Derivation of Bresenham Algorithm

Introduction

Consider a line segment from $P_1(x_1, y_1)$ to $P_2(x_2, y_2)$ with end points having integer coordinates and slope satisfying $0 \leq |m| \leq 1$. The DDA algorithm for line segment from P_1 to P_2 is shown below.

```
1  // 0 <= |m| <= 1
2  dx = x2 - x1; dy = y2 - y1
3  m = (float)dy/(float)dx
4  m = (m > 0.f) ? m : -m // 0 <= |m| <= 1
5  xstep = (dx > 0) ? 1 : -1
6  dx = (dx > 0) ? dx : -dx
7  ystep = (dy > 0) ? m : -m
8  set_pixel(x1, y1, color)
9  while (--dx) {
10   x1 += xstep
11   y1 += ystep
12   set_pixel(x1, (int)y1, color)
13 }
```

The DDA algorithm is an incremental algorithm and appears efficient. However, it has a few deficiencies:

1. The DDA algorithm implements floating-point arithmetic. In line 3, a floating-point division is required to compute the line's slope while line 11 requires floating-point addition to accumulate slope.
2. Floating-point representation is an approximate representation of the infinite set of real numbers in an interval. Therefore, an approximation of the line's mathematical slope $\frac{y_2 - y_1}{x_2 - x_1}$ is stored in floating-point variable `m`. Incremental calculations in line 11 `y1 += ystep` that continually add an approximation to approximate values make the accumulated value in `y1` drift further away from the mathematical y value. For specific applications, the relative error between the accumulated and mathematical values can be too large to ignore.
3. The conversion of floating-point values to integral values is not free and depending on the CPU and compiler combination can cause considerable performance decreases.
4. Not all embedded systems may have floating-point units. Even if a floating-point unit is present, power considerations would encourage using an integer-only algorithm.

Bresenham published a paper "Algorithm for Computer Control of a Digital Plotter" in 1965 that derived an incremental algorithm for moving between two points on a line segment using integer-only arithmetic and completely avoiding floating-point arithmetic. This seminal algorithm has spawned a variety of related algorithms that have become standard in hardware and software rasterizers. This document provides a mathematical development of the Bresenham algorithm for line segments with slopes $0 \leq |m| \leq 1$ using a slightly different approach than the original algorithm. The derivation and implementation of a corresponding algorithm for line segments with slopes $|m| > 1$ is left to the reader.

Categorization of Line Segments into Octants

Any implementation of the Bresenham algorithm must rasterize an infinite variety of line segments. To make the derivation of such an algorithm tractable, it would be convenient to squeeze these infinitely many varieties of line segments into a small number of categories. Lines passing through the origin having slopes 1 and -1 divide the infinite two-dimensional xy plane into eight octants. Any line segment to be rasterized can be categorized as belonging to one of these eight octants by translating the line segment so that its start point is at the origin. The vector from the origin to the translated end point categorizes the line segments into a particular octant.

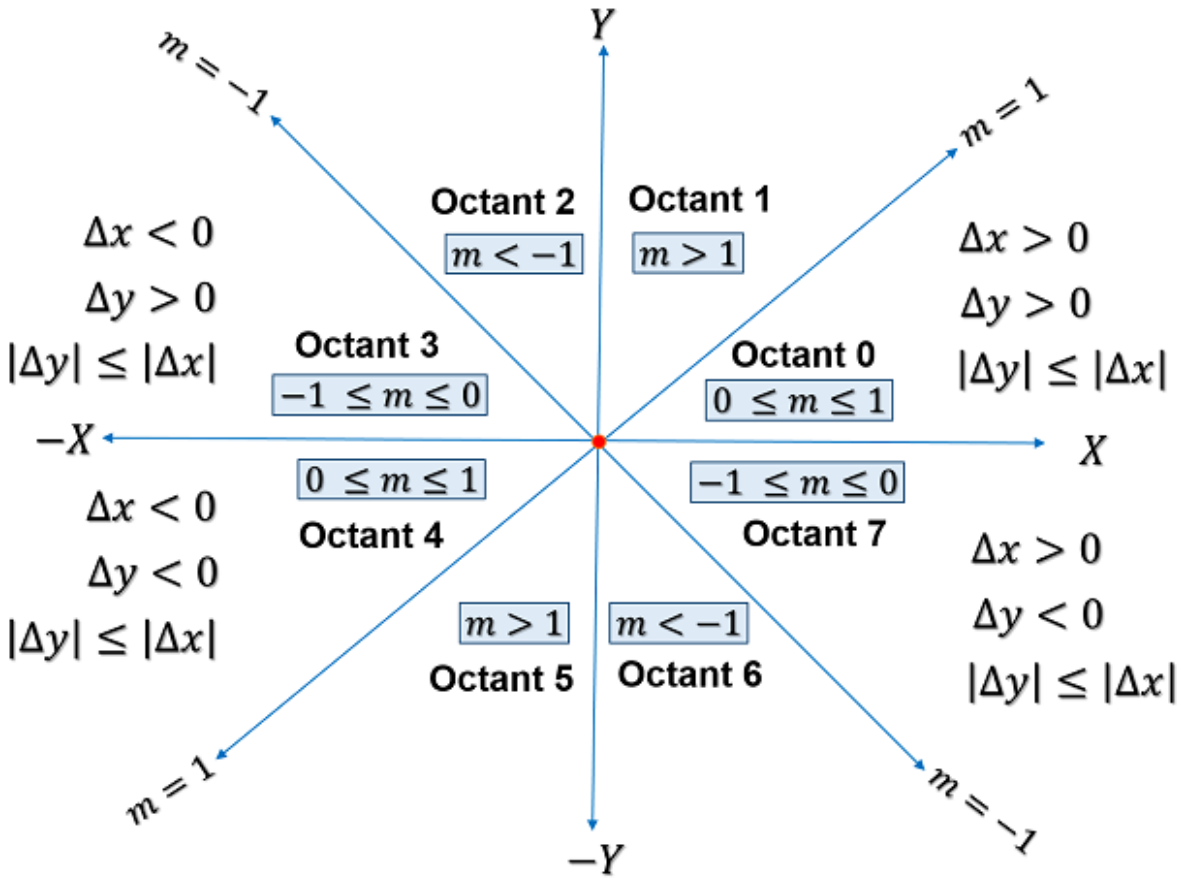

Figure 1: Categorizing line segments into octants

Figure 1 illustrates the categorical mapping of an infinite variety of line segments with slopes $-1 \leq |m| \leq 1$ to octants 0, 3, 4, and 7. Let's categorize a line segment from $P_1(111, 115)$ to $P_2(121, 120)$. Translating end points P_1 and P_2 by $\vec{t} = O - P_1 = (-111, -115)$ results in transformed line segment $P_1'(0, 0)$ to $P_2'(10, 5)$ which can be specified as an octant 0 line segment. Similarly, a line segment from $P_3(21, 20)$ to $P_4(11, 15)$ is determined as an octant 4 line segment since the translated line segment starts at $P_3'(0, 0)$ and ends at $P_4'(-10, -5)$.

A simple method to rasterize line segments with slopes $-1 \leq |m| \leq 1$ is to implement separate rasterization code for octants 0, 3, 4, and 7. Initial tests are performed to categorize an input line segment into a particular octant and the rasterization code associated with the octant is invoked. Although straightforward, the presence of redundant code means hardware implementations will have higher silicon and possibly higher power consumption requirements. A better implementation would extract geometrical relationships between lines in these four octants and combine their individual implementations into a single implementation. This document begins with the mathematical derivation of the Bresenham's algorithm for octant 0 line segments followed by an implementation of the algorithm. Next, a separate algorithm is derived and implemented for octant 4 lines. Both these algorithms are examined to extract commonalities

that yields a single implementation that rasterizes lines with slope $0 \leq m \leq 1$. Lines in octants 3 and 7 have slopes $-1 \leq m < 0$. Extracting geometrical symmetry between lines with slope $0 \leq m \leq 1$ and lines with slope $-1 \leq m < 0$ enables a single implementation of the Bresenham algorithm for line segments with slopes $-1 \leq |m| \leq 1$.

Bresenham Algorithm for Octant 0

OpenGL provides the ability to connect a set of end points using the `GL_LINE_LOOP` and `GL_LINE_STRIP` primitives. With `GL_LINE_LOOP` primitive, given a sequence of three vertices P_1, P_2, and P_3, OpenGL rasterizes three line segments $\overline{P_1P_2}$, $\overline{P_2P_3}$, and $\overline{P_3P_1}$. With `GL_LINE_STRIP` primitive and given the same set of vertices, OpenGL rasterizes two line segments $\overline{P_1P_2}$ and $\overline{P_2P_3}$. For efficiency purposes and to avoid artifacts, the algorithm must not rasterize the common vertex P_2 twice, first by line segment $\overline{P_1P_2}$ and the second time by line segment $\overline{P_2P_3}$. To ensure that only one line segment rasterizes the common vertex, this document follows the convention that a line segment's end point is not rasterized, except when both start and end points are coincident.

Assume a line segment that begins at $P_1(x_1, y_1)$ and ends at $P_2(x_2, y_2)$ with end points having integer coordinates and slope satisfying $0 \leq m \leq 1$. Let $y = mx + h$ be the equation of the line with $m = \dfrac{y_2 - y_1}{x_2 - x_1} = \dfrac{\Delta y}{\Delta x}$ and constant $h = y_1 - mx_1$. Figure 2 illustrates an additional assumption that the algorithm relies on: $x_2 > x_1 \implies \Delta x > 0$.

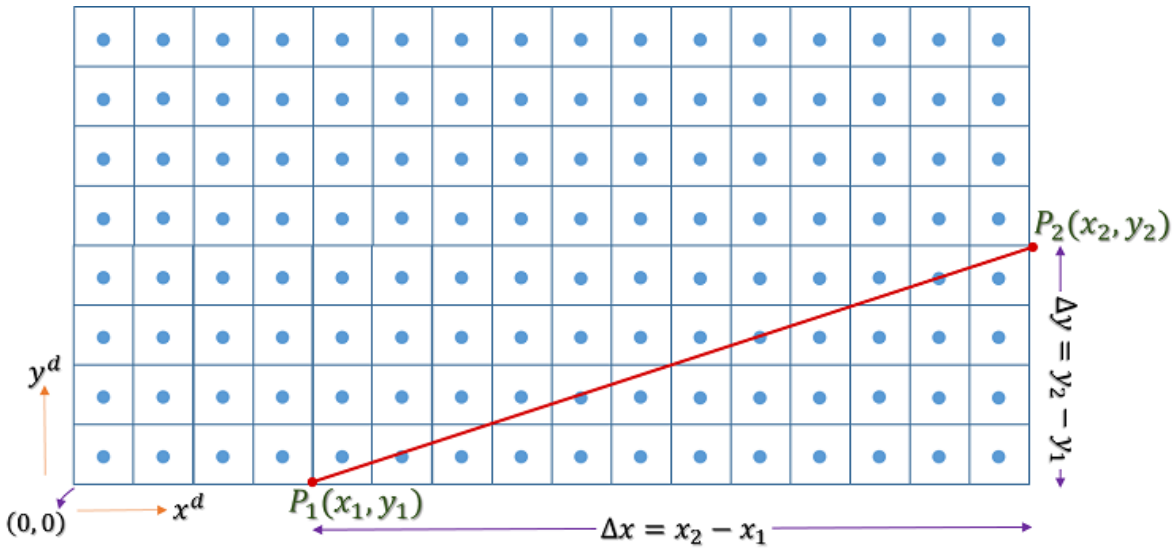

Figure 2: Line segment in window coordinates

The algorithm requires Δx iterations to compute fragments between P_1 to P_2 with the first iteration at start point P_1. Suppose fragment $C(x(k), y(k))$ has been chosen by the algorithm at iteration k. The algorithm aims to determine fragment $(x(k+1), y(k+1))$ at iteration $k+1$. Since $0 \leq m \leq 1$ and $\Delta x > 0$, the line segment is sampled along x-axis by incrementing x value in unit steps. Therefore, at the next iteration $k+1$, $x(k+1) = x(k) + 1$. The crux of the algorithm is determining $y(k+1)$. The fragments of interest at iteration $k+1$ are $E(x(k)+1, y(k))$ to the east of C and $NE(x(k)+1, y(k)+1)$ to the northeast of C. As shown in Figure 3, this decision can be based on the *vertical* distance of fragments E and NE from the mathematical value $y = m(x(k)+1) + h$ of the line segment at $x(k)+1$.

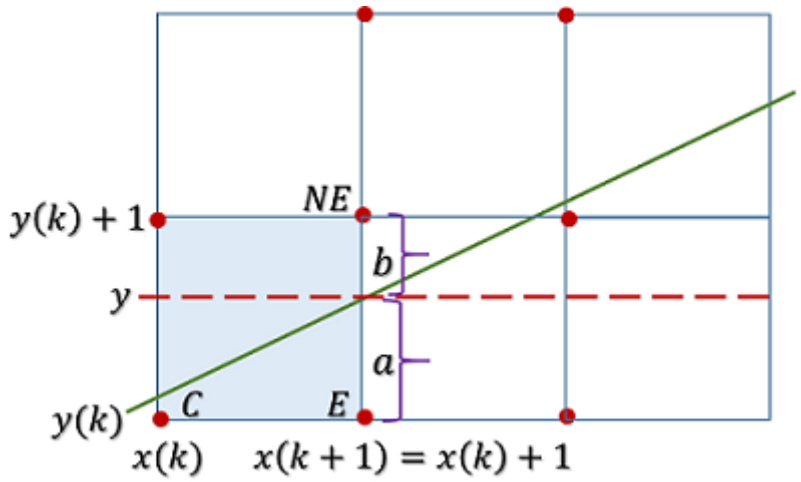

Figure 3: Decision parameter for choosing between east and northeast fragments

First, the vertical distances a and b are computed as:

$$\begin{aligned} a &= y - y(k) \\ b &= y(k) + 1 - y \end{aligned} \tag{1}$$

If distance a is less than or equal to distance b, that is, $a \leq b$, the line segment passes closer to the east fragment, so fragment $E(x(k) + 1, y(k))$ is chosen. Otherwise, if $a > b$, the line segment passes closer to the northeast fragment, so fragment $NE(x(k) + 1, y(k) + 1)$ is chosen. The same conclusion can be reached by computing the difference between the two distances:

$$\begin{aligned} a - b &= y - y(k) - (y(k) + 1 - y) \\ &= 2y - 2y(k) - 1 \\ \implies a - b &= 2mx(k) - 2y(k) + (2m + 2h - 1) \end{aligned} \tag{2}$$

If $(a - b) < 0$, then fragment $E(x(k) + 1, y(k))$ is chosen. Otherwise, fragment $NE(x(k) + 1, y(k) + 1)$ is chosen. However, Equation (2) requires floating-point calculations in the evaluation of slope m and line constant h. The first insight towards building an integer-only arithmetic algorithm comes from the assumption that end points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ have integer coordinates. This means that the line's slope $m = \dfrac{\Delta y}{\Delta x}$ is a rational number because numerator $\Delta x = x_2 - x_1$ and denominator $\Delta y = y_2 - y_1$ are integers. Floating-point calculations are converted to integer calculations by replacing the left hand side $a - b$ of Equation (2) with $(a - b)\Delta x$:

$$\begin{aligned} a - b &= 2mx(k) - 2y(k) + (2m + 2h - 1) \\ &= 2\frac{\Delta y}{\Delta x}x(k) - 2\frac{\Delta x}{\Delta x}y(k) + (2\frac{\Delta y}{\Delta x} + 2\frac{\Delta x}{\Delta x}y_1 - 2\frac{\Delta y}{\Delta x}x_1 - \frac{\Delta x}{\Delta x}) \\ \implies (a - b)\Delta x &= 2\Delta y x(k) - 2\Delta x y(k) + [2\Delta y + 2\Delta x y_1 - 2\Delta y x_1 - \Delta x] \\ \implies d(k) = (a - b)\Delta x &= 2\Delta y x(k) - 2\Delta x y(k) + c \end{aligned} \tag{3}$$

where $c = 2\Delta y + 2\Delta x y_1 - 2\Delta y x_1 - \Delta x$ is a constant value. Expression $d(k)$ in Equation (3) is called the ***decision parameter*** at iteration k because its value determines which fragment is chosen in iteration $k + 1$. Since $d(k)$ is computed by scaling $a - b$ with positive value Δx, the conclusions reached previously with Equation (2) continue to remain true with Equation (3): if $d(k) \leq 0$, east fragment $E(x(k) + 1, y(k))$ is chosen. Otherwise, northeast fragment $NE(x(k) + 1, y(k) + 1)$ is chosen.

Although Equation (3) uses integral computations, its right expression is computationally more complex than the DDA algorithm and is therefore not suitable as its replacement. Applying knowledge gained from DDA algorithm's development, a second insight is to use incremental calculations to compute $d(k+1)$.

$$d(k+1) = 2\Delta yx(k+1) - 2\Delta xy(k+1) + c \tag{4}$$

If $d(k) \leq 0$, east fragment E is chosen, and thus $x(k+1) = x(k) + 1$ and $y(k+1) = y(k)$. Substituting for $x(k+1)$ and $y(k+1)$ in Equation (4):

$$\begin{aligned} d(k+1) &= 2\Delta yx(k+1) - 2\Delta xy(k+1) + c \\ &= 2\Delta yx(k) - 2\Delta xy(k) + c + 2\Delta y \\ \implies d(k+1) &= d(k) + 2\Delta y \qquad \text{if} \quad d(k) \leq 0 \end{aligned} \tag{5}$$

If $d(k) > 0$, northeast fragment NE is chosen, and thus $x(k+1) = x(k) + 1$ and $y(k+1) = y(k) + 1$. Substituting for $x(k+1)$ and $y(k+1)$ in Equation (4):

$$\begin{aligned} d(k+1) &= 2\Delta yx(k+1) - 2\Delta xy(k+1) + c \\ &= 2\Delta yx(k) - 2\Delta xy(k) + c + 2\Delta y - 2\Delta x \\ \implies d(k+1) &= d(k) + 2\Delta y - 2\Delta x \qquad \text{if} \quad d(k) > 0 \end{aligned} \tag{6}$$

The results from Equations (5) and (6) can be combined:

$$d(k+1) = d(k) + \begin{cases} 2\Delta y, & \text{if } d(k) \leq 0 \\ 2\Delta y - 2\Delta x, & \text{if } d(k) > 0 \end{cases} \tag{7}$$

The efficiency of the algorithm has now considerably improved with the calculation of each successive fragment requiring just a sign test and addition of a constant value. Incremental calculations work only if there is an initial value to begin iterations from. The initial decision parameter $d(0)$ is computed from Equation (3) with $x(0)$ and $y(0)$ set to x_1 and y_1, respectively:

$$d(0) = 2\Delta y - \Delta x \tag{8}$$

Using Equations (7) and (8), the implementation of Bresenham's algorithm for line segments with $0 \leq m \leq 1,\ \Delta x > 0, \Delta y \geq 0$ is given below:

```
// assume 0 <= m <= 1, (x2 - x1) > 0, and (y2 - y1) >= 0
// to draw line loops correctly, the end point will not be rendered
void line_bresenham_octant0(int x1, int y1, int x2, int y2, Color clr) {
  int dx = x2 - x1, dy = y2 - y1, d = 2*dy - dx, de = 2*dy, dne = 2*dy -
2*dx;
  set_pixel(x1, y1, clr);
  while (--dx) {
    if (d <= 0) {
      d += de;
    } else {
      d += dne; ++y1;
    }
    ++x1;
    set_pixel(x1, y1, clr);
  }
}
```

Bresenham Algorithm for Octant 4

Since end points are not rasterized, line segments in octant 4 cannot be rasterized by simply swapping the start and end points and then using the algorithm for octant 0 line segments.

Consider a line segment from $P_1(x_1, y_1)$ to $P_2(x_2, y_2)$ with end points having integer coordinates and slope satisfying $0 \leq m \leq 1$. Let $y = mx + h$ be the equation of the line with $m = \dfrac{y_2 - y_1}{x_2 - x_1} = \dfrac{\Delta y}{\Delta x}$ and constant $h = y_1 - mx_1$. Figure 4 illustrates one additional assumption that the algorithm relies on: $x_2 < x_1 \implies \Delta x < 0$. and $y_2 \leq y_1 \implies \Delta y \leq 0$.

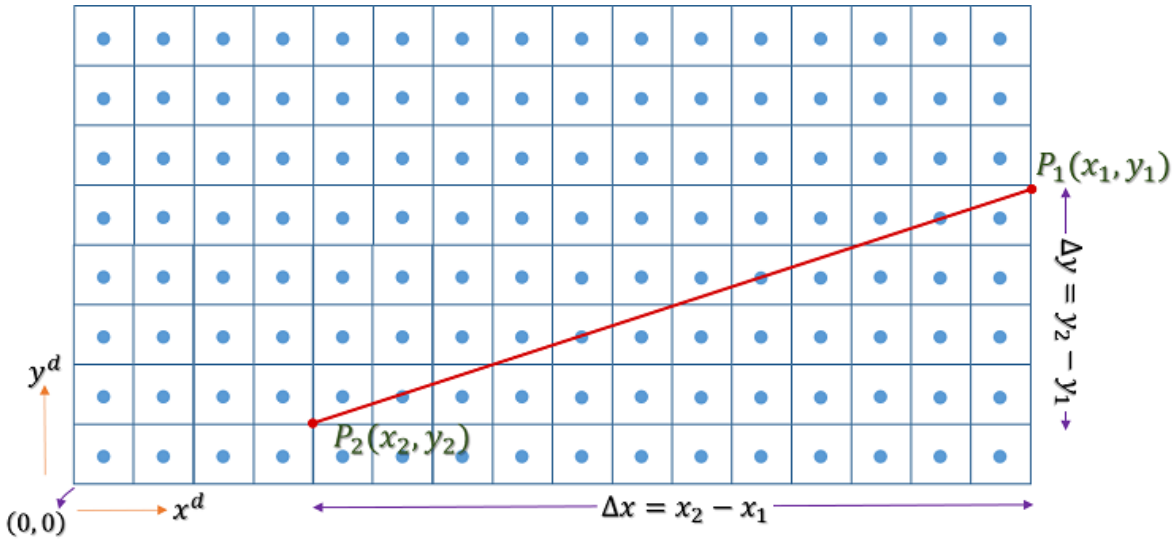

Figure 4: Octant 4 line segment in window coordinates

Suppose fragment $C(x(k), y(k))$ has been chosen by the algorithm at iteration k. The algorithm has to determine fragment $(x(k+1), y(k+1))$ at iteration $k+1$. Since $0 \leq m \leq 1$ and $\Delta x < 0$, the line segment is sampled along x-axis by decrementing x value in unit steps. Therefore, at the next iteration $k+1$, $x(k+1) = x(k) - 1$. The fragments of interest at iteration $k+1$ are $W(x(k) - 1, y(k))$ to the west of C and $SW(x(k) - 1, y(k) - 1)$ to the southwest of C. As shown in Figure 5, this decision can be based on the *vertical* distance of fragments W and SW from the mathematical $y = m(x(k) - 1) + h$ value of the line segment at $x(k) - 1$.

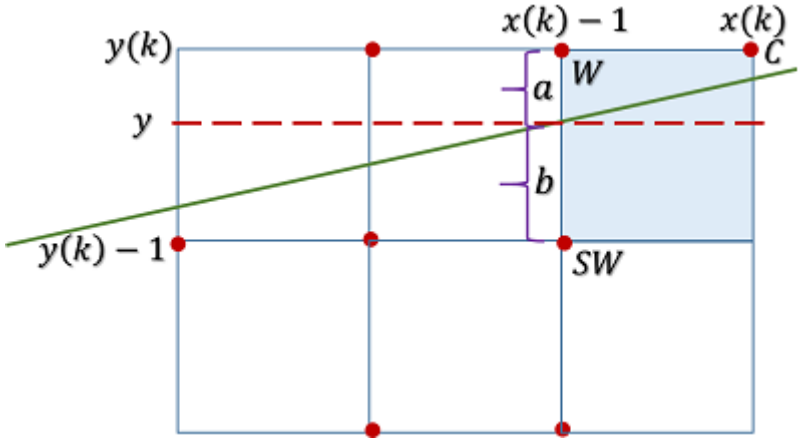

Figure 5: Decision parameter for choosing between west and southwest fragments

First, differences between vertical distances a and b are computed:

$$\begin{aligned} a &= y(k) - y \\ b &= y - (y(k) - 1) \\ a - b &= y(k) - y - [y - (y(k) - 1)] \\ \implies a - b &= -2mx(k) + 2y(k) + (2m + 2mx_1 - 2y_1 - 1) \end{aligned} \tag{9}$$

If $(a - b) < 0$, then fragment $W(x(k) - 1, y(k))$ is chosen. Otherwise, fragment $SW(x(k) - 1, y(k) - 1)$ is chosen. Since $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$ are integral values, floating-point calculations in right hand side of Equation (9) are converted to integer calculations and a decision parameter at iteration k is determined:

$$d(k) = (a-b)\Delta x = -2\Delta y x(k) + 2\Delta x y(k) + c \tag{10}$$

where $c = 2\Delta y + 2\Delta y x_1 - 2\Delta x y_1 - \Delta x$ is a constant value. Since $d(k)$ is computed by scaling $a - b$ with a *negative* value Δx, the conclusions reached previously with Equation (9) are ***negated***:

$$\begin{aligned}
\Delta x &< 0 \\
(a-b) &\leq 0 \implies \text{choose west fragment } W(x(k)-1, y(k)) \\
\implies (a-b)\Delta x &\geq 0 \\
\implies d(k) &\geq 0 \implies \text{choose west fragment } W(x(k)-1, y(k))
\end{aligned}$$

$$\begin{aligned}
\Delta x &< 0 \\
(a-b) &> 0 \implies \text{choose southwest fragment } SW(x(k)-1, y(k)-1) \\
\implies (a-b)\Delta x &< 0 \\
\implies d(k) &< 0 \implies \text{choose southwest fragment } SW(x(k)-1, y(k)-1)
\end{aligned} \tag{11}$$

Using Equation (10) which computes the decision parameter at iteration k, the decision parameter $d(k+1)$ at iteration $k+1$ evaluates as

$$d(k+1) = -2\Delta y x(k+1) + 2\Delta x y(k+1) + c \tag{12}$$

If $d(k) \geq 0$, west fragment W is chosen, and thus $x(k+1) = x(k) - 1$ and $y(k+1) = y(k)$. If $d(k) < 0$, southwest fragment SW is chosen, and thus $x(k+1) = x(k) - 1$ and $y(k+1) = y(k) - 1$. Using Equation (10) to compute the initial decision parameter at iteration 0 and substituting for $x(k+1)$ and $y(k+1)$ in Equation (12) when W and SW fragments are chosen, the decision parameter's initial and incremental evaluations are:

$$\begin{aligned}
d(0) &= 2\Delta y - \Delta x \\
d(k+1) &= d(k) + \begin{cases} 2\Delta y, & \text{if } d(k) \geq 0 \\ 2\Delta y - 2\Delta x, & \text{if } d(k) < 0 \end{cases}
\end{aligned} \tag{13}$$

Using Equation (13), Bresenham algorithm for line segments with $0 \leq m \leq 1,\ \Delta x < 0, \Delta y \leq 0$ can be implemented as:

```
// assume 0 <= m <= 1, (x2 - x1) < 0, and (y2 - y1) <= 0
// to draw line loops correctly, the end point will not be rendered
void line_bresenham_octant4(int x1, int y1, int x2, int y2, Color clr) {
  int dx = x2 - x1, dy = y2 - y1, d = 2*dy - dx, dw = 2*dy, dsw = 2*dy - 2*dx;
  set_pixel(x1, y1, clr);
  while (++dx) {
    if (d >= 0) {
      d += dw;
    } else {
      d += dsw;
      --y1;
    }
    --x1;
    set_pixel(x1, y1, clr);
  }
}
```

Bresenham Algorithm for Octants 0 and 4

For octant 0, consider a line segment from $P_1(x_1, y_1)$ to $P_2(x_2, y_2)$ with $\Delta x = x_2 - x_1 > 0$ and $\Delta y = y_2 - y_1 \geq 0$. For octant 4, use the same line segment except that it starts at $P_2 = (x_2, y_2)$ and ends at $P_1(x_1, y_1)$ with $\Delta x' = x_1 - x_2 < 0$ and $\Delta y' = y_1 - y_2 \leq 0$. Both line segments have the same slope $m = \dfrac{\Delta y}{\Delta x} = \dfrac{\Delta y'}{\Delta x'}$. Since both line segments are in octants 0 and 4, the condition $0 \leq m \leq 1$ holds. Using Equations (7), (8), and (13), results from mathematical derivations for octants 0 and 4 are compared.

Octant 0	**Octant** 4
$\Delta x = x_2 - x_1$	$\Delta x' = x_1 - x_2$
$\Delta y = y_2 - y_1$	$\Delta y' = y_1 - y_2$
$d(0) = 2\Delta y - \Delta x$	$d'(0) = 2\Delta y' - \Delta x'$
$d(k+1) = d(k) + 2\Delta y - 2\Delta x, \text{ if } d(k) > 0$	$d'(k+1) = d'(k) + 2\Delta y' - 2\Delta x', \text{if } d'(k) < 0$
$d(k+1) = d(k) + 2\Delta y, \text{ if } d(k) \leq 0$	$d'(k+1) = d'(k) + 2\Delta y', \text{ if } d'(k) \geq 0$

Replacing $\Delta x' = -\Delta x$ and $\Delta y' = -\Delta y$, decision parameter computations for octant 4 lines are negations of corresponding computations for octant 0 lines:

Octant 0	**Octant** 4
$\Delta x = x_2 - x_1$	$\Delta x' = -\Delta x = x_1 - x_2$
$\Delta y = y_2 - y_1$	$\Delta y' = -\Delta y = y_1 - y_2$
$d(0) = 2\Delta y - \Delta x$	$d'(0) = -(2\Delta y - \Delta x) = -d(0)$
$d(k+1) = d(k) + 2\Delta y - 2\Delta x, \text{ if } d(k) > 0$	$d'(k+1) = d'(k) - (2\Delta y - 2\Delta x), \text{ if } d'(k) < 0$
$d(k+1) = d(k) + 2\Delta y, \text{ if } d(k) \leq 0$	$d'(k+1) = d'(k) - 2\Delta y, \text{ if } d'(k) \geq 0$

Decision parameter computations for octant 4 lines are negations of corresponding computations for octant 0 lines:

Octant 0	**Octant** 4
$\Delta x = x_2 - x_1$	$\Delta x' = -\Delta x = x_1 - x_2$
$\Delta y = y_2 - y_1$	$\Delta y' = -\Delta y = y_1 - y_2$
$d(0) = 2\Delta y - \Delta x$	$-d(0) = -(2\Delta y - \Delta x)$
$d(k+1) = d(k) + 2\Delta y - 2\Delta x, \text{ if } d(k) > 0$	$-d(k+1) =$ $-d(k) - (2\Delta y - 2\Delta x), \text{ if } -d(k) < 0$
$d(k+1) = d(k) + 2\Delta y, \text{ if } d(k) \leq 0$	$-d(k+1) = -d(k) - 2\Delta y, \text{ if } -d(k) \geq 0$

Negating $\Delta x'$ and $\Delta y'$ leads to a negation of the decision parameters and their `if` conditions:

Octant 0	**Octant** 4
$\Delta x = x_2 - x_1$	$\Delta x' = \lvert x_1 - x_2 \rvert = \Delta x$
$\Delta y = y_2 - y_1$	$\Delta y' = \lvert y_1 - y_2 \rvert = \Delta y$
$d(0) = 2\Delta y - \Delta x$	$d(0) = 2\Delta y - \Delta x$
$d(k+1) = d(k) + 2\Delta y - 2\Delta x, \text{ if } d(k) > 0$	$d(k+1) = d(k) + 2\Delta y - 2\Delta x, \text{ if } d(k) > 0$
$d(k+1) = d(k) + 2\Delta y, \text{ if } d(k) \leq 0$	$d(k+1) = -d(k) + 2\Delta y, \text{ if } d(k) \leq 0$

The algorithm to rasterize octant 4 line segments now matches the algorithm to rasterize octant 0 line segments. Octant 4 lines require trivial negation of $\Delta x'$ and $\Delta y'$ values and their initial x and y values have to be decremented. The combined algorithm has the following implementation.

```
// assume 0 <= m <= 1
// to draw line loops correctly, the end point will not be rendered
void line_bresenham_octant04(int x1, int y1, int x2, int y2, Color clr) {
  int dx = x2 - x1, dy = y2 - y1;
  int xstep = (dx < 0) ? -1 : 1;
  int ystep = (dy < 0) ? -1 : 1;
  dx = (dx < 0) ? -dx : dx;
  dy = (dy < 0) ? -dy : dy;
  int d = 2*dy - dx, dmin = 2*dy, dmaj = 2*dy - 2*dx;
  set_pixel(x1, y1, clr);
  while (--dx) {
    y1 += (d > 0) ? ystep : 0;
    d  += (d > 0) ? dmaj  : dmin;
    x1  += xstep;
    set_pixel(x1, y1, clr);
  }
}
```

Bresenham Algorithm for Octants 0, 3, 4, and 7

Consider an octant 0 line segment from $P_1(0,0)$ to $P_2(x_2, y_2)$ with $\Delta x = x_2 > 0$ and $\Delta y = y_2 \geq 0$. A line segment from $P_1(0,0)$ to $P_2'(-x_2, y_2)$ describes an octant 3 line segment with $\Delta x' = -\Delta x < 0$ and $\Delta y' = y_2 \geq 0$. The octant 0 line segment has slope $m = \dfrac{\Delta y}{\Delta x}$ with condition $0 \leq m \leq 1$ while the octant 3 line segment has slope $m = -\dfrac{\Delta y}{\Delta x}$ with condition $-1 \leq m \leq 0$. These two lines are symmetrical about the vertical y axis. This means that an octant 3 line segment can be rasterized using the algorithm for the corresponding octant 0 line segment with the sole difference being an octant 3 line segment requires x values be decremented in unit steps from the start value. Function `line_bresenham_octant04` is programmed to sample the line segment in the appropriate x direction using variable `xstep`. Therefore, the previously implemented function `line_bresenham_octant04` can correctly rasterize octant 3 line segments.

As before, consider an octant 0 line segment from $P_1(0,0)$ to $P_2(x_2, y_2)$ with $\Delta x = x_2 > 0$ and $\Delta y = y_2 \geq 0$. A line segment from $P_1(0,0)$ to $P_2'(x_2, -y_2)$ describes an octant 7 line segment with $\Delta x' = \Delta x > 0$ and $\Delta y' = -y_2 \leq 0$. The octant 0 line segment has slope $m = \dfrac{\Delta y}{\Delta x}$ with condition $0 \leq m \leq 1$ while the octant 7 line segment has slope $m = -\dfrac{\Delta y}{\Delta x}$ with condition $-1 \leq m \leq 0$. These two lines are symmetrical about the horizontal x axis. This means that the octant 7 line segment can be rasterized by using the algorithm for the corresponding octant 0 line

segment while ensuring that y values are decremented in unit steps from the start value. Function `line_bresenham_octant04` is programmed to sample the line segment in the appropriate y direction using variable `ystep`. Therefore, the previously implemented function `line_bresenham_octant04` can correctly rasterize octant 7 line segments.

For the sake of completeness, the previous algorithm is presented with a more apt name but without any changes to its functionality.

```
// assume 0 <= |m| <= 1
// to draw line loops correctly, the end point will not be rendered
void line_bresenham_octant0347(int x1, int y1, int x2, int y2, Color clr) {
  int dx = x2 - x1, dy = y2 - y1;
  int xstep = (dx < 0) ? -1 : 1;
  int ystep = (dy < 0) ? -1 : 1;
  dx = (dx < 0) ? -dx : dx;
  dy = (dy < 0) ? -dy : dy;
  int d = 2*dy - dx, dmin = 2*dy, dmaj = 2*dy - 2*dx;
  set_pixel(x1, y1, clr);
  while (--dx) {
    y1 += (d > 0) ? ystep : 0;
    d  += (d > 0) ? dmaj  : dmin;
    x1  += xstep;
    set_pixel(x1, y1, clr);
  }
}
```