RAJEEV INSTITUTE OF TECHNOLOGY

HASSAN-573201

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Laboratory Manual

SEMESTER - VI

Computer Graphics & Visualization Laboratory (15CSL68)

[As per Choice Based Credit System (CBCS) scheme]

(Effective from the academic year 2015 -2016)

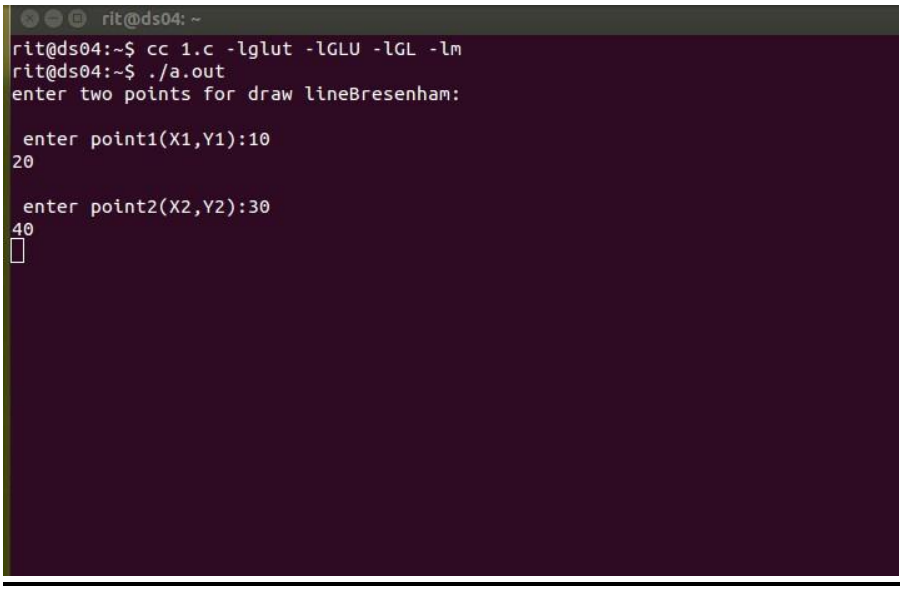
Program 1-

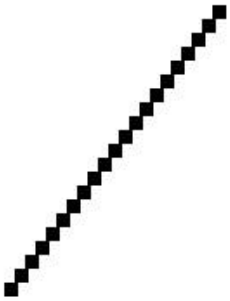
Implement Bresenham's line drawing algorithm for all types slope.

```
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>
GLint X1,Y1,X2,Y2;
void LineBres(void)
{
      glClear(GL_COLOR_BUFFER_BIT);
      int dx=abs(X2-X1),dy=abs(Y2-Y1);
      int p=2*dy-dx;
      int twoDy=2*dy,twoDyDx=2*(dy-dx);
      int x,y;
      if(X1>X2)
      {
            x=X2;
            y=Y2;
            X2=X1;
      }
      else
      {
            x=X1;
            y=Y1;
            X2=X2;
      }
      glBegin(GL_POINTS);
      glVertex2i(x,y);
      while(x<X2)
      {
            x++;
            if(p<0)
                  p+=twoDy;
            else
            {
                  y++;
                  p+=twoDyDx;
            }
            glVertex2i(x,y);
      }
      glEnd();
      glFlush();
}
void Init()
{
      glClearColor(1.0,1.0,1.0,0);
      glColor3f(0.0,0.0,0.0);
      glPointSize(8.0);
      glViewport(0,0,50,50);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(0,50,0,50);
}
int main(int argc,char **argv)
{
      printf("enter two points for draw lineBresenham:\n");
      printf("\n enter point1(X1,Y1):");
```

```
scanf("%d%d",&X1,&Y1);
printf("\n enter point2(X2,Y2):");
scanf("%d%d",&X2,&Y2);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(300,400);
glutInitWindowPosition(0,0);
glutCreateWindow("LineBresenham");
Init();
glutDisplayFunc(LineBres);
glutMainLoop();
}
```

Output:





Program 2-

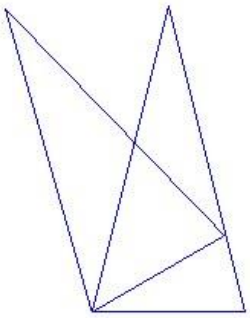
Create and rotate a triangle about the origin and a fixed point.

```
#define BLACK 0
#include <stdio.h>
#include <math.h>
#include <GL/glut.h>
GLfloat theta;
GLfloat
triangle[3][3]={{100.0,150.0,200.0},{100.0,300.0,100.0},{1.0,1.0,1.0}}
;
GLfloat rotatemat[3][3]={{0},{0},{0}};
GLfloat result[3][3]={{0}, {0}, {0} };
GLfloat arbitrary_x=100.0;
GLfloat arbitrary_y=100.0;
GLfloat rotation_angle;
void multiply()
{
int i,j,k;
for(i=0;i<3;i++)
for(j=0;j<3;j++)
{
result[i][j]=0;
for(k=0;k<3;k++)
result[i][j]=result[i][j]+rotatemat[i][k]* triangle[k][j];
}
}
void rotate()
{
GLfloat m,n;
rotation_angle=theta*3.1415/180.0;
m=-
arbitrary_x*(cos(rotation_angle)-1)+arbitrary_y*(sin(rotation_angle));
n=-arbitrary_y*(cos(rotation_angle)-1)-
arbitrary_x*(sin(rotation_angle));
rotatemat[0][0]=cos(rotation_angle);
rotatemat[0][1]=-sin(rotation_angle);
rotatemat[0][2]=m;
rotatemat[1][0]=sin(rotation_angle);
rotatemat[1][1]=cos(rotation_angle);
rotatemat[1][2]=n;
rotatemat[2][0]=0;
rotatemat[2][1]=0;
rotatemat[2][2]=1;
//multiply the two matrices
multiply();
}
void drawtriangle()
{
glColor3f(0.0, 0.0, 1.0);

glBegin(GL_LINE_LOOP);
glVertex2f(triangle[0][0], triangle[1][0]);
glVertex2f(triangle[0][1],triangle[1][1]);
glVertex2f(triangle[0][2], triangle[1][2]);
glEnd();
}
```

```
void drawrotatedtriangle()
{
glColor3f(0.0, 0.0, 1.0);
glBegin(GL_LINE_LOOP);
glVertex2f(result[0][0],result[1][0]);
glVertex2f(result[0][1],result[1][1]);
glVertex2f(result[0][2],result[1][2]);
glEnd();
}
void display()
{
 glClearColor(1.0,1.0,1.0,1.0);
glClear(GL_COLOR_BUFFER_BIT);

drawtriangle();
rotate();
drawrotatedtriangle();
glFlush();
}
int main(int argc, char** argv)
{
printf("Enter the rotation angle\n");
scanf("%f", &theta);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("triangle rotation");
glutDisplayFunc(display);
/*glMatrixMode(GL_PROJECTION);
glLoadIdentity();*/
gluOrtho2D(0.0,499.0,0.0,499.0);
glutMainLoop();
}
```

Output:

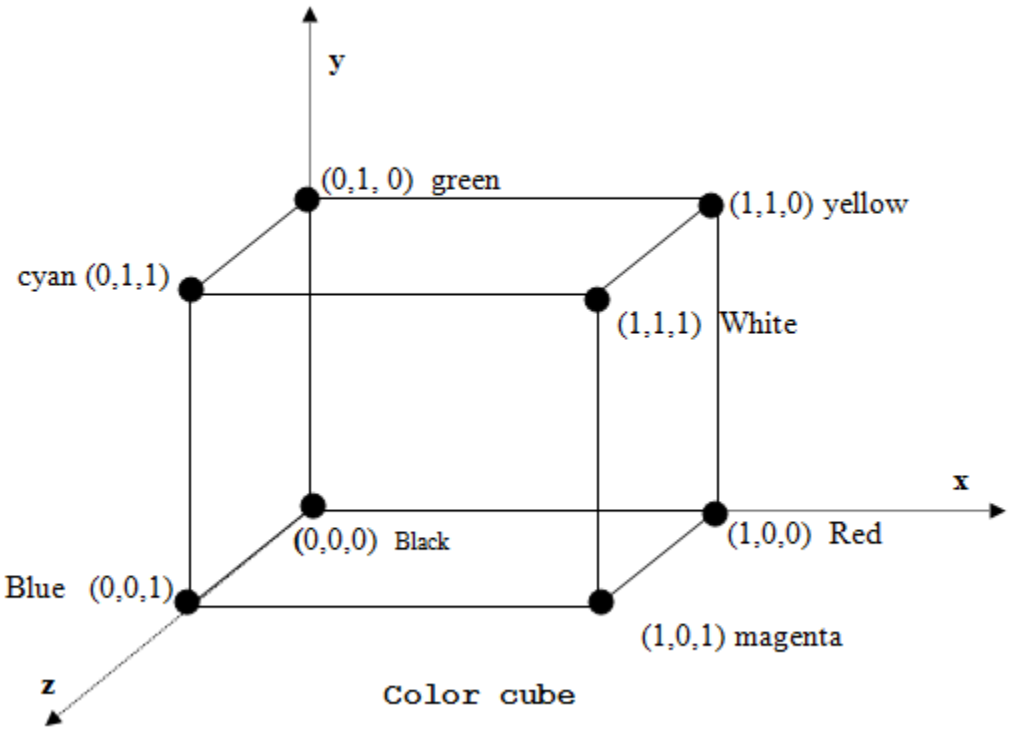
```
rit@ds04: ~
rit@ds04:~$ cc 2.c -lglut -lGLU -lGL -lm
rit@ds04:~$ ./a.out
Enter the rotation angle
30
```



Program 3-

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

Theory-



Color cube

The color cube can be rotated by using glRotatef() API of opengl.

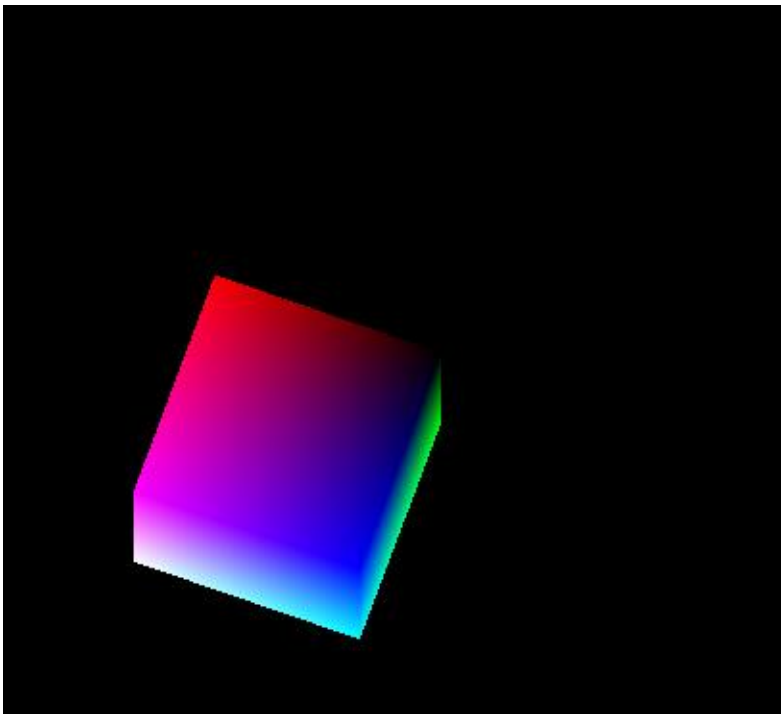
Program-

```
#include<GL/glut.h>
float
ver[]
[3]={{0,0,0},{1,0,0},{1,1,0},{0,1,0},{0,0,1},{1,0,1},{1,1,1},{0,
1,1}};
float theta[]={0,0,0};
int axis=0;
void polygon(int a,int b,int c,int d)
{
     glBegin(GL_POLYGON);
     glColor3fv(ver[a]);
     glVertex3fv(ver[a]);
     glColor3fv(ver[b]);
     glVertex3fv(ver[b]);
     glColor3fv(ver[c]);
     glVertex3fv(ver[c]);
     glColor3fv(ver[d]);
     glVertex3fv(ver[d]);
     glEnd();
}
```

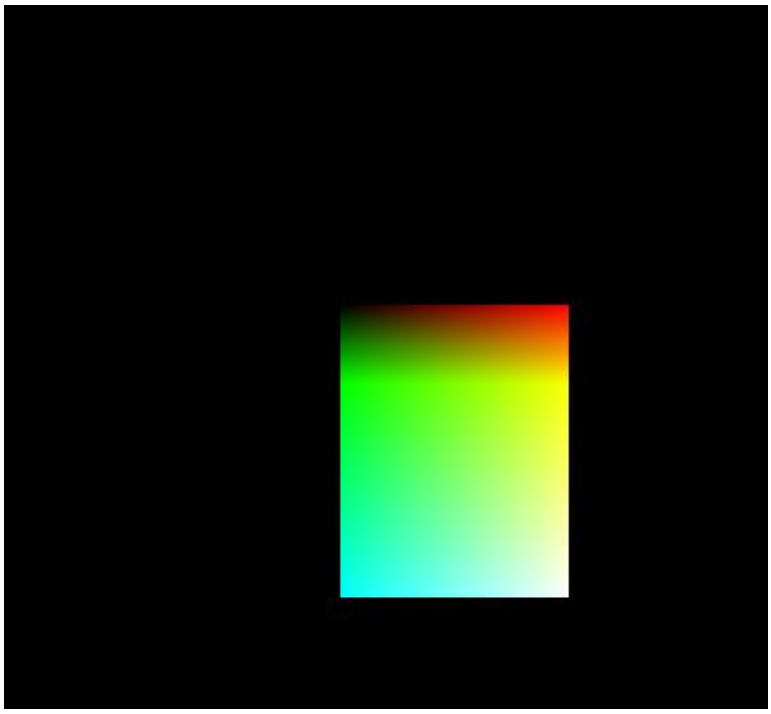
```
void Colorcube(void)
{
     polygon(0,3,2,1);
     polygon(2,3,7,6);
     polygon(0,4,7,3);
     polygon(1,2,6,5);
     polygon(4,5,6,7);
     polygon(0,1,5,4);
}
void display(void)
{
     glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
     glLoadIdentity();
     glRotatef(theta[0],1,0,0);
     glRotatef(theta[1],0,1,0);
     glRotatef(theta[2],0,0,1);
     Colorcube();
     glFlush();
     glutSwapBuffers();
}
void spincube()
{
     theta[axis]+=0.5;
     if(theta[axis]>360)
          theta[axis]=0;
     display();
}
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(-2,2,-2*h/w,2*h/w,-10,10);
     else
          glOrtho(-2*w/h,2*w/h,-2,2,-10,10);
     glMatrixMode(GL_MODELVIEW);
}
void main(int argc,char**argv)
{
     glutInit(&argc,argv);
     glutInitDisplayMode(GLUT_DOUBLE|GLUT_DEPTH);
     glutInitWindowSize(500,500);
     glutCreateWindow("SPININNG COLOR CUBE-1");
     glutReshapeFunc(myReshape);
```

```
        glutDisplayFunc(display);
        glutIdleFunc(spincube);
        glutMouseFunc(mouse);
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
}
```

Output:



Cube rotation along z-axis



Cube rotation along x-axis

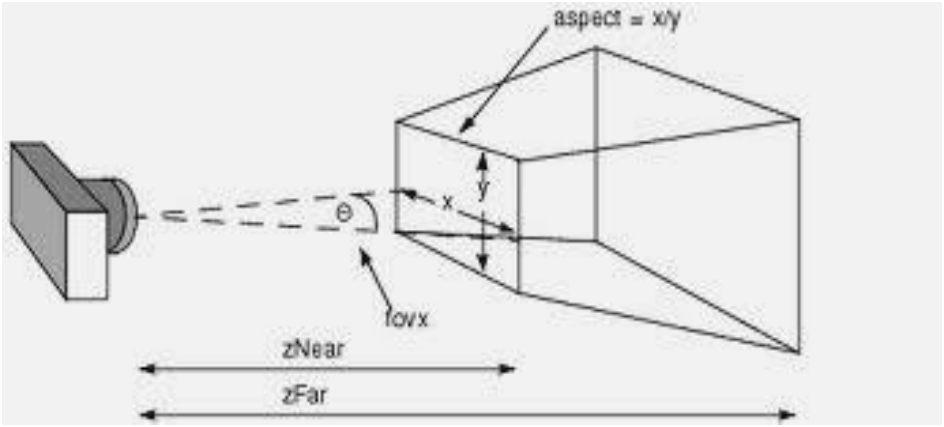
Program 4-

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

Theory-

Perspective viewing:

The opengl API for the perspective viewing is glFrustum(left,right,top,bottom,near,far) as shown below.



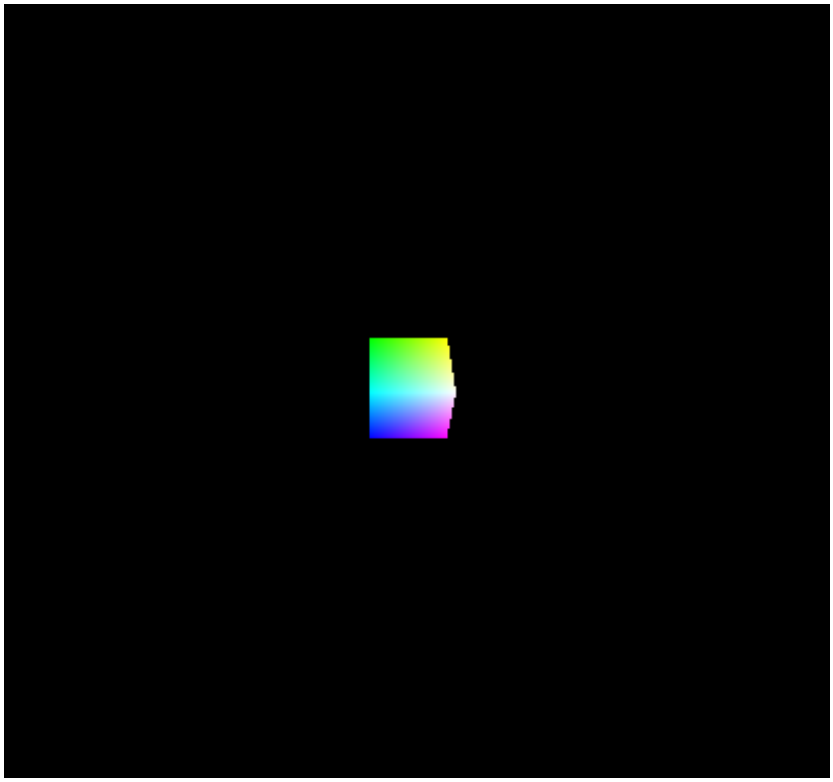
The above camera (vx,vy,vz)can be controlled by using the keyboard functions by changing the values of vx,vy and vz.

Program-

```
#include<GL/glut.h>
#include<stdio.h>
#include<stdlib.h>
float
ver[8][3]={{0,0,0},{1,0,0},{1,1,0},{0,1,0},{0,0,1},{1,0,1},{1,1,
1},{0,1,1}};
float v1[3]={0,0,5};
void polygon(int a,int b,int c,int d);
void polygon1();
void display()
{
     glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
     glLoadIdentity();
     gluLookAt(v1[0],v1[1],v1[2],0,0,0,0,1,0);
     polygon1();
     glFlush();
}
void polygon1()
{
     polygon(0,1,2,3);
     polygon(4,5,6,7);
     polygon(5,1,2,6);
     polygon(4,0,3,7);
     polygon(4,5,1,0);
     polygon(7,6,2,3);
}
```

```
void polygon(int a,int b,int c,int d)
{
     glBegin(GL_POLYGON);
     glColor3fv(ver[a]);
     glVertex3fv(ver[a]);
     glColor3fv(ver[b]);
     glVertex3fv(ver[b]);
     glColor3fv(ver[c]);
     glVertex3fv(ver[c]);
     glColor3fv(ver[d]);
     glVertex3fv(ver[d]);
     glEnd();
}
void key(char f,int x,int y)
{
     if(f=='x') v1[0]-=.10;
     if(f=='X') v1[0]+=.10;
     if(f=='y') v1[1]-=.10;
     if(f=='Y') v1[1]+=.10;
     if(f=='z') v1[2]-=.10;
     if(f=='Z') v1[2]+=.10;
     display();
}
void Reshape(int w,int h)
{
     glViewport(0,0,w,h);
     glMatrixMode(GL_PROJECTION);
     glLoadIdentity();
     if(w<=h)
          glFrustum(-2.0,2.0,-2.0*h/w,2.0*h/w,2.0,20);
     else
          glFrustum(-2.0*w/h,2.0*w/h,-2.0,2.0,2.0,20);
          glMatrixMode(GL_MODELVIEW);
}
void main(int argc,char**argv)
{
     glutInit(&argc,argv);
     glutInitDisplayMode(GLUT_SINGLE|GLUT_DEPTH);
     glutInitWindowSize(500,500);
     glutInitWindowPosition(10,10);
     glutCreateWindow("SPINNING COLOR CUBE-2");
     glutDisplayFunc(display);
     glutKeyboardFunc(key);
     glEnable(GL_DEPTH_TEST);
     glutReshapeFunc(Reshape);
     glutMainLoop();
}
```

Output:



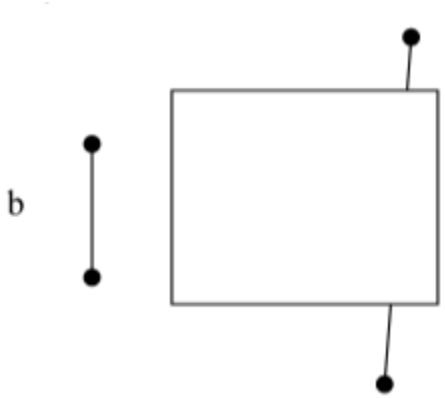
Program 5-

Clip a line using Cohen-Sutherland line clipping algorithm.

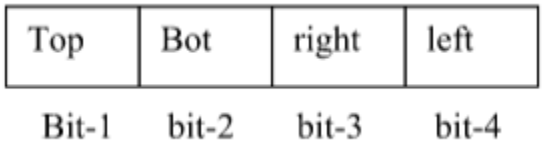
Theory-

We divide the line clipping process into 2 parts

1. Identify those lines which intersect the clipping window
2. Perform the clipping operation if all the lines fall into one of the following category
   - Visible: Both end points of the line lies within the windows
   - Not Visible: Both end points of the line completely outside the windows
   - Partly Visible: If the lines lies neither belong to category 1 or category 2.

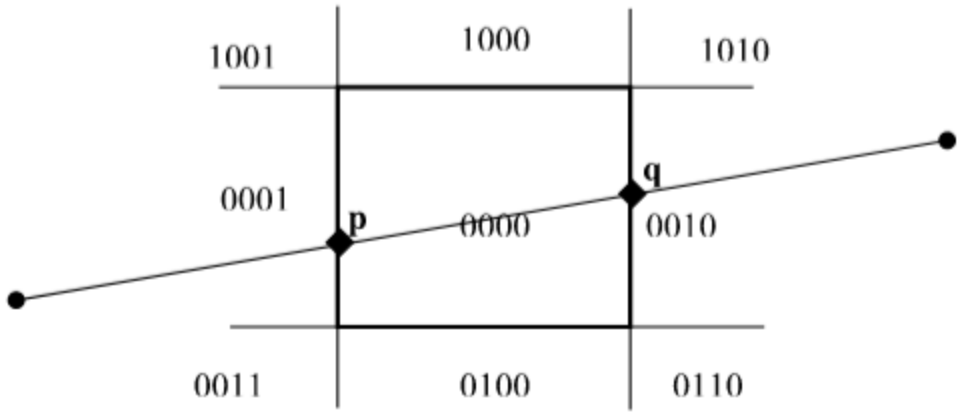


3. Assign a 4 bit region code to each point of a line. The code is determined as



Each bit is enabled as 1 based on the following conditions otherwise it is enabled as 0.

Bit1: The end points is above the clip window
Bit2: The end points is below the clip window
Bit3: The end points is right of the clip window
Bit4: The end points is left of the clip window



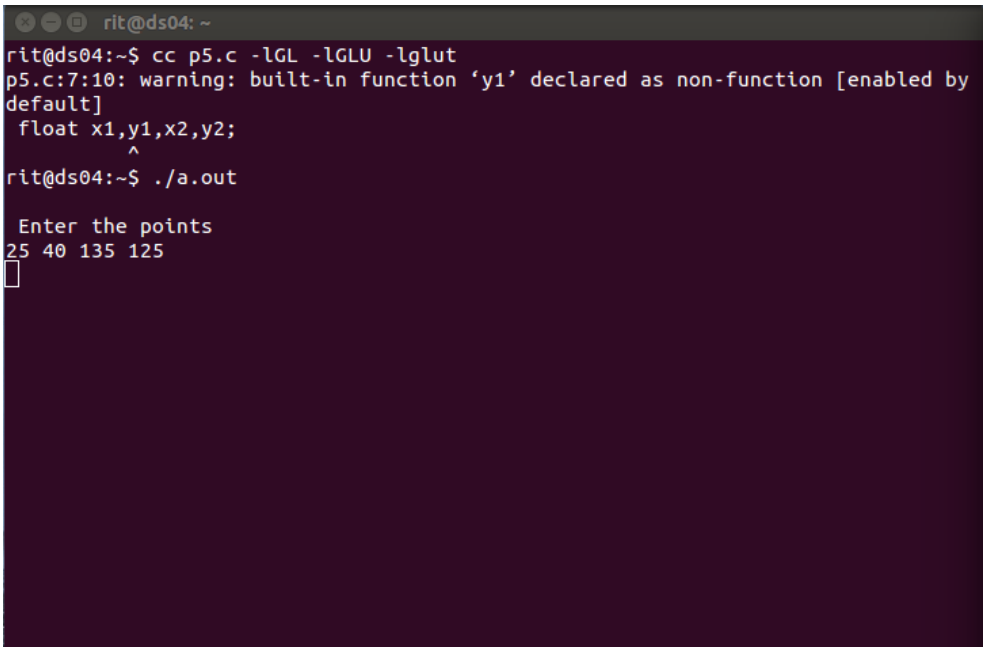
Program-

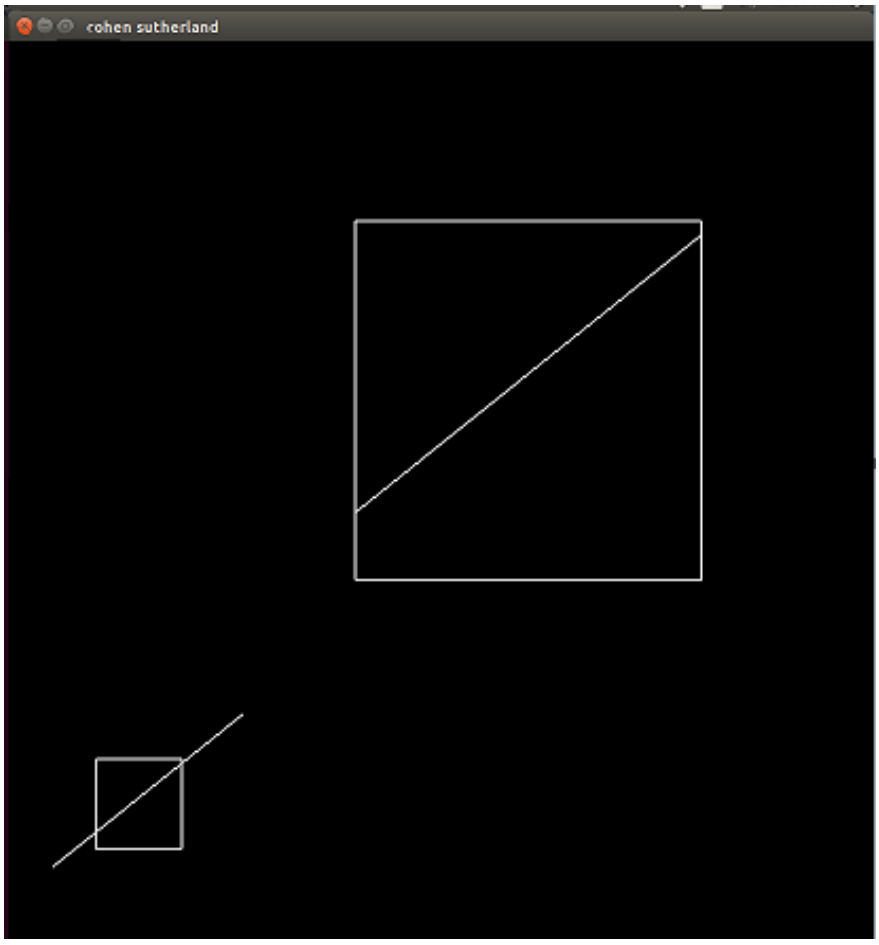
```
#include<stdio.h>
#include<GL/glut.h>
float xmin=50,ymin=50,xmax=100,ymax=100;
float xvmin=200,yvmin=200,xvmax=400,yvmax=400;
int RIGHT=8,LEFT=2,TOP=4,BOTTOM=1;
float sx,sy,vx1,vy1,vx2,vy2;
float x1,y1,x2,y2;
int compute(float x,float y)
{
      int code=0;
      if(y>ymax)
            code=TOP;
      else if(y<ymin)
            code=BOTTOM;
      if(x>xmax)
            code=RIGHT;
      if(x<xmin)
            code=LEFT;
            return code;
}
void cohen(float x1,float y1,float x2,float y2)
{
      float x,y;
      int accept=0,done=0,code_p,code_q,code;
      code_p=compute(x1,y1);
      code_q=compute(x2,y2);
      do
      {
            if(!(code_p|code_q))
            {
                  accept=1;
                  done=1;
            }
            else if(code_p & code_q)
                  done=1;
            else
            {
                  code=code_p?code_p:code_q;
                  if(code & TOP)
                  {
                        x=x1+(x2-x1)*(ymax-y1)/(y2-y1);
```

```
                        y=ymax;
                  }
                  else if(code & BOTTOM)
                  {
                        x=x1+(x2-x1)*(ymin-y1)/(y2-y1);
                        y=ymin;
                  }
                  else if(code & RIGHT)
                  {
                        y=y1+(y2-y1)*(xmax-x1)/(x2-x1);
                        x=xmax;
                  }
                  else
                  {
                        y=y1+(y2-y1)*(xmin-x1)/(x2-x1);
                        x=xmin;
                  }
                  if(code==code_p)
                  {
                        x1=x;
                        y1=y;
                        code_p=compute(x1,y1);
                  }
                  else
                  {
                        x2=x;
                        y2=y;
                        code_q=compute(x2,y2);
                  }
            }
      }while(!done);
      if(accept)
      {
            sx=(xvmax-xvmin)/(xmax-xmin);
            sy=(yvmax-yvmin)/(ymax-ymin);
            vx1=xvmin+(x1-xmin)*sx;
            vy1=xvmin+(y1-ymin)*sy;
            vx2=xvmin+(x2-xmin)*sx;
            vy2=xvmin+(y2-ymin)*sy;
      }
}
void display()
{
      glClear(GL_COLOR_BUFFER_BIT);
      glColor3f(1,1,1);
      glLineWidth(2);
      glBegin(GL_LINES);
      glVertex2d(x1,y1);
      glVertex2d(x2,y2);
      glEnd();
      glColor3f(1,1,1);
      glBegin(GL_LINE_LOOP);
      glVertex2f(xmin,ymin);
      glVertex2f(xmax,ymin);
      glVertex2f(xmax,ymax);
      glVertex2f(xmin,ymax);
      glEnd();
      cohen(x1,y1,x2,y2);
      glColor3f(1,1,1);
```

```
      glBegin(GL_LINE_LOOP);
      glVertex2f(xvmin,yvmin);
      glVertex2f(xvmax,yvmin);
      glVertex2f(xvmax,yvmax);
      glVertex2f(xvmin,yvmax);
      glEnd();
      glColor3f(1,1,1);
      glBegin(GL_LINES);
      glVertex2d(vx1,vy1);
      glVertex2d(vx2,vy2);
      glEnd();
      glFlush();
}
void myinit()
{
      glClearColor(0,0,0,1);
      gluOrtho2D(0,500,0,500);
}
void main(int argc,char**argv)
{
      printf("\n Enter the points \n");
      scanf("%f %f %f %f",&x1,&y1,&x2,&y2);
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
      glutInitWindowSize(500,500);
      glutCreateWindow("Cohen Sutherland");
      glutDisplayFunc(display);
      myinit();
      glutMainLoop();
}
```

Output:


rit@ds04: ~
rit@ds04:~$ cc p5.c -lGL -lGLU -lglut
p5.c:7:10: warning: built-in function ‘y1’ declared as non-function [enabled by
default]
float x1,y1,x2,y2;
^
rit@ds04:~$ ./a.out
Enter the points
25 40 135 125


cohen sutherland

Program 6-

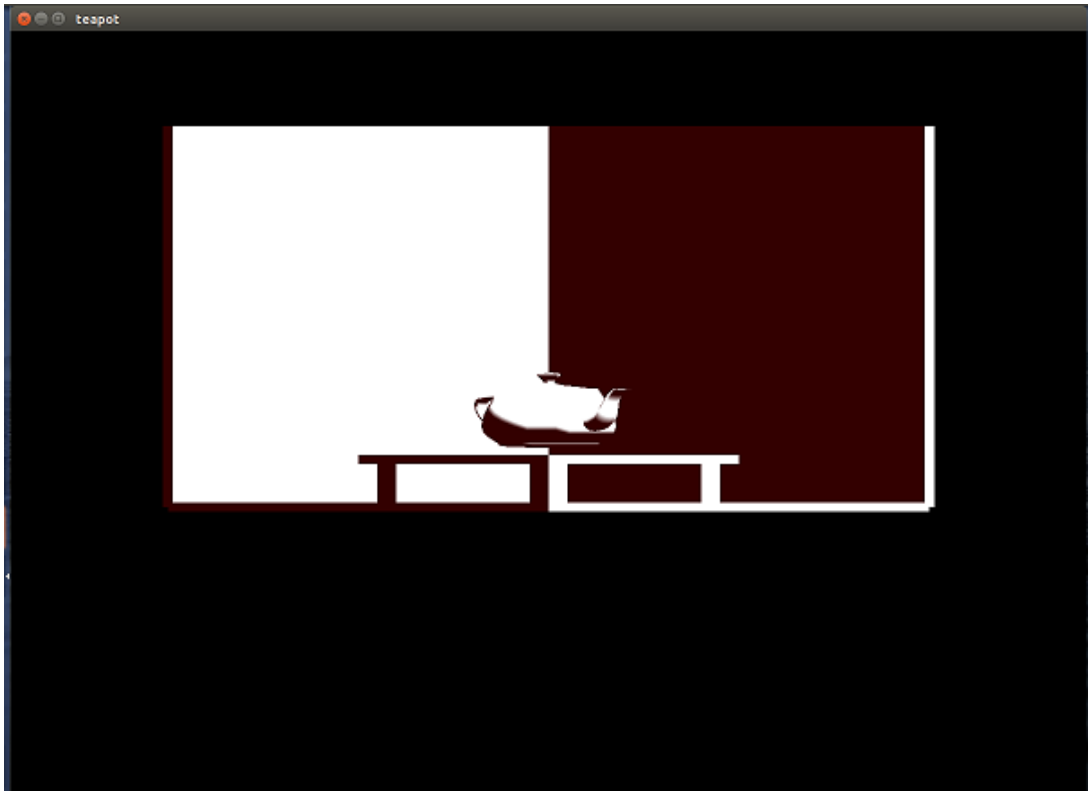
To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.

Program-

```
#include<stdio.h>
#include<GL/glut.h>
void init()
{
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-4,4,-4,4,-10,10);
      glMatrixMode(GL_MODELVIEW);
}
void wall()
{
      glPushMatrix();
      glScalef(2,0.05,2);
      glutSolidCube(2);
      glPopMatrix();
      glPushMatrix();
      glTranslatef(-2,2,0);
      glScalef(0.05,2,2);
      glutSolidCube(2);
      glPopMatrix();
      glPushMatrix();
      glTranslatef(0,2,-2);
      glScalef(2,2,0.05);
      glutSolidCube(2);
      glPopMatrix();
}
void table()
{
      glPushMatrix();
      glTranslatef(0,0.5,0);
      glScalef(1,0.05,1);
      glutSolidCube(2);
      glPopMatrix();
      glPushMatrix();
      glTranslatef(-0.8,0.2,0.8);
      glScalef(0.1,0.25,0.1);
      glutSolidCube(2);
      glPopMatrix();
      glPushMatrix();
      glTranslatef(0.8,0.2,0.8);
      glScalef(0.1,0.25,0.1);
      glutSolidCube(2);
      glPopMatrix();
      glPushMatrix();
      glTranslatef(0.8,0.2,-0.8);
      glScalef(0.1,0.25,0.1);
      glutSolidCube(2);
      glPopMatrix();
      glPushMatrix();
      glTranslatef(-0.8,0.2,-0.8);
```

```
      glScalef(0.1,0.25,0.1);
      glutSolidCube(2);
      glPopMatrix();
}
void tea()
{
      glPushMatrix();
      glTranslatef(0,1,0);
      glutSolidTeapot(0.5);
      glPopMatrix();
}
void display(void)
{
      float amb[]={1,0,0};
      float pos[]={2,4,1};
      glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
      glMaterialfv(GL_FRONT,GL_AMBIENT,amb);
      glLightfv(GL_LIGHT0,GL_POSITION,pos);
      gluLookAt(2,1,2,0,1,0,0,1,0);
      wall();
      table();
      tea();
      glFlush();
}
void main(int argc,char**argv)
{
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE|GLUT_DEPTH);
      glutInitWindowSize(1200,1200);
      glutCreateWindow("Teapot");
      init();
      glutDisplayFunc(display);
      glEnable(GL_DEPTH_TEST);
      glShadeModel(GL_SMOOTH);
      glEnable(GL_LIGHTING);
      glEnable(GL_LIGHT0);
      glEnable(GL_NORMALIZE);
      glutMainLoop();
}
```

Output:

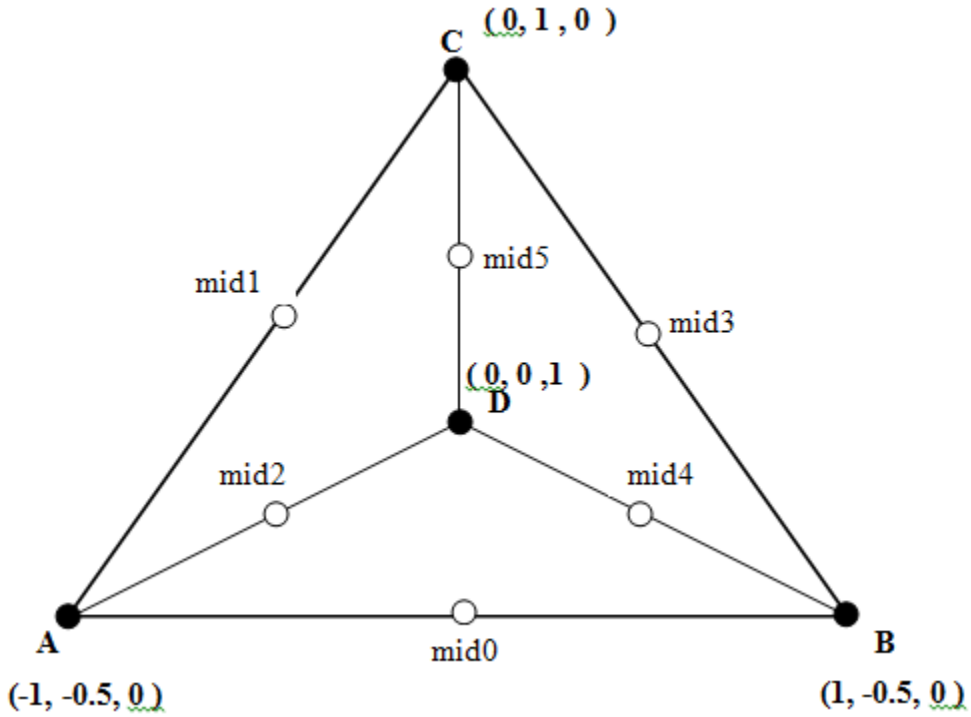


Program 7-

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.

Theory-

The coordinate system of the tetrahedron is shown below.



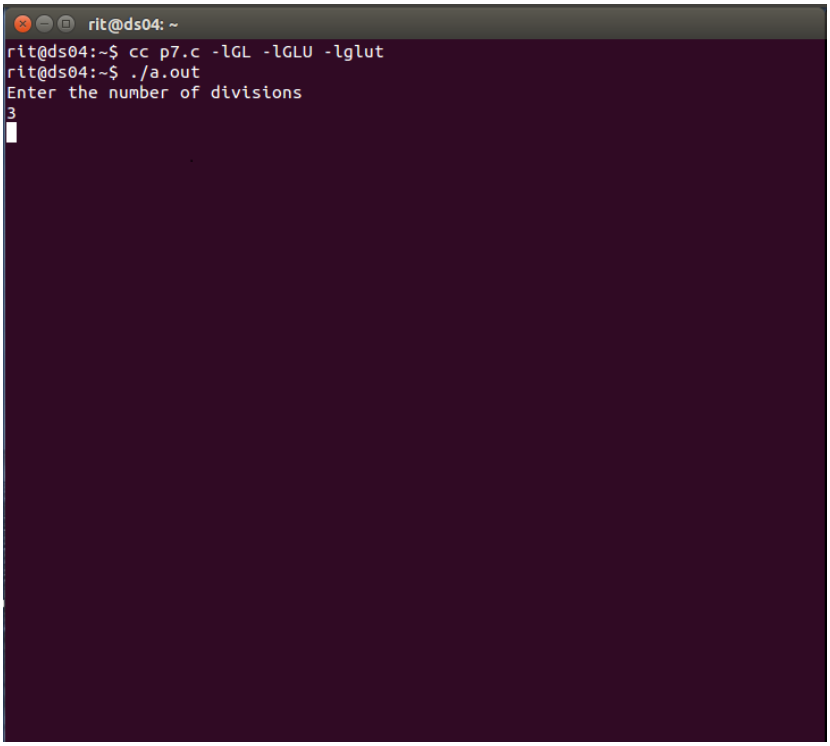
The mid-point of each face is caluculated as mid=(a+b)/2.

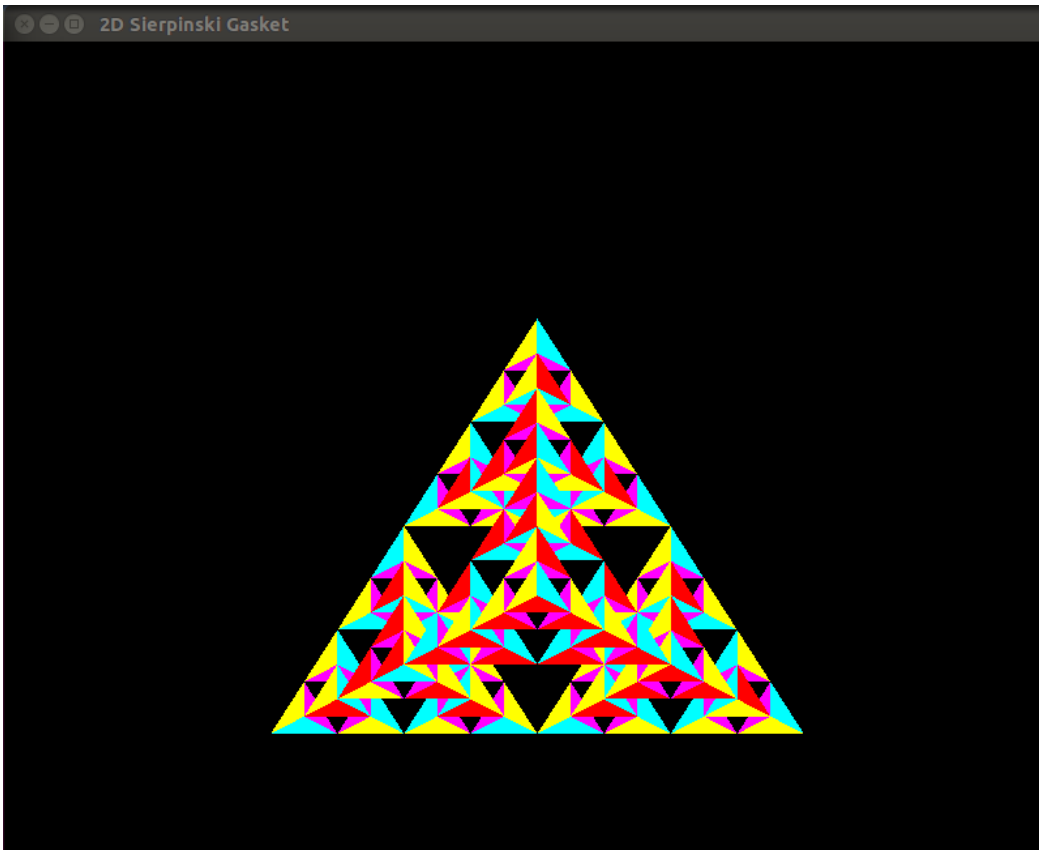
Program-

```
#include<stdio.h>
#include<GL/glut.h>
float v[][3]={{-1,-0.5,0},{1,-0.5,0},{0,1,0},{0,0,1}};
int n;
void triangle(float *p,float *q,float *r)
{
     glVertex3fv(p);
     glVertex3fv(q);
     glVertex3fv(r);
}
void tetra(float *a,float *b,float *c,float *d)
{
     glColor3f(1,0,0);
     triangle(a,b,c);
     glColor3f(1,1,0);
     triangle(a,b,d);
     glColor3f(1,0,1);
     triangle(b,c,d);
     glColor3f(0,1,1);
     triangle(a,d,c);
```

```
}
void divide_tetra(float *a,float *b,float *c,float *d,int m)
{
     float mid[6][3];
     int j;
     if(m>0)
     {
          for(j=0;j<3;j++)
          {
               mid[0][j]=(a[j]+b[j])/2;
               mid[1][j]=(a[j]+c[j])/2;
               mid[2][j]=(a[j]+d[j])/2;
               mid[3][j]=(b[j]+c[j])/2;
               mid[4][j]=(b[j]+d[j])/2;
               mid[5][j]=(c[j]+d[j])/2;
          }
          divide_tetra(a,mid[1],mid[0],mid[2],m-1);
          divide_tetra(b,mid[0],mid[3],mid[4],m-1);
          divide_tetra(c,mid[1],mid[3],mid[5],m-1);
          divide_tetra(d,mid[2],mid[4],mid[5],m-1);
     }
     else
     tetra(a,b,c,d);
}
void display()
{
     glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
     glBegin(GL_TRIANGLES);
     divide_tetra(v[0],v[1],v[2],v[3],n);
     glEnd();
     glFlush();
}
void init(void)
{
     glClearColor(0,0,0,1);
     glOrtho(-2,2,-2,2,-10,10);
}
int main(int argc,char**argv)
{
     glutInit(&argc,argv);
     printf("Enter the number of divisions");
     scanf("%d",&n);
     glutInitDisplayMode(GLUT_SINGLE|GLUT_DEPTH);
     glutInitWindowSize(700,700);
     glutCreateWindow("Sierpinski Gasket");
     init();
     glutDisplayFunc(display);
     glEnable(GL_DEPTH_TEST);
     glutMainLoop();
}
```

Output:


rit@ds04: ~
rit@ds04:~$ cc p7.c -lGL -lGLU -lglut
rit@ds04:~$ ./a.out
Enter the number of divisions
3


2D Sierpinski Gasket

Program 8-

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

```
#include<GL/glut.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define PI 3.1416

static int win,val=0,CMenu;

void CreateMenu(void);
void Menu(int value);

struct wcPt3D
{
GLfloat x, y, z;
};

GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = 0.0, xwcMax = 130.0;
GLfloat ywcMin = 0.0, ywcMax = 130.0;
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,struct wcPt3D *bezPt, GLint
nCtrlPts,struct 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(struct wcPt3D *ctrlPts, GLint nCtrlPts, GLint
nBezCurvePts)
{
struct 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;
struct 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);
//Indian Flag
if(val==1){
glPushMatrix();
glLineWidth(5);
glColor3f(1.0,0.5,0); //Indian flag: Orange color code
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glColor3f(1,1,1); //Indian flag: white color code
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glColor3f(0,1.0,0); //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();
}
//Karnataka Flag
if(val==2){
glPushMatrix();
glLineWidth(5);
glColor3f(1.0, 1.0, 0.0); //Karnataka flag: Yellow color code
for(int i=0;i<12;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glColor3f(1, 0.0, 0.0); //Karnataka flag: Red color code
for(int i=0;i<12;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 CreateMenu(void)
{
        CMenu= glutCreateMenu(Menu);//Creaate Menu Option

        glutAddMenuEntry("Indian Flag",1);
        glutAddMenuEntry("Karnataka Flag",2);
        glutAddMenuEntry("Exit",0);
```

```
        glutAttachMenu(GLUT_RIGHT_BUTTON);

}

void Menu(int value)
{
        if(value==0)
        {
        glutDestroyWindow(win);
        exit(0);
        }
        else {
        val=value;
        }
}

int main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowPosition(50, 50);
glutInitWindowSize(winWidth, winHeight);
glutCreateWindow("Prg. 8 Bezier Curve");
CreateMenu();
glutDisplayFunc(displayFcn);
glutReshapeFunc(winReshapeFun);
glutMainLoop();
}
```

NOTE:

This is a c++ program. We nedd to save this program file name with .cpp extension and for compilation we need to use following commands.

```
 g++ filename.cpp –lglut lGLU –lGL -lm
```

Output:


Prg. 8 Bezier Curve
Indian Flag
Karnataka Flag
Exit


Prg. 8 Bezier Curve
Indian Flag
Karnataka Flag
Exit

Program 8-

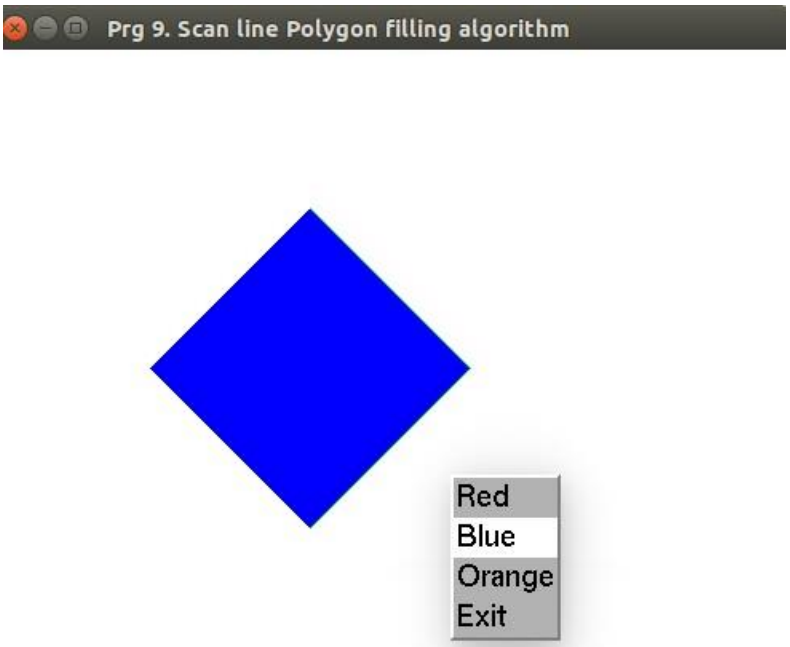
Develop a menu driven program to fill the polygon using scan line algorithm.

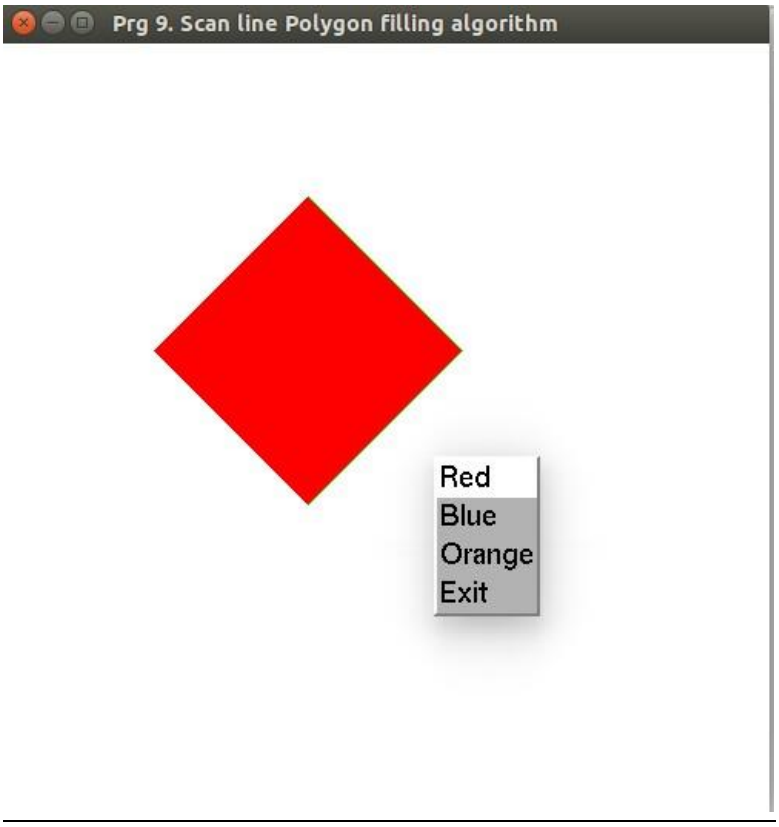
```
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
float x1,x2,x3,x4,y1,y2,y3,y4;
static int win,val=0,CMenu;

void CreateMenu(void);
void Menu(int value);

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)
{
     if(val==1)
     {
          glColor3f(1.0,0.0,0.0);
     }
     else if(val==2)
     {
          glColor3f(0.0,0.0,1.0);
     }
     else if(val==3)
     {
          glColor3f(1.0,0.5,0.0);
     }
```

```
     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=300.0;y4=300.0;
     glClear(GL_COLOR_BUFFER_BIT);
     glColor3f(0.0,1.0,0.0);
     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);
}
```

```
void CreateMenu(void)
{
     CMenu= glutCreateMenu(Menu);//Creaate Menu Option

     glutAddMenuEntry("Red",1);
     glutAddMenuEntry("Blue",2);
     glutAddMenuEntry("Orange",3);
     glutAddMenuEntry("Exit",0);

     glutAttachMenu(GLUT_RIGHT_BUTTON);

}

void Menu(int value)
{
     if(value==0)
     {
     glutDestroyWindow(win);
     exit(0);
     }
     else {
     val=value;
     }
}

int main(int argc,char** argv)
{
     glutInit(&argc,argv);
     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
     glutInitWindowSize(500,500);
     glutInitWindowPosition(0,0);

     win=glutCreateWindow("Prg 9. Scan line Polygon filling
algorithm");
     CreateMenu();

     glutDisplayFunc(display);
     myinit();
     glutMainLoop();
}
```

Output:


Prg 9. Scan line Polygon filling algorithm
Red
Blue
Orange
Exit


Prg 9. Scan line Polygon filling algorithm
Red
Blue
Orange
Exit