B.L.D.E.A's Vachana Pitamaha

Dr. P. G. Halakatti College of Engineering and Technology, Vijayapur-586103.

Department of Computer Science and Engineering

Laboratory Manual

Laboratory Name : **Computer Graphics Laboratory With Mini Project**

Laboratory Code : **15CSL68**

Class : **VI Semester B.E.**

2017-18

Institute Vision and Mission

Vision: To be a trend setting institution in Technical Education and Research, providing highly competent, efficient manpower to meet the ever-changing needs of the country, industry and the society.

Mission: To be an ideal institution providing quality Technical Education and Training to students in tune with the evolving challenges and social needs through a flexible and innovative learning process, enabling the students to excel in their professions and careers with a high degree of integrity and ethical standards.

Department Vision and Mission

Vision: To be a trend setting department in Technical Education and Research, providing highly competent, efficient manpower to meet the ever-changing need of the country, industry and the society.

Mission: To be an ideal department providing Technical Education and Training to students in tune with the evolving challenges and social needs through a flexible and innovative learning process, enabling the students to excel in their professions and careers with a high degree of integrity and ethical standards.

Programme Educational Objectives (PEOs)

I. A Graduate will be a successful IT professional and function effectively in multidisciplinary domains.

II. A Graduate will have the perspective of lifelong learning for continuous improvement of knowledge in Computer Science & Engineering, higher studies, and research.

III. A Graduate will be able to respond to local, national and global issues by imparting his/her knowledge of Computer Science & Engineering in Educational, Government, Financial and Private sectors.

IV. A Graduate will be able to function effectively as an individual, as a team member and as a team leader with highest professional and ethical standards.

Programme Outcomes (POs)

The programme enables students to achieve by the time of graduation,

a. An ability to apply knowledge of mathematics, computer science and engineering.
b. An ability to design and conduct experiments, as well as to analyze and interpret data in computer science and engineering.
c. An ability to design a computer based system, component, or product to meet desired needs within realistic constraints such as economic, environmental, social, political, ethical, health, safety, manufacturability, and sustainability.
d. An ability to function as an individual and as a member or leader in diverse or multidisciplinary teams.
e. An ability to identify, formulate and solve complex problems of computer science and engineering.
f. An understanding of professional and ethical responsibility.
g. Communicate effectively with various engineering communities, professional bodies and society at large.
h. The broad education is necessary in computer science and engineering to apply management principles to solve complex engineering problem in a global, economic, environmental and societal context.

i. A recognition of the needs, and an ability to engage in life-long learning in computer science and engineering.
j. A knowledge of contemporary issues in computer science and engineering.
k. An ability to use techniques, skills, and modern computer science and engineering tools necessary for engineering practice.

COMPUTER GRAPHICS LABORATORY WITH MINI PROJECT

Subject Code: 15CSL68 **I.A. Marks : 20**

Hours/Week : 03 **Exam Hours: 03**

Total Hours : 40 **Exam Marks: 80**

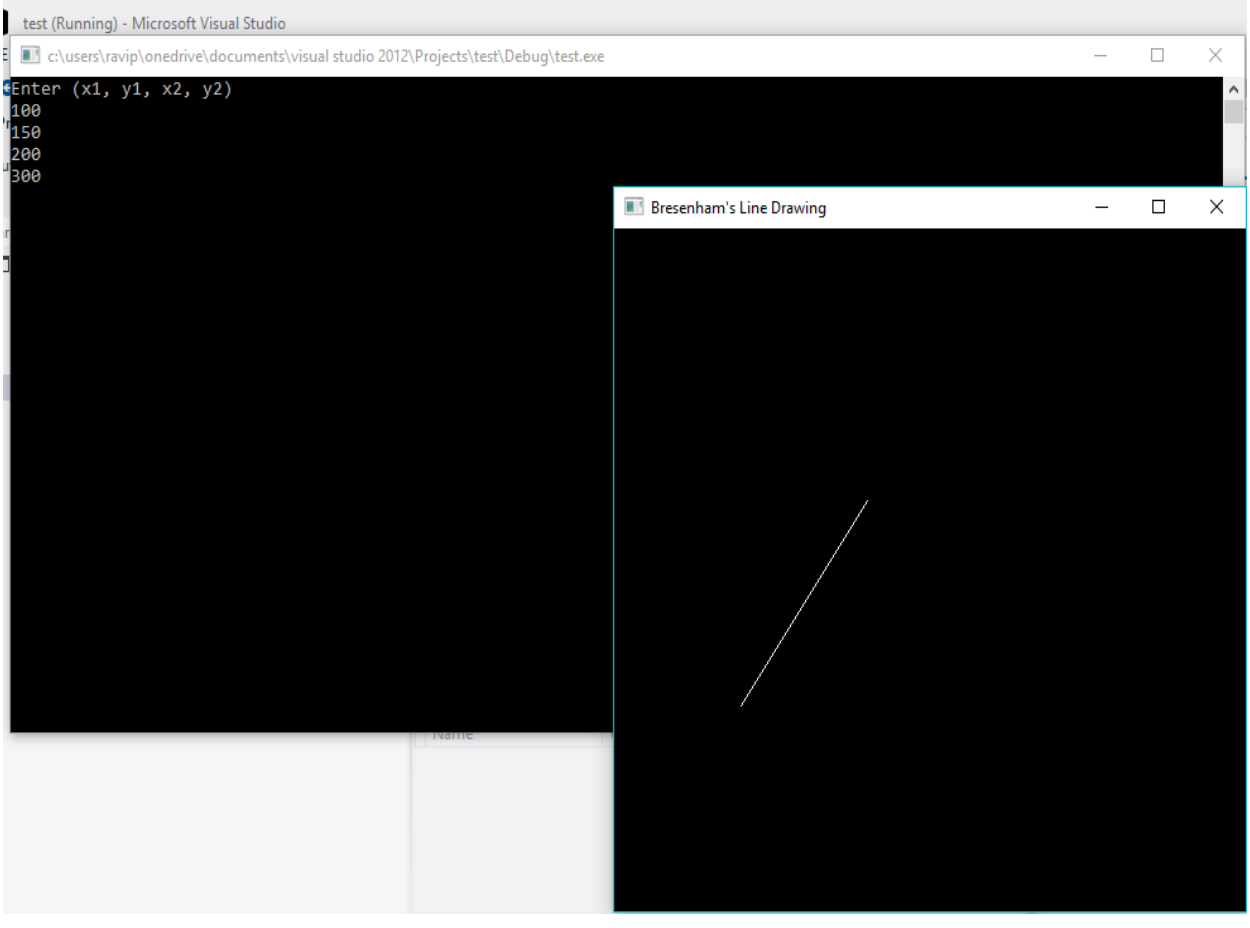
PROGRAM 1: Implement Brenham's line drawing algorithm for all types of slope.

```
#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:


test (Running) - Microsoft Visual Studio
c:\users\ravip\onedrive\documents\visual studio 2012\Projects\test\Debug\test.exe
Enter (x1, y1, x2, y2)
100
150
200
300
Bresenham's Line Drawing

PROGRAM 2: Create and rotate a triangle about the origin and a fixed point

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
GLsizei winWidth = 600, winHeight = 600;   /* Set initial display-window size. */
GLfloat xwcMin = 0.0, xwcMax = 225.0;   /* Set range for world coordinates. */
GLfloat ywcMin = 0.0, ywcMax = 225.0;

class wcPt2D
{
        public:
        GLfloat x, y;
};

typedef GLfloat Matrix3x3 [3][3];
Matrix3x3 matComposite;
const GLdouble pi = 3.14159;

void init (void)
{
        /* Set color of display window to white. */
        glClearColor (1.0, 1.0, 1.0, 0.0);
}

/* Construct the 3 x 3 identity matrix. */
void matrix3x3SetIdentity (Matrix3x3 matIdent3x3)

{
        GLint row, col;
        for (row = 0; row < 3; row++)
        for (col = 0; col < 3; col++)
```

```
        matIdent3x3 [row][col] = (row == col);
}

void matrix3x3PreMultiply (Matrix3x3 m1, Matrix3x3 m2)
{
        GLint row, col;
        Matrix3x3 matTemp;
        for (row = 0; row < 3; row++)
                for (col = 0; col < 3 ; col++)
                        matTemp [row][col] = m1 [row][0] * m2 [0][col] + m1
                        [row][1] * m2 [1][col] + m1 [row][2] * m2 [2][col];
        for (row = 0; row < 3; row++)
                for (col = 0; col < 3; col++)
                        m2 [row][col] = matTemp [row][col];
}

void translate2D (GLfloat tx, GLfloat ty)
{
        Matrix3x3 matTransl;
        /* Initialize translation matrix to identity. */
        matrix3x3SetIdentity (matTransl);

        matTransl [0][2] = tx;
        matTransl [1][2] = ty;
        /* Concatenate matTransl with the composite matrix. */
        matrix3x3PreMultiply (matTransl, matComposite);
}


void rotate2D (wcPt2D pivotPt, GLfloat theta)
{
        Matrix3x3 matRot;
        /* Initialize rotation matrix to identity. */
        matrix3x3SetIdentity (matRot);
        matRot [0][0] = cos (theta);
```

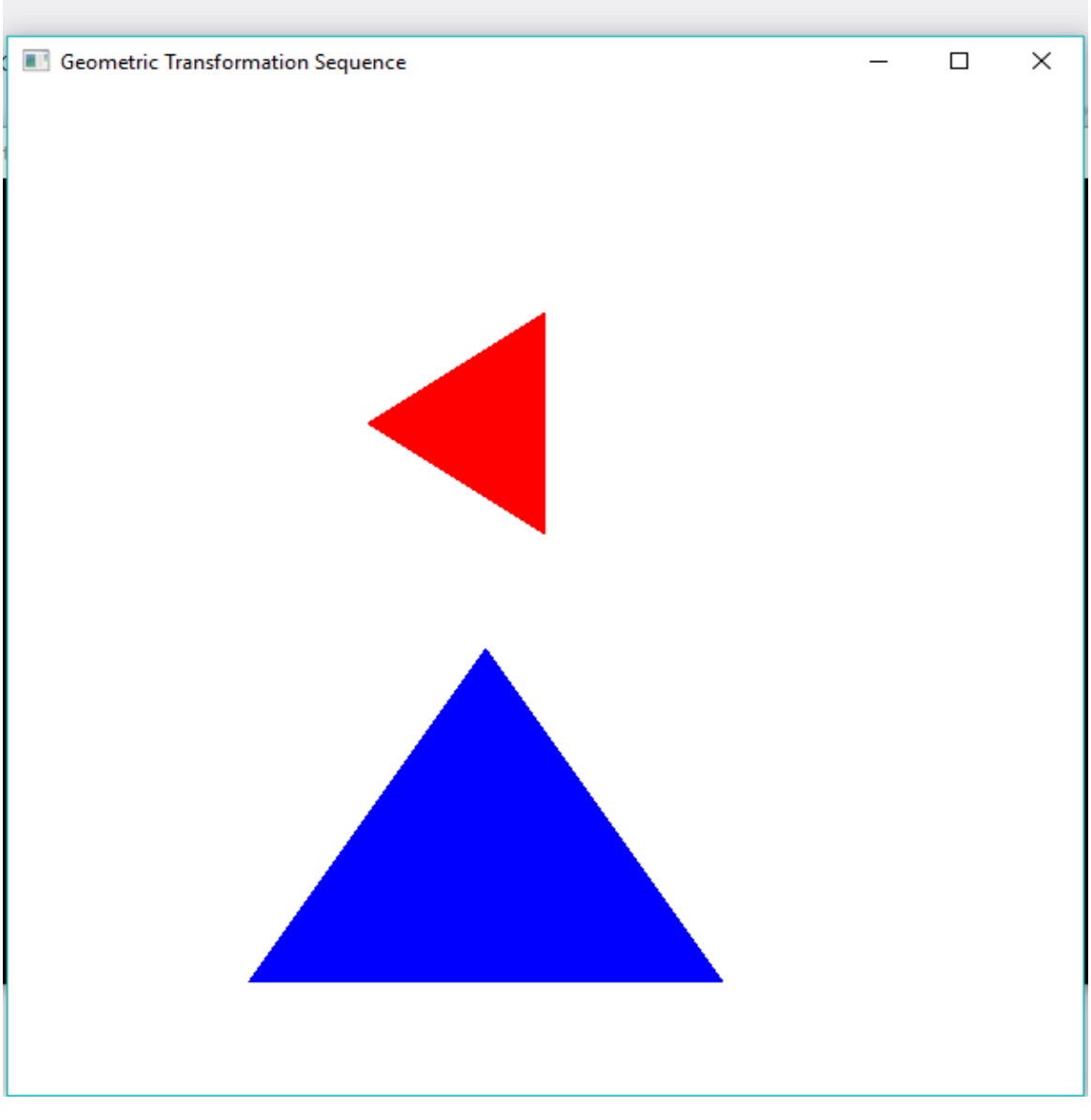
```
        matRot [0][1] = -sin (theta);
        matRot [0][2] = pivotPt.x * (1 - cos (theta)) +
                                                pivotPt.y * sin (theta);
        matRot [1][0] = sin (theta);
        matRot [1][1] = cos (theta);
        matRot [1][2] = pivotPt.y * (1 - cos (theta)) -
                                                pivotPt.x * sin (theta);

        /* Concatenate matRot with the composite matrix. */
        matrix3x3PreMultiply (matRot, matComposite);
}

void scale2D (GLfloat sx, GLfloat sy, wcPt2D fixedPt)
        {
                Matrix3x3 matScale;
                /* Initialize scaling matrix to identity. */
                matrix3x3SetIdentity (matScale);
                matScale [0][0] = sx;
                matScale [0][2] = (1 - sx) * fixedPt.x;
                matScale [1][1] = sy;
                matScale [1][2] = (1 - sy) * fixedPt.y;
                /* Concatenate matScale with the composite matrix. */
                matrix3x3PreMultiply (matScale, matComposite);
        }
                /* Using the composite matrix, calculate transformed coordinates. */
void transformVerts2D (GLint nVerts, wcPt2D * verts)
{
        GLint k;
        GLfloat temp;
        for (k = 0; k < nVerts; k++)
         {
                temp = matComposite [0][0] * verts [k].x + matComposite
                [0][1] * verts [k].y + matComposite [0][2];
```

```
            verts [k].y = matComposite [1][0] * verts [k].x + matComposite
            [1][1] * verts [k].y + matComposite [1][2];

            verts [k].x = temp;
        }
}

void triangle (wcPt2D *verts)
{
        GLint k;
        glBegin (GL_TRIANGLES);
        for (k = 0; k < 3; k++)
        glVertex2f (verts [k].x, verts [k].y);
        glEnd ( );
}

void displayFcn (void)
{
        /* Define initial position for triangle. */
        GLint nVerts = 3;
        wcPt2D verts [3] = { {50.0, 25.0}, {150.0, 25.0}, {100.0, 100.0} };
        /* Calculate position of triangle centroid. */
        wcPt2D centroidPt;
        GLint k, xSum = 0, ySum = 0;
        for (k = 0; k < nVerts; k++)
         {
            xSum += verts [k].x;
            ySum += verts [k].y;
        }
        centroidPt.x = GLfloat (xSum) / GLfloat (nVerts);
        centroidPt.y = GLfloat (ySum) / GLfloat (nVerts);
        /* Set geometric transformation parameters. */

        wcPt2D pivPt,fixedPt;
        pivPt = centroidPt;
        fixedPt = centroidPt;
```

```
    GLfloat tx = 0.0, ty = 100.0;
    GLfloat sx = 0.5, sy = 0.5;
    GLdouble theta = pi/2.0;
    // Clear display window.
    glClear (GL_COLOR_BUFFER_BIT);
    // Set initial fill color to blue.
    glColor3f (0.0, 0.0, 1.0);
    // Display blue triangle.
    triangle (verts);
    /* Initialize composite matrix to identity. */
    matrix3x3SetIdentity (matComposite);
    /* Construct composite matrix for transformation sequence. */
    scale2D (sx, sy, fixedPt); // First transformation: Scale.
    rotate2D (pivPt, theta); // Second transformation: Rotate
    translate2D (tx, ty); // Final transformation: Translate.
    /* Apply composite matrix to triangle vertices. */
    transformVerts2D (nVerts, verts);
    glColor3f (1.0, 0.0, 0.0); // Set color for transformed triangle.
    triangle (verts);
    glFlush ( );
}

void winReshapeFcn (GLint newWidth, GLint 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_SINGLE | GLUT_RGB);
    glutInitWindowPosition (50, 50);
```

```
        glutInitWindowSize (winWidth, winHeight);
        glutCreateWindow ("Geometric Transformation Sequence");
        init ( );
        glutDisplayFunc (displayFcn);
        glutReshapeFunc (winReshapeFcn);
        glutMainLoop ( );
        return 0;
}
```

OUTPUT:



PROGRAM 3: Program to draw a color cube and spin it using OpenGL transformation matrices.

Program Code:

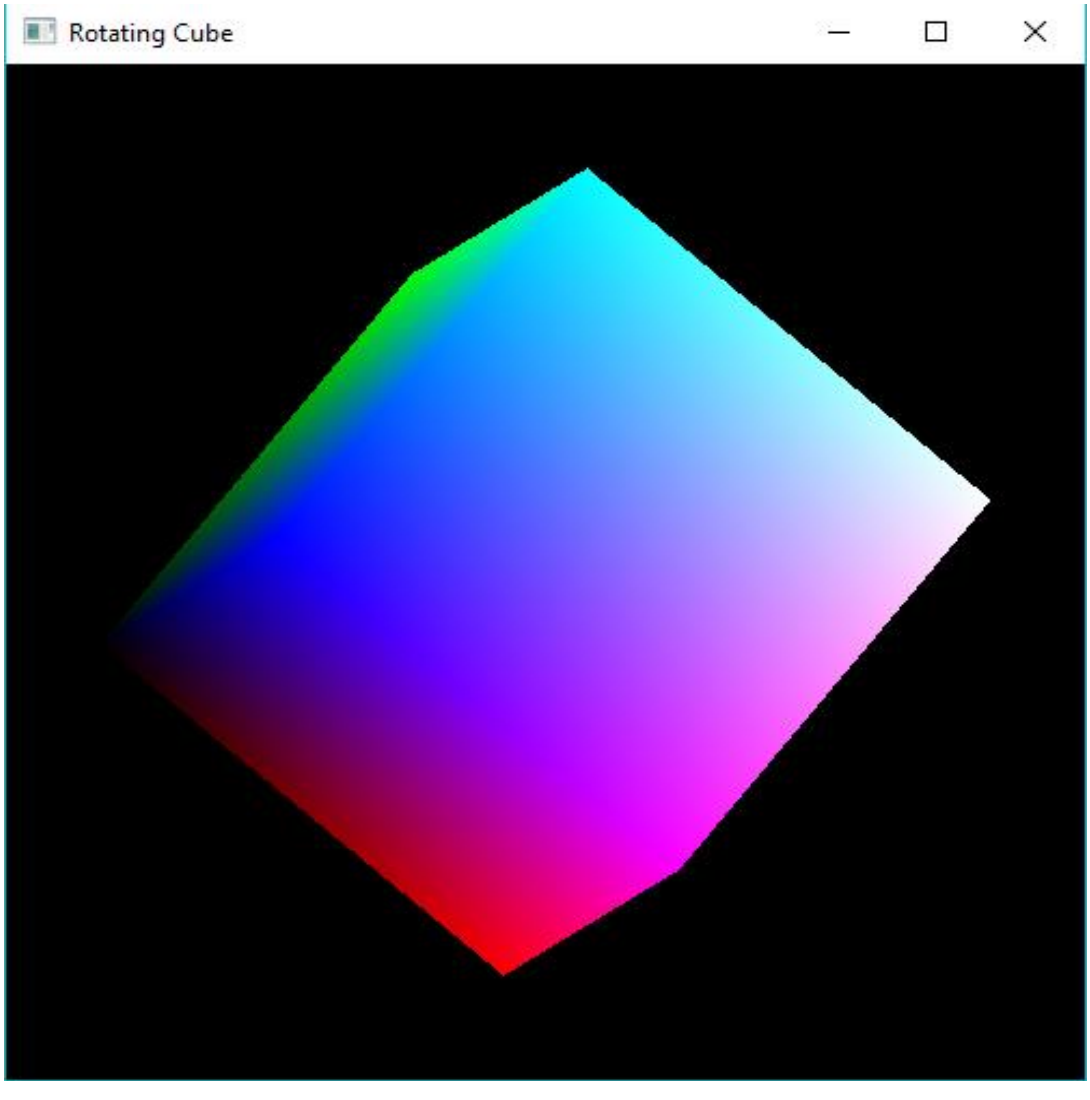
```
#include<stdlib.h>
#include<GL/glut.h>

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}};

static GLfloat theta[]={0.0,0.0,0.0};
static GLint axis=2;


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);
}

void display()
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        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()
{
        theta[axis]+=1.0;
        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 main(int argc, char **argv)
{
	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
	glutInitWindowSize(500,500);
	glutInitWindowPosition(0,0);
	glutCreateWindow("Rotating Cube");
	glutDisplayFunc(display);
	glutIdleFunc(spincube);
	glutMouseFunc(mouse);
	glutReshapeFunc(myreshape);
	glEnable(GL_DEPTH_TEST);
```

```
        glutMainLoop();
}
```

OUTPUT:



PROGRAM 4: Program to draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing. Use OpenGL functions.

Program Code:

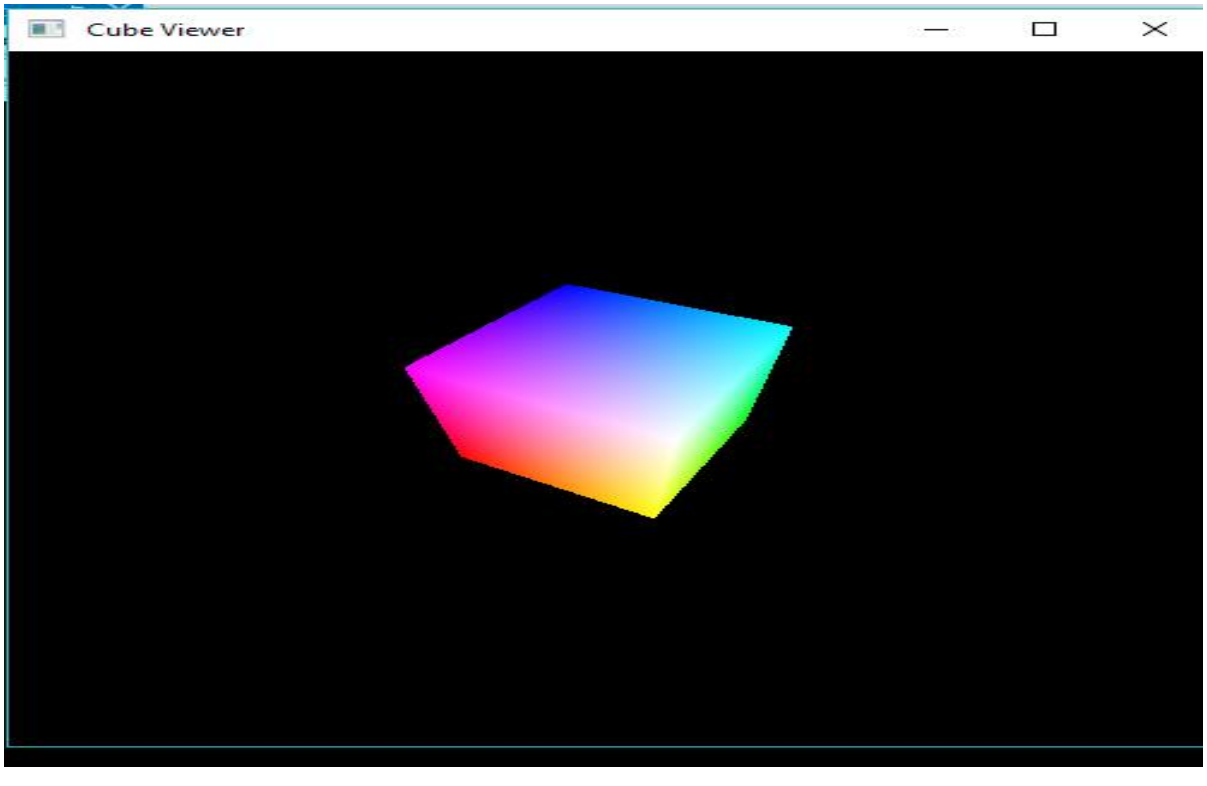
```
#include<stdlib.h>
#include<GL/glut.h>

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};

void display()
{
        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);
        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;
        glutPostRedisplay();
}

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*(GLfloat)w/(GLfloat)h,2.0*(GLfloat)w/(GLfloat)h,-
        2.0,2.0,2.0,20.0);

        glMatrixMode(GL_MODELVIEW);
}
```

```
void main(int argc, char **argv)
{
	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
	glutInitWindowSize(500,500);
	glutInitWindowPosition(0,0);
	glutCreateWindow(" Cube Viewer");
	glutReshapeFunc(myreshape);
	glutDisplayFunc(display);
	glutMouseFunc(mouse);
	glutKeyboardFunc(keys);
	glEnable(GL_DEPTH_TEST);
	glutMainLoop();
}
```

OUTPUT:



PROGRAM 5: Clip a lines using Cohen-Sutherland algorithm

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

double xmin=50,xmax=100,ymin=50,ymax=100;
double xvmin=200,xvmax=300,yvmin=200,yvmax=300;
float xc[10],yc[10];
int n;
typedef int outcode;
const int TOP=8;
const int BOTTOM=4;
const int RIGHT=2;
const int LEFT=1;

/*computing/ assigning region codes to end points of line*/
outcode ComputeCode(double x, double y)
{
        outcode code=0;
        if(y>ymax)
                code|=TOP;
        else if(y<ymin)
                code|=BOTTOM;
        if(x>xmax)
                code|=RIGHT;
        else if(x<xmin)
                code|=LEFT;
        return code;
}

void CohenSutherlandLineClipper(double x0,double y0,double x1,double
y1)
```

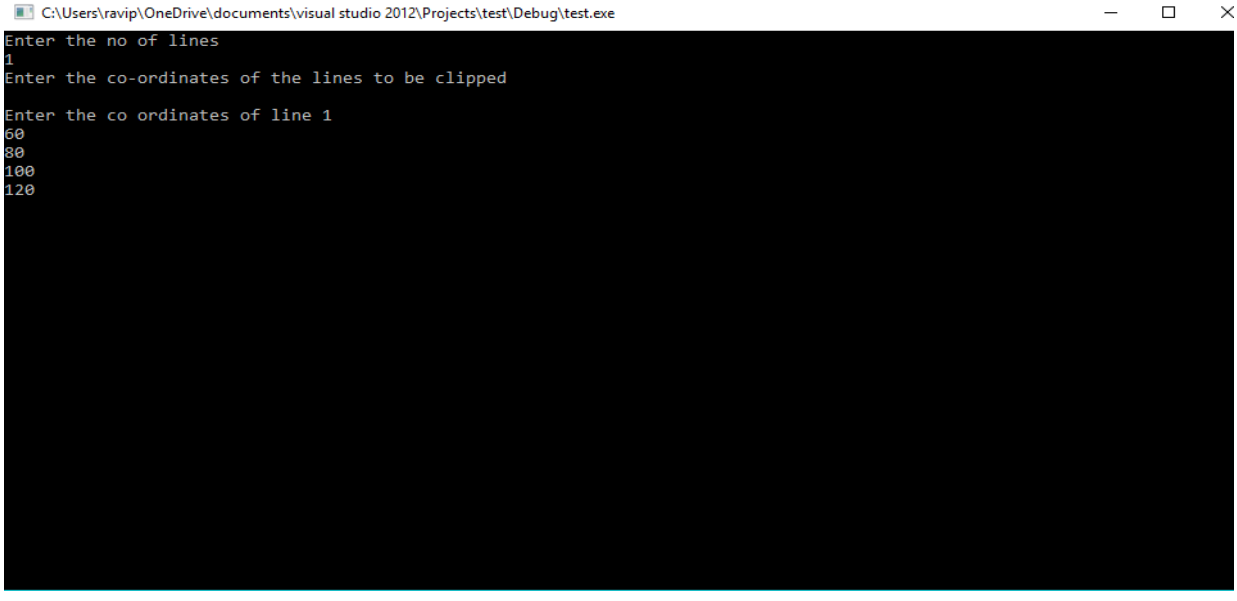
```
{
    outcode outcode0,outcode1,outcodeout;
    double x,y;
    bool accept=false,done=false;
    outcode0=ComputeCode(x0,y0);
    outcode1=ComputeCode(x1,y1);
    do
    {
        if(!(outcode0|outcode1))
        {
            accept=true;
            done=true;
        }
        else if(outcode0& outcode1)
            done=true;
        else
        {
            outcodeout=outcode0?outcode0:outcode1;
            if(outcodeout&TOP)
            {
                y=ymax;
                x=x0+(x1-x0)*(ymax-y0)/(y1-y0);
            }
            else if(outcodeout&BOTTOM)
            {
                y=ymin;
                x=x0+(x1-x0)*(ymin-y0)/(y1-y0);
            }
            else if(outcodeout & RIGHT)
            {
                x=xmax;
                y=y0+(y1-y0)*(xmax-x0)/(x1-x0);
            }
            else
```

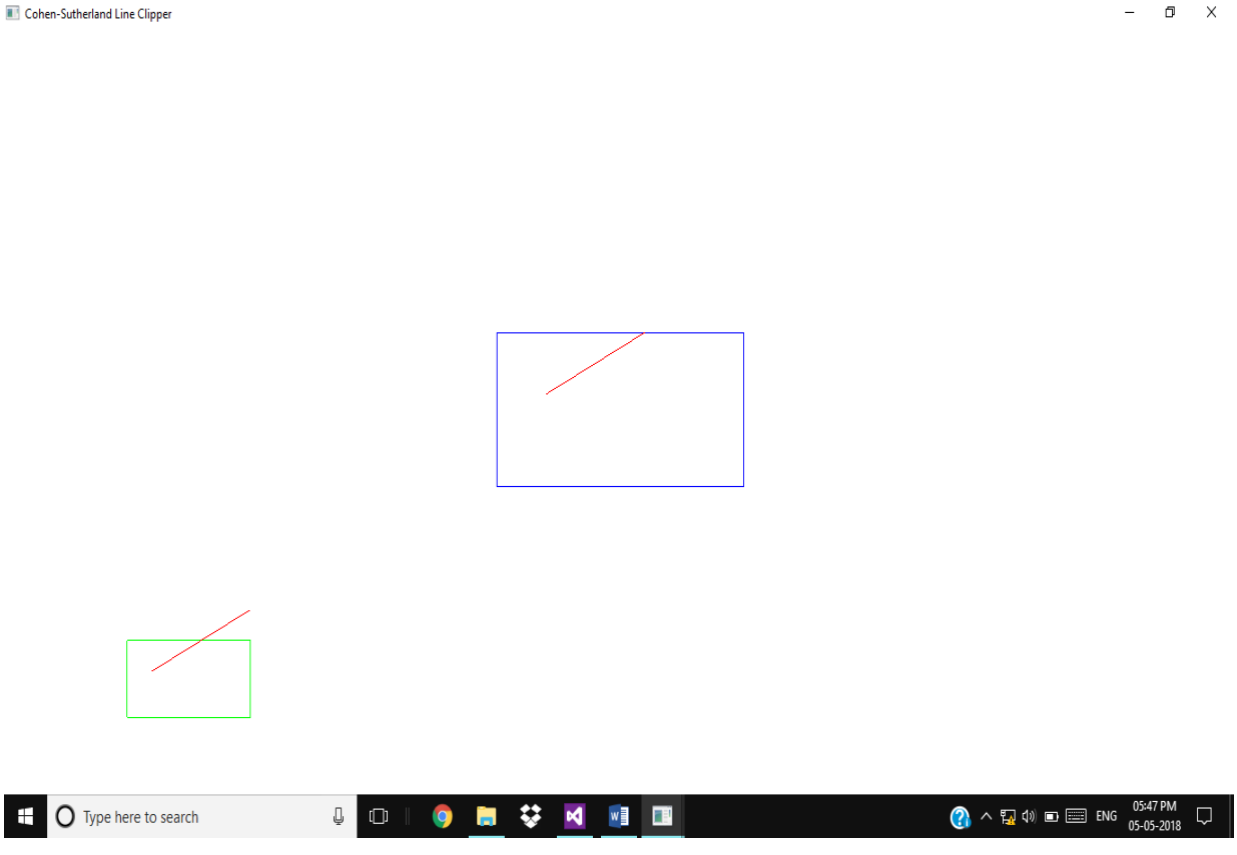
```
                {
                        x=xmin;
                        y=y0+(y1-y0)*(xmin-x0)/(x1-x0);
                }
                if(outcodeout==outcode0)
                {
                        x0=x;
                        y0=y;
                        outcode0=ComputeCode(x0,y0);
                }
                else
                {
                        x1=x;
                        y1=y;
                        outcode1=ComputeCode(x1,y1);
                }
        }
}while(done==false);
        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(0.0,0.0,1.0);
                glBegin(GL_LINE_LOOP);
                glVertex2f(xvmin,yvmin);
                glVertex2f(xvmax,yvmin);
                glVertex2f(xvmax,yvmax);
                glVertex2f(xvmin,yvmax);
                glEnd();
                glColor3f(1.0,0.0,0.0);
```

```
                glBegin(GL_LINES);
                glVertex2f(vx0,vy0);
                glVertex2f(vx1,vy1);
                glEnd();
            }
        }

void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0,1.0,0.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(xmin,ymin);
        glVertex2f(xmax,ymin);
        glVertex2f(xmax,ymax);
        glVertex2f(xmin,ymax);
        glEnd();
        glColor3f(1.0,0.0,0.0);
        glBegin(GL_LINES);
        for(int i=0;i<n*2;i=i+2)
        {
                glVertex2f(xc[i],yc[i]);
                glVertex2f(xc[i+1],yc[i+1]);
        }
        glEnd();
        for( i=0;i<n*2;i=i+2)
                CohenSutherlandLineClipper(xc[i],yc[i],xc[i+1],yc[i+1]);
        glFlush();
}

void myinit()
{
        glClearColor(1.0,1.0,1.0,1.0);
        gluOrtho2D(0.0,499.0,0.0,499.0);
```

```
}

void main(int argc, char **argv)
{
	int c=0;
	printf("Enter the no of lines\n");
	scanf("%d",&n);
	printf("Enter the co-ordinates of the lines to be clipped\n");
	for(int i=0;i<n;i++)
	{
		printf("\nEnter the co ordinates of line %d\n",i+1);
		for(int j=0;j<2;j++)
		{
			scanf("%f%f",&xc[c],&yc[c]);c++;
		}
	}

	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
	glutInitWindowPosition(0,0);
	glutInitWindowSize(500,500);
	glutCreateWindow("Cohen-Sutherland Line Clipper");
	myinit();
	glutDisplayFunc(display);
	glutMainLoop();
}
```

OUTPUT:


C:\Users\ravip\OneDrive\documents\visual studio 2012\Projects\test\Debug\test.exe
Enter the no of lines
1
Enter the co-ordinates of the lines to be clipped
Enter the co ordinates of line 1
60
80
100
120


Cohen-Sutherland Line Clipper
Type here to search
ENG
05:47 PM
05-05-2018

PROGRAM 6: Program, using OpenGL functions, 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 properties of the surfaces of the solid object used in the scene.

Program Code:

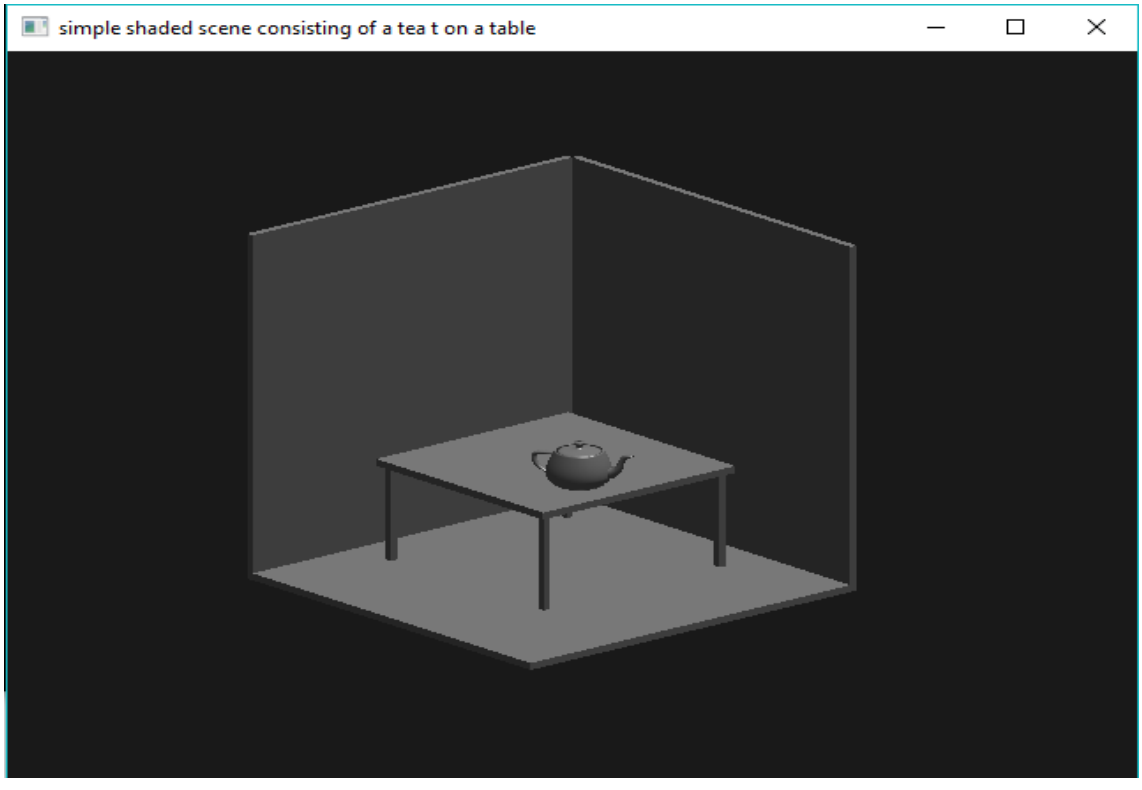
```
#include<GL/glut.h>
#include<stdio.h>
void wall(double thickness)
{
	glPushMatrix();
	glTranslated(0.5,0.5*thickness,0.5);
	glScaled(1.0,thickness, 1.0);
	glutSolidCube(1.0);
	glPopMatrix();
}
void tableLeg(double thick, double len)
{
	glPushMatrix();
	glTranslated(0,len/2,0);
	glScaled(thick,len,thick);
	glutSolidCube(1.0);
	glPopMatrix();
}

void table(double topWid, double topThick,double legThick, double
legLen)
{
	glPushMatrix();
	glTranslated(0,legLen,0);
	glScaled(topWid,topThick,topWid);
```

```
        glutSolidCube(1.0);
        glPopMatrix();
        double dist=0.95*topWid/2.0-legThick/2.0;
        glPushMatrix();
        glTranslated(dist,0,dist);
        tableLeg(legThick,legLen);
        glTranslated(0.0,0.0,-2*dist);
        tableLeg(legThick,legLen);
        glTranslated(-2*dist,0,2*dist);
        tableLeg(legThick,legLen);
        glTranslated(0,0,-2*dist);
        tableLeg(legThick,legLen);
        glPopMatrix();
}
void displaySolid(void)
{
        glLoadIdentity();
        //set properties of the surface material
        GLfloat mat_ambient[] = {0.7f, 0.7f, 0.7f, 1.0f}; // gray
        GLfloat mat_diffuse[] = {.5f, .5f, .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);
        //set the light source properties
        GLfloat lightIntensity[] = {0.9f, 0.9f, 0.9f, 1.0f};
        GLfloat light_position[] = {2.0f, 6.0f, 3.0f, 0.0f};
        glLightfv (GL_LIGHT0, GL_POSITION, light_position);
        glLightfv (GL_LIGHT0, GL_DIFFUSE, lightIntensity);
        //set the camera
        glMatrixMode (GL_PROJECTION);
        glLoadIdentity();
```

```
    double winHt = 1.0; //half-height of window
    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);
    //start drawing
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glTranslated (0.6, 0.38, 0.5);
    glRotated (30, 0, 1, 0);
    glutSolidTeapot (0.08);
    glPopMatrix ();
    glPushMatrix();
    glTranslated (0.25, 0.42, 0.35);
    glPopMatrix ();
    glPushMatrix();
    glTranslated (0.4, 0, 0.4);
    table (0.6, 0.02, 0.02, 0.3);
    glPopMatrix();
    wall (0.02);
    glPushMatrix();
    glRotated (90.0, 0.0, 0.0, 1.0);
    wall (0.02);
    glPopMatrix();
    glPushMatrix();
    glRotated (-90.0, 1.0, 0.0, 0.0);
    wall (0.02);
    glPopMatrix();
    glFlush();
}
void main(int argc, char ** argv)
```

```
{
        glutInit (&argc, argv);
        glutInitDisplayMode
        (GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
        glutInitWindowSize (640, 480);
        glutInitWindowPosition (100, 100);
        glutCreateWindow ("simple shaded scene consisting
        of a tea pot on a table");
        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:



PROGRAM 7: Program to recursively subdivide a tetrahedron to from 3D Sierpinski gasket. The number of recursive steps is to be specified by the user.

Program Code:

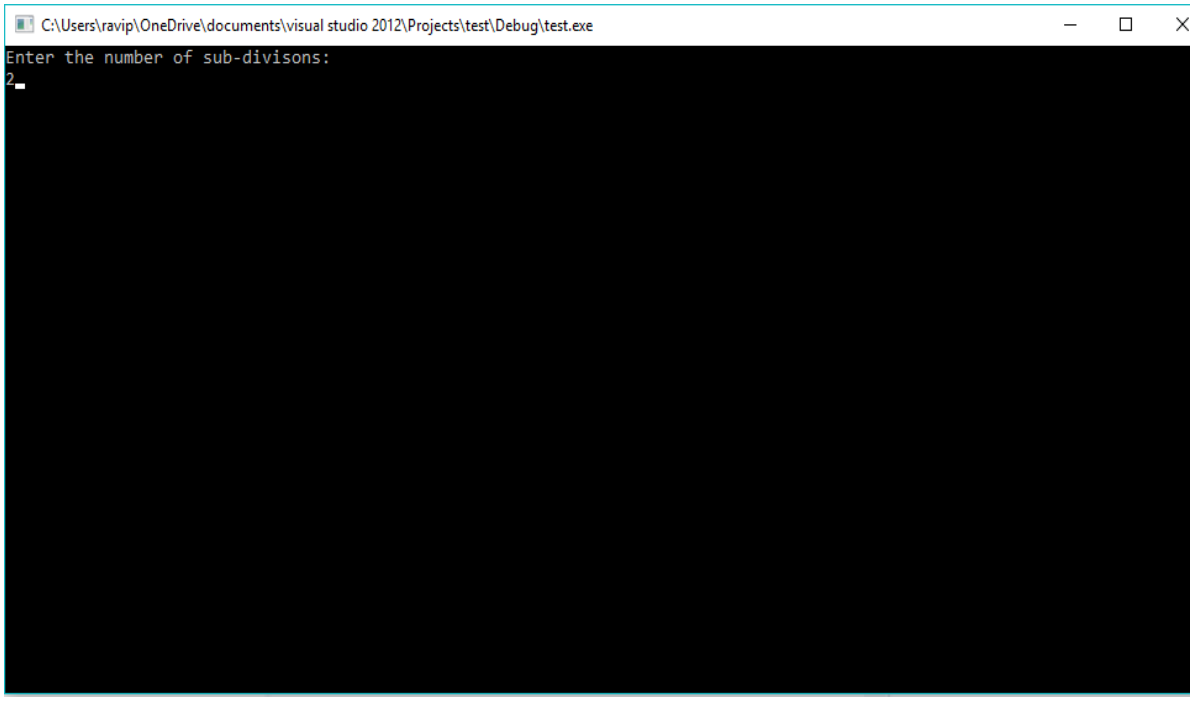
```
#include<GL/glut.h>
#include<stdio.h>
typedef float point[3];

point v[4]={{0.0,0.0,1.0},{0.0,0.942809,-0.33333},{-
0.816497,-0.471405,-0.333333},{0.816497,-0.471405,-
0.333333}};

int n;          /*recursive steps*/
void triangle(point a,point b,point c)
{
```

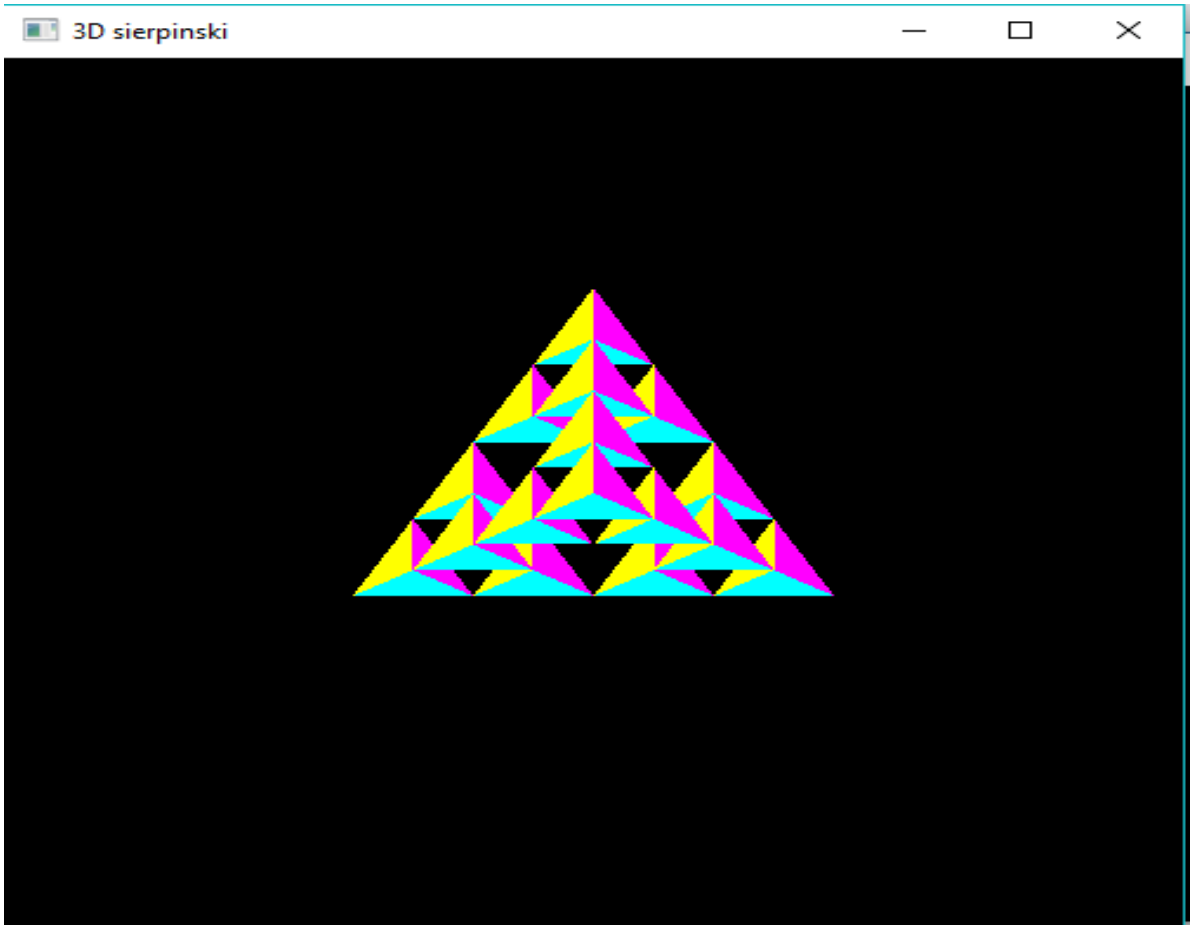
```
        glBegin(GL_TRIANGLES);
        glNormal3fv(a);
        glVertex3fv(a);
        glVertex3fv(b);
        glVertex3fv(c);
        glEnd();
}
void tetrahedron(point a, point b, point c, point d)
{
        glColor3f(1.0,1.0,0.0);
        triangle(a,b,c);
        glColor3f(0.0,1.0,1.0);
        triangle(a,c,d);
        glColor3f(1.0,0.0,1.0);
        triangle(a,d,b);
        glColor3f(0.0,0.0,0.0);
        triangle(b,d,c);
}
void divide_tetrahedron(point a, point b, point c, point d,
int n)
{
        int j;
        point v1,v2,v3,v4,v5,v6;/*variables to store six mid points*/
        if(n>0)
        {
                /*the six mid-points of the six edges of a tetrahedron*/
                for(j=0;j<3;j++)v1[j]=(a[j]+b[j])/2;        /*mid point
                of edge ab*/
                for(j=0;j<3;j++)v2[j]=(a[j]+c[j])/2;        /*mid point
                of edge ac*/
                for(j=0;j<3;j++)v3[j]=(a[j]+d[j])/2;        /*mid point
                of edge ad*/
                for(j=0;j<3;j++)v4[j]=(b[j]+c[j])/2;        /*mid point
                of edge bc*/
```

```
            for(j=0;j<3;j++)v5[j]=(c[j]+d[j])/2;        /*mid point
            of edge cd*/
            for(j=0;j<3;j++)v6[j]=(b[j]+d[j])/2;        /*mid point
            of edge bd*/
            /*a tetrahedron formed from vertices a,mid point of ab,ac,ad
        edge*/
            divide_tetrahedron(a, v1,v2,v3,n-1);
            /*a tetrahedron formed from vertices b,mid point of ab,bc,bd
        edge*/
            divide_tetrahedron( v1,b,v4,v6,n-1);
            /*a tetrahedron formed from vertices c,mid point of ac,bc,cd
            edge*/
            divide_tetrahedron( v2,v4,c,v5,n-1);
            /*a tetrahedron formed from vertices d,mid point of ad,cd,bd
            edge*/
            divide_tetrahedron( v3,v6,v5,d,n-1);
        }
        else
            tetrahedron(a,b,c,d);/*drawing the
        tetrahedrons*/
}
void display()
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_
        BIT);
        glEnable(GL_DEPTH_TEST);
        divide_tetrahedron(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
                (-2.0,2.0,-
                2.0*(GLfloat)h/(GLfloat)w,2.0*(GLfloat)h/(GLfloa
                t)w,-10.0,10.0);}
        else
                glOrtho(-
                2.0*(GLfloat)w/(GLfloat)h,2.0*(GLfloat)w/(GLflo
                at)h,-2.0,2.0,-10.0,10.0);
        glMatrixMode(GL_MODELVIEW);
        glutPostRedisplay();
        }
void main(int argc,char **argv)
{
        printf("Enter the number of sub-divisons:\n");
        scanf("%d",&n);
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE|GLUT_
        DEPTH);
        glutInitWindowPosition(0,0);
        glutInitWindowSize(500,500);
        glutCreateWindow("3D sierpinski");
        glutReshapeFunc(myreshape);
        glutDisplayFunc(display);
        glClearColor(0.0,0.0,0.0,1.0);
        glutMainLoop();
}
```

OUTPUT:


C:\Users\ravip\OneDrive\documents\visual studio 2012\Projects\test\Debug\test.exe
Enter the number of sub-divisons:
2


3D sierpinski

PROGRAM 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
GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = 0.0, xwcMax = 130.0;
GLfloat ywcMin = 0.0, ywcMax = 130.0;
typedef struct wcPt3D
{
        GLfloat x, y, z;
};
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, wcPt3D *bezPt, GLint
nCtrlPts, wcPt3D *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(wcPt3D *ctrlPts, GLint nCtrlPts, GLint
nBezCurvePts)
{
        wcPt3D 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()
{
        GLint nCtrlPts = 4, nBezCurvePts =20;
        static float theta = 0;
```

```
wcPt3D ctrlPts[4] = { {20, 100, 0}, {30, 110, 0}, {50, 90,
0},
                                                    {60,
                                                100, 0}};
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;
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 1.0, 1.0);
glPointSize(5);
glPushMatrix();
glLineWidth(5);
//Indian flag: Orange color code
glColor3f(255/255, 153/255.0, 51/255.0);
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 winReshapeFun(GLint newWidth, GLint newHeight)
{
        glViewport(0, 0, newWidth, newHeight);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
        glClear(GL_COLOR_BUFFER_BIT);
}
void main(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowPosition(50, 50);
        glutInitWindowSize(winWidth, winHeight);
        glutCreateWindow("Bezier Curve");
        glutDisplayFunc(displayFcn);
        glutReshapeFunc(winReshapeFun);
        glutMainLoop();
```

}

OUTPUT:

Program 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 r=0,g=0,b=0;//for colors menu (flag)
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)
{
//For red color menu
if (r==1){
glColor3f(1.0,0.0,0.0);
```

```
}//For green color menu
else if(g==1)
{
glColor3f(0.0,1.0,0.0);
}//For blue color menu
else if(b==1)
{
glColor3f(0.0,0.0,1.0);
}
else
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);
}
}
void display()
{
x1=200.0;y1=200.0;x2=100.0;y2=300.0;x3=200.0;y3=400.0;x4=30
0.0;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();
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);
}
// Menu exit
void handlemenu(int value)
{
switch (value) {
```

```
case 4:
exit(0);
break;
}
}
//Colors menu
void cmenu(int value){
switch(value){
case 1:
r=1;
```

```
g=0,b=0;
glutPostRedisplay();
break;


case 2:
g=1;
b=0;r=0;
glutPostRedisplay();
break;
case 3:
b=1;
g=0;r=0;
glutPostRedisplay();
break;
}
}
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");
int colors_menu=glutCreateMenu(cmenu);
glutAddMenuEntry("red", 1);
glutAddMenuEntry("green", 2);
glutAddMenuEntry("blue", 3);
glutCreateMenu(handlemenu);
glutAddSubMenu("color", colors_menu);
glutAddMenuEntry("Quit",4);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutDisplayFunc(display);
myinit();
glutMainLoop();
}
```

Output: