**EXPERIMENT NO 3**

~ ARYA RAUL

COMPS 20

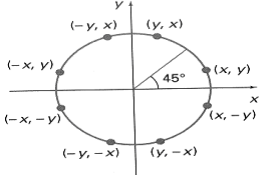
**Aim:**  To implement midpoint circle algorithm.

**Objective:**

Draw a circle using mid-point circle drawing algorithm by determining the points needed for rasterizing a circle. The mid-point algorithm to calculate all the perimeter points of the circle in the first octant and then print them along with their mirror points in the other octants.

**Theory:**

The shape of the circle is similar in each quadrant. We can generate the points in one section and the points in other sections can be obtained by considering the symmetry about x-axis and y-axis.

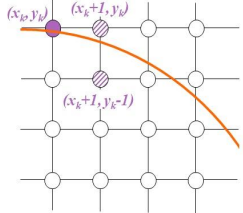


The equation of circle with center at origin is x2 + y2 = r2

Let the circle function is fcircle (x, y) -

* is < 0, if (x, y) is inside circle boundary,
* is = 0, if (x, y) is on circle boundary,
* is > 0, if (x, y) is outside circle boundary.

Consider the pixel at (xk, yk) is plotted,



Now the next pixel along the circumference of the circle will be either (xk + 1, yk) or (xk + 1, yk – 1) whichever is closer the circle boundary.

Let the decision parameter pk is equal to the circle function evaluate at the mid-point between two pixels.

If pk < 0, the midpoint is inside the circle and the pixel at yk is closer to the circle boundary.

Otherwise, the midpoint is outside or on the circle boundary and the pixel at yk – 1 is closer to the circle boundary.

**Algorithm –**

1) Accept the radius r and center of circle. Let the first point on the circumference of the circle is (x0, y0) = (0, r).

2) Calculate the initial value of decision parameter p0 as –

p0 = (5/4) – r

3) At each xk position starting at k = 0 perform the following test.

If pk < 0, then

xnext = xk + 1

ynext = yk

pk+1 = pk + 2 xk + 3

otherwise,

xnext = xk + 1

ynext = yk – 1

pk+1 = pk + 2xk – 2yk + 5

4) Determine the symmetry points in the other seven octants.

5) Translate each calculated pixel position by T(xc, yc) and display the pixels.

x = xk+1 + xc

y = yk+1 + yc

putpixel (x, y, Colour)

6) Repeat the steps 3 through 5 until x ⩾ y.

7) Stop

**Program –**

#include <iostream.h>

#include <conio.h>

#include <graphics.h>

#include <dos.h>

void drawCircle(int centerX, int centerY, int radius) {

int x = 0;

int y = radius;

int p = 1 - radius;

while (x <= y) {

putpixel(centerX + x, centerY + y, WHITE);

putpixel(centerX + y, centerY + x, WHITE);

putpixel(centerX - x, centerY + y, WHITE);

putpixel(centerX - y, centerY + x, WHITE);

putpixel(centerX + x, centerY - y, WHITE);

putpixel(centerX + y, centerY - x, WHITE);

putpixel(centerX - x, centerY - y, WHITE);

putpixel(centerX - y, centerY - x, WHITE);

if (p < 0) {

p += 2 \* x + 3;

} else {

p += 2 \* (x - y) + 5;

y--;

}

x++;

}

}

int main() {

int gd = DETECT, gm;

initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

int centerX, centerY, radius;

cout << "Enter the center coordinates (x y): ";

cin >> centerX >> centerY;

cout << "Enter the radius: ";

cin >> radius;

drawCircle(centerX, centerY, radius);

getch();

closegraph();

return 0;

}

**output –**

****

**Conclusion:**

1. Fast or slow: Fast

2. Draw one arc only and repeat the process in 8 quadrants: Drawing one arc using the midpoint algorithm and then repeating the process in all eight quadrants to form a complete circle is a clever optimization technique to reduce redundant calculations and improve efficiency.The midpoint algorithm calculates only one octant (one-eighth) of the circle and then mirrors and translates the results to draw the remaining seven octants. By doing so, it avoids redundant computations for each pixel in all eight octants, effectively reducing the number of calculations by a factor of 8.

3. Difference with line drawing method: The Midpoint Algorithm and Line Drawing Algorithm are two different algorithms used in computer graphics for drawing circles and lines, respectively. Here's a comparison between the two:

1. Purpose:
   * Midpoint Algorithm: It is used for drawing circles.
   * Line Drawing Algorithm: It is used for drawing straight lines.
2. Basic Principle:
   * Midpoint Algorithm: It calculates the positions of pixels to form a circle based on a decision parameter and adjusts it incrementally as it moves along the circumference.
   * Line Drawing Algorithm: It uses incremental calculations to determine the positions of pixels on a straight line.
3. Efficiency:
   * Midpoint Algorithm: It is efficient for drawing circles as it uses integer arithmetic and avoids costly floating-point operations.
   * Line Drawing Algorithm: It is also efficient for drawing lines, especially when using the Bresenham's Line Drawing Algorithm, as it minimizes calculations and uses integer arithmetic.
4. Complexity:
   * Midpoint Algorithm: It has a complexity of O(n) since it needs to calculate the pixels along the circumference of the circle.
   * Line Drawing Algorithm: It has a complexity of O(|dx| + |dy|), where dx and dy are the differences in x and y coordinates of the two endpoints.
5. Applicability:
   * Midpoint Algorithm: It is specifically designed for drawing circles and ellipses. It is not suitable for drawing lines.
   * Line Drawing Algorithm: It is designed for drawing lines and not suitable for circles or curves.
6. Visual Quality:
   * Midpoint Algorithm: It produces accurate circles without any jaggedness or visual artifacts.
   * Line Drawing Algorithm: Line Drawing Algorithm produces straight lines without jaggedness, but it can introduce slight visual artifacts due to the limited precision of integer arithmetic.