

The Bresenham Line Algorithm

Tom Carter

Computer Science
CSU Stanislaus

tom@csustan.csustan.edu

<http://csustan.csustan.edu/~tom/Lecture-Notes/Graphics/Bresenham-Line/Bresenham-Line.pdf>

October 8, 2014

<http://csustan.csustan.edu/~tom>



Our general topics:

Drawing Lines in a Raster	3
The Bresenham Line Algorithm (simple form)	6
The Bresenham Line Algorithm (all together)	15
Some Additional Notes	19

Drawing Lines in a Raster

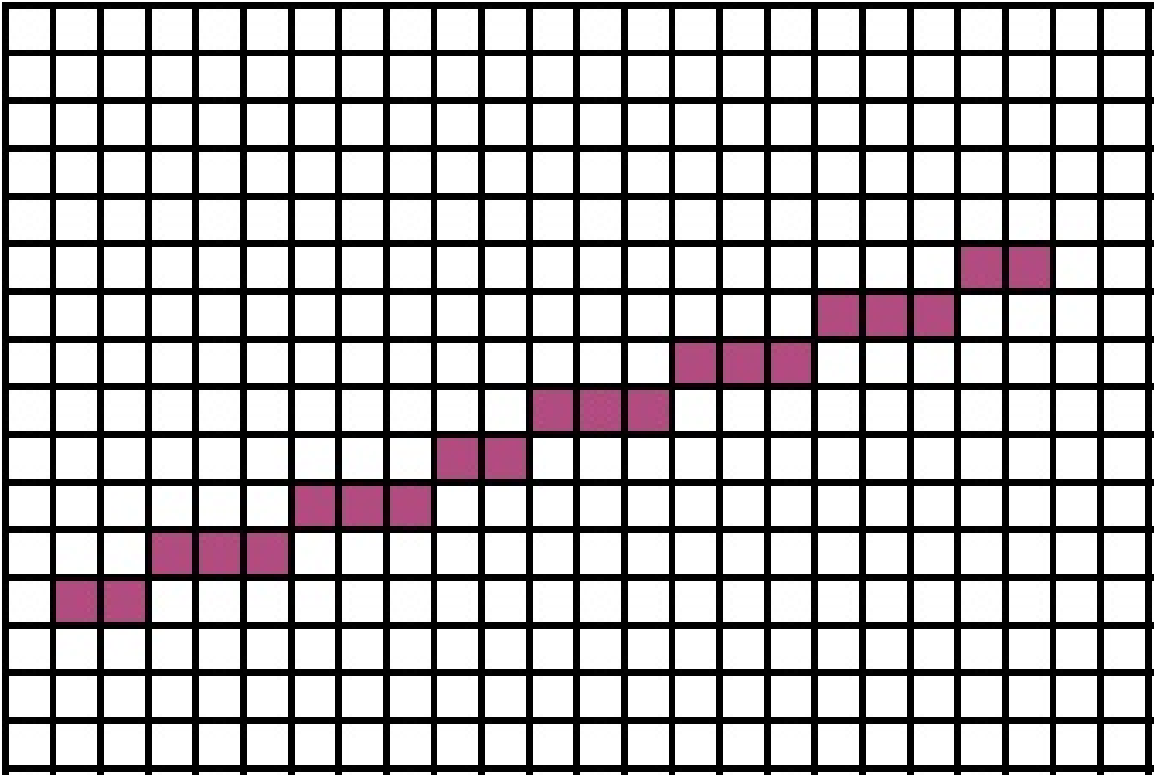


One of the most fundamental actions in computer graphics is drawing a (straight) line on a raster device. An optimized algorithm for drawing such a line is the *Bresenham Line Drawing Algorithm*. We want the algorithm to be as fast as possible, because in practice such an algorithm will be used a lot. We'll walk our way through a derivation of the algorithm.

The algorithm was originally published as: Jack E. Bresenham, *Algorithm for Computer Control of a Digital Plotter*, IBM Systems Journal, 4(1):25-30, 1965.

The *raster* is a two-dimensional array of pixels. We will assume we want to draw a line from one pixel to another in our raster $((X_0, Y_0) \rightarrow (X_1, Y_1))$ by turning on optimal pixels along the line connecting the end points.

Here's an example of what this will look like:



A line in a raster

We'll make some simplifying assumptions in the derivation (and pick up some of them later ...).

1. Our pixel coordinates go from left to right, and bottom to top, the way they do in the first quadrant in mathematics.
2. $X_0 < X_1$, and $Y_0 \leq Y_1$.
3. The slope of the line will be between 0 and 1 (i.e., $0 \leq m \leq 1$), so we will be drawing the line from lower left to upper right; and, for any X value between X_0 and X_1 , there will be exactly one pixel on. (Look at the example above ...)
4. Our pixels are either off or on (i.e., no gray-scale ...).

The Bresenham Line Algorithm (simple form) ←

With the assumptions we have made, the most straightforward algorithm is very simple:

```
For X = X_0 to X_1 step 1
    determine Y value
    SetPixel(X, Y)
Next X
```

Thus, if we can figure out a fast way to determine the Y value to turn on, we will be done.

We'll do this in an iterative fashion. As we step along in the X direction, we can see that either Y stays the same, or Y increases by 1. All we really need to do is figure out when to increment Y.

Algorithm, version 2:

```
X = X_0
Y = Y_0
SetPixel(X, Y)
While (X <= _X1)
    X = X + 1
    If we should increment Y
        Y = Y + 1
    SetPixel(X, Y)
EndWhile
```

Assuming our line is given by $y = mx + b$, what we are doing is setting $Y = \text{round}(mx + b)$. We will do this iteratively. Each (unit) step of x will increase y by m . If we let *fraction* be the amount y has increased since the last time we incremented, then we want to increment Y when $\text{fraction} \geq \frac{1}{2}$.

So our algorithm looks like:

```
X = X_0
Y = Y_0
fraction = start_value
fraction_increment = (Y_1 - Y_0) / (X_1 - X_0)
SetPixel(X, Y)
While (X <= X_1)
    X = X + 1
    fraction = fraction + fraction_increment
    If fraction >= 1/2
        Y = Y + 1
        fraction = fraction - 1
    SetPixel(X, Y)
EndWhile
```

There are some aspects of this that are slower than we want. In particular, we really don't want to be doing any floating point arithmetic, or floating point comparisons.

First, we have

$$m = \frac{Y_1 - Y_0}{X_1 - X_0}$$

To get rid of the fraction (so we can work with integers), we'll multiply by $X_1 - X_0$.

To get rid of the comparison to $\frac{1}{2}$, we'll multiply by 2.

Putting these two together, we will use

$$\begin{aligned} \textit{fraction_increment} &= m * 2 * (X_1 - X_0) \\ &= \frac{Y_1 - Y_0}{X_1 - X_0} * 2 * (X_1 - X_0) \\ &= 2 * (Y_1 - Y_0) \end{aligned}$$

Also, we generally would prefer to compare to 0 rather than compare to 1. What this all means is that our algorithm will now be:

```
X = X0
Y = Y0
fraction = start_value
fraction_increment = 2 * (Y_1 - Y_0)
SetPixel(X, Y)
While (X <= X1)
    X = X + 1
    fraction = fraction + fraction_increment
    If fraction >= 0
        Y = Y + 1
        fraction = fraction - 2 * (X_1 - X_0)
    SetPixel(X, Y)
EndWhile
```

The one other thing we need to do is figure out *start_value*.

What we really have is that *fraction* is a *decision variable*. We have started with our decision variable (after k steps in the X direction) as

$$\begin{aligned} DV(k) &= ((Y_0 + (k + 1) * m) \bmod 1) - \frac{1}{2} \\ &= (((k + 1) * m) \bmod 1) - \frac{1}{2} \end{aligned}$$

The $+ 1$ reflects the fact that we want to start on the right hand side of the first pixel (in the X direction), and then move forward at each step to the right side of the next pixel. The $-\frac{1}{2}$ normalizes us so that our decision variable will cause us to increment Y when we have moved up $\frac{1}{2}$ a pixel (or more). That is, we want our decision variable to be between $-\frac{1}{2}$ and $\frac{1}{2}$.

When we do the conversion so that we are working in integers, and compare with 0, our decision variable becomes:

$$\begin{aligned}
 fraction(k) &= (((k + 1) * m * 2 * (X_1 - X_0)) \\
 &\quad \text{mod } 2 * (X_1 - X_0)) - \frac{1}{2} * 2 * (X_1 - X_0) \\
 &= (((k + 1) * \frac{Y_1 - Y_0}{X_1 - X_0} * 2 * (X_1 - X_0)) \\
 &\quad \text{mod } 2 * (X_1 - X_0)) - \frac{1}{2} * 2 * (X_1 - X_0) \\
 &= (((k + 1) * (Y_1 - Y_0) * 2) \\
 &\quad \text{mod } (2 * (X_1 - X_0))) - (X_1 - X_0) \\
 &= ((k * 2 * (Y_1 - Y_0) + 2 * (Y_1 - Y_0)) \\
 &\quad \text{mod } (2 * (X_1 - X_0))) - (X_1 - X_0)
 \end{aligned}$$

Thus, when $k = 0$, we have

$$\begin{aligned} \mathit{fraction}(0) &= ((0 * 2 * (Y_1 - Y_0) + 2 * (Y_1 - Y_0)) \\ &\quad \text{mod } (2 * (X_1 - X_0)) - (X_1 - X_0)) \\ &= ((2 * (Y_1 - Y_0)) \\ &\quad \text{mod } (2 * (X_1 - X_0)) - (X_1 - X_0)) \\ &= 2 * (Y_1 - Y_0) - (X_1 - X_0) \end{aligned}$$

The Bresenham Line Algorithm (all together) ←

Now we can finalize everything. If we want to deal with positive or negative slope lines, we just adjust the step size to be $+1$ or -1 . If we want to deal with slopes greater than 1 (or less than -1), we just interchange X and Y , and do our step increment (or decrement) using Y instead of X , etc.

So, our final algorithm looks like:

```
void lineBresenham(int x0, int y0, int x1, int y1)
{
    int dx, dy;
    int stepx, stepy;

    dx = x1 - x0;
    dy = y1 - y0;

    if (dy < 0) { dy = -dy;   stepy = -1; } else { stepy = 1; }
    if (dx < 0) { dx = -dx;   stepx = -1; } else { stepx = 1; }
    dy <<= 1;                /* dy is now 2*dy */
    dx <<= 1;                /* dx is now 2*dx */
    if ((0 <= x0) && (x0 < RDim) && (0 <= y0) && (y0 < RDim))
theRaster[x0][y0] = 1;
```



```

if (dx > dy) {
    int fraction = dy - (dx >> 1);
    while (x0 != x1) {
        x0 += stepx;
        if (fraction >= 0) {
            y0 += stepy;
            fraction -= dx;
        }
        fraction += dy;
        if ((0 <= x0) && (x0 < RDim) && (0 <= y0) && (y0 < RDim))
            theRaster[x0][y0] = 1;
    }
} else {
    int fraction = dx - (dy >> 1);
    while (y0 != y1) {
        if (fraction >= 0) {

```

```
        x0 += stepx;
        fraction -= dy;
    }
    y0 += stepy;
    fraction += dx;
    if ((0 <= x0) && (x0 < RDim) && (0 <= y0) && (y0 < RDim))
        theRaster[x0][y0] = 1;
}
}
```

Some Additional Notes



There are other approaches to deriving the Bresenham Line Algorithm. Various parts are the same, but some details are presented differently.

With the assumptions above, we want to choose whether to keep the Y value the same or to move Y up a step when we take the next X step. We can make that decision based on the difference (“distance”) of those two pixels from the “true value” along the line.

We can let y be the “true value” along the line, and $Y(k)$ be the Y value we are at (after step k has been taken).

We can thus look at what would happen if we keep $Y(k + 1) = Y(k)$

$$D_1(k) = y - Y(k)$$

versus setting $Y(k + 1) = Y(k) + 1$:

$$D_2(k) = (Y(k) + 1) - y.$$

We can use these to make our decision:

- If $D_1(k) \leq D_2(k)$, we move to $(X(k) + 1, Y(k))$ (i.e., move right)
- if $D_1(k) > D_2(k)$, we move to $(X(k) + 1, Y(k) + 1)$ (i.e., move right and up)

Said differently (comparing with 0 ...):

- If $D_1(k) - D_2(k) \leq 0$, we move to $(X(k) + 1, Y(k))$ (i.e., move right)
- if $D_1(k) - D_2(k) > 0$, we move to $(X(k) + 1, Y(k) + 1)$ (i.e., move right and up)

We can calculate (using $y = m * x + b$):

$$\begin{aligned} D_1(k) - D_2(k) &= (y - Y(k)) - ((Y(k) + 1) - y) \\ &= y - Y(k) - (Y(k) + 1) + y \\ &= 2 * y - 2 * Y(k) - 1 \\ &= 2 * (m * (X(k) + 1) + b) - 2 * Y(k) - 1 \\ &= 2 * m * (X(k) + 1) - 2 * Y(k) + 2 * b - 1 \end{aligned}$$

Substituting in $b = Y_0 - mX_0$ and $m = \frac{Y_1 - Y_0}{X_1 - X_0} = \frac{\Delta Y}{\Delta X}$, we have

$$\begin{aligned}
 D_1(k) - D_2(k) &= 2m(X(k) + 1) - 2Y(k) + 2b - 1 \\
 &= 2m(X(k) + 1) - 2Y(k) + 2(Y_0 - mX_0) - 1 \\
 &= 2\frac{\Delta Y}{\Delta X}(X(k) + 1) - 2Y(k) + 2(Y_0 - \frac{\Delta Y}{\Delta X}X_0) - 1
 \end{aligned}$$

We don't want to do division (floating point operation), so we let $D(k) = \Delta X(D_1(k) - D_2(k))$, and we have:

$$\begin{aligned}
 D(k) &= \Delta X(D_1(k) - D_2(k)) \\
 &= \Delta X(2\frac{\Delta Y}{\Delta X}(X(k) + 1) - 2Y(k) + 2(Y_0 - \frac{\Delta Y}{\Delta X}X_0) - 1) \\
 &= 2\Delta Y(X(k) + 1) - 2\Delta X * Y(k) + 2\Delta X * Y_0 - 2\Delta Y * X_0 - \Delta X \\
 &= 2\Delta Y * X(k) - 2\Delta X * Y(k) \\
 &\quad + 2\Delta Y + 2\Delta X * Y_0 - 2\Delta Y * X_0 - \Delta X
 \end{aligned}$$

We can write this as:

$$D(k) = 2\Delta Y * X(k) - 2\Delta X * Y(k) + c$$

where $c = 2\Delta Y + 2\Delta X * Y_0 - 2\Delta Y * X_0 - \Delta X$ is a constant, depending only on the input values of (X_0, Y_0) and (X_1, Y_1) .

Note that since we are assuming $X_0 < X_1$, we have $\Delta X > 0$, and so the decision criterion is still the same.

Now all we need to do is calculate $D(k)$, compare with 0, and we know which pixel to select next. We can actually do this iteratively (note: $X(k+1) - X(k) = 1$):

$$\begin{aligned} D(k+1) - D(k) &= (2\Delta Y * X(k+1) - 2\Delta X * Y(k+1) + c) \\ &\quad - (2\Delta Y * X(k) - 2\Delta X * Y(k) + c) \\ &= 2\Delta Y(X(k+1) - X(k)) - 2\Delta X((Y(k+1) - Y(k))) \\ &= 2\Delta Y - 2\Delta X((Y(k+1) - Y(k))) \end{aligned}$$

There are two cases for $(Y(k+1) - Y(k))$. Either $(Y(k+1) = Y(k))$ (if $D(k) \leq 0$), or $(Y(k+1) = Y(k) + 1)$.

Thus, the increment in $D(k)$ is $2\Delta Y$ if $D(k) \leq 0$, or $2\Delta Y - 2\Delta X$ otherwise.

The only other thing we need is to find the initial value, $D(0)$. We can calculate this as

$$\begin{aligned} D(0) &= 2\Delta Y * X(0) - 2\Delta X * Y(0) + c \\ &= 2\Delta Y * X_0 - 2\Delta X * Y_0 + c \\ &= 2\Delta Y * X_0 - 2\Delta X * Y_0 + 2\Delta Y + 2\Delta X * Y_0 - 2\Delta Y * X_0 - \Delta X \\ &= 2\Delta Y - \Delta X \end{aligned}$$

[To top ←](#)