**Title**: To use **Midpoint Circle Drawing Algorithm** to draw a circle with given center and radius.

**Objective**:

1. To implement the Midpoint Circle Drawing Algorithm.
2. To evaluate how well the algorithm approximates a circle with pixels.
3. To analyze the efficiency and speed of the algorithm.
4. To visualize and display circle with varying parameters.

**Theory:**

A circle in a 2D plane is defined as the set of all points that are equidistant from a central point (the center) with a given radius. In computer graphics, similar to line drawing, we cannot directly render a perfect circle due to the discrete nature of pixels. Instead, we approximate the circle by determining and illuminating the closest pixels that best represent the circular path. The equation of the circle is x2 + y2 = r2. With center (0,0) and radius r, a circle function is defined as circle (x,y) = x2 + y2 – r2.

In graphics programming, the output screen acts as a coordinate system with the origin (0, 0) at the top-left corner. The x-coordinate increases to the right, and the y-coordinate increases downward. The Midpoint Circle Drawing Algorithm (MCDA) is an efficient method to compute the pixel coordinates that form a circle. It uses integer calculations to minimize computational overhead and ensures smooth, visually appealing circles by deciding the optimal pixels to activate.

The initial coordinates and decision parameter are (0,r) and Pk = 1 – r.

If Pk < 0, the next pixel is at (xk + 1, yk).

Pk+1 = Pk + 2xk+1 + 1

If Pk ≥ 0, the next pixel is at (xk + 1, yk - 1).

Pk+1 = Pk + 2xk+1 – 2yk+1 + 1

Using functions like `putpixel(x, y, color)` in C, MCDA allows us to render circle by illuminating the appropriate pixels on the screen, producing a precise and performance-friendly circle drawing.

****Midpoint Circle Drawing** Algorithm:**

1. Start
2. Declare variables xc, yc, x0, y0, p0, and r.
3. Read values of xc, yc, and r.
4. Initialize the x and y i.e. set the co-oordinates for the first point on the circumference of the circle centered at the origin as :  
    x0 = 0;

y0 = r;

1. Calculate the initial decision parameter :  
    p0 = 1 – r ;
2. At each xk position, starting from k = 0

If pk < 0

xk+1  = xk  + 1

yk+1  = yk

pk+1 = pk + 2xk+1 + 1

else

xk+1  = xk  + 1

yk+1  = yk  - 1

pk+1 = pk + 2xk+1 - 2yk+1  + 1

1. Determine the symmetry in other seven octants.
2. Move each calculated pixel position (x, y) onto the circular path centered on (xc, yc)
3. Plot the coordinates values

x = x + xc

y = y + yc

1. Repeat steps 6 to 9 until x ≥ y.
2. Stop

**Source Code:**

// Midpoint Circle Drawing Algorithm to draw a circle.

#include <stdio.h>

#include <graphics.h>

void drawCirclePoints(int xc, int yc, int x, int y, int col) {

putpixel(xc + x, yc + y, col);

putpixel(xc - x, yc + y, col);

putpixel(xc + x, yc - y, col);

putpixel(xc - x, yc - y, col);

putpixel(xc + y, yc + x, col);

putpixel(xc - y, yc + x, col);

putpixel(xc + y, yc - x, col);

putpixel(xc - y, yc - x, col);

}

void midpointCircle(int xc, int yc, int r, int col) {

int x = 0;

int y = r;

int p = 1 - r;

drawCirclePoints(xc, yc, x, y, col);

while (x < y) {

x++;

if (p < 0) {

p += 2 \* x + 1;

} else {

y--;

p += 2 \* (x - y) + 1;

}

drawCirclePoints(xc, yc, x, y, col);

delay(50); // Delay for visualization

}

}

int main() {

int gd = DETECT, gm;

initgraph(&gd, &gm, NULL);

int xc, yc, r;

printf("Created by Kushal Shah\nMid-Point Circle Drawing Algorithm\n");

printf("Enter center of the circle (xc yc): ");

scanf("%d %d", &xc, &yc);

printf("Enter radius of the circle: ");

scanf("%d", &r);

outtextxy(20, 20, "Kushal Shah");

for(int i=0;i<3;i++){

midpointCircle(xc, yc, r+i\*15, i+2);

}

delay(50000);

closegraph();

return 0;

}

**Output:**

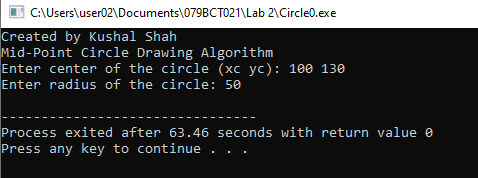


Figure 1: Inserting center and radius of the circle.

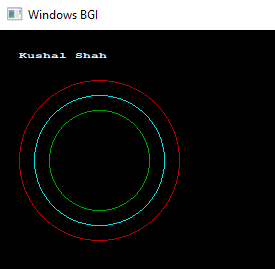


Figure 2: Drawing circle using MCDA

**Discussion**:

The Midpoint Circle Drawing Algorithm is an efficient method for rasterizing circles in computer graphics. By leveraging the symmetry of circles, the algorithm computes only one-eighth of the circle and mirrors the results to the other octants, significantly reducing the number of calculations required. This approach not only simplifies the implementation but also enhances performance, making it suitable for real-time applications where computational resources are limited.

The algorithm operates using integer arithmetic, which minimizes rounding errors and improves efficiency. It employs an incremental decision parameter to determine the next pixel to illuminate, allowing for smooth transitions along the circle's perimeter. However, while the algorithm excels in rendering perfect circles, it may struggle with precision in lower resolutions, leading to a pixelated appearance. Overall, the Midpoint Circle Drawing Algorithm remains a fundamental technique in computer graphics, balancing simplicity and efficiency in circle rendering.

**Conclusion**:

The Mid-Point Circle Algorithm is a robust technique for circle rendering in computer graphics. By utilizing symmetry and decision parameters, it efficiently computes pixel locations, minimizing computational load while ensuring visual accuracy. This algorithm is foundational in graphical applications, providing a reliable method for circle generation across various digital platforms. Its efficiency makes it suitable for real-time applications where performance is critical. The implementation stages outlined provide a clear pathway for effective coding and application of the algorithm in graphics programming.