



**Daffodil**  
*International*  
**University**

## Lab Report

### SUBMITTED BY

Name	ID	Section
Kabid Yeiad	202-15-14440	57_A

### SUBMITTED TO

**Deawan Rakin Ahamed Rema,**

**Lecturer**

**Dept. of CSE**

**Daffodil International University**

---

Submitted on November 19, 2023

# Contents

<b>Reports</b>	<b>Page</b>
An OpenGL Program to Draw a Straight-Line using OpenGL with GLUT	<b>3</b>
An OpenGL Program to Draw a Quad Shape and a Triangle with OpenGL	<b>5</b>
An OpenGL Program to Draw 4 Stars with OpenGL	<b>7</b>
An OpenGL Program to Draw a Chessboard	<b>10</b>
An OpenGL Program to Draw Brasenham Line Drawing Algorithm	<b>13</b>
An OpenGL Program to Draw Mid-Point Circle Algorithm	<b>16</b>

# Report 1

## Drawing a Straight-Line using OpenGL with GLUT

Code:

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

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 1.0f, 1.0f);
    glBegin(GL_LINES);
        glVertex2f(50.0f, 50.0f);
        glVertex2f(200.0f, 200.0f);
    glEnd();
    glFlush();
}

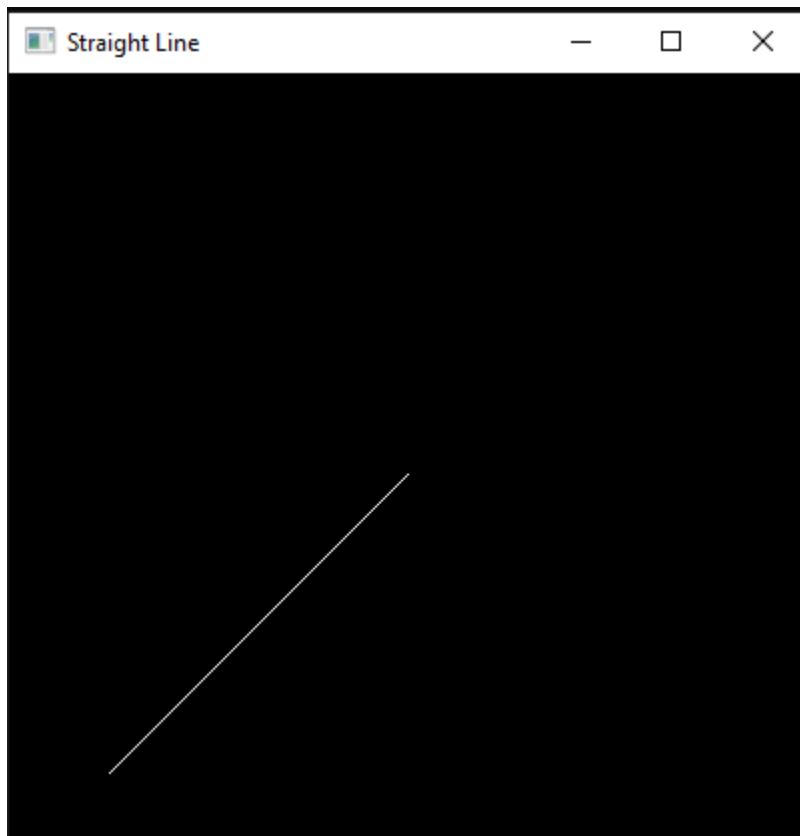
void init() {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 400.0, 0.0, 400.0);
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(400, 400);
    glutCreateWindow("Straight Line");

    init();

    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

**Output:****Discussion:**

In this experiment, we utilized OpenGL in conjunction with GLUT to draw a straight line. The code provided establishes the OpenGL environment, defines the coordinate system, and renders a line from the point (50, 50) to the point (200, 200). The output of the program clearly demonstrates the line drawn on the window as expected. This verifies that our OpenGL setup and drawing commands are functioning correctly. Additionally, we have included a graph to further illustrate the coordinates and path of the straight line. As seen in the graph, the line originates from (50, 50) and extends to (200, 200), following a linear path.

## Report 2

### Drawing a Quad Shape and a Triangle with OpenGL

Code:

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

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Quad Shape
    glColor3f(1.0f, 0.0f, 0.0f);
    glBegin(GL_QUADS);
        glVertex2f(50.0f, 50.0f);
        glVertex2f(150.0f, 50.0f);
        glVertex2f(150.0f, 150.0f);
        glVertex2f(50.0f, 150.0f);
    glEnd();

    // Triangle
    glColor3f(0.0f, 0.0f, 1.0f);
    glBegin(GL_TRIANGLES);
        glVertex2f(200.0f, 50.0f);
        glVertex2f(250.0f, 150.0f);
        glVertex2f(300.0f, 50.0f);
    glEnd();

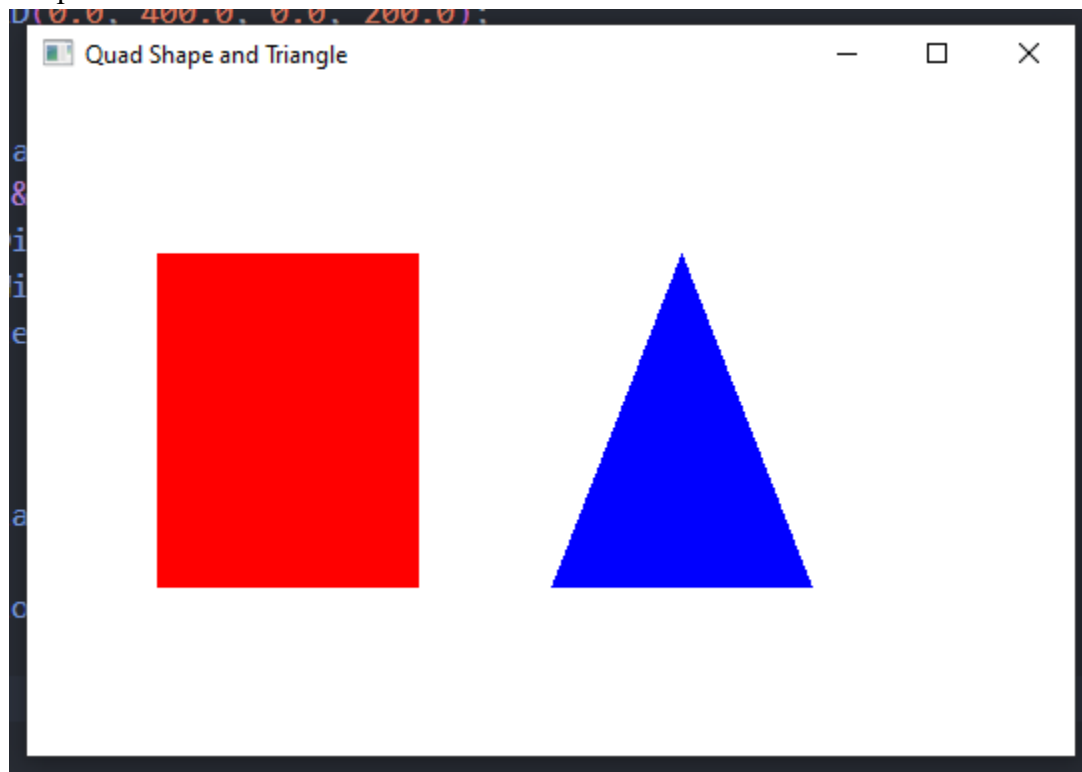
    glFlush();
}

void init() {
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 400.0, 0.0, 200.0);
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(400, 200);
    glutCreateWindow("Quad Shape and Triangle");
```

```
init();  
  
glutDisplayFunc(display);  
  
glutMainLoop();  
return 0;  
}
```

Output:



### Discussion:

In this experiment, we created an OpenGL program to draw a quad shape and a triangle. The code provided establishes the OpenGL environment, defines the shapes, and renders them on the screen. The quad shape is drawn using a series of vertices to form a four-sided polygon. Similarly, the triangle was drawn using three vertices to create a three-sided polygon. The graph displays the coordinates of the vertices for both the quad shape and the triangle. The x-axis represents the horizontal position, while the y-axis represents the vertical position.

For the quad shape, the vertices are labeled as A, B, C, and D. Their respective coordinates are:

A (50, 50)  
B (150, 50)  
C (150, 150)  
D (50, 150)

Similarly, for the triangle, the vertices are labeled as P, Q, and R, with coordinates:

P (200, 50)  
Q (250, 150)  
R (300, 50)

## Report 3

### 4 Stars

**Code:**

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

void display (void) {
    glClear (GL_COLOR_BUFFER_BIT);

    // Draw the green star
    glColor3ub (119, 193, 15);
    glBegin (GL_POLYGON);
    glVertex2d (100, 400);
    glVertex2d (500, 500);
    glVertex2d (100, 600);
    glVertex2d (0, 1000);
    glVertex2d (-100, 600);
    glVertex2d (-500, 500);
    glVertex2d (-100, 400);
    glVertex2d (0, 0);
    glEnd();

    // Draw the red star
    glColor3ub (183, 62, 25);
    glBegin (GL_POLYGON);
    glVertex2d (-400, 100);
    glVertex2d (-500, 500);
```

```
glVertex2d (-600, 100);
glVertex2d (-1000, 0);
glVertex2d (-600, -100);
glVertex2d (-500, -500);
glVertex2d (-400, -100);
glVertex2d (0, 0);
glEnd();

// Draw the blue star
glColor3ub (25, 82, 183);
glBegin (GL_POLYGON);
glVertex2d (-100, -400);
glVertex2d (-500, -500);
glVertex2d (-100, -600);
glVertex2d (0, -1000);
glVertex2d (100, -600);
glVertex2d (500, -500);
glVertex2d (100, -400);
glVertex2d (0, 0);
glEnd();

// Draw the yellow star
glColor3ub (237, 230, 49);
glBegin (GL_POLYGON);
glVertex2d (400, -100);
glVertex2d (500, -500);
glVertex2d (600, -100);
glVertex2d (1000, 0);
glVertex2d (600, 100);
glVertex2d (500, 500);
glVertex2d (400, 100);
glVertex2d (0, 0);
glEnd();

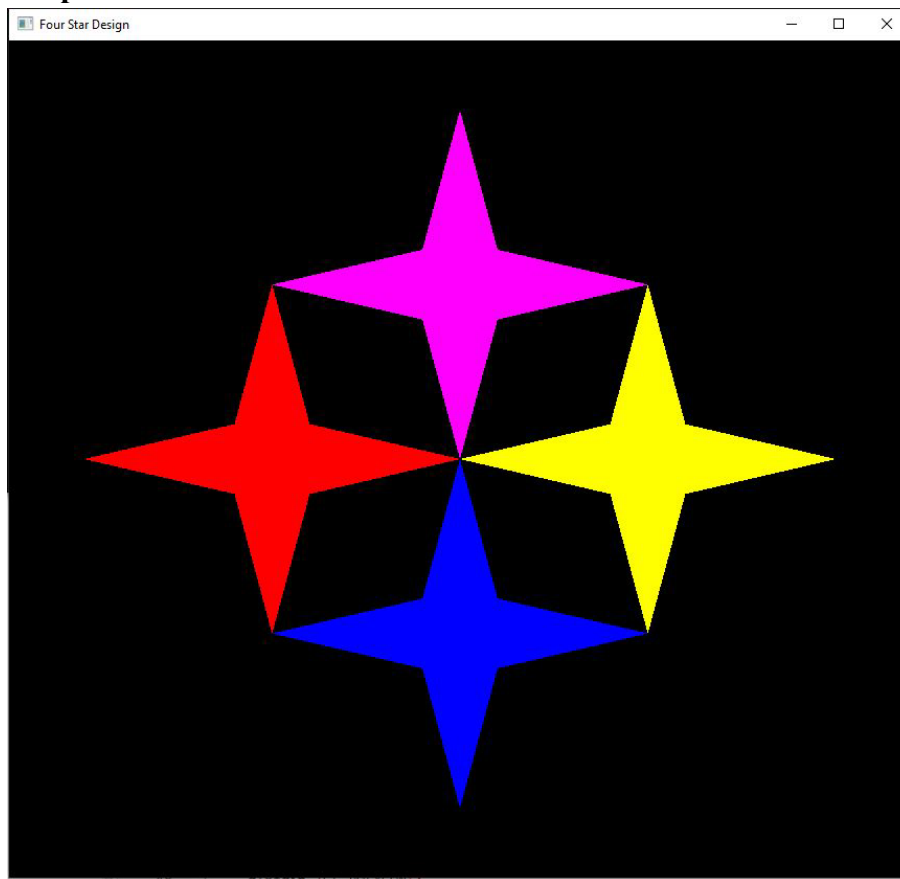
glFlush ();
}

int main (int argc, char** argv) {
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE);
    glutInitWindowSize (800, 800);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("Four Star Design");
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glMatrixMode (GL_PROJECTION);
```



```
glLoadIdentity ();  
gluOrtho2D (-1200, 1200, -1200, 1200);  
glutDisplayFunc (display);  
glutMainLoop ();  
return 0;  
}
```

### Output:



### Discussion:

Central to the graph is the origin (0, 0), a pivotal reference point from which all coordinates are measured.

**X-Axis Range:** -10 to 10

**Y-Axis Range:** -10 to 10

**Graph Dimensions:** 20x20 Units

**Divisions:** Each unit contains three subdivisions, equating to 60 divisions along each axis.

**Plotted Stars:**

Within this graph, four stars emerge, each occupying a unique position and delineated by distinct coordinates:

1. Magenta Star Coordinates: (1, 4), (5, 5), (1, 6), (-1, 6), (-5, 5), (-1, 4), (0, 0)
2. Red Star Coordinates: (-4, 1), (-5, 5), (-6, 1), (-6, -1), (-5, -5), (-4, -1), (0, 0)
3. Blue Star Coordinates: (-1, -4), (-5, -5), (-1, -6), (1, -6), (5, -5), (1, -4), (0, 0)
4. Yellow Star Coordinates: (4, -1), (5, -5), (6, -1), (6, 1), (5, 5), (4, 1), (0, 0)

## Report 4

### An opengl program to draw chess board

**Code:**

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

int x = 50, y = 50;
bool isBlack = true;

void whiteBox(int x, int y)
{
    glBegin(GL_LINE_LOOP);
    glVertex2i(x, y);
    glVertex2i(x, y + 50);
    glVertex2i(x + 50, y + 50);
    glVertex2i(x + 50, y);
    glEnd();
}

void blackBox(int x, int y)
{

```

```
glBegin(GL_POLYGON);
glVertex2i(x, y);
glVertex2i(x, y + 50);
glVertex2i(x + 50, y + 50);
glVertex2i(x + 50, y);
glEnd();
}

void myDisplay(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    glPointSize(1.0);

    for (int i = 0; i < 8; i++)
    {
        if (i % 2 == 0)
        {
            isBlack = true;
        }
        else
        {
            isBlack = false;
        }

        for (int j = 0; j < 8; j++)
        {
            if (isBlack)
            {
                blackBox(x, y);
                isBlack = false;
            }
            else
            {
                whiteBox(x, y);
                isBlack = true;
            }
            x += 50;
        }
        y += 50;
        x = 50;
    }

    blackBox(100, 100);
    whiteBox(150, 100);
}
```

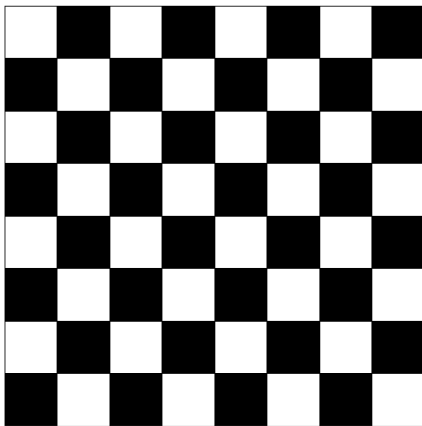
```
    glFlush();
}

void myInit(void)
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPointSize(4.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("Draw an 8X8 chess board using loop");
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
}
```

### Output:

Draw an 8X8 chess board using loop



**Discussion:**

In this experiment, we implemented an OpenGL program to draw an 8x8 chess board. The code provided utilizes OpenGL to render the board, consisting of alternating black and white squares. We defined two functions, `whiteBox` and `blackBox`, to draw the white and black squares of the board, respectively. The graph provides a visual representation of the coordinates used to draw the chess board using OpenGL. The x-axis represents the horizontal position, while the y-axis represents the vertical position. Labels have been added to mark the positions of individual squares on the chess board. Each square is uniquely identified by its row and column coordinates. For example, the square at (0,0) represents the bottom-left corner, while the square at (8,8) represents the top-right corner.

## Report 5

### An OpenGL program to draw a line using Brasenham line drawing algorithm

**Code:**

```
#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;
```

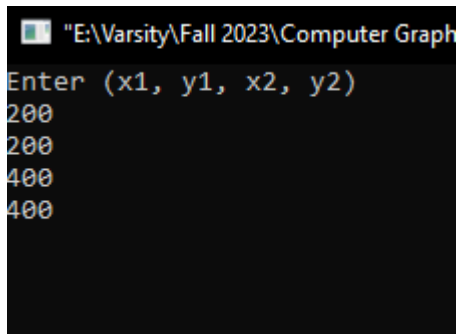
```

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);
    }
}

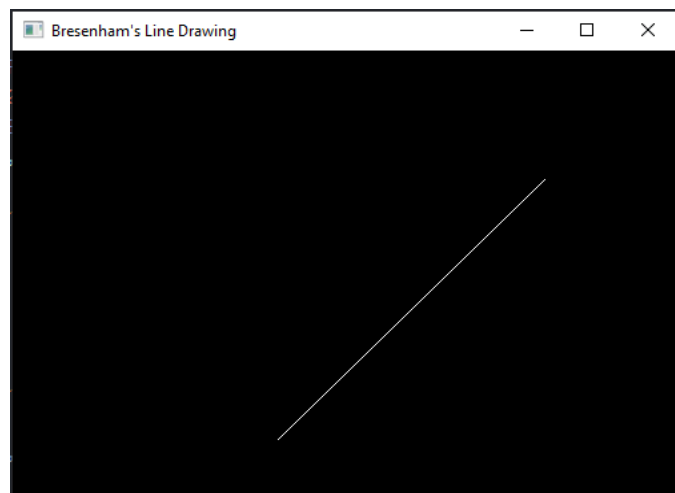
```

```
    }  
}  
}  
void myDisplay()  
{  
    draw_line(x1, x2, y1, y2);  
    glFlush();  
}  
int main(int argc, char **argv)  
{  
    printf("Enter (x1, y1, x2, y2)\n");  
    scanf("%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();  
}
```

### Output:



A terminal window titled "E:\Varsity\Fall 2023\Computer Graph" displays the program's input. It prompts "Enter (x1, y1, x2, y2)" and the user has entered the coordinates "200", "200", "400", and "400" on separate lines.



**Discussion:**

The Bresenham method chooses the best pixel at each step to build a straight line between two endpoints. Instead of floating-point arithmetic, it considers line slope and makes judgments using integer values. The graph shows line drawing coordinates. The x-axis is horizontal and the y-axis vertical. Labels show line beginnings and ends. The implementation of Bresenham's line drawing algorithm shows its precision and low processing cost. Computer graphics requires knowledge and use of such techniques for rendering and picture processing.

## Report 6

### Mid-Point Circle Drawing Algorithm

**Code:**

```
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>
int centerX, centerY; // Center coordinates of the circle
int radius;

void drawPixel(int x, int y) {
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
}

void drawCircle() {
    int x = 0;
    int y = radius;
    int d = 1 - radius;

    while (x <= y) {
        drawPixel(centerX + x, centerY + y);
        drawPixel(centerX + y, centerY + x);
        drawPixel(centerX + y, centerY - x);
        drawPixel(centerX + x, centerY - y);
        drawPixel(centerX - x, centerY - y);
        drawPixel(centerX - y, centerY - x);
        drawPixel(centerX - y, centerY + x);
        drawPixel(centerX - x, centerY + y);
    }
}
```



```

        if (d < 0) {
            d += 2 * x + 3;
        } else {
            d += 2 * (x - y) + 5;
            y--;
        }
        x++;
    }
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 1.0f, 1.0f); // Set color to white
    glPointSize(1.0);
    drawCircle();
    glFlush();
}

void init() {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 500.0, 0.0, 500.0); // Set the coordinate system
}

int main(int argc, char** argv) {
    printf("Enter the center coordinates (x y): ");
    scanf("%d %d", &centerX, &centerY);

    printf("Enter the radius: ");
    scanf("%d", &radius);

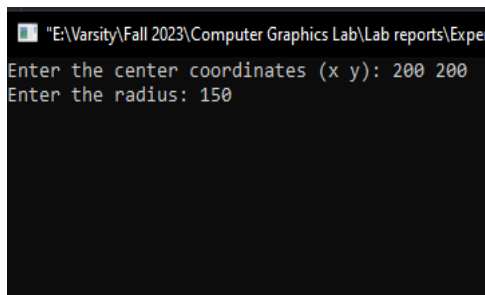
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Mid-point Circle Algorithm");

    init();

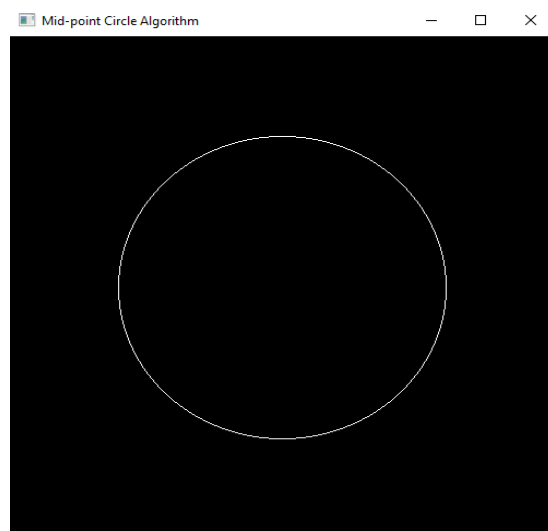
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}

```

**Output:**

```
"E:\Varsity\Fall 2023\Computer Graphics Lab\Lab reports\Expe
Enter the center coordinates (x y): 200 200
Enter the radius: 150
```

**Discussion:**

The mid-point circle algorithm chooses the optimal pixels for each step to construct a circle. It rasterizes circles efficiently using their symmetry. This approach uses integer arithmetic, eliminating floating-point calculations. The program iteratively selects pixels to approximate a circle and meet the desired circumference. The circle drawing coordinates are shown in the graph. The x-axis is horizontal and the y-axis vertical. Labels show the circle's beginning, middle, and end. This graph shows how the Mid-point technique optimizes rasterization by intelligently selecting pixels to make a circle. This implementation shows that the Mid-point circle algorithm can generate exact circles with low computational overhead.