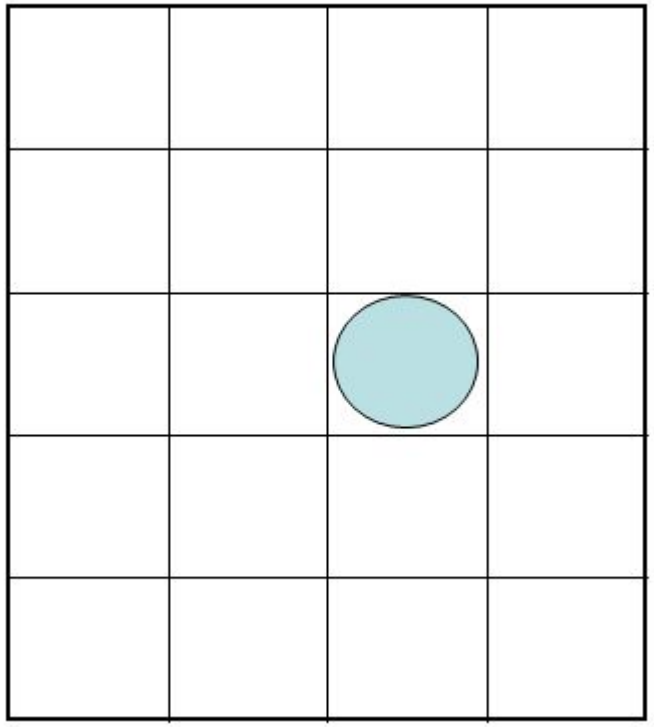
Module 2

Line Drawing Algorithm- DDA, Bresenham's algorithm – Circle Generation Algorithms –Mid point circle algorithm, Bresenham's algorithm- Scan Conversion-frame buffers – solid area scan conversion – polygon filling algorithms

Graphics Output Primitives

Points

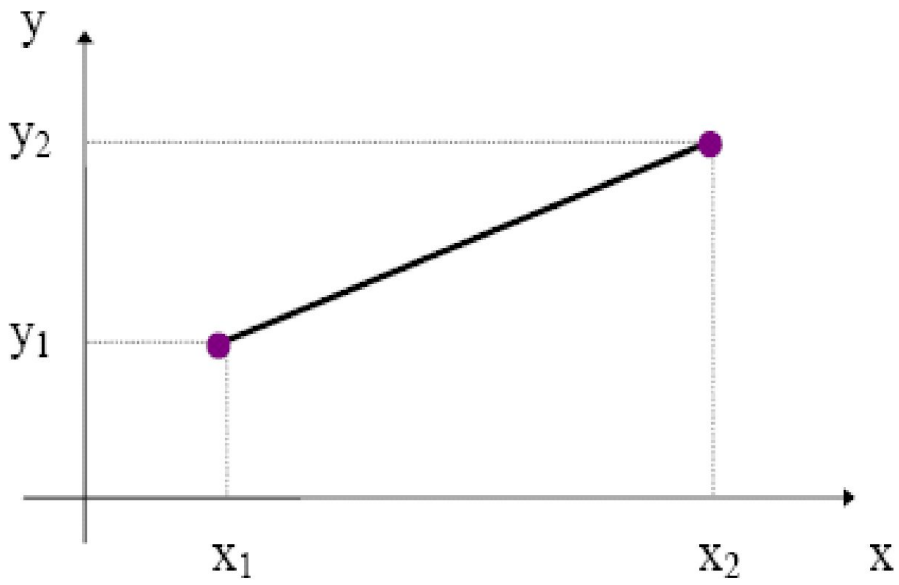
- A point is shown by illuminating a pixel on the screen



Lines

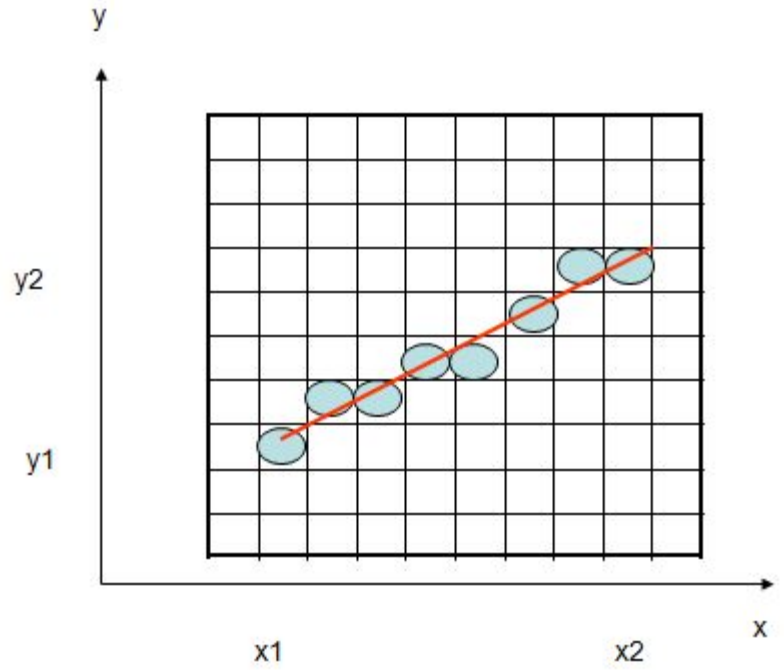
- A line segment is completely defined in terms of its two endpoints.
- A line segment is thus defined as:

  Line_Seg = { $(x_1, y_1), (x_2, y_2)$ }

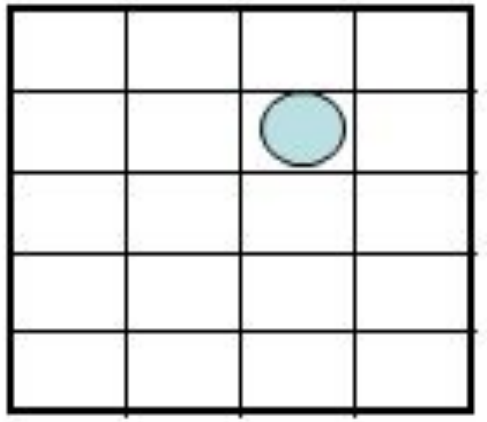


Lines

- A line is produced by means of illuminating a set of intermediary pixels between the two endpoints.

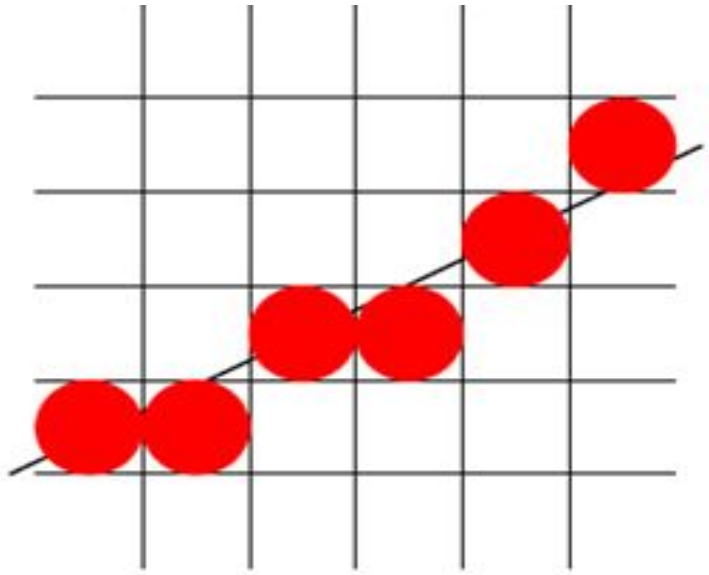


- Lines is digitized into a set of discrete integer positions that approximate the actual line path.
- Example: A computed line position of (10.48, 20.51) is converted to pixel position (10, 21).



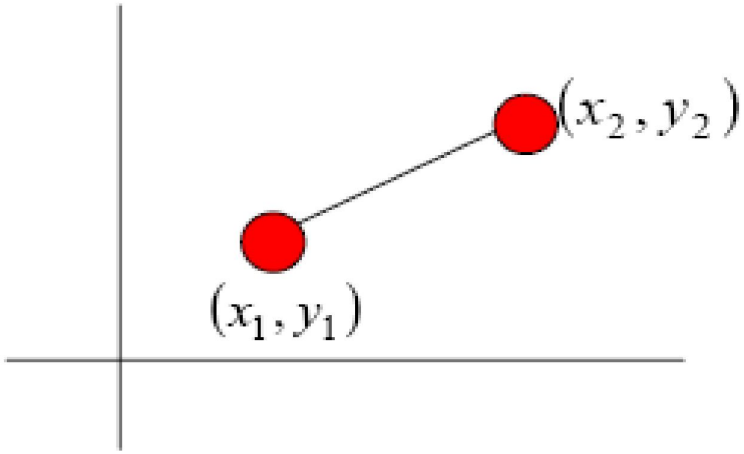
Line

- The rounding of coordinate values to integer causes all but horizontal and vertical lines to be displayed with a stair step appearance “the jaggies”.



Line Drawing Algorithms

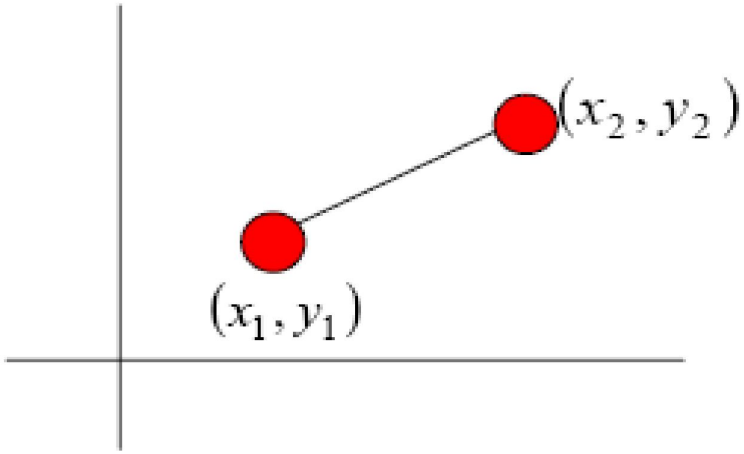
- A straight line segment is defined by the coordinate position for the end points of the segment.
- Given Points (x_1, y_1) and (x_2, y_2)



- **Line drawing is done by calculating intermediate positions along the line path between specified end points positions.**
- **In this type of system the lines are displayed with a staircase step appearance and it is known as jaggies.**
- **Standard algorithms are available to determine which pixels provide the best approximation to the desired line, one such algorithm is the DDA (Digital Differential Analyzer) algorithm.**

Line Drawing Algorithms

- A straight line segment is defined by the coordinate position for the end points of the segment.
- Given Points (x_1, y_1) and (x_2, y_2)

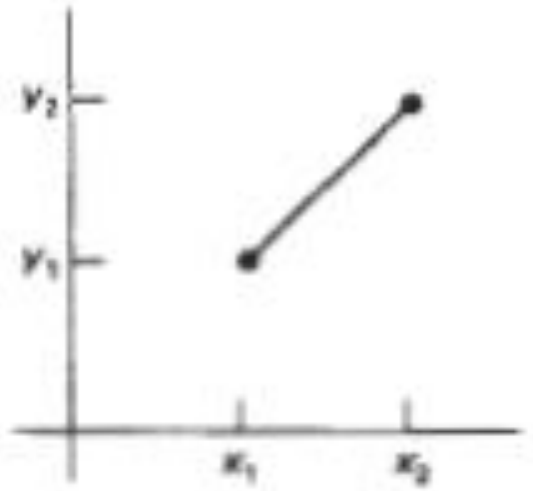


LINE-DRAWING ALGORITHMS

- The Cartesian slope-intercept equation for a straight line is

$$y = m \cdot x + b$$

- **with m representing the slope of the line and b as the intercept.**
- **Given that the two endpoints of a the segment are specified at positions (x1, y1) and (x2, y2), as shown in Fig. 3-3, we can determine values for the slope m and y intercept b with the following calculations:**



$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y_1 - m \cdot x_1$$

Figure 3-3
Line path between endpoint positions (x_1, y_1) and (x_2, y_2).

For any given x interval Ax along a line, we can compute the corresponding y Δy interval

Similarly, we can obtain the x interval Δx corresponding to a specified Δy as

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

Line

- All line drawing algorithms make use of the fundamental equations:

- Line Eqn. $y = m.x + b$
- Slope $m = y_2 - y_1 / x_2 - x_1$
  $= \Delta y / \Delta x$
- y-intercept $b = y_1 - m.x_1$
- x-interval→ $\Delta x = \Delta y / m$
- y-interval→ $\Delta y = m \Delta x$

DDA Algorithm (Digital Differential Analyzer)

- A line algorithm Based on calculating either Δy or Δx *using the above equations.*
- *There are two cases:*
  - *Positive slop*
  - *Negative slop*

DDA- Line with positive Slope

If $m \leq 1$ then take $\Delta x = 1$

- Compute successive y by

$$y_{k+1} = y_k + m \qquad (1)$$

- Subscript k takes integer values starting from 1, for the first point, and increases by 1 until the final end point is reached.
- Consider the first point as (x_k, y_k),then the next point is (x_{k+1}, y_{k+1})

- $m = (y_{k+1} - y_k) / (x_{k+1} - x_k)$ ie $x_{k+1} = x_k + 1$
- $m = (y_{k+1} - y_k)$

- If $m > 1$, reverse the role of x and y and take $\Delta y = 1$, calculate successive x from

$$x_{k+1} = x_k + 1/m \qquad (2)$$

- In this case, each computed x value is rounded to the nearest integer pixel position.
- The above equations are based on the assumption that lines are to be processed from left endpoint to right endpoint.

- Consider the first point as (x_k, y_k),then the next point is (x_{k+1}, y_{k+1})

- $m=(y_{k+1}-y_{k)} /(x_{k+1}- x_k)$ ie $y_{k+1} = y_k + 1$

- $m=1/(x_{k+1}- x_k)$

Merits + Demerits

- Faster than the direct use of line Eqn.
- It eliminates the multiplication in line Eqn.
- For long line segments, the true line Path may be mislead due to round off.
- Rounding operations and floating-point arithmetic are still time consuming.
- The algorithm can still be improved.
- Other algorithms, with better performance also exist.

ALGORITHM

- **Step1: Input two end point pixel positions (x1, y1) and (x2, y2)**
  Step2: Find horizontal and vertical difference between the end points
  dx = x2-x1
  dy = y2-y1
  Step3: The difference with the greater magnitude determines the value of parameter steps
  If abs (dx)>abs (dy) then
  steps = abs (dx)
  else
  steps = abs (dy)
  Step4: Starting with pixel position (x1, y1) be determined offset needed at each step to generate next pixel along the line path.
  xincrement = dx/steps
  yincrement =dy/steps
  Step5: Assign the values of x1, y1 to x, y
  x = x1
  y = y1
  Step6: Plot the pixel at (x, y) position on screen set pixel (round(x), round(y).
  step7: assign k=1 and perform steps 7 and 8 until k=steps.
  Calculate the values of x and y for the next pixel position.
  x = x + xincrement
  y = y + yincrement
  k++
  Step8:Plot the pixel at (x, y) position, set pixel (round(x), round(y).

Code for DDA Algorithm

```
Procedure lineDDA(xa,ya,xb,yb);
Var
        dx,dy,steps,k:
        xIncrement,yIncrement,x,y;
begin
        dx:=xb-xa;
        dy:=yb-ya;
        if abs(dx)>abs(dy) then
           steps:=abs(dx)
           else          steps:=abs(dy);
        xInc:=dx/steps;
        yinc:=dy/steps;
        x:=xa;
        y:=ya;
        putPixel(round(x),round(y));
        for (k=1;k<=steps;k++)
                begin
                        x:=x+xinc;
                        y:=y+yinc;
                        putPixel(round(x),round(y))
                end
end; {lineDDA}
```

Problems on DDA line drawing Algorithm

- **1.Draw a line with point(0,0) and (4,5)(using DDA)**
- dx= x2- x1=4-0=4 ,dy =y2 –y1= 5-0=5
- x =x1=0 , y=y1=0
- dx >dy=4<5
- steps=5
- xinc= dx/steps=4/5=0.8 ,yinc=dy/steps=5/5=1

- x=x+ xinc=0+0.8=0.8
- y=y+ yinc=0+1=1 putpixel(0.8,1)

- x=x+xinc=0.8+0.8=1.6
- y=y+yinc=1+1=2 putpixel(1.6,2)

- x=x+xinc=1.6+0.8=2.4
- y=y+yinc=2+1=3 putpixel(2.4,3)

- x=x+xinc=2.4+0.8=3.2
- y=y+yinc=3+1=4 putpixel(3.2,4)

- x=x+xinc=3.2+0.8=4.0
- y=y+yinc=4+1=5 putpixel(4,5)

2.Draw a line with point(20,10) and (30,18))(using DDA)

- dx= xb- xa=30-20=10 , dy =yb -ya= 18-10=8
- x =xa=20 , y=ya=10
- dx >dy=10>8
- steps=10
- xinc= dx/steps=10/10=1 , yinc=dy/steps=8/10=0.8

- x=x+ xinc=20+1=21
- y=y+ yinc=10+0.8=10.8 putpixel(21,10.8)

- x=x+xinc=21+1=22
- y=y+yinc=10.8+0.8=11.6 putpixel(22,11.6)

- x=x+xinc=22+1=23
- y=y+yinc=11.6+0.8=12.4 putpixel(23,12.4)

- x=x+ xinc=23+1=24
- y=y+ yinc=12.4+0.8=13.2 putpixel(24,13.2)

- x=x+xinc=24+1=25
- y=y+yinc=13.2+0.8=14.0 putpixel(25,14.0)

- x=x+xinc=25+1=26
- y=y+yinc=14.0+0.8=14.8 putpixel(26,14.8)

- x=x+xinc=26+1=27
- y=y+yinc=14.8+0.8=15.6 putpixel(27,15.6)

- x=x+xinc=27+1=28
- y=y+yinc=15.6+0.8=16.4 putpixel(28,16.4)

- x=x+xinc=28+1=29
- y=y+yinc=16.4+0.8=17.2 putpixel(29,17.2)

- x=x+xinc=29+1=30
- y=y+yinc=17.2+0.8=18 putpixel(30,18)

- DDA Algorithm is a faster method for calculating the pixel position but in some cases it requires some rounding operations and it is time consuming

Q1.Draw a line with point(2,4) and (9,9))(using DDA)
Q2.Draw a line with point(2,1) and (7,7))(using DDA)

Q3.Using DDA algorithm rasterize the line from (0, 0) to (4, 6) and plot the line.

- Identify dx, dy, xin, yin according to the DDA algorithm[2m]
- Identifying the points [7m]
- Plot the point[1m]

x	y	Plotting point
0.66	1	(1,1)
1.32	2	(1,2)
1.98	3	(2,3)
2.64	4	(3,4)
3.3	5	(3,5)
3.96	6	(4,6)

- **Advantages**
  - It eliminates the multiplication in Eq. (1) by making use of raster characteristics, so that appropriate increments are applied in the x or y direction to step to pixel positions along the line path.
  - Does not calculate coordinates based on the complete equation (uses offset method)

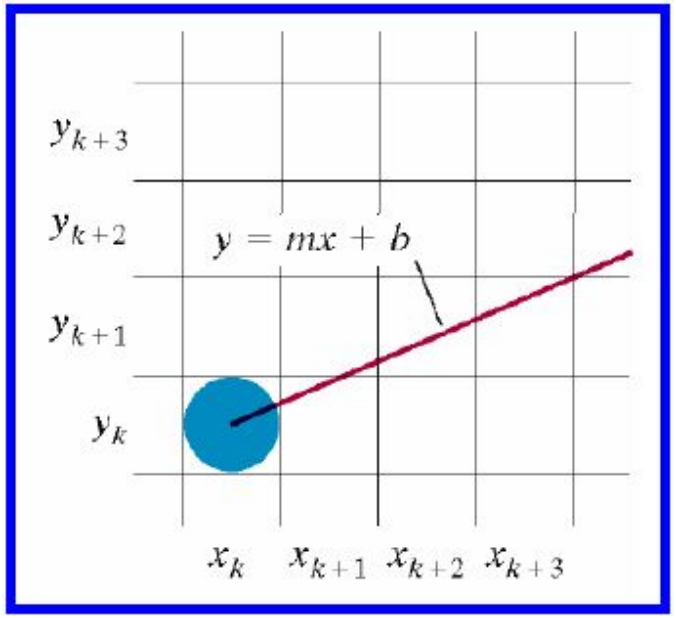
  Disadvantages:
  - Round-off errors are accumulated, thus line diverges more and more from straight line
  - Round-off operations take time
  - Perform integer arithmetic by storing float as integers in numerator and
  - denominator and performing integer arithmetic

Bresenham's Line Algorithm

- It is an efficient raster line generation algorithm.
- It can be adapted to display circles and other curves.
- The algorithm
  - After plotting a pixel position (x_k, y_k) , what is the next pixel to plot?
- Consider lines with positive slope.

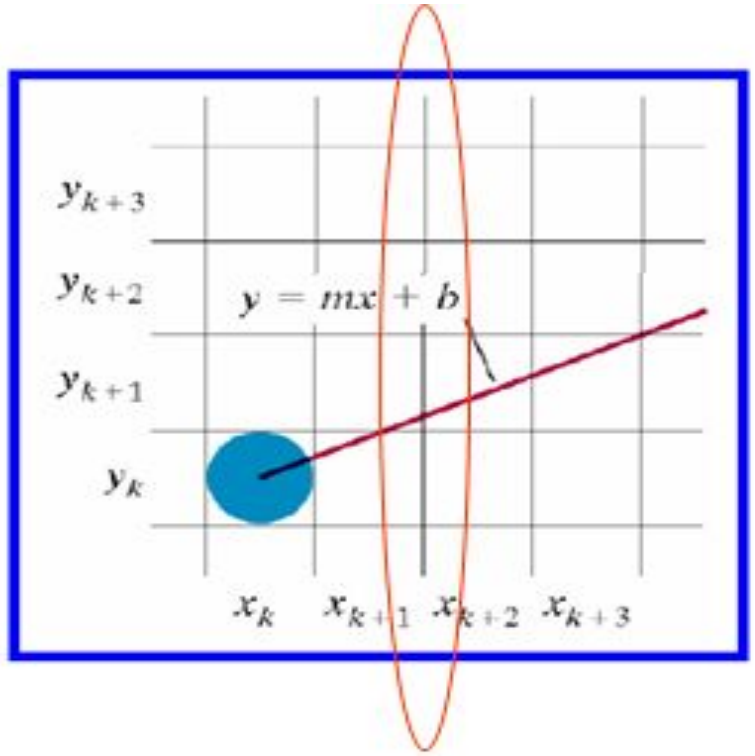
Bresenham's Line

- For a positive slope, $0 < m < 1$ and line is starting from left to right.
- After plotting a pixel position (x_k, y_k) we have two choices for next pixel:
  - $(x_k + 1, y_k)$
  - $(x_k + 1, y_k + 1)$



Bresenham's Line

- At position x_k +1, we pay attention to the intersection of the vertical pixel and the mathematical line path.



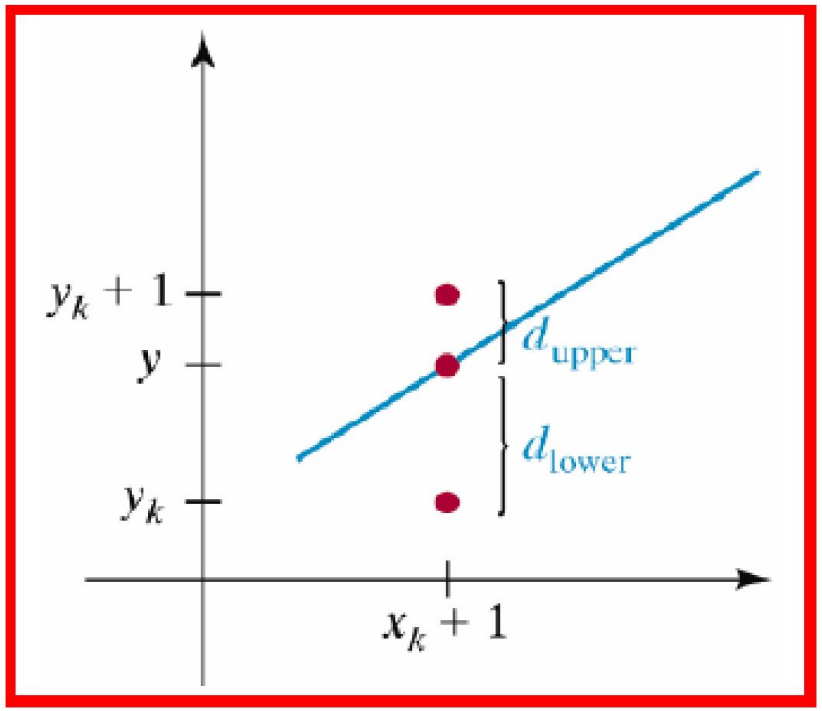
Bresenham's Line

- At position x_k +1, we label vertical pixel separations from the mathematical line path as

  d_{lower} , d_{upper}.

  d1= d_{lower}

- d2=dupper.



Bresenham's Line

- The y coordinate on the mathematical line at x_k+1 is calculated as

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

then

$$d_1 = y - y_k$$
$$= m(x_k + 1) + b - y_k$$

and

$$d_2 = (y_k + 1) - y$$
$$= y_k + 1 - m(x_k + 1) - b$$

Bresenham's Line

- To determine which of the two pixels is closest to the line path, we set an efficient test based on the difference between the two pixel separations

$$d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1$$

$$= 2(\Delta y / \Delta x)(x_k + 1) - 2y_k + 2b - 1$$

- Consider a decision parameter p_k such that

$$p_k = \Delta x (d_1 - d_2)$$

$$= \Delta x(2m(x_k + 1) - 2y_k + 2b - 1)$$

$$= \Delta x [2(\Delta y / \Delta x)(x_k + 1) - 2y_k + 2b - 1]$$

$$= 2\Delta y(x_k + 1) - 2\Delta x\, y_k + \Delta x\, 2b - \Delta x$$
$$= 2\Delta y(x_k + 1) - 2\Delta x\, y_k + \Delta x(2b - 1)$$

$$p_k = 2\,\Delta y\,(x_k) + 2\,\Delta y - 2\,\Delta x\,y_k + \Delta x(2b-1) \text{----------} 1$$

$$p_k = 2\Delta y.x_k - 2\Delta x.y_k + c$$

- where

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

$$p_k = 2\Delta y.x_k - 2\Delta x.y_k + c$$

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

- Subtracting these two equations

$$p_{k+1} - p_k = 2\Delta y\,(x_{k+1} - x_k) - 2\Delta x\,(y_{k+1} - y_k)$$

- But $x_{k+1} - x_k = 1$, Therefore

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x\,(y_{k+1} - y_k)$$

- $p_k = 2\,\Delta y\,(x_k) + 2\,\Delta y - 2\,\Delta x\,y_k + \Delta x(2b-1)$

- $K=0,\ p_0 = 2\,\Delta y\,(x_0) + 2\,\Delta y - 2\,\Delta x\,y_0 + \Delta x(2b-1)$----------2
- $b = y_0 - (\Delta y/\Delta x)\,x_0$
- $y = mx + b$
- $y_0 = mx_0 + b$
- $b = y_0 - mx_0$

  $b = y_0 - (\Delta y/\Delta x)\,x_0$

Substitute value of b in equation 2

- $p_0 = 2\,\Delta y\,(x_0) + 2\,\Delta y - 2\,\Delta x\,y_0 + \Delta x(2b-1)$

- $p_0 = 2\,\Delta y\,x_0 + 2\,\Delta y - 2\,\Delta x\,y_0 + \Delta x(2(y_0 - (\Delta y/\Delta x)\,x_0) - 1)$

$= 2\,\Delta y\,x_0 + 2\,\Delta y - 2\,\Delta x\,y_0 + \Delta x(2y_0 - 2\,\Delta y/\Delta x\,x_0 - 1$

$= 2\,\Delta y\,x_0 + 2\,\Delta y - 2\,\Delta x\,y_0 + \Delta x 2y_0 - \Delta x\,2\Delta y/\Delta x\,x_0 - \Delta x$

$= 2\,\Delta y\,x_0 + 2\,\Delta y - \underline{2\,\Delta x\,y_0} \;\underline{+\Delta x 2y_0} - 2\Delta y\,x_0 - \Delta x$

$p_0 = 2\,\Delta y - \Delta x$

Bresenham's Line-Drawing Algorithm

1. Input the two line end points and store the left end point in (x_0, y_0).

2. Load (x_0, y_0) into the frame buffer; that is, plot the first point.

3. Calculate the constants Δx, Δy, $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as

$$p_0 = 2\Delta y - \Delta x$$

4. At each x_k along the line, starting at $k = 0$, perform the following test: If $p_k < 0$, the next point to plot is (x_{k+1}, y_k) and

$$p_{k+1} = p_k + 2\Delta y$$

Otherwise, the next point to plot is (x_{k+1}, y_{k+1}) and

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

5. Repeat step 4, Δx times.

Problem

To illustrate the algorithm, we digitize the line with endpoints (20, 10) and (30, 18). This line has a slope of 0.8, with

$$\Delta x = 10, \qquad \Delta y = 8$$

The initial decision parameter has the value

$$\begin{aligned} p_0 &= 2\Delta y - \Delta x \\ &= 6 \end{aligned}$$

and the increments for calculating successive decision parameters are

$$2\Delta y = 16, \qquad 2\Delta y - 2\Delta x = -4$$

We plot the initial point $(x_0, y_0) = (20, 10)$, and determine successive pixel positions along the line path from the decision parameter as

- P_k=6> 0
- Next point (21,11)
- P_{k+1} = 6+2*8-2*10
  =2
- 2>0 next point(22,12)
- P_{k+1} = 2+2*8-2*10
  =2+16-20
  =-2

-2<0 next point(23,12)

- P_{k+1} = -2+2*8
  =-2+16
  =14

14>0 next point(24,13)

- P_{k+1} = 14+2*8-2*10
  =14+16-20
  =14-4=10

10>0 next point(25,14)

- P_{k+1} = 10+2*8-2*10
  =10+16-20
  =10-4=6

6>0 next point(26,15)

- P_{k+1} = 6+2*8-2*10
  =6+16-20
  =6-4=2

2>0 next point(27,16)

P_{k+1} = 2+2*8-2*10
=2+16-20
=2-4=-2

-2 < 0 next point(28,16)

P_{k+1} = -2+2*8
=-2+16
=14

14>0 next point(29,17)

P_{k+1} = 14+2*8-2*10
=14+16-20
=14-4=10

10>0 next point(30,18)

P_{k+1} = 10+2*8-2*10
=10+16-20
=10-4=6

Example

A plot of the pixels generated along this line path is shown in Fig.

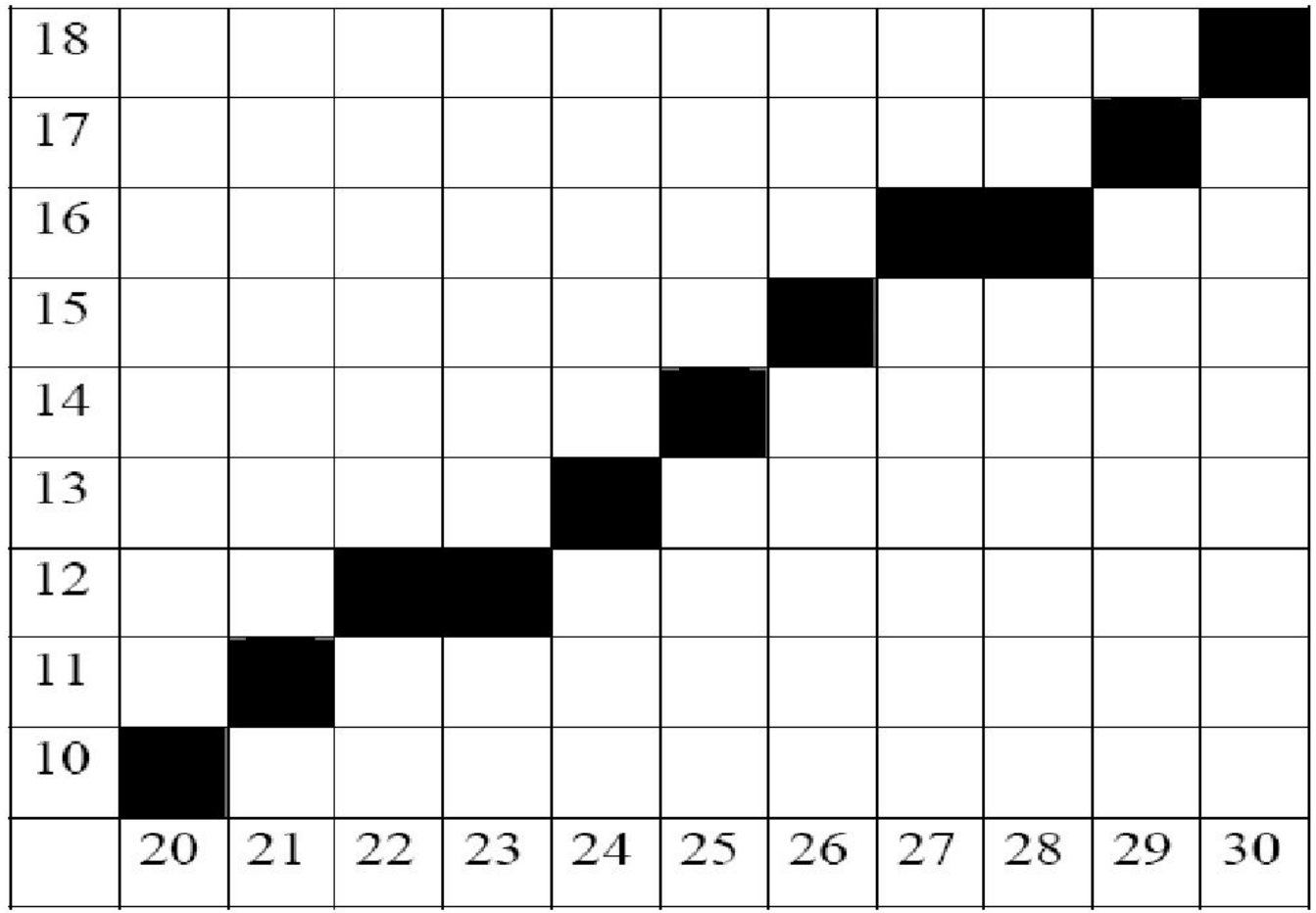


Figure: The Bresenham line from point (20,10) to point (30,18)

Q1. Compare DDA line drawing Algorithm with Bresenham's line drawing algorithm [ktu dec 2017, 3 marks]

	Digital Differential Analyzer Line Drawing Algorithm	Bresenhams Line Drawing Algorithm
Arithmetic	DDA algorithm uses floating points i.e. Real Arithmetic.	Bresenhams algorithm uses fixed points i.e. Integer Arithmetic.
Operations	DDA algorithm uses multiplication and division in its operations.	Bresenhams algorithm uses only subtraction and addition in its operations.
Speed	DDA algorithm is rather slowly than Bresenhams algorithm in line drawing because it uses real arithmetic (floating-point operations).	Bresenhams algorithm is faster than DDA algorithm in line drawing because it performs only addition and subtraction in its calculation and uses only integer arithmetic so it runs significantly faster.
Accuracy & Efficiency	DDA algorithm is not as accurate and efficient as Bresenham algorithm.	Bresenhams algorithm is more efficient and much accurate than DDA algorithm.
Round Off	DDA algorithm round off the coordinates to integer that is nearest to the line.	Bresenhams algorithm does not round off but takes the incremental value in its operation.
Expensive	DDA algorithm uses an enormous number of floating-point multiplications so it is expensive.	Bresenhams algorithm is less expensive than DDA algorithm as it uses only addition and subtraction.

Questions

Scan convert the line segment with end points (30,20) and (15,10) using DDA line drawing algorithm (4)

What are the advantages and disadvantages of DDA line drawing algorithm (2)

Rasterize the line segment from pixel coordinate (1, 1) to (8, 5) using Bresenham's line drawing algorithm. (5)

Generate the points between the end points of a line viz.(2,2) and (9,6) by using Bresenham's line drawing algorithm. (5)

Scan convert the line segment with end points (30,20) and (15,10) using DDA line drawing algorithm. (4)

CIRCLE DRAWING ALGORITHMS

The most important thing in drawing a circle is learning how the circle is drawn using 8-way symmetry. It is based on Mirror reflection. If we see Right hand in the mirror we will see Left hand, Similarly if we see pixel (x,y) in mirror we will see (y,x).

So point P1(x,y) will become P2(y,x) after reflection. Point P3 (-y,x) will become P4 (-x,y) and so on.

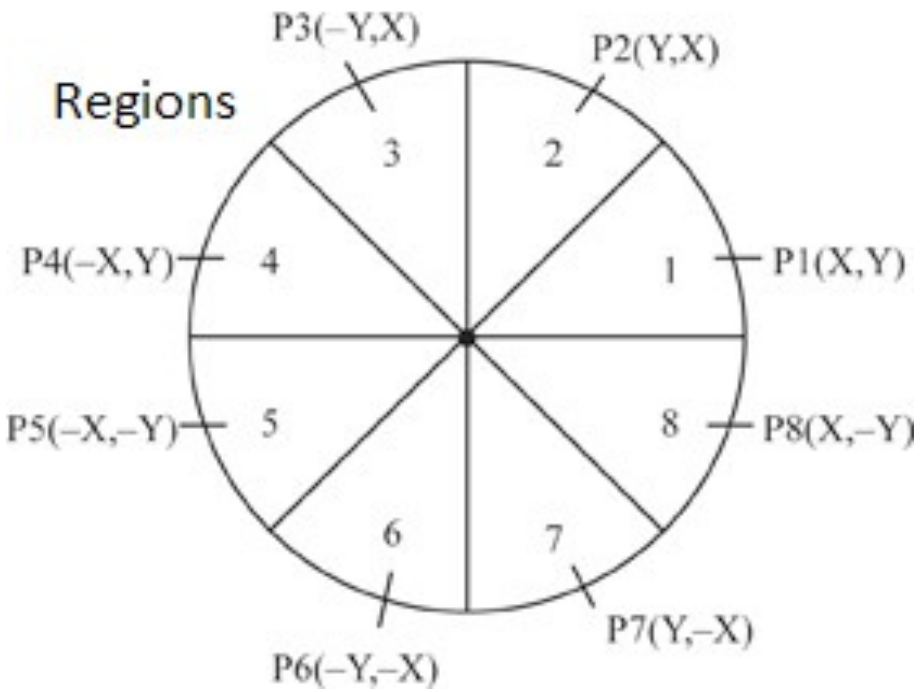
Lets understand more how these negative signs are taken. Remember following two things.

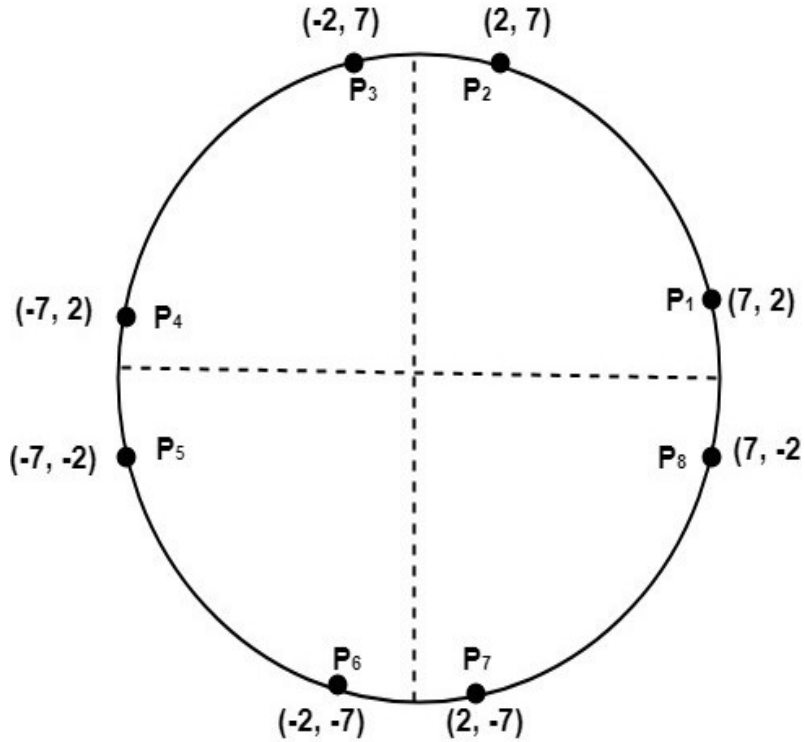
One – If we mirror reflect (x,y) with respect to x axis then y will change so it will become (x,-y). Similarly if we reflect point (x,y) with respect to y axis x will change , so the point will become (-x,y).

Two – Read Point (x,y) as : x is on x position and y is on y position, so when we reflect with respect to x axis then y position becomes negative and with respect to y axis x position becomes negative. Take this logic and reflection of point P2 (y,x) with respect to y axis will make the x position negative and not where x is written, so it will become P3 (-y,x).

Reflection of point P8 (x,-y) with respect to x axis will make y position negative and not where y is written, so it will become P1(x,y).

8-Way Symmetry Diagram





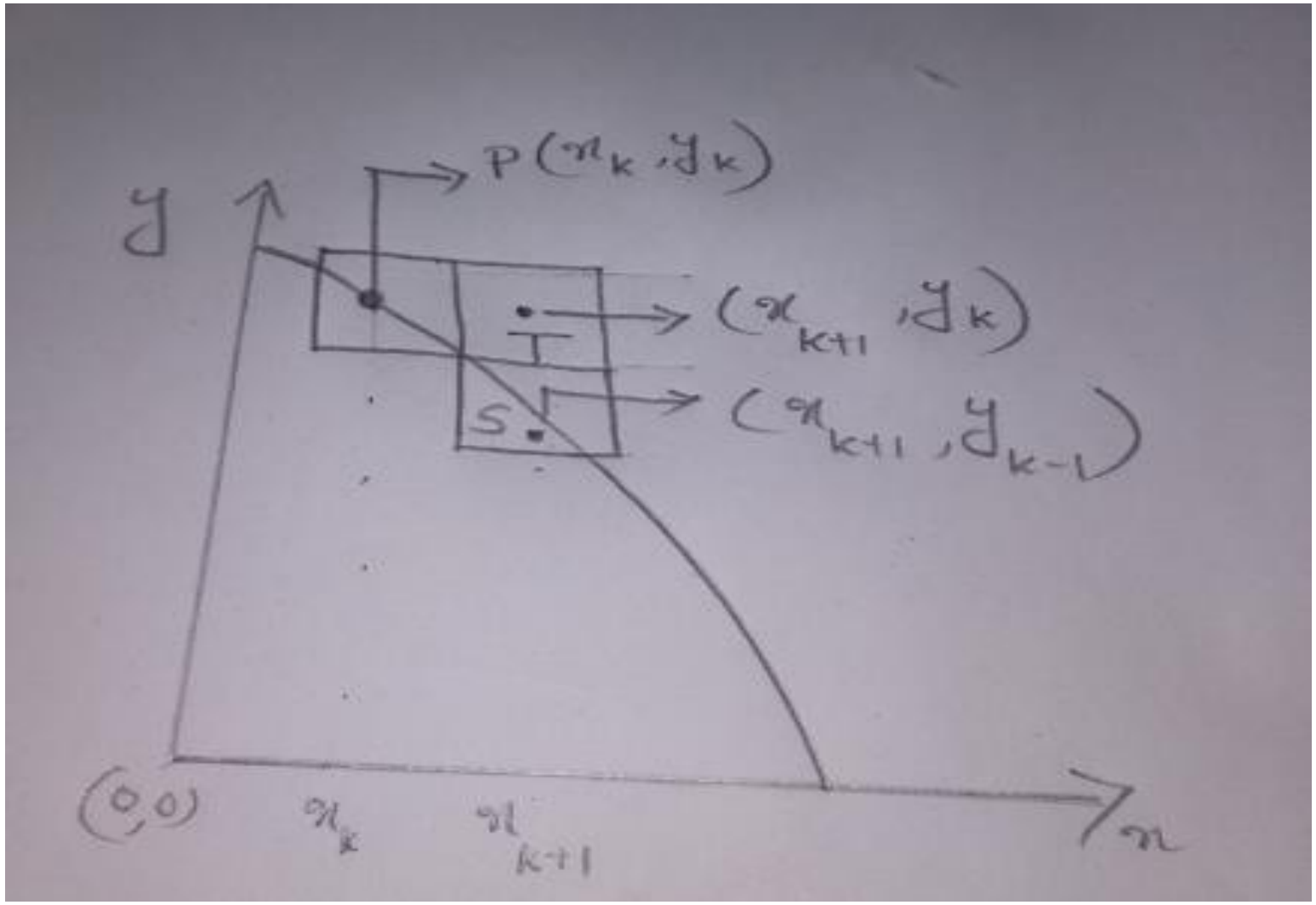
Eight way symmetry of a Circle

THERE ARE TWO ALGORITHMS FOR GENERATING A CIRCLE

- Bresenhm's circle drawing algorithm

- Midpoint circle drawing algorithm

BRESENHAM'S CIRCLE DRAWING ALGORITHM



The general eqn of a circle is

$$x^2 + y^2 = r^2$$

Let P be a point on the circle which will be accurately plotted with coordinate x_k & y_k. Let T be a point outside the circle with coordinates (x_{k+1}, y_k) and S be a point inside the circle with coordinate (x_{k+1}, y_{k-1})

If (x_k, y_k) is the last plotted pixel so for x_{k+1} we have 2 pixel values y_k and y_{k-1}

As to Bresenham's algo we have to find the pixel position closest to the scan line.

Distance of pixel T from origin (0,0)

$$D(T) = \sqrt{(x_k+1)^2 + y_k^2}$$

D(T) will be always +ve as it is outside true circle

Distance of pixel S from origin (0,0)

$$D(S) = \sqrt{(x_k+1)^2 + (y_k-1)^2}$$

D(S) will be -ve as it is inside true circle

Let d_1 be distance of pixel T from scan line

& d_2 be distance of pixel S to scan line

$$d_1 = \sqrt{(x_k+1)^2 + y_k^2} \quad - x$$

$$d_2 = \sqrt{(x_k+1)^2 + (y_k-1)^2} \quad - x$$

Squaring on both sides.

$$d_1 = (x_k+1)^2 + y_k^2 - x^2$$

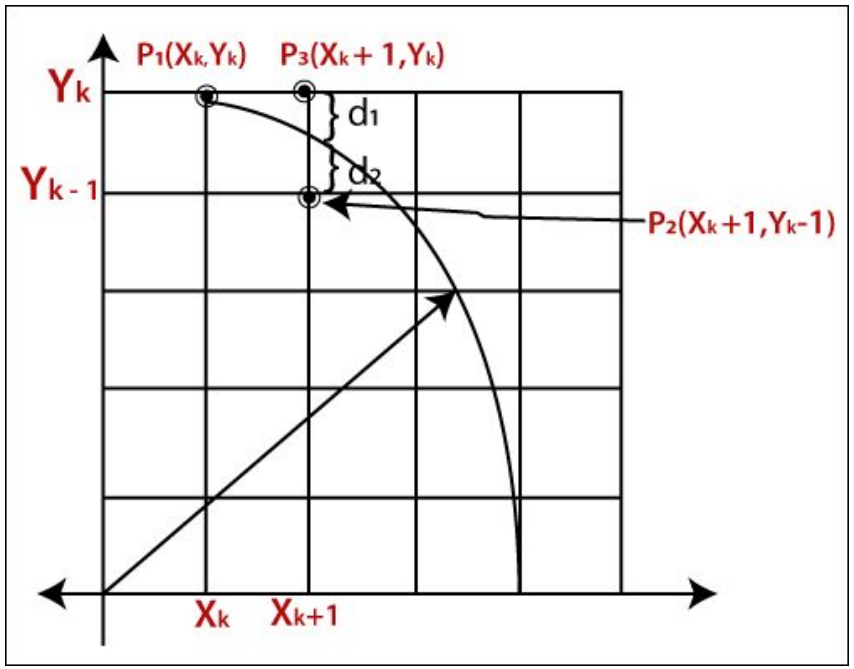
[value of d_1 is eql to zero
it is considered = 0]

$$d_2 = (x_k+1)^2 + (y_k-1)^2 - x^2$$

$$P_k = d_1 - d_2$$

$$= d_1 - (-d_2)$$

$$P_k = d_1 + d_2$$


P1(Xk,Yk)
P3(Xk + 1,Yk)
Yk
d1
d2
Yk - 1
P2(Xk +1,Yk-1)
Xk
Xk+1

- **$P_k < 0$ Point T is closer to circle boundary, and the final coordinates are-**

  $(x_{k+1}, y_k) = (x_k + 1, y_k)$

  $$P_{K+1} = P_K + 4X_K + 6$$

- **$P_k >= 0$ Point S is closer to circle boundary, and the final coordinates are-**

  $(x_{k+1}, y_k) = (x_k + 1, y_k - 1)$

  $$P_{K+1} = P_K + 4 (X_K - Y_K) + 10$$

- ***Initial decision parameter***

  $$Po = 3 - 2r$$

Algorithm

Step-1: Input radius r and circle center (X_C, Y_C) obtained the first point

$(X_0, Y_0) = (0, r)$

Step-2:Calculate the initial value of decision parameter as

$P_0 = 3-2r$

Step-3: At each X_K position, starting at k=0, perform, the following test:

If $P_k < 0$, the next point is (X_K+1, Y_K)

$P_{K+1}=P_K+ 4X_K+6$

Otherwise the next point is (X_K+1, Y_K-1)

$P_{K+1}=P_K+4(X_K-Y_K)+10$

Step-4: Determine the symmetry points in other seven octant

Step-5: Move each pixel position (X,Y) into circular path

$X=X+X_C$ and $Y=Y+Y_C$

Step-6: Repeat step 3 to 5 until $X \geq Y$

Example-:

Given a circle radius r=10, we demonstrate the Bresenham circle drawing algorithm by determining position along the circle octant in the first quadrant from X=0 to X=Y.

Solution:

The initial decision parameter P_0

$P_0=3-2r=3-20=-17 \quad \therefore P_0<0 \Longrightarrow (X_1,Y_1)=(1,10)$

For the circle centered on the coordinator origin, the initial point is $(X_0, Y_0)=(0, 10)$ and initial increment terms for calculating the decision parameter are

1) $P_1=P_0+4X_0+6$

$= -17+0+6$

$= -11 \quad \therefore P_1<0 \Longrightarrow (X_2,Y_2)=(2,10)$

2) $P_2=P_1+4X_1+6$

$= -11+4+6$

$= -1 \quad \therefore P_2<0 \Longrightarrow (X_3,Y_3)=(3,10)$

3) $P_3=P_2+4X_2+6$

$= -1+8+6$

$= 13 \quad \therefore P_3>0 \Longrightarrow (X_4,Y_4)=(4,9)$

4) $P_4=P_3+4(X_3-Y_3)+10$

$= 13- 28+ 10$

$= -5 \qquad \therefore P_4< 0 \Longrightarrow (X_5,Y_5)=(5,9)$

5) $P_5=P_4+4X_4+6$

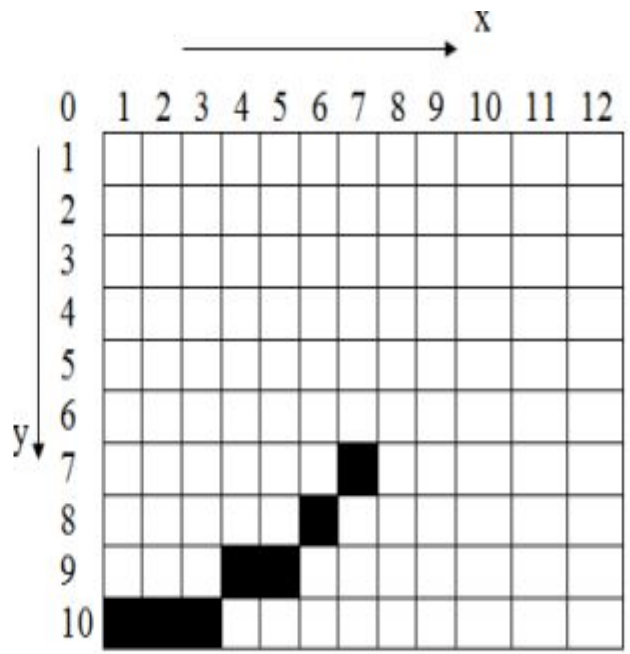
$= -5+ 16+ 6$

$= 17 \qquad \therefore P_5> 0 \Longrightarrow (X_6,Y_6)=(6,8)$

$P_6=P_5+4(X_5-Y_5)+10$

$= 11 \qquad \therefore P_6 >0 \Longrightarrow (X_7,Y_7)=(7,7)$

k	P	X	Y	(X,Y)
0	-17	1	10	(1, 10)
1	-11	2	10	(2, 10)
2	-1	3	10	(3, 10)
3	13	4	9	(4, 9)
4	-5	5	9	(5, 9)
5	17	6	8	(6, 8)
6	11	7	7	(7, 7)



- The radius of a circle is 8, and center point coordinates are (0, 0). Apply bresenham's circle drawing algorithm to plot all points of the circle.

(xo ,yo) =(0,r) =(0,8)

Now, we will calculate the initial decision parameter **(P_0)**

$$P_0 = 3 - 2 \times r$$
$$P_0 = 3 - 2 \times 8$$
$$P_0 = -13$$

The value of initial parameter $P_0 < 0$. So, case 1 is satisfied.

Thus,

$$x_{k+1} = x_k + 1 = 0 + 1 = 1$$
$$y_{k+1} = y_k = 8$$
$$P_{k+1} = P_k + 4x_{k+1} + 6 = -13 + (4 \times 1) + 6 = -3$$

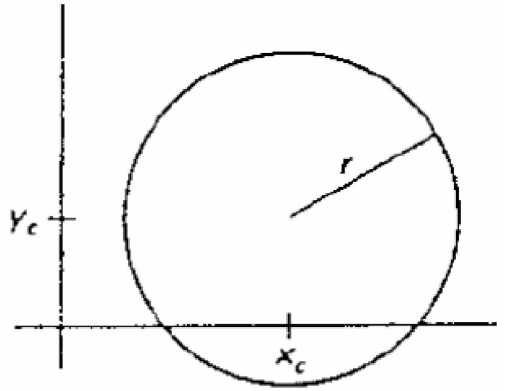
- Follow step 3 until we get **x >= y.**

k	p_k	p_{k+1}	(x_{k+1}, y_{k+1})
0			**(0, 8)**
1	**-13**	**-3**	**(1, 8)**
2	**-3**	**11**	**(2, 8)**
3	**11**	**5**	**(3, 7)**
4	**5**	**7**	**(4, 6)**
5	**7**		**(5, 5)**

- **Advantages**
- It is simple agorithm.
- It can be implement easily
- It is totally based on the equation of circle
- **Disadvantages**
- There is the problem of accuracy while generating points.
- This algorithm is not suitable for the complex and high graphic images.

Circle Generating Algorithms

- A circle is defined as the set of points that are all at a given distance r from a center point (x_c, y_c).



- For any circle point (x, y), this distance is expressed by the Equation

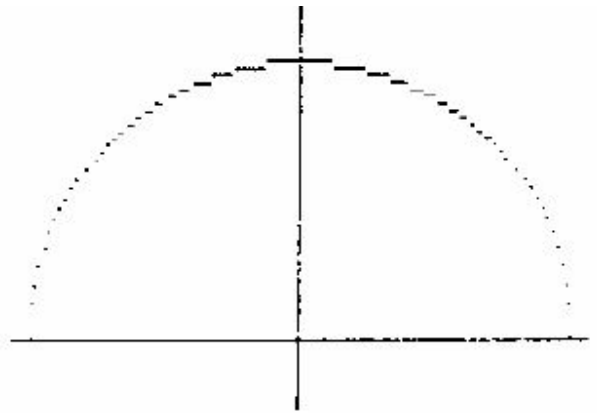
$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

- We calculate the points by stepping along the x-axis in unit steps from x_c-r to x_c+r and calculate y values as

$$y = y_c \pm \sqrt{r^2 - (x_c - x)^2}$$

- **There are some problems with this approach:**

1. Considerable computation at each step.
2. Non-uniform spacing between plotted pixels as in this Figure.



Mid point Circle Algorithm

- To apply the midpoint method, we define a circle function:

$$f_{circle}(x,y) = x^2 + y^2 - r^2 = 0 \qquad (2)$$

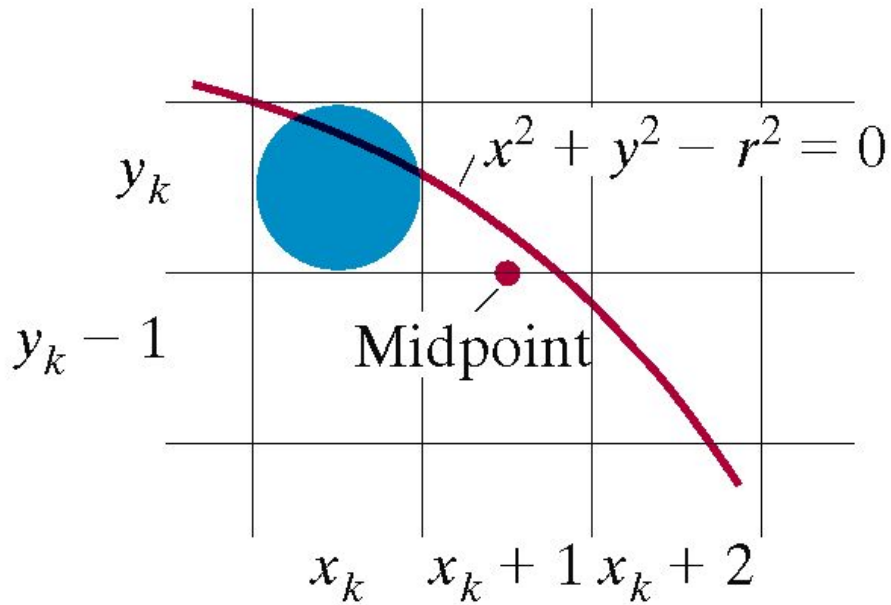
- Any point (*x,y*) on the boundary of the circle with radius *r* satisfies the equation $f_{circle}(x, y)= 0$.
- If the points is in the interior of the circle, the circle function is negative.
- If the point is outside the circle, the circle function is positive.

- To summarize, the relative position of any point (*x,y*) can be determined by checking the sign of the circle function:

$$f_{circle}(x,y) \begin{cases} < 0 & \text{if } (x,y) \text{ is inside the circle boundary} \\ = 0 & \text{if } (x,y) \text{ is on the circle boundary} \\ > 0 & \text{if } (x,y) \text{ is outside the circle boundary} \end{cases} \quad (3)$$

The circle function tests in (3) are performed for the mid positions between pixels near the circle path at each sampling step. Thus, the circle function is the decision parameter in the midpoint algorithm, and we can set up incremental calculations for this function as we did in the line algorithm

- Figure below shows the midpoint between the two candidate pixels at sampling position x_k +1. Assuming we have just plotted the pixel at (x_k , y_k), we next need to determine whether the pixel at position $(x_k +1, y_k)$ or the one at position $(x_k +1, y_k -1)$ is closer to the circle.



Q1. Derive the decision parameter in midpoint circle drawing algorithm and write the algorithm [ktu 8 marks, dec 2017]

- Our decision parameter is the circle function (2) evaluated at the midpoint between these two pixels:

$$p_k = f_{circle}\left(x_k + 1, y_k - \frac{1}{2}\right)$$

$$= (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2 \qquad (4)$$

- We obtain a recursive expression for the next decision parameter by evaluating the circle function

$$p_{k+1} = f_{circle}\left(x_{k+1}+1, y_{k+1}-\frac{1}{2}\right)$$

$$= \left[(x_k+1)+1\right]^2 + \left(y_{k+1}-\frac{1}{2}\right)^2 - r^2$$

or

$$p_{k+1} - p_k = (x_k + 1)^2 + 2(x_k + 1) + 1 + y_{k+1}{}^2 - y_{k+1} + {}^1 - r^2 - (x_k + 1)^2 - y_k{}^2 + y_k - {}^1 + r^2$$

$$p_{k+1} = p_k + 2(x_k+1) + \left(y^2{}_{k+1} - y^2{}_k\right) - (y_{k+1} - y_k) + 1$$

where y_{k+1} is either y_k or y_{k-1},depending on the sign of p_k.

- The initial decision parameter is obtained by evaluating the circle function at the start position $(x_0 , y_0)=(0, r)$:

$$p_0 = f_{circle}\left(1, r - \frac{1}{2}\right)$$

$$= 1 + \left(r - \frac{1}{2}\right)^2 - r^2$$

or

$$p_0 = \frac{5}{4} - r$$

- If the radius r is specified as an integer, we can simply round p_0 to

$$p_0 = 1 - r \quad (\text{for } r \text{ an integer})$$

since all increments are integers

Algorithm

1. Input radius r and circle center (x_c, y_c),and obtain the first point on the circumference of a circle centered on the origin as $(x_0, y_0) = (0, r)$
2. Calculate the initial value of the decision parameter as
   $$p_0 = \frac{5}{4} - r$$
3. At each x_k position, starting at $k = 0$, perform the following test: If $p_k < 0$, the next point along the circle centered on $(0,0)$ is (x_{k+1}, y_k) and
   $$p_{k+1} = p_k + 2x_{k+1} + 1$$
   Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ and
   $$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$
   where $2x_{k+1} = 2x_k + 2$ and $2y_{k+1} = 2y_k - 2$.
4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position (x, y) onto the circular path centered on (x_0, y_0) and plot the coordinate values:
   $x = x + x_c, y = y + y_c$
6. Repeat steps 3 through 5 until $x \geq y$.

Example

- Given a circle radius $r = 10$, we demonstrate the midpoint circle algorithm by determining positions along the circle octant in the first quadrant from $x = 0$ to $x = y$. The initial value of the decision parameter is

$$p_0 = 1 - r = -9$$

For the circle centered on the coordinate origin, the initial point is (x0 , y0) =(0,10)

- P_k=-9 <0
- Next point(1,10)
- $P_{k+1} = P_k+2x_{k+1}+1$
  =-9+2*1+1
  =-6 <0
- Next point(2,10)
- $P_{k+1} = P_k+2x_{k+1}+1$
  =-6+2*2+1
  =-1 <0
- Next point(3,10)
- $P_{k+1} = P_k+2x_{k+1}+1$
  =-1+2*3+1
  =6 >

Next point(4,9)

$P_{k+1} = P_k+2x_{k+1}+1 -2y_{k+1}$
=6+2*4+1-2*9
=-3 <0

Next point(5,9)

$P_{k+1} = P_k+2x_{k+1}+1$
=-3+2*5+1
=8 >0

Next point(6,8)

$P_{k+1} = P_k+2x_{k+1}+1-2y_{k+1}$
=8+2*6+1-2*8
=5>0

Next point(7,7)

$P_{k+1} = P_k+2x_{k+1}+1-2y_{k+1}$
=5+2*7+1-2*7
=6>0

Write the midpoint circle drawing algorithm. (4)

Use midpoint circle drawing algorithm to plot a circle whose radius =20 units and center is (50, 30). (5)

Use midpoint circle drawing algorithm to plot a circle whose radius = 20 units and center (50, 30)

Solution

Calculate points w.r.to origin (0,0)

$x_0 = 0$

$y_0 = R$

Calculate initial decision parameter P_0 as

$P_0 = 1 - R$

$= 1 - 20 = -19$

$P_k < 0$

$x_{k+1} = x_k + 1$

$y_{k+1} = y_k$

$P_{k+1} = P_k + 2 \cdot x_{k+1} + 1$

So plot next point $x = 1$

$y = 20$

$P_{k+1} = -19 + 2 + 1 = -16$

(0, 20)

P_k	x_{k+1}, y_{k+1}	$2x_{k+1}$	$2y_{k+1}$
-19	(1, 20)	2	40
-16	(2, 20)	4	40
-11	(3, 20)	6	40
-4	(4, 20)	8	20
5	(5, 19)	10	38
-22	(6, 19)	12	38
-9	(7, 19)	14	38
6	(8, 18)	16	36
-13	(9, 18)	18	36
6	(10, 17)	20	34

(3) $P_{k+1} = -1$
$= -$
$= -$

(4) $P_k = -11 +$
$= -4$

(5) $P_k = -4 -$
$= 5$

(6) $P_k = 5 +$
$= -$

(7) $P_k = -22 -$
$= -9$

(8) $P_k = -9$
$= 6$

(9) $P_k = 6 +$
$= -13$

(10) $P_k = -13$
$= 6$

(contd....)

	p_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
(11)	−7	(11, 17)	22	34
(12)	16	(12, 16)	24	32
(13)	9	(13, 15)	26	30
(14)	6	(14, 14)	28	28

(11) $p_k = 6 + 20$
$= -7$

(12) $p_k = -7 + 2$
$= 16$

(13) $p_k = 16 + 24$
$= 9$

(14) $p_k = 9 + 26$
$= 6$

Calculate points w.r.to given centre (50, 30)

(50, 30)

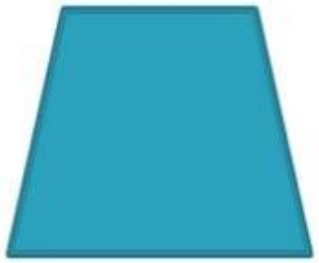
x_{k+1}, y_{k+1}	xplot, yplot
(0, 20)	(50, 50)
(1, 20)	(51, 50)
(2, 20)	(52, 50)
(3, 20)	(53, 50)
(4, 20)	(54, 50)
(5, 19)	(55, 49)
(6, 19)	(56, 49)
(7, 19)	(57, 49)
(8, 18)	(58, 48)
(9, 18)	(59, 48)
(10, 17)	(60, 47)
(11, 17)	(61, 47)
(12, 16)	(62, 46)
(13, 15)	(63, 45)
(14, 14)	(64, 44)

Draw a circle using Midpoint Circle Algorithm having radius as 10 and center of circle (100,100).

Midpoint Circle Algorithm (r=10,h,k=100)				
Step Number	**X**	**Y**	**d**	**Pixel**
1	0	10	-9	(x +h ,y +k) = 100,110
2	1	10	-9+2*1+3= – 4	101,110
3	2	10	-4+2*2+3=3	102,110
4	3	9	3+2*(3-9)+5 = -4	103,109
5	4	9	-4+2*4+3 = 7	104,109
6	5	8	7+2*(5-8)+5 = 6	105,108

AREA FILLING OR POLYGON FILLING ALGORITHMS

- Fill-Area algorithms are used to fill the interior of a polygonal shape.
- Many algorithms perform fill operations by first identifying the interior points, given the polygon boundary.



- Boundary fill algorithm
- Flood fill algorithm
- Scan line filling algorithm

Polygon Filling

- Filling the polygon means highlighting all the pixels which lie inside the polygon with any colour other than background colour.
- Polygons are easier to fill since they have linear boundaries. There are two basic approaches used to fill the polygon.
- One way to fill a polygon is to start from a given "seed", point known to be inside the polygon and highlight outward from this point i.e. neighbouring pixels until we encounter the boundary pixels.
- This approach is called seed fill because colour flows from the seed pixel until reaching the polygon boundary, like water flooding on the surface of the container
- Another approach to fill the polygon is to apply the inside test i.e. to check whether the pixel is inside the polygon or outside the polygon and then highlight pixels which lie inside the polygon". This approach is known as scan-line algorithm'

BOUNDARY FILL ALGORITHM

- Boundary Fill Algorithm starts at a pixel inside the polygon to be filled and paints the interior proceeding outwards towards the boundary.

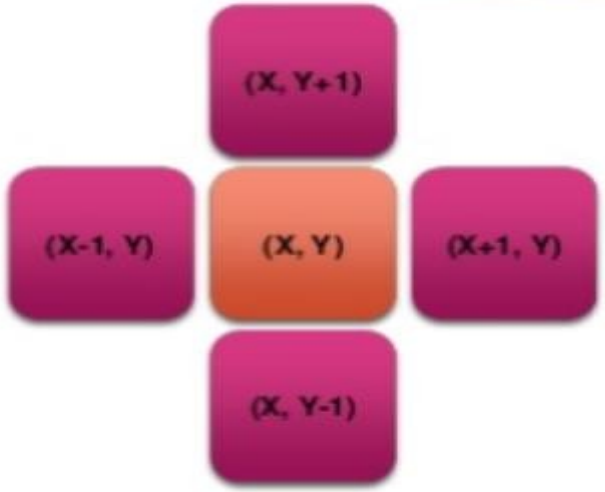
- This algorithm works **only if** the color with which the region has to be filled and the color of the boundary of the region are different.

- If the boundary is of one single color, this approach proceeds outwards pixel by pixel until it hits the boundary of the region.

- It takes an interior point(x, y), a fill color, and a boundary color as the input.
- The algorithm starts by checking the color of (x, y).
- If it's color is not equal to the fill color and the boundary color, then it is painted with the fill color and the function is called for all the neighbours of (x, y).
- If a point is found to be of fill color or of boundary color, the function does not call its neighbours and returns.
- This process continues until all points up to the boundary color for the region have been tested.
- The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

4-connected pixels

- After painting a pixel, the function is called for four neighboring points.
- These are the pixel positions that are right, left, above and below the current pixel.
- Areas filled by this method are called 4-connected

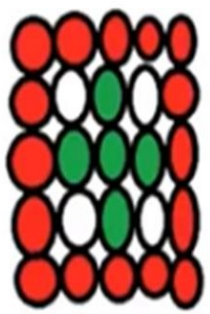


Boundary Fill Algorithm (Code)

```
void boundaryFill(int x, int y,
          int fillColor, int borderColor)
{
  getPixel(x, y, color);
  if ((color != borderColor)
            && (color != fillColor)) {
      setPixel(x,y);
      boundaryFill(x+1,y,fillColor,borderColor);
      boundaryFill(x-1,y,fillColor,borderColor);
      boundaryFill(x,y+1,fillColor,borderColor);
      boundaryFill(x,y-1,fillColor,borderColor);
  }
}
```

Problem in 4-connected

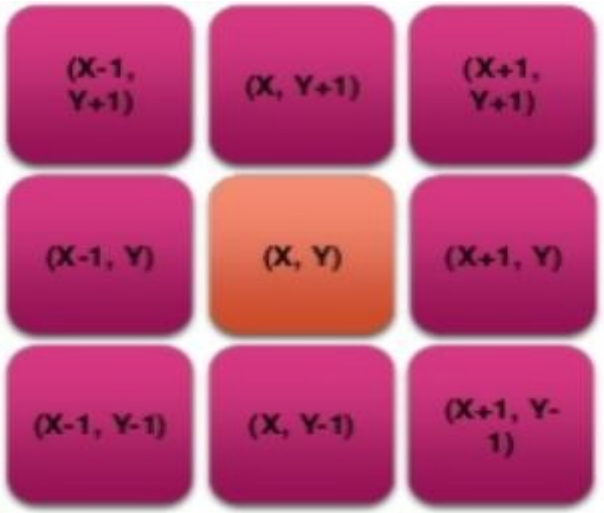
- There is a problem with this technique that 4-connected pixels technique cannot fill all the pixels.



Write the boundary fill algorithm for filling a polygon using eight connected approach. (4)

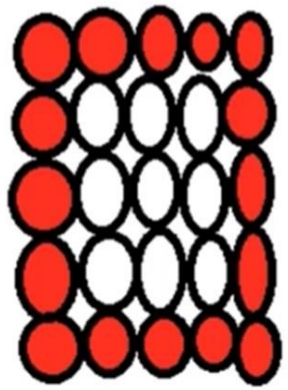
8-connected pixels

- More complex figures are filled using this approach.
- The pixels to be tested are the 8 neighboring pixels, the pixel on the right, left, above, below and the 4 diagonal pixels.
- Areas filled by this method are called 8-connected.

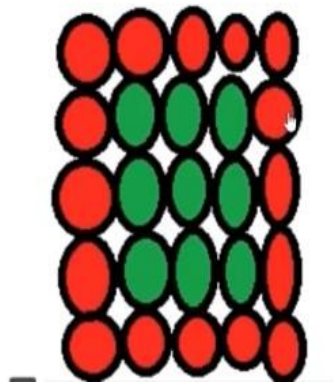


```
void boundaryFill8(int x, int y, int fill_color,int boundary_color)
{
    if(getpixel(x, y) != boundary_color &&
       getpixel(x, y) != fill_color)
    {
        putpixel(x, y, fill_color);
        boundaryFill8(x + 1, y, fill_color, boundary_color);
        boundaryFill8(x, y + 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y, fill_color, boundary_color);
        boundaryFill8(x, y - 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y - 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y + 1, fill_color, boundary_color);
        boundaryFill8(x + 1, y - 1, fill_color, boundary_color);
        boundaryFill8(x + 1, y + 1, fill_color, boundary_color);
    }
}
```

8-connected pixel filling



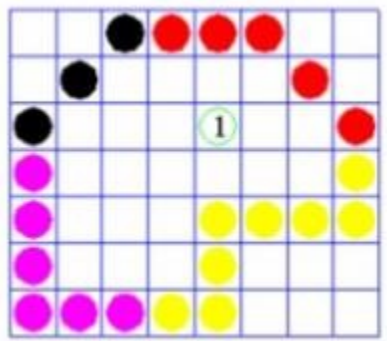
8-connected pixel filling



Write the flood fill algorithm for filling a polygon. (4)

Flood Fill Algorithm

- In this method, a point or seed which is inside region is selected.
- This point is called a seed point.
- Then four connected approaches or eight connected approaches is used to fill with specified color.
- This method is more suitable for filling multiple colors boundary.
- When boundary is of many colors and interior is to be filled with one color we use this algorithm.
- In fill algorithm, we start from a specified interior point (x, y) and reassign all pixel values are currently set to a given interior color with the desired color.
- Using either a 4-connected or 8-connected approaches, we then step through pixel positions until all interior points have been repainted.


1

```
Void floodFill4 (int x, int y, int fillColor, int oldColor)
{
    if (getpixel (x, y) == oldcolor)
    {
        setcolor (fillcolor);
        setpixel (x, y);
        floodFill4 (x+l, y, fillColor, oldColor);
        floodFill4 (x-1, y, fillColor, oldColor);
        floodFill4 (x, y+l, fillColor, oldColor);
        floodFill4 (x, y-1, fillColor, oldColor);
    }
}
```

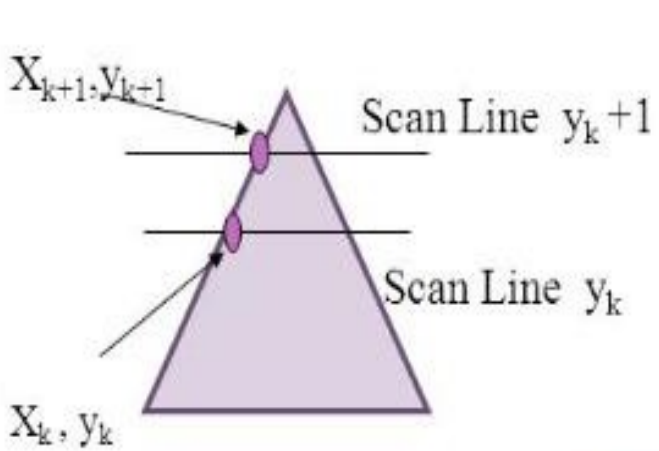
Flood Fill Algorithm	Boundary Fill Algorithm
Flood fill colors an entire area in an enclosed figure through interconnected pixels using a single color	Here area gets colored with pixels of a chosen color as boundary this giving the technique its name
So, Flood Fill is one in which all connected pixels of a selected color get replaced by a fill color.	Boundary Fill is very similar with the difference being the program stopping when a given color boundary is found.
A flood fill may use an unpredictable amount of memory to finish because it isn't known how many sub-fills will be spawned	Boundary fill is usually more complicated but it is a linear algorithm and doesn't require recursion
Time Consuming	It is less time Consuming

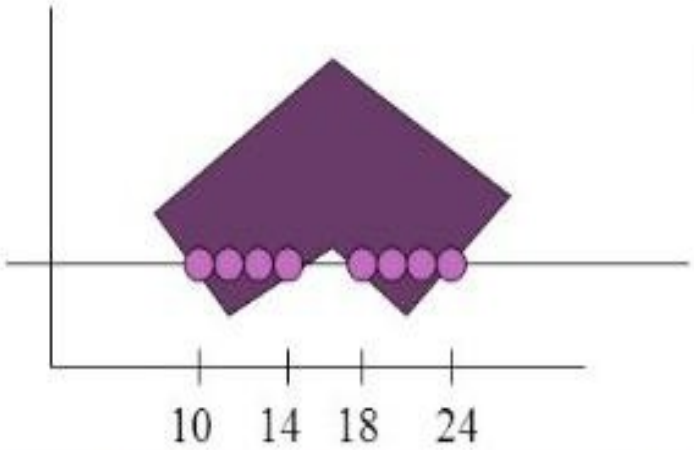
Write the scan line algorithm for filling a polygon. (4)

SCAN LINE POLYGON FILLING ALGORITHM

Steps

- Find the intersections of scan line with all edges of polygon
- Sort the intersection by increasing x co-ordinate ie,from left to right
- Make pairs of intersection and fill in color within all pixels inside the pair

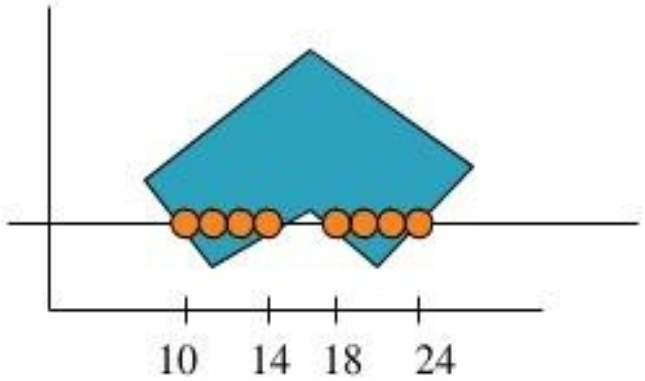




Interior pixels along a scan line passing through a polygon are

- For each scan line crossing a polygon are then sorted from left to right, and the corresponding frame buffer positions between each intersection pair are set to the specified color.
- These intersection points are then sorted from left to right , and the corresponding frame buffer positions between each intersection pair are set to specified color
- In the example, four pixel intersections define stretches from x=10 to x=14 and x=18 to x=24

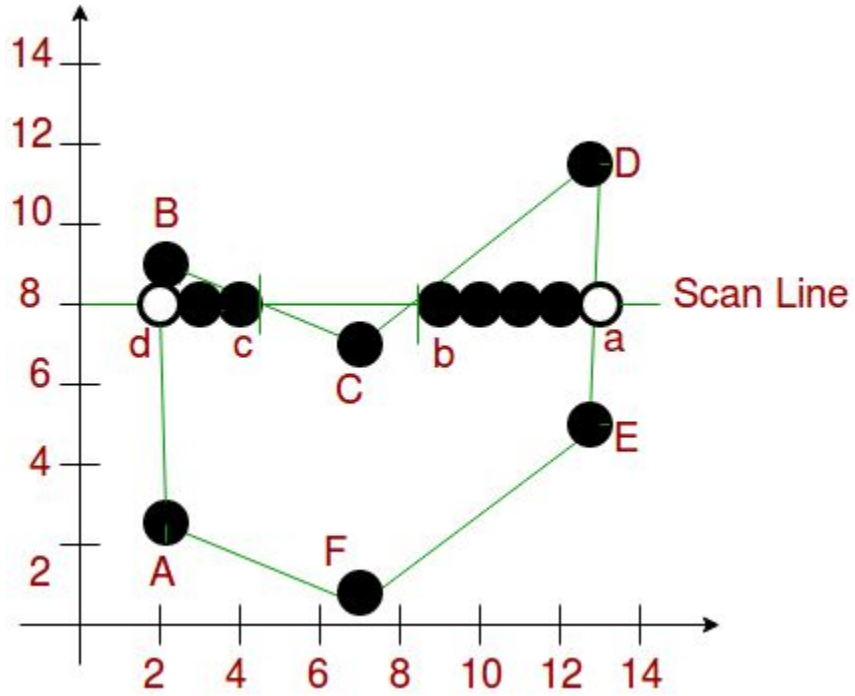
Scan Line Polygon Fill Algorithm



Interior pixels along a scan line passing through a polygon area

- For each scan line crossing a polygon ,the area filling algorithm locates the intersection points of the scan line with the polygon edges.
- These intersection points are then sorted from left to right , and the corresponding frame buffer positions between each intersection pair are set to specified fill color

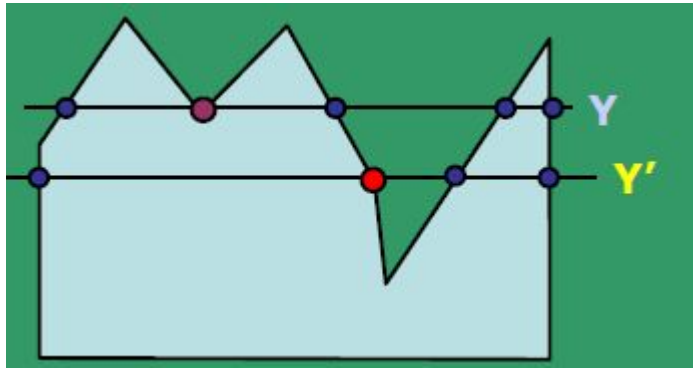
- This algorithm works by intersecting scan line with polygon edges and fills the polygon between pairs of intersections.



Special cases of polygon vertices:

- If both lines intersecting at the vertex are on the same side of the scan line, consider it as two points.
- If lines intersecting at the vertex are at opposite sides of the scan line, consider it as only one point.

- For Y, the edges at the vertex are on the same side of the scan line.
- Whereas for Y', the edges are on either/both sides of the vertex.
- In this case, we require additional processing.



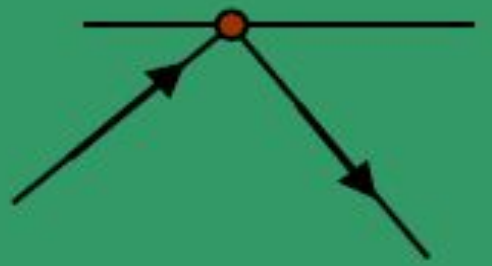
- Traverse along the polygon boundary clockwise (or counter- clockwise) and Observe the *relative change in Y-value* of the edges on either side of the vertex (i.e. as we move from
  one edge to another).

- Check the condition:

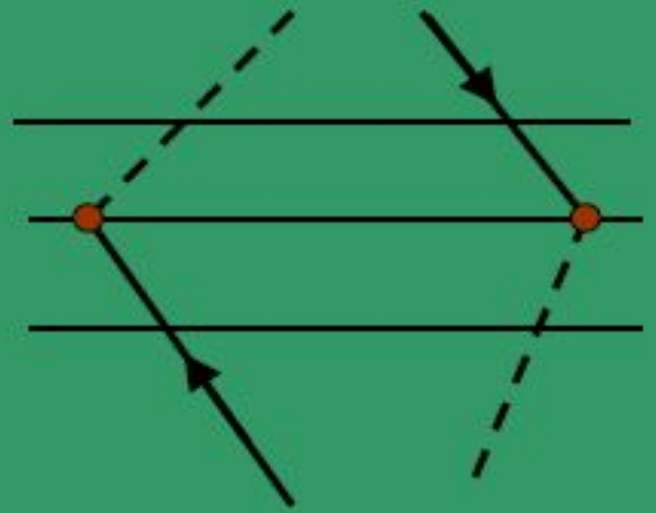
- If end-point Y values of two consecutive edges monotonically increase or decrease, count the middle vertex as a single intersection point for the scan line passing through it.

- Else the shared vertex represents a local maximum (or minimum) on the polygon boundary. Increment the intersection count.

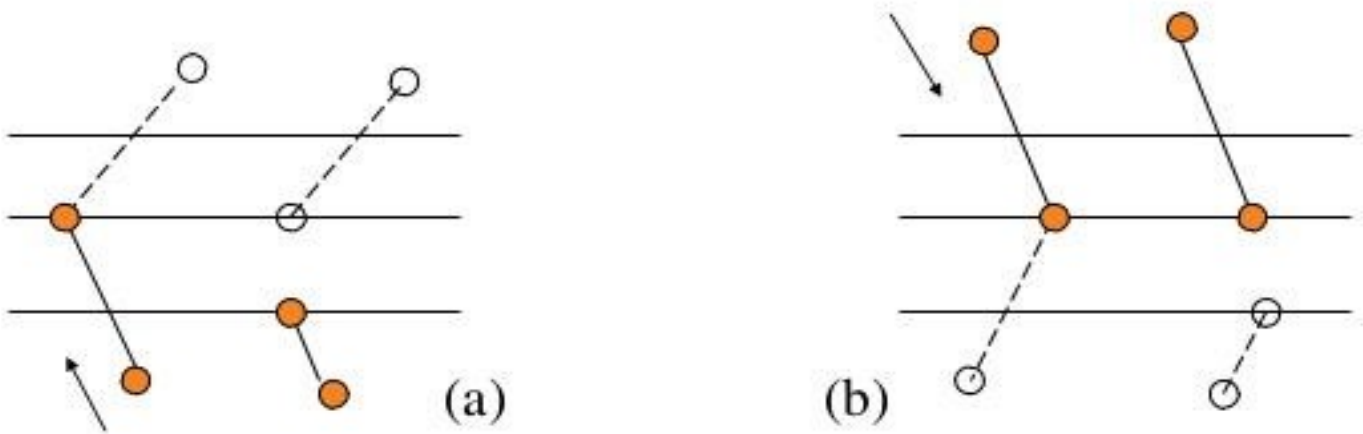
If the vertex is a ***local extrema***, consider (or add) two intersections for the scan line corresponding to such a shared vertex.



Must avoid this to happen in cases, such as:



Scan Line Polygon Fill Algorithm

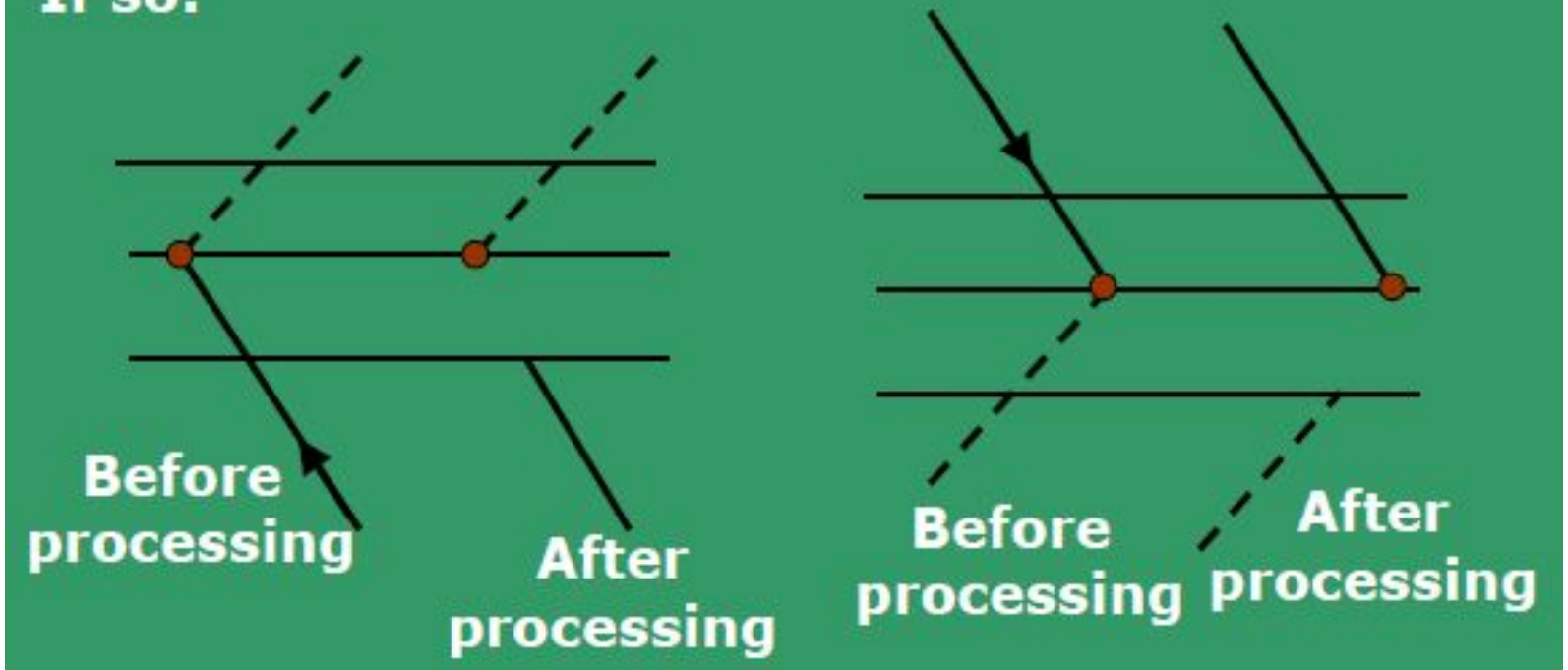


Adjusting endpoint values for a polygon, as we process edges in order around the polygon perimeter. The edge currently being processed is indicated as a solid like. In (a), the y coordinate of the upper endpoint of the current edge id decreased by 1. In (b), the y coordinate of the upper end point of the next edge is decreased by 1

To implement the above:

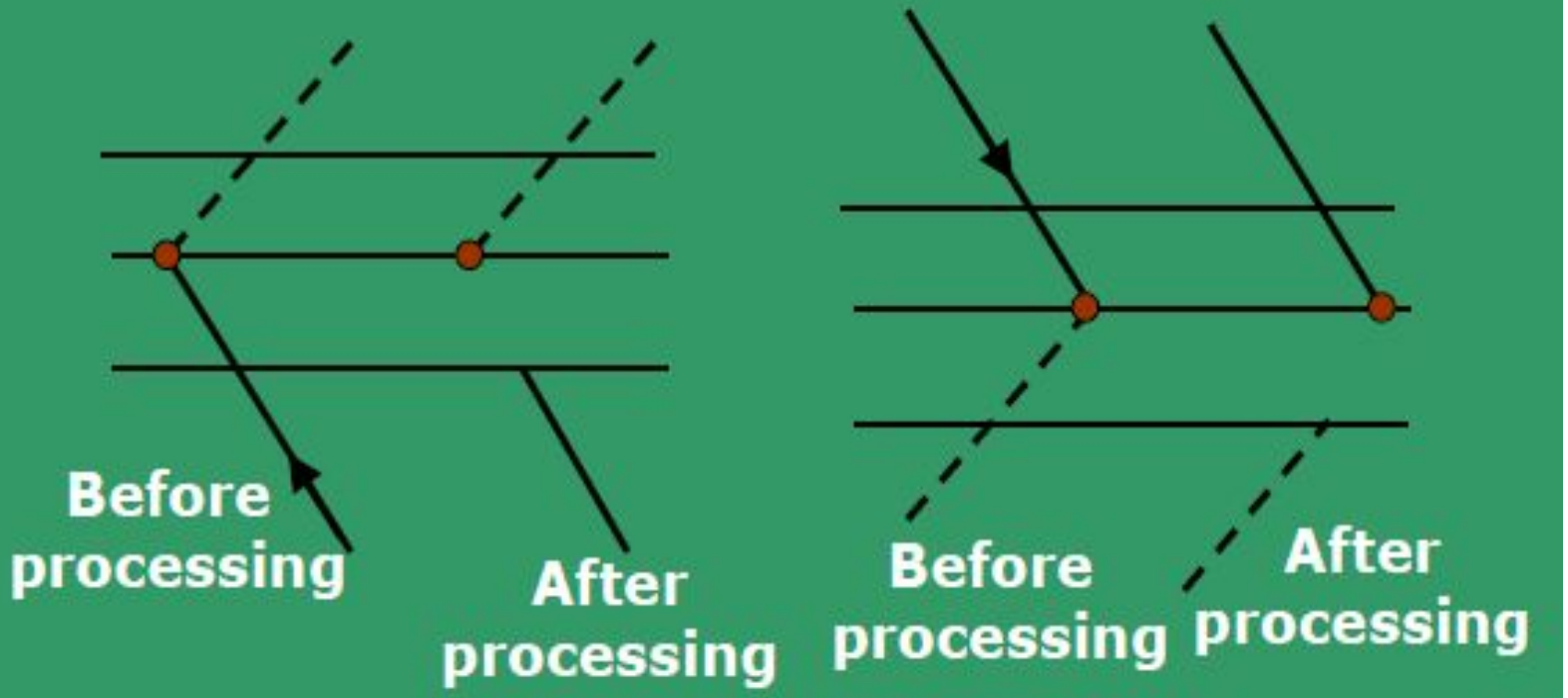
While processing non-horizontal edges (generally) along a polygon boundary in any order, check to determine the **condition of monotonically changing (increasing or decreasing) endpoint Y values.**

If so:



To implement the above:

Shorten the lower edge to ensure only one intersection point at the vertex.



Two important features of scanline-based polygon filling are:

- *scanline coherence* - values don't change much from one scanline to the next - the coverage (or visibility) of a face on one scanline typically differs little from the previous one.

- *edge coherence* - edges intersected by scanline "i" are typically intersected by scanline "i+1".

Data Structure Used (typical example):

SET (Sorted Edge table):

Contains all information necessary to process the scanlines efficiently.

SET is typically built using a bucket sort, with a many buckets as there are scan lines.

All edges are sorted by their Y_{min} coordinate, with a separate Y bucket for each scanline.

Within each bucket, edges sorted by increasing X of Y_{min} point.

Only non-horizontal edges are stored. Store these edges at the scanline position in the SET.

Edge structure
(sample record for each scanline):

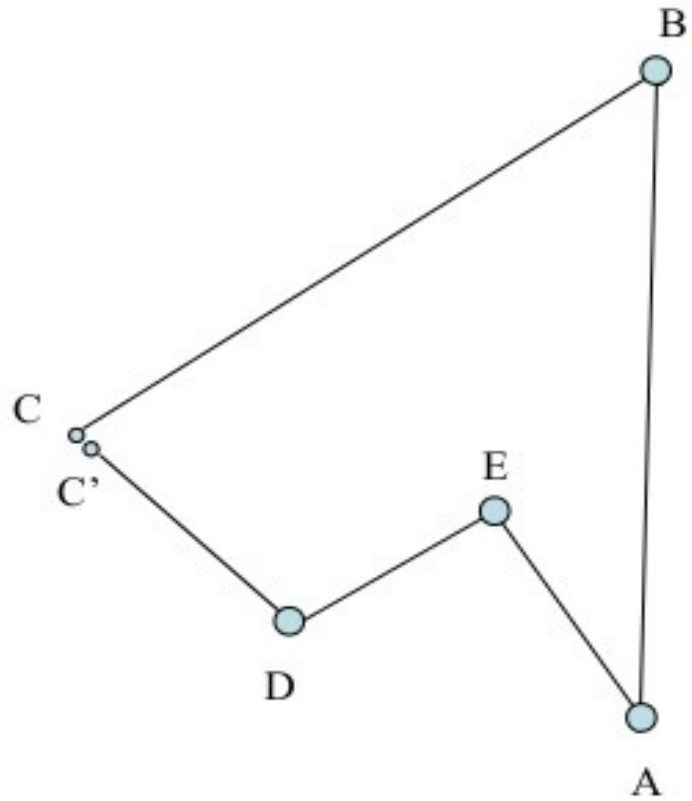
(Y_{max}, X_{min}, $\Delta X/\Delta Y$, pointer to next edge)

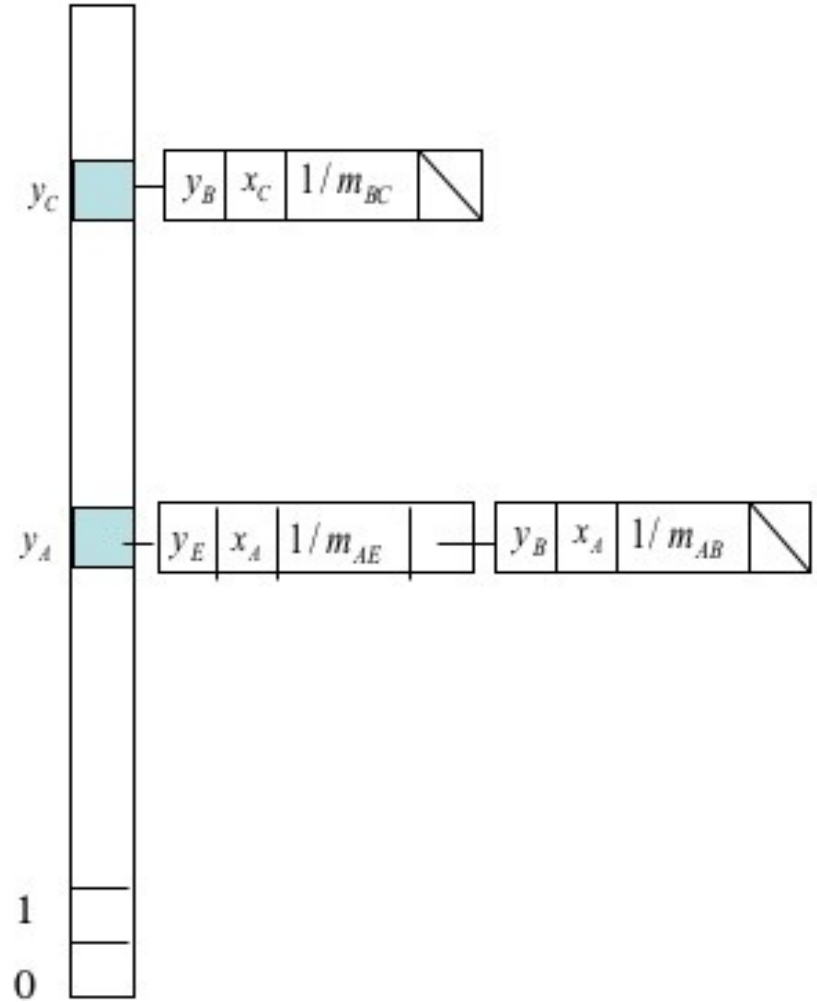
AEL (Active edge List):

Contains all edges crossed by a scanline at the current stage of iteration.

This is a list of edges that are active for this scanline, sorted by increasing X intersections.

Also called: Active Edge Table (AET).


B
C
C'
E
D
A

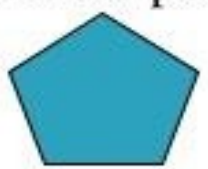

y_C
y_B x_C $1/m_{BC}$
y_A
y_E x_A $1/m_{AE}$
y_B x_A $1/m_{AB}$
1
0

Definition

- Polygon is a closed figure with many vertices and edges (line segments), and at each vertex exactly two edges meet and no edge crosses the other.

Types of Polygon:

- Regular polygons
  - No. of egdes equal to no. of angles.
- Convex polygons
  - Line generated by taking two points in the polygon must lie within the polygon.



- Concave polygons
  - Line generated by taking two points in the polygon may lie outside the polygon.

Polygon Inside/Outside Test

- This test is required to identify whether a point is inside or outside the polygon. This test is mostly used for identify the points in hollow polgons.

Two types of methods are there for this test:

I. Even-Odd Method,
II. Winding Number Method.

Even-Odd method:

1. Draw a line from any position P to a distant point outside the coordinate extents of the object and counting the number of edge crossings along the scan line.

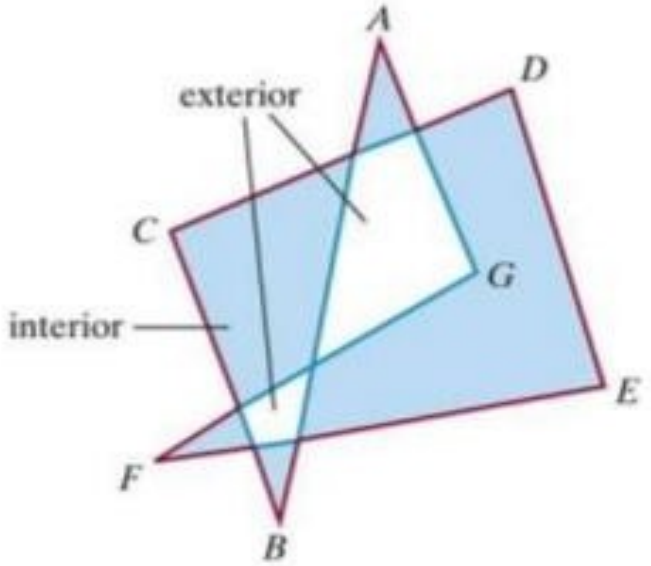
2. If the number of polygon edges crossed by this line is **odd** then

   P is an **interior** point.

   Else

   P is an **exterior** point

In some situations the logic described above doesn't hold, when a scan line passes through a vertex having two intersecting edges then if both the edges lies on the same side of the scan line the intersection is considered to be even.
Otherwise the intersection is considered to be odd i.e. if the two edges are on the opposite side of the scan line.



Odd-Even Rule

29