CITY ENGINEERING COLLEGE

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI

VI SEMESTER, CSE

LAB MANUAL

COMPUTER GRAPHICS AND VISUALIZATION

Subject Code: 18CSL67

Name: _____

Branch: _____

Semester: _____

Faculty In-charge

Prof. Ramesh B
Asst. Professor
Dept. of CSE, CEC

Prof. Nandini S B
Asst. Professor
Dept. of CSE, CEC

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CITY ENGINEERING COLLEGE, BENGALURU

Course Details

Course Name	Computer Graphics and Visualization
Course Code	18CSL67
Course Prerequisites	C Programming

Course Objectives

- Demonstrate simple algorithms using OpenGL Graphics primitive and attributes.
- Implementation of line drawing and clipping algorithms using OpenGL functions.
- Design and implementation of algorithms Geometric transformations on both 2D and 3D objects.

SYLLABUS

Subject Code: 18CSL67	**CIE Marks: 40**
No. of Practical Hrs/ Week: 04	**SEE Marks: 60**
Total number of Lab Contact Hrs: 36	**Exam Hrs: 60**

Description (if any):

Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal.

Programs List:

PART A

Design, develop, and implement the following programs using OpenGL API

1. Implement Brenham's line drawing algorithm for all types of slope.
2. Create and rotate a triangle about the origin and a fixed point.
3. Draw a colour cube and spin it using OpenGL transformation matrices.
4. Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.
5. Clip a lines using Cohen-Sutherland algorithm
6. 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.
7. 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.
8. Develop a menu driven program to animate a flag using Bezier Curve algorithm
9. Develop a menu driven program to fill the polygon using scan line algorithm

PART B MINI PROJECT

Student should develop mini project on the topics mentioned below or similar applications using OpenGL API. Consider all types of attributes like color, thickness, styles, font, background, speed etc., while doing mini project.

(**During the practical exam: the students should demonstrate and answer Viva-Voce**)

Sample Topics:

Simulation of concepts of OS, Data structures, algorithms etc.

Laboratory Outcomes:

The students should be able to:

- Apply the concepts of computer graphics.
- Implement computer graphics applications using OpenGL
- Animate real world problems using OpenGL

Conduct of Practical Examination:

- Experiment distribution

  For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.

  For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution *(Coursed to change in accordance with university regulations)*

  o) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks

  p) For laboratories having PART A and PART B

  i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks

  ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

CONTENTS

1.Introduction to OpenGL

OpenGL is a worldwide standard for 3D computer graphics programming. It's very widely used: in industry, in research laboratories, in computer games – and for teaching computer graphics.

OpenGL is a powerful, professional-level system, and it would take a manual much thicker than this one to describe all its facilities completely. We have selected a **subset** of OpenGL – a portion of OpenGL's functionality and sufficient to support its programming labs.

OpenGL has its origin in the IRIS Graphics Language invented by Silicon Graphics Inc. in the early 1990s as an API for programming their high-performance specialised graphics workstations. In 1992 Silicon Graphics created a new system based on IRIS GL called OpenGL.

OpenGL was specifically designed to be **platform-independent**, so it would work across a whole range of computer hardware – not just Silicon Graphics machines. The combination of OpenGL's power and portability led to its rapid acceptance as a **standard** for computer graphics programming.

OpenGL itself isn't a programming language, or a software library. It's the **specification** of an Application Programming Interface (API) for computer graphics programming. In other words, OpenGL defines a set of functions for doing computer graphics.

What exactly can OpenGL do? Here are some of its main features:

• It provides 3D geometric objects, such as lines, polygons, triangle meshes, spheres, cubes, quadric surfaces, NURBS curves and surfaces;

• It provides 3D modelling transformations, and viewing functions to create views of 3D scenes using the idea of a **virtual camera**;

• It supports high-quality rendering of scenes, including hidden-surface removal, multiple light sources, material types, transparency, textures, blending, fog;

• It provides display lists for creating graphics caches and hierarchical models. It also supports the interactive 'picking' of objects;

• It supports the manipulation of images as pixels, enabling frame-buffer effects such as anti-aliasing, motion blur, depth of field and soft shadows.

2. Software Installation

2.1 Steps to install Eclipse

Step 1: Download the Eclipse Installer (https://www.eclipse.org/).

Step 2: Start the Eclipse Installer executable

Step 3: Select the package to install

Step 4: Select your installation folder

Step 5**:** Launch Eclipse

2.2 Steps in install GCC:

Step 1: Download TDM GCC

Step 2: click the downloaded file

Step 3: click create

Step 4: select MinGW-w64/TDM64(32bit and 64 bit) and next

Step 5: you will get installation path click next

Step 6: select type of install as TDM-GCC Recommended, C/C++ and install.

Step 7: next and finish.

2.3 Steps in install freeGLUT:

Step 1: Download freeglut 3.0.0 for MinGW

Step 2: Unzip folders.

Step 3: click include --> GL --> copy all .h files

Step 4: go to c folder --> TDM-GCC-64 -->x86_64-w64-mingw32 --include --> GL --> paste it here.

Step 5: again from unzipped folder click lib --> x64 --> copy files

Step 6: go to c folder --> TDM-GCC-64 -->x86_64-w64-mingw32 --lib --> paste it here.

Step 7 : again from unzipped folder click bin --> copy .dll files

Step 8: c --> windows --> system32 --> paste here (click continue)

2.4 To create a New Project

Step 1: Once Eclipse is installed, we can make new C++ projects.

Step 2: From the start page, click --> New --> C Project" -> name the project -> GCC --->Finish

Step 3: right click on the project name --> new--> source file --> name the source file with extension --> Finish.

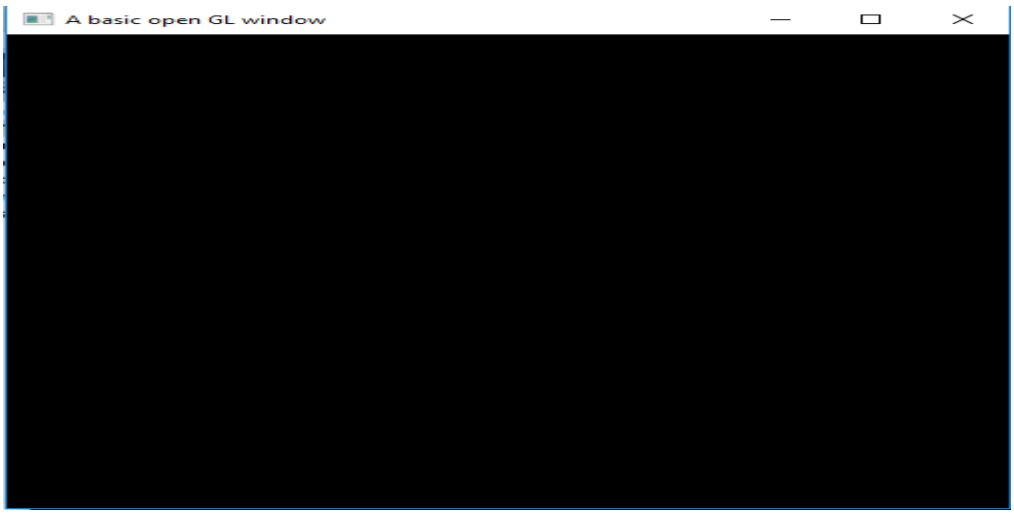
Step 4: Project --> properties --> c/c++ build --> settting --> GCC linker -- > libraries -->enter libraries(click add(+) and enter opengl32, glu32, freeglut) --> ok.

Sample Programs

1. A basic Open GL window

```
#include<GL/glut.h>
void display (void)
{
      glClearColor (0.0,0.0,0.0,1.0);
      glClear (GL_COLOR_BUFFER_BIT);
      glLoadIdentity ();
      glFlush ();
}
int main (intargc,char **argv)
{
      glutInit (&argc,argv);
      glutInitDisplayMode (GLUT_SINGLE);
      glutInitWindowSize (500,500);
      glutInitWindowPosition (100,100);
      glutCreateWindow ("A basic open GL window");
      glutDisplayFunc (display);
      glutMainLoop ();
      return 0;
}
```

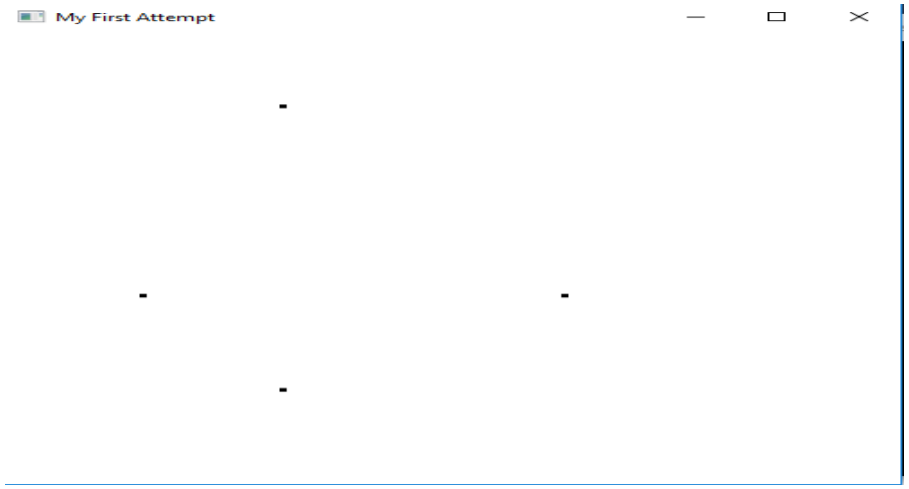
Output :



2. Program to draw/display points

```
#include<GL/glut.h>
#include<stdlib.h>
Void myInit(void)
{
      glClearColor(2.0,2.0,2.0,4.0);
      glColor3f(0.0f,0.0f,0.0f);
      glPointSize(4.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(0.0,640.0,0.0,480.0);
}
void display(void)
{
      glClear(GL_COLOR_BUFFER_BIT);
      glBegin(GL_POINTS);
      glVertex2i(100,200);
      glVertex2i(400,200);
      glVertex2i(200,100);
      glVertex2i(200,400);
      glEnd();
      glFlush();
}
void main(int argc, char** argv)
{
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
      glutInitWindowSize(500,500);
      glutInitWindowPosition(100,150);
      glutCreateWindow("My First Attempt");
      glutDisplayFunc(display);
      myInit();
      glutMainLoop();
}
```

Output:

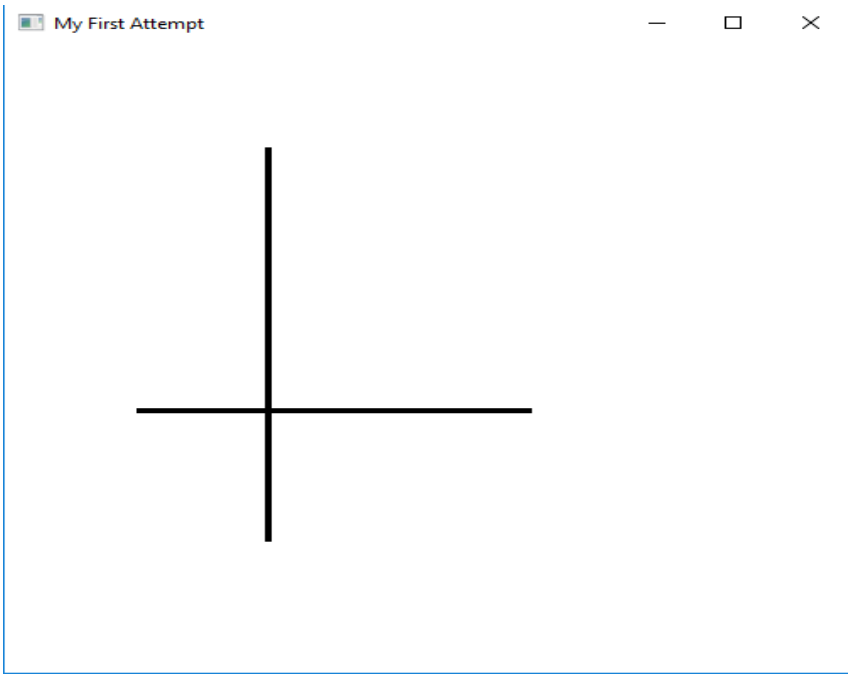


3.Program to implement horizontal and vertical lines

```
#include<GL/glut.h>
#include<stdlib.h>
void myInit(void)
{
      glClearColor(1.0,1.0,1.0,1.0);
      glColor3f(0.0f,0.0f,0.0f);
      glLineWidth(4.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(0.0,640.0,0.0,480.0);
}
void drawLineInt(GLint x1,GLint y1,GLint x2,GLint y2)
{
      glBegin(GL_LINES);
      glVertex2i(x1,y1);
      glVertex2i(x2,y2);
      glEnd();
```

```
}
void display(void)
{
      glClear(GL_COLOR_BUFFER_BIT);
      glBegin(GL_LINES);
      glVertex2i(100,200);
      glVertex2i(400,200);
      glVertex2i(200,100);
      glVertex2i(200,400);
      glEnd();
      glFlush();
}
void main(intargc, char** argv)
{
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
      glutInitWindowSize(500,500);
      glutInitWindowPosition(100,150);
      glutCreateWindow("My First Attempt");
      glutDisplayFunc(display);
      myInit();
      drawLineInt(100,200,40,60);
      glutMainLoop();
}
```

Output:



Lab Program - 1

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

```
#include<GL/glut.h>
#include<stdio.h>
int x1, y1, x2, y2;
void myInit()
{
	glClear(GL_COLOR_BUFFER_BIT);
	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;
```

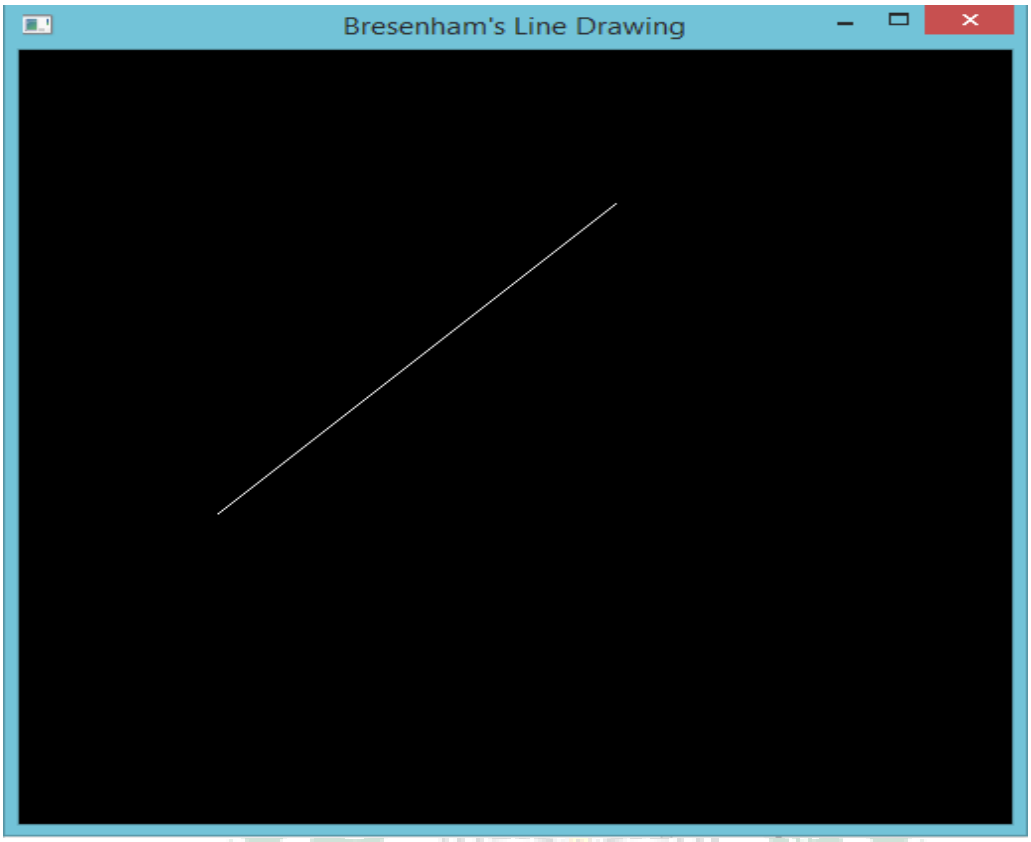
```
int incx, incy, inc1, inc2;
int x,y;
dx = x2-x1;
dy = y2-y1;
if (dx < 0)
     dx = -dx;
if (dy< 0)
     dy = -dy;
incx = 1;
if (x2 < x1)
incx = -1;

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()
```

```
{
      draw_line(x1, x2, y1, y2);
      glFlush();
}
int main(int argc, char **argv)
{
      printf("Enter (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;
}
```

Output:

```
Enter (x1, y1, x2, y2)
100
200
300
400
```



LAB PROGRAM - 2

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

```
#include<GL/glut.h>
#include<stdio.h>
int x,y;
int rFlag=0;
void draw_pixel(float x1,float y1)
{
      glColor3f(0.0,0.0,1.0);
      glPointSize(5.0);
      glBegin(GL_POINTS);
      glVertex2f(x1,y1);
```

```
	glEnd();
}
void triangle()
{
	glColor3f(1.0,0.0,0.0);
	glBegin(GL_POLYGON);
	glVertex2f(100,100);
	glVertex2f(250,400);
	glVertex2f(400,100);
	glEnd();
}
float th=0.0;
float trX=0.0,trY=0.0;
void display()
{
	glClear(GL_COLOR_BUFFER_BIT);
	glLoadIdentity();
	if(rFlag==1) //Rotate Around origin
	{
		trX=0.0;
		trY=0.0;
		th+=0.1;
		draw_pixel(0.0,0.0);
	}
	if(rFlag==2) //Rotate Around Fixed Point
	{
		trX=x;
		trY=y;
		th+=0.1;
		draw_pixel(x,y);
```

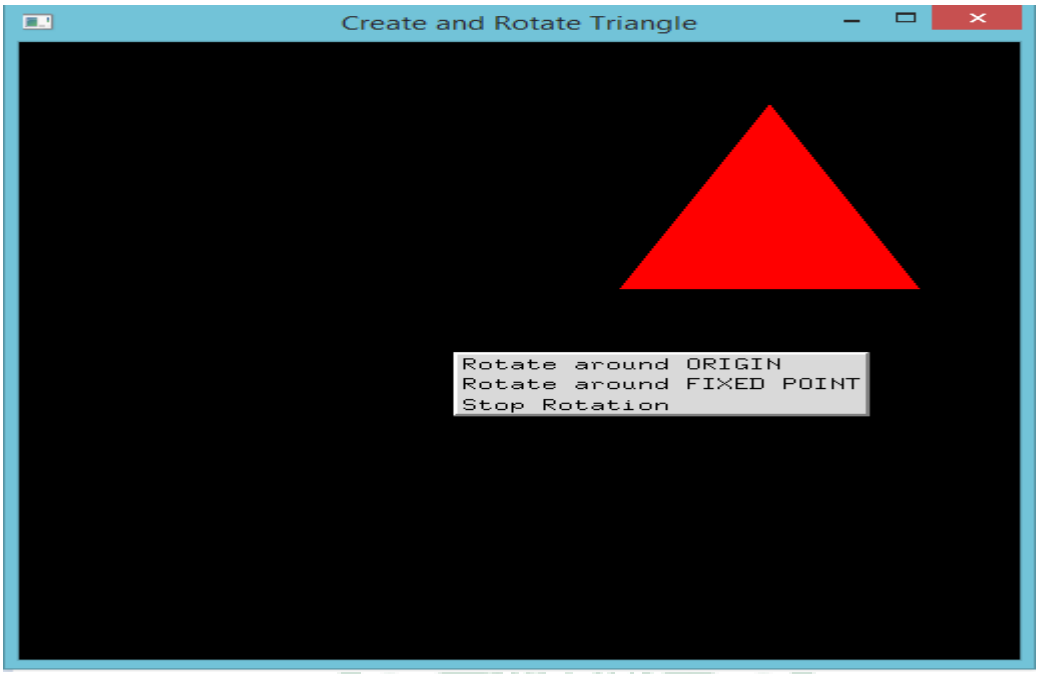
```
        }
        glTranslatef(trX,trY,0.0);
        glRotatef(th,0.0,0.0,1.0);
        glTranslatef(-trX,-trY,0.0);
        triangle();
        glutPostRedisplay();
        glutSwapBuffers();
}
void myInit()
{
        glClearColor(0.0,0.0,0.0,1.0);
        glMatrixMode(GL_PROJECTION);
        //glLoadIdentity();
        gluOrtho2D(-500.0, 500.0, -500.0, 500.0);
        glMatrixMode(GL_MODELVIEW);
}
void rotateMenu (int option)
{
        if(option==1)
                rFlag=1;
        if(option==2)
                rFlag=2;
        if(option==3)
                rFlag=3;
}
int main(int argc, char **argv)
{
        printf( "Enter Fixed Points (x,y) for Rotation: \n");
        fflush(stdout);
        scanf("%d %d", &x, &y);
```

```
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(0, 0);
        glutCreateWindow("Create and Rotate Triangle");
        myInit();
        glutDisplayFunc(display);
        glutCreateMenu(rotateMenu);
        glutAddMenuEntry("Rotate around ORIGIN",1);
        glutAddMenuEntry("Rotate around FIXED POINT",2);
        glutAddMenuEntry("Stop Rotation",3);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
        glutMainLoop();
}
```

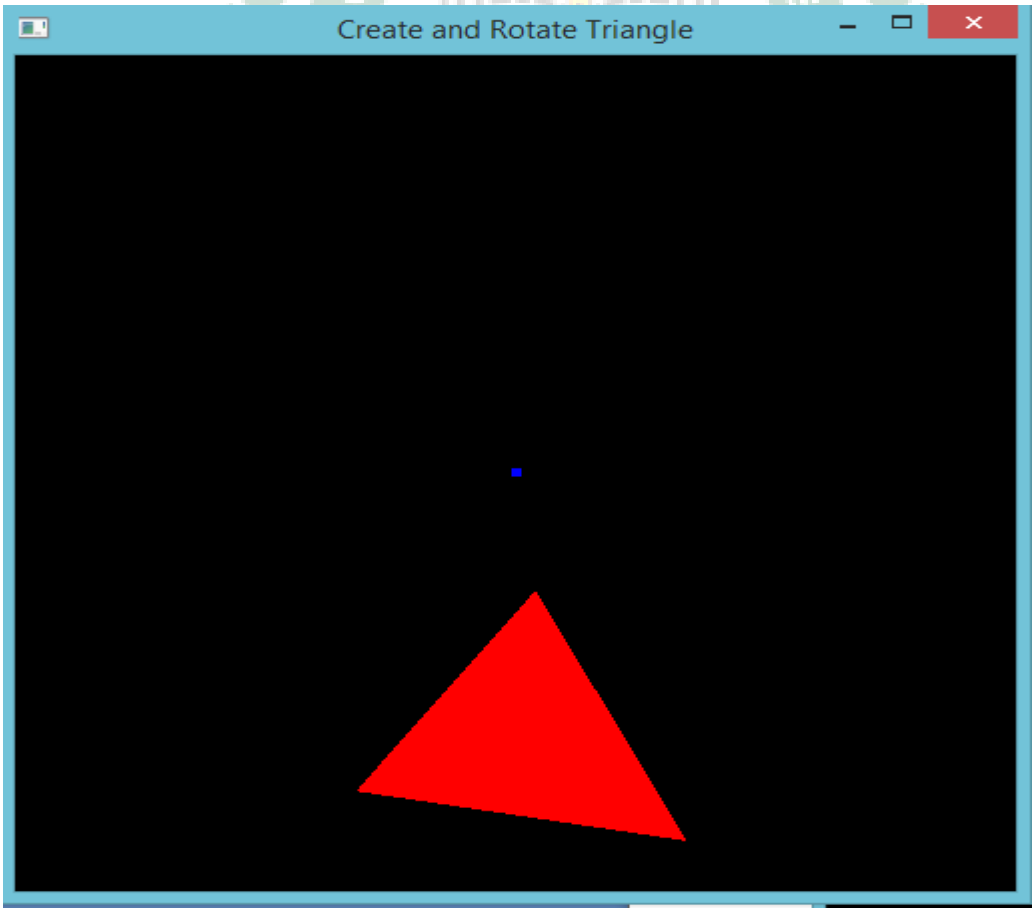
Output:

Enter Fixed Points (x,y) for Rotation:
100
200

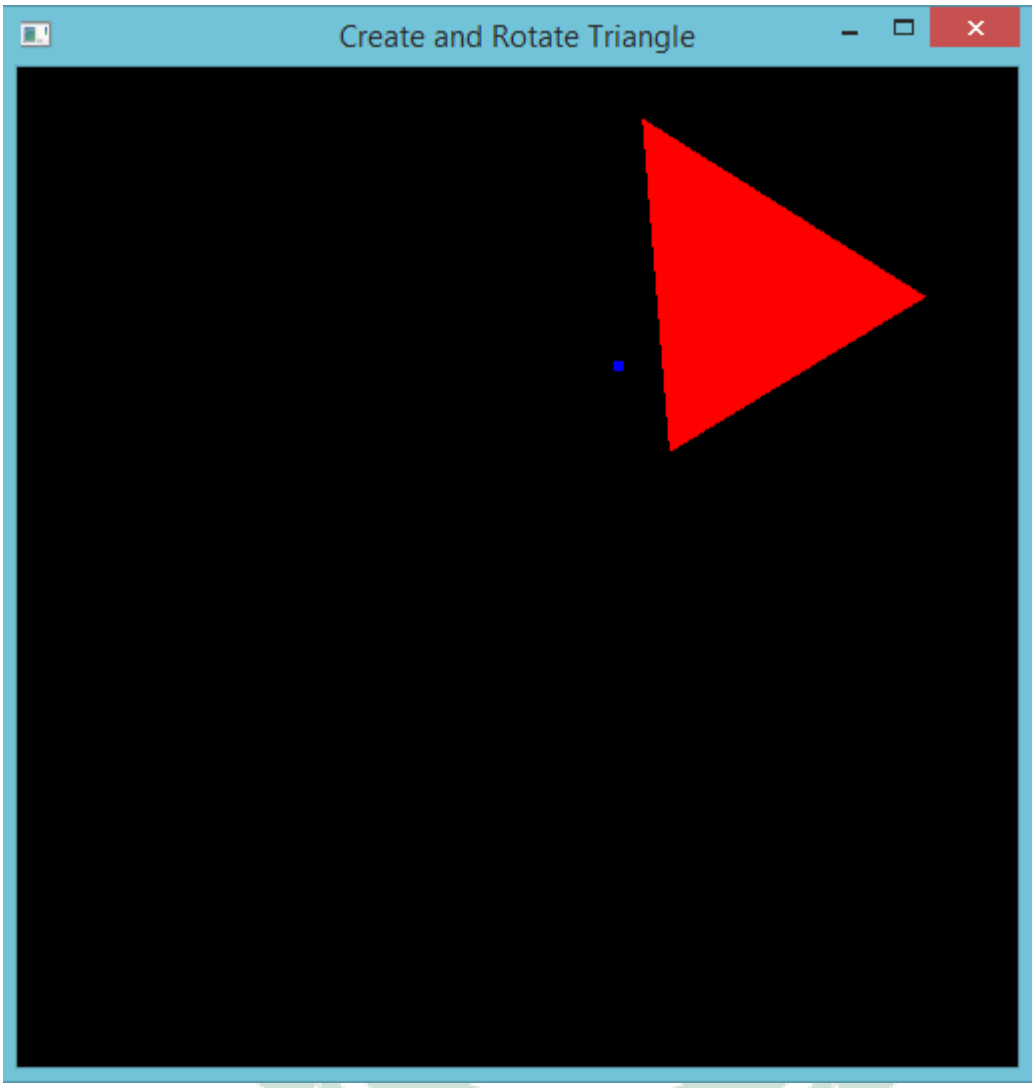
Case i : Triangle and options(on click of right button)



Case ii : Option 1 - Rotate around ORIGIN



Case iii: option 2 -- Rotate around FIXED POINT


Create and Rotate Triangle

LAB PROGRAM -3

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

```
#include<stdlib.h>
#include<GL/glut.h>
GLfloat vertices[] = {-1.0,-1.0,-1.0,
            1.0,-1.0,-1.0,
            1.0,1.0,-1.0,
           -1.0,1.0,-1.0,
           -1.0,-1.0,1.0,
           1.0,-1.0,1.0,
           1.0,1.0,1.0,
           -1.0,1.0,1.0};
GLfloat colors[] = {0.0,0.0,0.0,
                    1.0,0.0,0.0,
                    1.0,1.0,0.0,
                    0.0,1.0,0.0,
                    0.0,0.0,1.0,
                    1.0,0.0,1.0,
                    1.0,1.0,1.0,
                    0.0,1.0,1.0};
GLubyte cubeIndices[]={0,3,2,1,
                  2,3,7,6,
                  0,4,7,3,
                  1,2,6,5,
                  4,5,6,7,
                  0,1,5,4};
static GLfloat theta[] = {0.0,0.0,0.0};
static GLint axis = 2;
void display(void)
```

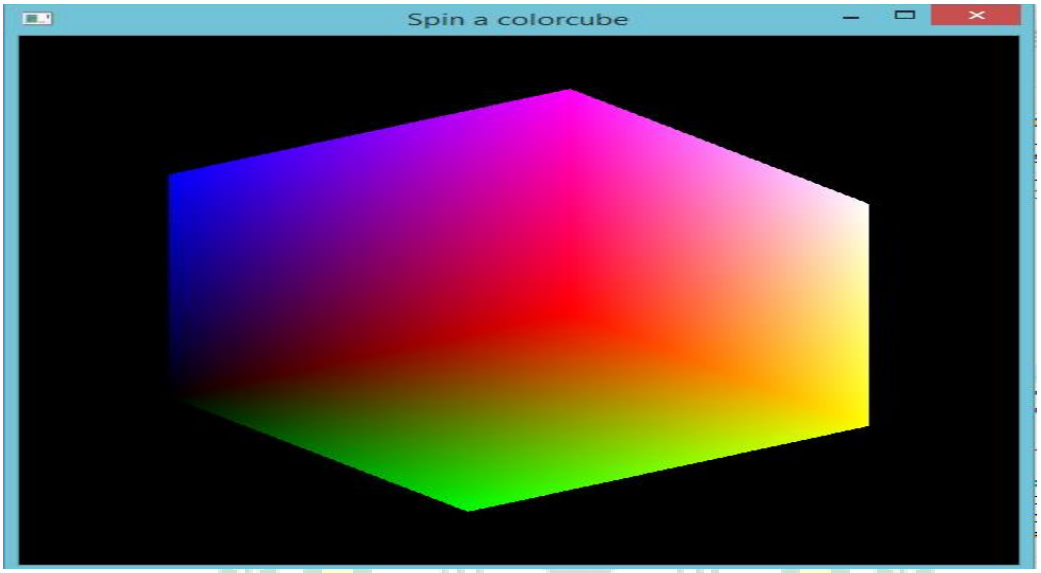
```
{
     /* display callback, clear frame buffer and z buffer,
     rotate cube and draw, swap buffers */
     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     glLoadIdentity();
     glRotatef(theta[0], 1.0, 0.0, 0.0);
     glRotatef(theta[1], 0.0, 1.0, 0.0);
     glRotatef(theta[2], 0.0, 0.0, 1.0);
     glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE,
cubeIndices);
     //glBegin(GL_LINES);
     //glVertex3f(0.0,0.0,0.0);
     //glVertex3f(1.0,1.0,1.0);
     //glEnd();
     glFlush();
     glutSwapBuffers();
}
void spinCube()
{
     /* Idle callback, spin cube 2 degrees about selected
axis */
     theta[axis] += 0.1;
     if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
     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();

    glOrtho(-2.0, 2.0, -2.0, 2.0, -10.0, 10.0);

    glMatrixMode(GL_MODELVIEW);

}

int main(int argc, char **argv)

{

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |
GLUT_DEPTH);

    glutInitWindowSize(1000, 1000);

    glutCreateWindow("Spin a colorcube");

    glutReshapeFunc(myReshape);

    glutDisplayFunc(display);

    glutIdleFunc(spinCube);

    glutMouseFunc(mouse);

    glEnable(GL_DEPTH_TEST); /*Enable hidden--surface--
removal */

    glEnableClientState(GL_COLOR_ARRAY);

    glEnableClientState(GL_VERTEX_ARRAY);

    glVertexPointer(3, GL_FLOAT, 0, vertices);

    glColorPointer(3,GL_FLOAT, 0, colors);

    //glColor3f(1.0,1.0,1.0);
```

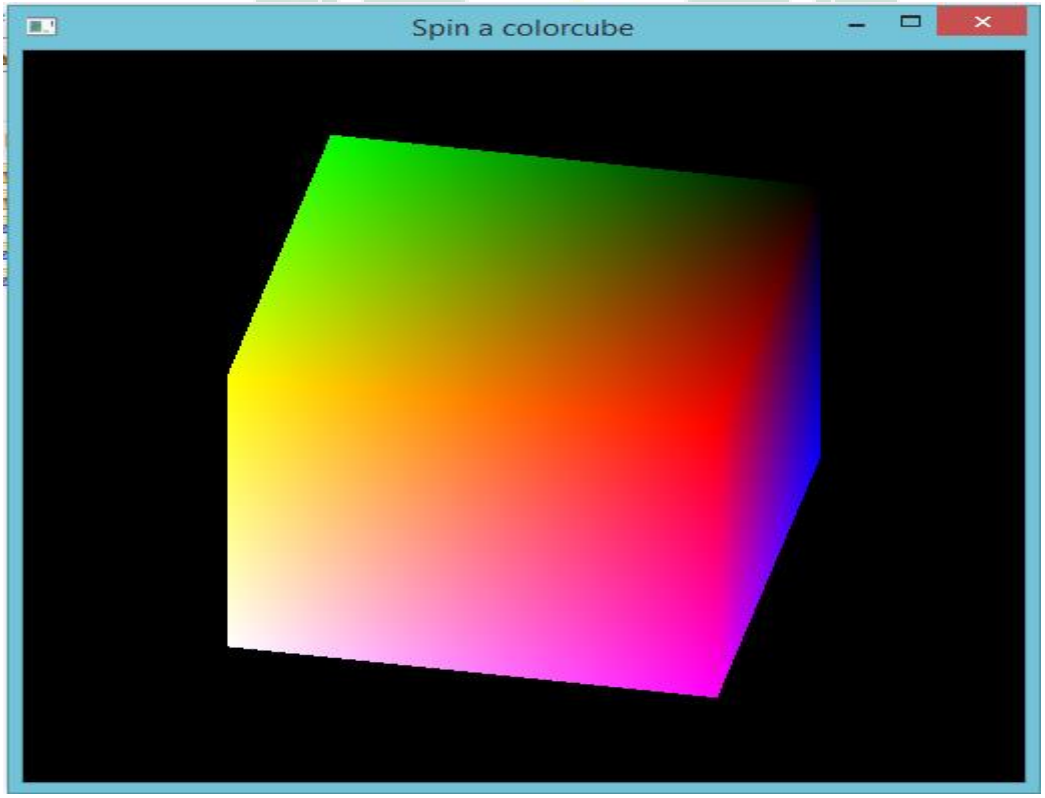
```
    glutMainLoop();
    return 0;
}
```

Output:

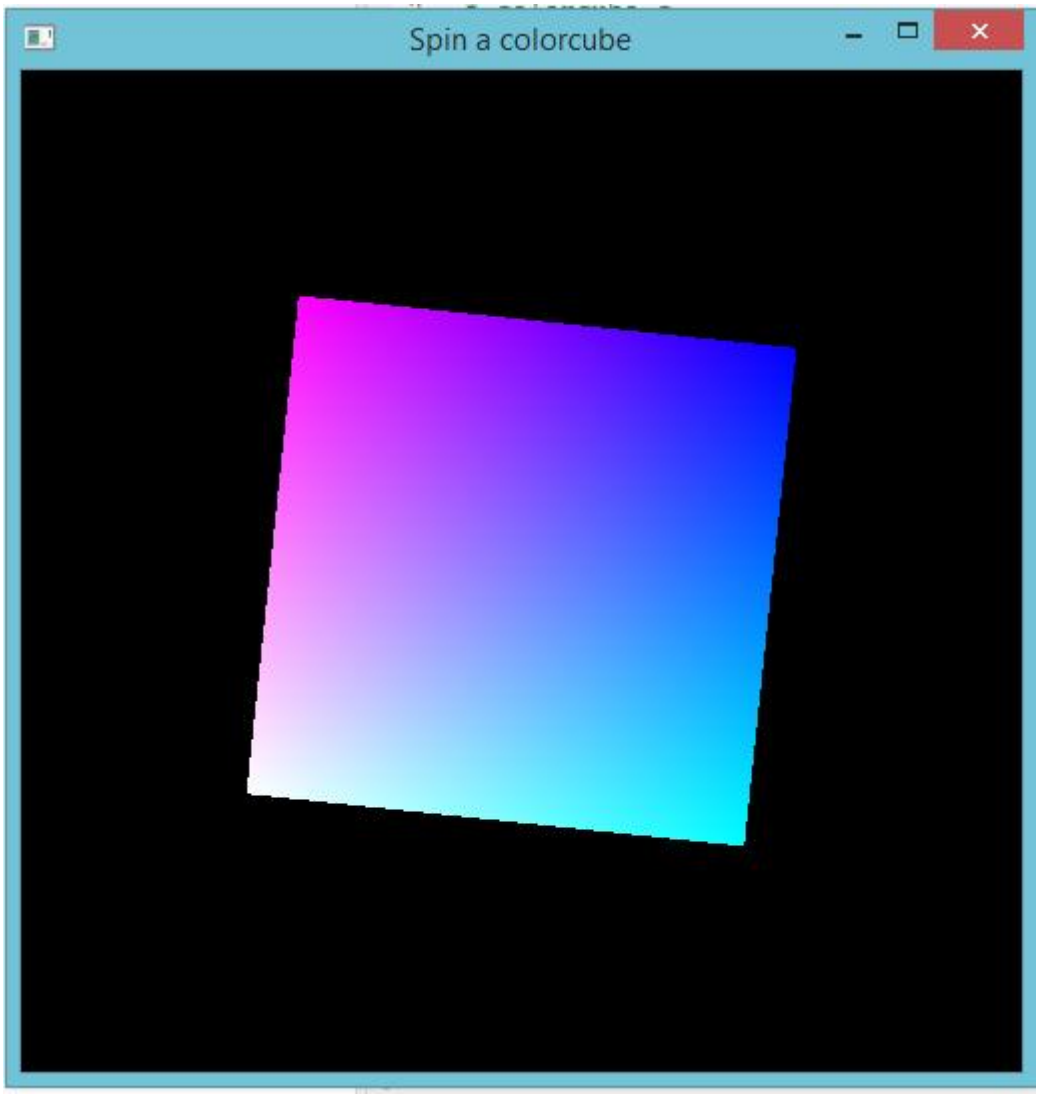
x-axis



y-axis



z axis



LAB PROGRAM -4

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

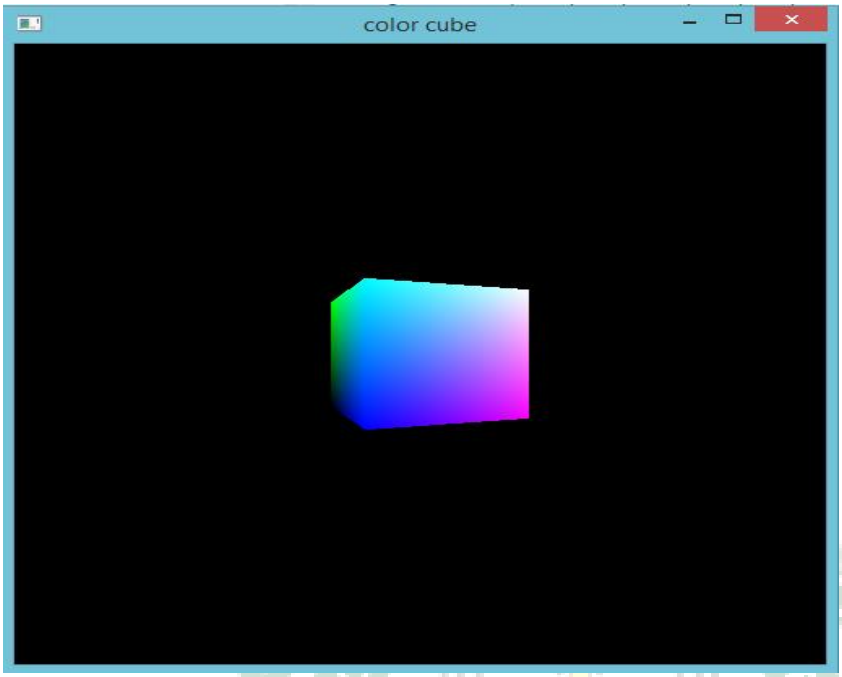
```
#include <GL/glut.h>
#include <stdio.h>
GLfloat vertices[ ]={
                     -1.0, -1.0, -1.0,
                      1.0, -1.0, -1.0,
                      1.0,  1.0, -1.0,
                     -1.0,  1.0, -1.0,
                     -1.0, -1.0,  1.0,
                      1.0, -1.0,  1.0,
`                     1.0,  1.0,  1.0,
                     -1.0,  1.0,  1.0
                      };
GLfloat normals[ ] ={
                     -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[ ]={
                    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};
GLubyte cubeIndices[]={0,3,2,1,
                       2,3,7,6,
                       0,4,7,3,
                       1,2,6,5,
                       4,5,6,7,
                       0,1,5,4 };
static GLdouble viewer[]={0.0,0.0,5.0};
```

```
void display(void)
{
      glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
      glLoadIdentity();
      gluLookAt(viewer[0],viewer[1],viewer[2],0.0,0.0,0.0,0.0
,1.0,0.0);
      //gluLookAt(4,3,5,0.0,0.0,0.0,0.0,1.0,0.0);
      glDrawElements(GL_QUADS,24,GL_UNSIGNED_BYTE,cubeIndices
);
      glFlush();
      glutSwapBuffers();
}

void keys(unsigned char key, int x, int y)
{     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;
      glutPostRedisplay();
}

void myReshape(int w, int h)
{
      glViewport(0,0,w,h);
      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);
      //glFrustum(-2.0,2.0,-2.0,2.0,2.0,20.0);
      glMatrixMode(GL_MODELVIEW);
}
int main(int argc, char **argv)
{
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |
GLUT_DEPTH);
      glutInitWindowSize(500,500);
      glutCreateWindow("color cube");
      glutReshapeFunc(myReshape);
      glutDisplayFunc(display);
      glutKeyboardFunc(keys);
      glEnable(GL_DEPTH_TEST);
      glEnableClientState(GL_COLOR_ARRAY);
      glEnableClientState(GL_VERTEX_ARRAY);
```

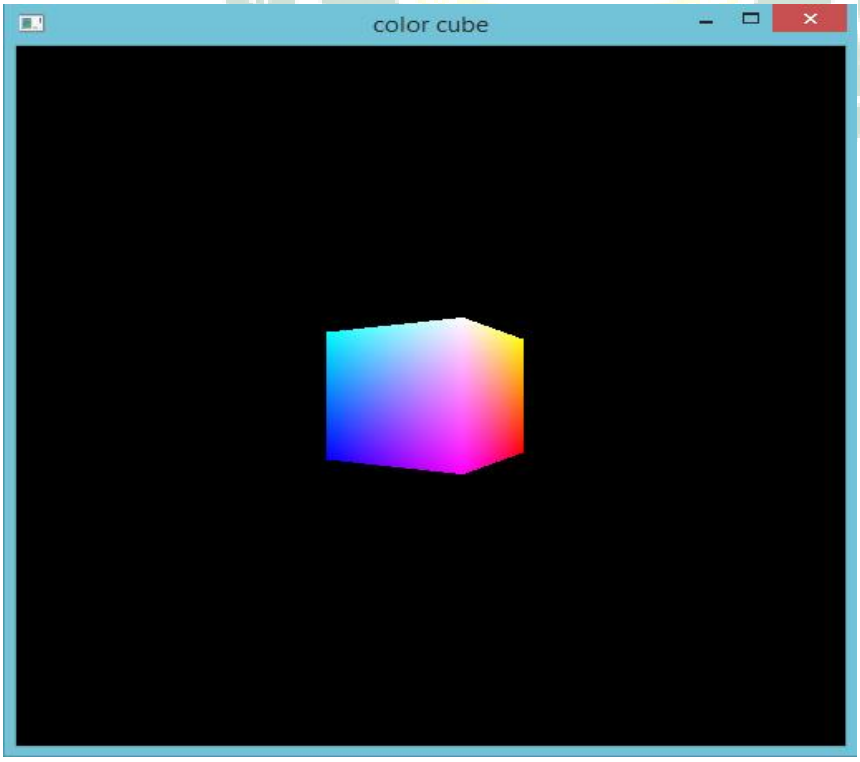
```
        glEnableClientState(GL_NORMAL_ARRAY);
        glVertexPointer(3,GL_FLOAT,0,vertices);
        glColorPointer(3,GL_FLOAT,0,colors);
        glNormalPointer(GL_FLOAT,0,normals);
        //glColor3f(1.0,1.0,1.0);
        glutMainLoop();
}
```

Output:

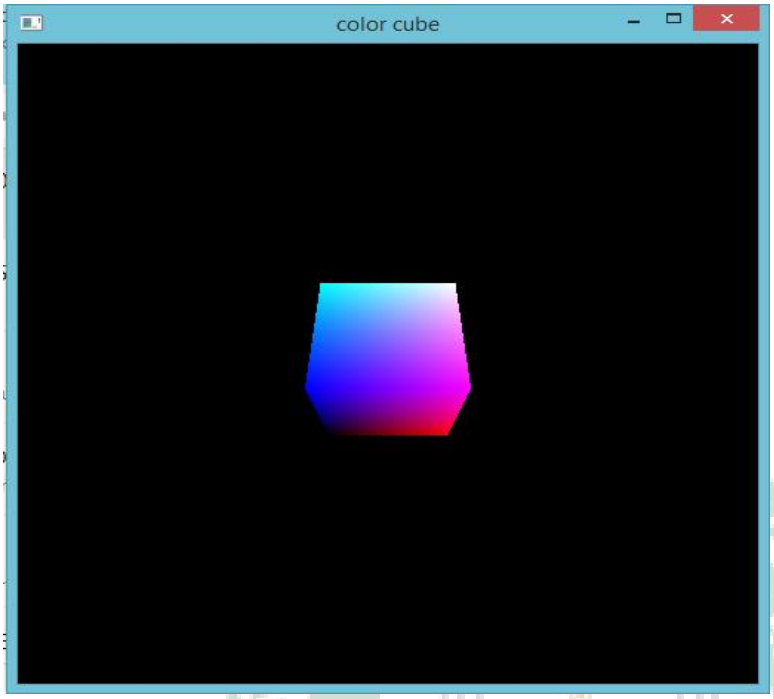
case i : On click of ' x '



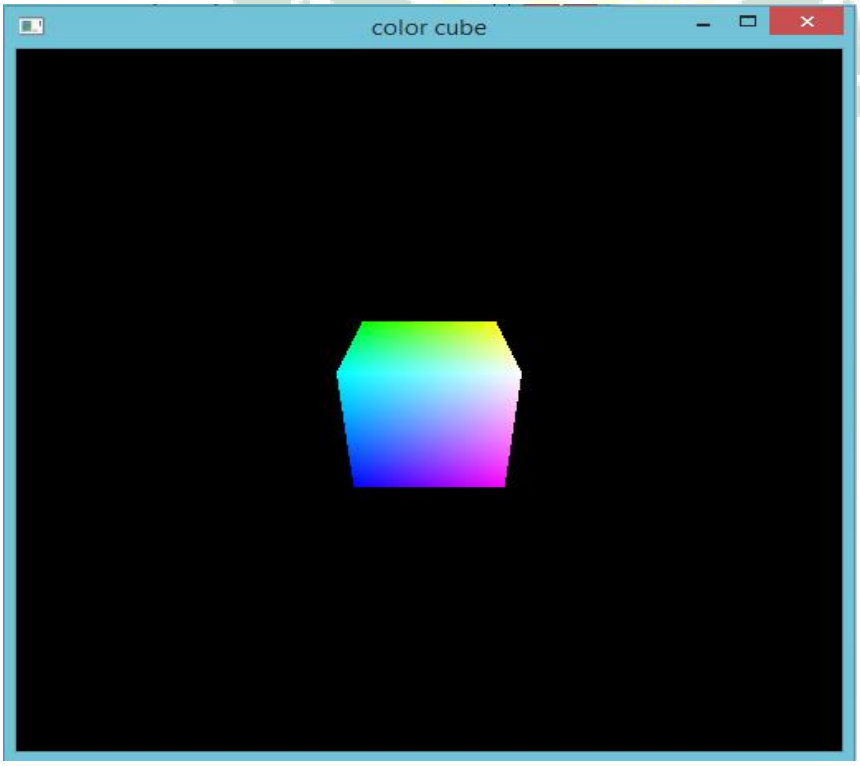
case ii : On click of 'X'



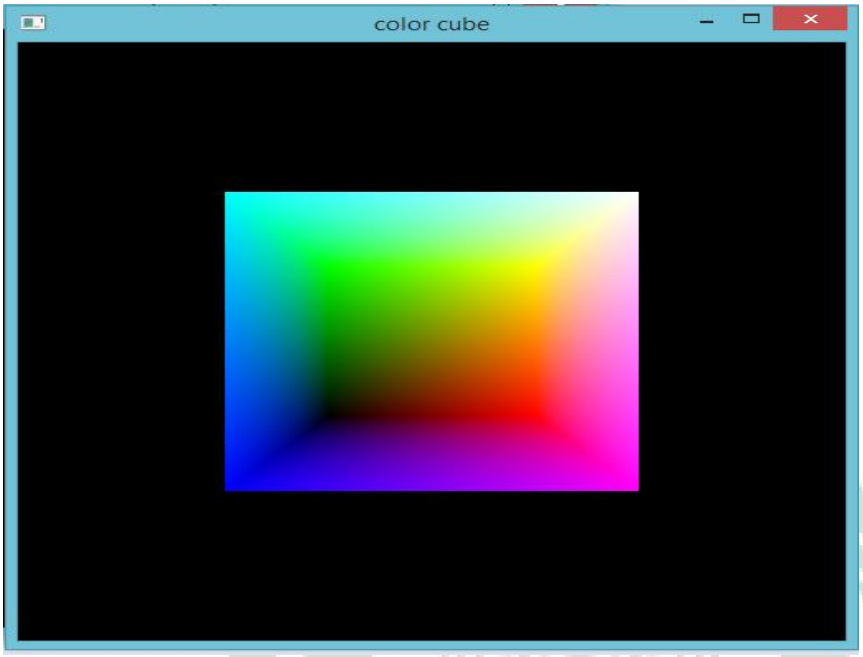
case iii: On click of 'y'



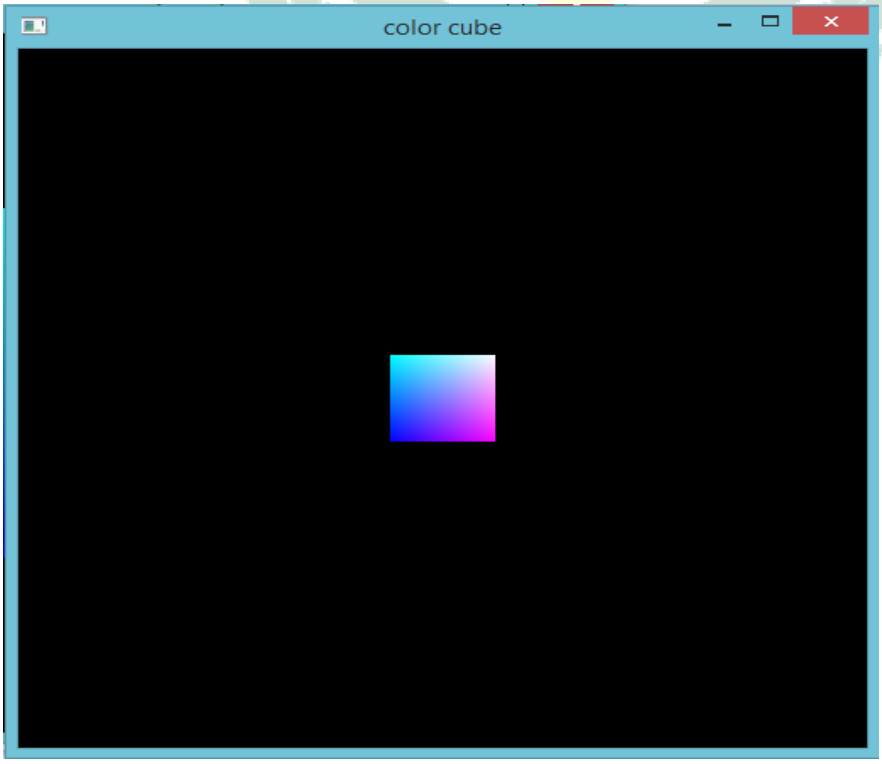
case iv : On click of 'Y'



case v : on click of ' z '



case vi : on click of ' Z '



LAB PROGRAM -5

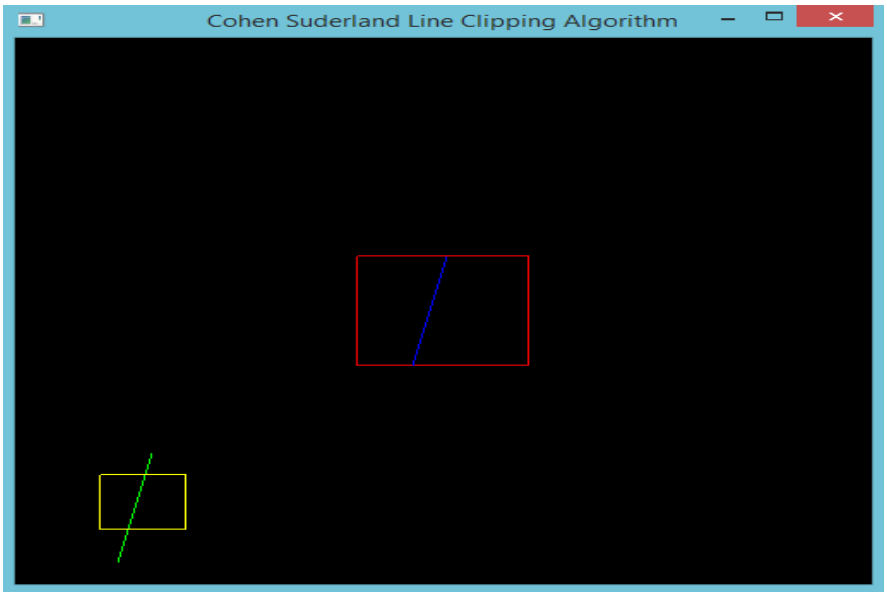
5. Clip a lines using Cohen-Sutherland algorithm.

```
#include <stdio.h>
#include <GL\glut.h>
#include <stdbool.h>
#define BOOL bool

double xmin=50,ymin=50, xmax=100,ymax=100;
double xvmin=200,yvmin=200,xvmax=300,yvmax=300;
const int RIGHT = 8; const int LEFT = 2; const int TOP = 4;
const int BOTTOM = 1;

int ComputeOutCode (double x, double y)
{
      int code = 0;
      if (y > ymax) code |= TOP; //above the clip window
      else if (y < ymin) code |= BOTTOM; //below the clip
window
      if (x > xmax) code |= RIGHT; //to the right of clip
window
      else if (x < xmin) code |= LEFT; //to the left of clip
window
      return code;
}
void CohenSutherland(double x0, double y0,double x1, double
y1)
{
      int outcode0, outcode1, outcodeOut;
      bool accept = false, done = false;
      outcode0 = ComputeOutCode (x0, y0);
      outcode1 = ComputeOutCode (x1, y1);
      do{
            if (!(outcode0 | outcode1))
            {
                  accept = true;
                  done = true;
            }
      else if (outcode0 & outcode1)
      done = true;
      else {
            double x, y;
            outcodeOut = outcode0? outcode0: outcode1;
            if (outcodeOut & TOP)
```

```
            {
                  x = x0 + (x1 - x0) * (ymax - y0)/(y1 - y0);
                  y = ymax;
            }
            else if (outcodeOut & BOTTOM)
            {
                  x = x0 + (x1 - x0) * (ymin - y0)/(y1 - y0); y
= ymin;
            }
      else if (outcodeOut & RIGHT)
      {
            y = y0 + (y1 - y0) * (xmax - x0)/(x1 - x0);
            x = xmax;
      }
      else
      {
            y = y0 + (y1 - y0) * (xmin - x0)/(x1 - x0);
            x = xmin;
      }
      if (outcodeOut == outcode0)
      {
            x0 = x; y0 = y;
            outcode0 = ComputeOutCode (x0, y0);
      }
      else
      {
            x1 = x; y1 = y;
            outcode1 = ComputeOutCode (x1, y1);
      }
}
}while (!done);

if (accept)
{
      double sx=(xvmax-xvmin)/(xmax-xmin);
      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;
      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);
      glBegin(GL_LINES);
      glVertex2d (vx0, vy0);
      glVertex2d (vx1, vy1);
```

```
        glEnd();
} }


void display()
{
        double x0=60,y0=20,x1=80,y1=120;
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0,1.0,0.0);
        glBegin(GL_LINES);
        glVertex2d (x0, y0);
        glVertex2d (x1, y1);
        glEnd();
        glColor3f(1.0, 1.0, 0.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(xmin, ymin);
        glVertex2f(xmax, ymin);
        glVertex2f(xmax, ymax);
        glVertex2f(xmin, ymax);
        glEnd();
        CohenSutherland(x0,y0,x1,y1);
        glFlush();
}
void myinit()
{
        glClearColor(0.0,0.0,0.0,1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0,500.0,0.0,500.0);
        glMatrixMode(GL_MODELVIEW);
}
int 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");
        myinit();
        glutDisplayFunc(display);
        glutMainLoop();
        return 0;
}
```

Output:



LAB PROGRAM - 6

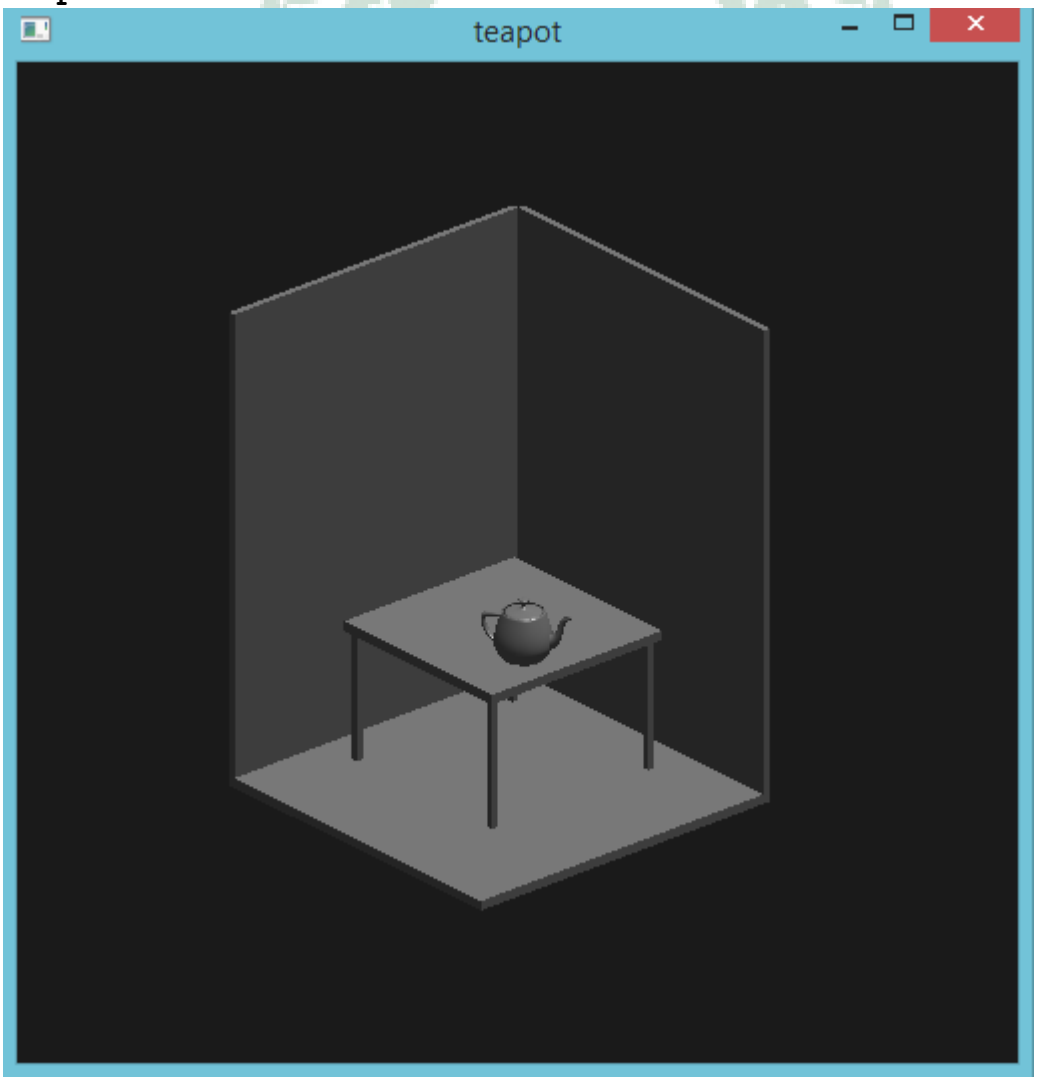
6. To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.

```
#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>
void wall(double thickness)
{
      glPushMatrix();
      glTranslated(0.5,0.5*thickness,0.5);
      glScaled(1.0,thickness,1.0);
      glutSolidCube(1.0);
      glPopMatrix();
}
void tableleg(double thick,double len)
{
      glPushMatrix();
      glTranslated(0,len/2,0);
      glScaled(thick,len,thick);
      glutSolidCube(1.0);
      glPopMatrix();
}
void table(double topw,double topt,double legt,double legl)
{
      glPushMatrix();
      glTranslated(0,legl,0);
      glScaled(topw,topt,topw);
      glutSolidCube(1.0);
      glPopMatrix();
      double dist=0.95*topw/2.0-legt/2.0;
      glPushMatrix();
```

```
        glTranslated(dist,0,dist);
        tableleg(legt,legl);
        glTranslated(0,0,-2*dist);
        tableleg(legt,legl);
        glTranslated(-2*dist,0,2*dist);
        tableleg(legt,legl);
        glTranslated(0,0,-2*dist);
        tableleg(legt,legl);
        glPopMatrix();
}
void displaysolid(void)
{
        GLfloat mat_ambient[]={0.7f,0.7f,0.7f,1.0f};
        GLfloat mat_diffuse[]={0.5f,0.5f,0.5f,1.0f};
        GLfloat mat_specular[]={1.0f,1.0f,1.0f,1.0f};
        GLfloat mat_shininess[]={50.0f};
        glMaterialfv(GL_FRONT,GL_AMBIENT,mat_ambient);
        glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);
        glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
        glMaterialfv(GL_FRONT,GL_SHININESS,mat_shininess);
        GLfloat lightint[]={0.7f,0.7f,0.7f,1.0f};
        GLfloat lightpos[]={2.0f,6.0f,3.0f,0.0f};
        glLightfv(GL_LIGHT0,GL_POSITION,lightpos);
        glLightfv(GL_LIGHT0,GL_DIFFUSE,lightint);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        double winht=1.0;
        glOrtho(-winht*64/48.0,winht*64/48.0,-
        winht,winht,0.1,100.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(2.3,1.3,2.0,0.0,0.25,0.0,0.0,1.0,0.0);
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
        glRotated(90.0,0.0,0.0,1.0);
        wall(0.02);
        glPopMatrix();
        wall(0.02);
        glPushMatrix();
        glRotated(-90.0,1.0,0.0,0.0);
        wall(0.02);
        glPopMatrix();
        glPushMatrix();
        glTranslated(0.4,0,0.4);
        table(0.6,0.02,0.02,0.3);
        glPopMatrix();
        glPushMatrix();
        glTranslated(0.6,0.38,0.5);
        glRotated(30,0,1,0);
        glutSolidTeapot(0.08);
        glPopMatrix();
```

```
        glFlush();
}
int main(int argc,char **argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("teapot");
        glutDisplayFunc(displaysolid);
        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();
}
```

Output:



LAB PROGRAM - 7

7. 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.

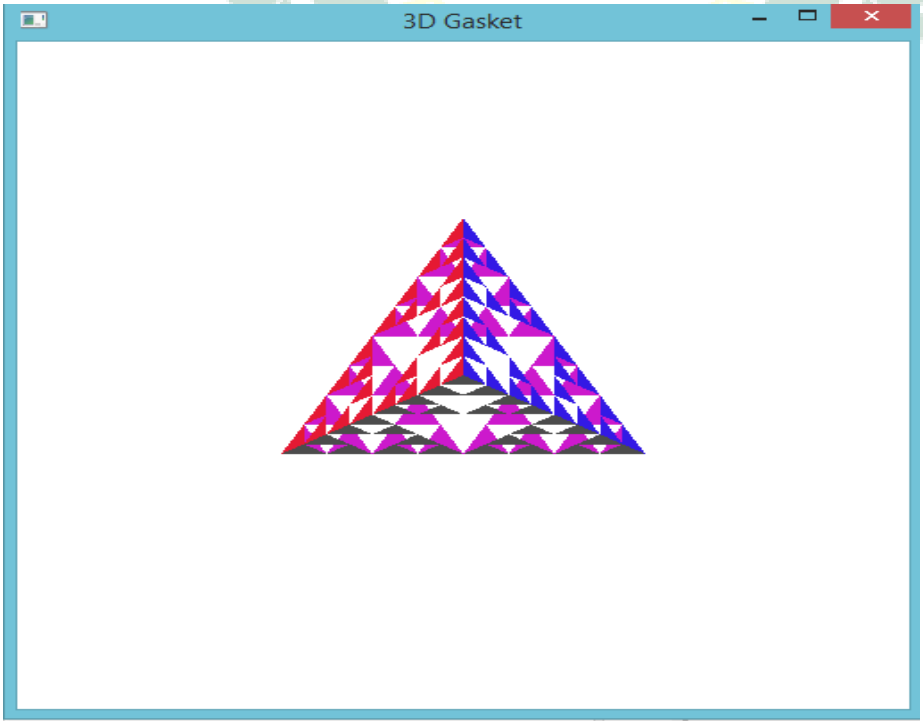
```
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
typedef float point[3];
point v[]= {{0.0,0.0,1.0},
           {0.0,0.942809,-0.33333},
           {-0.816497,-0.471405,-0.333333},
           {0.816497,-0.471405,-0.333333}
           }; //vertices of tetrahedron
int n;
void triangle(point a, point b, point c) //to draw triangles
{
      glBegin(GL_POLYGON);
      glVertex3fv(a);
      glVertex3fv(b);
      glVertex3fv(c);
      glEnd();
}
void divide_triangle(point a,point b,point c,int m)
{     point v1,v2,v3;
      int j;
      if(m>0) // triangle subdivision
      {
```

```
            for(j=0;j<3;j++)
            v1[j]=(a[j]+b[j])/2; //find midpoint of side1
            for (j=0;j<3;j++)
            v2[j]=(a[j]+c[j])/2; //find midpoint of side2
            for(j=0;j<3;j++)
            v3[j]=(b[j]+c[j])/2; //find midpoint of side3
            divide_triangle(a,v1,v2,m-1); // triangle
redivision ,
      recursive
            divide_triangle(c,v2,v3,m-1);
            divide_triangle(b,v3,v1,m-1);
      }
else (triangle (a,b,c));
}
void tetrahedron(int m) //tetrahedron division
{
      glColor3f(0.9,0.1,0.2);
      divide_triangle(v[0],v[1],v[2],m); //call for
triangular face-1
      division
      glColor3f(0.8,0.1,0.8);
      divide_triangle(v[3],v[2],v[1],m); //call for
triangular face-2
      division
      glColor3f(0.2,0.1,0.9);
      divide_triangle(v[0],v[3],v[1],m); //call for
triangular face-3
      division
      glColor3f(0.3,0.3,0.3);
      divide_triangle(v[0],v[2],v[3],m); //call for
triangular face-4
      division
}
void display(void)
{
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      glLoadIdentity();
      tetrahedron(n);
      glFlush();
}
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);
        glutPostRedisplay();
}
int main(int argc, char **argv)
{
        printf("\n Enter the number of division:\n");
        fflush(stdout);
        scanf("%d",&n);
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("3D Gasket");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glEnable(GL_DEPTH_TEST);
        glClearColor(1.0,1.0,1.0,1.0);
        glutMainLoop();
        return 0;
}
```

Output:

Enter the number of divisions: 3



LAB PROGRAM - 8

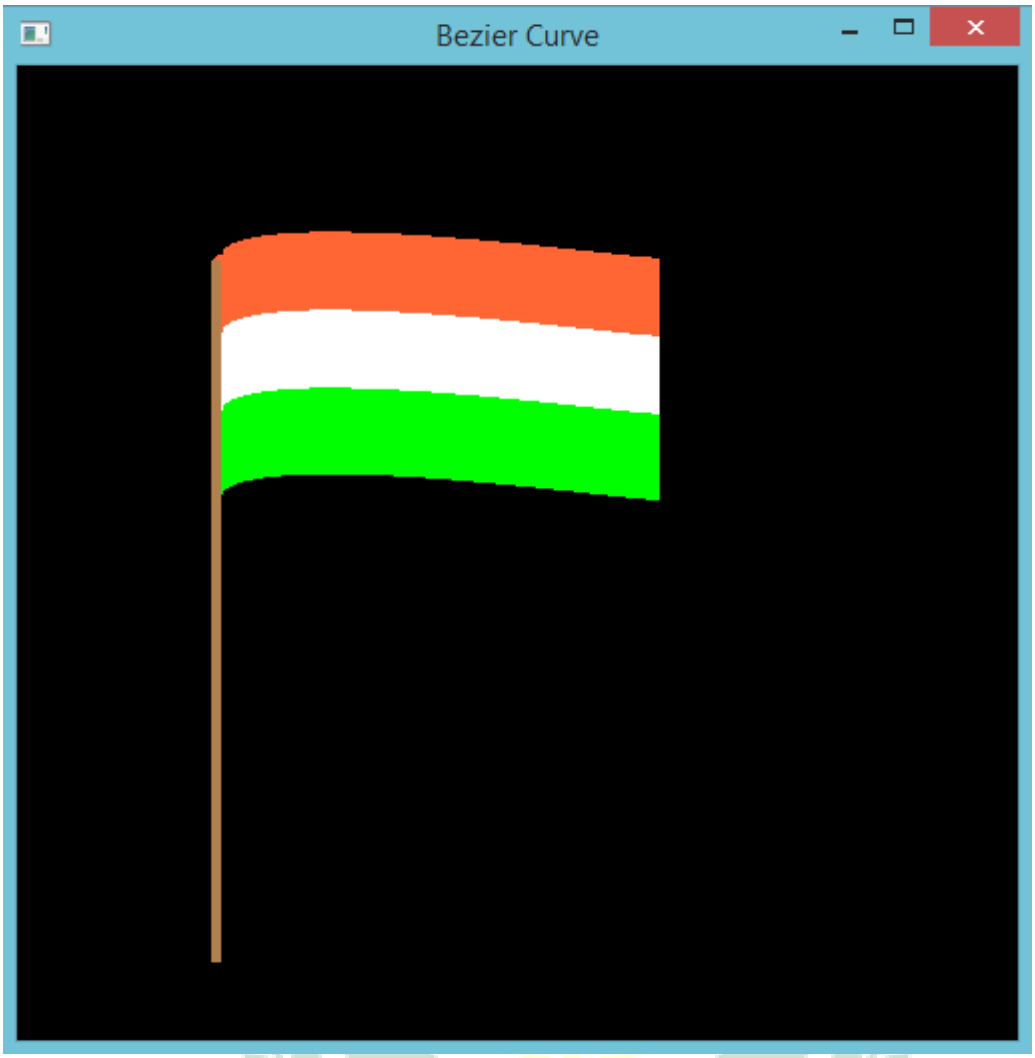
8. Develop a menu driven program to animate a flag using Bezier Curve algorithm.

```
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
#define PI 3.1416
typedef struct
{
GLfloat x, y, z;
}point;
void bino(int n, int *C)
{
int 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(float u, point *pt1, int cPt, point *pt2,
int *C)
{
```

```
int k, n=cPt-1;
float bFcn;
pt1 ->x =pt1 ->y = pt1->z=0.0;
for(k=0; k< cPt; k++)
{
bFcn = C[k] * pow(u, k) * pow( 1-u, n-k);
pt1 ->x += pt2[k].x * bFcn;
pt1 ->y += pt2[k].y * bFcn;
pt1 ->z += pt2[k].z * bFcn;
}
}void bezier(point *pt1, int cPt, int bPt)
{
point bcPt;
float u;
int *C, k;
C= new int[cPt];
bino(cPt-1, C);
glBegin(GL_LINE_STRIP);
for(k=0; k<=bPt; k++)
{
u=float(k)/float(bPt);
computeBezPt(u, &bcPt, cPt, pt1, C);
glVertex2f(bcPt.x, bcPt.y);
}
glEnd();
delete[ ]C;
}
float theta = 0;
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
int nCtrlPts = 4, nBCPts =20;
point ctrlPts[4] = {{100, 400, 0}, {150, 450, 0}, {250, 350,
0}, {300, 400, 0}};
ctrlPts[1].x +=50*sin(theta * PI/180.0);
ctrlPts[1].y +=25*sin(theta * PI/180.0);
ctrlPts[2].x -= 50*sin((theta+30) * PI/180.0);
ctrlPts[2].y -= 50*sin((theta+30) * PI/180.0);
ctrlPts[3].x -= 25*sin((theta) * PI/180.0);
ctrlPts[3].y += sin((theta-30) * PI/180.0);
theta+=0.2;
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 1.0, 1.0);
glPointSize(5);
glPushMatrix();
glLineWidth(5); glColor3f(1, 0.4, 0.2); //Indian flag:
Orange color code
for(int i=0;i<50;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBCPts);
```

```
}
glColor3f(1, 1, 1); //Indian flag: white color code
for(int i=0;i<50;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBCPts);
}
glColor3f(0, 1, 0); //Indian flag: green color code
for(int i=0;i<50;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBCPts);
}
glPopMatrix();
glColor3f(0.7, 0.5,0.3);
glLineWidth(5);
glBegin(GL_LINES);
glVertex2f(100,400);
glVertex2f(100,40);
glEnd();
glutPostRedisplay();
glutSwapBuffers();
}
void init()
{
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0,500,0,500);
}
int main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowPosition(0, 0);
glutInitWindowSize(500,500);
glutCreateWindow("Bezier Curve");
init();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
```

Output:



LAB PROGRAM - 9

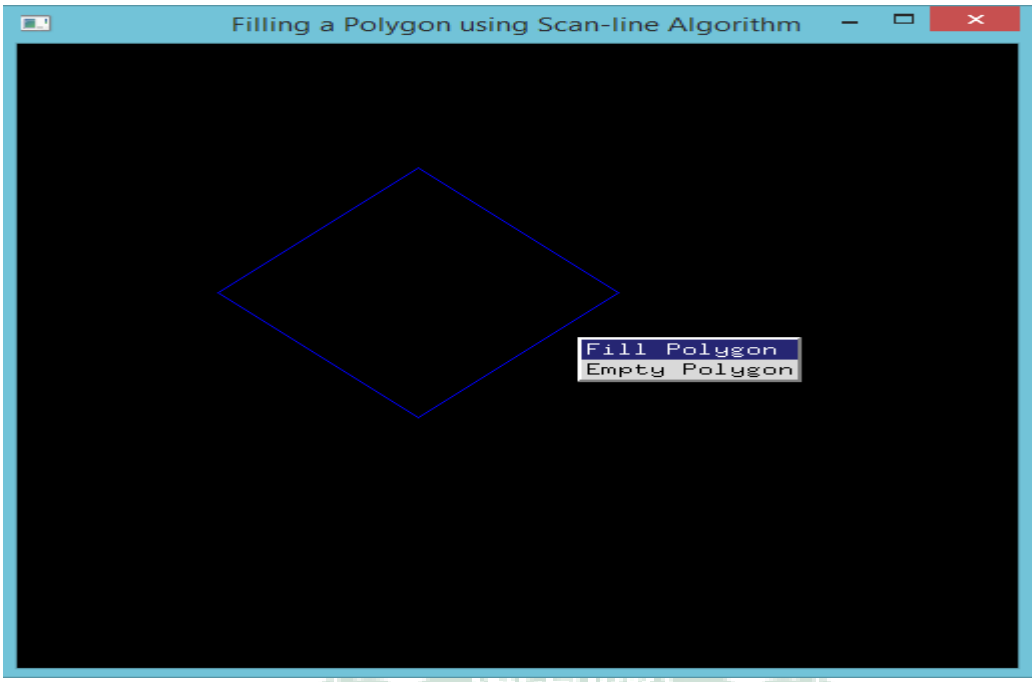
9. Develop a menu driven program to fill the polygon using scan line algorithm

```
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
float x1,x2,x3,x4,y1,y2,y3,y4;
int fillFlag=0;
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;
```

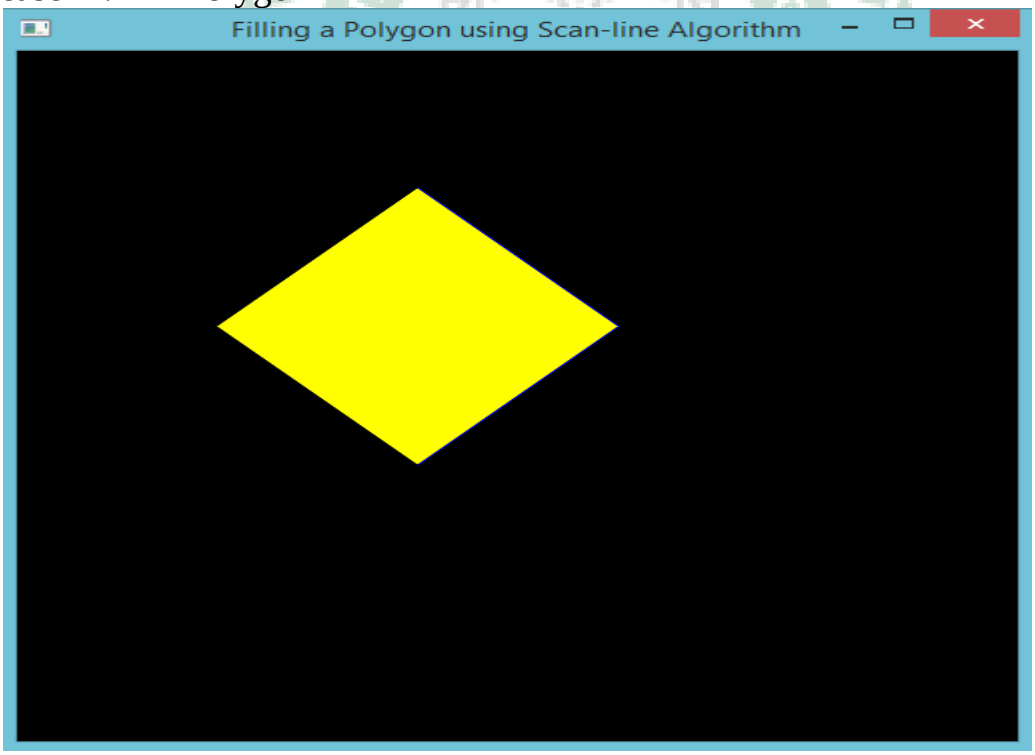
```
temp=x1;x1=x2;x2=temp;
}
if((y2-y1)!=0)
mx=(x2-x1)/(y2-y1);
else
mx=x2-x1;
x=x1;
for(i=y1;i<=y2;i++)
{
if(x<(float)le[i])
le[i]=(int)x;
if(x>(float)re[i])
re[i]=(int)x;
x+=mx;
}
}
void draw_pixel(int x,int y)
{
glColor3f(1.0,1.0,0.0);
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
}
void scanfill(float 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++)
{
for(i=(int)le[y];i<(int)re[y];i++)
draw_pixel(i,y);
}
}
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);
glColor3f(0.0, 0.0, 1.0);
```

```
glBegin(GL_LINE_LOOP);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glVertex2f(x3,y3);
glVertex2f(x4,y4);
glEnd();
if(fillFlag==1)
scanfill(x1,y1,x2,y2,x3,y3,x4,y4);
glFlush();
}
void init()
{
glClearColor(0.0,0.0,0.0,1.0);
glColor3f(1.0,0.0,0.0);
glPointSize(1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,499.0,0.0,499.0);
}
void fillMenu(int option)
{
if(option==1)
fillFlag=1;
if(option==2)
fillFlag=2;
display();
}
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");
init();
glutDisplayFunc(display);
glutCreateMenu(fillMenu);
glutAddMenuEntry("Fill Polygon",1);
glutAddMenuEntry("Empty Polygon",2);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutMainLoop();
}
```

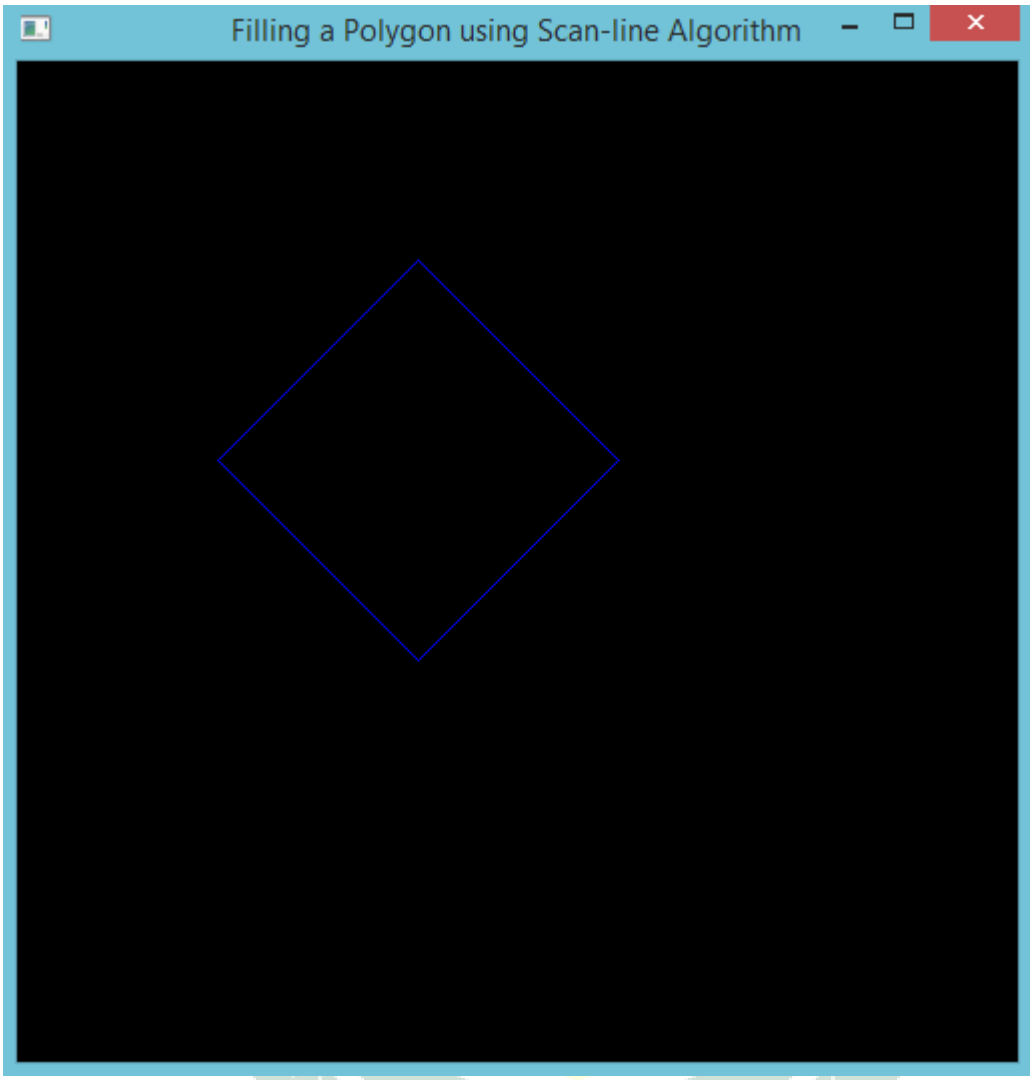
Output:
case i : Options



case ii : Fill Polygon



Case iii: Empty Polygon


Filling a Polygon using Scan-line Algorithm

VIVA QUESTIONS

1. What is OpenGL?

OpenGL was developed by silicon graphics Inc. OpenGL was developed by Silicon Graphics Inc. OpenGL (Open Graphics Library) is a cross-language, multi-platform application programming Interface (API) for rendering 2D and 3D vector graphics.

The API is typically used to interact with a Graphics processing unit (GPU), to achieve hardware accelerated rendering.

2. What are the applications of OpenGL?

OpenGL is widely used in CAD, virtual reality, scientific visualization, information visualization, flight simulation, and video games.

3. Explain the following:

i) void glutInit(int argc, char **argv);

glutInit() should be called before any other GLUT routine, because it initializes the GLUT library.

ii) void glutInitDisplayMode(unsigned int mode);

Specifies a display mode (such as RGBA or color-index, or single- or double-buffered) for windows created when glutCreateWindow() is called.
The mask argument is a bitwise ORed combination of GLUT_RGBA or GLUT_INDEX, GLUT_SINGLE or GLUT_DOUBLE, and the buffer-enabling flags: GLUT_DEPTH.
Example: for a double-buffered, RGBA-mode window with a depth, we use GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH
The default value is GLUT_RGBA | GLUT_SINGLE (an RGBA, single-buffered window).

iii)void glutInitWindowSize(int width, int height);
The width and height indicate the window's size (in pixels).

iv) void glutDisplayFunc( );
Specifies the function that's called whenever the contents of the window need to be redrawn.

4. Explain the concept of colors.

- A computer-graphics monitor emulates visible colors by lighting pixels with a combination of red, green, and blue light in proportions that excite the red-, green-, and blue-sensitive cones in the retina
- If humans had more types of cone cells, some that were yellow-sensitive for example, color monitors would probably have a yellow gun as well, and we'd use RGBY (red, green, blue, yellow) to specify colors.
- And if everyone were color-blind in the same way….??!!.

- glColor3f (1.0, 0.0, 0.0); (RGB -this colour is full red, no green, no blue.)
  A color monitor sends different proportions of red, green, and blue to each of the pixels

5. What does the glClearColor() do?

- glClearColor() establishes what color the window will be cleared to, and glClear() actually clears the window.
- Once the clearing color is set, the window is cleared to that color whenever glClear() is called.

6. What does glOrtho( ) do?

- It specifies the coordinate system OpenGL assumes as it draws the final image and how the image gets mapped to the screen.

7. What is Glut?

OpenGL Utility Toolkit (GLUT), e a popular library which standardizes and simplifies window and event management.

8. What is a color buffer?

They contain either color-index or RGB color data of every pixel.

9. What is a Depth buffer?

- The depth buffer stores a depth value for each pixel.
- The depth buffer is also called as the z buffer
- It is usually measured in terms of distance to the eye.
- The depth buffer's behavior can be modified as described in "Depth Test."

10. What is GL_DEPTH_TEST?

- The depth buffer is generally used for hidden-surface elimination. If a new candidate color for that pixel appears, it's drawn only if the corresponding object is closer than the previous object. Therefore only objects that aren't obscured by other items remain.
- Initially, the clearing value for the depth buffer is a value that's as far from the viewpoint as possible, so the depth of any object is nearer than that value comes previous value is overwritten with the new value which is closer to the eye.
- If this is how you want to use the depth buffer, you simply have to enable it by passing GL_DEPTH_TEST to glEnable().

11. What is scan conversion?
A major task of the display processor is digitizing a picture definition given in an application program into a set of pixel-intensity values for storage in the frame buffer. This digitization process is called scan conversion

12. What is rasterization?

The process of determining the appropriate pixels for representing picture or graphics object is known as rasterization

13. Name two techniques for producing colour displays with a CRT?
Beam penetration method, shadow mask method.

14. What is bitmap/ pixmap?
Some systems has only one bit per pixel; the frame buffer is often referred to as bitmap. Some system has multiple bits per pixel, the frame buffer is often referred to as pixmap.

15. What is persistence?
The time it takes the emitted light from the screen to decay one tenth of its original intensity is called as persistence.

16. What is Aspect ratio?
The ratio of vertical points to the horizontal points necessary to produce length of lines in both directions of the screen is called the Aspect ratio. Usually the aspect ratio is ¾.

17. What is frame buffer?
Picture definition is stored in a memory area called frame buffer or refresh buffer.

**********ALL THE BEST **********