GALGOTIAS UNIVERSITY

School of Computing Science and Engineering

Program: B.C.A.

Course Code: BCAS3003

Course Name: Computer Graphics

Vision

To be known globally as a premier department of Computer Science and Engineering for value-based education, multidisciplinary research and innovation.

Mission

- **M1:** Developing a strong foundation in fundamentals of computing science with responsiveness towards emerging technologies.
- **M2:** Establishing state-of-the-art facilities and adopt education 4.0 practices to analyze, develop, test and deploy sustainable ethical IT solutions by involving multiple stakeholders.
- **M3:** Establishing Centers of Excellence for multidisciplinary collaborative research in association with industry and academia.

Course Outcomes (COs)

CO Number	Title
CO1	Describe the fundamental concepts of Computer Graphics. (K1)
CO2	To demonstrate with the relevant mathematics of computer graphics, ex. line, circle and ellipse drawing algorithms. (K3)
CO3	To understand the attributes of output primitives of Graphics. (K2).
CO4	Apply simple and composite transformation on graphic objects/elements in two dimensions. (K3).
CO5	Analyze two dimensions modeling and clipping techniques. (K4).
CO6	List out the various contemporary research areas and tool in graphics domain. (K2).

Course Prerequisites

- Knowledge of Mathematics
- Fundamental knowledge of Computer

Syllabus

Unit 2 – Output Primitives (8 hours)

- Line Drawing Algorithms
- Circle Generation Algorithms
- Ellipse Generating Algorithm
- Pixel Addressing
- Filled-Area Primitives
- Fill Area Function,
- Cell Array, Character Generation

Recommended Books

Text books

- D. Hearn, P. Baker, "Computer Graphics - C Version", 2nd Edition, Pearson Education, 1997

Reference Book

- Heam Donald, Pauline Baker M: "Computer Graphics", PHI 2nd Edn. 1995.
- Harrington S: "Computer Graphics - A Programming Approach", 2nd Edn. Mc GrawHill.
- Shalini Govil-Pai, Principles of Computer Graphics, Springer, 2004

Additional online materials

- Coursera - https://www.coursera.org/learn/fundamentals-of-graphic-design
- https://www.youtube.com/watch?v=fwzYuhduME4&list=PLE4D97E3B8DB8A590
- NPTEL - https://nptel.ac.in/courses/106/106/106106090/
- https://www.coursera.org/learn/research-methods
- https://www.coursera.org/browse/physical-science-and-engineering/research-methods

Line Drawing Algorithms

- Scan Conversion Definition
- Scan Converting a Point
- Scan Converting a Straight Line
- DDA Algorithm
- Bresenham's Line Algorithm

Scan Conversion Definition

- It is a process of representing graphics objects a collection of pixels. The graphics objects are continuous. The pixels used are discrete. Each pixel can have either on or off state.
- 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.
- Most human beings think graphics objects as points, lines, circles, ellipses.
- For generating graphical object, many algorithms have been developed.

Scan Conversion Definition

- Examples of objects which can be scan converted are Point, Line, Sector, Arc, Ellipse, Rectangle, Polygon, Characters, Filled Regions
- The term pixel is a short form of the picture element. It is also called a point or dot.
- It is the smallest picture unit accepted by display devices. A picture is constructed from hundreds of such pixels.
- Pixels are generated using commands. Lines, circle, arcs, characters; curves are drawn with closely spaced pixels. To display the digit or letter matrix of pixels is used.
- The closer the dots or pixels are, the better will be the quality of picture. So the quality of the picture is directly proportional to the density of pixels on the screen.
- Pixels are also defined as the smallest addressable unit or element of the screen. Each pixel can be assigned an address as shown in Figure below:

Scan Conversion Definition

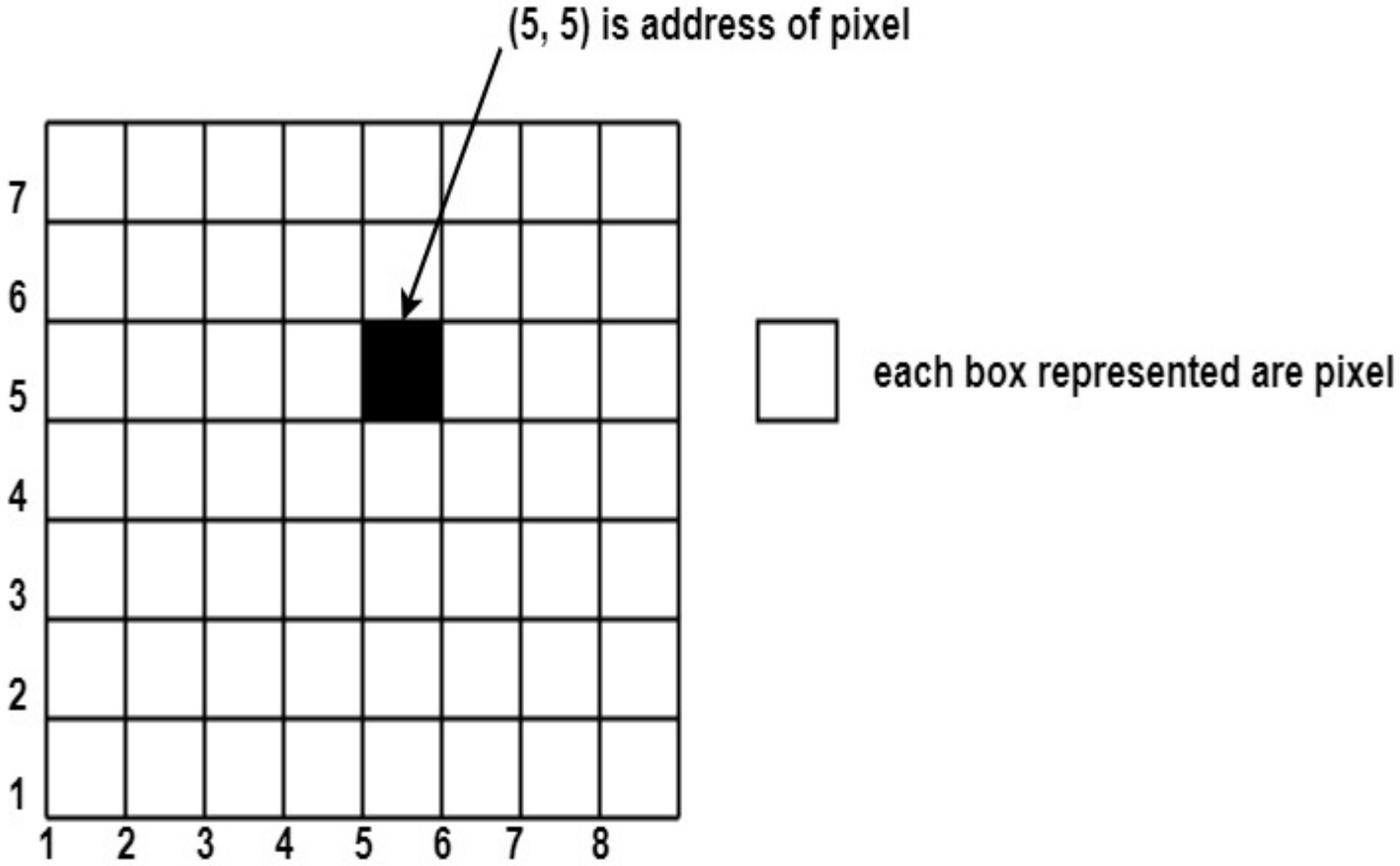


Figure 1: Pixel Representation

Scan Conversion Definition

- Different graphics objects can be generated by setting the different intensity of pixels and different colors of pixels.
- Each pixel has some co-ordinate value. The coordinate is represented using row and column.
- P (5, 5) used to represent a pixel in the 5th row and the 5th column. Each pixel has some intensity value which is represented in memory of computer called a **frame buffer or refresh buffer**.
- This memory is a storage area for storing pixels values using which pictures are displayed. It is also called as **digital memory**.
- Inside the buffer, image is stored as a pattern of binary digits either 0 or 1. So there is an array of 0 or 1 used to represent the picture.
- In black and white monitors, black pixels are represented using 1's and white pixels are represented using 0's. In case of systems having one bit per pixel frame buffer is called a **bitmap**. In systems with multiple bits per pixel it is called a **pixmap**.

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.
- **Example:** Display coordinates points P1(2¼, 1¾) and P2 (2 2/3, 1 ¼) would both be represented by pixel (2, 1). In general, a point p (x, y) is represented by the integer part of x & the integer part of y that is pixels [(INT (x), INT (y).

Scan Converting a Straight Line

- A straight line may be defined by two endpoints & an equation. In Fig 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.

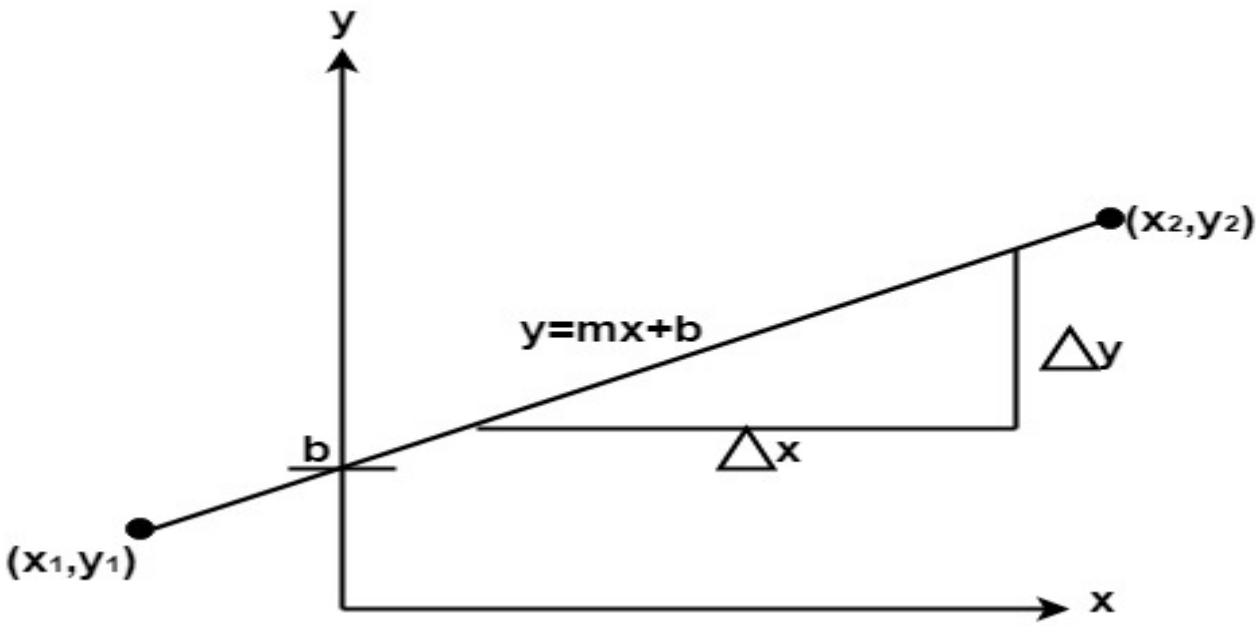


Figure 2: Line Representation

Scan Converting a Straight Line

- Using the equation of a straight line, y = mx + b where m = Δy/Δx and b = the y intercept.
- 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.

Properties of Good Line Drawing Algorithm

- **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.

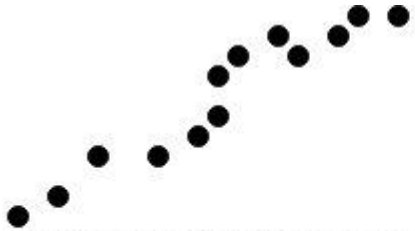


Fig: O/P from a poor line generating algorithm

Scan Converting a Straight Line

Properties of Good Line Drawing Algorithm

- **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.
- **Lines should terminate accurately:** Unless lines are plotted accurately, they may terminate at the wrong place.
- **Lines should have constant density:** Line density is proportional to the number of dots displayed divided by the length of the line. To maintain constant density, dots should be equally spaced.
- **Line should be drawn rapidly:** This computation should be performed by special-purpose hardware.

Algorithm for line Drawing

- **Direct use of line equation**
- **DDA (Digital Differential Analyzer)**
- **Bresenham's Algorithm**

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 P2 (x2' y2').
- Then m = (y2‘-y1')/(x2‘-x1') and b = $y_1^1 + m\ x_1^1$
- If value of |m|≤1 for each integer value of x. But do not consider x_1^1 and x_2^2
- If value of |m|>1 for each integer value of y. But do not consider y_1^1 and y_2^2
- **Example:** A line with starting point as P(0, 0) and ending point P (6, 18) is given. Calculate value of intermediate points and slope of line.
- Solution: P1 (0,0) P7 (6,18)

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

  $m = \Delta y/\Delta x = 3$

  We know equation of line is y =m x + b

  y = 3x + b.....equation (1)

Direct use of Line Equation:

- put value of x from initial point in equation (1), i.e., (0, 0) x =0, y=0

$0 = 3\ x^* 0 + b \Longrightarrow b=0$

put b = 0 in equation (1)

$y = 3x + 0 = 3x$

Now calculate intermediate points

Let $x = 1 \Longrightarrow y = 3 \times 1 \Longrightarrow y = 3$

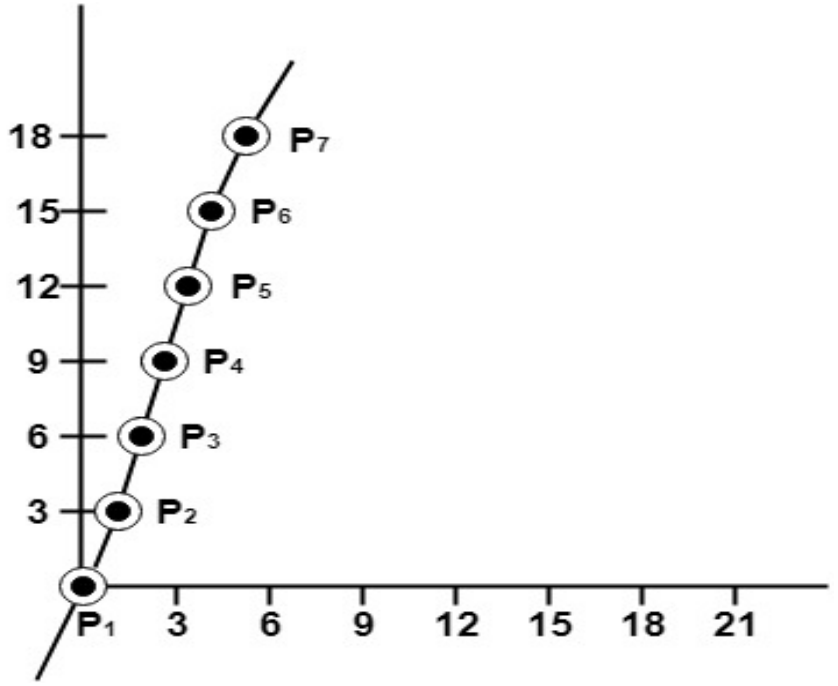
Let $x = 2 \Longrightarrow y = 3 \times 2 \Longrightarrow y = 6$

Let $x = 3 \Longrightarrow y = 3 \times 3 \Longrightarrow y = 9$

Let $x = 4 \Longrightarrow y = 3 \times 4 \Longrightarrow y = 12$

Let $x = 5 \Longrightarrow y = 3 \times 5 \Longrightarrow y = 15$

Let $x = 6 \Longrightarrow y = 3 \times 6 \Longrightarrow y = 18$



So points are P1 (0,0), P2 (1,3), P3 (2,6), P4 (3,9), P5 (4,12), P6 (5,15), P7 (6,18)

DDA Algorithm

- DDA stands for Digital Differential Analyzer. It is an incremental method of scan conversion of line.
- In this method calculation is performed at each step but by using results of previous steps.
- Suppose at step i, the pixels is (xi,yi)
- The line of equation for step i is

$$y_i = m^* x_i + b \text{.....equation 1}$$

- Next value will be

$$y_{i+1} = m^* x_{i+1} + b \text{.....equation 2}$$

$$m = \Delta y/\Delta x$$

$$y_{i+1}-y_i=\Delta y \text{.....equation 3}$$

$$x_{i+1}-x_i=\Delta x \text{.....equation 4}$$

$$y_{i+1} = y_i + \Delta y;\ \Delta y = m\,\Delta x;\ y_{i+1} = y_i + m\,\Delta x$$

$$\Delta x=\Delta y/m;\ x_{i+1}=x_i+\Delta x;\ x_{i+1}=x_i+\Delta y/m$$

DDA Algorithm

- Case1: When |m|<1 then (assume that x_1^2)

  x = x1, y=y1 set Δx = 1

  $y_{i+1} = y_1 + m$, x = x + 1

  Until x = x2

- Case2: When |M|<1 then (assume that y_1^2)

  x= x1, y=y1 set Δy = 1

  $x_{i+1} = 1/m$, y=y+1

  Until y → y2

DDA Algorithm

Advantage

- It is a faster method than method of using direct use of line equation.
- This method does not use multiplication theorem.
- It allows us to detect the change in the value of x and y ,so plotting of same point twice is not possible.
- This method gives overflow indication when a point is repositioned.
- It is an easy method because each step involves just two additions.
- Time Complexity of DDA algorithm O(max(|dx|,|dy|)).

Disadvantage

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

DDA Algorithm

Given-

Starting coordinates = (X0, Y0)

Ending coordinates = (Xn, Yn)

- **Step-01:** Calculate ΔX, ΔY and M from the given input. These parameters are calculated as-

$$\Delta X = Xn - X0$$

$$\Delta Y = Yn - Y0$$

$$M = \Delta Y / \Delta X$$

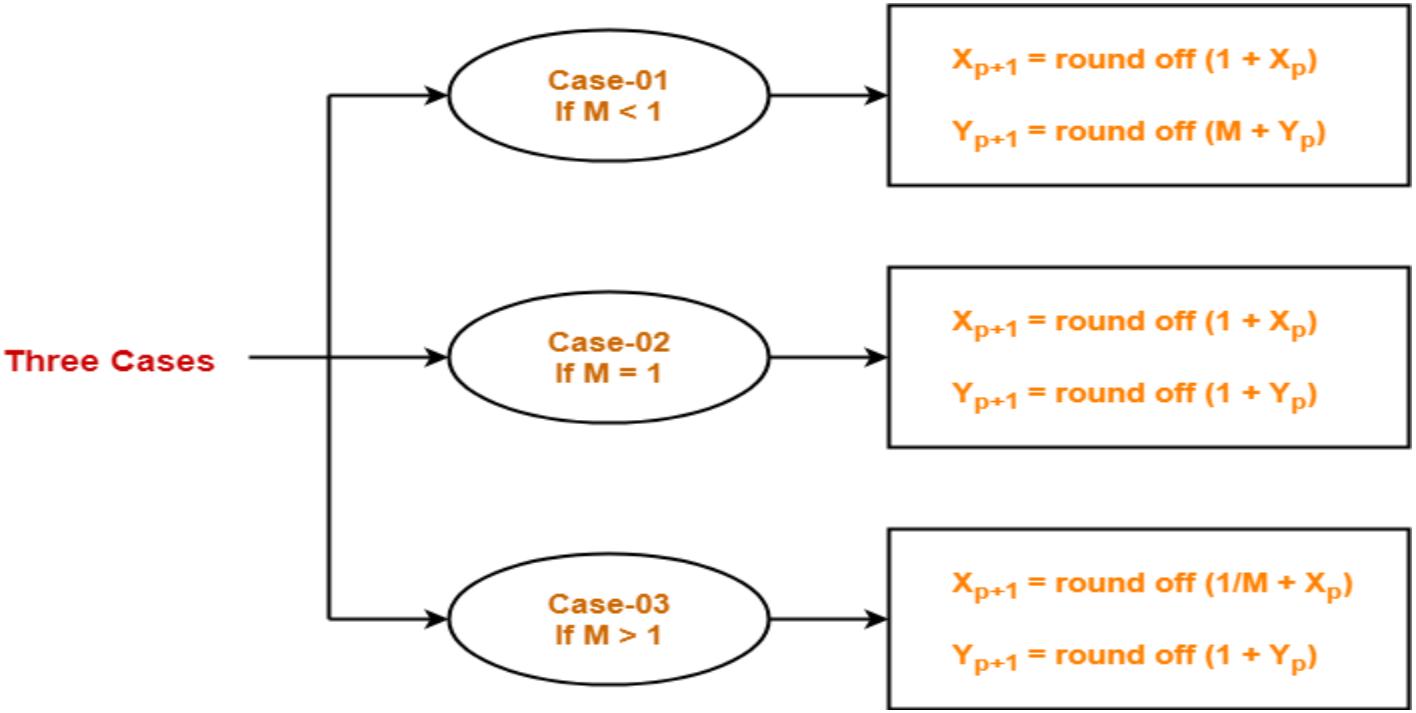
- **Step-02:** Find the number of steps or points in between the starting and ending coordinates.

```
if (absolute (ΔX) > absolute (ΔY))
        Steps = absolute (ΔX)
Else
        Steps = absolute (ΔY)
```

DDA Algorithm

- **Step-03**: Suppose the current point is (Xp, Yp) and the next point is (Xp+1, Yp+1).

Find the next point by following the below three cases-



DDA Example

- Example 1: If a line is drawn from (2, 3) to (6, 15) with use of DDA. How many points will needed to generate such line?
- Solution: P1 (2,3) P11 (6,15)

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

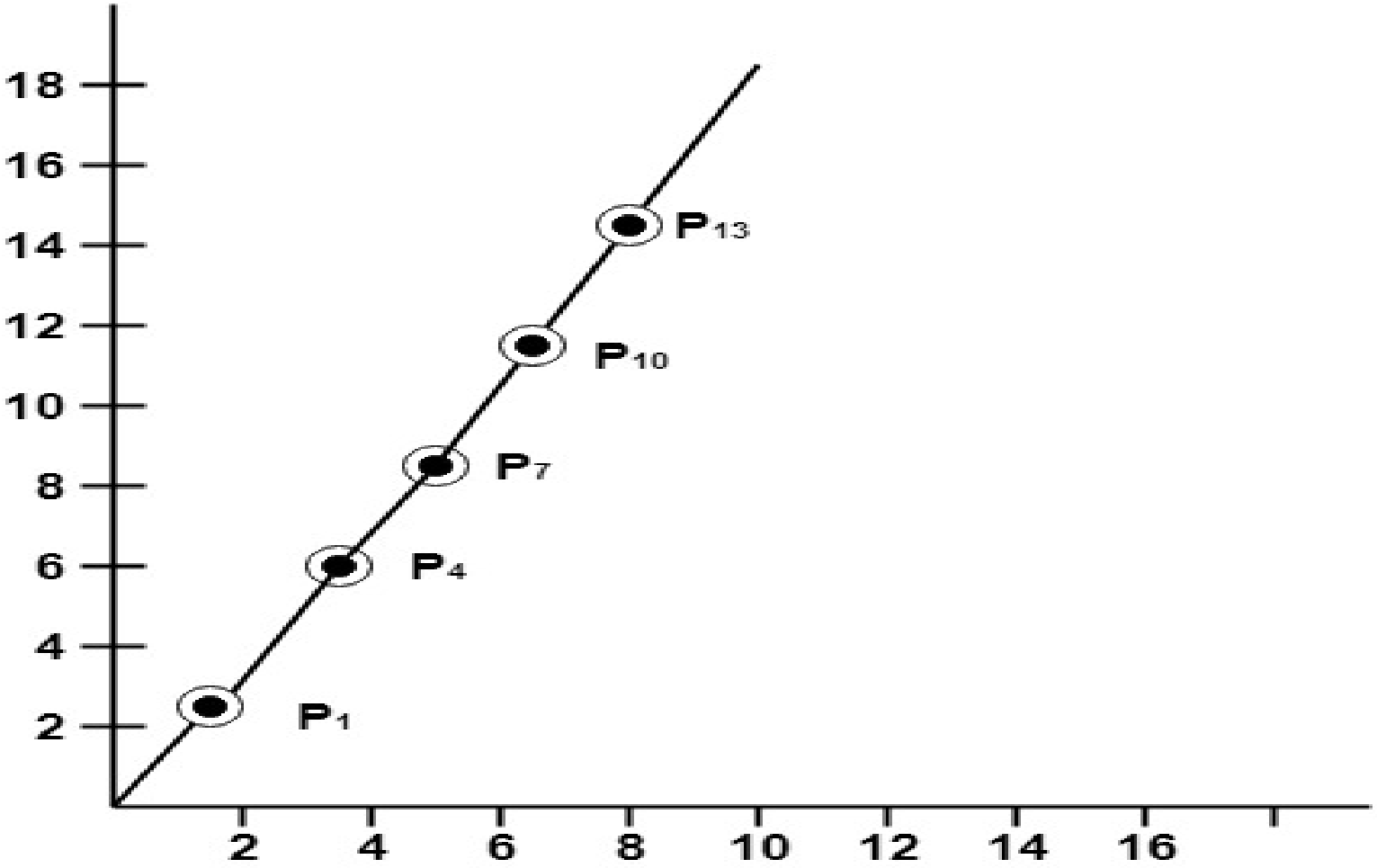
  dx = 6 - 2 = 4; dy = 15 - 3 = 12

  m = 12/4 = 3

  For calculating next value of x takes x = x + 1/m

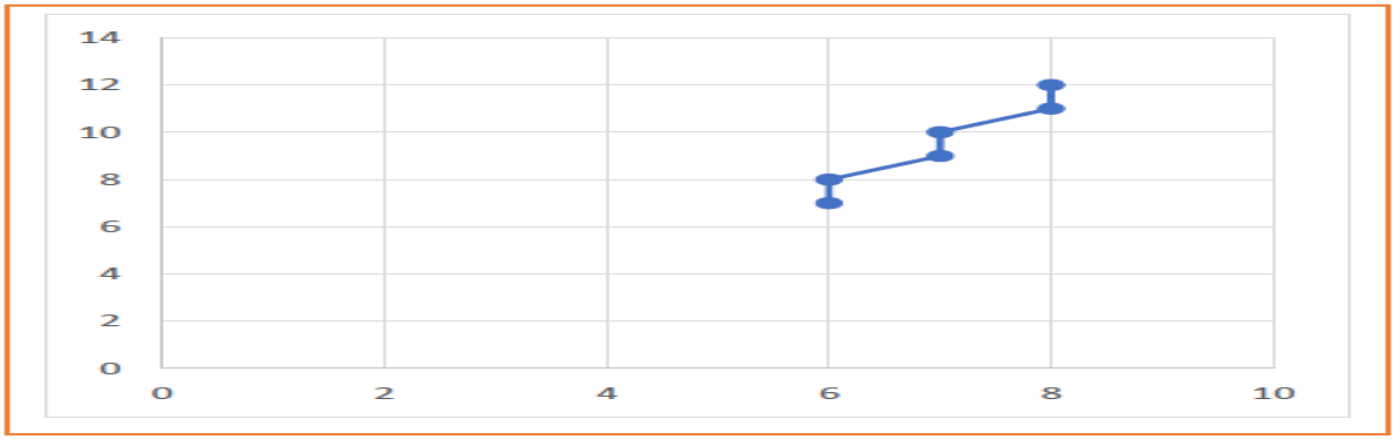
$P_1(2,3)$	point plotted	$P_5(3\frac{1}{3},7)$	point not plotted	$P_9(4\frac{2}{3},11)$	point not plotted
$P_2(2\frac{1}{3},4)$	point plotted	$P_6(3\frac{2}{3},8)$	point not plotted	$P_{10}(5,12)$	point plotted
$P_3(2\frac{2}{3},5)$	point not plotted	$P_7(4,9)$	point plotted	$P_{11}(5\frac{1}{3},13)$	point not plotted
$P_4(3,6)$	point plotted	$P_8(4\frac{1}{3},10)$	point not plotted	$P_{12}(5\frac{2}{3},14)$	point not plotted
				$P_{13}(6,15)$	point plotted

DDA Example



DDA Example

X_p	Y_p	X_{p+1}	Y_{p+1}	Round off (X_{p+1}, Y_{p+1})
5	6	5.5	7	(6, 7)
		6	8	(6, 8)
		6.5	9	(7, 9)
		7	10	(7, 10)
		7.5	11	(8, 11)
		8	12	(8, 12)



DDA Example

Example 1: Calculate the points between the starting point (5, 6) and ending point (8, 12).

Solution: Given-

Starting coordinates = (X0, Y0) = (5, 6)

Ending coordinates = (Xn, Yn) = (8, 12)

Step-01: Calculate ΔX, ΔY and M from the given input.

$$\Delta X = Xn - X0 = 8 - 5 = 3$$

$$\Delta Y = Yn - Y0 = 12 - 6 = 6$$

$$M = \Delta Y / \Delta X = 6 / 3 = 2$$

Step-02: Calculate the number of steps.

As $|\Delta X| < |\Delta Y| = 3 < 6$, so number of steps = $\Delta Y = 6$

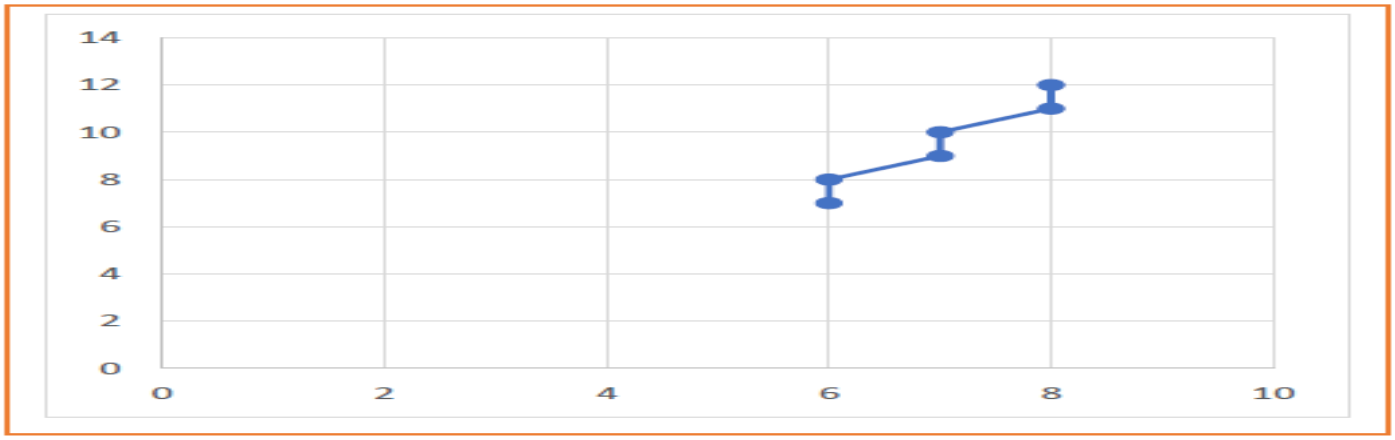
Step-03:

As M > 1, so case-03 is satisfied.

Now, Step-03 is executed until Step-04 is satisfied.

DDA Example

X_p	Y_p	X_{p+1}	Y_{p+1}	Round off (X_{p+1}, Y_{p+1})
5	6	5.5	7	(6, 7)
		6	8	(6, 8)
		6.5	9	(7, 9)
		7	10	(7, 10)
		7.5	11	(8, 11)
		8	12	(8, 12)



DDA Example

Example 2: Calculate the points between the starting point (5, 6) and ending point (13, 10).

Solution: Given-

Starting coordinates = (X0, Y0) = (5, 6)

Ending coordinates = (Xn, Yn) = (13, 10)

Step-01: Calculate ΔX, ΔY and M from the given input.

$$\Delta X = Xn - X0 = 13 - 5 = 8$$

$$\Delta Y = Yn - Y0 = 10 - 6 = 4$$

$$M = \Delta Y / \Delta X = 4 / 8 = 0.50$$

Step-02: Calculate the number of steps.

As $|\Delta X| > |\Delta Y| = 8 > 4$, so number of steps = $\Delta X = 8$

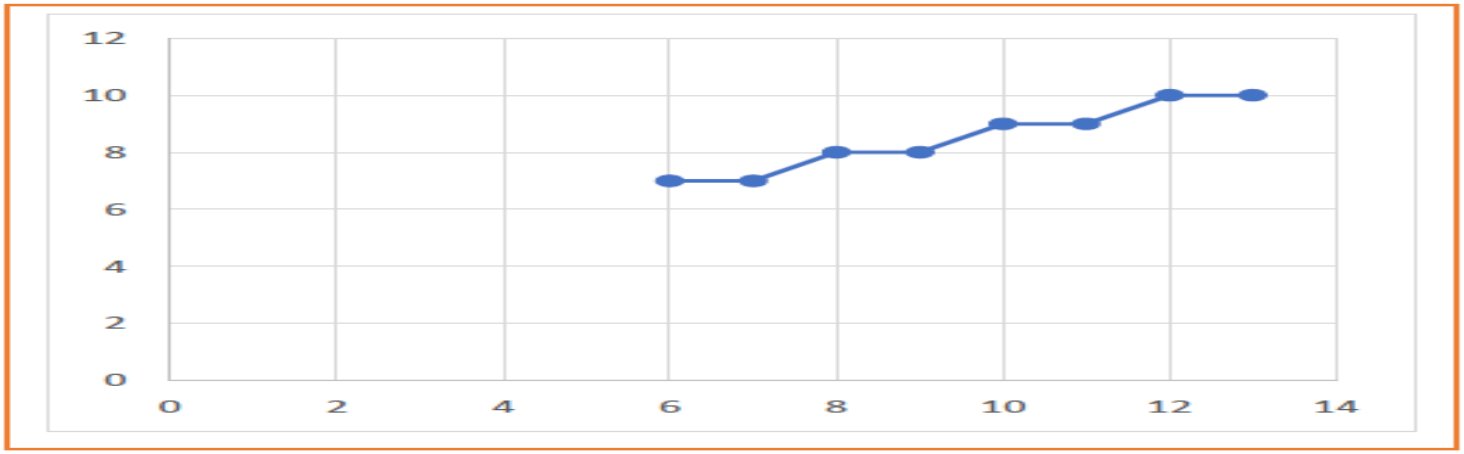
Step-03:

As M < 1, so case-01 is satisfied.

Now, Step-03 is executed until Step-04 is satisfied.

DDA Example

X_p	Y_p	X_{p+1}	Y_{p+1}	Round off (X_{p+1}, Y_{p+1})
5	6	6	6.5	(6, 7)
		7	7	(7, 7)
		8	7.5	(8, 8)
		9	8	(9, 8)
		10	8.5	(10, 9)
		11	9	(11, 9)
		12	9.5	(12, 10)
		13	10	(13, 10)



DDA Example

Example 3: Calculate the points between the starting point (1, 7) and ending point (11, 17).

Solution: Given-

Starting coordinates = (X0, Y0) = (1, 7)

Ending coordinates = (Xn, Yn) = (11, 17)

Step-01: Calculate ΔX, ΔY and M from the given input.

$$\Delta X = Xn - X0 = 11 - 1 = 10$$

$$\Delta Y = Yn - Y0 = 17 - 7 = 10$$

$$M = \Delta Y / \Delta X = 10 / 10 = 1$$

Step-02: Calculate the number of steps.

As $|\Delta X| = |\Delta Y| = 10 = 10$, so number of steps = $\Delta X = \Delta Y = 10$

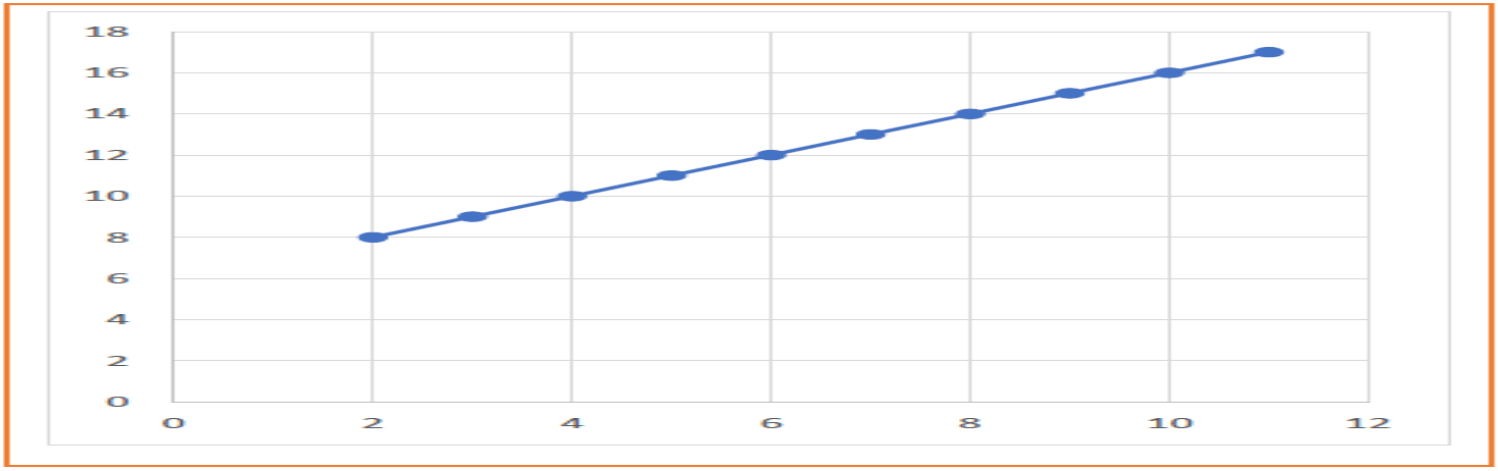
Step-03:

As M = 1, so case-02 is satisfied.

Now, Step-03 is executed until Step-04 is satisfied.

DDA Example

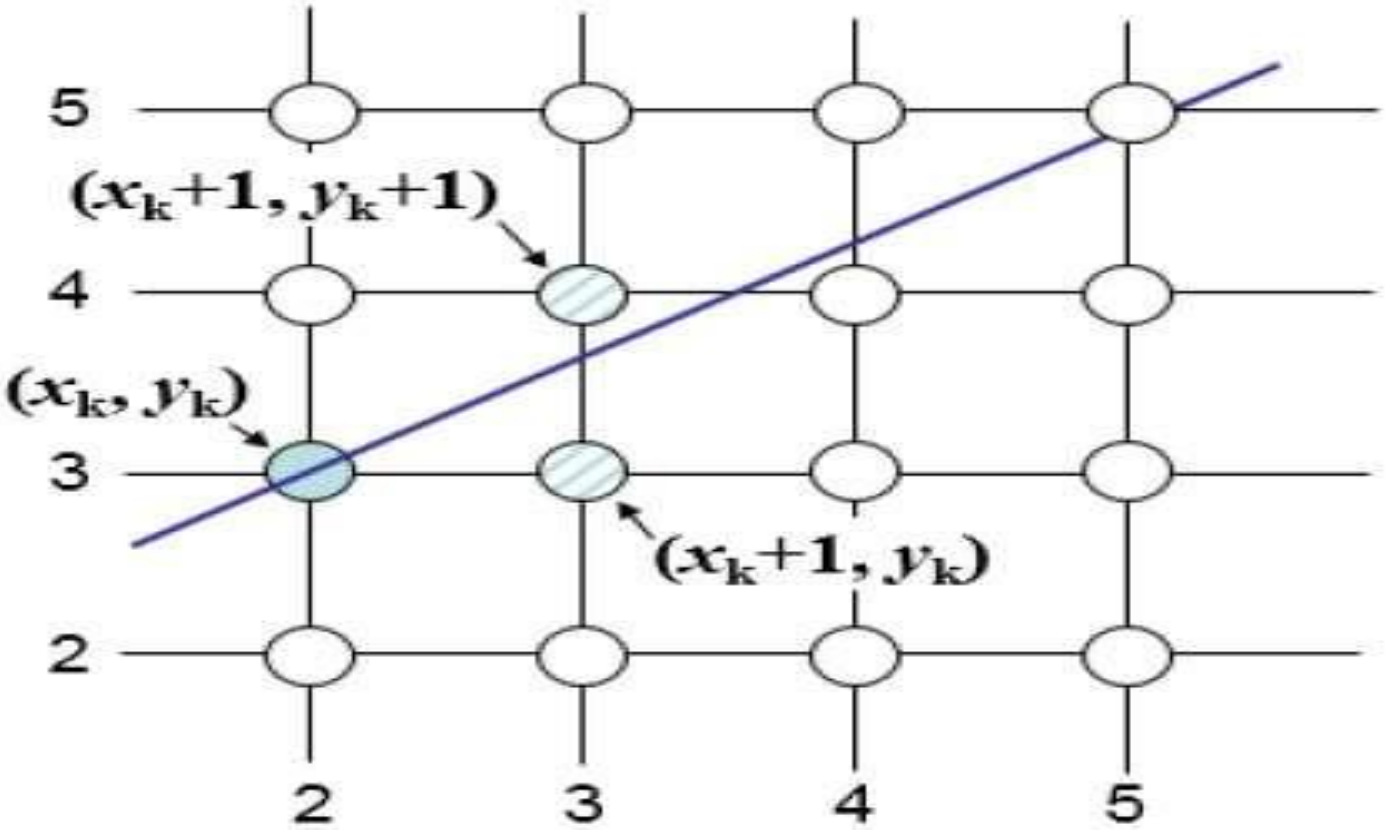
X_p	Y_p	X_{p+1}	Y_{p+1}	Round off (X_{p+1}, Y_{p+1})
1	7	2	8	(2, 8)
		3	9	(3, 9)
		4	10	(4, 10)
		5	11	(5, 11)
		6	12	(6, 12)
		7	13	(7, 13)
		8	14	(8, 14)
		9	15	(9, 15)
		10	16	(10, 16)
		11	17	(11, 17)



Bresenham's Line Drawing Algorithm

- 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.
- 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) we need to choose between (3,3) and (3,4).
- We would like the point that is closer to the original line.

Bresenham's Line Drawing Algorithm



Bresenham's Line Drawing Algorithm

- 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 P2'(x2',y2').
- The line is best approximated by those pixels that fall the least distance from the path between P_1',P_2'.

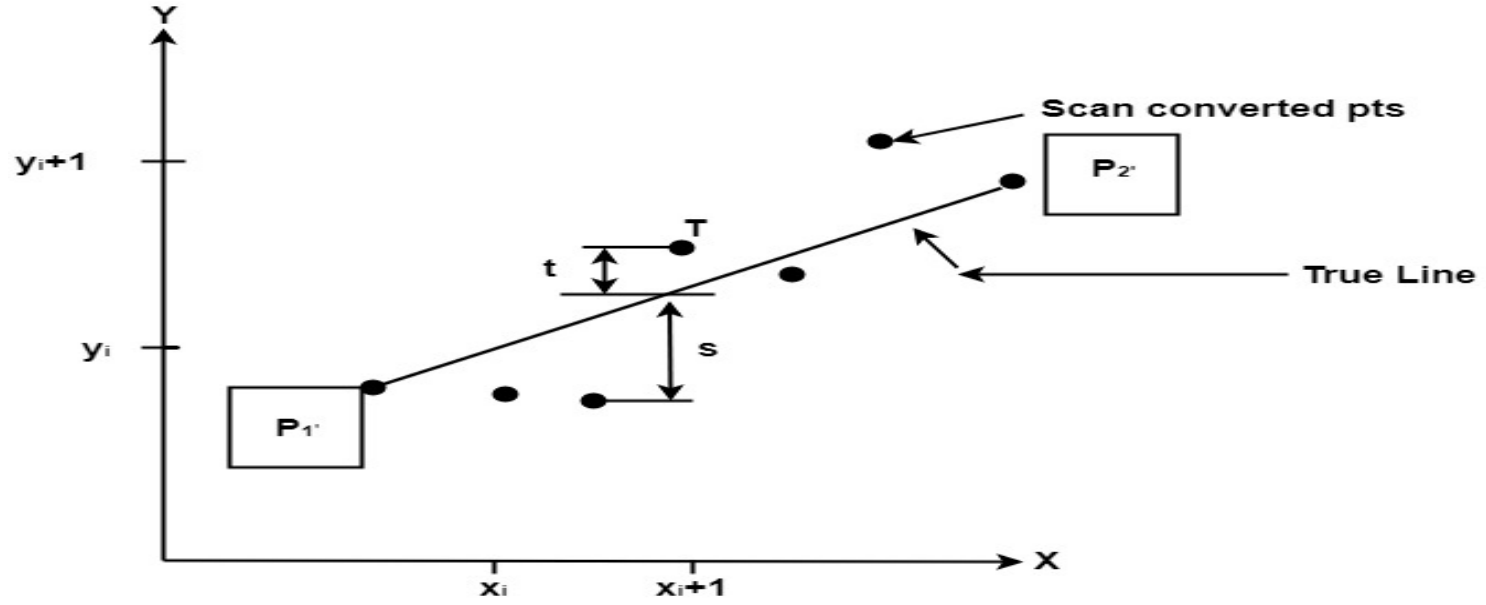


Fig: Scan Converting a line.

Bresenham's Line Drawing Algorithm

- 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

  $y = mx_{i+1} + b = m(x_i + 1) + b$

- 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

  $s - t$

Bresenham's Line Drawing Algorithm

- 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

$$s-t = (y-yi)-[(yi+1)-y]$$
$$= 2y - 2yi -1$$

$s-t = 2m(xi + 1) + 2b – 2yi – 1$ [putting value of 'y']

Substituting m by $\Delta y/\Delta x$ and introducing decision variable

$d_i=\Delta x\ (s-t)$

$d_i = \Delta x\ [2\ \Delta y/\Delta x(x_i+1) + 2b-2y_i-1)]$

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

$d_i = 2\Delta y.x_i-2\Delta x.y_i+c$; Where $c= 2\Delta y+\Delta x\ (2b-1)$

Bresenham's Line Drawing Algorithm

- 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., $d_i \geq 0) \Longrightarrow y_{i+1} = y_i+1$

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

- 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

$$d_1=\Delta x[2m(x_1+1)+2b-2y_1-1] =\Delta x[2(mx_1+b-y_1)+2m-1]$$

- Since $mx_1+b-y_i=0$ and m = , we have $d_1=2\Delta y-\Delta x$

Bresenham's Line Drawing Algorithm

Given-

Starting coordinates = (X0, Y0)

Ending coordinates = (Xn, Yn)

Step-01: Calculate ΔX, ΔY from the given input. These parameters are calculated as-

$$\Delta X = Xn - X0$$

$$\Delta Y = Yn - Y0$$

Step-02: Calculate the decision parameter Pk.

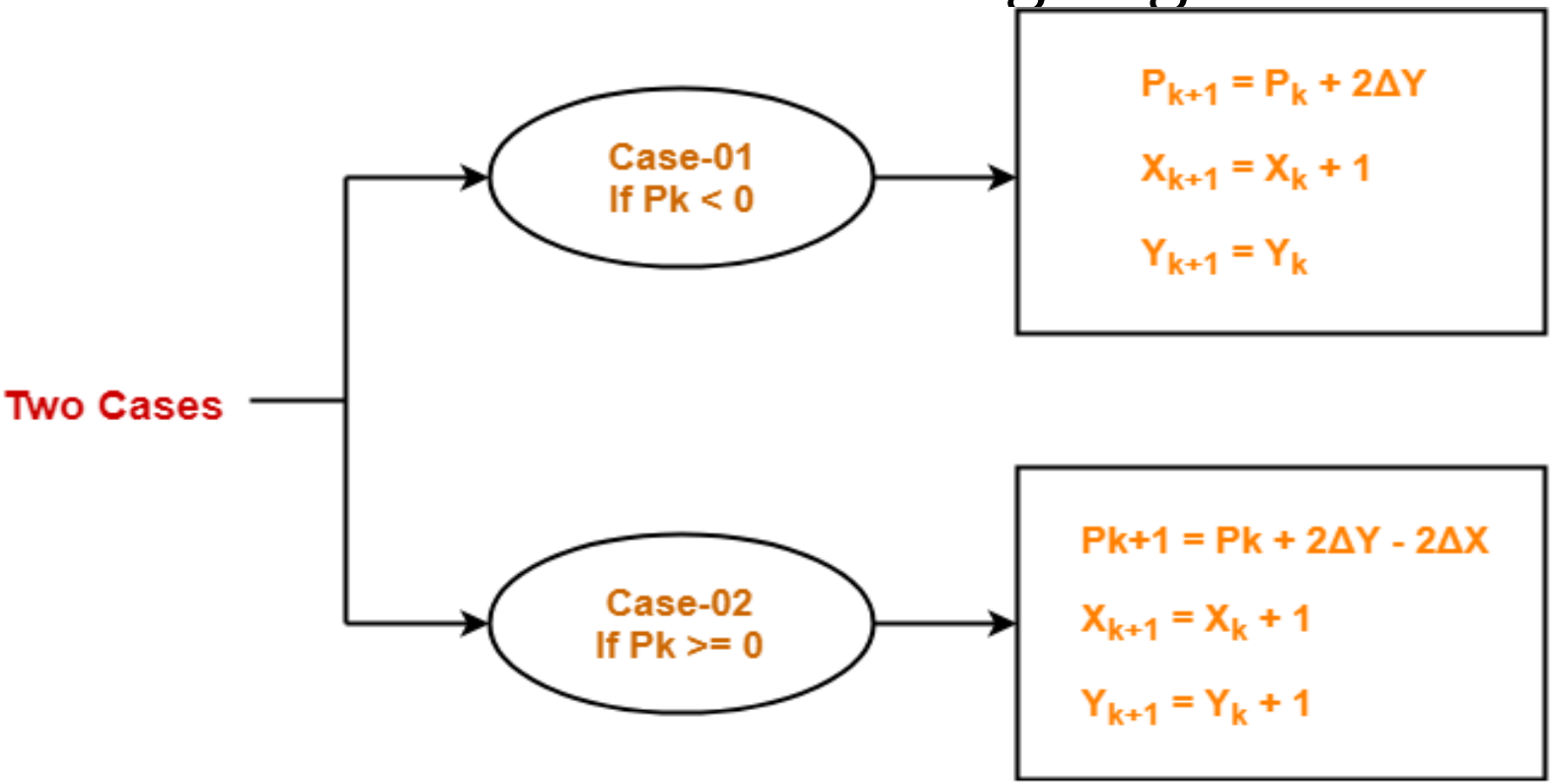
$$Pk = 2\Delta Y - \Delta X$$

Step-03: Suppose the current point is (Xk, Yk) and the next point is (Xk+1, Yk+1).

Find the next point depending on the value of decision parameter Pk.

Follow the below two cases-

Bresenham's Line Drawing Algorithm



Step-04: Keep repeating Step-03 until the end point is reached or number of iterations equals to (ΔX-1) times.

Bresenham's Line Drawing Algorithm

Advantage

- It involves only integer arithmetic, so it is simple.
- It avoids the generation of duplicate points.
- It can be implemented using hardware because it does not use multiplication and division.
- 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.

Bresenham's Line Drawing Algorithm

Problem 01:

Calculate the points between the starting coordinates (9, 18) and ending coordinates (14, 22).

Solution:

Given-

Starting coordinates = (X0, Y0) = (9, 18)

Ending coordinates = (Xn, Yn) = (14, 22)

Step-01: Calculate ΔX, ΔY from the given input.

$$\Delta X = Xn - X0 = 14 - 9 = 5$$

$$\Delta Y = Yn - Y0 = 22 - 18 = 4$$

Step-02: Calculate the decision parameter Pk.

$$Pk = 2\Delta Y - \Delta X = 2 \times 4 - 5 = 3$$

Bresenham's Line Drawing Algorithm

Step-03: As Pk >= 0, so case-02 is satisfied.

Thus,

$$P_{k+1} = P_k + 2\Delta Y - 2\Delta X = 3 + (2 \times 4) - (2 \times 5) = 1$$

$$X_{k+1} = X_k + 1 = 9 + 1 = 10$$

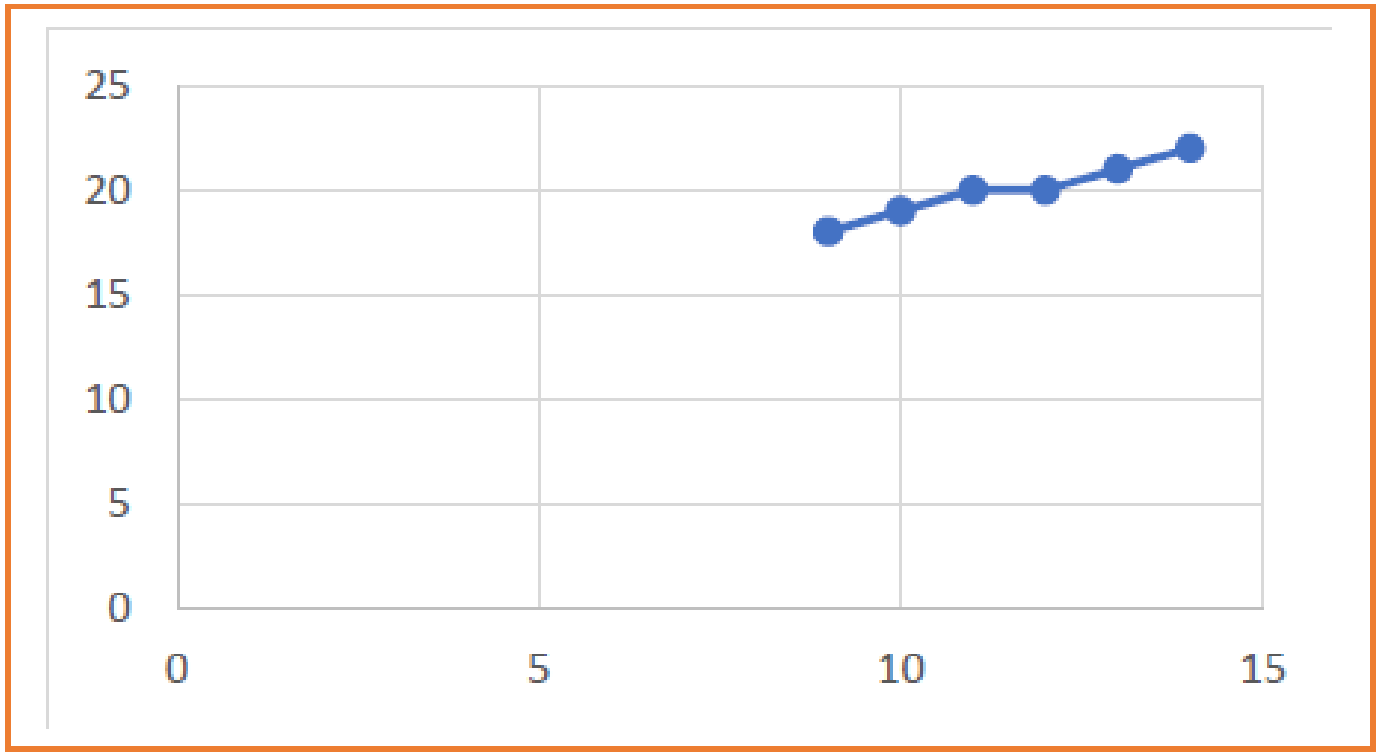
$$Y_{k+1} = Y_k + 1 = 18 + 1 = 19$$

Similarly, Step-03 is executed until the end point is reached or number of iterations equals to 4 times.

(Number of iterations = $\Delta X - 1 = 5 - 1 = 4$)

P_k	P_{k+1}	X_{k+1}	Y_{k+1}
		9	18
3	1	10	19
1	-1	11	20
-1	7	12	20
7	5	13	21
5	3	14	22

Bresenham's Line Drawing Algorithm



Bresenham's Line Drawing Algorithm

Problem 02:

Calculate the points between the starting coordinates (20, 10) and ending coordinates (30, 18).

Solution:

Given-

Starting coordinates = (X0, Y0) = (20, 10)

Ending coordinates = (Xn, Yn) = (30, 18)

Step-01: Calculate ΔX, ΔY from the given input.

$$\Delta X = Xn - X0 = 30 - 20 = 10$$

$$\Delta Y = Yn - Y0 = 18 - 10 = 8$$

Step-02: Calculate the decision parameter Pk.

$$Pk = 2\Delta Y - \Delta X = 2 \times 8 - 10 = 6$$

Bresenham's Line Drawing Algorithm

Step-03: As Pk >= 0, so case-02 is satisfied.

Thus,

$$P_{k+1} = P_k + 2\Delta Y - 2\Delta X = 6 + (2 \times 8) - (2 \times 10) = 2$$

$$X_{k+1} = X_k + 1 = 20 + 1 = 21$$

$$Y_{k+1} = Y_k + 1 = 10 + 1 = 11$$

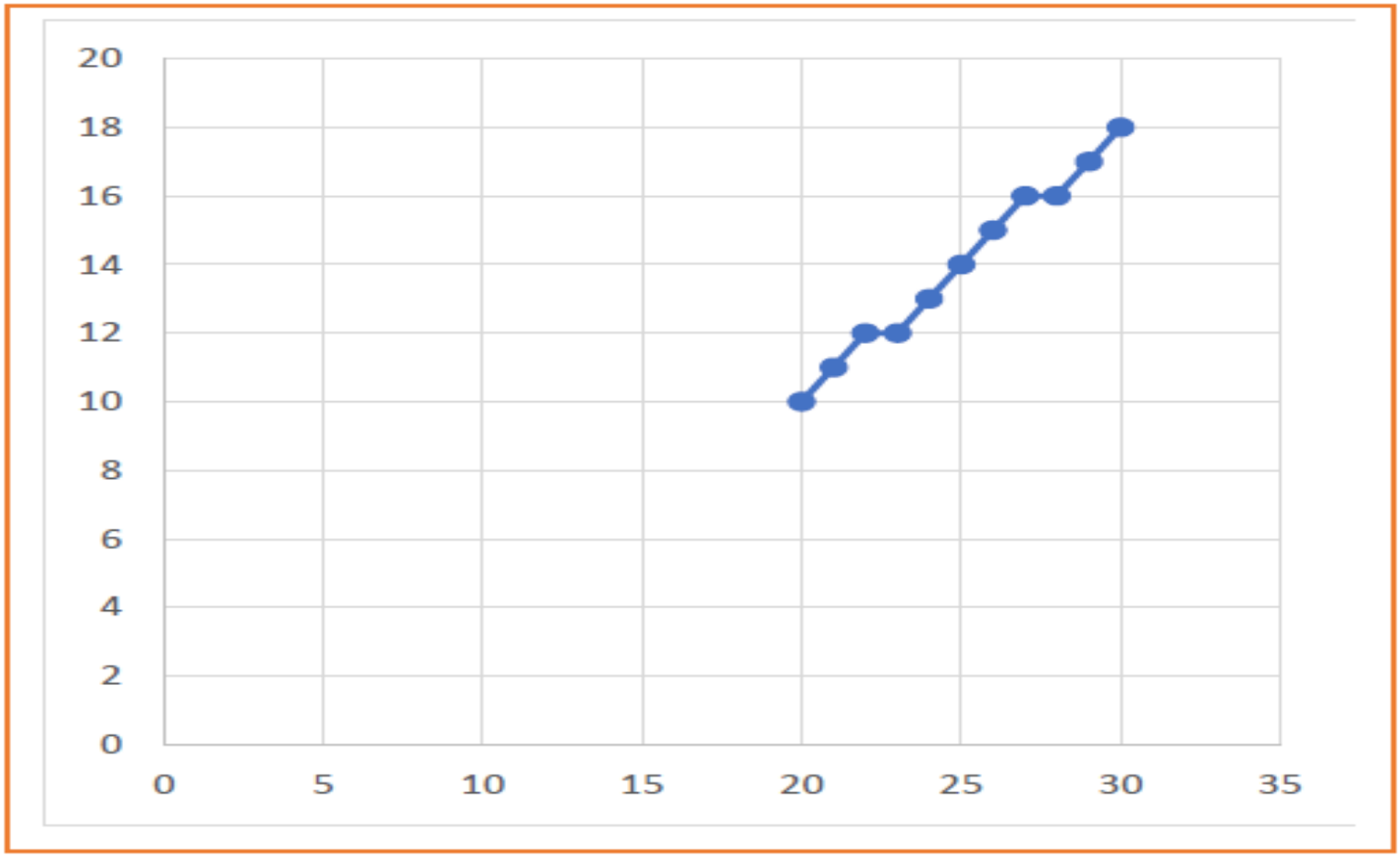
Similarly, Step-03 is executed until the end point is reached or number of iterations equals to 9 times.

(Number of iterations = $\Delta X - 1 = 10 - 1 = 9$)

Bresenham's Line Drawing Algorithm

P_k	P_{k+1}	X_{k+1}	Y_{k+1}
		20	10
6	2	21	11
2	-2	22	12
-2	14	23	12
14	10	24	13
10	6	25	14
6	2	26	15
2	-2	27	16
-2	14	28	16
14	10	29	17
10	6	30	18

Bresenham's Line Drawing Algorithm



DDA vs Bresenham's Algorithm

Sr. No	DDA	Bresenham
1	Comparatively less efficient	Comparatively more efficient
2	Calculation speed is lesser	Calculation speed is faster
3	It is costlier	It is cheaper
4	It has less precision or accuracy.	While it has more precision or accuracy.

Questions

- What do you mean by scan conversion?
- What is the computational complexity of the basic DDA line drawing algorithm? What about Bresenham's line drawing algorithm? What makes Bresenham's algorithm more efficient than the basic DDA?
- Explain DDA line drawing algorithm with its drawbacks.
- Explain Bresanham's line drawing algorithms along with its advantages and disadvantages.
- Rasterize the line from (10,5) to (15,9) using Bresenhams line drawing and DDA Algorithms.
- Compare DDA and Bbresanham's line drawing algorithms.


Thank You