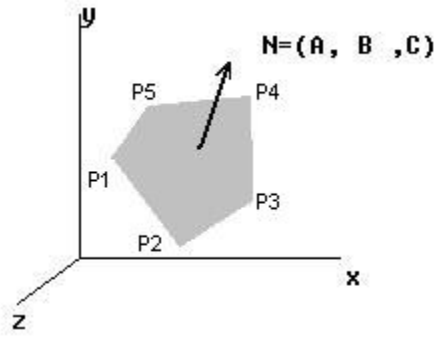

Figure 2.2: The normal vector N of a polygon

```
glBegin(GL_POLYGON)
        glVertex(P1)
        glVertex(P2)
        glVertex(P3)
        glVertex(P4)
        glVertex(P5)
glEnd()
```

2.2.4 GEOMETRIC TRANSFORMATIONS

For computer graphics, geometric transformations such as translation (i.e moving an object), rotation, and scaling are fundamental operations.

2.2.4.1 TRANSLATION

A translation moves an object from one position to another without a change of shape or scale. A point P = (x, y, z) in three dimensional space is shifted to a new location $P^* = (x^*, y^*, z^*)$ by adding shifts dx, dy, dz.

$$\begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_u \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Or

$$P^* = Tp$$

where T is the translation matrix.

In OpenGL, the 4 by 4 translation matrix T is produced by the function call

glTranslate(dx, dy, dy)

In OpenGL, with the following function, we can assign the identity matrix to the current matrix

glLoadIdentity()

2.2.4.2 ROTATION

The easiest three dimensional rotation is to rotate an object around the Cartesian coordinate axes. The transformation matrices are given by the following equations:

around z axis

$$\begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

around x axis

$$\begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

around y axis

$$\begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Or is represented as

$$P^* = R(\alpha).p$$

OpenGL provides the routine

glRotate(alfa, rx, ry, rz)

Here alfa is a rotation angle in degrees, and the r = (rx,ry,rz) defines the orientation of the rotation axis.

2.2.4.3 SCALING

To alter the size of an object, we apply a scaling transformation. A three dimensional scaling operation is performed by multiplying object positions (x, y, z) by scaling factor s_x, s_y and s_z to produce the transformed coordinates (x^*, y^*, z^*).

The transformation matrix is given by

$$\begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Or is represented as

$$P^* = S_p$$

2.3 HAND MOTION RECOGNITION

Gestures are a powerful means of communication among humans. In fact, gesturing is so deeply rooted in our communication that people often continue gesturing when speaking on the telephone.

Hand gestures provide a separate complementary modality to speech for expressing ones ideas. Information associated with hand gestures in a conversation is degree, discourse structure, spatial and temporal structure. So, a natural interaction between humans and computing devices can be achieved by using hand gestures for communication between them.

2.3.1 HAND POSTURE AND GESTURE RECOGNITION

The human hand has a complex anatomical structure consisting of many connected parts and joints, involving complex relations between them providing a total of roughly 27 degrees of freedom (DOFs))(Y. Wu and T.S. Huang 2001).

User Interface development requires a sound understanding of human hand's anatomical structure in order to determine what kind of postures and gestures are comfortable to make. Although hand postures and gestures are often considered identical, the distinctions between them need to be cleared. Hand posture is a static hand pose without involvement of movements.

In vision based hand gesture recognition system, the movement of the hand is recorded by video camera(s). This input video is decomposed into a set of features taking individual frames into account. Some form of filtering may also be performed on the frames to remove the unnecessary data, and highlight necessary components.

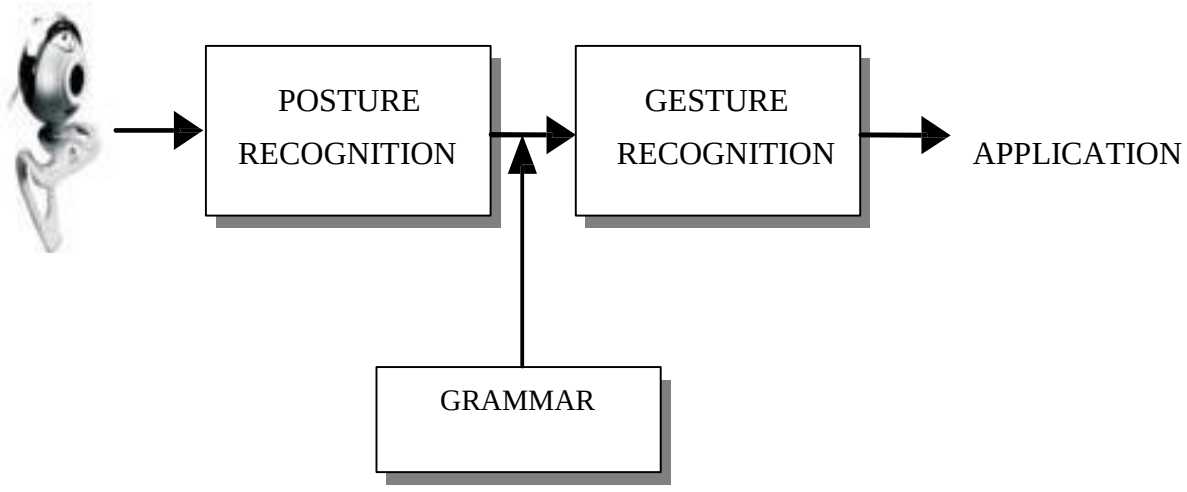

Figure 2.3 Hand Gesture Recognition Process

2.3.2 APPROACHES FOR HAND POSTURE AND GESTURE RECOGNITION

The approaches to Vision based hand posture and gesture recognition can be divided into two categories – 3 D hand model based approaches and appearance based approaches(H. Zhou, T.S. Huang 2003).

2.3.2.1 3 D HAND MODEL BASED APPROACH

Three dimensional hand model based approaches rely on the 3 D kinematic hand model with considerable DOF's, and try to estimate the hand parameters by comparison between the input images and the possible 2 D appearance projected by the 3-D hand model. Such an approach is ideal for realistic interactions in virtual environments. One of the earliest model based approaches to the problem of bare hand tracking was proposed by Rehg and Kanade (JM Rehg, T Kanade 1994). This article describes a model-based hand tracking system, called DigitEyes, which can recover the state of a 27 DOF hand model from ordinary gray scale images at speeds of up to10 Hz. The hand tracking problem is posed as an inverse problem: given an image frame (e.g. edge map) find the underlying parameters of the model. The inverse mapping is non-linear due to the trigonometric functions modeling the joint movements and the perspective image projection. A key observation is that the resulting image changes smoothly as the parameters are changed. Therefore, this problem is a promising candidate for assuming local linearity. Several iterative methods that assume local linearity exist for solving non-linear equations. Upon finding the solution for a frame the parameters are used as the initial parameters for the next frame and the fitting procedure is repeated. The approach can be thought of as a series of hypotheses and tests, where a hypothesis of model parameters at each step is generated in the direction of the parameter space (from the previous hypothesis) achieving the greatest decrease in mis-correspondence. These model parameters are then tested against the image. This approach has several disadvantages that has kept it from real-world use. First, at each frame the initial parameters have to be close to the solution, otherwise the approach is liable to find a suboptimal solution (i.e. local minima). Secondly, the fitting process is also sensitive to noise (e.g. lens aberrations, sensor noise) in the imaging process. Finally, the approach cannot handle the inevitable self-occlusion of the hand.

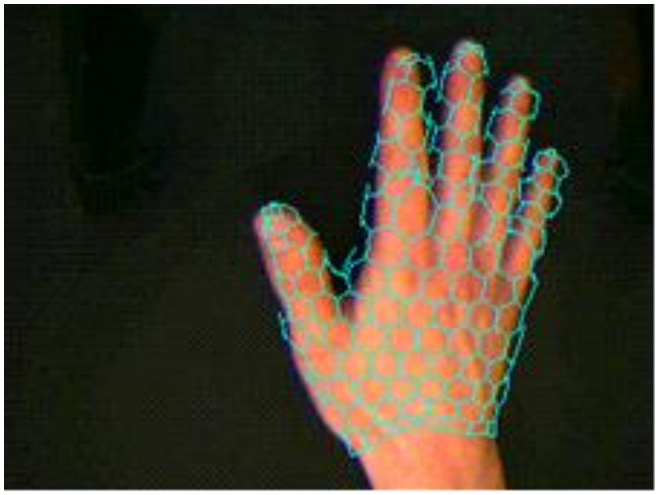

Figure 2.4 Snapshot of 3 D Tracker in action (Pragati G., Naveen A., Sanjeev S. 2009)

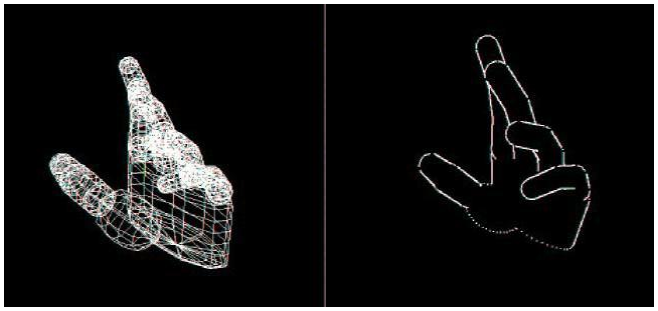

Figure 2.5 The 3 D model (left) and its generated contour (right) (Pragati G., Naveen A., Sanjeev S. 2009)

2.3.2.2 APPEARANCE BASED APPROACHES

Appearance based approaches use image features to model the visual appearance of the hand and compare these parameters with the extracted image features from the video input. Generally speaking, appearance based approaches have the advantage of real time performance due to the easier 2 D image features that are employed.

There have been a number of research efforts on appearance based methods in recent years. A straightforward and simple approach that is often utilized is to look for skin colored regions in the image (Elena Sánchez-Nielsen, Luis Antón-Canalís, Mario Hernández-Tejera 2004)(B. Stenger

2006). Although very popular, this has some drawbacks. First, skin color detection is very sensitive to lighting conditions. While practicable and efficient methods exist for skin color detection under controlled (and known) illumination, the problem of learning a flexible skin model and adapting it over time is challenging. Secondly, obviously, this only works if we assume that no other skin like objects are present in the scene. Lars and Lindberg(Lars Bretzner, Ivan Laptev and Tony Lindeberg, 2002) used scale-space color features to recognize hand gestures. Their gesture recognition method is based on feature detection and user independence, but the authors showed real time application only with no other skin coloured objects present in the scene. So, although, skin color detection is a feasible and fast approach given strictly controlled working environments, it is difficult to employ it robustly on realistic scenes.

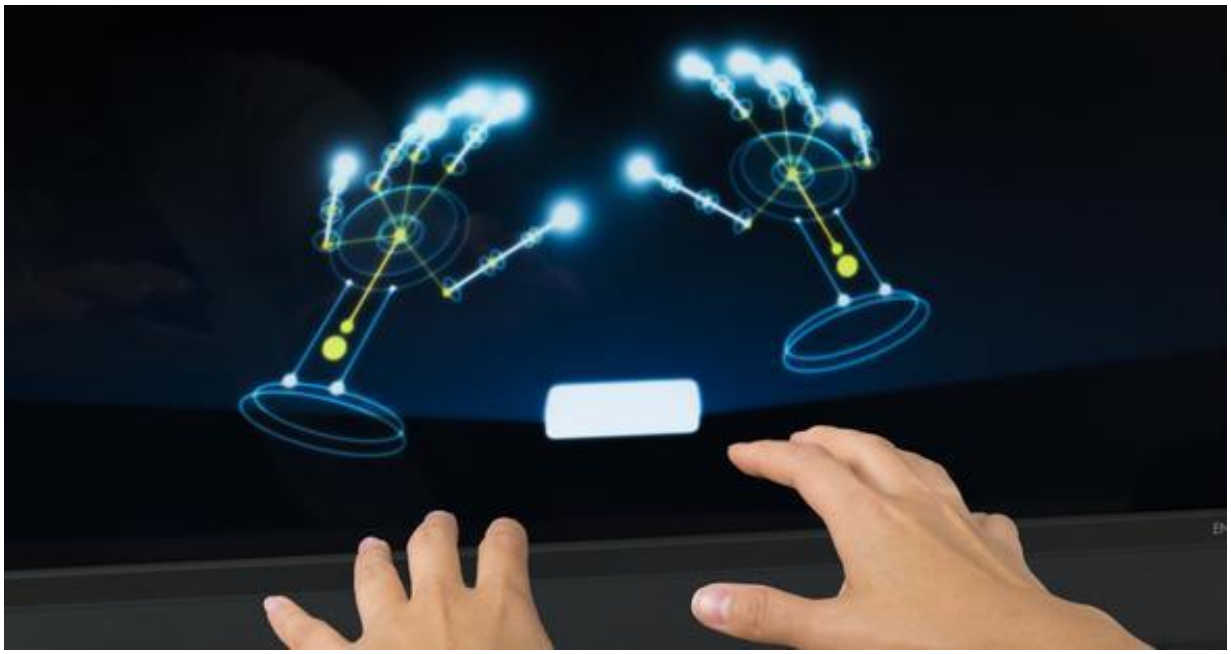

Figure 2.6 A hand motion recognition example

2.4 VISUALIZATION TOOLS FOR CULTURAL OBJECTS

Computer generated 3D graphics applied to draw historical objects for visualization them in last 20 years.We see today wide usage of renders, pictures but trend in visualizing is more in advanced techniques as animations and interactive real-time visualization.

Digital technologies are now mature for producing high quality digital replicas of Cultural Heritage (CH) artefacts. The research results produced in the last decade have shown an impressive evolution and consolidation of the technologies for acquiring high-quality digital 3D models (3D scanning) and for rendering those models at interactive speed. Technology is now mature enough to push us to go beyond the plain visualization of those assets, devising new tools able to extend our insight and intervention capabilities and to revise the current consolidated procedures for CH research and management. The paper presents a few recent experiences where high-quality 3D models have been used in CH research, restoration and conservation.

2.4.1 DIGITAL 3D MODELS - ACQUISITION AND VISUALIZATION

Technologies for the digital sampling of reality appeared around twenty years ago and consolidated in the last decade. The most well known approach is laser-based 3D scanning, but this is not the only approach available today 3D data can be sampled by adopting a scanning device or also by one of the recent image-based approaches.

Initially, the most obvious CH applications of 3D sampled data focused on different incarnations of visualization-oriented applications. Being able to present visually an artwork is valuable for several tasks and to different potential users.

2.4.1.1 VISUALIZATION

We do understand Visualization as communication process. We have data's, information's, knowledge, achievements, many experiences and even vision on side of experts. Visualization is easy understandable way of communication most of that. Some advantages of visualization are; possibilities to display eaxct data, photorealisic visuals, easy to use, wide software offer. Disadvantage of visualization is static presentation.

2.4.1.2 ANIMATION

Animation offers significantly better possibilities to display processes, wider areas as well as provide better description than renders. Technically animation consist of significant number of renders (average 25 – 30 per second) that create effect of continuous movie. Some advantages of animation are; possibilities to display eaxct data, photorealisic visuals,wider display, better effect. Disadvantages of animation are more difficult tool to manage,need to combine with other software.

Figure 2.7 Visualisation of the hypostyle hall of King Raneferef in Abusir (Egypt)

Figure 2.8 Visualisation of the hypostyle hall of King Raneferef in Abusir (Egypt)

2.4.1.3 RECONSTRUCTION

We see reconstruction as advanced visualization. Main difference is that we are not only testing to translate all given inputs into 3D visual but we are using those newly reconstructed reality to verify all idioms. Not only that while changing one or more axioms on digital 3D models it might bring additional learning's. Reconstruction might be used for artifacts, structures, houses, urban areas, events of all kinds etc.

Figure 2.9 Visual reconstruction of early iron age house in East Bogemia

3 SYSTEM SETUP

The system consists some visual datas such as scanned historical objects, their coordinates and color datas in 3D space, some visualization and texture mapping tools, OpenGL multi-platform application programming interface (API), SharpGL is a C# library that allows to use OpenGL in .NET framework.

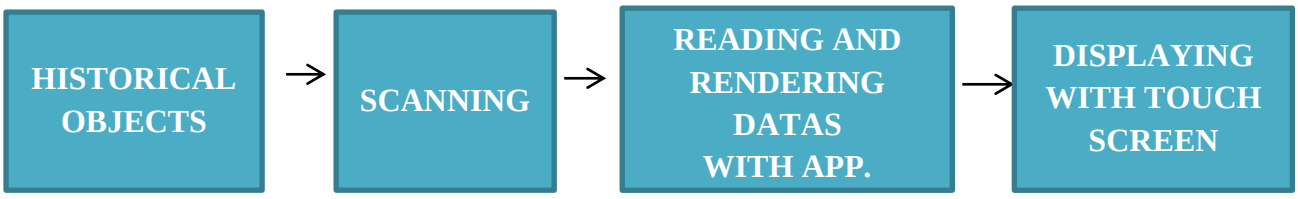

Figure 3.1 System Workflow

3.1 HISTORICAL OBJECTS

In this thesis is used Dolmabahçe Palace Saltanat Door for historical monument. This door is between two high wall with a long corridor. It has two leaf which are maden iron. It has enclosed in a large medallion panel and twin towers with inside and outside. And it is create interest for many tourist. All of the Dolmabahçe Palace Saltanat Door is scanned and collected some huge datas.

Figure 3.2 Dolmabahçe Palace Saltanat Door

3.2 GETTING DATAS

In this section we did some preparation process such as scanning,transferring,alignment, triangulation and crossing. For these processings we used some tools such as Leica Cyclone which is a laser scanner, a computer installed with Cyclone software, a point program which is written by Masetawa Kagesawa.

3.2.1 SCANNING

With Leica Cyclone laser scanner we can do good scanning in 360° horizantal and 270° vertical area or targeting only one object we can handle different resolutions.

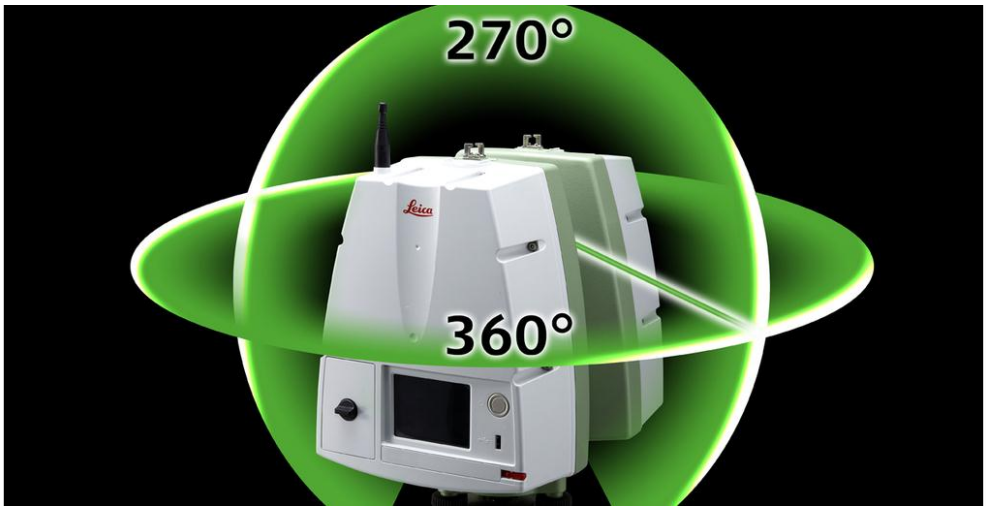

Figure 3.3 Scanning With Full Angle

We can do scanning process with laser scanner's little screen for an obvious area but this is very hard with little screen. Instead of this we can connect with a computer which is installed with Cyclone software and control to the scanning.

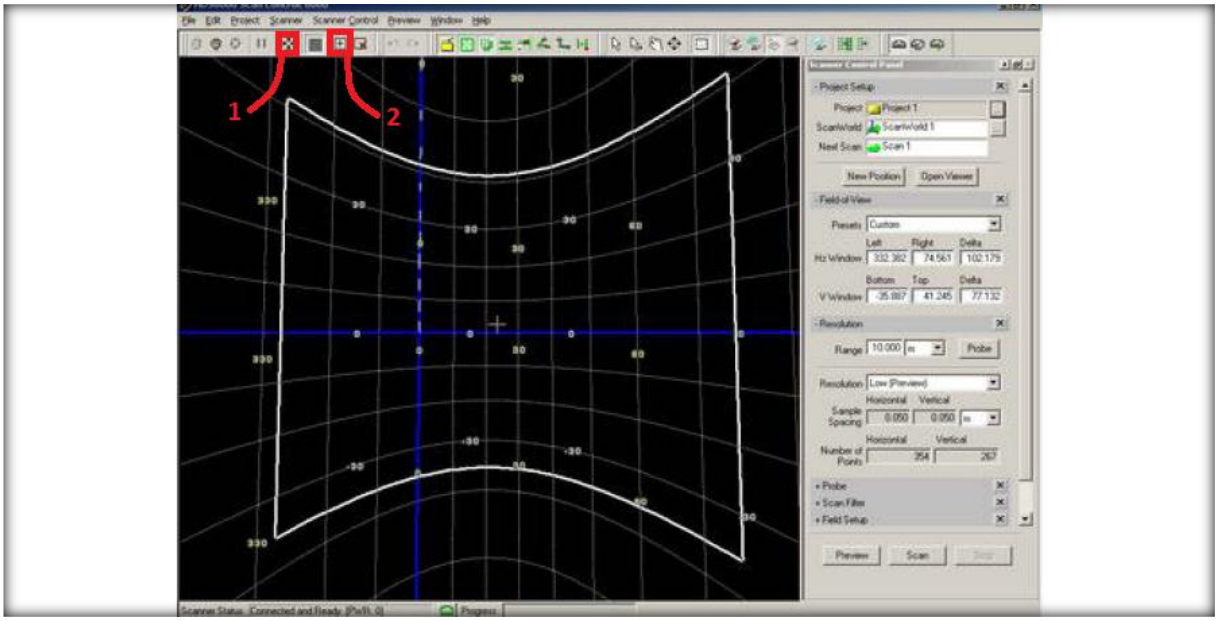

Figure 3.4 Taking a panoramic shot

We can take a panoramic scan of all area by using Cyclone Software in computer. Also we can choose any resolution for our scanning process.

3.2.2 TRANSFERRING

We must transfer scanned point clouds with cyclone software and it must be ptx file format. Because the alignment process working only ply files. So we changed plx files to ply file after the scanning.

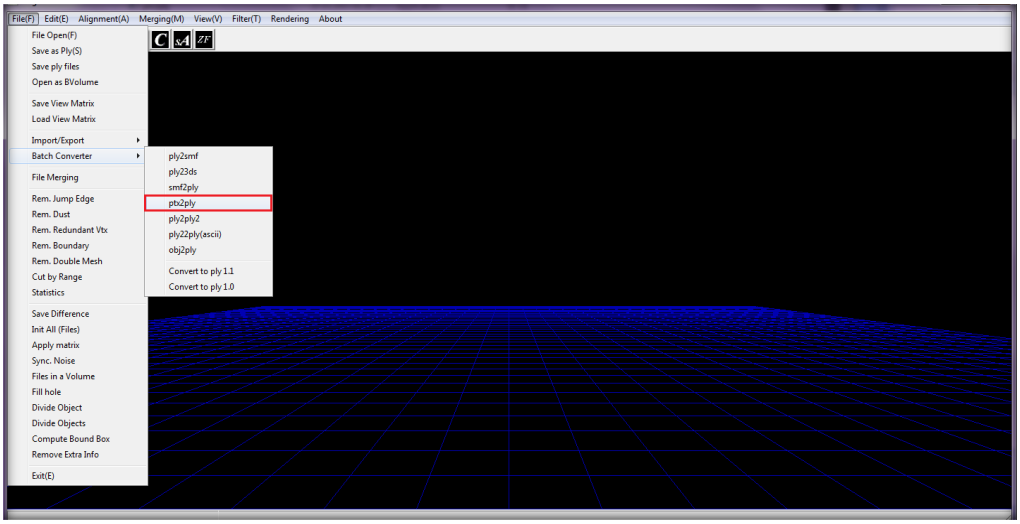

Figure 3.5 Exporting ply files

3.2.3 ALIGNMENT

Alignment is a process to combine two point clouds with only one coordinat system which they have same area.

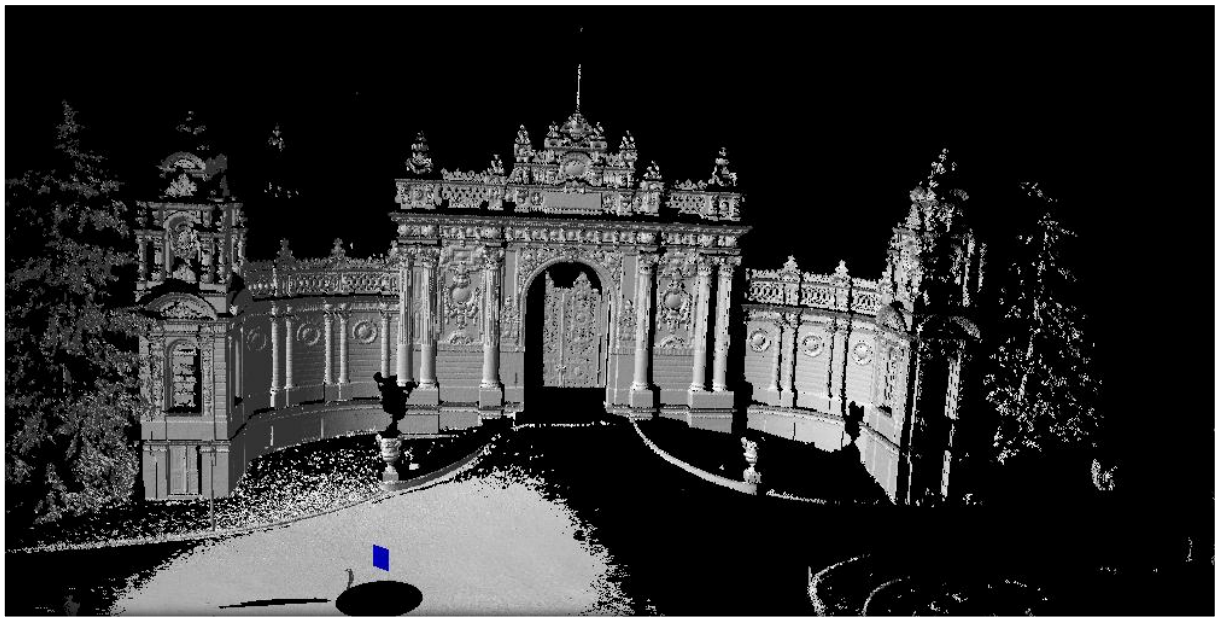

Figure 3.6 A scanning

Alignment make among two point clouds which have common intersecting points and these points are ply extension.

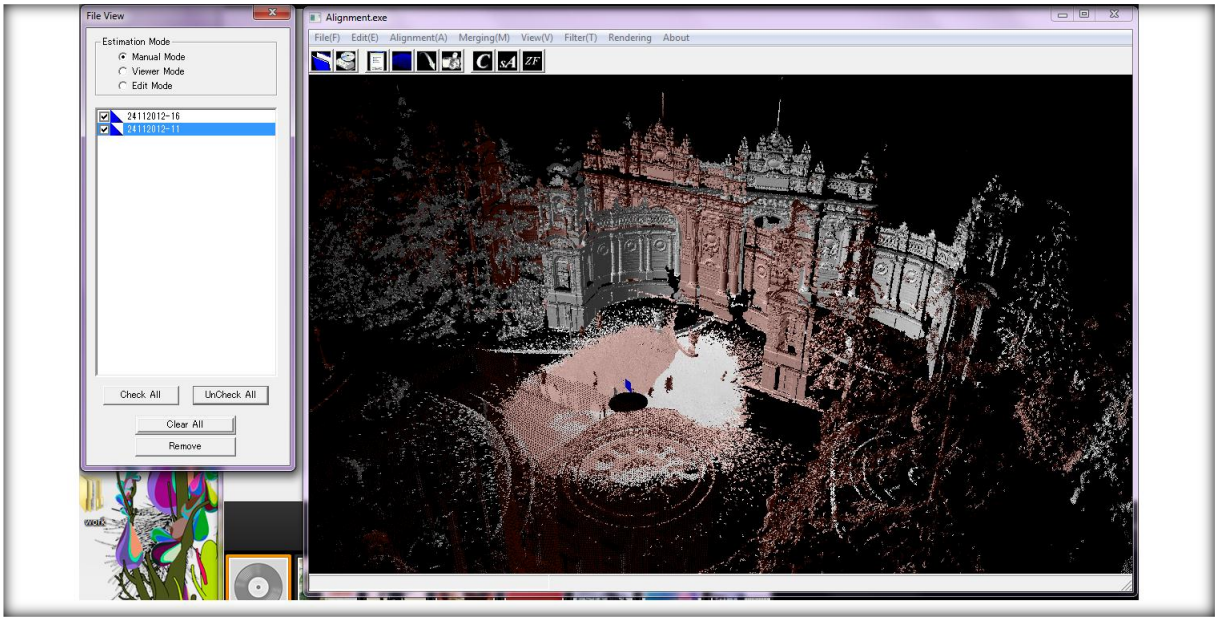

Figure 3.6 Alignment of a point cloud

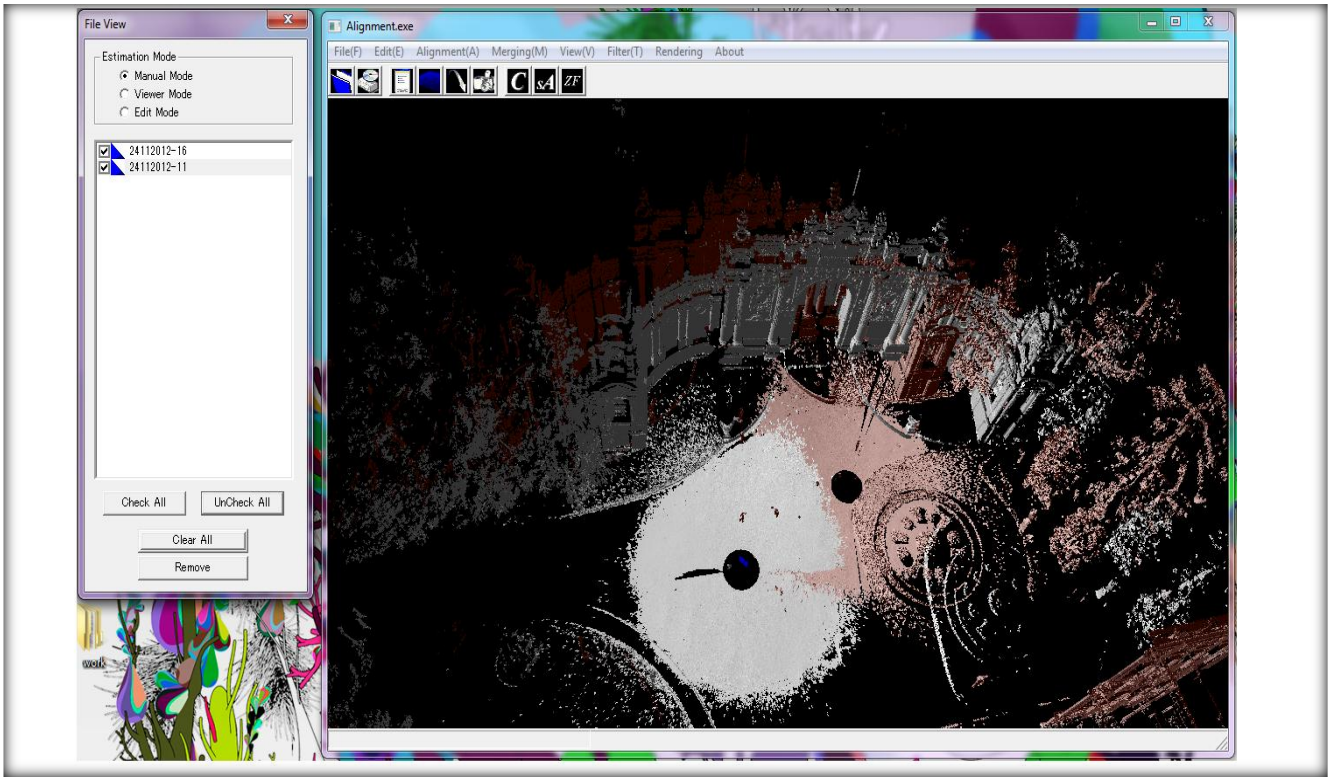

Figure 3.7 Approaching point clouds to each other

3.2.4 TRIANGULATION

After the alignment of some scanned images which are different resolution, we did triangulation with the best fit for surface but cyclone is not the best for this work. 3D Reshaper is choicen for the better triangulation process.

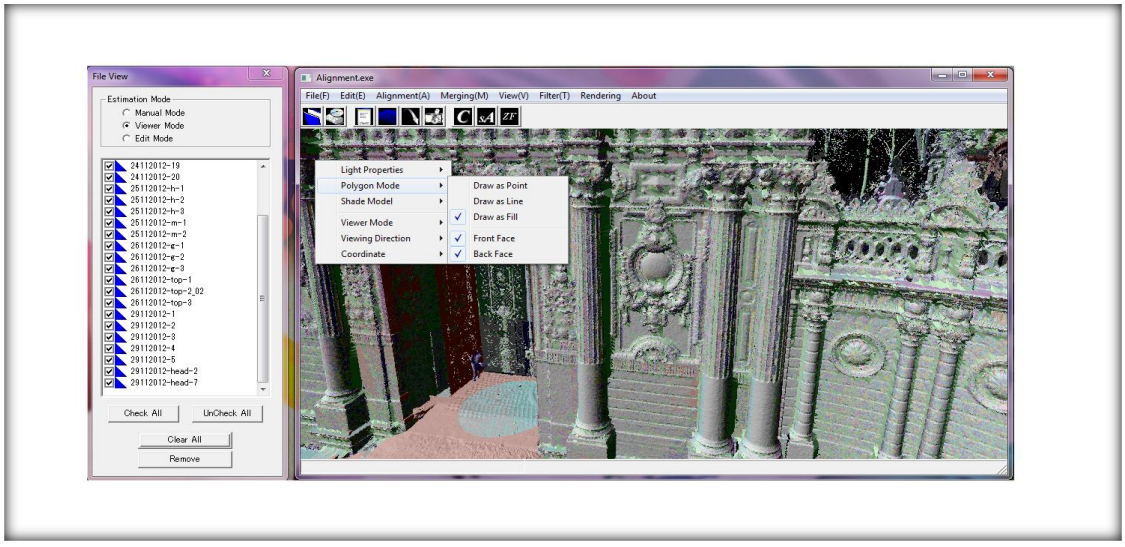

Figure 3.8 Triangulation

3.2.5 CROSSING

Crossing is analysis of an area of point clouds or analysis of an area of a model.

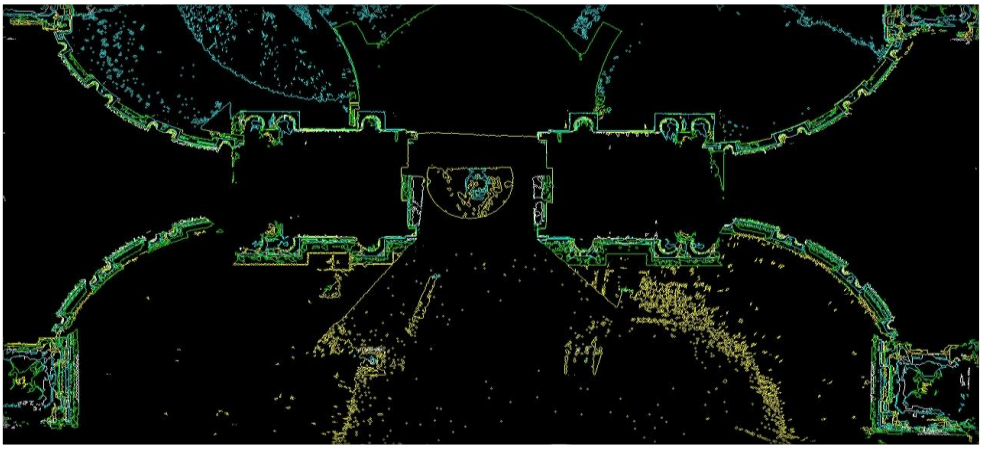

Figure 3.9 A crossing

3.3 DATA STRUCTURES

All of the Dolmabahçe Palace Saltanat Door is scanned with a scanner and stored in a Polygon File (PLY) format such as we explained previous section. Polygon File is a file format storing graphical objects that are described as a collection of polygons. This format was developed at Stanford University.To use this format is so simple and easy to implement. The file format has two sub-formats: an ASCII representation for easily getting started, and a binary version for compact storage and for rapid saving and loading.

The PLY format describes an object as a collection of vertices, faces and other elements, along with properties such as color and normal direction that can be attached to these elements. A PLY file contains the description of exactly one object. Sources of such objects include: hand-digitized objects, polygon objects from modeling programs, range data, triangles from marching cubes (isosurfaces from volume data), terrain data, radiosity models. Properties that might be stored with the object include: color, surface normals, texture coordinates, transparency, range data confidence, and different properties for the front and back of a polygon.

The PLY format is NOT intended to be a general scene description language, a shading language or a catch-all modeling format. This means that it includes no transformation matrices, object instantiation, modeling hierarchies, or object sub-parts. It does not include parametric patches, quadric surfaces, constructive solid geometry operations, triangle strips, polygons with holes, or texture descriptions (not to be confused with texture coordinates, which it does support!).

A typical PLY object definition is simply a list of (x,y,z) triples for vertices and a list of faces that are described by indices into the list of vertices. Most PLY files include this core information. Vertices and faces are two examples of "elements", and the bulk of a PLY file is its list of elements. Each element in a given file has a fixed number of "properties" that are specified for each element. The typical information in a PLY file contains just two elements, the (x,y,z) triples for vertices and the vertex indices for each face. Applications can create new properties that are attached to elements of an object. For example, the properties red, green and blue are commonly associated with vertex elements. New properties are added in such a way that old programs do not break when these new properties are encountered. Properties that are not understood by a program can either be carried along uninterpreted or can be discarded. In addition, one can create a new element type and define

the properties associated with this element. Examples of new elements are edges, cells (lists of pointers to faces) and materials (ambient, diffuse and specular colors and coefficients). New elements can also be carried along or discarded by programs that do not understand them.

This is the structure of a typical PLY file:

```
Header
Vertex List
Face List
(lists of other elements)
```

The header is a series of carriage-return terminated lines of text that describe the remainder of the file. The header includes a description of each element type, including the element's name (e.g. "edge"), how many such elements are in the object, and a list of the various properties associated with the element. The header also tells whether the file is binary or ASCII. Following the header is one list of elements for each element type, presented in the order described in the header.

Below is the complete ASCII description for a cow.

```
ply
format ascii 1.0           { ascii/binary, format version number }
comment this file is a cube{ comments keyword specified }
element vertex 8           { the number of vertex }
property float x           { float "x" coordinate vertex }
property float y           { y coordinate is also a vertex }
property float z           { z coordinate, too }
element face 6             { 6 "face" elements in the file }
property list uchar int vertex_index { "vertex_indices" }
end_header                 { delimits the end of the header }
0 0 0                      { start of vertex list }
0 0 1
0 1 1
0 1 0
1 0 0
1 0 1
1 1 1
1 1 0
4 0 1 2 3                  { start of face list }
4 7 6 5 4
4 0 4 5 1
4 1 5 6 2
4 2 6 7 3
4 3 7 4 0
```

3.4 APPLICATION

After the scanning and stored all of datas in the PLY file. We read all of data and visualize in our program. We develop our program with C#.Net developing language. Our application is a Windows Form Application. We used .NET Framework version 4.0 for develop this program. We can read both formats of PLY file such as ASCII and Binary. So we can draw any PLY file with this application except huge PLY files. For drawing process of 3D object we used SharpGL which is a C# library that allows to use OpenGL in .NET framework based application. Also our system have some visaul properties such as zoom-in, zoom-out, translation, rotation and multi touch controlling. Also application shows draw time and number of frame rate per a second. We can use our applicaiton any computer with Windows operation system.

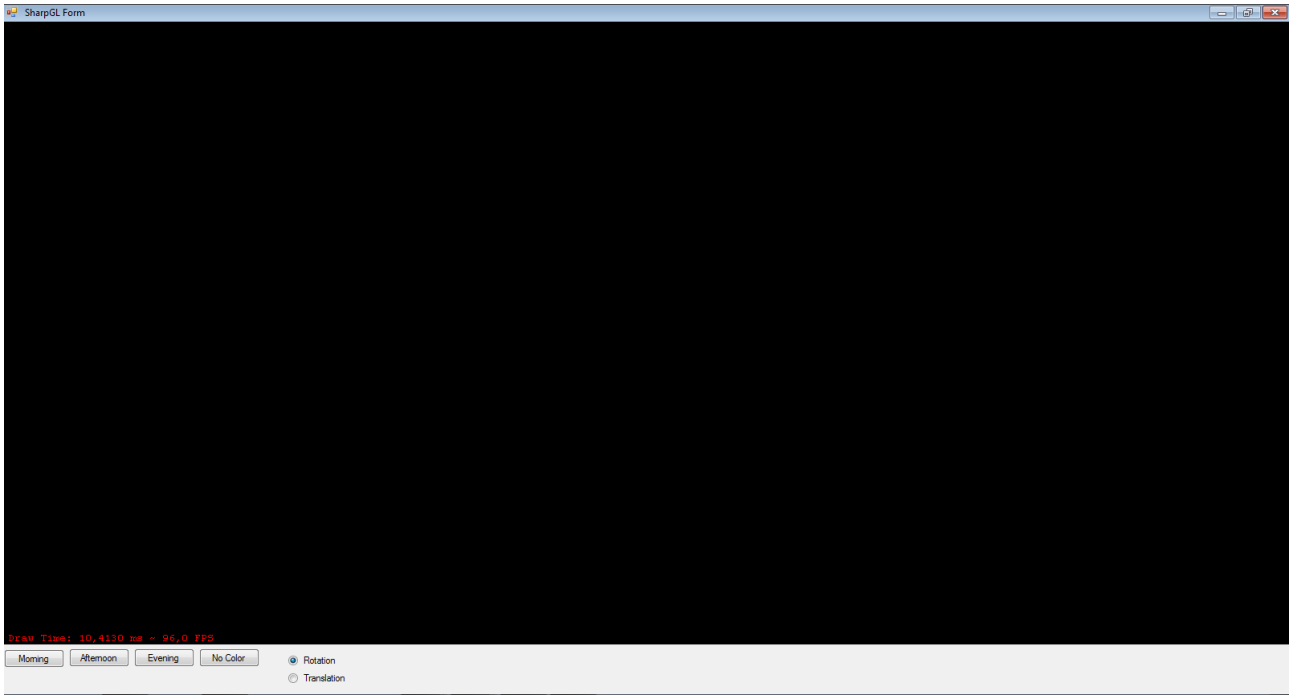

Figure 3.10 A screen of application

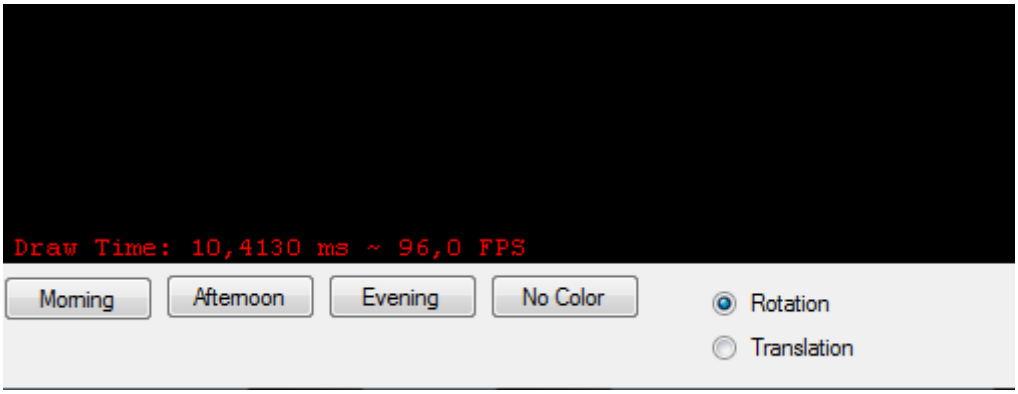

Figure 3.11 Application Menu

3.5 TOUCH SCREEN

We used Viewsonic TD220 for visualize and touch control 3D object. This screen is a multi-touch Full HD LED backlit monitor with 1920x1080 resolution. It has integrated speakers, Digital Visual Interface(DVI) & Video Graphics Array(VGA) inputs. Also it supports the Universal Serial Bus(USB) and Human Interface Device(HID) touch driver .We shows visual object with this multi touch screen.

4 RENDERER

Until now we describe to scanning process of historical object and keeping the datas of object in a PLY file. In this section we explain how to work our program for reading datas and visualizing objects. Until the drawing process we do not use any OpenGL code.

4.1 READING

Normally the PLY Files have two format such as ASCII and Binary. We did two different read method for these two different file format. Reading the ASCII file format is easy. For reading PLY wih ASCII format we use ReadAllLine() method which is reading all lines and keeping them in a list. After that we keep going to the read process line by line. Getting the number of vertex and number of faces from header section in the file. We holds the number of vertex in vertexCount and number of faces in faceCounts. After that we add all vertices in an array a loop. Also we do same thing for faces. After that we do normalization for vertices.

Our readAsci() method on below;

```
private void readAsci()
        {

            List<String> lines = File.ReadAllLines(filePath).ToList<String>();

            for (int i = 0; i < lines.Count; i++)
            {

                if (lines[i].StartsWith("format binary_little_endian 1.0"))
                {
                    fileFormat = 1;
                    break;
                }
            }

            for (int i = 0; i < lines.Count; i++)
            {
                if (lines[i].StartsWith("element vertex"))
                {
                    vertexCount = Convert.ToInt32(lines[i].Replace("element vertex",
"").Trim());
                    break;
                }
            }
```

```
            for (int i = 0; i < lines.Count; i++)
            {
                if (lines[i].StartsWith("element face"))
                {
                    faceCount = Convert.ToInt32(lines[i].Replace("element face",
"").Trim());
                    break;
                }
            }

            int startPoint = 0;
            for (int i = 0; i < lines.Count; i++)
            {
                if (lines[i].StartsWith("end_header"))
                {
                    startPoint = i + 1;
                    break;
                }
            }

            vertexX = new double[vertexCount];
            vertexY = new double[vertexCount];
            vertexZ = new double[vertexCount];

            vertex1 = new int[faceCount];
            vertex2 = new int[faceCount];
            vertex3 = new int[faceCount];

            for (int i = startPoint; i < startPoint + vertexCount; i++)
            {
                string[] temp = lines[i].Split(' ');
                vertexX[i - startPoint] = Convert.ToDouble(temp[0].Replace(".", ","));
                vertexY[i - startPoint] = Convert.ToDouble(temp[1].Replace(".", ","));
                vertexZ[i - startPoint] = Convert.ToDouble(temp[2].Replace(".", ","));
            }

            startPoint += vertexCount;
            for (int i = startPoint; i < startPoint + faceCount; i++)
            {
                string[] temp = lines[i].Split(' ');
                vertex1[i - startPoint] = Convert.ToInt32(temp[1]);
                vertex2[i - startPoint] = Convert.ToInt32(temp[2]);
                vertex3[i - startPoint] = Convert.ToInt32(temp[3]);
            }

            double smallest=0;
            double biggest=0;
            for (int i = 0; i < vertexY.Length; i++)
            {
                if (vertexY[i]<smallest)
                {
                    smallest = vertexY[i];
                }

                if (vertexY[i] > biggest)
                {
                    biggest = vertexY[i];
                }
            }
```

```
            double difference=(smallest+biggest)/2.0;
            for (int i = 0; i < vertexY.Length; i++)
            {
                vertexY[i] -= difference;
            }
        }
```

Also we read binary files with readLittleEndian() method but there is a differance. The binary files not like an ascii file. Binary files are read with BinaryReader. In this method we read all information in the binary file wirh BinaryReader().

Our readLittleEndian() method on below;

```
private void readLittleEndian()
        {
            FileStream fs = new FileStream(filePath, FileMode.Open);
            BinaryReader br = new BinaryReader(fs);

            char temp;
            string header="";
            string dnm = br.ReadString();

            while (true)
            {
                try
                {
                    temp = br.ReadChar();
                }
                catch (EndOfStreamException)
                {

                    break;
                }

                header += temp;
                if (header.EndsWith("end_header"))
                {
                    break;
                }

            }

            br.ReadChar();

            List <String> list1 = header.Split('\n').ToList<String>();

            foreach (String str in list1)
            {
                if (str.ToLower().StartsWith("element vertex"))
                {
                    vertexCount = Convert.ToInt32(str.Split(' ')[2]);
                }

                if (str.ToLower().StartsWith("element face"))
                {
                    faceCount = Convert.ToInt32(str.Split(' ')[2]);
```

```
        }

    }

    vertexX = null;
    vertexY = null;
    vertexZ = null;

    vertex1 = null;
    vertex2 = null;
    vertex3 = null;

    GC.Collect();

    vertexX = new double[vertexCount];
    vertexY = new double[vertexCount];
    vertexZ = new double[vertexCount];

    vertex1 = new int[faceCount];
    vertex2 = new int[faceCount];
    vertex3 = new int[faceCount];

    int ctr=0;

    while (true)
    {
        if (ctr>=vertexCount)
        {
            break;
        }

        vertexX[ctr] = br.ReadSingle();
        vertexY[ctr] = br.ReadSingle();
        vertexZ[ctr] = br.ReadSingle();

        ctr++;
    }

    ctr = 0;
    while (true)
    {
        if (ctr >= faceCount)
        {
            break;
        }

        br.ReadChar();
        vertex1[ctr] = br.ReadInt32();
        vertex2[ctr] = br.ReadInt32();
        vertex3[ctr] = br.ReadInt32();

        ctr++;
    }

    br.Close();
    fs.Close();
}
```

In the beginnings we used readLittleEndian() for read binary PLY files. After that we use readFile() method for read binary files. But there is some differances between both method. Last method reads color information from the file and keeping them in the colors[] array.

The readFile() method in on the below;

```
private void readFile()
        {


            FileStream fs = new FileStream(filePath, FileMode.Open);
            BinaryReader br = new BinaryReader(fs);

            char temp;
            string header = "";
            string dnm = br.ReadString();

            while (true)
            {
                try
                {
                    temp = br.ReadChar();
                }
                catch (EndOfStreamException)
                {

                    break;
                }

                header += temp;
                if (header.EndsWith("end_header"))
                {
                    break;
                }

            }

            br.ReadChar();

            List<String> list1 = header.Split('\n').ToList<String>();

            foreach (String str in list1)
            {
                if (str.ToLower().StartsWith("element vertex"))
                {
                    vertexCount = Convert.ToInt32(str.Split(' ')[2]);
                }

                if (str.ToLower().StartsWith("element face"))
                {
                    faceCount = Convert.ToInt32(str.Split(' ')[2]);
                }

            }

            vertexX = null;
            vertexY = null;
```

```
        vertexZ = null;
        colors  = null;
        vertex1 = null;
        vertex2 = null;
        vertex3 = null;

        GC.Collect();

        vertexX = new double[vertexCount];
        vertexY = new double[vertexCount];
        vertexZ = new double[vertexCount];

        colors = new Color[vertexCount];

        vertex1 = new int[faceCount];
        vertex2 = new int[faceCount];
        vertex3 = new int[faceCount];

        int ctr = 0;

        Color color = new Color();
        while (true)
        {
            if (ctr >= vertexCount)
            {
                break;
            }

            vertexX[ctr] = br.ReadSingle();
            vertexY[ctr] = br.ReadSingle();
            vertexZ[ctr] = br.ReadSingle();

            br.ReadSingle();
            colors[ctr] = Color.FromArgb(br.ReadByte(), br.ReadByte(), br.ReadByte());
            br.ReadByte();
            ctr++;
        }

        ctr = 0;
        while (true)
        {
            if (ctr >= faceCount)
            {
                break;
            }

            br.ReadChar();
            vertex1[ctr] = br.ReadInt32();
            vertex2[ctr] = br.ReadInt32();
            vertex3[ctr] = br.ReadInt32();

            ctr++;
        }

        br.Close();
        fs.Close();

    }
```

4.2 DRAWING

In the previous section we explained the reading process of our application and for reading we use only C#.NET codes. But in the drawing section we also use OpenGL codes and methods. For these processes we use SharpGL extension on the Microsoft Visual Studio. We explain before what is the SharpGL.

Before the explain drawing process we need to say why we use DisplayList() Method. DisplayList() is a method which is a group of OpenGL commands that have been stored (compiled) for later execution. Once a display list is created, all vertex and pixel data are evaluated and copied into the display list memory on the server machine. It is only one time process. After the display list has been prepared (compiled), you can reuse it repeatedly without re-evaluating and re-transmitting data over and over again to draw each frame(OpenGL Red Book Chapter 7). For use DisplayList() method we create a CreateList() method. And in this method firstly we generate a displayList(), after that foe define a list we use NewList() and EndList(), then we give every vertices coordinates and colors for faces in a loop. And this loop repeads to the number of faceCount.

CreateDisplayList() method is on the below,

```
private void CreateDisplayList(OpenGL gl)
        {
            //  Create the display list.
            displayList = new DisplayList();

            //  Generate the display list and
            displayList.Generate(gl);
            displayList.New(gl, DisplayList.DisplayListMode.CompileAndExecute);

            //  Draw the axies.
            gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT);



            gl.NewList(1, OpenGL.GL_COMPILE);
            gl.Begin(OpenGL.GL_TRIANGLES);


            readFile();
            GC.Collect();

            for (int i = 0; i < faceCount; i++)
            {
                if (time==4)
                    gl.Color(1.0f, 0.0f, 0.0f);
                else
```

```
                    gl.Color(colors[vertex1[i]].R / 255.0, colors[vertex1[i]].G / 255.0,
colors[vertex1[i]].B / 255.0);
                gl.Vertex(vertexX[vertex1[i]], vertexY[vertex1[i]], vertexZ[vertex1[i]]);

                if (time==4)
                    gl.Color(0.0f, 1.0f, 0.0f);
                else
                    gl.Color(colors[vertex2[i]].R / 255.0, colors[vertex2[i]].G / 255.0,
colors[vertex2[i]].B / 255.0);
                gl.Vertex(vertexX[vertex2[i]], vertexY[vertex2[i]], vertexZ[vertex2[i]]);

                if(time==4)
                    gl.Color(0.0f, 0.0f, 1.0f);
                else
                    gl.Color(colors[vertex3[i]].R / 255.0, colors[vertex3[i]].G / 255.0,
colors[vertex3[i]].B / 255.0);
                gl.Vertex(vertexX[vertex3[i]], vertexY[vertex3[i]], vertexZ[vertex3[i]]);
            }
            gl.End();
            gl.EndList();

            //  End the display list.
            displayList.End(gl);
        }
```

After the creating list with CreateDisplayList() method we use that method in the openGLControl_OpenGLDraw() methods. This method is a sharpGL method. The drawing process is doing on here. For drawing we calls to the CreateDisplayList() method.

openGLControl_OpenGLDraw() is on the below;

```
private void openGLControl_OpenGLDraw(object sender, PaintEventArgs e)
             {

                    //  Get the OpenGL object.
                    OpenGL gl = openGLControl.OpenGL;

            //uint ID = gl.CreateShader(OpenGL.GL_VERTEX_SHADER);

                    //  Clear the color and depth buffer.
                    gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT | OpenGL.GL_DEPTH_BUFFER_BIT);

                    //  Load the identity matrix.
                    gl.LoadIdentity();

            /*Convert to radiant*/
            float radAngle = (difX / 180.0f) * 3.1415f;

            //  Use the 'look at' helper function to position and aim the camera.
            gl.LookAt(0, 0.0, 0.0+zoonIn, 0, 0.0, 0, 0, 1, 0);



            gl.Translate(transX, transY, 0);
            gl.Rotate(rotationY, 1, 0, 0);
            gl.Rotate(-rotationX, 0, 1, 0);
```

```
gl.Rotate(0, 0, 0, 1);


if (displayList == null && time != 0 && timeChanged)
{
    CreateDisplayList(gl);
    timeChanged = false;
}
else if(displayList !=null)
    displayList.Call(gl);


rotationX += difX*2;
rotationY += difY*2;

transX += difX2 / 10f;
transY += difY2 / 10f;



difX = 0;
difY = 0;

difX2 = 0;
difY2 = 0;
  }
```

5 TOUCH SCREEN INTERFACE

In our project we also use hand motion recognition. The visualization program which we develop for show 3D visual model of Dolmabahce Saltanat Door has hand motion recognition system. The system was written with C#.NET software language. The 3D visual object in the screen was contolled hand motions on the multi touch screen and we can rotate and translate our object with one finger movement, we can zoom-in and zoom-out with two fingers. Also we can zoom-in with double clicking of one finger on the touch screen.

The software was developed with Windows Forms tool in Microsoft Visual Studio Editor 2012. Firstly some mouse event was used like a hand motion model. After that this movements was tested on the multi touch screen. And we observed that all of those movements are very successfully. Four different mouse events such as mouse move event, click event,double click event and mouse whell event was used on this system. Secondly our mouse events was used for hand motions on multi touch screen.

5.1 ZOOM-IN MOVEMENT

Zoom in movement is realized two different mouse events. First one is double click event and the second one is roolling to the mouse whell. When we click to the mouse left click two times or rolling to the mouse whell, zoom in movement is become true. If we want to use this event on the touch screen we must touch two times on the screen rapidly. Also if we touch to the screen with two fingers and we increase to angle between touched two fingers we can handle to the zoom in event.

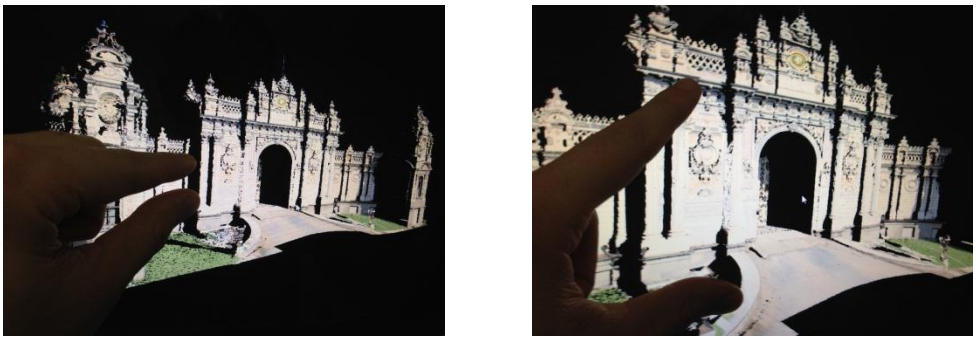

Figure 5.1 Zoom-In Movement

5.2 ZOOM-OUT MOVEMENT

Zoom out movement works similar of zoom in movement but this time if we rool to the mouse well contrary of the zoom in movement way or if we touch to the screen with two fingers and we decrease to angle between touched two fingers we can handle to the zoom out event.

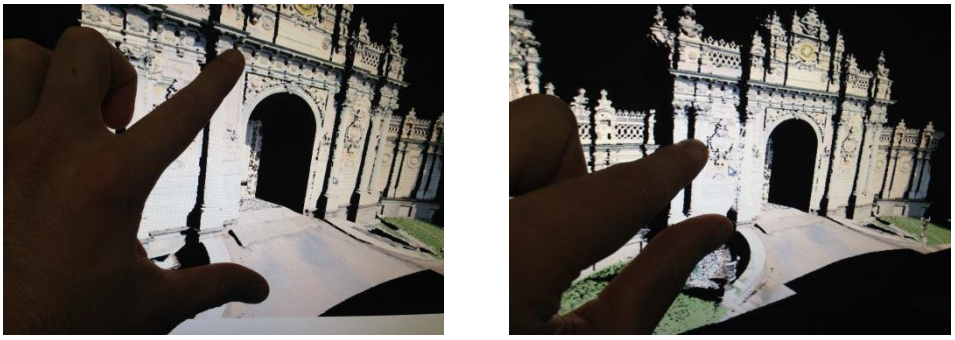

Figure 5.2 Zoom-Out Movement

Our system automatically recognize to the mouse whell movement empirically two finger movement on the touch screen for zoom in or zoom out movement.

5.3 ROTATION MOVEMENT

Rotation movement is depend of the mouse down and mouse move events. When we press left button of the mouse and hold the mouse cursor on the clicked image or model and move the mouse, it begin to the rotate. Also we can this movement with the finger and hand movements. Firstly we touch the model on the touch screen with one finger and then begin to the move of hand with a line on the screen our model rotates. But we do not cut to touch the screen when our hand movement. Our system automatically recognize mouse down and mouse move events together empirically one finger movement on the touch screen for rotation movement.

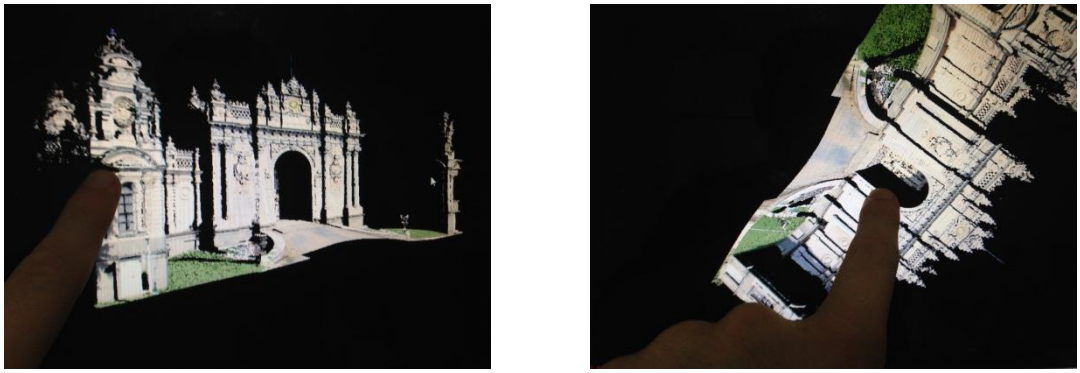

Figure 5.3 Rotation Movement

There is a some mathematical formulas on this property. When we down th mouse left button or touch the screen we keep the coordinate of the touch point on the screen. The coordinate of the first point on the X-axis is oldX and the coordinat of the second point on the Y-axis is oldY.And they are equal to the coordinates of the first touch of the finger.

```
bool isMouseDown = true; //A boolean variable

            oldX = e.X;  //first coordinates of the X
            oldY = e.Y;  //first coordinates of the Y
```

After that we again take the coordinates in the mouseMove event. But if isMouseDown is true then we doing this. If our movement is stop then we take the last coordinates. After that we compute to the differance between last coordinates and first coordinates such as

difX = oldX –e.X and difY = oldY-e.Y.

```
if (isMouseDown)
            {
               difX = oldX - e.X;//Differance between last coordinates and first coordinates
               difY = oldY - e.Y;//Differance between last coordinates and first coordinates
               oldX = e.X; //equivalent to new X position to the old X position
               oldY = e.Y; //equivalent to new Y position to the old Y position
            }
```

If we up finger of the screen then in the mouseUp events we equals to the isMouseDown to false. And oldX=0 and oldY=0.

```
isMouseDown = false;

            difX = 0;
            difY = 0;
```

Finally in the draw process we compute to the radiant angle for rotation function.

```
float radAngle = (difX / 180.0f) * 3.1415f; //Converting our angle to the radiant

gl.Rotate(rotationY, 1, 0, 0); //Rotation on the X axis
gl.Rotate(-rotationX, 0, 1, 0); //Rotation on the Y axis
gl.Rotate(0, 0, 0, 1);  //Rotation on the Z axis
```

5.4 TRANSLATION MOVEMENT

Actually for the translation movement we use same mouse events and finger movement with the rotation. The interface have two checkbox. First one is rotation and the second one is translation. If user choice to the rotation, one finger movement on the screen makes rotation movement, if user choice to the translation button it is makes translation movement.

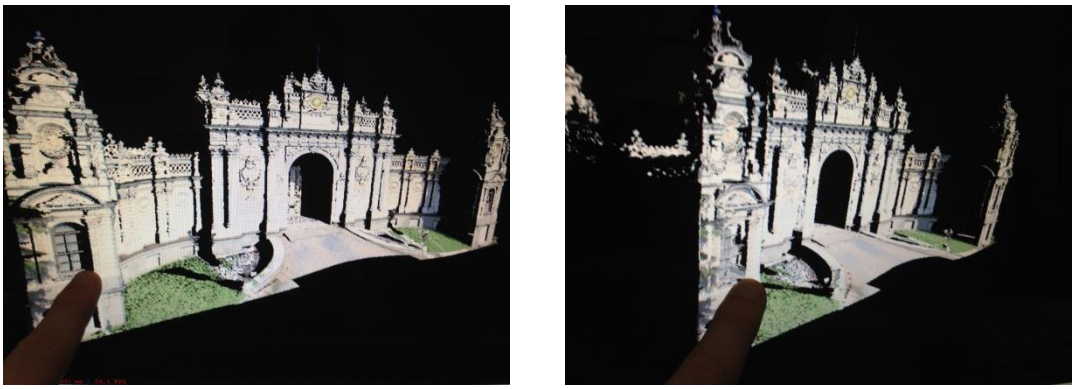

Figure 5.4 Translation Movement

In the translation movement we use oldX2 and oldY2 instead of oldX and oldY.

MouseDown or one finger touch on the screen method for rotation and translation on below,

```
private void openGLControl_MouseDown(object sender, MouseEventArgs e)
        {
            isMouseDown = true;

            if (radio)
            {
                oldX = e.X;
                oldY = e.Y;
            }
            else
            {
                oldX2 = e.X;
                oldY2 = e.Y;
```

```
            }
        }
```

MouseMove or one finger move on the screen method for rotation and translation on below,

```
private void openGLControl_MouseMove(object sender, MouseEventArgs e)
        {

            if (isMouseDown)
            {
                if (radio)
                {
                    difX = oldX - e.X;
                    difY = oldY - e.Y;
                    oldX = e.X;
                    oldY = e.Y;
                }
                else
                {
                    difX2 = oldX2 - e.X;
                    difY2 = oldY2 - e.Y;
                    oldX2 = e.X;
                    oldY2 = e.Y;
                }

            }
        }
```

MouseUp or one finger up from the screen method for rotation and translation on below,

```
       private void openGLControl_MouseUp(object sender, MouseEventArgs e)
        {
            isMouseDown = false;
            if (radio)
            {
                difX = 0;
                difY = 0;
            }
            else
            {
                difX2 = 0;
                difY2 = 0;
            }
        }
```

Finally in the draw process we we describe two value such as transX and transY. They are the value of how much we translate the object on the X and Y axis. And the values of transX and transY are;

```
transX += difX2 / 10f;
transY += difY2 / 10f;
```

So the translate function in of drawing is;

```
gl.Translate(transX, transY, 0);
```

6 EXPERIMENTAL RESULTS

In this thesis 5 different ply file is tested and observed all of results. Firstly 3D crow object was drawed and rotated with multi touch screen and noted results. After that Saltanat door was drawed, applied rotating process and noted results. Than same door was processed both drawing and rotating in a 3 different time such as morning,afternoon and evening and observed angle of sun light and shadows of this 3 different situation. Finally all of this process were tested 20 different people. A survey was applied for this peoples and analysied all results.

6.1 FIRST APROACH AND RESULTS

At the beginnig of this project using tiny datas about 181 Kb for testing performance. The cow.ply (Polygon PLY file) has 2903 vertex and 5804 faces. Firstly a 3-D cow object is drawed and observed to exchange of computer performance. In the processing time the CPU is increased 10% and the pysical memory reached about 100 megabyte. After the drawing process, 3-D cow object was rotated around of y axis or x axis with multi touch screen. In this time the cow object was rotating very slowly.

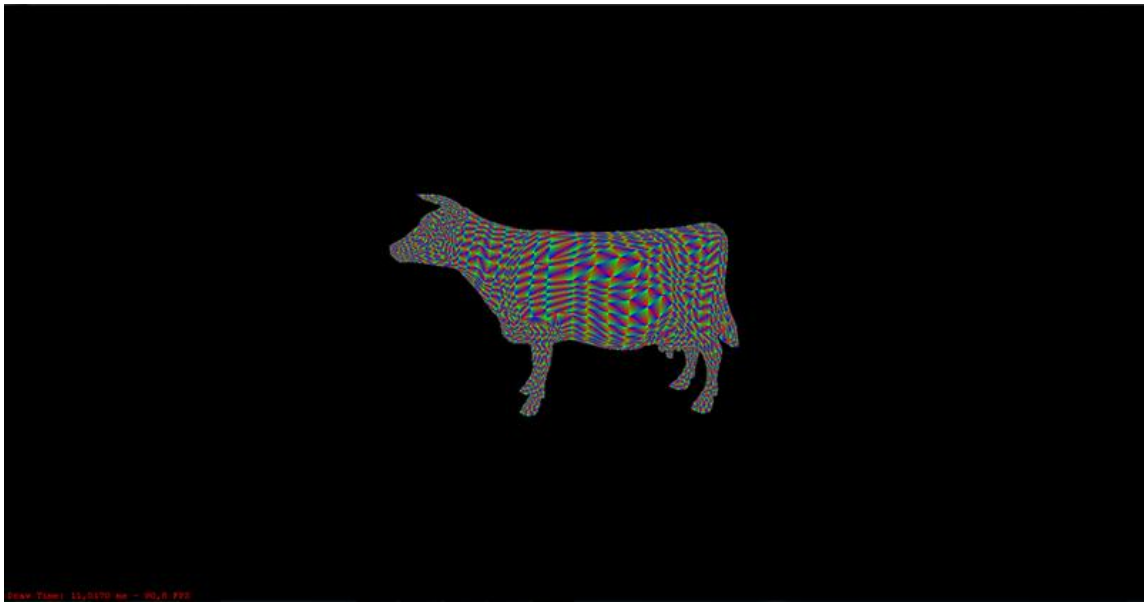

Figure 6.1 3D cow object before apply displayList()

After that this result was saw system is very slow beause the memory. Another method which is Display List was applied. Before program was shown 20 fps per a second after it was shown about 100 fps per a second. And rotation process was worked very fastly.

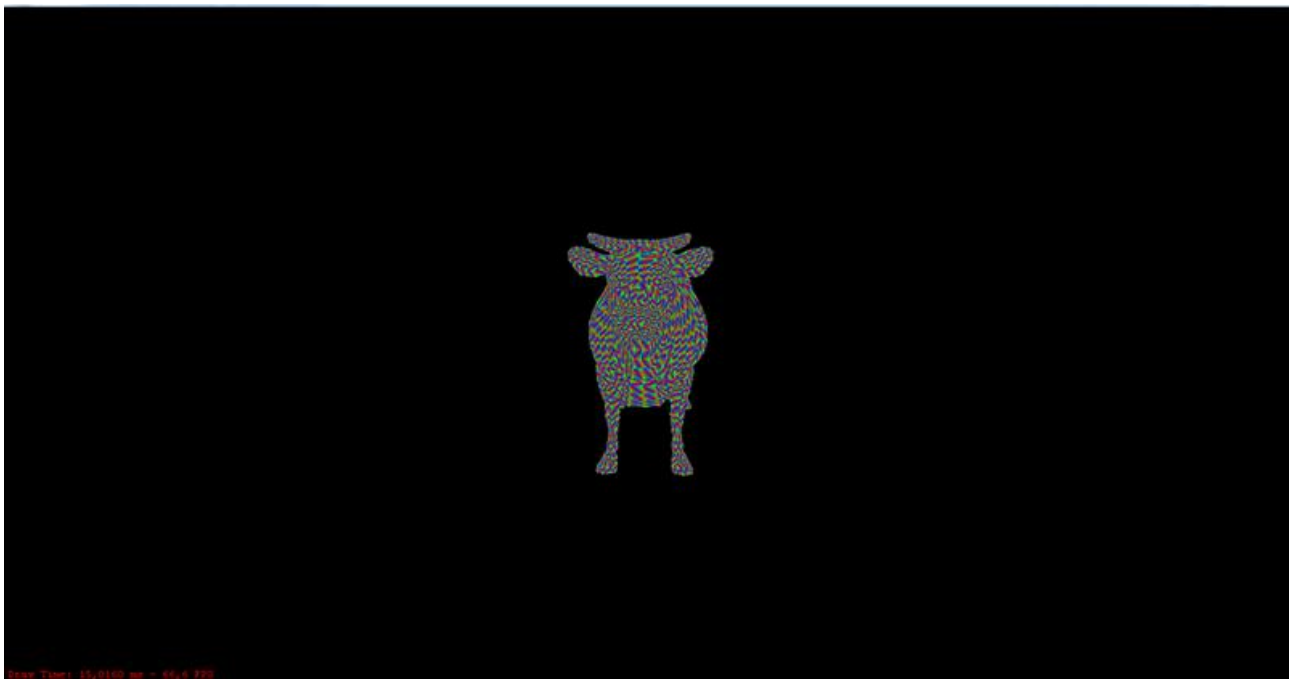

Figure 6.2 3D cow object after apply displayList() method

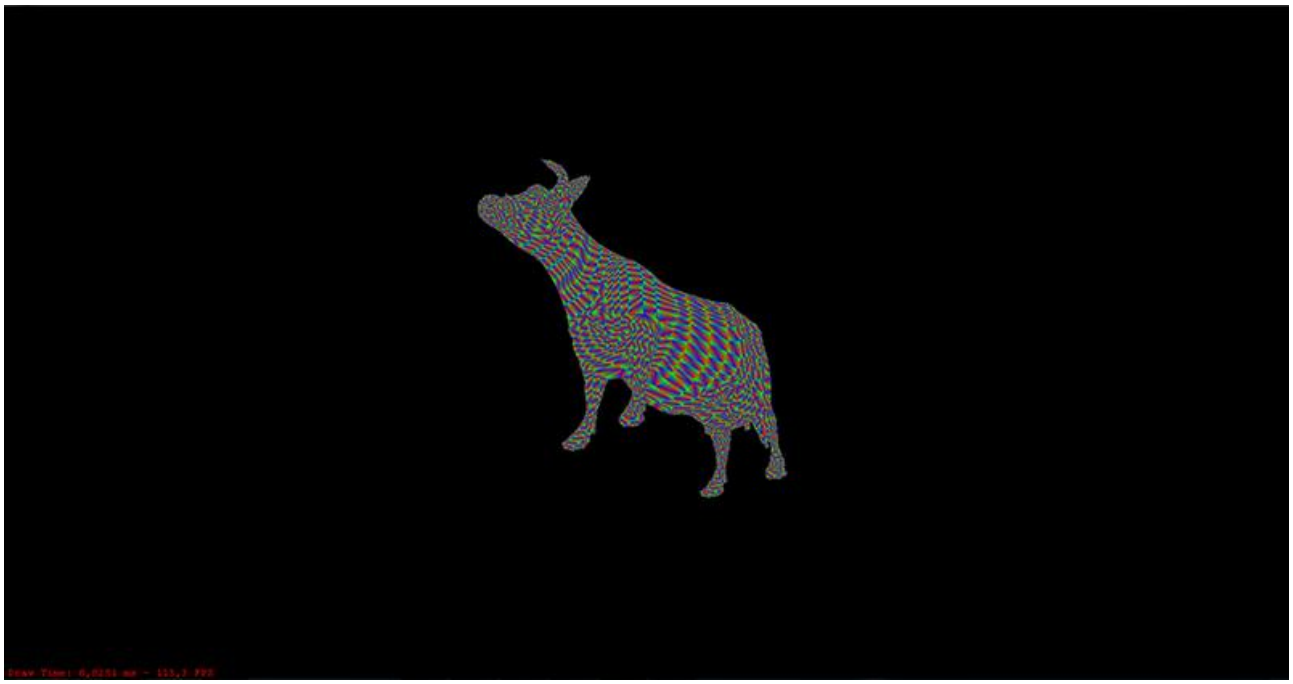

Figure 6.3 3D rotated cow object after apply displayList() method

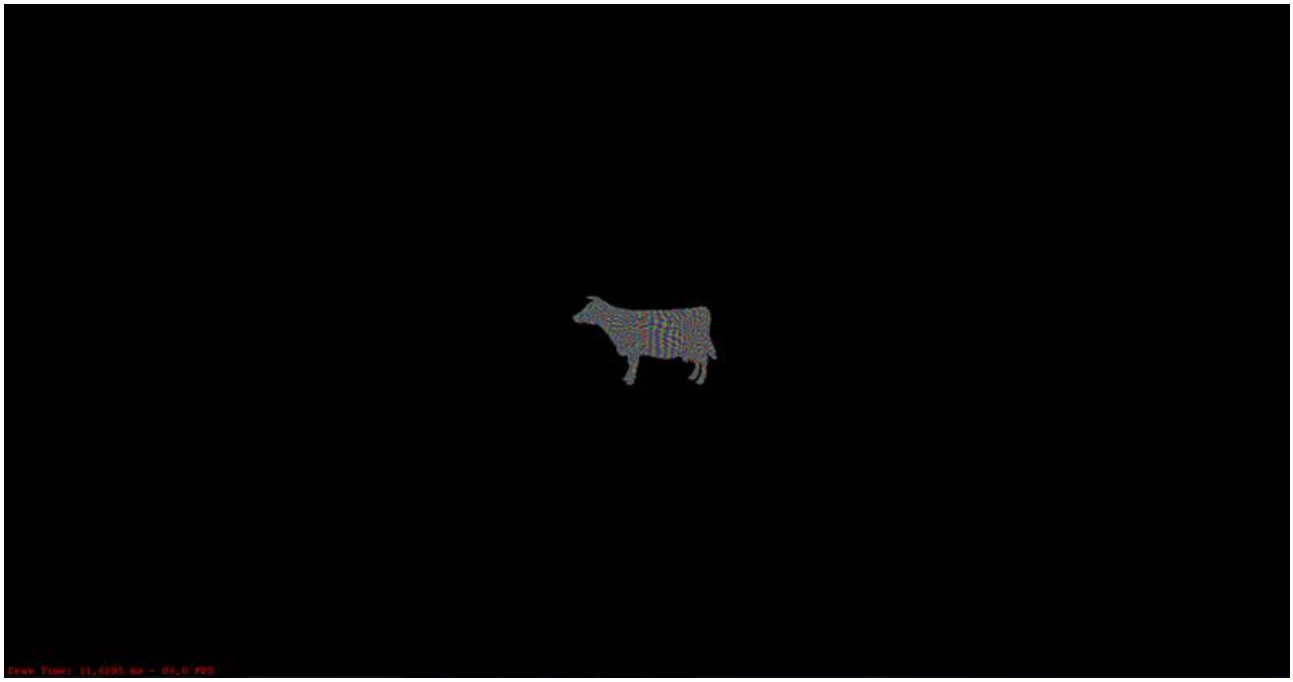

Figure 6.4 3D Zoom-out cow object after apply displayList() method

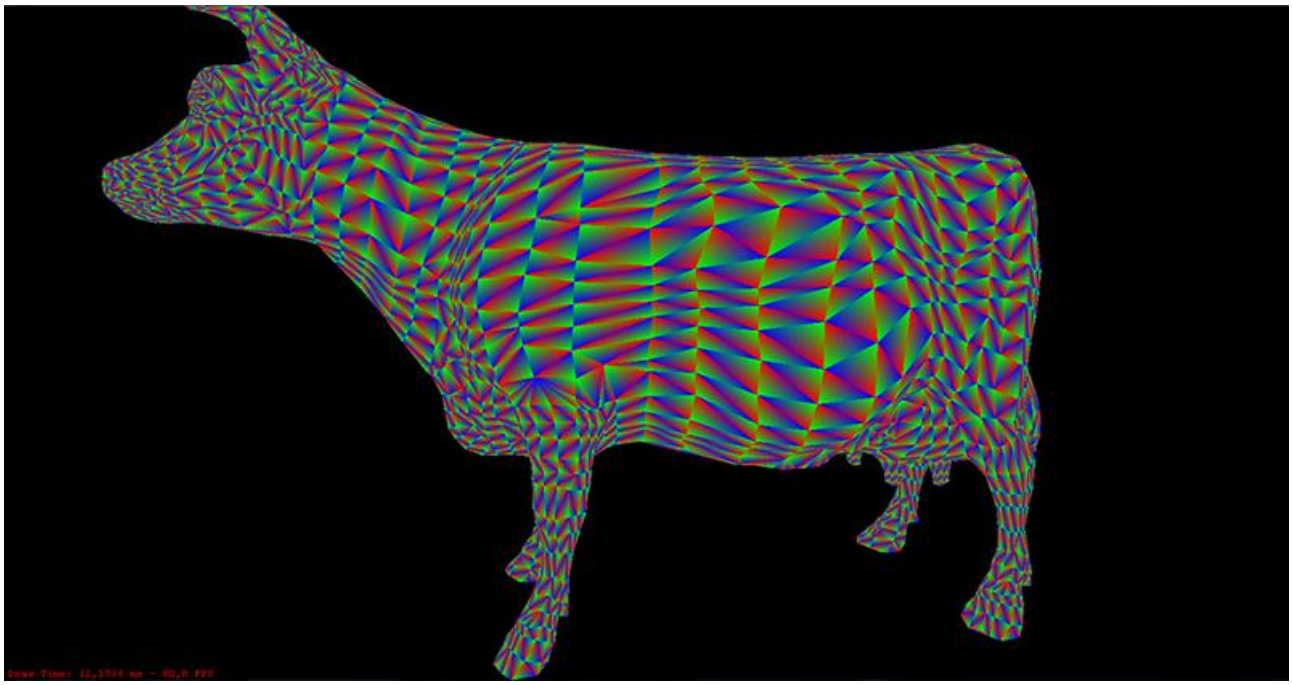

Figure 6.5 3D Zoom-in cow object after apply displayList() method

6.2 SECOND APPROACH AND RESULTS

Dolmabahce Palace Saltanat Door was tried to draw and observed results. The ply file that has vertex coordinates and and faces of Saltanat Door and the file size is about 20 Mb. It has about 1673312 vertex point and 3124759 faces.Also it has 4 different color properties such as red,green,blue and alpha.Because of the image was taken with Canon 650D photograph machine. In this thesis it was explained why we take this door photograph. Remind this photograph was taken for draw to 3D Saltanat Door with original color.

After the program was started the computer performance been very slow. CPU was increased %40 - %50 and pysical memory reached about 500 – 700 Mb.

A little bit shadow was saw on the left side of the Saltanat door. Because the sun is the right side of door in the morning.

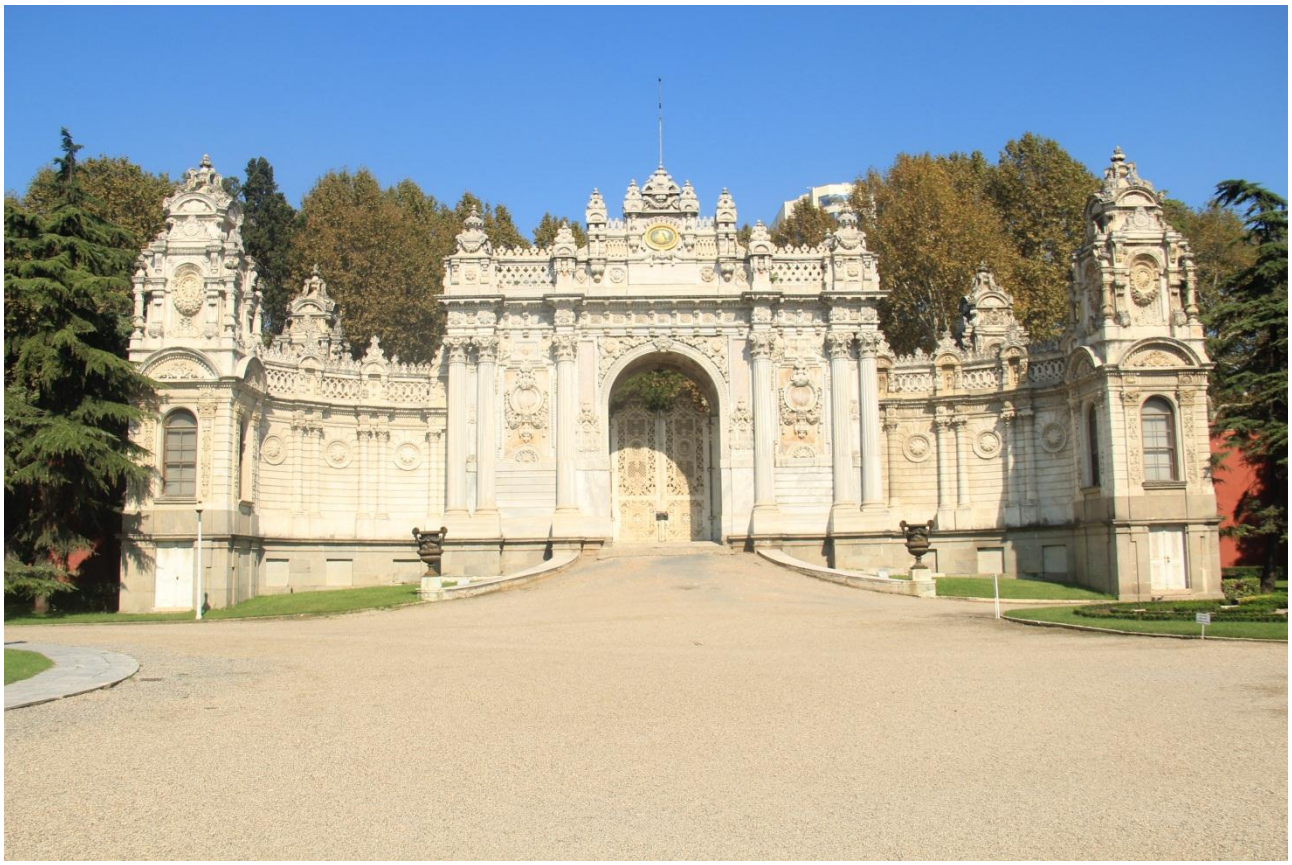

Figure 6.6 The original photograph of Saltanat door in the morning

In the program interface, if morning button is clicking then these 3D models will shown. This button shows to the door's colors and shadow in the morning.

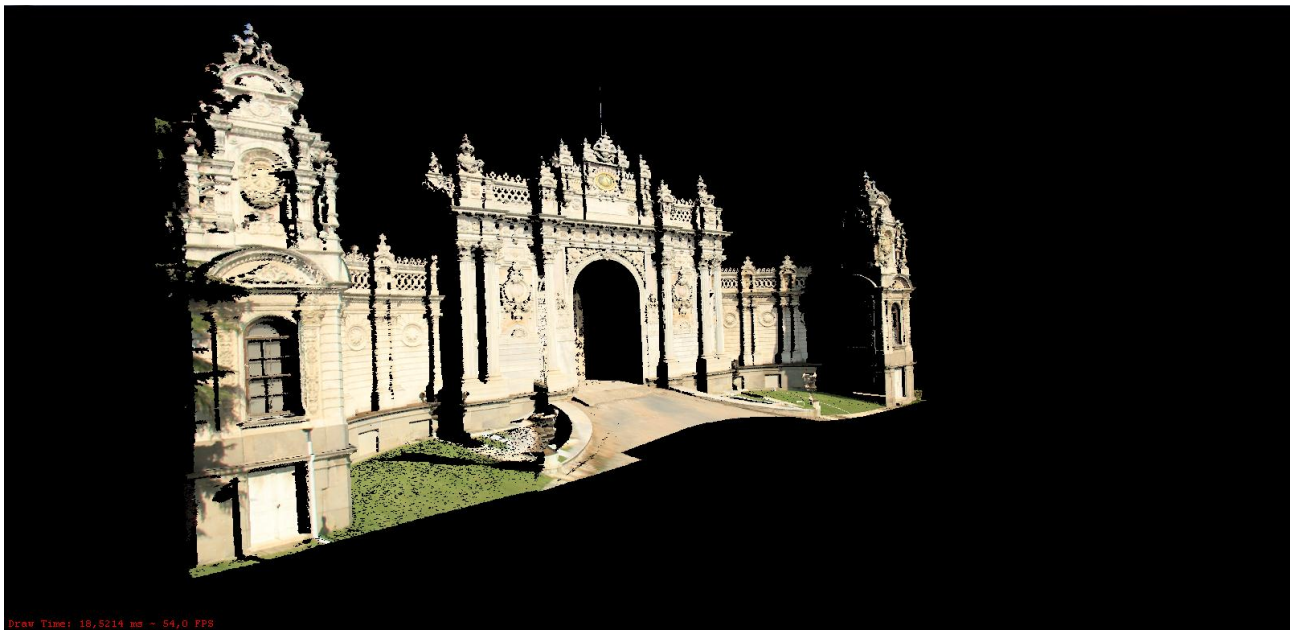

Figure 6.7 3D drawed object of Saltanat door in the morning

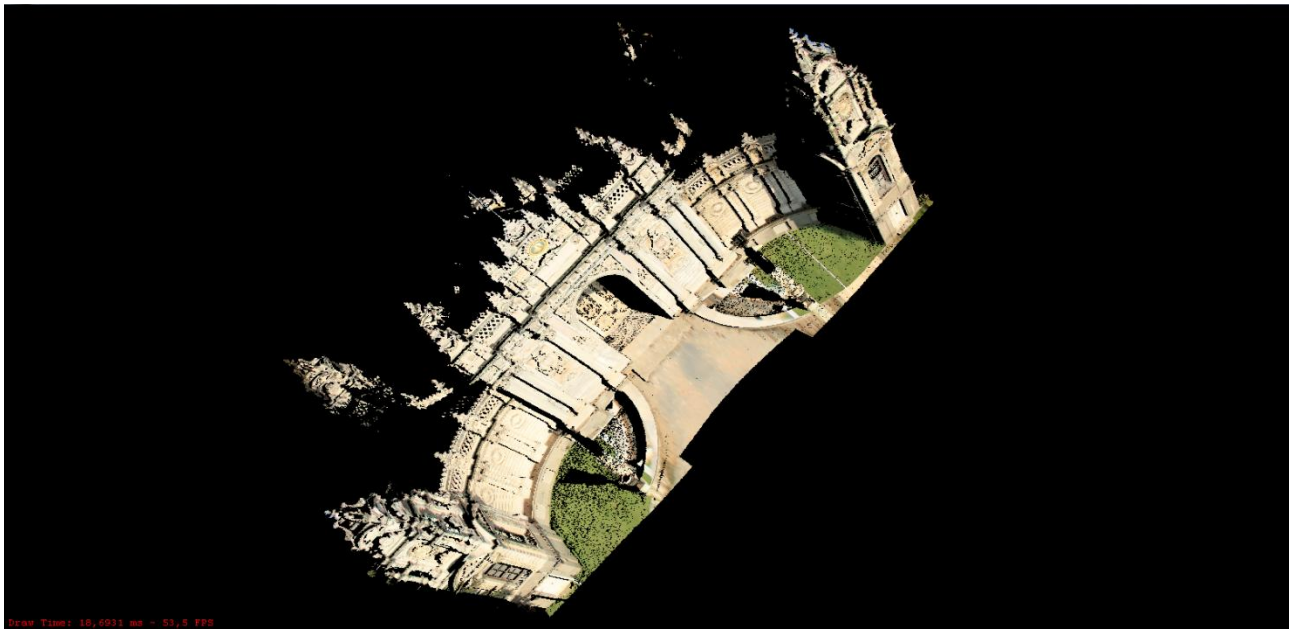

Figure 6.8 From upside of 3D drawed object of Saltanat door in the morning

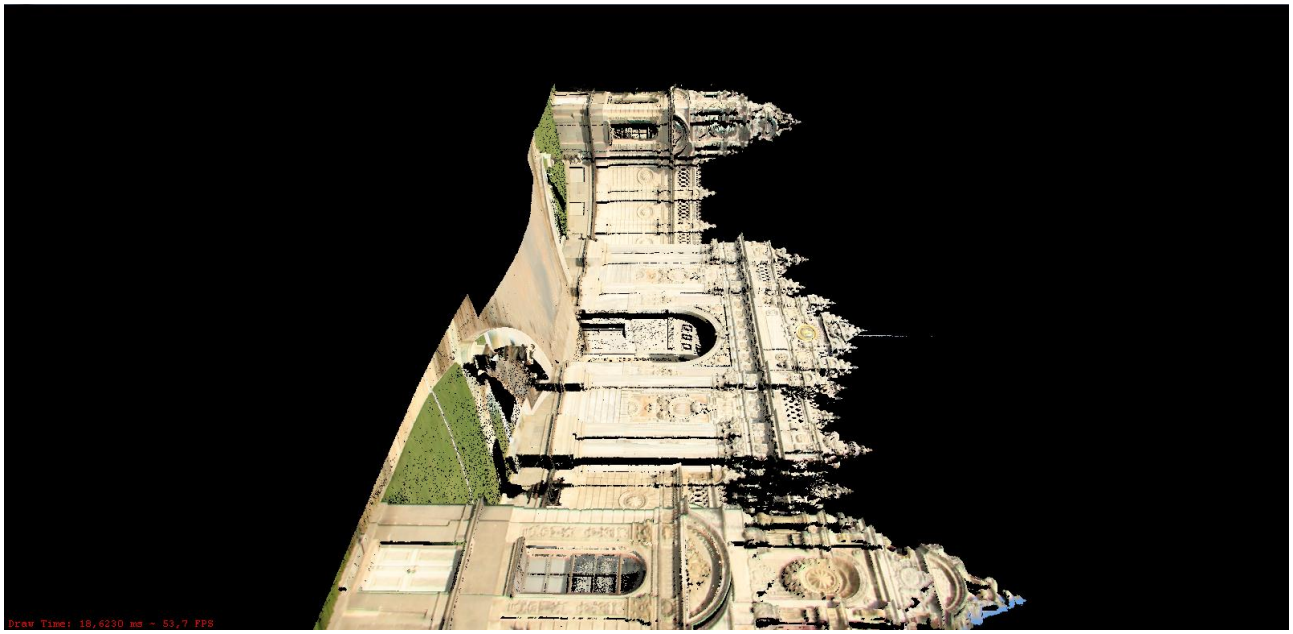

Figure 6.9 From beside of 3D drawed object of Saltanat door in the morning

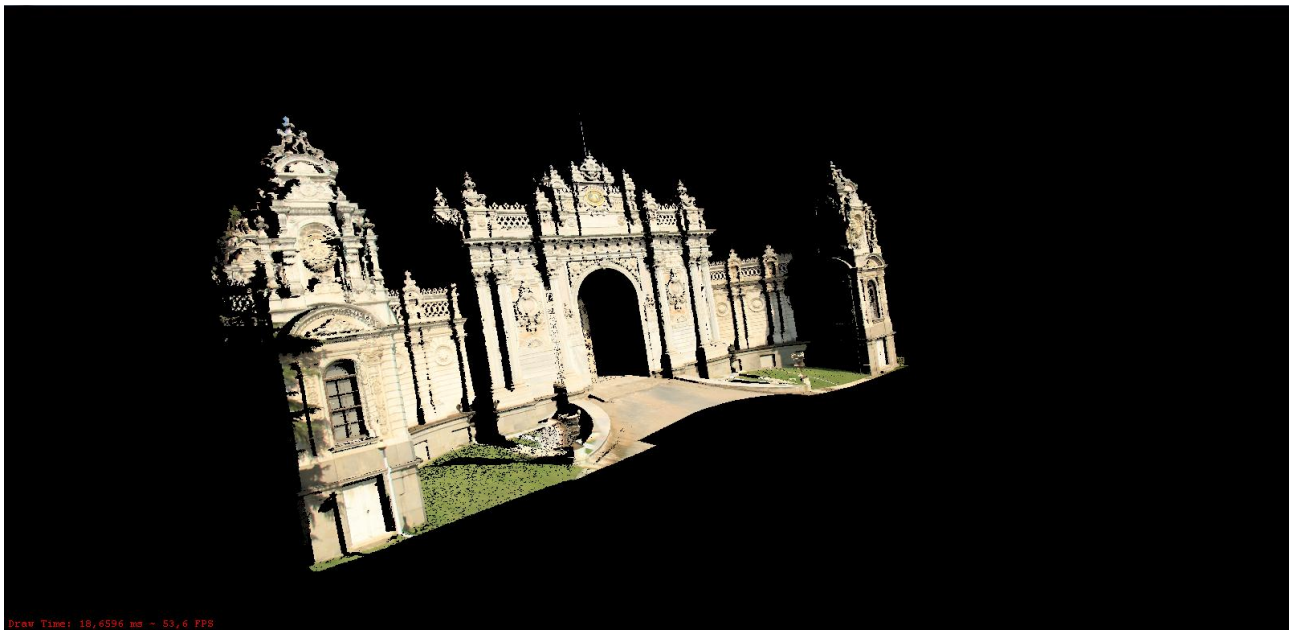

Figure 6.10 Zoom-out 3D drawed object of Saltanat door in the morning

Figure 6.11 Zoom-in 3D drawed object of Saltanat door in the morning

As we can see so much shadow was saw on the right side of the Saltanat door . Because the sun is up side of door in the afternon.

Figure 6.12 The original photograph of Saltanat door in the afternoon

In the program interface, if afternoon button is clicking then these 3D models will shown. This button shows to the door's colors and shadow in the afternoon.

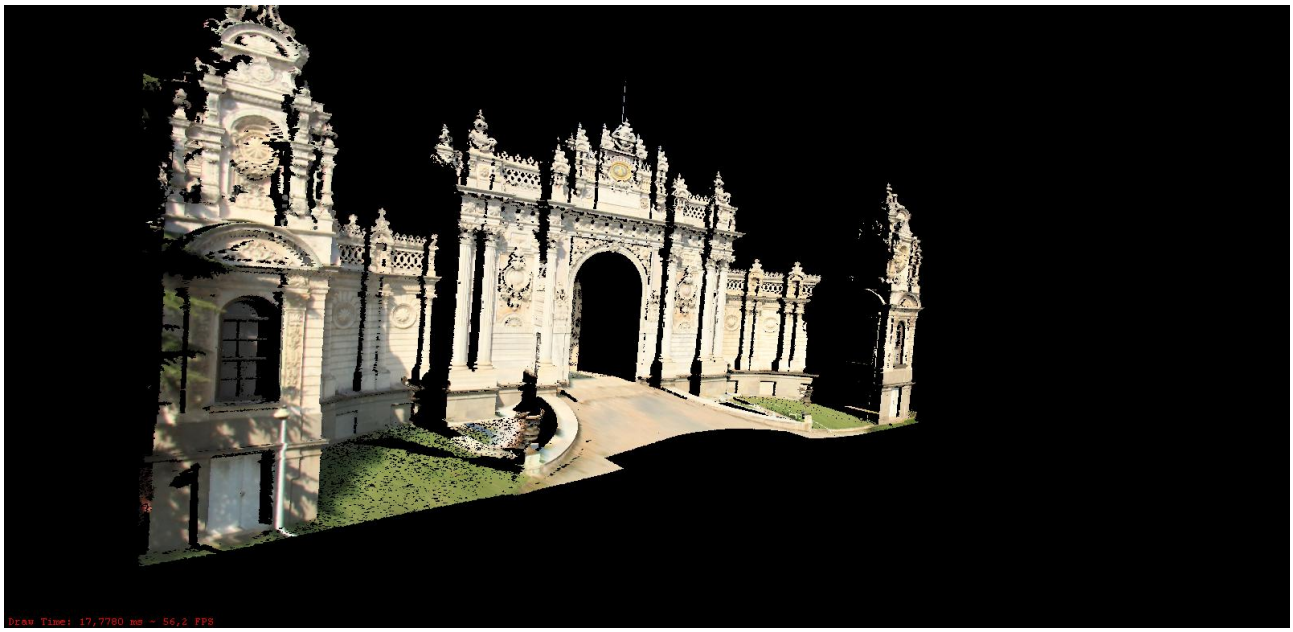

Figure 6.13 3D drawed object of Saltanat door in the afternoon

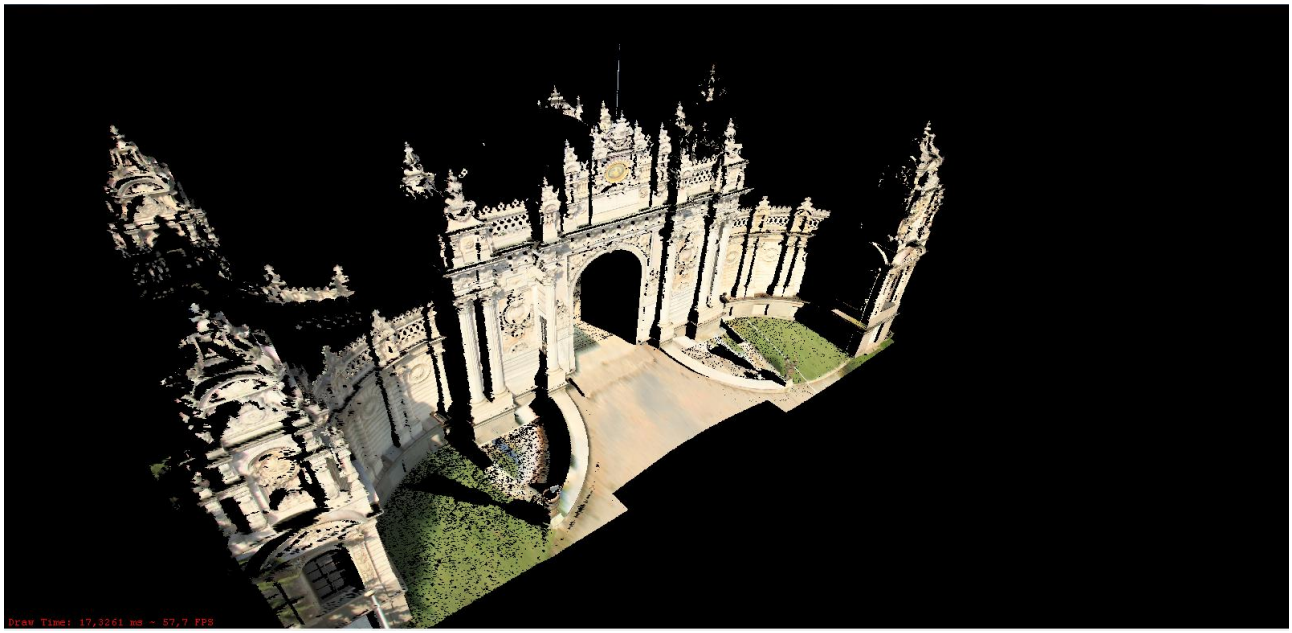

Figure 6.14 From upside of 3D drawed object of Saltanat door in the afternoon

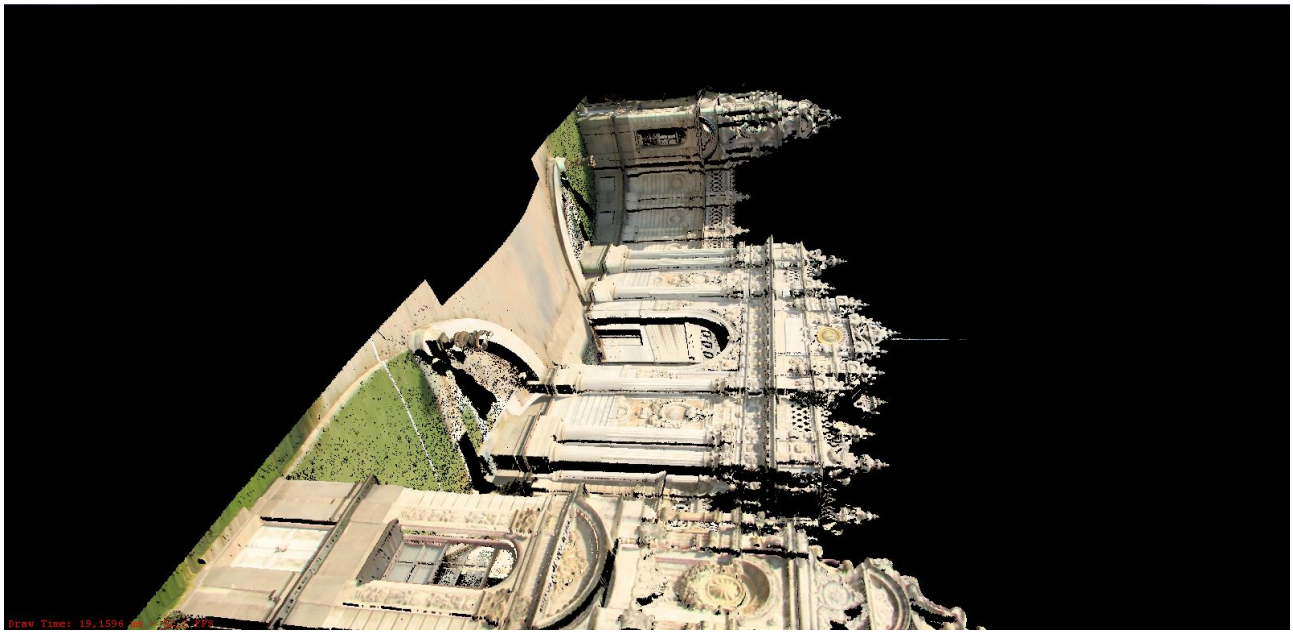

Figure 6.15 From beside of 3D drawed object of Saltanat door in the afternoon

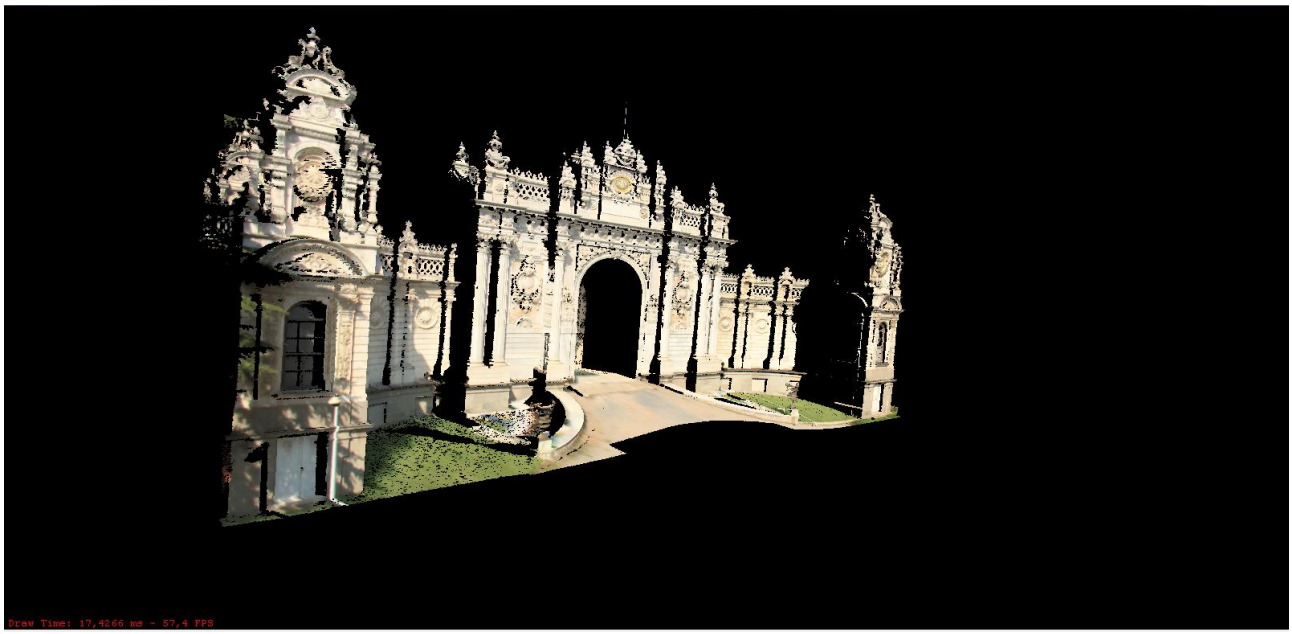

Figure 6.16 Zoom-out 3D drawed object of Saltanat door in the afternoon

Figure 6.17 Zoom-in 3D drawed object of Saltanat door in the afternoon

After all of processes a survey which is been some question about this system was applied some people. Totally twenty person answered those questions.

The questions are;

- How do you find the response time of the program?
- How do you find the rotation property of the program?
- How do you find the user interface of the program?
- How do you find the touch screen support of the program?
- How do you find the modelling quality of the program?
- How do you find zooming property of the program?

All of question have same five choices. From good answer to bad answer they are very satisfied, satisfied,neutral, not satisfied and very dissatisfied.

The answers of the question one are fiftheen very satisfied and five satisfied. Second question was taken eight very satisfied, eleven satisfied and one neutral answers. Third was taken question take three very satisfied, eleven satisfied, two neutral and three not satisfied answers. Fourth was taken question take fiftheen very satisfied and five satisfied answers. Fifth was taken question take thirteen very satisfied, five satisfied and two neutral answers.Last question was taken fiftheen very satisfied and five satisfied answers.

DEVELOPING AN INTERACTIVE VISUALIZATION TOOL FOR HISTORICAL BUILDINGS ON TOUCH SCREENS

How do you find the response time of the program?

- ☐ Very Satisfied
- ☐ Satisfied
- ☐ Neutral
- ☐ Not Satisfied
- ☐ Very Dissatisfied

How do you find the rotation property of the program?

- ☐ Very Satisfied
- ☐ Satisfied
- ☐ Neutral
- ☐ Not Satisfied
- ☐ Very Dissatisfied

How do you find the user interface of the program?

- ☐ Very Satisfied
- ☐ Satisfied
- ☐ Neutral
- ☐ Not Satisfied
- ☐ Very Dissatisfied

How do you find the touch screen support of the program?

- ☐ Very Satisfied
- ☐ Satisfied
- ☐ Neutral
- ☐ Not Satisfied
- ☐ Very Dissatisfied

How do you find the modelling quality of the program?

- ☐ Very Satisfied
- ☐ Satisfied
- ☐ Neutral
- ☐ Not Satisfied
- ☐ Very Dissatisfied

How do you find zooming property of the program?

- ☐ Very Satisfied
- ☐ Satisfied
- ☐ Neutral
- ☐ Not Satisfied
- ☐ Very Dissatisfied

Please list any other items/areas which have been unsatisfactory:

Please list any other items/areas which have been very satisfactory:

Figure 6.18 A Survey was asked some question to people

7 CONCLUSION

Interactive visualization tool is successfully applied to Dolmabahce Palace Saltanat Door's 3D visual model datas. The visual properties of 3D modal such as zoom-in, zoom-out and rotation is successfullu applied.

The method is applied by us for 3D visualization of historical objects that is very useful method and by this way we can apply this method for lots of historical objects even historical cities.

System performance results show that the server or computer process,memory and graphical card memory is so important for visualize of 3D modal. Also display lists is a very important command for a good performance of the system.

Survey results show that user interface and performance of the program is important for more than half visitors and needs to be improved to increase visitor satisfaction.

Since we have very good and consolidated technologies for 3D visualization of historical objects, it is now time to enlarge the offer of Augmented reality vision of historical objects.

REFERENCES

Nobuyuki Matsushita and Jun Rekimoto. Holowall: designing a finger, hand,body, and object sensitive wall. In UIST '97: Proceedings of the 10th annual ACM symposium on User interface software and technology, pages 209{210, New York, NY, USA, 1997. ACM Press. ISBN 0-89791-881-9. doi: http://doi.acm. org/10.1145/263407.263549.

Microsoft Corporation. MS surface. 2007. URL http://www.microsoft.com/surface/.

Shahram Izadi, Steve Hodges, Alex Butler, Alban Rrustemi, and Bill Buxton. Thinsight: integrated optical multi-touch sensing through thin form-factor displays. In EDT '07: Proceedings of the 2007 workshop on Emerging displays technologies, page 6, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-669- 1. doi: http://doi.acm.org/10.1145/1278240.1278246.

Wikipedia. Indium tin oxide | wikipedia, the free encyclopedia, 2008. [Online; accessed 28-July-2008].

Tyco Electronics. Elo TouchSystems. http://www.elotouch.com/. [Online; accessed 23-September-2008].

Nintendo Co., Ltd. Nintendo DS. http://www. nintendo.co.uk/NOE/en_GB/nintendo_ds_1023.html. [Online; accessed 23-September-2008].

J. Rekimoto. Smartskin: an infrastructure for freehand manipulation on interactive surfaces. In CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 113{120, New York, NY, USA, 2002. ACM.

J. Y. Han. Low-cost multi-touch sensing through frustrated total internal reection. In UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology, pages 115-118, New York, NY, USA, 2005. ACM Press.

J. Y. Han. Multi-touch interaction wall. In SIGGRAPH '06: ACM SIGGRAPH 2006 Emerging technologies, page 25, New York, NY, USA, 2006. ACM Press.

E. Costanza and J. Robinson. A region adjacency tree approach to the detection and design of fiducials. Vision, Video and Graphics (VVG, pages 63-70, 2003.

Multi-Touch Surfaces: A Technical Guide Technical Report TUM-I0833 Johannes Schöning, Peter Brandl, Florian Daiber, Florian Echtler, Otmar Hilliges, Jonathan Hook, Markus Löchtefeld, Nima Motamedi,Laurence Muller, Patrick Olivier, Tim Roth, Ulrich von Zadow 2008

[OpenGL ARB Guide] OpenGL Architecture Review Board, 2005, “OpenGL(R) Programming Guide: The Official Guide toLearning OpenGL(R), Version 2”, the Fifth Edition

Vision Based Hand Gesture Recognition: Pragati Garg, Naveen Aggarwal and Sanjeev Sofat World Academy Of Science, Engineering and Technology 49 2009

H. Zhou, T.S. Huang, “Tracking articulated hand motion with Eigen dynamics analysis”, In Proc. Of International Conference on Computer Vision, Vol 2, pp. 1102-1109, 2003

JM Rehg, T Kanade, “Visual tracking of high DOF articulated structures: An application to human hand tracking” ,In Proc. European Conference on Computer Vision, 1994

Elena Sánchez-Nielsen, Luis Antón-Canalís, Mario Hernández-Tejera, “Hand Getsure recognition for Human Machine Intercation”, In Proc. 12th International Conference on Computer Graphics, Visualization and Computer Vision : WSCG, 2004

B. Stenger, “Template based Hand Pose recognition using multiple cues”, In Proc. 7th Asian Conference on Computer Vision : ACCV 2006

Lars Bretzner, Ivan Laptev and Tony Lindeberg, “Hand gesture recognition using multiscale color features, hieracrchichal models and particle filtering”, in Proceedings of Int. Conf. on Automatic face and Gesture recognition, Washington D.C., May 2002

C C Wang, K C Wang, “Hand Posture recognition using Adaboost with SIFT for human robot interaction”, Springer Berlin, ISSN 0170-8643,
Volume 370/2008

R. Lienhart and J. Maydt, “An extended set of Haar-like features for rapid object detection,” in Proc. IEEE Int. Conf. Image Process., 2002, vol. 1, pp. 900–903.

Andre L. C. Barczak, Farhad Dadgostar, “Real-time hand tracking using a set of co-operative classifiers based on Haar-like features”, Res. Lett.
Inf. Math. Sci., 2005, Vol. 7, pp 29-42

Qing Chen , N.D. Georganas, E.M Petriu, “Real-time Vision based Hand Gesture Recognition Using Haar-like features”, IEEE Transactions on Instrumentation and Measurement -2007

B. Zheng, J. Takamatsu and K. Ikeuchi, "3d model segmentation and representation with implicit polynomials," IEICE Trans. on Information and Systems, E91-D(4): pp1149-1158, 2008.

APPENDICES

Appendix A Experimental Results Details

The answers of the survey question was answered by twenty people.

How do you find the response time of the program?	How do you find the rotation property of the program?	How do you find the user interface of the program?	How do you find the touch screen support of the program?	How do you find the modelling quality of the program?	How do you find zooming property of the program?
Very Satisfied	Very Satisfied	Very Satisfied	Very Satisfied	Very Satisfied	Very Satisfied
Very Satisfied	Satisfied	Neutral	Very Satisfied	Satisfied	Satisfied
Satisfied	Very Satisfied	Satisfied	Very Satisfied	Satisfied	Very Satisfied
Very Satisfied	Very Satisfied	Satisfied	Very Satisfied	Very Satisfied	Very Satisfied
Very Satisfied	Very Satisfied	Satisfied	Very Satisfied	Very Satisfied	Very Satisfied
Satisfied	Very Satisfied	Satisfied	Very Satisfied	Very Satisfied	Very Satisfied
Very Satisfied	Satisfied	Satisfied	Very Satisfied	Very Satisfied	Very Satisfied
Very Satisfied	Satisfied	Satisfied	Very Satisfied	Very Satisfied	Very Satisfied
Satisfied	Satisfied	Neutral	Satisfied	Satisfied	Satisfied
Very Satisfied	Very Satisfied	Satisfied	Very Satisfied	Very Satisfied	Very Satisfied
Very Satisfied	Very Satisfied	Satisfied	Very Satisfied	Very Satisfied	Very Satisfied
Very Satisfied	Very Satisfied	Very Satisfied	Very Satisfied	Very Satisfied	Very Satisfied
Very Satisfied	Satisfied	Satisfied	Very Satisfied	Very Satisfied	Very Satisfied
Satisfied	Satisfied	Not Satisfied	Satisfied	Neutral	Satisfied
Very Satisfied	Satisfied	Satisfied	Very Satisfied	Very Satisfied	Very Satisfied
Satisfied	Neutral	Not Satisfied	Satisfied	Neutral	Satisfied
Very Satisfied	Satisfied	Very Satisfied	Very Satisfied	Very Satisfied	Very Satisfied
Very Satisfied	Satisfied	Not Satisfied	Very Satisfied	Very Satisfied	Very Satisfied
Very Satisfied	Satisfied	Satisfied	Satisfied	Satisfied	Satisfied
Very Satisfied	Satisfied	Neutral	Satisfied	Satisfied	Very Satisfied