Unit-2

Scan Conversion Algorithm

The process in which the object is represented as the collection of discrete pixels is called scan conversion. It includes the concepts that are required for the understanding of two dimensional graphics. The graphics ~~used in~~ objects used in scan conversion are in continious but the pixels used are in discrete. Each pixel can have either on or off state.

Point, line, sector, arc, rectangle, ellipse, characters etc. are the examples of objects which can be scan converted. Different algorithms are used ~~to con~~ for scan conversion of these types of objects which are called scan conversion algorithms.

Advantage of developing algorithms for scan conversion:

- Algorithms can generate graphics objects at a faster rate.
- Using algorithms memory can be used efficiently.
- Algorithms can develop a higher level of graphical objects.

Line Drawing Algorithm:

A line drawing algorithm is a graphical algorithm for approximating a line segment on discrete graphical media. There are mainly three most widely used line drawing algorithms as follows:

i) Direct use of line equation.
ii) Digital Differential Analyzer Algorithm (DDA)
iii) Bresenham line Drawing Algorithm (BSA)

Here we study about only two DDA algorithm and BSA algorithm.

1.# Digital Differential Analyzer Algorithm (DDA):

In any 2-Dimensional plane if we connect any two points (x_0, y_0) and (x_1, y_1) we get a line segment. But in the case of computer graphics we can not directly join any two coordinate points, for that we should calculate intermediate points coordinate using a basic algorithm called DDA.

Working details of DDA Algorithm:

Step1:- Get the input of two end points (x_0, y_0) and (x_1, y_1).

Step2:- Calculate the difference between two end points as:

$$dx = x_1 - x_0$$
$$dy = y_1 - y_0$$

Step3:- Based on the calculated difference in step-2, now we need to identify the number of steps to put pixel as follows:-

also we can do with $m < 1$ where $m = \frac{dy}{dx}$

If $dx > dy$, then we need more steps in x co-ordinate; otherwise in y coordinate.

i.e, if (absolute(dx) > absolute(dy)), then steps = absolute (dx) else steps = absolute (dy).

Step4:- We calculate the increament in x coordinate and y coordinate as follows:-

$$x_{increament} = \frac{dx}{steps\ (float)}$$

$$y_{increament} = \frac{dy}{steps\ (float)}$$

Step5:- Finally we put the pixel by successfully incrementing x and y co-ordinates accordingly and complete the drawing of line.

```
for (int i=0; i<steps; i++){
    x = x + x_increament
    y = y + y_increament
    putpixel (Round(x), Round(y))
}
```

Example 1 (For $m<1$ OR $dx>dy$)

Q. Using DDA plot (5, 4) to (12, 7).

Sol^n

Given:-

$(x_0, y_0) = (5, 4)$

$(x_1, y_1) = (12, 7)$

$dx = x_1 - x_0 = 12 - 5 = 7$

$dy = y_1 - y_0 = 7 - 4 = 3$

Since $dx > dy$ OR, Slope $m = \frac{dy}{dx} = \frac{3}{7}$ i.e. <1

So, no. of steps = absolute (dx).

$= |7|$

$= 7$

Now,

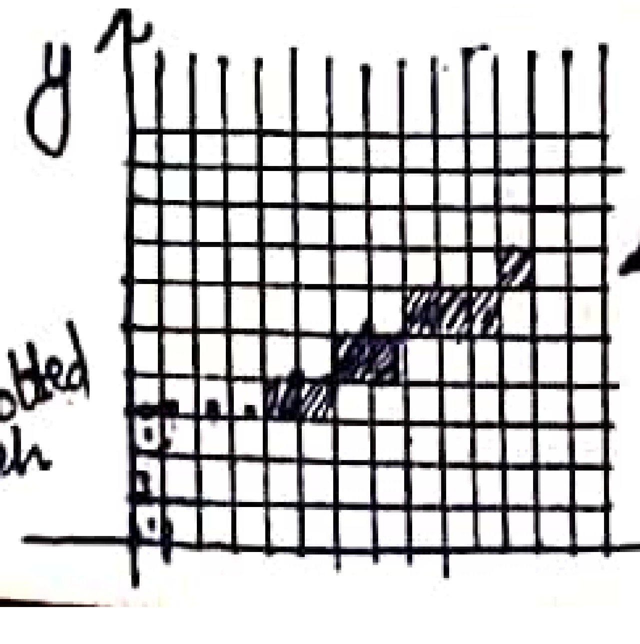
$x_{increment} = \frac{dx}{Steps} = \frac{7}{7} = 1$

$y_{increment} = \frac{dy}{Steps} = \frac{3}{7} = 0.4$

Steps	x	y	y (Rounded off)
0	5	4	4
1	6	4.4	4
2	7	4.8	5
3	8	5.2	5
4	9	5.6	6
5	10	6	6
6	11	6.4	6
7	12	6.8	7

Note:- If x or y comes in fraction we write as it is but if on decimal we round off them.

x & y are plotted on graph



In case we get rounded off we get this type of line not exactly straight line.

Example 2 (for $m>1$ OR $dx<dy$)

Q. Using DDA plot line (5, 7) to (10, 15).

Sol^n

Given, $(x_0, y_0) = (5, 7)$

$(x_1, y_1) = (10, 15)$.

$dx = x_1 - x_0 = 10 - 5 = 5$

$dy = y_1 - y_0 = 15 - 7 = 8$

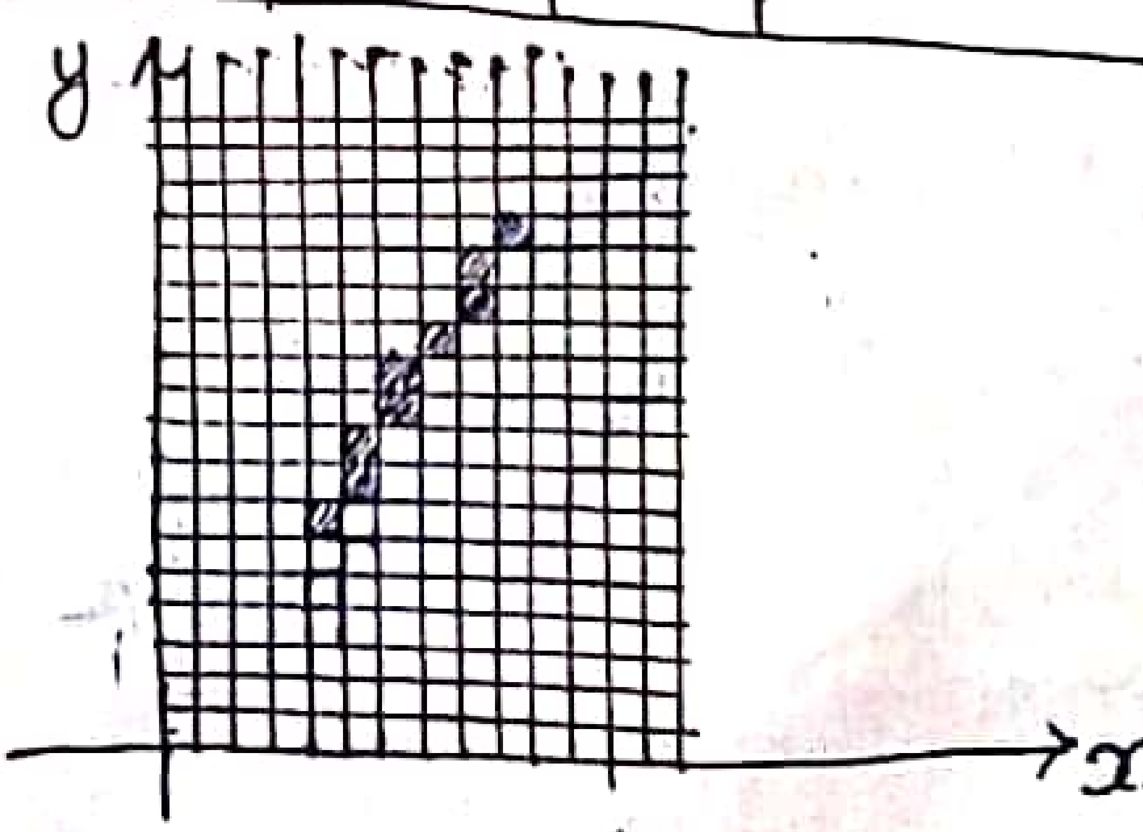
Now, $m = \frac{8}{5}$ i.e. >1

$\therefore$ Steps $= |8| = 8$.

$x_{increament} = \frac{dx}{steps} = \frac{5}{8} = 0.6$

$y_{increament} = \frac{dy}{steps} = \frac{8}{8} = 1$

steps	x	y	x (Rounded off)
0	5	7	5
1	5.6	8	6
2	6.2	9	6
3	6.8	10	7
4	7.4	11	7
5	8	12	8
6	8.6	13	9
7	9.2	14	9
8	9.8	15	10



Note:- If $dx = dy$ i.e., $m = 1$ then, steps $= |dx|$ or we can take $|dy|$ & We proceed the steps as we did for $m<1$ and $m>1$.

2. Bresenham Line Drawing Algorithm (BSA):

There are two problems arised when using DDA algorithm:

i) Lines may not be smooth since we take round off values if floating points are seen in co-ordinates.

ii) It takes floating points which takes extra time for calculation which makes DDA algorithm slower.

For to solve these two problems new algorithm was introduced called Bresenham line drawing algorithm.

Working details of BSA Algorithm (For positive slope) $\boxed{\text{Slope} = \frac{y_1 - y_0}{x_1 - x_0}}$

Step 1:- Get the input of two endpoints (x_0, y_0) and (x_1, y_1).

Step 2:- Calculate the difference between two endpoints as:

$\Delta x = \text{absolute}(x_1 - x_0)$

$\Delta y = \text{absolute}(y_1 - y_0)$

Step 3:- Calculate initial decision parameter as:

$P_k = 2\Delta y - \Delta x$.

Step 4:- if $x_1 > x_0$ then,

Set $x = x_0$;

Set $y = y_0$;

Set $x_{end} = x_1$;

Otherwise,

Set $x = x_1$;

Set $y = y_1$;

Set $x_{end} = x_0$;

Step 5:- Draw pixel at (x, y).

Step 6:- While $(x < x_{end})$ {

x++;

if $P_k < 0$ then

$P_{k+1} = P_k + 2\Delta y$

otherwise (i.e., for $P_k \geq 0$)

y++;

$P_{k+1} = P_k + 2\Delta y - 2\Delta x$

}

Draw pixel at (x, y).

Scanned with CamScanner

Example:

(*). Plot the line from the point (10, 15) to (15, 18) using BSA.

Soln

Given, $(x_0, y_0) = (10, 15)$

$(x_1, y_1) = (15, 18)$

Here, we calculate Δy and Δx as:

$$\Delta x = |x_1 - x_0| = |15 - 10| = 5$$

$$\Delta y = |y_1 - y_0| = |18 - 15| = 3$$

Now we calculate initial decision parameter as:

$$P_k = 2\Delta y - \Delta x = 2 \times 3 - 5 = 1$$

& since $x_1 > x_0$

So, $x = 10$; $y = 15$; $x\,end = 15$;

We know that $P_k \geq 0$ so, using condition,

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x = 1 + 2 \times 3 - 2 \times 5 = -3$$

Continuing this process up to ($x = x\,end$) on table as follows:-

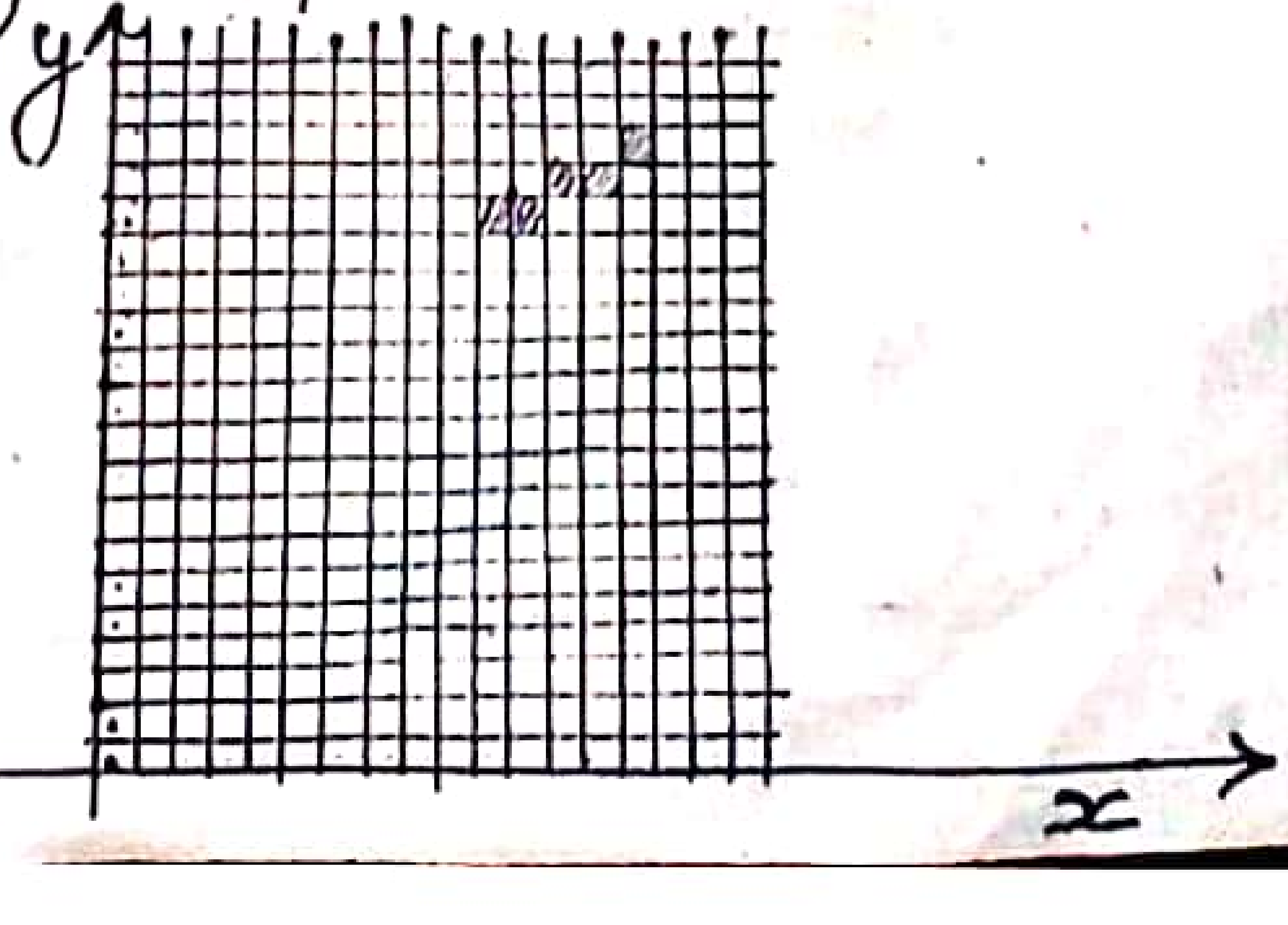
since $P_k > 0$
$P_{k+1} = P_k + 2\Delta y - 2\Delta y$
for $P_k < 0$
$P_{k+1} = P + 2\Delta y$

k	x	y	P_{k+1}	Pixel Points
0	10	15	1	(11, 16)
1	11	16	-3	(12, 16)
2	12	16	3	(13, 17)
3.	13	17	-1	(14, 17)
4	14	17	5	(15, 18)

We can calculate step by step showing algorithm or using table we can use any one method

→ x always increases
y increases only when P_{k+1} is greater or equal to zero.

Now we see finally plotting these points:-



Working details of BSA Algorithm (for negative slope questions):

Step1: Get the input of two end points (x_0, y_0) and (x_1, y_1).

Step2: Calculate the difference between two endpoints as:

$$\Delta x = absolute\,(x_1 - x_0).$$
$$\Delta y = absolute\,(y_1 - y_0):$$

Step3: Calculate initial decision parameter.

$$P_k = 2\Delta y - \Delta x.$$

Step4: for ($i=1$ to Δx){

Set pixel (x_s, y_s)

while ($P_k > 0$)

$y_0 = y_0 - 1$;

$P_k = P_k - 2\Delta x$.

end while, ($P_k > 0$)

$x_0 = x_0 + 1$

$P_k = P_k + 2\Delta y$

i++;

}

Set pixel (x_0, y_0).

Imp **Example:** Plot a line from (6,12) to (10,5) using BSA algorithm.

Sol^n.

$(x_0, y_0) = (6, 12)$

$(x_1, y_1) = (10, 5)$

Now we calculate Δx and Δy as follows:-

$\Delta x = absolute\,(10-6)$

$= 4$

$\Delta y = absolute\,(5-12)$

$= 7$

Now, initial decision parameter is,

$P_k = 2\Delta y - \Delta x$

$= 2 \times 7 - 4$

$= 10.$

for ($i=1$ to 4). using this condition we proceed as follows:-

For $i=1$ Set pixel $(6,12)$ while $(10>0)$

$$y_0 = 12-1$$
$$= 11$$
$$P_k = 10 - 2 \times 4$$
$$= 2$$

while $(2>0)$

$$y_0 = 11-1$$
$$= 10$$
$$P_k = 2 - 2 \times 4$$
$$= -6$$

we repeat while upto >0. - Since <0 so we go to end while

end while

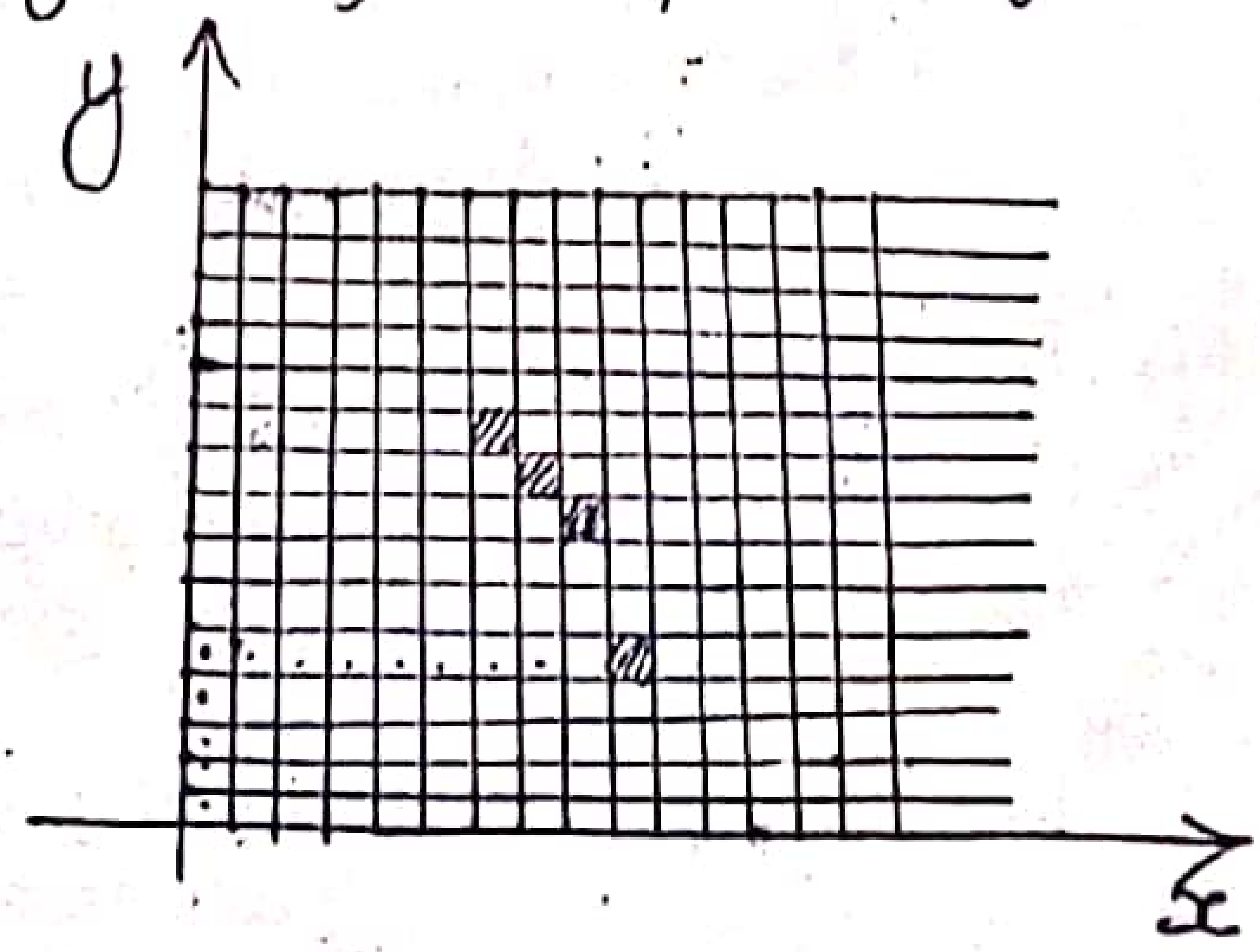
$$x_0 = x_0 + 1$$
$$= 6+1$$
$$= 7$$
$$P_k = P_k + 2\Delta y$$
$$= -6 + 2 \times 7$$
$$= 8$$

We can use similarly for $i = 2,3$ & 4 without making table step by step also

Now, continuing this process in table:-

i	x_0	y_0	P_K	Points
1	6	12	10	(7,10)
2	7	10	8	(8,9)
3	8	9	14	(9,7)
4	9	7	12	(10,5)

Now, finally plotting these points on graph



⊛ Differences between DDA line drawing algorithm and Bresenham line drawing algorithm:

Basis for difference	DDA line drawing algorithm	Bresenham line drawing algorithm.
Arithmetic	DDA algorithm uses floating points. i.e, Real Arithmetic.	Bresenham algorithm uses fixed points. i.e Integer Arithmetic.
Operations	DDA algorithm uses multiplication and division in its operations.	Bresenham algorithm uses only subtraction and addition in its operations.
Speed.	It is slower than Bresenham algorithm because it uses floating points.	It is faster than DDA algorithm because it uses only integer arithmetic.
Accuracy & Efficiency	DDA algorithm is not as accurate and efficient as Bresenham algorithm.	Bresenham algorithm is more efficient and much accurate than DDA algorithm.
Drawing	DDA algorithm can draw circles and curves but not accurate as Bresenham algorithm.	Bresenham algorithm can draw circles and curves with much more accuracy than DDA algorithm.
Round off	DDA algorithm round off the coordinates to integer that is nearest to the line.	Bresenham algorithm does not round off but takes the incremental value in its operation.
Expensive.	DDA algorithm uses an enermous number of floating-point multiplications so it is expensive.	Bresenham algorithm is less expensive than DDA algorithm as it uses only addition and subtraction.

3. Mid Point Circle Algorithm:

Algorithm:

Step1: Start

Step2: Input radius r and circle centre (h,k).

Step3: Initialize $x=0, y=r$ and $P_0 = 1-r$

Step4: While $(x \leq y)$

//plot 8 points as

Plot pixel $(x+h, y+k)$;
Plot pixel $(-x+h, y+k)$;
Plot pixel $(x+h, -y+k)$;
Plot pixel $(-x+h, -y+k)$;
Plot pixel $(y+h, x+k)$;
Plot pixel $(-y+h, x+k)$;
Plot pixel $(y+h, -x+k)$;
Plot pixel $(-y+h, -x+k)$;

x & y are co-ordinates
r → radius
P_0 → decision parameter.

Step5: If $P_0 < 0$ then,

Set, $P_0 = (P + 2x + 1)$
Set, $x = x+1$;

Otherwise,

Set, $P_0 = P_0 + 2x + 1 - 2y$
Set, $y = y-1$; and set, x=x+1;

Step6: End while loop.

Step7: Stop.

Example 1: Calculate the points to draw a circle having radius 10 and centre at (0,0).

Solution:

we have $r=10$.
centre $(h,k) = (0,0)$
Set $x=0, y=r=10$

Now, first pixel to be drawn $= (0,10)$

Initial decision parameter $(P_0) = 1-r$
$= 1-10$
$= -9$

Now, points on the octant are given by,

K	P_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
1	-9	(1,10)	2	20
2	-6	(2,10)	4	20
3	-1	(3,10)	6	20
4	6	(4,9)	8	18
5	-3	(5,9)	10	18
6	8	(6,8)	12	16
7	5	(7,7)	14	14

Now, using 8 point symmetry we get all points on the circle as below:-

(x,y)	(0,10)	(1,10)	(2,10)	(3,10)	(4,9)	(5,9)	(6,8)	(7,7)
(-x,y)	(0,10)	(-1,10)	(-2,10)	(-3,10)	(-4,9)	(-5,9)	(-6,8)	(-7,7)
(x,-y)	(0,-10)	(1,-10)	(2,-10)	(3,-10)	(4,-9)	(5,-9)	(6,-8)	(7,-7)
(-x,-y)	(0,-10)	(-1,-10)	(-2,-10)	(-3,-10)	(-4,-9)	(-5,-9)	(-6,-8)	(-7,-7)
(y,x)	(10,0)	(10,1)	(10,2)	(10,3)	(9,4)	(9,5)	(8,6)	(7,7)
(-y,x)	(-10,0)	(-10,1)	(-10,2)	(-10,3)	(-9,4)	(-9,5)	(-8,6)	(-7,7)
(y,-x)	(10,0)	(10,-1)	(10,-2)	(10,-3)	(9,-4)	(9,-5)	(8,-6)	(7,-7)
(-y,-x)	(-10,0)	(-10,-1)	(-10,-2)	(-10,-3)	(-9,-4)	(-9,-5)	(-8,-6)	(-7,-7)

Now plotting these points on graph we get circle.

<u>Example2</u>:- Digitize a circle $(x-2)^2+(y-3)^2=25$

<u>Solution</u>;

Given, $(x-2)^2+(y-3)^2=25$

Comparing with general equation of circle $(x-h)^2+(y-k)^2=r^2$

we get, centre $(h,k)=(2,3)$

radius $r=5$

First pixel $= (0,5)$ ($\because$ Set $x=0, y=r$).

Initial decision parameter $P_0 = 1-r$

$= 1-5$

$= -4$

Now. points on the octant are given by,

k	P_k	(x_{k+1}, y_{k+1})	$(2x_{k+1}$	$2y_{k+1}$	Actual pixel
1	-4	(1,5)	2	10	(3,8)
2	-1	(2,5)	4	10	(4,8)
3	4	(3,4)	6	8	(5,7)
4	3	(4,3)	8	6	(6,6)
5	6	(5,2)	10	4	(7,5)
6	13	(6,1)	12	2	(8,4)
7	24	(7,0)	14	0	(9,3)

Centre (0,0) भएको बेला यो पेरला Centre (0,0) बाहेक भएर यो गर्ने
We have centre (2,3) in this question so we add (2,3) to each pixel to get actual pixel.

Now using 8 point symmetry we get all points on the circle as we did in example 1 & Plotting these points on graph we get required circle.

Midpoint - Ellipse Algorithm

Algorithm:

Midpoint - Ellipse (a,b)

① $x=0$, $y=b$

② $d_1 = b^2 + \frac{a^2}{4} - a^2 b$

For Region 1

③ while $(a^2 y \geq b^2 x)$

Plot (x,y)

if $(d_1 \leq 0)$

$d_1 = d_1 + b^2(2x+3)$

$x = x+1$

else

$d_1 = d_1 + b^2(2x+3) + a^2(2-2y)$

$x = x+1$

$y = y-1$

For Region 2

④ $d_2 = (x+0.5)^2 b^2 + (y-1)^2 a^2 - a^2 b^2$

⑤ while $(y \geq 0)$

Plot (x,y)

if $(d_2 \leq 0)$

$d_2 = d_2 + b^2(2x+2) + a^2(3-2y)$

$x = x+1$

$y = y-1$

else

$dx = d_2 + (3-2y)a^2$

$y = y-1$.

Example:- Input ellipse parameters $r_x = 8$ and $r_y = 6$ the mid-point ellipse algorithm by determining raster position along the ellipse path in the first quadrant. Initial values and increments for the decision parameter calculates are:-

$$2r_y^2 x = 0 \quad \text{(with increment } 2r_y^2 = 72)$$

$$2r_x^2 y = 2r_x^2 r_y \quad \text{(with increment } -2r_x^2 = -128).$$

Solution:

Mid-point Ellipse (8, 6) (a, b)

$x = 0,\ y = 6$

For Region 1

Initial point is (0, 6)

according to algorithm formula

• & Initial decision parameter $(d_1) = 6^2 + \frac{8^2}{4} - 8^2 \times 6$

$= 36 + 16 - 384$

$= -332$

Now, the successive mid-point decision parameter values and the pixel positions along the ellipse are listed in following table:-

Since $b = 6$ & $x = 0$
$\therefore b^2 x = 6^2 \times 0$
$= 0$

Since $a = 8$
& $y = 6$
$\therefore a^2 y = 8^2 \times 6$
$= 384$

d_1	(x, y)	$b^2 x$	$a^2 y$
−332	(0, 6)	0	384
−224	(1, 6)	36	384
−44	(2, 6)	72	384
208	(3, 6)	108	384
−108	(4, 5)	144	320
288	(5, 5)	180	320
116	(6, 4)	216	256
272	(7, 3)	252	192

Since $d_1 < 0$ so according to algorithm if part runs that is in step 3

Since $d_1 \geq 0$ so according to algorithm else part that runs is in step 3

Since $(a^2 y \geq b^2 x)$ condition is false here so we stop here. We run until condition becomes true

For Region 2

Initial point is (8,2) from algorithm $x = x+1$, $y = y-1$ i.e. we have final point (7,3) $x = 7+1 = 8$, $y = 3-1 = 2$

& Initial decision parameter $(d_2) = (7+0.5)^2 \cdot 6 + (3-1)^2 \cdot 8^2 - 36 \times 64$

$= 2025 + 256 - 2304$

$= -23$

Now, the remaning positions along the ellipse path are as follows:-

d_2	(x,y)	b^2x	a^2y
-23	(8,2)	288	128
561	(9,1)	324	64
626	(9,0)	324	0

Since $y \geq 0$ condition meets so we stop here

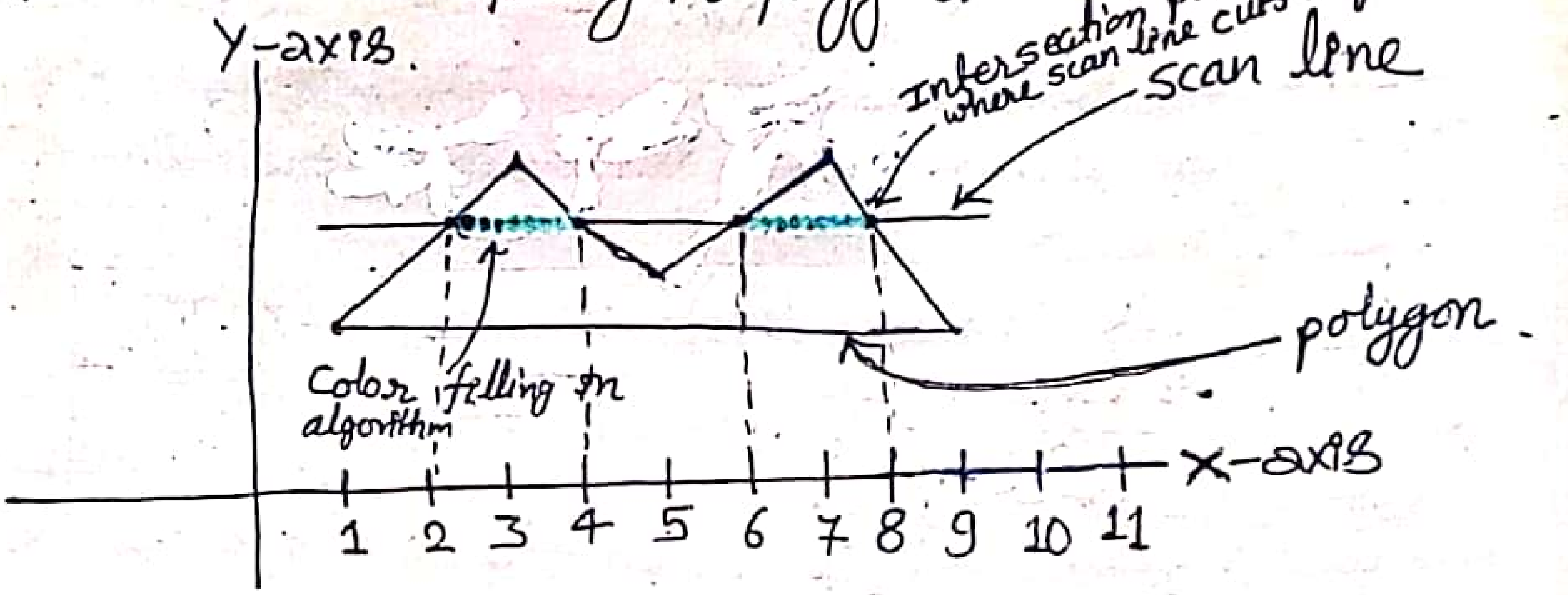
~~Now,~~ Finally plotting these points ~~on~~ [i.e.,(x,y)] on graph we get required ellipse.

Scanned with CamScanner

⊛ Area Filling:

1) Scan line Polygon fill Algorithm:

Basic concept → Scan line polygon filling algorithm is used for solid color filling in polygons.



Algorithm Steps:

1. The horizontal scanning of the polygon from its lowermost to topmost vertex is done.
2. Find all the intersections of the scan line with all edges of the polygon, from left to right.
   eg. In the above figure point of intersections are 2, 4, 6 and 8.
3. Sort the intersections by increasing x-coordinate i.e, from left to right.
4. Make pairs of the intersections and fill in colour within all the pixels inside the pair.
   e.g. (2,4) and (6,8) between these pixels colour is filled.

Special Cases:

Some scan-line intersections at polygon vertices require special handling.

i) A scan line passing through a vertex intersects two polygon edges at that position, adding two points to the list of intersections for the scan line.

For Example:

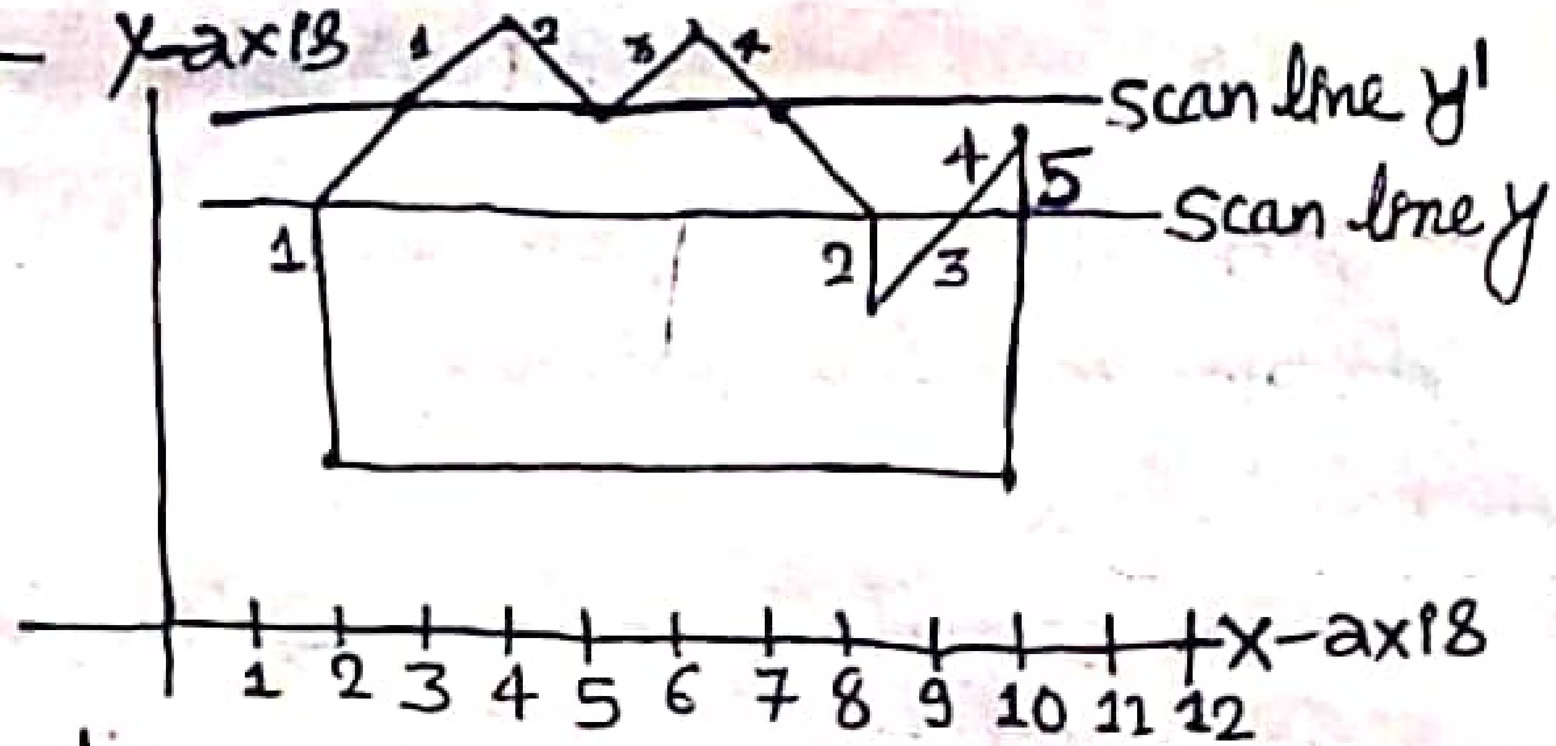


figure 1

In this example scan line y and scan line y' both passes through an vertex or an edge endpoint. Now in case of scan line y' the scan line is intersecting 4 edges i.e, even number of edges and also passing through a vertex/endpoint.

Also both the edges that are connected to the vertex are on same side, of the scan line. So we need to count the vertex/endpoint TWICE so that we can make pairs of intersection points as: (3,5) & (5,7).

Next Example: We consider same above figure 1 for this example. Now in case of scan line y, the scan line is intersecting with 5 different edges i.e, ODD NUMBER and also passing through a vertex/endpoint. So in this case we need to do some extra processing i.e, we need to check if the 2 edges at the endpoint through which the scan line is passing are they on opposite sides or on same sides?

In case of scan line y' they are on same sides and in case of scan line y both edges at the vertex are on opposite sides. So now we need to count the vertex as a single intersection point. Now we just need to sort the intersection points and make pairs of them and fill all pixels which lie inside the pair.

2) Inside – Outside test:

One simple way of finding whether the point is inside or outside a simple polygon is to test how many times a ray, intersects the edges of the polygon. If the point is on the outside of the polygon the ray will intersect its edge an even number of times. If the point is on the inside of the polygon the ray will intersect its edge an odd number of times. This method won't work if the point is on edge of polygon.

This method is also known as counting number method.
There are two methods by which we can identify whether particular point is inside an object or outside.

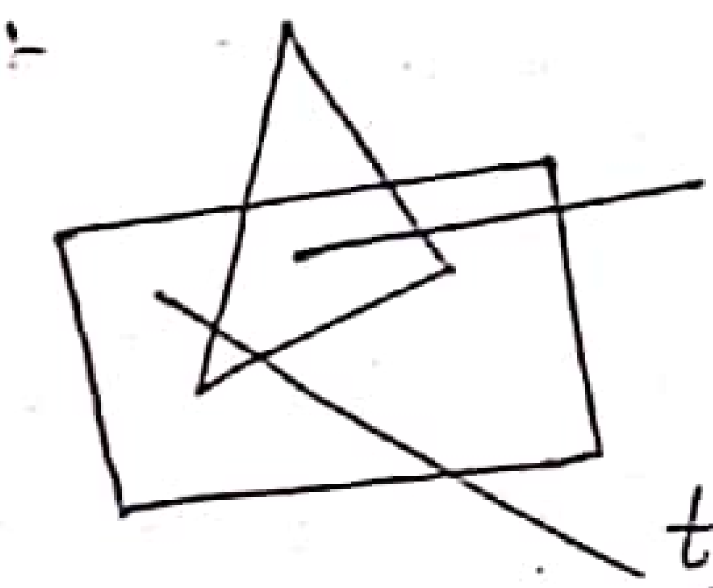
(a) Odd-Even method →

- In odd-even rule/method we draw a line from any position P to a distant point outside the coordinate extents of the object and count the number of edges crossings along the line.

  Condition → We need to make sure that the line drawn from point P does not intersect a vertex or endpoint.

- If the number of edges crossed by the line is ODD then the Point P is in the interior.
- If the number of edges crossed by the line is EVEN then the Point P is in the exterior.

Example:-



this line denotes EXTERIOR since it crossed 2 (i.e, even) number of edges.

this line denotes INTERIOR since it crossed 3 (i.e, odd) number of edges.

(b) Non-Zero winding number method →

- In non-zero winding number method we need to know the direction of each edge in the polygon, i.e, whether the edge is clockwise or counter-clockwise (i.e anti-clockwise)
- The winding number is the number of times the polygon edges wind around a particular point in the counter clockwise direction.

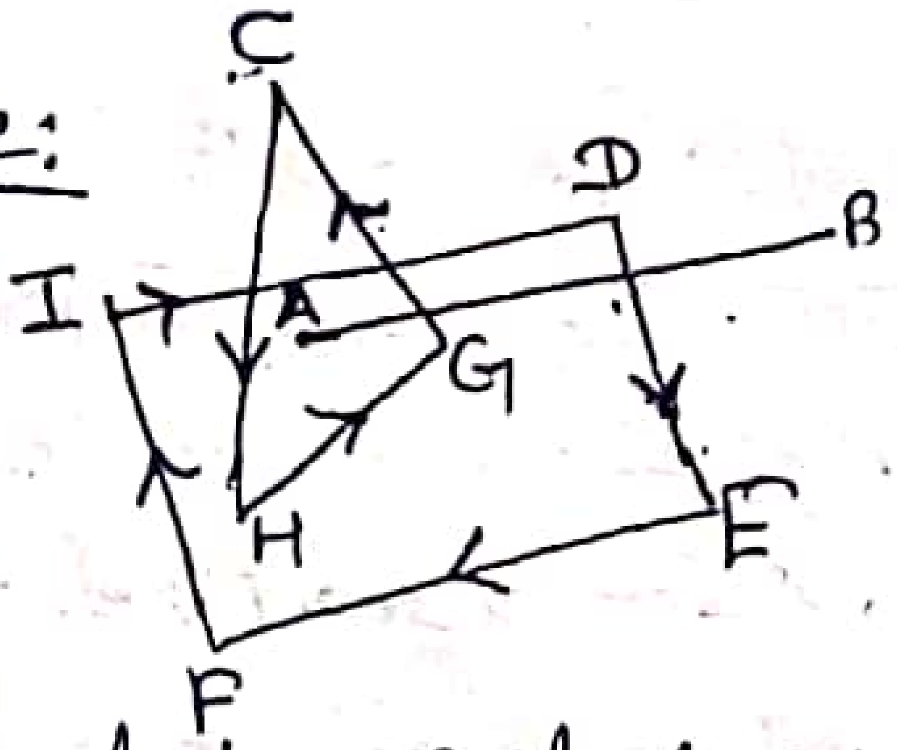
Steps to Apply this method:

1. First we keep the initial value of winding number = 0.
2. Then we imagine drawing a line from point P to outside the polygon which does not pass through any vertex.
3. Now we add 1 to the winding number every time we intersect a polygon edge that crosses the line from right to left, and we subtract 1 every time we intersect an edge that crosses from left to right.

utputs/Results

- The interior points are those which are having a non-zero value for winding number. (i.e, maybe +ve or -ve number except zero).
- Exterior points are those whose value of the winding number is zero.

Example:



Firstly we take initial value of winding number as zero (i.e, INITIAL VALUE = 0). Let we are testing at point A towards B. then G will be our rightside and C will be our leftside. the direction of G to C line is right to left (i.e anti-clockwise direction). So, we add 1 to winding number. Now we reach at point where AB intersects DE which is left to right (i.e, clockwise direction). So, we subtract 1 to winding number. Then finally +1 and -1 becomes zero in winding number. So, final winding number for line AB is zero which shows it as exterior region.

3) Scan line fill of Curved Boundry Area:-

(a). Boundry fill algorithm → In boundry fill algorithm the basic concept is filling the color in closed area by starting at a point inside a region and paint the interior outward towards the boundry. One requirement for boundry fill algorithm is that the boundry has to have a single color.

Working:-

- In boundry fill algorithm, we start from a point inside the region and fill the color interior outward towards the boundry pixel by pixel.
- We should check the default color of the pixel before filling, if the color is not boundry color, then we fill it with the fill color, and move to next pixel and check for the same criteria till we encounter the boundry colored pixel or boundry.

Methods of implementation → There are two methods in which the boundry fill algorithm can be implemented.

i) 4-connected pixel

ii) 8-connected pixel.

4-connected pixel method

In 4 connected pixel method we check 4 pixels adjcent to the current pixel, namely towards the left, right, top & bottom. So we fill the area with 4-connected pixel method by following steps:-

Step 1. First initialize the 4 values namely x, y, fill-color & Default-color. Where x and y are coordinate positions of the initial interior pixel, fill colour is the colour we want to fill and Default-color is the default color of the interior pixel.

Step 2. Define the value of the boundry pixel color or boundry color.

Step 3. Now check if the current pixel is of Default-color and if Yes then Repeat step 4 and step 5 till the boundry pixels are reached.

Step 4. Change the default color with the fill color at the current pixel.

Step 5. Repeat step 3 and step 4 for the neighbouring 4 pixels.

8-connected pixel method

There is a problem with 4-connected pixel method that it cannot fill all the pixels. So the solution for this problem is 8-connected pixel method. So in 8-connected pixel method we instead of filling just 4 adjcent pixels we fill also the adjcent diagonal pixel positions, such as $(x+1, y+1)$.

(b) Flood-fill algorithm → Flood-fill algorithm is useful in cases where there is no single color boundry for the polygon. i.e, the boundry has multiple colors. In flood fill algorithm instead of filling color till we encounter a specific boundry color we just fill the pixels with default color. It is used in the "bucket" fill tool of paint programs.

There are two methods in which flood-fill algorithm can be implemented: 4-connected pixel & 8-connected pixel.

4-connected pixel.

--------------------- (Similar lines as we wrote before in boundry fill)

Steps;

Step1. Initialize the 4 values first namely x, y, Fill-color & Default-color. -- (Step 1 is also similar as we wrote before).

Step2. If the color of node is not equal to default-color, return.

Step3. Set the color of node to replacement-color.

Step4. Set the color of node to the south of node to replacement-color.

Step5. Set the color of node to the north of node to replacement color.

Step6. Set the color of node to the east of node to replacement color.

Step7. Set the color of node to the west of node to replacement color.

8-connected pixel method

We write similar as we wrote for (8-connected pixel) of boundry fill algorithm.