Disadvantages of DDA

- Floating Point Addition
- Rounding Off Function

Bresenham Algorithm

- Incremental Algorithm as DDA
- No floating point usage
- No rounding function used
- It is an efficient method because it involves only integer addition, subtractions, and multiplication operations. These operations can be performed very rapidly so lines can be generated quickly.

Bresenham Line Generation

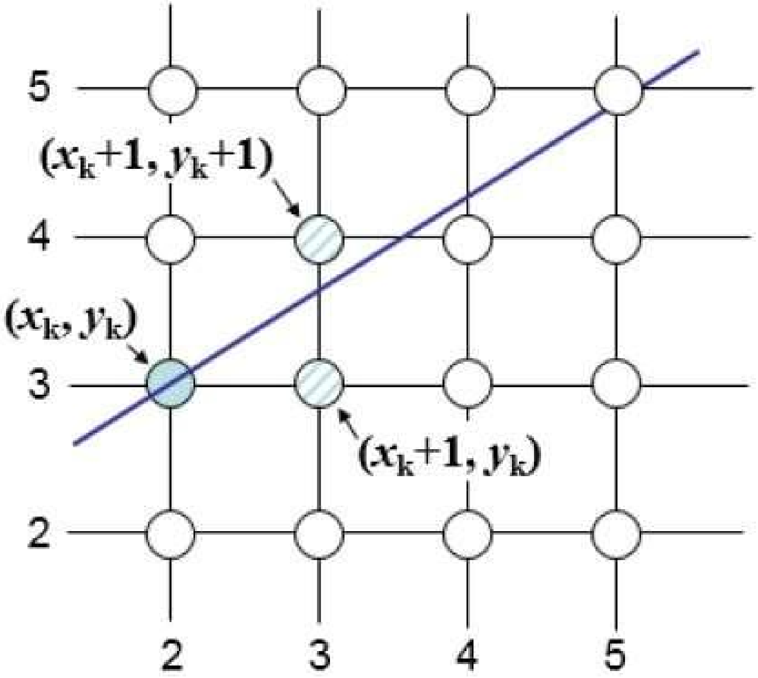

Image Source: https://www.tutorialspoint.com/computer_graphics/line_generation_algorithm.htm

Bresenham Line Generation

$Y = m\,(X_k + 1) + c$ -------

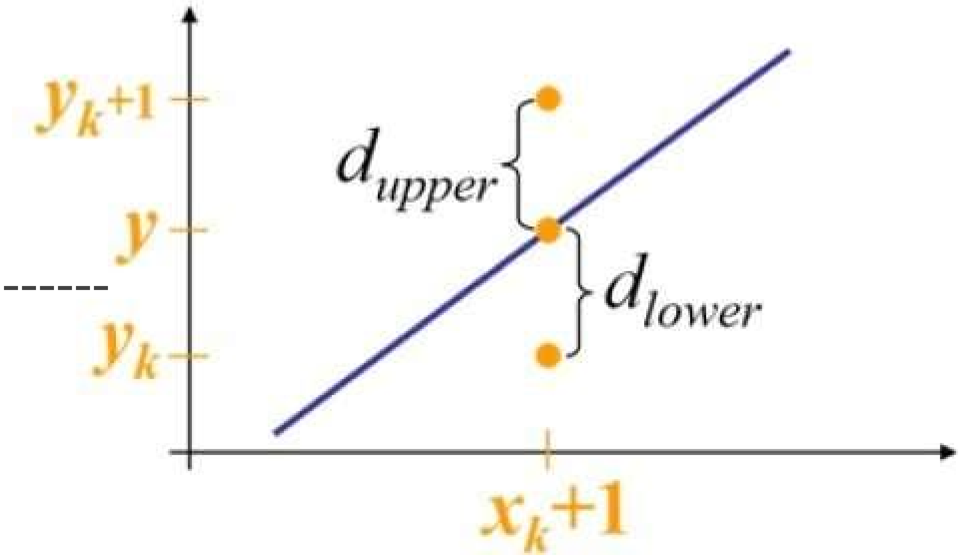

Image Source: https://www.tutorialspoint.com/computer_graphics/line_generation_algorithm.htm

What to choose

- $X_{next} = X_k + 1$
- $Y_{next} = Y_k + 1$ (OR) Y_k
- $Y = m(X_k + 1) + c$ ---- will be floating value but we need an integer value

Let's start the derivation

- $d_{lower} = Y - Y_k = [\, m(X_k + 1) + c \,] - Y_k = m(X_k + 1) + c - Y_k$
- $d_{upper} = Y_k + 1 - Y = Y_k + 1 - [\, m(X_k + 1) + c \,] = Y_k + 1 - m(X_k + 1) - c$
- To understand which one to choose from Y_k and $Y_k + 1$, we can verify by
  - If $(d_{lower} - d_{upper}) < 0$ then, $Y_{next} = Y_k$
  - If $(d_{lower} - d_{upper}) > 0$ then, $Y_{next} = Y_k + 1$
- So, $(d_{lower} - d_{upper})$ can act as a decision parameter

Derivation Continued……

- $d_{lower} - d_{upper} = [m(X_k + 1) + c - Y_k] - [Y_k + 1 - m(X_k + 1) - c]$
- $d_{lower} - d_{upper} = m(X_k + 1) + c - Y_k - Y_k - 1 + m(X_k + 1) + c$
- $d_{lower} - d_{upper} = 2m(X_k + 1) - 2Y_k + 2c - 1$
- Substituting m = �y / �x, but this might give a float.
- So how to get rid of it. We can get rid of the denominator �x altogether
- $d_{lower} - d_{upper}$ = 2(�y / �x)(X_k + 1) - 2Y_k + 2c - 1
- �x($d_{lower} - d_{upper}$) = �x[2(�y / �x)(X_k + 1) - 2Y_k + 2c - 1]
- �x($d_{lower} - d_{upper}$) = 2�y(X_k + 1) - 2�xY_k + 2�xc - �x

Derivation Continued……

- $�x(d_{lower} - d_{upper}) = 2�y(X_k + 1) - 2�xY_k + 2�xc - �x$
- $�x(d_{lower} - d_{upper}) = 2�yX_k - 2�xY_k + 2�y + 2�xc - �x$
- So we can now form our decision variable that is P_k
- $P_k = �x(d_{lower} - d_{upper}) = 2�yX_k - 2�xY_k + \underline{2�y + 2�xc - �x}$----Constant
- $P_k = �x(d_{lower} - d_{upper}) = 2�yX_k - 2�xY_k$
- $P_{next} = 2�yX_{next} - 2�xY_{next}$
- $P_{next} - P_k = [2�yX_{next} - 2�xY_{next}] - [2�yX_k - 2�xY_k]$
- $P_{next} - P_k = 2�yX_{next} - 2�xY_{next} - 2�yX_k + 2�xY_k$

Derivation Continued……

- $P_{next} - P_k = 2�y(X_{next} - X_k) - 2�x (Y_{next} - Y_k)$
- If $P_{next} - P_k < 0$, then remain on the same size for Y_{next} that is Y_k
  - $P_{next} = P_k + 2�y(X_k +1 - X_k) - 2�x (Y_k - Y_k)$
  - $P_{next} = P_k + 2�y$
- If $P_{next} - P_k >= 0$, then Y_{next} is $Y_k + 1$
  - $P_{next} = P_k + 2�y(X_k +1 - X_k) - 2�x (Y_k + 1 - Y_k)$
  - $P_{next} = P_k + 2�y - 2�x$

Derivation Continued……

- If $P_{k+1} - P_k < 0$, then remain on the same size for Y_{next} that is Y_k
    - $P_{k+1} = P_k + 2�y(X_k +1 - X_k) - 2�x (Y_k - Y_k)$
    - $P_{k+1} = P_k + 2�y$
- If $P_{k+1} - P_k >= 0$, then Y_{next} is $Y_k + 1$
    - $P_{k+1} = P_k + 2�y(X_k +1 - X_k) - 2�x (Y_k + 1 - Y_k)$
    - $P_{k+1} = P_k + 2�y - 2�x$

Derivation Continued……

- Now let’s find out the initial value of P_k
- $P_1 = 2�yX_1 - 2�xY_1 + \underline{2�y + 2�xc - �x}$
- As $y_1 = mx_1 + c$, we can substitute ‘c’ as $c = y_1 - mx_1 = y_1 - [�y / �x]\ x_1$
- $P_1 = 2�yX_1 - 2�xY_1 + 2�y + 2�x[y_1 - (�y / �x)\ x_1] - �x$
- $P_1 = 2�yX_1 - 2�xY_1 + 2�y + 2�xy_1 - 2�yx_1 - �x$
- $P_1 = 2�y - �x$

Bresenham Algo

```
Algo_Bresenham(x1, y1, x2, y2)
{
        dx = x2 – x1
        dy = y2 – y1

        P = 2dy - dx

                for(i = 0 to dx)
                {
                put_pixel ( x, y )
                x++
                        If (P < 0)
                                {
                                P = P + 2dy
                                }
                        else
                                {
                                P = P + 2dy - 2dx
                                y++
                                }
                }
}
```

Examples for Practice

- Indicate which raster locations would be chosen by Bresenham's algorithm when scan-converting a line from pixel coordinate (1,1) to pixel coordinate (8,5).
- Write and explain bresenham's line drawing alogrithm and find out which pixel would be turned on for the line with end points (3,2) to (7,4) using the same.
- What are the steps involved in Bresenham line drawing algorithm for line (0,0) to (-8,-5)?
- Explain Bresenham line drawing algorithm with proper mathematical analysis and identify the pixel positions along a line between A(10,10) and B(18,16). **(May-18/10 Marks)**

Summary

	Digital Differential Analyzer Line Drawing Algorithm	Bresenhams Line Drawing Algorithm
Arithmetic	DDA algorithm uses **floating points** i.e. **Real Arithmetic**.	Bresenhams algorithm uses **fixed points** i.e. **Integer Arithmetic**.
Operations	DDA algorithm uses **multiplication** and **division** in its operations.	Bresenhams algorithm uses only **subtraction** and **addition** in its operations.
Speed	DDA algorithm is rather **slow**ly than Bresenhams algorithm in line drawing because it uses real arithmetic (floating-point operations).	Bresenhams algorithm is faster than DDA algorithm in line drawing because it performs only addition and subtraction in its calculation and uses only integer arithmetic so it runs significantly **faster**.
Accuracy & Efficiency	DDA algorithm is not as accurate and efficient as Bresenham algorithm.	Bresenhams algorithm is more efficient and much accurate than DDA algorithm.
Drawing	DDA algorithm can draw circles and curves but that are not as accurate as Bresenhams algorithm.	Bresenhams algorithm can draw circles and curves with much more accuracy than DDA algorithm.
Round Off	DDA algorithm round off the coordinates to integer that is nearest to the line.	Bresenhams algorithm does not **round off** but takes the incremental value in its operation.
Expensive	DDA algorithm uses an enormous number of floating-point multiplications so it is expensive.	Bresenhams algorithm is less expensive than DDA algorithm as it uses only addition and subtraction.

References

- Hearn & Baker, “Computer Graphics C version”, 2nd Edition, Pearson Publication
- http://pseudobit.blogspot.com/2015/03/graphic-design-bresenham-line-algorithm.html