Midpoint Circle Drawing Algorithm

A Structured Study Note with Code Examples

Quick Notes

- Also known as Bresenham's Circle Algorithm (the integer-only version is often derived via the midpoint approach).
- Efficiently draws circles on raster displays using primarily **integer arithmetic**.
- Evaluates the circle function at the **midpoint** between two candidate pixels to decide which pixel is closer to the ideal circle.
- Exploits **8-way symmetry** for performance, calculating points for only one octant.

1. Initial Setup

Similar to other incremental algorithms, we define the circle and starting parameters. We focus on the second octant (90 to 45 degrees).

- Circle Center $(x_c, y_c) = (0, 0)$
- Radius R
- Starting Point: $(x_0, y_0) = (0, R)$
- Circle Function: $f(x, y) = x^2 + y^2 - R^2$. $f(x, y) < 0$ for points inside, $f(x, y) > 0$ for points outside.
- Initial Decision Parameter p_0: Evaluate $f(x, y)$ at the midpoint between the first two candidate pixels $(1, R)$ and $(1, R-1)$. The midpoint is $(1, R - \frac{1}{2})$.

  $$p_0 = f(1, R - \tfrac{1}{2}) = 1^2 + (R - \tfrac{1}{2})^2 - R^2 = 1 + R^2 - R + \tfrac{1}{4} - R^2 = \tfrac{5}{4} - R$$

  For integer-only arithmetic, we can define $p_k = f(x_k + 1, y_k - \frac{1}{2})$, which leads to an integer starting value:

  $$p_0 = 1 - R$$

  (This common simplification effectively shifts the decision boundary slightly but produces the same pixel choices).

- **Example:** For $R = 5$:

  $$p_0 = 1 - 5 = -4$$

2. Decision Parameter and Update Rules

At each step k, starting with (x_k, y_k), we consider the midpoint $M = (x_k + 1, y_k - \frac{1}{2})$ between the next two candidate pixels: East $E = (x_k + 1, y_k)$ and South-East $SE = (x_k + 1, y_k - 1)$. The decision parameter $p_k = f(M)$ determines which pixel is closer.

Table 1: Update Rules Based on Midpoint Decision Parameter p_k

Condition	**Midpoint Location (Relative to Circle)**	**Next Point Chosen** (x_{k+1}, y_{k+1})	**Update Rule for** p_{k+1}
$p_k < 0$	Midpoint is inside	$\text{E} = (x_k + 1, y_k)$	$p_{k+1} = p_k + 2x_k + 3$
$p_k \geq 0$	Midpoint is outside or on	$\text{SE} = (x_k + 1, y_k - 1)$	$p_{k+1} = p_k + 2(x_k - y_k) + 5$

The calculation proceeds from $x = 0$ and stops when $x \geq y$.

3. Step-by-Step Calculation (Example: R=5)

Given: $R = 5$, Center $(0, 0)$, Start Point $(0, 5)$, $p_0 = -4$.

Table 2: Midpoint algorithm calculation steps for R=5.

Step (k)	**Current Point** (x_k, y_k)	p_k	$p_k < 0$?	**Next Point** (x_{k+1}, y_{k+1})	**Calculation for** p_{k+1}
0	(0, 5)	-4	Yes	(1, 5)	$p_1 = p_0 + 2x_0 + 3$ $= -4 + 2(0) + 3 = -1$
1	(1, 5)	-1	Yes	(2, 5)	$p_2 = p_1 + 2x_1 + 3$ $= -1 + 2(1) + 3 = 4$
2	(2, 5)	4	No	(3, 4)	$p_3 = p_2 + 2(x_2 - y_2) + 5$ $= 4 + 2(2 - 5) + 5$ $= 4 - 6 + 5 = 3$
3	(3, 4)	3	No	(4, 3)	$p_4 = p_3 + 2(x_3 - y_3) + 5$ $= 3 + 2(3 - 4) + 5$ $= 3 - 2 + 5 = 6$
Stop: $x = 4, y = 3 \implies x \geq y$					

The points generated in the second octant are: $(0, 5), (1, 5), (2, 5), (3, 4), (4, 3)$. (These are identical to the points generated by the Bresenham algorithm derivation used previously).

4. Exploiting 8-Way Symmetry

The Midpoint algorithm, like Bresenham's, calculates points for only one octant. The full circle is obtained by reflecting these points across the axes and diagonals ($y = x, y = -x$). If (x, y) is a point calculated in the second octant (from $x = 0$ to $x = y$), the following points are also on the circle:

Table 3: Mapping a Point (x, y) to All 8 Octants (Same as Bresenham)

Octant	Symmetric Point	Transformation from (x, y)
1 ($0° - 45°$)	(y, x)	Swap x and y
2 ($45° - 90°$)	(x, y)	Original point (calculated)
3 ($90° - 135°$)	$(-x, y)$	Negate x
4 ($135° - 180°$)	$(-y, x)$	Swap x and y, negate new x
5 ($180° - 225°$)	$(-y, -x)$	Swap x and y, negate both
6 ($225° - 270°$)	$(-x, -y)$	Negate both
7 ($270° - 315°$)	$(x, -y)$	Negate y
8 ($315° - 360°$)	$(y, -x)$	Swap x and y, negate new y

Applying Symmetry to R=5 Example Points:
The calculated points $(0, 5), (1, 5), (2, 5), (3, 4), (4, 3)$ generate the same full set of circle points as shown in the Bresenham example when symmetry is applied.

5. Visualization (TikZ)

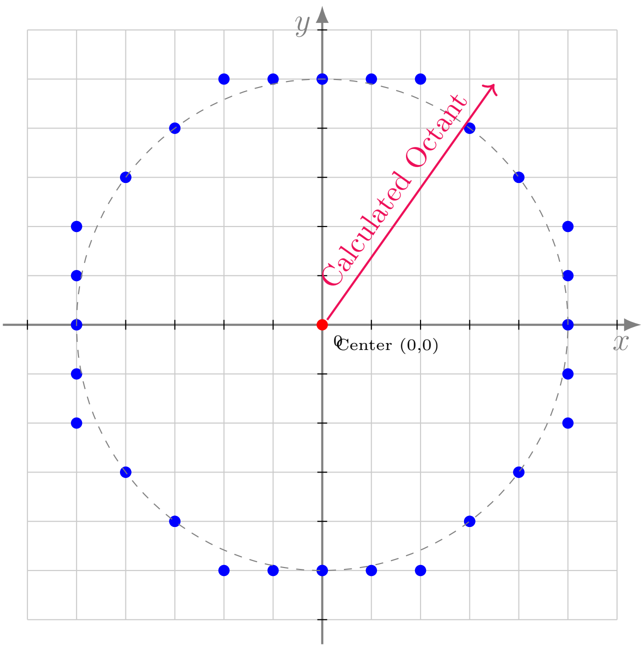

Figure 1: Plot of points for R=5 generated by the Midpoint Circle algorithm, showing 8-way symmetry.

6. Code Implementations

6.1. C++ / OpenGL Snippet

This snippet shows the Midpoint Circle algorithm logic. Assumes ‘drawPixel‘ and ‘drawCirclePoints‘ (for symmetry) exist as defined previously.

C++ / OpenGL Implementation Snippet

```
#include <GL/glut.h> // Or your preferred GL header

// Assumed: drawPixel(x, y) and drawCirclePoints(cx, cy, x, y) exist

// Midpoint Circle Drawing Algorithm
void drawCircleMidpoint(int cx, int cy, int r) {
    int x = 0;
    int y = r;
    int p = 1 - r; // Initial decision parameter (integer version)

    // Draw the first set of points based on the starting point (0, r)
    drawCirclePoints(cx, cy, x, y);

    // Calculate points for the second octant (from y-axis to y=x line)
    while (x < y) {
        x++; // Always increment x
        if (p < 0) {
            // Midpoint is inside, choose E pixel
            // Update p for the next E candidate: p_new = p + 2x_new + 1
            // Since x_new = x_old + 1, this is p + 2(x+1) + 1 = p + 2x + 3
            p = p + 2 * x + 3;
            // y remains the same
        } else {
            // Midpoint is outside or on the circle, choose SE pixel
            // Update p for the next SE candidate: p_new = p + 2x_new - 2y_new + 1
            // Since x_new = x+1, y_new = y-1, this is p + 2(x+1) - 2(y-1) + 1
            // p + 2x + 2 - 2y + 2 + 1 = p + 2(x - y) + 5
            y--; // Decrement y
            p = p + 2 * (x - y) + 5;
        }
        // Draw the calculated point and its symmetric counterparts
        drawCirclePoints(cx, cy, x, y);
    }
}

/*
// Example Usage in a GLUT display function:
void display() {
```

```
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 1.0, 0.0); // Set color to green

    glBegin(GL_POINTS); // Begin drawing points
    drawCircleMidpoint(0, 0, 50); // Draw circle at center
       (0,0) with radius 50
    glEnd(); // End drawing points

    glFlush();
}
*/
```

6.2. Python Snippet

Python implementation of the Midpoint Circle algorithm.

Python Implementation Snippet

```python
import matplotlib.pyplot as plt

def midpoint_circle_octant(r):
    """
    Calculates points for the second octant (90 to 45
       degrees)
    using the Midpoint Circle algorithm. Starts at (0, r).
    """
    x = 0
    y = r
    p = 1 - r  # Initial decision parameter (integer version)
    points = []

    # Add the starting point
    points.append((x, y))

    # Iterate until x >= y (end of the octant)
    while x < y:
        x += 1
        if p < 0:
            # Select E point (x+1, y)
            p = p + 2 * x + 3
        else:
            # Select SE point (x+1, y-1)
            y -= 1
            p = p + 2 * (x - y) + 5

        points.append((x, y))

    return points

```

```python
def get_symmetric_points(x, y):
    """Generates all 8 symmetric points for a given point (x, y)."""
    # (Same function as in Bresenham example)
    return [
        (x, y), (y, x), (-x, y), (-y, x),
        (x, -y), (y, -x), (-x, -y), (-y, -x)
    ]

def get_all_midpoint_circle_points(cx, cy, r):
    """
    Generates all points for a circle centered at (cx, cy) with radius r
    using the Midpoint algorithm.
    """
    octant_points = midpoint_circle_octant(r)
    all_points = set() # Use a set to avoid duplicates

    for x, y in octant_points:
        symmetric = get_symmetric_points(x, y)
        for sx, sy in symmetric:
            # Translate points relative to the center (cx, cy)
            all_points.add((cx + sx, cy + sy))

    return list(all_points)

# --- Example Usage ---
radius = 5
center_x = 0
center_y = 0

# Calculate points for the second octant
octant2_points_midpoint = midpoint_circle_octant(radius)
print(f"Points in the second octant (Midpoint, R={radius}): {octant2_points_midpoint}")

# Calculate all points for the circle
circle_points_midpoint = get_all_midpoint_circle_points(center_x, center_y, radius)
print(f"\nTotal unique points for the circle (Midpoint): {len(circle_points_midpoint)}")
# print(f"All points: {sorted(list(circle_points_midpoint))}") # Uncomment to see all points

# --- Optional: Plotting with Matplotlib ---
def plot_circle(points, title_suffix=""):
    if not points:
        print("No points to plot.")
```

```
            return

    x_coords, y_coords = zip(*points) # Unzip points into x
        and y lists

    plt.figure(figsize=(6, 6))
    plt.scatter(x_coords, y_coords, color='green', s=10) #
        Plot points (green)
    plt.axhline(0, color='grey', lw=0.5)
    plt.axvline(0, color='grey', lw=0.5)
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.gca().set_aspect('equal', adjustable='box') # Ensure
        aspect ratio is equal
    plt.title(f"Midpoint Circle Points{title_suffix}
        (R={radius}, Center=({center_x},{center_y}))")
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    # Set limits slightly larger than radius
    lim = radius + 1
    plt.xlim(-lim, lim)
    plt.ylim(-lim, lim)
    plt.show()

# Plot the calculated points
# plot_circle(circle_points_midpoint, title_suffix=" -
    Midpoint") # Requires matplotlib
```

7. Summary

Key Takeaways

- The Midpoint Circle Algorithm is another highly efficient, integer-based method for rasterizing circles.
- It explicitly uses the midpoint between candidate pixels to make decisions based on the circle equation $f(x, y) = x^2 + y^2 - R^2$.
- The resulting pixel coordinates and the update logic (using integer parameters) are often identical to those derived from Bresenham's approach.
- Like Bresenham's, it relies heavily on 8-way symmetry to minimize calculations.