Digital Differential Analyzer Algorithm (DDA)

Scan conversion is the process of representing continuous graphical objects as a collection of discrete pixel values

A **scan conversion algorithm** for lines computes the coordinates of the pixels that lie on a near an ideal infinitely thin straight line imposed on a 2D Raster grid.

It traces out successive (x,y) values by simultaneously incrementing x and y by small steps proportional to the first derivative of x and y.

Slope intercept equation of a straight line is $y = m x + c$

For points (x_1, y_1) and (x_2,y_2) slope $m = y_2 – y_1/x_2 - x_1$ or $\Delta y/\Delta x$

Algorithms for displaying lines depend on these equations for given interval x we can compute y interval

$\Delta y = m . \Delta x$ 'x' interval Δx is obtained by $\Delta x = \Delta y/m$

These equations form the basis for determining deflection voltages in analog devices

For lines with slope m = 1, Δy and Δx are equal, DDA scan conversion algorithm is based on calculating Δx, Δy

We sample line at unit interval in one coordinate and determine corresponding integer values nearest the line path for other coordinate

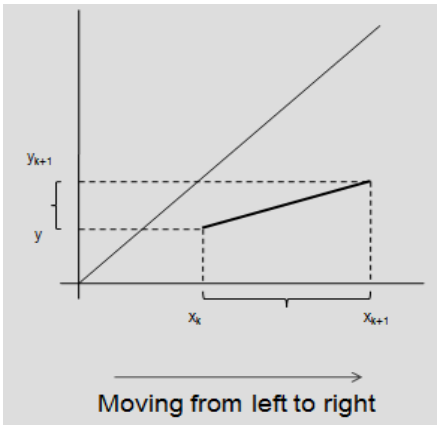
For Lines with Positive Slope

i. For Lines with |m| <=1 (Moving from Left to Right)

The simplest strategy for scan conversion of lines is to compute the slope 'm' as $\Delta y/\Delta x$ to increment x by 1 pixel starting with the leftmost point to calculate the next value of y_i, for each x and to intensify the pixel at $(x_i,|y_i|)$. Then

$m = y_{k+1} – y_k / x_{k+1} – x_k$

$y_{k+1} = y_k + m.(x_{k+1} – x_k)$



And if $\Delta x = 1$ then $y_{k+1} = y_k + m$. Thus the unit change in x changes y by m , which is the slope of the line. For all points (x_k,y_k) on the line, we know that if $x_{k+1} = x_k + 1$ then $y_{k+1} = y_k + m$ i.e. the values of x and y are defined in terms of their previous values

Similarly

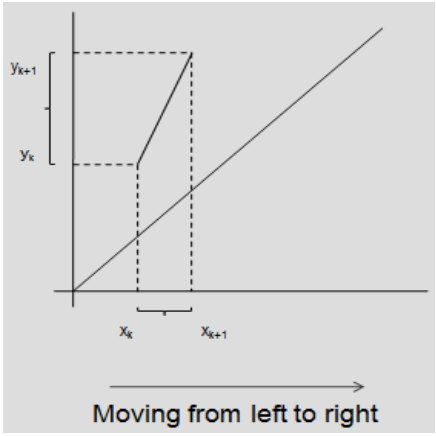
ii. For Lines with slope > 1 (Moving from Left to Right)

Perform unit increment in y direction $\Delta y = 1$ as $\Delta y > \Delta x$ i.e. $y_{k+1} – y_k = 1$ and compute successive x value as

$x_{k+1} – x_k = 1/m$

or $x_{k+1} = x_k + 1/m$

The x value computed must be rounded off to the nearest whole number



iii. For Lines with slope < 1 (Moving from Right to Left)

Perform unit increment in x direction $\Delta x = -1$ as $\Delta x > \Delta y$ i.e. $x_{k+1} - x_k = -1$ and compute successive y value as

$y_{k+1} - y_k = - m$

or $y_{k+1} = y_k - m$

The y value computed must be rounded off to the nearest whole number

iv. For Lines with slope > 1 (Moving from Right to Left)

Perform unit increment in y direction $\Delta y = -1$ as $\Delta y > \Delta x$ i.e. $y_{k+1} - y_k = -1$ and compute successive x value as

$x_{k+1} - x_k = -1/m$

or $x_{k+1} = x_k - 1/m$

The x value computed must be rounded off to the nearest whole number

For Lines with Negative Slope

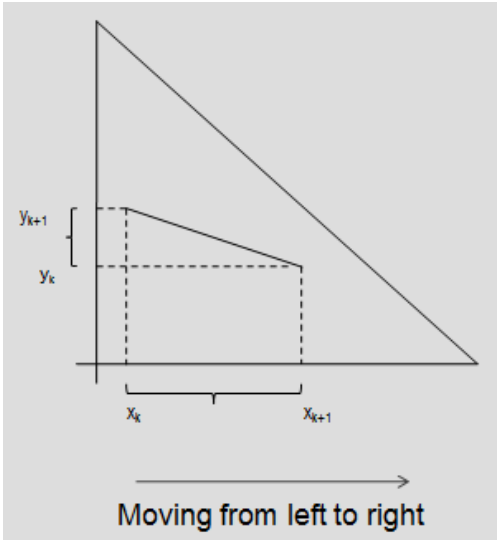
v. For Lines with |m| <=1 (Moving from Left to Right)

Perform unit increment in x direction $\Delta x = 1$ as $\Delta x > \Delta y$ i.e. $x_{k+1} - x_k = 1$ and compute successive y value as

$y_{k+1} - y_k = m$

or $y_{k+1} = y_k + m$

The y value computed must be rounded off to the nearest whole number



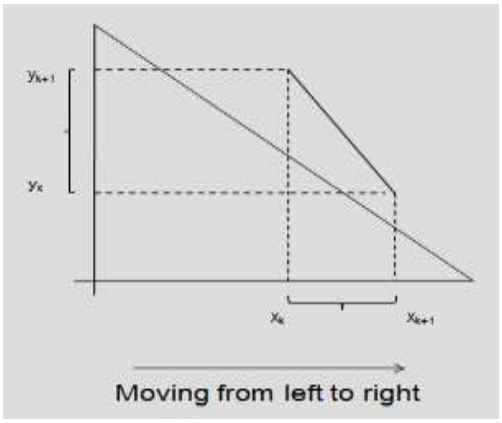
vi. For Lines with |m| > 1 (Moving from Left to Right)

Perform unit increment in y direction $\Delta y = -1$ as $\Delta y > \Delta x$ i.e. $y_{k+1} - y_k = -1$ and compute successive x value as

$x_{k+1} - x_k = - 1/m$

or $x_{k+1} = x_k - 1/m$

The x value computed must be rounded off to the nearest whole number



vii. For Lines with |m| < 1 (Moving from Right to Left)

Perform unit increment in x direction $\Delta x = - 1$ as $\Delta x > \Delta y$ i.e. $x_{k+1} - x_k = -1$ and compute successive y value as

$y_{k+1} - y_k = - m$

or $y_{k+1} = y_k - m$

The y value computed must be rounded off to the nearest whole number

viii. For Lines with |m| > 1 (Moving from Right to Left)

Perform unit increment in y direction $\Delta y = 1$ as $\Delta y > \Delta x$ i.e. $y_{k+1} - y_k = 1$ and compute successive x value as

$x_{k+1} - x_k = 1/m$

or $x_{k+1} = x_k + 1/m$

The x value computed must be rounded off to the nearest whole number

This algorithm is based on floating point arithmetic so it is slower than Bresenham's line drawing algorithm for drawing lines as Bresenham's line drawing algorithm is based on integer arithmetic approach.

Algorithm:

```
void lineDDA (int xa, int ya, int xb, int yb){
    int dx = xb - xa, dy = yb - ya, steps, k;
    float xIncrement, yIncrement, x = xa, y = ya;
        if (abs (dx) > abs (dy))  steps = abs (dx) ;
        else steps = abs (dy);
      xIncrement = dx / (float) steps;
      yIncrement = dy / (float) steps
      setpixel (ROUND(x), ROUND(y));
      for (k=0; k<steps; k++){
         x += xIncrement;
         y += yIncrement;
        setpixel (ROUND(x), ROUND(y));
      }
}
```

Advantages:

i. Lines are drawn with more accuracy

Disadvantages :

i. It is slower as it uses floating point arithmetic
ii. The accumulation of round off errors makes it error prone

Numerical

Digitize a line with end points A(5,2), B(10,4) using DDA

$m = 2/5 = 0.4$

As the slope of the line is less than one the equation used is

$x_{k+1} = x_k + 1$ $\quad\quad$ $y_{k+1} = y_k + m$

Starting pixel (5,2)

Step	x_{k+1}	y_{k+1}	Rounded Value of y_{k+1}	Pixel Plotted
1.	6	$y_1 = 2 + 0.4 = 2.4$	2	(6,2)
2.	7	$y_2 = 2.4 + 0.4 = 2.8$	3	(7,3)
3.	8	$y_3 = 2.8 + 0.4 = 3.2$	3	(8,3)
4.	9	$y_4 = 3.2 + 0.4 = 3.6$	4	(9,4)
5.	10	$y_5 = 3.6 + 0.4 = 4$	4	(10,4)