Problem Statement

Write a program that uses the Midpoint Ellipse Drawing Algorithm to draw an ellipse in a graphics window. The user should provide the center coordinates and the horizontal and vertical radii (rx and ry) of the ellipse as input. The program must compute all points on the ellipse using the Midpoint Ellipse Drawing Algorithm and render the shape using the graphics library. The implementation should utilize symmetry and decision parameters to efficiently plot the ellipse with pixel precision.

Background Theory

The Midpoint Ellipse Drawing Algorithm is a foundational technique in computer graphics used to draw ellipses accurately and efficiently using integer arithmetic. Unlike traditional methods that rely on complex mathematical functions and floating-point operations, this algorithm uses a decision parameter to determine the optimal pixel positions, thereby improving performance and precision.

An ellipse differs from a circle in that it has two radii: a horizontal radius (rx) and a vertical radius (ry). The ellipse is divided into two regions based on the slope of the curve. Region 1 handles the portion where the slope is less than 1, and Region 2 handles the portion where the slope is greater than 1. In each region, a different decision parameter is used to determine whether to move horizontally, vertically, or diagonally to the next pixel.

The algorithm takes advantage of the symmetry of ellipses, similar to circles. By calculating points in the first quadrant and reflecting them across the x- and y-axes, the complete ellipse can be drawn with minimal computation.

Algorithm

Step 1: Start

Step 2: Declare variables:

i. xc, yc as Center of ellipse

ii. rx, ry as Semi-major and semi-minor radii

iii. x, y as Current pixel position

iv. p1, p2 as Decision parameters for region 1 and region 2

Step 3: Read values of xc, yc, rx, ry

Step 4: Initialize starting point on the ellipse centered at origin:

```
    $x = 0$
    $y = ry$
```

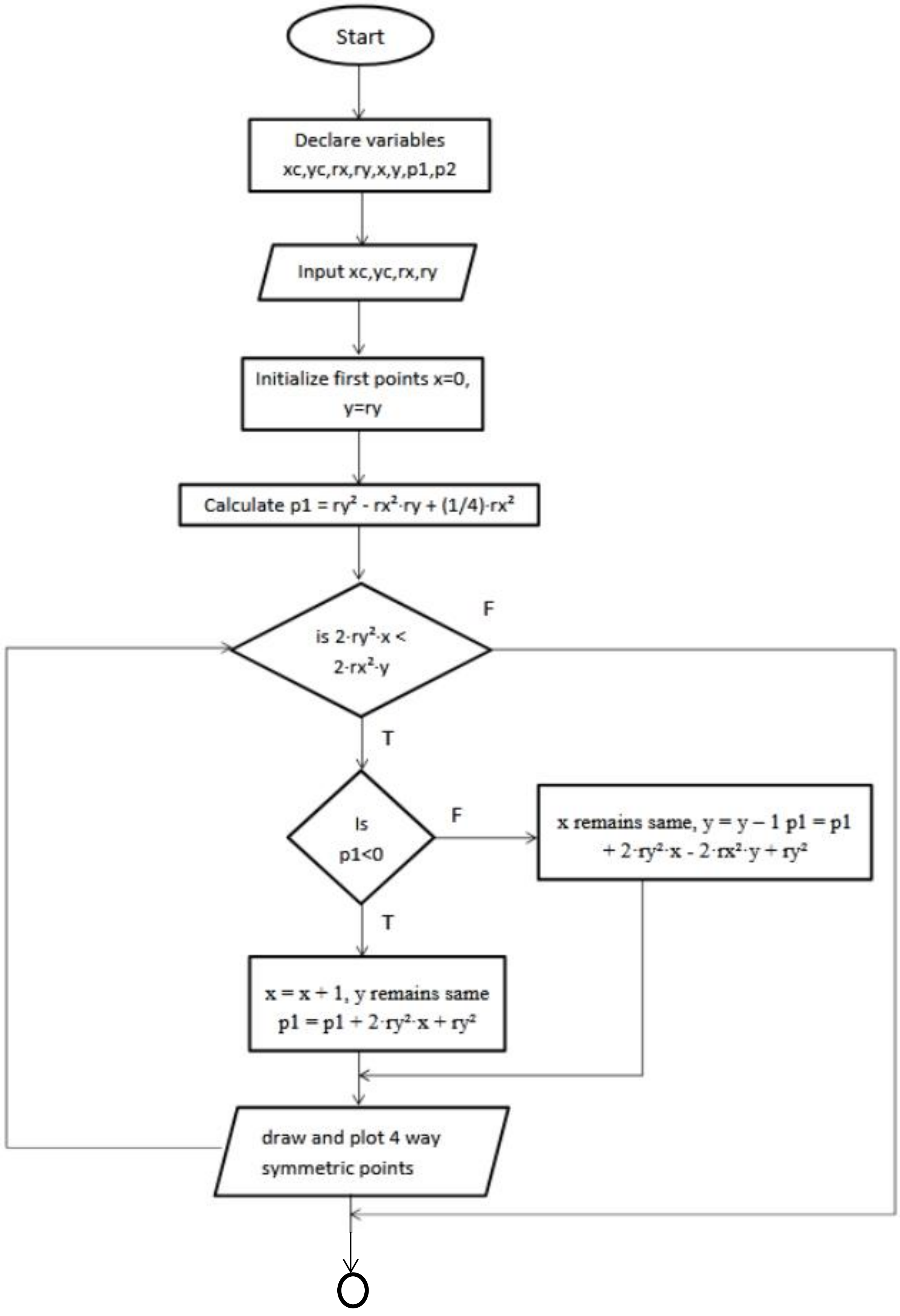
Step 5: Calculate the initial decision parameter for Region 1:

```
    $p1 = ry^2 - rx^2 \cdot ry + (1/4) \cdot rx^2$
```

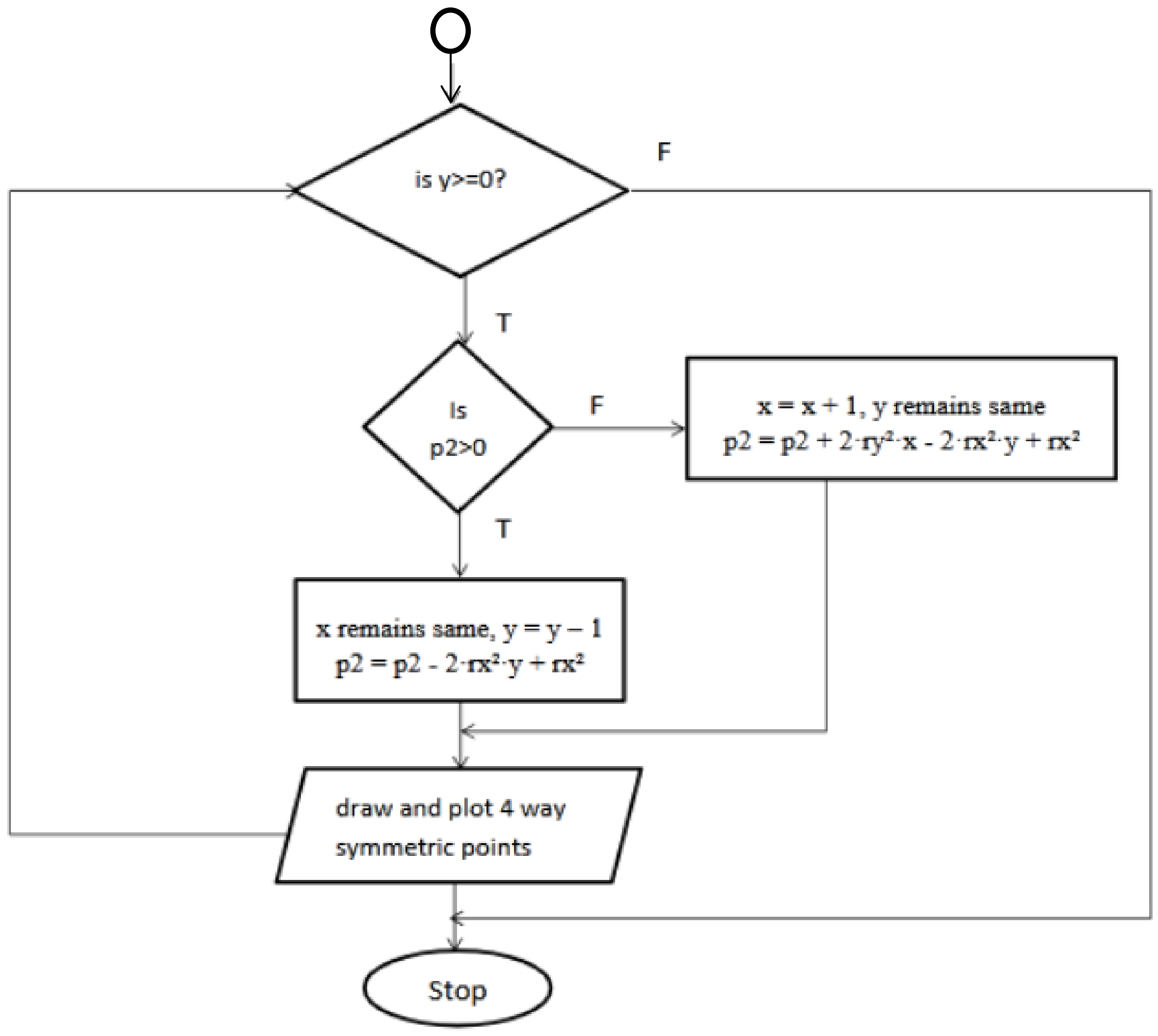
Step 6: For Region 1 (while $2 \cdot ry^2 \cdot x < 2 \cdot rx^2 \cdot y$):

```
    Repeat:
        If $p1 < 0$:
            $x = x + 1$
            $p1 = p1 + 2 \cdot ry^2 \cdot x + ry^2$
        Else:
            $x = x + 1$
            $y = y - 1$
            $p1 = p1 + 2 \cdot ry^2 \cdot x - 2 \cdot rx^2 \cdot y + ry^2$
        Plot symmetric points using 4-way symmetry
    Until $2 \cdot ry^2 \cdot x \geq 2 \cdot rx^2 \cdot y$
Step 7: Calculate the initial decision parameter for Region 2:
    $p2 = ry^2 \cdot (x + 0.5)^2 + rx^2 \cdot (y - 1)^2 - rx^2 \cdot ry^2$
Step 8: For Region 2 (while $y \geq 0$):
    Repeat:
      If $p2 > 0$:
        $y = y - 1$
        $p2 = p2 - 2 \cdot rx^2 \cdot y + rx^2$
      Else:
        $x = x + 1$
        $y = y - 1$
        $p2 = p2 + 2 \cdot ry^2 \cdot x - 2 \cdot rx^2 \cdot y + rx^2$
      Plot symmetric points using 4-way symmetry
Step 9: For each (x, y), determine 4 symmetric points:
    (xc + x, yc + y)
    (xc - x, yc + y)
    (xc + x, yc - y)
    (xc - x, yc - y)
Step 10: Plot these pixel positions using `putpixel()`
Step 11: Repeat until Region 2 is complete ($y < 0$)
Step 12: Stop
```

Flowchart



Start
Declare variables xc,yc,rx,ry,x,y,p1,p2
Input xc,yc,rx,ry
Initialize first points x=0, y=ry
Calculate p1 = ry² - rx²·ry + (1/4)·rx²
is 2·ry²·x < 2·rx²·y
F
T
Is p1<0
F
x remains same, y = y – 1 p1 = p1 + 2·ry²·x - 2·rx²·y + ry²
T
x = x + 1, y remains same p1 = p1 + 2·ry²·x + ry²
draw and plot 4 way symmetric points


is y>=0?
F
T
Is p2>0
F
$x = x + 1$, y remains same
$p2 = p2 + 2 \cdot ry^2 \cdot x - 2 \cdot rx^2 \cdot y + rx^2$
T
x remains same, $y = y - 1$
$p2 = p2 - 2 \cdot rx^2 \cdot y + rx^2$
draw and plot 4 way symmetric points
Stop

Source code

```
#include <graphics.h>

#include <iostream>

using namespace std;


void drawEllipsePoints(int xc, int yc, int x, int y) {

    putpixel(xc + x, yc + y, WHITE);

    putpixel(xc - x, yc + y, WHITE);

    putpixel(xc + x, yc - y, WHITE);

    putpixel(xc - x, yc - y, WHITE);
```

```
}


void midpointEllipse(int xc, int yc, int rx, int ry) {
    float dx, dy, p1, p2;
    int x = 0;
    int y = ry;

    // Initial decision parameter for region 1
    p1 = (ry * ry) - (rx * rx * ry) + (0.25 * rx * rx);
    dx = 2 * ry * ry * x;
    dy = 2 * rx * rx * y;

    // Region 1
    while (dx < dy) {
        drawEllipsePoints(xc, yc, x, y);
        x++;
        dx = dx + (2 * ry * ry);
        if (p1 < 0) {
            p1 = p1 + dx + (ry * ry);
        } else {
            y--;
            dy = dy - (2 * rx * rx);
            p1 = p1 + dx - dy + (ry * ry);
        }
    }

    // Initial decision parameter for region 2
    p2 = (ry * ry) * ((x + 0.5) * (x + 0.5)) +
         (rx * rx) * ((y - 1) * (y - 1)) -
         (rx * rx * ry * ry);

    // Region 2
    while (y >= 0) {
        drawEllipsePoints(xc, yc, x, y);
        y--;
        dy = dy - (2 * rx * rx);
        if (p2 > 0) {
            p2 = p2 + (rx * rx) - dy;
        } else {
            x++;
            dx = dx + (2 * ry * ry);
            p2 = p2 + dx - dy + (rx * rx);
        }
    }
}
```

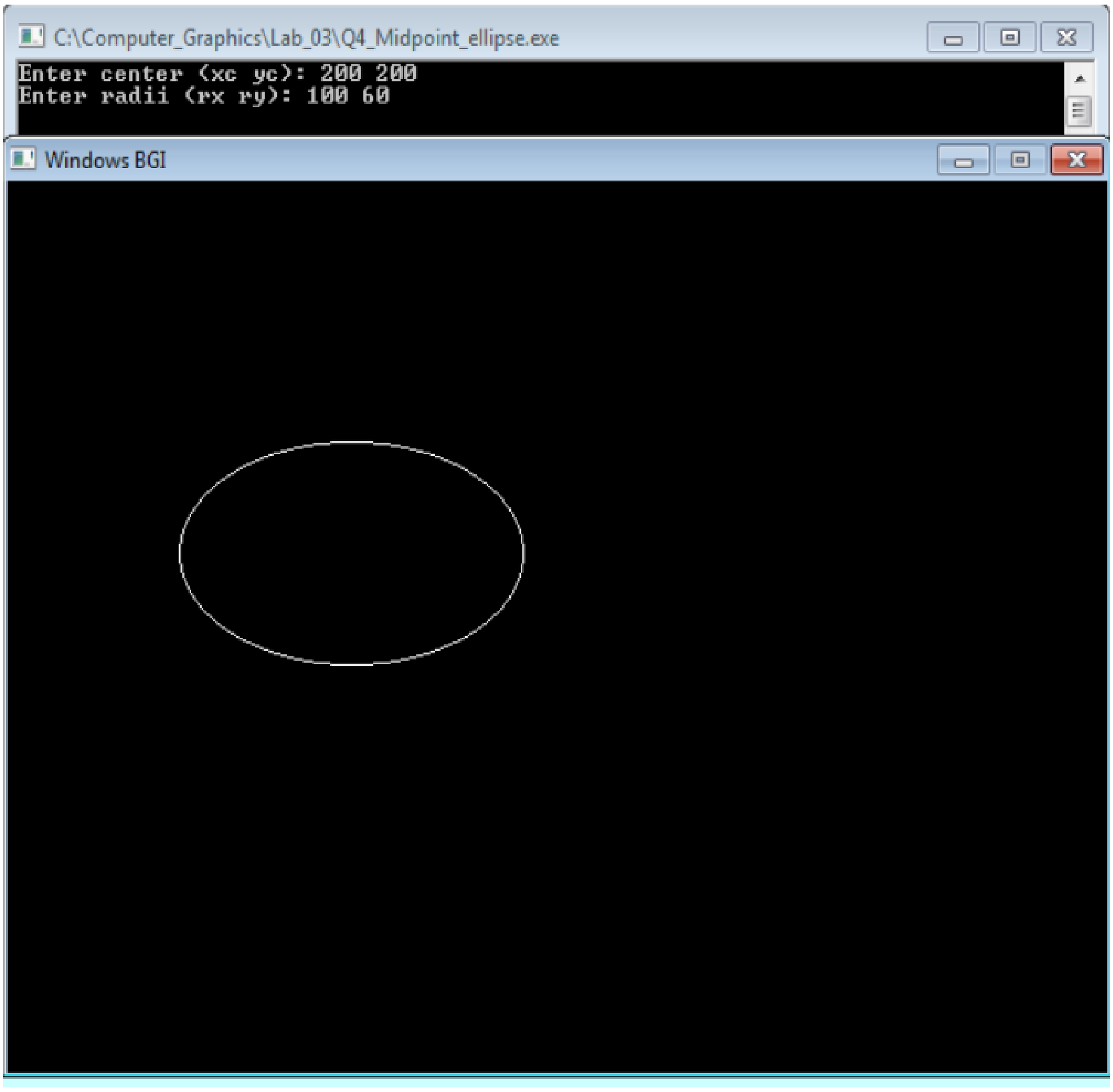
```
int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    int xc, yc, rx, ry;
    cout << "Enter center (xc yc): ";
    cin >> xc >> yc;
    cout << "Enter radii (rx ry): ";
    cin >> rx >> ry;

    midpointEllipse(xc, yc, rx, ry);

    getch();
    closegraph();
    return 0;
}
```

Sample input and output



```
Enter center (xc yc): 200 200
Enter radii (rx ry): 100 60
```

Conclusion:

The Midpoint Ellipse Drawing Algorithm provides an efficient and accurate method for rendering ellipses in computer graphics using only integer calculations. By avoiding floating-point operations and leveraging symmetry, the algorithm ensures optimal performance while maintaining visual accuracy. This lab effectively demonstrated the implementation of the Midpoint Ellipse Algorithm in C++ using the graphics.h library. The exercise reinforced the importance of mathematical modeling, incremental computation, and algorithmic optimization in real-time graphics programming and the broader field of computer graphics.