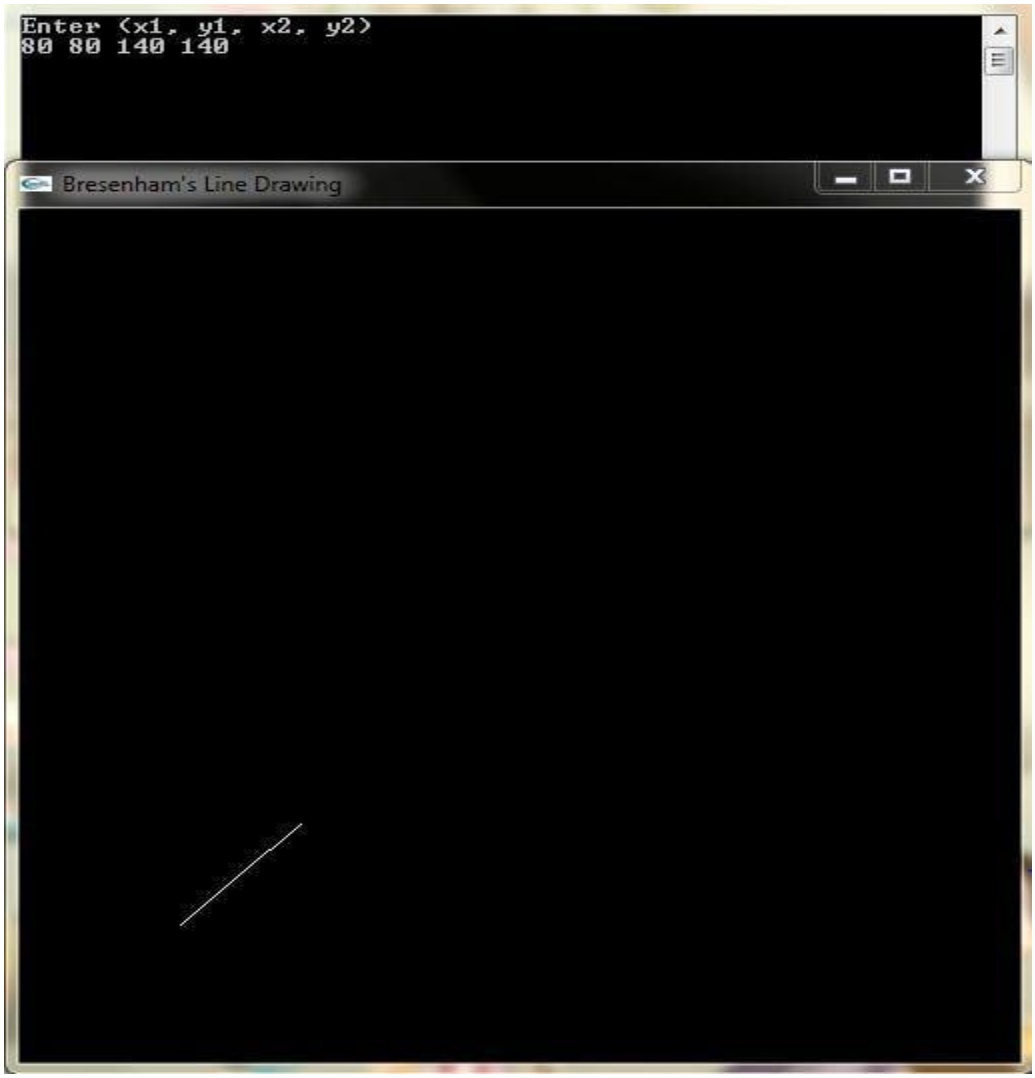
Program 1:-Implement Bresenham's line drawing algorithm for all types of slope

```
#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, x, y, incx, incy, inc1, inc2;
   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;
}
```

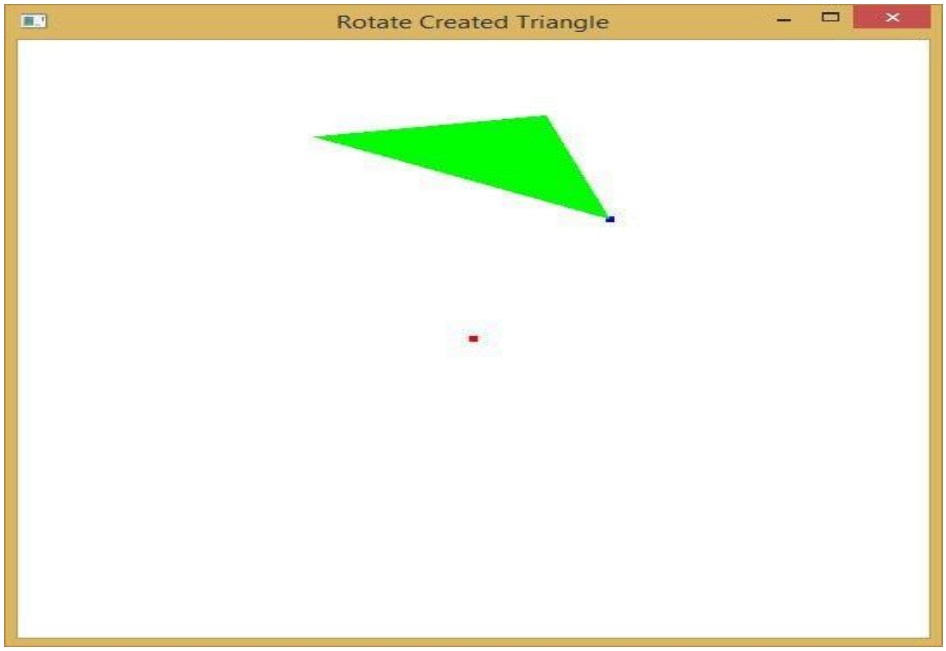
Program 2:- Create and rotate a triangle about the origin and a fixed point.




```
#include<stdio.h>
#include<GL/glut.h>
int x,y;
int where_to_rotate=0;
float translate_x=0.0,translate_y=0.0,rotate_angle=0.0;
void draw_pixel(float x1,float y1)
{
   glPointSize(5.0);
   glBegin(GL_POINTS);
      glVertex2f(x1,y1);
   glEnd();
}
void triangle(int x,int y)
{
   glColor3f(0.0,1.0,0.0); // set interior color of triangle to green
   glBegin(GL_POLYGON);
      glVertex2f(x,y);
      glVertex2f(x+400,y+400);
      glVertex2f(x+300,y+0);
   glEnd();
   glFlush();
}
```

```
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glLoadIdentity();
   glColor3f(1.0,0.0,0.0); //color of point
   draw_pixel(0.0,0.0);
   if(where_to_rotate==1)
   {
      translate_x=0.0;
      translate_y=0.0;
      rotate_angle+=0.9;
   }
   if(where_to_rotate==2)
   {
      translate_x=x;
      translate_y=y;
      rotate_angle+=0.9;
      glColor3f(0.0,0.0,1.0);
      draw_pixel(x,y);
   }
   glTranslatef(translate_x,translate_y,0.0);
   glRotatef(rotate_angle,0.0,0.0,1.0);
   glTranslatef(-translate_x,-translate_y,0.0);
   triangle(translate_x,translate_y);
   glutPostRedisplay();
   glutSwapBuffers();
}
void myInit()
{
   glClearColor(1.0,1.0,1.0,1.0); //background color to white
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-800.0,800.0,-800.0,800.0);
   glMatrixMode(GL_MODELVIEW);
}
void rotate_menu(int option)
{
   if(option==1)
      where_to_rotate=1;
   if(option==2)
      where_to_rotate=2;
   if(option==3)
      where_to_rotate=3;
   display();
}
int main(int argc,char **argv)
{
   printf("\nEnter fixed points for rotation (x,y) : ");
   scanf("%d%d",&x,&y);
   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
   glutInitWindowSize(800,800);
   glutInitWindowPosition(0,0);
   glutCreateWindow("Rotate Created Triangle");
      myInit();
```

```
   glutDisplayFunc(display);
   glutCreateMenu(rotate_menu);
      glutAddMenuEntry("Rotate Around Origin",1);
      glutAddMenuEntry("Rotate Around Fixed Points",2);
      glutAddMenuEntry("Stop Rotation",3);
   glutAttachMenu(GLUT_RIGHT_BUTTON);
   glutMainLoop();
}
```

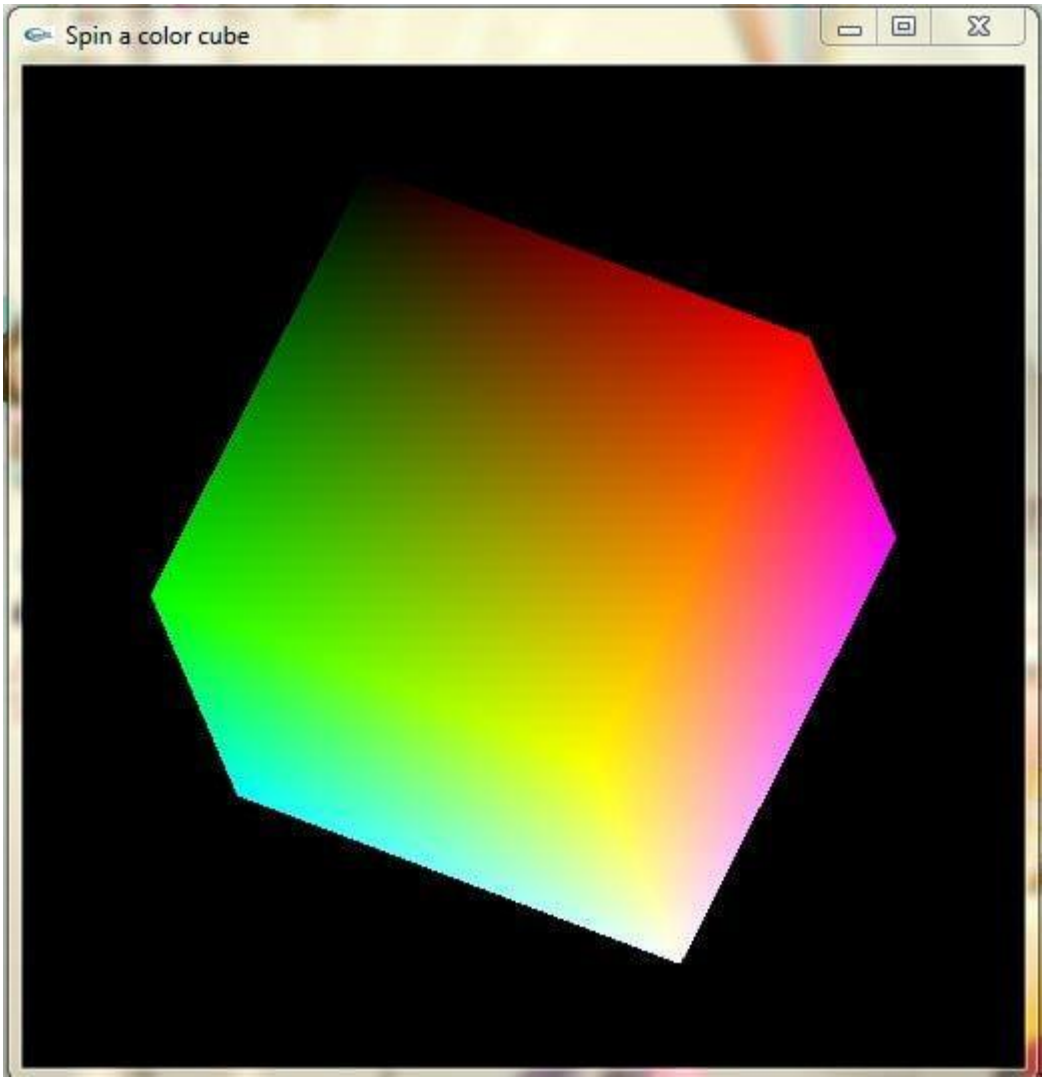


Program 3:- Draw a color cube and spin it using OpenGL transformation matrices.

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

        GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
        {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},
        {1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};

        GLfloat normals[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
        {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},
        {1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};

        GLfloat colors[][3] = {{0.0,0.0,0.0},{1.0,0.0,0.0},
        {1.0,1.0,0.0}, {0.0,1.0,0.0}, {0.0,0.0,1.0},
        {1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0}};
```

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

void colorcube(void)
{

/* map vertices to faces */

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

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

void display(void)
{
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        glRotatef(theta[0], 1.0, 0.0, 0.0);
        glRotatef(theta[1], 0.0, 1.0, 0.0);
        glRotatef(theta[2], 0.0, 0.0, 1.0);
 colorcube();
 glFlush();
 glutSwapBuffers();
}

void spinCube()
```

```
{
        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);
//   glOrtho(l,b,n,r,t,f);
   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);
   glutCreateWindow("Spin a Color Cube");
   glutReshapeFunc(myReshape);
   glutDisplayFunc(display);
   glutIdleFunc(spinCube);
   glutMouseFunc(mouse);
   glEnable(GL_DEPTH_TEST);
   glutMainLoop();
}
```



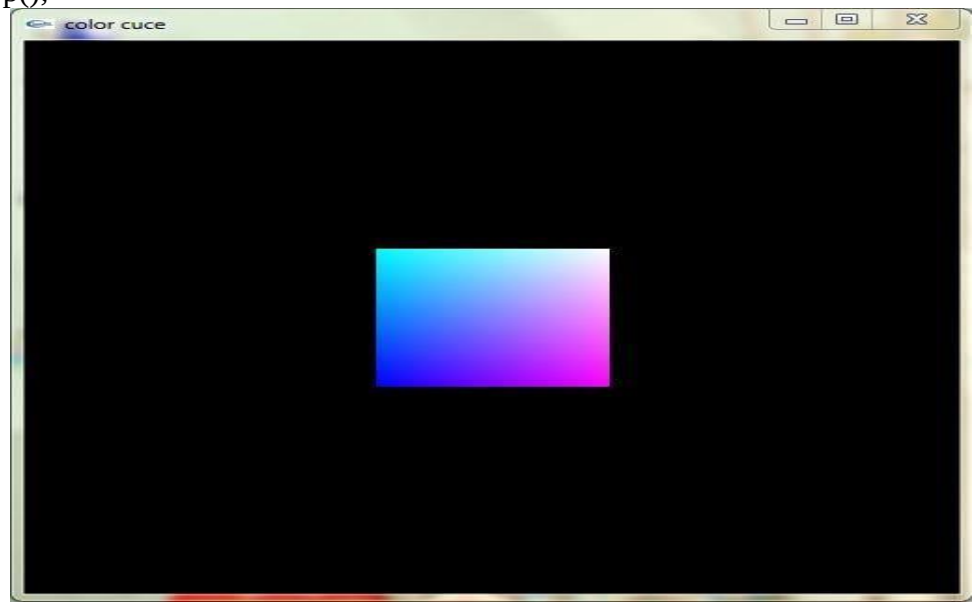
Program 4:- Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing

```
/* Rotating cube with viewer movement */

/* We use the Lookat function in the display callback to point
the viewer, whose position can be altered by the x,X,y,Y,z, and Z keys.
The perspective view is set in the reshape callback */

#include <stdlib.h>
#include <GL/glut.h>
        GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
        {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},
        {1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};

        GLfloat normals[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
        {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},
        {1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};

        GLfloat colors[][3] = {{0.0,0.0,0.0},{1.0,0.0,0.0},
        {1.0,1.0,0.0}, {0.0,1.0,0.0}, {0.0,0.0,1.0},
```

```
        {1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0}};

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

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

void display(void)
{

 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

/* Update viewer position in modelview matrix */

        glLoadIdentity();
        gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

/* rotate cube */

        glRotatef(theta[0], 1.0, 0.0, 0.0);
        glRotatef(theta[1], 0.0, 1.0, 0.0);
        glRotatef(theta[2], 0.0, 0.0, 1.0);

 colorcube();
```

```
 glFlush();
        glutSwapBuffers();
}

void mouse(int btn, int state, int x, int y)
{
        if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
        if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
        if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
        theta[axis] += 2.0;
        if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
        display();
}
void keys(unsigned char key, int x, int y)
{

/* Use x, X, y, Y, z, and Z keys to move viewer */

   if(key == 'x') viewer[0]-= 1.0;
   if(key == 'X') viewer[0]+= 1.0;
   if(key == 'y') viewer[1]-= 1.0;
   if(key == 'Y') viewer[1]+= 1.0;
   if(key == 'z') viewer[2]-= 1.0;
   if(key == 'Z') viewer[2]+= 1.0;
   display();
}

void myReshape(int w, int h)
{
 glViewport(0, 0, w, h);

/* Use a perspective view */

 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
        if(w<=h) glFrustum(-2.0, 2.0, -2.0 * (GLfloat) h/ (GLfloat) w,
      2.0* (GLfloat) h / (GLfloat) w, 2.0, 20.0);
        else glFrustum(-2.0, 2.0, -2.0 * (GLfloat) w/ (GLfloat) h,
      2.0* (GLfloat) w / (GLfloat) h, 2.0, 20.0);

/* Or we can use gluPerspective */

 /* gluPerspective(45.0, w/h, -10.0, 10.0); */

 glMatrixMode(GL_MODELVIEW);
}

void  main(int argc, char **argv)
{
```

```
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
 glutInitWindowSize(500, 500);
 glutCreateWindow("Color cube");
 glutReshapeFunc(myReshape);
 glutDisplayFunc(display);
glutMouseFunc(mouse);
glutKeyboardFunc(keys);
glEnable(GL_DEPTH_TEST);
 glutMainLoop();
}
```



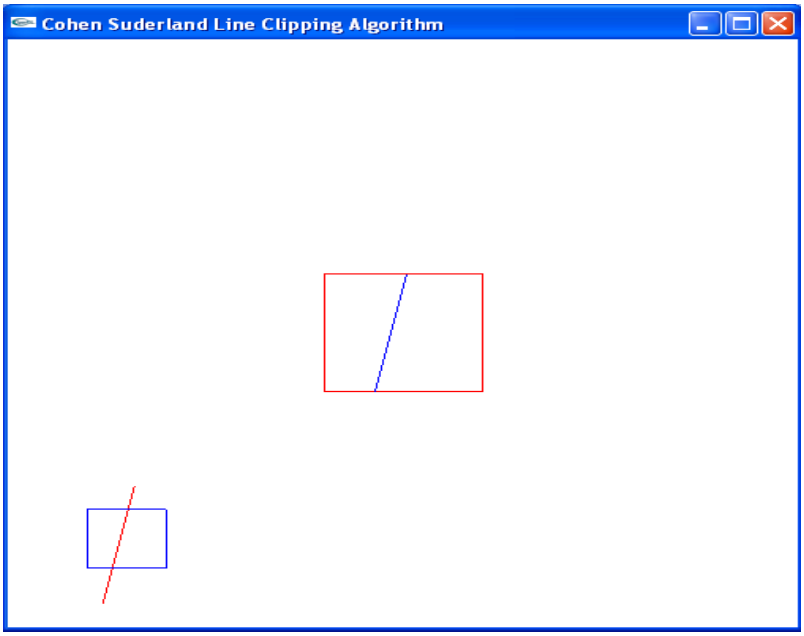
Program 5:- Clip a lines using Cohen-Sutherland algorithm.

```
#include<stdio.h>
#include<GL/glut.h>
#define outcode int

double xmin=50,ymin=50,xmax=100,ymax=100;
double xvmin=200,yvmin=200,xvmax=300,yvmax=300;
double x0,y0,x1,y1;
const int RIGHT=8;
const int LEFT=2;
const int TOP=4;
const int BOTTOM=1;
outcode ComputeOutCode(double x, double y);

void CohenSutherland(double x0, double y0, double x1, double y1)
{
   outcode outcode0, outcode1, outcodeOut;
   bool accept=false, done=false;
   outcode0=ComputeOutCode(x0,y0);
   outcode1=ComputeOutCode(x1,y1);
   do
   {
      if(!(outcode0 | outcode1))
      {
```

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

if(accept)
{
    double sx=(xvmax-xvmin)/(xmax-xmin);
    double sy=(yvmax-yvmin)/(ymax-ymin);
    double vx0=xvmin+(x0-xmin)*sx;
    double vy0=yvmin+(y0-ymin)*sy;
    double vx1=xvmin+(x1-xmin)*sx;
    double vy1=yvmin+(y1-ymin)*sy;
    glColor3f(1.0,0.0,0.0);
    glBegin(GL_LINE_LOOP);
        glVertex2f(xvmin, yvmin);
        glVertex2f(xvmax, yvmin);
        glVertex2f(xvmax, yvmax);
        glVertex2f(xvmin, yvmax);
    glEnd();
```

```
        glColor3f(0.0,0.0,1.0);
        glBegin(GL_LINES);
            glVertex2d(vx0,vy0);
            glVertex2d(vx1,vy1);
        glEnd();
    }
}

outcode ComputeOutCode(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 display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,0.0,0.0);
    glBegin(GL_LINES);
        glVertex2d(x0,y0);
        glVertex2d(x1,y1);
    glEnd();
    glColor3f(0.0,0.0,1.0);
    glBegin(GL_LINE_LOOP);
        glVertex2f(xmin, ymin);
        glVertex2f(xmax, ymin);
        glVertex2f(xmax, ymax);
        glVertex2f(xmin, ymax);
    glEnd();
    CohenSutherland(x0,y0,x1,y1);
    glFlush();
}

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

void main(int argc, char** argv)
{
    printf("Enter the end points of the line: ");
    scanf("%lf%lf%lf%lf", &x0,&y0,&x1,&y1);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Cohen-Sutherland Line Clipping");
    glutDisplayFunc(display);
```

```
    myinit();
    glutMainLoop();
}
```

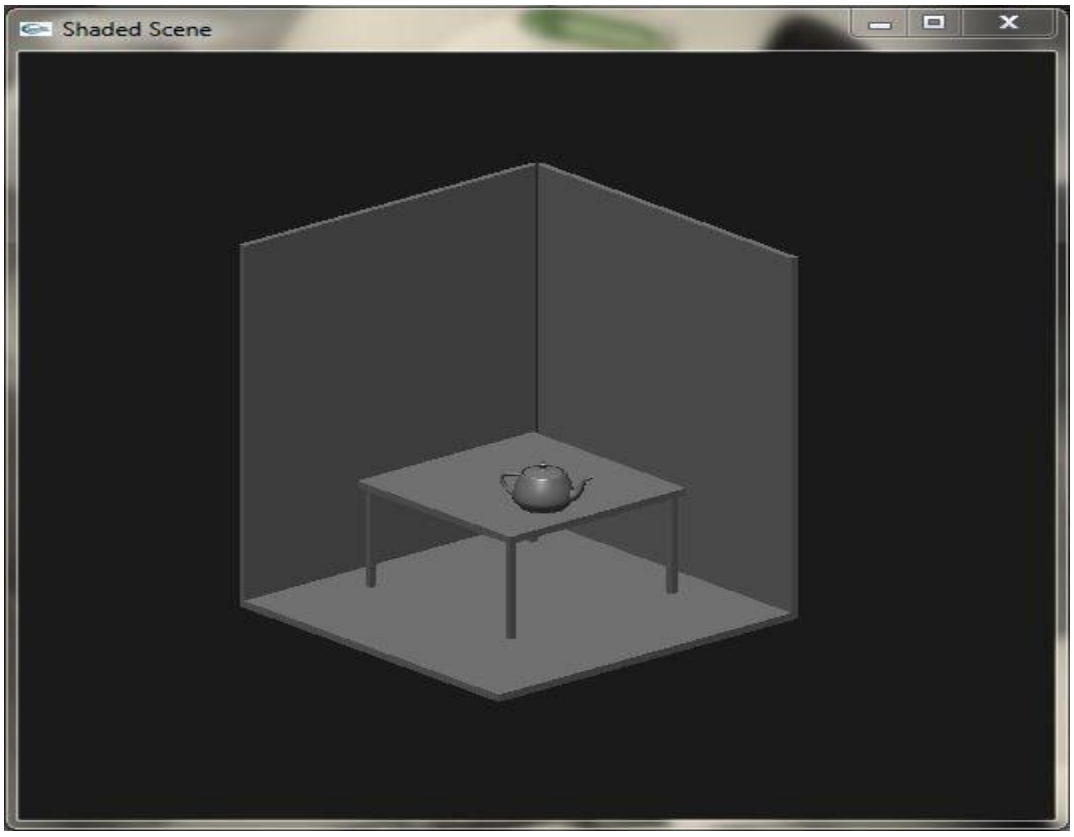


Program 6:- To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene

```
#include<GL/glut.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)
{
    GLfloat mat_ambient[]={0.7f,0.7f,0.7f,1.0f};
    GLfloat mat_diffuse[]={0.5f,0.5f,0.5f,1.0f};
    GLfloat mat_specular[]={1.0f,1.0f,1.0f,1.0f};
    GLfloat mat_shininess[]={50.0f};
    glMaterialfv(GL_FRONT,GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT,GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT,GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT,GL_SHININESS, mat_shininess);

    GLfloat lightintensity[]={0.7f,0.7f,0.7f,1.0f};
    GLfloat lightposition[]={2.0f,6.0f,3.0f,0.0f};
    glLightfv(GL_LIGHT0, GL_POSITION, lightposition);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightintensity);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    double winht=1.0;
    glOrtho(-winht*64/48, winht*64/48, -winht, winht,
                                0.1, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(2.3,1.3,2.0,0.0,0.25,0.0,0.0,1.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glTranslated(0.6,0.38,0.5);
    glRotated(30,0,1,0);
    glutSolidTeapot(0.08);
    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);
    glutInitWindowPosition(50,50);
    glutInitWindowSize(400,300);
    glutCreateWindow("Shaded Scene");
    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();
}
```

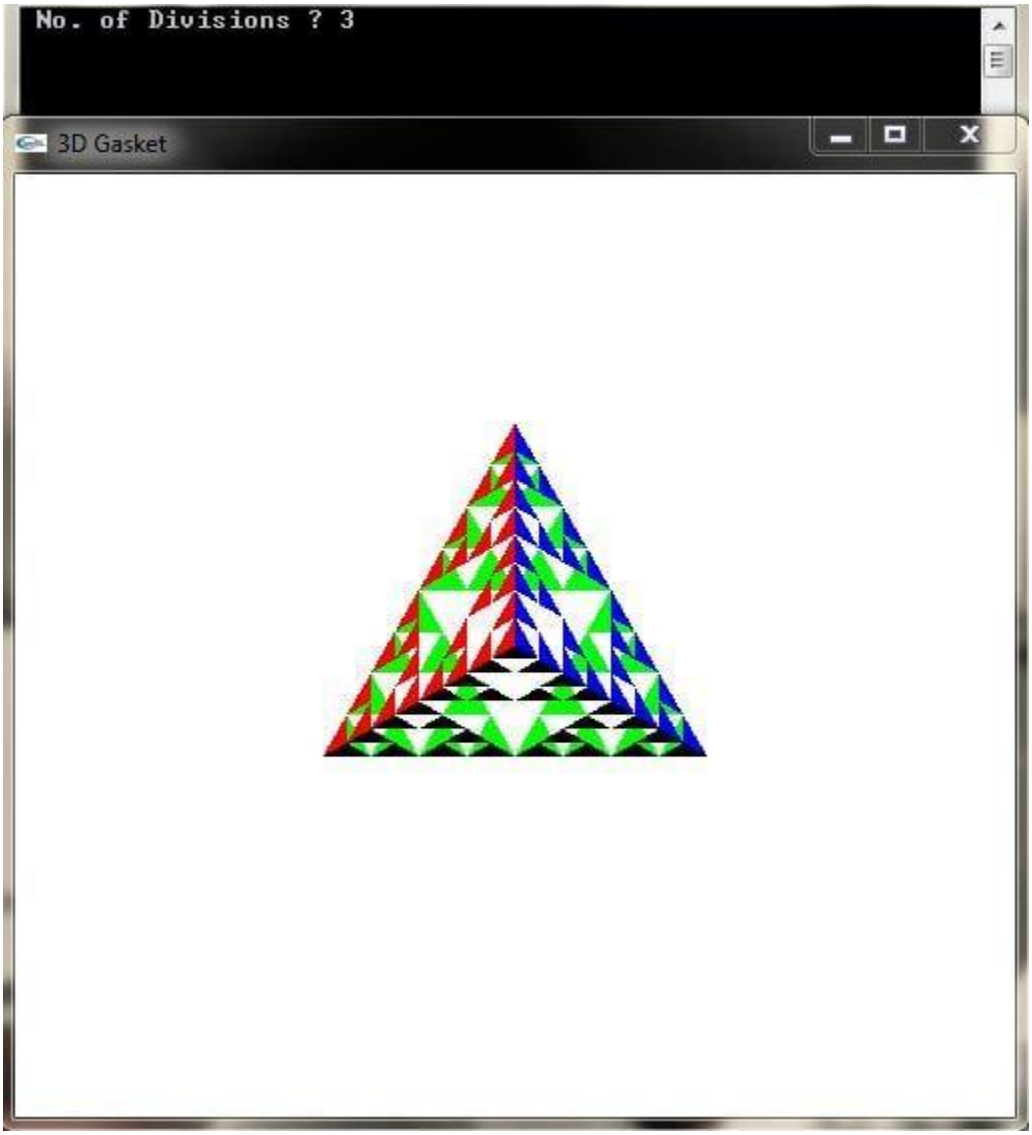


Program 7:- Design, develop and implement recursively subdivide a tetrahedron to form 3D Sierpinski gasket. The number of recursive steps is to be specified by the user.

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

typedef float point[3];
```

```
point v[]={ {0.0, 0.0, 1.0},
          {0.0, 0.942809, -0.33333},
          {-0.816497, -0.471405, -0.333333},
          {0.816497, -0.471405, -0.333333}};
static GLfloat theta[] = {0.0,0.0,0.0};
int n;

void triangle( point a, point b, point c)
{
   glBegin(GL_POLYGON);
   glNormal3fv(a);
   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, -2.0, 2.0, -10.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
    glutPostRedisplay();
}

void main(int argc, char **argv)
{
    printf(" No. of Divisions ? ");
    scanf("%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();
}
```


No. of Divisions ? 3
3D Gasket

Program 8:- Develop a menu driven program to animate a flag using Bezier Curve algorithm
//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;
int animate=1;
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()
{
if(animate)
{
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);
glColor3f(255/255, 153/255.0, 51/255.0); //Indian flag: Orange color code
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glColor3f(1, 1, 1); //Indian flag: white color code
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glColor3f(19/255.0, 136/255.0, 8/255.0); //Indian flag: green color code
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
```

```
glPopMatrix();
glColor3f(0.7, 0.5,0.3);
glLineWidth(5);
glBegin(GL_LINES);
glVertex2f(20,100);
glVertex2f(20,40);
glEnd();
glFlush();
glutPostRedisplay();
glutSwapBuffers();
}
}
// Menu exit
void handlemenu(int value)
{
switch (value) {case 4:
exit(0);
break;
}
}
//Colors menu
void cmenu(int value){
switch(value){
case 1:
animate=1;
glutPostRedisplay();
break;
case 2:
animate=0;
glutPostRedisplay();
break;
}
}
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");
int a_menu=glutCreateMenu(cmenu);
```

```
glutAddMenuEntry("start", 1);
glutAddMenuEntry("stop", 2);
glutCreateMenu(handlemenu);
glutAddSubMenu("animate", a_menu);
glutAddMenuEntry("Quit",4);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutDisplayFunc(displayFcn);
glutReshapeFunc(winReshapeFun);
glutMainLoop();
}
```

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

```
#include<stdio.h>
#include<math.h>
#include<iostream>
#include<GL/glut.h>
int le[500], re[500], flag=0 ,m;
void init()
{
gluOrtho2D(0, 500, 0, 500);
}
void edge(int x0, int y0, int x1, int y1)
{
if (y1<y0)
{
int tmp;
tmp = y1;
y1 = y0;
y0 = tmp;
tmp = x1;
x1 = x0;
x0 = tmp;
}
int x = x0;
m = (y1 - y0) / (x1 - x0);
for (int i = y0; i<y1; i++)
{
if (x<le[i])
le[i] = x;
if (x>re[i])
re[i] = x;
x += (1 / m);
}
}
void display()
{
glClearColor(1, 1, 1, 1);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0, 0, 1);
glBegin(GL_LINE_LOOP);
glVertex2f(200, 100);
glVertex2f(100, 200);
glVertex2f(200, 300);
glVertex2f(300, 200);
glEnd();
for (int i = 0; i<500; i++)
{
le[i] = 500;
re[i] = 0;
}
```

```
edge(200, 100, 100, 200);
edge(100, 200, 200, 300);
edge(200, 300, 300, 200);
edge(300, 200, 200, 100);
if (flag == 1)
{
for (int i = 0; i < 500; i++)
{
if (le[i] < re[i])
{
for (int j = le[i]; j < re[i]; j++)
{
glColor3f(1, 0, 0);
glBegin(GL_POINTS);
glVertex2f(j, i);
glEnd();
}
}
}
}
glFlush();
}
void ScanMenu(int id)
{
if (id == 1) {
flag = 1;
}
else if (id == 2) {
flag = 0;
}
else { exit(0); }
glutPostRedisplay();
}
int main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitWindowPosition(100, 100);
glutInitWindowSize(500, 500);
glutCreateWindow("scan line");
init();
glutDisplayFunc(display);
glutCreateMenu(ScanMenu);
glutAddMenuEntry("scanfill", 1);
glutAddMenuEntry("clear", 2);
glutAddMenuEntry("exit", 3);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutMainLoop();
return 0;
```