EXPERIMENT 3:- MIDPOINT ALGORITHM

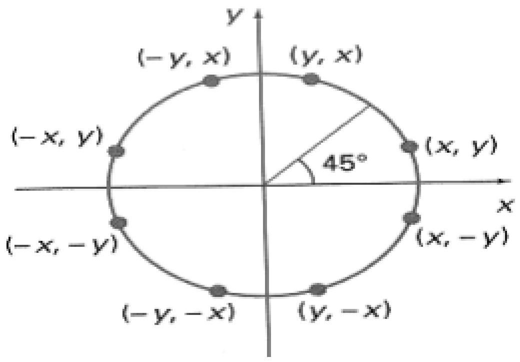
Aim: To implement midpoint circle algorithm.

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.

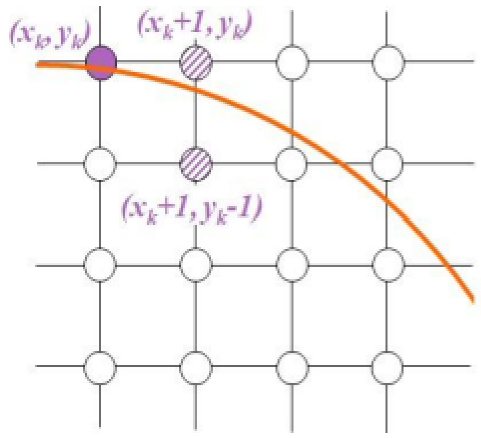


The equation of circle with center at origin is $x^2 + y^2 = r^2$

Let the circle function is f_{circle} (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 –

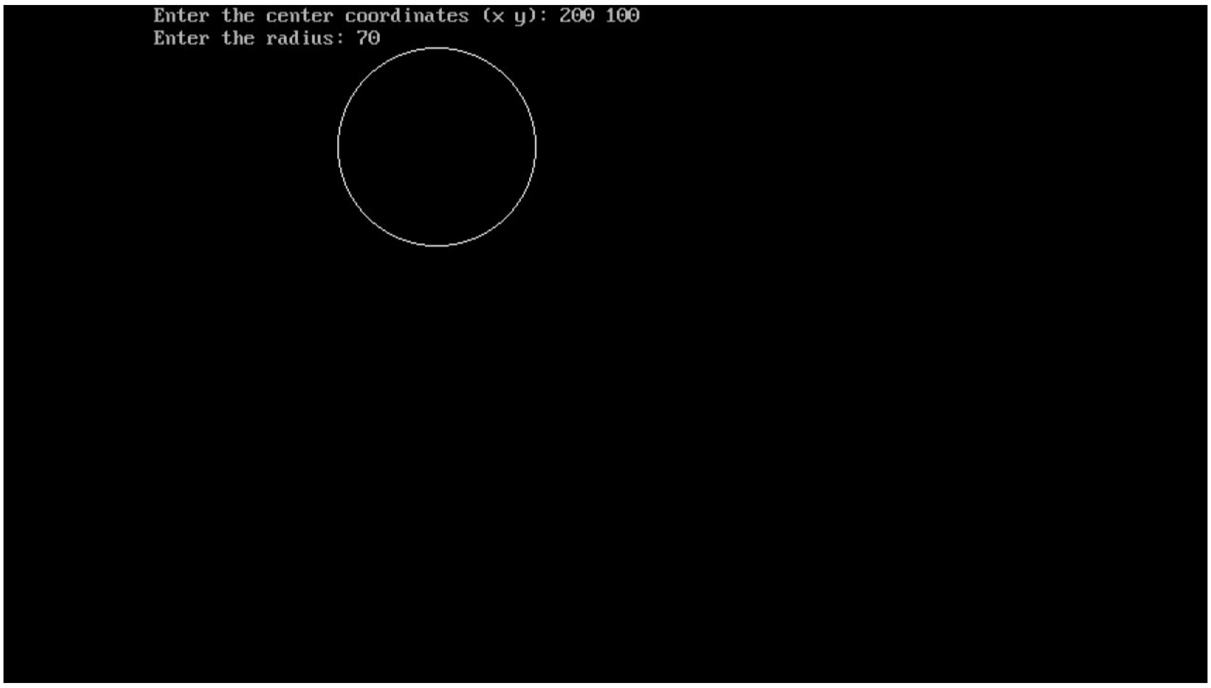
```
1) Accept the radius r and center of circle. Let the first point on the circumference of the
circle is (x0, y0) = (0, r).
2) Calculate the initial value of decision parameter p0 as –
        p0 = (5/4) – r
3) At each xk position starting at k = 0 perform the following test.
        If pk < 0, then
                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 –



```
Enter the center coordinates (x y): 200 100
Enter the radius: 70
```

Conclusion:

The midpoint algorithm is a powerful tool for rasterizing circles and ellipses on a discrete grid, offering efficient and accurate results in computer graphics. Its ability to minimize floating-point calculations and its simplicity make it an appealing choice for circle-based rendering tasks. By leveraging the midpoint algorithm, developers can achieve smooth and visually pleasing circular shapes without compromising performance