Bresenham’s Circle Algorithm

Kurt Rosenfeld

The purpose of Bresenham’s circle algorithm is to generate the set of points that approximate a circle on a pixel-based display. We generate the points in the first octant of the circle and then use symmetry to generate the other seven octants. We start at \((r, 0)\). This pixel is chosen trivially. Next we need to find the pixel for the \(y = 1\) row. We must make a decision between two candidate points.

1 Decision Rule

Assume that we having just drawn the pixel at \((x_p, y_p)\). When drawing the first octant of a circle, the two candidates for the next point are as follows:

\[
\text{northwestNeighbor} = (x_p - 1, y_p + 1) \quad (1)
\]
\[
\text{northNeighbor} = (x_p, y_p + 1) \quad (2)
\]

We check the midpoint of the two candidate points to see whether the midpoint is inside or outside the circle. If the midpoint is inside the circle, we go north. If the midpoint is outside the circle, we go northwest.

A variable, \(d\), is used to represent the relationship of the midpoint of the two candidate points to the ideal circle point for that row. Updating \(d\) is done using an incremental trick. The trick is that if we have the old \(d\) value, for the midpoint to the left of \((x_p, y_p)\), after choosing which candidate pixel we will draw, we can calculate the new \(d\) value for the midpoint to the left of the candidate pixel that we chose. We have one formula for incrementally updating \(d\) in the case where we go north and another formula for incrementally updating \(d\) in case we go northwest. The incremental update rules come from expanding the \(d\) formula for the old midpoint, and then observing that both of the new midpoints, when their \(d\) values are formulated in terms
of the previously drawn point, share some terms with the formula for the old midpoint. When subtracting the formula for the old midpoint from the formula for the new midpoint, the common quadratic terms cancel, leaving simple linear update rules to be evaluated in each iteration of the loop.

2 Incremental Update Rules

The previously drawn point is \((x_p, y_p)\). The midpoint to the left of \((x_p, y_p)\) is \((x_p - 0.5, y_p)\). We define

\[
d(x, y) = x^2 + y^2 - r^2,
\]

and therefore

\[
d(x_p - 0.5, y_p) = x_p^2 - x_p + 0.25 + y_p^2 - r^2
\]

(4)

The candidate point to the north is \((x_p, y_p + 1)\). The candidate point to the northwest is \((x_p - 1, y_p + 1)\). The midpoint between these two points is

\[
\text{candidateMidpoint} = (x_p - 0.5, y_p + 1)
\]

(5)

and by expanding this, we get

\[
d(x_p - 0.5, y_p + 1) = x_p^2 - x_p + 0.25 + y_p^2 + 2y_p + 1 - r^2.
\]

(6)

Now we get the incremental update rule by subtracting the expression for \(d(x_p - 0.5, y_p)\) from the expression for \(d(x_p - 0.5, y_p + 1)\). We are left with \(2y_p + 1\). So in cases where we choose the north neighbor, we have

\[
d_{\text{new}} = d_{\text{old}} + 2y_p + 1.
\]

(7)

Next we obtain the incremental update rule for cases where we choose the northwest neighbor.

\[
d(x_p - 1.5, y_p + 1) = x_p^2 - 3x_p + 2.25 + y_p^2 + 2y_p + 1 - r^2
\]

(8)

Subtracting \(d(x_p - 0.5, y_p)\) from \(d(x_p - 1.5, y_p + 1)\), we are left with \(-2x_p + 2 + 2y_p + 1\). Simplifying, we get

\[
d_{\text{new}} = d_{\text{old}} + 2y_p - 2x_p + 3.
\]

(9)
3 Example Code

#include<stdio.h>
#include<math.h>

void plot_8_ways(int x, int y);

int main() {
    int r = 200, ymax, x, y, yp, xp, d;
    ymax = r / sqrt(2);
    x = r; /* start at the bottom of the first octant */
    /* d measures whether the midpoint of two pixel locations is inside or outside the ideal circle. positive means outside */
    d = -1*r; /* to be precise, this should be r^2 - (r - 0.5)^2 */
    xp = r; yp = -1; /* these hold the old values across iterations */
    for(y=0; y<=ymax; y++) {
        plot_8_ways(x,y);
        if (d > 0) { /* midpoint is outside circle, go NW */
            x--;
            d += (2*yp - 2*xp + 3);
        }
        else { /* midpoint is inside circle, go N */
            d += (2*yp + 1);
        }
        yp = y;
        xp = x;
    }
    return(0);
}

void plot_8_ways(int x, int y) {
    printf("%d %d\n", x, y);
    printf("%d %d\n", -1*x, y);
    printf("%d %d\n", x, -1*y);
    printf("%d %d\n", -1*x, -1*y);
    printf("%d %d\n", y, x);
    printf("%d %d\n", -1*y, x);
    printf("%d %d\n", y, -1*x);
}
printf("%d %d\n", -1*y, -1*x);