

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 two-dimensional Cartesian plane is formally defined as the locus of all points equidistant from a fixed central point, with this constant distance being the radius (r). The standard equation for a circle centered at the origin $(0,0)$ is given by $x^2+y^2=r^2$. Alternatively, a circle function can be defined as $\text{circle}(x,y)=x^2+y^2-r^2$.

In the context of computer graphics, the discrete nature of display pixels precludes the direct rendering of geometrically perfect circles. Consequently, algorithms are employed to approximate the circular path by selectively illuminating the most appropriate pixels. This process involves determining the pixel coordinates that best represent the continuous curve on a grid of discrete points.

The **Midpoint Circle Drawing Algorithm (MCDA)** is an optimized method for computing the pixel coordinates required to render a circle. This algorithm is particularly efficient due to its reliance on integer-only arithmetic, which minimizes computational overhead. The MCDA operates by evaluating a decision parameter at the midpoint between candidate pixels, thereby determining the optimal pixel to activate. This approach ensures the generation of visually smooth and aesthetically pleasing circular approximations.

The initial coordinates and decision parameter are $(0,r)$ and $P_k = 1 - r$.

If $P_k < 0$, the next pixel is at $(x_k + 1, y_k)$.

$$P_{k+1} = P_k + 2x_{k+1} + 1$$

If $P_k \geq 0$, the next pixel is at $(x_k + 1, y_k - 1)$.

$$P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1$$

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:

Step 1: Start

Step 2: Declare variables x_c , y_c , x_0 , y_0 , p_0 , and r .

Step 3: Read values of x_c , y_c , and r .

Step 4: Initialize the x and y i.e. set the co-ordinates for the first point on the circumference of the circle centered at the origin as :

$$x_0 = 0;$$

$$y_0 = r;$$

Step 5: Calculate the initial decision parameter :

$$p_0 = 1 - r ;$$

Step 6: At each x_k position, starting from $k = 0$

If $p_k < 0$

$$x_{k+1} = x_k + 1$$

$$y_{k+1} = y_k$$

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

else

$$x_{k+1} = x_k + 1$$

$$y_{k+1} = y_k - 1$$

$$p_{k+1} = p_k + 2x_{k+1} - 2y_{k+1} + 1$$

Step 7: Determine the symmetry in other seven octants.

Step 8: Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c)

Step 9: Plot the coordinates values

$$x = x + x_c$$

$$y = y + y_c$$

Step 10: Repeat steps 6 to 9 until $x \geq y$.

Step 11: 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 Shishir Timilsina\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);

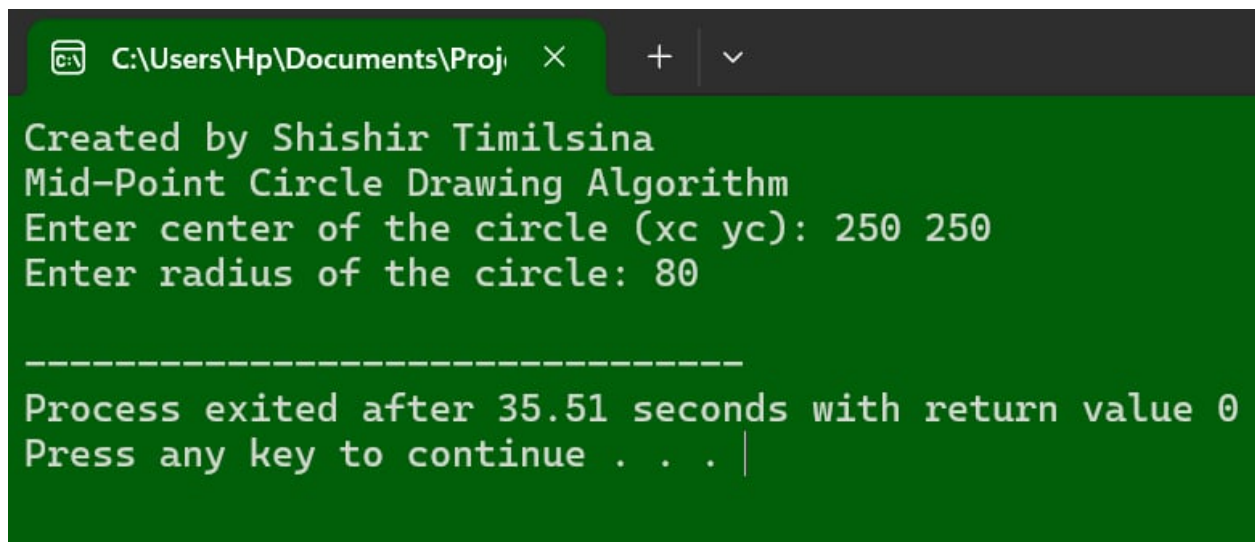
```

```
outtextxy(230, 300, "Shishir Timilsina");

for(int i=0;i<3;i++){
    midpointCircle(xc, yc, r+i*15, i+2);
}

delay(50000);
closegraph();
return 0;
}
```

Output:



```
C:\Users\Hp\Documents\Proj >
Created by Shishir Timilsina
Mid-Point Circle Drawing Algorithm
Enter center of the circle (xc yc): 250 250
Enter radius of the circle: 80

-----
Process exited after 35.51 seconds with return value 0
Press any key to continue . . . |
```

Figure 1: Inserting center and radius of the circle.

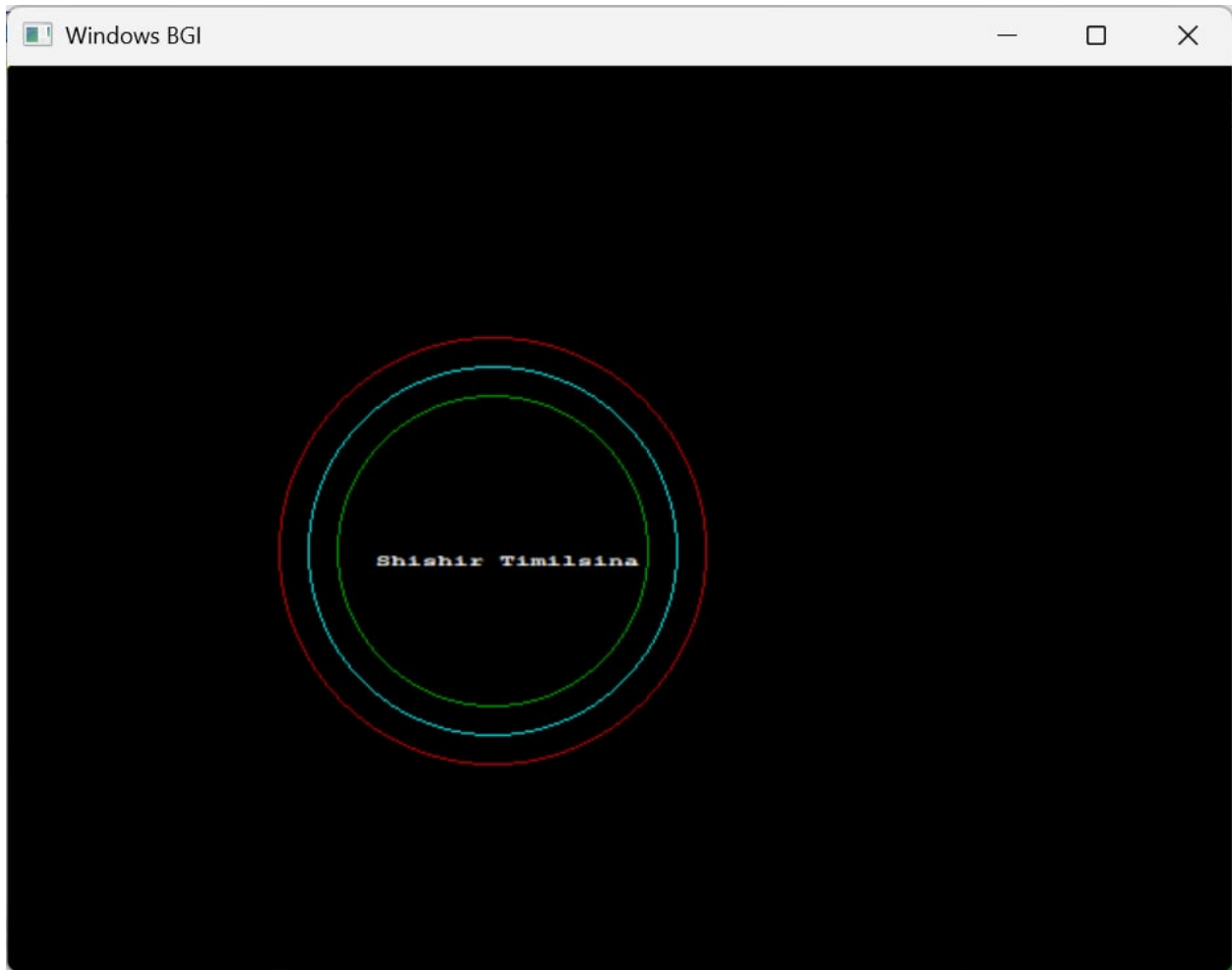


Figure 2: Drawing circle using MCDA

Discussion and Conclusion:

The Midpoint Circle Drawing Algorithm (MCDA) efficiently rasterizes circles by exploiting eight-way symmetry, calculating only one-eighth of the points and mirroring them across the coordinate axes and diagonals. This strategic approach significantly reduces the computational overhead, as it avoids redundant calculations for symmetrical points. The reduced computational burden makes the MCDA particularly well-suited for real-time graphics applications, where processing speed and resource optimization are critical. Furthermore, its reliance on integer arithmetic is a significant advantage, as it inherently minimizes cumulative rounding errors that can occur with floating-point operations, thereby improving both the accuracy and efficiency of the pixel selection process. The algorithm employs an incremental decision parameter, which is continuously updated to determine the optimal next pixel to illuminate, ensuring smooth and visually appealing transitions along the circle's perimeter.

While the MCDA is highly effective for rendering precise circles, especially at higher resolutions where individual pixels are less noticeable, its visual fidelity may decrease when applied to

lower resolutions. In such scenarios, the discrete nature of the pixels becomes more pronounced, potentially leading to a more noticeable "pixelated" or "jagged" appearance of the circle. Despite this limitation, the MCDA remains a fundamental and widely taught technique in computer graphics, successfully balancing algorithmic simplicity with practical rendering efficiency.

Conclusion

Thus, as shown in the program above, we can draw a Circle by plotting individual pixels using Mid-point Circle Drawing Algorithm with the graphics functions provided in the graphics.h header file.