# BRESHENHAM'S ALGORITHM

Kenneth I. Joy
Visualization and Graphics Research Group
Department of Computer Science
University of California, Davis

## Overview

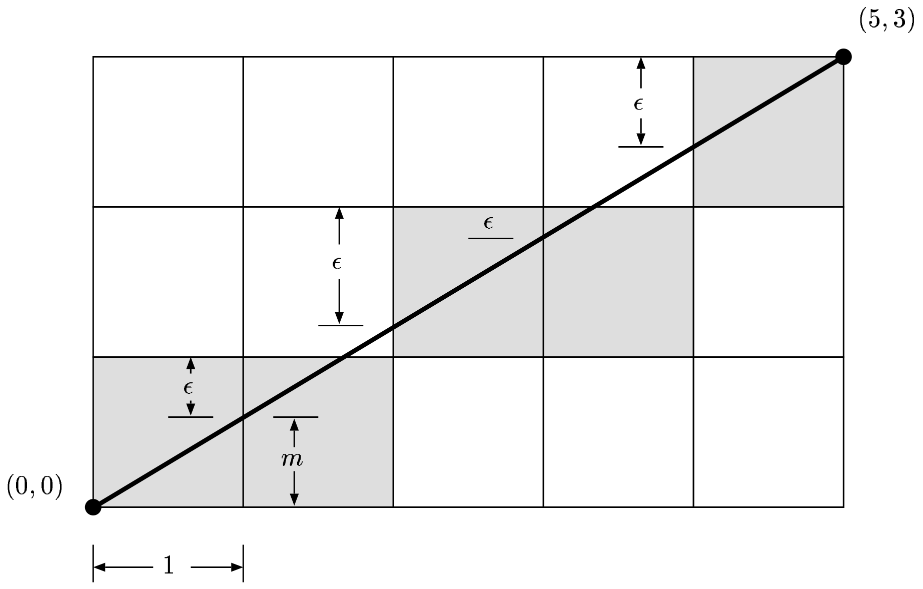
The basic "line drawing" algorithm used in computer graphics is Bresenham's Algorithm. This algorithm was developed to draw lines on digital plotters, but has found wide-spread usage in computer graphics. The algorithm is fast – it can be implemented with integer calculations only – and very simple to describe.

---

## Bresenham's Algorithm

Consider a line with initial point $(x_1, y_1)$ and terminal point $(x_2, y_2)$ in device space. If $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$, we define the *driving axis* (*DA*) to be the $x$-axis if $|\Delta x| \geq |\Delta y|$, and the $y$-axis if $|\Delta y| > |\Delta x|$. The *DA* is used as the "axis of control" for the algorithm and is the axis of maximum movement. Within the main loop of the algorithm, the coordinate corresponding to the *DA* is incremented by one unit. The coordinate corresponding to the other axis (usually denoted the *passive axis* or *PA*) is only incremented as needed.

The best way to describe Bresenham's algorithm is to work through an example. Consider the following example, in which we wish to draw a line from $(0, 0)$ to $(5, 3)$ in device space.



Bresenham's algorithm begins with the point $(0, 0)$ and "illuminates" that pixel. Since $x$ is the *DA* in this example, it then increments the $x$ coordinate by one. Rather than keeping track of the $y$ coordinate (which increases by $m = \Delta y/\Delta x$, each time the $x$ increases by one), the algorithm keeps an error bound $\epsilon$ at each stage, which represents the negative of the distance from the point where the line exits the pixel to the top edge of the pixel (see the figure). This value is first set to $m - 1$, and is incremented by $m$ each time the $x$ coordinate is incremented by one. If $\epsilon$ becomes greater than zero, we know that the line has moved upwards one pixel, and that we must increment our $y$ coordinate and readjust the error to represent the distance from the top of the new pixel – which is done by subtracting one from $\epsilon$.

The reader can examine the above illustration and the following table to see the complete operation of the algorithm on this example.

| $(x, y)$ | $\epsilon$ | description |
|---|---|---|
| $(0,0)$ | -0.4 | illuminate pixel $(0,0)$ |
| | 0.2 | increment $\epsilon$ by $0.6$ |
| $(1,0)$ | | increment $x$ by 1 |
| $(1,0)$ | 0.2 | illuminate pixel $(1,0)$ |
| | | since $\epsilon > 0$ |
| $(1,1)$ | | increment $y$ by 1 |
| | -0.8 | decrement $\epsilon$ by 1 |
| | -0.2 | increment $\epsilon$ by $0.6$ |
| $(2,1)$ | | increment $x$ by 1 |
| $(2,1)$ | -0.2 | illuminate pixel $(2,1)$ |
| | 0.4 | increment $\epsilon$ by $0.6$ |
| $(3,1)$ | | increment $x$ by 1 |
| $(3,1)$ | 0.4 | illuminate pixel $(3,1)$ |
| | | since $\epsilon > 0$ |
| $(3,2)$ | | increment $y$ by 1 |
| | -0.6 | decrement $\epsilon$ by 1 |
| | 0.0 | increment $\epsilon$ by $0.6$ |
| $(4,2)$ | | increment $x$ by 1 |
| $(4,2)$ | 0.0 | illuminate pixel $(4,2)$ |

Assuming that the *DA* is the $x$-axis, an algorithmic description of Bresenham's algorithm is as follows:

**Bresenham's Algorithm**

```
The points $(x_1, y_1)$ and $(x_2, y_2)$ are assumed
    not equal and integer valued.
$\epsilon$ is assumed to be real.

Let $\Delta x = x_2 - x_1$
Let $\Delta y = y_2 - y_1$
Let $m = \frac{\Delta y}{\Delta x}$
Let $j = y_1$
Let $\epsilon = m - 1$

for $i = x_1$ to $x_2 - 1$
    illuminate $(i, j)$
    if $(\epsilon \geq 0)$
        $j += 1$
        $\epsilon -= 1.0$
        end if
      $i += 1$
      $\epsilon += m$
      next i

  finish
```

All algorithms presented in these notes assume that $\Delta x$ and $\Delta y$ are positive. If this is not the case, the algorithm is essentially the same except for the following:

- $\epsilon$ is calculated using $|\frac{\Delta y}{\Delta x}|$.
- $x$ and $y$ are decremented (instead of incremented) by one if the sign of $\Delta x$ or $\Delta y$ is less than zero, respectively.

---

**The Integer Bresenham's Algorithm**

Bresenham's Algorithm, as given in the previous section, requires the use of floating point arithmetic to calculate the slope of the line and to evaluate the error term. We note that $\epsilon$ is initialized to

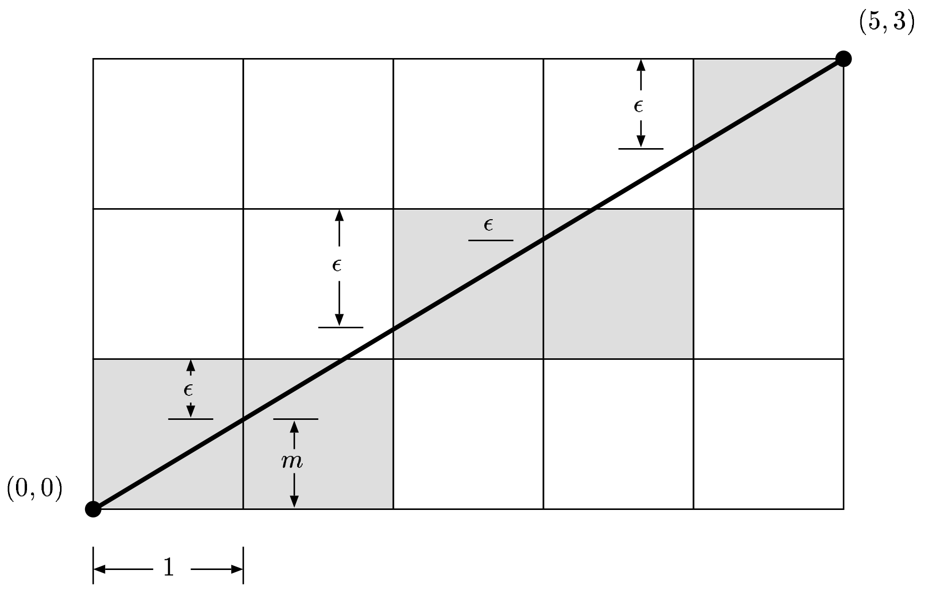
$$\epsilon = \frac{\Delta y}{\Delta x} - 1$$

and is incremented by $\frac{\Delta y}{\Delta x}$ at each step. Since both $\Delta y$ and $\Delta x$ are integer quantities, we can convert to an all integer format by multiplying the operations through by $\Delta x$. That is, we will consider the integer quantity $\bar{\epsilon}$, where $\bar{\epsilon}$ is initialized to

$$\bar{\epsilon} = \Delta x \epsilon = \Delta y - \Delta x$$

and we will increment $\bar{\epsilon}$ by $\Delta y$ at each step, and decrement it by $\Delta x$ when $\epsilon$ becomes positive.

Our example from the section above, which attempts to draw a line from $(0, 0)$ to $(5, 3)$ in screen space, can now be converted to an integer algorithm. Consider the figure and table below, where $\Delta x = 5$, $\Delta y = 3$ and $\bar{\epsilon} = \Delta x - \Delta y = -2$.



| $(x,y)$ | $\bar{\epsilon}$ | description |
|---|---|---|
| $(0,0)$ | -2 | illuminate pixel $(0,0)$ |
| | 1 | increment $\epsilon$ by $\Delta y$ |
| $(1,0)$ | | increment $x$ by 1 |
| $(1,0)$ | 1 | illuminate pixel $(1,0)$ |
| | | since $\bar{\epsilon} > 0$ |
| $(1,1)$ | | increment $y$ by 1 |
| | -4 | decrement $\bar{\epsilon}$ by 5 |
| | -1 | increment $\epsilon$ by $\Delta y$ |
| $(2,1)$ | | increment $x$ by 1 |
| $(2,1)$ | -1 | illuminate pixel $(2,1)$ |
| | 2 | increment $\epsilon$ by $\Delta y$ |
| $(3,1)$ | | increment $x$ by 1 |
| $(3,1)$ | 2 | illuminate pixel $(3,1)$ |
| | | since $\bar{\epsilon} > 0$ |
| $(3,2)$ | | increment $y$ by 1 |
| | -3 | decrement $\bar{\epsilon}$ by 5 |
| | 0 | increment $\epsilon$ by $\Delta y$ |
| $(4,2)$ | | increment $x$ by 1 |
| $(4,2)$ | 0 | illuminate pixel $(4,2)$ |

Thus the integer version of Bresenham's algorithm is constructed as follows:

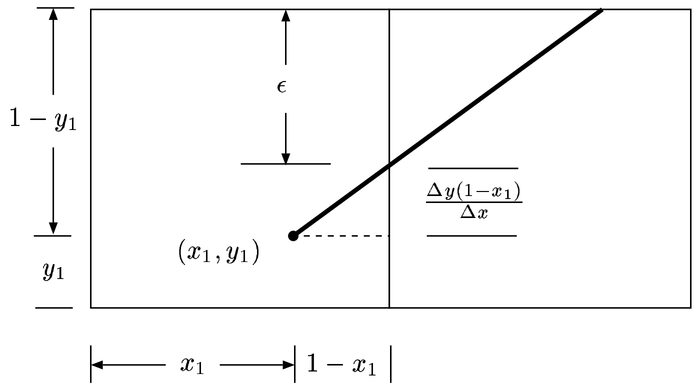
**Bresenham's Algorithm using Integer Arithmetic**

```
The points $(x_1, y_1)$ and $(x_2, y_2)$ are assumed
    not equal and integer valued.
$\bar{\epsilon}$ is assumed to be integer valued.

Let $\Delta x = x_2 - x_1$
Let $\Delta y = y_2 - y_1$
Let $j = y_1$
Let $\bar{\epsilon} = \Delta y - \Delta x$

for $i = x_1$ to $x_2 - 1$
    illuminate $(i, j)$
    if $(\bar{\epsilon} \geq 0)$
        $j += 1$
        $\bar{\epsilon} -= \Delta x$
        end if
    $i += 1$
    $\bar{\epsilon} += \Delta y$
    next i

finish
```

## Bresenham's Algorithm for Lines with Arbitrary Endpoints

Bresenham's algorithm, as described in the sections above, is limited by the fact that the lines to be drawn have endpoints with integer coordinates. In this section, we consider a version of Bresenham's algorithm for lines that have endpoints with real coordinates. The only problem to overcome is the initial setting of the error. Once this is done, the algorithm proceeds as before.

Consider a line with initial point $(x_1, y_1)$ and terminal point $(x_2, y_2)$ in device space, where we assume the points are not the same. To calculate the correct $\epsilon$ in this case, we refer to the following figure.

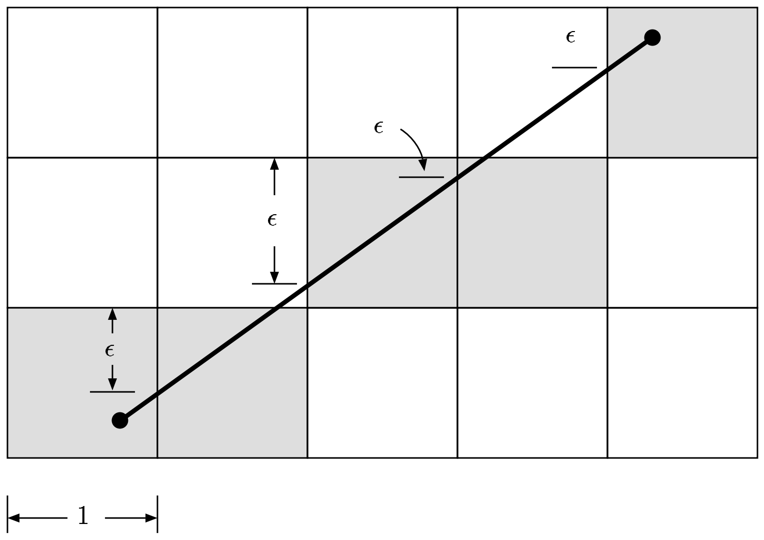


If we consider the lower-left-hand corner of the grid to be $(0, 0)$, then it is easily seen that the initial $\epsilon$ is

$$\epsilon \ = \ -\left(1 - y_1 - \frac{\Delta y(1 - x_1)}{\Delta x}\right)$$

We now utilize this to modify Bresenham's algorithm accordingly.

Consider the following example, which attempts to draw a line from $(0.75, 0.125)$ to $(4.3, 2.8)$ in screen space,



Bresenham's algorithm calculates the new $\epsilon$ as

$$\begin{aligned}\epsilon &= -\left(1 - 0.125 - \frac{2.55(1-.75)}{3.55}\right)\\ &= -\left(0.75 - \frac{.6375}{3.55}\right)\\ &= -\left(0.75 - 0.17958\right)\\ &= -0.57042\end{aligned}$$

The algorithm begins with the point $(0,0) = (\lfloor 0.75 \rfloor, \lfloor 0.125 \rfloor)$ and then proceeds in exactly the same way as Bresenham's algorithm for lines having endpoints with integer coordinates.

The reader can examine the above illustration and the following table to see the complete operation of the algorithm on this example. In this case $m = \frac{2.55}{3.55} = .71831$, and the lower left corner of the grid is $(\lfloor x_1 \rfloor, \lfloor x_2 \rfloor)$.

| $(x, y)$ | $\epsilon$ | description |
|---|---|---|
| $(0,0)$ | -0.57042 | illuminate pixel $(0,0)$ |
| | 0.14789 | increment $\epsilon$ by 0.71831 |
| $(1,0)$ | | increment $x$ by 1 |
| $(1,0)$ | 0.14789 | illuminate pixel $(1,0)$ |
| | | since $\epsilon > 0$ |
| $(1,1)$ | | increment $y$ by 1 |
| | -0.85211 | decrement $\epsilon$ by 1 |
| | -0.1338 | increment $\epsilon$ by 0.71831 |
| $(2,1)$ | | increment $x$ by 1 |
| $(2,1)$ | -0.1338 | illuminate pixel $(2,1)$ |
| | 0.58451 | increment $\epsilon$ by 0.71831 |
| $(3,1)$ | | increment $x$ by 1 |
| $(3,1)$ | 0.58451 | illuminate pixel $(3,1)$ |
| | | since $\epsilon > 0$ |
| $(3,2)$ | | increment $y$ by 1 |
| | -0.41549 | decrement $\epsilon$ by 1 |
| | 0.30282 | increment $\epsilon$ by 0.71831 |
| $(4,2)$ | | increment $x$ by 1 |
| $(4,2)$ | 0.30282 | illuminate pixel $(4,2)$ |

Assuming that the *DA* is the $x$-axis, the algorithmic description of Bresenham's algorithm for lines with arbitrary endpoints is as follows:

**Bresenham's Algorithm**

```
The points $(x_1, y_1)$ and $(x_2, y_2)$ are assumed not equal
    and have arbitrary real coordinates
$\epsilon$ is assumed to be real.

Let $\Delta x = x_2 - x_1$
Let $\Delta y = y_2 - y_1$
Let $m = \frac{\Delta y}{\Delta x}$
Let $i_1 = \lfloor x_1 \rfloor$
Let $j = \lfloor y_1 \rfloor$
Let $i_2 = \lfloor x_2 \rfloor$
Let $\epsilon = -(1 - (y_1 - j) - \frac{\Delta y(1-(x_1-i_1))}{\Delta x})$
for $i = i_1$ to $i_2$
    illuminate $(i, j)$
    if $(\epsilon \geq 0)$
        $j += 1$
        $\epsilon -= 1.0$
        end if
    $i += 1$
    $\epsilon += m$
    next i

finish
```

**The Integer Bresenham's Algorithm for Lines with Arbitrary Endpoints**

Bresenham's Algorithm, as given in the previous section, was adapted to lines that have endpoints with arbitrary real coordinates. This algorithm again requires the use of floating point arithmetic to calculate the slope of the line and to evaluate the error term. We note that $\epsilon$ is initialized to

$$\epsilon = -\left(1 - (y_1 - \lfloor y_1 \rfloor) - \frac{\Delta y(1 - (x_1 - \lfloor x_1 \rfloor))}{\Delta x}\right)$$

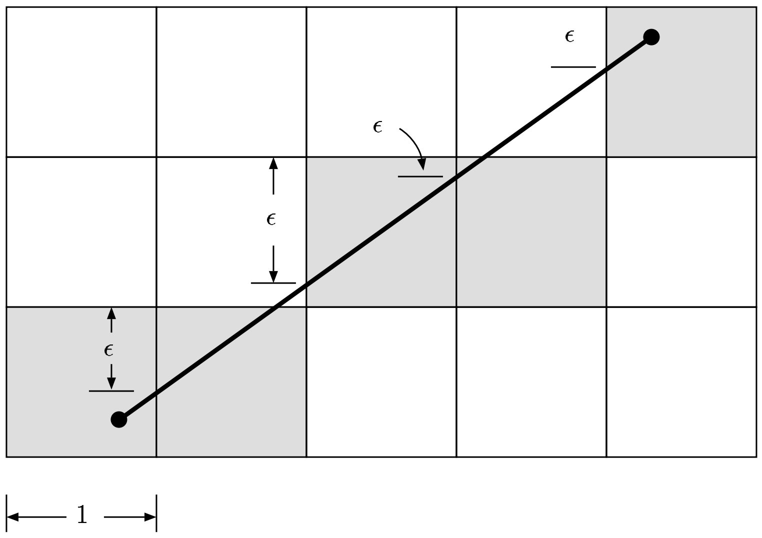
and is incremented by $\frac{\Delta y}{\Delta x}$ at each step. However, we cannot do the same simplifications with this algorithm as we did with the integer algorithm above, since in this case both $\Delta y$ and $\Delta x$ are real – not integer.

If we multiply through by $\Delta x$, we again obtain an approximation for $\bar{\epsilon}$ that, at least, does not require division.

$$\bar{\epsilon} \; = \; \Delta x \epsilon \; = \; -\Delta x(1 - (y_1 - \lfloor y_1 \rfloor)) + \Delta y(1 - (x_1 - \lfloor x_1 \rfloor))$$

However, to bring this algorithm back to integer form we assume that our basic pixel element is no longer $1 \times 1$ in size, but now it is $m \times m$ in size, and utilize increments and decrements in the algorithm of $\lfloor m\Delta x \rfloor$ and $\lfloor m\Delta y \rfloor$, respectively. The error term is first set to $\lfloor m\bar{\epsilon} \rfloor$ and the algorithm then proceeds as does the version with endpoints that have integer coordinates.

If we address the example, which attempts to draw a line from $(0.75, 0.125)$ to $(4.3, 2.8)$ in screen space.



and if we let $m = 1024$[1], then we have that

$$\begin{aligned} \lfloor m\Delta x \rfloor &= \lfloor 1024(3.55) \rfloor \\ &= \lfloor 3635.2 \rfloor \\ &= 3635 \end{aligned}$$

[1]We note that if $m = 2^k$ for some $k$, then this algorithm can be made especially fast

$$\begin{aligned}\lfloor m\Delta y\rfloor &= \lfloor 1024(2.55)\rfloor \\ &= \lfloor 2611.2\rfloor \\ &= 2611\end{aligned}$$

and

$$\begin{aligned}m\epsilon &= -1024(\Delta x(1-0.125) - \Delta y(1-0.75)) \\ &= -1024(3.55(1-0.125) - 2.55(1-0.75) \\ &= -1024(2.6625 - .6375) \\ &= -2073.6\end{aligned}$$

and so we initialize the integer quantity $\bar{\epsilon} = -2073$, and the algorithm proceeds as follows:

| $(x, y)$ | $\bar{\epsilon}$ | description |
|---|---|---|
| $(0,0)$ | -2073 | illuminate pixel $(0,0)$ |
| | 538 | increment $\epsilon$ by 2611 |
| $(1,0)$ | | increment $x$ by 1 |
| $(1,0)$ | 538 | illuminate pixel $(1,0)$ |
| | | since $\bar{\epsilon} > 0$ |
| $(1,1)$ | | increment $y$ by 1 |
| | -3097 | decrement $\bar{\epsilon}$ by 3635 |
| | -486 | increment $\epsilon$ by 2611 |
| $(2,1)$ | | increment $x$ by 1 |
| $(2,1)$ | -486 | illuminate pixel $(2,1)$ |
| | 2125 | increment $\epsilon$ by 2611 |
| $(3,1)$ | | increment $x$ by 1 |
| $(3,1)$ | 2125 | illuminate pixel $(3,1)$ |
| | | since $\bar{\epsilon} > 0$ |
| $(3,2)$ | | increment $y$ by 1 |
| | -1510 | decrement $\bar{\epsilon}$ by 3635 |
| | 1101 | increment $\epsilon$ by 2611 |
| $(4,2)$ | | increment $x$ by 1 |
| $(4,2)$ | 1101 | illuminate pixel $(4,2)$ |

Thus the integer version for Bresenham's algorithm with arbitrary endpoints is constructed as follows:

**Integer Bresenham's Algorithm**

```
The points $(x_1, y_1)$ and $(x_2, y_2)$ are assumed not equal
    and have arbitrary real coordinates
$\bar{\epsilon}$ is assumed to be integer valued.

Let $\Delta x = m(x_2 - x_1)$
Let $\Delta y = m(y_2 - y_1)$
Let $i_1 = \lfloor x_1 \rfloor$
Let $i_2 = \lfloor x_2 \rfloor$
Let $j = \lfloor y_1 \rfloor$
Let $\bar{\epsilon} = \lfloor \Delta y(1 - (x_1 - i) - \Delta x(1 - y_1 - j) \rfloor$
    for $i = i_1$ to $i_2$
    illuminate $(x, y)$
    if $(\bar{\epsilon} \geq 0)$
        $y += 1$
        $\bar{\epsilon} -= \Delta x$
        end if
    $x += 1$
    $\bar{\epsilon} += \Delta y$
    next i

finish
```

---

**Specifying the Driving Axis**

If $x$ is the driving axis, Bresenham's algorithm produces only one illuminated cell per column in the matrix of pixels. This feature allows the algorithm to be useful in the rasterization of polygons in image space. In this algorithm, we require only one illuminated pixel per row be produced – which is possible with Bresenham's algorithm by fixing the driving axis as the $y$ axis. This can be done by a simple change to the main loop of the algorithm.

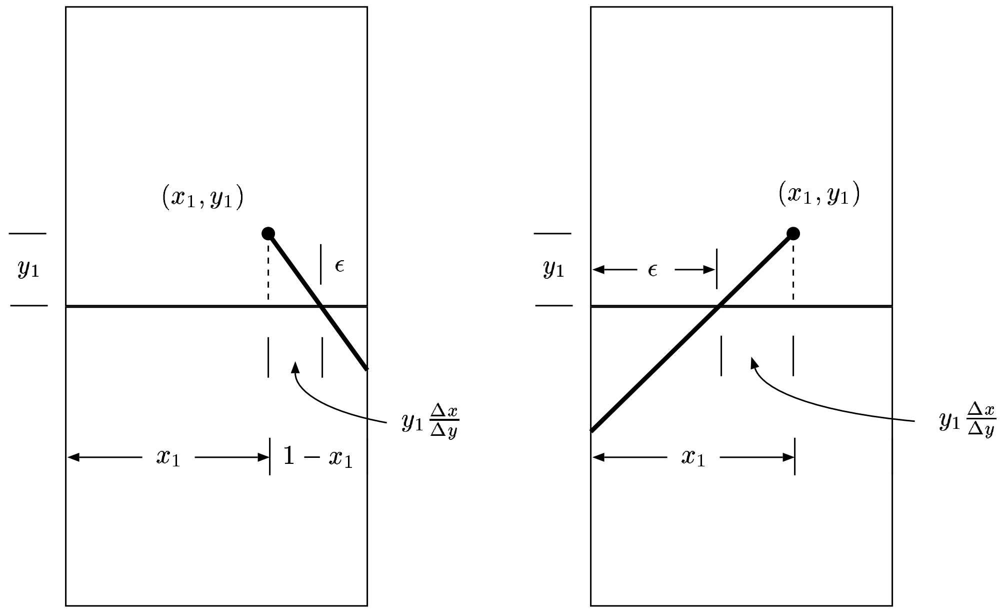
Normally, within the main loop of the algorithm, the coordinate corresponding to the *DA* is incremented by one unit and the coordinate corresponding to the other axis need only be incremented occasionally. When we fix the driving axis, the coordinate on the *DA* is still incremented by one unit, however the coordinate on the other axis may be incremented several times. This is actually a simple change to the algorithm, and and can be implemented by replacing the statement

**if** $(\bar{\epsilon} \geq 0)$

with

**while** $(\bar{\epsilon} \geq 0)$

The algorithm now appears as follows[2] We also note we have changed the algorithm to reflect that the $y$ axis is the driving axis. We note that $m = \frac{\Delta x}{\Delta y}$, and that according to the following picture



we must consider two possible initial values for $\epsilon$, one if $\Delta x > 0$ (the left-hand illustration) and one if $\Delta x < 0$ (the right-hand illustration). Referring to the illustration, we have either

$$\epsilon = -(1 - x_1 - y_1 \frac{\Delta x}{\Delta y})$$

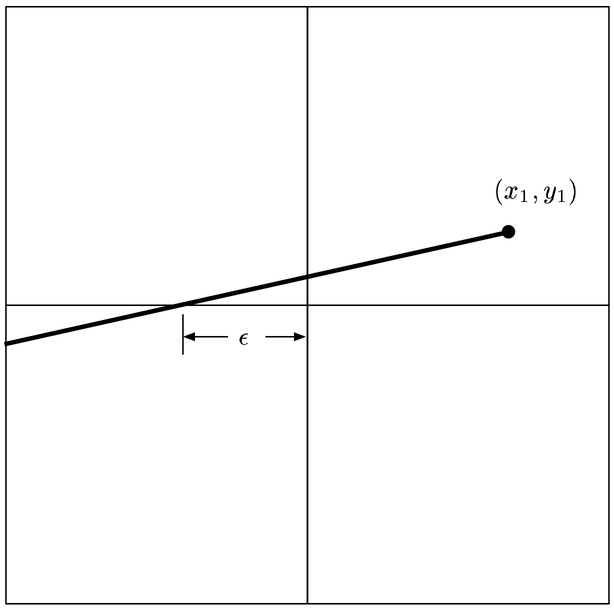
in the first case, or

$$\epsilon = -(x_1 - y_1 \frac{\Delta x}{\Delta y})$$

in the second. In general $x_1$ must be replaced by $x_1 - \lfloor x_1 \rfloor$ and $y_1$ by $y_1 - \lfloor y_1 \rfloor$ as the figure is drawn as if the lower-left-hand corner of the pixel is $(0, 0)$.

We also need to recognize that $\epsilon$ may be greater than zero immediately, as in the following figure. Therefore, we must move the "illuminate" step in our algorithm below the "while" statement. This will insure that we are at the correct boundary pixel for the trapezoid.

[2]We are using the non-integer form of the algorithm, which is clearer in its presentation. The conversion to the integer algorithm is straightforward.



The algorithm now appears as follows:

**Bresenham's Algorithm with $y$ as the Driving Axis**

```
The points $(x_1, y_1)$ and $(x_2, y_2)$ are assumed not equal
$\epsilon$ is assumed to be real.

Let $\Delta x = x_2 - x_1$
Let $\Delta y = y_2 - y_1$
Let $m = |\frac{\Delta x}{\Delta y}|$
Let $i = \lfloor x_1 \rfloor$
Let $j_1 = \lfloor y_1 \rfloor$
Let $j_2 = \lfloor y_2 \rfloor$

if $\Delta x > 0$
        $i_{inc} = 1$
        $\epsilon = -(1 - (x_1 - i) - (y_1 - j_1)\frac{\Delta x}{\Delta y})$
    else
        $i_{inc} = -1$
        $\epsilon = -((x_1 - i) - (y_1 - j_1)\frac{\Delta x}{\Delta y})$
    end if

for $j = j_1$ to $j_2$

    while $(\epsilon \geq 0)$
        $i += 1$
        $\epsilon -= 1.0$
        end while
        **illuminate** $(i, j)$

        $j += 1$
        $\epsilon += m$
        **next** j

    **finish**
```

This algorithm will continue to increment the $x$ value (and decrement $\epsilon$) as long as $\epsilon$ is greater than zero. It illuminates only one pixel per row on the line.

We are assuming that $\Delta y \neq 0$ which is valid since we are only considering the non-horizontal edges of the trapezoids that we want to rasterize. It should also be mentioned that the above algorithm is written for the case where $m$ is positive. If $m = 0$, then $\epsilon = -(1 - (x_1 - i))$ and is never incremented. In this case, we have a vertical line, and the $x$ value will never need to be incremented. If $m < 0$, then we must decrement $i$ in the algorithm, and add $|m|$ to $\epsilon$.

---

## Summary

Bresenham's Algorithm is a fundamental algorithm in computer graphics. Its basic use it to draw lines on raster graphics devices, however it is useful as a driving engine for many other graphics routines.

---