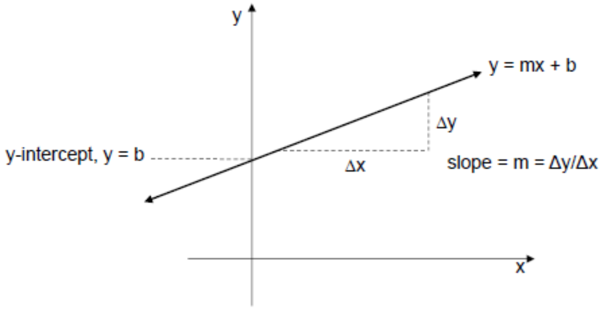
Digital Differential Analyzer (DDA) algorithm

A line can be defined by…

… its slope m and its y-intercept b.

Points on the line satisfy the equation y = mx + b



The Cartesian *slope-intercept equation* for a straight line is

$$y = m \cdot x + b \qquad (3\text{-}1)$$

with *m* representing the slope of the line and b as they intercept. Given that the two endpoints of the segment at positions *(x,* y,) and *(x,* y), we can determine values for the slope m and y intercept b with the following calculations

$$m = \frac{y_2 - y_1}{x_2 - x_1} \qquad (3\text{-}2)$$

$$b = y_1 - m \cdot x_1 \qquad (3\text{-}3)$$

Algorithms for displaying straight lines are based on the line equation 3-1 and the calculations given in Eqs. 3-2 and 3-3.

For any given *x* interval Δx along a line, we can compute the corresponding y interval Δy from Eq. 3-2 as

$$\Delta y = m \, \Delta x \qquad (3\text{-}4)$$

Similarly, we can obtain the *x* interval Δx

Line DDA algorithm is the simple line generation algorithm which is explained step by step here.

LineDDA algorithm

Step 1 − Get the input of two end points (X0,Y0) and (X1,Y1)

Step 2 − Calculate the difference between two end points.

```
  dx = $X_1$ - $X_0$
  dy = $Y_1$ - $Y_0$
```

Step 3 − Based on the calculated difference in step-2, you need to identify the number of steps to put pixel. If dx > dy, then you need more steps in x coordinate; otherwise in y coordinate.

```
if (dx > dy) then  Steps = absolute(dx);
   else    Steps = absolute(dy);
```

Step 4 − Calculate the increment in x coordinate and y coordinate.

```
Xincrement = dx / (float) steps;
Yincrement = dy / (float) steps;
```

Step 5 − Put the pixel by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

```
    X=X0, Y=Y0
    For I=1 to step do
        Plot(X,Y,color)
        X=X+Xincrement
        Y=Y+Yincremen
```

Square1 algorithm

```
Step 1 − Get the input of two points  in square (X1,Y1), (X2,y2)
Step 2 − Call LinDDA(X1,Y1,X2,y1,color)
Step 3 − Call LinDDA(X2,Y1,X2,y2,color)
Step 4 − Call LinDDA(X2,Y2,X1,y2,color)
Step 5 − Call LinDDA(X1,Y2,X1,y1,color)
```

Square2 algorithm

```
Step 1 − Get the input of Center point (Xc,Yc) and width S.

Step 2 − Calculate the two points of square.

  X1=Xc-(S div 2)
  Y1=Yc-(S div 2)
  X2=Xc+(S div 2)
  Y2=Yc+(S div 2)

 Step 3 − Call Square (X1,Y1,X2,Y2,color)
```

Polyline algorithm

```
Step 1 − Get the input of Number of Vertex N.
Step 2 − Get the input of coordinate of points in two arrays X,Y.

Step 3 – Draw the edge of polyline .

  if N=1 then plot (x[1],y[1],color);
    else    For k=1 to N-1 do
           Call  Lindda (X[k],y[k],X[k-1],Y[k-1],color)
```

-

Bresenham’s Line Generation

The Bresenham algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that, it uses only integer calculations.

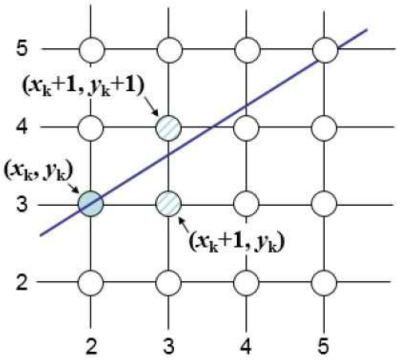
Difference Between DDA Line Drawing Algorithm and Bresenhams Line Drawing Algorithm.

	Digital Differential Analyzer	***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 slowly 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.*

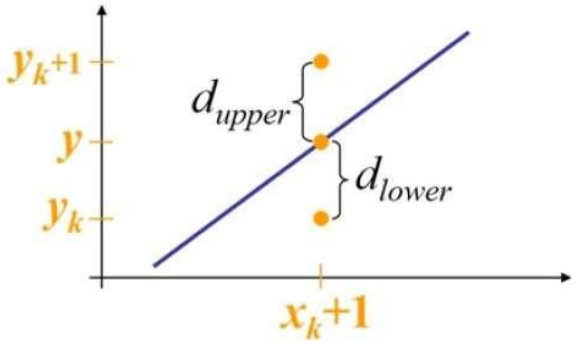
Expensive	*DDA algorithm uses an enormous number of floating-point multiplications so it is expensive.*	*Bresenhams algorithm is less expensive than DDA algorithm as it uses only addition and subtraction.*

Moving *across the x axis* in unit intervals and at each step choose between two different y coordinates.

For example, as shown in the following illustration, from position (2, 3) you need to choose between (3, 3) and (3, 4). You would like the point that is closer to the original line.



At sample position X_k+1, the vertical separations from the mathematical line are labeled as d_{upper} and d_{lower}.



y_k+1
y
y_k
d_{upper}
d_{lower}
x_k+1