**Program 1:**

**Implement Brenham’s line drawing algorithm for all types of slope.**

#include <GL/glut.h>

#include <stdio.h>

int x1, y1, x2, y2;

void myInit()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(0.0, 0.0, 0.0, 1.0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, 500, 0, 500);

}

void draw\_pixel(int x, int y)

{

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

void draw\_line(int x1, int x2, int y1, int y2)

{

int dx, dy, i, e; int incx, incy, inc1, inc2;

int x, y;

dx = x2 - x1;

dy = y2 - y1;

if (dx < 0) dx = -dx;

if (dy < 0) dy = -dy;

incx = 1;

if (x2 < x1) incx = -1;

incy = 1;

if (y2 < y1) incy = -1;

x = x1; y = y1;

if (dx > dy)

{

draw\_pixel(x, y);

e = 2 \* dy - dx;

inc1 = 2 \* (dy - dx);

inc2 = 2 \* dy;

for (i = 0; i < dx; i++)

{

if (e >= 0)

{

y += incy;

e += inc1;

}

else

e += inc2;

x += incx;

draw\_pixel(x, y);

}

}

else

{

draw\_pixel(x, y);

e = 2 \* dx - dy;

inc1 = 2 \* (dx - dy);

inc2 = 2 \* dx;

for (i = 0; i < dy; i++)

{

if (e >= 0)

{

x += incx;

e += inc1;

}

else

e += inc2;

y += incy;

draw\_pixel(x, y);

}

}

}

void myDisplay()

{

draw\_line(x1, x2, y1, y2);

glFlush();

}

int main(int argc, char\*\* argv)

{

printf("Enter (x1, y1, x2, y2)\n");

scanf\_s ("%d %d %d %d", &x1, &y1, &x2, &y2);

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

glutCreateWindow("Bresenham's Line Drawing");

myInit();

glutDisplayFunc(myDisplay);

glutMainLoop();

return 0;

}

**OUTPUT:**

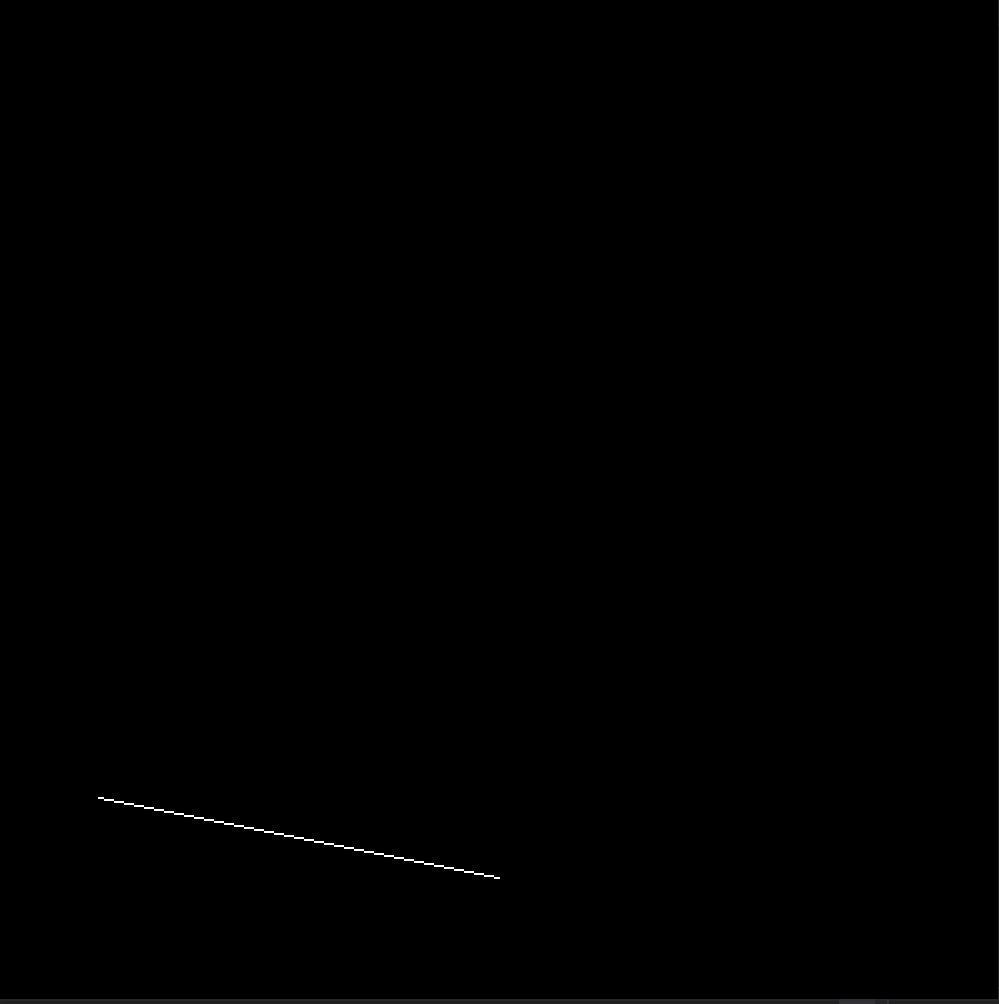
Enter (x1,y1,x2,y2)

50

100

250

60

****

**Program 2**

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

#include<GL/glut.h>

#include<stdio.h>

int x, y; int rFlag = 0;

void draw\_pixel(float x1, float y1)

{

glColor3f(0.0, 0.0, 1.0);

glPointSize(5.0);

glBegin(GL\_POINTS);

glVertex2f(x1, y1);

glEnd();

}

void triangle()

{

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON);

glVertex2f(100, 100);

glVertex2f(250, 400);

glVertex2f(400, 100);

glEnd();

}

float th = 0.0;

float trX = 0.0, trY = 0.0;

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

if (rFlag == 1) //Rotate Around origin

{

trX = 0.0; trY = 0.0; th += 0.1; draw\_pixel(0.0, 0.0);

}

if (rFlag == 2) //Rotate Around Fixed Point

{

trX = x; trY = y; th += 0.1; draw\_pixel(x, y);

} glTranslatef(trX, trY, 0.0);

glRotatef(th, 0.0, 0.0, 1.0);

glTranslatef(-trX, -trY, 0.0);

triangle();

glutPostRedisplay();

glutSwapBuffers();

}

void myInit()

{

glClearColor(0.0, 0.0, 0.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-500.0, 500.0, -500.0, 500.0);

glMatrixMode(GL\_MODELVIEW);

}

void rotateMenu(int option)

{

if (option == 1)

rFlag = 1; if (option == 2)

rFlag = 2; if (option == 3)

rFlag = 3;

}

int main(int argc, char\*\* argv)

{

printf("Enter Fixed Points (x,y) for Roration: \n");

scanf\_s("%d %d", &x, &y);

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

glutCreateWindow("Create and Rotate Triangle");

myInit();

glutDisplayFunc(display);

glutCreateMenu(rotateMenu);

glutAddMenuEntry("Rotate around ORIGIN", 1);

glutAddMenuEntry("Rotate around FIXED POINT", 2);

glutAddMenuEntry("Stop Rotation", 3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

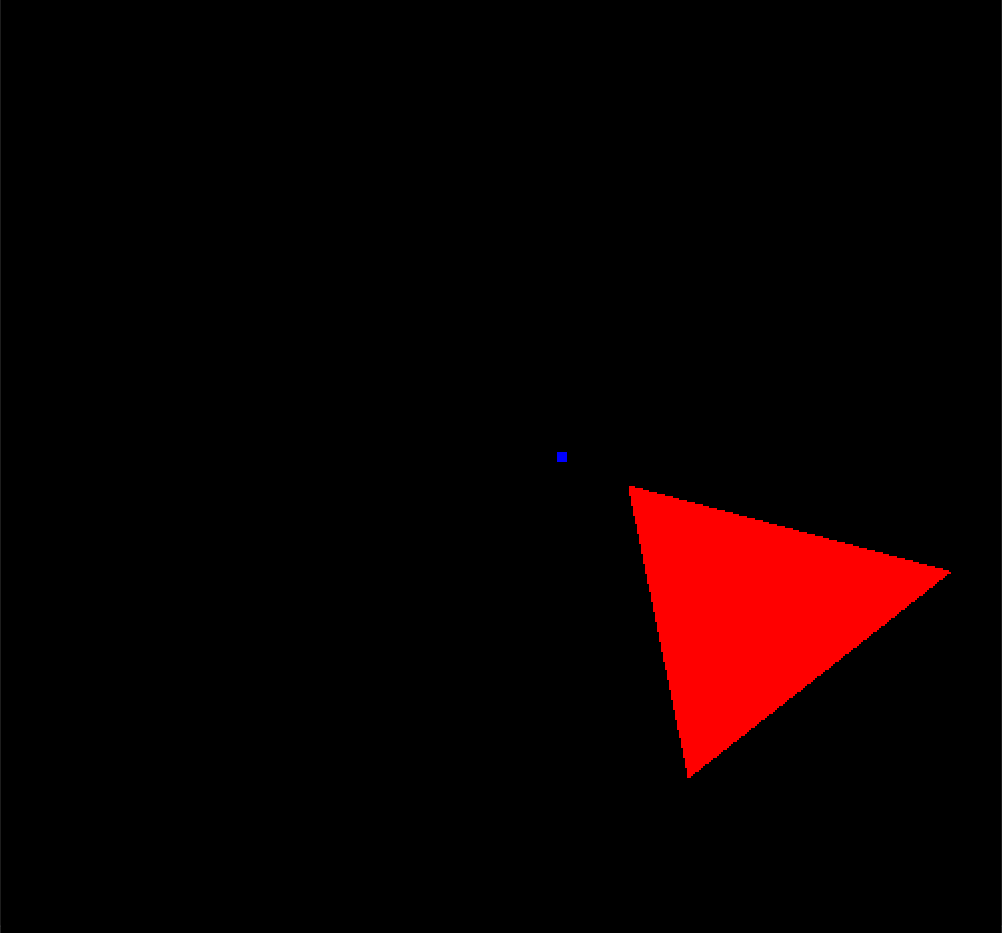
glutMainLoop();

}

**OUTPUT:**

Enter fixed points (x,y) for Rotation:

60 40

****

**Program 3**

**Draw a colour cube and spin it using OpenGL transformation matrices.**

#include<stdlib.h>

#include<GL/glut.h>

//1. Define global arrays for vertices and colors

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(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);

}

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);

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);

}

int main(void)

{

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutCreateWindow("Rotating a color cube");

glutReshapeFunc(myreshape);

glutDisplayFunc(display);

glutIdleFunc(spincube);

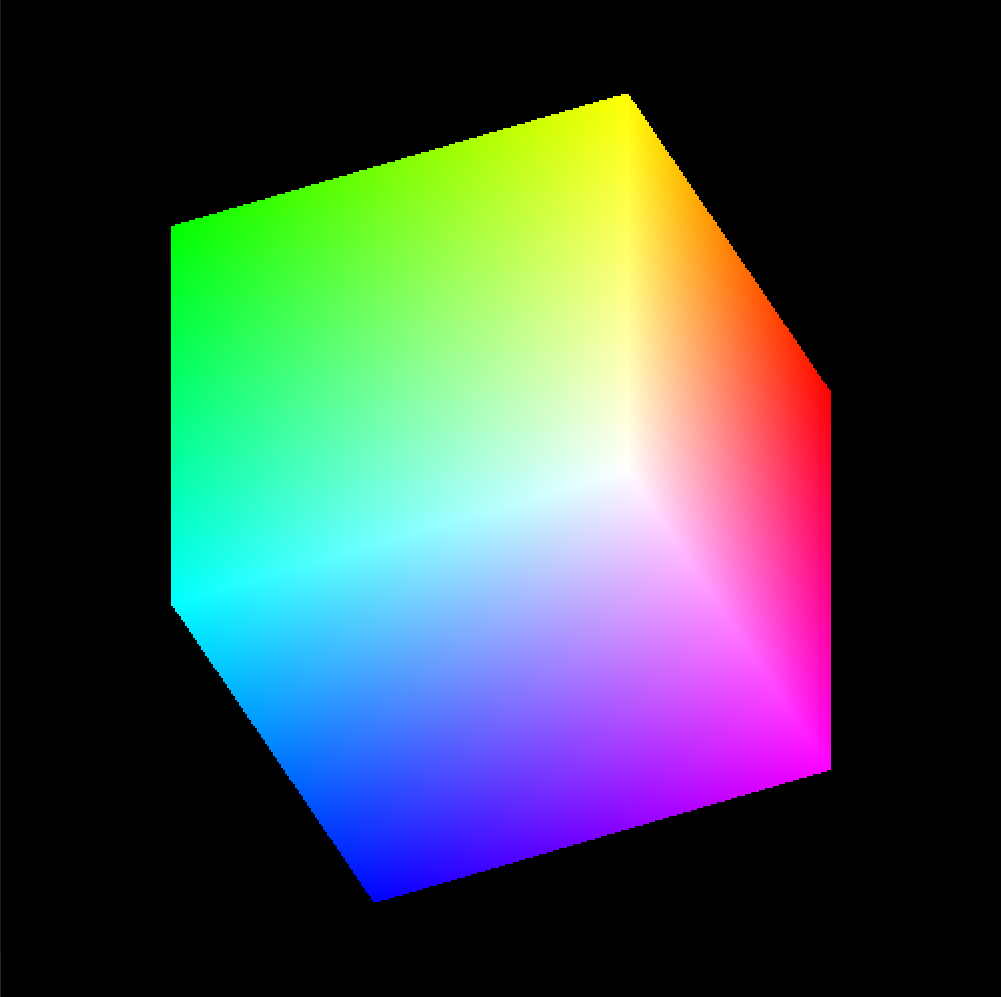
glutMouseFunc(mouse);

glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

}

**OUTPUT:**

****

**Program 4**

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

#include<stdio.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} };

GLfloatcolors[][3]={{0.0,0.0,0.0},{1.0,0.0,0.0},{1.0,1.0,0.0}, {0.0,1.0,0.0},{0.0,0.0,1.0},{1.0,0.0,1.0}, {1.0,1.0,1.0},{0.0,1.0,1.0} };

void polygon(int a, int b, int c, int d)

{

glBegin(GL\_POLYGON);

glColor3fv(colors[a]);

glNormal3fv(normals[a]);

glVertex3fv(vertices[a]);

glColor3fv(colors[b]);

glNormal3fv(normals[b]);

glVertex3fv(vertices[b]);

glColor3fv(colors[c]);

glNormal3fv(normals[c]);

glVertex3fv(vertices[c]);

glColor3fv(colors[d]);

glNormal3fv(normals[d]);

glVertex3fv(vertices[d]);

glEnd();

}

void colorcube()

{

polygon(0, 3, 2, 1);

polygon(2, 3, 7, 6);

polygon(0, 4, 7, 3);

polygon(1, 2, 6, 5);

polygon(4, 5, 6, 7);

polygon(0, 1, 5, 4);

}

static GLfloat theta[] = { 0.0,0.0,0.0 };

static GLint axis = 2;

static GLdouble viewer[] = { 0.0,0.0,5.0 };

void display(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(void)

{

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutCreateWindow("colorcube viewer");

glutReshapeFunc(myreshape);

glutDisplayFunc(display);

glutMouseFunc(mouse);

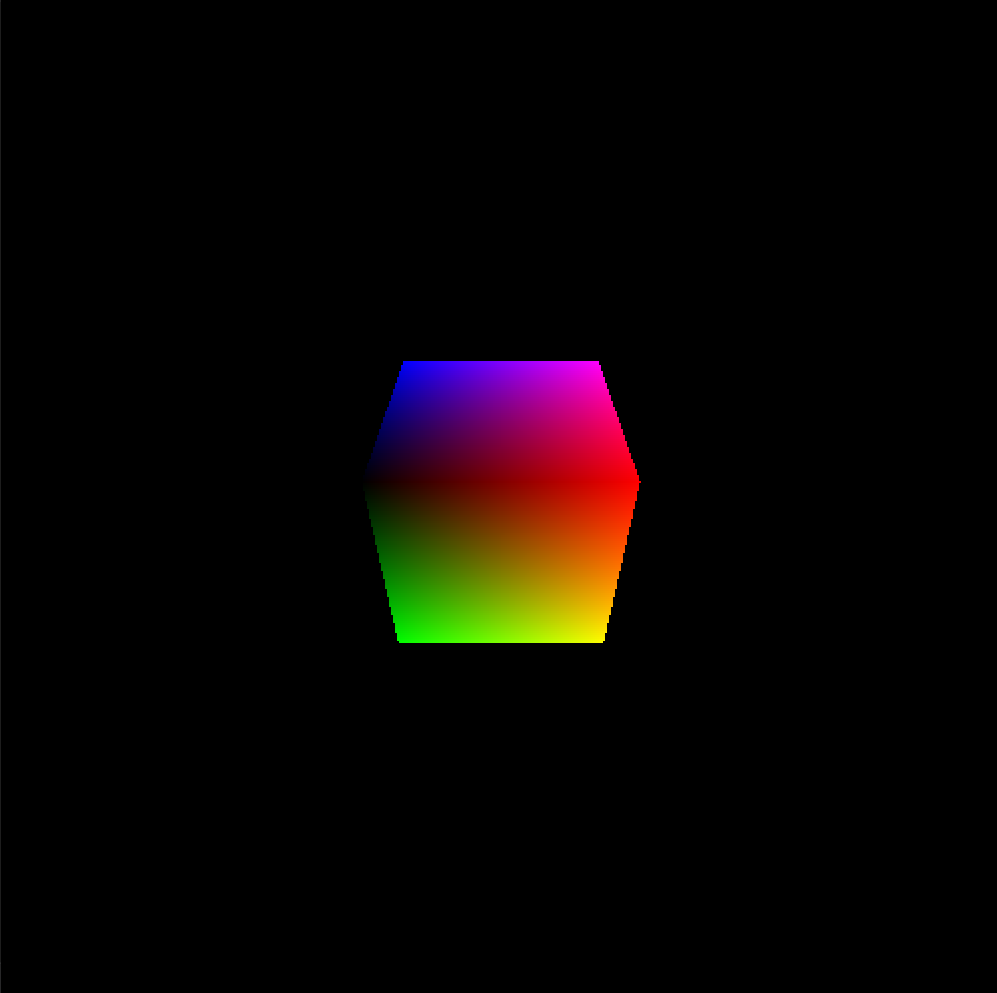
glutKeyboardFunc(keys);

glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

}

**OUTPUT:**

****

**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;

const int RIGHT = 8;

const int LEFT = 2;

const int TOP = 4;

const int BOTTOM = 1;

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;

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()

{

double x0 = 120, y0 = 10, x1 = 40, y1 = 130;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2d(x0, y0);

glVertex2d(x1, y1);

glVertex2d(60, 20);

glVertex2d(80, 120);

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);

cohensutherlandlineclipanddraw(60, 20, 80, 120);

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(void)

{

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

glutCreateWindow("Cohen Suterland Line Clipping Algorithm");

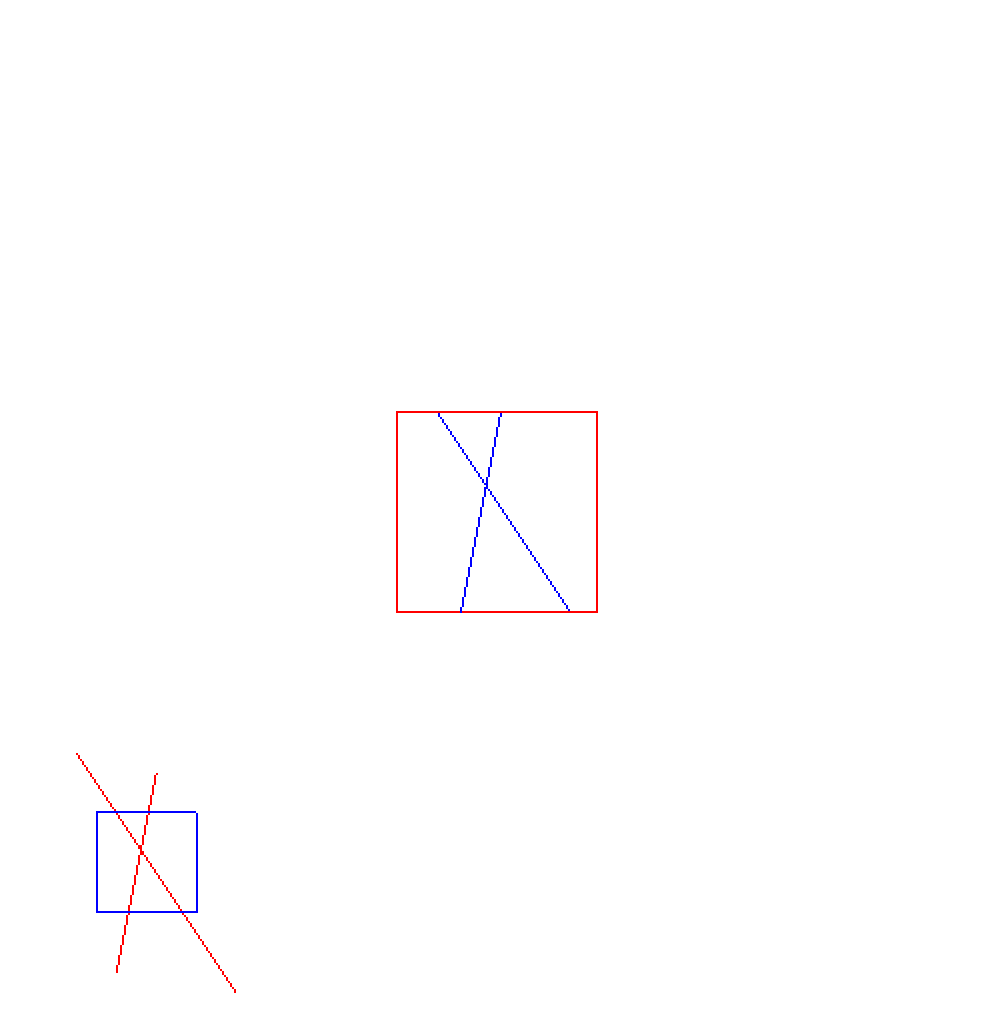
glutDisplayFunc(display);

myinit();

glutMainLoop();

}

**OUTPUT:**

****

**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>

#include <stdio.h>

#include <stdlib.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 topw, double topt, double legt, double legl)

{

glPushMatrix();

glTranslated(0, legl, 0);

glScaled(topw, topt, topw);

glutSolidCube(1.0);

glPopMatrix();

double dist = 0.95 \* topw / 2.0 - legt / 2.0;

glPushMatrix();

glTranslated(dist, 0, dist);

tableleg(legt, legl);

glTranslated(0, 0, -2 \* dist);

tableleg(legt, legl);

glTranslated(-2 \* dist, 0, 2 \* dist);

tableleg(legt, legl);

glTranslated(0, 0, -2 \* dist);

tableleg(legt, legl);

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 lightint[] = { 0.7f,0.7f,0.7f,1.0f };

GLfloat lightpos[] = { 2.0f,6.0f,3.0f,0.0f };

glLightfv(GL\_LIGHT0, GL\_POSITION, lightpos);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightint);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

double winht = 1.0;

glOrtho(-winht \* 64 / 48.0, winht \* 64 / 48.0, -winht, winht, 0.1, 100.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(2.3, 1.3, 2.0, 0.0, 0.25, 0.0, 0.0, 1.0, 0.0);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glPushMatrix();

glRotated(90.0, 0.0, 0.0, 1.0);

wall(0.02);

glPopMatrix();

wall(0.02);

glPushMatrix();

glRotated(-90.0, 1.0, 0.0, 0.0);

wall(0.02);

glPopMatrix();

glPushMatrix();

glTranslated(0.4, 0, 0.4);

table(0.6, 0.02, 0.02, 0.3);

glPopMatrix();

glPushMatrix();

glTranslated(0.6, 0.38, 0.5);

glRotated(30, 0, 1, 0);

glutSolidTeapot(0.08);

glPopMatrix();

glFlush();

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

glutCreateWindow("teapot");

glutDisplayFunc(displaysolid);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glShadeModel(GL\_SMOOTH);

glEnable(GL\_DEPTH\_TEST);

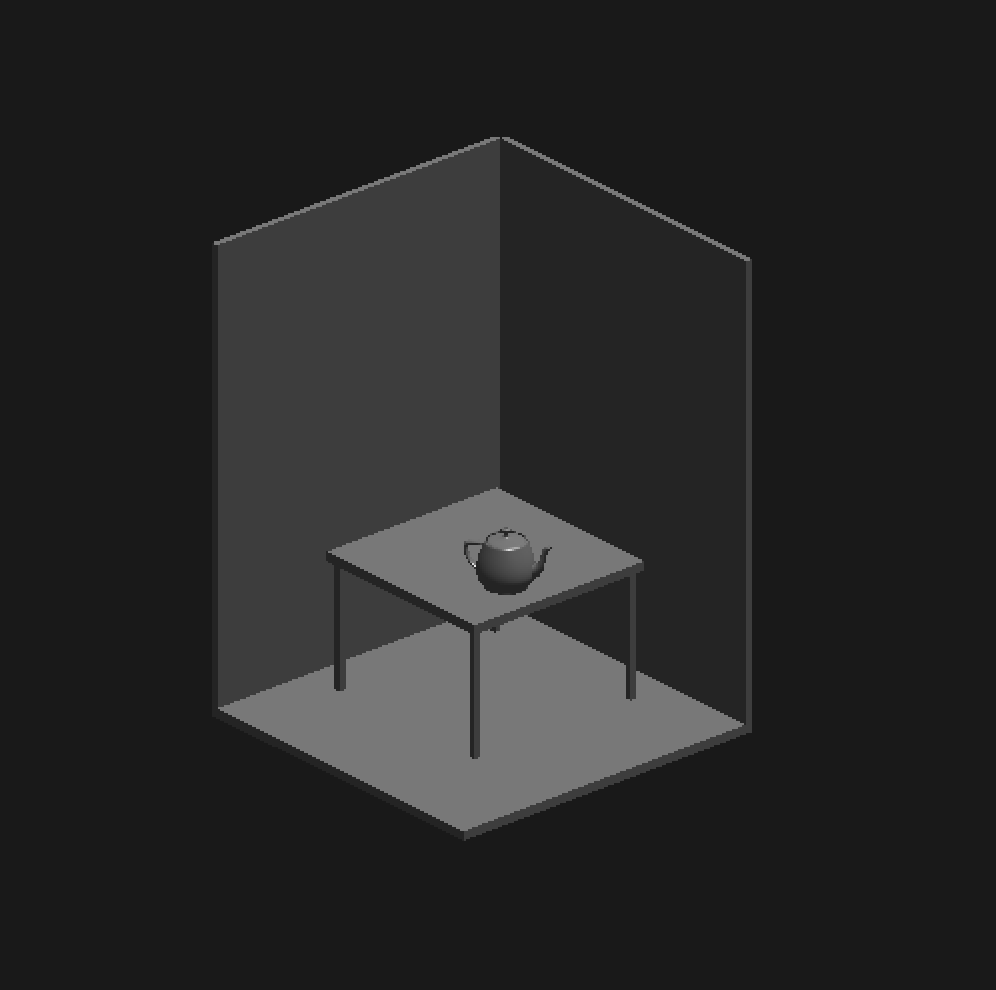
glEnable(GL\_NORMALIZE);

glClearColor(0.1, 0.1, 0.1, 0.0);

glViewport(0, 0, 640, 480);

glutMainLoop();}

**OUTPUT:**

****

**Program 7**

**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} };

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\_s("%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);

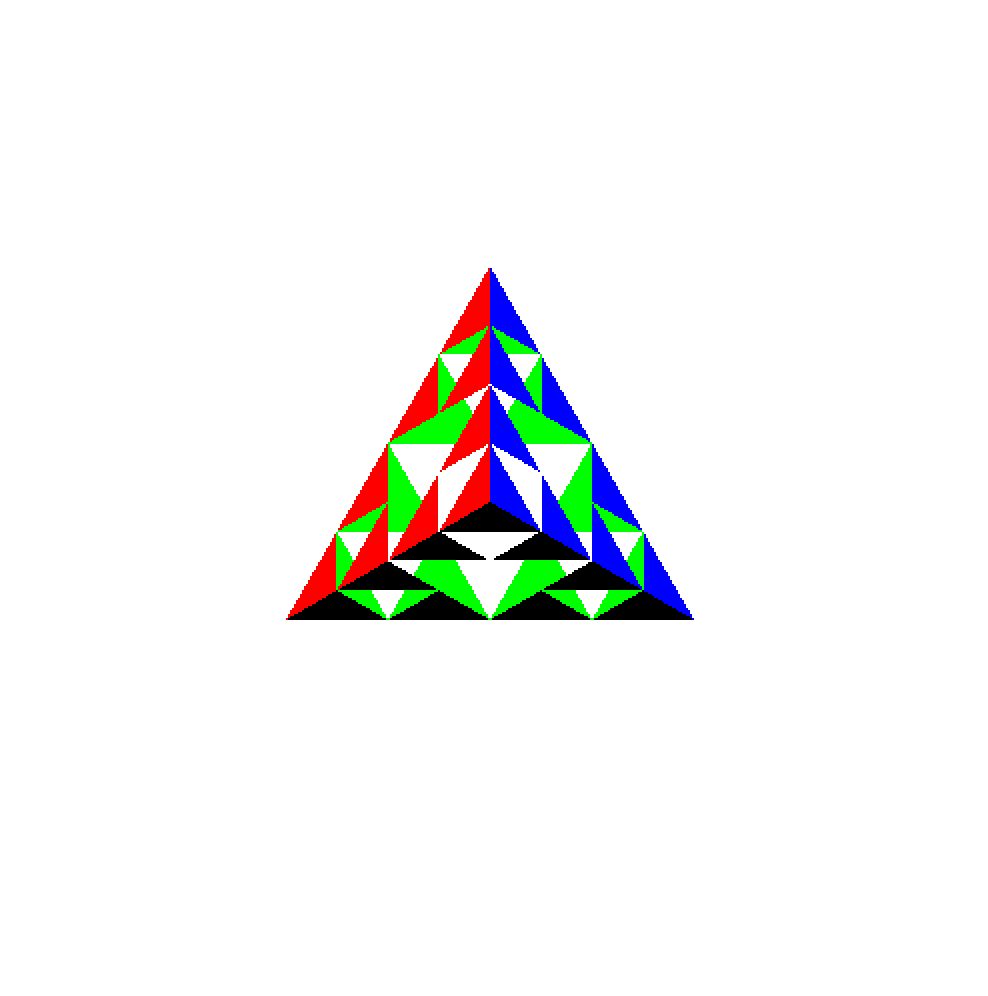
glClearColor(1.0, 1.0, 1.0, 1.0);

glutMainLoop();

}

**OUTPUT:**

No. of Divisions: **2**

****

**Program 8**

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

#include<GL/glut.h>

#include<math.h>

#include<stdio.h>

void bezierCoefficients(int n, int\* c)

{

int k, i;

for (k = 0; k <= n; k++)

{

c[k] = 1;

for (i = n; i >= k + 1; i--)

c[k] \*= i;

for (i = n - k; i >= 2; i--)

c[k] /= i;

}

}

void display()

{

int cp[4][2] = { {10,10},{100,200},{200,50},{300,300} };

int c[4], k, n = 3;

float x, y, u, blend;

bezierCoefficients(n, c);

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 0.0, 0.0);

glLineWidth(5.0);

glBegin(GL\_LINE\_STRIP);

for (u = 0; u < 1.0; u += 0.01)

{

x = 0; y = 0;

for (k = 0; k < 4; k++)

{

blend = c[k] \* pow(u, k) \* pow(1 - u, n - k);

x += cp[k][0] \* blend;

y += cp[k][1] \* blend;

}

glVertex2f(x, y);

}

glEnd();

glFlush();

}

void myinit()

{

glClearColor(1.0, 1.0, 1.0, 1.0);

glColor3f(1.0, 0.0, 0.0);

glPointSize(5.0);

gluOrtho2D(0.0, 600, 0.0, 600.0);

}

int main(void)

{

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(600, 600);

glutCreateWindow("Bezier Curve");

glutDisplayFunc(display);

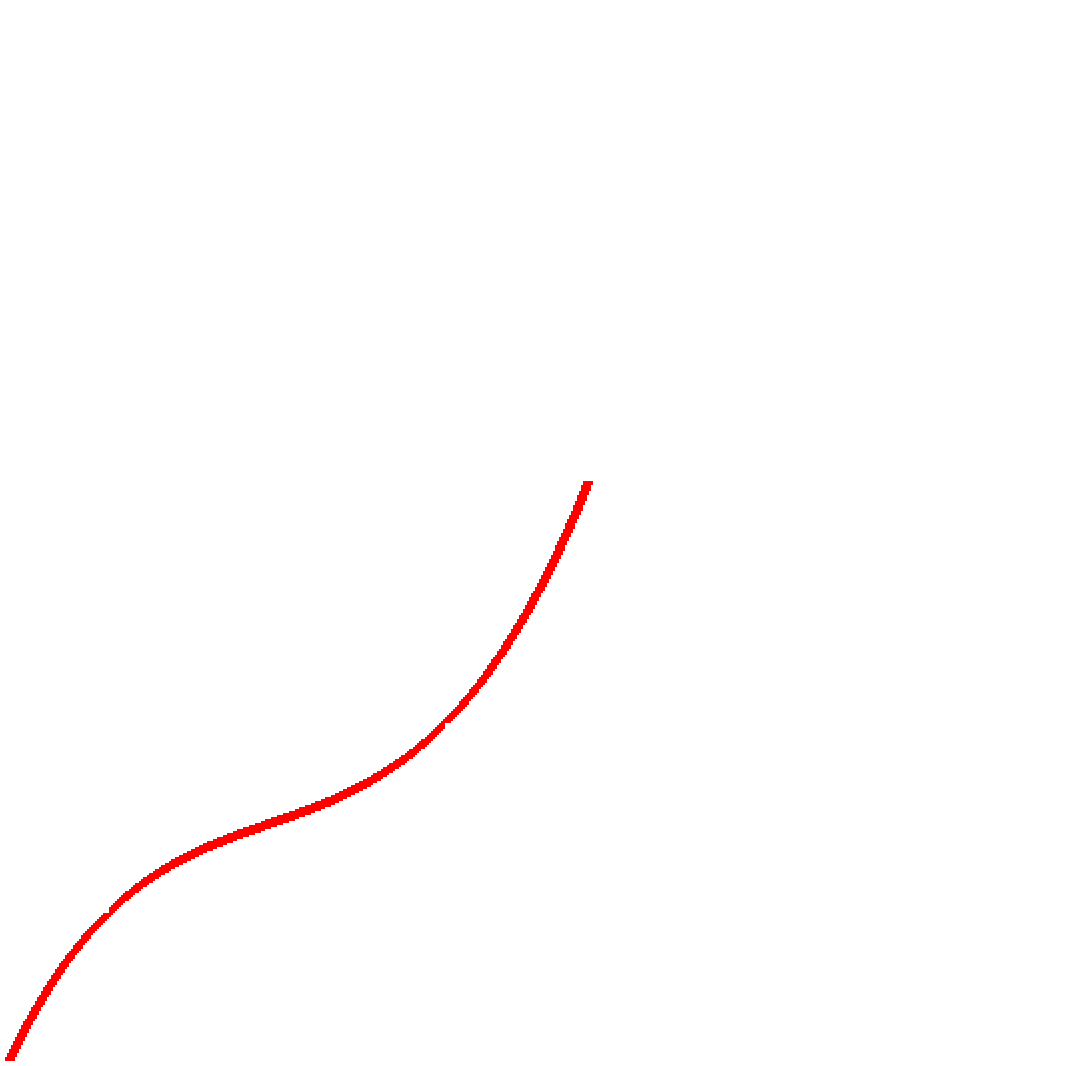
myinit();

glutMainLoop();

return 0;

}

**OUTPUT:**

****

**Program 9**

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

#include <stdlib.h>

#include <stdio.h>

#include <GL/glut.h>

float x1, x2, x3, x4, y1, y2, y3, y4;

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)

{

glColor3f(1.0, 1.0, 0.0);

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

void scanfill(float x1, float y1, float x2, float y2, float x3, float y3, float x4, float y4)

{

int le[500], re[500];

int i, y;

for (i = 0; i < 500; i++)

{

le[i] = 500;

re[i] = 0;

}

edgedetect(x1, y1, x2, y2, le, re);

edgedetect(x2, y2, x3, y3, le, re);

edgedetect(x3, y3, x4, y4, le, re);

edgedetect(x4, y4, x1, y1, le, re);

for (y = 0; y < 500; y++)

{

if (le[y] <= re[y])

for (i = (int)le[y]; i < (int)re[y]; i++)

draw\_pixel(i, y);

}

}

void display()

{

x1 = 200.0; y1 = 200.0;

x2 = 100.0; y2 = 300.0;

x3 = 200.0; y3 = 400.0;

x4 = 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);

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

glutCreateWindow("Filling a Polygon using Scan-line Algorithm");

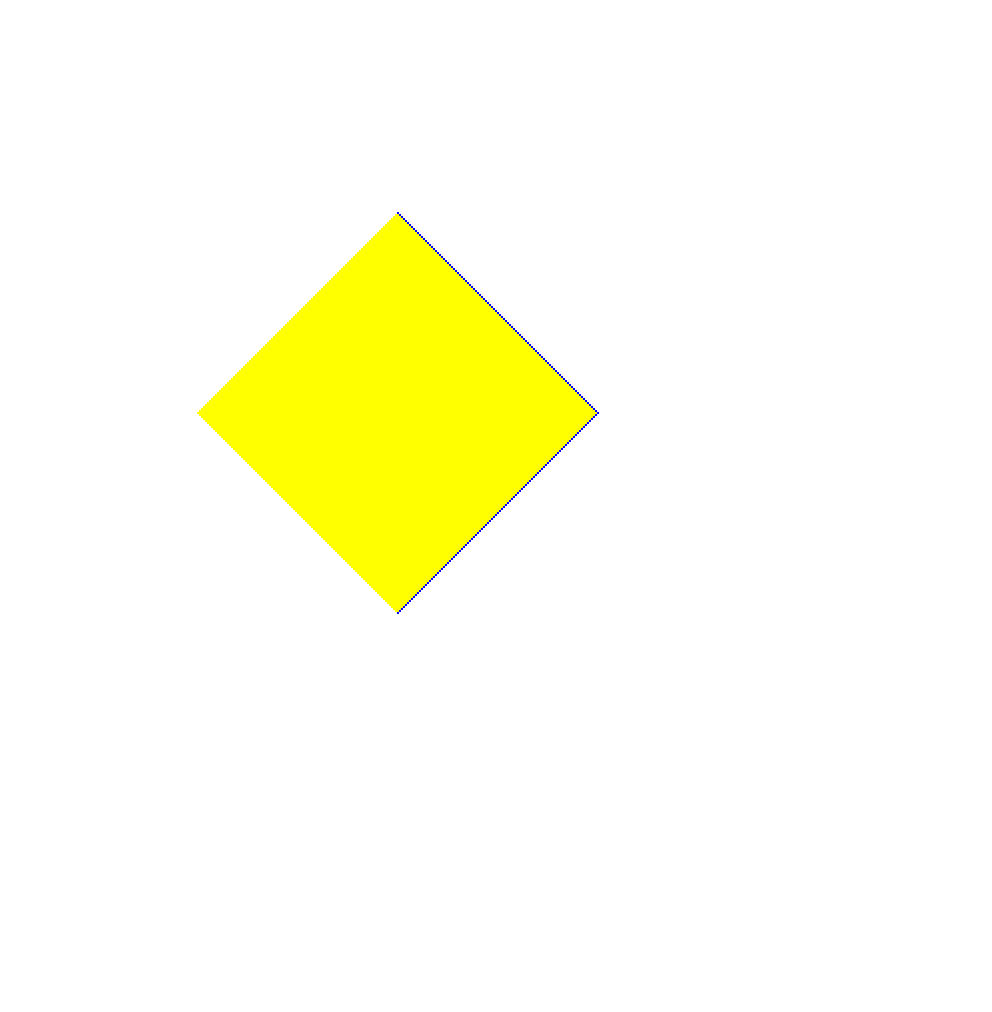
glutDisplayFunc(display);

myinit();

glutMainLoop();

}

**OUTPUT:**

****