Report No: 02

Experiment Name: Bresenham Circle Drawing Algorithm.

Description:

Bresenham's Circle Drawing Algorithm is an efficient technique used in computer graphics to draw circles using only integer arithmetic, avoiding the need for floating-point calculations. It uses a decision parameter to determine the next pixel position along the circumference of the circle. The algorithm starts at the topmost point and calculates points for one-eighth of the circle. By exploiting the symmetry of the circle, it mirrors each calculated point to the remaining seven octants. This approach significantly reduces computation time and enhances performance in raster-based systems. The decision parameter is updated incrementally at each step, depending on the midpoint's position relative to the ideal circle. Due to its speed and simplicity, this algorithm is commonly used in low-level graphics programming and embedded systems.

Algorithm:

Given:

Center coordinates: (Xc, Yc)

Radius: R

Step 1: Initialize Values

Start at the top of the circle:

$X = 0$

$Y = R$

Step 2: Calculate the Initial Decision Parameter

The decision parameter P is calculated as:

$P = 3 - 2 * R$

Step 3: Decision Parameter Cases

The algorithm uses two conditions to decide the next point:

Case 1: If $P < 0$

$X = X + 1$

$P = P + 4 * X + 6$

Case 2: If $P \geq 0$

$X = X + 1$

$Y = Y - 1$

P = P + 4 * X - 4 * Y + 10

Step 4: Plot Symmetrical Points

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

(Xc + X, Yc + Y)

(Xc - X, Yc + Y)

(Xc + X, Yc - Y)

(Xc - X, Yc - Y)

(Xc + Y, Yc + X)

(Xc - Y, Yc + X)

(Xc + Y, Yc - X)

(Xc - Y, Yc - X)

Step 5: Repeat Until X ≥ Y

Continue calculating points and plotting them until the condition X ≥ Y is satisfied.

Step 6: Generate Points for All Octants

The points calculated in one octant are mirrored across the other seven octants using the symmetry property of the 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 = 3 - 2 * r;

while (x <= y) {

      // 8 symmetric points

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

```
        putpixel(xc - x, yc + y, WHITE);
        putpixel(xc + x, yc - y, WHITE);
        putpixel(xc - x, yc - y, WHITE);
        putpixel(xc + y, yc + x, WHITE);
        putpixel(xc - y, yc + x, WHITE);
        putpixel(xc + y, yc - x, WHITE);
        putpixel(xc - y, yc - x, WHITE);
 x++;
 if (p < 0) {
            p = p + 4 * x + 6;
        } else {
            y--;
            p = p + 4 * (x - y) + 10;
        }
    }
}
int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");
 int xc, yc, r;
    cout << "Enter center point (xc yc): ";
    cin >> xc >> yc;
    cout << "Enter radius r: ";
    cin >> r;
 drawCircle(xc, yc, r);
 getch();
    closegraph();
    return 0;
```

}

Output:

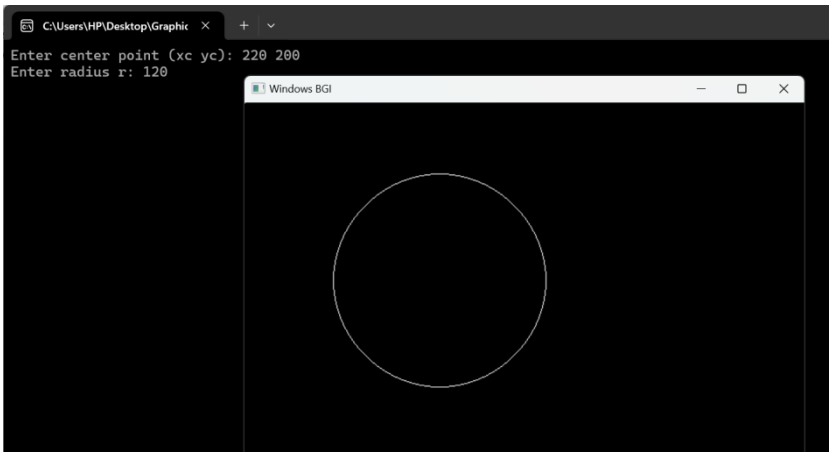

Figure: Output of Bresenham Circle Drawing Algorithm.

Conclusion:

Bresenham's Circle Drawing Algorithm provides an efficient and accurate method for rendering circles in computer graphics. By relying solely on integer arithmetic and exploiting the symmetry of the circle, it minimizes computational complexity and enhances performance, especially in real-time applications. Its use of a decision parameter to determine pixel positions makes it ideal for systems with limited processing power. Overall, the algorithm is a foundational technique in computer graphics and remains widely used due to its simplicity, speed, and effectiveness.