**LAB-4**

**Midpoint Ellipse Drawing Algorithm**

**Objective: -**

* Efficiently determine ellipse pixel positions using integer arithmetic, avoiding floating-point calculations.
* Utilize ellipse symmetry to reduce computations by calculating points for one-fourth only.

**Theory: -**

The Midpoint Ellipse Drawing Algorithm efficiently draws a circle by applying integer arithmetic and symmetry. By focusing on one-fourth of the ellipse, it minimizes redundant calculations and computational load. The algorithm incrementally determines pixel positions, ensuring smooth curves and consistency, while avoiding complex operations such as trigonometry or floating-point arithmetic.

**Algorithm: -**

1. Initialize Variables:  
 a. Set the initial coordinates (x, y) as (0, ry).  
 b. Compute rx^2, ry^2, 2 \* rx^2, and 2 \* ry^2.  
 c. Define the initial decision parameter p1 for Region 1 as ry^2 - (rx^2 \* ry) + 0.25 \* rx^2.

a. Set the initial coordinates (x, y) as (0, radius).

b. Define the initial decision parameter (d) as 1 - radius.

c. Prepare a list or storage structure to store the calculated points.

2. Region 1 Iteration:  
 a. While 2 \* ry^2 \* x <= 2 \* rx^2 \* y:  
 - Plot points for all four quadrants using symmetry.  
 - Update x and p1:  
 i. If p1 < 0, update p1 as p1 + 2 \* ry^2 \* x + ry^2.  
 ii. Otherwise, decrement y and update p1 as p1 + 2 \* ry^2 \* x - 2 \* rx^2 \* y + ry^2.

a. Use the symmetry of the circle to plot the initial points in all eight octants.

b. For a circle centered at (0, 0), the octant points are:

(x, y), (y, x), (-x, y), (-y, x), (-x, -y), (-y, -x), (x, -y), (y, -x).

3. Region 2 Initial Decision Parameter:  
 a. Compute p2 for Region 2 as ry^2 \* (x + 0.5)^2 + rx^2 \* (y - 1)^2 - rx^2 \* ry^2.

a. Evaluate the decision parameter (d):

i. If d < 0, the midpoint is inside the circle, so update d as d + 2 \* x + 3.

ii. If d ≥ 0, the midpoint is outside or on the circle, so:

- Update d as d + 2 \* (x - y) + 5.

- Decrement y by 1.

b. Increment x by 1.

4. Region 2 Iteration:  
 a. While y >= 0:  
 - Plot points for all four quadrants using symmetry.  
 - Update y and p2:  
 i. If p2 > 0, decrement y and update p2 as p2 - 2 \* rx^2 \* y + rx^2.  
 ii. Otherwise, increment x and decrement y, updating p2 as p2 + 2 \* ry^2 \* x - 2 \* rx^2 \* y + rx^2.

a. Using the updated (x, y) values, calculate and plot the points for all eight octants using symmetry.

5. Complete the Ellipse:  
 a. Ensure all calculated points are plotted, forming a complete ellipse.

a. Continue updating (x, y) and plotting points until x equals or surpasses y.

* a. Ensure all calculated points are plotted, forming a complete circle.

**Code: -**

import matplotlib.pyplot as plt

rx = int(input('Enter the horizontal radius'))

ry = int(input('Enter the vertical radius'))

xc = int(input('Enter the x coords of center'))

yc = int(input('Enter the y coords of center'))

def plot(x, y):

    plt.plot(x,y,marker='o')

def midpoint\_ellipse(rx, ry, xc, yc):

    # Step 1: Initialize variables

    x = 0

    y = ry

    rx2 = rx \* rx

    ry2 = ry \* ry

    two\_rx2 = 2 \* rx2

    two\_ry2 = 2 \* ry2

    # Step 2: Region 1 Initial Decision Parameter

    p1 = ry2 - (rx2 \* ry) + (0.25 \* rx2)

    # Step 3: Iterate through Region 1

    while two\_ry2 \* x <= two\_rx2 \* y:

        # Plot points for all four quadrants

        plot(x + xc, y + yc)

        plot(-x + xc, y + yc)

        plot(x + xc, -y + yc)

        plot(-x + xc, -y + yc)

        # Update x and decision parameter

        if p1 < 0:

            x += 1

            p1 += two\_ry2 \* x + ry2

        else:

            x += 1

            y -= 1

            p1 += two\_ry2 \* x - two\_rx2 \* y + ry2

    # Step 4: Region 2 Initial Decision Parameter

    p2 = (ry2 \* (x + 0.5) \*\* 2) + (rx2 \* (y - 1) \*\* 2) - (rx2 \* ry2)

    # Step 5: Iterate through Region 2

    while y >= 0:

        # Plot points for all four quadrants

        plot(x + xc, y + yc)

        plot(-x + xc, y + yc)

        plot(x + xc, -y + yc)

        plot(-x + xc, -y + yc)

        # Update y and decision parameter

        if p2 > 0:

            y -= 1

            p2 -= two\_rx2 \* y + rx2

        else:

            x += 1

            y -= 1

            p2 += two\_ry2 \* x - two\_rx2 \* y + rx2

midpoint\_ellipse(rx,ry,xc,yc)

plt.show()

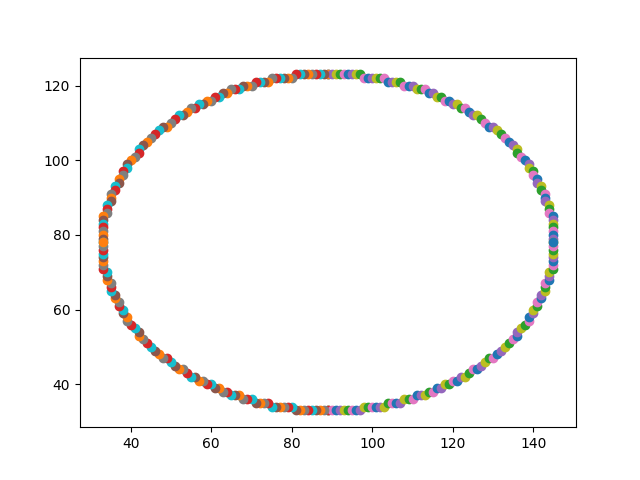
**Output: -**

Enter the horizontal radius:-56

Enter the vertical radius:-45

Enter the x coords of center:-89

Enter the y coords of center:-78



**Discussion**

The Midpoint Ellipse Drawing Algorithm is highly efficient because it leverages integer arithmetic and symmetry, calculating only one-eighth of the circle's points and reflecting them to reduce redundant calculations. This efficiency makes it ideal for real-time rendering of geometric shapes. The decision parameter is updated incrementally, avoiding complex operations like trigonometry and ensuring both speed and precision.

However, the algorithm is best suited for perfect circles. Modifying it to draw other shapes, such as ellipses, requires additional modifications, increasing complexity. Additionally, its dependence on symmetry limits its application to symmetrical shapes, making it less effective for non-standard or irregular curves.

**Conclusion**

The Midpoint Ellipse Drawing Algorithm is a fundamental technique in computer graphics for efficiently rendering circles. By utilizing symmetry and integer-based calculations, it ensures accurate and smooth rendering on raster displays. Despite its limitations, its simplicity and computational efficiency make it essential for graphics applications and visual computing.