Bresenham Line Algorithm

Choses y representing the pixel center that is closest to the ideal y for the same x. Y can remain the same or increate by 1, the general equation of the line through the endpoints is given by:

$$\frac{y - y_0}{y_1 - y_0} = \frac{x - x_0}{x_1 - x_0}$$

Since we know the column x, the pixel row y is given by rounding.

$$y = \frac{y_1 - y_0}{x_1, x_0}(x - x_0) + y_0$$

The slope formula $\frac{y_1 - y_0}{x_1 - x_0}$ depends only on the endpoint cordinates. The y should be computed by repeatedly adding the slope to x (starting at y_0).

How Computers Draw Lines

Since there are a finite number of points (could be viewed as pixels), we can not use a simple point slope algorithm.

Since we have this issue we need to check if the y should stay the same or move, since in a line the y may remain the same in order to appear as if it is a straight line. Take a mouse pointer for example, when examining the pixels they arent perfectly vertical! They are square points that occasionally appear in succession in order to give the illusion of a line.

Another Example of this maybe minecraft. Look at the example below for a better idea.

```
#
 ##
  ###
   ####
    #####
     ######
      #######
```

In this example the the shape displays as triangular, while having a varying slope.

One Method

$$f(x_0 + 1, y_0 + \frac{1}{2})$$

Case 1 - > 0 : y increments

Case 2 - $<= 0$: y stays the same

Another Method

Find the difference:

$$D = f(x_0 + 1, y_0 + \frac{1}{2}) - f(x_0, y_0)$$

$$= [A(x_0 + 1) + B(y_0 + 1/2) + C] - [Ax_0 + By_0 + C]$$

Simplifying:

$$= [Ax_0 + By_0 + C + A + \frac{1}{2}B] - [Ax_0 + By_0 + C]$$

$$D = A + \frac{1}{2}B$$

Defining A and B and D

$$D = \Delta y - \frac{1}{2}\Delta x$$

$$A = \Delta y$$

$$B = -\Delta x$$

Case 1: - D > 0 : y increments - $(x_0 + 1, y_0 + 1)$ - $\Delta D = \Delta y - \Delta x$

Case 2: - D $<= 0$: y stays the same - $(x_0 + 1, y_0)$

However, we should avoid using fractions in our solution so we should multiply the entire RHS by 2, eliminating the need for the fraction.

$$D = 2\Delta y - \Delta x$$

Fixing this algorithm

You may have noticed that we will have a few potential issues within this algorithm. First and foremost this algorithm will only work for lines with small increasing slopes. We also have no method of implementing a negative slope with the current bounds that I have laid forth.

For addressing the issue of negative slopes, we can simply fix this through implementing an algorithm to check weather y needs to increase or decrease in order to reach the end point (the last point in the line). We can apply the same methodology to x, decrementing it.

Through swapping the values of the x and y corrdinates we can cover steep slopes.