Coordinate conversion & Scan conversion of line

Course Title: Computer Graphics

Course Code: CSE - 413

Outlines

- Scan converting a point
- Scan converting a line
  - Direct line draw method
  - DDA line draw method
  - Bresenham line draw method

Must Read Questions From This Slide

- **Properties** Of Good Line Drawing Algorithm
- Direct Line Drawing Method : Ex-1,2
- Write Down The DDA **Algorithm**
- DDA Algorithm Math: Ex-1,2, Assignment-4, 7
- **Uses & Issues** Of DDA Line Drawing Algorithm
- Bresenham Line Draw Algorithm math: Ex-1,2 Assignment-3,4
- DDA Vs Bresenham **Difference**

Scan Conversion

We know, the graphics objects are continuous and the pixels are discrete. The process of representing continuous graphics object as a collection of discrete pixels is known as scan conversion. The circuitry of the video display device of the computer is capable of converting binary values (0, 1) into a pixel on and pixel off information. 0 is represented by pixel off. 1 is represented using pixel on. Using this ability graphics computer represent picture having discrete dots. Any model of graphics can be reproduced with a dense matrix of dots or points. Most human beings think graphics objects as points, lines, circles, ellipses. For generating graphical object, many algorithms have been developed. The job of conversion of every primitive object represented on the graphics screen into a set of pixels that is its basic form is termed as scan conversion or rasterization. The conversion of the graphic screen lines and objects depicted in pixels is scan conversion.

Need of scan conversion algorithms

This helps in accelerating the process of scan conversion and can generate graphic objects at a faster rate comparatively. Using these algorithms memory is also used efficiently and the production quality of the objects on the screen is also improved. We can use a random scan system for this process. It uses a beam of electrons that are accelerated through the potential difference and operates like a pencil on the screen of the cathode ray tube thus producing objects on the screen. These are constructed through a particular sequence of lines drawn. The Scan Converting rate is a technique that is used in video processing. In this technique, we can change the horizontal and vertical frequencies for different purposes.

Examples of objects which can be scan converted :- 1. Point 2. Line 3. Circle 4. Sector 5. Arc 6. Ellipse 7. Rectangle 8. Polygon 9. Characters 10. Filled Regions

The algorithms implementation varies from one computer system to another computer system. Some algorithms are implemented using the software. Some are performed using hardware or firmware. Some are performed using various combinations of hardware, firmware, and software.

Advantages of Scan Conversion

- This scan conversion methodology is used for many purposes including TV, the card used to capture a video, LCD monitor, projectors used for video display any more.
- This conversion technique has large and many applications in daily life.
- One can efficiently perform this scan conversion using high and good speed circuits that are integrated.

Disadvantages of Scan Conversion

- While digital scan conversion, the analog video signal gets converted into digital signals of data.
- We can apply scan conversion with only LSI (Large Scale Integration) and VLSI (Very Large Scale Integration) integrated circuits.

Scan Converting a Point

Each pixel on the graphics display does not represent a mathematical point. Instead, it means a region which theoretically can contain an infinite number of points.

Scan-Converting a point involves illuminating the pixel that contains the point.

Pixel position will referenced according to scan-line number and column number which is illustrated by following figure.

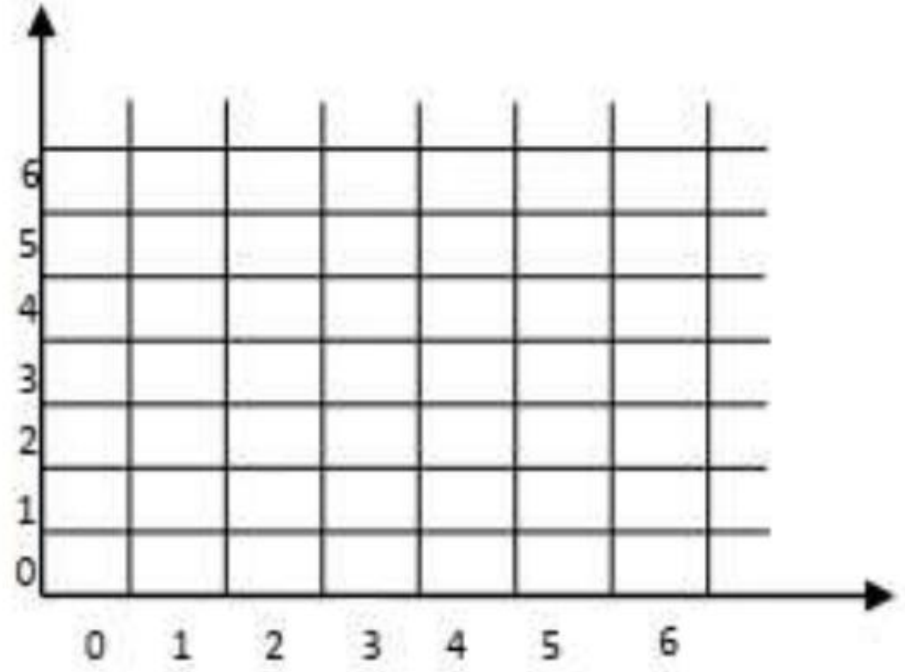
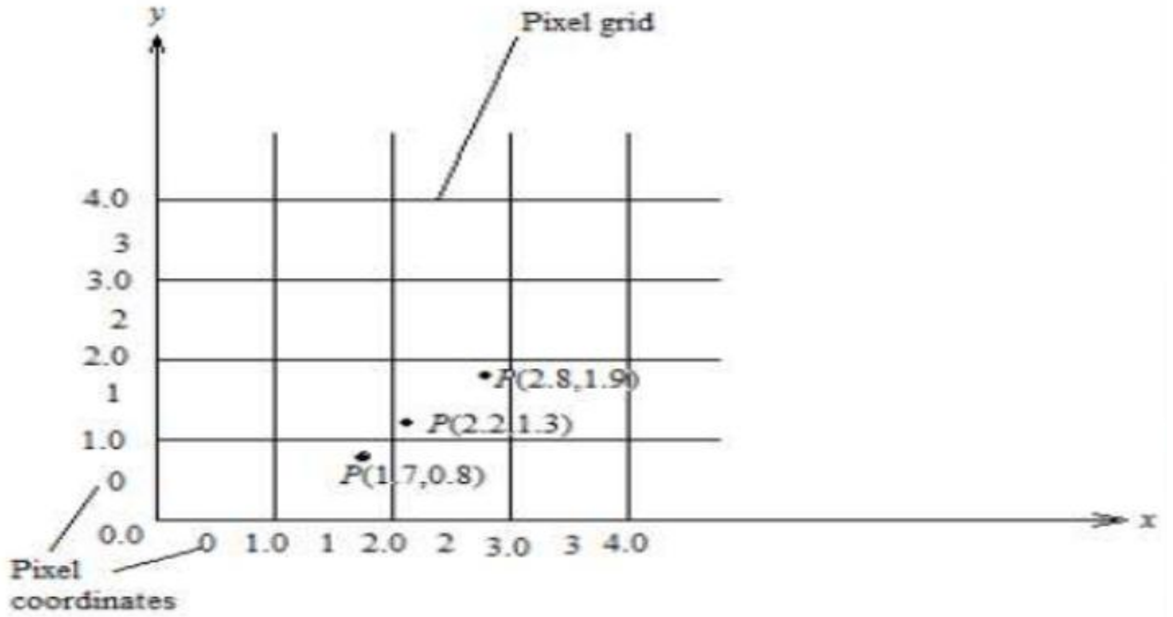


Fig. Pixel positions referenced by scan-line number and column number.

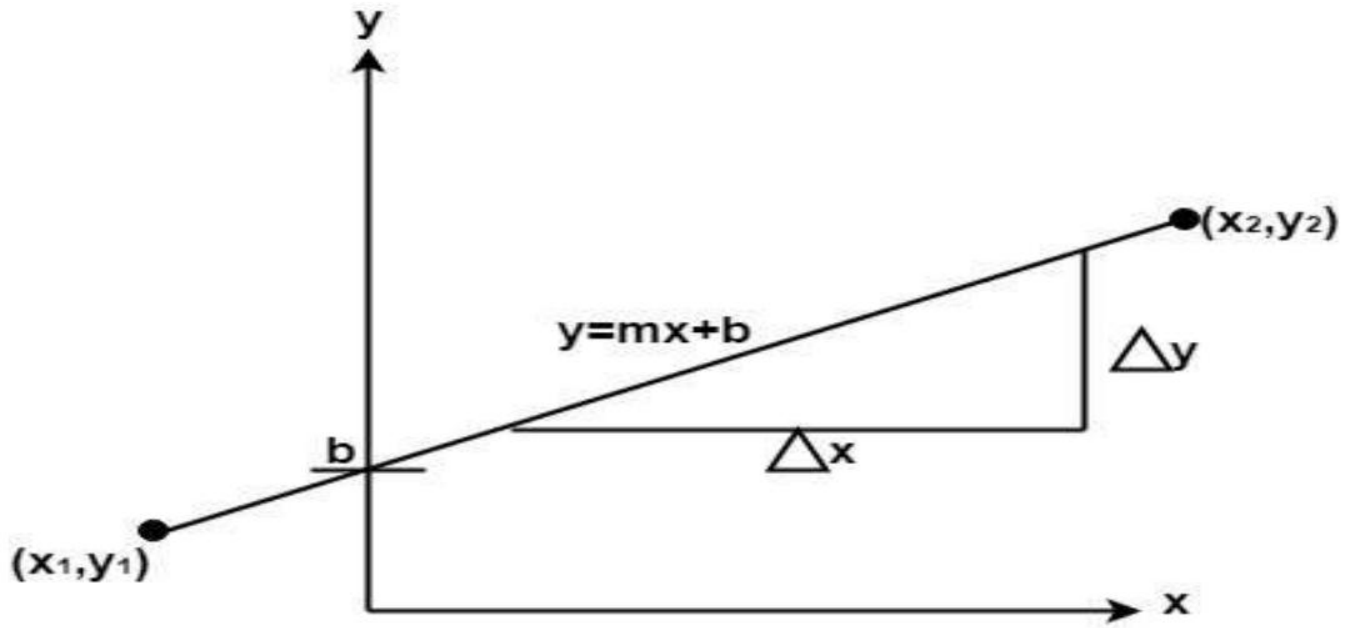
Approach

1. A mathematical point (x, y) where x and y are real numbers within an image area, needs to be scan converted to a pixel at location (x’, y’).
2. This may be done by making x’ to be the integer part of x, and y’ to be the integer part of y.
3. In other words, x’ = floor(x) and y’ = floor(y), where function floor returns the largest integer that is less than or equal to the arguments.
4. Doing so in essence places the origin of a continuous coordinate system for (x, y) at the lower left corner of the pixel grid in the image space.
5. All the points that satisfy x’<= x < x’ + 1 and y’<= y < y’ + 1 are mapped to pixel (x’, y’).
6. For example, a point P1(1.7, 0.8) is represented by pixel (1, 0). Points P2 (2.2, 1.3) and P3(2.8, 1.9) are both represented by pixel (2, 1).



Scan Converting a Straight Line

For the scan conversion of a straight line, we need the two endpoints. In normal life, if we want to draw a line, we simply draw it by using a scale or ruler. But we can't draw a line on the computer by using a ruler. We have to do some programming for it. The computer draws a line by finding the intermediate points between the two endpoints of a line. The line is drawn on the screen when the computer has endpoints and then the computer fills the pixel of the intermediate point's value. The computer can't take the intermediate points in fraction value. For example, after using any of the algorithms if the intermediate point was found (5.1, 7.8) then the computer will round off the point values which are (5,8). The computer takes the nearest integer from that fraction value. This happens because the computer fills the pixels in the screen and pixels are present at the integer values. Either the pixel will be filled or it will not be filled. A pixel can't be partially filled. That is why the line is drawn in the computer is not always a straight line. In the following figure, the two endpoints are described by (x1,y1) and (x2,y2). The equation of the line is used to determine the x, y coordinates of all the points that lie between these two endpoints.



Using the equation of a straight line, $y = mx + b$ where $m =$ & $b =$ the y interrupt, we can find values of y by incrementing x from $x = x1$, to $x = x2$. By scan-converting these calculated x, y values, we represent the line as a sequence of pixels.

A line may have three forms with respect to slope i.e., it may have slope = 1(slope=45^0) as shown in following figure (a), or may have slope < 1 (slope<45^0) as shown in figure (b) or it may have slope > 1 ((slope>45^0) as shown in figure (c).

Now if a line has slope = 1 it is very easy to draw the line by simply starting form one point and go on incrementing the x and y coordinates till they reach the second point. So that is a simple case but if slope < 1 or is > 1 then there will be some problem.

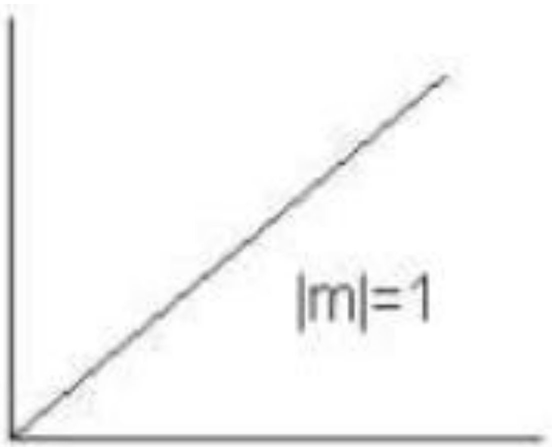


figure (a)

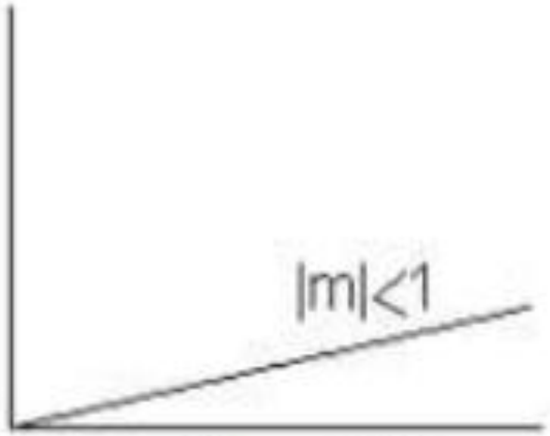


figure (b)

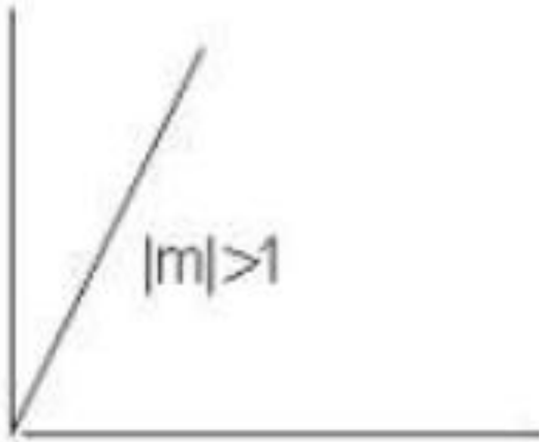


figure (c)

Properties of Good Line Drawing Algorithm:

1. Line should appear Straight: We must appropriate the line by choosing addressable points close to it. If we choose well, the line will appear straight, if not, we shall produce crossed lines.
The lines must be generated parallel or at 45° to the x and y-axes. Other lines cause a problem: a line segment through it starts and finishes at addressable points, may happen to pass through no another addressable points in between.

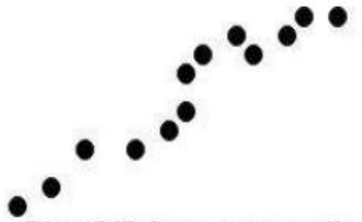


Fig: O/P from a poor line generating algorithm

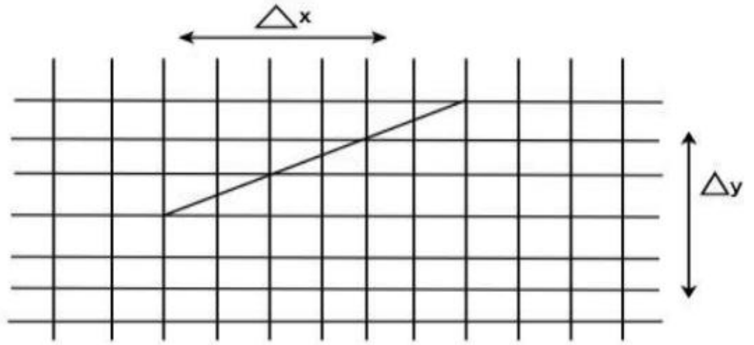


Fig: A straight line segment connecting 2 grid intersection may fail to pass through any other grid intersections.

2. Lines should terminate accurately: Unless lines are plotted accurately, they may terminate at the wrong place.

3. Lines should have constant density: Line density is proportional to the no. of dots displayed divided by the length of the line. To maintain constant density, dots should be equally spaced.
4. Line **density should be independent** of line length and angle: This can be done by computing an approximating line length estimate and to use a line-generation algorithm that keeps line density constant to within the accuracy of this estimate.
5. Line should be **drawn rapidly:** This computation should be performed by special-purpose hardware.

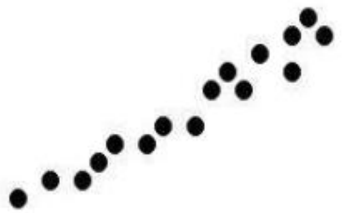


Fig: Uneven line density caused by bunching of dots.

❑ Direct use of Line Equation

It is the simplest form of conversion. First of all, scan P1 and P2 points. P1 has co-ordinates (x1', y1') and (x2', y2') respectively. Then m = (y2'- y1')/ (x2'- x1') and b = y1' -m x1'

If value of |m|≤1 for each integer value of x but excluding x1' and x2'

If value of |m|>1 for each integer value of y but excluding y1' and y2'

```
Algorithm for direct line drawing method:
Step-1:-  accept start point(x1, y1) and end point(x2, y2) coordinate.
Step-2:- calculate dx = x2- x1
                   dy = y2- y1
Step-3:- calculate m = dy/dx
                   b = y1 - m*x1
Step-4:-  Set (x, y) equal to starting point (i.e., lowest point) and xend equal to largest value.
Step-5:-
    if dx<0 then          if dx>0 then
    x = x2                x = x1
    y = y2                y = y1
    $x_{end}$ = x1           $x_{end}$ = x2

Step-6:- plot(x, y)
Step-7:- increment, x =  x+1
             y = m*x + b
             until x<= $x_{end}$
```

Example 01: A line with starting point as (0, 0) and ending point (6, 18) is given. Calculate value of intermediate points and slope of line.

Solution: P1 (0,0) and P2 (6,18)

x1=0 x2=6
y1=0 y2=18

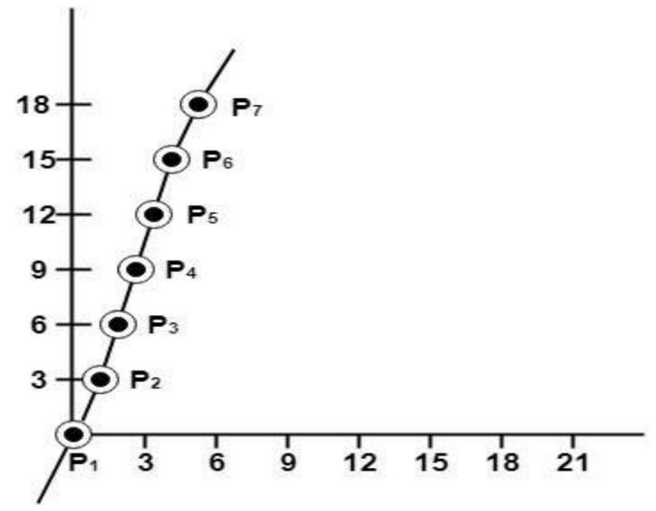
dx = x2 – x1 = 6 – 0 =6
dy = y2 – y1 = 18 – 0 = 18

m = dy / dx = 18/6 = 3
b = y1 – m * x1 = 0 - 3*0 = 0

set starting point (x, y)
here **dx > 0**
x = x1 = 0
y = y1 = 0
x_{end} =x2 = 6
plot (x,y) = (0,0)

x = x +1 --------(i)
y = mx + b = 3x + 0 = 3x ---------(ii)

x_i	y_i	Plot (x_i, y_i)	x_{i+1}=x+1	Y_{i+1}=3x (Here x is x_{i+1})
0	0	(0, 0)	0+1=1	3.1=3
1	3	(1,3)	1+1=2	3.2=6
2	6	(2, 6)	2+1=3	3.3=9
3	9	(3, 9)	3+1=4	3.4=12
4	12	(4, 12)	4+1=5	3.5=15
5	15	(5, 15)	5+1=6	3.6=18
6	18	(6, 18)	Since x_{end}= x_2 =6; so, it stops here.	



Example 02: A line with starting point as (5, 5) and ending point (1, 1) is given. Calculate value of intermediate points and slope of line.

Solution: P1 (5,5) and P2 (1,1)

x1=5	x2=1
y1=5	y2=1

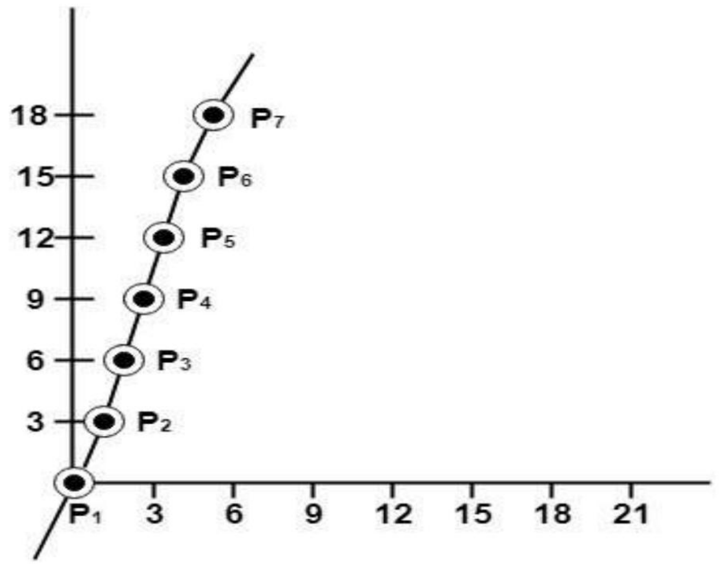
dx = x2 – x1 = 1 – 5 = - 4
dy = y2 – y1 = 1 – 5 = - 4

m = dy / dx = -4 / -4 = 1
b = y1 – m * x1 = 5 - 1*5 = 0

set starting point (x, y)
here **dx < 0**
x = x2 = 1
y = y2 = 1
x_{end} =x1 = 5
plot (x,y) = (1,1)

x = x +1 --------(i)
y = mx + b = 1*x + 0 = x ---------(ii)

X	Y	Plot(x,y)	X=x+1	Y=mx+b = x
1	1	(1,1)	1+1=2	2
2	2	(2,2)	2+1=3	3
3	3	(3,3)	3+1=4	4
4	4	(4,4)	4+1=5	5
5	5	(5,5) [x ==x_{end}]		
Here, x <=x_{end}, so it stops further executions.				



DDA Algorithm

A DDA (Digital Differential Analyzer) algorithms is a scan-conversion method for drawing a line which follows an incremental approach. In this algorithm to draw a line the difference in the pixel points is analyzed then according to that the line is drawn. The method is said to be incremental because it performs computations at each step and uses the outcome of the previous step.

Before understanding DDA algorithm, we must understand what is a line and how it is defined? When two points in a plane connected by a line segment and falls under the line equation is known as a line.

The line equation mentioned above is

$y=mx+b$ where m is the slope (i.e., $m = \Delta y/\Delta x$) and

y is the intercept of the line (the value of y at the points of the line).

Now for DDA,

let us assume at i we have computed (x_i, y_i) to be a point on the line as the next point (x_i+1, y_i+1) should $\Delta y/\Delta x = m$ where $\Delta y = (y_i+1) - y_i$ and

$\Delta x = (x_i+1) - x_i$

The line of equation for step i

$y_i=mx_i+b$……………………(1)

Next value will be

$y_i+1=mx_i+1+b$………………(2)

Now we get,

$y_i+1=y_i+\Delta y$

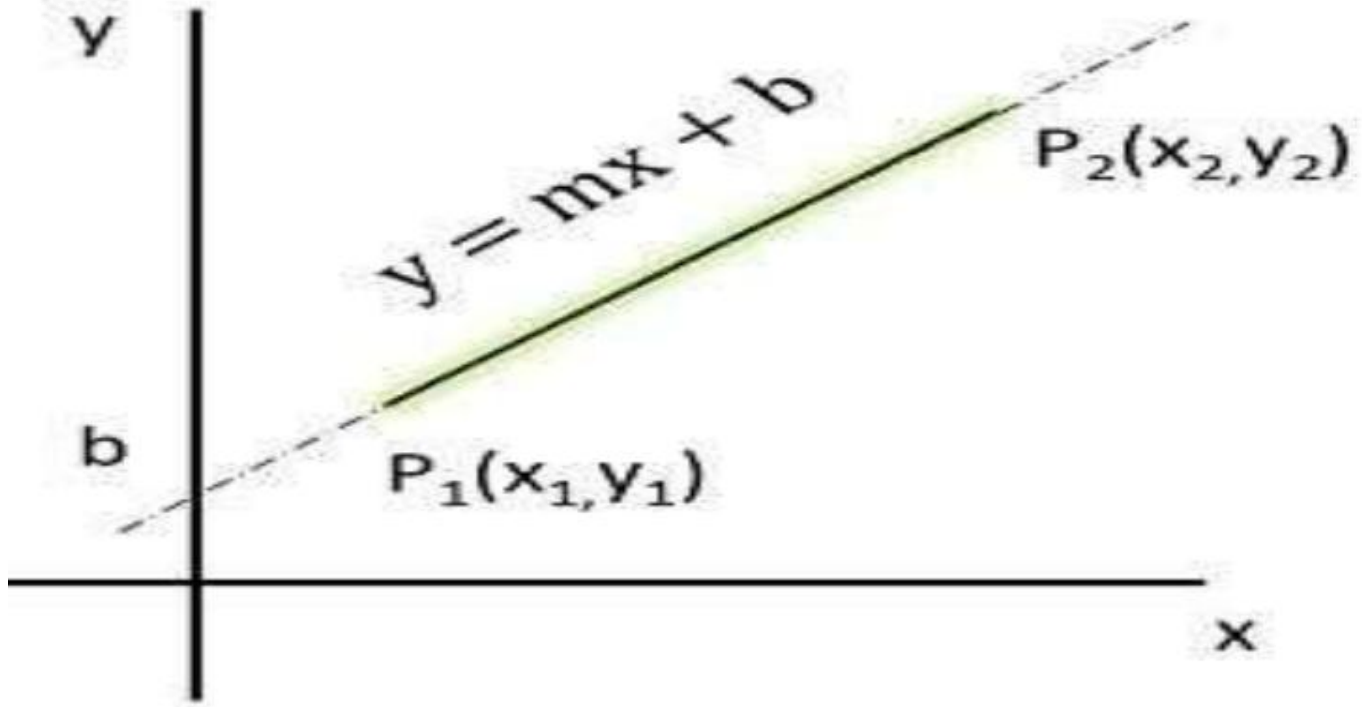
and $\Delta y=m\Delta x$

So, we can write $y_i+1=y_i+m\Delta x$…………(3)

Again, we get $\Delta x=\Delta y/m$

and $x_i+1=x_i+\Delta x$

So, we can write $x_i+1=x_i+\Delta y/m$……………(4)



Scan converting a line

Case1: When |M|<1 then (assume that x12)

x= x1, y=y1 set Δx=1

yi+1=y1+m, x=x+1

Until x = x2

Case2: When |M|<1 then (assume that y12)

x= x1, y=y1 set Δy=1

xi+1= 1/m, y=y+1

Until y → y2

❑ Criticism on DDA Line Drawing Algorithm:

There is serious criticism on the algorithm that is use of floating-point calculation. They say that when we have to draw points that should have integers as coordinates then why to use floating point calculation, which requires more space as well as they have more computational cost. Therefore, there is need to develop an algorithm which would be based on integer type calculations. Therefore, work is done and finally we will come up with an algorithm “Bresenham Line Drawing algorithm” which will be discussed next.

❑ Advantage:

1. It is a faster method than method of using direct use of line equation.
2. This method does not use multiplication theorem.
3. It allows us to detect the change in the value of x and y, so plotting of same point twice is not possible.
4. This method gives overflow indication when a point is repositioned.
5. It is an easy method because each step involves just two additions.

❑ Disadvantage:

1. It involves floating point additions rounding off is done. Accumulations of round off error cause accumulation of error.
2. Rounding off operations and floating-point operations consumes a lot of time.
3. It is more suitable for generating line using the software. But it is less suited for hardware implementation.

❑ Uses of DDA Algorithm:

DDA (Digital Differential Analyzer) algorithm is commonly used in computer graphics for line drawing. It has a wide range of applications, including:

1.Creating basic graphics primitives: DDA algorithm can be used to draw simple shapes such as lines, polygons, and rectangles. By using a series of line segments generated using DDA, more complex shapes can also be created.

2.Computer-aided design (CAD): In CAD software, DDA algorithm is used to draw lines between two points, which are used to create 2D and 3D models.

3.Image processing: DDA algorithm can be used in image processing for tasks such as edge detection and image segmentation.

4.Video game development: DDA algorithm is used for rendering lines and polygons in real-time graphics rendering for video games.

5.Simulation and modeling: DDA algorithm is used to simulate physical phenomena such as ray tracing, which is used in computer graphics to create realistic images of 3D objects.

❑ Issues in DDA Algorithm:

some limitations and issues, which are:

1.Floating point arithmetic: The DDA algorithm requires floating-point arithmetic, which can be slow on some systems. This can be a problem when dealing with large datasets.

2.Limited precision: The use of floating-point arithmetic can lead to limited precision in some cases, especially when the slope of the line is very steep or shallow.

3.Round-off errors: Round-off errors can occur during calculations, which can lead to inaccuracies in the generated line. This is particularly true when the slope of the line is close to 1.

4.Inability to handle vertical lines

5.Slow for complex curves

6.Aliasing: Aliasing occurs when the line segments generated using the DDA algorithm do not accurately represent the line being drawn, resulting in a jagged appearance.

7.Not suitable for thick lines

DDA(Digital Differential Analyzer) algorithm

```
Step-1: Accept start point(x1, y1) and end point(x2, y2) coordinate
Step-2: calculate dx = x2 - x1 and
                  dy = y2 - y1
Step-3: calculate m = dy/dx
Step-4: if m <= 1
            $x_{i+1} = x_i + 1$
            $y_{i+1} = y_i + m$
        until $x_{i+1}$ <= x2
Step-5: else if m > 1
            $x_{i+1} = x_i + 1/m$
            $y_{i+1} = y_i + 1$
        until $y_{i+1}$ <= y2

Step-6: plot pixel ($x_{i+1}$, $y_{i+1}$) as the rounded.
```

Example 01: If a line is drawn from (2, 3) to (6, 15) with use of DDA. How many points will be needed to generate such line?

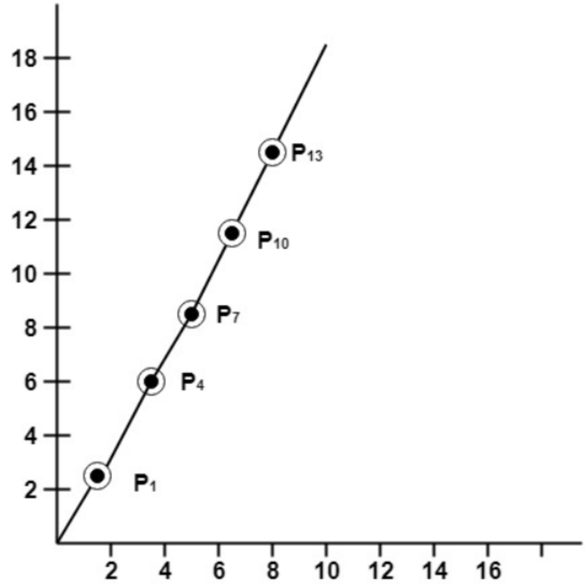
Solution:

Given (2,3) and (6,15)
x1=2 x2= 6
y1=3 y2=15

dx = x2 - x1
= 6 - 2 = 4
dy = y2 – y1
= 15 - 3 = 12
m = dy / dx = 12 / 4 = 3

Here slope (m) is greater than 1 (m>1)
Then new points will be:-

$x_{i+1} = x_i + 1/3$
$y_{i+1} = y_i + 1$
until y_{i+1} <= y2 is reached
and plot pixels rounded



Steps	x_i	y_i	$x_{i+1} = x_i + 1/3$	$y_{i+1} = y_i + 1$	(x_{i+1}, y_{i+1})	Pixel to be Plotted
01	2	3	2+1/3= 2.33	3+1=4	(2.33, 4)	(2, 4)
02	2.33	4	2.33+1/3=2.66	4+1=5	(2.66, 5)	(3, 5)
03	2.66	5	2.66+1/3= 3	5+1=6	(3, 6)	(3, 6)
04	3	6	3+1/3=3.33	6+1=7	(3.33, 7)	(3, 7)
05	3.33	7	3.33+1/3=3.66	7+1=8	(3.66, 8)	(4, 8)
06	3.66	8	3.66+1/3=4	8+1=9	(4, 9)	(4, 9)
07	4	9	4+1/3=4.33	9+1=10	(4.33, 10)	(4, 10)
08	4.33	10	4.33+1/3=4.66	10+1=11	(4.66, 11)	(5, 11)
09	4.66	11	4.66+1/3=5	11+1=12	(5, 12)	(5, 12)
10	5	12	5+1/3=5.33	12+1=13	(5.33, 13)	(5, 13)
11	5.33	13	5.33+1/3=5.66	13+1=14	(5.66, 14)	(6, 14)
12	5.66	14	5.66+1/3=6	14+1=15	(6, 15)	(6, 15)
At step 12, y reaches y=y2 and we get the coordinate (6, 15); Hence it stops here.						

Example- 02: Draw a line from (1,1) to (8,7) using DDA algorithm.

Solution: Given (1,1) and (8,7)

x1=1 x2= 8

y1=1 y2=7

dx = x2 – x1 = 8 - 1 = 7

dy = y2 – y1 = 7 – 1 = 6

m = dy/dx=6/7 = 0.86

Here slope (m) is lesser than 1 (m<1)

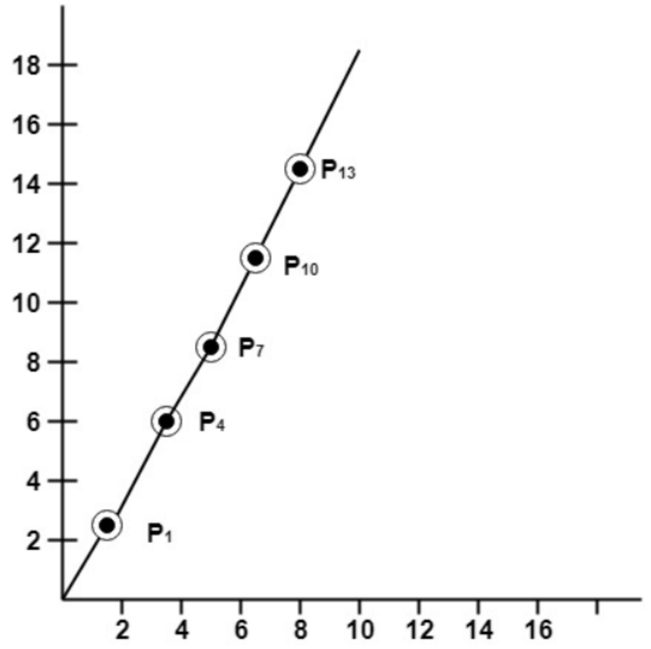
$x_{i+1} = x_i + 1$

$y_{i+1} = y_i + 0.86$

until $x_{i+1} <= x2$ is reached

And Plot the pixels rounded

Steps	x_i	y_i	$x_{i+1} = x_i + 1$	$y_{i+1} = y_i + 0.86$	(x_{i+1}, y_{i+1})	Pixel to be Plotted
01	1	1	1+1=2	1+0.86=1.86	(2, 1.86)	(2, 2)
02	2	1.86	2+1=3	1.86+0.86=2.72	(3, 2.72)	(3, 3)
03	3	2.72	3.+1=4	2.72+0.86=3.58	(4, 3.58)	(4, 4)
04	4	3.58	4+1=5	3.58+0.86=4.44	(5, 4.44)	(5, 4)
05	5	4.44	5+1=6	4.44+0.86=5.3	(6, 5.3)	(6, 5)
06	6	5.3	6+1=7	5.3+0.86=6.16	(7, 6.16)	(7, 6)
07	7	6.16	7+1=8	6.16+0.86=7.02	(8, 7.02)	(8, 7)



The algorithm will stop here as the x value has reached x=x2=8.

Assignment

01. Draw a line using DDA Algorithm from (0,0) to (4,6).

02.Draw a line from (0,0) to (7,7) using DDA algorithm.

03.Use DDA Algorithm to draw a line from (2,3) to (9,8).

04.Calculate the points between the starting point (5, 6) and ending point (8, 12) by using DDA algorithm.

05.Calculate the points between the starting point (5, 6) and ending point (13, 10) by using DDA algorithm.

06.Calculate the points between the starting point (1, 7) and ending point (11, 17) by using DDA algorithm.

07.Indicate which raster locations would be chosen by DDA algorithm when scan converting a line from pixel coordinate (1,1) to pixel coordinate (8,5).

08.A line has a starting point (9,18) and ending point (14,22). Apply the DDA Line Drawing algorithm to plot a line.

09.A line has a starting point (20,10) and ending point (30,18). Apply the DDA Line Drawing algorithm to plot a line.

10.Draw a line from A (2, 2) to B (5, 5) using the DDA algorithm.

1. Draw a line using DDA Algorithm from (0,0) to (4,6).

Solution:

Given coordinates: (0,0) and (4,6)

x1 = 0   x2 = 4

y1 = 0   y2 = 6

Slope(m) = (y2-y1) / (x2-x1) = (6-0) / (4-0) = 1.5 (where m>1)

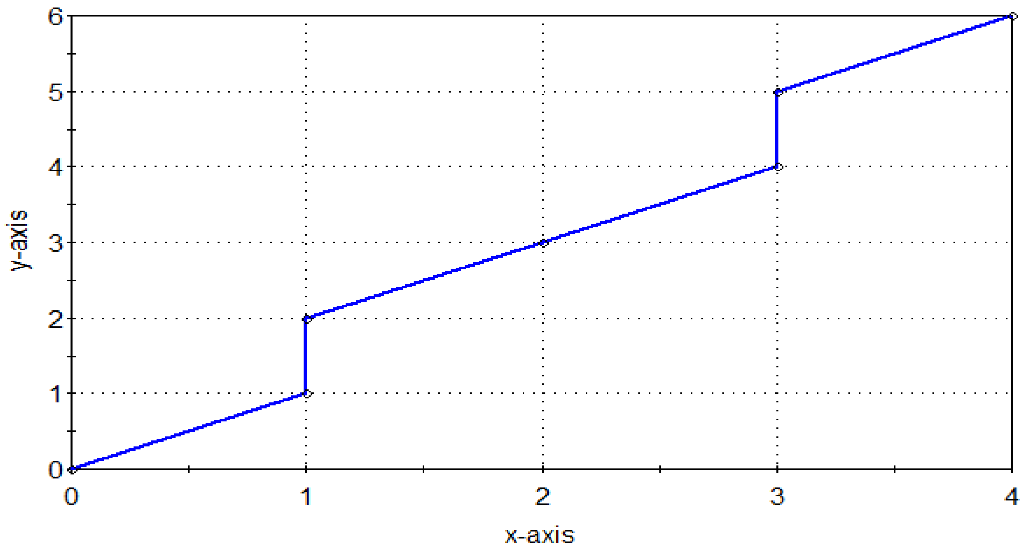
Set initial position (x, y) = (0,0)

So, the formula to calculate next coordinates: xi+1 = xi + (1/m) = xi + (1/1.5) = xi + 0.67 yi+1 = yi + 1 (Until y1 <= y2)

Plot positions(round(xi+1), round(yi+1))

Steps	x_i	y_i	$x_{i+1} = x_i + 0.67$	$y_{i+1} = y_i + 1$	(x_{i+1}, y_{i+1})	Pixel to be Plotted
01	0	0	0+0.67 = 0.67	0+1=1	(0.67, 1)	(1,1)
02	0.67	1	0.67+0.67 = 1.34	1+1=2	(1.34, 2)	(1, 2)
03	1.34	2	1.34+0.67 = 2	2+1=3	(2, 3)	(2, 3)
04	2	3	2+0.67 = 2.67	3+1=4	(2.67, 4)	(3, 4)
05	2.67	4	2.67+0.67 = 3.34	4+1=5	(3.34, 5)	(3, 5)
06	3.34	5	3.34+0.67 = 4	5+1=6	(4, 6)	(4, 6)
At step 6, x reaches x=x2 and we get the coordinate (4, 6); Hence it stops here.						

pixel plotted in the graph



02. Draw a line using DDA Algorithm from (0,0) to (7,7).

Solution:

Given coordinates: (0,0) and (7,7)

x1 = 0 x2 = 7

y1 = 0 y2 = 7

Slope(m) = (y2-y1) / (x2-x1) = (7-0) / (7-0) = 1 (where m=1)

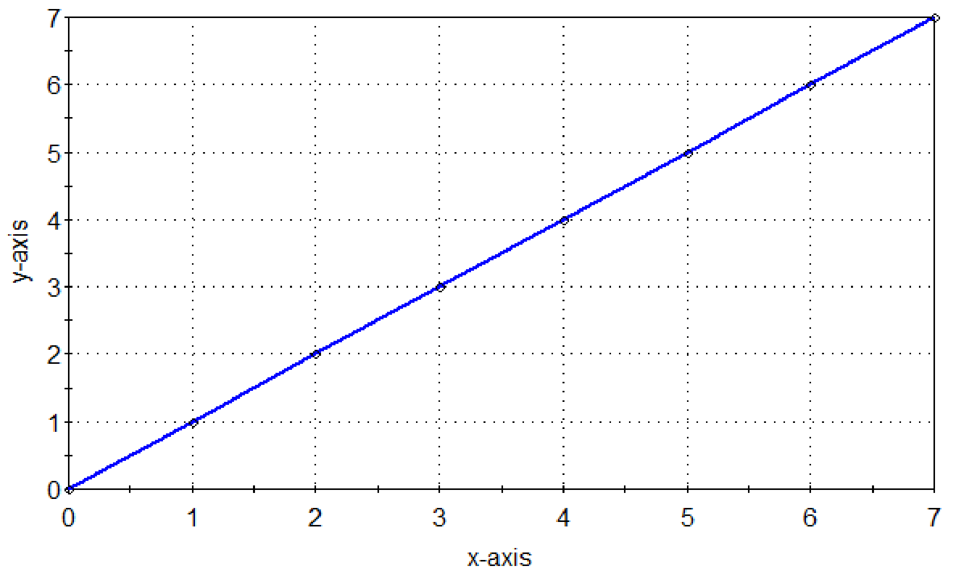
Set initial position (x, y) = (0,0)

So, the formula to calculate next coordinates: xi+1 = xi + 1 , yi+1 = yi + 1 (Until x1 <= x2)

Plot positions(round(xi+1), round(yi+1))

Steps	xi	yi	xi+1 = xi +1	yi+1 = yi +1	(xi+1, yi+1)	Pixel to be Plotted
01	0	0	0+1= 1	0+1=1	(1, 1)	(1,1)
02	1	1	1+1=2	1+1=2	(2, 2)	(2, 2)
03	2	2	2+1= 3	2+1=3	(3, 3)	(3, 3)
04	3	3	3+1=4	3+1=4	(4, 4)	(4, 4)
05	4	4	4+1=5	4+1=5	(5, 5)	(5, 5)
06	5	5	5+1=6	5+1=6	(6, 6)	(6, 6)
07	6	6	5+1=7	6+1=7	(7, 7)	(7, 7)
At step 7, x reaches x=x2 and we get the coordinate (7, 7); Hence it stops here.						

pixel plotted in the graph



03. Draw a line using DDA Algorithm from (2,3) to (9,8).

Solution:

Given coordinates: (2,3) and (9,8)

x1 = 2   x2 = 9

y1 = 3   y2 = 8

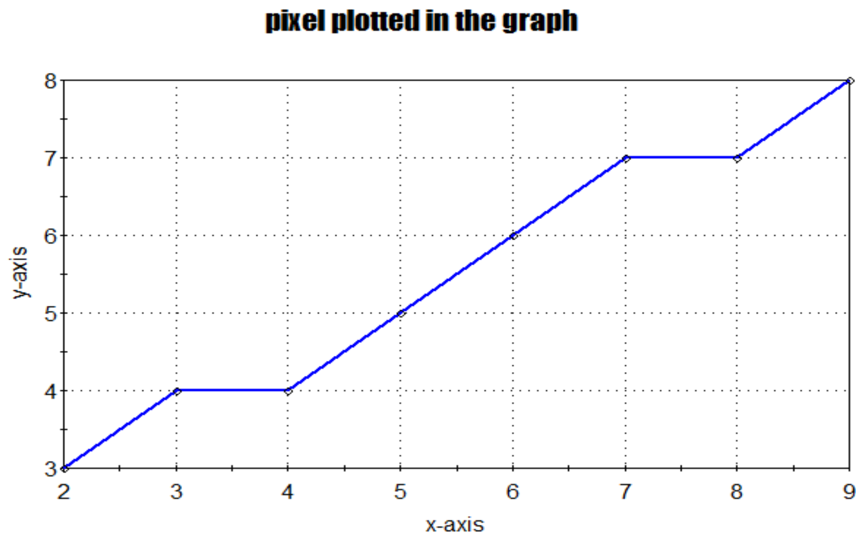
Slope(m) = (y2-y1) / (x2-x1) = (8-3) / (9-2) = 0.71 (where m<1)

Set initial position (x, y) = (2,3)

So, the formula to calculate next coordinates: xi+1 = xi + 1,   yi+1 = yi + 0.71 (Until x1 <= x2)

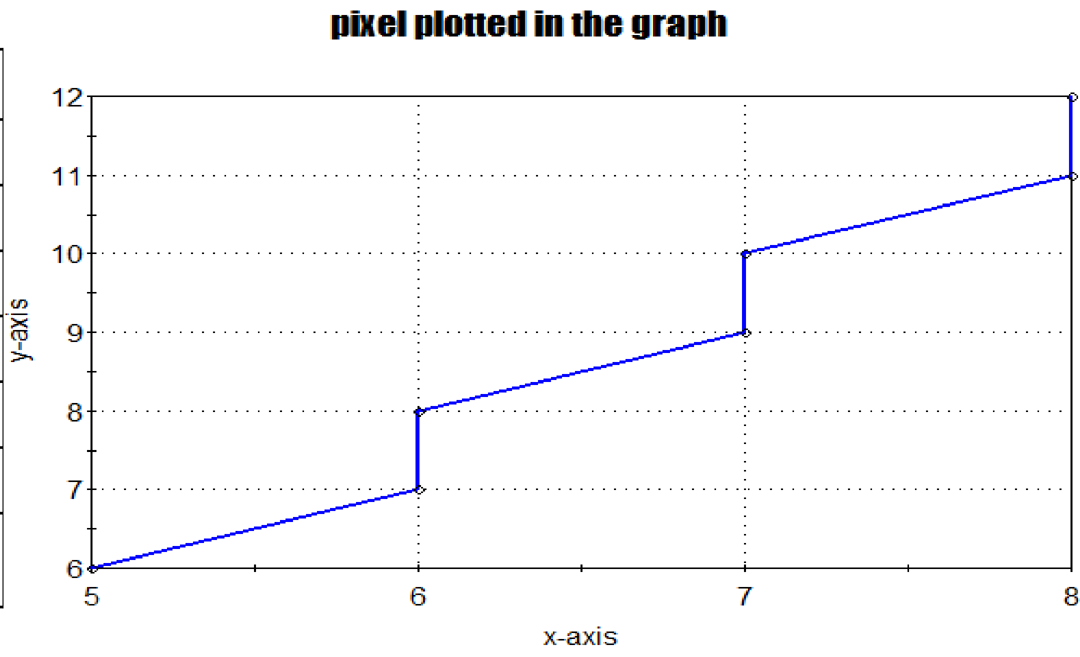
Plot positions(round(xi+1), round(yi+1))

Steps	xi	yi	xi+1 = xi +1	yi+1 = yi +0.71	(xi+1, yi+1)	Pixel to be Plotted
01	2	3	2+1= 3	3+0.71=3.71	(3, 3.71)	(3,4)
02	3	3.71	3+1=4	3.71+0.71=4.42	(4, 4.42)	(4, 4)
03	4	4.42	4+1=5	4.42+0.71=5.13	(5, 5.13)	(5, 5)
04	5	5.13	5+1=6	5.13+0.71=5.84	(6, 5.84)	(6, 6)
05	6	5.84	6+1=7	5.84+0.71=6.55	(7, 6.55)	(7, 7)
06	7	6.55	7+1=8	6.55+0.71=7.26	(8, 7.26)	(8, 7)
07	8	7.26	8+1-9	7.26+0.71=7.97	(9, 7.97)	(9, 8)
At step 7, x reaches x=x2 and we get the coordinate (9,10); Hence it stops here.						



4.Draw a line using DDA Algorithm from (5,6) to (8,12).
Solution:
Given coordinates: (5,6) and (8,12)
x1 = 5 x2 = 8
y1 = 6 y2 = 12
Slope(m) = (y2-y1) / (x2-x1) = (12-6) / (8-5) = 2 (where m>1)
Set initial position (x, y) = (5,6)
So, the formula to calculate next coordinates: xi+1 = xi + (1/m) = xi + (1/2) = xi + 0.5 , yi+1 = yi + 1 (Until y1 <= y2)
Plot positions(round(xi+1), round(yi+1))

Steps	xi	yi	xi+1 = xi +0.5	yi+1 = yi +1	(xi+1, yi+1)	Pixel to be Plotted
01	5	6	5+0.5= 5.5	6+1=7	(5.5, 7)	(6,7)
02	5.5	7	5.5+0.5=6	7+1=8	(6, 8)	(6, 8)
03	6	8	6+0.5= 6.5	8+1=9	(6.5, 9)	(7, 9)
04	6.5	9	6.5+0.5=7	9+1=10	(7, 10)	(7, 10)
05	7	10	7+0.5=7.5	10+1=11	(7.5, 11)	(8, 11)
06	7.5	11	7.5+0.5=8	11+1=12	(8, 12)	(8, 12)
At step 6, x reaches x=x2 and we get the coordinate (11, 12); Hence it stops here.						



05. Draw a line using DDA Algorithm from (5,6) to (13,10).

Solution:

Given coordinates: (5,6) and (13,10)

x1 = 5 x2 = 13

y1 = 6 y2 = 10

Slope(m) = (y2-y1) / (x2-x1) = (10-6) / (13-5) = 0.5 (where m<1)

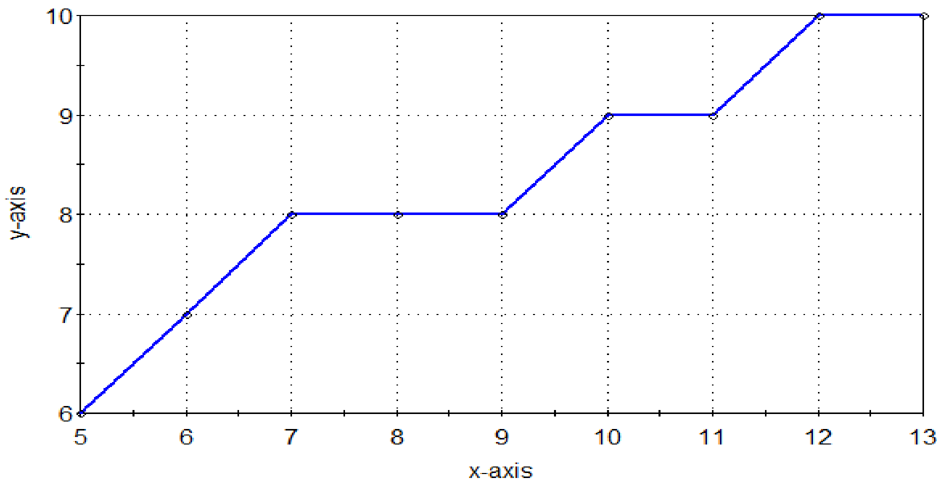
Set initial position (x, y) = (5,6)

So, the formula to calculate next coordinates: xi+1 = xi + 1, yi+1 = yi + 0.5 (Until x1 <= x2)

Plot positions(round(xi+1), round(yi+1))

Steps	x_i	y_i	$x_{i+1} = x_i + 1$	$y_{i+1} = y_i + 0.5$	(x_{i+1}, y_{i+1})	Pixel to be Plotted
01	5	6	5+1=6	6+0.5 =6.5	(6, 6.5)	(6,7)
02	6	6.5	6+1=7	6.5+0.5=7	(7, 7)	(7, 8)
03	7	7	7+1=8	7+0.5= 7.5	(8, 7.5)	(8, 8)
04	8	7.5	8+1=9	7.5+0.5=8	(9, 8)	(9, 8)
05	9	8	9+1=10	8+0.5=8.5	(10, 8.5)	(10, 9)
06	10	8.5	10+1=11	8.5+0.5=9	(11, 9)	(11, 9)
07	11	9	11+1=12	9+0.5=9.5	(12,9.5)	(12,10)
08	12	9.5	12+1=13	9.5+0.5=10	(13,10)	(13,10)
At step 8, x reaches x=x2 and we get the coordinate (13, 14); Hence it stops here.						

pixel plotted in the graph



6. Draw a line using DDA Algorithm from (1,7) to (11,17).

Solution:

Given coordinates: (1,7) and (11,17)

x1 = 1 x2 = 11

y1 = 7 y2 = 17

Slope(m) = (y2-y1) / (x2-x1) = (17-7) / (11-1) = 1 (where m=1)

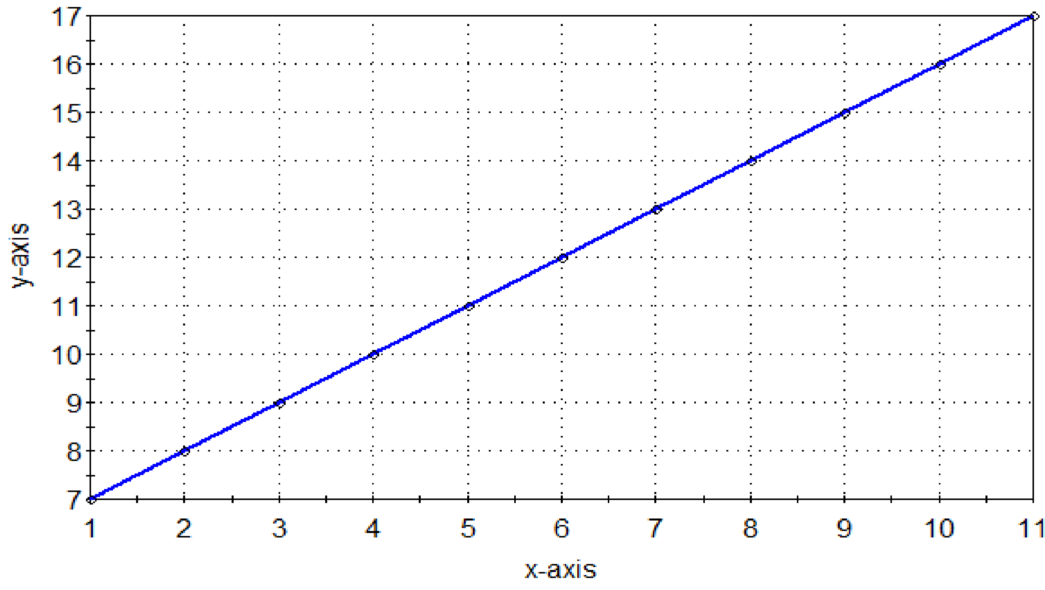
Set initial position (x, y) = (1,7)

So, the formula to calculate next coordinates: xi+1 = xi + 1, yi+1 = yi + 1 (Until x1 <= x2)

Plot positions(round(xi+1), round(yi+1))

Steps	x_i	y_i	$x_{i+1} = x_i + 1$	$y_{i+1} = y_i + 1$	(x_{i+1}, y_{i+1})	Pixel to be Plotted
01	1	7	1+1=2	7+1=8	(2, 8)	(2, 8)
02	2	8	2+1=3	8+1=9	(3, 9)	(3, 9)
03	3	9	3+1=4	9+1=10	(4, 10)	(4, 10)
04	4	10	4+1=5	10+1=11	(5, 11)	(5, 11)
05	5	11	5+1=6	11+1=12	(6, 12)	(6, 12)
06	6	12	6+1=7	12+1=13	(7, 13)	(7, 13)
07	7	13	7+1=8	13+1=14	(8,14)	(8,14)
08	8	14	8+1=9	14+1=15	(9,15)	(9,15)
09	9	15	9+1=10	15+1=16	(10,16)	(10,16)
10	10	16	10+1=11	16+1=17	(11,17)	(11,17)
At step 10, x reaches x=x2 and we get the coordinate (11, 17); Hence it stops here.						

pixel plotted in the graph



07. Indicate which raster locations would be chosen by DDA algorithm when scan converting a line from pixel coordinate(1,1) to pixel coordinate (8,5).

Solution:

Given coordinates: (1,1) and (8,5)

$x1 = 1$ $x2 = 8$

$y1 = 1$ $y2 = 5$

Slope(m) = (y2-y1) / (x2-x1) = (5-1) / (8-1) = 0.57 (where m<1)

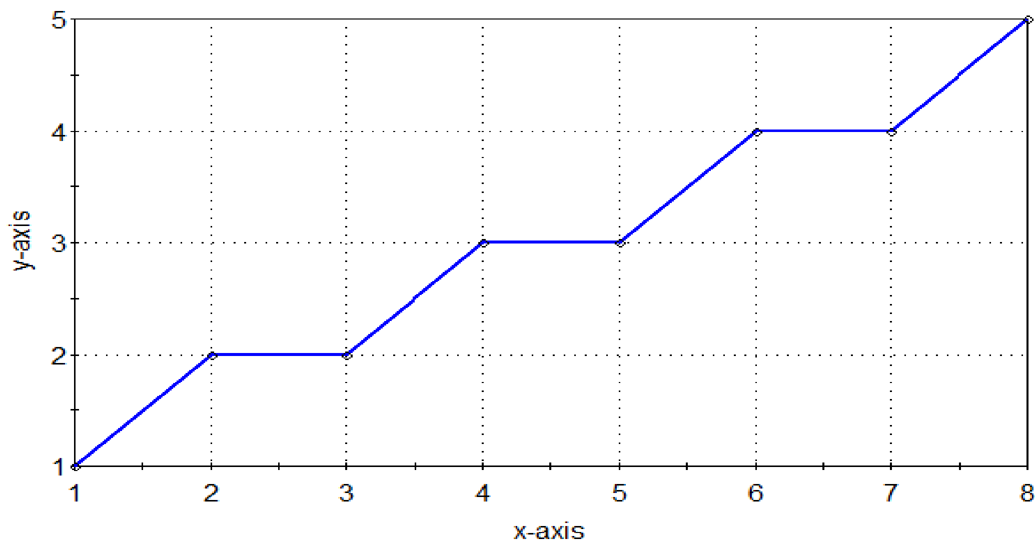
Set initial position (x, y) = (1,1)

So, the formula to calculate next coordinates: xi+1 = xi + 1, yi+1 = yi + 0.57 (Until x1 <= x2)

Plot positions(round(xi+1), round(yi+1))

Steps	x_i	y_i	$x_{i+1} = x_i + 1$	$y_{i+1} = y_i + 0.57$	(x_{i+1}, y_{i+1})	Pixel to be Plotted
01	1	1	1+1=2	1+0.57 =1.57	(2, 1.57)	(2,2)
02	2	1.57	2+1=7	1.57+0.57=2.14	(3, 2.14)	(3, 2)
03	3	2.14	3+1=8	2.14+0.57= 2.71	(4, 2.71)	(4, 3)
04	4	2.71	4+1=9	2.71+0.57=3.28	(5, 3.28)	(5, 3)
05	5	3.28	5+1=10	3.28+0.57=3.85	(6, 3.85)	(6, 4)
06	6	3.85	6+1=11	3.85+0.57=4.42	(7, 4.42)	(7, 4)
07	7	4.42	7+1=12	4.42+0.57=5	(8,5)	(8,5)
At step 7, x reaches x=x2 and we get the coordinate (8, 5); Hence it stops here.						

pixel plotted in the graph



08. A line has a starting point (9,18) and ending point (14,22). Apply DDA line drawing algorithm to plot a line.

Solution:

Given coordinates: (9,18) and (14,22)

x1 = 9 x2 = 14

y1 = 18 y2 = 22

Slope(m) = (y2-y1) / (x2-x1) = (22-18) / (14-9) = 0.8 (where m<1)

Set initial position (x, y) = (9,8)

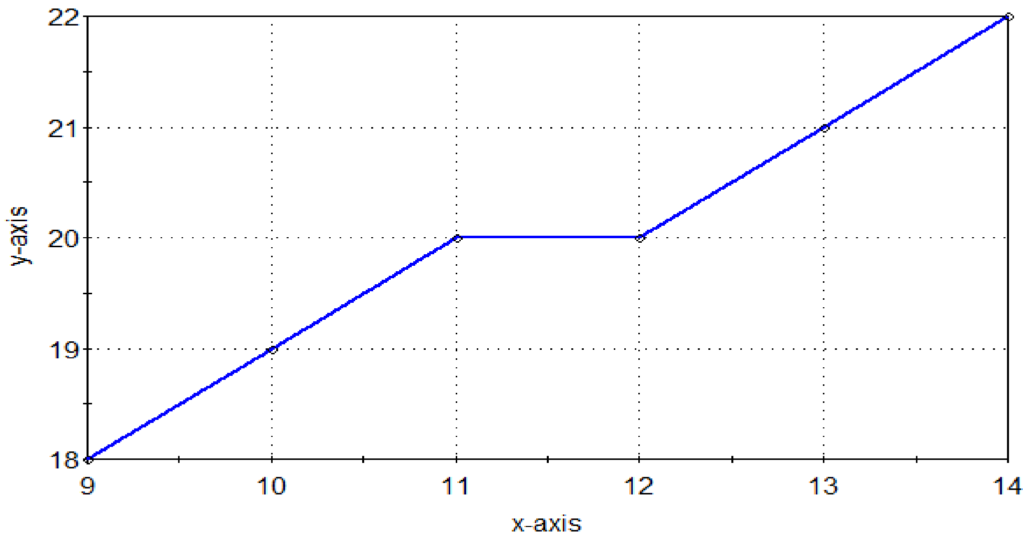
So, the formula to calculate next coordinates: xi+1 = xi + 1, yi+1 = yi + 0.8 (Until x1 <= x2)

Plot positions(round(xi+1), round(yi+1))

Steps	x_i	y_i	$x_{i+1} = x_i + 1$	$y_{i+1} = y_i + 0.8$	(x_{i+1}, y_{i+1})	Pixel to be Plotted
01	9	18	9+1=10	18+0.8 =18.8	(10, 18.8)	(10,19)
02	6	18.8	10+1=11	18.8+0.8=19.6	(11, 19.6)	(11, 20)
03	7	19.6	11+1=12	19.6+0.8= 20.4	(12, 20.4)	(12, 20)
04	8	20.4	12+1=13	20.4+0.8=21.2	(13, 21.4)	(13, 21)
05	9	21.2	13+1=14	21.2+0.8=22	(14,22)	(14, 22)

At step 5, x reaches x=x2 and we get the coordinate (14, 2); Hence it stops here.

pixel plotted in the graph



9. A line has a starting point (20,10) and ending point (30,18). Apply DDA line drawing algorithm to plot a line.

Solution:

Given coordinates: (20,10) and (30,18)

x1 = 20 x2 = 30

y1 = 10 y2 = 18

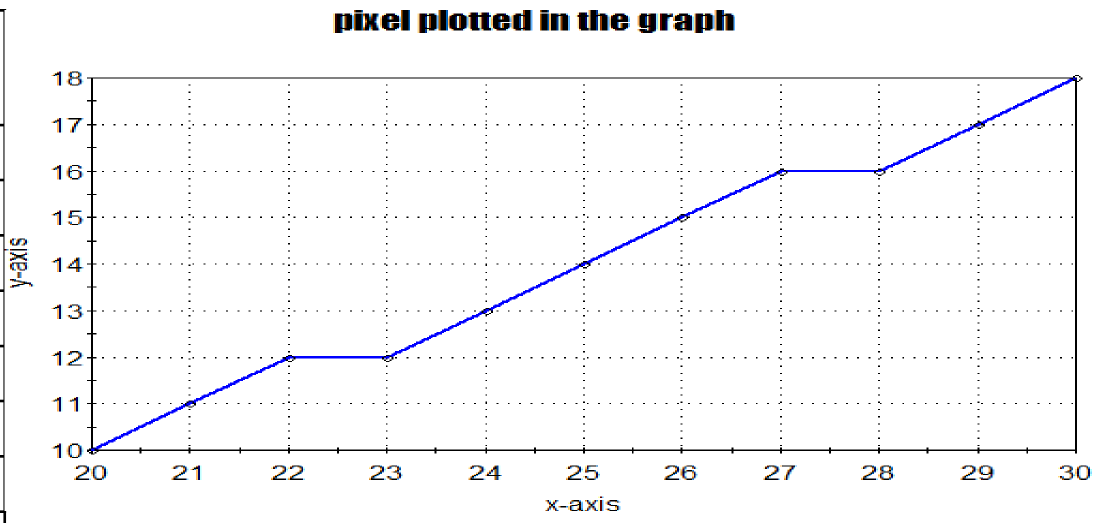
Slope(m) = (y2-y1) / (x2-x1) = (18-10) / (30-20) = 0.8 (where m<1)

Set initial position (x, y) = (20,10)

So, the formula to calculate next coordinates: xi+1 = xi + 1, yi+1 = yi + 0.8 (Until x1 <= x2)

Plot positions(round(xi+1), round(yi+1))

Steps	x_i	y_i	$x_{i+1} = x_i + 1$	$y_{i+1} = y_i + 0.8$	(x_{i+1}, y_{i+1})	Pixel to be Plotted
01	20	10	20+1=21	10+0.8 =10.8	(21, 10.8)	(21,11)
02	21	10.8	21+1=22	10.8+0.8=11.6	(22, 11.6)	(22, 12)
03	22	11.6	22+1=23	11.6+0.8= 12.4	(23, 12.4)	(23, 12)
04	23	12.4	23+1=24	12.4+0.8=13.2	(24, 13.2)	(24, 13)
05	24	13.2	24+1=25	13.2+0.8=14	(25, 14)	(25, 14)
06	25	14	25+1=26	14+0.8=14.8	(26, 14.8)	(26, 15)
07	26	14.8	26+1=27	14.8+0.8=15.6	(27,15.6)	(27,16)
08	27	15.6	27+1=28	15.6+0.8=16.4	(28,16.4)	(28,16)
09	28	16.4	28+1=29	16.4+0.8=17.2	(29,17.2)	(29,17)
10	29	17.2	29+1=30	17.2+0.8=18	(30,18)	(30,18)
At step 10, x reaches x=x2 and we get the coordinate (30, 18); Hence it stops here.						



10. Draw a line from A(2,2) to B(5,5) using the DDA algorithm.

Solution:

Given coordinates: (2,2) and (5,5)

x1 = 2   x2 = 5

y1 = 2   y2 = 5

Slope(m) = (y2-y1) / (x2-x1) = (5-2) / (5-2) = 1 (where m=1)

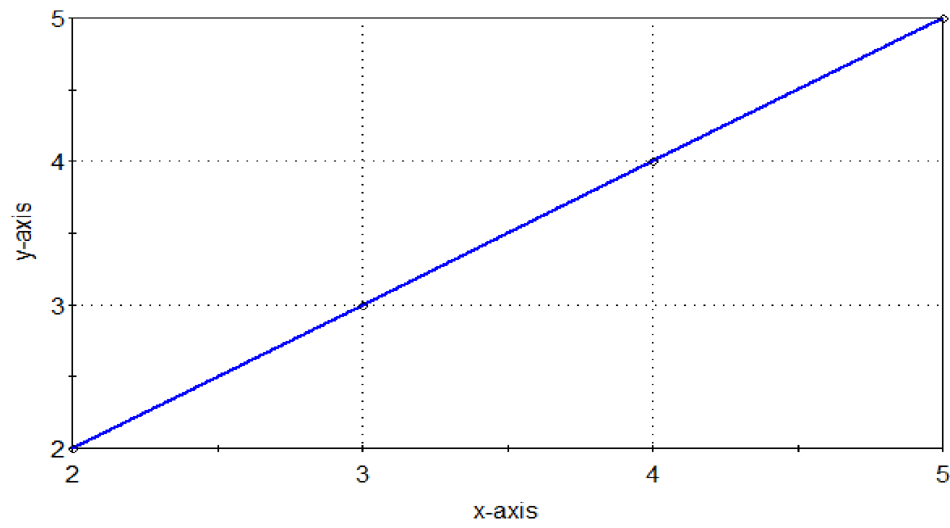
Set initial position (x, y) = (2,2)

So, the formula to calculate next coordinates: xi+1 = xi + 1, yi+1 = yi + 1 (Until x1 <= x2)

Plot positions(round(xi+1), round(yi+1))

Steps	x_i	y_i	$x_{i+1} = x_i + 1$	$y_{i+1} = y_i + 0.5$	(x_{i+1}, y_{i+1})	Pixel to be Plotted
01	2	2	2+1=3	2+1=3	(3 3)	(3,3)
02	3	3	3+1=4	3+1=4	(4, 4)	(4, 4)
03	4	4	4+1=5	4+1=5	(5, 5)	(5, 5)
At step 3, x reaches x=x2 and we get the coordinate (5,5); Hence it stops here.						

pixel plotted in the graph



❑ Scan Conversion (Bresenham’s Line Drawing Algorithm)

This algorithm was introduced by “Jack Elton Bresenham” in 1962. This algorithm helps us to perform scan conversion of a line. It is a powerful, useful, and accurate method. It is also an incremental method for creating a line i.e., we use incremental integer calculations to draw a line. 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.

Working of the Bresenham's Algorithm Bresenham's algorithm finds the closest integer coordinates to the actual line, using only integer math.

Suppose we have to draw a line PQ with coordinates P (x1, y1) and Q (x2, y2).

To draw the line using Bresenham's line drawing algorithm, first of all, calculate the slope of the line from the given coordinates by using,

m = dy/dx Where, dy = x2 - x1 and dx = y2 - y1

There can be three values of slope (m) i.e.,

1. m < 1 (slope is positive but less than 1 or 0^0 to 45^0)
2. m>1 (slope>45^0) &
3. m=1 (slope=45^0)

On the basis of the value of the slope, the decision parameter is calculated which gives the decision about the selection of the next pixel point which has the least distance from the true line.

We can calculate the decision parameter by the following way with two cases:

Case I: slope (m) < 1 (slope is positive but less than 1 or 0^0 to 45^0) or slope (m) <=1

Case II: slope(m) >= 1 or slope(m) > 1

Here we will derive the decision parameter for both Case I and II and at the end of the

derivation we will look Bresenham’s algorithm for

i) slope (m) < 1 or slope (m) <= 1

ii) slope(m) >1 or slope(m) >= 1

iii) slope (m) < 1 & slope(m) >= 1 and

iv) slope (m) <= 1 & slope(m) >1

The next pixel to the line will be either to its top or to its bottom. If the current point is (x_i, y_i),

the next point can be either (x_{i+1}, y_i) or (x_{i+1}, y_{i+1}) .

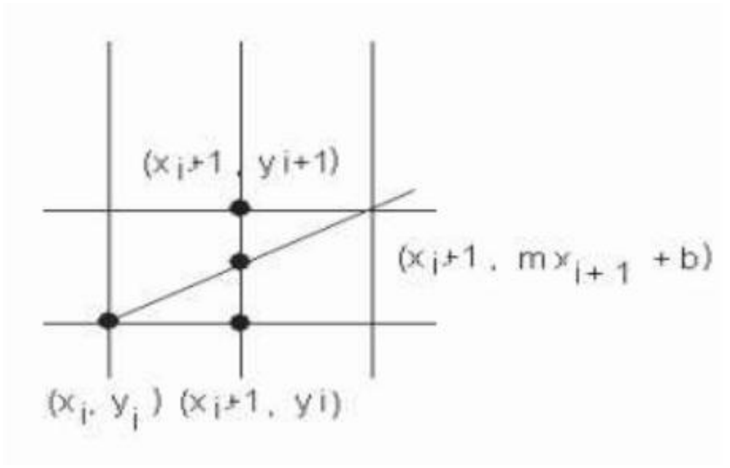
If we choose the top pixel, the points will become,

x_{next} or $x_{i+1} = x_i+1$ and y_{next} or $y_{i+1} = y_i + 1$

If we choose bottom pixel, the points will become,

x_{next} or $x_{i+1} = x_i+1$ and y_{next} or $y_{i+1} = y_i$

Since Bresenham's algorithm depends upon the distance. So, calculating the distance between the true point, and the two alternative pixel positions available gives:
d1 = y – y_i
= m * (x_i +1)+b- y_i
d2 = y_i + 1 - y
= y_i + 1 – m (xi + 1) – b
If d1 - d2<0, then the closest pixel will be the bottom pixel and y_{next} or $y_{i+1} = y_i$
if d1 - d2>0, then the closest pixel will be the top pixel y_{next} or $y_{i+1} = y_i + 1$
Let us magically define a decision function p, to determine which distance is closer to the true point. By taking the difference between the distances, the decision function will be positive if d1 is larger, and negative otherwise. A positive scaling factor is added to ensure that no division is necessary, and only integer math need be used.
Pi = dx (d1 - d2)
= dx [m * (xi +1)+b-yi]- [yi + 1 – m (xi + 1) – b]
= dx (2m * (xi+1) + 2b – 2yi-1)
= dx [2 (dy/dx) * (xi+1) + 2b – 2yi-1]
= 2 dy (xi+1) –2 dx yi + dx (2b-1)......................(i)
= 2 dy xi – 2 dx yi + k ....(ii)
where k=2 dy + dx (2b-1)
Then we can calculate Pi+1 or Pnext in terms of pi without any xi , yi or k .
Pnext or Pi+1 = 2 dy xnext – 2 dx ynext + k
= 2 dy xi+1 – 2 dx yi+1 + k
= 2 dy (xi + 1) - 2 dx yi+1 + k [since xi+1= xi + 1]
= 2 dy xi + 2 dy- 2 dx yi+1 + k ....(iii)

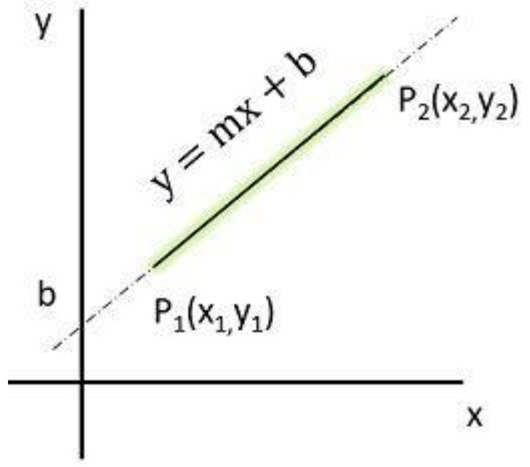


Now subtracting (ii) from (iii), we get
Pi+1 - Pi = 2 dy - 2 dx (yi+1 - yi)
=> Pi+1 = Pi+ 2 dy - 2 dx (yi+1 - yi) ....(iv)
Case I: If the next point is: (xi+1, yi) i.e., if chosen pixel is at the bottom pixel (i.e., d1-d2<0) ⟹
yi+1=yi then we get from (iv)
Pi+1 = Pi+ 2 dy - 2 dx (yi+1 - yi)
= Pi+ 2 dy - 2 dx (yi - yi)
= Pi+ 2 dy
Case II: If the next point is: (xi+1, yi+1) i.e., if chosen pixel is at the top pixel (i.e., d1-d2>0) ⟹
yi+1= yi + 1 then we get from (iv)
Pi+1 = Pi+ 2 dy - 2 dx (yi+1 - yi)
= Pi+ 2 dy - 2 dx (yi + 1 - yi)
= Pi+ 2 dy - 2 dx
= Pi+ 2 (dy - dx)
The Pi is our decision variable, and calculated using integer arithmetic from pre-computed constants and its previous value. Now a question is remaining how to calculate initial value of p1. For that use equation (i) and put values (x1, y1)
Pi = 2 dy (xi+1) –2 dx yi + dx (2b-1)
where b = y – mx implies that b = y1 – mx1 for (x1, y1)

$p1 = 2\,dy\,x1 + 2\,dy - 2\,dx\,y1 + dx\,(2\,(y1 - mx1) - 1)$
$p1 = 2\,dy\,x1 + 2\,dy - 2\,dx\,y1 + 2\,dx\,y1 - 2\,dy\,x1 - dx$
$p1 = 2\,dy - dx$
Derivation of the decision parameter Pi for slope(m) > 1 & slope(m) = 1
Let us consider a straight line passing through a point (x, yi+1)
Where ynext or yi+1= yi+1
and x satisfies the equation of the line i.e.,
yi+1= mx+b (floating value)
=> mx+b= yi+1
=> mx= yi+1-b
⇒ x=1/m(yi+1-b)
The next pixel to the line will be either to its right or to its left. If the current point is (xi,yi),
the next point can be either (xi, yi+1) or (xi+1,yi+1) .
If we choose the right pixel, the points will become,
xnext or xi+1= xi+1 and ynext or yi+1 = yi + 1
If we choose left pixel, the points will become,
xnext or xi+1= xi and ynext or yi+1 = yi + 1
Since Bresenham's algorithm depends upon the distance. So, calculating the distance between
the true point, and the two alternative pixel positions available gives:

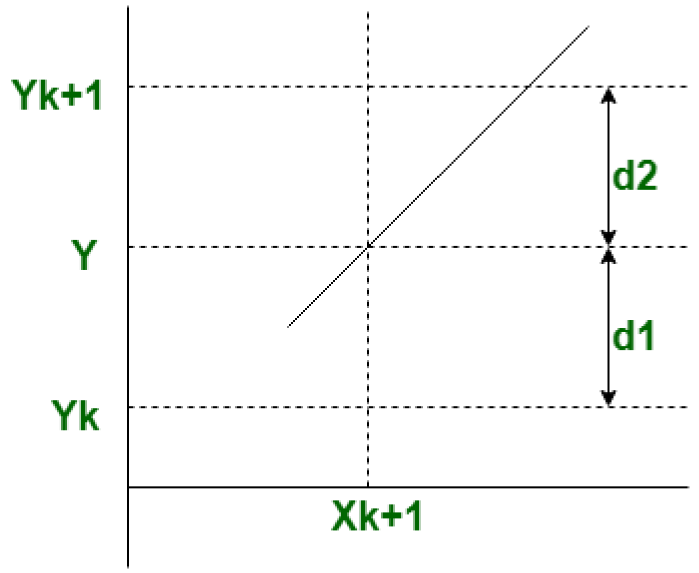
d1 = x – xi
= 1/m(yi+1-b)-xi
=dx/dy(yi+1-b)-xi
d2 = xi + 1 - x
=xi + 1 – 1/m(yi+1-b)
= xi + 1 – dx/dy(yi+1-b)
If d1 - d2<0, then the closest pixel will be the left pixel and xnext or xi+1 = xi
if d1 - d2>0, then the closest pixel will be the right pixel xnext or xi+1 = xi + 1
Let us magically define a decision function p, to determine which distance is closer to the true point. By taking the difference between the distances, the decision function will be positive if d1 is larger, and negative otherwise. A positive scaling factor is added to ensure that no division is necessary, and only integer math need be used.
Pi = dy (d1 - d2)
= dyd1 – dyd2
= dy [dx/dy(yi+1-b)-xi]- dy[xi + 1 – dx/dy(yi+1-b)]
= dx(yi+1-b)-dyxi- dy(xi + 1) + dx(yi+1-b)
= dx(yi+1)-dxb-dyxi- dyxi -dy + dx(yi+1)-dx b
= 2dx(yi+1) –2dyxi - 2dx b – dy
= 2dx yi –2dy xi + (2dx - 2dx b – dy) ....(i)

= 2dx yi –2dy xi + k ....(ii)
where k=2dx - 2dx b – dy
Then we can calculate Pi+1 or Pnext in terms of Pi without any xi , yi or k .
Pnext or Pi+1 = 2dx ynext –2dy xnext + k
 = 2 dx yi+1 – 2 dy xi+1 + k
 = 2 dx (yi + 1) - 2 dy xi+1 + k [since yi+1= yi + 1]
 = 2 dx yi + 2 dx- 2 dy xi+1 + k ....(iii)
Now subtracting (ii) from (iii), we get
Pi+1 - Pi = 2 dx - 2 dy (xi+1 - xi)
 => Pi+1 = Pi+ 2 dx - 2 dy (xi+1 - xi) ....(iv)
Case I: If the next point is: (xi, yi+1) i.e., if chosen pixel is at the left pixel, then xi+1=xi and we get
from (iv)
Pi+1 = Pi+ 2 dx - 2 dy (xi+1 - xi)
 = Pi+ 2 dx - 2 dy (xi - xi)
 = Pi+ 2 dx
Case II: If the next point is: (xi+1, yi+1) i.e., if chosen pixel is at the right pixel, then xi+1= xi + 1 and we get from (v)
 Pi+1 = Pi+ 2 dx - 2 dy (xi+1 - xi)
 = Pi+ 2 dx - 2 dy (xi + 1 - xi)
 = Pi+ 2 dx - 2 dy
 = Pi+ 2 (dx – dy)



Scan converting a line

The Pi is our decision variable, and calculated using integer arithmetic from pre-computed constants and its previous value. Now a question is remaining how to calculate initial value of p1. For that use equation (i) and put values (x1, y1)
Pi = 2dx yi –2dy xi + (2dx - 2dx b – dy)
Pi = 2dx yi –2dy xi + 2dx – 2dx b – dy
where b = y – mx implies that b = y1 – mx1 for (x1, y1)
p1 = 2 dx y1 +2 dx – 2 dy x1 – 2 dx (y1 – mx1) -dy
p1 = 2 dx y1 +2 dx – 2 dy x1 – 2 dx y1 +2dx mx1 -dy
p1 = 2 dx – 2 dy x1 +2dx (dy/dx) x1 -dy
p1 = 2 dx – 2 dy x1 +2dy x1 -dy
p1 = 2 dx – dy



Distance b/w pixels in Bresenham's algo

Bresenham line drawing algorithm

Step-1: accept start point(x1, y1) and end point (x2, y2) coordinate

Step-2: calculate dx = x2 - x1 and
dy = y2 - y1

Step-3: calculate m = dy/dx

Step-4: calculate Initial decision parameter(P)

If $m < 1$
Initial decision parameter(P) = $2dy - dx$

- If $P < 0$

  $x_{i+1} = x_i + 1$
  $y_{i+1} = y_i$

  $P_{i+1} = P_i + 2dy$

- If $P >= 0$

  $x_{i+1} = x_i + 1$
  $y_{i+1} = y_i + 1$

  $P_{i+1} = P_i + 2dy-2dx$

If $m >= 1$
Initial decision parameter(P) = $2dx - dy$

- If $P < 0$

  $x_{i+1} = x_i$
  $y_{i+1} = y_i + 1$

  $P_{i+1} = P_i + 2dx$

- If $P >= 0$

  $x_{i+1} = x_i + 1$
  $y_{i+1} = y_i + 1$

  $P_{i+1} = P_i + 2dx-2dy$

Step-5: update new points (x_{i+1} , y_{i+1}) and new decision parameter(P)

Step-6: plot pixel points (x_{i+1} , y_{i+1})

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:

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.

DDA Algorithm	Bresenham's Line Algorithm
1. DDA Algorithm use floating point, i.e., Real Arithmetic. This is the major reason that made the computations in DDA difficult than the bresenham algorithm.	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.
DDA is not optimized and less expensive.	Bresenham algorithm is optimized
DDA rounds off the value to the closest integer value.	The values in Bresenham never rounded off.
DDA output lines is not smooth	Bresenham output lines is smooth

Example-01: If a line is drawn from (2, 3) to (6, 15) with use of Bresenham's Line Drawing algorithm. How many points will be needed to generate such line? Solution:

x1 = 2 x2=6
y1 = 3 y2=15

dx = x2 – x1 = 6 -2 = 4
dy = y2- y1 = 15 – 3 = 12

m = dy / dx = 12 / 4 = 3
Here, m >= 1

initial decision parameter(P_i) = 2dx – dy
= (2*4 – 12)
P_i = - 4

x	y	Plot points (x, y)	Previous decision parameter(P)	Next decision parameter(P)
				P<0 : P = P + 2dx = P + 2*4 = P + 8 — x = x , y = y+ 1 P>=0 : P = P + 2dx – 2dy = P + (2*4) – (2*12) = P - 16 — x = x + 1 , y = y+ 1

2	3	=(2, 3)		Initial P = - 4
				X = 2
				Y = 3+1=4
				New points = (2, 4)

2	4	= (2, 4)	- 4	P = P+8 = -4 + 8 = 4
				X = 2+1 = 3
				Y = 4+1 = 5
				New points = (3, 5)

3	5	= (3, 5)	4	P = P – 16 = 4 – 16 = -12
				X = 3
				Y =5 +1 = 6
				New points = (3,6)

x	y	Plot points (x, y)	Previous decision parameter(P)	Next decision parameter(P) P<0 : P = P + 2dx = P + 2*4 = P + 8 P>=0 : P = P + 2dx – 2dy = P + (2*4) – (2*12) = P - 16 x = x , y = y+ 1 x = x + 1 , y = y+ 1

3	6	= (3,6)	-12	P = P + 8 = -12 + 8 = -4
				X = 3
				Y = 6+1 = 7
				New points = (3,7)

3	7	=(3,7)	-4	P = P + 8 = -4 + 8 = 4
				X = 3 + 1 = 4
				Y = 7 + 1 = 8
				New points = (4,8)

4	8	= (4,8)	4	P = P – 16 = 4 – 16 = -12
				X = 4
				Y = 8+1 = 9
				New points = (4,9)

4	9	= (4,9)	-12	P = P + 8 = -12 + 8 = -4
				X = 4
				Y = 9+1 =10
				New points = (4, 10)

4	10	= (4, 10)	-4	P = P + 8 = -4 + 8 = 4
				X = 4+1 =5
				Y = 10 +1 =11
				New points = (5, 11)

x	y	Plot points (x, y)	Previous decision parameter(P)	Next decision parameter(P) P<0 : P = P + 2dx = P + 2*4 = P + 8 P>=0 : P = P + 2dx – 2dy = P + (2*4) – (2*12) = P - 16 x = x , y = y+ 1 x = x + 1 , y = y+ 1

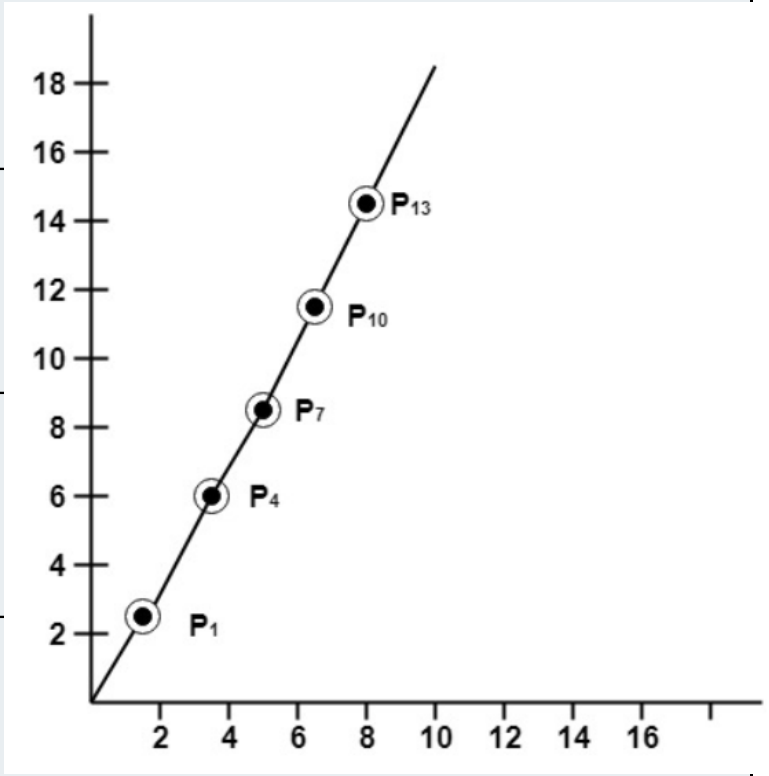
5	11	= (5, 11)	4	P = P – 16 = 4 – 16 = -12
				X = 5
				Y = 11+1=12
				New points = (5, 12)

5	12	= (5, 12)	-12	P = P + 8 = -12 + 8 = -4
				X = 5
				Y = 12+1=13
				New points = (5, 13)

5	13	= (5, 13)	-4	P = P + 8 = -4 + 8 = 4
				X = 5 +1 =6
				Y = 13+1 = 14
				New points = (6,14)

6	14	= (6,14)	4	P = P – 16 = 4 – 16 = -12
				X = 6
				Y = 14 + 1 = 15
				New points = (6,15)

6	15	= (6,15)	-12	**Since x = x2 and y = y2, so it will stop further execution.**



Example-02: If a line is drawn from (1,1) to (8,7) with use of Bresenham's Line Drawing algorithm. How many points will be needed to generate such line? Solution:

x1 = 1 x2=8
y1 = 1 y2=7

dx = x2 – x1 = 8 - 1 = 7
dy = y2- y1 = 7 – 1 = 6

m = dy / dx = 6 / 7 = 0.86
Here, m < 1

initial decision parameter(P_i) = 2dy - dx
= (2 * 6) - 7
P_i = 5

x	y	Plot points (x, y)	Previous decision parameter(P)	Next decision parameter(P)
				P<0 : P = P + 2dy = P + 2*6 = P + 12 → x = x+1 , y = y P>=0 : P = P + 2dy – 2dx = P + (2*6) – (2*7) = P – 2 → x = x + 1 , y = y+ 1

1	1	=(1,1)		Initial P = 5
				X = 1 + 1 = 2
				Y = 1 + 1 = 2
				New points = (2, 2)

2	2	= (2, 2)	5	P = P – 2 = 5 – 2 = 3
				X = 2 + 1= 3
				Y = 2 + 1 =3
				New points = (3, 3)

3	3	= (3, 3)	3	P = P – 2 = 3 -2 = 1
				X = 3 + 1 =4
				Y = 3 + 1 =4
				New points = (4, 4)

x	y	Plot points (x, y)	Previous decision parameter(P)	Next decision parameter(P) P<0 : P = P + 2dy = P + 2*6 = P + 12 P>=0 : P = P + 2dy – 2dx = P + (2*6) – (2*7) = P – 2 x = x+1 , y = y x = x + 1 , y = y+ 1

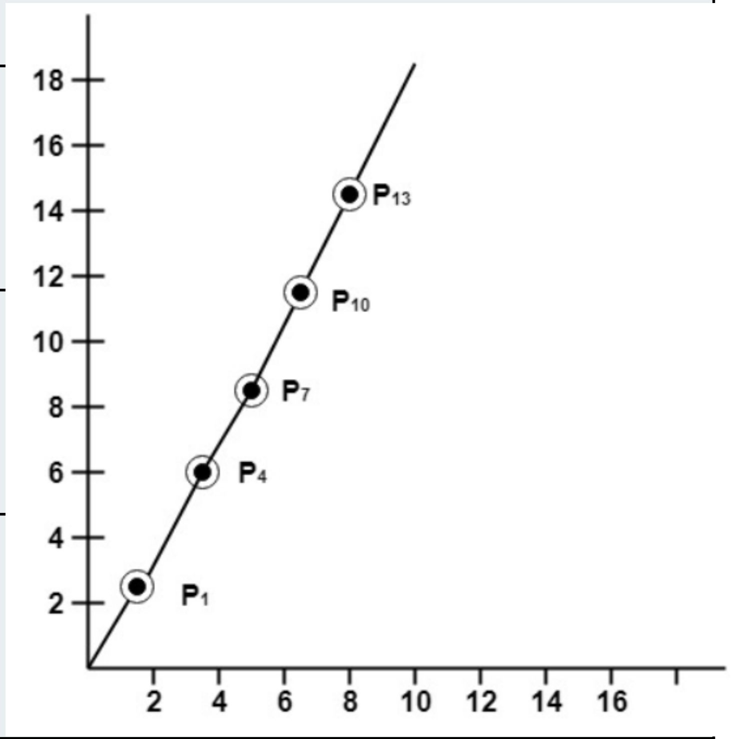
4	4	= (4, 4)	1	P = P – 2 = 1 – 2 = -1
				X = 4 + 1 = 5
				Y = 4
				New points = (5, 4)

5	4	= (5, 4)	-1	P = P + 12 = -1 + 12 = 11
				X = 5 + 1 =6
				Y = 4 + 1 = 5
				New points = (6,5)

6	5	= (6,5)	11	P = P – 2 = 11 – 2 = 9
				X = 6 + 1 = 7
				Y = 5 + 1 = 6
				New points = (7, 6)

7	6	= (7, 6)	9	P = P – 2 = 9 – 2 = 7
				X = 7 + 1 = 8
				Y = 6 + 1 = 7
				New points = (8, 7)

8	7	= (8, 7)	7	Since x=x2 and y=y2 So it stops further execution



Assignment

01. Draw a line using Bresenham’s Algorithm from (0,0) to (4,6).
02. Draw a line from (0,0) to (7,7) using Bresenham’s algorithm.
03. Use Bresenham’s Algorithm to draw a line from (2,3) to (9,8).
04. Calculate the points between the starting point (5, 6) and ending point (8, 12) by using Bresenham’s algorithm.

1. Draw a line using Bresenham's Algorithm from (0,0) to (4,6).

Solution:

x1 = 0 x2=4
y1 = 0 y2=6

dx = x2 – x1 = 4 - 0 = 4
dy = y2- y1 = 6 - 0 = 6

m = dy / dx = 6 / 4 = 1.5
Here, m >= 1

initial decision parameter(P_i) = 2dx – dy
= (2*4 – 6)
P_i = 2

x	y	Plot points (x, y)	Previous decision parameter(P)	Next decision parameter(P) P<0 : P = P + 2dx = P + 2*4 = P + 8 x = x , y = y+ 1 P>=0 : P = P + 2dx – 2dy = P + (2*4) – (2*6) = P - 4 x = x + 1 , y = y+ 1

0	0	= (0,0)		Initial P = 2
				X = x + 1 = 0 + 1 = 1
				Y = y + 1 = 0 + 1 = 1
				New points = (1, 1)

1	1	= (1, 1)	2	P = P – 4 = 2 – 4 = -2
				X = 1
				Y = 1+1 = 2
				New points = (2, 2)

1	2	= (1, 2)	-2	P = P + 8 = - 2 + 8 = 6
				X = x+1 = 1+1 =2
				Y =y+1 = 2 +1 =3 New points = (2, 3)

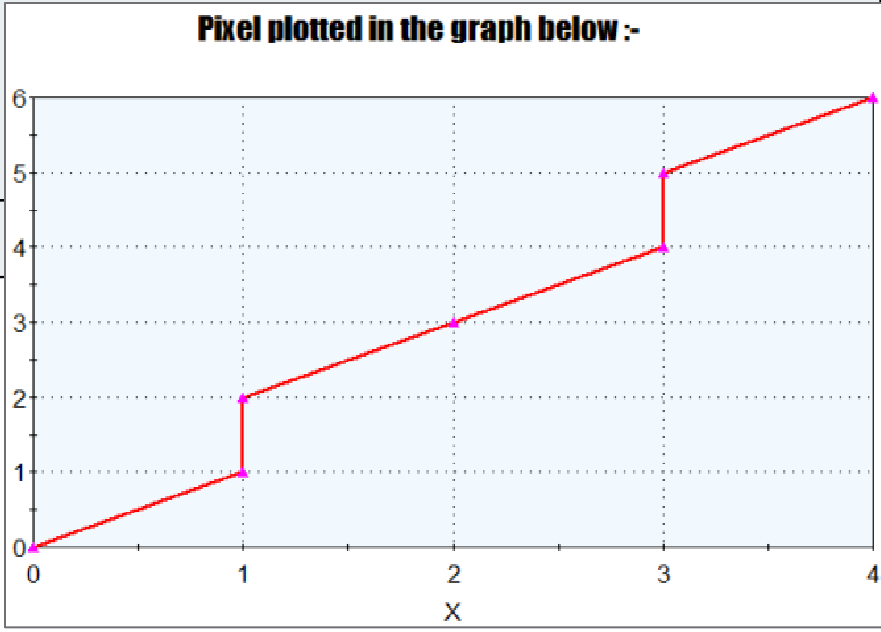
x	y	Plot points (x, y)	Previous decision parameter(P)	Next decision parameter(P) P<0 : P = P + 2dx = P + 2*4 = P + 8 x = x , y = y+ 1 P>=0 : P = P + 2dx – 2dy = P + (2*4) – (2*6) = P - 4 x = x + 1 , y = y+ 1

4	4	=(4,4)	7	P = P – 4 = 6 – 4 = 2
				X = x+1 = 2+1= 3
				Y = y+1 = 3+1= 4 New points = (3, 4)

3	4	= (3,4)	7	P = P – 4 = 2 – 4= - 2
				X = 3
				Y = y+1 = 4 + 1 = 5 New points = (3, 5)

3	5	= (3, 5)	7	P = P + 8 = -2+8 = 6
				X = x+1=3+1=4
				Y = y+1=5+1=6
				New points = (4, 6)

4	6	= (4, 6)	$x = x_{end}$, So the process is stopped	



02. Draw a line from (0,0) to (7,7) using Bresenham's Algorithm.

Solution:

x1 = 0 x2=7
y1 = 0 y2=7

dx = x2 – x1 = 7 - 0 = 7
dy = y2- y1 = 7 - 0 = 7

m = dy / dx = 7 / 7 = 1
Here, m >= 1

initial decision parameter(P_i) = 2dx – dy
= (2*7 – 7)
P_i = 7

x	y	Plot points (x, y)	Previous decision parameter(P)	Next decision parameter(P) P<0 : P = P + 2dx = P + 2*7 = P + 14 x = x , y = y + 1 P>=0 : P = P + 2dx – 2dy = P + (2*7) – (2*7) = P x = x + 1 , y = y + 1

0	0	= (0,0)		Initial P = 7
				X = x + 1 = 0 + 1 = 1
				Y = y + 1 = 0 + 1 = 1 New points = (1, 1)

1	1	= (1, 1)	7	P = 7
				X = 1+1 = 2
				Y = 1+1 = 2 New points = (2, 2)

2	2	= (2, 2)	7	P = 7
				X = x+1 = 2+1 = 3
				Y = y+1 = 2 +1 =3 New points = (3, 3)

3	3	= (3, 3)	7	P = 7
				X = x+1 = 3+1 = 4
				Y = y+1 = 3 +1 =4 New points = (4, 4)

x	y	Plot points (x, y)	Previous decision parameter(P)	Next decision parameter(P)
				P<0 : P = P + 2dx = P + 2*7 = P + 14 x = x , y = y + 1 P>=0 : P = P + 2dx – 2dy = P + (2*7) – (2*7) = P x = x + 1 , y = y + 1

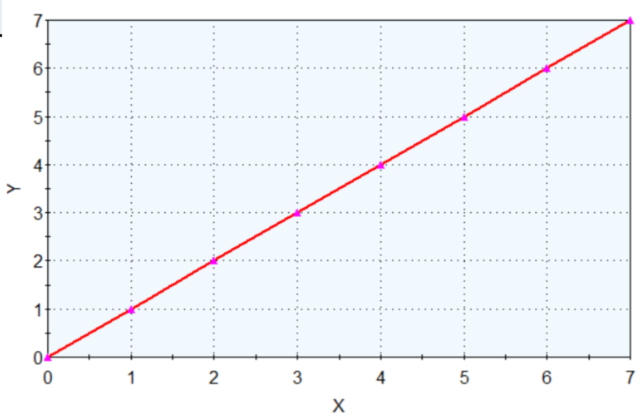
4	4	= (4, 4)	7	P = 7
				X = x+1 = 4+1 = 5
				Y = y+1 = 4 +1 = 5 New points = (5, 5)

5	5	= (5, 5)	7	P = 7
				X = 5+1 = 6
				Y = 5+1 = 6 New points = (6, 6)

6	6	= (6, 6)	7	P = 7
				X = x+1 = 6+1 = 7
				Y = y+1 = 6 +1 =7 New points = (7, 7)

7	7	= (7, 7)	Here, $x = x_{end,}$ So it stops further execution	

Pixel plotted in the graph below :-



03.Use Bresenham's Algorithm to Draw a line from (2,3) to (9,8) .

Solution:

x1 = 2 x2=9
y1 = 3 y2=8

dx = x2 – x1 = 9 - 2 = 7
dy = y2 - y1 = 8 - 3 = 5

m = dy / dx = 5 / 7 = 0.71
Here, m < 1

initial decision parameter(P_i) = 2dy - dx
= (2 * 5) - 7
P_i = 3

x	y	Plot points (x, y)	Previous decision parameter(P)	Next decision parameter(P)
				Next decision parameter(P) P<0 : P = P + 2dy = P + 2*5= P + 10 P>=0 : P = P + 2dy – 2dx = P + (2*5) – (2*7) = P – 4 x = x+1 , y = y x = x + 1 , y = y+ 1

2	3	=(2, 3)		Initial P = 3
				X = 2 + 1 = 3
				Y = 3 + 1 = 4 New points = (3, 4)

3	4	= (3, 4)	3	P = P – 4 = 3 – 4 = -1
				X = x+1 = 3+1 = 4
				Y = 4 New points = (4, 4)

4	4	= (4, 4)	-1	P = P + 10 = -1 + 10 = 9
				X = x+1 = 4+1 = 5
				Y = y+1 = 4+1 = 5 New points = (5, 5)

5	5	= (5, 5)	9	P = P – 4 = 9- 4=5
				X = x+1 = 5 + 1 = 6
				Y = y+1 = 5 + 1 = 6 New points = (6, 6)

x	y	Plot points (x, y)	Previous decision parameter(P)	Next decision parameter(P) P<0 : P = P + 2dy = P + 2*5= P + 10 P>=0 : P = P + 2dy – 2dx = P + (2*5) – (2*7) = P – 4 x = x+1 , y = y x = x + 1 , y = y+ 1

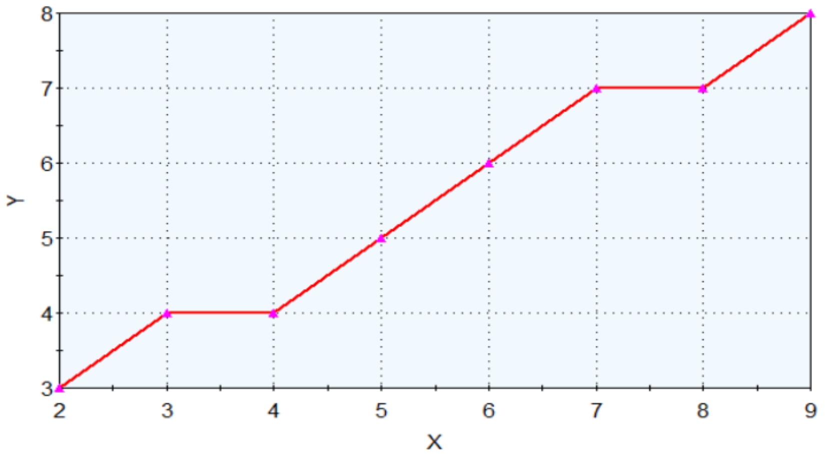
6	6	= (6, 6)	1	P = P – 4 = 1 – 4 = -3
				X = x+1 = 6+1 = 7
				Y=6 New points = (7, 6)

7	6	= (7, 6)	-3	P = P + 10 = -3 + 10 = 7
				X = x+1 = 7 + 1 = 8
				Y = y+1 = 6 + 1 = 7 New points = (8, 7)

8	7	= (8, 7)	7	P = P – 4 = 7 – 4 =3
				X = x+1 = 8 + 1 = 9
				Y = y+1 = 7 + 1 = 8 New points = (9, 8)

9	8	= (9, 8)	Here, x = x_{end}, So it stops further execution	

Pixel plotted in the graph below :-



04. Calculate the points between the starting point (5,6) and ending point (8,12) by using Bresenham's Algorithm.
Solution:

x1 = 5 x2=8
y1 = 6 y2=12

dx = x2 – x1 = 8 - 5 = 3
dy = y2- y1 = 12 - 6 = 6

m = dy / dx = 6/ 3 = 2
Here, m >= 1

initial decision parameter(P_i) = 2dx – dy
= (2*3 – 6)
P_i = 0

x	y	Plot points (x, y)	Previous decision parameter(P)	Next decision parameter(P) P<0 : P = P + 2dx = P + 2*3 = P + 6 (x = x , y = y + 1) P>=0 : P = P + 2dx – 2dy = P + (2*3) – (2*6) = P - 6 (x = x + 1 , y = y + 1)

5	6	=(5, 6)		Initial P = 0
				X = x + 1 = 5 + 1 = 6
				Y = y + 1 = 6 + 1 = 7 New points = (6, 7)

6	7	= (6, 7)	0	P = P – 6 = 0 – 6 = -6
				X = 6
				Y = 7+1 = 8 New points = (6, 8)

6	8	= (6, 8)	-6	P = P + 6 = -6 + 6 = 0
				X = x + 1 = 6 + 1 = 7
				Y = y + 1 = 8 + 1 = 9 New points = (7, 9)

7	9	= (7, 9)	0	P = P – 6 = 0 – 6 = -6
				X = 7
				Y = 9+1 = 10 New points = (7, 10)

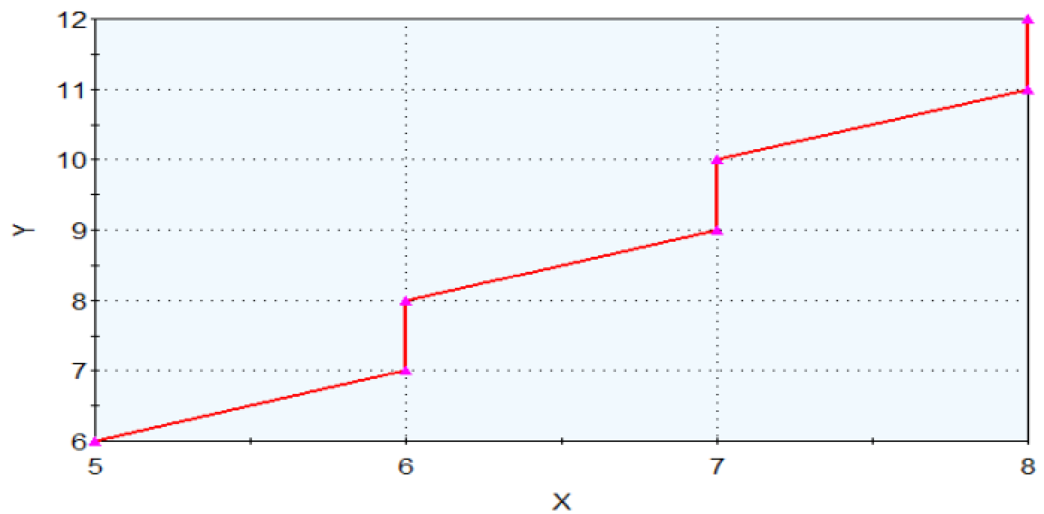
x	y	Plot points (x, y)	Previous decision parameter(P)	Next decision parameter(P) P<0 : P = P + 2dx = P + 2*3 = P + 6 (x = x , y = y + 1) P>=0 : P = P + 2dx – 2dy = P + (2*3) – (2*6) = P - 6 (x = x + 1 , y = y + 1)

7	10	= (7, 10)	-6	P = P + 6 = -6 + 6 = 0
				X = x + 1 = 7 + 1 = 8
				Y = y + 1 = 10 + 1 = 11 New points = (8, 11)

8	11	= (8, 11)	0	P = P – 6 = 0 – 6 = -6
				X = 8
				Y = 11+1 = 12 New points = (8, 12)

8	12	= (8, 12)	Here, x = x_{end}, So it stops further execution	

Pixel plotted in the graph below :-



12
11
10
9
8
7
6
y
5
6
7
8
x