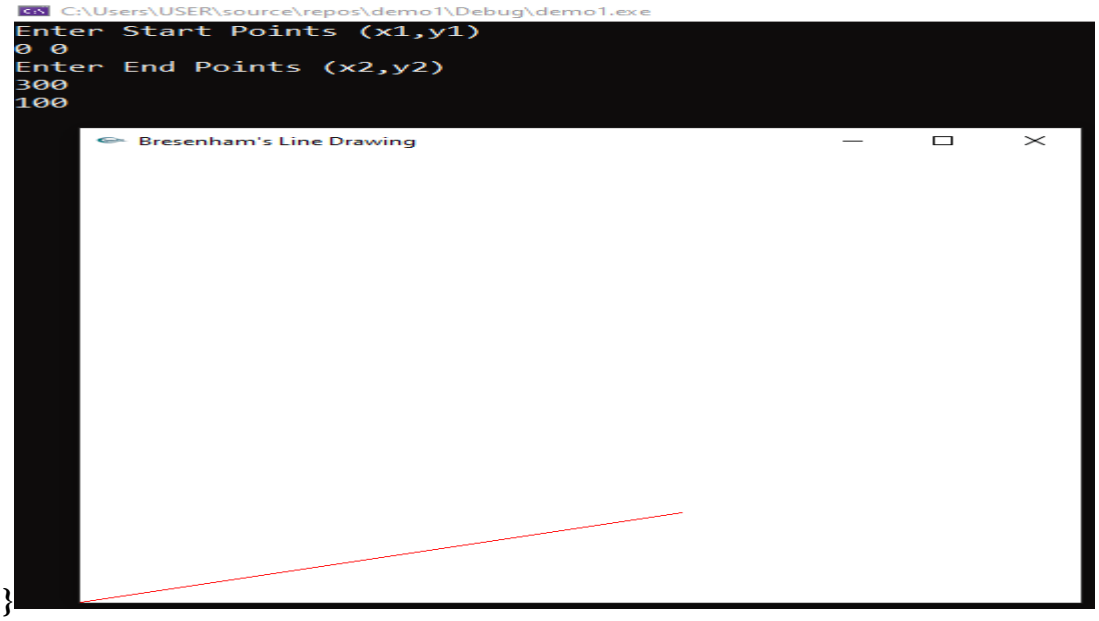
1.Implement Brenham's line drawing algorithm for all types of slope.
Refer:Text-1: Chapter 3.5
Refer:Text-2: Chapter 8

```
#include<glut.h>
#include<stdio.h>
int x1, y1, x2, y2;
void draw_pixel(int x, int y)
{
        glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_POINTS);
        glPointSize(2.0);
        glVertex2i(x, y);
        glEnd();
        glFlush();
}
void bresenhams_line_draw(int x1, int y1, int x2, int y2)
{
        int x, y;
        int dx = x2 - x1; // x difference
        int dy = y2 - y1; // y difference
        int m = dy / dx; // slope
        if (x1 > x2)
        {
                x = x2;
                y = y2;
                x2 = x1;
        }
        else
        {
                x = x1;
                y = y1;

        }
        if (m < 1)
        {
                int decision_parameter = 2 * dy - dx;
                draw_pixel(x, y); // plot a point
                while (x < x2) // from 1st point to 2nd point
                {
                        if (decision_parameter >= 0)
                        {
                                x = x + 1;
```

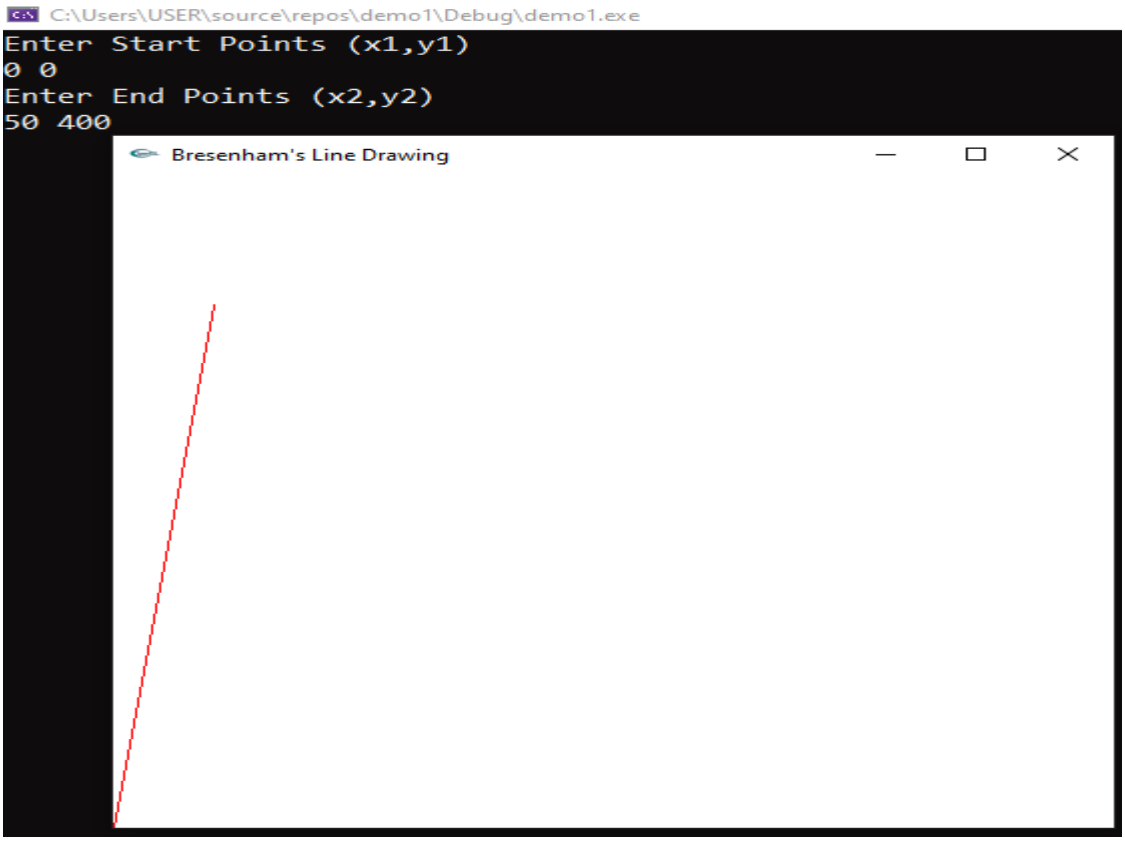
```
                y = y + 1;
        decision_parameter = decision_parameter + 2 * dy - 2 * dx * (y + 1 - y);
            }
            else
            {
                x = x + 1;
                y = y;
        decision_parameter = decision_parameter + 2 * dy - 2 * dx * (y - y);
            }
            draw_pixel(x, y);
        }
    }
    else if (m > 1)
    {
        int decision_parameter = 2 * dx - dy;
        draw_pixel(x, y);
        while (y < y2)
        {
            if (decision_parameter >= 0)
            {
                x = x + 1;
                y = y + 1;
        decision_parameter = decision_parameter + 2 * dx - 2 * dy * (x + 1 - x);
            }
            else
            {
                y = y + 1;
                x = x;
        decision_parameter = decision_parameter + 2 * dx - 2 * dy * (x - x);
            }
            draw_pixel(x, y);
        }
    }
    else if (m == 1)
    {
        draw_pixel(x, y);
        while (x < x2)
        {
            x = x + 1;
            y = y + 1;
            draw_pixel(x, y);
        }
    }
```

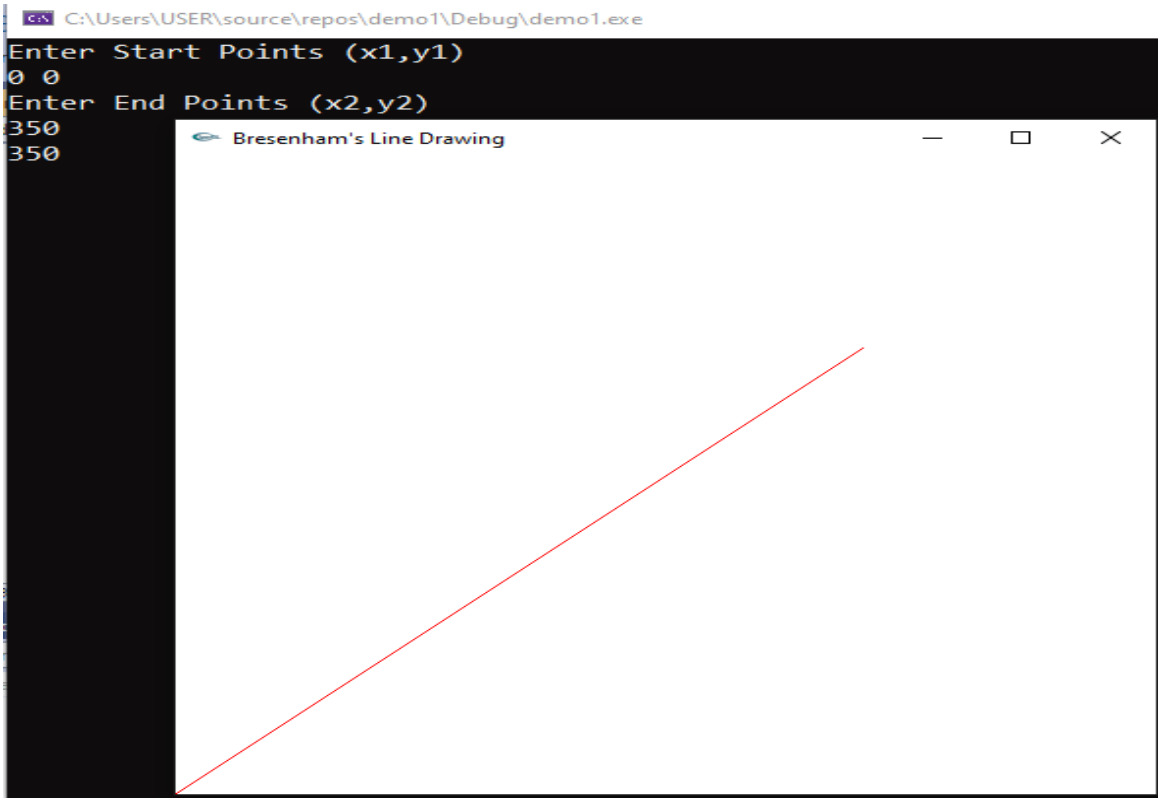
```
}
void init()
{
        glClearColor(1, 1, 1, 1);
gluOrtho2D(0.0, 500.0, 0.0, 500.0); // left ->0, right ->500, bottom ->0, top ->500
}
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        bresenhams_line_draw(x1, y1, x2, y2);
        glFlush();
}
int main(int argc, char** argv)
{
        printf("Enter Start Points (x1,y1)\n");
        scanf_s("%d %d", &x1, &y1); // 1st point from user
        printf("Enter End Points (x2,y2)\n");
        scanf_s("%d %d", &x2, &y2); // 2nd point from user
        glutInit(&argc, argv); // initialize graphics system
        glutInitWindowSize(500, 500); // 500 by 500 window size
        glutInitWindowPosition(220, 200); // where do you wanna see your window
        glutCreateWindow("Bresenham's Line Drawing"); // the title of your window
        init(); // initialize the canvas
        glutDisplayFunc(display); // call display function
        glutMainLoop(); // run forever
        return 0;
}
```

OUTPUT:



}


C:\Users\USER\source\repos\demo1\Debug\demo1.exe
Enter Start Points (x1,y1)
0 0
Enter End Points (x2,y2)
50 400
Bresenham's Line Drawing

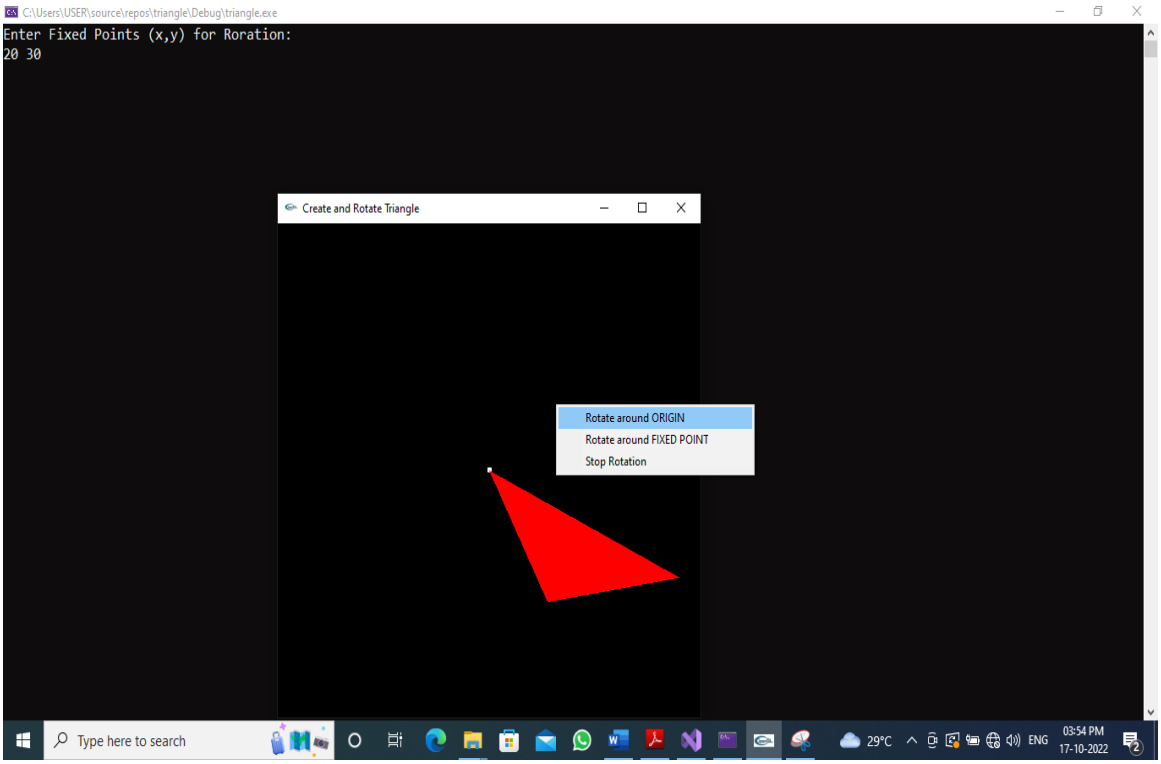

C:\Users\USER\source\repos\demo1\Debug\demo1.exe
Enter Start Points (x1,y1)
0 0
Enter End Points (x2,y2)
350
350
Bresenham's Line Drawing

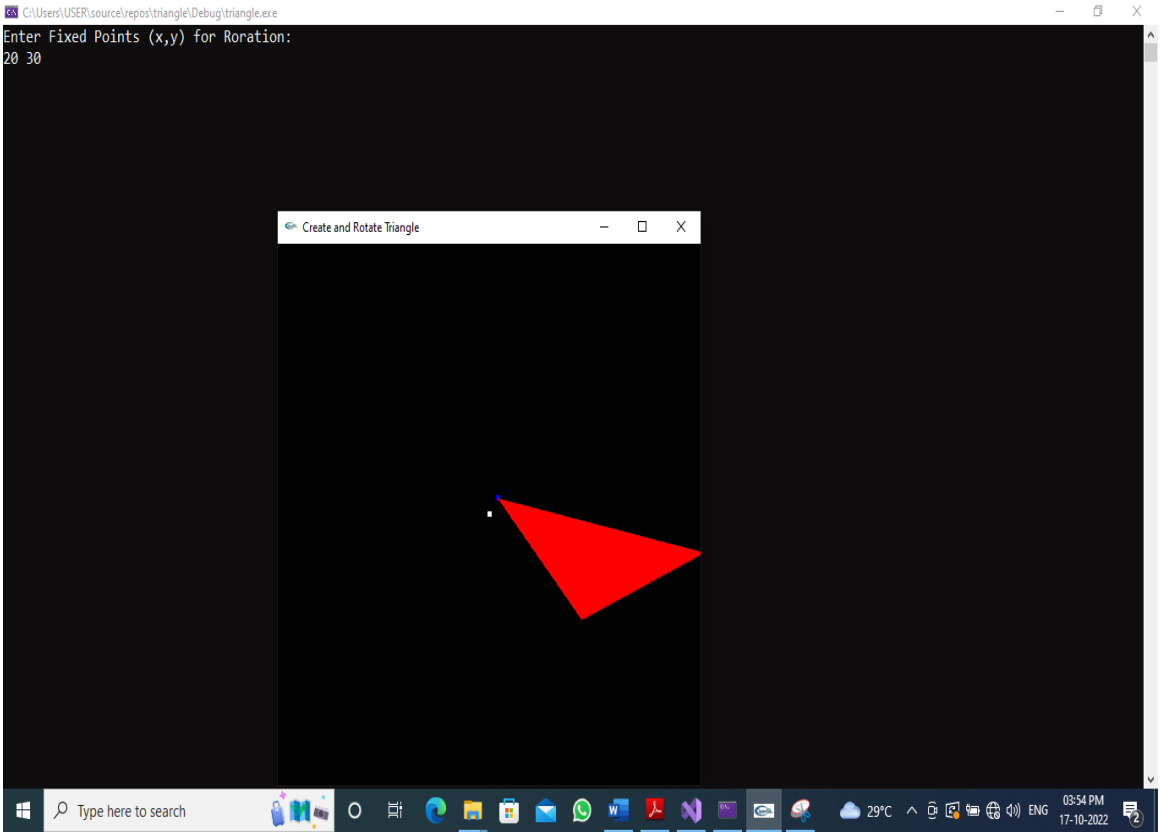
2.Create and rotate a triangle about the origin and a fixed point.
Refer: Text-1: Chapter 5-4

```
#include<glut.h>
#include<stdio.h>
int x, y;
int rFlag = 0;// don't rotate initially
float th = 0.0; // initial angle
float trX = 0.0, trY = 0.0;// initial translation
void draw_pixel(float x1, float y1)
{
        glPointSize(5.0);
        glBegin(GL_POINTS);
        glVertex2f(x1, y1);
        glEnd();
}
void triangle(int x, int y)
{
        glColor3f(1, 0, 0);
        glBegin(GL_POLYGON); // drawing a Triangle
        glVertex2f(x, y);
        glVertex2f(x + 400, y + 300);
        glVertex2f(x + 300, y + 0);
        glEnd();
}
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glLoadIdentity();
        glColor3f(1, 1, 1); // mark origin point as white dot
        draw_pixel(0, 0); // plot origin - white colour
        if (rFlag == 1) //Rotate Around origin
        {
                trX = 0.0; // no translation for rotation around origin
                trY = 0.0;
                th += 0.1; // the amount of rotation angle
        }
        if (rFlag == 2) //Rotate Around Fixed Point
        {
                trX = x;// SET the translation to wherever the user says
                trY = y;
                th += 0.1; // the amount of rotation angle
                glColor3f(0, 0, 1);// mark the user coordinate as blue dot
                draw_pixel(x, y);// plot the user coordinate - blue colour
```

```
        }
        glTranslatef(trX, trY, 0.0);// ACTUAL translation +ve
        glRotatef(th, 0.0, 0.0, 1.0);// rotate
        glTranslatef(-trX, -trY, 0.0);// ACTUAL translation -ve
        triangle(trX,trY); // what to rotate ? – TRIANGLE boss
        glutPostRedisplay(); // call display function again and again
        glutSwapBuffers();// show the output
}
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;
}
void main(int argc, char** argv)
{
        printf("Enter Fixed Points (x,y) for Roration: \n");
        scanf_s("%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(); // run forever
}
```

OUTPUT:


C:\Users\USER\source\repos\triangle\Debug\triangle.exe
Enter Fixed Points (x,y) for Roration:
20 30
Create and Rotate Triangle
Rotate around ORIGIN
Rotate around FIXED POINT
Stop Rotation
Type here to search
29°C
ENG
03:54 PM
17-10-2022


C:\Users\USER\source\repos\triangle\Debug\triangle.exe
Enter Fixed Points (x,y) for Roration:
20 30
Create and Rotate Triangle
Type here to search
29°C
ENG
03:54 PM
17-10-2022

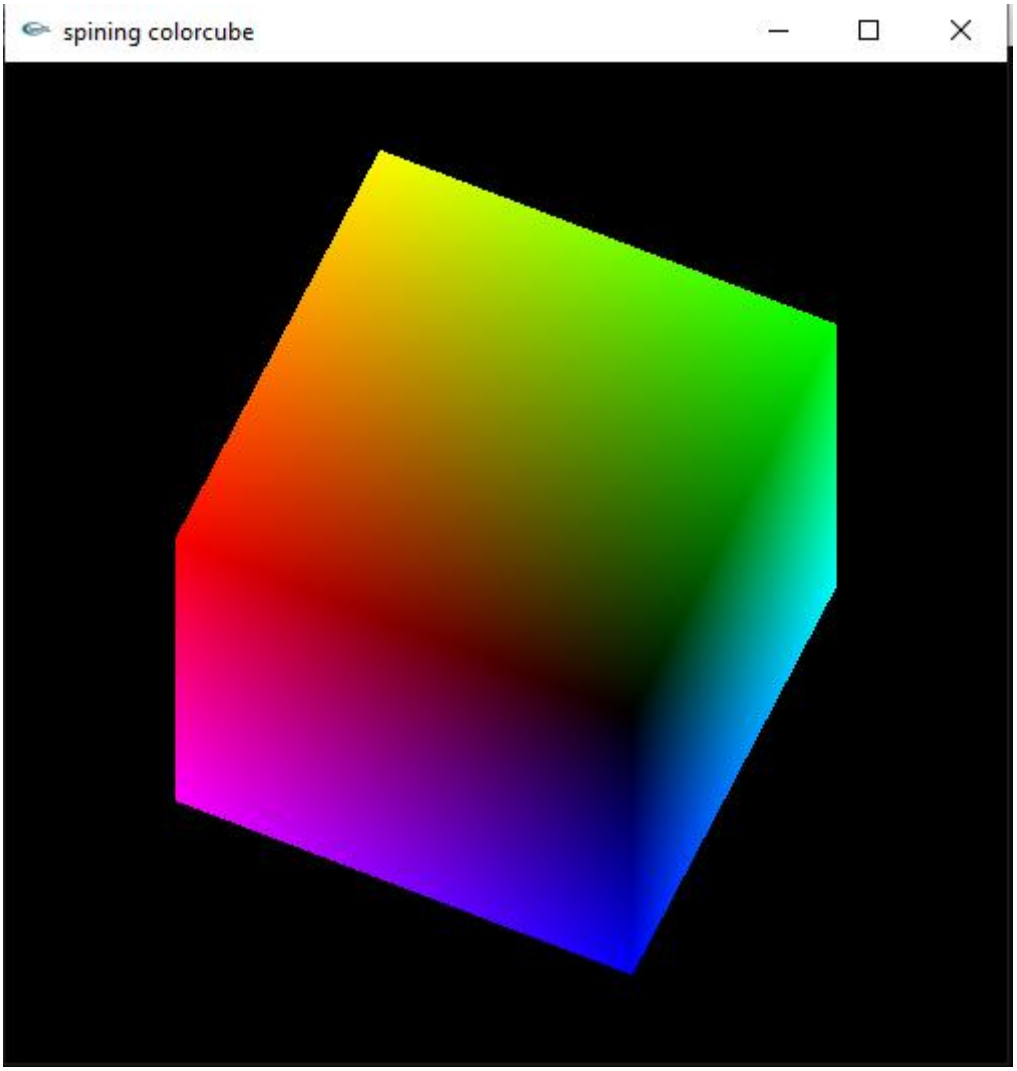
3.Draw a colour cube and spin it using OpenGL transformation matrices.
Refer:Text-2: Modelling a Coloured Cube

```
#include <stdlib.h>
#include <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}
};
//cube vertices
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 face(int a, int b, int c, int d)
{
        glBegin(GL_POLYGON);
        glColor3fv(colors[a]);
        glVertex3fv(vertices[a]);

        glColor3fv(colors[b]);
        glVertex3fv(vertices[b]);

        glColor3fv(colors[c]);
        glVertex3fv(vertices[c]);

        glColor3fv(colors[d]);
        glVertex3fv(vertices[d]);
```

```
        glEnd();
}
void colorcube(void)
{
        face(4, 5, 6, 7);//front face
        face(0, 1, 2, 3); //back
        face(0, 4, 7, 3);//left
        face(1, 2, 6, 5);//right
        face(2, 3, 7, 6);//top
        face(0, 4, 5, 1);//bottom
}
GLfloat theta[] = { 0.0,0.0,0.0 };
GLint axis = 2;
void display(void)
{
        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();
        glutSwapBuffers();
}
void spinCube()
{
        theta[axis] += 0.15;
        if (theta[axis] > 360.0) theta[axis] -= 360.0;
        glutPostRedisplay();
}
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;
}
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 MyInit()
{
        glClearColor(0, 0, 0, 1);
        glEnable(GL_DEPTH_TEST);//enable z buffer to view back buffer
}
void main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(500, 500);
        glutCreateWindow("spining colorcube");
        MyInit();
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glutIdleFunc(spinCube);
        glutMouseFunc(mouse);
        glutMainLoop();
}
```

OUTPUT:


spining colorcube

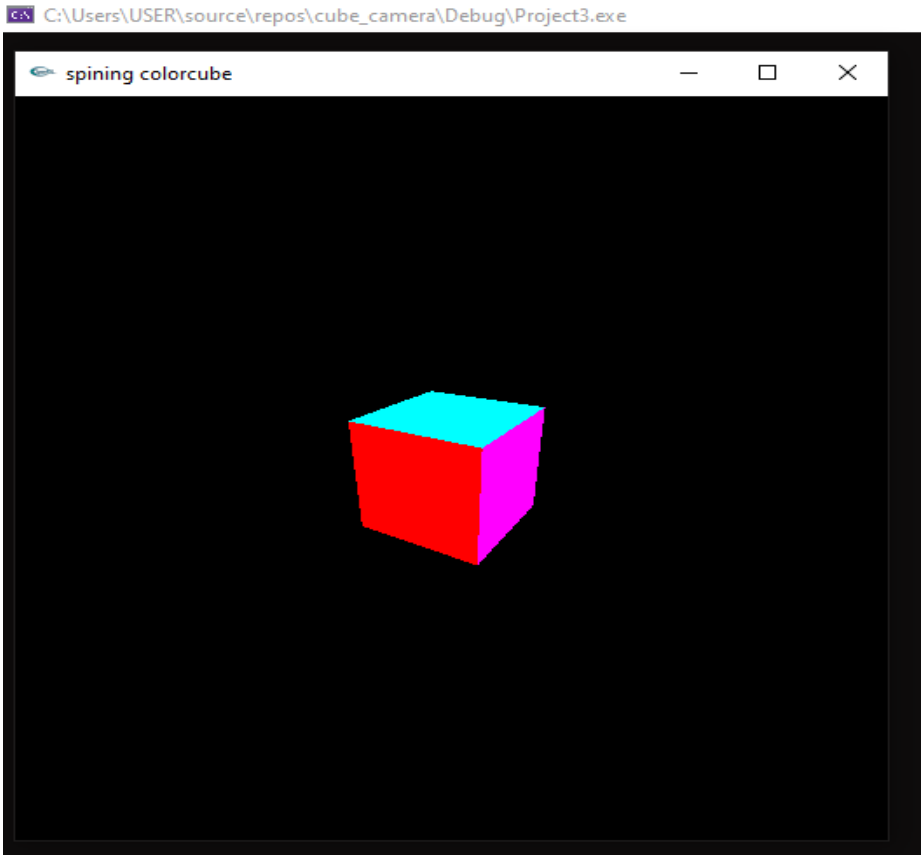
4. Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.

Refer:Text-2: Topic: Positioning of Camera

```
#include <stdlib.h>
#include <glut.h>
static GLdouble viewer[] = { 0.0,0.0,5.0 };
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}
};
//cube vertices

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 face(int a, int b, int c, int d)
{
        glBegin(GL_POLYGON);
        //glColor3fv(colors[a]);

        glVertex3fv(vertices[a]);
        //glColor3fv(colors[b]);

        glVertex3fv(vertices[b]);
        //glColor3fv(colors[c]);

        glVertex3fv(vertices[c]);
```

```
        //glColor3fv(colors[d]);

        glVertex3fv(vertices[d]);
        glEnd();
}
void colorcube(void)
{
        glColor3f(1.0, 0.0, 0.0);
        face(4, 5, 6, 7);//front

        glColor3f(0.0, 1.0, 0.0);
        face(0, 3, 2, 1);//back

        glColor3f(1.0, 1.0, 0.0);
        face(0, 4, 7, 3);//left

        glColor3f(0.0, 0.0, 1.0);
        face(5, 4, 0, 1);//bottom

        glColor3f(0.0, 1.0, 1.0);
        face(2, 3, 7, 6);//top
        glColor3f(1.0, 0.0, 1.0);
        face(1, 2, 6, 5);//right
}
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);
        colorcube();
        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);
        glMatrixMode(GL_MODELVIEW);
}
void MyInit()
{
        glClearColor(0, 0, 0, 1);
        glEnable(GL_DEPTH_TEST);//enable z buffer to view back buffer
}
void main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(500, 500);
        glutCreateWindow("spining colorcube");
        MyInit();
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glutKeyboardFunc(keys);
        glutMainLoop();
}
```


C:\Users\USER\source\repos\cube_camera\Debug\Project3.exe
spining colorcube

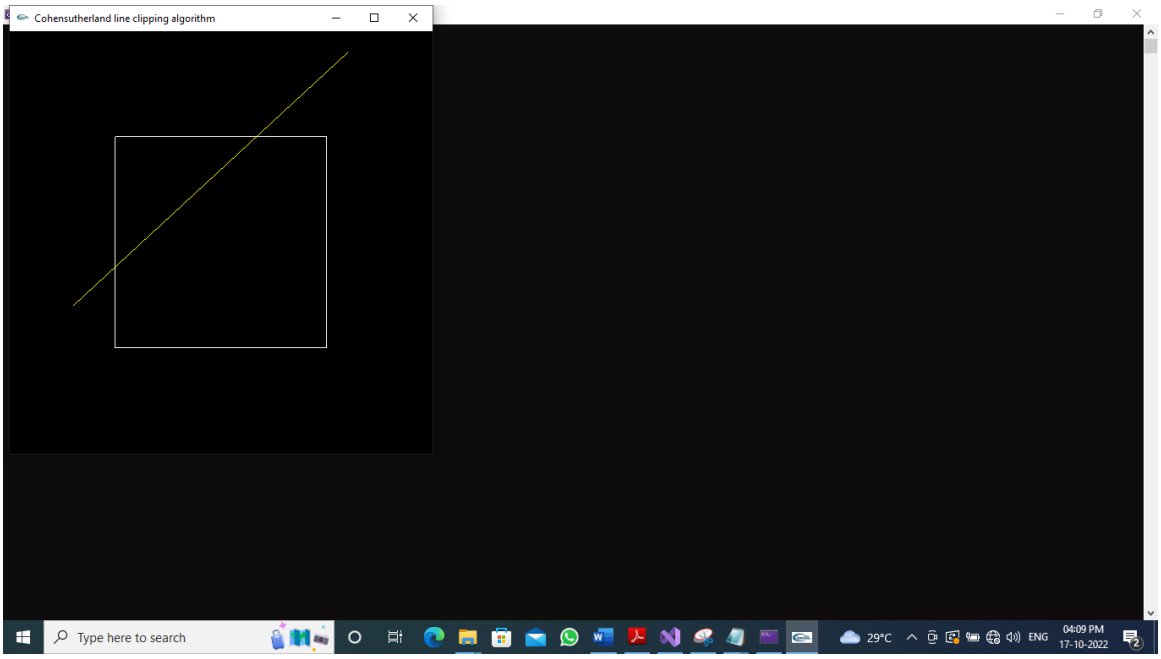
5. Clip a lines using Cohen-Sutherland algorithm
Refer:Text-1: Chapter 6.7
Refer:Text-2: Chapter 8

```
#include<stdio.h>
#include<glut.h>
GLfloat xmin = -0.5, ymin = -0.5, xmax = 0.5, ymax = 0.5;
GLfloat x1 = -0.7, y1 = -0.3, x2 = 0.6, y2 = 0.9;
int LEFT = 1, RIGHT = 2, BOTTOM = 4, TOP = 8;
int c1, c2;
int clip_flag = 0, flag = 1;
int get_code(GLfloat x, GLfloat y)//to compute the region code
{
        int code = 0;
        if (y > ymax)
                code = code | TOP;
        else if (y < ymin)
                code = code | BOTTOM;
        if (x > xmax)
                code = code | RIGHT;
        else if (x < xmin)
                code = code | LEFT;
        return code;
}
void clip()
{
        int c;
        GLfloat x, y;
        if (c1)
                c = c1;
        else
                c = c2;

        if (c & LEFT)
        {
                x = xmin;
                y = y1 + (y2 - y1) * ((xmin - x1) / (x2 - x1));
        }
        if (c & RIGHT)
        {
                x = xmax;
                y = y1 + (y2 - y1) * ((xmax - x1) / (x2 - x1));
        }
        if (c & BOTTOM)
```

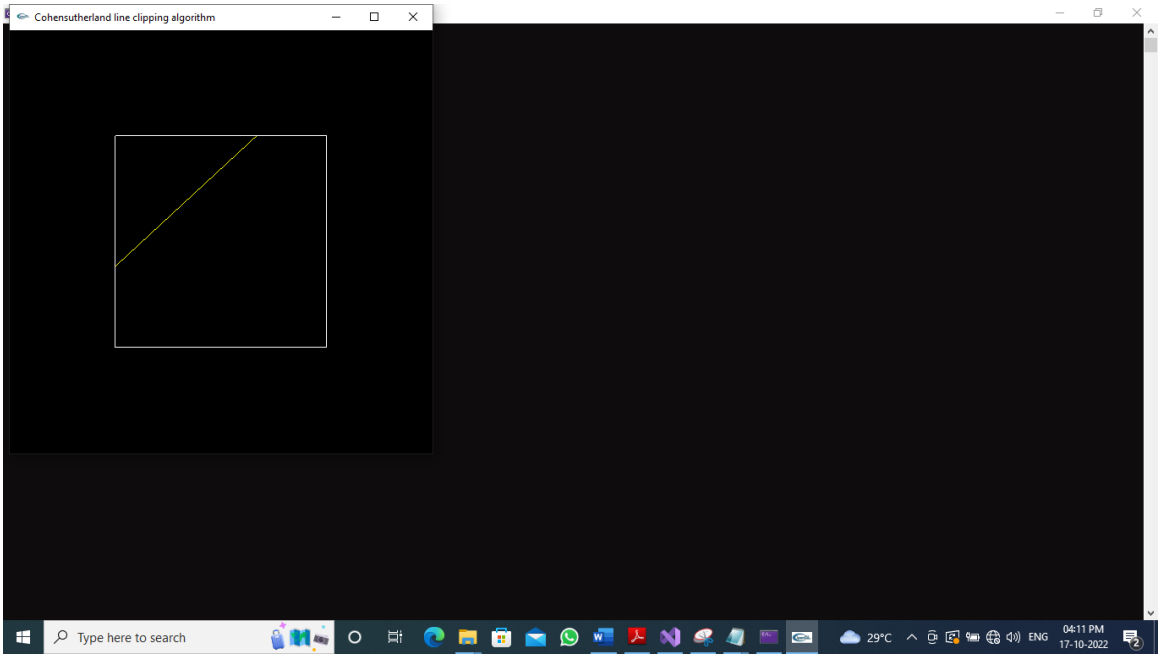
```
        {
                y = ymin;
                x = x1 + (x2 - x1) * ((ymin - y1) / (y2 - y1));
        }
        if (c & TOP)
        {
                y = ymax;
                x = x1 + (x2 - x1) * ((ymax - y1) / (y2 - y1));
        }
        if (c == c1)
        {
                x1 = x;
                y1 = y;
        }
        else
        {
                x2 = x;
                y2 = y;
        }
}
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1, 1, 1);
        glBegin(GL_LINE_LOOP);
        glVertex2f(xmin, ymin);
        glVertex2f(xmax, ymin);
        glVertex2f(xmax, ymax);
        glVertex2f(xmin, ymax);
        glEnd();
        glColor3f(1, 1, 0);
        if (flag == 1)
        {
                glBegin(GL_LINES);
                glVertex2f(x1, y1);
                glVertex2f(x2, y2);
                glEnd();
        }
        while (1 && clip_flag == 1)
        {
                c1 = get_code(x1, y1);
                c2 = get_code(x2, y2);
                if ((c1 | c2) == 0)
```

```
                    break;
                else if ((c1 & c2) != 0)
                {
                    flag = 0;
                    break;
                }
                else clip();
        }
        glFlush();
}
void key(unsigned char ch, int x, int y)
{
        clip_flag = 1;
        glutPostRedisplay();
}
void main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitWindowSize(500, 500);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutCreateWindow("Cohensutherland line clipping algorithm");
        glutDisplayFunc(display);
        glutKeyboardFunc(key);
        glutMainLoop();
}
```

OUTPUT:



Press any key


Cohensutherland line clipping algorithm
Type here to search
29°C
ENG
04:11 PM
17-10-2022

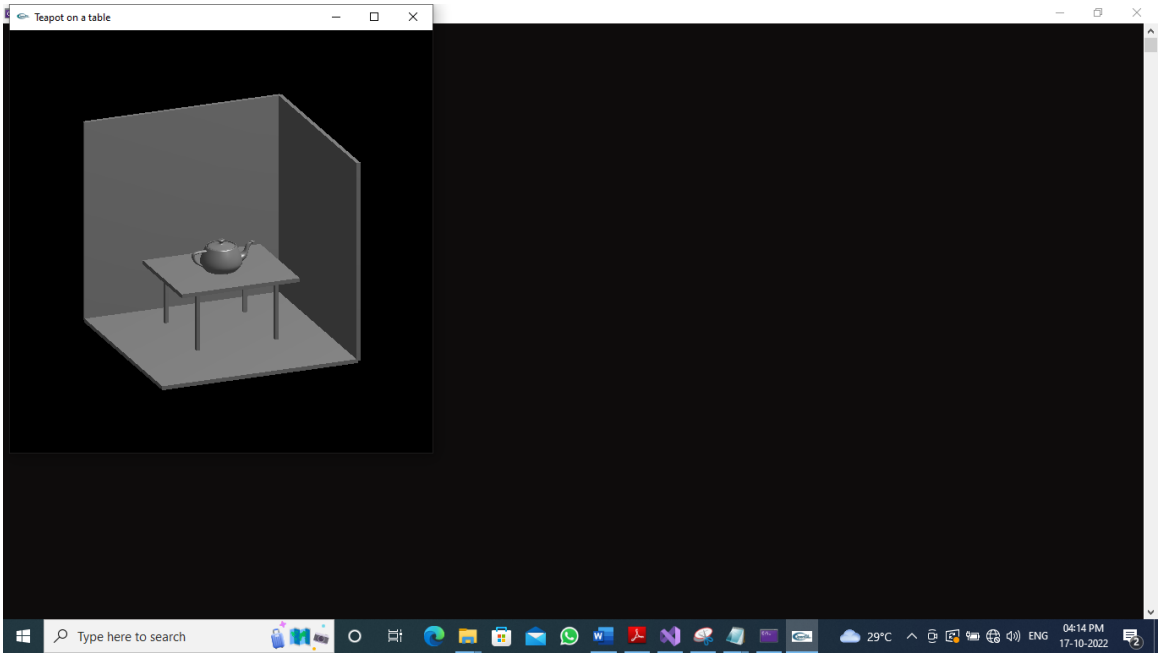
6.To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.
Refer:Text-2: Topic: Lighting and Shading

```
#include<glut.h>
void teapot(GLfloat x, GLfloat y, GLfloat z)
{
        glPushMatrix(); //save the current state
        glTranslatef(x, y, z); //move your item appropriately
        glutSolidTeapot(0.1); //render your teapot
glPopMatrix(); //get back your state with the recent changes that you have done
}
void tableTop(GLfloat x, GLfloat y, GLfloat z) // table top which is actually a CUBE
{
        glPushMatrix();
        glTranslatef(x, y, z);
        glScalef(0.6, 0.02, 0.5);
        glutSolidCube(1);
        glPopMatrix();
}
void tableLeg(GLfloat x, GLfloat y, GLfloat z) // table leg which is actually a CUBE
{
        glPushMatrix();
        glTranslatef(x, y, z);
        glScalef(0.02, 0.3, 0.02);
        glutSolidCube(1);
        glPopMatrix();
}
void wall(GLfloat x, GLfloat y, GLfloat z) // wall which is actually a CUBE
{
        glPushMatrix();
        glTranslatef(x, y, z);
        glScalef(1, 1, 0.02);
        glutSolidCube(1);
        glPopMatrix();
}
void light() // set the lighting arrangements
{
        GLfloat mat_ambient[] = { 1, 1, 1, 1 }; // ambient colour
        GLfloat mat_diffuse[] = { 0.5, 0.5, 0.5, 1 };
        GLfloat mat_specular[] = { 1, 1, 1, 1 };
        GLfloat mat_shininess[] = { 50.0f }; // shininess value
```

```
        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 light_position[] = { 2, 6, 3, 1 };
        GLfloat light_intensity[] = { 0.7, 0.7, 0.7, 1 };
        glLightfv(GL_LIGHT0, GL_POSITION, light_position);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, light_intensity);
}
void display()
{
        GLfloat teapotP = -0.07, tabletopP = -0.15, tablelegP = 0.2, wallP = 0.5;
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        gluLookAt(-2, 2, 5, 0, 0, 0, 0, 1, 0); // camera position & viewing
        light(); //Adding light source to your project
        teapot(0, teapotP, 0); //Create teapot
        tableTop(0, tabletopP, 0); //Create table's top
        tableLeg(tablelegP, -0.3, tablelegP); //Create 1st leg
        tableLeg(-tablelegP, -0.3, tablelegP); //Create 2nd leg
        tableLeg(-tablelegP, -0.3, -tablelegP); //Create 3rd leg
        tableLeg(tablelegP, -0.3, -tablelegP); //Create 4th leg
        wall(0, 0, -wallP); //Create 1st wall
        glRotatef(90, 1, 0, 0);
        wall(0, 0, wallP); //Create 2nd wall
        glRotatef(90, 0, 1, 0);
        wall(0, 0, wallP); //Create 3rd wall
        glFlush(); // show the output to the user
}
void init()
{
        glClearColor(0, 0, 0, 1); // black colour background
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-1, 1, -1, 1, -1, 10);
        glMatrixMode(GL_MODELVIEW);
}
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(0, 0);
```

```
        glutCreateWindow("Teapot on a table");
        init();
        glutDisplayFunc(display);
        glEnable(GL_LIGHTING); // enable the lighting properties
        glEnable(GL_LIGHT0); // enable the light source
glShadeModel(GL_SMOOTH); // for smooth shading (select flat or smooth shading)
glEnable(GL_NORMALIZE); // If enabled and no vertex shader is active, normal
//vectors  are normalized to unit length after transformationand beforelighting.
glEnable(GL_DEPTH_TEST); // do depth comparisons and update the depth buffer.
        glutMainLoop();
}
```

OUTPUT:

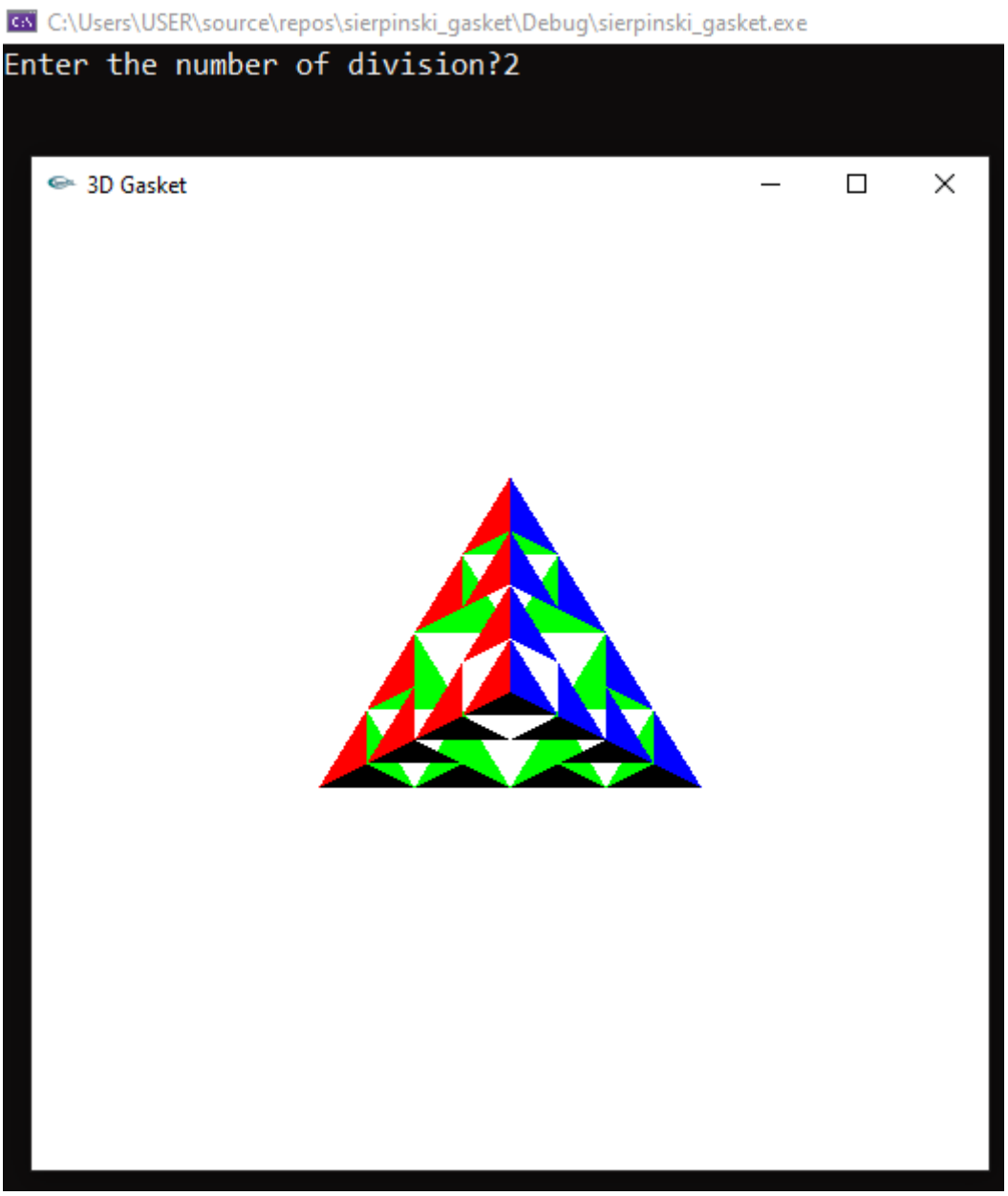


7.Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user.
Refer: Text-2: Topic: sierpinski gasket.

```
#include<stdlib.h>
#include<stdio.h>
#include<glut.h>

typedef float point[3];
point v[] = { {0.0,0.0,1.0},
              {0.0,0.9,0.3},
              {-0.8,-0.4,-0.3},
              {0.8,-0.4,-0.3} };
int n;
void triangle(point a, point b, point c)
{
    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)
    {
        for (j = 0; j < 3; j++)
            v1[j] = (a[j] + b[j]) / 2;
        for (j = 0; j < 3; j++)
            v2[j] = (a[j] + c[j]) / 2;
        for (j = 0; j < 3; j++)
            v3[j] = (b[j] + c[j]) / 2;
        divide_triangle(a, v1, v2, m - 1);
        divide_triangle(c, v2, v3, m - 1);
        divide_triangle(b, v3, v1, m - 1);
    }
    else(triangle(a, b, c));
}
void tetrahedron(int m)
{
    glColor3f(1.0, 0.0, 0.0);
    divide_triangle(v[0], v[1], v[2], m);
```

```
        glColor3f(0.0, 1.0, 0.0);
        divide_triangle(v[3], v[2], v[1], m);
        glColor3f(0.0, 0.0, 1.0);
        divide_triangle(v[0], v[3], v[1], m);
        glColor3f(0.0, 0.0, 0.0);
        divide_triangle(v[0], v[2], v[3], m);
}
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, -.0, 2.0, -10.0,
10.0);
        glMatrixMode(GL_MODELVIEW);
        glutPostRedisplay();
}
int main(int argc, char** argv)
{
        printf("Enter the number of division?");
        scanf_s("%d", &n);
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(500, 500);
        glutCreateWindow("3D Gasket");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glEnable(GL_DEPTH_TEST);
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glutMainLoop();
}
```

OUTPUT:


C:\Users\USER\source\repos\sierpinski_gasket\Debug\sierpinski_gasket.exe
Enter the number of division?2
3D Gasket

8.Develop a menu driven program to animate a flag using Bezier Curve algorithm
Refer: Text-1: Chapter 8-10

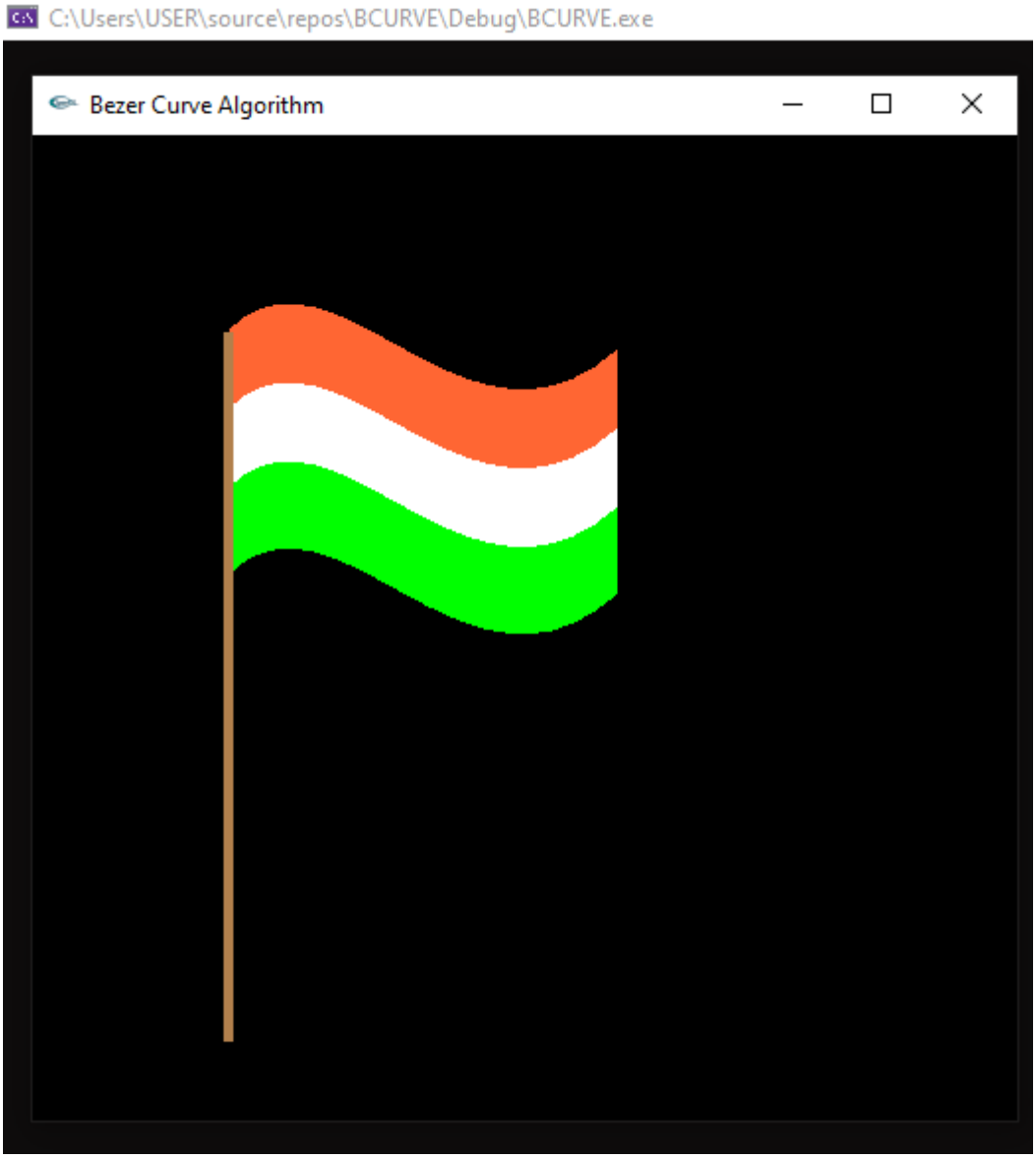
```
#include<stdio.h>
#include<glut.h>
#include<math.h>
#define Pi 3.1416
typedef struct point
{
	GLfloat x, y, z;
};
void bino(int n, int* c)
{
	int k, j;
	for (k = 0; k <= n; k++)
	{
		c[k] = 1;
		for (j = n; j >= k + 1; j--)
			c[k] *= j;
		for (j = n - k; j >= 2; j--)
			c[k] /= j;
	}
}
void computebezPt(float 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} };
        // for animating the flag
        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 += 25 * sin((theta - 30) * Pi / 180.0);

        theta += 0.2; //animating speed
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0, 1.0, 1.0);
        glPointSize(5);
        glPushMatrix();
        glLineWidth(5);
        glColor3f(1.0, 0.4, 0.2); //Indian flag: Saffron color code
        for (int i = 0; i < 50; i++)
        {
                glTranslatef(0.0, -0.8, 0.0);
                bezier(ctrlPts, nctrlPts, nBcPts);
        }
        glColor3f(1, 1, 1);
        for (int i = 0; i < 50; i++)
        {
                glTranslatef(0, -0.8, 0);
                bezier(ctrlPts, nctrlPts, nBcPts);
        }
```

```
        glColor3f(0, 1, 0);
        for (int i = 0; i < 50; i++)
        {
                glTranslatef(0, -0.8, 0);
                bezier(ctrlPts, nctrlPts, nBcPts);
        }
        glPopMatrix();
        glColor3f(0.7, 0.5, 0.3); //pole colour
        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("Bezer Curve Algorithm");
        init();
        glutDisplayFunc(display);
        glutMainLoop();
        return 0;
}
```

O

OUTPUT:


C:\Users\USER\source\repos\BCURVE\Debug\BCURVE.exe
Bezer Curve Algorithm

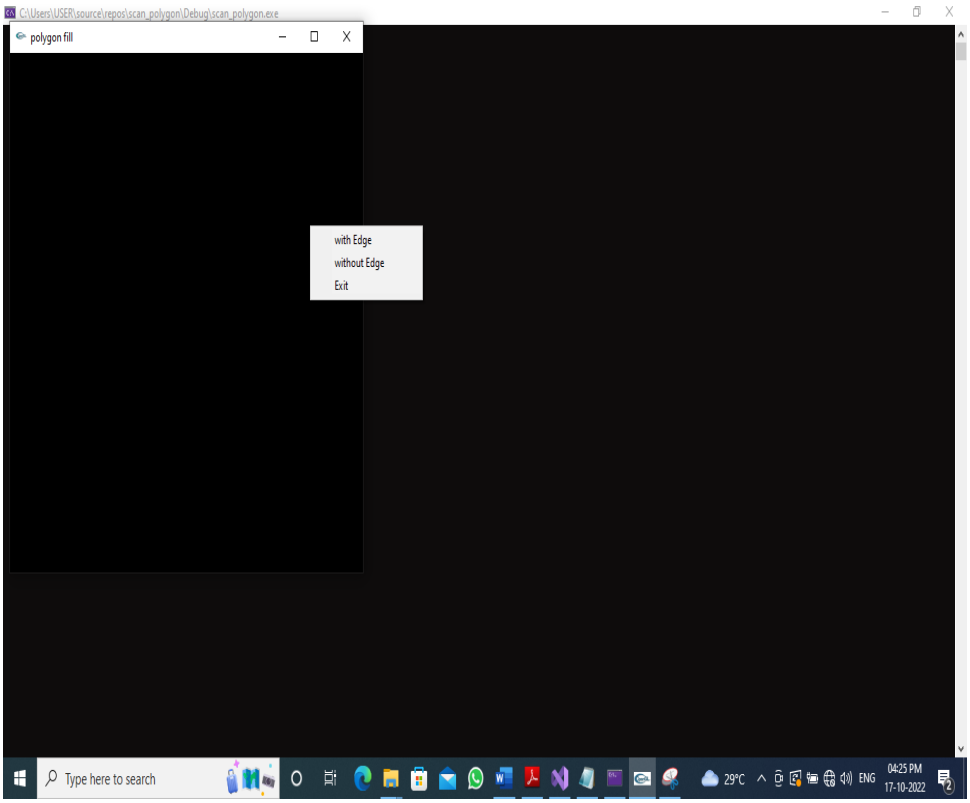
9. Develop a menu driven program to fill the polygon using scan line algorithm

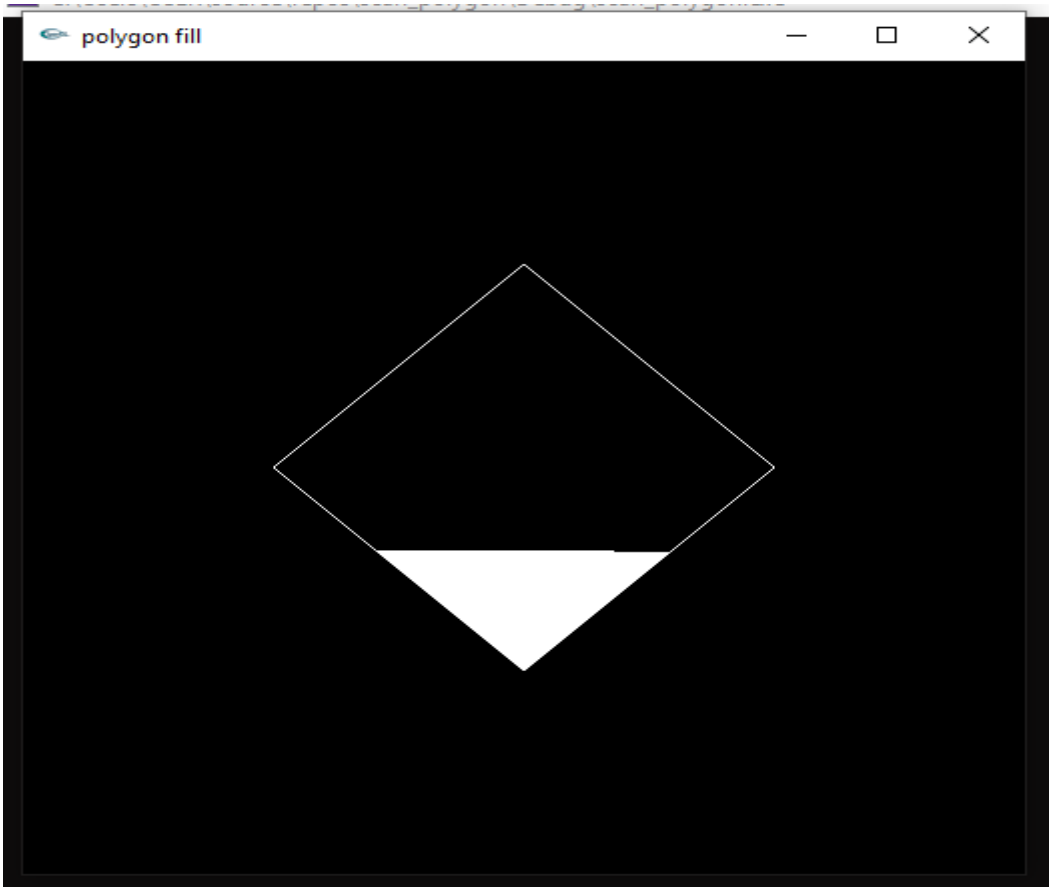
```
#include<stdio.h>
#include<glut.h>
#include<stdlib.h>
float LE[500], RE[500];
int  Edgeflag = 0, FillFlag = 0;
void Menu(int id)
{
        if (id == 1)
                Edgeflag = 1;
        else if (id == 2)
                Edgeflag = 0;
        else if (id == 3)
                exit(0);
        FillFlag = 1;
        glutPostRedisplay();
}
void MyInit()
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0, 500, 0, 500);
        glMatrixMode(GL_MODELVIEW);
        glutCreateMenu(Menu);
        glutAddMenuEntry("with Edge", 1);
        glutAddMenuEntry("without Edge", 2);
        glutAddMenuEntry("Exit", 3);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
}
void intersection_point(GLint x1, GLint y1, GLint x2, GLint y2)
{
        float M, x;
        int t, y;
        if (y1 > y2)
        {
                t = x1;
                x1 = x2;
                x2 = t;
                t = y1;
                y1 = y2;
                y2 = t;
        }
        if ((y2 - y1) == 0)
```

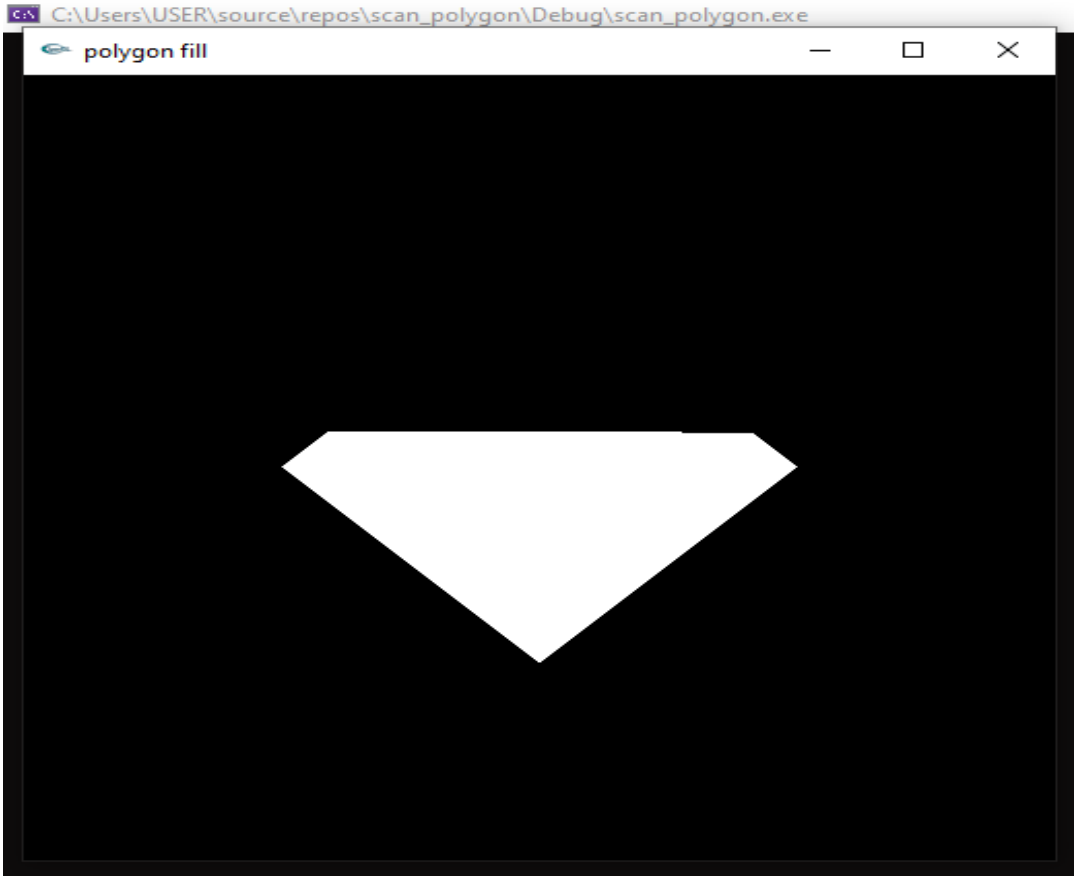
```
                M = (x2 - x1);
        else
                M = (x2 - x1) / (y2 - y1);
        x = x1;
        for (y = y1; y <= y2; y++)
        {
                if (x < LE[y])
                        LE[y] = x;
                if (x > RE[y])
                        RE[y] = x;
        x = x + M;
        }
}
void Draw()
{
        GLint P1[2] = { 125,250 }, P2[2] = { 250,125 }, P3[2] = { 375,250 }, P4[2] = {
250,375 };
        glClear(GL_COLOR_BUFFER_BIT);
        for (int i = 0; i < 500; i++)
        {
                LE[i] = 500;
                RE[i] = 0;
        }
        if (Edgeflag == 1)
        {
                glBegin(GL_LINE_LOOP);
                glVertex2iv(P1);
                glVertex2iv(P2);
                glVertex2iv(P3);
                glVertex2iv(P4);
                glEnd();
        }
        intersection_point(P1[0], P1[1], P2[0], P2[1]);
        intersection_point(P2[0], P2[1], P3[0], P3[1]);
        intersection_point(P3[0], P3[1], P4[0], P4[1]);
        intersection_point(P4[0], P4[1], P1[0], P1[1]);
        if (FillFlag == 1)
        {
                for (int y = 0; y < 500; y++)
                {
                        for (int x = LE[y]; x < RE[y]; x++)
                        {
                                glBegin(GL_POINTS);
```

```
                                glVertex2i(x, y);
                                glEnd();
                                glFlush();
                        }
                }
        }
        glFlush();
}
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(10, 50);
        glutCreateWindow("polygon fill");
        MyInit();
        glutDisplayFunc(Draw);
        glutMainLoop();
        return 0;
}
```

OUTPUT:




polygon fill


C:\Users\USER\source\repos\scan_polygon\Debug\scan_polygon.exe
polygon fill