What Is 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. Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of *geometric primitives* - points, lines, and polygons.

A sophisticated library that provides these features could certainly be built on top of OpenGL. The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadric surfaces and NURBS (Non-Uniform Rational B-Splines) curves and surfaces. GLU is a standard part of every OpenGL implementation. Also, there is a higher-level, object-oriented toolkit, Open Inventor, which is built atop OpenGL, and is available separately for many implementations of OpenGL.

OpenGL-Related Libraries

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of the windowing system. A number of libraries exist to allow you to simplify your programming tasks, including the following:

- The OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. This library is provided as part of every OpenGL implementation. GLU routines use the prefix **glu**.
- For every window system, there is a library that extends the functionality of that window system to support OpenGL rendering. For machines that use the X Window System, the OpenGL Extension to the X Window System (GLX) is provided as an adjunct to OpenGL. GLX routines use the prefix **glX**. For Microsoft Windows, the WGL routines provide the Windows to OpenGL interface. All WGL routines use the prefix **wgl**. For IBM OS/2, the PGL is the Presentation Manager to OpenGL interface, and its routines use the prefix **pgl**.
- The OpenGL Utility Toolkit (GLUT) is a window system-independent toolkit, written by Mark Kilgard, to hide the complexities of differing window system APIs. GLUT routines use the prefix **glut.**
- Open Inventor is an object-oriented toolkit based on OpenGL which provides objects and methods for creating interactive three-dimensional graphics applications. Open

Inventor, which is written in C++, provides prebuilt objects and a built-in event model for user interaction, high-level application components for creating and editing three-dimensional scenes, and the ability to print objects and exchange data in other graphics formats. Open Inventor is separate from OpenGL.

Include Files

For all OpenGL applications, you want to include the gl.h header file in every file. Almost all OpenGL applications use GLU, the aforementioned OpenGL Utility Library, which requires inclusion of the glu.h header file. So almost every OpenGL source file begins with

```
#include <GL/gl.h>
#include <GL/glu.h>
```

If you are directly accessing a window interface library to support OpenGL, such as GLX, AGL, PGL, or WGL, you must include additional header files. For example, if you are calling GLX, you may need to add these lines to your code

```
#include <X11/Xlib.h>
#include <GL/glx.h>
```

If you are using GLUT for managing your window manager tasks, you should include

```
#include <GL/glut.h>
```

Note that glut.h includes gl.h, glu.h, and glx.h automatically, so including all three files is redundant. GLUT for Microsoft Windows includes the appropriate header file to access WGL.

GLUT, the OpenGL Utility Toolkit

As you know, OpenGL contains rendering commands but is designed to be independent of any window system or operating system. Consequently, it contains no commands for opening windows or reading events from the keyboard or mouse. Unfortunately, it's impossible to write a complete graphics program without at least opening a window, and most interesting programs require a bit of user input or other services from the operating system or window system.

In addition, since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), GLUT includes several routines that create more complicated three-dimensional objects such as a sphere, a torus, and a teapot. This way, snapshots of program output can be interesting to look at. (Note that the OpenGL Utility Library, GLU, also has quadrics routines that create some of the same three-dimensional objects as GLUT, such as a sphere, cylinder, or cone.)

Important features of OpenGL Utility Toolkit (GLUT)

- Provides functionality common to all window systems.
- Open a window.
- Get input from mouse and keyboard.
- Menus.
- Event-driven.
- Code is portable, but GLUT lacks the functionality of a good toolkit for a specific platform.

- No slide bars.
- OpenGL is not object oriented. Hence, there are multiple functions for a given logical function
- glVertex3f
  - glVertex2i
  - glVertex3dv
- Underlying storage mode is the same easy to create overloaded functions in C++ but issue is efficiency.
- **OpenGL Interface**
  - GL (OpenGL in Windows)
  - GLU (graphics utility library)
    uses only GL functions, creates common objects (such as spheres)
  - GLUT (GL Utility Toolkit)
    interfaces with the window system
  - GLX: glue between OpenGL and Xwindow, used by GLUT

Window Management

Five routines perform tasks necessary to initialize a window.

- **glutInit**(int **argc*, char ***argv*) initializes GLUT and processes any command line arguments (for X, this would be options like -display and -geometry). **glutInit()** should be called before any other GLUT routine.
- **glutInitDisplayMode**(unsigned int *mode*) specifies whether to use an *RGBA* or color-index color model. You can also specify whether you want a single- or double-buffered window. (If you're working in color-index mode, you'll want to load certain colors into the color map; use **glutSetColor()** to do this.) Finally, you can use this routine to indicate that you want the window to have an associated depth, stencil, and/or accumulation buffer. For example, if you want a window with double buffering, the RGBA color model, and a depth buffer, you might call **glutInitDisplayMode**(*GLUT_DOUBLE* | *GLUT_RGB* | *GLUT_DEPTH*).
- **glutInitWindowPosition**(int *x*, int *y*) specifies the screen location for the upper-left corner of your window.
- **glutInitWindowSize**(int *width*, int *size*) specifies the size, in pixels, of your window.
- int **glutCreateWindow**(char **string*) creates a window with an OpenGL context. It returns a unique identifier for the new window. Be warned: Until **glutMainLoop()** is called (see next section), the window is not yet displayed.

The Display Callback

glutDisplayFunc(void (**func*)(void)) is the first and most important event callback function you will see. Whenever GLUT determines the contents of the window need to

be redisplayed, the callback function registered by **glutDisplayFunc()** is executed. Therefore, you should put all the routines you need to redraw the scene in the display callback function.

If your program changes the contents of the window, sometimes you will have to call **glutPostRedisplay**(void), which gives **glutMainLoop()** a nudge to call the registered display callback at its next opportunity.

Running the Program

The very last thing you must do is call **glutMainLoop**(void). All windows that have been created are now shown, and rendering to those windows is now effective. Event processing begins, and the registered display callback is triggered. Once this loop is entered, it is never exited!

Example: Simple OpenGL Program Using GLUT: hello.c

```
#include <GL/gl.h>
#include <GL/glut.h>

void display(void)
{
/*  clear all pixels  */
   glClear (GL_COLOR_BUFFER_BIT);

/*  draw white polygon (rectangle) with corners at
 *  (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0)
 */
   glColor3f (1.0, 1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f (0.25, 0.25, 0.0);
      glVertex3f (0.75, 0.25, 0.0);
      glVertex3f (0.75, 0.75, 0.0);
      glVertex3f (0.25, 0.75, 0.0);
   glEnd();

/*  don't wait!
 *  start processing buffered OpenGL routines
 */
   glFlush ();
}

void init (void)
{
/*  select clearing (background) color       */
   glClearColor (0.0, 0.0, 0.0, 0.0);
```

```
/*  initialize viewing values  */
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

/*  Declare initial window size, position, and display mode
 *  (single buffer and RGBA).  Open window with "hello"
 *  in its title bar.  Call initialization routines.
 *  Register callback function to display graphics.
 *  Enter main loop and process events.
 */

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (250, 250);
   glutInitWindowPosition (100, 100);
   glutCreateWindow ("hello");
   init ();
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;   /* ISO C requires main to return int. */
}
```

Instructions for using OpenGL and GLUT for Windows XP machines with Microsoft Visual Studio.Net 2003

If you are using OpenGL on your computer at home:

- All Windows systems since Windows 98 have OpenGL preinstalled.
- Download GLUT. Go to the CSCI 4250 pages web site (http://www.mtsu.edu/~csjudy/4250) and visit the “Links to OpenGL sites”. You will find a site for downloading GLUT.
- When you down load it, it will be a zip file. You will have to unzip it. Then place the **glut32.dll** file in the **windows\system32** directory.
- If your system is configured like the ones at MTSU, you will need to place the **glut.h** file in the **\Program Files\Microsoft Visual Studio .Net 2003\vc7\PlatformSDK\Include\ GL** folder and the **glut32.lib** file in the **Program Files\Microsoft Visual Studio .Net 2003\vc7\PlatformSDK\Lib** folder.

To use Microsoft Visual Studio.Net 2003:

- Start Microsoft Visual C++ -- click on **Start**, **Programs, Microsoft Visual Studio .Net**

2003, Microsoft Visual Studio .Net 2003

- Create a new project by selecting **File New Project… Visual C++ Projects Win32 Project.** Give your project a name (use your last name followed by **the number of the project** for the name of the project), select the appropriate disk or hard drive, and click on **OK.** On the subsequent window, select **Application Settings**. In the next window, select **Empty Project** and **Console Application** then click on **Finish.**
- To create the files that should be placed in the project, you should select **Project, Add New Item,** and then select the type of files that you are creating.
- To create an executable, compile **Build Build Solution or Build Rebuild Solution** (Compiles and Links.)
- You can then run your executable by typing **(CTRL + F5)** or selecting **Debug Start Without Debugging**

Graphics Programming Using OpenGL in Visual C++

- Opengl32.dll and glu32.dll should be in the system folder.
- Opengl32.lib and glu32.lib should be in the lib folder for VC++.
- gl.h and glu.h should be in a folder called GL under the include folder for VC++.
- Get glut32.lib, glut32.dll and glut.h from the course homepage and put them in the same places as the other files.
- Fire up visual studio.
- Create a new project as the following :
  File New Project (input your project name, then a directory (workspace) with the same name will be built)
  Win32 Console Application
  An Empty Application
- In the workspace click on “File View” to access (expand) the source code tree.
- In “Source Files”, add in the source code (*.cpp files).
- Select Project Settings Link and in “Object/library modules” add “Opengl32.lib glu32.lib glut32.lib”.
- Press “F7” to build “your_project.exe”.

Software Installation steps in Linux

OS	-	LINUX with Internet Connection
To Install Package	-	Open Terminal Type “yum install freeglut-devel” press enter (user should be root)
To edit the Program	-	Open Terminal Type gedit “filename.c or cpp”

To execute Program - cc filename.c –lGL –lGLU –lglut

Output - ./a.out

Basic OpenGL Elements - Programming 2D Applications

A vertex is a location in space

glVertex*

* is in the form of **nt** or **ntv**

n is the number of dimensions
t denotes the data type:
integer (i), float (f), or double (d);
pointer to an array (v)

Examples :

glVertex2i(GLint xi, GLint yi)

glVertex3f(GLfloat x, GLfloat y, GLfloat z)

GLfloat vertex[3]

glVertex3fv(vertex)

OpenGL Object Examples

glBegin (mode);
glVertex2f (x_1, y_1);
glVertex2f (x_2, y_2);
glVertex2f (x_3, y_3);
glVertex2f (x_4, y_4);
glEnd ();

where mode = GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_POLYGON, GL_LINE_LOOP, etc.

Lines of zero length are not visible.

A vertex is considered to be just a location with zero mass.

Only 10 of the OpenGL functions are legal for use between glBegin and glEnd.

- Examples : glBegin(GL_LINES);
  glVertex2f(x1, y1);
  glVertex2f(x2, y2);

```
glEnd();
```

- ```
  glBegin(GL_POINTS);
      glVertex2f(x1, y1);
      glVertex2f(x2, y2);
  glEnd();
  ```

Graphics Functions

- Primitive functions:
  points, line segments, polygons, pixels, text, curves, surfaces
- Attributes functions:
  color, pattern, typeface

Some Primitive Attributes

glClearColor (red, green, blue, alpha); - Default = (0.0, 0.0, 0.0, 0.0)
glColor3f (red, green, blue); - Default = (1.0, 1.0, 1.0)
glLineWidth (width); - Default = (1.0)
glLineStipple (factor, pattern) - Default = (1, 0xffff)
glEnable (GL_LINE_STIPPLE);
glPolygonMode (face, mode) - Default = (GL_FRONT_AND_BACK, GL_FILL)
glPointSize (size); - Default = (1.0)

- Viewing functions:
  position, orientation, clipping
- Transformation functions:
  rotation, translation, scaling
- Input functions:
  keyboards, mice, data tablets
- Control functions:
  communicate with windows, initialization, error handling
- Inquiry functions: number of colors, camera parameters/values

Matrix Mode

There are two matrices in OpenGL:

- Model-view: defines COP and orientation
- Projection: defines the projection matrix

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 500.0, 0.0, 500.0);
glMatrixMode(GL_MODELVIEW);
```

Control Functions

- OpenGL assumes origin is bottom left
```

- glutInit(int *argcp, char **argv);
- glutCreateWindow(char *title);
- glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
- glutInitWindowSize(480,640);
- glutInitWindowPosition(0,0);
- OpenGL default: RGB color, no hidden-surface removal, single buffering

Obtaining Values of OpenGL State Variables

glGetBooleanv (paramname, *paramlist);
glGetDoublev (paramname, *paramlist);
glGetFloatv (paramname, *paramlist);
glGetIntegerv (paramname, *paramlist);

Saving and Restoring Attributes

glPushAttrib (group);
glPopAttrib ( );

where group = GL_CURRENT_BIT, GL_ENABLE_BIT, GL_LINE_BIT, GL_POLYGON_BIT, etc.

Projection Transformations

glMatrixMode (GL_PROJECTION);
glLoadIdentity ( );
glFrustum (left, right, bottom, top, near, far);
gluPerspective (fov, aspect, near, far);
glOrtho (left, right, bottom, top, near, far);
- Default = (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0)
gluOrtho2D (left, right, bottom, top);

Modelview Transformations

glMatrixMode (GL_MODELVIEW);
glLoadIdentity ( );
gluLookAt (eye_x, eye_y, eye_z, at_x, at_y, at_z, up_x, up_y, up_z);
glTranslatef (dx, dy, dz);
glScalef (sx, sy, sz);
glRotatef (angle, axisx, axisy, axisz);

Writing Bitmapped Text

glPixelStorei (GL_UNPACK_ALIGNMENT, 1);
glColor3f (red, green, blue);
glRasterPos2f (x, y);
glutBitmapCharacter (font, character);
where font = GLUT_BITMAP_8_BY_13, GLUT_BITMAP_HELVETICA_10, etc.

Managing the Frame Buffer

glutInit (&argc, argv);

glutInitDisplayMode (GLUT_RGB | mode);
glutInitWindowSize (width, height);
glutInitWindowPosition (x, y);
glutCreateWindow (label);
glClear (GL_COLOR_BUFFER_BIT);
glutSwapBuffers ( );
where mode = GLUT_SINGLE or GLUT_DOUBLE.

Registering Callbacks

glutDisplayFunc (callback);
glutReshapeFunc (callback);
glutDisplayFunc (callback);
glutMotionFunc (callback);
glutPassiveMotionFunc (callback);
glutMouseFunc (callback);
glutKeyboardFunc (callback);
id = glutCreateMenu (callback);
glutMainLoop ( );

Display Lists

glNewList (number, GL_COMPILE);
glEndList ( );
glCallList (number);
glDeleteLists (number, 1);

Managing Menus

id = glutCreateMenu (callback);
glutDestroyMenu (id);
glutAddMenuEntry (label, number);
glutAttachMenu (button);
glutDetachMenu (button);
where button = GLUT_RIGHT_BUTTON or GLUT_LEFT_BUTTON.

Character Primitives Functions

glRasterPos2f(GLfloat x, GLfloat y);
glutBitmapCharacter (font, character);
where font = GLUT_BITMAP_8_BY_13, GLUT_BITMAP_HELVETICA_10,
possibilities GLUT_BITMAP_9_BY_15, GLUT_BITMAP_HELVETICA_12
GLUT_BITMAP_HELVETICA_18
GLUT_BITMAP_TIMES_ROMAN_10 or 12

Sample Program for Displaying Text

```
Drawtext(char *s)
{
    glRasterPos2f(50.0,50.0);
    int j=0;
    while(s[j]!="0\")
```

```
{
        glutBitmapCharacter (GLUT_BITMAP_HELVETICA_10, s[j]);
        j++
}
}
```

This function as to be called in the Display callback function

PART A

Design, develop, and implement the following programs in C/C++ using OpenGL API.

Lab Program – 1: Bresenham's Line Drawing

Implement Bresenham's line drawing algorithm for all types of slope.

PREAMBLE

Bresenham's line algorithm is an algorithm that determines the points of an n-dimensional raster that should be selected in order to form a close approximation to a straight line between two points. It is commonly used to draw line primitives in a bitmap image (e.g. on a computer screen), as it uses only integer addition, sub-traction and bit shifting, all of which are very cheap operations in standard computer architectures. It is an incremental error algorithm. It is one of the earliest algorithms developed in the _eld of computer graphics. An extension to the original algorithm may be used for drawing circles. While algorithms such as Wu's algorithm are also frequently used in modern com -puter graphics because they can support antialiasing, the speed and simplicity of Bresenham's line algorithm means that it is still important. The algorithm is used in hardware such as plotters and in the graphics chips of modern graphics cards. It can also be found in many software graphics libraries. Because the algorithm is very simple, it is often implemented in either the firmware or the graphics hardware of modern graphics cards.

CODE:

```
#include <GL/glut.h>
#include <stdio.h>
int x1, y1, x2, y2;

void myInit()
{
   glClearColor(0.0, 0.0, 0.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(0, 500, 0, 500);
}

void draw_pixel(int x, int y)
{
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   glEnd();
}
```

```
void draw_line(int x1, int x2, int y1, int y2)
{
   int dx, dy, i, e;
   int incx, incy, inc1, inc2;
   int x,y;
   dx = x2-x1;
   dy = y2-y1;
   if (dx < 0) dx = -dx;
   if (dy < 0) dy = -dy;
   incx = 1;
   if (x2 < x1) incx = -1;
   incy = 1;
   if (y2 < y1) incy = -1;
   x = x1; y = y1;
   if (dx > dy)
   {
         draw_pixel(x, y);
         e = 2 * dy-dx;
                  inc1 = 2*(dy-dx);
         inc2 = 2*dy;
                  for (i=0; i<dx; i++)
                   {
                            if (e >= 0)
                            {
                                     y += incy;e += inc1;
                            }
                            else
                             e += inc2;
                             x += incx;
                             draw_pixel(x, y);
                    }
   }
  else
  {
      draw_pixel(x, y);
      e = 2*dx-dy;
      inc1 = 2*(dx-dy);
      inc2 = 2*dx;
            for (i=0; i<dy; i++)
           {
                    if (e >= 0)
                    {
                            x+= incx;
```

```
                    e += inc1;
                }
                else
                e += inc2;
                y += incy;
                draw_pixel(x, y);
        }
  }
}

void myDisplay()
{
        glClear(GL_COLOR_BUFFER_BIT);
        draw_line(x1, x2, y1, y2);
        glFlush();
}

int main(int argc, char **argv)
 {
   printf( "Enter end points of the Line (x1, y1, x2, y2)\n");
   scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(0, 0);
   glutCreateWindow("Bresenham's Line Drawing");
   myInit();
   glutDisplayFunc(myDisplay);
   glutMainLoop();
   return 0;
}
```

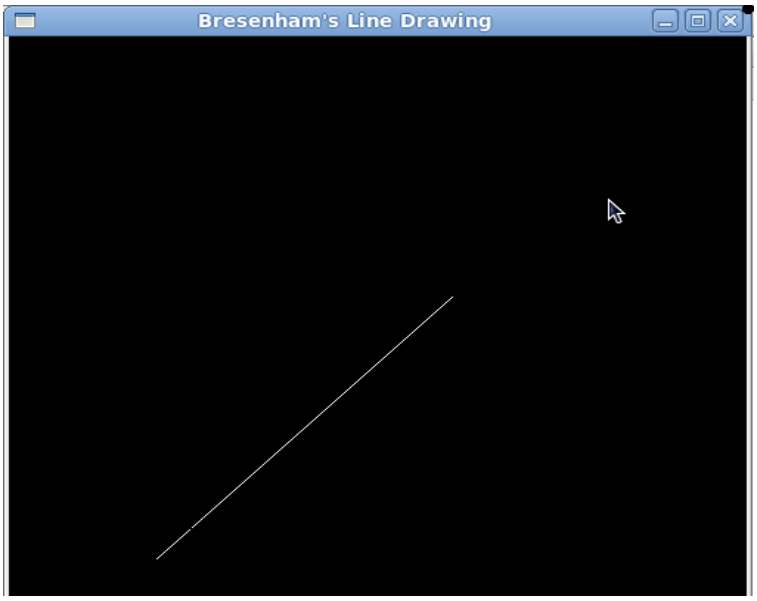
RUN:

gcc bresenham.cpp -lGL -lGLU -lglut

[root@localhost ~]# ./a.out

1. Enter end points of the Line (x1, y1, x2, y2)
   100 100 300 300

**** Output ***



Lab Program-2: Triangle Rotation

Create and rotate a triangle about the origin and a fixed point.

PREAMBLE

In linear algebra, a rotation matrix is a matrix that is used to perform a rotation in Euclidean space. For example the matrix rotates points in the xy-Cartesian plane counterclockwise through an angle Θ about the origin of the Cartesian coordinate system. To perform the rotation using a rotation matrix R, the position of each point must be represented by a column vector v, containing the coordinates of the point. A rotated vector is obtained by using the matrix multiplication Rv. Since matrix multiplication has no effect on the zero vector (i.e., on the coordinates of the origin), rotation matrices can only be used to describe rotations about the origin of the coordinate system. Rotation matrices provide a simple algebraic description of such rotations, and are used extensively for computations in geometry, physics, and computer graphics. In 2-dimensional space, a rotation can be simply described by an angle Θ of rotation, but it can be also represented by the 4 entries of a rotation matrix with 2 rows and 2 columns. In 3-dimensional space, every rotation can be interpreted as a rotation by a given angle about a single fixed axis of rotation (see Euler's rotation theorem), and hence it can be simply described by an angle and a vector with 3 entries. However, it can also be represented by the 9 entries of a rotation matrix with 3 rows and 3 columns. The notion of rotation is not commonly used in dimensions higher than 3; there is a notion of a rotational displacement, which can be represented by a matrix, but no associated single axis or angle.

CODE:

```
#define BLACK 0
#include <stdio.h>
#include <math.h>
#include <GL/glut.h>

GLfloat house[3][3]={{100.0,250.0,175.0},{100.0,100.0,300.0},{1.0,1.0,1.0}};
GLfloat rotatemat[3][3]={{0},{0},{0}};
GLfloat result[3][3]={{0},{0},{0}};
GLfloat arbitrary_x=0;
GLfloat arbitrary_y=0;
float rotation_angle;

void multiply()
{
        int i,j,k;
        for(i=0;i<3;i++)
```

```
        for(j=0;j<3;j++)
        {
                result[i][j]=0;
                for(k=0;k<3;k++)
                result[i][j]=result[i][j]+rotatemat[i][k]*house[k][j];
        }
}


void rotate()
{
        GLfloat m,n;
        m=-arbitrary_x*(cos(rotation_angle) -1) + arbitrary_y * (sin(rotation_angle));
        n=-arbitrary_y * (cos(rotation_angle) - 1) -arbitrary_x * (sin(rotation_angle));
        rotatemat[0][0]=cos(rotation_angle);
        rotatemat[0][1]=-sin(rotation_angle);
        rotatemat[0][2]=m;
        rotatemat[1][0]=sin(rotation_angle);
        rotatemat[1][1]=cos(rotation_angle);
        rotatemat[1][2]=n;
        rotatemat[2][0]=0;
        rotatemat[2][1]=0;
        rotatemat[2][2]=1;
        //multiply the two matrices
        multiply();
}

void drawhouse()
{
        glColor3f(0.0, 0.0, 1.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(house[0][0],house[1][0]);
        glVertex2f(house[0][1],house[1][1]);
        glVertex2f(house[0][2],house[1][2]);
        glEnd();
}

void drawrotatedhouse()
{
        glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(result[0][0],result[1][0]);
        glVertex2f(result[0][1],result[1][1]);
        glVertex2f(result[0][2],result[1][2]);
        glEnd();
```

```
}


 void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        drawhouse();
        drawrotatedhouse();
        glFlush();
}

void myinit()
{
        glClearColor(1.0,1.0,1.0,1.0);
        glColor3f(1.0,0.0,0.0);
        glPointSize(1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0,499.0,0.0,499.0);
}

int main(int argc, char** argv)
{
        int ch;
        printf("Enter your choice \n1: Rotation about origin \n2: Rotation about a Fixed point\n");
        scanf("%d",&ch);

        switch(ch)
        {
                case 1: printf("Enter the rotation angle in degree :");
                scanf("%f", &rotation_angle);
                rotation_angle= (3.14 * rotation_angle) / 180;
                rotate();
                break;

                case 2: printf("Enter the fixed points :");
                scanf("%f%f", &arbitrary_x,&arbitrary_y);
                printf("Enter rotation angle in degree :");
                scanf("%f", &rotation_angle);
                rotation_angle= (3.14 * rotation_angle) / 180;
                rotate();
                break;

        }
```

```
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("house rotation");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
        return 0;
}
```

RUN:

TriangleRotation.cpp -lglut -lGL -lGLUT

./a.out

**** Output ***

Enter your choice

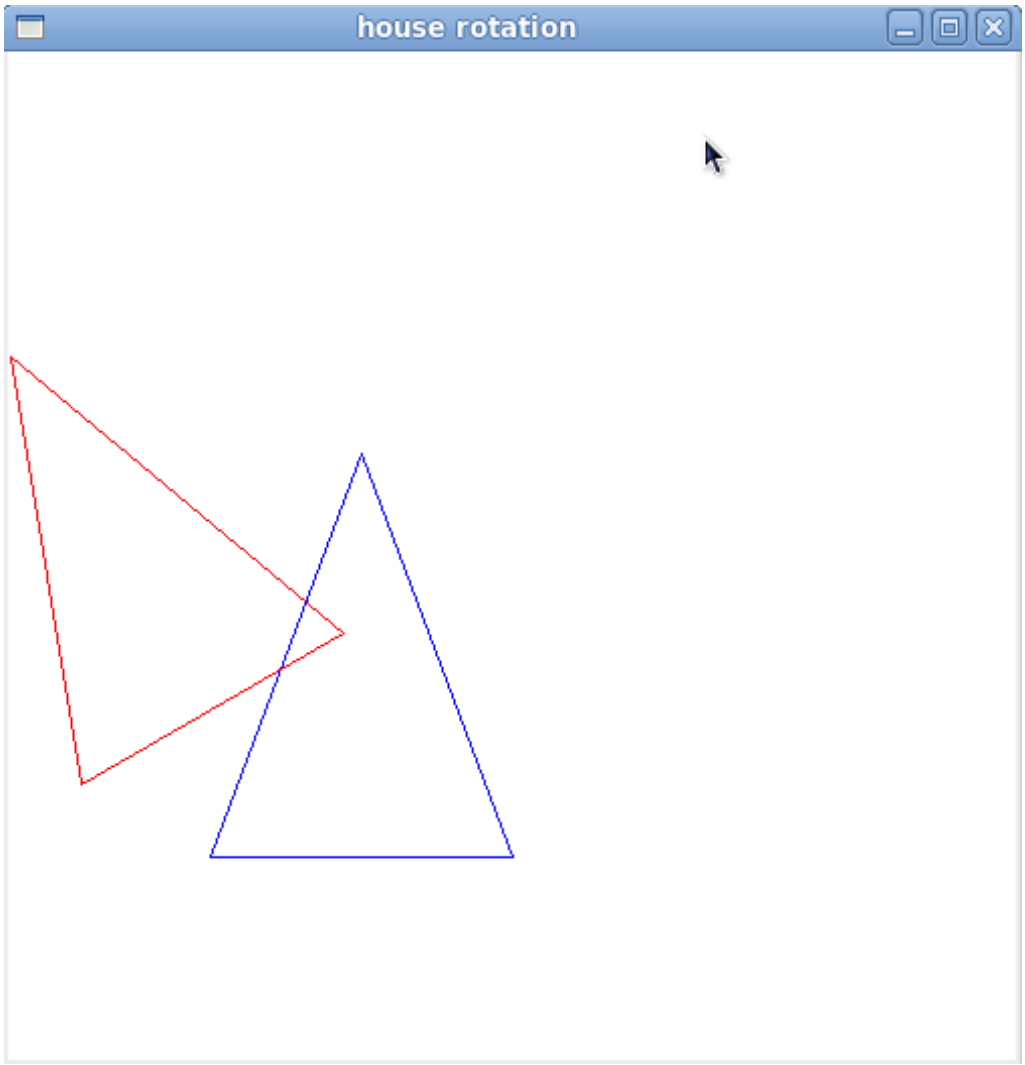
1: Rotation about origin

2: Rotation about a Fixed point

1

Rotation about origin

Enter the rotation angle in degree :30



Enter your choice

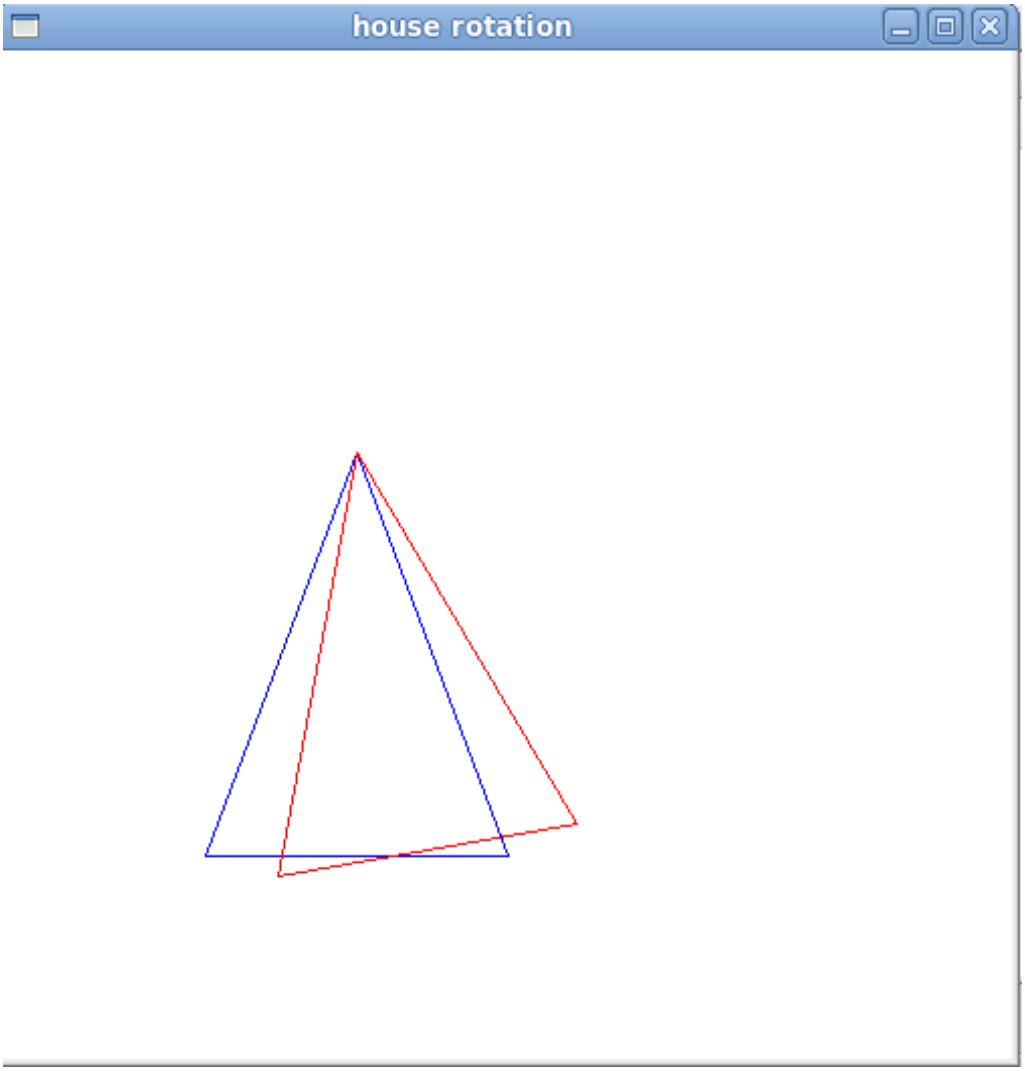
1: Rotation about origin

2: Rotation about a Fixed point

2

Enter the fixed points :175 300

Enter rotation angle in degree :10



Lab Program –3: Color cube and spin

Draw a color cube and spin it using OpenGL transformation matrices.

PREAMBLE

Algorithm:

Modeling a color cube with simple data structures
+ Define vertices with centre of cube as origin
+ Define normals to identify faces
+ Define colors

Rotating the cube
+ Define angle of rotation [or accept it as input]
+ Define/Input rotation axis
+ Use swap-buffer to ensure smooth rotation

Using callback functions to indicate either or both of the following:
+ Angle of rotation
+ Axis of rotation

1. Define global arrays for vertices and colors

```
GLfloat vertices[][3] = {{-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}};
GLfloat colors[][3] = {{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}};
```

2. Draw a polygon from a list of indices into the array vertices and use color corresponding to first index

```
void polygon(int a, int b, int c, int d)
{
        glBegin(GL_POLYGON);
        glColor3fv(colors[a]);
        glVertex3fv(vertices[a]);
        glVertex3fv(vertices[b]);
        glVertex3fv(vertices[c]);
        glVertex3fv(vertices[d]);
        glEnd();
}
```

3.Draw cube from faces.

```
void colorcube( )
{
        polygon(0,3,2,1);
```

```
        polygon(2,3,7,6);
        polygon(0,4,7,3);
        polygon(1,2,6,5);
        polygon(4,5,6,7);
        polygo n(0,1,5,4);
}
```

4. Define the Display function to clear frame buffer , Z-buffer ,rotate cube and draw,swap buffers.

```
void display(void)
{
        /* display callback, clear frame buffer and z buffer,
        rotate cube and draw, swap buffers */
        glLoadIdentity();
        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);
        colorcube();
        glutSwapBuffers();
}
```

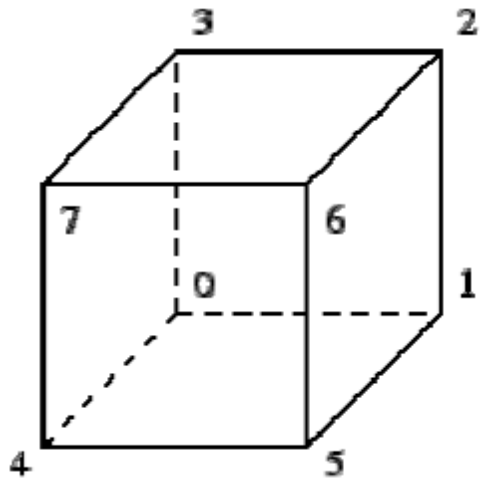
5. Define the spincube function to spin cube 2 degrees about selected axis.

6. Define mouse callback to select axis about which to rotate.

7. Define reshape function to maintain aspect ratio.

```
void myReshape(int w, int h)
{
        glViewport(0, 0, w, h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        if (w <= h)
        glOrtho(-2.0, 2.0, -2.0 *(GLfloat)h/(GLfloat)w,2.0*(GLfloat)h/(GLfloat)w,-10.0,10.0);
        else
        glOrtho(-2.0*(GLfloat)w/(GLfloat)h,2.0*(GLfloat)w/(GLfloat) h, -2.0, 2.0, -10.0, 10.0);
        glMatrixMode(GL_MODELVIEW);
}
```

8. Define main function to create window and to call all function.



CODE:

```
#include <stdlib.h>
#include <GL/glut.h>
GLfloat vertices[][3] = {{-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}};
GLfloat normals[][3] = {{-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}};
GLfloat colors[][3] = {{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}};

void  polygon(int a, int b, int c , int d)
{
        /* draw a polygon via list of vertices */
        glBegin(GL_POLYGON);
        glColor3fv(colors[a]);
        glNormal3fv(normals[a]);
        glVertex3fv(vertices[a]);
        glColor3fv(colors[b]);
        glNormal3fv(normals[b]);
        glVertex3fv(vertices[b]);
        glColor3fv(colors[c]);
        glNormal3fv(normals[c]);
        glVertex3fv(vertices[c]);
        glColor3fv(colors[d]);
        glNormal3fv(normals[d]);
        glVertex3fv(vertices[d]);
        glEnd();
```

```
}

void  colorcube(void)
{
        /* map vertices to faces */
        polygon(0,3,2,1);
        polygon(2,3,7,6);
        polygon(0,4,7,3);
        polygon(1,2,6,5);
        polygon(4,5,6,7);
        polygon(0,1,5,4);
}
static GLfloat theta[] = {0.0,0.0,0.0};
static GLint axis = 2;

void  display(void)
{
        /* display callback, clear frame buffer and z buffer,
        rotate cube and draw, swap buffers */
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        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);
        colorcube();
        glFlush();
        glutSwapBuffers();
}

void  spinCube()
{
        /* Idle callback, spin cube 2 degrees about selected axis */
        theta[axis] += 1.0;
        if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
        /* display(); */
        glutPostRedisplay();
}

void  mouse(int btn, int state, int x, int y)
{
        /* mouse callback, selects an axis about which to rotate */
        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;
}
```

```
void  myReshape(int w, int h)
{
        glViewport(0, 0, w, h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        if (w <= h)
        glOrtho(-2.0, 2.0, -2.0 * (GLfloat) h / (GLfloat) w, 2.0 * (GLfloat) h / (GLfloat) w, -10.0,
        10.0);
        else
        glOrtho(-2.0 * (GLfloat) w / (GLfloat) h, 2.0 * (GLfloat) w / (GLfloat) h, -2.0, 2.0, -10.0,
        10.0);
        glMatrixMode(GL_MODELVIEW);
}

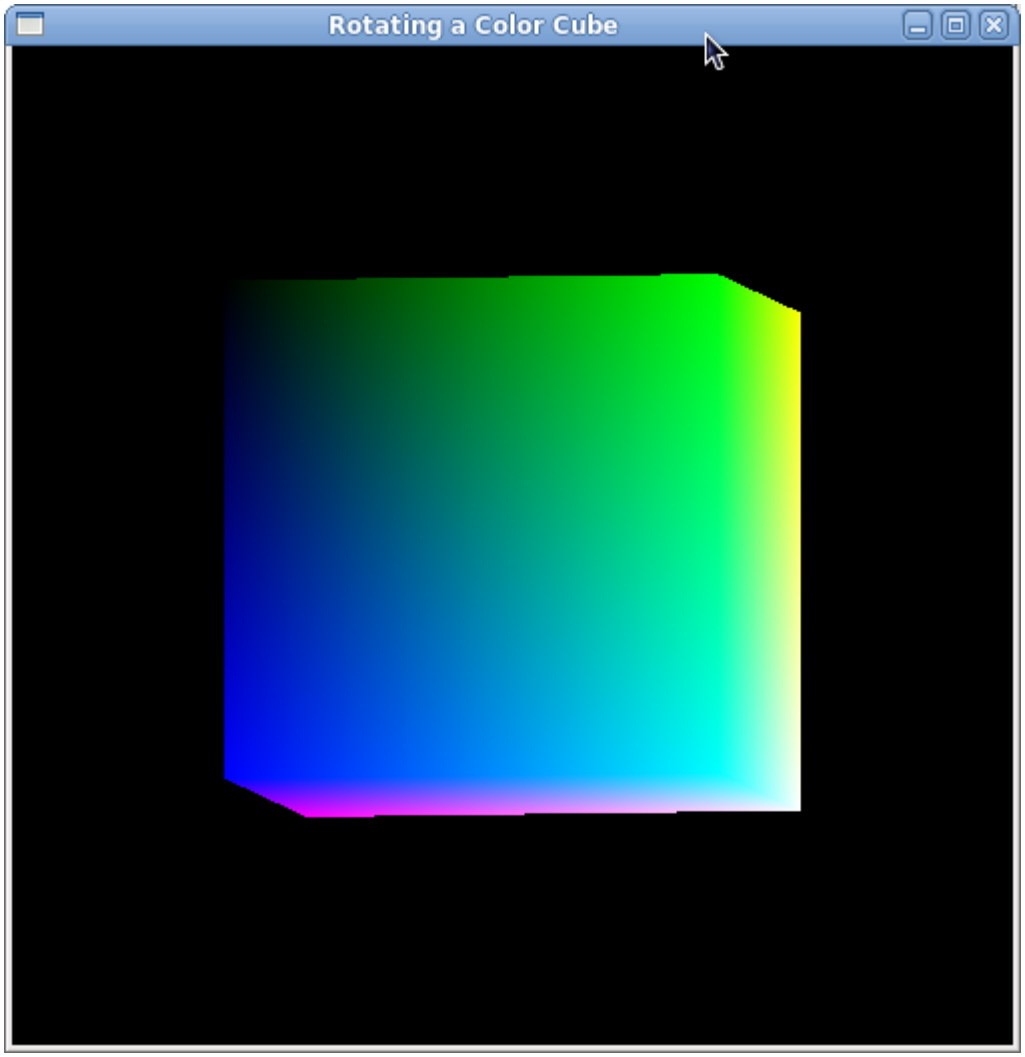
void  main(int argc, char **argv)
{
         glutInit(&argc, argv);
        /* need both double buffering and z buffer */
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(500, 500);
        glutCreateWindow("Rotating a Color Cube");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glutIdleFunc(spinCube);
        glutMouseFunc(mouse);
        glEnable(GL_DEPTH_TEST); /* Enable hidden--surface--removal */
        glutMainLoop();
}
```

RUN:

```
gcc 3.cpp  -lglut  -lGL  -lGLUT

./a.out
```

*** Output ***



Lab Program –4: Color cube with perspective viewing

Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.

PREAMBLE

Concepts Used:

+ Data structures for representation of a cube
+ Perspective viewing
+ Defining and moving the camera
+ Input functions using Keyboard and mouse

Algorithm:

1. Modeling a color cube with simple data structures
+ Define vertices with centre of cube as origin
+ Define normals to identify faces
+ Define colors

2. Camera Manipulations
+ Define initial camera position - take care to define outside the cube.

3. Callback Function
+ Handling mouse inputs - use the 3 mouse buttons to define the 3 axes of rotation
+ Handling Keyboard Inputs - use the 3 pairs of keys to move the viewer along +ive and -ive directions of the X and Y axes respectively.

Pseudo Code

1. Define global arrays for vertices and colors

```
GLfloat vertices[][3] = {{-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}};
GLfloat colors[][3] = {{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}};
```

2. Draw a polygon from a list of indices into the array vertices and use color corresponding to first index

```
void polygon(int a, int b, int c, int d)
{
        glBegin(GL_POLYGON);
        glColor3fv(colors[a]);
        glVertex3fv(vertices[a]);
```

```
        glVertex3fv(vertices[b]);
        glVertex3fv(vertices[c]);
        glVertex3fv(vertices[d]);
        glEnd();
}
```

3.Draw cube from faces.

```
void colorcube( )
{
        polygon(0,3,2,1);
        polygon(2,3,7,6);
        polygon(0,4,7,3);
        polygon(1,2,6,5);
        polygon(4,5,6,7);
        polygo n(0,1,5,4);
}
```

4. Intialize the theta value and initial viewer location and axis

```
static GLfloat theta[]={0.0,0.0,0.0}
static Glint axis =2;
static GLdouble viewer[]={0.0,0.0,5,0}
```

5. Define display function to upadate viewer position in model view matrix

```
void display(void)
{
        /* display callback, clear frame buffer and z buffer, rotate cube and draw, swap
        buffers */
        glLoadIdentity();
        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);
        colorcube();
        glutSwapBuffers();
}
```

6. Define mouse callback function to rotate about axis.

7. Define key function to move viewer position .use x,X,y,Y z,Z keys to move viewer position.

8. Define reshape function to maintain aspect ratio and then use perspective view.

9. Define main function to create window and to call all function.

CODE:

```
#include <stdlib.h>
#include <GL/glut.h>
GLfloat vertices[][3] = {{-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}};
GLfloat normals[][3] = {{-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}};
GLfloat colors[][3] = {{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}};

void  polygon(int a, int b, int c , int d)
{
        glBegin(GL_POLYGON);
        glColor3fv(colors[a]);
        glNormal3fv(normals[a]);
        glVertex3fv(vertices[a]);
        glColor3fv(colors[b]);
        glNormal3fv(normals[b]);
        glVertex3fv(vertices[b]);
        glColor3fv(colors[c]);
        glNormal3fv(normals[c]);
        glVertex3fv(vertices[c]);
        glColor3fv(colors[d]);
        glNormal3fv(normals[d]);
        glVertex3fv(vertices[d]);
        glEnd();
}
void colorcube()
{
        polygon(0,3,2,1);
        polygon(2,3,7,6);
        polygon(0,4,7,3);
        polygon(1,2,6,5);
        polygon(4,5,6,7);
        polygon(0,1,5,4);
}

static GLfloat theta[] = {0.0,0.0,0.0};
static GLint axis = 2;
static GLdouble viewer[]= {0.0, 0.0, 5.0}; /* initial viewer location */
```

```
void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        /* Update viewer position in modelview matrix */
        glLoadIdentity();
        gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
        /* rotate cube */
        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);
        colorcube();
        glFlush();
        glutSwapBuffers();
}

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;
        display();
}

void keys(unsigned char key, int x, int y)
{
        /* Use x, X, y, Y, z, and Z keys to move viewer */
        if(key == 'x') viewer[0]-= 1.0;
        if(key == 'X') viewer[0]+= 1.0;
        if(key == 'y') viewer[1]-= 1.0;
        if(key == 'Y') viewer[1]+= 1.0;
        if(key == 'z') viewer[2]-= 1.0;
        if(key == 'Z') viewer[2]+= 1.0;
        display();
}

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);
        /* Or we can use gluPerspective */
        /* gluPerspective(45.0, w/h, -10.0, 10.0); */
        glMatrixMode(GL_MODELVIEW);
}

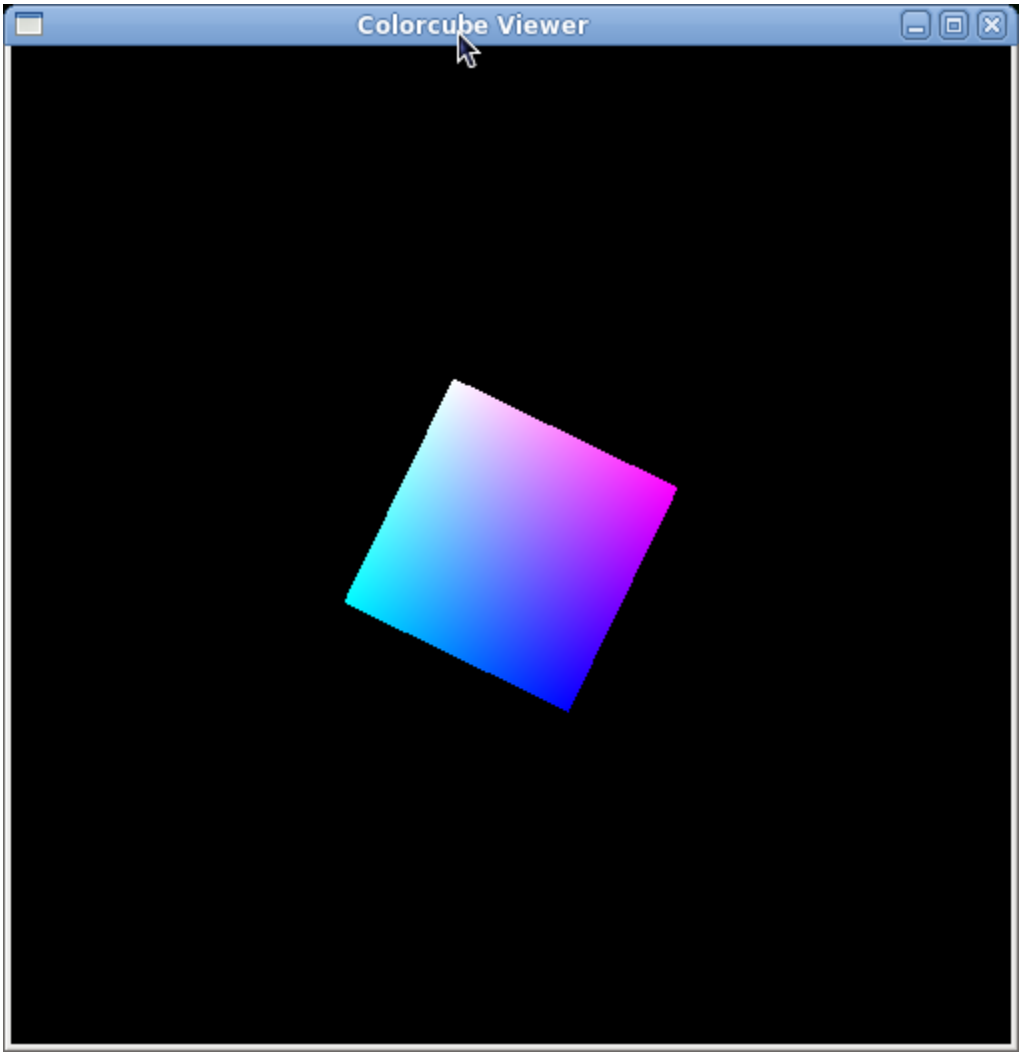
void main(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(500, 500);
        glutCreateWindow("Colorcube Viewer");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glutMouseFunc(mouse);
        glutKeyboardFunc(keys);
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
}
```

RUN:

```
gcc 4.c  -lglut  -lGL  -lGLUT
./a.out
```

*** Output ***

Rotation with respect to x - axis



Lab Program – 5: Cohen-Sutherland

Clip a line using Cohen-Sutherland algorithm.

PREAMBLE

The Cohen-Sutherland algorithm uses a divide-and-conquer strategy. The line segment's end-points are tested to see if the line can be trivally accepted or rejected. If the line cannot be trivally accepted or rejected, an intersection of the line with a window edge is determined and the trivial reject/accept test is repeated. This process is continued until the line is accepted. To perform the trivial acceptance and rejection tests, we extend the edges of the window to divide the plane of the window into the nine regions. Each end point of the line segment is then assigned the code of the region in which it lies.

Algorithm:

Every line endpoint is assigned a 4 bit Region code. The appropriate bit is set depending on the location of the endpoint with respect to that window component as shown below:

1001	1000	1010
0001	0000	0010
0101	0100	0110

Endpoint Left of window then set bit 1 Endpoint Right of window then set bit 2 Endpoint Below window then set bit 3 Endpoint Above window then set bit 4

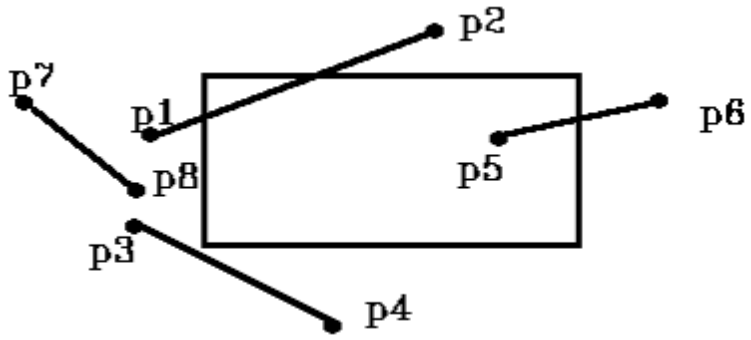
1. Given a line segment with endpoint P1=(x1,y1) and P2=(x2,y2)

2. Compute the 4-bit codes for each endpoint.
If both codes are 0000,(bitwise OR of the codes yields 0000) line lies completely inside the window: pass the endpoints to the draw routine.
If both codes have a 1 in the same bit position (bitwise AND of the codes is not 0000), the line lies outside the window. It can be trivially rejected.

3. If a line cannot be trivially accepted or rejected, at least one of the two endpoints must lie outside the window and the line segment crosses a window edge. This line must be clipped at the window edge before being passed to the drawing routine.

4. Examine one of the endpoints, say . Read 's 4-bit code in order:
Left-to-Right, Bottom-to-Top.

5. When a set bit (1) is found, compute the intersection I of the corresponding window edge with the line from to . Replace with I and repeat the algorithm.
Example1:

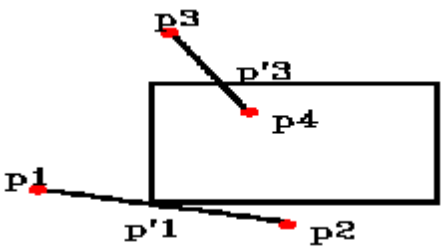


Can determine the bit code by testing the endpoints with window as follows:

If x is less than Xwmin then set bit 1
If x is greater than Xwmax then set bit 2
If y is less than Ywmin then set bit 3
If y is greater than Ywmax then set bit 4

Note: can use 4 element Boolean matrix and set C[Left] = true / false (1/0). If both endpoints = 0000 (in window) then display line. If both endpoints have a bit set in same position (P7, P8) then the line is completely outside the window and is rejected. So: can do logical AND of region codes and reject if result is 0000 Can do logical OR of region codes and accept if result = 0000
For the rest of the lines we must check for intersection with window. May still be outside, e.g. P3 - P4 in the above image. If point is to Left of window then compute intersection with Left window boundary. Do the same for Right, Bottom, and Top. Then recompute region code restest. So the algorithm is as follows:
1. Compute region code for endpoints
2. Check for trivial accept or reject
3. If step 2 unsuccessful then compute window intersections in order:\\
Left, Right, Bottom, Top (only do 1)
4. Repeat steps 1, 2,3 until done.

Example2:



I. 1) P1 = 0001 2) no 3) P1 = P'1

P2 = 1000

II. 1) P'1= 0000 2) no 3) P2 = P' 2
P2 = 1000

III. 1) P'1 = 0000 2) Yes - accept & display
P'2 = 0000

Look at how to compute the line intersections
for P'1: m = dy/dx = (y1 - y'1)/(x1 - x'1) P1(x1, y1)
P'1(x'1, y'1)

or y'1 = y1 + m(x'1 - x1)

but for Left boundary x'1 = Xwmin
for Right boundary x'1 = Xwmax

Similarly for Top / Bottom, e.g. P'3
x'3 = x3 + (y'3 - y3) / m
for Top y'3 = Ywmax for Bottom y'3 = Ywmin

CODE:

```
#include <stdio.h>
#include <GL/glut.h>
#include<stdbool.h>
double xmin=50,ymin=50, xmax=100,ymax=100; // Window boundaries
double xvmin=200,yvmin=200,xvmax=300,yvmax=300; // Viewport boundaries
//bit codes for the right, left, top, & bottom
const int TOP = 8;
const int BOTTOM = 4;
const int RIGHT= 2;
const int LEFT = 1;
//used to compute bit codes of a intersecting point

int ComputeOutCode (double x, double y);
//Cohen-Sutherland clipping algorithm clips a line from
//P0 = (x0, y0) to P1 = (x1, y1) against a rectangle with
//diagonal from (xmin, ymin) to (xmax, ymax).

void CohenSutherlandLineClipAndDraw (double x0, double y0,double x1, double y1)
{
        //Outcodes for P0, P1, and whatever point lies outside the clip rectangle
```

```
int outcode0, outcode1, outcodeOut;
bool accept = false, done = false;
//compute outcodes
outcode0 = ComputeOutCode (x0, y0);
outcode1 = ComputeOutCode (x1, y1);
do
{
        if ((outcode0 |outcode1)==0) //logical or is 0 Trivially accept & exit
        {
                accept = true;
                done = true;
        }
        else if (outcode0 & outcode1) //logical and is not 0. Trivially reject and exit
        done = true;
        else
        {
                //failed both tests, so calculate the line segment to clip
                //from an outside point to an intersection with clip edge
                double x, y;
                //At least one endpoint is outside the clip rectangle; pick it.
                outcodeOut = outcode0? outcode0: outcode1;
                float slope=(y1-y0)/(x1-x0);
                //Now find the intersection point;
                //use formulas y = y0 + slope * (x - x0), x = x0 + (1/slope)* (y - y0)
                if (outcodeOut & TOP) //point is above the clip rectangle
                {
                        x = x0 + (1/slope) * (ymax - y0);
                        y = ymax;
                }
                else if (outcodeOut & BOTTOM) //point is below the clip rectangle
                {
                        x = x0 + (1/slope) * (ymin - y0);
                        y = ymin;
                }
                else if (outcodeOut & RIGHT) //point is to the right of clip rectangle
                {

                        y = y0 + slope * (xmax - x0);
                        x = xmax;
                }
                else //point is to the left of clip rectangle
                {
                        y = y0 + slope* (xmin - x0);
                        x = xmin;
                }
```

```
                    //Now we move outside point to intersection point to clip
                    //and get ready for next pass.
                    if (outcodeOut == outcode0)
                    {
                            x0 = x;
                            y0 = y;
                            outcode0 = ComputeOutCode (x0, y0);
                    }
                    else
                    {
                            x1 = x;
                            y1 = y;
                            outcode1 = ComputeOutCode (x1, y1);
                    }
            }
    }
    while (!done);
    if (accept)
    {
            // Window to viewport mappings
            double sx=(xvmax-xvmin)/(xmax-xmin); // Scale parameters
            double sy=(yvmax-yvmin)/(ymax-ymin);
            double vx0=xvmin+(x0-xmin)*sx;
            double vy0=yvmin+(y0-ymin)*sy;
            double vx1=xvmin+(x1-xmin)*sx;
            double vy1=yvmin+(y1-ymin)*sy;
            //draw a red colored viewport
            glColor3f(1.0, 0.0, 0.0);
            glBegin(GL_LINE_LOOP);
            glVertex2f(xvmin, yvmin);
            glVertex2f(xvmax, yvmin);
            glVertex2f(xvmax, yvmax);
            glVertex2f(xvmin, yvmax);
            glEnd();
            glColor3f(0.0,0.0,1.0); // draw blue colored clipped line
            glBegin(GL_LINES);
            glVertex2d (vx0, vy0);
            glVertex2d (vx1, vy1);
            glEnd();
    }
}
//Compute the bit code for a point (x, y) using the clip rectangle
//bounded diagonally by (xmin, ymin), and (xmax, ymax)
int ComputeOutCode (double x, double y)
{
```

```
        int code = 0;
        if (y > ymax) //above the clip window
        code |= TOP;
        else if (y < ymin) //below the clip window
        code |= BOTTOM;
        if (x > xmax) //to the right of clip window
        code |= RIGHT;
        else if (x < xmin) //to the left of clip window
        code |= LEFT;
        return code;
}

void display()
{
        double x0=60,y0=20,x1=80,y1=120;
        glClear(GL_COLOR_BUFFER_BIT);
        //draw the line with red color
        glColor3f(1.0,0.0,0.0);
        //bres(120,20,340,250);
        glBegin(GL_LINES);
        glVertex2d (x0, y0);
        glVertex2d (x1, y1);
        glEnd();
        //draw a blue colored window
        glColor3f(0.0, 0.0, 1.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(xmin, ymin);
        glVertex2f(xmax, ymin);
        glVertex2f(xmax, ymax);
        glVertex2f(xmin, ymax);
        glEnd();
        CohenSutherlandLineClipAndDraw(x0,y0,x1,y1);
        glFlush();
}

void myinit()
{
        glClearColor(1.0,1.0,1.0,1.0);
        glColor3f(1.0,0.0,0.0);
        glPointSize(1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0,499.0,0.0,499.0);
}
```

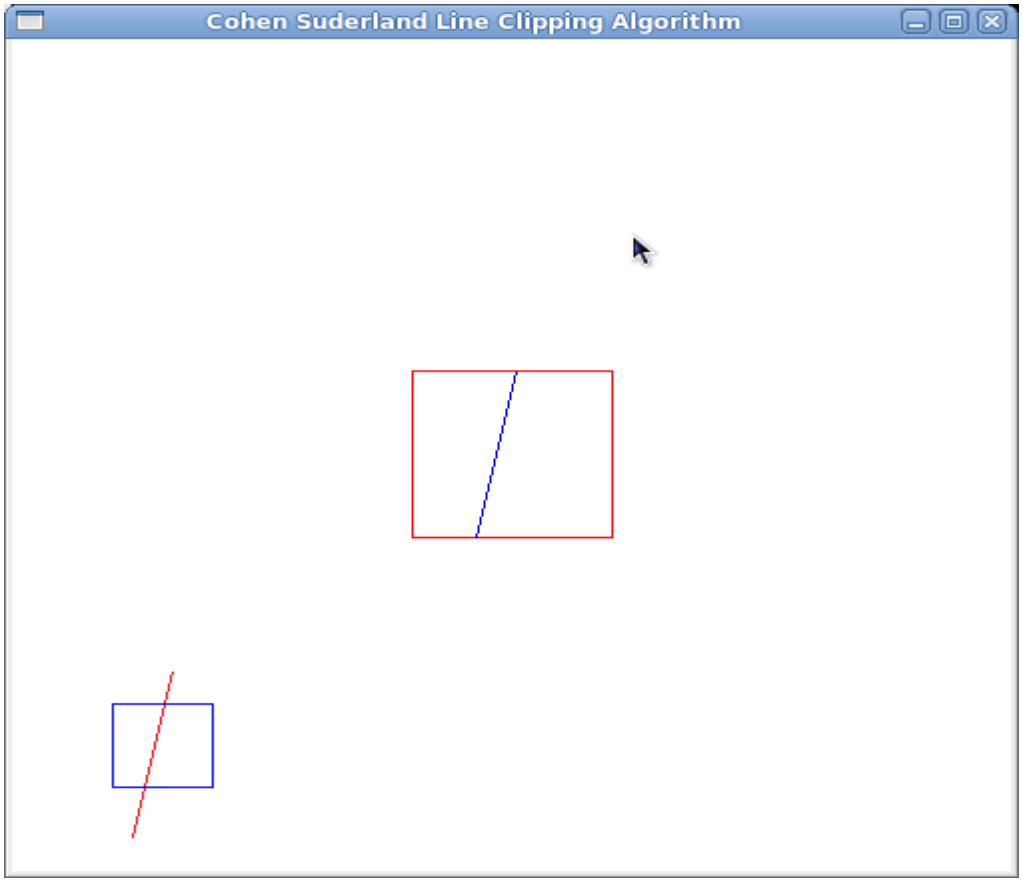
```
void main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("Cohen Suderland Line Clipping Algorithm");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

RUN:

gcc 5.c -lglut -lGL -lGLUT

./a.out

***** Output *****



Lab Program –6: Tea pot

To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.

PREAMBLE

Concepts Used:

+ Translation, Scaling
+ Material Properties, Light Properties
+ Depth in 3D Viewing
+ Using predefined GLU library routines

Algorithm[High Level]:

1. Recognise the objects in the problem domain and their components
+ Teapot
+ Table
    - Table top
    - 4 legs
+ Walls
    - Left wall
    - Right wall
    - Floor

2. If depth testing is NOT used ensure that the background is drawn first

3. Enable lighting first - since lighting is applied at the time objects are drawn.

4. Draw the objects.

Algorithm[Low Level]:

+ Main Function:
+ Initialization for Display, Mode , Window
+ Enable lighting, shading, depth test
+ Register display and callback function
+ Define viewport appropriately
+ Call mainloop
+ Display function:
+ Define Material properties
+ Define lighting properties
+ Set the camera by defining
    - projection parameters
    - camera parameters
+ Plan the required centre position for each of the components

+ Draw each component using translation, scaling and rotation as required

Pseudo Code

1. Include glut heaeder files.
2. Define wall function as following

```
void wall(double thickness)
{
        glPushMatrix();
        glTranslated(0.5,0.5*thickness,0.5);
        glScaled(1.0,thickness,1.0);
        glutSolidCube(1.0);
        gPopMatrix();
}
```

3. Draw one tableleg using tableLeg function as following

```
void tableLeg(double thick,double len)
{
        glpushMatrix();
        glTranslated(0,len/2,0);
        glScaled(thick,len,thick);
        glutsolidCube(1.0);
        glPopMatrix();
}
```

4. Draw the table using table function.

i) draw the table top using glutSolidCube(1.0) function.Before this use gltranslated and glScaled function to fix table in correct place.

ii) Draw four legs by calling the function tableleg .before each call use gltranslated function to fix four legs in correct place.

5. Define the function displaySolid

i) Initialize the properties of the surface material and set the light source properties.
ii) Set the camera position.
iii) Draw the teapot using glutSolidTeapot(0.08).before this call gltranslated and glRotated.
iv) Call table function and wall function
v) Rotate wall about 90 degree and then call wall function.
vi) Rotate wall about -90 degree and then call wall function.

6. Define the main function to create window and enable lighting function using glEnable(GL_LIGHTING)

CODE:

```
#include<GL/glut.h>
#include<stdio.h>

        GLfloat mat_ambient[]={0.7,0.7,0.7,1.0};
        GLfloat mat_diffuse[]={0.5,0.5,0.5,1.0};
        GLfloat mat_specular[]={1.0,1.0,1.0,1.0};
        const  GLfloat mat_shininess[] = {50.0};
        GLfloat light_intensity[]= {0.7,0.7,0.7,1.0};
        GLfloat light_position[]={2.0,6.0,3.0,0.0};
void init()
{
        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);
        glLightfv(GL_LIGHT0,GL_POSITION,light_position);
        glLightfv(GL_LIGHT0,GL_DIFFUSE,light_intensity);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-2.0,2.0,-2.0,2.0,-10.0,10.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(2.0,1.0,2.0,0.0,0.2,0.2,0.0,1.0,0.0);
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
}
void teapot()
{
        glPushMatrix();
        glTranslated(0.4,0.0,0.4);
        glRotated(30,0,1,0);
        glutSolidTeapot(0.2);
        glPopMatrix();
}
void tabletop()
{
        glPushMatrix();
        glTranslated(0.0,-0.3,0.0);
        glScaled(7.0,0.5,7.0);
        glutSolidCube(0.2);
        glPopMatrix();

}


void frontleg()
```

```
{
        glPushMatrix();
        glTranslated(0.5,-0.7,0.5);
        glScaled(0.5,7.0,0.5);
        glutSolidCube(0.1);
        glPopMatrix();
}
void leftleg()
{
        glPushMatrix();
        glTranslated(-0.5,-0.7,0.5);
        glScaled(0.5,7.0,0.5);
        glutSolidCube(0.1);
        glPopMatrix();
}
void rightleg()
{
        glPushMatrix();
        glTranslated(0.5,-0.7,-0.5);
        glScaled(0.5,7.0,0.5);
        glutSolidCube(0.1);
        glPopMatrix();
}
void backleg()
{
        glPushMatrix();
        glTranslated(-0.5,-0.7,-0.5);
        glScaled(0.5,7.0,0.5);
        glutSolidCube(0.1);
        glPopMatrix();
}
void leftwall()
{
        glPushMatrix();
        glTranslated(-1.0,-0.0,0.0);
        glScaled(0.1,10.0,10.0);
        glutSolidCube(0.2);
        glPopMatrix();
}
void bottomfloor()
{
        glPushMatrix();
        glTranslated(0.0,-1.0,0.0);
        glScaled(10.1,0.1,10.0);
        glutSolidCube(0.2);
```

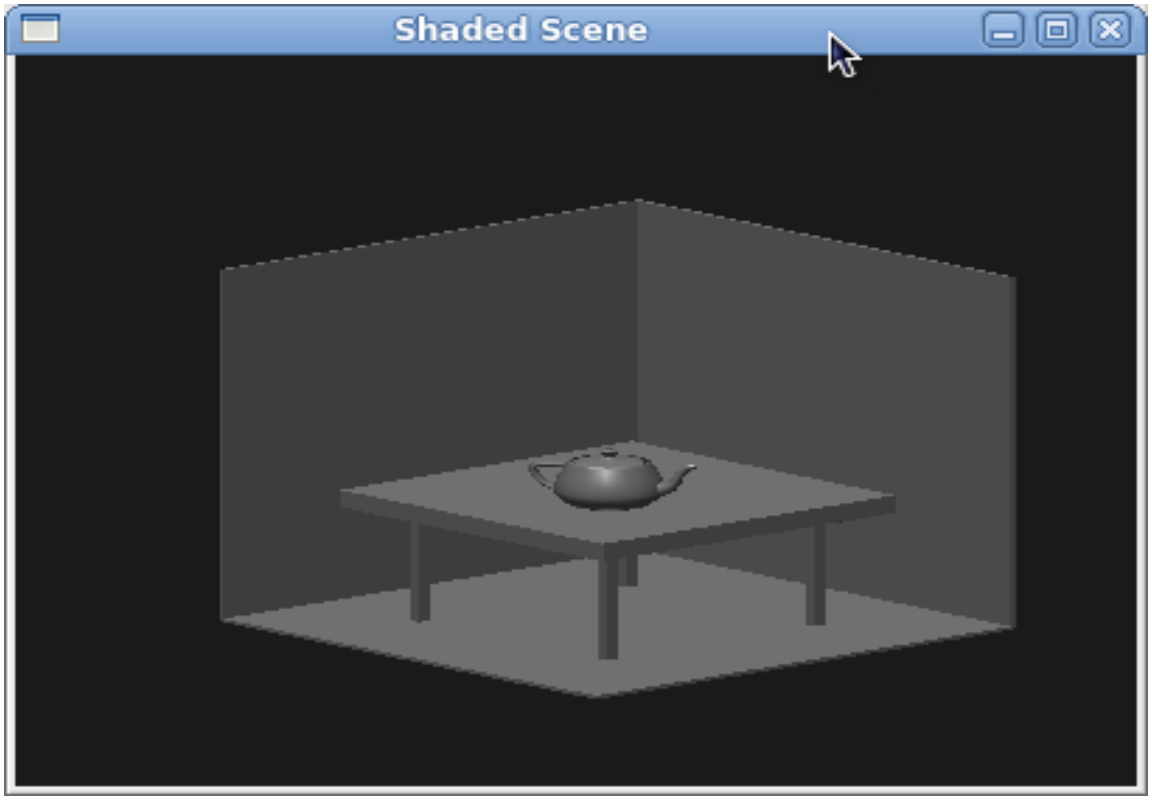
```
        glPopMatrix();
}
void rightwall()
{
        glPushMatrix();
        glTranslated(0.0,0.0,-1.0);
        glScaled(10.0,10.0,0.1);
        glutSolidCube(0.2);
        glPopMatrix();
}
void display()
{
        init();
        teapot();
        tabletop();
        frontleg();
        leftleg();
        rightleg();
        backleg();
        bottomfloor();
        rightwall();
        leftwall();
        glFlush();
}
void main(int argc, char **argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
        glutInitWindowPosition(50,50);
        glutInitWindowSize(400,300);
        glutCreateWindow("shaded Scene");
        glutDisplayFunc(display);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glShadeModel(GL_SMOOTH);
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_NORMALIZE);
        glClearColor(0.1,0.1,0.1,0.0);
        glViewport(0,0,640,480);
        glutMainLoop();


}
```

RUN:

```
gcc 6.c  -lglut  -lGL  -lGLUT
./a.out
```

*** Output ***



Lab Program –7: 3D Sierpinski gasket

Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user.

PREAMBLE

Sierpinski's Triangle is a very famous fractal that's been seen by most advanced math students. This fractal consists of one large triangle, which contains an in_nite amount of smaller triangles within. The in_nite amount of triangles is easily understood if the fractal is zoomed in many levels. Each zoom will show yet more previously unseen triangles embedded in the visible ones.

Creating the fractal requires little computational power. Even simple graphing calculators can easily make this image. The fractal is created pixel by pixel, using random numbers; the fractal will be slightly different each time due to this. Although, if you were to run the program repeatedly, and allow each to use an infinite amount of time, the results would be always identical. No one has an infinite amount of time, but the differences in the infinite versions are very small.

A fractal is generally "a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole".

To generate this fractal, a few steps are involved.
We begin with a triangle in the plane and then apply a repetitive scheme of operations to it (when we say triangle here, we mean a blackened, filled-in' triangle). Pick the midpoints of its three sides. Together with the old vertices of the original triangle, these midpoints define four congruent triangles of which we drop the center one. This completes the basic construction step. In other words, after the first step we have three congruent triangles whose sides have exactly half the size of the original triangle and which touch at three points which are common vertices of two contiguous triangles. Now we follow the same procedure with the three remaining triangles and repeat the basic step as often as desired. That is, we start with one triangle and then produce 3, 9, 27, 81, 243, . . . triangles, each of which is an exact scaled down version of the triangles in the preceeding step.

Concepts Used:

+ Data structures for representing 3D vertices
+ Tetrahedron sub-division using mid-points

Algorithm:

+ Input: Four 3D vertices of tetrahedron, no. of divisions
+ Recursively sub-divide the each triangle by finding the mid-point

Pseudo Code

1. Define and initializes the array to hold the vertices as follows:

GLfloat v[3][3]={{-1.0,-0.5,0.0},{1.0,-0.5,0.0},{0.0,1.0,0.0}, {0.0,0.0,1.0}};

2. Define Triangle function that uses points in three dimensions to display one triangle
Use glBegin(GL_POLYGON)
glVertex3fv(a)
glVertex3fv(b)
glVertex3fv(c)
glEnd();

3. Subdivide a tetrahedron using divide_triangle function
i) if no of subdivision(m) > 0 means perform following functions
ii) Compute six midpoints using for loop.
iii) Create 4 tetrahedrons by calling divide_tetra function
iv) Else draw triangle at end of recursion.

4. Define tetrahedron function to apply triangle subdivision to faces of tetrahedron by Calling the function divide_tetra

5. Define display function to clear the color buffer and to call tetrahedron function.

6. Define main function to create window and to call display function.

CODE:

```
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
typedef GLfloat point[3];
point v[]={{-1.0,-0.5,0.0},{1.0,-0.5,0.0},{0.0,1.0,0.0}, {0.0,0.0,1.0}};
GLfloat colors[4][3]={{1.0,0.0,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0},{1.0,1.0,0.0}};
int n;
void triangle(point a,point b,point c)
{
        glBegin(GL_POLYGON);
        glVertex3fv(a);
        glVertex3fv(b);
        glVertex3fv(c);
        glEnd();
}

void tetra(point a,point b,point c,point d)
{
        glColor3fv(colors[0]);
        triangle(a,b,c);
        glColor3fv(colors[1]);
```

```
        triangle(a,c,d);
        glColor3fv(colors[2]);
        triangle(a,d,b);
        glColor3fv(colors[3]);
        triangle(b,d,c);
}

void divide_tetra(point a,point b,point c,point d,int m)
{
        point mid[6];
        int j;
        if(m>0)
        {
                for(j=0;j<3;j++)
                {
                        mid[0][j]=(a[j]+b[j])/2.0;
                        mid[1][j]=(a[j]+c[j])/2.0;
                        mid[2][j]=(a[j]+d[j])/2.0;
                        mid[3][j]=(b[j]+c[j])/2.0;
                        mid[4][j]=(c[j]+d[j])/2.0;
                        mid[5][j]=(b[j]+d[j])/2.0;
                }
                divide_tetra(a,mid[0],mid[1],mid[2],m-1);
                divide_tetra(mid[0],b,mid[3],mid[5],m-1);


                divide_tetra(mid[1],mid[3],c,mid[4],m-1);
                divide_tetra(mid[2],mid[5],mid[4],d,m-1);
        }
        else
        tetra(a,b,c,d);
}
void display()
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glClearColor(1.0,1.0,1.0,1.0);
        divide_tetra(v[0],v[1],v[2],v[3],n);
        glFlush();
}

void myReshape(int w,int h)
{
        glViewport(0,0,w,h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
```

```
        if(w<=h)
        glOrtho(-1.0,1.0,-1.0*((GLfloat)h/(GLfloat)w), 1.0*((GLfloat)h/(GLfloat)w),-1.0,1.0);
        else
        glOrtho(-1.0*((GLfloat)w/(GLfloat)h),1.0*((GLfloat)w/(GLfloat)h),-1.0,1.0,-1.0,1.0);
        glMatrixMode(GL_MODELVIEW);
        glutPostRedisplay();
}

void main(int argc,char ** argv)
{
        printf( "No of Division?: ");
        scanf("%d",&n);
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
        glutInitWindowSize(500,500);
        glutCreateWindow( "3D gasket" );
        glutDisplayFunc(display);
        glutReshapeFunc(myReshape);
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
}
```

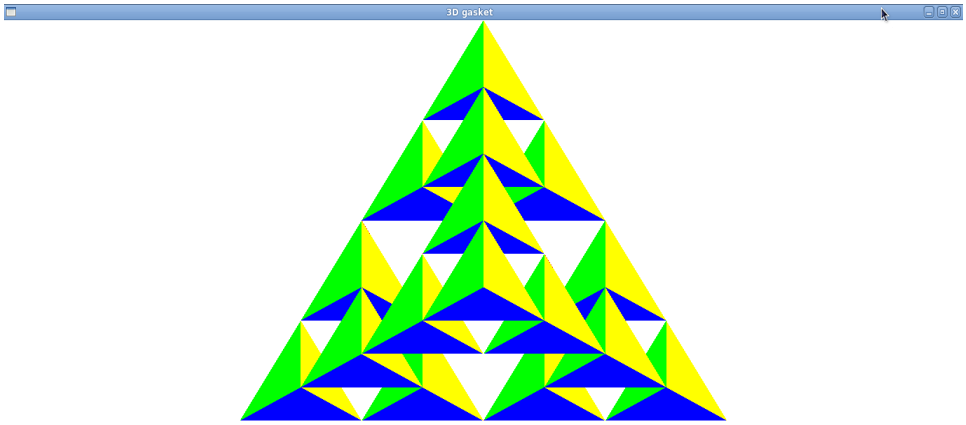
RUN:

```
gcc 7.c  -lglut  -lGL  -lGLUT

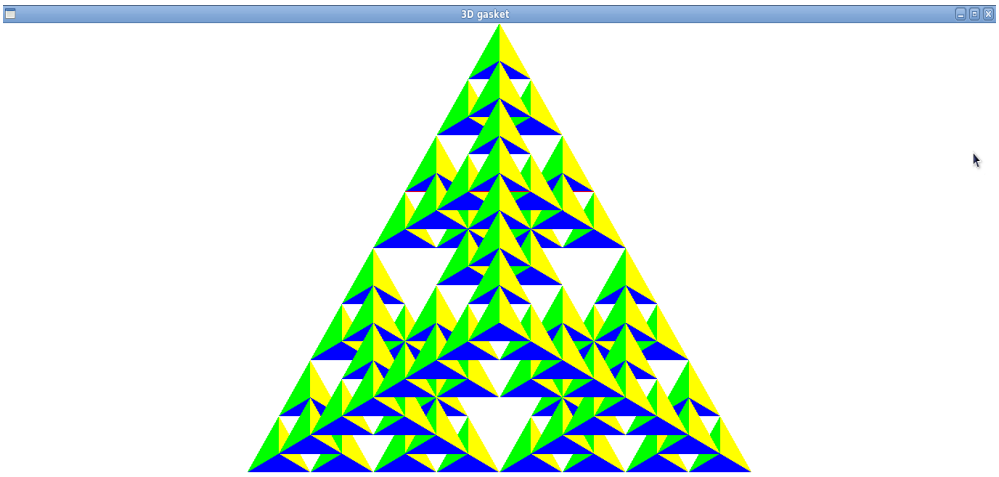
./a.out
```

**** Output ***

No. of Divisions ? 2



No. of Divisions ? 3



Lab Program –8: Bezier Curve

Develop a menu driven program to animate a flag using Bezier Curve algorithm.

PREAMBLE

Bezier curve is discovered by the French engineer Pierre Bezier. These curves can be generated under the control of other points. Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathematically as

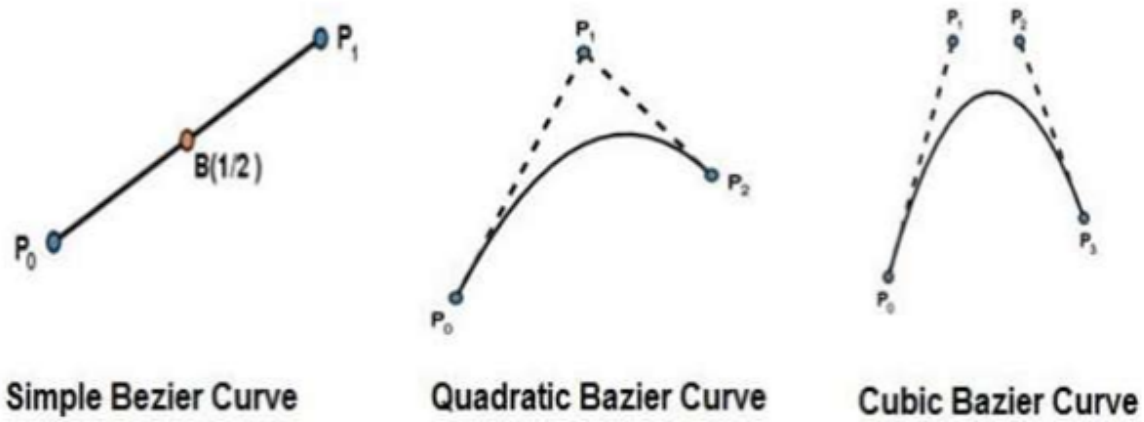
$$\sum_{k=0}^{n} P_i B_i^n(t)$$

Where pi is the set of points and Bni(t) represents the Bernstein polynomials which are given by

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i$$

Where n is the polynomial degree, i is the index, and t is the variable.

The simplest Bezier curve is the straight line from the point P0 to P1. A quadratic Bezier curve is determined by three control points. A cubic Bezier curve is determined by four control points.



Simple Bezier Curve Quadratic Bazier Curve Cubic Bazier Curve

Bezier curves generally follow the shape of the control polygon, which consists of the segments joining the control points and always pass through the first and last control points. They are contained in the convex hull of their defining control points. The degree of the polynomial defining the curve segment is one less that the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial. A Bezier curve generally follows the shape of the defining polygon. The

direction of the tangent vector at the end points is same as that of the vector determined by first and last segments. The convex hull property for a Bezier
Curve ensures that the polynomial smoothly follows the control points. No straight line intersects a Bezier curve more times than it intersects its control polygon. They are invariant under an affine transformation. Bezier curves exhibit global control means moving a control point alters the shape of the whole curve. A given Bezier curve can be subdivided at a point t=t0 into two Bezier segments which join together at the point corresponding to the parameter value t=t0.

CODE:

```
// Program to generate menu
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<GL/glut.h>
#define PI 3.1416
GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = 0.0, xwcMax = 130.0;
GLfloat ywcMin = 0.0, ywcMax = 130.0;
int size,submenu;
GLint nCtrlPts = 4, nBezCurvePts =20;
static float theta = 0;

struct wcPt3D
  {
        GLfloat x;
        GLfloat y;
        GLfloat z;
  };
wcPt3D ctrlPts[4] = {
          {20, 100, 0},
         {30, 110, 0},
         {50, 90, 0},
        {60, 100, 0}};

typedef struct wcPt3D cp;
void bino(GLint n, GLint *C)
  {
    GLint k, j;
    for(k=0;k<=n;k++)
    {
        C[k]=1;
        for(j=n;j>=k+1; j--)
            C[k]*=j;
        for(j=n-k;j>=2;j--)
```

```
                C[k]/=j;
        }
    }
void computeBezPt(GLfloat u, cp *bezPt, GLint nCtrlPts, cp *ctrlPts, GLint *C)
{
    GLint k, n=nCtrlPts-1;
    GLfloat bezBlendFcn;
    bezPt ->x =bezPt ->y = bezPt->z=0.0;
    for(k=0; k< nCtrlPts; k++)
    {
        bezBlendFcn = C[k] * pow(u, k) * pow( 1-u, n-k);
        bezPt ->x += ctrlPts[k].x * bezBlendFcn;
        bezPt ->y += ctrlPts[k].y * bezBlendFcn;
        bezPt ->z += ctrlPts[k].z * bezBlendFcn;
    }
}

void bezier(cp *ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)
  {
    cp bezCurvePt;
    GLfloat u;
    GLint *C, k;
    C= new GLint[nCtrlPts];
    bino(nCtrlPts-1, C);
    glBegin(GL_LINE_STRIP);
        for(k=0; k<=nBezCurvePts; k++)
        {
            u=GLfloat(k)/GLfloat(nBezCurvePts);
            computeBezPt(u, &bezCurvePt, nCtrlPts, ctrlPts, C);
            glVertex2f(bezCurvePt.x, bezCurvePt.y);
        }
    glEnd();
    delete[]C;
  }

 void displayFcn()
  {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glPointSize(5);
    glPushMatrix();
    glLineWidth(5);
    glColor3f(255/255, 153/255.0, 51/255.0); //Indian flag: Orange color code
    for(int i=0;i<8;i++)
    {
```

```
            glTranslatef(0, -0.8, 0);
            bezier(ctrlPts, nCtrlPts, nBezCurvePts);
        }

        glColor3f(1, 1, 1);             //Indian flag: white color code
        for(int i=0;i<8;i++)
        {
            glTranslatef(0, -0.8, 0);
            bezier(ctrlPts, nCtrlPts, nBezCurvePts);
        }
        glColor3f(19/255.0, 136/255.0, 8/255.0);        //Indian flag: green color code
        for(int i=0;i<8;i++)
        {
            glTranslatef(0, -0.8, 0);
            bezier(ctrlPts, nCtrlPts, nBezCurvePts);
        }
        glPopMatrix();
        glColor3f(0.7, 0.5,0.3);
        glLineWidth(5);
        glBegin(GL_LINES);
            glVertex2f(20,100);
            glVertex2f(20,40);
        glEnd();
        glFlush();
        glutPostRedisplay();
        glutSwapBuffers();
    }

void menufunc(int n)
{
            switch(n)
            {
            case 1:
                        ctrlPts[1].x +=10*sin(theta * PI/180.0);
                        ctrlPts[1].y +=5*sin(theta * PI/180.0);
                        ctrlPts[2].x -= 10*sin((theta+30) * PI/180.0);
                        ctrlPts[2].y -= 10*sin((theta+30) * PI/180.0);
                        ctrlPts[3].x -= 4*sin((theta) * PI/180.0);
                        ctrlPts[3].y += sin((theta-30) * PI/180.0);
                        theta+=0.1;

                       break;
            case 2:
                        ctrlPts[1].x -=10*sin(theta * PI/180.0);
                        ctrlPts[1].y -=5*sin(theta * PI/180.0);
```

```
                    ctrlPts[2].x +=10*sin((theta+30) * PI/180.0);
                    ctrlPts[2].y +=10*sin((theta+30) * PI/180.0);
                    ctrlPts[3].x +=4*sin((theta) * PI/180.0);
                    ctrlPts[3].y -=sin((theta-30) * PI/180.0);
                    theta-=0.1;

                   break;

        case 3: exit(0);


        }
        //glutPostRedisplay();
}

void winReshapeFun(GLint newWidth, GLint newHeight)
  {
     glViewport(0, 0, newWidth, newHeight);
     glMatrixMode(GL_PROJECTION);
     glLoadIdentity();
     gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
     glClear(GL_COLOR_BUFFER_BIT);
  }

int main(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowPosition(50, 50);
        glutInitWindowSize(winWidth, winHeight);
        glutCreateWindow("Bezier Curve");
        submenu=glutCreateMenu(menufunc);
        glutCreateMenu(menufunc);
        glutAddMenuEntry("Dwn-mov",1);
        glutAddMenuEntry("Up-mov",2);
        glutAddMenuEntry("Exit",3);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
        glutDisplayFunc(displayFcn);
         glutReshapeFunc(winReshapeFun);
        glutMainLoop();

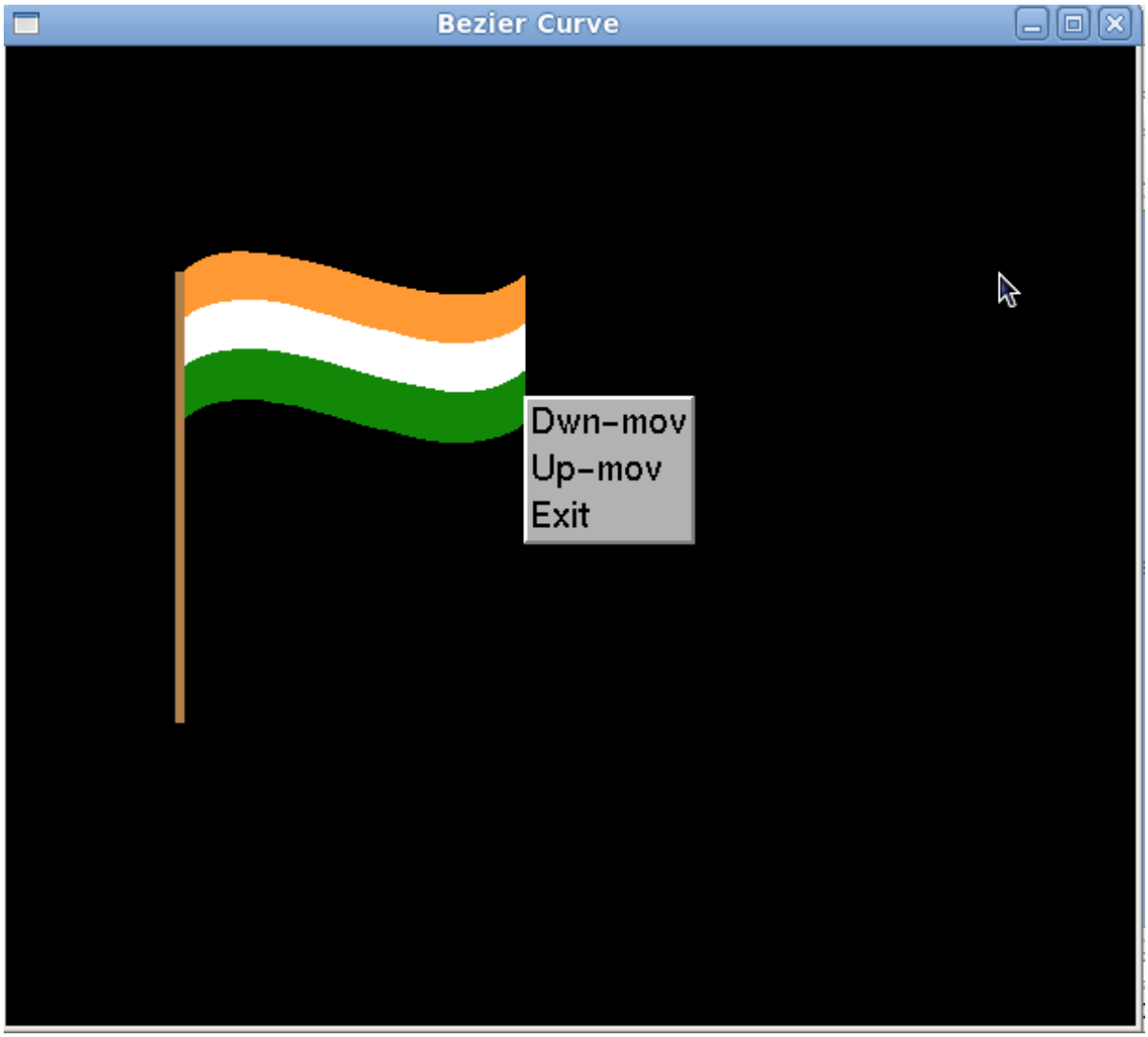
}
```

RUN:

gcc 8.cpp -lglut -lGL -lGLUT

./a.out

****** Output *****



Lab Program –9: Scan-line

Develop a menu driven program to fill the polygon using scan line algorithm.

PREAMBLE

The scan conversion algorithm works as follows

i. Intersect each scanline with all edges
ii. Sort intersections in x
iii. Calculate parity of intersections to determine in/out iv. Fill the "in" pixels

Special cases to be handled:
i. Horizontal edges should be excluded
ii. For vertices lying on scanlines,
i. count twice for a change in slope.
ii. Shorten edge by one scanline for no change in slope

. Coherence between scanlines tells us that
. Edges that intersect scanline y are likely to intersect y + 1
. X changes predictably from scanline y to y + 1

We have 2 data structures: Edge Table and Active Edge Table
. Traverse Edges to construct an Edge Table
. Eliminate horizontal edges
. Add edge to linked-list for the scan line corresponding to the lower vertex.

Store the following:
. yupper: last scanline to consider
. xlower: starting x coordinate for edge
. 1/m: for incrementing x; compute
. Construct Active Edge Table during scan conversion. AEL is a linked list of active edges on the current scanline, y. Each active edge line has the following information
. yupper: last scanline to consider
. xlower: edge's intersection with current y
. 1/m: x increment

The active edges are kept sorted by x Concepts Used:
+ Use the straight line equation y2-y1=m(x2-x1) to compute the x values corresponding to lines increment in y value.
+ Determine which are the left edge and right edges for the the closed polygon.for each value of y within the polygon.
+ Fill the polygon for each value of y within the polygon from x=left edge to x=right edge.

Algorithm:

1. Set y to the smallest y coordinate that has an entry in the ET;
i.e, y for the first nonempty bucket.

2. Initialize the AET to be empty.

3. Repeat until the AET and ET are empty:
3.1 Move from ET bucket y to the AET those edges whose y_min = y (entering edges).
3.2 Remove from the AET those entries for which y = y_max (edges not involved in the next scanline), the sort the AET on x (made easier because ET is presorted).
3.3 Fill in desired pixel values on scanline y by using pairs of x coordinates from AET.
3.4 Increment y by 1 (to the coordinate of the next scanline).
3.5 For each nonvertical edge remaining in the AET, update x for the new y.

Extensions:

1. Multiple overlapping polygons - priorities
2. Color, patterns Z for visibility

// Function scanfill
Inputs : vertices of the polygon.
Output : filled polygon.

Procressing :
1. Initialize array LE to 500 and RE to 0 for all values of y(0-->499)
2. Call function EDGEDETECT for each edge of the polygon one by one to set the value of x for each value of y within that line.
3. For each value of y in the screen draw the pixels for every value of x provided . It is greater than right edge value of x.

//function Display
Inputs : Globally defined vertices of Polygon
Output : Filled polygon display

Processing :
1. Draw the polygon using LINE,LOOP
2. Fill the polygon using SCANFILL

//Function EDGEDETECT:

Inputs :
1. End co-ordinates of edge
2. Adress of LE and RE

Output :
The updated value of x for left edge of the polygon and the right edge of the polygon.

Processing :
1. Find the inverse of the slope.
2. Starting from the lesser integer value of y to the greater integer value of y.
3. Compute the value of x for left edge and right edge and update the value of x for both the edges.

CODE:

```
#define BLACK 0
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
int submenu;
float x1,x2,x3,x4,y1,y2,y3,y4;
void edgedetect(float x1,float y1,float x2,float y2,int *le,int *re)
{
        float mx,x,temp;
        int i;
        if((y2-y1)<0)
        {
                temp=y1;y1=y2;y2=temp;
                temp=x1;x1=x2;x2=temp;
        }
        if((y2-y1)!=0)
        mx=(x2-x1)/(y2-y1);
        else
        mx=x2-x1;
        x=x1;
        for(i=y1;i<=y2;i++)
        {
```

```
                if(x<(float)le[i])
                le[i]=(int)x;
                if(x>(float)re[i])
                re[i]=(int)x;
                x+=mx;
        }
}
void draw_pixel(int x,int y,int value)
{
        //glColor3f(1.0,1.0,0.0);
        glBegin(GL_POINTS);
        glVertex2i(x,y);
        glEnd();
}
void scanfill(float x1,float y1,float x2,float y2,float x3,float y3,float x4,float y4)
{
        int le[500],re[500];
        int i,y;
        for(i=0;i<500;i++)
        {
                le[i]=500;
                re[i]=0;
        }
        edgedetect(x1,y1,x2,y2,le,re);
        edgedetect(x2,y2,x3,y3,le,re);
        edgedetect(x3,y3,x4,y4,le,re);
        edgedetect(x4,y4,x1,y1,le,re);
        for(y=0;y<500;y++)
        {
                if(le[y]<=re[y])
                for(i=(int)le[y];i<(int)re[y];i++)
                draw_pixel(i,y,BLACK);
        }
```

```
}
void display()
{
        x1=200.0;y1=200.0;x2=100.0;y2=300.0;x3=200.0;y3=400.0;x4=300.0;y4=300.0;
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_LINE_LOOP);
        glVertex2f(x1,y1);
        glVertex2f(x2,y2);
        glVertex2f(x3,y3);
        glVertex2f(x4,y4);
        glEnd();
        scanfill(x1,y1,x2,y2,x3,y3,x4,y4);
        glFlush();
}
void myinit()
{
        glClearColor(1.0,1.0,1.0,1.0);
        //glColor3f(1.0,0.0,0.0);
        glPointSize(1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0,499.0,0.0,499.0);
}
void menufunc(int n)
{
        switch(n)
        {
        case 1: glColor3f(1.0, 0.0, 0.0);

                break;
        case 2:
                glColor3f(0.0, 1.0, 0.0);
```

```
                break;

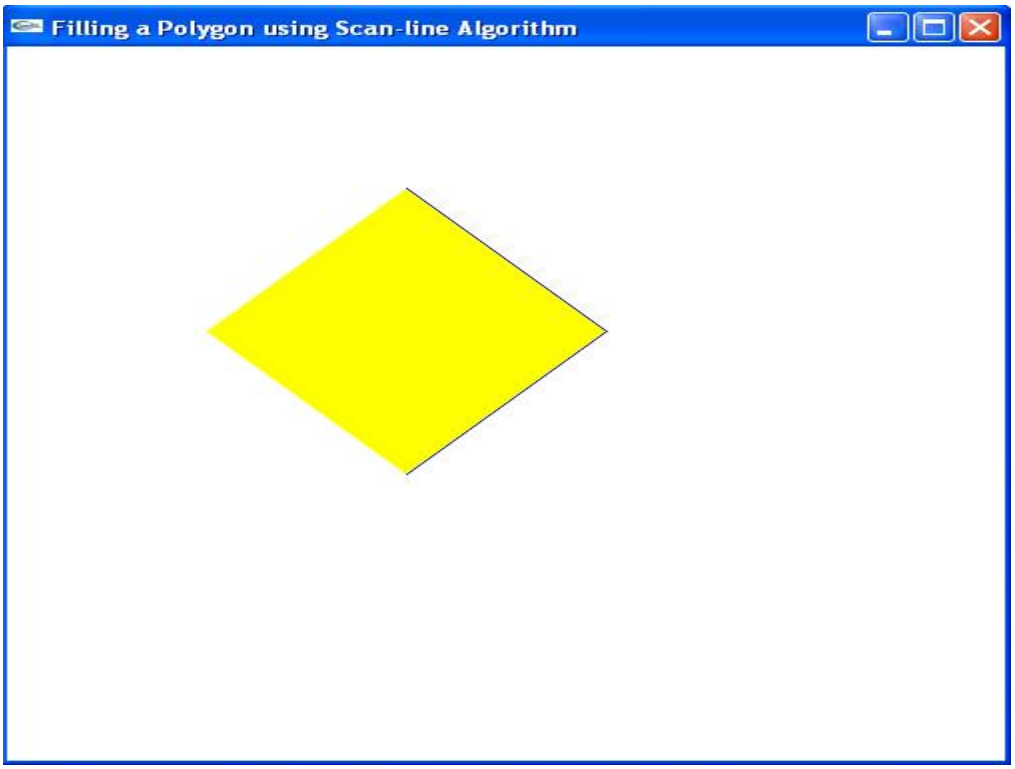
        case 3: glColor3f(0.0, 0.0, 1.0);
        }
}

int main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("Filling a Polygon using Scan-line Algorithm");
        submenu=glutCreateMenu(menufunc);
        glutCreateMenu(menufunc);
        glutAddMenuEntry("RED",1);
        glutAddMenuEntry("GREEN",2);
        glutAddMenuEntry("BLUE",3);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
        return 0;
}
```

RUN:

cc 9.c -lGL -lGLU -lglut
./a.out

*** Output ***



PART B

Develop a suitable Graphics package to implement the skills learnt in the theory and the exercises indicated in PART A. Use the OpenGL.

Criteria for CG Project

Students per batch should be THREE or less.

One Three-Dimensional OpenGL Graphics Project using features from at least THREE

CATEGORIES listed below:

Category I

. Input and Interaction
. Menus, Display Lists

Category II

. Transformations
. Camera Movement

Category III

. Coloring
. Texturing
. Lighting/ Shading

Category IV

. Animation
. Hidden Surface Removal

Viva questions

1. Explain all the OpenGL functions used in this program?
2. What is the principle of Sierpinski gasket?
3. Difference between additive and subtractive color?
4. What is the Graphics Architecture used in OpenGL?
5. What is Rasterisation?
6. Explain sierpinski Gasket (using points).
7. Explain sierpinski Gasket (using polye)
8. Difference between 2D Tetrohedron and 3D Tetrahedron.
9. What do you mean by clipping?
10. How is Liang-Barsky clipping different from CohenSutherland Clipping Algorithm?
11. How do you set the color attributes in Opengl?
12. What is the command for clearscreen?
13. What is Event callback function?
14. Explain Liang-Barsky line clipping algorithm?
15. Explain window to viewpoint mapping?
16. What are vertex arrays?
17. How are the faces of the color cube modeled?
18. How do you consider the inward and outward pointing of the faces?
19. Explain the OpenGL function used to rotate the color cube?
20. What is the difference between 2D and 3D orthographic projection statements?
21. Explain the Inward and Outward pointing face?
22. Explain the data structure for object representation?
23. Explain the vertex list representation of a cube?
24. What is transformation?
25. Explain the OpenGL functions used for translation, rotation and scaling?
26. What is the order of transformation?
27. State the difference between modelview and projection?
28. What is Homogeneous-coordinate representation?
29. Define the rotation matrix and object matrix.
30. Explain the procedure to obtain the resultant matrix using rotation matrix and object matrix.
31. What is the principle of Cohen-Sutherland Algorithm?
32. State the advantages and disadvantages of Cohen-Sutherland Algorithm?
33. What is an outcode?
34. What is Synthetic Camera Model?
35. What are the Camera Specifications?
36. Explain the cases of outcodes in Cohen-Sutherland algorithm.
37. Explain the cohen-sutherland line clipping algorithm
38. Mention the difference between Liang-Barsky and Cohen-Sutherland line clipping algorithm.
39. Explain about gluLookAt(...), glFrustum(...), gluPerspective?
40. Explain the different types of projections?
41. Explain Z-buffer algorithm

42. What is antialiasing?
43. What is Center Of Projection(COP), Direction Of Projection(DOP)?
44. What is midpoint circle drawing algorithm
45. How do you get the equation d+=2x+3, d+=2(x-y)+5
46. What is gluOrtho2D function
47. Explain plot pixel function
48. Why do we use GLFlush function in Display
49. Explain Specular, Diffuse and Translucent surfaces.
50. What is ambient light?
51. What is umbra, penumbra?
52. Explain Phong lighting model.
53. Explain glLightfv(...), glMaterialfv(...).
54. What is glutSolidCube function ? what are its Parameters
55. What are the parameters to glScale function
56. Explain Push & Pop matrix Functions
57. What is Materialfv function & its Parameters
58. Explain GLULookAt Function
59. Explain the keyboard and mouse events used in the program?
60. What is Hidden surface Removal? How do you achieve this in OpenGL?
61. What are the functions for creating Menus in OpenGL?
62. Explain about fonts in GLUT?
63. Explain about glutPostRedisplay()?
64. Explain how the cube is constructed
65. Explain rotate function
66. What is GLFrustum function what are its Parameters
67. What is viewport
68. What is glutKeyboard Function what are its Parameters
69. Explain scanline filling algorithm?
70. What is AspectRatio,Viewport?
71. How do you use timer?
72. What are the different frames in OpenGL?
73. What is fragment processing?
74. Explain Polygon Filling algorithm
75. What is slope
76. How the edges of polygon are detected
77. Why you use GL_PROJECTION in MatrixMode Function
78. What is dx & dy
79. What is maxx & maxy
80. What is glutPostRedisplay
81. Why do we use glutMainLoop function
82. What do you mean by GL_LINE_LOOP in GL_Begin function
83. Define Computer Graphics.
84. Explain any 3 uses of computer graphics applications.
85. What are the advantages of DDA algorithm?
86. What are the disadvantages of DDA algorithm?

87. Define Scan-line Polygon fill algorithm.
88. What are Inside-Outside tests?
89. Define Boundary-Fill algorithm.
90. Define Flood-Fill algorithm.
91. Define attribute parameter. Give examples.
92. What is a line width? What is the command used to draw the thickness of lines.
93. What are the three types of thick lines? Define.
94. What are the attribute commands for a line color?
95. What is color table? List the color codes.
96. What is a marker symbol and where it is used?
97. Discuss about inquiry functions.
98. Define translation and translation vector.
99. Define window and view port.
100. Define viewing transformation.
101. Give the equation for window to viewport transformation.
102. Define view up vector.
103. What is meant by clipping? Where it happens?
104. What is point clipping and what are its inequalities?
105. What is line clipping and what are their parametric representations?
106. How is translation applied?
107. What is referred to as rotation?
108. Write down the rotation equation and rotation matrix.
109. Write the matrix representation for scaling, translation and rotation.
110. Draw the block diagram for 2D viewing transformation pipeline.
111. Mention the equation for homogeneous transformation.
112. What is known as composition of matrix?
113. Write the composition transformation matrix for scaling, translation and Rotation.
114. Discuss about the general pivot point rotation?
115. Discuss about the general fixed point scaling.
116. Explain window, view port and window - to - view port transformation
117. Mention the three raster functions available in graphics packages.
118. What is known as region codes?
119. Why Cohen Sutherland line clipping is popular?
120. Mention the point clipping condition for the liang-barsky line clipping.
121. What is called as an exterior clipping?
122. How is the region code bit values determined?
123. Why liang-barsky line clipping is more efficient than Cohen Sutherland line Clipping?
124. Differentiate uniform and differential scaling
125. Explain soft fill procedures.
126. Explain the three primary color used in graphics
127. Explain in detail about color and grey scale levels?
128. Explain color and grey scale levels.
129. Explain the area fill attributes and character attributes.
130. Explain character attributes in detail.
131. Briefly discuss about basic transformations.

132. Explain matrix representations.
133. Discuss about composite transformations.
134. Explain about reflection and shear.
135. Explain the following transformation with the matrix representations.
Give suitable diagram for illustration translation .ii scaling.iii rotation.
136. How the rotation of an object about the pivot point is performed?
137. How window-to-viewport coordinate transformation happens.
138. Explain clipping with its operation in detail.
139. Explain Cohen- Sutherland line clipping.
140. Discuss the logical classifications of input devices.
141. Explain the details of 2d viewing transformation pipeline.
142. Explain point, line, curve, text, exterior clipping?
143. Discuss the properties of light.
144. Define chromaticity, complementary colors, color gamut and primary colors.
145. What is color model?
146. Define hue, saturation and value.
147. Explain XYZ color model.
148. Explain RGB color model.
149. Explain YIQ color model.
150. Explain CMY color model.
151. Explain HSV color model.
152. Give the procedure to convert HSV & RGB color model.
153. What is the use of chromaticity diagram?
154. What is illumination model?
155. What are the basic illumination models?
156. What is called as lightness?
157. Explain the conversion of CMY to RGB representation.
158. What is animation?
159. Define Morphing.
160. What are the steps involved in designing an animation sequence?
161. How to draw dots using OPENGL?
162. How to draw lines using OPENGL?
163. How to draw convex polygons using OPENGL?
164. What is the command used in OPENGL to clear the screen?
165. What are the various OPENGL data types?

Viva questions and answers

1. What is Computer Graphics?
Answer: Computer graphics are graphics created using computers and, more generally, the
representation and manipulation of image data by a computer.

2. What is OpenGL?
Answer: OpenGL is the most extensively documented 3D graphics API (Application Program Interface) to date. It is used to create Graphics.

3. What is GLUT?
Answer: The OpenGL Utility Toolkit (GLUT) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system.

4. What are the applications of Computer Graphics?
Answer: Gaming Industry, Animation Industry and Medical Image Processing Industries. The sum total of these industries is a Multi Billion Dollar Market. Jobs will continue to increase in this arena in the future.

5. Explain in breif 3D Sierpinski gasket?
Answer: The Sierpinski triangle (also with the original orthography Sierpinski), also called the Sierpinski gasket or the Sierpinski Sieve, is a fractal named after the Polish mathemati-
cian Waclaw Sierpinski who described it in 1915. Originally constructed as a curve, this is one of the basic examples of self-similar sets, i.e. it is a mathematically generated pattern
that can be reproducible at any magnification or reduction.

6. What is Liang-Barsky line clipping algorithm?
Answer: In computer graphics, the Liang-Barsky algorithm is a line clipping algorithm. The
Liang-Barsky algorithm uses the parametric equation of a line and inequalities describing the range of the clipping box to determine the intersections between the line and the clipping box. With these intersections it knows which portion of the line should be drawn.

7. Explain in brief Cohen-Sutherland line-clipping algorithm?
Answer: The Cohen-Sutherland line clipping algorithm quickly detects and dispenses with two common and trivial cases. To clip a line, we need to consider only its endpoints. If both endpoints of a line lie inside the window, the entire line lies inside the window. It is trivially accepted and needs no clipping. On the other hand, if both endpoints of a line lie entirely to one side of the window, the line must lie entirely outside of the window. It is trivially rejected and needs to be neither clipped nor displayed.

8. Explain in brief scan-line area filling algorithm?
Answer: The scanline fill algorithm is an ingenious way of filling in irregular polygons. The algorithm begins with a set of points. Each point is connected to the next, and the line between them is considered to be an edge of the polygon. The points of each edge are adjusted to ensure that the point with the smaller y value appears first. Next, a data structure is created that contains a list of edges that begin on each scanline of the image. The program progresses from the first scanline upward. For each line, any pixels that contain an intersection between this scanline and an edge of the polygon are filled in. Then, the algorithm progresses along the scanline, turning on when it reaches a polygon pixel and turning off when it reaches another one, all the way across the scanline.

9. Explain Midpoint Line algorithm
Answer: The Midpoint line algorithm is an algorithm which determines which points in an n-dimensional raster should be plotted in order to form a close approximation to a straight
line between two given points. It is commonly used to draw lines on a computer screen, as it uses only integer addition, subtraction and bit shifting, all of which are very cheap operations in standard computer architectures.

10. What is a Pixel?
Answer: In digital imaging, a pixel (or picture element) is a single point in a raster image. The Pixel is the smallest addressable screen element; it is the smallest unit of picture which
can be controlled. Each Pixel has its address. The address of Pixels corresponds to its coordinate. Pixels are normally arranged in a 2-dimensional grid, and are often represented
using dots or squares.

11. What is Graphical User Interface?
Answer: A graphical user interface (GUI) is a type of user interface item that allows people
to interact with programs in more ways than typing such as computers; hand-held devices such as MP3 Players, Portable Media Players or Gaming devices; household appliances and
once equipment with images rather than text commands.

12. What is the general form of an OpenGL program?
Answer: There are no hard and fast rules. The following pseudocode is generally recognized
as good OpenGL form.

```
program_entrypoint
{
// Determine which depth or pixel format should be used.
// Create a window with the desired format.
```

```
// Create a rendering context and make it current with the window.
// Set up initial OpenGL state.
// Set up callback routines for window resize and window refresh.
}
handle_resize
{
glViewport(...);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
// Set projection transform with glOrtho, glFrustum, gluOrtho2D, gluPerspective, etc.
}
handle_refresh
{
glClear(...);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
// Set view transform with gluLookAt or equivalent
// For each object (i) in the scene that needs to be rendered:
// Push relevant stacks, e.g., glPushMatrix, glPushAttrib.
// Set OpenGL state speci_c to object (i).
// Set model transform for object (i) using glTranslatef, glScalef, glRotatef, and/or equiv-
alent.
// Issue rendering commands for object (i).
// Pop relevant stacks, (e.g., glPopMatrix, glPopAttrib.)
// End for loop.
// Swap bu_ers.
}
```

13. What support for OpenGL does Open,Net,FreeBSD or Linux provide?
Answer: The X Windows implementation, XFree86 4.0, includes support for OpenGL using
Mesa or the OpenGL Sample Implementation. XFree86 is released under the XFree86 license. http://www.xfree86.org/

14. What is the AUX library?
Answer: The AUX library was developed by SGI early in OpenGL's life to ease creation of small OpenGL demonstration programs. It's currently neither supported nor maintained. Developing OpenGL programs using AUX is strongly discouraged. Use the GLUT instead. It's more _exible and powerful and is available on a wide range of platforms. Very important: Don't use AUX. Use GLUT instead.

15. How does the camera work in OpenGL?
Answer: As far as OpenGL is concerned, there is no camera. More specifically, the camera
is always located at the eye space coordinate (0., 0., 0.). To give the appearance of

moving
the camera, your OpenGL application must move the scene with the inverse of the camera transformation.

16. How do I implement a zoom operation?
Answer: A simple method for zooming is to use a uniform scale on the ModelView matrix. However, this often results in clipping by the zNear and zFar clipping planes if the model is scaled too large. A better method is to restrict the width and height of the view volume in the Projection matrix.

17. What are OpenGL coordinate units?
Answer: Depending on the contents of your geometry database, it may be convenient for your application to treat one OpenGL coordinate unit as being equal to one millimeter or one parsec or anything in between (or larger or smaller). OpenGL also lets you specify your
geometry with coordinates of differing values. For example, you may find it convenient to model an airplane's controls in centimeters, its fuselage in meters, and a world to fly around
in kilometers. OpenGL's ModelView matrix can then scale these different coordinate systems into the same eye coordinate space. It's the application's responsibility to ensure that the Projection and ModelView matrices are constructed to provide an image that keeps the viewer at an appropriate distance, with an appropriate field of view, and keeps the zNear
and zFar clipping planes at an appropriate range. An application that displays molecules in micron scale, for example, would probably not want to place the viewer at a distance of 10 feet with a 60 degree field of view.

18. What is Microsoft Visual Studio?
Answer: Microsoft Visual Studio is an integrated development environment (IDE) for developing windows applications. It is the most popular IDE for developing windows applications or windows based software.

19. What does the .gl or .GL _Je format have to do with OpenGL?
Answer: .gl _Jes have nothing to do with OpenGL, but are sometimes confused with it. .gl is a file format for images, which has no relationship to OpenGL.

20. Who needs to license OpenGL? Who doesn't? Is OpenGL free software?
Answer: Companies which will be creating or selling binaries of the OpenGL library will need to license OpenGL. Typical examples of licensees include hardware vendors, such as Digital Equipment, and IBM who would distribute OpenGL with the system software on their workstations or PCs. Also, some software vendors, such as Portable Graphics and Template Graphics, have a business in creating and distributing versions of OpenGL, and they need to license OpenGL. Applications developers do NOT need to license OpenGL. If a developer wants to use OpenGL that developer needs to obtain copies of a

linkable OpenGL library for a particular machine. Those OpenGL libraries may be bundled in with the development and/or run-time options or may be purchased from a third-party software vendor, without licensing the source code or use of the OpenGLtrademark.

21. How do we make shadows in OpenGL?
Answer: There are no individual routines to control neither shadows nor an OpenGL state for shadows. However, code can be written to render shadows.

22. What is the use of Glutinit?
Answer: void glutInit(int *argcp, char **argv);
glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized.

23. Describe the usage of glutInitWindowSize and glutInitWindowPosition?
Answer: void glutInitWindowSize(int width, int height);
void glutInitWindowPosition(int x, int y);
Windows created by glutCreateWindow will be requested to be created with the current initial window position and size. The intent of the initial window position and size values is to provide a suggestion to the window system for a window's initial size and position. The window system is not obligated to use this information. Therefore, GLUT programs should not assume the window was created at the speci_ed size or position. A GLUT program should use the window's reshape callback to determine the true size of the window.

24. Describe the usage of glutMainLoop?
Answer: 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.