Lab Report No:01

Experiment Name: Implementation of Mid-Point Circle Algorithm.

Description:

The Midpoint Circle Algorithm is a simple and efficient way to draw circles in computer graphics using only basic math and pixel plotting. Instead of calculating every point on the circle using complex formulas, it starts from the top of the circle and uses a decision-based method to choose the next point, while taking advantage of the circle's symmetry to draw all eight parts at once. This makes the drawing process much faster and accurate using only integers. In this project, we implemented the algorithm to see how a perfect circle can be drawn step by step on the screen using this clever technique.

The Midpoint Circle Drawing Algorithm is a basic and important method in computer graphics used to draw circles. It uses only integer calculations and takes advantage of the circle's symmetry to draw points faster and more efficiently. By calculating points for just one octant (or part of the circle), it mirrors them across the other seven octants to complete the full circle.

Algorithm:

Given:

Center coordinates: (Xc, Yc)

Radius: R

Step 1: Initialize Values

Set the starting point at the top of the circle:

X = 0

Y = R

Step 2: Calculate the initial value parameter.

P = 1 - R

Step 3: Decision Parameter Cases

Suppose the current point is (X, Y), and we want to find the next point. The following cases apply based on the value of the decision parameter P:

Case 1: If P < 0

X = X + 1

P = P + 2 * X + 1

Case 2: If P ≥ 0

X = X + 1

Y = Y - 1

P = P + 2 * X - 2 * Y + 1

Step 4: Plot Symmetrical Points

For each (X, Y) calculated, plot the 8 symmetric points using the center (Xc, Yc):

If the given centre point (X0, Y0) is not (0, 0), then do the following and plot the point-

Xplot = Xc + X0

Yplot = Yc + Y0

Here, (Xc, Yc) denotes the current value of X and Y coordinates.

Step 5: Repeat until X ≥ Y

Continue the loop and plot points until the condition X ≥ Y is satisfied.

Step 6:

Step-05 generates all the points for one octant. To find the points for other seven octants, follow the eight symmetry property of circle.

Code:

```
#include <graphics.h>
#include <conio.h>
#include <iostream>
using namespace std;
void drawCircle(int xc, int yc, int r) {
    int x = 0;
    int y = r;
    int p = 1 - r;
while (x < y) {
```

```
        x++;
        if (p < 0) {
            p = p + 2 * x + 1;
        } else {
            y--;
            p = p + 2 * (x - y) + 1;
        }
// 8 symmetric points
        putpixel(xc + x, yc + y, 5);
        putpixel(xc - x, yc + y, 3);
        putpixel(xc + x, yc - y, 5);
        putpixel(xc - x, yc - y, 3);
        putpixel(xc + y, yc + x, 5);
        putpixel(xc - y, yc + x, 3);
        putpixel(xc + y, yc - x, 5);
        putpixel(xc - y, yc - x, 3);
    }
}
int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");
 setbkcolor(WHITE); // Set background color to White
    cleardevice(); // Apply the background color
    int xc, yc, r;
    cout << "Enter center point coordinate (xc yc): ";
    cin >> xc >> yc;
    cout << "Enter radius of circle R: ";
    cin >> r;
 drawCircle(xc, yc, r);
    getch();
    closegraph();
```

```
    return 0;
}
```

Output:

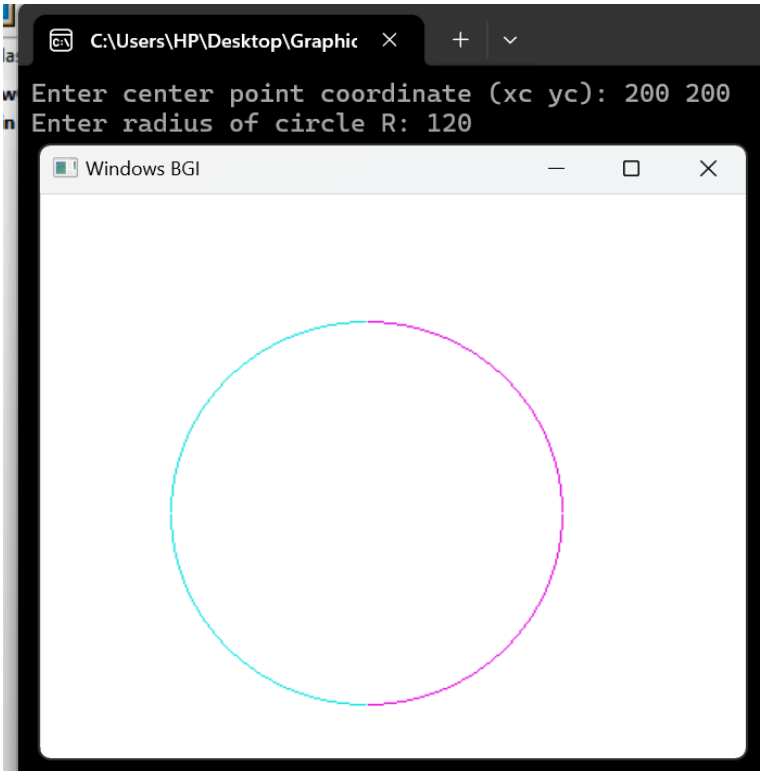

Figure: Output of Mid-Point Circle Algorithm.

Conclusion:

Implementing the Midpoint Circle Algorithm helped us understand how simple mathematical logic can be used to draw accurate and smooth circles using only integer calculations. By using symmetry and a decision parameter, the algorithm efficiently determines the pixels needed without relying on complex operations. This hands-on experience deepened our knowledge of fundamental computer graphics techniques and showed how algorithms can make graphical rendering faster and more resource-friendly. It lays a strong foundation for exploring more advanced shape-drawing and rendering methods in the future.

Remarks: Using App: