What Does Pixel Mean?

A pixel is the smallest unit of a digital image or graphic that can be displayed and represented on a digital display device.

A pixel is the basic logical unit in digital graphics. Pixels are combined to form a complete image, video, text, or any visible thing on a computer display.

A pixel is also known as a picture element (pix = picture, el = element).

Frame buffer -

A frame buffer is a large, contiguous piece of computer memory. At a minimum there is one memory bit for each pixel in the rater; this amount of memory is called a bit plane. The picture is built up in the frame buffer one bit at a time.

Resolution in Computer Graphics:

The number of horizontal and vertical pixels on a display screen is called Resolution.

Image Resolution: It refers to the pixel spacing the distance from one pixel to the next pixel. In other words, the resolution of an image is the total number of pixels along with the entire height and width of the image.

Example: A full-screen image with resolution 800×600 dpi means that there are 800 columns of dot pixels per inch and each column comprising 600 dot pixels per inch.

A total of 800×600 = 48000 dot pixels in sq. inches image area.

Screen Resolution: Screen resolution is the number of pixels a screen, both horizontally and vertically. So, a screen that has a resolution of 3840 x 2160 (It is also known as 4k UHD), it can display 2160 pixels vertically, and 3840 pixels horizontally.

Aspect Ratio:
The **Aspect Ratio** is the ratio of the number of X pixels to the number of Y pixels. The standard aspect ratio for PCs is 4:3. Some common resolutions, the respective number of pixels and standard aspect ratio are given below:

Resolution	Number of Pixels	Aspect Ratio
320x240	76800	4:3
640x480	307200	4:3
800x600	480000	4:3
1024x768	786432	4:3
1280x720	921600	16:9
1920x1080	2073600	16:9

Line Drawing Algorithms

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

$y_i = mx_i + b$.....equation 1

Next value will be

$y_{i+1} = mx_{i+1} + b$.....equation 2

m =DDA Algorithm

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

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

$y_{i+1} = y_i + \Delta y$

$\Delta y = m\Delta x$

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

$\Delta x = \Delta y / m$

$x_{i+1} = x_i + \Delta x$

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

Case1: When $|M|<1$ then (assume that $x_1 2$)

$x= x_1, y=y_1$ set $\Delta x=1$

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

Until $x = x_2$

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

$x= x_1, y=y_1$ set $\Delta y=1$

$x_{i+1}=\frac{1}{m}$, $y=y+1$

Until $y \rightarrow y_2$

- 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.

- DDA Algorithm:
- **Step1:** Start Algorithm
- **Step2:** Declare x_1, y_1, x_2, y_2, dx, dy, x, y as integer variables.
- **Step3:** Enter value of x_1, y_1, x_2, y_2.
- **Step4:** Calculate dx = $x_2 - x_1$
- **Step5:** Calculate dy = $y_2 - y_1$
- **Step6:** **If ABS (dx) > ABS (dy)**
  Then step = abs (dx)
  Else step = abs (dy)

Step7: xinc=dx/step
yinc=dy/step
assign x = x1
assign y = y1

Step8: Set pixel (x, y)

Step9: x = x + xinc
y = y + yinc
Set pixels (Round (x), Round (y))

Step10: Repeat step 9 until x = x2

Step11: End Algorithm

Problem-01:

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

Solution-

Given-

- Starting coordinates = $(X_0, Y_0) = (5, 6)$
- Ending coordinates = $(X_n, Y_n) = (8, 12)$

Step-01:

Calculate ΔX, ΔY and M from the given input.

- $\Delta X = X_n - X_0 = 8 - 5 = 3$
- $\Delta Y = Y_n - Y_0 = 12 - 6 = 6$
- $M = \Delta Y / \Delta X = 6 / 3 = 2$

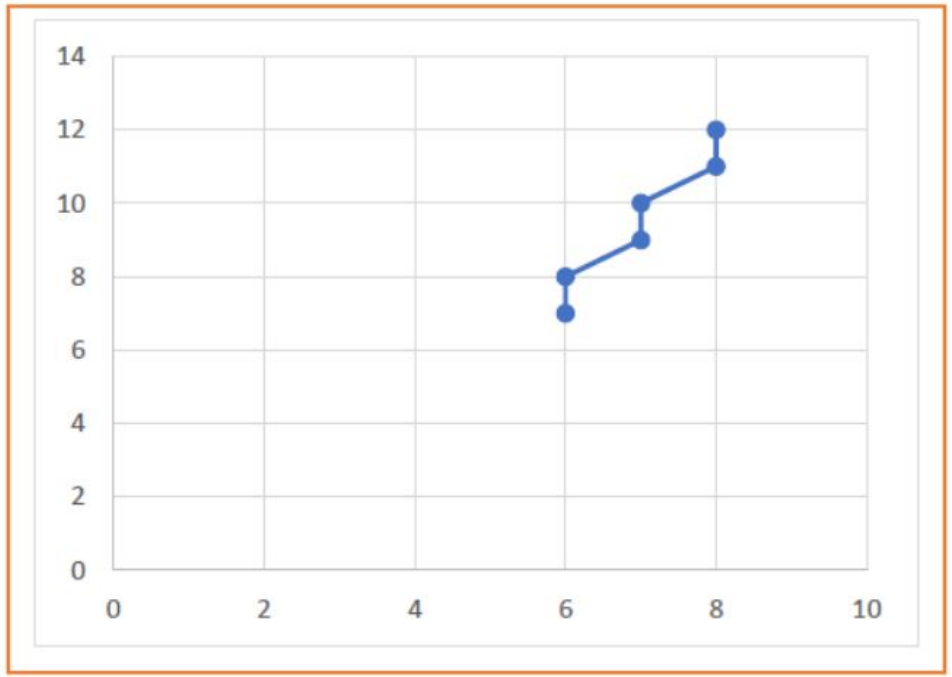
Step-02:

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.

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)



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.

Count.

- he 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'.

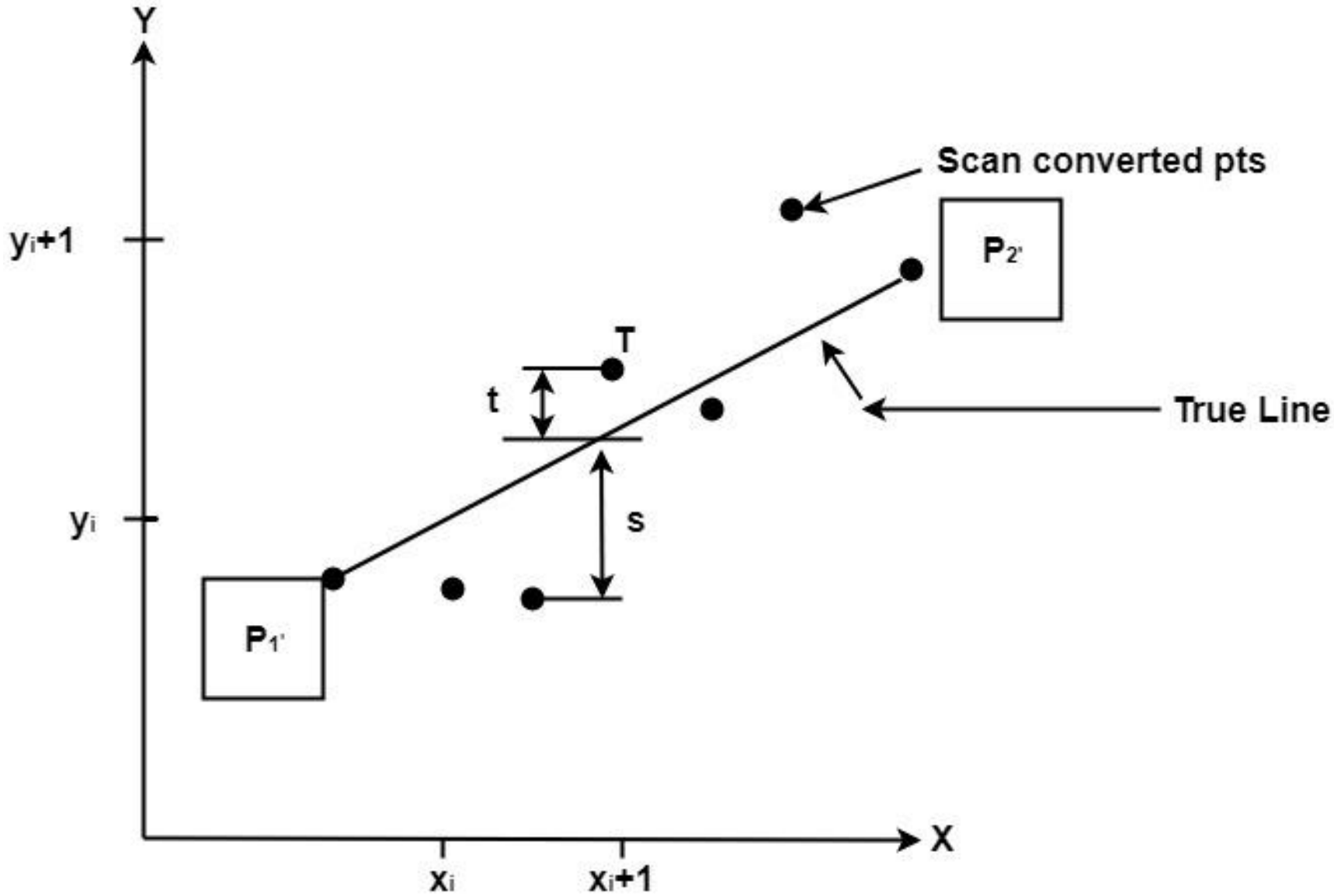


Fig: Scan Converting a line.

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

If S is chosen

We have xi+1=xi+1 and yi+1=yi

If T is chosen

We have xi+1=xi+1 and yi+1=yi+1

The actual y coordinates of the line at x = xi+1is

y=mxi+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 $(s-t) \geq 0 \Longrightarrow s < t$

The closest pixel is T

This difference is

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

$= 2y - 2y_i - 1$

$$s - t = 2m(x_i + 1) + 2b - 2y_i - 1$$

[Putting the value of (1)]

Substituting m by $\frac{\Delta y}{\Delta x}$ and introducing decision variable

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

$$d_i = \Delta x\left(2\frac{\Delta y}{\Delta x}(x_i+1)+2b-2y_i-1\right)$$

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

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

Where $c = 2\Delta y + \Delta x\,(2b-1)$

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

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

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

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 x1,x2,y1,y2,d,i1,i2,dx,dy

Step3: Enter value of x1,y1,x2,y2

Where x1,y1are coordinates of starting point

And x2,y2 are coordinates of Ending point

Step4: Calculate dx = x2-x1

Calculate dy = y2-y1

Calculate i1=2*dy

Calculate i2=2*(dy-dx)

Calculate d=i1-dx

Step5: Consider (x, y) as starting point and xendas maximum possible value of x.

If dx < 0

Then x = x2

y = y2

xend=x1

If dx > 0

Then x = x1

y = y1

xend=x2

Step6: Generate point at (x,y)coordinates.

Step7: Check if whole line is generated.

If x > = xend

Stop.

Step8: Calculate co-ordinates of the next pixel

If d < 0

Then d = d + i1

If d ≥ 0

Then d = d + i2

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

Problem:

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 and ΔY from the given input.

ΔX = Xn – X0 = 14 – 9 = 5

ΔY =Yn – Y0 = 22 – 18 = 4

Step-02:

Calculate the decision parameter. Pk

= 2ΔY – ΔX

= 2 x 4 – 5

= 3

So, decision parameter Pk = 3

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