ACKNOWLEDGEMENT

It gives us immense pleasure to present before you our project titled **‘3D MODEL OF TOMB’.** The joy and satisfaction that accompany the successful completion of any task would be incomplete without the mention of those who made it possible. We are glad to express our gratitude towards our prestigious institution **DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT** for providing us with utmost knowledge, encouragement and the maximum facilities in undertaking this project.

We express our deepest gratitude and special thanks to **Dr. C. Nandini**, **Prof & H.O.D, Dept. Of Computer Science Engineering**, for all her guidance and encouragement.

We sincerely acknowledge the guidance and constant encouragement of our mini-project guides, **Assistant Prof. Mr. Raghu. M. T**

AISHWARYA NAIK (1DT15CS007),

NIVEDITHA S. (1DT15CS074)

AND

JACINTH (1DT15CS042)

TABLE OF CONTENTS

Chapter No.	Name	Page No.
1.	**INTRODUCTION**	
	1.1 Computer Graphics	1
	1.2 OpenGL Technology	2
	1.3 Project Description	4
	1.4 Functions Used	5
2.	**REQUIREMENT SPECIFICATION**	
	2.1 Hardware Requirements	12
	2.2 Software Requirements	12
3.	**INTERFACE AND ARCHITECTURE**	
	3.1 Interface	13
	3.2 Architecture	14
4.	**IMPLEMENTATION**	15
5.	**SNAPSHOTS**	39
6.	**FUTURE ENHANCEMENT**	42
7.	**CONCLUSION**	43
8.	**REFERENCES**	44

Chapter-1

INTRODUCTION

1.1 Computer Graphics

Computer graphics are graphics created using computers and the representation of image data by a computer specifically with help from specialized graphic hardware and software.

The interaction and understanding of computers and interpretation of data has been made easier because of computer graphics. Computer graphic development has had a significant impact on many types of media and have revolutionized animation, movies and the video game industry.

Computer graphics is widespread today. Computer imagery is found on television, in newspapers, for example in weather reports, or for example in all kinds of medical investigation and surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. In the media "such graphs are used to illustrate papers, reports, thesis", and other presentation material.

Computer generated imagery can be categorized into several different types: two dimensional (2D), three dimensional (3D), and animated graphics. As technology has improved, 3D computer graphics have become more common, but 2D computer graphics are still widely used. Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like information visualization, and scientific visualization more concerned with "the visualization of three dimensional phenomena (architectural, meteorological, medical, biological, etc.), where the emphasis is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic (time) component".

OpenGL Utility Toolkit (GLUT) was developed by Mark Kilgard, it Hides the complexities of differing window system APIs, Default user interface for class projects, Glut routines have prefix glut, Eg- glutCreateWindow().

1.2 OpenGL Technology

OpenGL is a graphics application programming interface (API) which was originally developed by Silicon Graphics. OpenGL is not in itself a programming language, like C++, but functions as an API which can be used as a software development tool for graphics applications. The term Open is significant in that OpenGL is operating system independent. GL refers to graphics language. OpenGL also contains a standard library referred to as the OpenGL Utilities (GLU). GLU contains routines for setting up viewing projection matrices and describing complex objects with line and polygon approximations.

OpenGL gives the programmer an interface with the graphics hardware. OpenGL is a lowlevel, widely supported modelling and rendering software package, available on all platforms. It can be used in a range of graphics applications, such as games, CAD design, modelling.

OpenGL is the core graphics rendering option for many 3D games, such as Quake 3. The providing of only low-level rendering routines is fully intentional because this gives the programmer a great control and flexibility in his applications. These routines can easily be used to build high-level rendering and modelling libraries. The OpenGL Utility Library (GLU) does exactly this, and is included in most OpenGL distributions! OpenGL was originally developed in 1992 by Silicon Graphics, Inc, (SGI) as a multi-purpose, platform independent graphics API. Since 1992 all of the development of OpenGL.

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications.

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using.

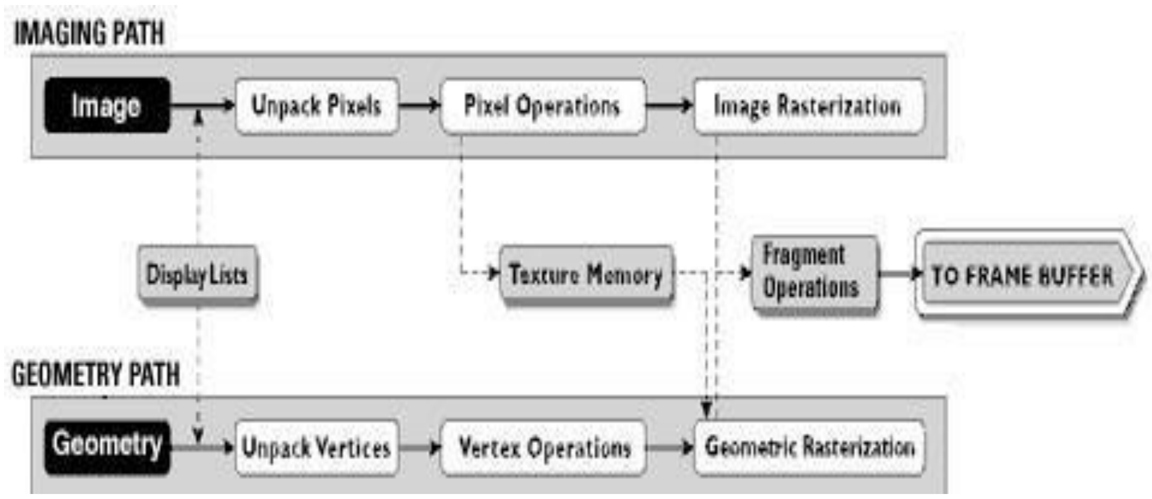


Figure. 1.1: OpenGl Visualization Programming Pipeline

OpenGL operates on image data as well as geometric primitives. Simplifies Software Development, Speeds Time-to-Market .

Routines simplify the development of graphics software—from rendering a simple geometric point, line, or filled polygon to the creation of the most complex lighted and texturemapped NURBS curved surface. OpenGL gives software developers access to geometric and image primitives, display lists, modeling transformations, lighting and texturing, anti-aliasing, blending, and many other features. Every conforming OpenGL implementation includes the full complement of OpenGL functions. The well-specified OpenGL standard has language bindings for C, C++, Fortran, Ada, and Java. All licensed OpenGL implementations come from a single specification and language binding document and are required to pass a set of conformance tests. Applications utilizing OpenGL functions are easily portable across a wide array of platforms for maximized programmer productivity and shorter time-to-market.

All elements of the OpenGL state—even the contents of the texture memory and the frame buffer—can be obtained by an OpenGL application. OpenGL also supports visualization applications with 2D images treated as types of primitives that can be manipulated just like 3D geometric objects. As shown in the OpenGL visualization programming pipeline diagram above.

1.3 PROJECT DESCRIPTION:

Buddhist temple is the place of worship for Buddhists, the followers of Buddhism. They include the structures called vihara, stupa, wat and pagoda in different regions and languages. Temples in Buddhism represent the pure land or pure environment of a Buddha. Traditional Buddhist temples are designed to inspire inner and outer peace. Its structure and architecture varies from region to region. Usually, the temple consists not only of its buildings, but also the surrounding environment. The Buddhist temples are designed to symbolize 5 elements: Fire, Air, Earth, Water, and Wisdom.

A 3D Model is a mathematical representation of any three-dimensional object (real or imagined) in a 3D software environment. Unlike a 2D image, 3D models can be viewed in specialized software suites from any angle, and can be scaled, rotated, or freely modified. 3D modeling is used in various industries like films, animation and gaming, interior designing and architecture. They are also used in the medical industry for the interactive representations of anatomy.

We make use of C with OpenGl for entire coding purpose along with some features of Windows. The OpenGl Utility is a Programming Interface. The toolkit supports much functionalities like multiple window rendering, callback event driven processing using sophisticated input devices etc.

1.4 OpenGl Functions Used:

This project is developed using CodeBlocks and this project is implemented by making extensive use of library functions offered by graphics package of OpenGl, a summary of those functions follows:

1.4.1 **glBegin() :**

Specifies the primitives that will be created from vertices presented between glBegin and subsequent glEnd.. GL_POLYGON, GL_LINE_LOOP etc.

1.4.2 **glEnd(void) :**

It ends the list of vertices.

1.4.3 **glPushMatrix() :** *void **glPushMatrix**(void)* glPushMatrix pushes the current matrix stack down by one level, duplicating the current matrix.

1.4.4 **glPopMatrix() :**

*void **glPopMatrix(**void)*

glPopMatrix pops the top matrix off the stack, destroying the contents of the popped matrix. Initially, each of the stacks contains one matrix, an identity matrix.

1.4.5 **glTranslate() :**

*void **glTranslate**(GLdouble x, GLdouble y, GLdouble z)*

Translation is an operation that displaces points by a fixed distance in a given direction. *Parameters x*, *y*, *z* specify the *x*, *y*, and *z* coordinates of a translation vector. Multiplies current matrix by a matrix that translates an object by the given x, y and z-values.

1.4.6 **glClear() :**

*void **glClear**(GLbitfield mask)* glClear takes a single argument that is the bitwise *or* of several values indicating which buffer is to be cleared.

GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT, GL_ACCUM_

BUFFER_BIT, and GL_STENCIL_BUFFER_BIT. Clears the specified buffers to their current clearing values.

1.4.7 **glClearColor() :**

*void **glClearColor**(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)*

Sets the current clearing color for use in clearing color buffers in RGBA mode. The red, green, blue, and alpha values are clamped if necessary to the range [0,1]. The default clearing color is (0, 0, 0, 0), which is black.

1.4.8 **glMatrixMode() :**

*void **glMatrixMode**(GLenum mode)*

It accepts three values GL_MODELVIEW, GL_PROJECTION and GL_TEXTURE. It specifies which matrix is the current matrix. Subsequent transformation commands affect the specified matrix.

1.4.9 **glutInitWindowPosition() :**

*void **glutInitWindowPosition**(int x, int y);*

This API will request the windows created to have an initial position. The arguments x, y indicate the location of a corner of the window, relative to the entire display.

1.4.10 **glLoadIdentity() :**

*void **glLoadIdentity**(void);*

It replaces the current matrix with the identity matrix.

1.4.11 **glutInitWindowSize() :** *void*

***glutInitWindowSize**(int width, int height);*

The API requests windows created to have an initial size. The arguments width and height indicate the window's size (in pixels). The initial window size and position are hints and may be overridden by other requests.

1.4.12 **glutInitDisplayMode** *void*

__glutInitDisplayMode__(unsigned int mode);

Specifies the display mode, normally the bitwise OR-ing of GLUT display mode bit *masks*. This API specifies a display mode (such as RGBA or color-index, or single or double-buffered) for windows.

1.4.13 **glFlush() :**

void __glFlush__(void);

The glFlush function forces execution of OpenGL functions in finite time.

1.4.14 **glutCreateWindow() :**

*int __glutCreateWindow__(char *name);*

The parameter *name* specifies any name for window and is enclosed in double quotes. This opens a window with the set characteristics like display mode, width, height, and so on. The string name will appear in the title bar of the window system. The value returned is a unique integer identifier for the window. This identifier can be used for controlling and rendering to multiple windows from the same application.

1.4.15 **glutDisplayFunc() :**

*void __glutDisplayFunc__(void (*func)(void))*

Specifies the new display callback function. The API specifies the function that's called whenever the contents of the window need to be redrawn. All the routines need to be redraw the scene are put in display callback function.

1.4.16 **glVertex2f** *void **glVertex2f**(GLfloatx,GLfloat y);*   *x*   Specifies the x-coordinate of a vertex.

y   Specifies the y-coordinate of a vertex.

The glVertex function commands are used within glBegin/glEnd pairs to specify point, line, and polygon vertices. The current color, normal, and texture coordinates are associated with the vertex when glVertex is called. When only x and y are specified, z defaults to 0.0 and w defaults to 1.0. When x, y, and z are specified, w defaults to 1.0.

1.4.17 **glColor3f** *void glColor3f(GLfloat red, GLfloat green, GLfloat blue);*

PARAMETERS:

1. Red: The new red value for the current color.
2. Green: The new green value for the current color.
3. Blue: The new blue value for the current color.

Sets the current color.

1.4.18 **glRotate**():

void glRotate(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);

PARAMETERS:

angle: The angle of rotation, in degrees.

x: The x coordinate of a vector. y: The y

coordinate of a vector. z: The z

coordinate of a vector.

The glRotated and glRotatef functions multiply the current matrix by a rotation matrix.

1.4.19 **gluPerspective():** *void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar);*

PARAMETERS: fovy : Specifies the field of view angle, in degrees, in the y direction. aspect: Specifies the aspect ratio that determines the field of view in the x direction.

The aspect ratio is the ratio of x (width) to y (height).

zNear: Specifies the distance from the viewer to the near clipping plane (always positive). zFar : Specifies the distance from the viewer to the far clipping plane (always positive).
Sets up a perspective projection matrix.

1.4.20 **glMaterialfv():** *void glMaterialfv(GLenum face, GLenum pname, const GLfloat params);*

PARAMETERS:

face : The face or faces that are being updated. Must be one of the following: GL_FRONT, GL_BACK, or GL_FRONT and GL_BACK.

Pname: The material parameter of the face or faces being updated.
The parameters that can be specified using glMaterialfv, and their interpretations by the lighting equation, are as follows.
GL_SPECULAR: The params parameter contains four integer or floating-point values that specify the seculars RGBA reflectance of the material. Integer values are mapped linearly such that the most positive represent able value maps to 1.0, and the most negative represent able value maps to

-1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The default specular reflectance for both front-facing and back-facing materials is (0.0, 0.0, 0.0, 1.0).

The glMaterialfv function specifies material parameters for the lighting model.

1.4.21 **glutInit():**

*glutInit(int *argcp, char **argv);*

PARAMETERS:

argcp : A pointer to the program's unmodified argc variable from main. Upon return, the value pointed to by argcp will be updated, because glutInit extracts any command line options intended for the GLUT library.

argv : The program's unmodified argv variable from main. Like argcp, the data for argv will be updated because glutInit extracts any command line options understood by the GLUT library.

glutInit(&argc,argv); glutInit is used

to initialize the GLUT library.

1.4.22 **glutMainLoop ():** void *glutMainLoop(void);*

glutMainLoop(); glutMainLoop enters the GLUT event processing loop.

1.4.23 **glLightfv():**

*void glLightfv(GLenum light, GLenum pname, GLfloat *params);*

The glLightfv function returns light source parameter values.

PARAMETERS:

Light: The identifier of a light. The number of possible lights depends on the implementation, but at least eight lights are supported. They are identified by symbolic names of the form GL_LIGHTi where i is a value: 0 to GL_MAX_LIGHTS - 1.

Pname: A light source parameter for light. The following symbolic names are accepted:

GL_DIFFUSE: The params parameter contains four integer or floating-point values that specify the diffuse RGBA intensity of the light. Integer values are mapped linearly such that the most positive represent able value maps to 1.0, and the most negative represent able value maps to 1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The default diffuse intensity is (0.0, 0.0, 0.0, 1.0) for all lights other than light zero. The default diffuse intensity of light zero is (1.0, 1.0, 1.0, 1.0).
GL_SPECULAR: The params parameter contains four integer or floating-point values that specify the specular RGBA intensity of the light. Integer values are mapped linearly such that the most positive represent able value maps to 1.0, and the most negative represent able value maps to 1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The default specular intensity is (0.0, 0.0, 0.0, 1.0) for all lights other than light zero. The default specular intensity of light zero is (1.0, 1.0, 1.0, 1.0).
GL_AMBIENT: The params contains four integer or floating-point values that specify the ambient RGBA intensity of the light. Integer values are mapped linearly such that the most positive represent able value maps to 1.0, and the most negative representable value maps to -1.0 . Floatingpoint values are mapped directly. Neither integer nor floating-point values are clamped. The initial ambient light intensity is (0,0, 0, 1).
glLightfv(GL_LIGHT1,GL_POSITION,pos);

1.4.24 **glEnable():**

void glEnable(GLenum cap);

glEnable(GL_CULL_FACE); PARAMETERS:

cap: A symbolic constant indicating an OpenGL capability.

Chapter-2

REQUIREMENTS SPECIFICATION

2.1 Hardware requirements:

- ❖ Pentium or higher processor.
- ❖ 128 MB or more RAM.
- ❖ A standard keyboard, and Microsoft compatible mouse
- ❖ VGA monitor.
- ❖ Hard Disk Space: 4.0 GB

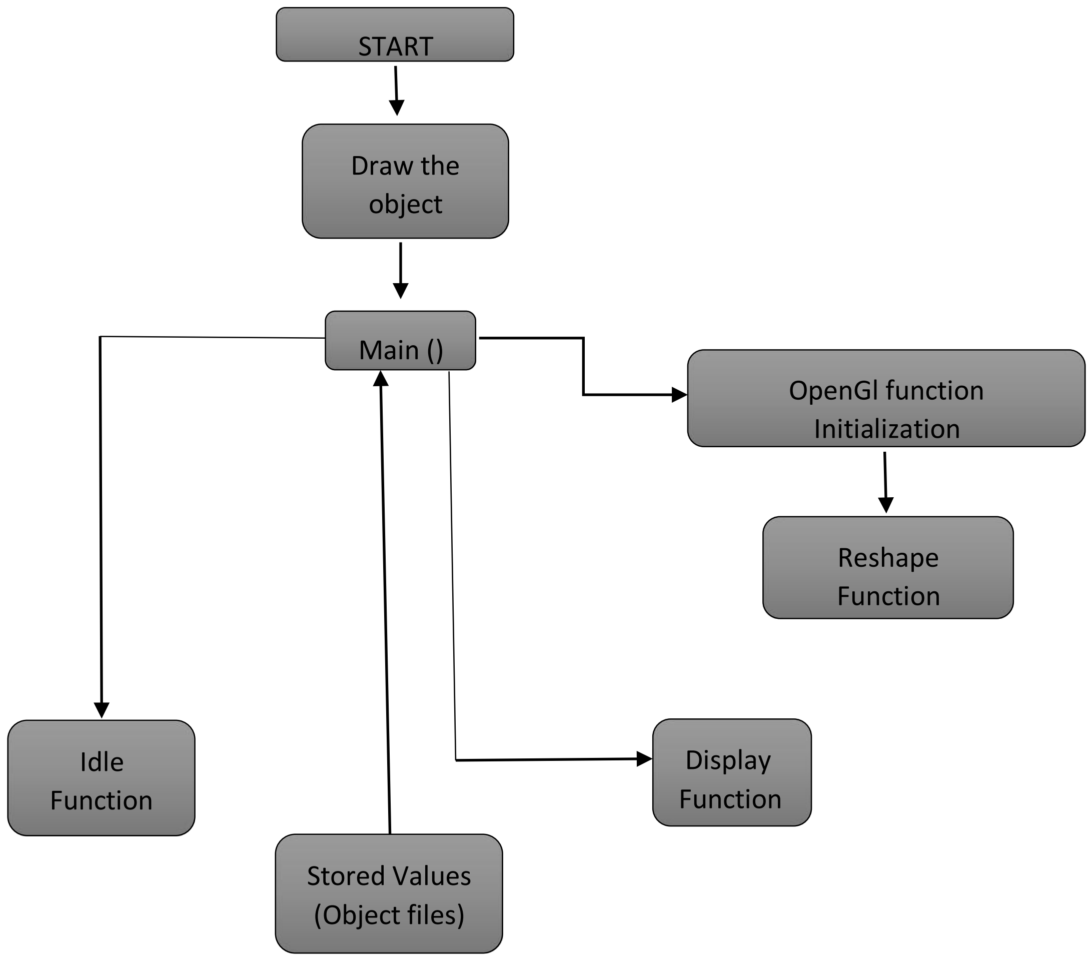
2.2 Software requirements:

- ❖ The graphics package has been designed for OpenGL; hence the machine must have Dev C++.
- ❖ Software installed preferably 6.0 or later versions with mouse driver installed.
- ❖ GLUT libraries, Glut utility toolkit must be available.
- ❖ Operating System**:** Windows
- ❖ Version of Operating System**:** Windows XP, Windows NT and Higher
- ❖ The package is implemented using Microsoft visual C++, under the windows platform. OpenGL and associated toolkits are used for the package development.
- ❖ OpenGL , a software interface for graphics hardware with built in graphics libraries like glut and glut32, and header files like glut.h.

Chapter-3

INTERFACE AND ARCHITECTURE

3.1 Flow Diagram:



The flow diagram of 3D model of Fort

OpenGL is a software tool for developing the graphics objects. OpenGL library called GLUT i.e. Graphics Library Utility toolkit supports graphics system with the necessary modelling and rendering techniques. The Lighting system is a technique for displaying graphic objects on the monitor and displaying the light effects. It provides the following functionalities.

3.2 Initialization

This function is the initial stage of the system where the system initializes the various aspects of the graphics system based on the user requirements, which include Command line processing, window system initialization and also the initial window creation state is controlled by these routines.

3.3 Event Processing

This routine enters GLUT's event processing loop. This routine never returns, and it continuously calls GLUT callback as and when necessary. This can be achieved with the help of the callback registration functions. These routines register callbacks to be called by the GLUT event processing loop.

Chapter-4

IMPLEMENTATION

Source Code

```
#include<windows.h>
#include<GL/glut.h>
void wall()
{
glPushMatrix();
glutSolidCube(1.0);
glPopMatrix();
}
void walls()
{
glPushMatrix();
glutSolidCube(2.0);
glPopMatrix();
}
void pillars(int time)
{
   if(time==0)
   {
   glPushMatrix();
   glScaled(0.2,0.6,0.1);
   glTranslated(0.0,0.025,0.0);
   wall();
   glPopMatrix();
```

```
glPushMatrix();
glRotated(90,0.0,1.0,0.0);
glScaled(0.2,0.6,0.1);
glTranslated(0.0,0.025,0.0);
wall();
glPopMatrix();
glPushMatrix();
glRotated(45,0.0,1.0,0.0);
glScaled(0.2,0.6,0.1);
glTranslated(0.0,0.025,0.0);
wall();
glPopMatrix();
glPushMatrix();
glRotated(-45,0.0,1.0,0.0);
glScaled(0.2,0.6,0.1);
glTranslated(0.0,0.025,0.0);
wall();
glPopMatrix();
glPushMatrix();
glRotated(90,1.0,0.0,0.0);
glTranslated(0.0,0.0,-0.325);
glutSolidCone(0.2,0.4,50,50);
glPopMatrix();
}
else if(time==1)
{
   glPushMatrix();
   glScaled(0.8,0.7,0.8);
   glTranslated(0.0,-0.15,0.0);
```

```
glPushMatrix();
glScaled(0.2,0.6,0.1);
glTranslated(0.0,0.025,0.0);
wall();
glPopMatrix();
glPushMatrix();
glRotated(90,0.0,1.0,0.0);
glScaled(0.2,0.6,0.1);
glTranslated(0.0,0.025,0.0);
wall();
glPopMatrix();
glPushMatrix();
glRotated(45,0.0,1.0,0.0);
glScaled(0.2,0.6,0.1);
glTranslated(0.0,0.025,0.0);
wall();
glPopMatrix();
glPushMatrix();
glRotated(-45,0.0,1.0,0.0);
glScaled(0.2,0.6,0.1);
glTranslated(0.0,0.025,0.0);
wall();
glPopMatrix();
glPushMatrix();
glRotated(90,1.0,0.0,0.0);
glTranslated(0.0,0.0,-0.325);
glutSolidCone(0.2,0.4,50,50);
glPopMatrix();
glPopMatrix();
```

```
}
else
{
    glPushMatrix();
    glScaled(0.7,0.5,0.7);
    glTranslated(0.0,-0.25,0.0);
    glPushMatrix();
    glScaled(0.2,0.6,0.1);
    glTranslated(0.0,0.025,0.0);
    wall();
    glPopMatrix();
    glPushMatrix();
    glRotated(90,0.0,1.0,0.0);
    glScaled(0.2,0.6,0.1);
    glTranslated(0.0,0.025,0.0);
    wall();
    glPopMatrix();
    glPushMatrix();
    glRotated(45,0.0,1.0,0.0);
    glScaled(0.2,0.6,0.1);
    glTranslated(0.0,0.025,0.0);
    wall();
    glPopMatrix();
    glPushMatrix();
    glRotated(-45,0.0,1.0,0.0);
    glScaled(0.2,0.6,0.1);
    glTranslated(0.0,0.025,0.0);
    wall();
    glPopMatrix();
```

```
        glPushMatrix();
        glRotated(90,1.0,0.0,0.0);
        glTranslated(0.0,0.0,-0.325);
        glutSolidCone(0.2,0.4,50,50);
        glPopMatrix();
        glPopMatrix();
        glPushMatrix();
        glRotated(-90,1.0,0.0,0.0);
        glTranslated(0.0,0.0,0.0);
        glutSolidCone(0.1,0.1,50,50);
        glPopMatrix();
        glPushMatrix();
        glTranslated(0.0,0.2,0.0);
        glutSolidSphere(0.12,50,50);
        glPopMatrix();
    }
}
void allPillars(int time)
{
        float dist;
    dist=2.0*0.75/2.0-0.2/2.0;
    glPushMatrix();
    glTranslated(dist,0,dist);
    pillars(time);
    glTranslated(0,0,-2*dist);
    pillars(time);
    glTranslated(-2*dist,0,2*dist);
    pillars(time);
    glTranslated(0,0,-2*dist);
    pillars(time);
    glPopMatrix();


}
void wallsabove()
{
glPushMatrix();
glColor3f(1.0,0.0,0.0);
glTranslated(0.0,0.8,0.65);
glRotated(90,1.0,0.0,0.0);
glScaled(1.4,0.03,0.3);
wall();
glPopMatrix();
}
void wallbelow()
{
    glPushMatrix();
    glColor3f(0.576,0.8588,0.439);
    glTranslated(0.45,0.06,0.65);
    glScaled(0.4,0.7,0.05);
    wall();
    glPopMatrix();
    glPushMatrix();
    glColor3f(0.576,0.8588,0.439);
    glTranslated(-0.45,0.06,0.65);
    glScaled(0.4,0.7,0.05);
    wall();
    glPopMatrix();
}
```

```
void fort()
{



glPushMatrix();
glColor3f(0.576,0.8588,0.439);
glScaled(0.7,0.9,0.7);
glTranslated(0.0,0.0,0.0);





glPushMatrix();
glTranslated(0.0,-0.3,0.0);
glScaled(2.0,0.05,2.0);
wall();
glPopMatrix();
allPillars(0);
glPushMatrix();
glTranslated(0.0,0.600,0.0);
allPillars(1);
glPopMatrix();
glPushMatrix();
glTranslated(0.0,0.975,0.0);
allPillars(2);
glPopMatrix();


glPushMatrix();
```

```
glTranslated(0.0,0.555,0.0);
glScaled(1.4,0.3,1.4);
wall();
glPopMatrix();
glPushMatrix();
glTranslated(0.0,0.720,0.0);
glScaled(1.6,0.03,1.6);
wall();
glPopMatrix();
wallsabove();
glPushMatrix();
glRotated(90,0.0,1.0,0.0);
wallsabove();
glRotated(90,0.0,1.0,0.0);
wallsabove();
glRotated(90,0.0,1.0,0.0);
wallsabove();
glPopMatrix();
wallbelow();
glPushMatrix();
glRotated(90,0.0,1.0,0.0);
wallbelow();
glRotated(90,0.0,1.0,0.0);
wallbelow();
glRotated(90,0.0,1.0,0.0);
wallbelow();
glPopMatrix();
glPopMatrix();


}
static GLfloat theta[] = {0.0,0.0,0.0};
 static GLint axis = 2;
void displaySolid()
{
GLfloat mat_ambient[]={0.7f,0.7f,0.7f,1.0f};
GLfloat mat_diffuse[]={0.5f,0.5f,0.5f,1.0f};
GLfloat mat_specular[]={1.0f,1.0f,1.0f,1.0f};
GLfloat mat_shininess[]={50.0f};
glMaterialfv(GL_FRONT,GL_AMBIENT,mat_ambient);
glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);
glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
glMaterialfv(GL_FRONT,GL_SHININESS,mat_shininess);
GLfloat lightIntensity[]={0.7f,0.7f,0.7f,1.0f};
GLfloat light_Position[]={2.0f,6.0f,3.0f,0.0f};
glLightfv(GL_LIGHT0,GL_POSITION,light_Position);
glLightfv(GL_LIGHT0,GL_DIFFUSE,lightIntensity);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
double winHt =1.0;
glOrtho(-winHt*1300/700 ,winHt*1300/700,-winHt,winHt,0.1,100.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(2.3,1.3,2.0,0.0,0.25,0.0,0.0,1.0,0.0);
glRotatef(theta[0], 1.0, 0.0, 0.0);
  glRotatef(theta[1], 0.0, 1.0, 0.0);
  glRotatef(theta[2], 0.0, 0.0, 1.0);
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);


glPushMatrix();
glColor3f(0.576,0.8588,0.439);
glScaled(0.5,0.7,0.5);
glTranslated(0.0,0.0,0.0);




glPushMatrix();


glTranslated(0.0,-0.3,0.0);
glScaled(1.6,0.01,1.6);
walls();
glPopMatrix();
allPillars(0);
glPushMatrix();
glTranslated(0.0,0.600,0.0);
allPillars(1);
glPopMatrix();
glPushMatrix();
glTranslated(0.0,0.975,0.0);
allPillars(2);
glPopMatrix();



glPushMatrix();
glTranslated(0.0,0.555,0.0);
```

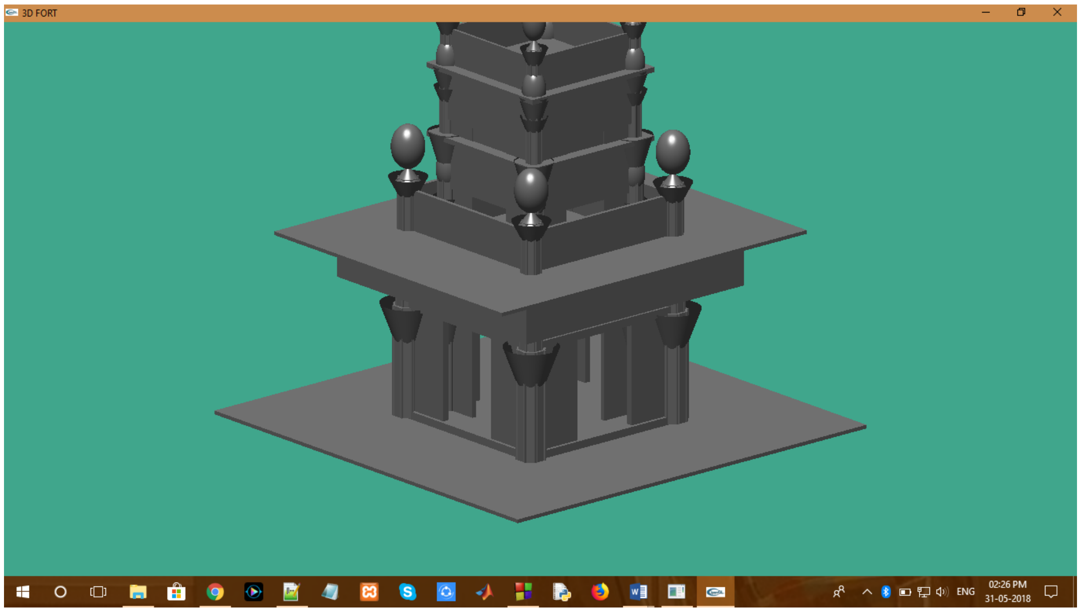
```
glScaled(1.0,0.099,1.0);
walls();
glPopMatrix();
glPushMatrix();
glTranslated(0.0,0.720,0.0);
glScaled(1.2,0.0099,1.4);
glPopMatrix();
wallsabove();
wallsabove();
glRotated(90,0.0,1.0,0.0);
wallsabove();
glPopMatrix();
wallbelow();
glPushMatrix();
glRotated(90,0.0,1.0,0.0);
wallbelow();
glRotated(90,0.0,1.0,0.0);
wallbelow();
glRotated(90,0.0,1.0,0.0);
wallbelow();
glPopMatrix();


glPopMatrix();
```

```
glFlush();
}
void mouse(int btn, int state, int x, int y) {
    if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
 if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
  if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
   theta[axis] += 2.0;
    if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
    displaySolid();
    }
void myReshape(int w, int h) {
     glViewport(0, 0, w, h);

/* Use a perspective view */

 glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
   if(w<=h) glFrustum(-2.0, 2.0, -2.0 * (GLfloat) h/ (GLfloat) w,2.0* (GLfloat) h / (GLfloat) w, 2.0, 20.0);
   else glFrustum(-2.0, 2.0, -2.0 * (GLfloat) w/ (GLfloat) h,2.0* (GLfloat) w / (GLfloat) h, 2.0, 20.0);

 glMatrixMode(GL_MODELVIEW); }

int main(int argc,char** argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH|GL_COLOR_BUFFER_BIT);
glutInitWindowSize(700,500);
```

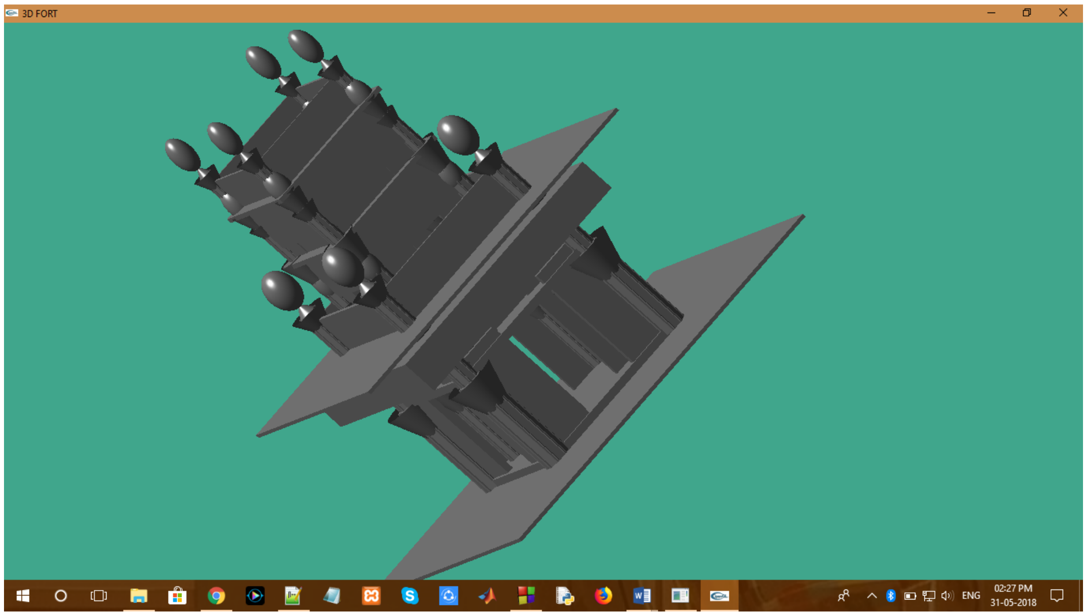
```
glutInitWindowPosition(100,100);
glutCreateWindow("3D FORT");
glutReshapeFunc(myReshape);
glutDisplayFunc(displaySolid);
glutMouseFunc(mouse);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glShadeModel(GL_SMOOTH);
glEnable(GL_DEPTH_TEST);
glEnable(GL_NORMALIZE);
glClearColor(0.25,0.65,0.55,0.0);
glViewport(0,0,1300,700);
glutMainLoop();
return 0;
}
```

Chapter-5

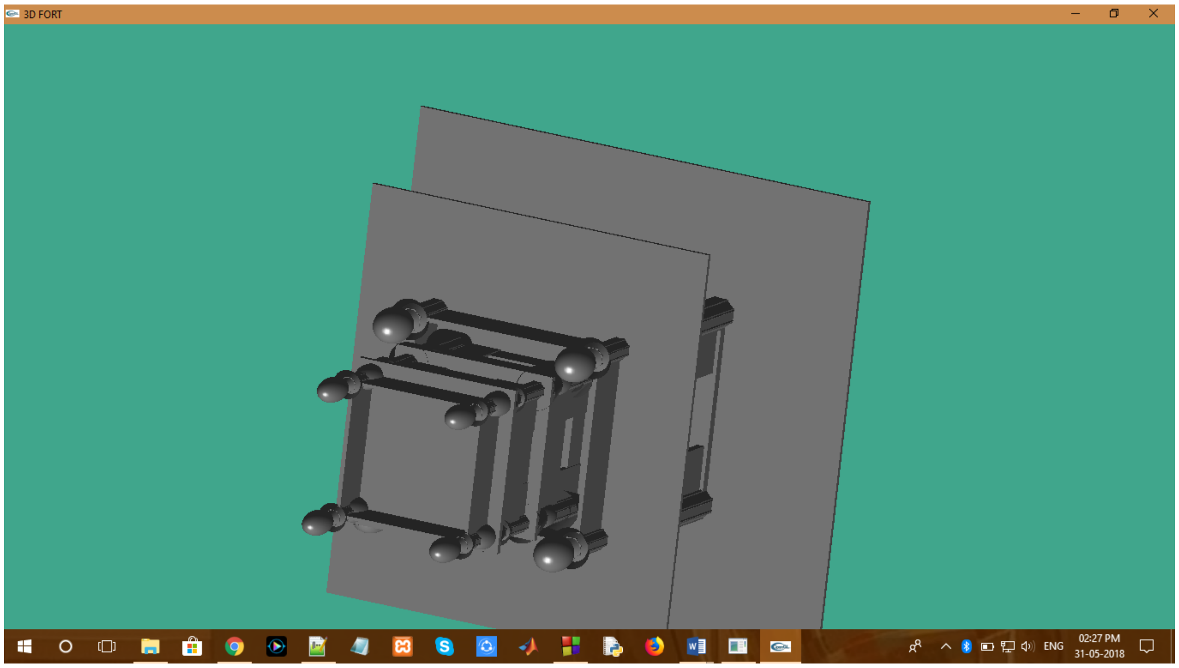
SNAPSHOTS



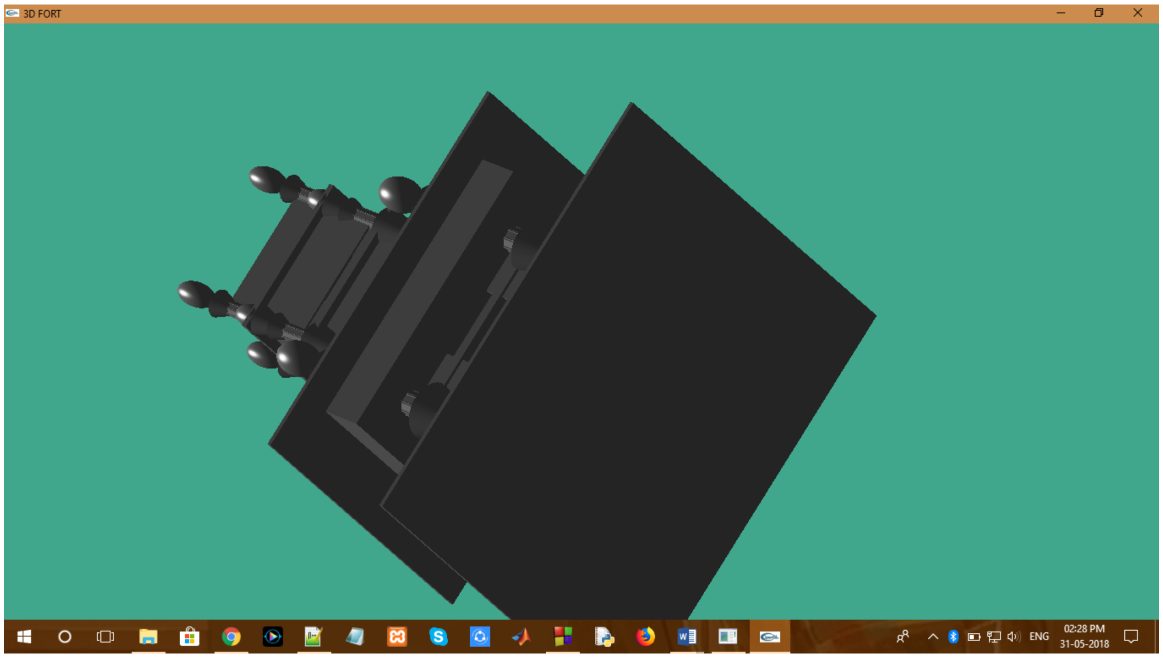
Snapshot1:Front View of Buddhist Temple



Snapshot2: Side View of Buddhist Temple



Snapshot3: Top View of buddhist temple



Snapshot4: Bottom View of Buddhist Temple

Chapter-6

FUTURE ENHANCEMENT

- We can try to add actual color to the object drawn in this program.
- We can try to improve the quality of the object drawn.

Chapter-7

CONCLUSION

The described project demonstrates the power of Viewing which is implemented using different modes of viewing. The lighting and material functions of OpenGl library add effect to the objects in animation.

The aim in developing this program was to design a simple program using Open GL application software by applying the skills we learnt in class, and in doing so, to understand the algorithms and the techniques underlying interactive graphics better.

The designed program will incorporate all the basic properties that a simple program must possess.The program is user friendly as the only skill required in executing this program is the knowledge of graphics.

Chapter-8

REFERENCES

Books:

[1] The Red Book –OpenGL Programming Guide,6th edition.

[2] Rost , Randi J. : OpenGL Shading Language, Addison-Wesley

[3] Interactive Computer Graphics-A Top Down Approach Using OpenGL, Edward Angel, Pearson-5th edition.