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 low-level, widely supported modeling and rendering software package, available on all platforms. It can be used in a range of graphics applications, such as games, CAD design, modeling.

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 modeling 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. The OpenGL

Visualization Programming Pipeline:

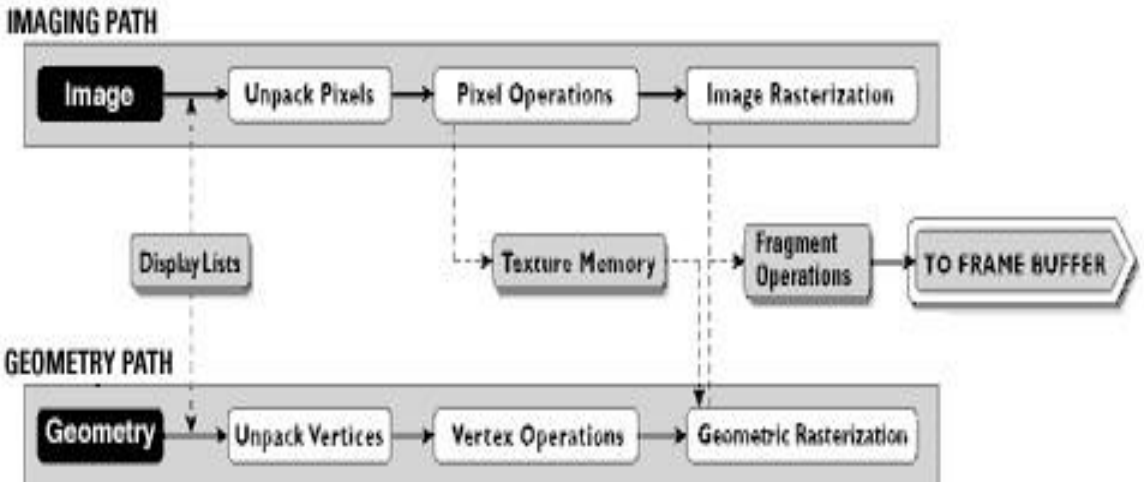


Figure. 1.1: OpenGl Visualization Programming Pipeline

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 texture mapped 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:

This project aims at creating a 3D shapes in the space using the polygon primitives in OpenGl and glut libraries. The viewer is presented with simple keyboard buttons to navigate within the boundary. On choosing a particular point, user can move the cube in any 3 dimension. User has to select four points on space .User can then select which color has to be implemented for each of these shapes. Once the shape is completed user can move to any shape irrespective of the previous shape. Appropriate texture mapping is incorporated to each of these planes.

We make use of OpenGl and glut library 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)*

glPushMatrixpushes the current matrix stack down by one level, duplicating the current matrix.

1.4.4 **glPopMatrix() :**

*void **glPopMatrix(**void)*

glPopMatrixpops 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**(GLdoublex, GLdoubley,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 **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.

1.4.17 **glRotate**():

voidglRotate(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.

1.4.18 **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.

1.4.19 **glutMainLoop():**

void glutMainLoop(void);

glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

1.4.20 **glEnable():**

void glEnable(GLenum cap);

glEnable(GL_CULL_FACE);

PARAMETERS: cap: A symbolic constant indicating an OpenGL capability.

The glEnable enable OpenGL capabilities.

Chapter-2

LITERATURE SURVEY

People use the term "computer graphics" to mean different things in different context. Computer graphics are pictures that are generated by a computer. Everywhere you look today, you can find examples, especially in magazines and on television. Some images look so natural you can't distinguish them from photographs of a real scene. Others have an artificial look, intended to achieve some visual effects.

There are several ways in which the graphics generated by the program can be delivered.

- Frame- by- frame: A single frame can be drawn while the user waits.
- Frame-by-frame under control of the user: A sequence of frames can be drawn, as in a corporate power point presentation; the user presses to move on to the next slide, but otherwise has no way of interacting with the slides
- Animation: A sequence of frames proceeds at a particular rate while the user watches with delight.
- Interactive program: An interactive graphics presentation is watched, where the user controls the flow from one frame to another using an input device such as a mouse or keyboard, in a manner that was unpredictable at the time the program was written. This can delight the eye.

2.1 History

OpenGL was developed by **'Silicon Graphics Inc'** (SGI) on 1992 and is popular in the gaming industry where it competes with the Direct3D in the Microsoft Windows platform. OpenGL is broadly used in CAD (Computer Aided Design), virtual reality, scientific visualization, information visualization, flight simulation and video games development.

OpenGL is a standard specification that defines an API that is multi-language and multi-platform and that enables the codification of applications that output computerized graphics in 2D and 3D.

The interface consists in more than 250 different functions, which can be used to draw complex tridimensional scenes with simple primitives. It consists of many functions that help to create a real world object and a particular existence for an object can be given.

2.2 Characteristics

- OpenGL is a better documented API.
- OpenGL is also a cleaner API and much easier to learn and program.
- OpenGL has the best demonstrated 3D performance for any API.
- Microsoft's Direct3D group is already planning a major API change called Direct Primitive that will leave any existing investment in learning Direct3D immediate mode largely obsolete.

2.3 Computer Graphics Library Organization

OpenGL stands for Open Source Graphics Library. Graphics Library is a collection of APIs (Application Programming Interfaces).

Graphics Library functions are divided in three libraries. They are as follows-

i. GL Library (OpenGL in Windows)
ii. GLU (OpenGL Utility Library)
iii. GLUT (OpenGL Utility Toolkit)

Functions in main GL library name function names that begin with the letter ‘gl’.

- GLU library uses only GL functions but contains code for creating objects and simplify viewing.
- To interface with the window system and to get input from external devices GLUT library is used, which is a combination of three libraries GLX for X windows, ‘wgl’ for Windows and ‘agl’ for Macintosh.
- These libraries are included in the application program using preprocessor directives. E.g.: #include<GL/glut.h>
- The following figure shows the library organization in OpenGL.

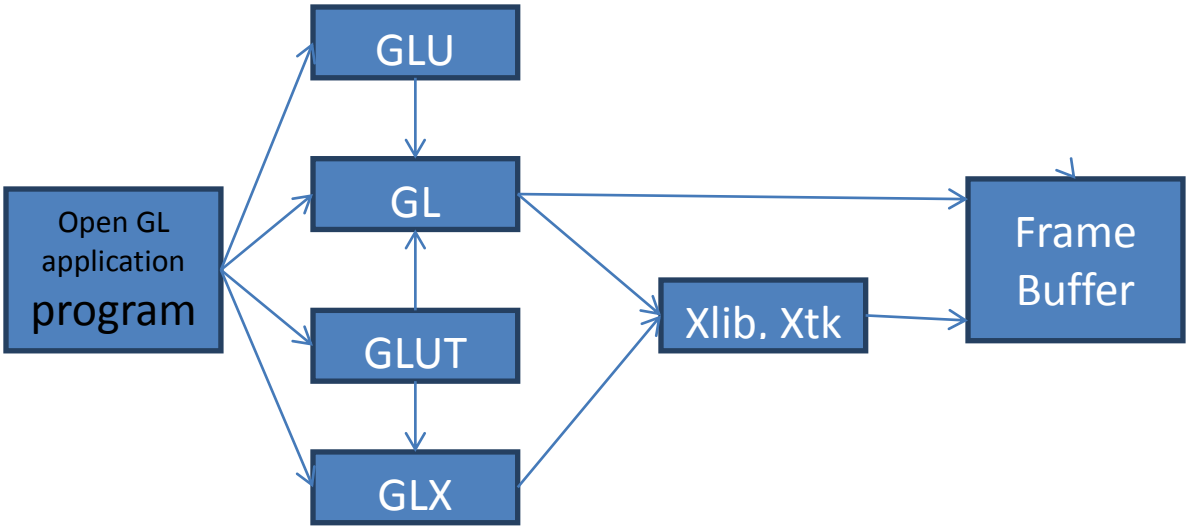


Fig 2.1 Library Organization

2.4 Graphics System and Functions

- Graphics system and functions can be considered as a black box, a term used to denote a system whose properties are only described by its inputs and output without knowing the internal working.
- Inputs to graphics system are functions calls from application program, measurements from input devices such as mouse and keyboard.
- Outputs are primarily the graphics sent to output devices.

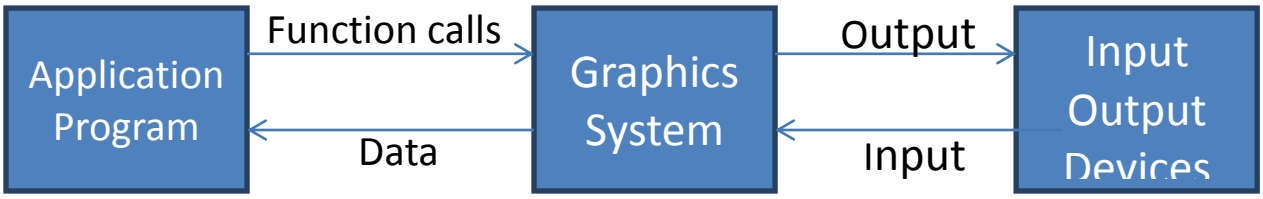


Fig 2.2 Graphics System as a Black Box

API's are described through functions in its library. These functions are divided into seven major groups.

1) Primitive Functions:

   Primitive functions define the low level objects or atomic entities that a system can display, the primitives include line segments, polygons, pixels, points, text and various types of curves and surfaces.

2) Attribute Functions:

   Attribute Functions allow us to perform operations ranging from choosing the color to display a line segment, to packing a pattern to fill inside any solid figure.

3) Viewing Functions:

   Viewing functions allow us to specify various views.

4) Transformation Functions:

   Transformation functions allow us to carry out transformation of objects such as rotation, translation and scaling.

5) Input Functions:

   Input functions allow us to deal with the diverse forms of input that characterize modern graphics system. It deals with devices such as keyboard, mouse and data tablets.

6) Control Functions:

   Control Functions enable us to communicate with the window system, to initialize the programs, and to deal with any errors that occur during the execution of the program.

7) Query Functions:

   Query Functions provides information about the API.

Chapter-3

REQUIREMENTS SPECIFICATION

3.1 Hardware requirements:

- Minimum of 2GB of main memory
- Minimum of 3GB of storage
- Mouse
- Display Unit
- Dual-Core or AMD with minimum of 1.5GHz speed

3.2 Software requirements:

- Windows – XP/7/8/8.1/10
- OpenGL Files
- DirectX 8.0 and above versions

Header Files

- glut.h

Object File Libraries

- glut32.lib

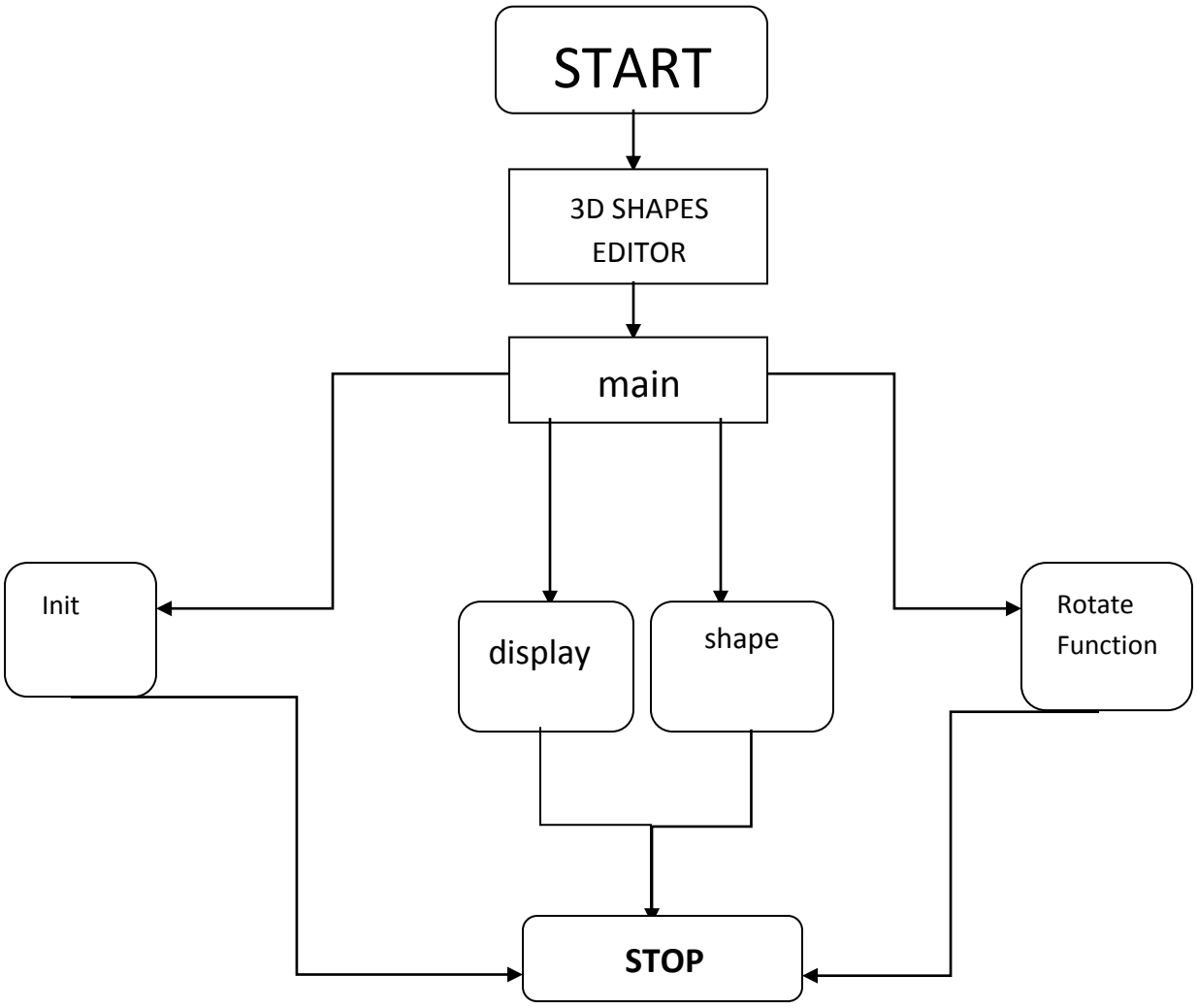
DLL files

- glut32.dll

Chapter 4

INTERFACE AND ARCHITECTURE

4.1 Flow Diagram



The flow diagram of 3D SHAPES EDITOR

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.

4.2 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.

4.3 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 5

IMPLEMENTATION

5.1 The code for building the editor:

```
#include<windows.h>
#include<iostream>
#include<GL/glut.h>
using namespace std;
int cx=0,cy=0,cz=0;
int cn;
static GLfloat theta[]={0.0,0.0,0.0};
static GLint axis=2;
static GLdouble viewer[]={0.0,7.0,7.0};
GLfloat colors[] = {0.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,0.0, 0.0,1.0,0.0,
0.0,0.0,1.0, 1.0,0.0,1.0, 1.0,1.0,1.0, 0.0,1.0,1.0};

GLfloat vertices[] = {-1.0,-1.0,-1.0,1.0,-1.0,-1.0,1.0,1.0,-1.0, -
1.0,1.0,-1.0, -1.0,-1.0,1.0,1.0,-1.0,1.0, 1.0,1.0,1.0, -1.0,1.0,1.0};



struct Quads
{
    int x1,y1,z1,x2,y2,z2,x3,z3,y3,x4,y4,z4;
    float r,g,b;
    int state;
    int total;
};Quads Q[100];

void addQuad()
{
    Q[0].state++;if(Q[0].state>4){Q[0].state=1;}
    int st=Q[0].state;

    if(st==1){Q[0].total++;cn=Q[0].total;}
    if(st==1){Q[cn].x1=cx; Q[cn].y1=cy; Q[cn].z1=cz;}
    if(st==1||st==2){Q[cn].x2=cx; Q[cn].y2=cy; Q[cn].z2=cz;}
    if(st==1||st==2||st==3){Q[cn].x3=cx; Q[cn].y3=cy; Q[cn].z3=cz;}
    if(st==1||st==2||st==3||st==4){Q[cn].x4=cx; Q[cn].y4=cy; Q[cn].z4=cz;}
}

void drawQuads()
{
    int i;
    for(i=1;i<Q[0].total+1;i++)
    {

        glBegin(GL_QUADS);
        glColor3f(Q[i].r,Q[i].g,Q[i].b);
        glVertex3f(Q[i].x1,Q[i].y1,Q[i].z1);
```

```
        glVertex3f(Q[i].x2,Q[i].y2,Q[i].z2);
        glVertex3f(Q[i].x3,Q[i].y3,Q[i].z3);
        glVertex3f(Q[i].x4,Q[i].y4,Q[i].z4);
        glEnd();
    }


}
void basicsquare()
{
    glPushMatrix();
    glColor3f(1,1,1);
    glTranslatef(cx,cy,cz);
glutSolidCube(0.4);
glPopMatrix();
}


void drawGrid()
{
    int i;
    for(i=0;i<40;i++)
    {
        glPushMatrix();
        if(i<20)
            {
                glTranslatef(0,0,i);
        }
        if(i>=20)
        {
            glTranslatef(i-20,0,0);
            glRotatef(-90,0,1,0);
        }

        glBegin(GL_LINES);
        glColor3f(1,0,0);glLineWidth(1);
        glVertex3f(0,-0.1,0);glVertex3f(19,-0.1,0);
        glEnd();
        glPopMatrix();
    }
}
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;
glutPostRedisplay();
}
```

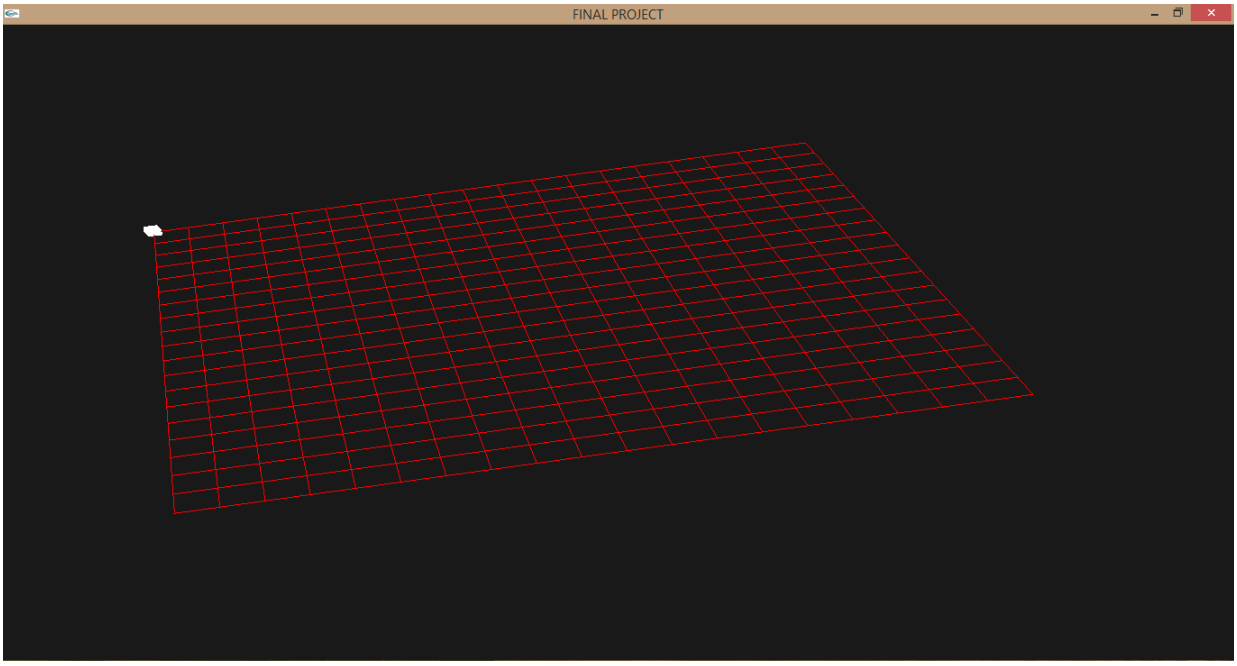
```
void display(void)
{
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 glTranslatef(-13,0,-45);
glRotatef(40,1,1,0);

//drawGrid();

//DrawGrid(20);
// Code for cube rotate
gluLookAt(viewer[0],viewer[1],viewer[2],0.0,0.0,0.0,0.0,1.0,0.0);
        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);
        drawGrid();
        drawQuads();
        basicsquare();

//glFlush();
glutSwapBuffers();
}
void init()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(35,1.0f,0.1f,1000);
    glMatrixMode(GL_MODELVIEW);
    glEnable(GL_DEPTH_TEST);
    glClearColor(0.1,0.1,0.1,1);

}

void keyboard(unsigned char key,int x,int y)
{
    if(key=='w'){cz-=1;}if(key=='s'){cz+=1;}
    if(key=='a'){cx-=1;}if(key=='d'){cx+=1;}
    if(key=='z'){cy-=1;}if(key=='q'){cy+=1;}


    if(key==32)
        {addQuad();
        }
    if(key=='r'){Q[cn].r=1;Q[cn].g=0;Q[cn].b=0;}
    if(key=='g'){Q[cn].r=0;Q[cn].g=1;Q[cn].b=0;}
    if(key=='b'){Q[cn].r=0;Q[cn].g=0;Q[cn].b=1;}
    if(key=='y'){Q[cn].r=1;Q[cn].g=1;Q[cn].b=0;}


    glutPostRedisplay();

}
```

```
int main(int argc,char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowSize(800,600);
    glutCreateWindow("FINAL PROJECT");
    glutMouseFunc(mouse);
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    glEnableClientState(GL_COLOR_ARRAY);
    glEnableClientState(GL_VERTEX_ARRAY);
   glVertexPointer(3, GL_FLOAT, 0, vertices);
    glColorPointer(3,GL_FLOAT, 0, colors);
    glColor3f(1.0,1.0,1.0);
    init();
    glutMainLoop();
    return 0;
}
```

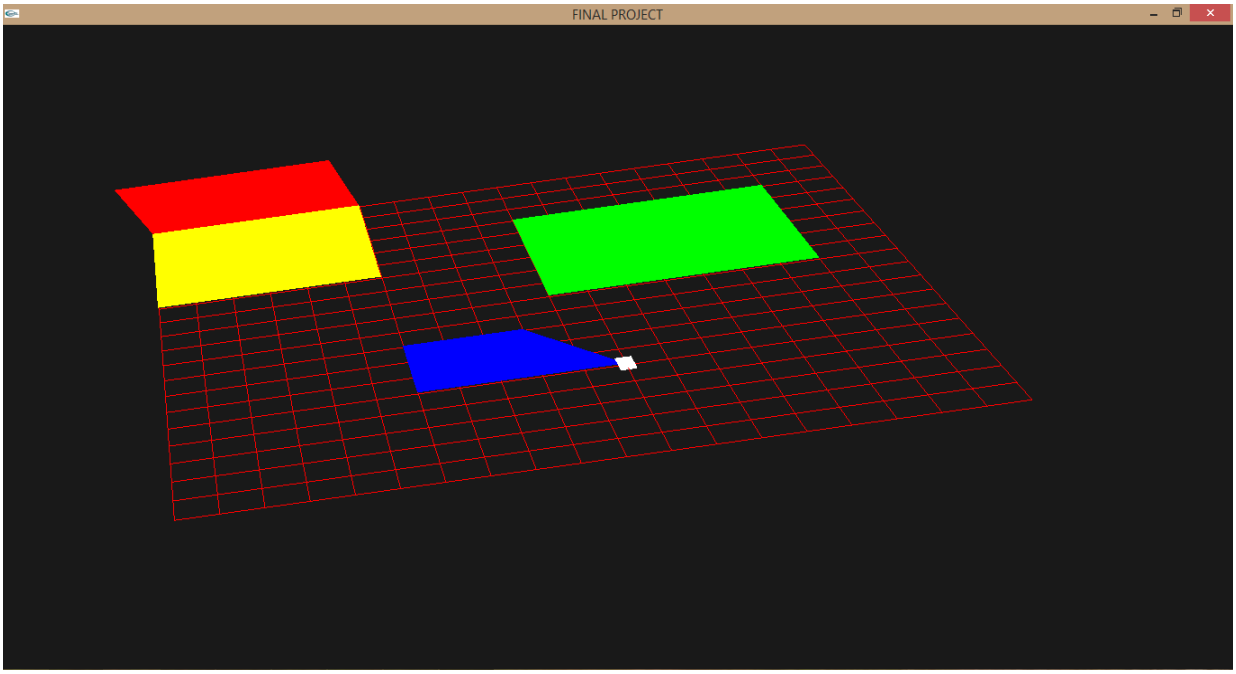
Chapter 6

SNAPSHOTS

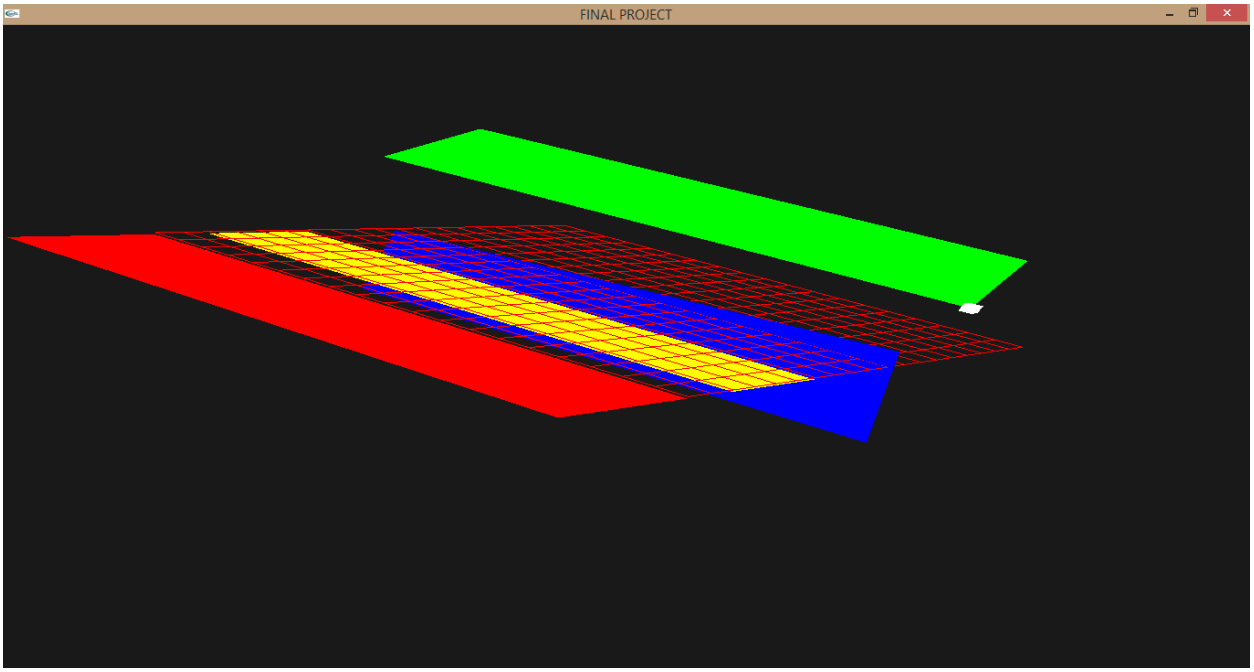
6.1 First glimpse of the project

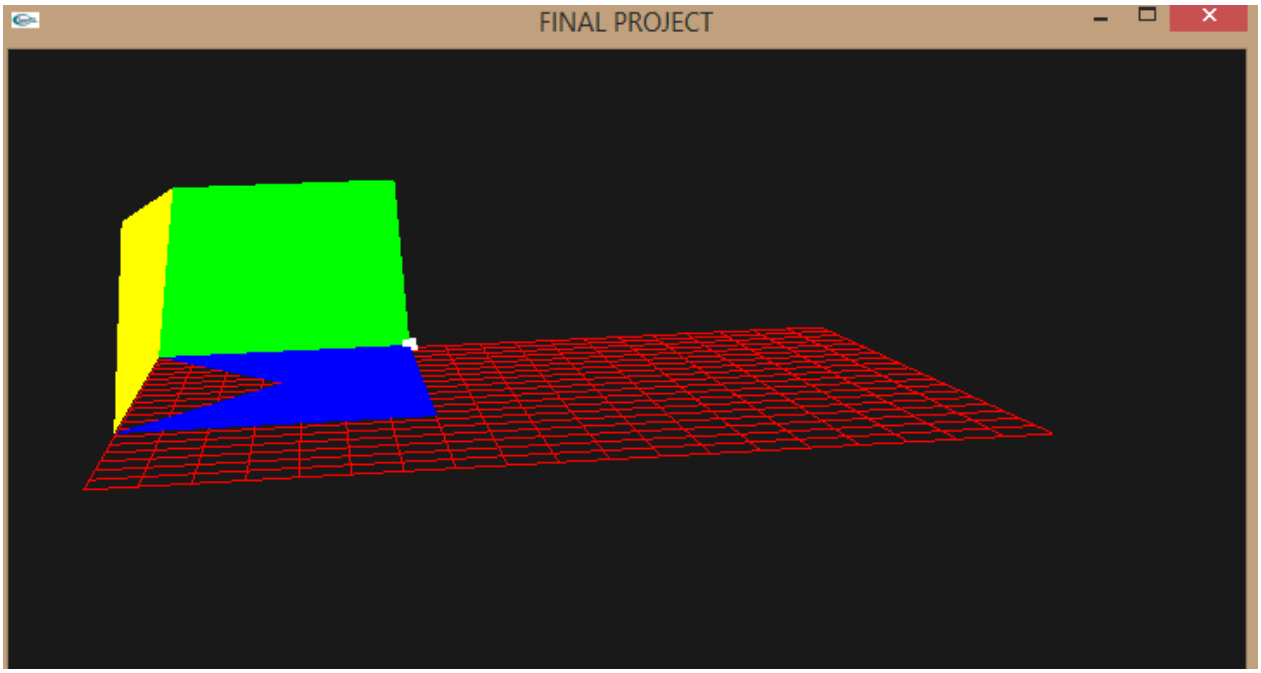


6.2 Shapes on plane



6.3 Building shapes on space





Chapter 7

CONCLUSION & FUTURE ENHANCEMENT

Our editor will be able to save the coordinate of the different shapes that was given as the input for the creation of the shapes. This may be used for creating of various models without having to memorize the coordinate values. Our project mainly aims in developing of various models prior to implementing in the project.

By user's point of view, the OpenGL software is very easy to use. Also it is the most widely used application in interactivity. The designers developed the GLUT (an OpenGl Utility Toolkit) which provides all the functionalities that is expected on virtual systems, which allows for the portability of the system to work in a variety of environments. Using which we can develop the physically based models etc.

In Future:

There are textures which can be implemented to provide a real-life, more enhanced and minutely detailed view of the various designs. We can use this in the field of architecture and construction engineering to make various models.

Chapter 8

REFERENCES

Books Referred

[1] Interactive Computer Graphics-Edward Angel

[2] Fundamentals of Computer Graphics -Peter Shirley

Sites Referred

[1] www.computergraphics.com

[2] en.wikipedia.org/wiki/OpenGL

[3] www.opengl.com

[4] www.google.com

Chapter 9

APPENDIX

8.1 User Manual:

- Mouse button is used for rotation of the plane
- The keys A and D are used for the movement of the cube in the x coordinate
- The keys Q and Z are used for the movement of the cube in the y coordinate
- The keys W and S are used for the movement of the cube in the z coordinate

8.2 Personal Details:

NAME: PRATEEK T L [1DT15CS081]
SREE VISHNU RAO C R [1DT15CS109]

SEMESTER: VI

DEPARTMENT: COMPUTER SCIENCE AND ENGINEERING

COLLEGE: DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT

EMAIL ID: tlprateektl@gmail.com
vishnurao12@gmail.com