Program 1: Implement Brensenham'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) {
      glPointSize(3.0);
      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();
}

void 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();
}
```

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

```
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>
GLfloat
house[3][3]={{100.0,150.0,200.0},{100.0,150.0,100.0},{1.0,1.0,1.0}};
GLfloat rot_mat[3][3]={{0},{0},{0}};
GLfloat result[3][3]={{0},{0},{0}};
GLfloat h;
GLfloat k;
GLfloat theta,rad;
int ch;

void multiply()
{
      int i,j,l;
      for(i=0;i<3;i++)
            for(j=0;j<3;j++)
            {
                  result[i][j]=0;
                  for(l=0;l<3;l++)
                        result[i][j]=result[i][j]+rot_mat[i][l]*house[
                  l][j];
            }
}

void rotate()
{
      GLfloat m,n;
      m=-h*(cos(theta)-1)+k*(sin(theta));
      n=-k*(cos(theta)-1)-h*(sin(theta));
      rot_mat[0][0]=cos(theta);
      rot_mat[0][1]=-sin(theta);
      rot_mat[0][2]=m;
      rot_mat[1][0]=sin(theta);
      rot_mat[1][1]=cos(theta);
      rot_mat[1][2]=n;
      rot_mat[2][0]=0;
      rot_mat[2][1]=0;
      rot_mat[2][2]=1;
      multiply();
}

void drawhouse(GLfloat mat[3][3])
{
      glBegin(GL_TRIANGLES);
      glVertex2f(mat[0][0],mat[1][0]);
      glVertex2f(mat[0][1],mat[1][1]);
      glVertex2f(mat[0][2],mat[1][2]);
      glEnd();
}
```

```
void rotation()
{
      int ch;
      printf("enter the choice to rotate\n1.fixed
      point\n2.origin\n");
      scanf("%d",&ch);
      switch(ch)
      {
            case 1:h=100,k=100;
                  rotate();
                  glColor3f(1.0,0.0,0.0);
                  drawhouse(rot_mat);
                  break;
            default:printf("\ninvalid option");
      }
glutPostRedisplay();
}

void display()
{
      glClear(GL_COLOR_BUFFER_BIT);
      theta=rad;
      glColor3f(0.0,0.3,0.7);
      drawhouse(house);
      if(ch==1)
      {
            h=100;
            k=100;
            rotate();
            glColor3f(0.0,1.0,0.6);
      }
      if(ch==2)
      {
            h=(house[0][0]+house[0][1]+house[0][2])/3;
            k=(house[1][0]+house[1][1]+house[1][2])/3;
            rotate();
            glColor3f(1.0,0.4,1.0);
      }
      drawhouse(result);
      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);
}
```

```
int main(int argc,char** argv)
{
      printf("\nenter the rotation angle:");
      scanf("%f",&theta);
      printf("\nenter the choice1.fixed point2.origin");
      scanf("%d",&ch);
      rad=theta*(3.14/180.0);
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
      glutInitWindowSize(500,500);
      glutInitWindowPosition(0,0);
      glutCreateWindow("trianglerotation");
      glutDisplayFunc(display);
      myinit();
      glutMainLoop();
}
```

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

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

GLfloat vertices[8][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[8][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}};

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]);
      glVertex3fv(vertices[a]);
      glColor3fv(colors[b]);
      glVertex3fv(vertices[b]);
      glColor3fv(colors[c]);
      glVertex3fv(vertices[c]);
      glColor3fv(colors[d]);
      glVertex3fv(vertices[d]);
      glEnd();
}

void colorcube(void)
{
      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 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)h/(GLfloat)w,
            2.0*(GLfloat)h/(GLfloat)w,-2.0,2.0,-10.0,10.0);
      glMatrixMode(GL_MODELVIEW);
}
```

```
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] +=0.05;
      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 main(int argc, char** argv)
{
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
      glutInitWindowSize(750,750);
      glutCreateWindow("Rotating 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.

```
#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] += 20;
      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);

 glMatrixMode(GL_MODELVIEW);
}

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

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;
float x0,y0,x1,y1;
const int RIGHT=2;
const int LEFT=1;
const int TOP=8;
const int BOTTOM=4;
outcode ComputeOutCode(double x,double y);

void CohenSutherlandLineClipAndDraw(double x0,double y0,double
x1,double y1)
{
      outcode outcode0,outcode1,outcodeOut;
      int accept=0,done=0;
      outcode0=ComputeOutCode(x0,y0);
      outcode1=ComputeOutCode(x1,y1);
      do
      {
            if(!(outcode0 | outcode1))
            {
                  accept=1;
                   done=1;
            }
            else if(outcode0 & outcode1)
                  done=1;
            else
            {
                  double x,y,m;
                  m=(y1-y0)/(x1-x0);
                  outcodeOut=outcode0?outcode0:outcode1;
                  if(outcodeOut & TOP)
                  {
                        x=x0+(ymax-y0)/m;
                        y=ymax;
                  }
                  else if(outcodeOut & BOTTOM)
                  {
                        x=x0+(ymin-y0)/m;
                        y=ymin;
                  }
                  else if(outcodeOut & RIGHT)
                  {
                        y=y0+(xmax-x0)*m;
                        x=xmax;
                  }
```

```
                else
                {
                      y=y0+(xmin-x0)*m;
                      x=xmin;
                }
                if(outcodeOut==outcode0)
                {
                      x0=x;
                      y0=y;
                      outcode0=ComputeOutCode(x0,y0);
                }
                else
                {
                      x1=x;
                      y1=y;
                      outcode1=ComputeOutCode(x1,y1);
                }
          }
     }
     while(!done);
     glColor3f(1.0,0.0,0.0);
     glBegin(GL_LINE_LOOP);
     glVertex2f(xvmin,yvmin);
     glVertex2f(xvmax,yvmin);
     glVertex2f(xvmax,yvmax);
     glVertex2f(xvmin,yvmax);
     glEnd();
     printf("\n %f %f:%f %f",x0,y0,x1,y1);
     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_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();
  CohenSutherlandLineClipAndDraw(x0,y0,x1,y1);
  glFlush();
}

void myinit()
{
      glClearColor(1.0,1.0,1.0,1.0);
      glColor3f(1.0,1.0,0.0);
      glPointSize(1.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(0.0,499.0,0.0,499.0);
}

void main(int argc,char **argv)
{
      printf("Enter end points: ");
      scanf("%f %f %f %f",&x0,&y0,&x1,&y1);
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
      glutInitWindowSize(500,500);
      glutCreateWindow("When Sutherland line clipping algorithm");
      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 obj(double tx,double ty,double tz,double sx,double sy,
double sz)
{
      glRotated(50,0,1,0);
      glRotated(10,-1,0,0);
      glRotated(11.7,0,0,-1);
      glTranslated(tx,ty,tz);
      glScaled(sx,sy,sz);
      glutSolidCube(1.0);
      glLoadIdentity();
}

void display()
{
      glViewport(0,500,500,500);
      glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
      obj(0,0,0.5,1,1,0.04);
      obj(-0.5,0,0,0.04,1,1);
      obj(0,-3.0,0,0.02,0.2,0.02);
      obj(0,-0.3,-0.4,0.02,0.2,0.02);
      obj(0.4,-0.3,0,0.02,0.2,0.02);
      obj(0.4,-0.3,-0.4,0.02,0.2,0.02);
      obj(0.2,-0.18,-0.2,0.8,0.02,0.8);
      obj(0,-0.5,0.02,1,0.02,1);
      glTranslated(0.03,-0.2,-0.5);
      glutSolidTeapot(0.1);
      glLoadIdentity();
      glFlush();
}

void main(int argc,char** argv)
{
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
      GLfloat ambient[]={0.3,0.4,0.5,1};
      GLfloat Light_pos[]={27,80,2,3};
      glutInitWindowSize(1000,1000);
      glutInitWindowPosition(10,10);
      glutCreateWindow("Teapot and Table");
      glutDisplayFunc(display);
      glEnable(GL_LIGHTING);
      glEnable(GL_LIGHT0);
      glMaterialfv(GL_FRONT,GL_AMBIENT,ambient);
      glLightfv(GL_LIGHT0,GL_POSITION,Light_pos);
      glEnable(GL_DEPTH_TEST);
      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<stdio.h>
#include<GL/glut.h>

GLfloat vertices[4][3]={{0.0,0.0,1.0},{0.0,0.94,-0.33},
{-0.82,-0.47,-0.33},{0.82,-0.47,0.33}};

GLfloat colors[4][3]={{1.0,0.0,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0},
{0.0,0.0,0.0}};

int n;

void triangle(GLfloat *va,GLfloat *vb,GLfloat *vc)
{
     glVertex3fv(va);
     glVertex3fv(vb);
     glVertex3fv(vc);
}
void tetra(GLfloat *a,GLfloat *b,GLfloat *c,GLfloat *d)
{
     glColor3fv(colors[0]);
     triangle(a,b,c);

     glColor3fv(colors[1]);
     triangle(a,c,d);

     glColor3fv(colors[2]);
     triangle(a,d,b);

     glColor3fv(colors[3]);
     triangle(b,d,c);
}

void divide_tetra(GLfloat *a,GLfloat *b,GLfloat *c,GLfloat *d,int m)
{
     GLfloat mid[6][3];
     int j;
     if(m>0)
     {
          for(j=0;j<3;j++) mid[0][j]=(a[j]+b[j])/2;
          for(j=0;j<3;j++) mid[1][j]=(a[j]+c[j])/2;
          for(j=0;j<3;j++) mid[2][j]=(a[j]+d[j])/2;
          for(j=0;j<3;j++) mid[3][j]=(b[j]+c[j])/2;
          for(j=0;j<3;j++) mid[4][j]=(b[j]+d[j])/2;
          for(j=0;j<3;j++) mid[5][j]=(c[j]+d[j])/2;

          divide_tetra(a,mid[0],mid[1],mid[2],m-1);
          divide_tetra(mid[0],b,mid[3],mid[4],m-1);
          divide_tetra(mid[1],mid[3],c,mid[5],m-1);
          divide_tetra(mid[2],mid[4],mid[5],d,m-1);
     }
```

```
      else
            tetra(a,b,c,d);
}

void display(void)
{
      glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
      glClearColor(0.0,0.0,0.0,0.0);
      glBegin(GL_TRIANGLES);
      divide_tetra(vertices[0],vertices[1],vertices[2],vertices[3],
      n);
      glEnd();
      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)h/(GLfloat)w,
            2.0*(GLfloat)h/(GLfloat)w,-2.0,2.0,-10.0,10.0);
      glMatrixMode(GL_MODELVIEW);
      glutPostRedisplay();
}

void main(int argc,char** argv)
{
      printf("Enter no of recursive steps to divide tetrahedron");
      scanf("%d",&n);
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
      glutInitWindowSize(500,500);
      glutCreateWindow("3D sierpinski gasket");
      glutReshapeFunc(myReshape);
      glutDisplayFunc(display);
      glEnable(GL_DEPTH_TEST);
      glutMainLoop();
}
```

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);
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();
}
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();
}
```

Program 9 : Program to fill any given polygon using scan-line area filling algorithm.

```
// Scan-Line algorithm for filling a
polygon #define BLACK 0
#include
<stdlib.h>
#include
<stdio.h>
#include
<GL/glut.h>
float x1,x2,x3,x4,y1,y2,y3,y4;
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,int value)
{
      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,BLACK);
	}

}
void display()
{
x1=200.0;y1=200.0;x2=100.0;y2=300.0;x3=200.0;y3=400.0;x4=300.0;y4=300.0;

glClear(GL_COLOR_BUFFER_BIT
); glColor3f(0.0, 0.0,
1.0);
glBegin(GL_LINE_LOOP);
  glVertex2f(x1,y1);
  glVertex2f(x2,y2);
  glVertex2f(x3,y3);
  glVertex2f(x4,y4);
  glEnd();
  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 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"); glutDisplayFunc(display);
	myinit();
	glutMainLoop();
}
```