Line Drawing Algorithms

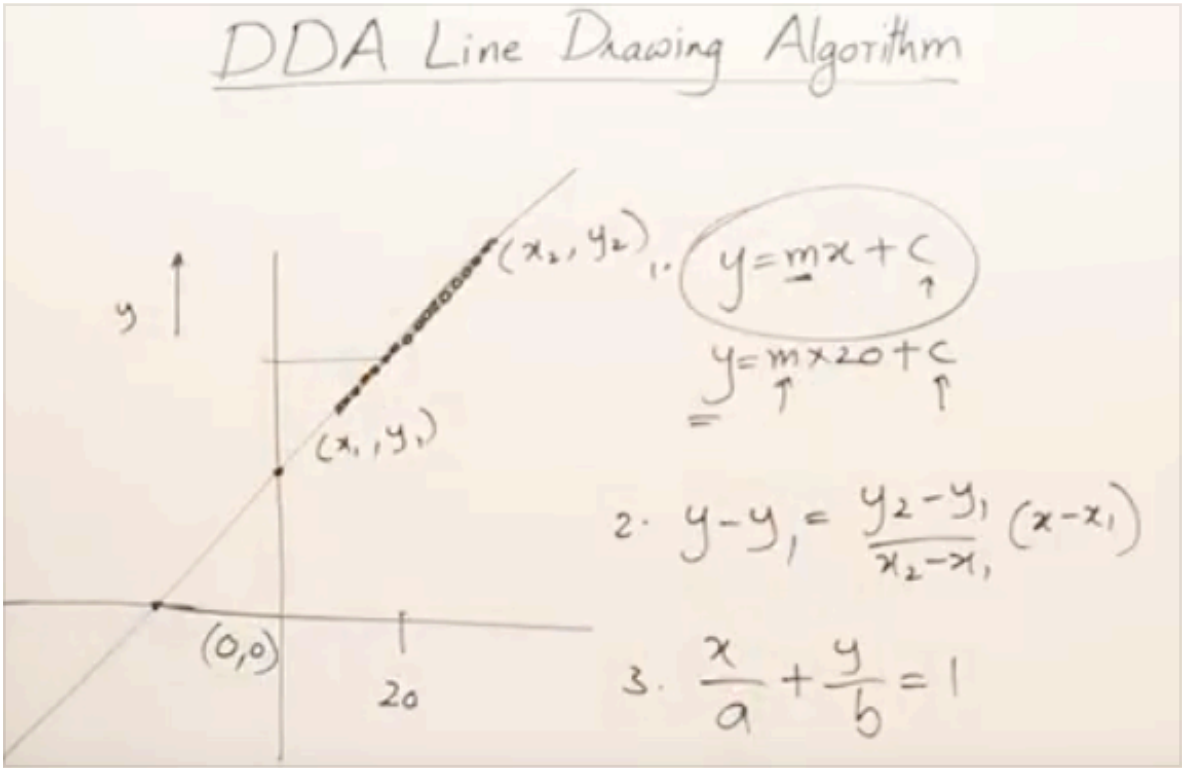
All these line algorithms are designed to demonstrate how raster scan actually draws lines on the screen in pixels grid. The problem occurs when we need to draw lines at some angles where
We got confused how to choose between 2 pixels at a weird angle of line maybe 37 or 63 something.

We have the general equation for a straight line in mathematics is :
Y = mx + c
Where y, & x are both of our axis &
C = the point where the line cuts the y axis
M = slope or gradient we can say sometimes

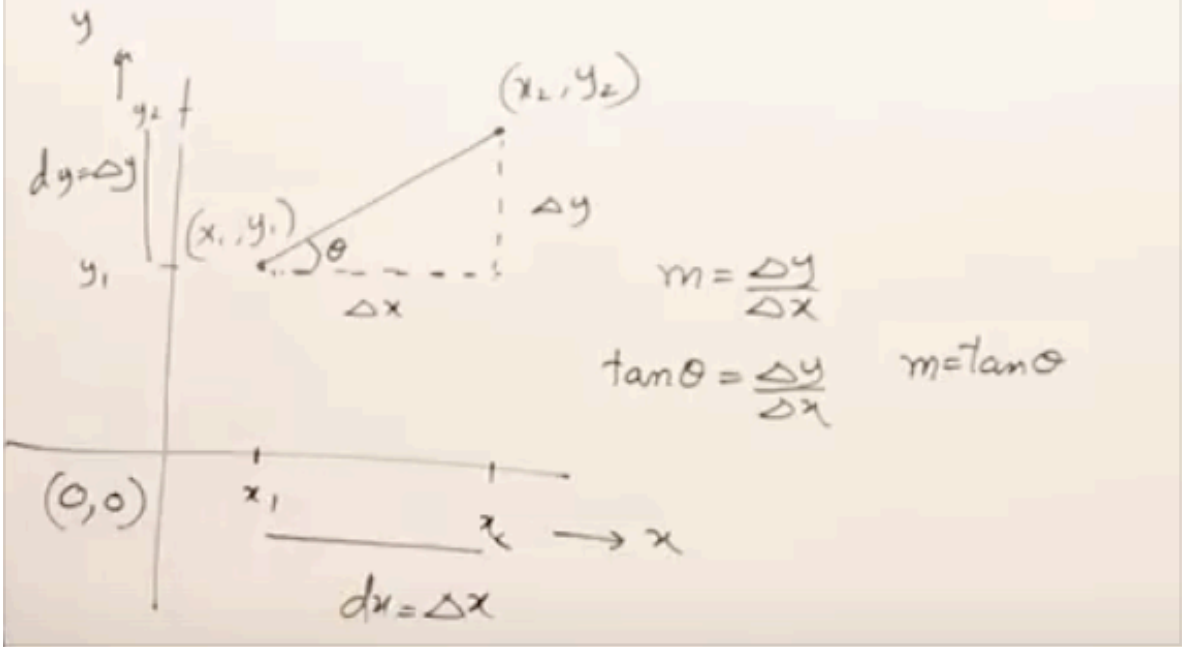
Slope of line can be calculated by m = y2 - y1 / x2 - x1 = del y / del x = dy/dx ;

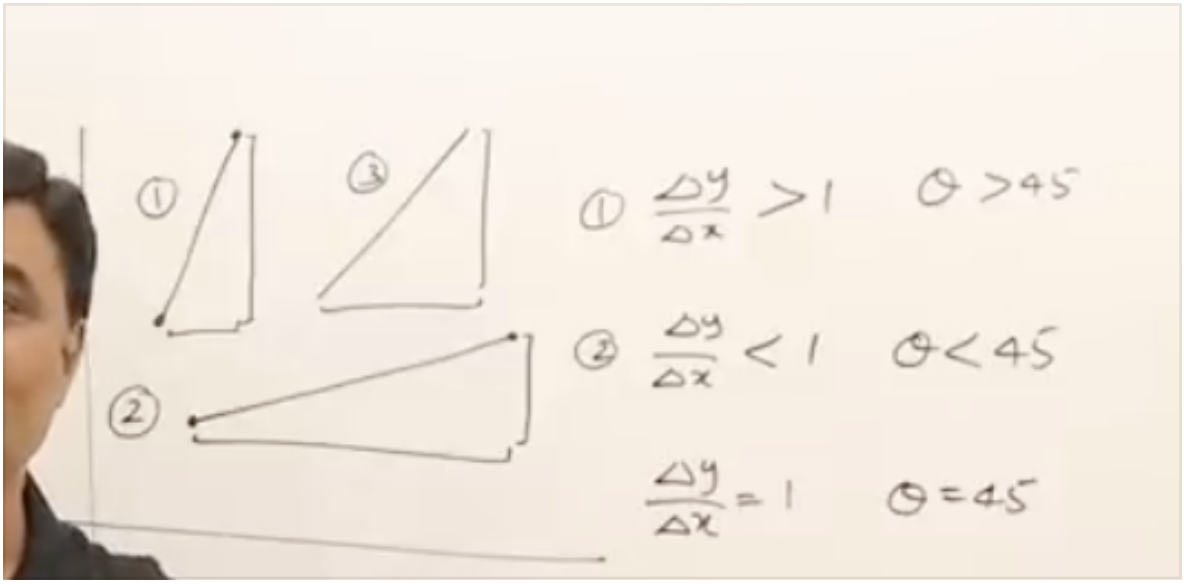
We also have other equations for a line

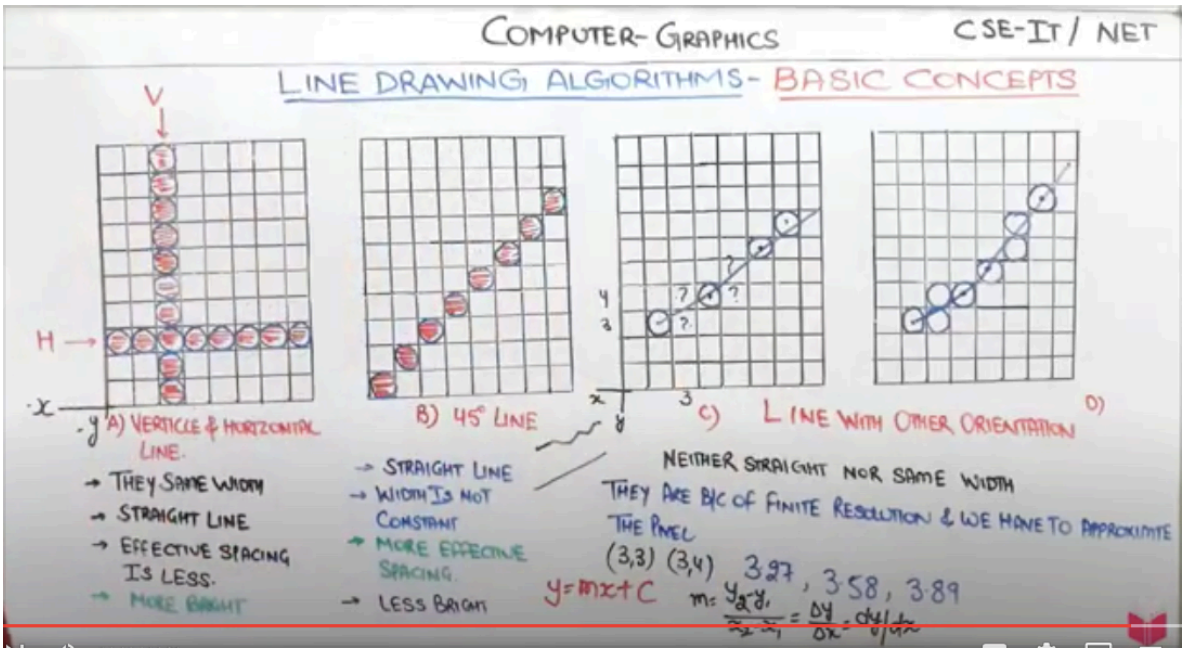
Sin.	Cos.	Tan
P	B.	P
H	H.	B
Cosec	Sec	Cot



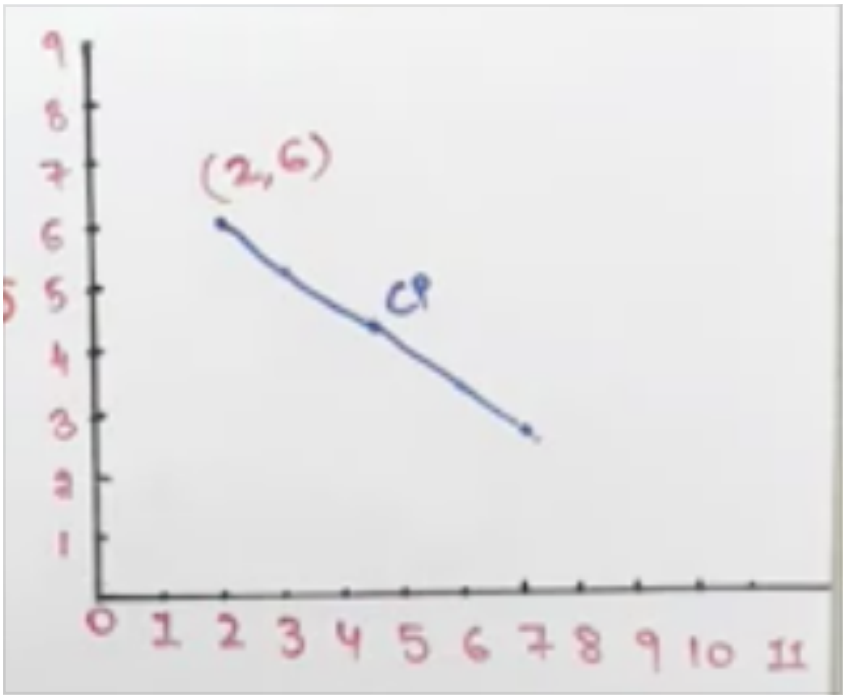
DDA Line Drawing Algorithm







Line Increment Algorithm :



1. Curr Position = Start
   Step = Increment
2. If(|currPosition-end| < Accuracy then foto step5)
   If(currPosition < End) Then goto step 3
   If(currPosition > End) Then goto step 4
3. CurrPosition = CurrPosition + Step
   Goto Step 2
4. CurrPosition = CurrPosition- Step goto step 2
5. Stop

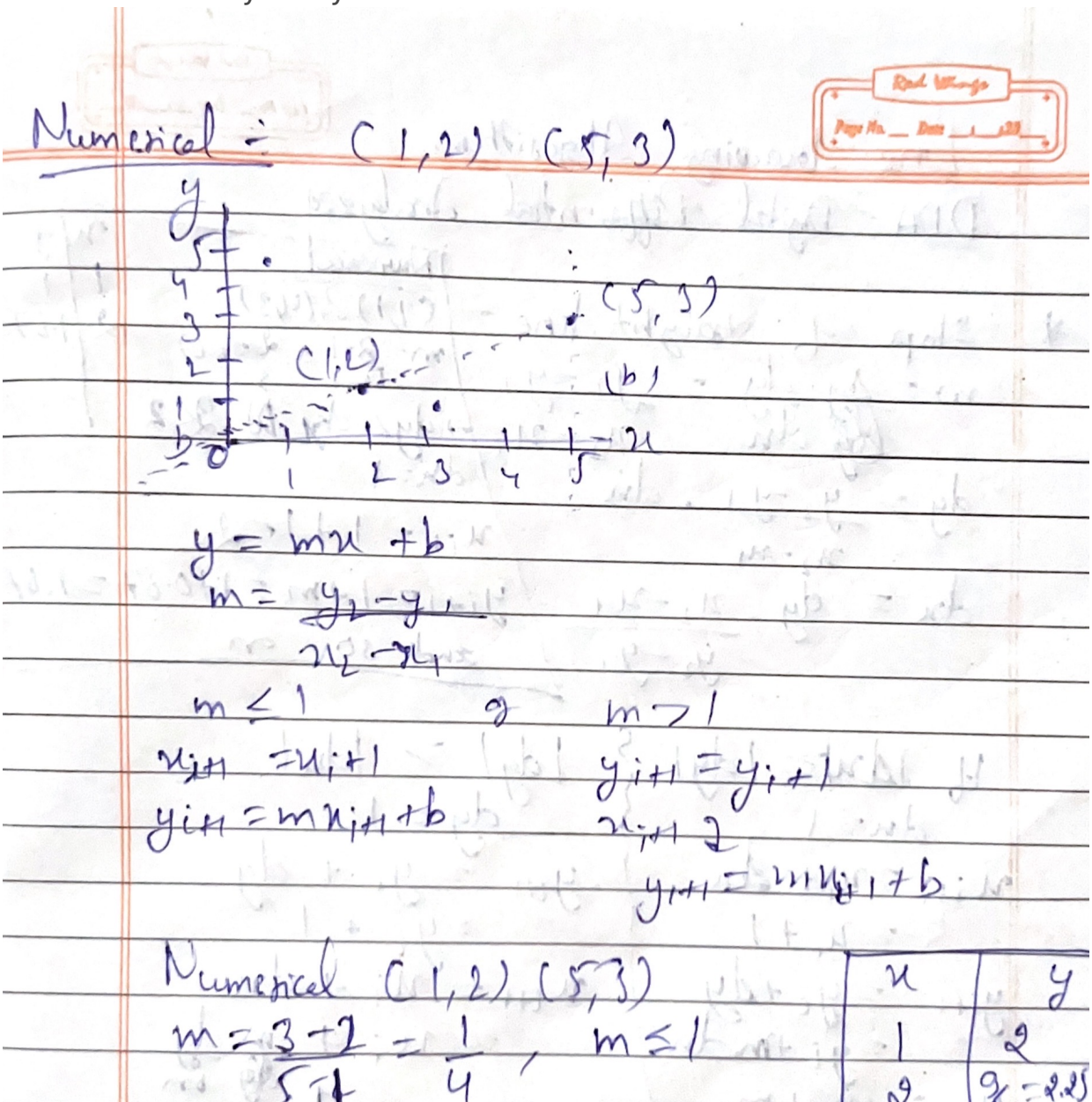
Direct Method Of Line Drawing

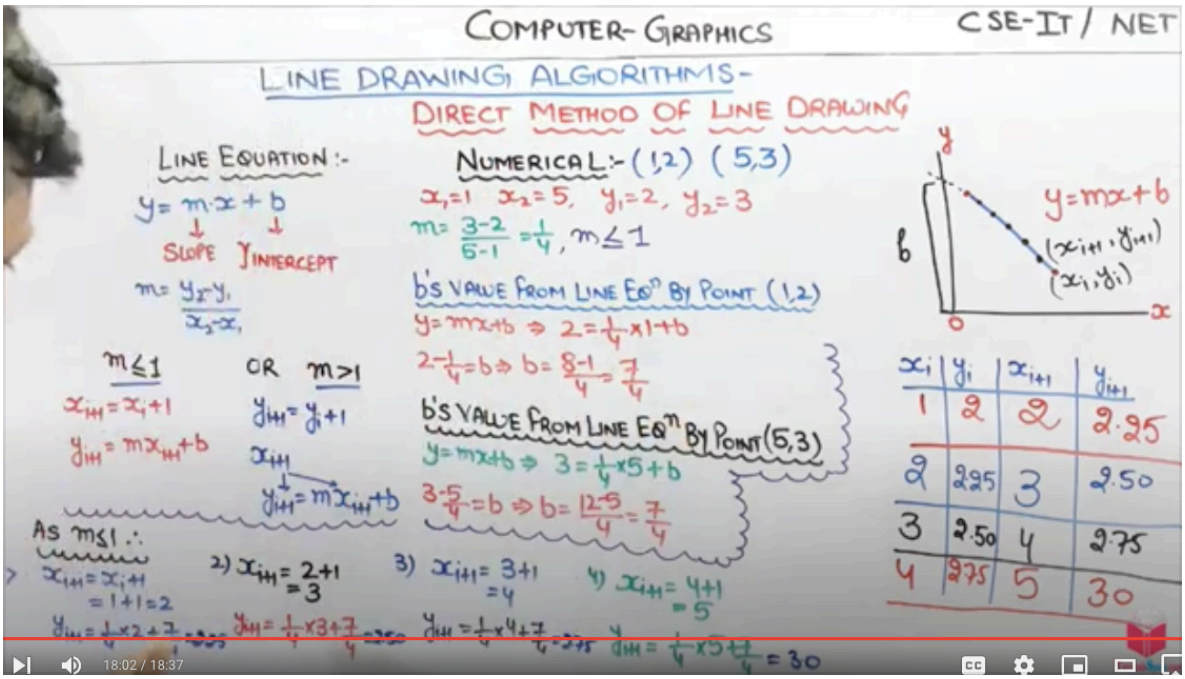
As we know now, that the Line equation is y = mx + b
Where b = intersection point with y axis
And m is the slope of line which usually calculated my the angle of the line with x axis m = tan @ = p/b or m = y2-y1/x2-x1

Now with the value of m comes out to be <= 1
Xi+1 = xi + 1;
Yi+1 = mxi+1+b;

Or if m comes out be m > 1
Yi+1=yi + 1;
xi+1 = (yi+1=mx+1+b)

Yes I solved it my self yeaasssssssss !

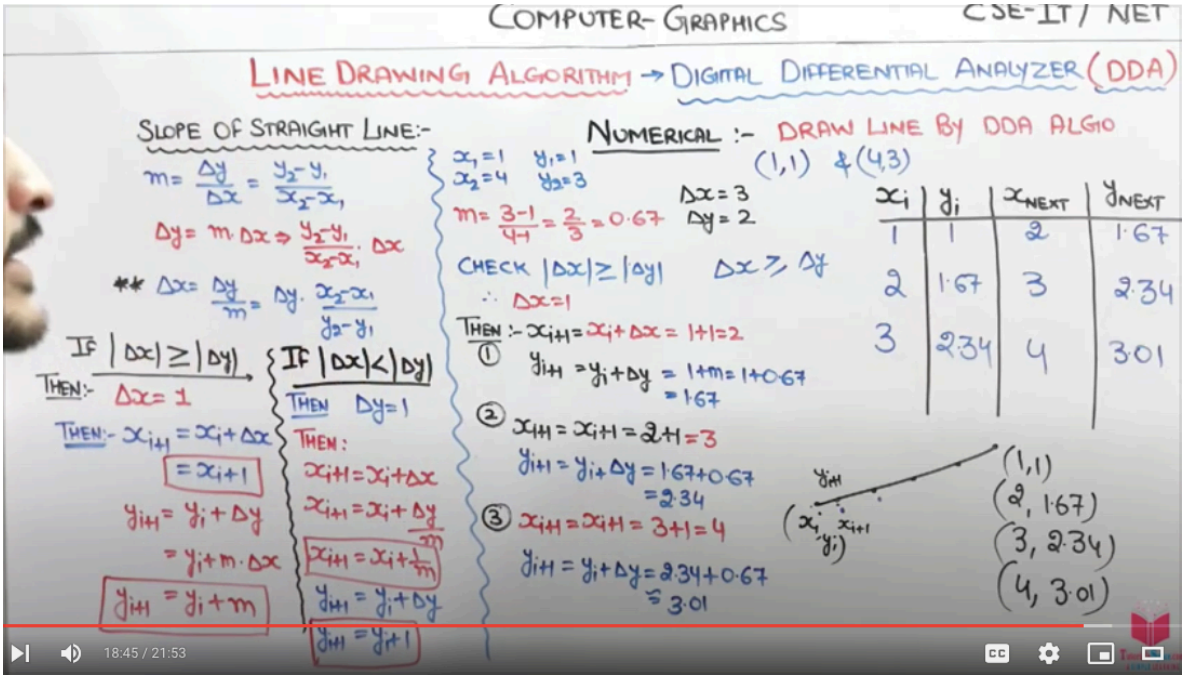




DDA - Digital Differential Analyzer, Line Drawing Algorithm

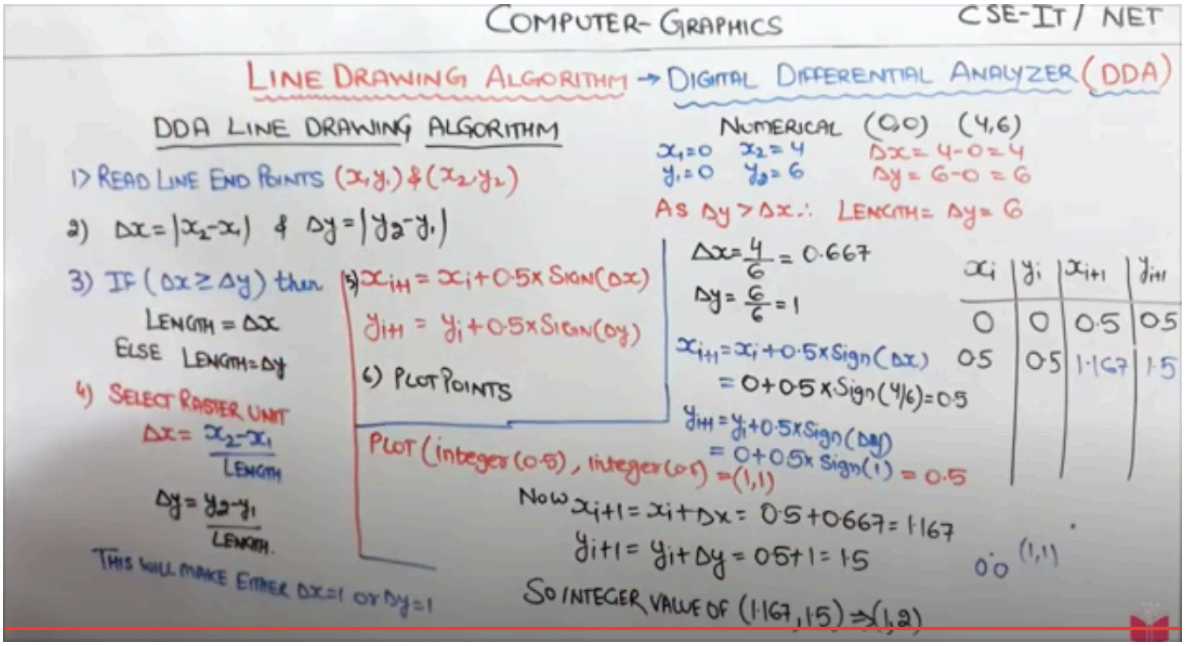
Steps for 1st DDA Algorithm

1). M = dy/dx = y2-y1/x2-x1
2). Dy = m.dx = y2-y1/x2-x1 * dx
3). Dx = dy/m = dy * x2-x1/y2-y1
4). If(|dx| >= |dy| x = 1. xi+1 = xi + dx = xi+1 && yi+1 = yi + dy = yi+m.dx = yi+m
Else y = 1 xi+1=xi+dx. xi+1 = xi+dy/m = xi + 1/m && yi+1=yi+dy = yi+1



Actually we can do the dda algorithm via 2 ways in the first one we have problem the the raster values that we got after calculation for a line
Are in points which make the line we draw kind of a zigzag line. And we can use this 2nd method where we approx the value with 0.5 sin(dx) and
Make it a integer for raster value but keep same for calculating other values

2nd Algorithm steps
1). Read Line End Points (x, y) & (x2, y2)
2). Dx = [x2 - x1] & dy=|y2-y1|
3). If(dx >= dy) lengrh = dx
Else length = dy
4). Selected Raster Unit will be
Dx = x2-x1/length
Dy = y2-y1/length
This will make either dx=1 or dy=1
5). xi+1 = xi + 0.5 Sign(dx)
yi+! = yi + 0.5 Sign(dy)
6). Plot Points -> round of end of the points for values but for calculation we gonna use exact points ....



Bresenham's Line Algorithm

This algorithm is used for scan converting a line. It was developed by Bresenham. 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.

In this method, next pixel selected is that one who has the least distance from true line.

The method works as follows:

Assume a pixel $P_1'(x_1',y_1')$,then select subsequent pixels as we work our may to the night, one pixel position at a time in the horizontal direction toward $P_2'(x_2',y_2')$.

Once a pixel in choose at any step

The next pixel is

1. Either the one to its right (lower-bound for the line)
2. One top its right and up (upper-bound for the line)

The line is best approximated by those pixels that fall the least distance from the path between P_1',P_2'.

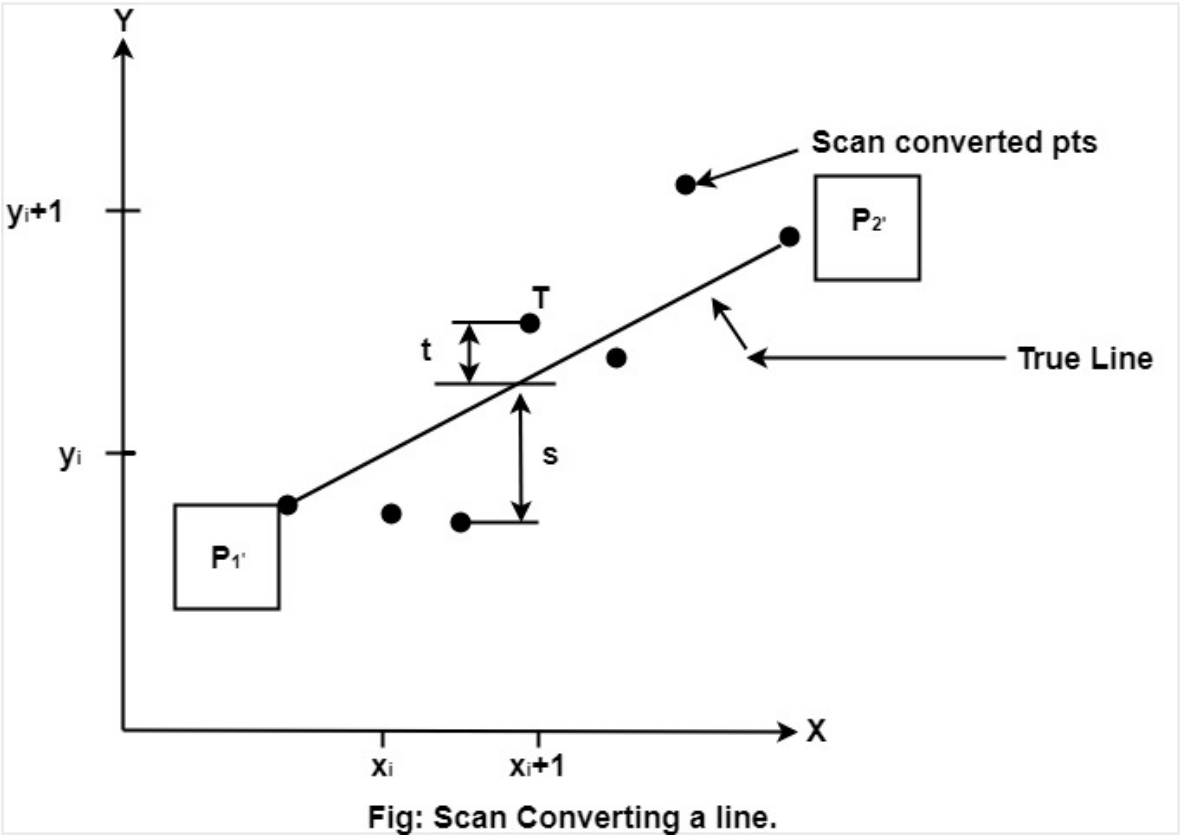


Fig: Scan Converting a line.

To chooses the next one between the bottom pixel S and top pixel T.

If S is chosen

We have $x_{i+1}=x_i+1$ and $y_{i+1}=y_i$

If T is chosen

We have $x_{i+1}=x_i+1$ and $y_{i+1}=y_i+1$

The actual y coordinates of the line at $x = x_{i+1}$ is

$y=mx_{i+1}+b$

$$y = m(x_i + 1) + b$$

The distance from S to the actual line in y direction

$s = y-y_i$

The distance from T to the actual line in y direction

$t = (y_i+1)-y$

Now consider the difference between these 2 distance values

$s - t$

When $(s-t) < 0 \Longrightarrow s < t$

The closest pixel is S

When $(s-t) \geq 0 \Longrightarrow t < s$

The closest pixel is T

This difference is

s-t = (y-yi)-[(yi+1)-y]

= 2y - 2yi -1

Putting value of y not 1.

$$s - t = 2m(x_i + 1) + 2b - 2y_i - 1$$ [Putting the value of (1)]

Substituting m by

$\frac{\Delta y}{\Delta x}$

and introducing decision variable $d_i=\Delta x\ (s-t)$

$d_i=\Delta x\ (2$

$\frac{\Delta y}{\Delta x}$

$(x_i+1)+2b-2y_i-1)$

$=2\Delta xy_i-2\Delta y-1\Delta x.2b-2y_i\Delta x-\Delta x$

$d_i=2\Delta y.x_i-2\Delta x.y_i+c$

Where c= 2Δy+Δx (2b-1)

We can write the decision variable d_{i+1} for the next slip on

$d_{i+1}=2\Delta y.x_{i+1}-2\Delta x.y_{i+1}+c$

$d_{i+1}-d_i=2\Delta y.(x_{i+1}-x_i)-2\Delta x(y_{i+1}-y_i)$

Since x_(i+1)=x_i+1,we have

$d_{i+1}+d_i=2\Delta y.(x_i+1-x_i)-2\Delta x(y_{i+1}-y_i)$

Special Cases

If chosen pixel is at the top pixel T (i.e., $d_i\geq0)\Longrightarrow y_{i+1}=y_i+1$

$d_{i+1}=d_i+2\Delta y-2\Delta x$

If chosen pixel is at the bottom pixel T (i.e., $d_i<0)\Longrightarrow y_{i+1}=y_i$

$d_{i+1}=d_i+2\Delta y$

Finally, we calculate d_1

$d_1=\Delta x[2m(x_1+1)+2b-2y_1-1]$

$d_1=\Delta x[2(mx_1+b-y_1)+2m-1]$

Since $mx_1+b-y_i=0$ and m =

$\frac{\Delta y}{\Delta x}$

, we have $d_1=2\Delta y-\Delta x$

Advantage:

1. It involves only integer arithmetic, so it is simple.
2. It avoids the generation of duplicate points.
3. It can be implemented using hardware because it does not use multiplication and division.
4. It is faster as compared to DDA (Digital Differential Analyzer) because it does not involve floating point calculations like DDA Algorithm.

Disadvantage:

1. This algorithm is meant for basic line drawing only Initializing is not a part of Bresenham's line algorithm. So to draw smooth lines, you should want to look into a different algorithm.

Bresenham's Line Algorithm:

Step1: Start Algorithm

Step2: Declare variable x_1,x_2,y_1,y_2,d,i_1,i_2,dx,dy

Step3: Enter value of x_1,y_1,x_2,y_2

Where x_1,$y_1$are coordinates of starting point

And x_2,y_2 are coordinates of Ending point

Step4: Calculate dx = x_2-x_1

Calculate dy = y_2-y_1

Calculate i_1=2*dy

Calculate i_2=2*(dy-dx)

Calculate d=i_1-dx

Step5: Consider (x, y) as starting point and x_{end}as maximum possible value of x.

If dx < 0

Then x = x_2

y = y_2

x_{end}=x_1

If dx > 0

Then x = x_1

y = y_1

x_{end}=x_2

Step6: Generate point at (x,y)coordinates.
Step7: Check if whole line is generated.

If $x >= x_{end}$

Stop.

Step8: Calculate co-ordinates of the next pixel

If $d < 0$

Then $d = d + i_1$

If $d \geq 0$

Then $d = d + i_2$

Increment $y = y + 1$

Step9: Increment $x = x + 1$
Step10: Draw a point of latest (x, y) coordinates
Step11: Go to step 7
Step12: End of Algorithm
Example: Starting and Ending position of the line are (1, 1) and (8, 5). Find intermediate points.
Solution: $x_1=1$

$y_1=1$

$x_2=8$

$y_2=5$

$dx= x_2-x_1=8-1=7$

$dy=y_2-y_1=5-1=4$

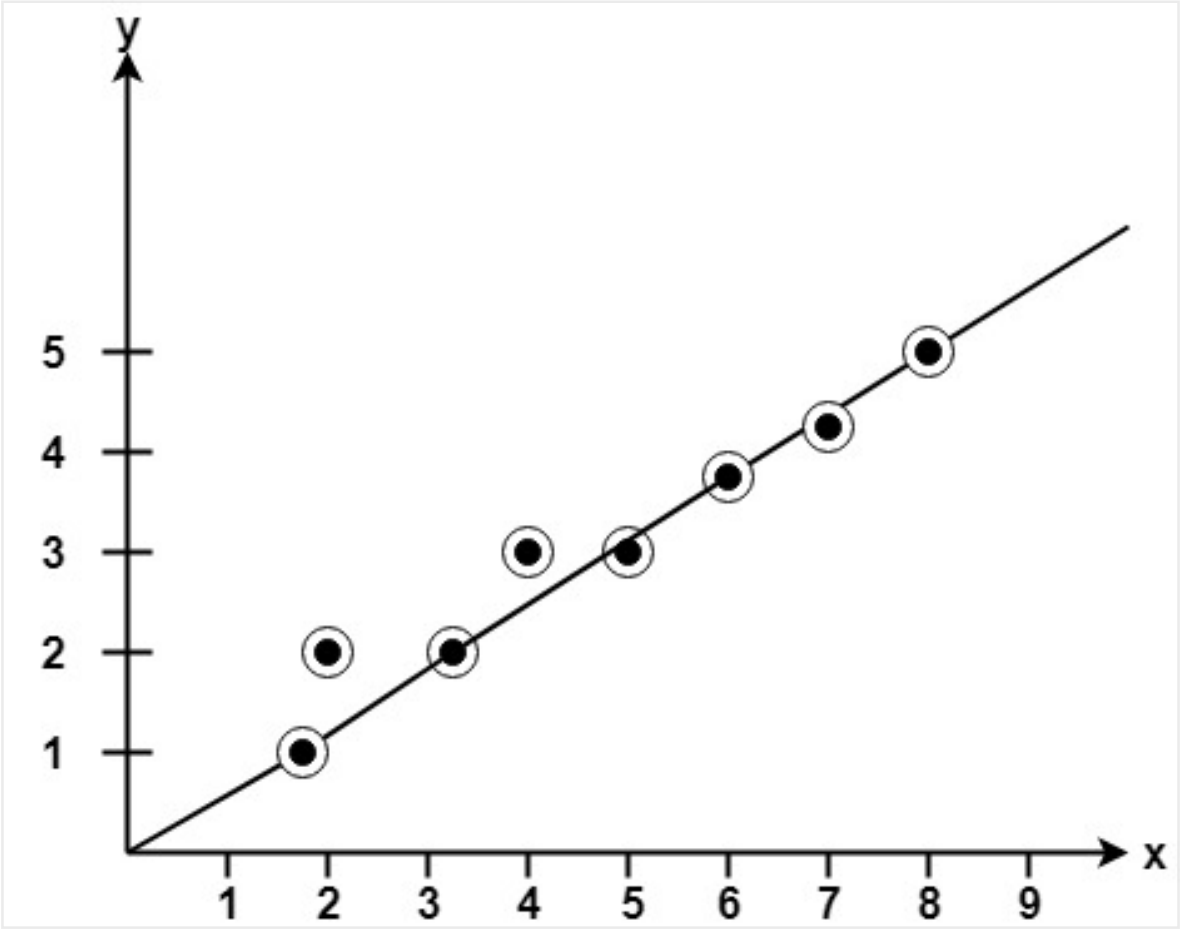
$I_1=2* \Delta y=2*4=8$

$I_2=2*(\Delta y-\Delta x)=2\times(4-7)=-6$

$d = I_1-\Delta x=8-7=1$

x	y	$d=d+I_1$ or I_2
1	1	$d+I_2=1+(-6)=-5$
2	2	$d+I_1=-5+8=3$
3	2	$d+I_2=3+(-6)=-3$
4	3	$d+I_1=-3+8=5$
5	3	$d+I_2=5+(-6)=-1$
6	4	$d+I_1=-1+8=7$
7	4	$d+I_2=7+(-6)=1$

8	5	

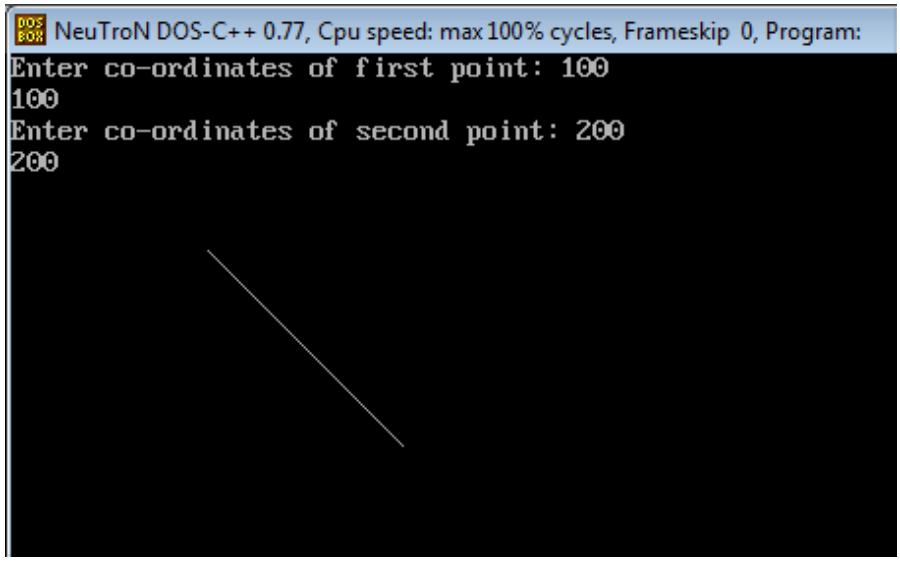


Program to implement Bresenham's Line Drawing Algorithm:

```
#include<stdio.h>
#include<graphics.h>
void drawline(int x0, int y0, int x1, int y1)
{
    int dx, dy, p, x, y;
    dx=x1-x0;
    dy=y1-y0;
    x=x0;
    y=y0;
    p=2×dy-dx;
    while(x<x1)
    {
        if(p>=0)
        {
            putpixel(x,y,7);
            y=y+1;
            p=p+2×dy-2×dx;
```

```
18.         }
19.         else
20.         {
21.             putpixel(x,y,7);
22.             p=p+2*dy;}
23.             x=x+1;
24.         }
25. }
26. int main()
27. {
28.     int gdriver=DETECT, gmode, error, x0, y0, x1, y1;
29.     initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");
30.     printf("Enter co-ordinates of first point: ");
31.     scanf("%d%d", &x0, &y0);
32.     printf("Enter co-ordinates of second point: ");
33.     scanf("%d%d", &x1, &y1);
34.     drawline(x0, y0, x1, y1);
35.     return 0;
36. }
```

Output:



Differentiate between DDA Algorithm and Bresenham's Line Algorithm:

DDA Algorithm	Bresenham's Line Algorithm
1. DDA Algorithm use floating point, i.e., Real Arithmetic.	1. Bresenham's Line Algorithm use fixed point, i.e., Integer Arithmetic

2. DDA Algorithms uses multiplication & division its operation	2.Bresenham's Line Algorithm uses only subtraction and addition its operation
3. DDA Algorithm is slowly than Bresenham's Line Algorithm in line drawing because it uses real arithmetic (Floating Point operation)	3. Bresenham's Algorithm is faster than DDA Algorithm in line because it involves only addition & subtraction in its calculation and uses only integer arithmetic.
4. DDA Algorithm is not accurate and efficient as Bresenham's Line Algorithm.	4. Bresenham's Line Algorithm is more accurate and efficient at DDA Algorithm.
5.DDA Algorithm can draw circle and curves but are not accurate as Bresenham's Line Algorithm	5. Bresenham's Line Algorithm can draw circle and curves with more accurate than DDA Algorithm.

Next TopicDefining a Circle