**LAB-3**

**Midpoint Circle Drawing Algorithm**

**Objective: -**

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

**Theory: -**

The Midpoint Circle Drawing Algorithm efficiently draws a circle by applying integer arithmetic and symmetry. By focusing on one-eighth of the circle, 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, radius).

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

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

2. Plot Initial Points:

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. Iterate While x < y:

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. Plot New Points:

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

5. Repeat Steps 3-4 Until x >= y:

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

6. Complete the Circle:

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

**Code: -**

import matplotlib.pyplot as plt

def MCA():

    def SP(x,y):

        xes.append(x)

        yes.append(y)

        xes.extend([x, -x, -x, x, y, -y, -y, y])

        yes.extend([y, y, -y, -y, x, x, -x, -x])

    r=int(input('Enter the radius: '))

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

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

    x,y=0,r

    p=1-r

    xes=[]

    yes=[]

    SP(x,y)

    while(x<y):

        x=x+1

        if(p<0):

            p=p+2\*x+1

        else:

            y=y-1

            p=p+2\*(x-y)+1

        SP(x,y)

    xes = [xc + xi for xi in xes]

    yes = [yc + yi for yi in yes]

    plt.scatter(xes, yes, color='blue', s=10)

    plt.title("Midpoint Circle Algorithm")

    plt.xlabel("X")

    plt.ylabel("Y")

    plt.grid()

    plt.show()

MCA()

**Output: -**

Enter the radius: 78

Enter the x coord of center: 34

Enter the y coord of center: 56

A graph with a blue circle

Description automatically generated

**Discussion**

The Midpoint Circle 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 Circle 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.