# Breshenham and Gupta Sproull Line Algorithm

Carl Henrik Ek

February 24, 2019

**Abstract**

This document outlines the derivation of the Breshenham and Goupta Sproull line drawing algorithms. The idea underlying Breshenhams line algorithm is to create a decision function which is completely based on integer computations. Its a good example of how to think about making fast code, reduce the computations to integers, split up the algorithm the several cases which can be detected quickly each one being less specific and try to re-use as many computations as we can.

## 1 Breshenham

A line can be written as,

$$y = \frac{\Delta y}{\Delta x}x + c,$$

as we are drawing in screen space both $\Delta x$ and $\Delta y$ are integers. This means that we can re-write the line as an implicit surface such as,

$$f(x, y) = \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot c.$$

This implicit surface specifies a half-plane where every-point on one side will take a negative value and all values on the other will be positive while only points on the line will be zero. We will use this as a test to see which pixel lies on the line.

We will first split lines into 8 separate cases, we will derive the method for the case where $\Delta x$ and $\Delta y$ are both positive and $\Delta x > \Delta y$. Deriving the remaining 7 cases is trivial and works just the same, when you do so remember to use symmetries. As an example the case $\Delta x$ negative $\Delta y$ positive can be made into the latter by flipping start and end-point of x for the line.

Now if we have drawn the pixel $(x_p, y_p)$ as we know that the line moves "quicker" along $x$ than $y$ we know that we should increase $x$ by one, the question is which pixel in $y$ to draw? As we know that the slope is positive, the two pixels we could draw is either $(x + 1, y)$ or $(x + 1, y + 1)$. As we want to draw the pixel that is closest to the line we can check if $(x + 1, y + \frac{1}{2})$ is above or below the line as,

$$\begin{array}{rr} (x_p + 1, y + \frac{1}{2}) \text{ above the line} & \text{draw } (x_p + 1, y_p) \\ (x_p + 1, y + \frac{1}{2}) \text{ below the line} & \text{draw } (x_p + 1, y_p + 1). \end{array} \tag{1}$$

The nice thing about the implicit representation of the line is that it is very easy to test which half-plane the point $(x + 1, y + \frac{1}{2})$ lies in as,

$$\begin{array}{rr} f(x_p + 1, y + \frac{1}{2}) > 0 & \text{draw } (x_p + 1, y_p) \\ f(x_p + 1, y + \frac{1}{2}) < 0 & \text{draw } (x_p + 1, y_p + 1) \\ f(x_p + 1, y + \frac{1}{2}) = 0 & \text{draw random.} \end{array} \tag{2}$$

Now we come to the real speed-up when using this algorithm. We will try to derive a formulation that allows us to re-use the computation from previous decision to compute the next. We will do this for the two possible cases for this line.

### 1.1 $f(x_p + 1, y_p + \frac{1}{2}) < 0$

In this case we have just draw $(x + 1, y)$ so the new test point is $(x + 2, y + \frac{1}{2})$,

$$f(x_p + 2, y_p + \frac{1}{2}) = \Delta y(x_p + 2) - \Delta x(y_p + \frac{1}{2}) + \Delta x \cdot c.$$

When deciding to draw $(x+1, y)$ we had to calculate $f(x_p+1, y+\frac{1}{2})$ so we can reuse this computation. The idea is to formulate an itterative computation as,

$$f(x_p + 2, y_p + \frac{1}{2}) = f(x_p + 1, y + \frac{1}{2}) + \Delta f.$$

Deriving $\Delta f$ is as simple as,

$$\begin{aligned}
\Delta f &= f(x_p + 2, y_p + \frac{1}{2} - f(x_p + 1, y + \frac{1}{2}) && (3)\\
&= \Delta y(x_p + 2) - \Delta x(y_p + \frac{1}{2}) + \Delta x \cdot c - (\Delta y(x_p + 1) - \Delta x(y_p + \frac{1}{2}) + \Delta x \cdot c) && (4)\\
&= \Delta y && (5)
\end{aligned}$$

So rather than re-computing the whole decision function we can just add $\Delta y$ each iteration we do this choice, importantly this value is an integer as well so nothing computationally expensive.

### 1.2 $f(x + 1, y + \frac{1}{2}) > 0$

If we in the previous iteration drew $(x + 1, y + 1)$ the same structure can be used except that we now are new test point is $(x + 2, y + \frac{3}{2})$.

$$\begin{aligned}
\Delta f &= f(x_p + 2, y_p + \frac{3}{2} - f(x_p + 1, y + \frac{1}{2}) && (6)\\
&= \Delta y(x_p + 2) + \Delta x(y_p + \frac{3}{2}) + \Delta x \cdot c - (\Delta y(x_p + 1) + \Delta x(y_p + \frac{1}{2})\Delta x \cdot c) && (7)\\
&= \Delta y - \Delta x && (8)
\end{aligned}$$

### 1.3 First point

We have now written the itterative update for the intermediate steps so the only thing that is left to do is to write the test for the first point on the line $f(x_0 + 1, y_0 + \frac{1}{2})$.

$$f(x_0 + 1, y_0 + \frac{1}{2}) = \Delta y(x_0 + 1) - \Delta x(y_0 + \frac{1}{2}) + \Delta x \cdot c = \underbrace{\Delta y \cdot x_0 - \Delta x \cdot y_0 + \Delta x \cdot c}_{f(x_0,y_0)} + \Delta y - \frac{1}{2}\Delta x.$$

Now because the first point on the line is clearly on the line this means that $f(x_0, y_0) = 0$ so the first value of the test function is,

$$f(x_0 + 1, y_0 + \frac{1}{2} = \Delta y - \frac{1}{2}\Delta x$$

This is slightly annoying as this means that the first test is no longer necessary an integer due to the fraction of $\Delta x$. However, as we are only interested in the sign of the test function. This means that we can multiply the first test by 2 and everything is in terms of integers.

## 2 Gupta-Sproull

Breshenham algorithm makes a hard decision of which pixel should be drawn. This can, especially in lower resolutions look a bit too blocky and we want to include some anti-aliasing. There are many different algorithms to do so. Probably the most common is Xiaolin Wu's line algorithm, commonly known as WU-lines[1]. What we will look at here is a method that is a bit more correct and also fits very well in with Breshenhams algorithms, in many ways it is an extension of the former to include anti-aliasing. We will just as with Breshenhams algorithm look at the same case where we have a postive slope and the change in $x$ is "faster" than the one in $y$. First if we look at Figure 1 what we want to do is to colour the three closest pixels to the line. Specifically we want to shade them based on the distance they have to the line, this means that we want to colour the pixel below according to $D_{bellow}$ and the one above according to $D_{up}$ and the one in the middle with respect to $D$. So our task is to compute these three distances.

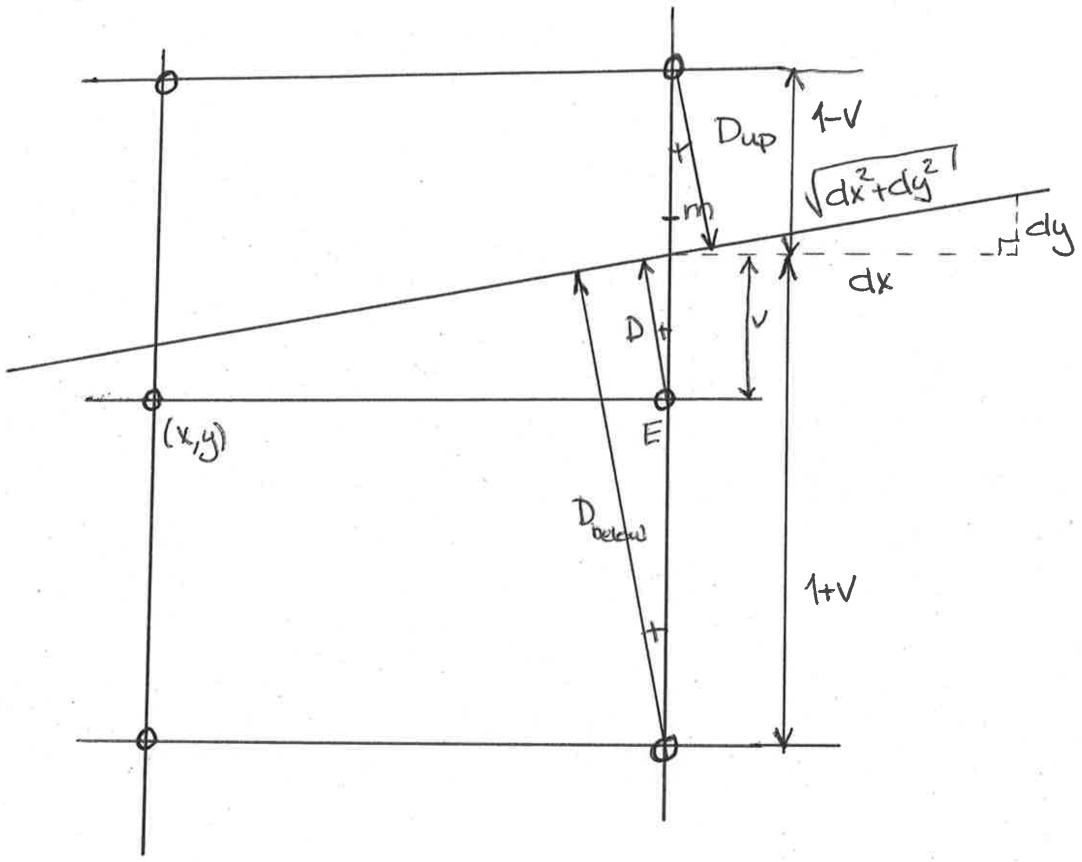

Figure 1: The Gupta-Sproull algorithm for anti-aliased lines

The first thing we want to compute is $v$ the vertical distance from the center pixel to the line. Using the same formulation of the line we had before but including the multiplication by 2 to keep it on integer form.

$$f(x, y) = 2(\Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot c)$$

As we know the implicit surface is 0 on the line we can solve the above to get the $y$ coordinate for the intersection of the line and the vertical between the pixels $y_v$ as,

$$f(x, y) = 2(\Delta y \cdot (x + 1) - \Delta x \cdot y_v + \Delta x \cdot c) = 0 \tag{9}$$

$$y_v = \frac{\Delta y(x + 1) + \Delta x \cdot c}{\Delta x}, \tag{10}$$

this means that $v$ can be computed by removing the current $y$ value,

$$v = y_v - y = \frac{\Delta y(x + 1) + \Delta x \cdot c}{\Delta x} - y.$$

[1]https://en.wikipedia.org/wiki/Xiaolin_Wu%27s_line_algorithm

As we want to make use of the same computations as we have previously done for Breshenham we now want to re-write this in terms of the decision function we had before,

$$\Delta x \cdot v = \Delta y(x+1) + \Delta x \cdot c - \Delta x \cdot y = \frac{1}{2} f(x+1, y)$$

Now we are nearly there, our decision variable was the decision function evaluated at $f(x+1, y+\frac{1}{2})$ lets see if we can re-write the previous expression in terms of this,

$$\begin{aligned} 2\Delta x \cdot v = f(x+1,y) &= 2\Delta y \cdot (x+1) - 2\Delta x \cdot y + 2\Delta x \cdot c && (11)\\ &= 2\Delta y \cdot (x+1) - 2(\Delta x \cdot (y+\frac{1}{2}) - \Delta x \frac{1}{2}) + 2\Delta x \cdot c && (12)\\ &= f(x+1, y+\frac{1}{2}) + \Delta x. && (13) \end{aligned}$$

Now we have written $v$ as a function of our previous decision variable and as can be seen it is something that we get for free adding just a single addition to our code. We are now ready to compute the distance from each of the three pixels to the line. We will first calculate $D$ and we will do so through two similar triangles,

$$\begin{aligned} \frac{D}{v} &= \frac{\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}} && (14)\\ D &= \frac{v \cdot \Delta x}{\sqrt{\Delta x^2 + \Delta y^2}} && (15)\\ &= \frac{f(x+1, y+\frac{1}{2}) + \Delta x}{2(\sqrt{\Delta x^2 + \Delta y^2})} && (16)\\ &= A \cdot (f(x+1, y+\frac{1}{2}) + \Delta x). && (17) \end{aligned}$$

The value $A$ is a constant for each line so only needs to be computed once for each line. Now we can use exactly the same relationship to compute the two remaining distances,

$$\begin{aligned} D_{up} &= \frac{(1-v)\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}} && (18)\\ &= \frac{\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}} - \frac{v\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}} && (19)\\ &= B - A \cdot (f(x+1, y+\frac{1}{2}) + \Delta x) && (20)\\ D_{below} &= \frac{(1+v)\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}} && (21)\\ &= \frac{\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}} + \frac{v\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}} && (22)\\ &= B + A \cdot (f(x+1, y+\frac{1}{2}) + \Delta x && (23) \end{aligned}$$

So now we have all the distances needed and can come up with a shading such that the further away from the line a pixel is the less bright it will be. Now the last part of the algorithm includes $A$ and $B$ which clearly are fractions smaller than 1. However, we can still keep the whole thing as integers by scaling all the numbers as we know the upper bound on the distances.