

Oblique Projection

3D Computer Graphics and Animation Programming Assignment 2



Ida Bagus Bhaskara (001201500016)

Vera Debora Vitamas (001201500076)

Jonathan Surya Sandjaya (001201500055)

CIT 2 2015

**Table of Contents**

Cover Page 1

**Table of Contents 2**

[Chapter 1 Introduction 2](#_Toc506154720)

[Chapter 2 Basic Theory 2](#_Toc506154721)

[2.1 Transforming 3D Object into 2D Image 2](#_Toc506154722)

[2.2 Parallel Projections 2](#_Toc506154723)

[2.3 Axonometric and Oblique Projection and Its Differences 5](#_Toc506154724)

[2.4 Oblique Projection Transformation 11](#_Toc506154725)

[Chapter 3 Implementation 14](#_Toc506154726)

[3.1 Main Interface of the Application 14](#_Toc506154727)

[3.2 Features of the Applicaton 15](#_Toc506154728)

[Chapter 4 Design 15](#_Toc506154729)

[4.1 Variables Used in the Program 15](#_Toc506154730)

[4.2 Representation of Edge and Point in the Application 16](#_Toc506154731)

[4.3 Initializing the Cube 17](#_Toc506154732)

[4.4 Setting the Edge 18](#_Toc506154733)

[4.5 Setting the Vertex 18](#_Toc506154734)

[4.6 Drawing the Cube 18](#_Toc506154735)

[4.7 Hiding the Cube 19](#_Toc506154736)

[4.8 Setting the Matrix 19](#_Toc506154737)

[4.9 Multiplying the Matrix 19](#_Toc506154738)

[4.10 Rotation Tick Method 20](#_Toc506154739)

[Chapter 5 Evaluation 21](#_Toc506154740)

[5.1 Display the Oblique Projection with Different Value of Phi 21](#_Toc506154741)

[5.2 Display the Oblique Projections with Different Values of Alpha 22](#_Toc506154742)

[Chapter 6 Work Log 23](#_Toc506154743)

[Chapter 7 Conclusion and Remarks 24](#_Toc506154744)

# **Introduction**

1. About the Application

This is a 3D oblique projection simulator. It applies the oblique projection which uses computer graphic animation algorithms to rotate and to create an oblique view of a cube. This simulator allows the user to initialize the alpha and theta of the view, rotating the cube in 3 different coordinates (, and ), and stop the cube rotation.

This program uses Microsoft Visual Studio as the programming platform and Visual Basic as the programming language. This report covers the basic theory, implementation, design, evalution, work log, and conclusion.

# **Basic Theory**

## Transforming 3D Object into 2D Image

In order to transform 3D object to 2D image, the object's location in the Object Coordinate System will have to be determined. Then, it should be transformed to WCS (World Coordinate System) using (World Transformation) matrix. Transformation from WCS to VCS should occur, and it can be done using; consists of transformation matrices such as the rotation, translation, scaling, and shearing matrices. It is then translated to SCS (Screen Coordinate System) using (Screen Transformation).

## Parallel Projections

A line joining a point in a first plane and corresponding image are parallel in a projection from one plane to another plane. Z-coordinate and parallel lines are discarded on each vertex of the object which is extended until intersect in a view plane. The direction of the projection is specified from the center of projection. The distance from the center of projection to the project plane is an infinite one. It is less realistic and at the same time it is good for accurate measurement.

There are three types of Parallel Projections:

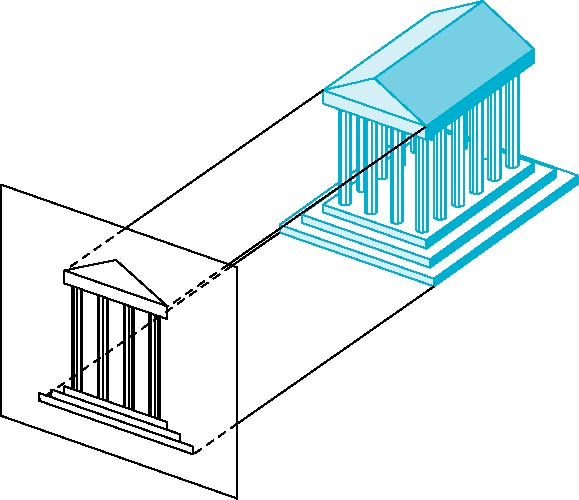
* Orthographic Projection

Figure 1 Projectors are orthogonal to projection surface.

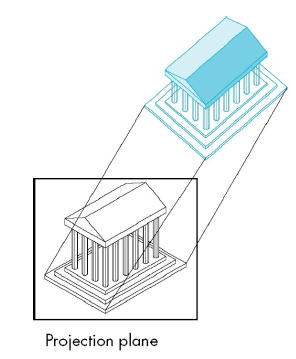
* Requires multiple views (x=0, y=0, or z=0 planes). One view is not adequate.
* True size and shape for lines.
* Requires multiple views (x=0, y=0, or z=0 planes). One view is not adequate.
* True size and shape for lines.
* Axonometric Projection

Figure 2 Additional rotation, translation, and then projection on the z = 0 plane.

* Classified by number of corners on a projected cube having the same angle.

None : trimetric

Two : dimetric

Three : isometric

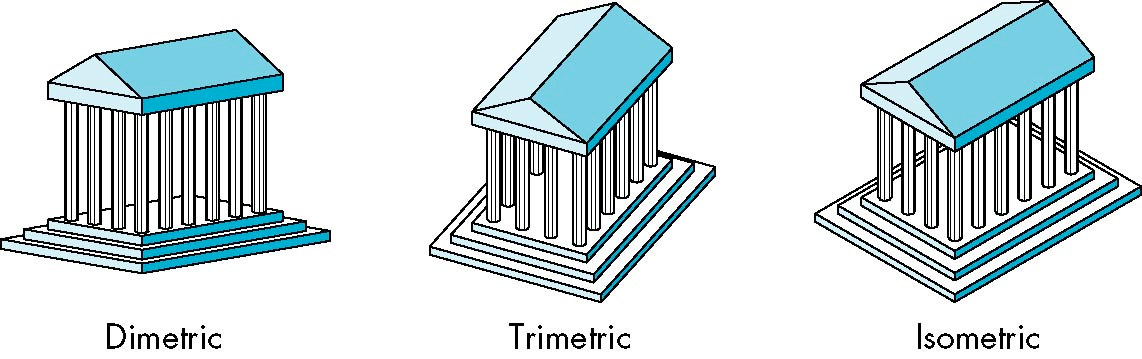


Figure 3 Axonometric Classifications

* Lines are scaled (foreshortened) but can find scaling factors.
  + Lines preserved but angles are not.
  + Projection of a circle in a plane not parallel to the projection plane is an ellipse.
* Can see three principal faces of a box-like object.
  + Some optical illusions possible.
  + Parallel lines appear to diverge.
* Does not look real because far objects are scaled the same as near objects.
* Oblique Projection

A pictorial view of an object showing its elevation, plan, or section to scale with parallel lines projected from the corners, at 45 degrees or any other angle, indicating the other sides.

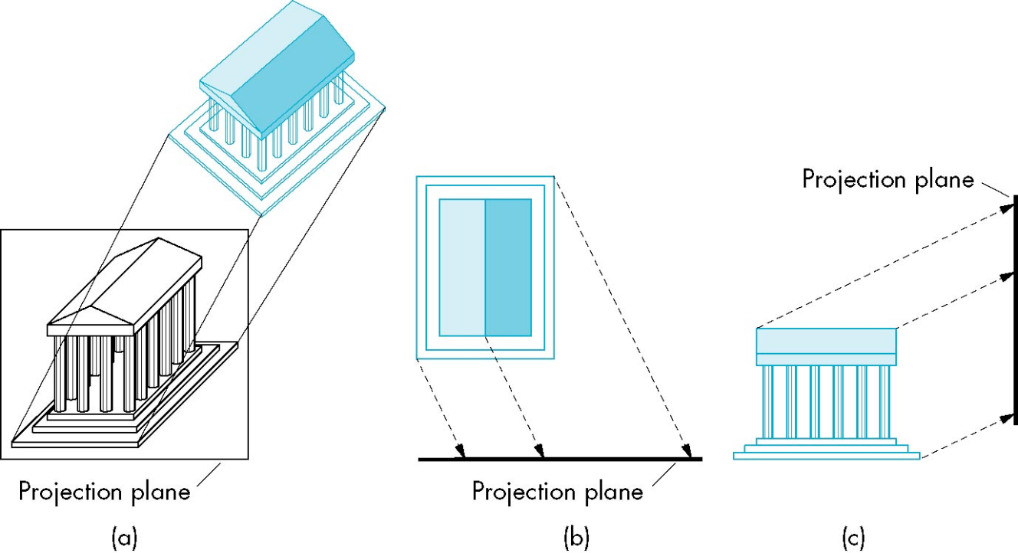


Figure 4 Arbitrary relationship between projectors.

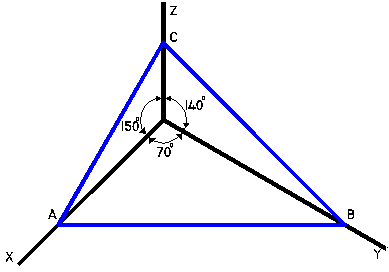
* Can pick the angles to emphasize a particular face.
* Angles in faces parallel to projection plane are preserved while we can still see “around” side.
* In physical world, cannot be created with simple camera; possible with bellows camera or special lens (architectural).

## Axonometric and Oblique Projection and Its Differences

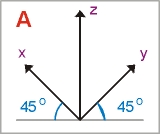
* 1. **Axonometric Parallel Projection**

Axonometric parallel projection is referred to as a projection that do not have vanishing points as the conventional perspective drawing glossary-tag consequently, all line on a common axis are draw as. It is how the object is rotated about the center point that creates the variations in the way it is constructed. parallel. There are three types of Axonometric.

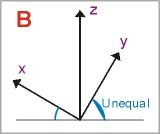
First is Trimetric Projection. In Trimetric Projection the projection of the three angles between the axes are unequal. Thus, three separate scales are needed to generate a trimetric projection of an object. The scales are constructed using the same method described in isometric and dimetric projection. An example below shows the method of construction for each of the scales.



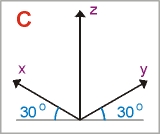
Second is Isometric Projection. Centered Isometric Projection is where the x & y axis are set at 45 degrees to the picture plane. As a result, the x & y axis are orthogonal to each other (or at 90 degrees).



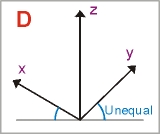
And for the rotated Isometric Projection is where the x & y axis are set at any angle to the picture plane yet are orthogonal to each other (or at 90 degrees).



Third is Dimetric Projection. Centered Dimetric Projection is where both the x & y axis is setup at less or greater than 45 degrees and are the same angle to the picture plane. As a result, the x & y axis is not orthogonal to each other.



Rotated Dimetric Projection is where both the x & y axis is setup at a different angle to the picture plane and are not orthogonal to each other.

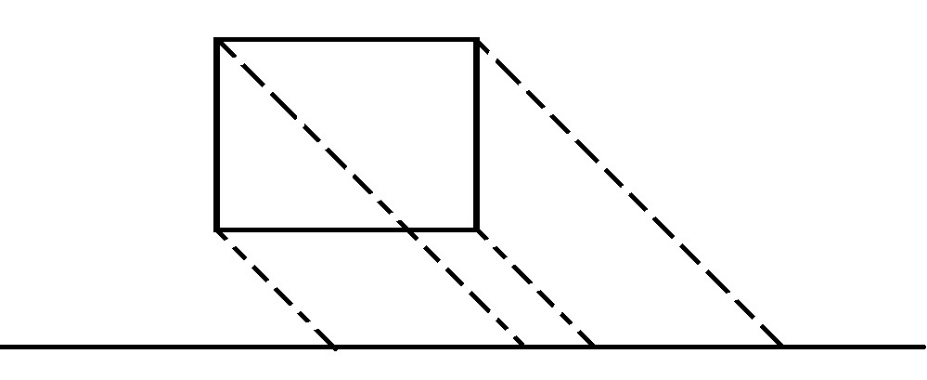


* 1. **Oblique Parallel Projection**

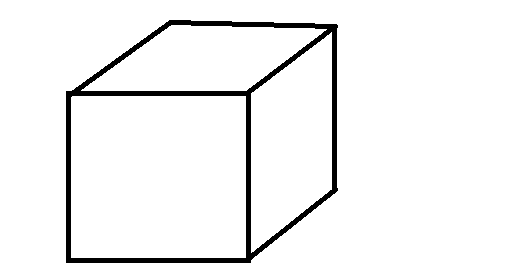
In parallel-projection view of scene is obtained by transferring object descriptions to the view plane along projection that can be any selected direction relative to the view-plane normal vector. When the projection path is not perpendicular to the view plane, this mapping called an **oblique parallel projection.** Using this projection can produce combinations such as front, side, and top view of an object. Oblique parallel projections are defined by a vector direction for the projection lines, and this direction can be specified in various ways.

Oblique parallel projections are often specified with two angles α and φ. A spatial position *(x, y, z),* in this illustration, is projected to ( on a view plane,

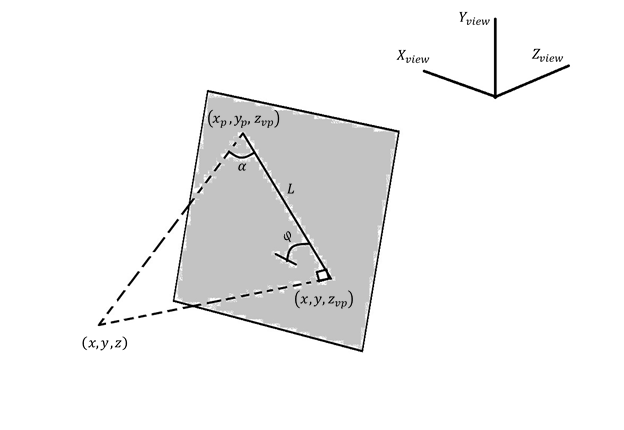
1. View plane



1. View plane



Oblique parallel projection of coordinate position (x, y, z) to position on a projection plane at position along the axis.

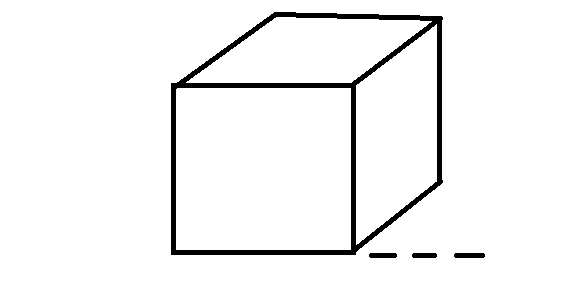


Which is at location along the viewing z axis. Position (x, y, ) is the corresponding orthogonal-projection point. The oblique parallel projection line from (x, y, z) to ( has an intersection angle α with the line on the projection plane that joins (, ) and (x, y, ). This view-plane line, with length L is at an angle φ with the horizontal direction in the projection plane. Angle α can be assigned a value between 0 and 90, and angle φ can be vary from 0 to 360. We can express the projection coordinates in terms of x, y, L, and φ as

Length L depends on the angle α and the perpendicular distance of the point (x, y, z) from the view plane:

Where = , which is also the value of L when . And the equation could be:

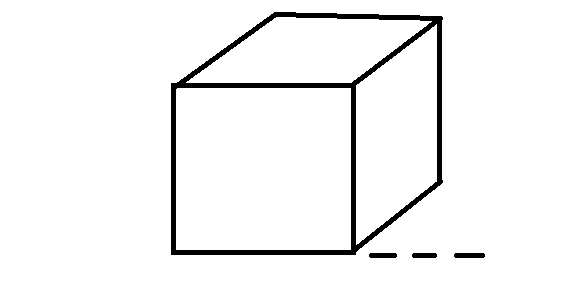
An orthogonal projection is obtained when = 0 (which occurs at the projection angle α = 90). Those equations above represent z-axis shearing transformation. In fact, the effect of an oblique parallel projection is to shear planes of constant z and project them onto the view plane. The (x, y) positions on each plane of constant z are shifted by an amount proportional to the distance of the plane from the view plane, so that angles, distances, and parallel lines in the plane are projected accurately.



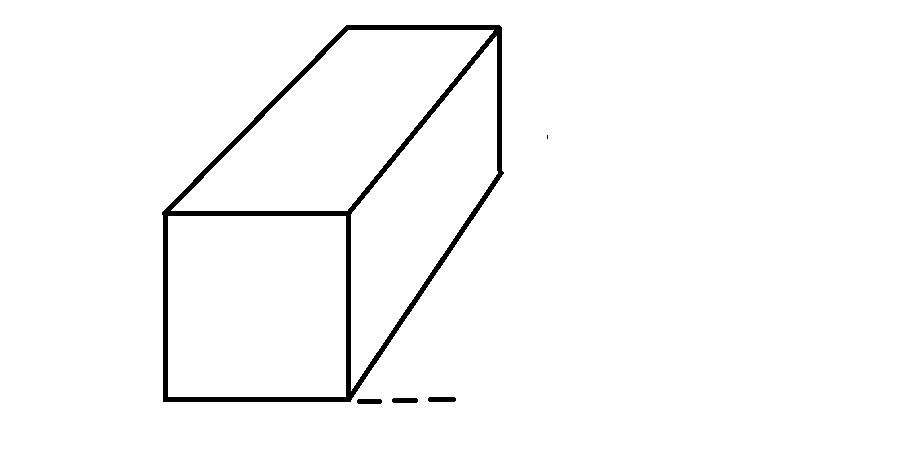
Where the view plane is positioned at the front face a cube. The back plane of the cube is sheared and overlapped with the front plane in the projection to the viewing surface. A side edge of the cube connecting the front and back planes is projected into a line of length that makes and angle φ with a horizontal line in the projection plane.

Typical choices for angle φ are 30 and 45, which display a combination view of the front, side, and top (or front, side, and bottom) of an object. Two commonly used values for are those for which = 1 and tan = 2. For the first case, and the views obtained are called **cavalier** projections. All lines perpendicular to the projection plane are projected with no change with length.

a)



b)



(cavalier projections of cube onto a view plane for two values of angle . The depth of the cubes projected with a length equal to that width and height)

When the projection angle is chosen so that tan = 2, the resulting view is called a cabinet projection. For this angle (, lines perpendicular to the viewing surface are projected at half their length. Cabinet projections appear more realistic than cavalier projections because of this reduction in length of perpendiculars.

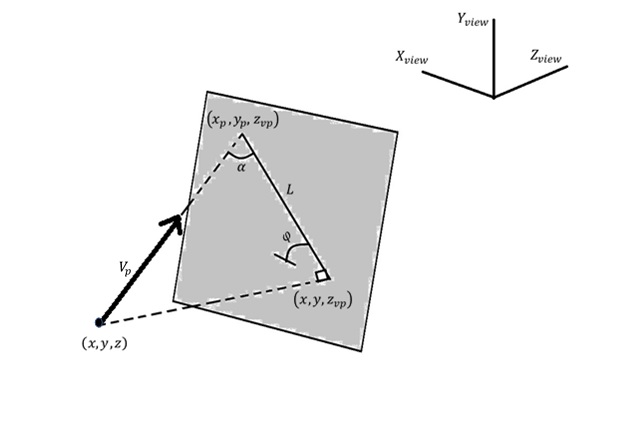
* 1. **Difference Between Axonometric and Oblique Parallel Projection**

The difference between Axonometric parallel projection and Oblique parallel projection is the front side of Axonometric Parallel Projection is could be changed, but on Oblique parallel projection.

## Oblique Parallel Projection Transformation

In graphics programming libraries that support oblique parallel projections, the direction of projection to the view plane specified with a **parallel-projection vector**, . This direction vector can be designated with a reference position relative to the view point, as we did with the view-plane normal vector, or with any other two points. Some packages use a reference point relative to center of the clipping window to define the direction for a parallel projection. If the projection vector is specified in world coordinates, it must be transformed to viewing coordinates using the rotation matrix.

Once the projection vector is established in viewing coordinates, all points in the scene are transferred to the view plane along lines that are parallel to this vector.



Illustrates an oblique parallel projection of a spatial point to the view plane. It can denote the components of the projection vector relative to the viewing-coordinate frame as = ( where . By using the picture above, get this equation:

In terms of the projection vector as

The oblique parallel-projection coordinates on equation above reduce to the orthogonal projection, when .

Using the projection-vector parameters from the equation above, can express the elements of the transformations matrix for an oblique parallel projection as

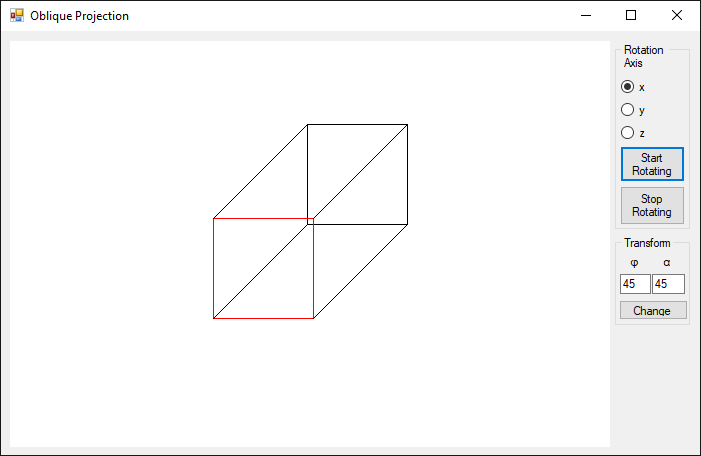
This matrix shifts the values of the x and y coordinates by an amount proportional to the distance from the view plane, which is at position on the axis. The z values of spatial positions are unchanged. If , using an orthogonal projection and using matrix is reduce to the identify matrix.

For general oblique parallel projection, matrix represents a z-axis shearing transformation.

# **Implementation**

## Main Interface of the Application

This application has one interface, which it uses as its main interface. The interface is pictured below:



7

6

5

4

3

2

1

The interface consists of the following components:

1. Canvas Picture Box

The canvas is used to display the cube using the application.

1. Rotation Axis Radio Buttons

The radio buttons are used to change the , and coordinate of cube rotation.

1. Start Rotating Button

The button is used to rotate the cube in the canvas.

1. Stop Rotating Button

The button is used to stop the rotation of the cube.

1. Phi Textbox

The input box is used to determine the value of the phi.

1. Alpha Textbox

The input box is used to determine the value of the alpha.

1. Change Button

The change button is used to change the value of alpha and theta of the cube based on the value entered on the textboxes.

## Features of the Applicaton

The application has several features, namely:

1. Rotating the 3D Cube Around the , and Axes

To rotate the cube, the user must choose 1 out of 3 axes shown in the radio button and then click the start rotating button to rotate the 3D cube.

1. Changing the Values of Alpha () and Phi ()

To change the value of alpha and theta the user must input the values in the textboxes and click change button.

1. Stopping the Rotation of the Cube

To stop the cube rotation the user must click the stop rotating button.

# Design

## Variables Used in the Program

The following variables are used in this program:

* Dim graphics As Graphics  
  This variable is used to pass the canvas variable to the Graphics class so that functions from the Graphics class, for example the DrawLine function, can be used to draw the cube’s edges in PictureBox1, which acts as the canvas.
* Dim canvas As Bitmap  
  This variable is used to initialize the canvas as a New Bitmap with PictureBox1’s width and height as its properties.
* Dim phi As Single

This variable used to store the value of the cube.

* Dim alpha As Single

This variable used to store the value of the cube.

* Dim vertex(7) As Point

This array variable is used to store the cube’s vertices.

* Dim edges(12) As Edge

This array variable is used to store the cube’s edges.

* Dim view(3, 3), screen(3, 3) As Single

These 2D array variable is used to store the and row matrices.

* Dim VR(7), VS(7) As Point

These 2D array variable is used to store the result of multiplying the cube’s vertex with the and matrices respectively.

* Dim deg As Single = 0

This variable is used to store the degree that is used to calculate the rotation matrix and will be incremented each tick in the RotationTick\_Tick.

## Representation of Edge and Point in the Application

The edge and point are both represented as a Structure in this application. Inside Structure Edge there are two variables defined, namely point1 and point2. These variables will define the two vertices that creates an edge. There is one variable that is defined as Structure Edge in this application, and that is the edges(12) variable.

Structure Point consists of four variables, namely x, y, z, w. These variables will define the 1x4 matrix that determine the position of the vertex. There are three variables that are defined as Structure Point in this application, they are vertex(7), VR(7), and VS(7).

## Initializing the Cube

The cube is initialized by the Sub Init. The Sub initalized the cube by calling several Subs, namely SetPoint, SetColMat, SetEdge, MultiplyMat, and DrawCube. First, the Sub fills the vertex array using SetPoint. Then, it sets the and matrix column-by-row with SetColMat. Edges are then set using SetEdge. It then multiplies the vertices with and by multiplying each vertex with and using MultiplyMat and looping it for 8 times. It then calls DrawCube to show the 3D cube on the canvas.

Sub Init(phi As Single, alpha As Single)

SetPoint(vertex(0), -1, -1, 1)

SetPoint(vertex(1), 1, -1, 1)

SetPoint(vertex(2), 1, 1, 1)

SetPoint(vertex(3), -1, 1, 1)

SetPoint(vertex(4), -1, -1, -1)

SetPoint(vertex(5), 1, -1, -1)

SetPoint(vertex(6), 1, 1, -1)

SetPoint(vertex(7), -1, 1, -1)

SetColMat(screen, 0, 50, 0, 0, 300)

SetColMat(screen, 1, 0, -50, 0, 180)

SetColMat(screen, 2, 0, 0, 0, 0)

SetColMat(screen, 3, 0, 0, 0, 1)

SetColMat(view, 0, 1, 0, (CotDegree(phi) \* CosDegree(alpha)) \* 2, 0)

SetColMat(view, 1, 0, 1, (CotDegree(phi) \* SinDegree(alpha)) \* 2, 0)

SetColMat(view, 2, 0, 0, 0, 0)

SetColMat(view, 3, 0, 0, 0, 1)

SetEdge(edges(0), 0, 1)

SetEdge(edges(1), 1, 2)

SetEdge(edges(2), 2, 3)

SetEdge(edges(3), 3, 0)

SetEdge(edges(4), 4, 5)

SetEdge(edges(5), 5, 6)

SetEdge(edges(6), 6, 7)

SetEdge(edges(7), 7, 4)

SetEdge(edges(8), 0, 4)

SetEdge(edges(9), 1, 5)

SetEdge(edges(10), 2, 6)

SetEdge(edges(11), 3, 7)

For i = 0 To 7

VR(i) = MultiplyMat(vertex(i), view)

VS(i) = MultiplyMat(VR(i), screen)

Next

DrawCube()

End Sub

## Setting the Edge

The edge is defined by the edge variable, which is defined as Structure Edge. It accepts three parameters; the edge array, the first point, and the second point.

Sub SetEdge(ByRef edge As Edge, n1 As Integer, n2 As Integer)

edge.point1 = n1

edge.point2 = n2

End Sub

## Setting the Vertex

The vertex is defined by the vertex variable, which is defined as Structure Point. It accepts four parameters; the vertex array and the x,y, and z axis position of the vertex.

Sub SetPoint(ByRef point As Point, x As Integer, y As Integer, z As Integer)

point.x = x

point.y = y

point.z = z

point.w = 1

End Sub

## Drawing the Cube

The cube is drawn using DrawCube. Showing the front side of the cube as a red-outlined square is done by drawing the first four edges stored in the edges array black and then the next four red. The cube is then completed by drawing the rest of the edges black.

Sub DrawCube()

Dim i, j, k As Integer

For i = 0 To 3

graphics.DrawLine(Pens.Black, VS(edges(i).point1).x, VS(edges(i).point1).y, VS(edges(i).point2).x, VS(edges(i).point2).y)

Next

For j = 4 To 7

graphics.DrawLine(Pens.Red, VS(edges(j).point1).x, VS(edges(j).point1).y, VS(edges(j).point2).x, VS(edges(j).point2).y)

Next

For k = 8 To 11

graphics.DrawLine(Pens.Black, VS(edges(k).point1).x, VS(edges(k).point1).y, VS(edges(k).point2).x, VS(edges(k).point2).y)

Next

PictureBox1.Image = canvas

End Sub

## Hiding the Cube

Hiding the cube is done using the DrawLine method, similar to DrawCube. But instead of drawing the edges black, in this method the cube is drawn white; similar to the canvas.

Sub HideCube()

Dim i As Integer

For i = 0 To 11

graphics.DrawLine(Pens.White, VS(edges(i).point1).x, VS(edges(i).point1).y, VS(edges(i).point2).x, VS(edges(i).point2).y)

Next

PictureBox1.Image = canvas

End Sub

## Setting the Matrix

In this application, matrices are defined column-by-column. Defining the matrix is done via Sub SetColMat. It accepts six parameters; the transformation matrix, the column index, and the values to be set into the matrix, which is a, b, c, and d value.

Sub SetColMat(ByRef Matrix(,) As Single, col As Integer, a As Double, b As Double, c As Double, d As Double)

Matrix(0, col) = a

Matrix(1, col) = b

Matrix(2, col) = c

Matrix(3, col) = d

End Sub

## Multiplying the Matrix

Multiplying the matrix is done via Function MultiplyMat and returns Point. The function accepts two parameters, namely the vertex and the multiplier matrix. It multiplies the vertex with the matrix column-by-row.

Function MultiplyMat(point As Point, M(,) As Single) As Point

Dim result As Point

result.x = (point.x \* M(0, 0) + point.y \* M(1, 0) + point.z \* M(2, 0) + point.w \* M(3, 0))

result.y = (point.x \* M(0, 1) + point.y \* M(1, 1) + point.z \* M(2, 1) + point.w \* M(3, 1))

result.z = (point.x \* M(0, 2) + point.y \* M(1, 2) + point.z \* M(2, 2) + point.w \* M(3, 2))

result.w = 1

Return result

End Function

## Rotation Tick Method

In this application, the cube can be rotated based on the axis the user selected. The cube is then rotated per tick. This means that the cube changes it degree of rotation every tick. At the beginning of the tick, it hid the cube beforehand using HideCube. It then increments the deg variable by 5. Before setting the Rotation matrix, it checks which axis radio button is currenly selected by the user, whether it’s the x, y, or the z axis radio button. It then multiplies the vertices with , and by multiplying each vertex with and using MultiplyMat and looping it for 8 times. It then calls DrawCube to show the 3D cube on the canvas. It also refreshes the canvas after it has drawn the cube.

Private Sub RotationTick\_Tick(sender As Object, e As EventArgs) Handles RotationTick.Tick

Dim Rot(3, 3) As Single

HideCube()

deg = deg + 5

If XButton.Checked = True Then

SetColMat(Rot, 0, 1, 0, 0, 0)

SetColMat(Rot, 1, 0, CosDegree(deg), -SinDegree(deg), 0)

SetColMat(Rot, 2, 0, SinDegree(deg), CosDegree(deg), 0)

SetColMat(Rot, 3, 0, 0, 0, 1)

ElseIf YButton.Checked = True Then

SetColMat(Rot, 0, CosDegree(deg), 0, SinDegree(deg), 0)

SetColMat(Rot, 1, 0, 1, 0, 0)

SetColMat(Rot, 2, -SinDegree(deg), 0, CosDegree(deg), 0)

SetColMat(Rot, 3, 0, 0, 0, 1)

ElseIf ZButton.Checked = True Then

SetColMat(Rot, 0, CosDegree(deg), -SinDegree(deg), 0, 0)

SetColMat(Rot, 1, SinDegree(deg), CosDegree(deg), 0, 0)

SetColMat(Rot, 2, 0, 0, 1, 0)

SetColMat(Rot, 3, 0, 0, 0, 1)

End If

For i = 0 To 7

VR(i) = MultiplyMat(vertex(i), Rot)

VR(i) = MultiplyMat(VR(i), view)

VS(i) = MultiplyMat(VR(i), screen)

Next

DrawCube()

PictureBox1.Refresh()

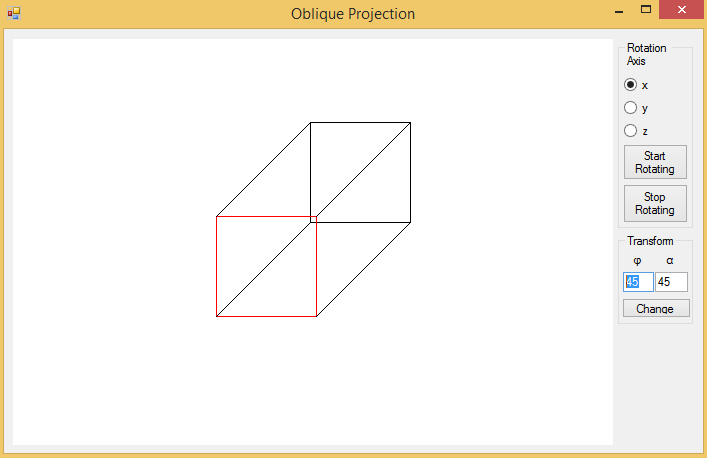
End Sub

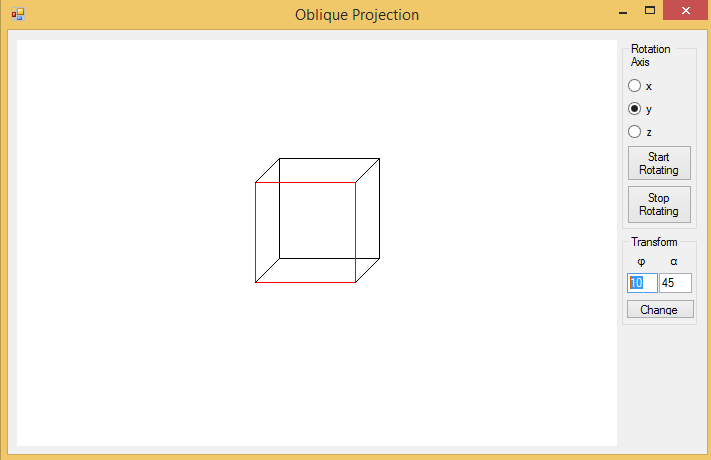
# Evaluation

To ensure that the application works as it is designed, test cases are conducted. These test cases are as follows:

## Display the Oblique Projection with Different Value of Phi

In this case, the user tries to change the theta value in the projection.

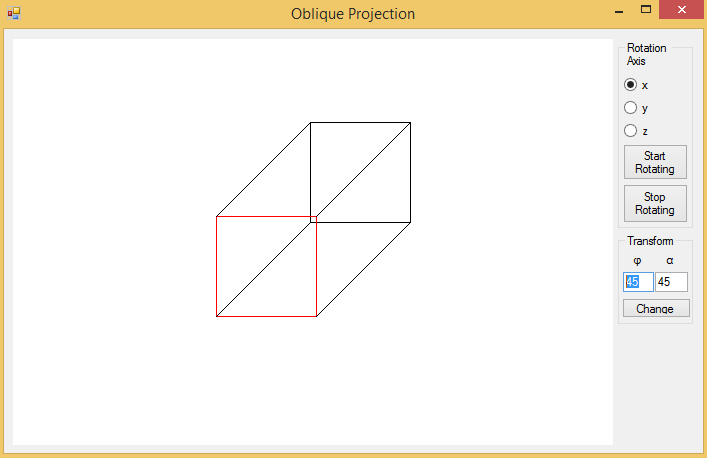


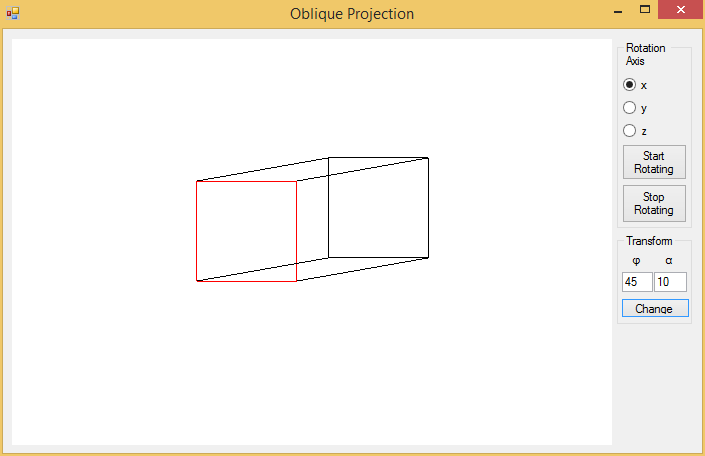


This case is successful because the program can change the view of the cube.

## Display the Oblique Projections with Different Values of Alpha

In this case, the user tries to change the alpha value in the projection.





This case is successful because the program can change the view of the cube.

# Work Log

The work log is extracted directly from Visual Studio’s Git Log History, which is also available publicly at <https://github.com/bakanui/ObliqueProjection/commits/master>.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Author | Date | Time | Commit Message |
| 1 | Bhaskara Ida Bagus | 1/26/2018 | 5:39:49 PM | Initialized project |
| 2 | Vera Debora Vitamas | 1/29/2018 | 1:57:02 PM | Trying to create matrix structure |
| 3 | Vera Debora Vitamas | 1/29/2018 | 3:30:21 PM | Trying Matrix Multiplication |
| 4 | Vera Debora Vitamas | 1/29/2018 | 7:05:34 PM | Trying to initialize cube |
| 5 | Bhaskara Ida Bagus | 2/2/2018 | 11:05:35 PM | Add report template |
| 6 | Bhaskara Ida Bagus | 2/3/2018 | 3:24:05 PM | Buggy cube; shows a square instead of a cube |
| 7 | Bhaskara Ida Bagus | 2/3/2018 | 3:50:18 PM | Square is now a Cube, but not Oblique |
| 8 | Jonathan Surya Sandjaya | 2/4/2018 | 8:12:53 PM | Cube is now oblique |
| 9 | Bhaskara Ida Bagus | 2/4/2018 | 9:14:46 PM | Phi and Theta can now be set by user, buggy rotation |
| 10 | Bhaskara Ida Bagus | 2/4/2018 | 10:19:30 PM | Declaring DegToRad as a function |
| 11 | Bhaskara Ida Bagus | 2/5/2018 | 11:26:05 PM | Rotation in the x axis works |
| 12 | Bhaskara Ida Bagus | 2/6/2018 | 6:30:29 PM | Beautifying UI, Rotation on x, y, and z axis can now be done |
| 13 | Bhaskara Ida Bagus | 2/6/2018 | 6:45:40 PM | Fix Rot Matrix, Renaming theta to alpha |
| 14 | Bhaskara Ida Bagus | 2/7/2018 | 1:43:12 AM | Update work log in report |
| 15 | Jonathan Surya Sandjaya | 2/7/2018 | 7:06:55 PM | creating design and introduction in report |
| 16 | Jonathan Surya Sandjaya | 2/11/2018 | 7:03:14 PM | Add reset button |
| 17 | Jonathan Surya Sandjaya | 2/11/2018 | 7:33:47 PM | update conclusion and remarks in report |
| 18 | Vera Debora Vitamas | 2/11/2018 | 8:21:45 PM | Updating basic theory |
| 19 | Bhaskara Ida Bagus | 2/11/2018 | 10:28:32 PM | Update design, work log of report. Minor grammar checks. Finalizing report. |

# Conclusion and Remarks

The program works well. The 3D cube rotation, oblique projection, and reset work as expected. This can be proven by the success of the program in conducting each test case. Overall, this program doesn’t have anything bug.

Through this programming assignment, we learn some important things such as it’s definitely hard to do this programming assignment in 2 weeks , we can know the ability and work ethic of each member, how to divide the time efficiently between doing this programming assignment and other assignments, and the most important thing is through this programming assignment, we can improve our coding skill where we are forced to work harder in order to reach a goal in the certain limit time.