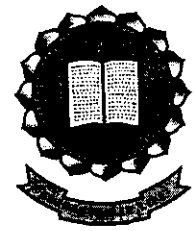
P-3127

3D MODELING OF BASIC OBJECTS



A PROJECT REPORT

Submitted by

DIVYAPRABHA S	**71206104012**
GAYATHRILAKSHMI N	**71206104014**
JINJU K	**71206104018**

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY : CHENNAI 600 025

APRIL 2010

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"3D MODELING OF BASIC OBJECTS"** is the bonafide work of **"DIVYAPRABHA S, GAYATHRILAKSHMI N** and **JINJU K"** who carried out the project under my supervision.

SIGNATURE

Dr. S. Thangasamy

DEAN

Department of Computer Science & Engg

Kumaraguru College of Technology,

Chinnavedampatti Post,

Coimbatore – 641606

SIGNATURE

Dr. S. Thangasamy

SUPERVISOR

DEAN

Department of Computer Science & Engg

Kumaraguru College of Technology,

Chinnavedampatti Post,

Coimbatore – 641606

The candidates with University Register Nos. **71206104012, 71206104014** and **71206104018** were examined by us in the project viva-voce examination held on 16-4-2010

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

We hereby declare that the project entitled "**3D Modeling of Basic Objects**" is a record of original work done by us and to the best of our knowledge, a similar work has not been submitted to Anna University or any Institutions, for fulfilment of the requirement of the course study.

The report is submitted in partial fulfilment of the requirement for the award of the Degree of Bachelor of Computer Science and Engineering of Anna University, Chennai.

Place: Coimbatore

Date: 16-4-2010

S. Divya

(S. Divya Prabha)

N. Gayathri

(N.Gayathri Lakshmi)

K. Jinju

(K. Jinju)

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

We are extremely grateful to **Dr. S. Ramachandran**, Principal, Kumaraguru College of Technology for having given us the opportunity to embark on this project.

We have immense pleasure to express my deep sense of gratitude and indebtedness to our Guide, **Dr. S. Thangasamy**, Dean, Department of Computer Science and Engineering, Kumaraguru College of Technology, for suggesting this problem and for his valuable suggestions and useful criticism, constant encouragement and inspiration evinced throughout the course of the project and in the preparation of the thesis.

We wish to place on record our heartfelt gratitude to our project coordinator, **Mrs. P. Devaki**, for her encouragement and constructive suggestions. We express our ineffable and deep sense of gratitude to **Mrs. R. Kalaiselvi**, our class advisor, for her helpful guidance, constant inspiration and good wishes during the tenure of study.

We reciprocate the kindness shown to us by the staff members of our college, parents and our beloved friends who have contributed in the form of ideas and encouragement for the successful completion of the project.

Above all, we humbly acknowledge the grace and blessings of the supreme power that capacitated us to fulfil this well nurtured dream.

ABSTRACT

ABSTRACT

Many industries exhibit demand for the construction of 3D models, which they use either for analysis or development purpose.3D models of human bodies, automobiles, geographic models and buildings are inevitably in demand. Many of the readymade models that are frequently in need for the clients are readily available with the online community and the 3D market is a very costly issue. Moreover, construction of 3D models is not dynamic; it takes quite a while to analyse the data and bring the 3D perspective of the required model. This discussion implies that there is a huge requirement for a tool that constructs 3D models from minimum data in a simple and dynamic manner.

The project presents an interactive modeling system that constructs 3D model from a collection of 2D images. As a prototype to the concept of 3D model construction from images, this project has limited the modeling process to the 3D surface that takes the shape of a parallelepiped only.

We have also implemented two different approaches for Rotation of the constructed object – the Euler angle rotation and Plane projection methods. An analysis for studying the performance factors of both the methods is performed and the results are documented. The rotation time taken by the plane projection method is 55times lesser (on a typical PC) than the time taken by Euler angle rotation, which proves the former method to be efficient. The rotation time is calculated as the minimum period of time taken by each of the methods to rotate the constructed 3D model around any arbitrary axes. Zooming and cropping are the two additional features of the tool, that adds to the convenience of the user. Constraints encountered while trying to extend the same approach to asymmetric objects are also discussed.

LIST OF FIGURES

TABLE OF CONTENTS

INTRODUCTION

1. INTRODUCTION

3D models and animation add a new era in the field of computer graphics. Modeling is the process of taking a shape and molding it into a completed 3D mesh. The most typical means of creating a 3D model is to take a simple object, called a primitive, and extend or "grow" it into a shape that can be refined and detailed. Primitives can be anything from a single point (called a vertex), a two-dimensional line (an edge), a curve (a spline), to three dimensional objects (faces or polygons).

1.1 TYPES OF MODELING

There are three basic methods that can be used to create a 3D model:

- Spline or patch modeling
- Box modeling
- Poly modeling / edge extrusion

1.1.1 SPLINE OR PATCH MODELING

A spline is a curve in 3D space defined by at least two control points. The most common splines used in 3D art are Bezier curves and NURBS (the software Maya has a strong NURBS modeling foundation.) Using splines to create a model is perhaps the oldest, most traditional form of 3D modeling available. A cage of splines is created to form a "skeleton" of the object that is to be created. A patch of polygons can then be created to extend between two splines, forming a 3D skin around the shape. Spline modeling is not used very often these days for character creation, due to how long it takes to create good models. The

models that are produced usually aren't useful for animation without a lot of modification. When creating a 3D scene that requires curved shapes, spline modeling will be extremely helpful.

1.1.2 BOX MODELING

Box modeling is possibly the most popular technique, and bears a lot of resemblance to traditional sculpting. In box modeling, one starts with a primitive (usually a cube) and begins adding detail by "slicing" the cube into pieces and extending faces of the cube to gradually create the form we're after. People use box modeling to create the basic shape of the model. Once practiced, the technique is very quick to get acceptable results. The downside is that the technique requires a lot of tweaking of the model along the way. Also, it is difficult to create a model that has a surface topology that lends well to animation.

Box modeling is useful as a way to create organic models, like characters. Box modelers can also create hard objects like buildings; however precise curved shapes may be more difficult to create using this technique.

1.1.3 POLY MODELING / EDGE EXTRUSION

While it's not the easiest to get started with, poly modeling is perhaps the most effective and precise technique. In poly modeling, one creates a 3D mesh point-by-point, face-by-face. Often one will start out with a single quad (3D object consisting of 4 points) and extrude an edge of the quad, creating a second quad attached to the first. The 3D model is created gradually in this way. While poly modeling is not as fast as box modeling, it requires less tweaking of the mesh to get it "just right," and we

can plan out the topology for animation ahead of time. Poly modelers use the technique to create either organic or hard objects, though poly modeling is best suited for organic models.

1.2 APPLICATIONS OF 3D MODELS

3D models are used in a wide variety of fields.

- The *medical industry* uses detailed models of organs.
- The *movie industry* uses them as characters and objects for animated and real-life motion pictures.
- The *video game industry* uses them as assets for computer and video games.
- The *science sector* uses them as highly detailed models of chemical compounds.
- The *architecture industry* uses them to demonstrate proposed buildings and landscapes through Software Architectural Models.
- The *engineering community* uses them as designs of new devices, vehicles and structures as well as a host of other uses.
- In recent decades the *earth science community* has started to construct 3D geological models as a standard practice.

1.3 3D MODEL MARKET

A large market for 3D models (as well as 3D-related content, such as textures, scripts, etc.) still exists - either for individual models or large collections. Online marketplaces for 3D content allow individual artists to sell content that they have created. In most cases, the artist retains ownership of the 3d model; the customer only buys the right to use and present the model.

1.4 NEED FOR THE 3D MODELING TOOL

All these factors determine that 3D modeling is a costly issue and also a time consuming tedious process. Most 3D models are built using high level software like CAD. In this point of view, constructing 3D model from its images obviously is an important approach. This proposed approach is concerned with building the model of the object from its 2D images in a dynamic way.

If the user is able to create a 3D model with multiple 2D images of the same object, then it solves the problem of involving huge amounts in buying 3D models or using an expensive 3D scanner. Moreover, the existing technology of constructing a 3D model is a not a dynamic process. Instead experts take specific duration of time to construct a 3D model and this has to be avoided. But the proposed project now confines to the implementation of the concept for a rectangular parallelepiped using the technique of box modeling, which is explained before. By extending the functionalities, the desired results can be obtained for other complex objects.

LITERATURE REVIEW

2. LITERATURE REVIEW

2.1 PROBLEM DOMAIN

A great deal of effort has been expended on 3D scene reconstruction from image sequences using computer vision techniques. Unfortunately, the results from most automatic modeling systems are disappointing and unreliable due to the complexity of the real scene and the fragility of the vision techniques. Part of the reason is the demand for accurate correspondences (e.g., point correspondence) required by many computer vision techniques. Moreover, such correspondences may not be available if the scene consists of images with complex pixel data.

This project presents an interactive modeling system that constructs 3D models from a collection of 2D images of the object. There are 2 techniques employed to achieve the desired goal of constructing the 3D model from multiple images and rotating them about arbitrary axes:

- Euler Rotation method
- Plane projection method

Both the methods are implemented in the same platform and an analysis is carried out to study the performance of both of the methods. The implementation of the technique is confined to the construction of any object that is in the form of rectangular parallelepiped. The factors that influenced the performance of rotation like computational overhead, quality of picture, and size of the picture for both the methods are also observed and discussed.

In an attempt to implement the procedure for asymmetric objects, a sample model is constructed which is a parallelepiped with curved surfaces.

The constraints which limited the project scope to symmetrical objects are inferred and documented.

2.2 EXISTING SYSTEM

2.2.1 3D SCANNER

A **3D scanner** is a device that analyzes a real-world object or environment to collect data on its shape and possibly its appearance (i.e. color). The collected data can then be used to construct digital, three dimensional models useful for a wide variety of applications. The purpose of a 3D scanner is usually to create a point cloud of geometric samples on the surface of the subject. These points can then be used to extrapolate the shape of the subject (a process called reconstruction). If color information is collected at each point, then the colors on the surface of the subject can also be determined.

For most situations, a single scan will not produce a complete model of the subject. Multiple scans, even hundreds, from many different directions are usually required to obtain information about all sides of the subject. These scans have to be brought in a common reference system, a process that is usually called *alignment* or *registration*, and then merged to create a complete model. This whole process, going from the single range map to the whole model, is usually known as the 3D scanning pipeline.

Drawback: 3D scanners are expensive. Their costs range from $14,000 - $140,000 which may not be affordable for small financial projects.

2.2.2 3D MODELING APPLIATIONS (DESIGN)

A 3D model is usually originated on the computer by engineer using some kind of 3d modeling tools. Creating 3D models is not easy and the software alone can cost a fortune. But there are quite a number of open source 3D modeling tools that help a user in creating or designing 3D models of our choice, with exhaustive features. Certain tools have simple drag and drop options that facilitate construction of complex models from simple 3D primitives.

Example: Blender, K-3D, Seamless3D

Drawback: The tools allow the user to create a model of their own and does not reconstruct from its 2D images.

2.2.3 3D MODELS FROM PANORAMIC MOSAICS

This tool presents an interactive modeling system that constructs 3D models from a collection of panoramic image mosaics. A panoramic mosaic consists of a set of images taken around the same viewpoint, and a transformation matrix associated with each input image, which are registered together to form one large image. Panorama Tools provides a powerful framework for re-projecting and blending multiple source images into immersive panoramic of many types.

Example: Panorama Tools

Drawback: This tool allows the user to create a 3D scene of physical location. This cannot be applied to construct a model of a 3D object dynamically.

2.3 PROPOSED SYSTEM

We propose the implementation of 3D modeling concept from multiple 2D images which can be accomplished in 2 methods – Euler

rotation method and Plane projection method. Projection of any 3 dimensional object over a 2D surface is a challenging task. Here a 3D surface is already created and the user is expected to feed the input images which are processed and reproduced over the 3D surface as a 3D model.

The analysis of performance factors of the method is also discussed. In addition to the modeling process, rotation of the model, zooming options are also featured. Rotation of the object is with reference to any arbitrary axis.

An additional 'Cropping' feature is integrated with the system that enables the user to crop the picture to the desired size and feed in as input.

2.4 SCOPE OF THE PROJECT

The project implements the technique for 3D objects taking up the shape of a rectangular parallelepiped. This construction was from the 6 multiple 2D image inputs. When tried to extend the system for asymmetric objects, it did not prove effective.

The model of a mobile phone was tried for reconstruction with its multiple 2D images. The mobile phone considered is also a parallelepiped with curved ends. In order to bring the curved surfaces on to the 3D surface, each corner required 5 little arcs projected at angles with small differences. Therefore 20 images were required to reconstruct the corners of the model. Considering this concept of modeling for models with more complex shapes, it can be said that the scope of the project is limited for such cases. More the asymmetry, lesser is the scope.

SYSTEM DESIGN

3. SYSTEM DESIGN

3.1 CONSTRUCTION OF A 3D SURFACE

3.1.1 NEED FOR 3D SURFACE

Taking a 2D object and mapping on to a 2D screen is pretty straight forward. The window is the same plane as the 2D world. If we are to take 3D objects and map them onto 2D screen, then we need to separate the model world from its rendered image. So a 3D surface is required to present the 3D system over the screen.

3D surface for the proposed method is constructed as a Rectangular Parallelepiped, which is projected to the 2D screen with the Plane Projection mechanism. Plane projection is a class that is supported by WPF technology of Microsoft.

3.2 PLANE PROJECTION

To provide these 3D effects, we need to implement a plane projection, which is a transformation that gives perspective to a 2D plane surface. It represents a perspective transform (a 3-D-like effect) on an object.

PlaneProjection has a few attributes each with X, Y and Z variants:

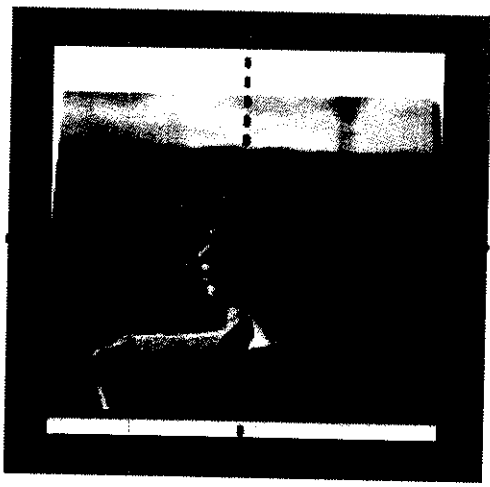
- Rotation.
- CenterOfRotation
- LocalOffset
- GlobalOffset

3.2.1 ROTATION

This attribute is specified using 3 properties – RotationX, RotationY and RotationZ and they are used to specify rotation angles.

Example:

The RotationX property specifies rotating around the horizontal axis of the object. The following illustration demonstrates how the RotationX property rotates an object around the x-axis.



RotateX = "-35"

Fig 3.1 RotateX property

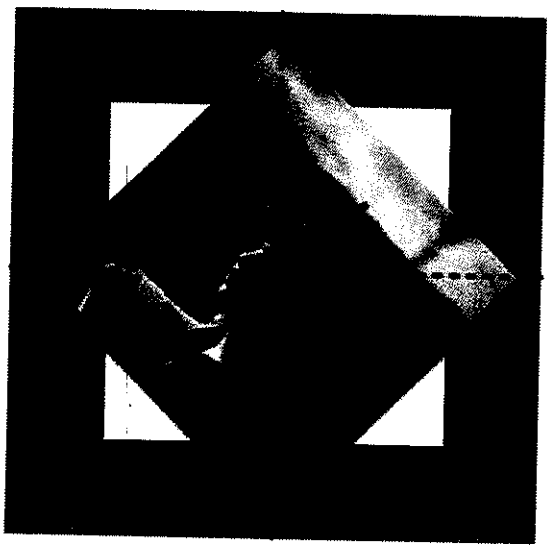
The illustration also says how the y-axis and z-axis are also shown. The point at which all three axes intersect is called the center of rotation.

The RotationY property rotates around the y-axis of the center of rotation.

RotateY = "-35"

Fig 3.2 RotateY property

The RotationZ property rotates around the z-axis of the center of rotation (a line that goes directly through the plane of the object).



RotateZ = "-45"

Fig 3.3 RotateZ property

3.2.2 CENTER OF ROTATION

- The center of rotation can be fixed by using the CenterOfRotationX, CenterOfRotationY, and CenterOfRotationZ properties.
- By default, the axes of rotation run directly through the center of the object, causing the object to rotate around its center; however, the center of rotation is moved to the outer edge of the object, it will rotate around that edge.
- The default values for CenterOfRotationX and CenterOfRotationY are 0.5, and the default value for CenterOfRotationZ is 0.

EXAMPLE ILLUSTRATION

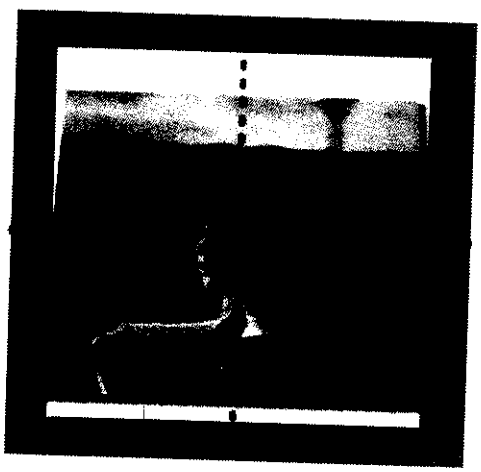


Fig 3.4 CenterOfRotationY default

CenterOfRotationY = "0.5" (default)

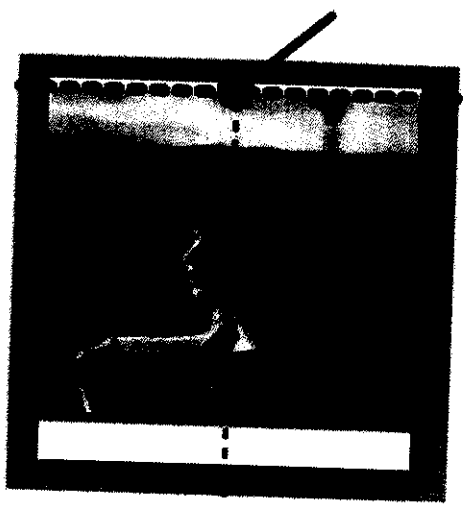


Fig 3.5 CenterOfRotationY modified

CenterOfRotationY = "0.1"

(Upper edge)

3.2.3 POSITIONING AN OBJECT

It is possible to position these rotated objects in space, relative to one another by using the following properties:

- LocalOffsetX - translates an object along the x-axis of the plane of a rotated object.
- LocalOffsetY - translates an object along the y-axis of the plane of a rotated object.
- LocalOffsetZ - translates an object along the z-axis of the plane of a rotated object.
- GlobalOffsetX - translates an object along the screen-aligned x-axis.
- GlobalOffsetY - translates an object along the screen-aligned y-axis
- GlobalOffsetZ - translates an object along the screen-aligned z-axis

3.3 CONSTRUCTION OF THE 3D SURFACE

The surface required for projection of the object over 3D screen is constructed as shown in the figure:

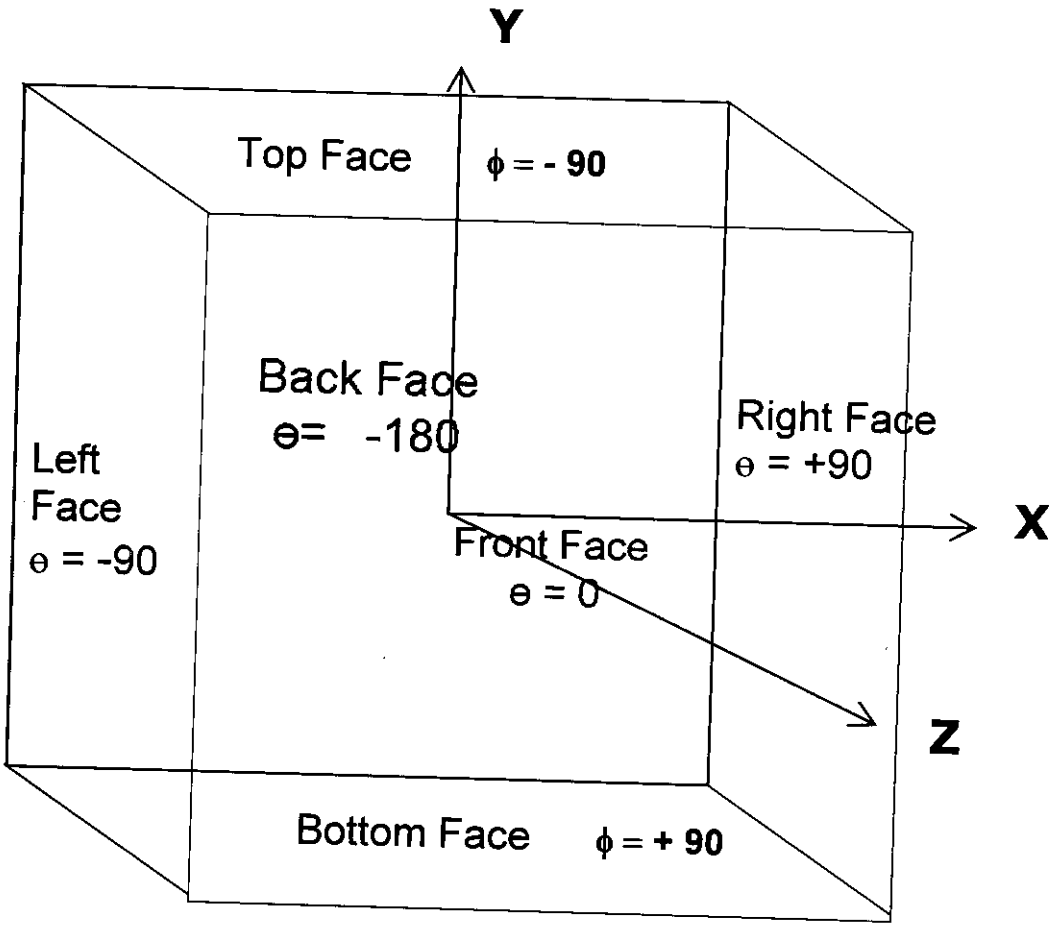


Fig 3.6 3D Surface

ϕ - X- axis rotation Angle

θ - Y axis Rotation Angle

SOFTWARE DESCRIPTION

4. SOFTWARE DESCRIPTION

4.1 PROGRAMMING LANGUAGE: C# 4.0

C# 4.0 is the latest version of the C# programming language, which has been finalized and is in beta testing as of May 2009. Microsoft has released the 4.0 runtime and development environment in a public beta of Visual Studio 2010. It is designed for building a variety of applications that run on the .NET Framework. C# is simple, powerful, type-safe, and object-oriented. The many innovations in C# enable rapid application development while retaining the expressiveness and elegance of C-style languages.

Visual C# is an implementation of the C# language by Microsoft. Visual Studio supports Visual C# with a full-featured code editor, compiler, project templates, designers, code wizards, a powerful and easy-to-use debugger, and other tools. The .NET Framework class library provides access to many operating system services and other useful, well-designed classes that speed up the development cycle significantly.

4.2 IDE: MICROSOFT VISUAL STUDIO 2010

Microsoft Visual Studio, the Integrated Development Environment from Microsoft can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft Windows, Windows Mobile,

Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight.

4.3 TECHNOLOGIES

4.3.1 WINDOWS PRESENTATION FOUNDATION

Windows Presentation Foundation (or **WPF**) is a graphical subsystem for rendering user interfaces in Windows-based applications. WPF was initially released as part of .NET Framework 3.0.It is built on DirectX, which provides hardware acceleration and enables modern UI features like transparency, gradients, and transforms.

WPF also offers a new markup language, known as XAML, which is an alternative means for defining UI elements and relationships with other UI elements. A WPF application can be deployed on the desktop or hosted in a web browser. It also enables rich control, design, and development of the visual aspects of Windows programs. It aims to unify a number of application services: user interface, 2D and 3D drawing, vector graphics, raster graphics and animation.

4.3.2 FRAMEWORK: MICROSOFT SILVERLIGHT 3.0

Silverlight is a web application framework that provides functionalities of integrating multimedia, graphics, animations and interactivity into a single runtime environment. Initially released as a video streaming plug-in, later versions brought additional interactivity features and support for development tools. The current version, 3.0, was released on July 9, 2009.

Silverlight provides a retained mode graphics system similar to Windows Presentation Foundation, and integrates multimedia, graphics, animations and interactivity into a single runtime environment. In Silverlight applications, user interfaces are declared in Extensible Application Markup Language (XAML) and programmed using a subset of the .NET Framework. XAML can be used for marking up the vector graphics and animations.

Silverlight 3 supports *perspective 3D* which enables 3D transformations of 2D elements. These transformations, as well as many 2D operations like stretches, alpha blending etc are hardware accelerated.

IMPLEMENTATION DETAILS

5. IMPLEMENTATION DETAILS

5.1 MODULES

1. 3D Surface Creation
2. Rotation using Euler angles
3. Rotation using PlaneProjection
4. Construction of asymmetric objects
5. Zooming
6. Cropping

5.2 MODULE DESCRIPTION

5.2.1 3D SURFACE CREATION

The construction of the 3D surface is as described in the system design section. When the user feeds the input images, the frames get re-sized to the size of the image. Based on the face, their positional angle is determined and the frame (which is actually the image here) is fixed over the panel. The positional angles are as follows:

Front Face : 0° rotation along Y-axis

Back Face : -180° rotation along Y-axis

Right Face : +90° rotation along Y-axis

Left Face : -90° rotation along Y-axis

Top Face : -90° rotation along X-axis

Bottom Face : +90° rotation along X-axis

5.2.2 ROTATION USING EULER ANGLES

Rotation is turning of an object or coordinate system by an angle about a fixed point. Euler's rotation theorem states that "an arbitrary rotation can be parameterized using three parameters" these parameters are commonly taken as the Euler angles. Euler angles can be used to describe the rotation of an object in 3D space. To generate a rotation transformation for an object using Euler angles method, we must designate:

- An Axis of rotation
- Amount of angular rotation

Rotations are implemented using the following transformation matrices:

Rotation about X –axis:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & \sin\phi & 0 \\ 0 & -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$x' = x$$

$$y' = y\cos\phi + z\sin\phi$$

$$z' = -y\sin\phi + z\cos\phi$$

Rotation about Y –axis:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$x' = x \cos\theta + z \sin\theta$

$y' = y$

$z' = -x \sin\theta + z \cos\theta$

Rotation about Z –axis:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 & 0 \\ -\sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$x' = x \cos\varphi + y \sin\varphi$

$y' = -x \sin\varphi + y \cos\varphi$

$z' = z$

where ϕ, ϵ and φ are the rotations around the X, Y and Z axes respectively.

Translation Matrix:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$x' = x + tx$

$y' = y + ty$

$z' = z + tz$

Flowchart Depicting the Composite Transformations – Rotation about Arbitrary Axes

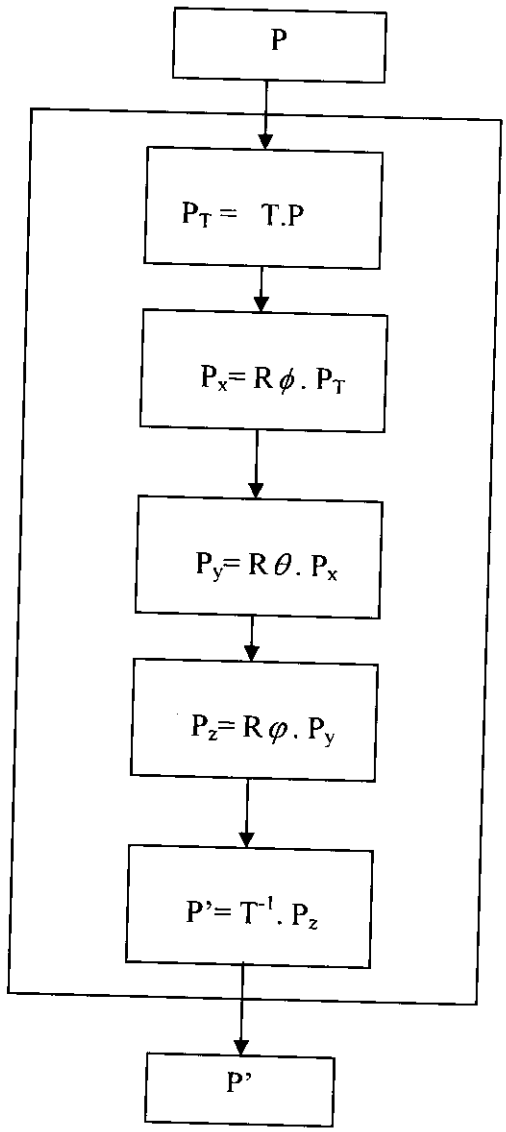


Fig 5.1 Flowchart depicting Euler Rotation algorithm

ROTATION AROUND ARBITRARY AXES:

When an object is to be rotated about an axis that is not parallel to one of the coordinate axes, we need to perform some additional transformations. In this case, we also need rotations to align the axis with a selected coordinate axis and to bring back the axis back to its original orientation.

Given the specifications for the rotation axis and the rotation angle we can accomplish the required rotation in three steps:

1. Translate the object so that the rotation axis passes through the coordinate origin.
2. Perform the specified rotation about the coordinate axis.
3. Apply the inverse translation to bring the rotation axis back to its original position.

CONVERSION OF 3D CO-ORDINATES TO 2D CO-ORDINATES:

To convert 3D co-ordinates to 2D co-ordinates it is essential to know about the reference of camera points. The camera points are in the same direction as the Z axis and the points to draw are in front of it. The camera is a kind of a rectangular based pyramid where the peak of the pyramid is set at the origin (0, 0, 0) and the base serves as a projection screen for the 3D universe and the points would be projected on the projection screen. What the user will see will be the image of this 3D world on the rectangular base. The focal distance (Fc) of the camera is the distance between the origin (0, 0, 0) and the surface which contains the base of the pyramid.

In the following figure, the OABCD pyramid is the camera with its projection screen as ABCD, the line PQ is a 3D object to be projected on the 2D screen.

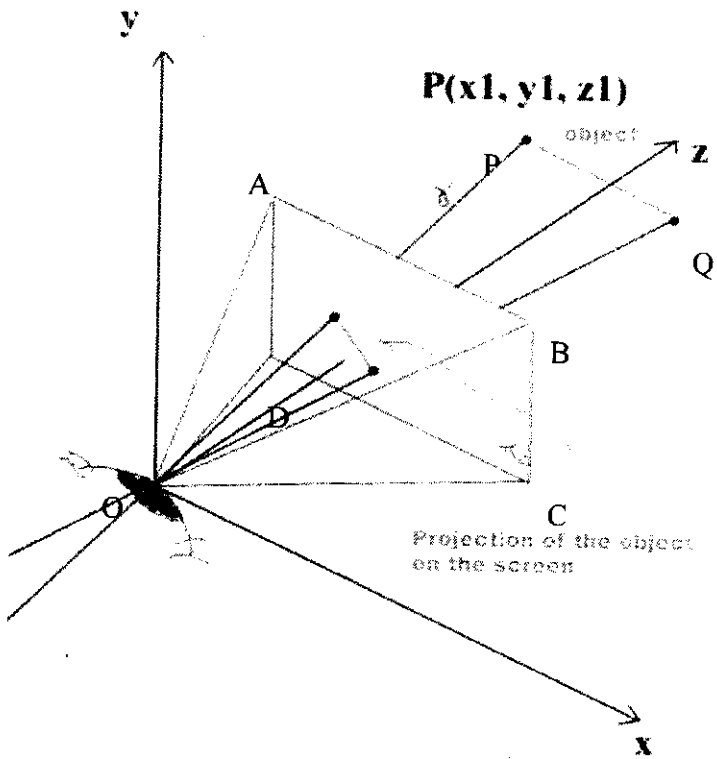


Fig 5.2 Projection of 3D object on 2D screen

Say the camera's at (Xc, Yc, Zc) and the point to be projected is P = (X, Y, Z). The distance from the camera to the 2D plane onto which we are projecting is F (so the equation of the plane is Z-Zc=F). The 2D coordinates of P projected onto the plane are (X', Y').

Then,

$X' = ((X - Xc) * (F/Z)) + Xc$

$Y' = ((Y - Yc) * (F/Z)) + Yc$

If your camera is the origin, then this simplifies to:

$X' = X * (F/Z)$

$Y' = Y * (F/Z)$

PROCEDURE TO ROTATE THE 3D OBJECT IN ARBITRARY AXES

1) Determine the size of the cuboid as follows:

From the input image, calculate

Width=frontface.width =backface.width

Height =frontface.height=backface.height

Depth =leftface.width=rightface.width

2) Set cubeorigin= (width/2, height/2, depth/2).

3) Set origin = (0, 0, 0).

4) Find the 3D coordinates of the vertices based on the size of the cuboid.

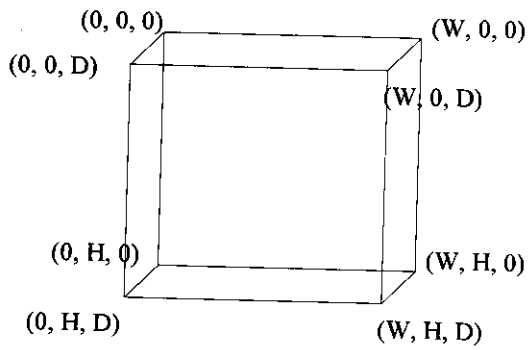


Fig 5.3 3D coordinates of the cuboid vertices

5) Get the values of angleX, angleY, angleZ.

6) Set cameraposition (X_C, Y_C, Z_C) = (width/2, height/2, Fc).

7) For each face of the cuboid do:

For each Z vary from 0 to face. depth do

For each Y vary from 0 to face. height do

For each X vary from 0 to face. width do

Point P= (X, Y, Z)

P_T = Translate (P, cubeorigin, origin).

P_X = RotateX (P_T, angleX)

P_Y = Rotate Y (P_X , angleY)

P_Z= RotateZ (P_Y, angleZ)

P' = Translate (P_Z, origin, cubeorigin)

$P_{2D}.X = ((P'.X - X_C) * ((Z - Z_C) / Z)) + P'.X$

$P_{2D}.Y = ((P'.Y - Y_C) * ((Z - Z_C) / Z)) + P'.Y$

Draw the pixel in P_{2D}.

TRANSLATE (Point P, Point oldorigin , Point neworigin)

1) Compute difference = neworigin – oldorigin.

2) P_T = P + difference

RotateX (Point P , AngleX)

Compute

$P_X.X = P.X$

$P_X.Y = P.Y * \cos(AngleX), + P.Z * \sin(AngleX)$

$P_X.Z = -P.Y * \sin(AngleX) + P.Z * \cos(AngleX)$

RotateY (Point P , AngleY)

Compute

$P_Y.X = P.X * \cos(AngleY) + P.Y * \sin(AngleY)$

$P_Y.Y = P.Y$

$P_Y.Z = -P.X * \sin(AngleY) + P.Z * \cos(AngleY)$

RotateZ (Point P , AngleZ)

Compute

$P_{Z.}X = P.X * \cos(AngleZ) + P.Y * \sin(AngleZ)$

$P_{Z.}Y = -P.X * \sin(AngleZ) + P.Y * \cos(AngleZ)$

$P_{Z.}Z = P.Z$

5.2.3 ROTATION USING PLANEPROJECTION

In this method rotation of 3D object is achieved using the following two steps:

- Mouse Position Capture

  Rotation angle = (current position-previous position)

  This rotation angle is used to rotate the generated 3D object along Y- axis.

- Translation

  Based on the angle of rotation along Y- axis, each face of the 3D object is translated along X and Z axes. The magnitude of translation is given by:

  x = depth * Cos (rotateY+90)

  z = depth * Sin (rotateY+90)

For every angle change, the values are computed and the faces are projected with respective transformations (rotation along the axis and translation along other axes) that cause the object to rotate.

Time required to rotate the object around arbitrary axes = **1.06 seconds**

5.2.4 CONSTRUCTION OF ASYMMETRIC OBJECTS

The above discussed modules of surface construction and rotation is extended to asymmetric objects. But there is a short-coming for this method as the user had to feed multiple image inputs.

Consider the example of a mobile phone. To bring a curved corner to the 3D model, it needed 5 images, projected at slightly varying angles. Applying it to all the corners, the user needs to feed 20 images to bring rounded ends at the corner. This is a big constraint of the system because of which dynamic modelling of 3D objects with asymmetry is a tough task for both the users and designers.

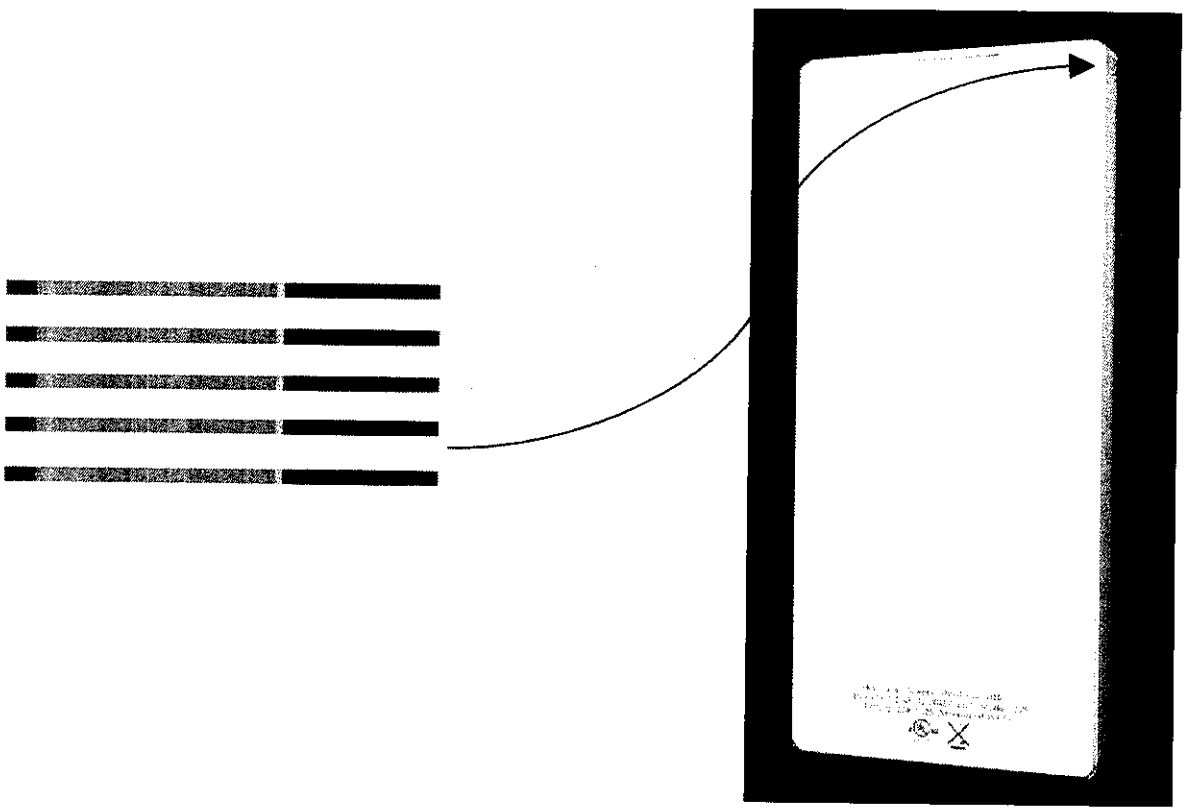


Fig 5.4 Asymmetric 3D object construction

5.2.5 ZOOMING

Zooming is moving the generated 3D object back and forth in 3D space. While zooming, the generated 3D object is being scaled and translated according to their positions in that space. This scaling and movement is based on their respective z values of each face.

For Zoom-in: The Z-value of each of the faces is increased.

For Zoom-out: The Z-value of each of the faces is decreased.

This is implemented by changing the CenterOfRotationZ of each plane continually and projecting each plane according to the new CentreOfRotationZ.

5.2.6 CROPPING

A digital image is made up of pixels. Cropping is the process of removing selected pixels from a digital image. In this system we use multiple images of an object to generate its 3D model. Cropping is included in this system to extract the required data.

The cropping process is accomplished in the following way:

- The user loads the image to be cropped
- A rectangular clip window is defined by the user.
- The portion of the image within the clip window gets saved as separate image.

PERFORMANCE ANALYSIS

6. PERFORMANCE ANALYSIS

6.1 NEED FOR ANALYSIS

The two methods for rotation (Euler angles and PlaneProjection) have been discussed in detail. During implementation, there were factors that influenced the performance of the methods, processing time being the critical factor. This led to the analysis of finding the method with better efficiency.

6.2 COMPLEXITY INVOLVED IN THE EULER ANGLE METHOD

6.2.1 COMPUTATIONAL OVERHEAD

In the system, we are constructing 3D object from 6 bitmap images. These images are fed as input to the system by the user.

A bitmap is a data structure representing a generally rectangular grid of pixels, or points of color, viewable via a monitor, paper, or other display medium. Raster images are stored in image files with varying formats. It is technically characterized by the width and height of the image in pixels and by the number of bits per pixel.

To rotate an object like a cuboid, we need to rotate each of the 6 faces. Each face will hold a bitmap image which contains array of pixels. According to the Euler rotation method that we have discussed earlier, **rotation of single pixel in 3D space** will involve the following 18 arithmetic operations:

(i) 4 x 3 multiplication operations (X, Y & Z axes)

(ii) 2 x 3 addition operations (X, Y & Z axes)

Considering an image of size 100x100,

Total no. of Mathematical operation involved

in rotating the image in 3D space = 18 x 100 x 100 = 180000.

Assuming each of the six faces in 3D object hold an image of size 100 x 100,

Then

Total no. of Mathematical operation involved

in rotating the 3D object = 6 x 18 x 100 x 100.

If the user gives a high quality image as input such as an image of size 2048×1536, it has a total of 3,145,728 pixels or 3.1 mega pixels, for all the 6 faces, to rotate this 3D object,

Total no. of Mathematical operation involved

in rotating the 3D object = 6 x 18 x 2048 x 1536

= 339738624

If we are rotating the object continuously, for each angle change in any of the three axes, it will have to perform all the 39738624 mathematical operations. This will reduce the system performance significantly.

Time Consumption

- For Single_axis rotation 14.55 seconds
- For Two – axis rotation 36.33 seconds
- For Three – axis rotation 55 seconds

GIMBAL LOCK

More over Euler angles methods suffers from Gimbal Lock. Gimbal lock is the loss of one degree of freedom that occurs when the axes of two of the three gimbals are driven into the same place and cannot compensate for rotations around one axis in the three dimensional space.

Loss of a degree of freedom with Euler angles

A rotation in 3D space can be represented numerically with matrices in several ways. One of these representations is:

$$R = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

with α and γ constrained in the interval $[-\pi,\pi]$, and β constrained in the interval $[0,\pi]$.

For example, when $\beta = 0$ ($\cos 0 = 1$ and $\sin 0 = 0$) the above expression becomes:

$$R = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The second matrix is the identity matrix and has no effect on the product. Carrying out matrix multiplication of first and third matrices:

$$R = \begin{bmatrix} \cos\alpha\cos\gamma - \sin\alpha\sin\gamma & -\cos\alpha\sin\gamma - \sin\alpha\cos\gamma & 0 \\ \sin\alpha\cos\gamma + \cos\alpha\sin\gamma & -\sin\alpha\sin\gamma + \cos\alpha\cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

And by using the trigonometry formulae:

$$R = \begin{bmatrix} \cos(\alpha+\gamma) & -\sin(\alpha+\gamma) & 0 \\ \sin(\alpha+\gamma) & \cos(\alpha+\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Changing α's and γ's value in the above matrix has the same effects: the rotation's angle $\alpha + \gamma$ changes, but the rotation's axis remains in the Z direction. The last column and the last line in the matrix won't change: one degree of freedom has been lost.

6.3 BETTER PERFORMANCE OF PLANE PROJECTION

When the analysis factor is process time, plane projection will definitely take advantage over the other method. To project a plane in a particular angle along a particular axis, plane projection class uses Quaternions instead of Euler Angles approach. Quaternion does not have the Gimbal lock problem.

6.3.1 QUATERNIONS

In mathematics, the quaternions are a number system that extends the complex numbers. Quaternions find uses in both theoretical and applied mathematics, in particular for calculations involving three-dimensional rotations such as in three-dimensional computer graphics and epipolar geometry. They can be used alongside other methods, such as

Euler angles and matrices, or as an alternative to them depending on the application.

Unit quaternions provide a convenient mathematical notation for representing orientations and rotations of objects in three dimensions. Compared to Euler angles they are simpler to compose and avoid the problem of Gimbal lock. Compared to rotation matrices they are more numerically stable and may be more efficient.

6.3.1.1 ROTATIONS WITH QUATERNIONS

Let (*w*,*x*,*y*,*z*) be the coordinates of a rotation as previously described. Define the quaternion

$$q = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k} = w + (x, y, z) = \cos(\alpha/2) + \vec{u}\sin(\alpha/2)$$

where $\vec{u}$ is a unit vector. Let also $\vec{v}$ be an ordinary vector in 3 dimensional space, considered as a quaternion with a real coordinate equal to zero. Then it can be shown that the quaternion product

$$q\vec{v}q^{-1}$$

yields the vector $\vec{v}$ rotated by an angle α around the $\vec{u}$ axis.

Steps to change the rotation represented by a quaternion:

1. Generate a temporary quaternion, to represent change in rotation.

   Eg: If the current rotation is changed by rotating backwards over the X-axis a little bit, this change is represented in temporary quaternion.

2. Multiply the two quaternions (the temporary and permanent quaternions) together; this will generate a new permanent quaternion, which has been changed by the rotation described in the temporary quaternion.

Pseudo code for quaternion rotation:

temporary_rotation.w = cos(Angle/2)

temporary_rotation.x = axis.x * sin(Angle/2)

temporary_rotation.y = axis.y * sin(Angle/2)

temporary_rotation.z = axis.z * sin(Angle/2)

new_permanent_rotation = temporary_ rotation * permanent_rotation

where axis is a unit vector.

6.3.2 ADVANTAGES OF QUATERNIONS

The representation of a rotation as a quaternion (4 numbers) is more compact than the representation as an orthogonal matrix (9 numbers). Furthermore, for a given axis and angle, one can easily construct the corresponding quaternion, and conversely, for a given quaternion one can easily read off the axis and the angle. Both of these are much harder with matrices or Euler angles.

Quaternions also avoid the phenomenon called Gimbal lock. This can be explained intuitively by the fact that a quaternion describes a rotation in one single move while the Euler angles are made of three successive rotations.

Interpolating using spherical linear interpolation between two orientations using quaternions is also the smoothest way to interpolate angles.

Thus quaternions help in improving the performance of 3D object rotation in the PlaneProjection approach.

CONCLUSION

7. CONCLUSION

Understanding the requirements of the 3D world, the project has devised a simple approach to construct 3D models from multiple 2D images. As an initial development, a 3D surface is designed to project the three dimensional object over the 2D screen. The surface is restricted to any object taking up the shape of a parallelepiped. The process takes input from the users which are images of the object at various angles. The frames are resized to the size of the image and they are positioned accordingly. Rotation of the object is accomplished using two methods – plane projection and Euler rotation methods.

An analysis of both the methods is carried out and factors influencing performance are studied and summarized. From the analysis, it can be concluded that with respect to rotation time, the plane projection approach (implemented with the WPF technology of Microsoft) works 55 times faster (on a typical PC) than the Euler Rotation method. There can be slight variations with the jobs scheduled in the processor, which anyway will not influence the time factor in a big way. Zooming of the constructed 3D object is enabled to give an enlarged view of the model. A simple cropper is added with the project, which facilitates clipping of the desired portion of the image that is to be given as input.

To promote the approach to asymmetric objects, a sample model is constructed in a static way. The constraints involved in extending the same technique for asymmetric objects are also studied. Overall, the approach has produced a simple tool, which applies the 3D modelling technique (for parallelepiped surfaces) in a user-friendly way for objects that exhibits symmetry with respect to any two axes.

APPENDIX

APPENDIX

SOURCE CODE

CUBE WITH IMAGES ON 6 SIDES:

cube.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Windows.Media.Imaging;


namespace ImageCubeTrial
{
  public partial class MainPage : UserControl
  {

    public bool myflag = true;

    BitmapImage backbitmap = new BitmapImage();
    BitmapImage topbitmap = new BitmapImage();
    BitmapImage frontbitmap = new BitmapImage();
    BitmapImage bottombitmap = new BitmapImage();
    BitmapImage rightbitmap = new BitmapImage();
    BitmapImage leftbitmap = new BitmapImage();
    int flag = 1;
    DateTime startTime,stopTime ;
    private Point pt;
    public MainPage()
    {
      InitializeComponent();

    }
    void MainPage_Loaded(object sender, RoutedEventArgs e)
```

```
    {
        LayoutRoot.MouseMove += new
MouseEventHandler(LayoutRoot_MouseMove);
        CompositionTarget.Rendering += new
EventHandler(CompositionTarget_Rendering);
        timer();
    }
    void LayoutRoot_MouseMove(object sender, MouseEventArgs e)
    {
        pt = e.GetPosition(LayoutRoot);
    }

    void timer()
    {
        if (flag==1)
        {
            stopTime = DateTime.Now;
            TimeSpan duration = stopTime - startTime;
            MessageBox.Show("seconds:" + duration.TotalSeconds);
            flag = 0;

        }
    }

    void CompositionTarget_Rendering(object sender, EventArgs e)
    {
        if (myflag)
        {
            Image1Projection.RotationY += ((pt.X - (LayoutRoot.ActualWidth / 2)) /
LayoutRoot.ActualWidth) * 10;
            Image2Projection.RotationY += ((pt.X - (LayoutRoot.ActualWidth / 2)) /
LayoutRoot.ActualWidth) * 10;
            Image3Projection.RotationY += ((pt.X - (LayoutRoot.ActualWidth / 2)) /
LayoutRoot.ActualWidth) * 10;
            Image4Projection.RotationY += ((pt.X - (LayoutRoot.ActualWidth / 2)) /
LayoutRoot.ActualWidth) * 10;
            Image5Projection.RotationY += ((pt.X - (LayoutRoot.ActualWidth / 2)) /
LayoutRoot.ActualWidth) * 10;
            Image6Projection.RotationY += ((pt.X - (LayoutRoot.ActualWidth / 2)) /
LayoutRoot.ActualWidth) * 10;

            Image1Projection.RotationX += ((pt.Y - (LayoutRoot.ActualHeight / 2)) /
LayoutRoot.ActualHeight) * 10;
            Image2Projection.RotationX += ((pt.Y - (LayoutRoot.ActualHeight / 2)) /
LayoutRoot.ActualHeight) * 10;
```

```
            Image3Projection.RotationX += ((pt.Y - (LayoutRoot.ActualHeight / 2)) /
LayoutRoot.ActualHeight) * 10;
            Image4Projection.RotationX += ((pt.Y - (LayoutRoot.ActualHeight / 2)) /
LayoutRoot.ActualHeight) * 10;
            Image5Rotation.Angle -= ((pt.Y - (LayoutRoot.ActualHeight / 2)) /
LayoutRoot.ActualHeight) * 10;


            Image6Rotation.Angle += ((pt.Y - (LayoutRoot.ActualHeight / 2)) /
LayoutRoot.ActualHeight) * 10;

        }
    }

    private void imgface_Click(object sender, RoutedEventArgs e)
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();

        openFileDialog.Filter = "All Files (*.*)|*.*";

        bool? userClickedOK = openFileDialog.ShowDialog();

        if (userClickedOK == true)
        {
            System.IO.Stream fileStream = openFileDialog.File.OpenRead();
            frontbitmap.SetSource(fileStream);
            fileStream.Close();
        }

    }


    private void cmdtop_Click(object sender, RoutedEventArgs e)
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();

        openFileDialog.Filter = "All Files (*.*)|*.*";

        bool? userClickedOK = openFileDialog.ShowDialog();

        if (userClickedOK == true)
        {
            System.IO.Stream fileStream = openFileDialog.File.OpenRead();
            topbitmap.SetSource(fileStream);
            this.imgboxtop.Source =topbitmap;
            fileStream.Close();
```

```
    }

  }

  private void cmdright_Click(object sender, RoutedEventArgs e)
  {
     OpenFileDialog openFileDialog = new OpenFileDialog();

     openFileDialog.Filter = "All Files (*.*)|*.*";

     bool? userClickedOK = openFileDialog.ShowDialog();

     if (userClickedOK == true)
     {
        System.IO.Stream fileStream = openFileDialog.File.OpenRead();
        rightbitmap.SetSource(fileStream);
        this.imgboxright.Source = rightbitmap;
        fileStream.Close();
     }

  }

  private void cmdbottom_Click(object sender, RoutedEventArgs e)
  {
     OpenFileDialog openFileDialog = new OpenFileDialog();

     openFileDialog.Filter = "All Files (*.*)|*.*";

     bool? userClickedOK = openFileDialog.ShowDialog();

     if (userClickedOK == true)
     {
        System.IO.Stream fileStream = openFileDialog.File.OpenRead();
        bottombitmap.SetSource(fileStream);
        this.imgboxbottom.Source = bottombitmap;
        fileStream.Close();
     }
  }

  private void cmdback_Click(object sender, RoutedEventArgs e)
  {
     OpenFileDialog openFileDialog = new OpenFileDialog();
     openFileDialog.Filter = "All Files (*.*)|*.*";

     bool? userClickedOK = openFileDialog.ShowDialog();
```

```
    if (userClickedOK == true)
    {
      System.IO.Stream fileStream = openFileDialog.File.OpenRead();
      backbitmap.SetSource(fileStream);
      this.imgboxback.Source = backbitmap;
      fileStream.Close();
    }
  }

  private void cmdfront_Click(object sender, RoutedEventArgs e)
  {
    OpenFileDialog openFileDialog = new OpenFileDialog();

    openFileDialog.Filter = "All Files (*.*)|*.*";

    bool? userClickedOK = openFileDialog.ShowDialog();

    if (userClickedOK == true)
    {
      System.IO.Stream fileStream = openFileDialog.File.OpenRead();
      frontbitmap.SetSource(fileStream);
      this.imgboxfront.Source = frontbitmap;
      fileStream.Close();
    }
  }

  private void cmdleft_Click(object sender, RoutedEventArgs e)
  {
    OpenFileDialog openFileDialog = new OpenFileDialog();

    openFileDialog.Filter = "All Files (*.*)|*.*";

    bool? userClickedOK = openFileDialog.ShowDialog();

    if (userClickedOK == true)
    {
      System.IO.Stream fileStream = openFileDialog.File.OpenRead();
      leftbitmap.SetSource(fileStream);
      this.imgboxleft.Source = leftbitmap;
      fileStream.Close();
    }
  }
```

CUBE ROTATION BY EULER ANGLES:

cubeeulerangles.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;

namespace EulerRotation
{

  public partial class Form1 : Form
  {
    Cube cube;
   public  static string bitmapfile;

    public Form1()
    {
      InitializeComponent();
    }

    private void btnReset_Click(object sender, EventArgs e)
    {
      tX.Value = 0;
      tY.Value = 0;
      tZ.Value = 0;
      render(tX.Value, tY.Value, tZ.Value);


    }


  internal class Math3D
  {
    public class Point3D
    {
      //The Point3D class is rather simple, just keeps track of X Y and Z values,
      //and being a class it can be adjusted to be comparable
      public double X;
```

```
        public double Y;
        public double Z;

        public Point3D(int x, int y, int z)
        {
            X = x;
            Y = y;
            Z = z;
        }

        public Point3D(float x, float y, float z)
        {
            X = (double)x;
            Y = (double)y;
            Z = (double)z;
        }

        public Point3D(double x, double y, double z)
        {
            X = x;
            Y = y;
            Z = z;
        }

        public Point3D()
        {
        }

        public override string ToString()
        {
            return "(" + X.ToString() + ", " + Y.ToString() + ", " + Z.ToString() + ")";
        }
    }

    public class Camera
    {
        //For 3D drawing we need a point of perspective, thus the Camera
        public Point3D Position = new Point3D();
    }

    public static Point3D RotateX(Point3D point2D, double degrees)
    {
        //Here we use Euler's matrix formula for rotating a 3D point x degrees around
the x-axis

        //[ a  b  c ] [ x ]   [ x*a + y*b + z*c ]
```

```
        //[ d  e  f ] [ y ] = [ x*d + y*e + z*f ]
        //[ g  h  i ] [ z ]   [ x*g + y*h + z*i ]

        //[ 1   0       0  ]
        //[ 0   cos(x)  sin(x)]
        //[ 0   -sin(x) cos(x)]

        double cDegrees = (Math.PI * degrees) / 180.0f; //Convert degrees to radian for
.Net Cos/Sin functions
        double cosDegrees = Math.Cos(cDegrees);
        double sinDegrees = Math.Sin(cDegrees);

        double y = (point2D.Y * cosDegrees) + (point2D.Z * sinDegrees);
        double z = (point2D.Y * -sinDegrees) + (point2D.Z * cosDegrees);

        return new Point3D(point2D.X, y, z);
    }

    public static Point3D RotateY(Point3D point2D, double degrees)
    {
        //Y-axis

        //[ cos(x)   0    sin(x)]
        //[  0       1     0  ]
        //[-sin(x)   0    cos(x)]

        double cDegrees = (Math.PI * degrees) / 180.0; //Radians
        double cosDegrees = Math.Cos(cDegrees);
        double sinDegrees = Math.Sin(cDegrees);

        double x = (point2D.X * cosDegrees) + (point2D.Z * sinDegrees);
        double z = (point2D.X * -sinDegrees) + (point2D.Z * cosDegrees);

        return new Point3D(x, point2D.Y, z);
    }

    public static Point3D RotateZ(Point3D point2D, double degrees)
    {
        //Z-axis

        //[ cos(x)  sin(x) 0]
        //[ -sin(x) cos(x) 0]
        //[    0     0    1]

        double cDegrees = (Math.PI * degrees) / 180.0; //Radians
        double cosDegrees = Math.Cos(cDegrees);
```

```
        double sinDegrees = Math.Sin(cDegrees);

        double x = (point2D.X * cosDegrees) + (point2D.Y * sinDegrees);
        double y = (point2D.X * -sinDegrees) + (point2D.Y * cosDegrees);

        return new Point3D(x, y, point2D.Z);
    }

    public static Point3D Translate(Point3D points3D, Point3D oldOrigin, Point3D
newOrigin)
    {
        //Moves a 3D point based on a moved reference point
        Point3D difference = new Point3D(newOrigin.X - oldOrigin.X, newOrigin.Y -
oldOrigin.Y, newOrigin.Z - oldOrigin.Z);
        points3D.X += difference.X;
        points3D.Y += difference.Y;
        points3D.Z += difference.Z;
        return points3D;
    }

    //These are to make the above functions workable with arrays of 3D points
    public static Point3D[] RotateX(Point3D[] points3D, double degrees)
    {
        for (int i = 0; i < points3D.Length; i++)
        {
            points3D[i] = RotateX(points3D[i], degrees);
        }
        return points3D;
    }

    public static Point3D[] RotateY(Point3D[] points3D, double degrees)
    {
        for (int i = 0; i < points3D.Length; i++)
        {
            points3D[i] = RotateY(points3D[i], degrees);
        }
        return points3D;
    }

    public static Point3D[] RotateZ(Point3D[] points3D, double degrees)
    {
        for (int i = 0; i < points3D.Length; i++)
        {
            points3D[i] = RotateZ(points3D[i], degrees);
        }
        return points3D;
```

```
    }

    public static Point3D[] Translate(Point3D[] points3D, Point3D oldOrigin, Point3D
newOrigin)
    {
      for (int i = 0; i < points3D.Length; i++)
      {
        points3D[i] = Translate(points3D[i], oldOrigin, newOrigin);
      }
      return points3D;
    }

  }

  internal class Cube
  {
    //Example cube class to demonstrate the use of 3D points and 3D rotation

    public int width = 0;
    public int height = 0;
    public int depth = 0;

    double xRotation = 0.0;
    double yRotation = 0.0;
    double zRotation = 0.0;

    Math3D.Camera camera1 = new Math3D.Camera();
    Math3D.Point3D cubeOrigin;

    public double RotateX
    {
      get { return xRotation; }
      set { xRotation = value; }
    }

    public double RotateY
    {
      get { return yRotation; }
      set { yRotation = value; }
    }

    public double RotateZ
    {
      get { return zRotation; }
      set { zRotation = value; }
    }
```

```
public Cube(int side)
{
   width = side;
   height = side;
   depth = side;
   cubeOrigin = new Math3D.Point3D(width / 2, height / 2, depth / 2);
}

public Cube(int side, Math3D.Point3D origin)
{
   width = side;
   height = side;
   depth = side;
   cubeOrigin = origin;
}

public Cube(int Width, int Height, int Depth)
{
   width = Width;
   height = Height;
   depth = Depth;
   cubeOrigin = new Math3D.Point3D(width / 2, height / 2, depth / 2);
}

public Cube(int Width, int Height, int Depth, Math3D.Point3D origin)
{
   width = Width;
   height = Height;
   depth = Depth;
   cubeOrigin = origin;
}

//Finds the othermost points. Used so when the cube is drawn on a bitmap,
//the bitmap will be the correct size
static Rectangle getBounds(PointF[] points)
{
   double left = points[0].X;
   double right = points[0].X;
   double top = points[0].Y;
   double bottom = points[0].Y;
   for (int i = 1; i < points.Length; i++)
   {
      if (points[i].X < left)
         left = points[i].X;
      if (points[i].X > right)
```

```
            right = points[i].X;
          if (points[i].Y < top)
            top = points[i].Y;
          if (points[i].Y > bottom)
            bottom = points[i].Y;
        }

        return new Rectangle(0, 0, (int)Math.Round(right - left),
(int)Math.Round(bottom - top));
      }

      public Bitmap drawCube(Point drawOrigin)
      {
        //FRONT FACE
        //Top Left - 7
        //Top Right - 4
        //Bottom Left - 6
        //Bottom Right - 5

        //Vars
        PointF[] point2D = new PointF[24]; //Will be actual 2D drawing points
        Point tmpOrigin = new Point(0, 0);

        Math3D.Point3D point0 = new Math3D.Point3D(0, 0, 0); //Used for reference

        //Zoom factor is set with the monitor width to keep the cube from being
 distorted
        double zoom = (double)Screen.PrimaryScreen.Bounds.Width / 1.5;

        //Set up the cube
        Math3D.Point3D[] cubePoints = fillCubeVertices(width, height, depth);
        Math3D.Point3D[] sPoints = fillCubeVertices(width, height, depth);

        //Calculate the camera Z position to stay constant despite rotation
        Math3D.Point3D anchorPoint = (Math3D.Point3D)cubePoints[4]; //anchor
 points
        double cameraZ = 1000; //-(((anchorPoint.X - cubeOrigin.X) * zoom) /
 cubeOrigin.X) + anchorPoint.Z;
       // MessageBox.Show(""+cameraZ);

        camera1.Position = new Math3D.Point3D(cubeOrigin.X, cubeOrigin.Y,
 cameraZ);

        //Apply Rotations, moving the cube to a corner then back to middle
        cubePoints = Math3D.Translate(cubePoints, cubeOrigin, point0);
        cubePoints = Math3D.RotateX(cubePoints, xRotation); //The order of these
```

```
        cubePoints = Math3D.RotateY(cubePoints, yRotation); //rotations is the source
        cubePoints = Math3D.RotateZ(cubePoints, zRotation); //of Gimbal Lock
        cubePoints = Math3D.Translate(cubePoints, point0, cubeOrigin);
        Math3D.Point3D vec; int nx, ny;

        //Convert 3D Points to 2D

        for (int i = 0; i < point2D.Length; i++)
        {
           vec = cubePoints[i];
           if (vec.Z - camera1.Position.Z >= 0)
           {
              point2D[i].X = (int)((double)-(vec.X - camera1.Position.X) / (-0.1f) *
zoom) + drawOrigin.X;
              point2D[i].Y = (int)((double)(vec.Y - camera1.Position.Y) / (-0.1f) *
zoom) + drawOrigin.Y;
           }
           else
           {
            // MessageBox.Show("i : " +i +" X: " + vec.X + " Y: " + vec.Y + " Z: " +
vec.Z);
              point2D[i].X =(int)( (float)((vec.X - camera1.Position.X) / (vec.Z -
camera1.Position.Z) * zoom + drawOrigin.X));
              point2D[i].Y = (int) ( (float)(-(vec.Y - camera1.Position.Y) / (vec.Z -
camera1.Position.Z) * zoom + drawOrigin.Y));

           }
        }


        //Now to plot out the points
        Rectangle bounds = getBounds(point2D);
        bounds.Width += drawOrigin.X;
        bounds.Height += drawOrigin.Y;

        Bitmap tmpBmp = new Bitmap(bounds.Width, bounds.Height);
        Graphics g = Graphics.FromImage(tmpBmp);


        Bitmap bmp = new Bitmap(Form1.bitmapfile);

        int x,y,z;

        // Back Face
           for (z = (int)sPoints[0].Z; z <=sPoints[2].Z; z++)
           {
```

```
            for (y = (int)sPoints[0].Y; y <= sPoints[2].Y; y++)
            {
               for (x = (int)sPoints[0].X; x <= sPoints[2].X; x++)
               {
                  Color c;
                  if (x < bmp.Width && y < bmp.Height)
                     c = bmp.GetPixel(x, y);
                  else
                     c = Color.White;


                  Math3D.Point3D p = new Math3D.Point3D(x, y, z);

                  p = Math3D.Translate(p, cubeOrigin, point0);
                  p = Math3D.RotateX(p, xRotation); //The order of these
                  p = Math3D.RotateY(p, yRotation); //rotations is the source
                  p = Math3D.RotateZ(p, zRotation); //of Gimbal Lock
                  p = Math3D.Translate(p, point0, cubeOrigin);
                  vec = p;
                  if (vec.Z - camera1.Position.Z >= 0)
                  {
                     nx = (int)((double)-(vec.X - camera1.Position.X) / (-0.1f) * zoom)
+ drawOrigin.X;
                     ny = (int)((double)(vec.Y - camera1.Position.Y) / (-0.1f) * zoom) +
drawOrigin.Y;
                  }
                  else
                  {
                     nx = (int)((float)((vec.X - camera1.Position.X) / (vec.Z -
camera1.Position.Z) * zoom + drawOrigin.X));
                     ny = (int)((float)(-(vec.Y - camera1.Position.Y) / (vec.Z -
camera1.Position.Z) * zoom + drawOrigin.Y));
                  }

                  DrawPixel(g, new Pen(c, 2.0F), new PointF(nx, ny));
               }
            }
         }

         //// Front Face
         for (z = (int)sPoints[4].Z; z <= sPoints[6].Z; z++)
         {
            for (y = (int)sPoints[4].Y; y < sPoints[6].Y; y++)
            {
               for (x = (int)sPoints[4].X; x < sPoints[6].X; x++)
               {
```

```
                    Color c = bmp.GetPixel(x, y);

                    Math3D.Point3D p = new Math3D.Point3D(x, y, z);

                    p = Math3D.Translate(p, cubeOrigin, point0);
                    p = Math3D.RotateX(p, xRotation); //The order of these
                    p = Math3D.RotateY(p, yRotation); //rotations is the source
                    p = Math3D.RotateZ(p, zRotation); //of Gimbal Lock
                    p = Math3D.Translate(p, point0, cubeOrigin);
                    vec = p;
                    if (vec.Z - camera1.Position.Z >= 0)
                    {
                       nx = (int)((double)-(vec.X - camera1.Position.X) / (-0.1f) * zoom)
+ drawOrigin.X;
                       ny = (int)((double)(vec.Y - camera1.Position.Y) / (-0.1f) * zoom) +
drawOrigin.Y;
                    }
                    else
                    {
                       nx = (int)((float)((vec.X - camera1.Position.X) / (vec.Z -
camera1.Position.Z) * zoom + drawOrigin.X));
                       ny = (int)((float)(-(vec.Y - camera1.Position.Y) / (vec.Z -
camera1.Position.Z) * zoom + drawOrigin.Y));
                    }


                    DrawPixel(g, new Pen(c, 2.0F), new PointF(nx, ny));


                }
             }
          }


       for (int f = 8; f < 16; f += 4)
       {

          for ( z = (int)sPoints[f].Z; z <= sPoints[f + 2].Z; z++)
          {

             for ( y = (int)sPoints[f].Y; y <= sPoints[f + 2].Y; y++)
             {


                for (x = (int)sPoints[f].X; x <= sPoints[f + 2].X; x++)
                {
```

```
                    Color c;
                    if (y < bmp.Height && z < bmp.Width)
                      c = bmp.GetPixel( y,z);
                    else
                      c = Color.White;
                    Math3D.Point3D p = new Math3D.Point3D(x, y, z);

                    p = Math3D.Translate(p, cubeOrigin, point0);
                    p = Math3D.RotateX(p, xRotation); //The order of these
                    p = Math3D.RotateY(p, yRotation); //rotations is the source
                    p = Math3D.RotateZ(p, zRotation); //of Gimbal Lock
                    p = Math3D.Translate(p, point0, cubeOrigin);
                    vec = p;
                    if (vec.Z - camera1.Position.Z >= 0)
                    {
                      nx = (int)((double)-(vec.X - camera1.Position.X) / (-0.1f) * zoom)
+ drawOrigin.X;
                      ny = (int)((double)(vec.Y - camera1.Position.Y) / (-0.1f) * zoom) +
drawOrigin.Y;
                    }
                    else
                    {
                      nx = (int)((float)((vec.X - camera1.Position.X) / (vec.Z -
camera1.Position.Z) * zoom + drawOrigin.X));
                      ny = (int)((float)(-(vec.Y - camera1.Position.Y) / (vec.Z -
camera1.Position.Z) * zoom + drawOrigin.Y));
                    }



                      DrawPixel(g, new Pen(c, 2.0F), new PointF(nx, ny));

                  }
                }
              }
            }



            for (int f = 16; f < 24; f += 4)
            {

              for (z = (int)sPoints[f].Z; z <= sPoints[f + 2].Z; z++)
              {

                for (y = (int)sPoints[f].Y; y <= sPoints[f + 2].Y; y++)
```

```
            {


                for (x = (int)sPoints[f].X; x <= sPoints[f + 2].X; x++)
                {
                    Color c;
                    if (x < bmp.Width && z < bmp.Width)
                        c = bmp.GetPixel(x,z);
                    else
                        c = Color.White;
                    Math3D.Point3D p = new Math3D.Point3D(x, y, z);

                    p = Math3D.Translate(p, cubeOrigin, point0);
                    p = Math3D.RotateX(p, xRotation); //The order of these
                    p = Math3D.RotateY(p, yRotation); //rotations is the source
                    p = Math3D.RotateZ(p, zRotation); //of Gimbal Lock
                    p = Math3D.Translate(p, point0, cubeOrigin);
                    vec = p;
                    if (vec.Z - camera1.Position.Z >= 0)
                    {
                        nx = (int)((double)-(vec.X - camera1.Position.X) / (-0.1f) * zoom)
+ drawOrigin.X;
                        ny = (int)((double)(vec.Y - camera1.Position.Y) / (-0.1f) * zoom) +
drawOrigin.Y;
                    }
                    else
                    {
                        nx = (int)((float)((vec.X - camera1.Position.X) / (vec.Z -
camera1.Position.Z) * zoom + drawOrigin.X));
                        ny = (int)((float)(-(vec.Y - camera1.Position.Y) / (vec.Z -
camera1.Position.Z) * zoom + drawOrigin.Y));
                    }


                    DrawPixel(g, new Pen(c,2.0F), new PointF(nx, ny));
                }
            }
        }
    }


    //Back Face
    g.DrawLine(Pens.Red, point2D[0], point2D[1]);
    g.DrawLine(Pens.Red, point2D[1], point2D[2]);
    g.DrawLine(Pens.Red, point2D[2], point2D[3]);
    g.DrawLine(Pens.Red, point2D[3], point2D[0]);
```

```
    //Front Face
    g.DrawLine(Pens.Blue, point2D[4], point2D[5]);
    g.DrawLine(Pens.Blue, point2D[5], point2D[6]);
    g.DrawLine(Pens.Blue, point2D[6], point2D[7]);
    g.DrawLine(Pens.Blue, point2D[7], point2D[4]);

    //Right Face
    g.DrawLine(Pens.Blue, point2D[8], point2D[9]);
    g.DrawLine(Pens.Blue, point2D[9], point2D[10]);
    g.DrawLine(Pens.Blue, point2D[10], point2D[11]);
    g.DrawLine(Pens.Red, point2D[11], point2D[8]);

    //Left Face
    g.DrawLine(Pens.Blue, point2D[12], point2D[13]);
    g.DrawLine(Pens.Blue, point2D[13], point2D[14]);
    g.DrawLine(Pens.Blue, point2D[14], point2D[15]);
    g.DrawLine(Pens.Red, point2D[15], point2D[12]);

    //Bottom Face
    g.DrawLine(Pens.Blue, point2D[16], point2D[17]);
    g.DrawLine(Pens.Blue, point2D[17], point2D[18]);
    g.DrawLine(Pens.Blue, point2D[18], point2D[19]);
    g.DrawLine(Pens.Red, point2D[19], point2D[16]);

    //Top Face
    g.DrawLine(Pens.Blue, point2D[20], point2D[21]);
    g.DrawLine(Pens.Blue, point2D[21], point2D[22]);
    g.DrawLine(Pens.Blue, point2D[22], point2D[23]);
    g.DrawLine(Pens.Red, point2D[23], point2D[20]);

    g.Dispose(); //Clean-up

    return tmpBmp;
}

public void DrawPixel(Graphics g, Pen color,PointF p)
{
    g.DrawLine(color, p, new PointF(p.X + 1, p.Y + 1));
}


public static Math3D.Point3D[] fillCubeVertices(int width, int height, int depth)
{
    Math3D.Point3D[] verts = new Math3D.Point3D[24];
```

```
        //front face
        verts[0] = new Math3D.Point3D(0, 0, 0);
        verts[1] = new Math3D.Point3D(0, height, 0);
        verts[2] = new Math3D.Point3D(width, height, 0);
        verts[3] = new Math3D.Point3D(width, 0, 0);

        //back face
        verts[4] = new Math3D.Point3D(0, 0, depth);
        verts[5] = new Math3D.Point3D(0, height, depth);
        verts[6] = new Math3D.Point3D(width, height, depth);
        verts[7] = new Math3D.Point3D(width, 0, depth);

        //left face
        verts[8] = new Math3D.Point3D(0, 0, 0);
        verts[9] = new Math3D.Point3D(0, height, 0);
        verts[10] = new Math3D.Point3D(0, height, depth);
        verts[11] =  new Math3D.Point3D(0, 0, depth);

        //right face
        verts[12] = new Math3D.Point3D(width, 0, 0);
        verts[13] = new Math3D.Point3D(width, 0, depth);
        verts[14] = new Math3D.Point3D(width, height, depth);
        verts[15] = new Math3D.Point3D(width, height, 0);

        //top face
        verts[16] = new Math3D.Point3D(0, height, 0);
        verts[17] = new Math3D.Point3D(0, height, depth);
        verts[18] = new Math3D.Point3D(width, height, depth);
        verts[19] = new Math3D.Point3D(width, height, 0);

        //bottom face
        verts[20] = new Math3D.Point3D(0, 0, 0);
        verts[21] = new Math3D.Point3D(0, 0, depth);
        verts[22] = new Math3D.Point3D(width, 0, depth);
        verts[23] = new Math3D.Point3D(width, 0, 0);

        return verts;
    }
  }
}
```

cubeeuleranglesdesigner.cs

```
namespace EulerRotation
{
  partial class Form1
```

```
{
    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    private void InitializeComponent()
    {
        this.btnReset = new System.Windows.Forms.Button();
        this.tZ = new System.Windows.Forms.TrackBar();
        this.tY = new System.Windows.Forms.TrackBar();
        this.tX = new System.Windows.Forms.TrackBar();
        this.label3 = new System.Windows.Forms.Label();
        this.label2 = new System.Windows.Forms.Label();
        this.label1 = new System.Windows.Forms.Label();
        this.picCube = new System.Windows.Forms.PictureBox();
        this.timer1 = new System.Windows.Forms.Timer();
        this.btnRotate = new System.Windows.Forms.Button();
        this.txtX = new System.Windows.Forms.TextBox();
        this.txtZ = new System.Windows.Forms.TextBox();
        this.txtY = new System.Windows.Forms.TextBox();
        this.button1 = new System.Windows.Forms.Button();
        this.label4 = new System.Windows.Forms.Label();
        this.label5 = new System.Windows.Forms.Label();
        this.label6 = new System.Windows.Forms.Label();
        this.label7 = new System.Windows.Forms.Label();
        this.pictureBox1 = new System.Windows.Forms.PictureBox();
        this.SuspendLayout();
        //
        // btnReset
        //
        this.btnReset.Location = new System.Drawing.Point(311, 294);
        this.btnReset.Name = "btnReset";
        this.btnReset.Size = new System.Drawing.Size(47, 21);
        this.btnReset.TabIndex = 21;
        this.btnReset.Text = "Reset";
        this.btnReset.UseVisualStyleBackColor = true;
        this.btnReset.Click += new System.EventHandler(this.btnReset_Click);
        //
        // tZ
        //
```

```
this.tZ.AutoSize = false;
this.tZ.Location = new System.Drawing.Point(311, 269);
this.tZ.Maximum = 360;
this.tZ.Name = "tZ";
this.tZ.Size = new System.Drawing.Size(306, 19);
this.tZ.TabIndex = 20;
this.tZ.TickStyle = System.Windows.Forms.TickStyle.None;
this.tZ.Scroll += new System.EventHandler(this.tZ_Scroll);
//
// tY
//
this.tY.AutoSize = false;
this.tY.Location = new System.Drawing.Point(311, 244);
this.tY.Maximum = 360;
this.tY.Name = "tY";
this.tY.Size = new System.Drawing.Size(306, 19);
this.tY.TabIndex = 19;
this.tY.TickStyle = System.Windows.Forms.TickStyle.None;
this.tY.Scroll += new System.EventHandler(this.tY_Scroll);
//
// tX
//
this.tX.AutoSize = false;
this.tX.Location = new System.Drawing.Point(311, 219);
this.tX.Maximum = 360;
this.tX.Name = "tX";
this.tX.Size = new System.Drawing.Size(306, 19);
this.tX.SmallChange = 10;
this.tX.TabIndex = 18;
this.tX.TickStyle = System.Windows.Forms.TickStyle.None;
this.tX.Scroll += new System.EventHandler(this.tX_Scroll);
//
// label3
//
this.label3.AutoSize = true;
this.label3.Location = new System.Drawing.Point(296, 273);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(17, 13);
this.label3.TabIndex = 17;
this.label3.Text = "Z:";
//
// label2
//
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(296, 247);
this.label2.Name = "label2";
```

```
this.label2.Size = new System.Drawing.Size(17, 13);
this.label2.TabIndex = 16;
this.label2.Text = "Y:";
//
// label1
//
this.label1.AutoSize = true;
this.label1.Location = new System.Drawing.Point(296, 221);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(17, 13);
this.label1.TabIndex = 15;
this.label1.Text = "X:";
//
// picCube
//
this.picCube.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
this.picCube.Location = new System.Drawing.Point(12, 25);
this.picCube.Name = "picCube";
this.picCube.Size = new System.Drawing.Size(275, 275);
this.picCube.TabIndex = 14;
this.picCube.TabStop = false;
//
// btnRotate
//
this.btnRotate.Location = new System.Drawing.Point(688, 347);
this.btnRotate.Name = "btnRotate";
this.btnRotate.Size = new System.Drawing.Size(97, 37);
this.btnRotate.TabIndex = 23;
this.btnRotate.Text = "Rotate";
this.btnRotate.UseVisualStyleBackColor = true;
this.btnRotate.Click += new System.EventHandler(this.btnRotate_Click);
//
// txtX
//
this.txtX.Location = new System.Drawing.Point(722, 238);
this.txtX.Name = "txtX";
this.txtX.Size = new System.Drawing.Size(81, 20);
this.txtX.TabIndex = 24;
//
// txtZ
//
this.txtZ.Location = new System.Drawing.Point(722, 305);
this.txtZ.Name = "txtZ";
this.txtZ.Size = new System.Drawing.Size(81, 20);
this.txtZ.TabIndex = 25;
//
```

```
        // txtY
        //
        this.txtY.Location = new System.Drawing.Point(722, 269);
        this.txtY.Name = "txtY";
        this.txtY.Size = new System.Drawing.Size(81, 20);
        this.txtY.TabIndex = 26;
        //
        // button1
        //
        this.button1.Location = new System.Drawing.Point(362, 70);
        this.button1.Name = "button1";
        this.button1.Size = new System.Drawing.Size(129, 48);
        this.button1.TabIndex = 27;
        this.button1.Text = "Load Image";
        this.button1.UseVisualStyleBackColor = true;
        this.button1.Click += new System.EventHandler(this.button1_Click);
        //
        // label4
        //
        this.label4.AutoSize = true;
        this.label4.Font = new System.Drawing.Font("Microsoft Sans Serif", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
        this.label4.Location = new System.Drawing.Point(682, 238);
        this.label4.Name = "label4";
        this.label4.Size = new System.Drawing.Size(34, 24);
        this.label4.TabIndex = 28;
        this.label4.Text = "X: ";
        //
        // label5
        //
        this.label5.AutoSize = true;
        this.label5.Font = new System.Drawing.Font("Microsoft Sans Serif", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
        this.label5.Location = new System.Drawing.Point(684, 269);
        this.label5.Name = "label5";
        this.label5.Size = new System.Drawing.Size(32, 24);
        this.label5.TabIndex = 29;
        this.label5.Text = "Y: ";
        //
        // label6
        //
        this.label6.AutoSize = true;
        this.label6.Font = new System.Drawing.Font("Microsoft Sans Serif", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
        this.label6.Location = new System.Drawing.Point(684, 305);
        this.label6.Name = "label6";
```

```
        this.label6.Size = new System.Drawing.Size(27, 24);
        this.label6.TabIndex = 30;
        this.label6.Text = "Z:";
        //
        // label7
        //
        this.label7.AutoSize = true;
        this.label7.Font = new System.Drawing.Font("Microsoft Sans Serif", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
        this.label7.Location = new System.Drawing.Point(684, 198);
        this.label7.Name = "label7";
        this.label7.Size = new System.Drawing.Size(106, 24);
        this.label7.TabIndex = 31;
        this.label7.Text = "Angle Input";
        //
        // pictureBox1
        //
        this.pictureBox1.Location = new System.Drawing.Point(526, 25);
        this.pictureBox1.Name = "pictureBox1";
        this.pictureBox1.Size = new System.Drawing.Size(157, 134);
        this.pictureBox1.TabIndex = 32;
        this.pictureBox1.TabStop = false;
        //
        // Form1
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(871, 438);
        this.Controls.Add(this.pictureBox1);
        this.Controls.Add(this.label7);
        this.Controls.Add(this.label6);
        this.Controls.Add(this.label5);
        this.Controls.Add(this.label4);
        this.Controls.Add(this.button1);
        this.Controls.Add(this.txtY);
        this.Controls.Add(this.txtZ);
        this.Controls.Add(this.txtX);
        this.Controls.Add(this.btnRotate);
        this.Controls.Add(this.picCube);
        this.Controls.Add(this.btnReset);
        this.Controls.Add(this.tZ);
        this.Controls.Add(this.tY);
        this.Controls.Add(this.tX);
        this.Controls.Add(this.label3);
        this.Controls.Add(this.label2);
        this.Controls.Add(this.label1);
```

```
        this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
        this.MaximizeBox = false;
        this.Name = "Form1";
        this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
        this.Text = "3D Matrix Method Rotation ";
        this.Load += new System.EventHandler(this.Form1_Load);
        this.ResumeLayout(false);
        this.PerformLayout();

    }

    #endregion

    private System.Windows.Forms.Button btnReset;
    private System.Windows.Forms.TrackBar tZ;
    private System.Windows.Forms.TrackBar tY;
    private System.Windows.Forms.TrackBar tX;
    private System.Windows.Forms.Label label3;
    private System.Windows.Forms.Label label2;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.PictureBox picCube;
    private System.Windows.Forms.Timer timer1;
    private System.Windows.Forms.Button btnRotate;
    private System.Windows.Forms.TextBox txtX;
    private System.Windows.Forms.TextBox txtZ;
    private System.Windows.Forms.TextBox txtY;
    private System.Windows.Forms.Button button1;
    private System.Windows.Forms.Label label4;
    private System.Windows.Forms.Label label5;
    private System.Windows.Forms.Label label6;
    private System.Windows.Forms.Label label7;
    private System.Windows.Forms.PictureBox pictureBox1;
  }
}
```

ProcessorTime.cs

```
using System;
using System.Diagnostics;

namespace ProcessSample
{
  class ProcessMonitorSample
  {
    public static void Main()
```

```
    {

        // Define variables to track the peak
        // memory usage of the process.
        long peakPagedMem = 0,
           peakWorkingSet = 0,
           peakVirtualMem = 0;

        Process myProcess = null;

        try
        {
           // Start the process.
           myProcess = Process.Start("D:/Paper and Projects/Final Year Proj/Full
Project/3DImageRotation/bin/Debug/EulerRotation.exe");



           // Display the process statistics until
           // the user closes the program.
           do
           {
              if (!myProcess.HasExited)
              {
                 // Refresh the current process property values.
                 myProcess.Refresh();
                 Console.WriteLine();
                 // Display current process statistics.
Console.WriteLine("{0} -", myProcess.ToString());
                 Console.WriteLine("-----------------------------------");

                 Console.WriteLine("  user processor time: {0}",
                    myProcess.UserProcessorTime);
                 // Update the values for the overall peak memory statistics.
                 peakPagedMem = myProcess.PeakPagedMemorySize64;
                 peakVirtualMem = myProcess.PeakVirtualMemorySize64;
                 peakWorkingSet = myProcess.PeakWorkingSet64;

                 if (myProcess.Responding)
                 {
                    Console.WriteLine("Status = Running");
                 }
                 else
                 {
                    Console.WriteLine("Status = Not Responding");
                 }
```

```
            }
        }
        while (!myProcess.WaitForExit(1000));
        Console.WriteLine();
        Console.WriteLine("Process exit code: {0}",
            myProcess.ExitCode);

        // Display peak memory statistics for the process.
        Console.WriteLine("Peak physical memory usage of the process: {0}",
            peakWorkingSet);
        Console.WriteLine("Peak paged memory usage of the process: {0}",
            peakPagedMem);
        Console.WriteLine("Peak virtual memory usage of the process: {0}",
            peakVirtualMem);

      }
      finally
      {
        if (myProcess != null)
        {
          myProcess.Close();
        }
      }
    }

  }
}
```

StaticModel.cs

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Windows.Threading;
using System.ComponentModel;

namespace _Dto3DImages
{
  public class RoundBox3D : Panel
```

```
{
    private Point mouseStart, mouseNow;

    private FrameworkElement front_el, reflex_el, colorArt_el, bwArt_el, back_el,
engrave1_el, engrave2_el, engrave3_el;
    private FrameworkElement artists_el, controlSticker_el, arrows_el;
    private FrameworkElement left_el, leftb1_el, leftb2_el, leftb3_el, leftb4_el,
leftb5_el, leftt1_el, leftt2_el, leftt3_el, leftt4_el, leftt5_el;
    private FrameworkElement right_el, rightb1_el, rightb2_el, rightb3_el, rightb4_el,
rightb5_el, rightt1_el, rightt2_el, rightt3_el, rightt4_el, rightt5_el;

    private DispatcherTimer clock;

    #region Dependency Properties

    public static readonly DependencyProperty RotateYProperty =
DependencyProperty.Register("RotateY", typeof(double), typeof(RoundBox3D), new
PropertyMetadata(RoundBox3D.RotateYChanged));
    public static readonly DependencyProperty IsDragOnProperty =
DependencyProperty.Register("IsDragOn", typeof(bool), typeof(RoundBox3D), new
PropertyMetadata(RoundBox3D.IsDragChanged));
    public static readonly DependencyProperty DistanceProperty =
DependencyProperty.Register("Distance", typeof(double), typeof(RoundBox3D), new
PropertyMetadata(RoundBox3D.DistanceChanged));

    private static void RotateYChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e) { ((RoundBox3D)sender).Refresh(); }
    private static void IsDragChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e) {
((RoundBox3D)sender).OnIsDragChanged(e); }
    private static void DistanceChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e) { ((RoundBox3D)sender).Refresh(); }

    public static readonly DependencyProperty FrontProperty =
DependencyProperty.Register("Front", typeof(ImageSource), typeof(RoundBox3D),
new PropertyMetadata(RoundBox3D.FrontChanged));
    public static readonly DependencyProperty ReflexProperty =
DependencyProperty.Register("Reflex", typeof(ImageSource), typeof(RoundBox3D),
new PropertyMetadata(RoundBox3D.ReflexChanged));
```

```
    public static readonly DependencyProperty BackProperty =
DependencyProperty.Register("Back", typeof(ImageSource), typeof(RoundBox3D),
new PropertyMetadata(RoundBox3D.BackChanged));
    public static readonly DependencyProperty LeftProperty =
DependencyProperty.Register("Left", typeof(ImageSource), typeof(RoundBox3D),
new PropertyMetadata(RoundBox3D.LeftChanged));
    public static readonly DependencyProperty Leftb1Property =
DependencyProperty.Register("Leftb1", typeof(ImageSource), typeof(RoundBox3D),
new PropertyMetadata(RoundBox3D.Leftb1Changed));
    public static readonly DependencyProperty Leftt1Property =
DependencyProperty.Register("Leftt1", typeof(ImageSource), typeof(RoundBox3D),
new PropertyMetadata(RoundBox3D.Leftt1Changed));
    public static readonly DependencyProperty Rightb1Property =
DependencyProperty.Register("Rightb1", typeof(ImageSource), typeof(RoundBox3D),
new PropertyMetadata(RoundBox3D.Rightb1Changed));
    public static readonly DependencyProperty Rightt1Property =
DependencyProperty.Register("Rightt1", typeof(ImageSource), typeof(RoundBox3D),
new PropertyMetadata(RoundBox3D.Rightt1Changed));
    public static readonly DependencyProperty RightProperty =
DependencyProperty.Register("Right", typeof(ImageSource), typeof(RoundBox3D),
new PropertyMetadata(RoundBox3D.RightChanged));



    private static void FrontChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e) { ((RoundBox3D)sender).Refresh(); }
    private static void ReflexChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e) { ((RoundBox3D)sender).Refresh(); }
    private static void BackChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e) { ((RoundBox3D)sender).Refresh(); }
    private static void LeftChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e) { ((RoundBox3D)sender).Refresh(); }
    private static void Leftb1Changed(DependencyObject sender,
DependencyPropertyChangedEventArgs e) { ((RoundBox3D)sender).Refresh(); }
    private static void Leftt1Changed(DependencyObject sender,
DependencyPropertyChangedEventArgs e) { ((RoundBox3D)sender).Refresh(); }
    private static void Rightb1Changed(DependencyObject sender,
DependencyPropertyChangedEventArgs e) { ((RoundBox3D)sender).Refresh(); }
    private static void Rightt1Changed(DependencyObject sender,
DependencyPropertyChangedEventArgs e) { ((RoundBox3D)sender).Refresh(); }
    private static void RightChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e) { ((RoundBox3D)sender).Refresh(); }

    // Getters and setters of dependency properties
    [Category("RoundBox 3D")]
    public double RotateY
```

```
{
    get { return (double)this.GetValue(RoundBox3D.RotateYProperty); }
    set { this.SetValue(RoundBox3D.RotateYProperty, value); }
}

[Category("RoundBox 3D")]
public bool IsDragOn
{
    get { return (bool)this.GetValue(RoundBox3D.IsDragOnProperty); }
    set { this.SetValue(RoundBox3D.IsDragOnProperty, value); }
}

[Category("RoundBox 3D")]
public double Distance
{
    get { return (double)this.GetValue(RoundBox3D.DistanceProperty); }
    set { this.SetValue(RoundBox3D.DistanceProperty, value); }
}

[Category("RoundBox 3D Faces")]
public ImageSource Front
{
    get { return (ImageSource)this.GetValue(RoundBox3D.FrontProperty); }
    set { this.SetValue(RoundBox3D.FrontProperty, value); }
}

[Category("RoundBox 3D Faces")]
public ImageSource Reflex
{
    get { return (ImageSource)this.GetValue(RoundBox3D.ReflexProperty); }
    set { this.SetValue(RoundBox3D.ReflexProperty, value); }
}

[Category("RoundBox 3D Faces")]
public ImageSource Back
{
    get { return (ImageSource)this.GetValue(RoundBox3D.BackProperty); }
    set { this.SetValue(RoundBox3D.BackProperty, value); }
}

[Category("RoundBox 3D Faces")]
public ImageSource Left
{
    get { return (ImageSource)this.GetValue(RoundBox3D.LeftProperty); }
    set { this.SetValue(RoundBox3D.LeftProperty, value); }
}
```

```
    [Category("RoundBox 3D Faces")]
    public ImageSource Leftb1
    {
      get { return (ImageSource)this.GetValue(RoundBox3D.Leftb1Property); }
      set { this.SetValue(RoundBox3D.Leftb1Property, value); }
    }

    [Category("RoundBox 3D Faces")]
    public ImageSource Leftt1
    {
      get { return (ImageSource)this.GetValue(RoundBox3D.Leftt1Property); }
      set { this.SetValue(RoundBox3D.Leftt1Property, value); }
    }

    [Category("RoundBox 3D Faces")]
    public ImageSource Right
    {
      get { return (ImageSource)this.GetValue(RoundBox3D.RightProperty); }
      set { this.SetValue(RoundBox3D.RightProperty, value); }
    }

    [Category("RoundBox 3D Faces")]
    public ImageSource Rightb1
    {
      get { return (ImageSource)this.GetValue(RoundBox3D.Rightb1Property); }
      set { this.SetValue(RoundBox3D.Rightb1Property, value); }
    }

    [Category("RoundBox 3D Faces")]
    public ImageSource Rightt1
    {
      get { return (ImageSource)this.GetValue(RoundBox3D.Rightt1Property); }
      set { this.SetValue(RoundBox3D.Rightt1Property, value); }
    }

    private void OnIsDragChanged(DependencyPropertyChangedEventArgs e)
    {
      this.MouseLeftButtonDown += new
MouseButtonEventHandler(this.RoundBox3D_MouseLeftButtonDown);
      this.MouseLeftButtonUp += new
MouseButtonEventHandler(this.RoundBox3D_MouseLeftButtonUp);

    }

    #endregion
```

```
    #region Mouse

    private void RoundBox3D_MouseLeftButtonDown(object sender,
MouseButtonEventArgs e)
    {

      this.CaptureMouse();
      this.mouseStart = e.GetPosition(this);
      this.mouseNow = this.mouseStart;

      if (this.clock == null)
      {
        this.clock = new DispatcherTimer();
        this.clock.Interval = new TimeSpan(0, 0, 0, 0, 10);
        this.clock.Start();
        this.clock.Tick += new EventHandler(this.Clock_Tick);
        this.clock.Start();
      }
      else
      {
        this.clock.Stop();
        this.clock.Start();
      }

      this.MouseMove -= new MouseEventHandler(this.RoundBox3D_MouseMove);
      this.MouseMove += new
MouseEventHandler(this.RoundBox3D_MouseMove);
    }

    private void RoundBox3D_MouseLeftButtonUp(object sender,
MouseButtonEventArgs e)
    {
      this.ReleaseMouseCapture();
      this.MouseMove -= new MouseEventHandler(this.RoundBox3D_MouseMove);
    }

    private void RoundBox3D_MouseMove(object sender, MouseEventArgs e)
    {

      if (!this.clock.IsEnabled)
      {
        this.clock.Start();
      }
      this.mouseNow = e.GetPosition(this);
```

```
    }

    #endregion

    #region Update Screen

    private void Clock_Tick(object sender, EventArgs e)
    {

      this.RotateY -= ((this.mouseNow.X - this.mouseStart.X) ) ;
      this.mouseStart.X += ((this.mouseNow.X - this.mouseStart.X));
      this.mouseStart.Y += ((this.mouseNow.Y - this.mouseStart.Y));
      if ((Math.Abs(this.mouseNow.X - this.mouseStart.X) < 2) &&
(Math.Abs(this.mouseNow.Y - this.mouseStart.Y) < 2))
      {
        this.clock.Stop();
      }
    }

    private void Refresh()
    {
      double angle = this.RotateY;
      const double depth = 26;

      this.UpdateFace(this.front_el, this.Front, depth, angle, 0, angle, 0, 0, 0, 0, 0.5,
0.5, 0.5, 0);
      this.UpdateFace(this.reflex_el, this.Reflex, depth + 0.5, angle, 0, angle, 0, 0, 0,
0, 0.5, 0.5, 0.5, 0);

      angle -= 180;
      this.UpdateFace(this.back_el, this.Back, depth, angle, 0, angle, 0, 0, 0, 0, 0.5,
0.5, 0.5, 0);

      angle -= 90;
      this.UpdateFace(this.left_el, this.Left, 149, angle, 0, angle, 0, 0, 0, 0, 0.5, 0.5,
0.5, 0);
      this.UpdateFace(this.leftb1_el, this.Leftb1, 149, angle, 0, angle, 0, 0, 318, 0, 0.5,
0.5, 0.5, 0);
      this.UpdateFace(this.leftb2_el, this.Leftb1, 149, angle, 0, angle, 0, 0, 320, 0, 0.5,
0.5, 0.5, 0);
      this.UpdateFace(this.leftb3_el, this.Leftb1, 148, angle, 0, angle, 0, 0, 322, 0, 0.5,
0.5, 0.5, 0);
      this.UpdateFace(this.leftb4_el, this.Leftb1, 146, angle, 0, angle, 0, 0, 324, 0, 0.5,
0.5, 0.5, 0);
      this.UpdateFace(this.leftb5_el, this.Leftb1, 144, angle, 0, angle, 0, 0, 326, 0, 0.5,
0.5, 0.5, 0);
```

```
        this.UpdateFace(this.leftt1_el, this.Leftt1, 149, angle, 0, angle, 0, 0, -318, 0, 0.5,
0.5, 0.5, 0);
        this.UpdateFace(this.leftt2_el, this.Leftt1, 149, angle, 0, angle, 0, 0, -320, 0, 0.5,
0.5, 0.5, 0);
        this.UpdateFace(this.leftt3_el, this.Leftt1, 148, angle, 0, angle, 0, 0, -322, 0, 0.5,
0.5, 0.5, 0);
        this.UpdateFace(this.leftt4_el, this.Leftt1, 146, angle, 0, angle, 0, 0, -324, 0, 0.5,
0.5, 0.5, 0);
        this.UpdateFace(this.leftt5_el, this.Leftt1, 144, angle, 0, angle, 0, 0, -326, 0, 0.5,
0.5, 0.5, 0);

        angle += 180;
        this.UpdateFace(this.right_el, this.Right, 149, angle, 0, angle, 0, 0, 0, 0, 0.5, 0.5,
0.5, 0);
        this.UpdateFace(this.rightb1_el, this.Rightb1, 149, angle, 0, angle, 0, 0, 318, 0,
0.5, 0.5, 0.5, 0);
        this.UpdateFace(this.rightb2_el, this.Rightb1, 149, angle, 0, angle, 0, 0, 320, 0,
0.5, 0.5, 0.5, 0);
        this.UpdateFace(this.rightb3_el, this.Rightb1, 148, angle, 0, angle, 0, 0, 322, 0,
0.5, 0.5, 0.5, 0);
        this.UpdateFace(this.rightb4_el, this.Rightb1, 146, angle, 0, angle, 0, 0, 324, 0,
0.5, 0.5, 0.5, 0);
        this.UpdateFace(this.rightb5_el, this.Rightb1, 144, angle, 0, angle, 0, 0, 326, 0,
0.5, 0.5, 0.5, 0);
        this.UpdateFace(this.rightt1_el, this.Rightt1, 149, angle, 0, angle, 0, 0, -318, 0,
0.5, 0.5, 0.5, 0);
        this.UpdateFace(this.rightt2_el, this.Rightt1, 149, angle, 0, angle, 0, 0, -320, 0,
0.5, 0.5, 0.5, 0);
        this.UpdateFace(this.rightt3_el, this.Rightt1, 148, angle, 0, angle, 0, 0, -322, 0,
0.5, 0.5, 0.5, 0);
        this.UpdateFace(this.rightt4_el, this.Rightt1, 146, angle, 0, angle, 0, 0, -324, 0,
0.5, 0.5, 0.5, 0);
        this.UpdateFace(this.rightt5_el, this.Rightt1, 144, angle, 0, angle, 0, 0, -326, 0,
0.5, 0.5, 0.5, 0);
    }

    private void UpdateFace(FrameworkElement face, ImageSource image, double
radius, double angle, double rotateX, double rotateY, double rotateZ, double localX,
double localY, double localZ, double centerX, double centerY, double centerZ, int
zIndex)

    {
        if (face != null)
        {
            double x = radius * Math.Cos( Math.PI * (angle+90 )/180);
            double z = radius * Math.Sin( Math.PI * (angle+90 )/180);
```

```
            PlaneProjection ppFace = new PlaneProjection();
            if(face.Name=="front")
            // MessageBox.Show("Face: " + face.Name + "angle: " + angle+"mousenow
x: "+this.mouseNow.X+"mousestart x: "+mouseStart.X);

            ppFace.RotationX =0;
            ppFace.RotationY = rotateY;
            ppFace.RotationZ = 0;
            ppFace.GlobalOffsetX = x;
            ppFace.GlobalOffsetY = 0;
            ppFace.GlobalOffsetZ =  z + this.Distance;
            ppFace.LocalOffsetY = localY;
            face.Projection = ppFace;


            double alignX = face.DesiredSize.Width / 2;
            double alignY = face.DesiredSize.Height / 2;
            if (image != null)
            {
               Image f = face as Image;
               if (f.Source != image)
               {
                  f.Source = image;
               }
            }

            face.Arrange(new Rect(Width / 2 - alignX, Height / 2 - alignY,
face.DesiredSize.Width, face.DesiredSize.Height));
         }
      }

      #endregion



      #region Panel Overrides

      protected override Size MeasureOverride(Size availableSize)
      {

         Size resultSize = new Size(0, 0);
         foreach (UIElement child in this.Children)
         {
            child.Measure(availableSize);
            resultSize.Width = Math.Max(resultSize.Width, child.DesiredSize.Width);
            resultSize.Height = Math.Max(resultSize.Height, child.DesiredSize.Height);
```

```
}

resultSize.Width =
   double.IsPositiveInfinity(availableSize.Width) ?
   resultSize.Width : availableSize.Width;

resultSize.Height =
   double.IsPositiveInfinity(availableSize.Height) ?
   resultSize.Height : availableSize.Height;

this.front_el = (FrameworkElement)this.FindName("front");
this.reflex_el = (FrameworkElement)this.FindName("reflex");
this.colorArt_el = (FrameworkElement)this.FindName("colorArt");
this.bwArt_el = (FrameworkElement)this.FindName("bwArt");
this.back_el = (FrameworkElement)this.FindName("back");
this.engrave1_el = (FrameworkElement)this.FindName("engrave1");
this.engrave2_el = (FrameworkElement)this.FindName("engrave2");
this.engrave3_el = (FrameworkElement)this.FindName("engrave3");

this.artists_el = (FrameworkElement)this.FindName("Artists");
this.controlSticker_el = (FrameworkElement)this.FindName("controlSticker");
this.arrows_el = (FrameworkElement)this.FindName("arrows");

this.left_el = (FrameworkElement)this.FindName("left");
this.leftb1_el = (FrameworkElement)this.FindName("leftb1");
this.leftb2_el = (FrameworkElement)this.FindName("leftb2");
this.leftb3_el = (FrameworkElement)this.FindName("leftb3");
this.leftb4_el = (FrameworkElement)this.FindName("leftb4");
this.leftb5_el = (FrameworkElement)this.FindName("leftb5");
this.leftt1_el = (FrameworkElement)this.FindName("leftt1");
this.leftt2_el = (FrameworkElement)this.FindName("leftt2");
this.leftt3_el = (FrameworkElement)this.FindName("leftt3");
this.leftt4_el = (FrameworkElement)this.FindName("leftt4");
this.leftt5_el = (FrameworkElement)this.FindName("leftt5");

this.right_el = (FrameworkElement)this.FindName("right");
this.rightb1_el = (FrameworkElement)this.FindName("rightb1");
this.rightb2_el = (FrameworkElement)this.FindName("rightb2");
this.rightb3_el = (FrameworkElement)this.FindName("rightb3");
this.rightb4_el = (FrameworkElement)this.FindName("rightb4");
this.rightb5_el = (FrameworkElement)this.FindName("rightb5");
this.rightt1_el = (FrameworkElement)this.FindName("rightt1");
this.rightt2_el = (FrameworkElement)this.FindName("rightt2");
this.rightt3_el = (FrameworkElement)this.FindName("rightt3");
this.rightt4_el = (FrameworkElement)this.FindName("rightt4");
this.rightt5_el = (FrameworkElement)this.FindName("rightt5");
```

```
            return resultSize;
        }

        protected override Size ArrangeOverride(Size finalSize)
        {
            this.Refresh();
            return base.ArrangeOverride(finalSize);
        }

        #endregion
    }
}
```

Dynamic3DModel.cs

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Windows.Threading;
using System.ComponentModel;
using System.Windows.Media.Imaging;

namespace _Dto3DImages
{
    public class RoundBox3D : Panel
    {
        public double frontDepth { get; set; }
        public double sideDepth { get; set; }

        private Point mouseStart, mouseNow;


        private FrameworkElement front_el;
        private FrameworkElement left_el;
        private FrameworkElement right_el,back_el,top_el,bottom_el;

        private DispatcherTimer clock;
```

```
    #region Dependency Properties

    // Dependency properties exposed in Blend
    // (Properties Panel > RoundBox 3D)
    public static readonly DependencyProperty RotateYProperty =
DependencyProperty.Register("RotateY", typeof(double), typeof(RoundBox3D), new
PropertyMetadata(RoundBox3D.RotateYChanged));
    public static readonly DependencyProperty RotateXProperty =
DependencyProperty.Register("RotateX", typeof(double), typeof(RoundBox3D), new
PropertyMetadata(RoundBox3D.RotateXChanged));
    public static readonly DependencyProperty RotateZProperty =
DependencyProperty.Register("RotateZ", typeof(double), typeof(RoundBox3D), new
PropertyMetadata(RoundBox3D.RotateZChanged));

    public static readonly DependencyProperty IsDragOnProperty =
DependencyProperty.Register("IsDragOn", typeof(bool), typeof(RoundBox3D), new
PropertyMetadata(RoundBox3D.IsDragChanged));
    public static readonly DependencyProperty DistanceProperty =
DependencyProperty.Register("Distance", typeof(double), typeof(RoundBox3D), new
PropertyMetadata(RoundBox3D.DistanceChanged));

    private static void RotateYChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e)
    {
      ((RoundBox3D)sender).Refresh();
    }
    private static void RotateXChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e)
    {
      ((RoundBox3D)sender).Refresh();
    }
    private static void RotateZChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e)
    {
      ((RoundBox3D)sender).Refresh();
    }
    private static void IsDragChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e)
    {
      ((RoundBox3D)sender).OnIsDragChanged(e);
    }
    private static void DistanceChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e)
    {
      ((RoundBox3D)sender).Refresh();
    }
```

```
        // Dependency properties exposed in Blend
        // (Properties Panel > RoundBox 3D Faces)
        public static readonly DependencyProperty FrontProperty =
DependencyProperty.Register("Front", typeof(ImageSource), typeof(RoundBox3D),
new PropertyMetadata(RoundBox3D.FrontChanged));
        public static readonly DependencyProperty BackProperty =
DependencyProperty.Register("Back", typeof(ImageSource), typeof(RoundBox3D),
new PropertyMetadata(RoundBox3D.BackChanged));
        public static readonly DependencyProperty LeftProperty =
DependencyProperty.Register("Left", typeof(ImageSource), typeof(RoundBox3D),
new PropertyMetadata(RoundBox3D.LeftChanged));
        public static readonly DependencyProperty RightProperty =
DependencyProperty.Register("Right", typeof(ImageSource), typeof(RoundBox3D),
new PropertyMetadata(RoundBox3D.RightChanged));
        public static readonly DependencyProperty TopProperty =
DependencyProperty.Register("Top", typeof(ImageSource), typeof(RoundBox3D), new
PropertyMetadata(RoundBox3D.TopChanged));
        public static readonly DependencyProperty BottomProperty =
DependencyProperty.Register("Bottom", typeof(ImageSource), typeof(RoundBox3D),
new PropertyMetadata(RoundBox3D.BottomChanged));

        private static void FrontChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e)
        {
            ((RoundBox3D)sender).Refresh();
        }
        private static void BackChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e)
        {
            ((RoundBox3D)sender).Refresh();
        }
        private static void LeftChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e)
        {
            ((RoundBox3D)sender).Refresh();
        }
        private static void RightChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e)
        {
            ((RoundBox3D)sender).Refresh();
        }
        private static void TopChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e)
        {
            ((RoundBox3D)sender).Refresh();
```

```
    }
    private static void BottomChanged(DependencyObject sender,
DependencyPropertyChangedEventArgs e)
    {
      ((RoundBox3D)sender).Refresh();
    }

    // Getters and setters of dependency properties
    [Category("RoundBox 3D")]
    public double RotateY
    {
      get { return (double)this.GetValue(RoundBox3D.RotateYProperty); }
      set { this.SetValue(RoundBox3D.RotateYProperty, value); }
    }

    [Category("RoundBox 3D")]
    public double RotateX
    {
      get { return (double)this.GetValue(RoundBox3D.RotateXProperty); }
      set { this.SetValue(RoundBox3D.RotateXProperty, value); }
    }

    [Category("RoundBox 3D")]
    public double RotateZ
    {
      get { return (double)this.GetValue(RoundBox3D.RotateZProperty); }
      set { this.SetValue(RoundBox3D.RotateZProperty, value); }
    }

    [Category("RoundBox 3D")]
    public bool IsDragOn
    {
      get { return (bool)this.GetValue(RoundBox3D.IsDragOnProperty); }
      set { this.SetValue(RoundBox3D.IsDragOnProperty, value); }
    }

    [Category("RoundBox 3D")]
    public double Distance
    {
      get { return (double)this.GetValue(RoundBox3D.DistanceProperty); }
      set { this.SetValue(RoundBox3D.DistanceProperty, value); }
    }

    [Category("RoundBox 3D Faces")]
    public ImageSource Front
    {
```

```
        get { return (ImageSource)this.GetValue(RoundBox3D.FrontProperty); }
        set { this.SetValue(RoundBox3D.FrontProperty, value); }
    }

    [Category("RoundBox 3D Faces")]
    public ImageSource Back
    {
        get { return (ImageSource)this.GetValue(RoundBox3D.BackProperty); }
        set { this.SetValue(RoundBox3D.BackProperty, value); }
    }

    [Category("RoundBox 3D Faces")]
    public ImageSource Left
    {
        get { return (ImageSource)this.GetValue(RoundBox3D.LeftProperty); }
        set { this.SetValue(RoundBox3D.LeftProperty, value); }
    }
    [Category("RoundBox 3D Faces")]
    public ImageSource Right
    {
        get { return (ImageSource)this.GetValue(RoundBox3D.RightProperty); }
        set { this.SetValue(RoundBox3D.RightProperty, value); }
    }
    [Category("RoundBox 3D Faces")]
    public ImageSource Top
    {
        get { return (ImageSource)this.GetValue(RoundBox3D.TopProperty); }
        set { this.SetValue(RoundBox3D.TopProperty, value); }
    }
    [Category("RoundBox 3D Faces")]
    public ImageSource Bottom
    {
        get { return (ImageSource)this.GetValue(RoundBox3D.BottomProperty); }
        set { this.SetValue(RoundBox3D.BottomProperty, value); }
    }


    private void OnIsDragChanged(DependencyPropertyChangedEventArgs e)
    {
        this.MouseLeftButtonDown += new
MouseButtonEventHandler(this.RoundBox3D_MouseLeftButtonDown);
        this.MouseLeftButtonUp += new
MouseButtonEventHandler(this.RoundBox3D_MouseLeftButtonUp);
    }

    #endregion
```

```
    #region Mouse

    private void RoundBox3D_MouseLeftButtonDown(object sender,
MouseButtonEventArgs e)
    {
      this.CaptureMouse();
      this.mouseStart = e.GetPosition(this);
      this.mouseNow = this.mouseStart;

      if (this.clock == null)
      {
        this.clock = new DispatcherTimer();
        this.clock.Tick += new EventHandler(this.Clock_Tick);
        this.clock.Start();
      }
      else
      {
        this.clock.Stop();
        this.clock.Start();
      }

      this.MouseMove += new
MouseEventHandler(this.RoundBox3D_MouseMove);
    }

    private void RoundBox3D_MouseLeftButtonUp(object sender,
MouseButtonEventArgs e)
    {
      this.ReleaseMouseCapture();
      this.MouseMove -= new MouseEventHandler(this.RoundBox3D_MouseMove);
    }

    private void RoundBox3D_MouseMove(object sender, MouseEventArgs e)
    {
      if (!this.clock.IsEnabled)
      {
        this.clock.Start();
      }
      this.mouseNow = e.GetPosition(this);
    }

    #endregion

    #region Update Screen
```

```
private void Clock_Tick(object sender, EventArgs e)
{

   this.RotateY -= ((this.mouseNow.X - this.mouseStart.X));
   this.mouseStart.X += ((this.mouseNow.X - this.mouseStart.X));
   this.mouseStart.Y += ((this.mouseNow.Y - this.mouseStart.Y));

}

private void Refresh()
{

   double angleY = this.RotateY;
   double angleX = this.RotateX;
   double angleZ = this.RotateZ;
   double fdepth = 0;
   double sdepth = 0;
   if(this.Front!=null)
   fdepth = frontDepth;
   if(this.Left!=null)
   sdepth = sideDepth;

    this.UpdateFace(this.front_el, this.Front, fdepth,  angleX,angleY, angleZ);

   angleY -= 180;
   angleX = -angleX;
   this.UpdateFace(this.back_el, this.Back, fdepth,  angleX, angleY, angleZ);


   angleY -= 90;
   this.UpdateFace(this.left_el, this.Left, sdepth, angleX, angleY, angleZ);

   angleY += 180;
   this.UpdateFace(this.right_el, this.Right, sdepth, angleX, angleY, angleZ);

   //angle += 90;
   //  this.UpdateFace(this.top_el, this.Top, sdepth, 0, angle, 0);

   //  angle += 180;
   //  this.UpdateFace(this.bottom_el, this.Bottom, sdepth, 0, angle, 0);


}

   private void UpdateFace(FrameworkElement face, ImageSource image, double
radius,  double rotateX, double rotateY, double rotateZ)
```

```
{
   if (face != null)
   {
      double x, y, z;

      x = radius * Math.Cos(Math.PI * (rotateY + 90) / 180);
      y = radius * Math.Cos(Math.PI * (rotateX+90) / 180);

      z = radius * Math.Sin(Math.PI * (rotateY + 90) / 180);

      y = 0;
      TextBox t;
      if (face == this.front_el)
      {
         t = (TextBox)this.FindName("roty");
         t.Text = "" + (rotateY % 181);
         t = (TextBox)this.FindName("fx");
         t.Text = "x: " + x;
         t = (TextBox)this.FindName("fz");
         t.Text = "z: " + z;

      }
      if (face == this.back_el)
      {

         t = (TextBox)this.FindName("bx");
         t.Text = "x: " + x;
         t = (TextBox)(TextBox)this.FindName("bz");
         t.Text = "z: " + z;

      }
      if (face == this.right_el)
      {

         t = (TextBox)this.FindName("rx");
         t.Text = "x: " + x;
         t = (TextBox)this.FindName("rz");
         t.Text = "z: " + z;

      }
      if (face == this.left_el)
      {

         t = (TextBox)this.FindName("lx");
         t.Text = "x: " + x;
         t = (TextBox)this.FindName("lz");
```

```
            t.Text = "z: " + z;

        }
        PlaneProjection ppFace = new PlaneProjection();
        ppFace.RotationX = rotateX;
        ppFace.RotationY = rotateY;
        ppFace.RotationZ = 0;
        ppFace.GlobalOffsetX = x;
        ppFace.GlobalOffsetY = y;
        ppFace.GlobalOffsetZ = z+this.Distance;
        ppFace.LocalOffsetX = 0;
        ppFace.LocalOffsetY = 0;
        ppFace.LocalOffsetZ = 0;
        ppFace.CenterOfRotationX = 0.5;
        ppFace.CenterOfRotationY = 0.5;
        ppFace.CenterOfRotationZ = 0;

        face.Projection = ppFace;

        double alignX = face.DesiredSize.Width / 2;
        double alignY = face.DesiredSize.Height / 2;

        face.Arrange(new Rect(Width/2 - alignX, Height / 2 - alignY,
face.DesiredSize.Width, face.DesiredSize.Height));
      }
    }

    #endregion

    #region Panel Overrides

    protected override Size MeasureOverride(Size availableSize)
    {
      Size resultSize = new Size(0, 0);

      foreach (UIElement child in this.Children)
      {
        child.Measure(availableSize);
        resultSize.Width = Math.Max(resultSize.Width, child.DesiredSize.Width);
        resultSize.Height = Math.Max(resultSize.Height, child.DesiredSize.Height);
      }

      resultSize.Width =
        double.IsPositiveInfinity(availableSize.Width) ?
        resultSize.Width : availableSize.Width;
```

```
            resultSize.Height =
               double.IsPositiveInfinity(availableSize.Height) ?
               resultSize.Height : availableSize.Height;

            this.front_el = (FrameworkElement)this.FindName("front");
            this.back_el = (FrameworkElement)this.FindName("back");
            this.left_el = (FrameworkElement)this.FindName("left");
            this.right_el = (FrameworkElement)this.FindName("right");
          //   this.top_el = (FrameworkElement)this.FindName("top");
            // this.bottom_el = (FrameworkElement)this.FindName("bottom");
            return resultSize;
        }

        protected override Size ArrangeOverride(Size finalSize)
        {
            this.Refresh();
            return base.ArrangeOverride(finalSize);
        }

        #endregion
    }
}
```

Dynamic3DMain.cs

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Windows.Media.Imaging;
using System.IO;
using System.IO.IsolatedStorage;

namespace _Dto3DImages
{
    public partial class MainPage : UserControl
    {

        BitmapImage bImage = new BitmapImage();
        BitmapImage fImage = new BitmapImage();
        BitmapImage rImage = new BitmapImage();
```

```
BitmapImage lImage = new BitmapImage();

public MainPage()
{
   // Required to initialize variables
   InitializeComponent();
}

private void UserControl_Loaded(object sender, RoutedEventArgs e)
{


}

private void button1_Click_1(object sender, RoutedEventArgs e)
{
   OpenFileDialog openFileDialog = new OpenFileDialog();

   openFileDialog.Filter = "All Files (*.*)|*.*";

   bool? userClickedOK = openFileDialog.ShowDialog();

   if (userClickedOK == true)
   {
      System.IO.Stream fileStream = openFileDialog.File.OpenRead();
      fImage.SetSource(fileStream);
      this.frontimg.Source = fImage;
      fileStream.Close();
   }


}

private void button2_Click(object sender, RoutedEventArgs e)
{
   OpenFileDialog openFileDialog2 = new OpenFileDialog();

   openFileDialog2.Filter = "All Files (*.*)|*.*";
   bool? userClickedOK2 = openFileDialog2.ShowDialog();

   if (userClickedOK2 == true)
   {

      System.IO.Stream fileStream = openFileDialog2.File.OpenRead();
      bImage.SetSource(fileStream);
      this.backimg.Source = bImage;
```

```
        fileStream.Close();
    }

}

private void button3_Click(object sender, RoutedEventArgs e)
{
    OpenFileDialog openFileDialog3 = new OpenFileDialog();
    openFileDialog3.Filter = "All Files (*.*)|*.*";

    bool? userClickedOK3 = openFileDialog3.ShowDialog();

    if (userClickedOK3 == true)
    {

        System.IO.Stream fileStream = openFileDialog3.File.OpenRead();
        rImage.SetSource(fileStream);

        this.rightimg.Source = rImage;

        fileStream.Close();
    }

}

private void button4_Click(object sender, RoutedEventArgs e)
{
    BitmapImage image = new BitmapImage();
    OpenFileDialog openFileDialog1 = new OpenFileDialog();

    openFileDialog1.Filter = "All Files (*.*)|*.*";

    bool? userClickedOK = openFileDialog1.ShowDialog();

    if (userClickedOK == true)
    {

        System.IO.Stream fileStream = openFileDialog1.File.OpenRead();
        Image.SetSource(fileStream);
        this.leftimg.Source = lImage;

        fileStream.Close();
    }
}

private void button5_Click(object sender, RoutedEventArgs e)
```

```
{
    R1.frontDepth = lImage.PixelWidth / 2;
    R1.sideDepth = fImage.PixelWidth / 2;

    R1.Front = fImage;
    R1.Back = bImage;
    R1.Left = lImage;
    R1.Right = rImage;
    R1.Distance = -500;

    //back
    Image Backimg = new Image();
    Backimg.Source = bImage;
    Backimg.Name = "back";

    Backimg.HorizontalAlignment = System.Windows.HorizontalAlignment.Left;
    Backimg.VerticalAlignment = System.Windows.VerticalAlignment.Center;
    Backimg.Width = bImage.PixelWidth;
    Backimg.Height = bImage.PixelHeight;
    Backimg.IsHitTestVisible = false;

    //front
    Image frntimg = new Image();
    frntimg.Source = fImage;
    frntimg.Name = "front";

    frntimg.HorizontalAlignment = System.Windows.HorizontalAlignment.Left;
    frntimg.VerticalAlignment = System.Windows.VerticalAlignment.Center;
    frntimg.Width = fImage.PixelWidth;
    frntimg.Height = fImage.PixelHeight;
    frntimg.IsHitTestVisible = false;


    //left
    Image leftimg = new Image();
    leftimg.Source = lImage;
    leftimg.Name = "left";

    leftimg.HorizontalAlignment = System.Windows.HorizontalAlignment.Left;
    leftimg.VerticalAlignment = System.Windows.VerticalAlignment.Top;
    leftimg.Width = lImage.PixelWidth;
    leftimg.Height = lImage.PixelHeight;
    leftimg.IsHitTestVisible = false;

    Image topimg = new Image();
    topimg.Source = lImage;
```

```
        topimg.Name = "top";

        topimg.HorizontalAlignment = System.Windows.HorizontalAlignment.Left;
        topimg.VerticalAlignment = System.Windows.VerticalAlignment.Top;
        topimg.Width = lImage.PixelWidth;
        topimg.Height = lImage.PixelHeight;
        topimg.IsHitTestVisible = false;

        Image bottomimg = new Image();
        bottomimg.Source = lImage;
        bottomimg.Name = "bottom";

        bottomimg.HorizontalAlignment =
System.Windows.HorizontalAlignment.Left;
        bottomimg.VerticalAlignment = System.Windows.VerticalAlignment.Top;
        bottomimg.Width = lImage.PixelWidth;
        bottomimg.Height = lImage.PixelHeight;
        bottomimg.IsHitTestVisible = false;


        //right
        Image rightimg = new Image();
        rightimg.Source = rImage;
        rightimg.Name = "right";

        rightimg.HorizontalAlignment = System.Windows.HorizontalAlignment.Left;
        rightimg.VerticalAlignment = System.Windows.VerticalAlignment.Top;
        rightimg.Width = rImage.PixelWidth;
        rightimg.Height = rImage.PixelHeight;
        rightimg.IsHitTestVisible = false;


        //childrens
        R1.Children.Add(frntimg);
        R1.Children.Add(Backimg);
        R1.Children.Add(leftimg);
        R1.Children.Add(rightimg);
        R1.Children.Add(topimg);
        R1.Children.Add(bottomimg);
      }

      private void sldrzoom_ValueChanged(object sender,
  RoutedPropertyChangedEventArgs<double> e)
      {
        R1.Distance = sldrzoom.Value;
```

```
    }
  }
}
```

ImageCropper.cs

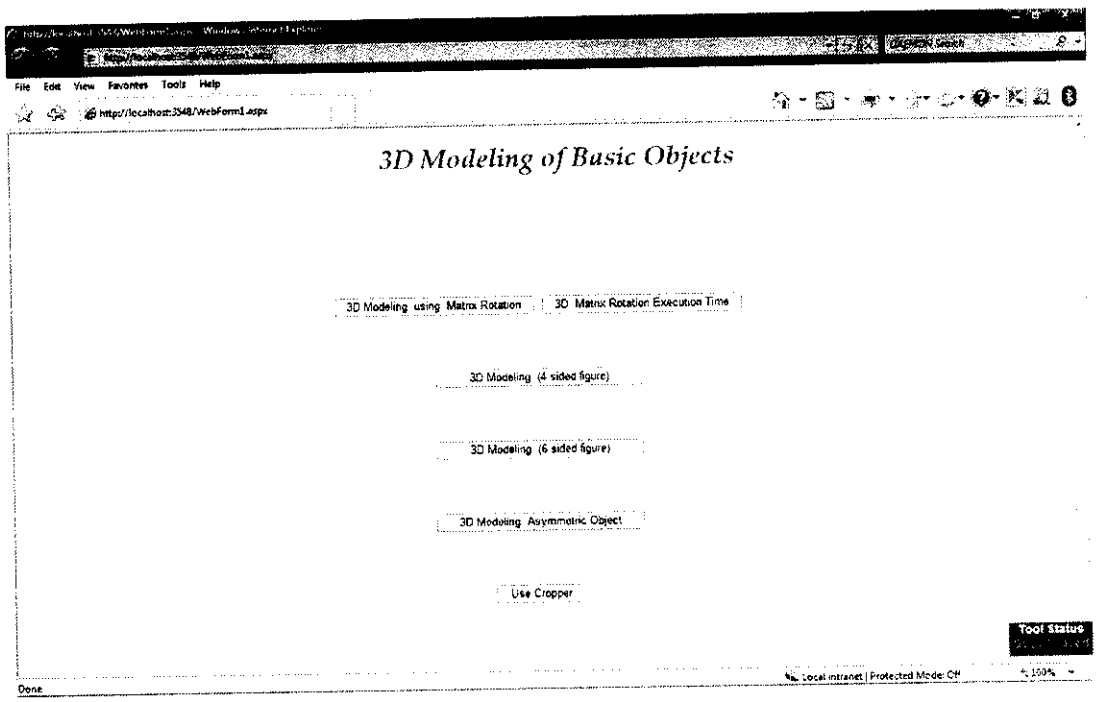
```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace ImageCropper
{
  public partial class SelectionCanvas : Canvas
  {
    #region Instance fields
    private Point mouseLeftDownPoint;
    private Style cropperStyle;
    public Shape rubberBand = null;
    public readonly RoutedEvent CropImageEvent;
    #endregion

    #region Events
    public event RoutedEventHandler CropImage
    {
      add { AddHandler(this.CropImageEvent, value); }
      remove { RemoveHandler(this.CropImageEvent, value); }
    }
    #endregion

    #region Ctor


    public SelectionCanvas()
    {
```

```
        this.CropImageEvent = EventManager.RegisterRoutedEvent("CropImage",
          RoutingStrategy.Bubble, typeof(RoutedEventHandler),
typeof(SelectionCanvas));
    }
    #endregion

    #region Public Properties
    public Style  CropperStyle
    {
      get { return cropperStyle; }
      set { cropperStyle = value; }
    }
    #endregion

    #region Overrides

    protected override void OnMouseLeftButtonDown(MouseButtonEventArgs e)
    {
      base.OnMouseLeftButtonDown(e);
      if (!this.IsMouseCaptured)
      {
        mouseLeftDownPoint = e.GetPosition(this);
        this.CaptureMouse();
      }
    }

    protected override void OnMouseLeftButtonUp(MouseButtonEventArgs e)
    {
      base.OnMouseLeftButtonUp(e);

      if (this.IsMouseCaptured && rubberBand != null)
      {
        this.ReleaseMouseCapture();

        RaiseEvent(new RoutedEventArgs(this.CropImageEvent, this));
      }
    }

    protected override void OnMouseMove(MouseEventArgs e)
    {
      base.OnMouseMove(e);

      if (this.IsMouseCaptured)
      {
        Point currentPoint = e.GetPosition(this);
```

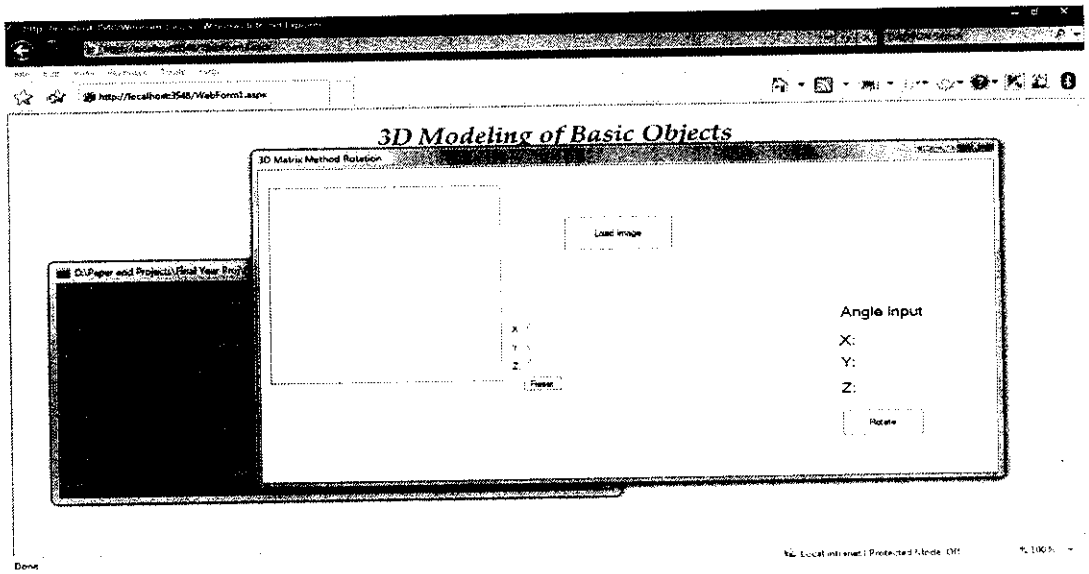
```
            if (rubberBand == null)
            {
               rubberBand = new Rectangle();
               if (cropperStyle != null)
                  rubberBand.Style = cropperStyle;
               this.Children.Add(rubberBand);
            }

            double width = Math.Abs(mouseLeftDownPoint.X - currentPoint.X);
            double height = Math.Abs(mouseLeftDownPoint.Y - currentPoint.Y);
            double left = Math.Min(mouseLeftDownPoint.X, currentPoint.X);
            double top = Math.Min(mouseLeftDownPoint.Y, currentPoint.Y);

            rubberBand.Width = width;
            rubberBand.Height = height;
            Canvas.SetLeft(rubberBand, left);
            Canvas.SetTop(rubberBand, top);
         }
      }
      #endregion
   }
}
```

SCREEN SHOTS

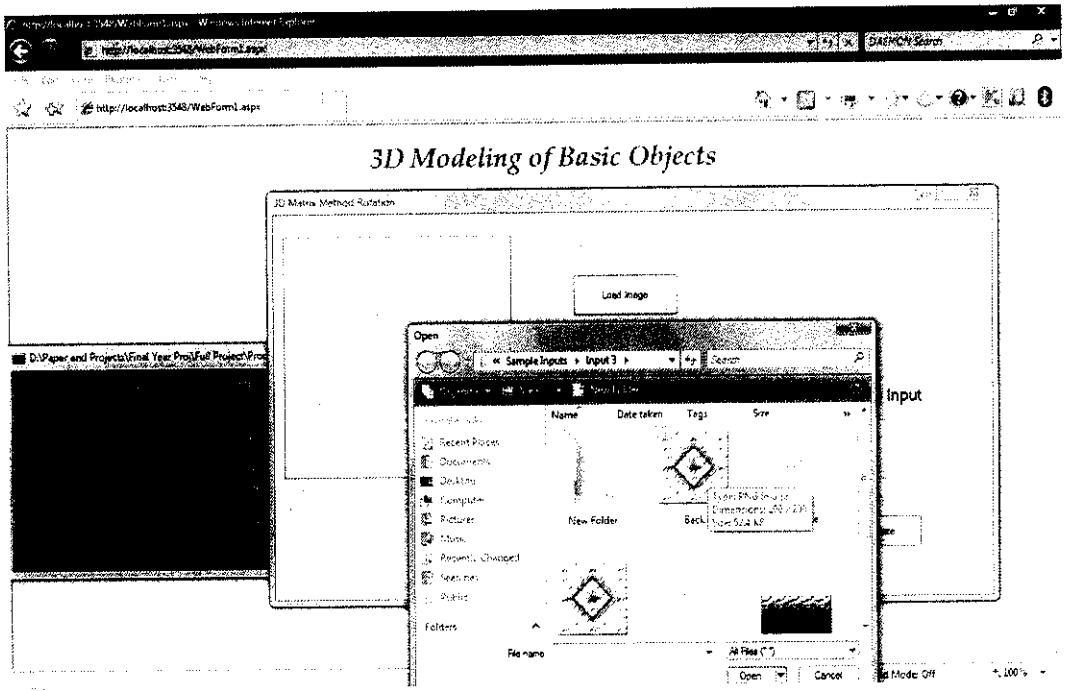
Main page



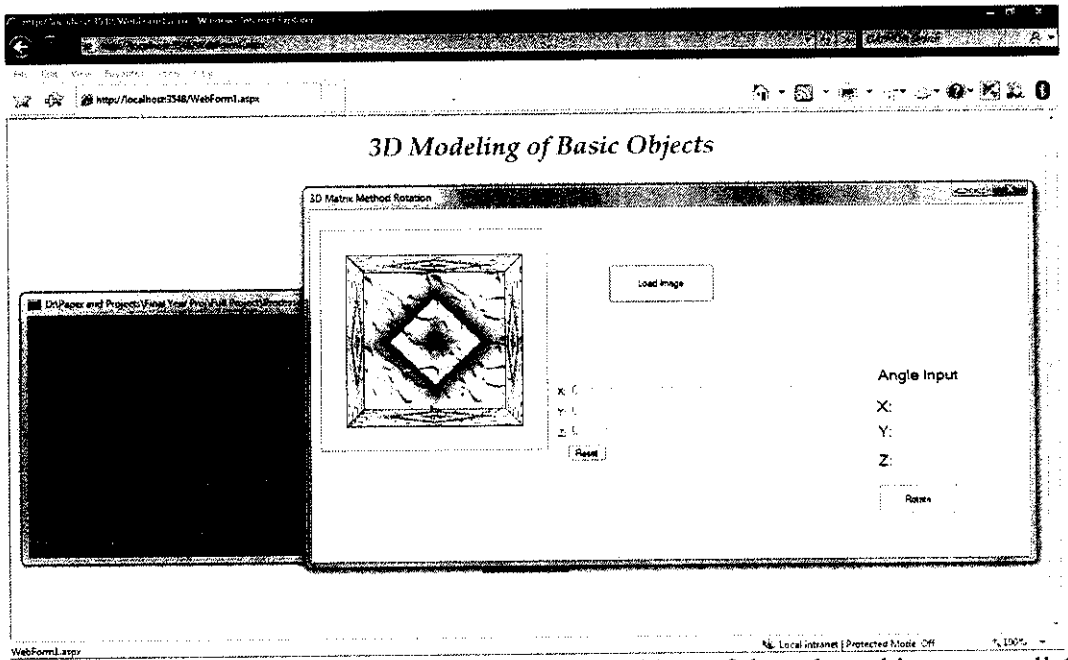
Rotation by Euler Angles



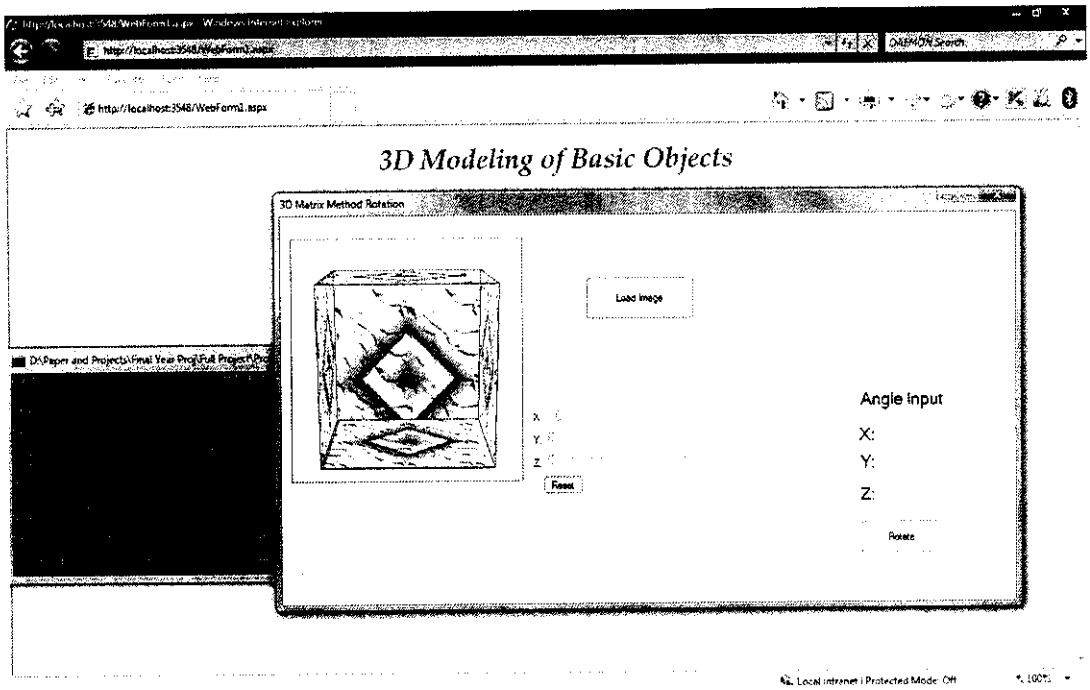
Euler Rotation: The figure shows the Euler rotation front page with no images loaded as input. The console window shows the processing time and status of the application.



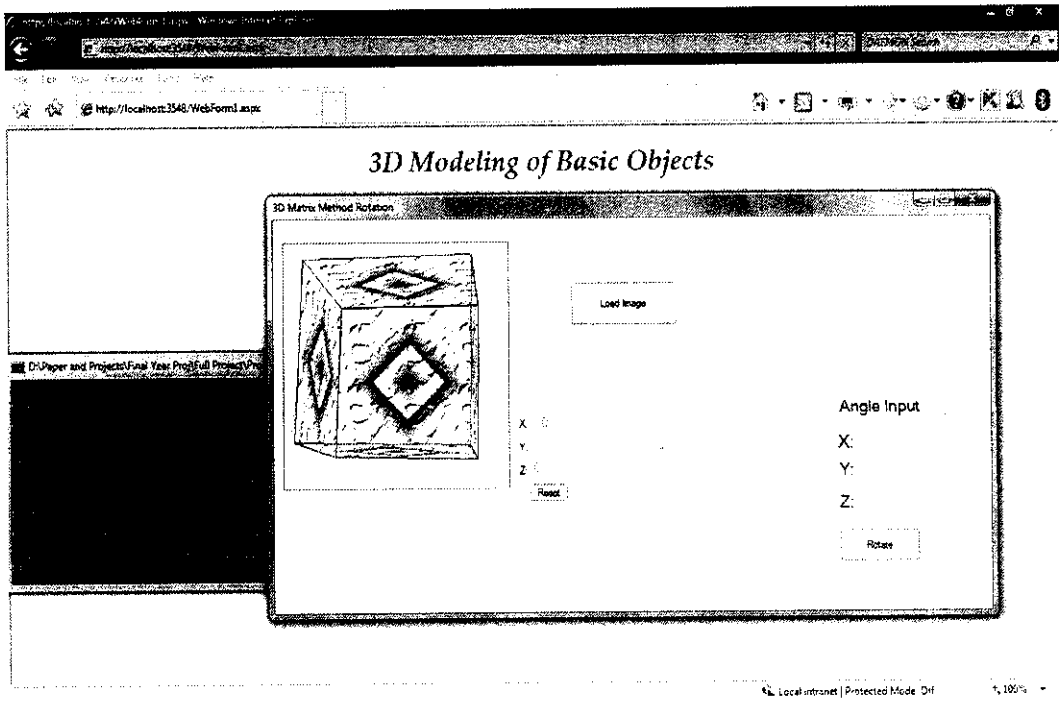
Euler Rotation: The figure shows the how the user can select input images to generate 3D object.



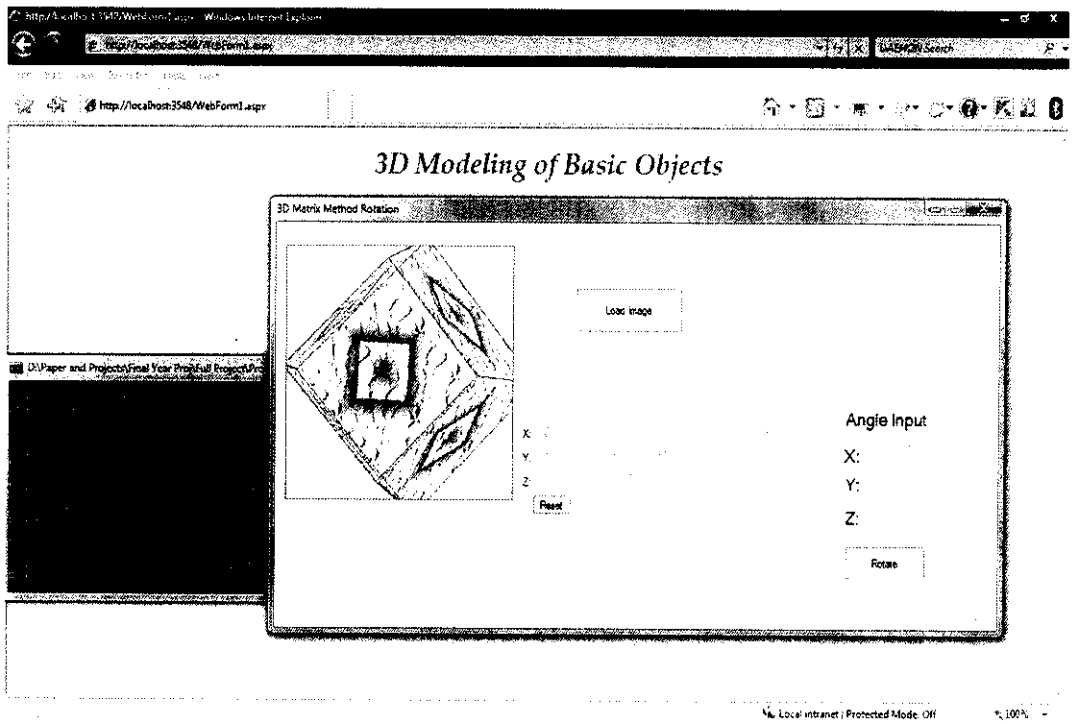
Euler Rotation: figure shows the generated 3D object of the selected image on all 6 sides. The console window shows the status of the process and the processing time as 8 seconds.



Euler Rotation: The figure shows the rotation of 3D object along X-axis. The console displays the time taken as 26.317 seconds.

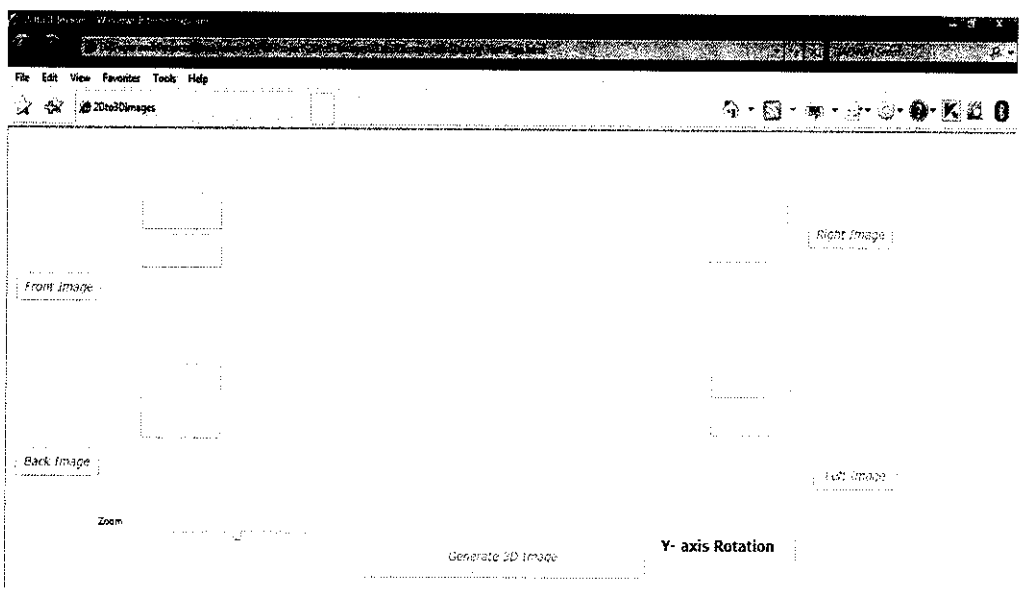


Euler Rotation: The figure shows the rotation of 3D object along X and Y-axes. The console window displays the time taken as 40.477 seconds.

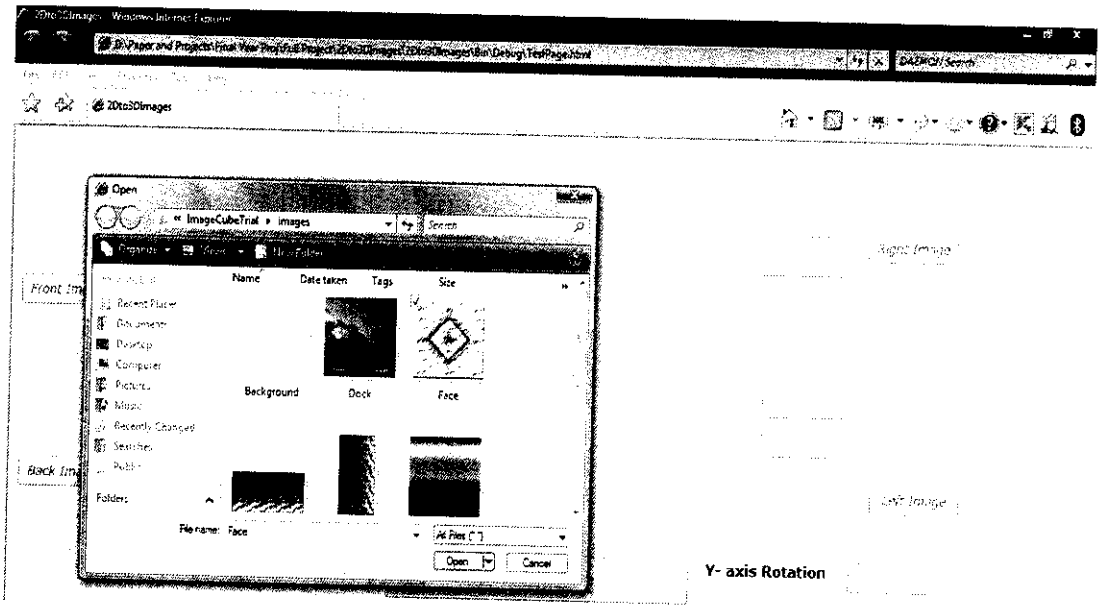


Euler Rotation: The figure shows the rotation of 3D object along X ,Y and Z-axes. The console displays the time taken as 55.271 seconds.

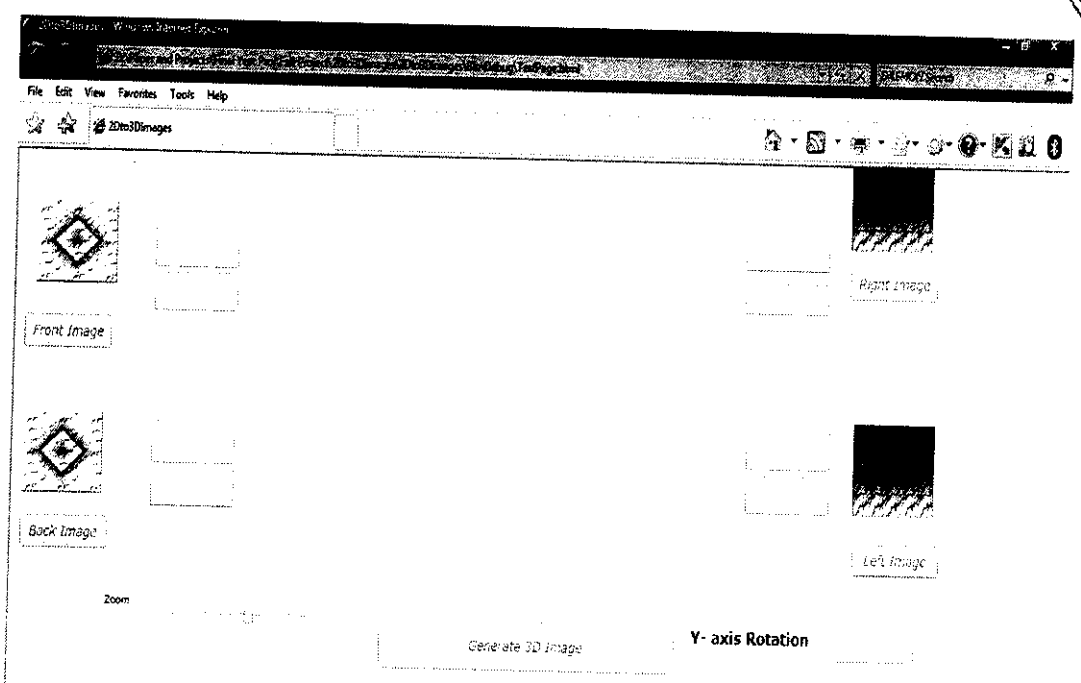
Rotation by PlaneProjection



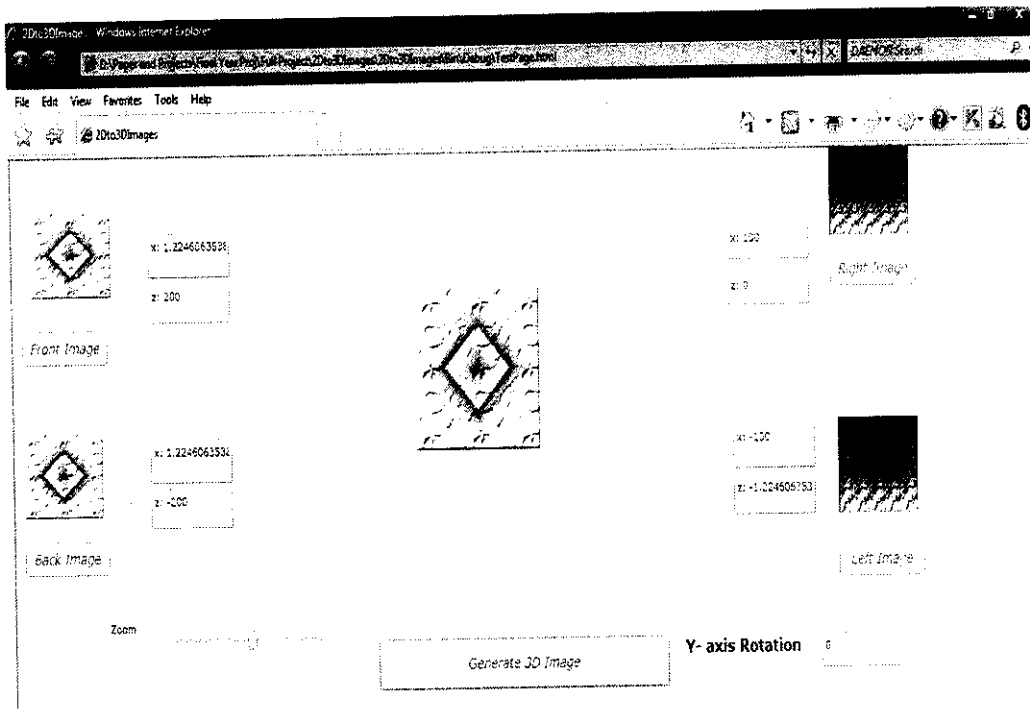
Rotation by PlaneProjection: The figure shows the main page of the module with no images loaded.



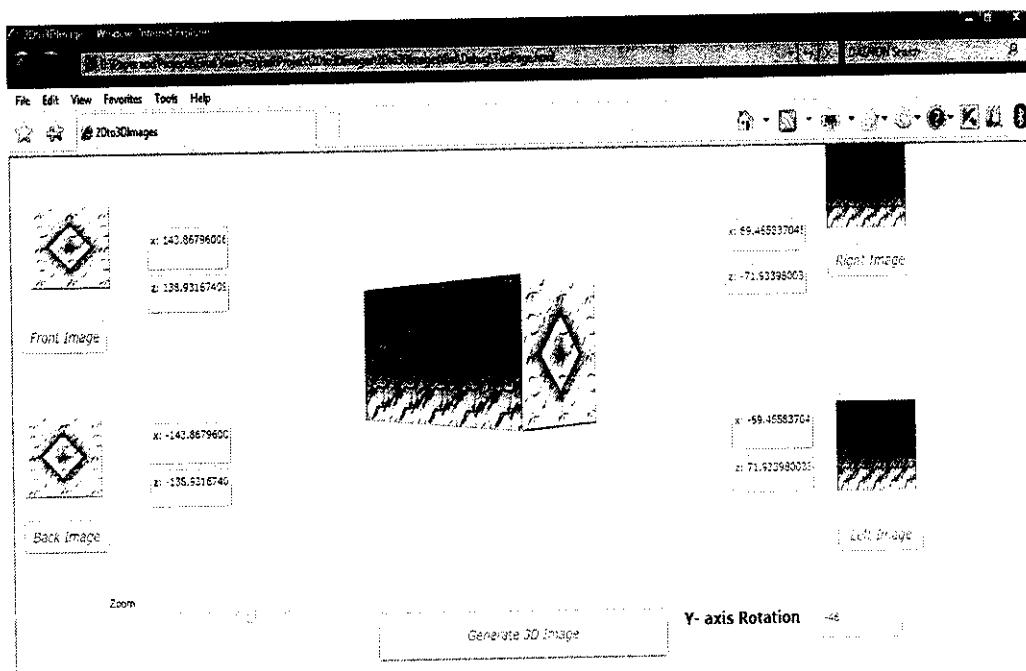
Rotation by PlaneProjection: The figure shows the selection of an input image for the front face to generate 3D object.



Rotation by PlaneProjection: The figure shows the application with all 4 input images loaded to generate 3D object.

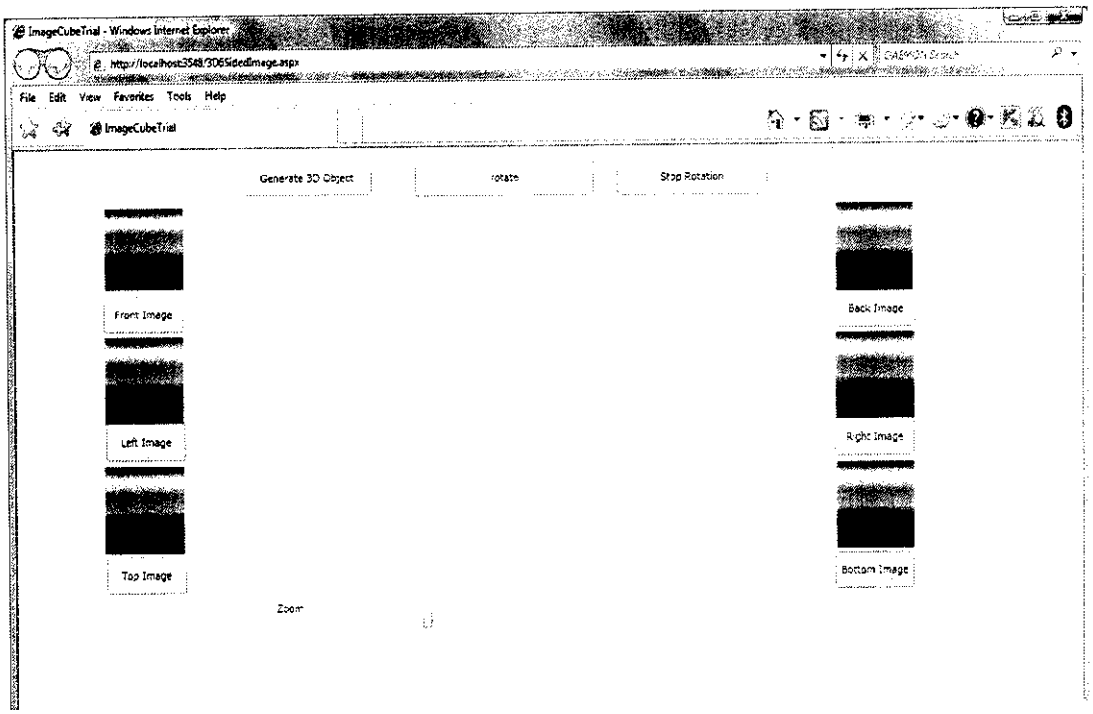


Rotation by PlaneProjection: The figure shows the generated 3D object with Y-axis rotation as 0.

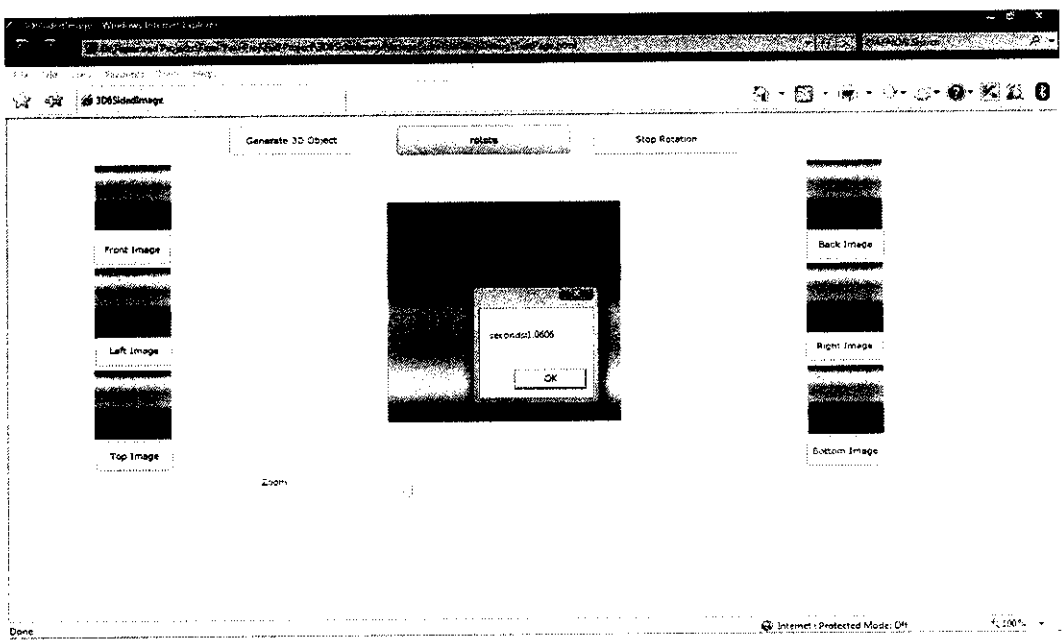


Rotation by PlaneProjection: The figure shows the generated 3D object rotated -46 degree along Y-axis.

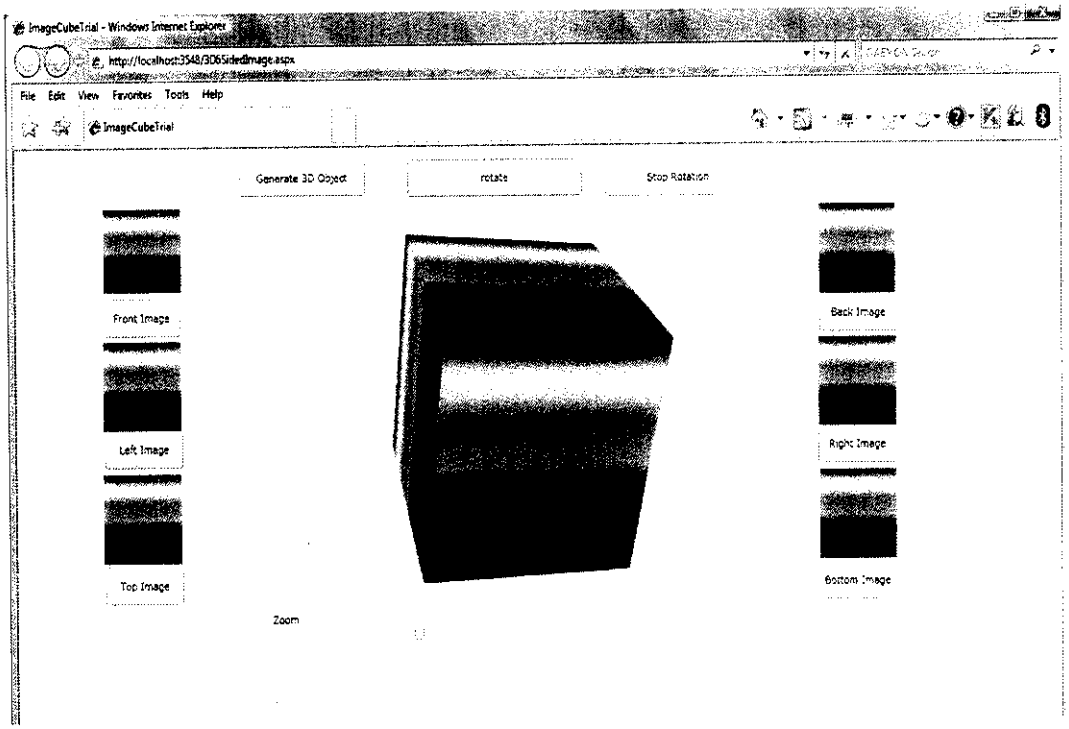
Rotation around Arbitrary Axes (Plane Projection)



Rotation by PlaneProjection: The figure shows the application with all 6 input images loaded to generate 3D object.

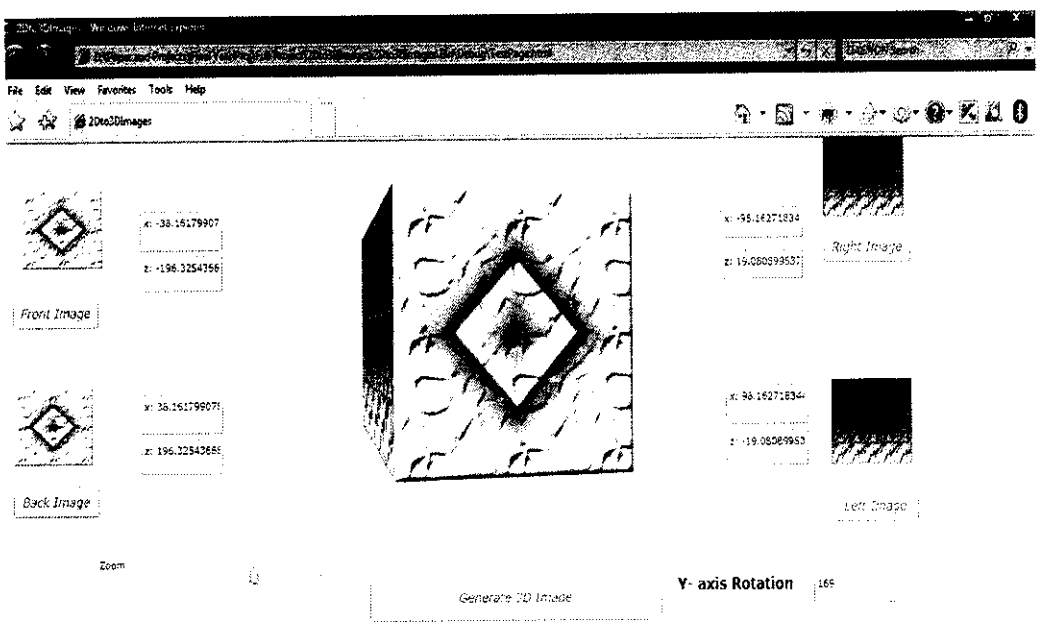


Rotation by PlaneProjection: The figure shows the time take to generate 3D object - 1.06 seconds.

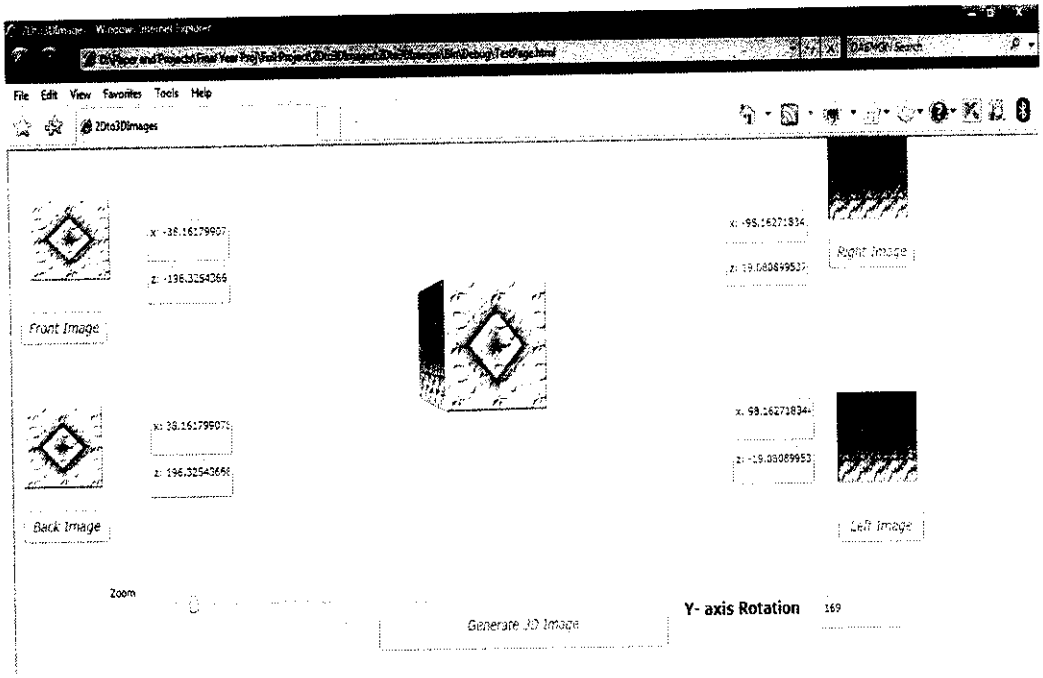


Rotation by PlaneProjection: The figure shows the generated 3D object rotated along all 3 axes.

Zooming

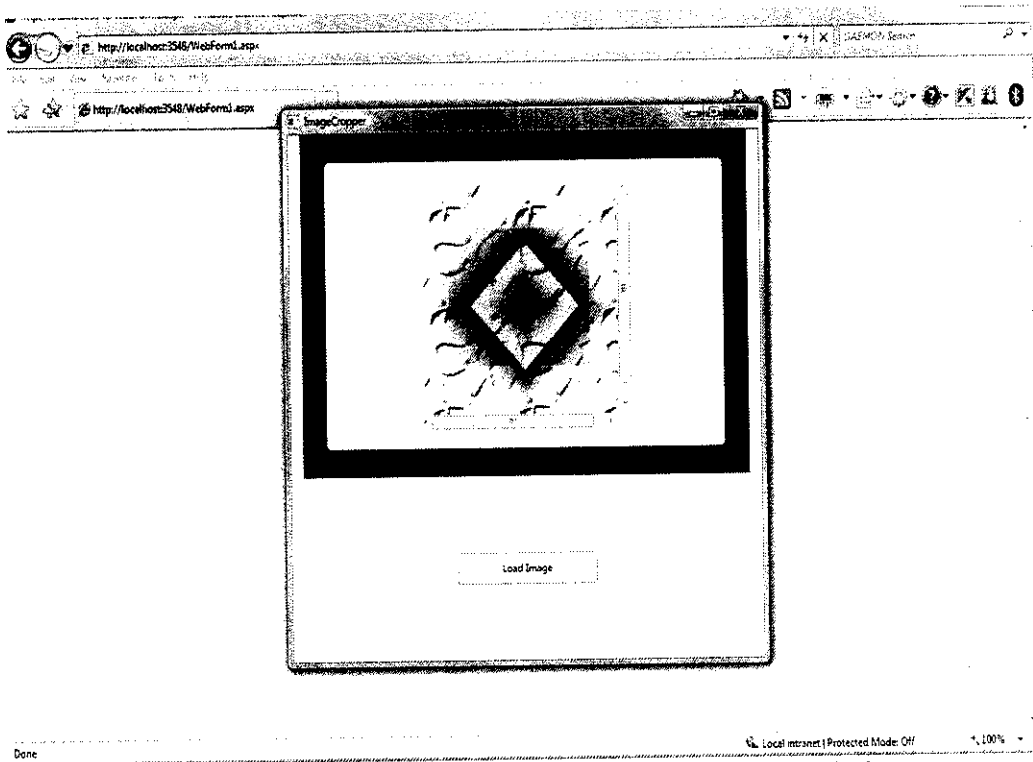


Zooming: The figure shows the 3D object zoomed in.



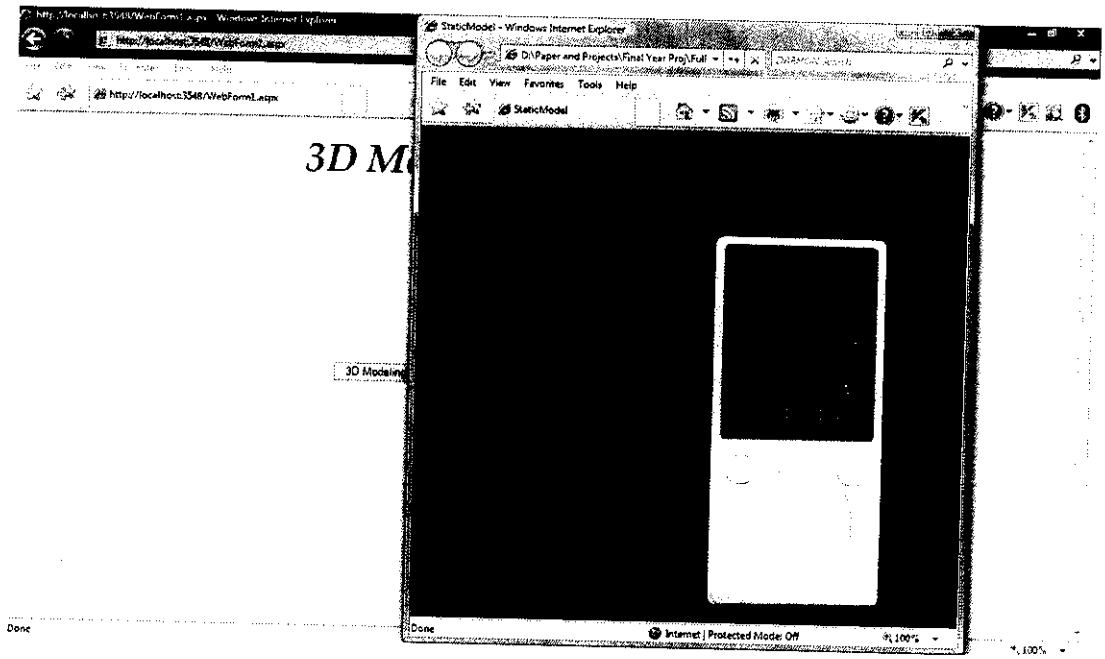
Zooming: The figure shows the 3D object zoomed out.

Cropping

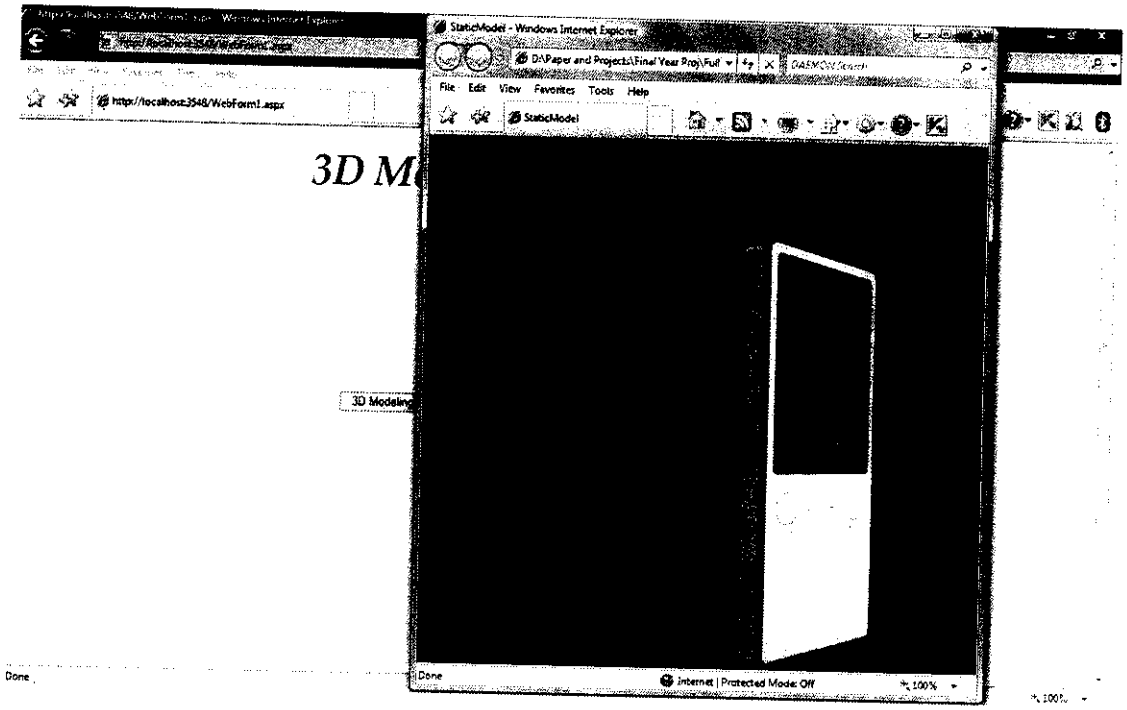


Cropping: The figure shows the image to be cropped and clip window.

ASYMMETRIC OBJECT:



Asymmetric Object: The figure shows a 3D object of a cell phone(with curved ends).



Asymmetric Object: The figure shows curved edges of the cell phone.

REFERENCES

REFERENCES

1. Donald Hearn and M.Pauline Baker, "Computer Graphics C Version", Pearson Education, 2003

2. http://mathworld.wolfram.com/Rotation.html

3. http://msdn.microsoft.com

4. http://www.c-sharpcorner.com

5. www.silverlight.net

6. www.wpftutorial.net

7. www.fact-index.com/q/qu/**quaternions**_and_spatial_rotation.html

8. www.wikipedia.org