Bresenham’s algorithms are fast and efficient techniques for rasterizing lines and circles. Of particular importance to you are the fact that they involve only integers (no floating point), no rounding, no division, and only multiplication by the constants 2 and 4 (why is that so helpful?).

We start with the simplifying assumption that we are drawing a line with a slope between 0 and 1 (how can you generalize this to all lines?). In that case, Bresenham exploits the fact that once we have decided to draw a pixel at \((r,q)\), the only two possible next pixels are \((r+1,q)\) and \((r+1,q+1)\). Thus, we only need to decide which of the two the true line is closest to when it crosses the \(r+1\) vertical line. Referring to the above figure, we only need to know whether \(s\) or \(t\) is larger. More specifically, we check the sign of \((s-t)\).

Since the line crosses the \(r+1\) vertical at \(y = \frac{dy}{dx} \cdot (r+1)\), then \(s = \frac{dy}{dx} \cdot (r+1) - q\) and \(t = q + 1 - \frac{dy}{dx} \cdot (r+1)\). Therefore, \(s - t = 2 \cdot \frac{dy}{dx} \cdot (r+1) - 2q - 1\). Also,

\[ dx(s-t) = 2(r \cdot dy - q \cdot dx) + 2dy - dx. \]

Since \(dx\) is defined to be positive, \(dx(s-t)\) is also positive whenever \((s-t)\) is positive, so we can just as easily use that as our test condition. If we define \(d_i = dx(s-t)\) and replace \(r\) and \(q\) with instances of \(x\) and \(y\), we get:

\[ d_i = 2x_i dy - 2y_{i-1} dx + 2dy - dx \]

for the previous point and

\[ d_{i+1} = 2x_i dy - 2y_i dx + 2dy - dx \]  \[ \text{[1]} \]

for the current point. Thus we can always calculate the next \(d_i\) from the previous:

\[ d_{i+1} = d_i + 2dy - 2dx(y_i - y_{i-1}) \]

and since \(x_i - x_{i-1} = 1\) by definition:

\[ d_{i+1} = d_i + 2dy - 2dx(y_i - y_{i-1}) \]

If \(d_i \geq 0\) then the upper pixel is chosen and therefore we know \(y_i = y_{i-1} + 1\) so

\[ d_{i+1} = d_i + 2dy - 2dx \]

Otherwise the lower pixel is chosen and then \(y_i = y_{i-1}\) so

\[ d_{i+1} = d_i + 2dy \]

We only need to initialize \(d_i\), then, for this algorithm to work. The first \(d_i\) should be independent of the initial values, and depend only on \(dx\) and \(dy\), so if we just insert the point \((0,0)\), for simplicity, into equation [1], we get \(d = 2dy - dx\). The pseudocode on the next page implements this algorithm.
Procedure Bresenham_Line(x1,y1,x2,y2) { /*this only works when the slope is between 0 and 1*/

dx = abs(x2 – x1)
dy = abs(y2 – y1)
d = 2 * dy – dx
incr1 = 2 * dy /* constant used for increment if d < 0 */
incr2 = 2 * (dy – dx) /* constant used for increment if d >= 0 */
if (x1 > x2) {
    x = x2
    y = y2
    xend = x1
} /*start at end with smaller x */
else {
    x = x1
    y = y1
    xend = x2
}
draw_pixel(x,y)
while (x < xend) {
    x = x + 1
    if (d < 0)
        d = d + incr1
    else {
        y = y + 1
        d = d + incr2
    }
draw_pixel(x,y)
}
For the circle algorithm, Bresenham uses the same simplifications, applying the equation for a circle, however, instead of the equation for a line. We can further simplify this algorithm by observing that if we know one point on the circle, we can calculate 7 others, as so:

\[
\begin{align*}
(x,y) & \quad (-x,y) \\
(x,-y) & \quad (-y,x) \\
(-x,-y) & \quad (-y,-x) \\
(y,x) & \quad (y,-x)
\end{align*}
\]

(note that this assumes the circle is centered at (0,0), but is easily generalized). Because of this, we need only compute the pixels for 1/8 of the circle (45°).

I’m not going to derive the circle algorithm. You can find it in any good graphics text, and even find several nice descriptions on the web.

Procedure circle(radius) { /* assumes circle is centered at 0,0 – you need to generalize */

\[
\begin{align*}
x & = 0 \\
y & = \text{radius} \\
d & = 3 - 2 \times \text{radius}
\end{align*}
\]

while (x < y) {

\[
\begin{align*}
\text{circle_points}(x,y) & \quad /*draw all eight points derived from this one point*/ \\
\text{if} & \quad (d < 0) \\
\text{d} & = d + 4 \times x + 6 \\
\text{else} & \quad \{ \\
\text{d} & = d + 4 \times (x - y) + 10 \\
y & = y - 1 \\
\}\n\}
\]

\[
\begin{align*}
x & = x + 1 \\
\text{if} & \quad (x == y) \\
\text{circle_points}(x,y)
\}
\]

}