1. BRESENHAM LINE

```
#include<GL/glut.h>
#include<stdio.h>
int x1, y1, x2, y2;
void draw_pixel(int x, int y)
{
glColor3f(1.0,0.0,0.0);
glBegin(GL_POINTS);
glVertex2i(x, y);
glEnd();
}
void bresenhams_line_draw(int x1, int y1, int x2, int y2)
{
int dx = x2 - x1;
int dy = y2 - y1;
if(dx>dy)
{
int decision_parameter = 2*dy - dx;
int x = x1; // initial x
int y = y1; // initial y
if(dx < 0)
{
x = x2;
y = y2;
x2 = x1;
}
draw_pixel(x, y);
while(x < x2)
{
if(decision_parameter >= 0)
{
x = x+1;
y = y+1;
decision_parameter = decision_parameter + 2*dy - 2*dx * (y+1 - y);
}
else
{
x = x+1;
y = y;
decision_parameter = decision_parameter + 2*dy;
}
draw_pixel(x, y);
```

```
}
}
else if(dx<dy)
{
int decision_parameter = 2*dx - dy;
int x = x1; // initial x
int y = y1; // initial y
if(dy < 0)
{
x = x2;
y = y2;
y2 = y1;
}

draw_pixel(x, y);
while(y < y2)
{
if(decision_parameter >= 0)
{
x = x+1;
y = y+1;
decision_parameter = decision_parameter + 2*dx - 2*dy * (x+1 - x);
}
else
{
y = y+1;
x = x;
decision_parameter = decision_parameter + 2*dx;
}
draw_pixel(x, y);
}
}
else
{
int x = x1;
int y = y1;
draw_pixel(x, y);
while(x < x2)
{
x = x+1;
y = y+1;
draw_pixel(x, y);
}
```

```
}
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
bresenhams_line_draw(x1, y1, x2, y2);
glFlush();
}
int main(int argc, char **argv)
{
printf( "Enter Start Points (x1,y1)\n");
scanf("%d %d", &x1, &y1);
printf( "Enter End Points (x2,y2)\n");
scanf("%d %d", &x2, &y2);
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutCreateWindow("Bresenham's Line Drawing");
glClearColor(1,1,1,1);
gluOrtho2D(0.0, 500.0, 0.0, 500.0);
glutDisplayFunc(display);
glutMainLoop();
}
```

2. TRIANGLE ROTATE

```
#include<GL/glut.h>
#include<stdio.h>
int x,y;
int where_to_rotate=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(1.0,0.0,0.0);
	glBegin(GL_POLYGON);
	glVertex2f(x,y);
	glVertex2f(x+400,y+300);
	glVertex2f(x+300,y+0);
	glEnd();
}
float rotate_angle=0.0;
float translate_x=0.0,translate_y=0.0;

void display()
{
	glClear(GL_COLOR_BUFFER_BIT);
	glLoadIdentity();
	glColor3f(1,1,1);
	draw_pixel(0.0,0.0);
	if(where_to_rotate==1) //Rotate Around origin
	{
	translate_y=translate_x=0.0;  // move the triangle to origin
	rotate_angle+=.1;  //speed of rotation
	}
	if(where_to_rotate==2) //Rotate Around Fixed Point
	{
	translate_x=x;
	translate_y=y;		//move triangle to the points
	rotate_angle+=.1;  // speed of rotation
```

```
        glColor3f(0.0,0.0,1.0);
        draw_pixel(x,y);
        }
        glTranslatef(translate_x,translate_y,0.0);      // to keep triangle to the origin point
        glRotatef(rotate_angle,0.0,0.0,1.0);
        glTranslatef(-translate_x,-translate_y,0.0);  // to keep triangle to the fixed point
        triangle(translate_x,translate_y);
        glutPostRedisplay();  // rotating
        glutSwapBuffers();      // to display red triangle on screen
}
void rotateMenu (int option)
{
        if(option==1)
        where_to_rotate=1;
        if(option==2)
        where_to_rotate=2;
        if(option==3)
        where_to_rotate=3;
}
int main(int argc, char **argv)
{
        printf( "Enter Fixed Points (x,y) for Rotation: \n");
        scanf("%d %d", &x, &y);

        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
        glutInitWindowSize(800, 800);
        glutCreateWindow("Create and Rotate Triangle");
        // start with init functions
        glClearColor(0.0,0.0,0.0,1.0); //setting to black
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-800.0, 800.0, -800.0, 800.0);
        glMatrixMode(GL_MODELVIEW);
        // end of init functions
       glutDisplayFunc(display);
        glutCreateMenu(rotateMenu);
        glutAddMenuEntry("Rotate around ORIGIN",1);
        glutAddMenuEntry("Rotate around FIXED POINT",2);
        glutAddMenuEntry("Stop Rotation",3);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
        glutMainLoop();
}
```

3. CUBE ROTATE

```
#include <stdlib.h>
 #include <GL/glut.h>
 GLfloat vertices[][3] = {{-1,-1,-1},{1,-1,-1},{1,1,-1},{-1,1,-1},
 {-1,-1,1},{1,-1,1}, {1,1,1},{-1,1,1}};
 GLfloat colors[][3] = {{1,0,0},{1,1,0},{0,1,0},{0,0,1},
 {1,0,1},{1,1,1},{0,1,1},{0.5,0.5,0.5}};
 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(4,5,6,7);
 polygon(0,4,7,3);
 polygon(5,4,0,1);
 polygon(2,3,7,6);
 polygon(1,2,6,5);
 }
 GLfloat theta[] = {0.0,0.0,0.0};
 GLint axis = 2;
 void display(void)
 {
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 glRotatef(theta[0], 1.0, 0.0, 0.0);  //left button rotation
 glRotatef(theta[1], 0.0, 1.0, 0.0);   //right button rotation
 glRotatef(theta[2], 0.0, 0.0, 1.0);       //middle button rotation
 colorcube();
 glutSwapBuffers();
 }
 void spinCube()
```

```
{
theta[axis] += 0.1;   // speed
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(10, 10, w, h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-2.0, 2.0, -2.0 ,2.0 , -10.0, 10.0);
glMatrixMode(GL_MODELVIEW);
}
int main(int argc, char *argv[])
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowSize(500, 500);
glutCreateWindow("Rotating a Color Cube");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glutIdleFunc(spinCube);
glutMouseFunc(mouse);
glEnable(GL_DEPTH_TEST); /* Enable hidden--surface--removal */
glutMainLoop();
}
```

4. CUBE CAMERA ROTATE

```
#include <stdlib.h>
#include <GL/glut.h>
GLfloat vertices[][3] = { {-1,-1,-1},
{1,-1,-1},
{1,1,-1},
{-1,1,-1},
{-1,-1,1},
{1,-1,1},
{1,1,1},
{-1,1,1}
};
GLfloat colors[][3] = { {1,0,0},
{1,1,0},
{0,1,0},
{0,0,1},
{1,0,1},
{1,1,1},
{0,1,1},
{0.5,0.5,0.5}
};
GLfloat theta[] = {0.0,0.0,0.0};
GLint axis = 2;
GLdouble viewer[]= {0.0, 0.0, 5.0}; /* initial viewer location */
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(0,4,7,3);
polygon(5,4,0,1);
```

```
polygon(2,3,7,6);
polygon(1,2,6,5);
polygon(4,5,6,7);
}
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
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)
{
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);
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);
glutMainLoop();
}
```

5. COHEN_SUTHERLAND

```
 #include<GL/glut.h>
#include<math.h>
#include<stdio.h>
#include<iostream>
using namespace std;
void display();
float xmin=-100;
float ymin=-100;
float xmax=100;
float ymax=100;
float xd1,yd1,xd2,yd2;

int code(float x,float y)
{
	int c=0;
	if(y>ymax)c=8;
	if(y<ymin)c=4;
	if(x>xmax)c=c|2;
	if(x<xmin)c=c|1;
	return c;
}

void cohen_Line(float x1,float y1,float x2,float y2)
{
	int c1=code(x1,y1);
	int c2=code(x2,y2);
	float xi=x1;float yi=y1;
	float m=(y2-y1)/(x2-x1);
	while((c1|c2)>0)
	{
	if((c1 & c2)>0)
	{
	exit(0);
	}

	int c=c1;
	if(c==0)
	{
	c=c2;
	xi=x2;
	yi=y2;
```

```
        }
        float x,y;
        if((c & 8)>0)
        {
        y=ymax;
        x=xi+ 1.0/m*(ymax-yi);
        }
        else
        if((c & 4)>0)
        {
        y=ymin;
        x=xi+1.0/m*(ymin-yi);
        }
        else
        if((c & 2)>0)
        {
        x=xmax;
        y=yi+m*(xmax-xi);
        }
        else
        if((c & 1)>0)
        {
        x=xmin;
        y=yi+m*(xmin-xi);
        }

        if(c==c1)
        {
        xd1=x;
        yd1=y;
        c1=code(xd1,yd1);
        }

        if(c==c2)
        {
        xd2=x;
        yd2=y;
        c2=code(xd2,yd2);
        }
}
 display();
}
```

```
void mykey(unsigned char key,int x,int y)
{
        if(key=='c')
        {
        cohen_Line(xd1,yd1,xd2,yd2);
        }
}
void display()
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0.0,1.0,0.0);
  glBegin(GL_LINE_LOOP);
  glVertex2i(xmin,ymin);
  glVertex2i(xmin,ymax);
  glVertex2i(xmax,ymax);
  glVertex2i(xmax,ymin);
  glEnd();
  glColor3f(1.0,0.0,0.0);
  glBegin(GL_LINES);
  glVertex2i(xd1,yd1);
  glVertex2i(xd2,yd2);
  glEnd();
  glFlush();
}
int main(int argc,char** argv)
{
        printf("Enter line co-ordinates:");
        cin>>xd1>>yd1>>xd2>>yd2;
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(600,600);
        glutCreateWindow("Clipping");
        glutDisplayFunc(display);
        glutKeyboardFunc(mykey);
        glClearColor(0.0,0,0,0);
        glMatrixMode(GL_PROJECTION);
        gluOrtho2D(-300,300,-300,300);
        glutMainLoop();
        return 0;
}
```

6. TEA POT

```
#include<GL/glut.h>
void teapot(GLfloat x,GLfloat y,GLfloat z)
{
glPushMatrix();
glTranslatef(x,y,z);
glutSolidTeapot(0.1);
glPopMatrix();
}
void tableTop(GLfloat x,GLfloat y,GLfloat z)
{
glPushMatrix();
glTranslatef(x,y,z);
glScalef(0.6,0.02,0.5);
glutSolidCube(1.0);
glPopMatrix();
}
void tableLeg(GLfloat x,GLfloat y,GLfloat z)
{
glPushMatrix();
glTranslatef(x,y,z);
glScalef(0.02,0.3,0.02);
glutSolidCube(1.0);
glPopMatrix();
}
void wall(GLfloat x,GLfloat y,GLfloat z)
{
glPushMatrix();
glTranslatef(x,y,z);
glScalef(1.0,1.0,0.02);
glutSolidCube(1.0);
glPopMatrix();
}
void light()
{
GLfloat mat_ambient[]= {1.0,1.0,1.0,1.0};
GLfloat mat_diffuse[]= {0.5,0.5,0.5,1.0};
GLfloat mat_specular[]= {1.0,1.0,1.0,1.0};
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 light_position[]= {2.0,6.0,3.0,1.0};
GLfloat lightIntensity[]= {0.7,0.7,0.7,1.0};
glLightfv(GL_LIGHT0,GL_POSITION,light_position);
glLightfv(GL_LIGHT0,GL_DIFFUSE,lightIntensity);
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
gluLookAt(-2.0,2.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0); // look in 3d
light(); //Adding light source to your project
teapot(0.0,-0.07,0.0); //Create teapot teapot=0.07
tableTop(0.0,-0.15,0.0); //Create table's top tabletop=0.15
tableLeg(0.2,-0.3,0.2); //Create 1st leg leg=0.2
tableLeg(-0.2,-0.3,0.2); //Create 2nd leg
tableLeg(-0.2,-0.3,-0.2); //Create 3rd leg
tableLeg(0.2,-0.3,-0.2); //Create 4th leg
wall(0.0,0.0,-0.5); //Create 1st wall wall =0.5
glRotatef(90.0,1.0,0.0,0.0);
wall(0.0,0.0,0.5); //Create 2nd wall
glRotatef(90.0,0.0,1.0,0.0);
wall(0.0,0.0,0.5); //Create 3rd wall
glFlush();
}

int main(int argc,char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(500,500);
glutCreateWindow("Teapot on a table");
// init functions
glClearColor(0.0,0.0,0.0,1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1.0,1.0,-1.0,1.0,-1.0,10.0);
glMatrixMode(GL_MODELVIEW);
// end of init functions
glutDisplayFunc(display);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_NORMALIZE);
```

```
glEnable(GL_DEPTH_TEST);
glutMainLoop();
}
```

7. SIERPENSKI GASKET

```
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
typedef float point[3];
point v[]= {{0.0,0.0,1.0},{0.0,1.0,0.0},
{-1.0,-0.5,0.0}, {1.0,-0.5,0.0}};
int n;
void triangle(point a,point b,point c)
{
glBegin(GL_POLYGON);
glVertex3fv(a);
glVertex3fv(b);
glVertex3fv(c);
glEnd();
}
void divide_triangle(point a,point b,point c,int m)
{
point v1,v2,v3;
int j;
if(m>0)
{
for(j=0; j<3; j++)
v1[j]=(a[j]+b[j])/2;
for(j=0; j<3; j++)
v2[j]=(a[j]+c[j])/2;
for(j=0; j<3; j++)
v3[j]=(c[j]+b[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)   // 4 faces
{
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)
{
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-2.0,2.0,-2.0,2.0,-10.0,10.0);
glMatrixMode(GL_MODELVIEW);
glutPostRedisplay();
}
int main(int argc,char ** argv)
{
printf("No of Recursive steps/Division: ");
scanf("%d",&n);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutCreateWindow("3D Sierpinski gasket");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glEnable(GL_DEPTH_TEST);
glClearColor(1.0,1.0,1.0,0.0);
glutMainLoop();
return 0;
}
```

8. BEZIER CURVE

```
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
#define PI 3.1416
typedef struct point
{
GLfloat x, y, z;
};
void bino(int n, int *C)
{
int k, j;
for(k=0; k<=n; k++)
{
C[k]=1;
for(j=n; j>=k+1; j--)
C[k]*=j;
for(j=n-k; j>=2; j--)
C[k]/=j;
}
}
void computeBezPt(float u, point *pt1, int cPt, point *pt2, int *C)
{
int k, n=cPt-1;
float bFcn;
pt1 ->x =pt1 ->y = pt1->z=0.0;
for(k=0; k< cPt; k++)
{
bFcn = C[k] * pow(u, k) * pow( 1-u, n-k);
pt1 ->x += pt2[k].x * bFcn;
pt1 ->y += pt2[k].y * bFcn;
pt1 ->z += pt2[k].z * bFcn;
}
}
void bezier(point *pt1, int cPt, int bPt)
{
point bcPt;
float u;
int *C, k;
C= new int[cPt];
bino(cPt-1, C);
glBegin(GL_LINE_STRIP);
```

```
for(k=0; k<=bPt; k++)
{
u=float(k)/float(bPt);
computeBezPt(u, &bcPt, cPt, pt1, C);
glVertex2f(bcPt.x, bcPt.y);
}
glEnd();
delete[]C;
}
float theta = 0;
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
int nCtrlPts = 4, nBCPts =20;
point ctrlPts[4] = {{100, 400, 0}, {150, 450, 0}, {250, 350, 0},
{300, 400, 0}
};
ctrlPts[1].x +=50*sin(theta * PI/180.0);
ctrlPts[1].y +=25*sin(theta * PI/180.0);
ctrlPts[2].x -= 50*sin((theta+30) * PI/180.0);
ctrlPts[2].y -= 50*sin((theta+30) * PI/180.0);
ctrlPts[3].x -= 25*sin((theta) * PI/180.0);
ctrlPts[3].y += sin((theta-30) * PI/180.0);
theta+=0.2;
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 1.0, 1.0);
glPointSize(5);
glPushMatrix();
glLineWidth(5);
glColor3f(1, 0.4, 0.2); //Indian flag: Orange color code
for(int i=0; i<50; i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBCPts);
}
glColor3f(1, 1, 1); //Indian flag: white color code
for(int i=0; i<50; i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBCPts);
}
glColor3f(0, 1, 0); //Indian flag: green color code
for(int i=0; i<50; i++)
```

```
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBCPts);
}
glPopMatrix();
glColor3f(0.7, 0.5,0.3);
glLineWidth(5);
glBegin(GL_LINES);
glVertex2f(100,400);
glVertex2f(100,40);
glEnd();
glutPostRedisplay();
glutSwapBuffers();
}
void init()
{
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0,500,0,500);
}
int main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowPosition(0, 0);
glutInitWindowSize(500,500);
glutCreateWindow("Bezier Curve");
init();
glutDisplayFunc(display);
glutMainLoop();
}
```

9. SCANLINE POLYGON FILL

```
// Scan-Line algorithm for filling a polygon
#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 left_edge[],int right_edge[])
{
float x_slope,x,temp;
int i;
if((y2-y1)<0)
{
temp = y1;
y1 = y2;
y2 = temp;
temp = x1;
x1 = x2;
x2 = temp;
}
if((y2-y1)!=0)
x_slope = (x2-x1)/(y2-y1);
else
x_slope = x2-x1;
x = x1;
for(i = y1; i <= y2; i++)
{
if(x < left_edge[i])
left_edge[i] = x;
if(x > right_edge[i])
right_edge[i] = x;
x = x + x_slope;
}
}
void draw_pixel(int x,int y)
{
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 left_edge[500],right_edge[500];
int i,y;
for(i = 0; i <= 500; i++)
{
left_edge[i]=500;
right_edge[i]=0;
}
edgedetect(x1,y1,x2,y2,left_edge,right_edge);
edgedetect(x2,y2,x3,y3,left_edge,right_edge);
edgedetect(x3,y3,x4,y4,left_edge,right_edge);
edgedetect(x4,y4,x1,y1,left_edge,right_edge);
for(y = 0; y <= 500; y++)
{
if(left_edge[y] <= right_edge[y])
{
for(i = left_edge[y]; i <= right_edge[y]; i++)
{
draw_pixel(i,y);
glFlush();
}
}
}
}
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);
}
int main(int argc, char** argv)
{
glutInit(&argc,argv);
```

```
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutCreateWindow("Filling a Polygon using Scan-line Algorithm");
glClearColor(1.0,1.0,1.0,1.0);
gluOrtho2D(0,499,0,499);
glutDisplayFunc(display);
glutMainLoop();
}
```