Submitted By:

Aashutosh Sapkota

Submitted To:

Prashant Poudel

Lab	Title
1	To familiarize students with the tools to be used for implementing algorithms studied in the course
2	To draw a straight line using Digital Differential Analyzer Algorithm
3	To draw a straight line using Bresenham's Line Drawing Algorithm
4	To digitize a circle using Midpoint Circle Drawing Algorithm
5	To digitize an Ellipse using Ellipse Drawing Algorithm
6	To draw graphical output primitives using OpenGL
7	To draw Two-dimensional objects on the screen and perform two-dimensional transformations
8	To fill objects using Boundary Fill and Flood Fill approaches
9	To draw the projected view of a 3D object using perspective and parallel projection and perform three dimensional transformations to it
10	To draw a smooth curve using Bezier curve with designated control points

Lab 1: To familiarize students with the tools to be used for implementing algorithms studied in the course

Steps for setup:

1. Download Dev C++
   Link: https://sourceforge.net/projects/orwelldevcpp/
2. Download Freeglut library (Download freeglut 3.0.0 for MinGW).
   Link: https://www.transmissionzero.co.uk/software/freeglut-devel/
3. Copy Freeglut files into (Program Flies (x86)\Dev-Cpp\MinGW64\x86_64-w64-mingw32) specific folder like include and lib.

- Copy downloaded files from GL folder (freeglut/include) and paste it to include/GL folder from C: drive of opened file (Program Flies(x86)\Dev-Cpp\MinGW64\x86_64-w64-mingw32).
- Now from the downloaded package goto lib/x64 copy all files and paste it to the respective folder in C: drive (Program Flies (x86)\Dev-Cpp\MinGW64\x86_64-w64-mingw32) lib folder.

4. Copy freeglut.dll from bin/x64 of download package folder and copy it to C:\windows\system32

1. Right click on open file DDA, Goto project option.
2. Click on Parameters tab On Linker text box type:
   -lopengl32
   -lfreeglut
   -lglu32

Start Coding:

1. Header Files:
   #include<GL/glut.h>
   #include <GL/gl.h>
   #include<stdlib.h>
   #include<stdio.h>

2. Round off a floating-point number to the nearest integer

   #define ROUND(x) ((int)(x+0.5))

3. Initialization Function:

   void init(void) {

   glClearColor (0.0, 0.0, 0.0, 1.0);
   glOrtho(-100.0, 100.0, -100.0, 100.0, -1.0, 1.0);

   }

   The init function sets up the initial state of the OpenGL environment. It specifies the clear color and sets up an orthographic projection.

4. Display Function:

   void display(void) {

   // Algorithm starts here
   // ... (see next point for details)
   glutSwapBuffers();

   }

5. Main Function:

   void Main() {

   }

Lab 2: To draw a straight line using Digital Differential Analyzer Algorithm

Algorithm

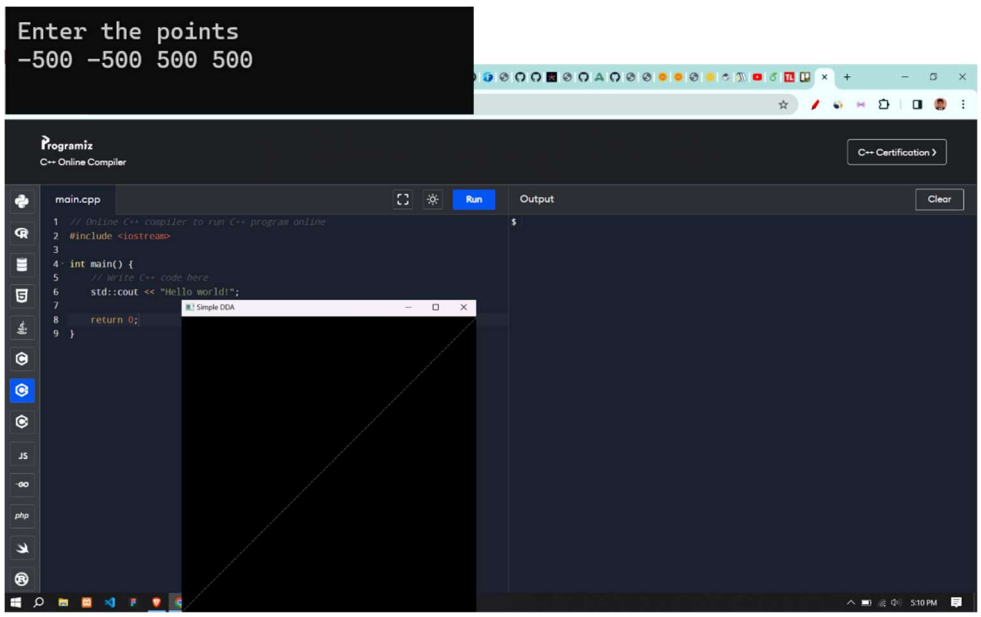
i. Input the coordinates of the two endpoints defining the line segment, denoted as (x1, y1) and (x2, y2).
ii. Compute the disparities between the x-coordinates and y-coordinates of the endpoints, assigning them as dx and dy respectively.
iii. Determine the slope of the line, expressed as m = dy/dx.
iv. Establish the starting point of the line at (x1, y1).
v. Iterate through the x-coordinates of the line, incrementing by one during each iteration. Calculate the corresponding y-coordinate using the formula y = y1 + m (x – x1).
vi. Plot the pixel at the computed (x, y) coordinate.
vii. Repeat steps (v) and (vi) until the endpoint (x2, y2) is reached.

Program

```
#include<GL/glut.h>
#include <GL/gl.h>
#include<stdlib.h>
#include<stdio.h>
#define ROUND(x) ((int)(x+0.5))
int xa,xb,ya,yb;
void init(void){
	glClearColor (0.0, 0.0, 0.0, 1.0);
	glOrtho(-100.0, 100.0, -100.0, 100.0, -1.0, 1.0);
}
void display (void){
	//DDA Algorithm starts here
	int dx = xb-xa,dy = yb-ya,steps,k;
	float xIncrement,yIncrement, x = xa, y = ya;
	if(abs(dx)>abs(dy))
	steps = abs(dx);
	else steps = abs(dy);
	xIncrement = dx/(float)steps;
	yIncrement = dy/(float)steps;
	glClear (GL_COLOR_BUFFER_BIT);
	glColor3f (1.0, 1.0, 1.0);
	glBegin(GL_POINTS);
	glVertex2s(ROUND(x),ROUND(y));
```

```
        for(k=0;k<steps;k++){
                x = x+xIncrement;
                y = y+yIncrement;
                glVertex2s(ROUND(x),ROUND(y));
        }
        glEnd();
        glutSwapBuffers();
}
int main(int argc, char** argv){
        printf("Enter the points\n");
        scanf("%d %d %d %d",&xa,&ya,&xb,&yb);
        glutInit(&argc, argv);
        glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowSize (500, 500);
        glutInitWindowPosition (100, 100);
        glutCreateWindow ("Simple DDA ");
        init ();
        glutDisplayFunc(display);
        glutMainLoop();
        return 0;
}
```

Output:



Lab 3: To draw a straight line using Bresenham's Line Drawing Algorithm

Algorithm

i. Input the coordinates of the two endpoints that define the line segment, denoted as (x1, y1) and (x2, y2).
ii. Calculate the differences between the x-coordinates and y-coordinates of the endpoints, represented as dx and dy.
iii. Compute the decision parameter, often referred to as P, using the initial coordinates (x1, y1): P = 2 * (dy - dx).
iv. Set the initial point of the line as (x1, y1).
v. Iterate through the x-coordinates of the line, incrementing by one during each iteration.
vi. At each iteration, determine whether to increment the y-coordinate or not based on the decision parameter. If (P < 0), update (P) and the pixel coordinates (x, y):
Set P = P + 2dy, set y = y + 1
Else only update P : P = P + 2dy - 2dx
vii. Plot the pixel at the calculated (x, y) coordinate.
viii. Repeat steps (v)-(vii) until reaching the endpoint (x2, y2).

Program

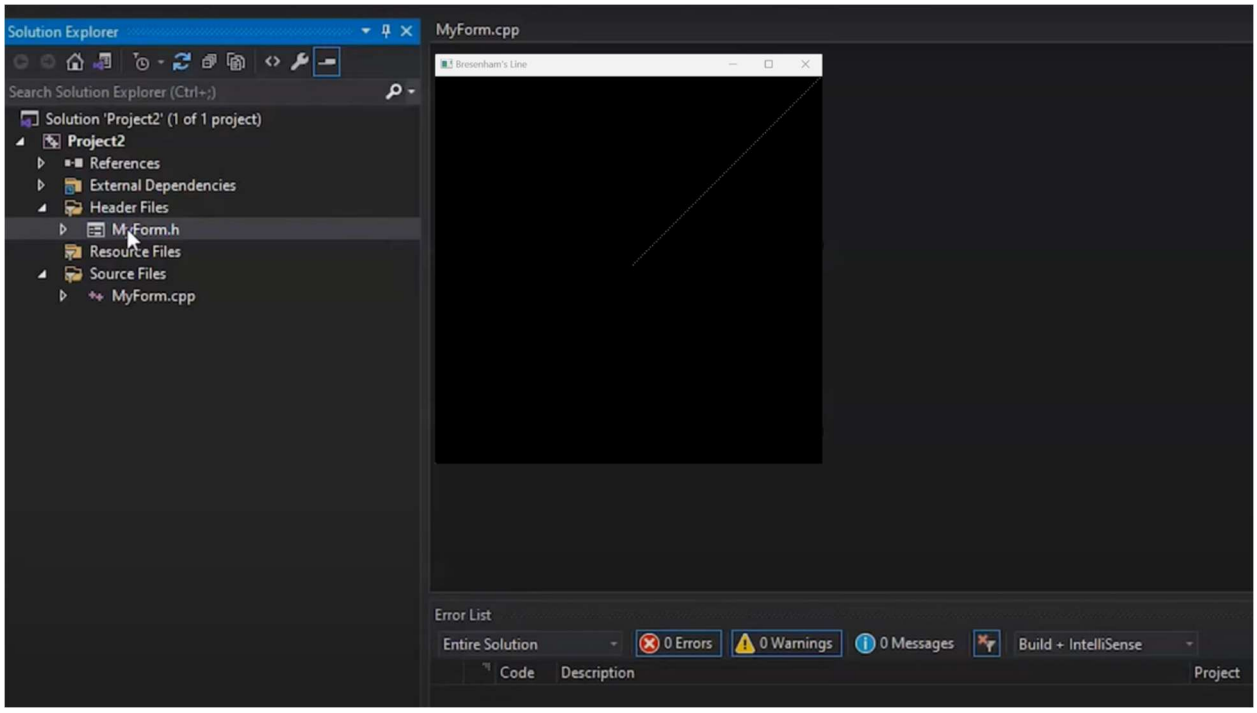
```
#include <GL/glut.h>
#include <GL/gl.h>
#include <stdlib.h>
#include <stdio.h>
int xa, xb, ya, yb;
void init(void) {
	glClearColor(0.0, 0.0, 0.0, 1.0);
	glOrtho(-100.0, 100.0, -100.0, 100.0, -1.0, 1.0);
}
void display(void) {
	// Bresenham's Algorithm starts here
	int dx = xb - xa, dy = yb - ya, p, x, y, const1, const2;
	const1 = 2 * dy;
	const2 = 2 * (dy - dx);
	if (dx > 0) {
		x = xa;
		y = ya;
	} else {
```

```
        x = xb;
        y = yb;
        xb = xa;
        yb = ya;
}
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 1.0, 1.0);
glBegin(GL_POINTS);
glVertex2s(x, y);
if (dx > dy) {
        p = 2 * dy - dx;
        for (int i = 0; i < dx; i++) {
        x++;
        if (p < 0) {
                p += const1;
        } else {
                y++;
                p += const2;
        }
        glVertex2s(x, y);
        }
} else {
        p = 2 * dx - dy;
        for (int i = 0; i < dy; i++) {
                y++;
                if (p < 0) {
                        p += const1;
                } else {
                        x++;
                        p += const2;
                }
                glVertex2s(x, y);
        }
}
glEnd();
```

```
        glutSwapBuffers();
}
int main(int argc, char** argv) {
        printf("Enter the points\n");
        scanf("%d %d %d %d", &xa, &ya, &xb, &yb);
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(100, 100);
        glutCreateWindow("Bresenham's Line");
        init();
        glutDisplayFunc(display);
        glutMainLoop();
        return 0;
}
```

Output :

```
Enter the points
2 3 200 300
```



Lab 4: To digitize a circle using Midpoint Circle Drawing Algorithm

Algorithm

i. **Initialization:** Start with the center coordinates (x_c, y_c) of the circle and its radius (r).
   Set two variables, x and y, to the initial values (r, 0).
   These represent the coordinates of the first point on the circle in the first octant ($0 \leq x \leq \pi/2$, $0 \leq y \leq \pi/2$).

ii. **Decision based on error term:** Calculate an error term, p, that represents the distance between the current point (x, y) and the ideal point on the circle. This can be done using the formula: p = 1 - r^2 + 2x + 3y

iii. **Plot the point and update coordinates:** Based on the value of p, determine the next point on the circle:
   - If $p \leq 0$, the current point is already on or inside the circle, so plot it and simply increment x: x = x + 1
   - If $p > 0$, the current point is outside the circle, so plot the next point diagonally and update both x and y: x = x - 1 y = y + 1 p = p + 2(x - y) + 5

iv. **Repeat:** Repeat steps (ii) and (iii) until $x \geq y$. This will generate all the points on the circle in the first octant.

v. **Symmetry:** Since a circle is symmetrical, you can obtain the points in all other octants by reflecting the points in the first octant across the axes.

Program

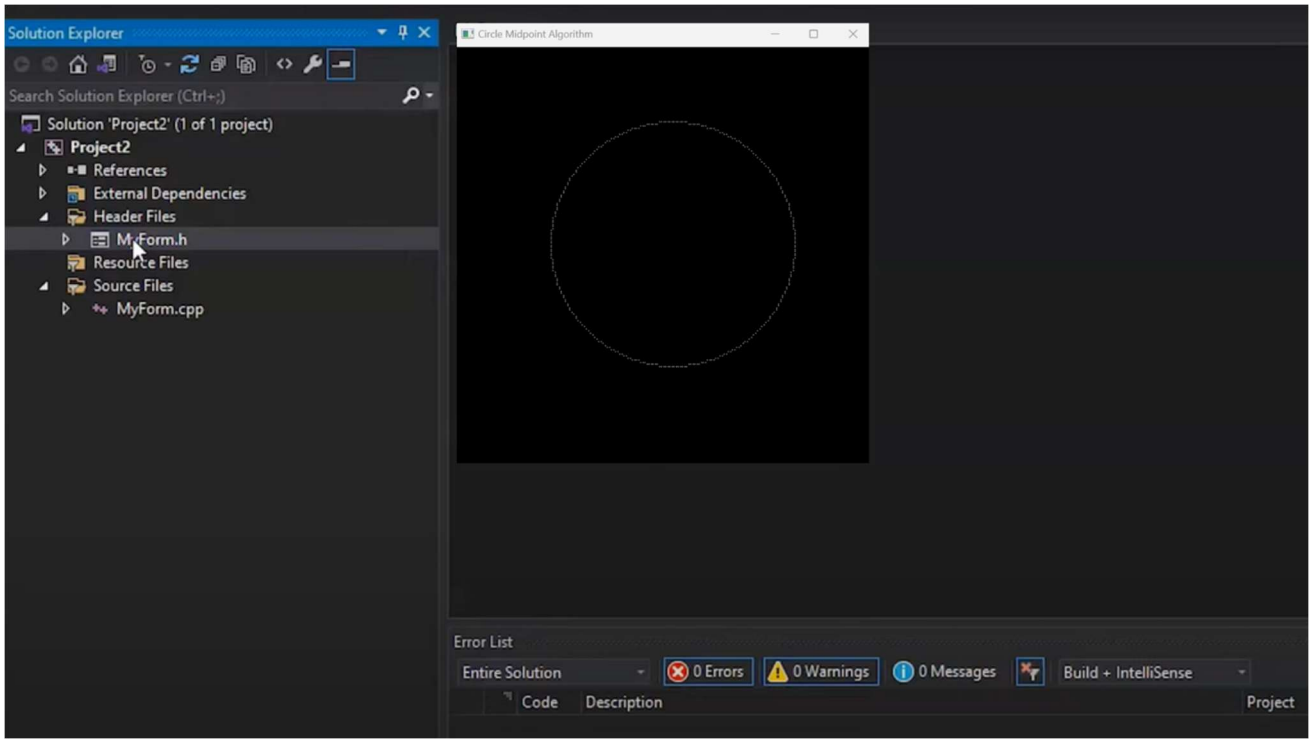
```
#include <GL/glut.h>
#include <GL/gl.h>
#include <stdlib.h>
#include <stdio.h>
#define ROUND(x) ((int)(x + 0.5))
int xc, yc, r; // Center coordinates and radius
void init(void) {
        glClearColor(0.0, 0.0, 0.0, 1.0);
        glOrtho(-100.0, 100.0, -100.0, 100.0, -1.0, 1.0);
}
void display(void) {
        int x = 0, y = r;
        int p = 1 - r; // Decision parameter
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0, 1.0, 1.0);
        glBegin(GL_POINTS);
        // Plot initial points
```

```
        glVertex2s(ROUND(xc + x), ROUND(yc + y));
        glVertex2s(ROUND(xc - x), ROUND(yc + y));
        glVertex2s(ROUND(xc + x), ROUND(yc - y));
        glVertex2s(ROUND(xc - x), ROUND(yc - y));
        glVertex2s(ROUND(xc + y), ROUND(yc + x));
        glVertex2s(ROUND(xc - y), ROUND(yc + x));
        glVertex2s(ROUND(xc + y), ROUND(yc - x));
        glVertex2s(ROUND(xc - y), ROUND(yc - x));
        while (x < y) {
                x++;
                if (p < 0) {
                        p = p + 2 * x + 1;
                } else {
                        y--;
                        p = p + 2 * (x - y) + 1;
                }
                // Plot octants using symmetry
                glVertex2s(ROUND(xc + x), ROUND(yc + y));
                glVertex2s(ROUND(xc - x), ROUND(yc + y));
                glVertex2s(ROUND(xc + x), ROUND(yc - y));
                glVertex2s(ROUND(xc - x), ROUND(yc - y));
                glVertex2s(ROUND(xc + y), ROUND(yc + x));
                glVertex2s(ROUND(xc - y), ROUND(yc + x));
                glVertex2s(ROUND(xc + y), ROUND(yc - x));
                glVertex2s(ROUND(xc - y), ROUND(yc - x));
        }
        glEnd();
        glutSwapBuffers();
}
int main(int argc, char** argv) {
        printf("Enter center coordinates and radius: ");
        scanf("%d %d %d", &xc, &yc, &r);
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowSize(500, 500);
```

```
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Circle Midpoint Algorithm");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Output :

```
Enter center coordinates and radius: 4 5 59
```



Lab 5: To digitize an Ellipse using Ellipse Drawing Algorithm

Algorithm

i. **Initialization:** Define the ellipse's center coordinates (Cx, Cy), major radius (a), and minor radius (b).
Set initial variables:
x = a (representing x-coordinate), y = 0 (representing y-coordinate)
d1 = b^2 + (a^2 - b^2) * x (decision parameter1), d2 = b^2 * (2x + 1) (decision parameter2)

ii. **Iterative calculations:** While x > 0:

Calculate new decision parameters:

d1 = d1 - 2xy - 3y^2 + b^2 (region 1), d2 = d2 - 4x (region 2)

Based on the parameters, plot the next point:

- If d1 ≤ 0 and d2 ≤ 0 (inside or on ellipse):

  Plot (x, y) and update y: y = y + 1
- If d1 > 0 (outside ellipse in region 1):

  Plot (x, y) and update both x and y: x = x - 1, y = y + 1
- If d2 > 0 (outside ellipse in region 2):

  Plot (x - 1, y)

iii. **Symmetry and completion:** Use the four-way symmetry of the ellipse to plot corresponding points in other quadrants based on the first quadrant points:
(-x, y) for second quadrant
(x, -y) for third quadrant
(-x, -y) for fourth quadrant

Repeat steps (ii) and (iii) until x = 0, generating all points on the ellipse.

Program

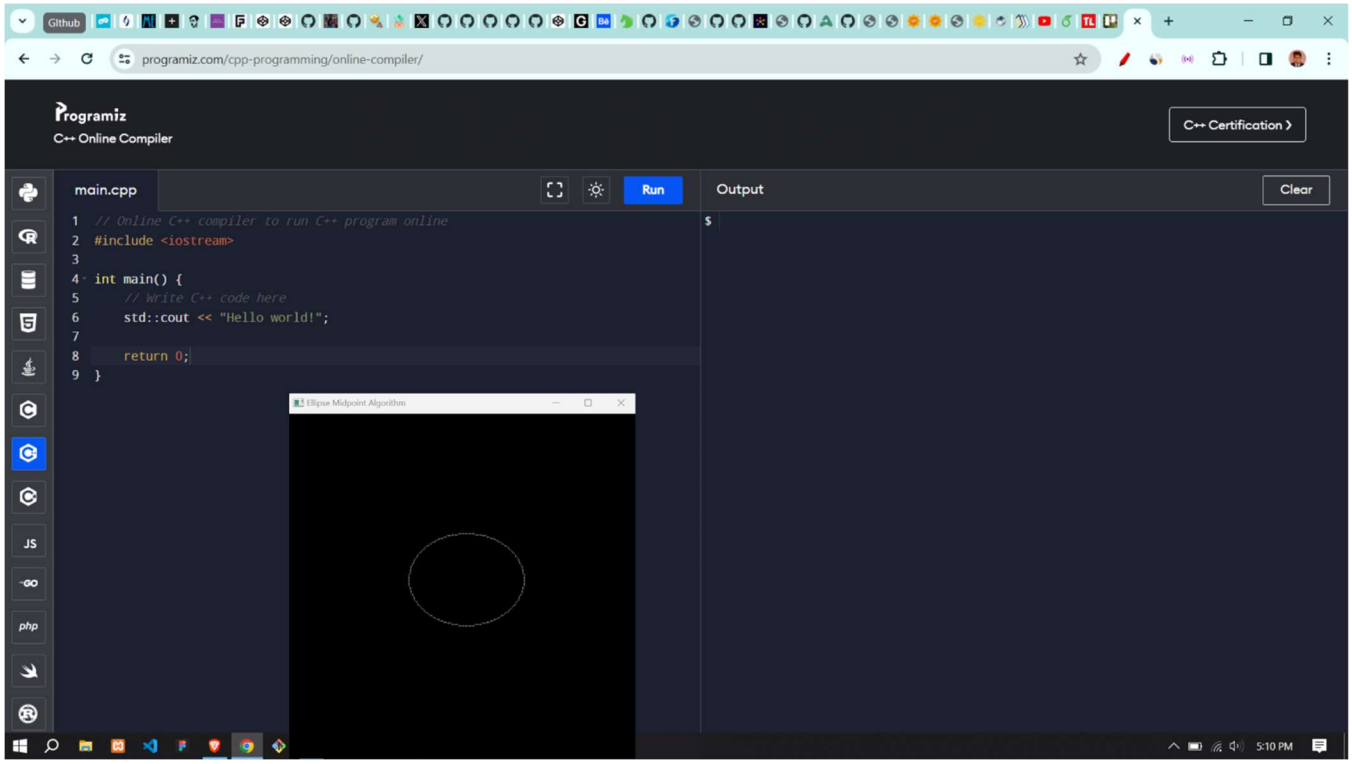
```
#include <GL/glut.h>
#include <GL/gl.h>
#include <stdlib.h>
#include <stdio.h>
#define ROUND(x) ((int)(x + 0.5))
int xc, yc, rx, ry; // Center coordinates and radii along x and y axes
```

```
void init(void) {
	glClearColor(0.0, 0.0, 0.0, 1.0);
	glOrtho(-150.0, 150.0, -150.0, 150.0, -1.0, 1.0);
}
void drawEllipsePoints(int x, int y) {
	glVertex2s(ROUND(xc + x), ROUND(yc + y));
	glVertex2s(ROUND(xc - x), ROUND(yc + y));
	glVertex2s(ROUND(xc + x), ROUND(yc - y));
	glVertex2s(ROUND(xc - x), ROUND(yc - y));
}
void display(void) {
	int x = 0, y = ry;
	int rxSq = rx * rx;
	int rySq = ry * ry;
	int twoRxSq = 2 * rxSq;
	int twoRySq = 2 * rySq;
	int px = 0, py = twoRxSq * y;
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(1.0, 1.0, 1.0);
	glBegin(GL_POINTS);
	// Initial points
	drawEllipsePoints(x, y);
	// Region 1
	int p1 = rySq - rxSq * ry + 0.25 * rxSq;
	while (px < py) {
		x++;
		px += twoRySq;
		if (p1 < 0)
			p1 += rySq + px;
		else {
			y--;
			py -= twoRxSq;
			p1 += rySq + px - py;
		}
		drawEllipsePoints(x, y);
```

```
        }
        // Region 2
        int p2 = rySq * (x + 0.5) * (x + 0.5) + rxSq * (y - 1) * (y - 1) -
        rxSq * rySq;
        while (y > 0) {
                y - -;
                py -= twoRxSq;
                if (p2 > 0)
                        p2 += rxSq - py;
                else {
                        x++;
                        px += twoRySq;
                        p2 += rxSq - py + px;
                }
                drawEllipsePoints(x, y);
        }
        glEnd();
        glutSwapBuffers();
}
int main(int argc, char **argv) {
        printf("Enter center coordinates and radii along x and y axes: ");
        scanf("%d %d %d %d", &xc, &yc, &rx, &ry);
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(100, 100);
        glutCreateWindow("Ellipse Midpoint Algorithm");
        init();
        glutDisplayFunc(display);
        glutMainLoop();
        return 0;
        }
```

Output :

```
Enter center coordinates and radii along x and y axes: 3 7 50 40
```



Lab 6: To draw graphical output primitives using OpenGL

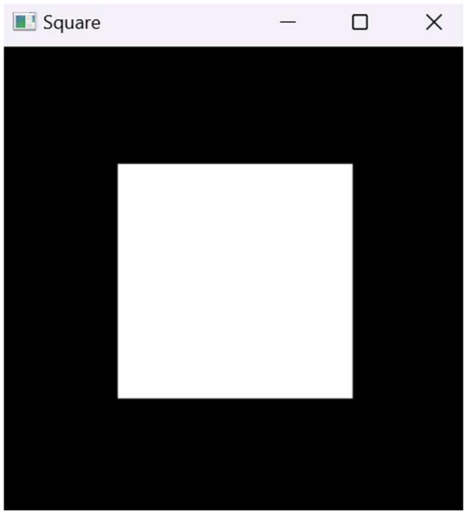
Algorithm

i. **Initialize OpenGL:** Set up the OpenGL environment and create a window.
ii. **Define Vertices:** Define the coordinates of the vertices.
iii. **Draw Object:** Use OpenGL functions to draw the object using the defined vertices.

Program: To draw square

```
#include <GL/glut.h>
void display() {
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_QUADS);
	glVertex2f(-0.5, -0.5);
	glVertex2f(0.5, -0.5);
	glVertex2f(0.5, 0.5);
	glVertex2f(-0.5, 0.5);
	glEnd();
	glFlush();
}
int main(int argc, char** argv) {
	glutInit(&argc, argv);
	glutCreateWindow("Square");
	glutDisplayFunc(display);
	glutMainLoop();
	return 0;
}
```

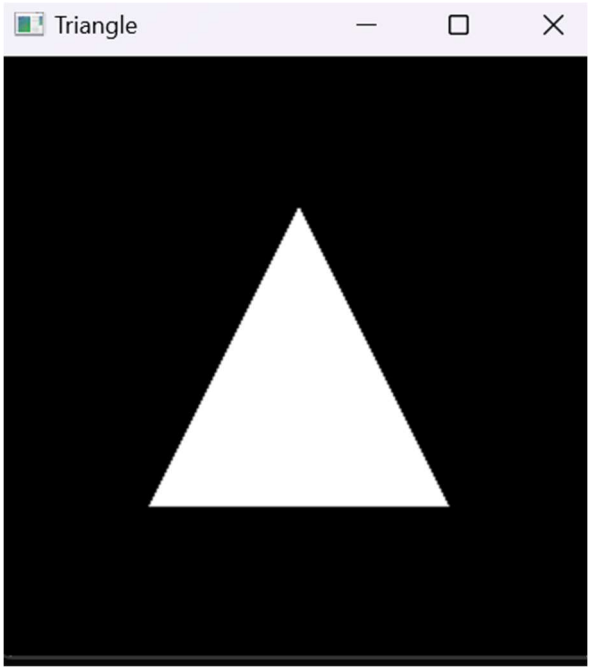
Output



Program: To draw triangle

```
#include <GL/glut.h>
void display() {
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_TRIANGLES);
        glVertex2f(-0.5, -0.5);
        glVertex2f(0.5, -0.5);
        glVertex2f(0.0, 0.5);
        glEnd();
        glFlush();
}
int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutCreateWindow("Triangle");
        glutDisplayFunc(display);
        glutMainLoop();
        return 0;
}
```

Output



Lab 7: To draw Two-dimensional objects on the screen and perform two-dimensional transformations

Algorithm

i. **Initialization:**
   Set initial value of translateX = 0.0 and translateY = 0.0
   Set initial rotation angle = 0.0
   Set initial value of scaleX = 1.0 and scaleY = 1.0

ii. **Define an object:**
   Define the coordinates of the vertices.

iii. **Define Transformation function:**

   glTranslatef(translateX, translateY, 0.0)

   glRotatef(angle, 0.0, 0.0, 1.0)

   glScalef(scaleX, scaleY, 1.0)

iv. **Define keyboard Function:**

   Switch on the pressed key:

   - 'w': Increment translateY for translation up.
   - 's': Decrement translateY for translation down.
   - 'a': Decrement translateX for translation left.
   - 'd': Increment translateX for translation right.
   - 'q': Increment angle for rotation counter clockwise.
   - 'e': Decrement angle for rotation clockwise.
   - 'z': Decrement scaleX for scaling up.
   - 'x': Increment scaleY for scaling down.
   - 27: Exit if ESC key is pressed.

   Request a redraw using glutPostRedisplay()

Program

```
#include <GL/glut.h>
float translateX = 0.0;
float translateY = 0.0;
float rotateAngle = 0.0;
float scaleX = 1.0;
float scaleY = 1.0;
void drawRectangle() {
    glBegin(GL_QUADS);
```

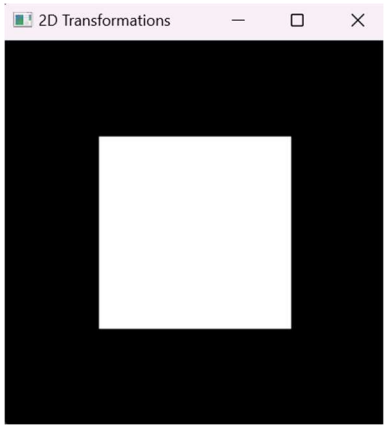
```
        glVertex2f(-0.5, -0.5);
        glVertex2f(0.5, -0.5);
        glVertex2f(0.5, 0.5);
        glVertex2f(-0.5, 0.5);
        glEnd();
}
void display() {
        glClear(GL_COLOR_BUFFER_BIT);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        // Apply transformations
        glTranslatef(translateX, translateY, 0.0);
        glRotatef(rotateAngle, 0.0, 0.0, 1.0);
        glScalef(scaleX, scaleY, 1.0);
        // Draw the rectangle
        drawRectangle();
        glFlush();
}
void keyboard(unsigned char key, int x, int y) {
        switch (key) {
                case 'w':
                translateY += 0.1;
                break;
                case 's':
                translateY -= 0.1;
                break;
                case 'a':
                translateX -= 0.1;
                break;
                case 'd':
                translateX += 0.1;
                break;
                case 'q':
                rotateAngle += 5.0;
                break;
```

```
                case 'e':
                rotateAngle -= 5.0;
                break;
                case 'z':
                scaleX *= 1.5;
                scaleY *= 1.5;
                break;
                case 'x':
                scaleX /= 1.5;
                scaleY /= 1.5;
                break;
                case 27: // ASCII code for 'Esc' key
                exit(0);
        }
        glutPostRedisplay();
}
int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutCreateWindow("2D Transformations");
        glutDisplayFunc(display);
        glutKeyboardFunc(keyboard);
        glutMainLoop();
        return 0;
}
```

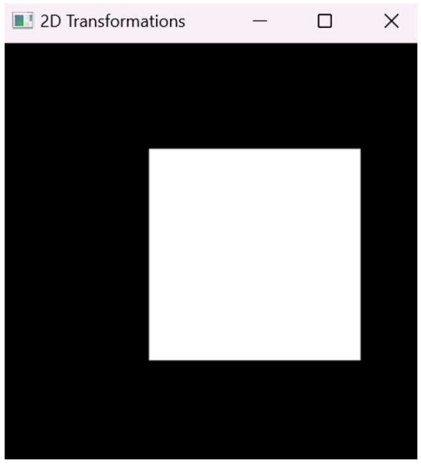
transltion by 0.2

Output

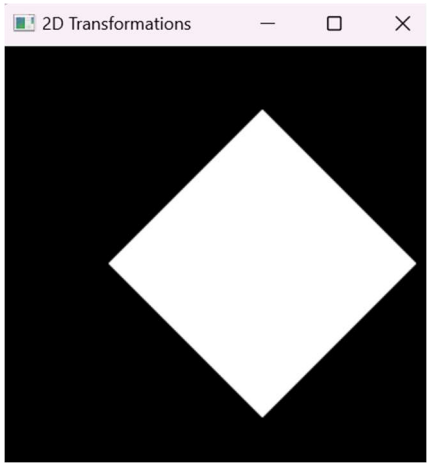
i. Original object



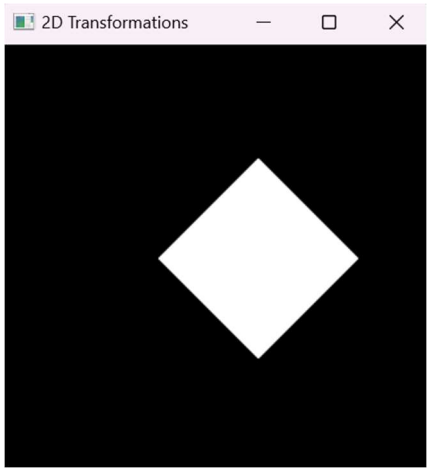
ii. Translation by 0.2 in positive X-axis



iii. Rotation by an angle of 45^0 in counter clockwise direction



iv. Scaling down by 1.5



Lab 8: To fill objects using Boundary Fill and Flood Fill approaches

Boundary Fill Algorithm

i. Initialize Parameters:
   - Set the fill color.
   - Set the boundary color (color of the object).
   - Choose a seed point (x, y) inside the object.

ii. Boundary Fill Function:

```
function boundaryFill(x, y, fillColor, borderColor):
    if (getPixelColor(x, y) is not borderColor and getPixelColor(x, y) is not fillColor):
        setPixelColor(x, y, fillColor)
        boundaryFill(x + 1, y, fillColor, borderColor)  // Fill right
        boundaryFill(x - 1, y, fillColor, borderColor)  // Fill left
        boundaryFill(x, y + 1, fillColor, borderColor)  // Fill up
        boundaryFill(x, y - 1, fillColor, borderColor)  // Fill down
```

iii. Call Boundary Fill:

Call the boundaryFill function with the seed point.

Flood Fill Algorithm:

i. Initialize Parameters:

Set the fill color.

Set the boundary color (color of the object).

Choose a seed point (x, y) inside the object.

ii. Flood Fill Function:

```
function floodFill(x, y, fillColor, borderColor):
    if (getPixelColor(x, y) is not borderColor and getPixelColor(x, y) is not fillColor):
        setPixelColor(x, y, fillColor)
        floodFill(x + 1, y, fillColor, borderColor)  // Fill right
        floodFill(x, y + 1, fillColor, borderColor)  // Fill up
        floodFill(x, y - 1, fillColor, borderColor)  // Fill down
```

iii. Call Flood Fill:
Call the floodFill function with the seed point.

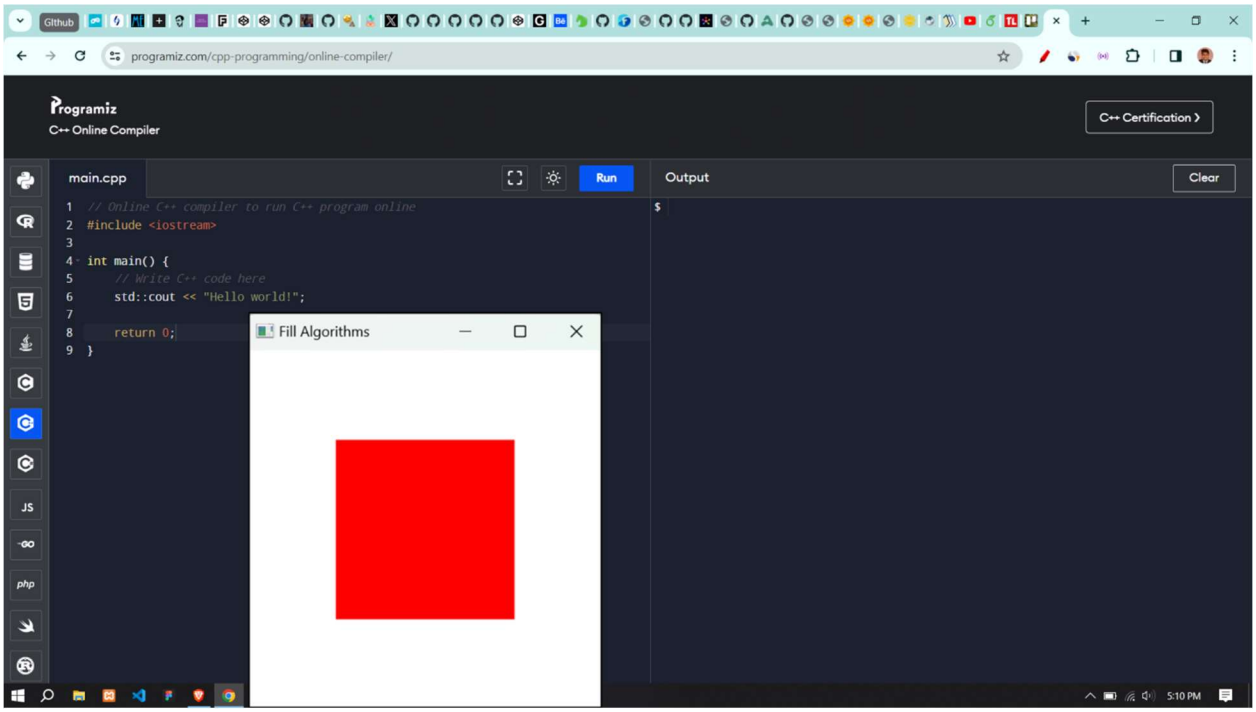
Program

```
#include <GL/glut.h>
void drawSquare(float x, float y, float size) {
	glBegin(GL_QUADS);
	glVertex2f(x, y);
	glVertex2f(x + size, y);
	glVertex2f(x + size, y + size);
	glVertex2f(x, y + size);
	glEnd();
}
void boundaryFill(int x, int y, float fillColor[3], float borderColor[3]) {
	float currentColor[3];
	glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, currentColor);
	if (!(currentColor[0] == borderColor[0] && currentColor[1] == borderColor[1] &&
	currentColor[2] == borderColor[2]) &&
	!(currentColor[0] == fillColor[0] && currentColor[1] == fillColor[1] && currentColor[2]
	== fillColor[2])) {
		glColor3fv(fillColor);
		glBegin(GL_POINTS);
		glVertex2i(x, y);
		glEnd();
		glFlush();
		boundaryFill(x + 1, y, fillColor, borderColor);
		boundaryFill(x - 1, y, fillColor, borderColor);
		boundaryFill(x, y + 1, fillColor, borderColor);
		boundaryFill(x, y - 1, fillColor, borderColor);
	}
}
void floodFill(int x, int y, float fillColor[3], float targetColor[3]) {
	float currentColor[3];
	glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, currentColor);
	if (currentColor[0] == targetColor[0] && currentColor[1] == targetColor[1] &&
	currentColor[2] == targetColor[2]) {
		glColor3fv(fillColor);
```

```
                glBegin(GL_POINTS);
                glVertex2i(x, y);
                glEnd();
                glFlush();
                floodFill(x + 1, y, fillColor, targetColor);
                floodFill(x - 1, y, fillColor, targetColor);
                floodFill(x, y + 1, fillColor, targetColor);
                floodFill(x, y - 1, fillColor, targetColor);
        }
}
void display() {
        glClear(GL_COLOR_BUFFER_BIT);
        float borderColor[] = {1.0, 0.0, 0.0}; // Red color for border
        float fillColor[] = {0.0, 1.0, 0.0}; // Green color for fill
        glColor3fv(borderColor);
        drawSquare(100, 100, 200);
        // Boundary Fill
        boundaryFill(150, 150, fillColor, borderColor);
        // Flood Fill
        floodFill(250, 250, fillColor, borderColor);
        glFlush();
}
void init() {
        glClearColor(1.0, 1.0, 1.0, 1.0);
        gluOrtho2D(0, 400, 0, 400);
}
int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutCreateWindow("Fill Algorithms");
        init();
        glutDisplayFunc(display);
        glutMainLoop();
```

```
    return 0;
}
```

Output



Lab 9: To draw the projected view of a 3D object using perspective and parallel projection and perform three dimensional transformations to it

Algorithm

i. **Define 3D Object:**

Represent the 3D object using a set of vertices and faces.

ii. **For Perspective Projection:**

Set Camera Parameters:

- Define the position and orientation of the camera.
- Specify the distance between the viewer and the projection plane.

Perform Perspective Projection:

- For each vertex (x, y, z) in the 3D object:
  Calculate the projected coordinates (x', y') on the projection plane using perspective projection formulas.

For Parallel Projection:

Set Projection Direction:

- Choose a projection direction (e.g., top view, front view, side view).

Perform Parallel Projection:

- For each vertex (x, y, z) in the 3D object:

  Calculate the projected coordinates (x', y') on the projection plane using parallel projection formulas.

iii. **Define keyboard Function:**

Switch on the pressed key:

- 'w': rotate counter clockwise in X-axis.
- 's': rotate clockwise in X-axis.
- 'a': rotate counter clockwise in Y-axis.
- 'd': rotate clockwise in Y-axis.
- 'z': Increment translateZ for translation up.
- 'x': Decrement translateZ for translation down.
- 27: Exit if ESC key is pressed.

Request a redraw using glutPostRedisplay()

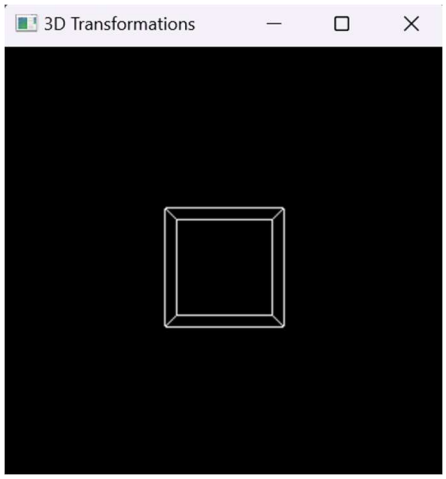
Program

```
#include <GL/glut.h>
float rotateX = 0.0;
float rotateY = 0.0;
float translateZ = -5.0;
void drawCube() {
	glutWireCube(1.0);
}
void display() {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	// Perspective Projection
	gluPerspective(45.0, 1.0, 1.0, 10.0);
	glTranslatef(0.0, 0.0, translateZ);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	// Apply 3D transformations
	glRotatef(rotateX, 1.0, 0.0, 0.0);
	glRotatef(rotateY, 0.0, 1.0, 0.0);
	// Draw the cube
	drawCube();
	glFlush();
	glutSwapBuffers();
}
void keyboard(unsigned char key, int x, int y) {
	switch (key) {
		case 'w':
		rotateX += 5.0;
		break;
		case 's':
		rotateX -= 5.0;
		break;
		case 'a':
```

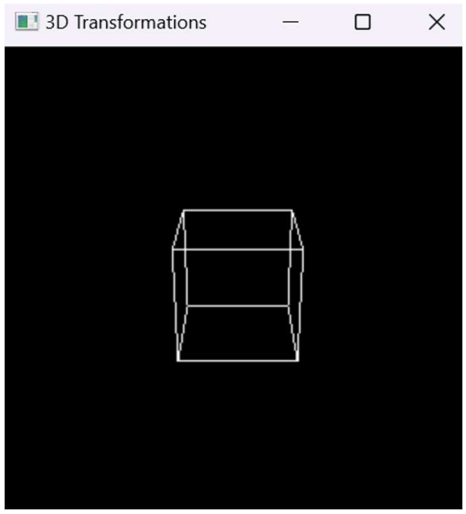
```
                rotateY += 5.0;
                break;
                case 'd':
                rotateY -= 5.0;
                break;
                case 'z':
                translateZ += 0.5;
                break;
                case 'x':
                translateZ -= 0.5;
                break;
                case 27: // ASCII code for 'Esc' key
                exit(0);
        }
        glutPostRedisplay();
}
void reshape(int width, int height) {
        glViewport(0, 0, width, height);
}
int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutCreateWindow("3D Transformations");
        glEnable(GL_DEPTH_TEST);
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutKeyboardFunc(keyboard);
        glutMainLoop();
        return 0;
}
```

Output

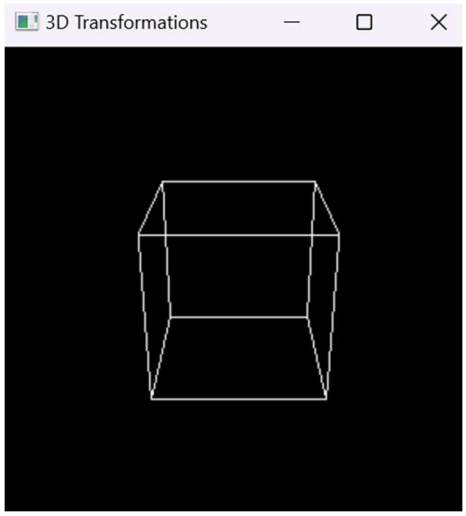
i. Original object



ii. Rotation by an angle 25^0 in clockwise direction in X-axis



iii. Translate by 1.5 in positive Z-axis



Lab 10: To draw a smooth curve using Bezier curve with designated control points

Algorithm

i. **Define Control Points**
Specify four control points (P0, P1, P2, P3) for the cubic Bezier curve.

P0 and P3 are the endpoints, and P1 and P2 are the control points that influence the shape of the curve.

ii. **Calculate Bezier Curve Points:**

For a parameter **t** in the range **[0, 1]**:

Calculate the Bezier curve point **(B(t))** using the cubic Bezier formula:
$B(t)=(1-t)^3*P0+3*(1-t)^2*t*P1+3*(1-t)*t^2*P2+t^3*P3$

iii. Draw the Bezier Curve:

Connect consecutive Bezier curve points with line segments or use a plotting function to visualize the smooth curve.

Program

```
#include <GL/glut.h>
GLfloat controlPoints[4][3] = {
        {-0.5, -0.5, 0.0},
        {-0.25, 0.5, 0.0},
        {0.25, 0.5, 0.0},
        {0.5, -0.5, 0.0}
};
void drawBezierCurve() {
        glBegin(GL_LINE_STRIP);
        for (float t = 0.0; t <= 1.0; t += 0.01) {
                GLfloat x = (1 - t) * (1 - t) * (1 - t) * controlPoints[0][0] +
                3 * t * (1 - t) * (1 - t) * controlPoints[1][0] +
                3 * t * t * (1 - t) * controlPoints[2][0] +
                t * t * t * controlPoints[3][0];
                GLfloat y = (1 - t) * (1 - t) * (1 - t) * controlPoints[0][1] +
                3 * t * (1 - t) * (1 - t) * controlPoints[1][1] +
                3 * t * t * (1 - t) * controlPoints[2][1] +
```

```
            t * t * t * controlPoints[3][1];
            GLfloat z = (1 - t) * (1 - t) * (1 - t) * controlPoints[0][2] +
            3 * t * (1 - t) * (1 - t) * controlPoints[1][2] +
            3 * t * t * (1 - t) * controlPoints[2][2] +
            t * t * t * controlPoints[3][2];
            glVertex3f(x, y, z);
        }
        glEnd();
}
void display() {
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0, 0.0, 0.0); // It provides
```


```
red color for the Bezier curve
        glLineWidth(2.0);
        drawBezierCurve();
        glFlush();
}
void init() {
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
```

```
        gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
}
int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutCreateWindow("Bezier Curve");
        init();
        glutDisplayFunc(display);
        glutMainLoop();
        return 0;
}
```

Output: