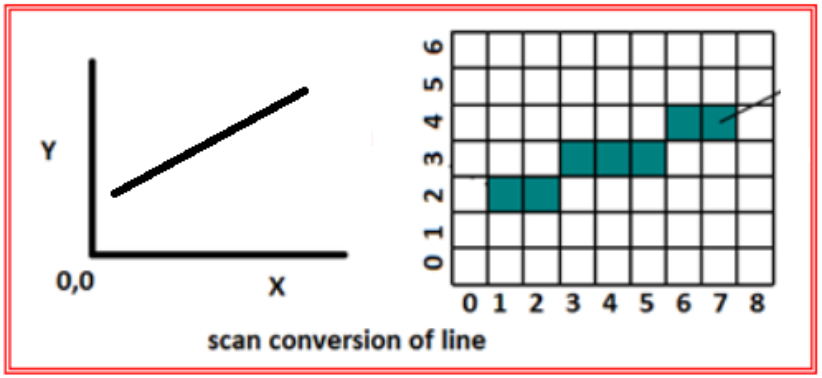
Chapter 2

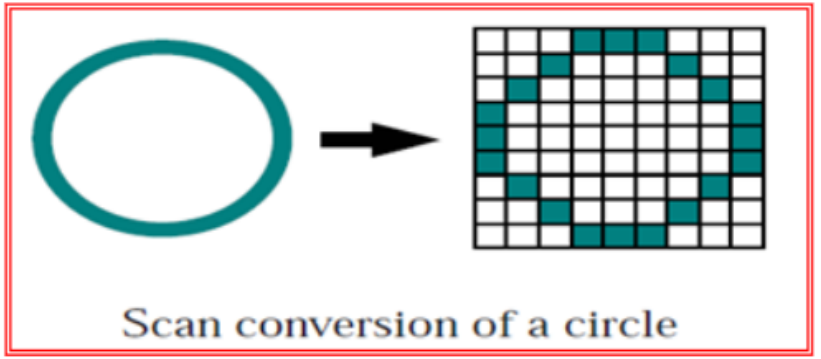
Raster Graphics and Algorithms

What is scan conversion?

- The process of representing continuous graphics objects as a collection of discrete pixels is called **scan conversion**.(Also called **rasterization**)

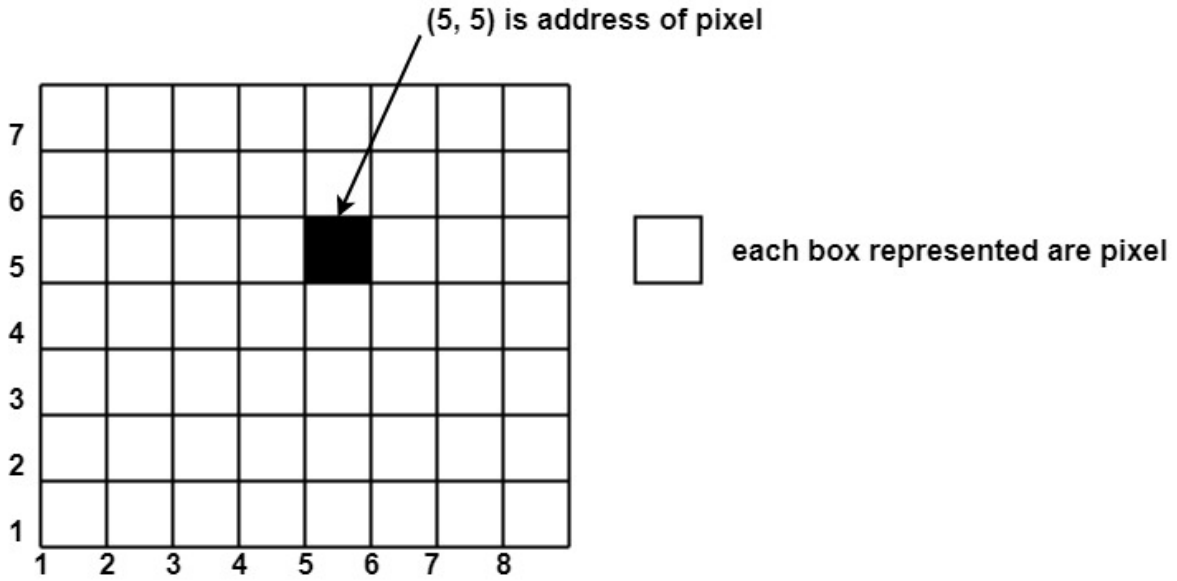


scan conversion of line


Scan conversion of a circle

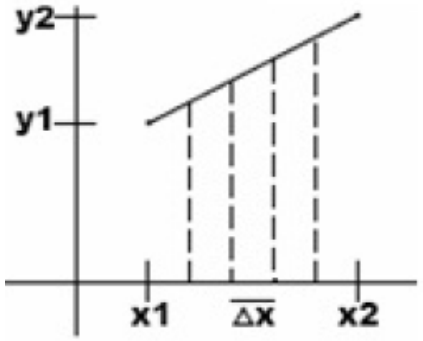
Scan Converting a point

- A point can in represented in a raster screen by just illuminating corresponding pixel.



Scan Converting a straight line

- On raster systems, lines are plotted with pixels, and step sizes in the horizontal and vertical directions are constrained by pixel separations.
- Idea: A line is sampled at unit intervals in one coordinate and the corresponding integer values nearest the line path are determined for the other coordinate.
- We have different algorithm for computation



DDA (Digital Differential Analyzer) Algorithm

- DDA algorithm is an incremental scan-conversion method. Such an approach is characterized by performing calculations at each step using results from the preceding step.
- A line is sampled at unit intervals in one coordinate and the corresponding integer values nearest the line path are determined for the other coordinate.
- Straight Line equations

Slope-Intercept Equation

$$y = m.x + b$$

Slope

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

Interval Calculation

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

DDA (Digital Differential Analyzer) Algorithm

- Main Calculation

$$x_{k+1} = x_k + \Delta x$$

$$y_{k+1} = y_k + \Delta y$$

- Four Different cases

Case 1: line with positive slope and magnitude less than 1 (*/m/<1*)

Case 2: line with positive slope and magnitude more than 1 (*/m/>1*)

Case 3: line with negative slope and magnitude less than 1 (*/m/<1*)

Case 4: line with negative slope and magnitude more than 1 (*/m/>1*)

DDA (Digital Differential Analyzer) Algorithm

- **Considering line with a positive slope:**

Case I: If slope (m) <=1 then sample at unit x intervals' (Δx=1) and compute each successive y value because the increment in x is more than increment in y.

So, set,

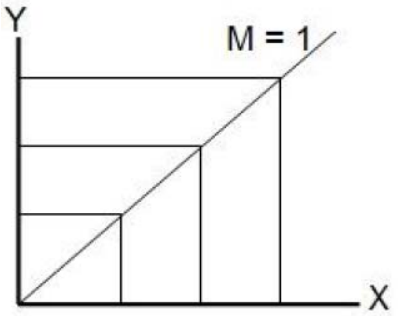
Δx=1 So, Δy=m

i.e.

$x_{k+1}=x_k+\Delta x=x_k+1$

$y_{k+1}=y_k+\Delta y=y_k+m$

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



Case II: If m > 1, then the increment in x (i.e. Δx) is smaller than increment in y (i.e. Δy),

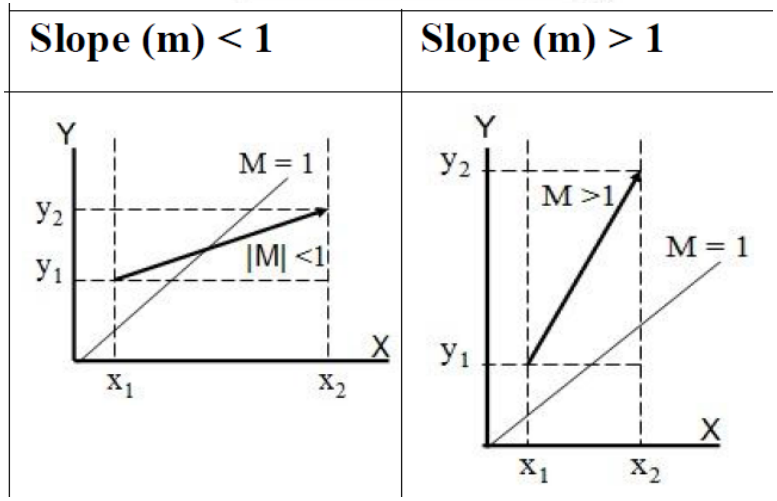
So set,

Δy=1 So, Δx=1/m

i.e.

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

$y_{k+1}=y_k+1$



DDA (Digital Differential Analyzer) Algorithm

- Here, in the both case m < = 1 and m>1, the algorithm based on the assumption that lines are to be processed from the left end point to the right end point. If this process is reversed, relation required to change in both case.

Case I: if $m<=1$, $\Delta x=-1$, so, $\Delta y=-m$

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

$y_{k+1}=y_k-m$

Case II: if $m>1$, $\Delta y=-1$ and $\Delta x=-1/m$

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

$y_{k+1}=y_k-1$

DDA (Digital Differential Analyzer) Algorithm

- **Considering line with a negative slope:**

Case I: If / m/< = 1 then assume start end point is the left then Δx=1 and Δy = m (m is negative).

i.e.

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

$y_{k+1}=y_k+m$

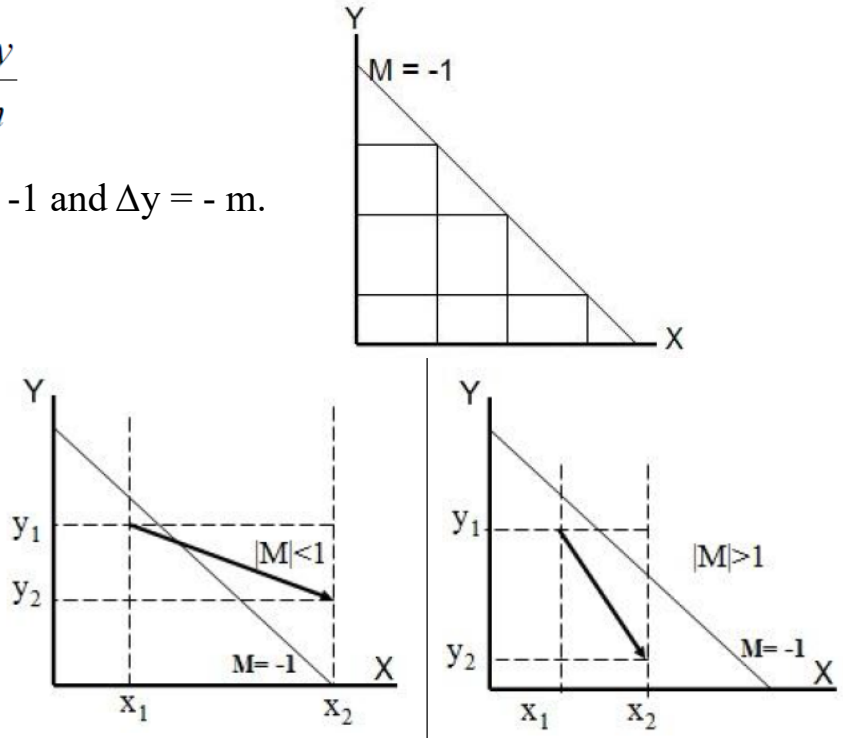
$$\Delta y = m.\Delta x \qquad \Delta x = \frac{\Delta y}{m}$$

If algorithm is required to proceed right to left then set on Δx= -1 and Δy = - m.

i.e.

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

$y_{k+1}=y_k-m$



DDA (Digital Differential Analyzer) Algorithm

- **Considering line with a negative slope:**

Case II: If / m/ > 1 then assume start end is at left and set $\Delta y = -1$ and $\Delta x = 1/m$ (m is negative)

i.e.

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

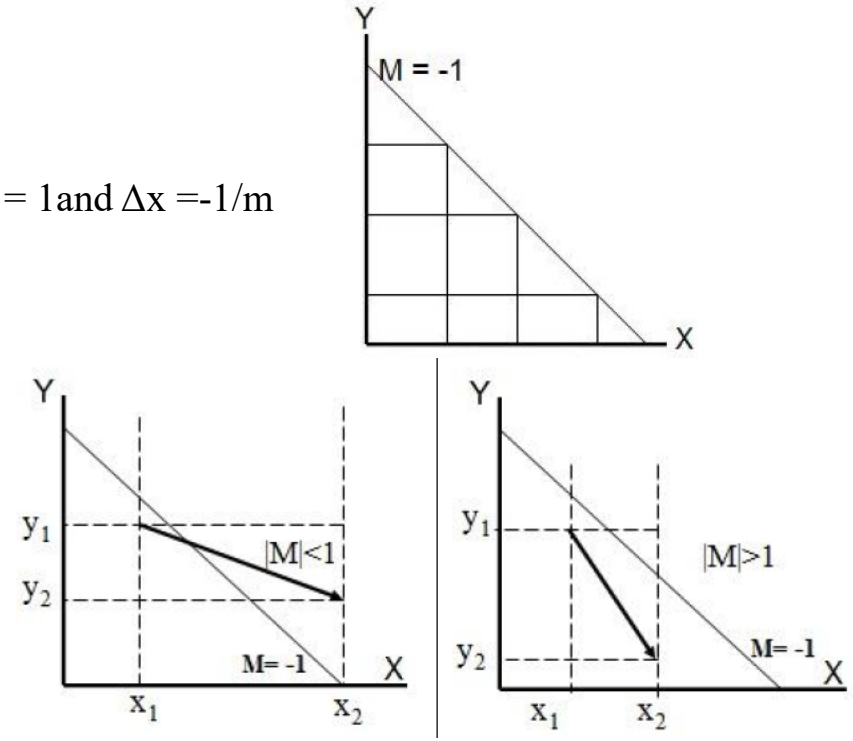
$y_{k+1}=y_k-1$

If the algorithm is required to proceed right to left then set $\Delta y = 1$and $\Delta x = -1/m$

i.e.

$x_{k+1}=x_k \quad +1/m$

$y_{k+1}=y_k+1$



DDA (Digital Differential Analyzer) Algorithm

1. Input points (x1,y1) and (x2,y2)
2. Compute dx = x2- x1 & dy= y2- y1
3. If |dx|>|dy| then steps=|dx|

   otherwise steps =|dy|
4. x_{inc} = dx/ steps

   y_{inc} = dy/ steps
5. x= x1 , y= y1 , k=0
6. do:

   plot(round (x), round (y));

   x=x+ x_{inc}

   y=y+ x_{inc}

   k ++

while (k<=steps)

DDA (Digital Differential Analyzer) Algorithm

Advantages

- Faster method than simple line drawing algorithm for calculating pixel position as it eliminates multiplication.
- Avoids directly deal of equation of st. line $y = mx+c$.

Disadvantages

- 'm' is usually stored in floating point number.
- There could be round off error.
- The line will move away from the true line path, especially when it is long due to successive round of error.
- Accumulation of round off error in successive additions of floating point increment.

DDA (Digital Differential Analyzer) Algorithm

Example-1: Digitize the line with end points (2, 1) and (8, 3) using DDA.

Solution

Here,

Starting point of line = (x1, y1) = (2, 1) and

Ending point of line = (x2, y2) = (8, 3)

Thus, slope of line, m = Δy / Δx = *y2-y1 / x2-x1* = (3-1) / (8-2) = 1/3 =0.3333

As the given points, it is clear that the line is moving left to right with the slope

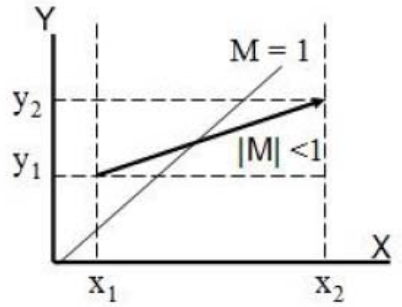
m = 1/3 <1

Thus,

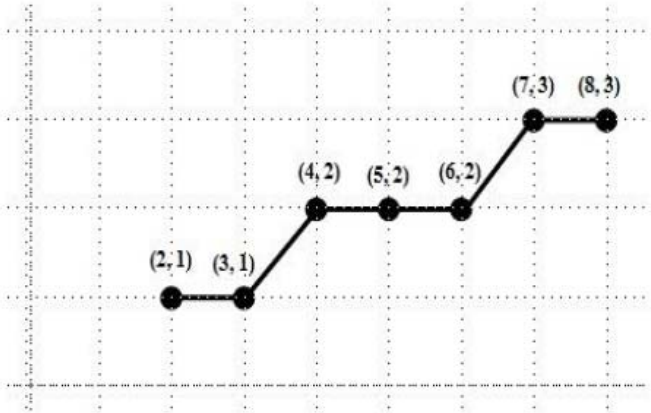
Δx=1
Δy=m

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

$$y_{k+1} = y_k + m$$



X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1})
2	1	(2,1)
2+1 = 3	1+0.3333 = 1.333 ≈ 1	(3, 1)
3+1 = 4	1+2 *0.3333 = 1.666 ≈ 2	(4, 2)
4+1 = 5	1+3*0.3333 = 1.999 ≈ 2	(5, 2)
5+1 = 6	1+4*0.3333 = 2.333 ≈ 2	(6, 2)
6+1 = 7	1+5*0.3333 = 2.666 ≈ 3	(7, 3)
7+1 = 8	1+6*0.3333 = 2.999 ≈ 3	(8, 3)



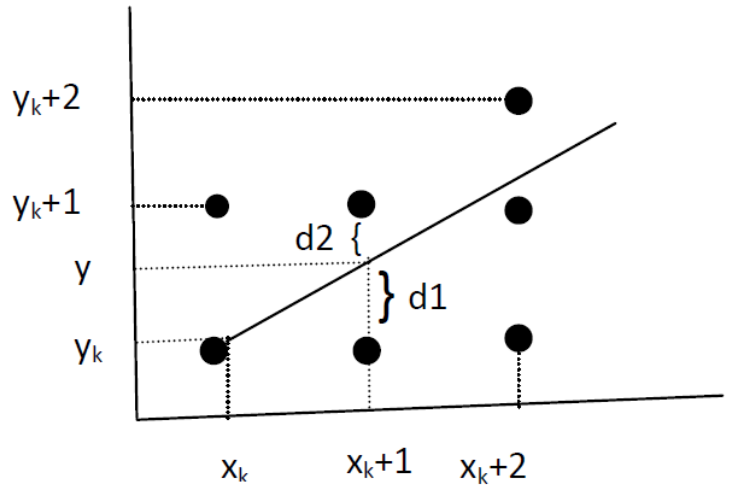
Bresenham's Line Algorithm

- Accurate and efficient line drawing algorithm.
- Brenham algorithm use only integer arithmetic to find the next position to be plot.
- It avoids incremental error.
- The major concept of Bresenham algorithm is to determine the nearest pixel position.

Bresenham's Line Algorithm

- Pixel positions are determined by sampling at unit x intervals.
- - Starting from left end position (x_0, y_0) of a given line, we step to each successive column (x-position) and plot the pixel whose scan line y value is closed to the line path.
- - Assuming the pixel at (x_k, y_k) to be displayed is determined, we next to decide which pixel to plot in column x_k +1, our choices are the pixels at positions.

(x_k + 1, y_k) and (x_k + 1, y_k + 1)



Bresenham's Line Algorithm

- At sampling position $x_k + 1$, we label vertical pixel separations from the mathematical line path d1 and d2.
- The y-co-ordinate on the mathematical at pixel column position $x_k + 1$ is calculated as,

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

Then,

$$d1=y-y_k =m(x_k+1)+b-y_k.....(2)$$

and

$$d2=(y_k+1)-y = y_k+1-m(x_k+1)-b.....(3)$$

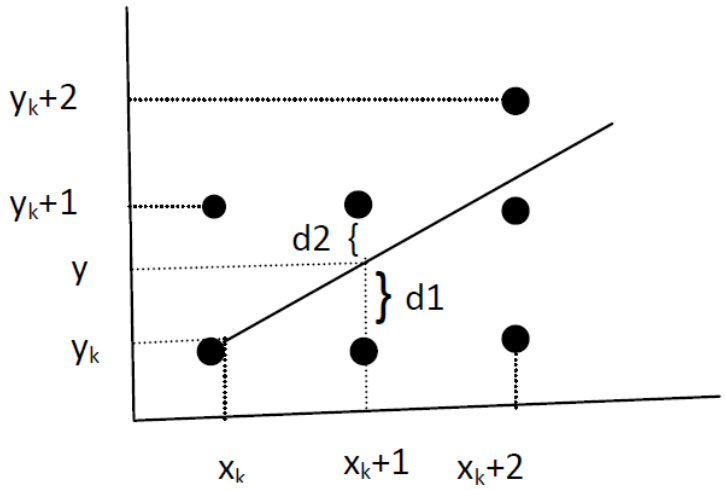
now,

$$d1-d2=m(x_k+1)+b-y_k-y_k-1+m(x_k+1)+b$$

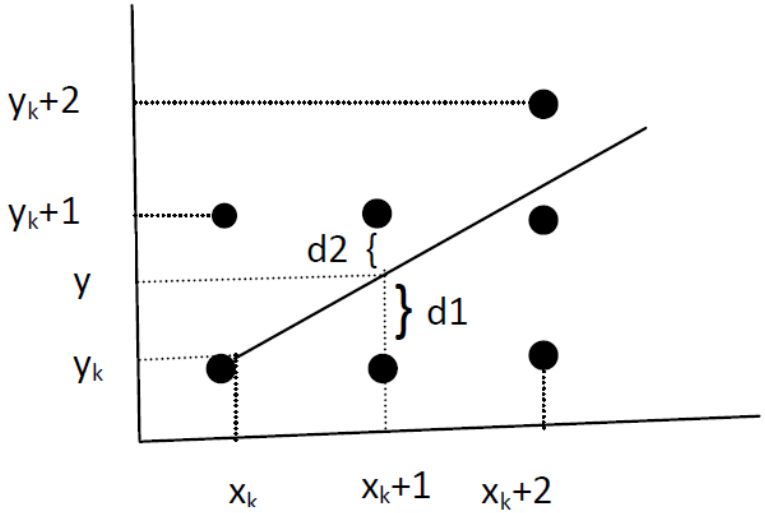
$$=2*m(x_k+1)-2y_k+2b-1$$

$$=2\Delta y/\Delta x(x_k+1)-2y_k+2b-1 \text{ [since } m=\Delta y/\Delta x]$$

$$\Delta x(d1-d2)= 2\Delta y(x_k+1)-2 \Delta xy_k+2\Delta xb-\Delta x$$



Bresenham's Line Algorithm



Defining decision parameter $p_k=\Delta x(d1-d2)$

$p_k=2\Delta y(x_k+1)-2\Delta xy_k+2\Delta xb-\Delta x$

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

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

$=2\Delta yx_k-2\Delta xy_k+c$.....(4)

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

- Case I:

If $p_k>=0$ then $d2 < d1$ which implies that $y_k + 1$ is nearer than y_k. So pixel at $(y_k + 1)$ is better to choose which reduce error than pixel at y_k. This determine next pixel co-ordinate to plot is $(x_k + 1, y_k + 1)$

- Case II:

If $p_k< 0$ then $d1< d2$ which implies pixel at y_k is nearer than pixel at $(y_k + 1)$. So pixel at y_k is better to choose which reduce error than pixel at $(y_k + 1)$. This determine next pixel co-ordinate to plot is $(x_k + 1, y_k)$

Bresenham's Line Algorithm

Now,

Similarly, pixel as $(x_k + 2)$ can be determine whether it is $(x_k + 2, y_{k+1})$ or $(x_k + 2, y_{k+2})$ by looking the sign of deciding parameter p_{k+1} assuming pixel as $(x_k + 1)$ is known.

$p_{k+1} = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + c$ where c is same as in p_k

Now,

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

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

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

$= 2\Delta y - 2\Delta x(y_{k+1} - yk)$

Bresenham's Line Algorithm

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

This implies that decision parameter for the current column can be determined if the decision parameter of the last column is known.

Here $(y_{k+1} - y_k)$ could either 0 or 1 which depends on sign of p_k

If $p_k >= 0$ (i.e.d2< d1), $y_{k+1} = y_k + 1$ which implies $(y_{k+1} - y_k) = 1$

That is at $p_k >= 0$ the pixel to plot is $(x_k + 1, y_k + 1)$ and

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

If $p_k <0$(d1<d2), y_{k+1}=yk which implies ($y_{k+1}-y_k=0$)

That is at $p_k<0$ then pixel to be plotted is (x_k+1,y_k) and

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

Bresenham's Line Algorithm

- **Initial decision parameter (p_0)**

  $p_k=2\Delta yx_k-2\Delta xy_k+2\Delta y+\Delta x(2b-1)$

  $p_0=2\Delta yx_0-2\Delta xy_0+2\Delta y+\Delta x(2b-1)$

  We have $b=y_0-mx_0=y_0-\Delta y/\Delta x(x_0)$

  $=2\Delta yx_0-2\Delta xy_0+2\Delta y+2\Delta xy_0-2\Delta yx_0-\Delta x$

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

Bresenham's Line Algorithm

Step 1. Start

Step 2. Declare variables x_1,y_1,x_2,y_2,lx,ly, Δx, Δy,p0, p_k, p_{k+1}.

Step 3. Read Values of x_1, y_1,x_2,y_2.

Step 4. Calculate Δx=absolute(x_2-x_1)

Δy=absolute(y_2-y_1)

Step 5. if (x_2>x_1)

assign lx=1

else

assign lx=-1

Step 6. if (y_2>y_1)

assign ly=1

else

assign ly=-1

Bresenham's Line Algorithm

Step 7. Plot (x_1, y_1)

Step 8. if $\Delta x > \Delta y$(i.e. /m/<1)

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

starting at k=0 to Δx times, repeat

if $(p_k < 0)$

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

$y_{k+1} = y_k$

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

else

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

$y_{k+1} = y_k + ly$

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

plot(x_{k+1}, y_{k+1})

Bresenham's Line Algorithm

else

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

starting at k=0 to Δy times, repeat

if($p_k<0$)

$x_{k+1}=x_k$

$y_{k+1}=y_k+ly$

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

else

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

$y_{k+1}=y_k+ly$

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

Plot(x_{k+1},y_{k+1})

Step 9. Stop

Bresenham's Line Algorithm

Difference between DDA and BLA (Advantage of BLA over DDA)

- In DDA algorithm each successive point is computed in floating point, so it requires more time and more memory space. While in BLA each successive point is calculated in integer value. So it requires less time and less memory space.
- In DDA, since the calculated point value is floating point number, it should be rounded at the end of calculation but in BLA it does not need to round, so there is no accumulation of rounding error.
- Due to rounding error, the line drawn by DDA algorithm is not accurate, while in BLA line is accurate.
- DDA algorithm can not be used in other application except line drawing, but BLA can be implemented in other application such as circle, ellipse and other curves.

Bresenham's Line Algorithm

Example-1: Digitize the line with end points (20, 10) and (30, 18) using BLA.

Solution

Here, Starting point of line = (x1, y1) = (20, 10) and

Ending point of line = (x2, y2) = (30, 18)

Thus, slope of line, $m = \Delta y / \Delta x = y2\text{-}y1 / x2\text{-}x1 = (18\text{-}10) / (30\text{-}20) = 8/10$

As the given points, it is clear that the line is moving left to right with the positive slope,

$|m| = 0.8 < 1$

Thus,

The initial decision parameter $(P_0) = 2\Delta y - \Delta x = 2*8 - 10 = 6$

Since, for the Bresenham's Line drawing Algorithm of slope, $|m| \leq 1$, we have

- If $P_k > 0$,
  $P_{k+1} = P_k + 2\Delta y - 2\Delta x$, and $y_{k+1} = y_k + 1$ & $x_{k+1} = x_k + 1$
- Else
  If $P_k < 0$,
  $P_{k+1} = P_k + 2\Delta y$, and $y_{k+1} = y_k$ & $x_{k+1} = x_k + 1$

Bresenham's Line Algorithm

Thus,

k	P_k	X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1})
0.	6	20+1 = 21	11	(21, 11)
1.	6 + 2*8 -2*10 = 2	21+1 = 22	12	(22, 12)
2.	2 + 2*8 -2*10 = -2	22+1 = 23	12	(23, 12)
3.	-2 + 2*8 = 14	23+1 = 24	13	(24, 13)
4.	14 + 2*8 -2*10 = 10	24+1 = 25	14	(25, 14)
5.	10 + 2*8 -2*10 = 6	25+1 = 26	15	(26, 15)
6.	6 + 2*8 -2*10 = 2	26+1 = 27	16	(27, 16)
7.	2 + 2*8 -2*10 = -2	27+1 = 28	16	(28, 16)
8.	-2 + 2*8 = 14	28+1 = 29	17	(29, 17)
9.	14 + 2*8 -2*10 = 10	29+1 = 30	18	(30, 18)

Bresenham's Line Algorithm

Conclusion: Bresenham's Line drawing Algorithm for slope, $|m| \leq 1$

a) Calculate the starting value of the decision parameter: $P_0 = 2\Delta y - \Delta x$.

b) At each x_k along the line, starting at $k = 0$, perform the following test.

- If $P_k > 0$,
  $P_{k+1} = P_k + 2\Delta y - 2\Delta x$, and $y_{k+1} = y_k + 1$ & $x_{k+1} = x_k + 1$
- Else If $P_k < 0$,
  $P_{k+1} = P_k + 2\Delta y$, and $y_{k+1} = y_k$ & $x_{k+1} = x_k + 1$

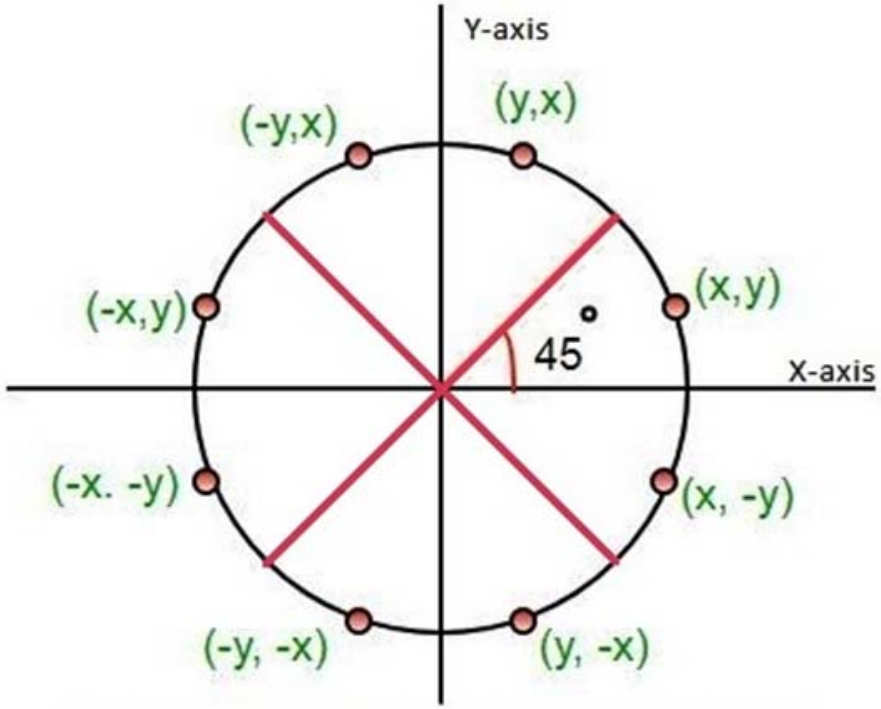
Scan Converting a circle

- The equation of circle in Cartesian form

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

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

- We sample at unit intervals and determine the closest pixel position to the specified circle at each steps.
- Non uniform spacing of plotted pixel is a problem
- To solve the computational complexity, we use symmetry of circle i.e. Calculate for one octant and use symmetry for others.
- Bresenham's line algorithm for raster displays is adapted for circle generation by setting up decision parameters for finding the closest pixel to the circumference at each sampling step.



For each pixel (x,y) all possible pixels in 8 octants

- We test the halfway position between two pixels to determine if this midpoint is inside or outside the circle boundary.

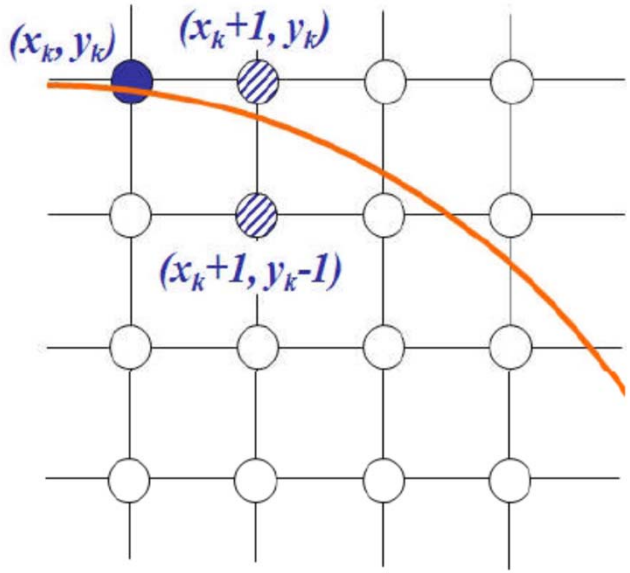
Midpoint Circle Algorithm

- The equation of circle with center (0, 0) is $x^2 + y^2 = r^2$
- With center (0,0) and radius r, function of circle $f_{circle}(x, y) = x^2 + y^2 - r^2$
- We know,

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

- Assume (x_k, y_k) is plotted, then next point close to the circle is

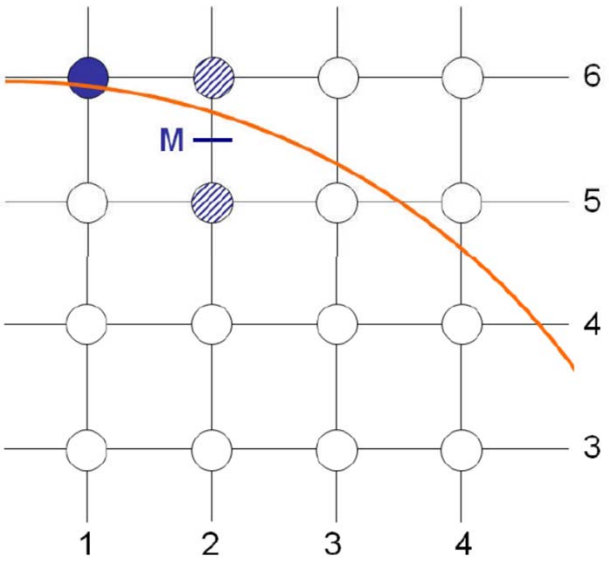
  (x_{k+1}, y_k) or (x_{k+1}, y_{k-1})
- Decision parameter is the circle function evaluated at the mid point between these two points.

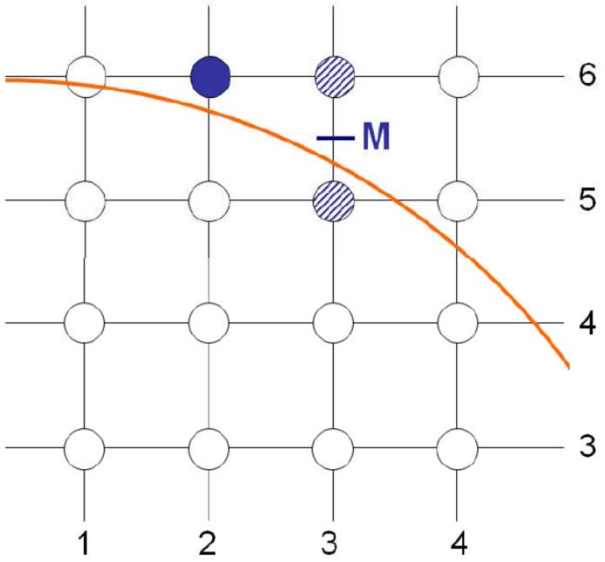


Midpoint Circle Algorithm

- $P_k = f_{circle}(x_k+1, y_k - ½)$
- $P_k = (x_k+1)^2 + (y_k - ½)^2 - r^2$
- So, if $P_k<0$ then midpoint is inside the circle and y_k is closer to circle boundary.

  else, midpoint is outside the circle. And y_k-1 is closer to the circle boundary





Midpoint Circle Algorithm

- successive decision parameter are obtaining using incremental calculations
- ie. Next decision parameter is obtained at

$x_{k+1}+1$ and $y_{k+1}- ½$ (mid point of y_{k+1} and $y_{k+1}-1$)

$P_{k+1} = f_{circle}(x_{k+1}+1, y_{k+1} – ½)$

$= (x_{k+1}+1)^2 +(y_{k+1} - ½)^2 – r^2$

$= (x_k+1+1)^2 +(y_{k+1} - ½)^2 – r^2$ because $x_{k+1} = x_k+1$

Now,

$P_{k+1} – P_k = (x_k+1+1)^2 +(y_{k+1} - ½)^2 – r^2 – (x_k+1)^2 - (y_k – ½)^2 + r^2$

$= (x_k+1)^2 +2(x_k+1) +1 + y_{k+1}^2 – y_{k+1} + ¼ - r^2 - (x_k+1)^2 – y_k^2 + y_k - ¼ + r^2$

$= 2(x_k+1) + (y_{k+1}^2 – y_k^2) – (y_{k+1}- y_k) +1$

Midpoint Circle Algorithm

$P_{k+1} - P_k = 2(x_k+1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$

where y_{k+1} is either y_k or y_k-1 depending on sign of P_k

- If $P_k<0$ then next pixel is at (x_k+1, y_k)

  $P_{k+1}=P_k+2(x_k+1) +1$

- If $P_k>=0$ then next pixel is at (x_k+1, y_k-1)

  $P_{k+1}=P_k+2(x_k+1) + [(y_k-1)^2 - y_k^2] - (y_k-1- y_k) +1$

  $= P_k+2(x_k+1) + (y_k^2 - 2y_k +1 - y_k^2) +1 +1$

  $= P_k+2(x_k+1) - 2y_k + 2 +1$

  $= P_k+2(x_k+1) - 2(y_k - 1) +1$

  $= P_k+2x_{k+1} - 2y_{k+1} +1$ because $x_{k+1} = x_k+1$ and $y_{k+1} = y_k-1$

Midpoint Circle Algorithm

- **Initial decision parameter**

The initial decision parameter is obtained by evaluating the circle function at starting point (x0, y0)= (0,r)

$P_0 = 1 + (r - ½)^2 - r^2$ because $P_k = (x_k+1)^2 + (y_k - ½)^2 - r^2$

$= 1 + r^2 - r + ¼ - r^2$

$P_0 = 5/4 - r$

If the radius 'r' is specified as an integer, we can simply round to $P_0=1-r$

Midpoint Circle Algorithm

Conclusion

1. Calculate the initial decision parameter (P_0) $= 1 - r$
2. If $P < 0$

Plot $(x_k + 1, y_k)$

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

Else ($P \geq 0$)

Plot $(x_k + 1, y_k - 1)$

$P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1$

Note : If the center of the circle is not at origin then first calculate the octant points of the circle in the same way as the center at origin & then add the given circle center on each calculated pixels.

Midpoint Circle Algorithm

Step 1. Start

Step 2. Declare variables x_c, y_c, r, x_0, y_0, P_0, P_k, P_{k+1}.

Step 3. Read Values of x_c, y_c, r.

Step 4. Initialize the x_0 and y_0 i. e. set the co-ordinates for the first point on the circumference of the circle centered at origin as.

$x_0=0$

$y_0=r$

Step 5. Calculate initial value of decision parameter

$P_0 = 5/4 - r$

Step 6. At each x_k position, starting from k=0

If $P_k<0$

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

$y_{k+1}= y_k$

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

Midpoint Circle Algorithm

else

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

$y_{k+1}= y_k-1$

$P_{k+1}=P_k+2x_{k+1}-2y_{k+1}+1$

Step 7. Determine the symmetry in other seven octants.

Step 8. Move each calculated pixel position (x,y) onto the circular path centered on (x_c, y_c)

Step 9. Plot the co-ordinates values

$x=x+x_c$

$y=y+y_c$

Step 10. Repeat steps 6 to 9 until $x \geq y$

Step 11. Stop

Midpoint Circle Algorithm

Example: Digitize a circle with radius 9 and center at (6, 7).

Solution

Here, the initial decision parameter (P_0)

$P_0 = 1 - r = 1 - 9 = -8$

Since, for the Midpoint Circle Algorithm of starting point (0, r) & centre at origin (0, 0) rotating at clockwise direction, we have

If $P < 0$

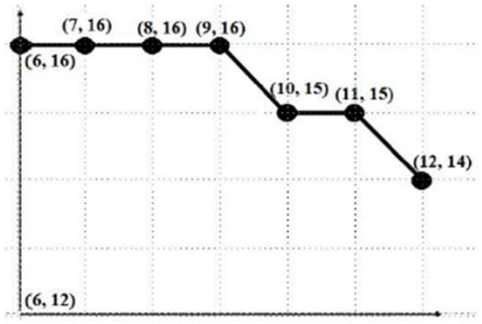
Plot $(x_k + 1, y_k)$

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

Else ($P \geq 0$)

Plot $(x_k + 1, y_k - 1)$

$P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1$

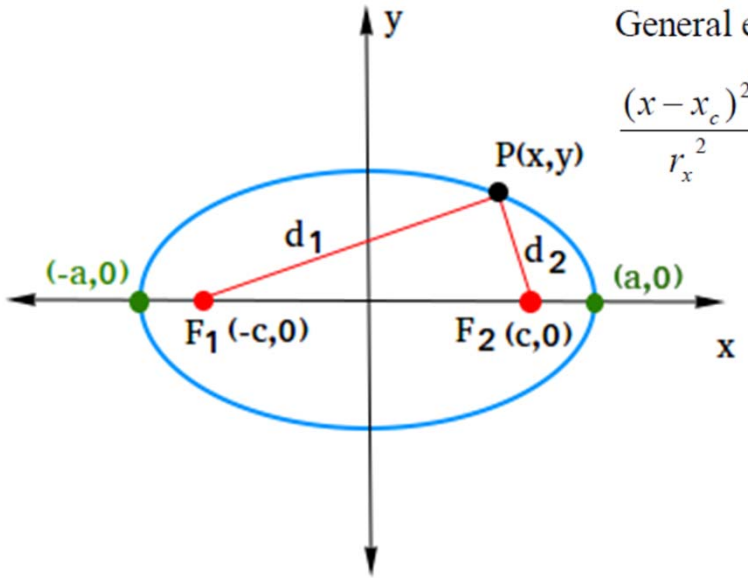


Thus,

k	P_k	X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1}) At (0, 0)	(X_{k+1}, Y_{k+1}) At (6, 7)
0.	-8	0+1 = 1	9	(1, 9)	(1+6, 9+7) = (7, 16)
1.	-8 + 2*1 +1= -5	1+1 = 2	9	(2, 9)	(2+6, 9+7) = (8, 16)
2.	-5 + 2*2 +1= 0	2+1 = 3	8	(3, 8)	(3+6, 8+7) = (9, 15)
3.	0 + 2*3 - 2*8 +1= -9	3+1 = 4	8	(4, 8)	(4+6, 8+7) = (10, 15)
4.	-9 + 2*4 +1= 0	4+1 = 5	7	(5, 7)	(5+6, 7+7) = (11, 14)
5.	0 + 2*5 - 2*7 +1= -3	5+1 = 6	7	(6, 7) (Halt)	(6+6, 7+7) = (12, 14)
6.	-3 + 2*6 +1= 10	6+1 = 7	6	(7, 6) (Only for checking)	

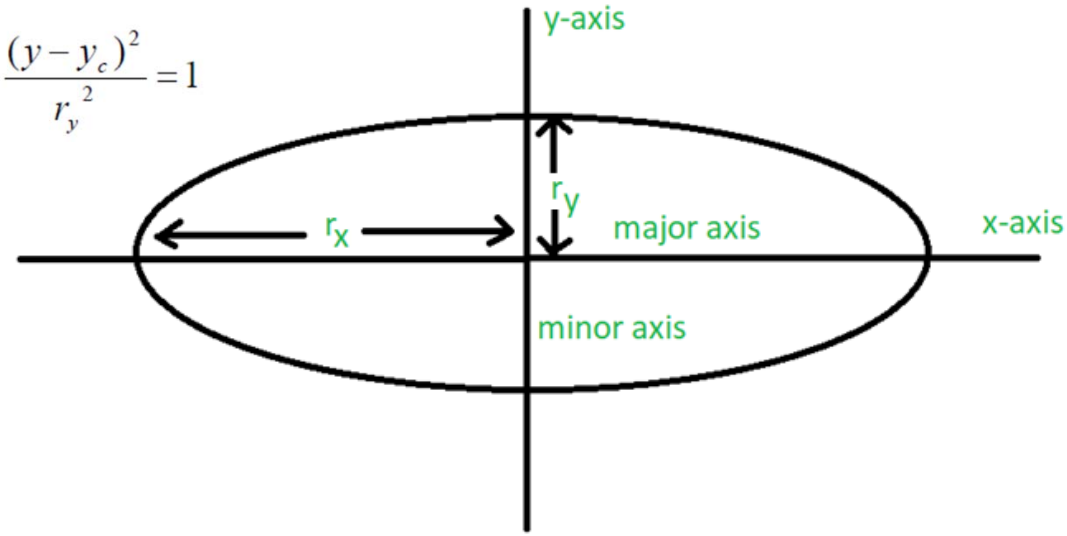
Scan Converting a ellipse

- An ellipse is described as a curve on a plane that surrounds two focal points such that the sum of the distances to the two focal points is constant for every point on the curve.



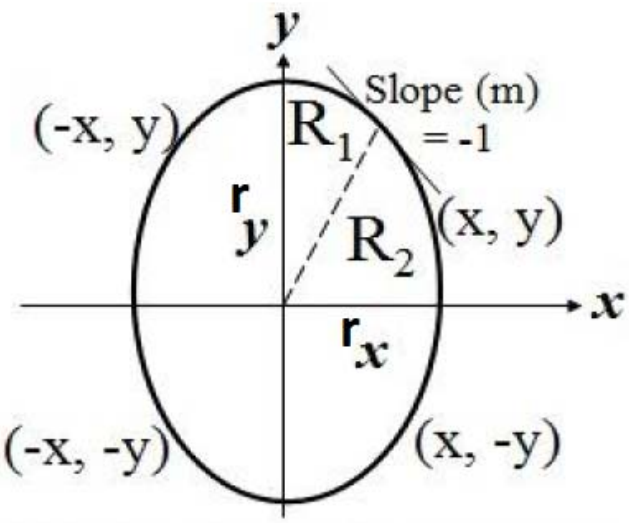
General equation:

$$\frac{(x-x_c)^2}{r_x^{\,2}}+\frac{(y-y_c)^2}{r_y^{\,2}}=1$$



Midpoint Ellipse Algorithm

- This algorithm is applied throughout the 1st quadrant in two parts
- Figure shows the division of 1st quadrant according to the slope of an ellipse with $r_x < r_y$
- process this quadrant by taking unit steps in x-direction where slope of curve has magnitude less than 1 i.e. Region 1 (R1), and taking unit step in y-direction where slope has magnitude greater than 1 i.e. Region 2 (R2).
- Two approach
  - start at position $(0, r_y)$ i.e. R1 and step clock wise along the elliptical path in first quadrants, shifting from unit step in x to unit step in y when magnitude of slope becomes greater than 1 i.e R2
  - alternatively, start at position $(r_x, 0)$ i.e R2 and select points in counter clockwise, shifting from unit step in y to unit step in x when the magnitude of slope becomes less than 1 i.e. R1.
- Here, we start at position $(0, r_y)$
- -we define an ellipse function with $(x_c, y_c)=(0,0)$

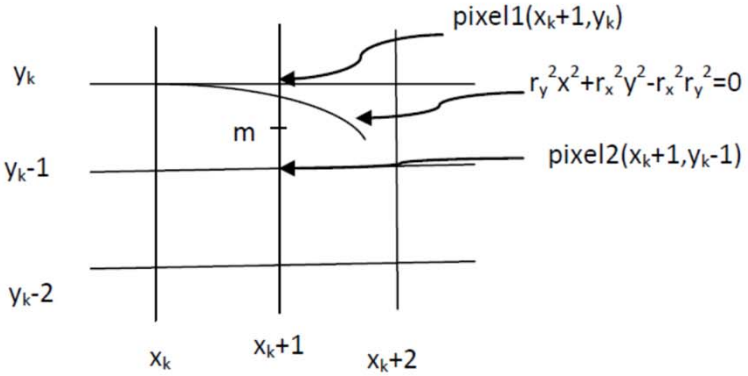


Midpoint Ellipse Algorithm

$f_{ellipse}(x,y)=r_y^2x^2+r_x^2y^2-r_x^2r_y^2$ .....(i)

With properties

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



- Thus ellipse function serves as the decision parameter
- At each sampling position, we select the next pixel along the ellipse path according to the sign of the ellipse function evaluated at the midpoint between the two candidate pixels.

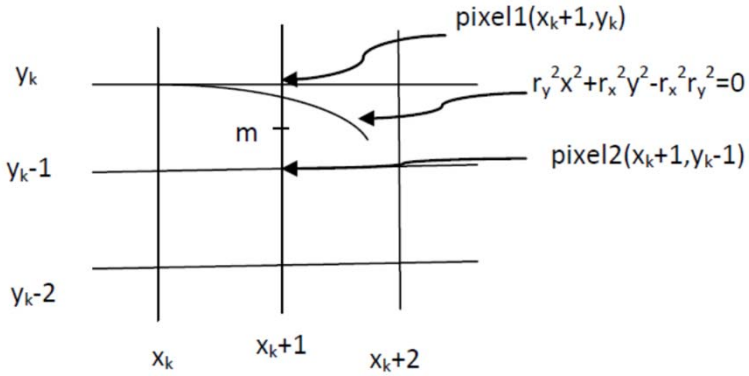
Midpoint Ellipse Algorithm

- at each step, we test the value of the slope of the curve,
- We have, $r_y^2x^2+r_x^2y^2= r_x^2r_y^2$

Differentiating with respect to x,

$$2r_y^2x + 2r_x^2y\frac{dy}{dx} = 0$$

$$\therefore \frac{dy}{dx} = -\frac{2r_y^2x}{2r_x^2y} \quad \text{.....(ii)}$$



-at the boundary between region 1 and region 2, slope=-1, so,

$$\frac{dy}{dx} = -1 = \frac{-2x_y^2x}{2r_x^2y}$$

$$\therefore 2r_y^2x = 2r_x^2y$$

-we move out of region 1, whenever

$$2r_y^2x \geq 2r_x^2y$$

Midpoint Ellipse Algorithm

-Assuming (x_k, y_k) has been illuminated (selected) we determine the next position along the ellipse path by evaluating the decision parameter at the midpoint $(x_k+1, y_k-\frac{1}{2})$

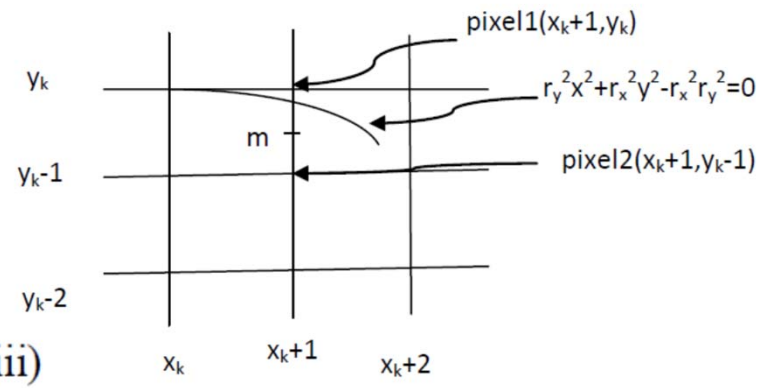
we have to determine the next point

(x_k+1, y_k) or (x_k+1, y_k-1)

We define decision parameter at mid-point as

$$P_{1k}=f_{ellipse}(x_k+1, y_k-\frac{1}{2})$$

$$P_{1k}==r_y^2(x_k+1)^2+r_x^2(y_k-\frac{1}{2})^2-r_x^2r_y^2 \quad \text{.....(iii)}$$



If $p_{1k}<0$, the mid-point is inside the ellipse and the pixel on scan line y_k is closer to the ellipse boundary

Otherwise,

The mid-point is outside or on the boundary and we select the pixel on scan line y_k-1

Midpoint Ellipse Algorithm

- At the next sampling position ($x_{k+1}+1=x_k+2$), the decision parameter for region 1 is evaluated as $P_{1k+1}=f_{ellipse}(x_{k+1}+1, y_{k+1}-\frac{1}{2})$

- $p_{1k+1} = r_y^2[(x_k+1)+1]^2 + r^2 x(y_{k+1}-\frac{1}{2})^2 - r_x^2 r_y^2$ .....(iv)

Subtracting equation (iii) from (iv)

$$p_{1k+1} - p_{1k} = r_y^2[(x_k+1+1)^2 - (x_k+1)^2] + r_x^2\left[\left(y_{k+1}-\frac{1}{2}\right)^2 - \left(y_k-\frac{1}{2}\right)^2\right] - r_x^2 r_y^2 + r_x^2 r_y^y$$

$$= r_y^2\left[(x_k+1)^2 + 2(x_k+1) + 1 - (x_k+1)^2\right] + r_x^2\left[y^2_{k+1} - 2y_{k+1}\frac{1}{2} + \frac{1}{4} - y_k^2 + 2y_k.\frac{1}{2} - \frac{1}{4}\right]$$

$$= r_y^2.2(x_k+1) + r_y^2 + r_x^2\left[\left(y^2_{k+1} - y_k^2\right) - (y_{k+1} - y_k)\right]$$

$$= 2r_y^2(x_k+1) + r_y^2 + r_x^2\left[\left(y^2_{k+1} - y_k^2\right) - (y_{k+1} - y_k)\right]$$

Midpoint Ellipse Algorithm

Where y_{k+1} is either y_k or y_k-1, depending on the sign of p_{1k}

$p_{1k} \le 0$ i.e. $y_{k+1}=y_k$ then decision parameter is

$$p_{1k+1} = p_{1k} + 2r_y^{\,2}(x_k + 1) + r_y^{\,2}$$

If $p_{1k} > 0$, i.e. $y_{k+1} = y_k - 1$, then decision parameter is

$$p_{1k+1} = p_{1k} + 2r_y^{\,2}(x_k + 1) + r_y^{\,2} - 2r_x^{\,2}y_{k+1}$$

The initial decision parameter is evaluated at start position $(x_0, y_o)=(0, r_y)$ as

$$P_{10} = f_{ellipse}\left(1, r_y - \frac{1}{2}\right)$$

$$= r_y^{\,2} + r_x^{\,2}\left(r_y - \frac{1}{2}\right)^2 - r_x^{\,2}y_x^{\,2}$$

$$p_{10} = r_y^{\,2} - r_x^{\,2}r_y + \frac{1}{4}\,r_x^{\,2} \quad \ldots\ldots\ldots\ldots \quad (v)$$

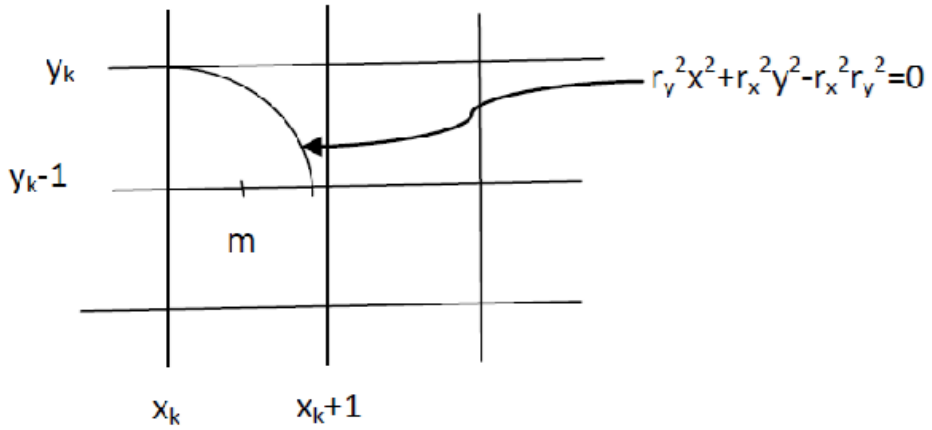
Midpoint Ellipse Algorithm

Region 2

We sample at unit steps in the negative y direction and the mid point is now taken between horizontal pixels at each step.

The decision parameter is

$P_{2k} = f_{ellipse}(x_k + \frac{1}{2}, y_k - 1)$



$$p_{2k} = r_y^2\left(x_k + \frac{1}{2}\right)^2 + r_x^2(y_k - 1)^2 - r_x^2 r_y^2 \quad \text{.....} \quad \text{(vi)}$$

If $p_{2k} > 0$, the mid-point is outside the ellipse boundary and we select the pixel x_k If $p_{2k} \leq 0$, the midpoint is inside or on the ellipse boundary and we select pixel position x_k+1

Note: Initial decision parameter for region 2 is calculated by taking last point of region 1 as (x_k, y_k)

Midpoint Ellipse Algorithm

Now, at next sampling position $y_{k+1}-1=y_k-2$

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

$$p_{2k+1} = r_y^2\left(x_{k+1}+\frac{1}{2}\right)^2 + r_x^2(y_{k+1}-1)^2 - r_x^2 r_y^2 \quad \text{.....(vii)}$$

Subtracting equation (vi) from (vii)

$$p_{2k+1} = p_{2k} + r_y^2\left[\left(x_{k+1}+\frac{1}{2}\right)^2 - \left(x_k+\frac{1}{2}\right)^2\right] + r_x^2\left[(y_k-1-1)^2 - (y_k-1)^2\right]$$

$$= \left[p_{2k} + r_x^2(y_k-1)^2 - 2(y_k-1)+1-(y_k-1)^2\right] + r_y^2\left[\left(x_{k+1}+\frac{1}{2}\right)^2 - \left(x_k+\frac{1}{2}\right)^2\right]$$

If $p_{2k} > 0$, then

$x_{k+1}=x_k$

$$p_{2k}+1 = p_{2k} - 2r_x^2 y_{k+1} + r_x^2$$

If $p_{2k} \le 0$, then

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

$$p_{2k+1} = p_{2k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

Midpoint Ellipse Algorithm

Step 1. Start

Step 2. Declare variables x_c, y_c, r_x, r_y, x_0, y_0, p_{10}, p_{1k}, p_{1k+1}, p_{20}, p_{2k}, p_{2k+1}

Step 3. Read Values of x_c, y_c, r_x, r_y,

Step 4. obtain the first point on an ellipse centered on origin (x_0, y_0) by initializing the x_0 and y_0 as $x_0=0$, $y_0=r$

Step 5. Calculate the initial value of the decision parameter in region 1 as

$$p_{10} = r_y^2 - r_x^2 r_y + \frac{1}{4} r_y^2$$

Step 6. For each x_k position in region 1, starting at k=0, perform the following test.

If $P_{1k}<0$,

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

$y_{k+1}=y_k$

$$p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} + r_y^2$$

else

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

$y_{k+1}=y_k-1$

$$p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} - 2r_y^2 y_{k+1} + r_y^2$$

and continue until $2r_y^2 x \geq 2r_y^2 y$

Midpoint Ellipse Algorithm

Step 7. Calculate the initial decision parameter in region 2 using the last point (x_0, y_0) calculated in region 1 as

$$p_{20} = r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2$$

Step 8. At each y_k position in region 2, starting at k=0, perform the following test.

If $P_{2k}>0$,

$x_{k+1}=x_k$

$y_{k+1}=y_k-1$

$$p_{2k+1} = p_{2k} - 2r_x^2 y_{k+1} + r_x^2$$

else

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

$y_{k+1}=y_k-1$

$$p_{2k+1} = p_{2k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

Repeat the steps for region 2 until $y<0$.

Step 9. Determine the symmetry points in the other three quadrants.

Step 10. Move each calculated pixel position(x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values.

$x=x+x_c$ and $y=y+y_c$

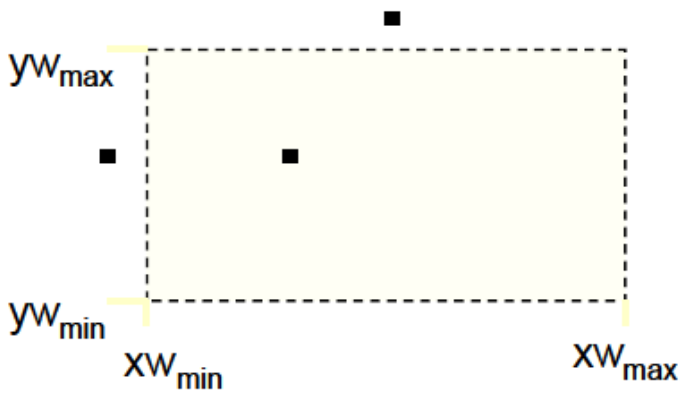
Step 11. Stop.

Clipping

- Any procedure that identifies those portions of a picture that are either inside or outside of a specified region of a space
- A **clip window** can be polygon or curved boundaries
- World- coordinate clipping removes the primitives outside the window from further consideration; thus eliminating the processing necessary to transform these primitives to device space.
- **Clipping type-** point ,line, area, curve and text clipping

Point Clipping

Assume clipping window is Rectangle, point P=(x,y) is saved for display if following inequalities are satisfied

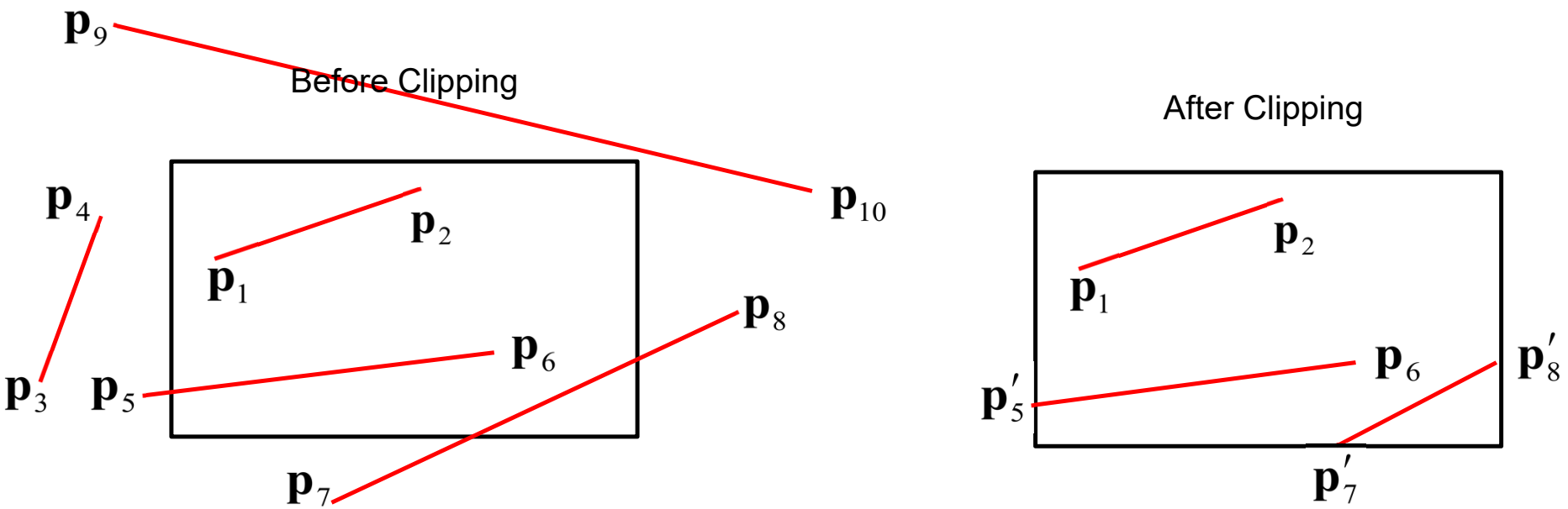


$$xw_{min} \leq x \leq xw_{max}$$
$$yw_{min} \leq y \leq yw_{max}$$

$(xw_{min}, xw_{max}, yw_{min}, yw_{max})$ → can be either world coordinate window boundaries or viewport boundaries

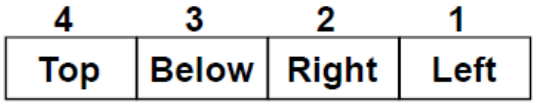
If all the four inequalities are satisfied for a point with co-ordinate (x,y), the point is accepted; i.e not clipped

Line Clipping



Cohen Sutherland Line Clipping Algorithm

- This is an efficient method of accepting or rejecting lines that do not intersect the window edges.
- Divide 2D space into 3x3 = 9- regions.
- Middle region is the **clipping window.**
- Each region is assigned **a** 4-bit code.
- Region codes indicate the position of the regions with respect to the window



Region Code Legend

bit 4 : $y > yw_{max}$

bit 3 : $y < yw_{min}$

bit 2 : $x > xw_{max}$

bit 1 : $x < xw_{min}$

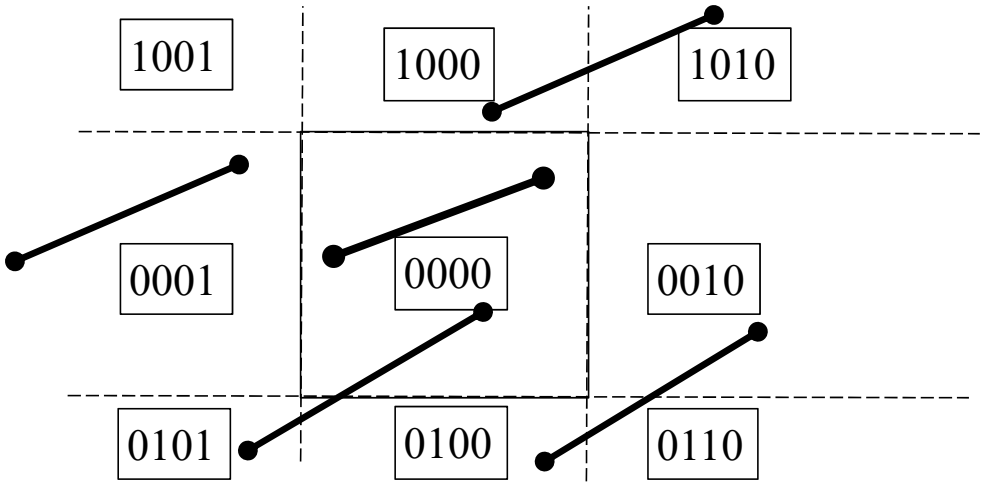
TBRL

1001	1000	1010
0001	0000 Window	0010
0101	0100	0110

Cohen Sutherland Line Clipping Algorithm

1. Assign a region code for each endpoints.
2. If both endpoints have a **region code 0000** →**trivially accept** these line.
3. Else, perform the logical AND operation for both region codes.
   - 3.1 **if** the result is **NOT** 0000 → **trivially reject** the line.
   - 3.2 **else** (i.e. **result = 0000**, need clipping)
     - 3.2.1. Choose an endpoint of the line that is outside the window.
     - 3.2.2. Find the intersection point at the window boundary (based on region code).
     - 3.2.3. Replace endpoint with the intersection point and update the region code.
     - 3.2.4. Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.
4. Repeat step 1 for other lines.

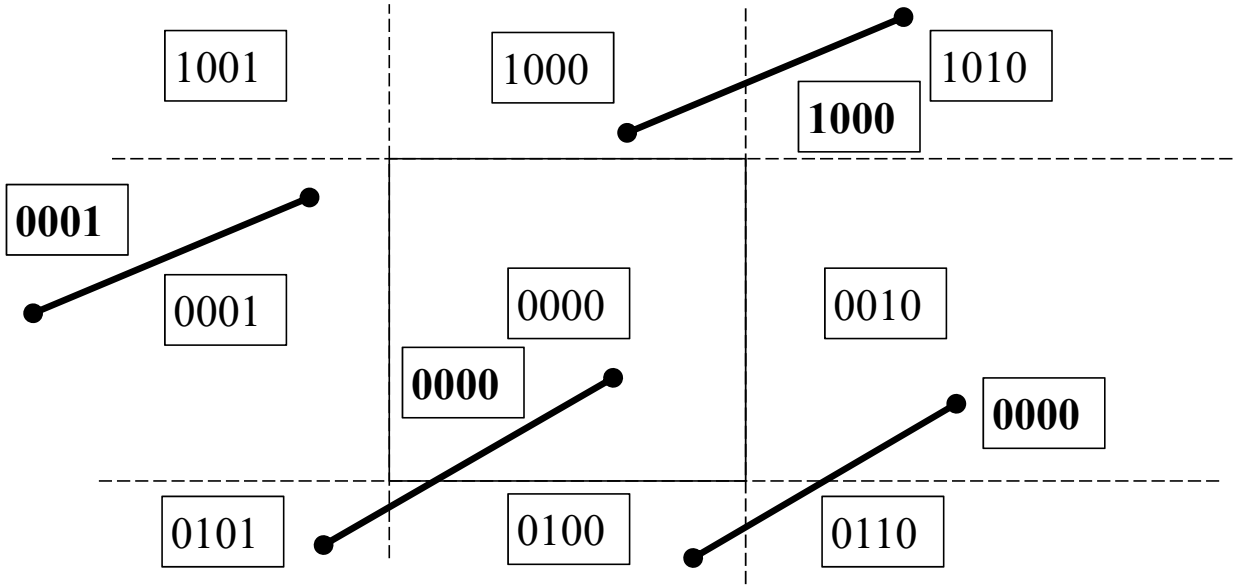
Cohen-Sutherland Algorithm



Both endpoint codes 0000, trivial acceptance, else:

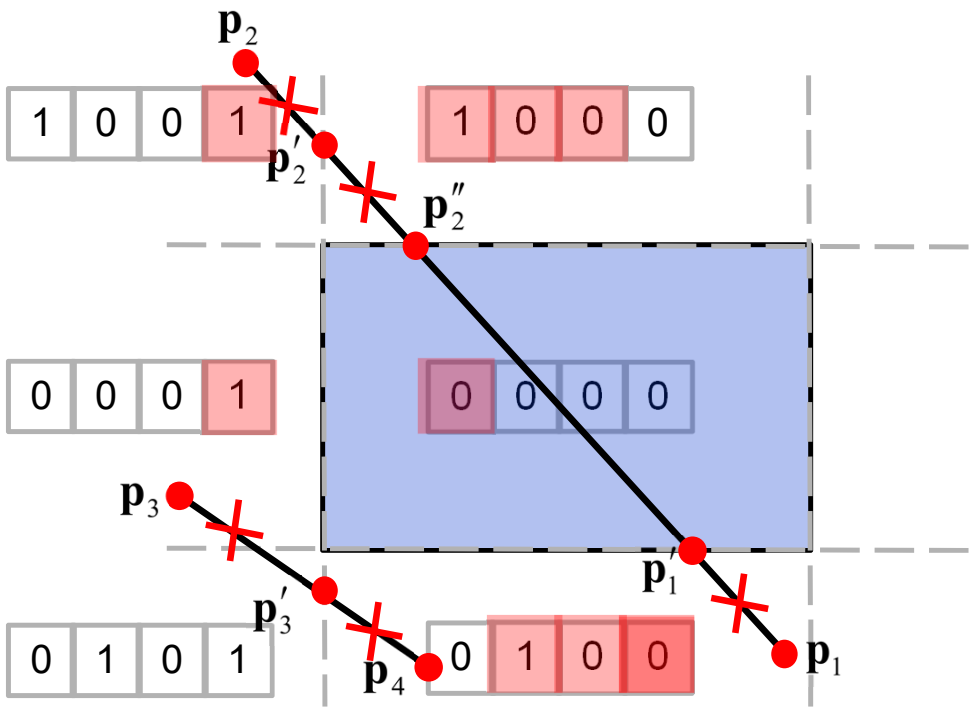
Do logical AND of Outcodes (reject if non-zero)

Cohen-Sutherland Algorithm



Logical AND between codes for 2 endpoints,
Reject line if non-zero – trivial rejection.

Cohen-Sutherland Algorithm



Cohen-Sutherland Algorithm

Intersection calculations:

Intersection with vertical boundary

$y = y_1 + m(x-x_1)$

Where

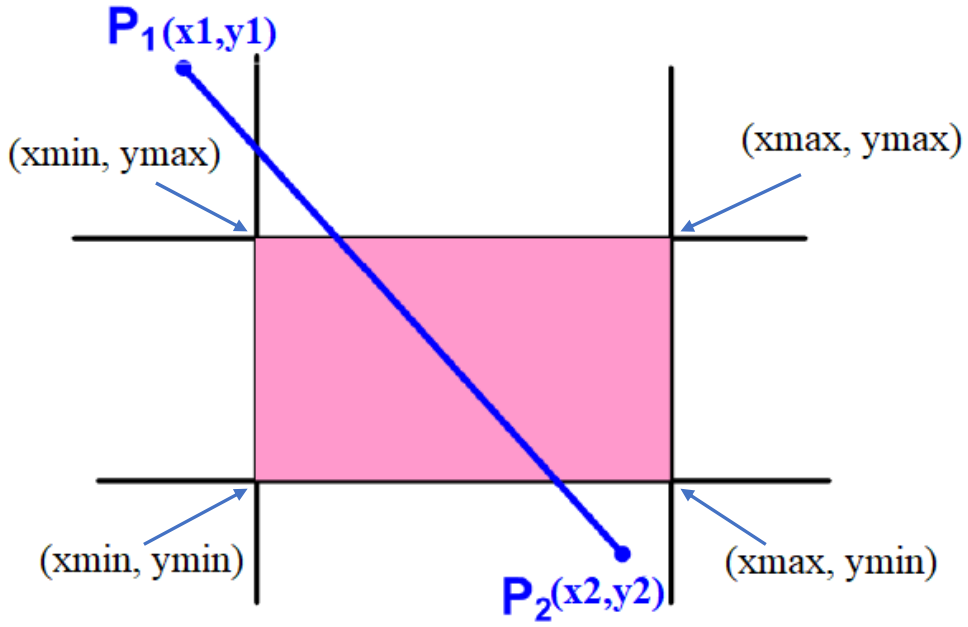
$x = x_{Wmin}$ or x_{Wmax}

Intersection with horizontal boundary

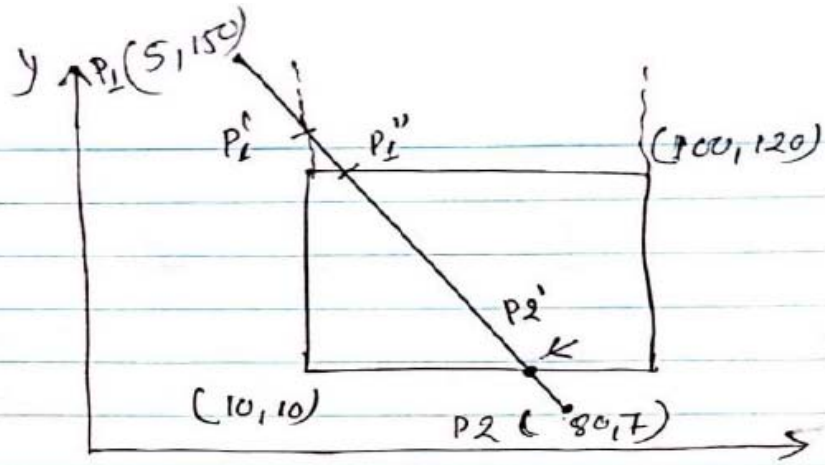
$x = x_1 + (y-y_1)/m$

Where

$y = y_{Wmin}$ or y_{Wmax}



⊛ Example-1 :



4	3	2	1
Top	Below	Right	Left

Region Code Legend

bit 4 : $y > yw_{max}$

bit 3 : $y < yw_{min}$

bit 2 : $x > xw_{max}$

bit 1 : $x < xw_{min}$

- Clip P1P2 against given window.

Sol'n Region code for P1

↓

1001

$x - xw_{min} < 0 \Rightarrow$ bit1 = 1

$yw_{max} - y < 0 \Rightarrow$ bit4 = 1

Region code for P2,

↓

0100

$x - xw_{min} > 0 \Rightarrow$ bit1 = 0

$xw_{max} - x > 0 \Rightarrow$ bit2 = 0

$y - yw_{in} < 0 \Rightarrow$ bit3 = 1

$yw_{max} - y > 0 \Rightarrow$ bit4 = 0

Logical AND of two Region codes

$$\begin{array}{r} 1001 \\ 0100 \\ \hline 0000 \end{array}$$

Choose P2, to calculate the intersection with window boundary.

$$P2 = (x', y_{Wmin})$$

$$x' = x_1 + \frac{y_{Wmin} - y_2}{m}$$

$$m = \frac{7-150}{80-5} \Rightarrow \frac{-143}{75} = -1.91$$

$$x' = 80 + \frac{10-7}{-1.91} \Rightarrow 80 + \frac{3}{1.91}$$

$$\Rightarrow 80 - 1.57$$

$$\Rightarrow 78.43$$

$$x' \simeq 78$$

$$\boxed{P_2' = (78, 10)}$$

, Region code (0000)

- Logical ORing of two Region code

```
  1001
  0000
 ------
 [1001]  → Need to clip.
```

Choose P_1, to calculate the intersection with window Vertical boundary, $P_1' = (10, y')$

$$y' = y_1 + m(x - x_1)$$

$$= 150 + -1.91(10 - 5)$$

$$= 150 - 9.55$$

$$= 140.45$$

$$y' \simeq 141$$

$P_1' = (10, 141)$

↓ Region code

1000

Logical ORing of two region codes,

0000
1000
1000 → Need to clip again.

Calculate the intersection with horizontal boundary,

$P_1'' = (?, y_{wmax})$

$$x'' = x_1 + \frac{y - y_1}{m}$$

$$= 5 + \frac{120 - 150}{-1.91}$$

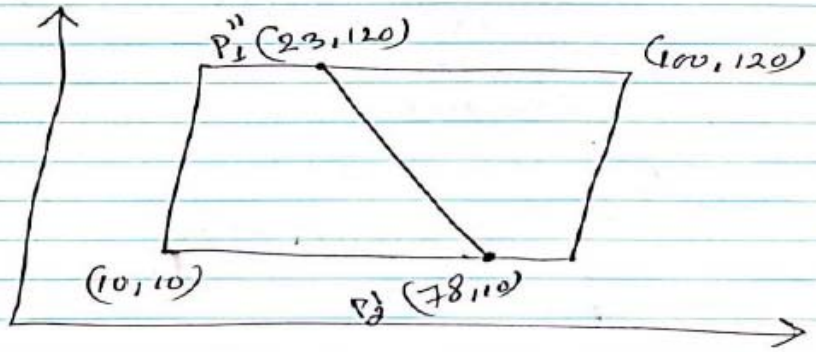
$$= 5 + 15.71$$

$$= 22.71$$

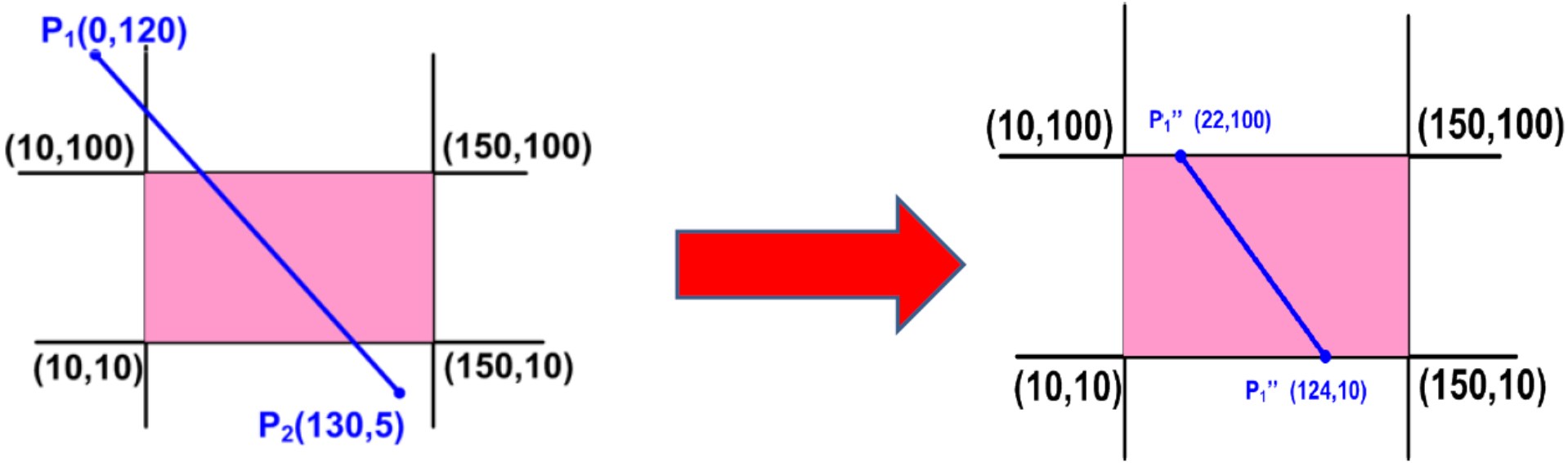
$x'' \simeq 23$

$P_1'' = (23, 120)$
$P_2' = 78, 110$

Both Region 0000 Accepted & Clipped.



Obtain the endpoints of line P1P2 after cohen-sutherland clipping



Liang Barsky Line Clipping Algorithm

- Uses parametric equation of a line and inequalities describing the range of the clipping window to determine the intersections between the line and the clip window.
- Efficient algorithm than Cohen Sutherland Algorithm, as not computing the coordinate values at irrelevant vertices
- Parametric equation of a line

  $x = x_1 + t\,(x_2 - x_1)$ ……..(1)

  $y = y_1 + t\,(y_2 - y_1)$ ……..(2)
- A point is inside the clipping window if

  $xw_{min} \le x \le xw_{max}$ i.e. $xw_{min} \le x_1 + t\Delta x \le xw_{max}$ where $\Delta x = x_2 - x_1$

  And,

  $yw_{min} \le y \le yw_{max}$ i.e. $yw_{min} \le y_1 + t\Delta y \le yw_{max}$ where $\Delta y = y_2 - y_1$

Liang Barsky Line Clipping Algorithm

- Each of these inequalities can be written as

$- t\Delta x \leq x1 - xw_{min}$ (multiplying both sides by – in $t\Delta x \geq xw_{min} - x_1$)

$t\Delta x \leq xw_{max} - x_1$

$- t\Delta y \leq y1 - yw_{min}$ (multiplying both sides by – in $t\Delta y \geq yw_{min} - y_1$)

$t\Delta y \leq yw_{max} - y_1$

- These 4 inequalities can be expressed as

$tp_k \leq q_k$ for k = 1, 2, 3, 4

where,

$p_1 = -\Delta x$ and $q_1 = x1 - xw_{min}$ (left boundary)

$p_2 = \Delta x$ and $q_2 = xw_{max} - x_1$ (right boundary)

$p_3 = -\Delta y$ and $q_3 = y1 - yw_{min}$ (bottom boundary)

$p_4 = \Delta y$ and $q_4 = yw_{max} - y_1$ (top boundary)

Liang Barsky Line Clipping Algorithm

Algorithm

1. If $p_k = 0$ for some k, then the line is parallel to the clipping boundary, now test q_k

   If $q_k < 0$ for these k, then the line is outside

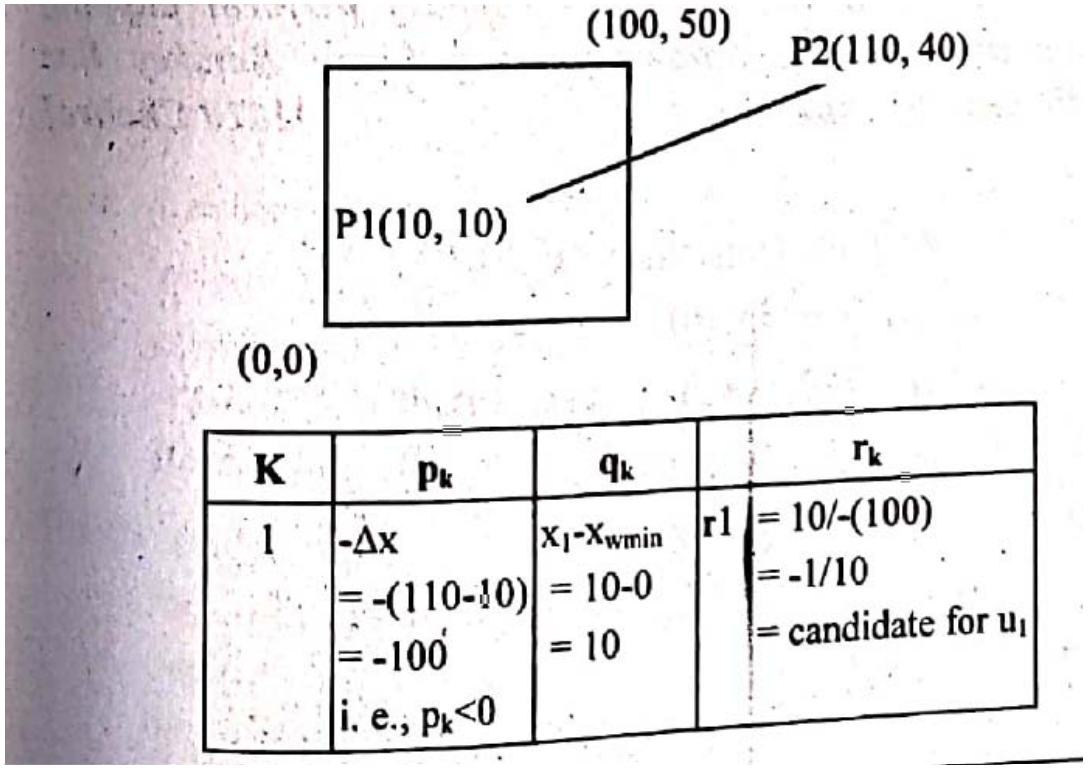
   If $q_k \geq 0$ for these k, then some portion of the line is inside

1. For all $p_k < 0$ (i.e. line proceeds from outside to inside boundary) calculate $t_1 = \max(0, r_k)$ to determine intersection point with clipping boundary and obtain new point for that line at t_1 ($r_k = q_k / p_k$)
2. For all $p_k > 0$ (i.e. line proceeds from inside to outside boundary) calculate $t_2 = \min(1, r_k)$ to determine intersection point with clipping boundary and obtain new point for that line at t_1 ($r_k = q_k / p_k$)
3. If t1 > t2, then discard the line
4. Else new points are calculated as

$$x_{1new} = x_1 + t_1\Delta x \quad \text{and} \quad y_{1new} = y_1 + t_1\Delta y$$

$$x_{2new} = x_1 + t_2\Delta x \quad \text{and} \quad y_{2new} = y_1 + t_2\Delta y$$

Example of Liang Barsky Line Clipping Algorithm



K	p_k	q_k	r_k
1	$-\Delta x$ $= -(110-10)$ $= -100$ i. e., $p_k<0$	x_1-x_{wmin} $= 10-0$ $= 10$	$r1$ $= 10/-(100)$ $= -1/10$ $=$ candidate for u_1

K	p_k	q_k	r_k
2	Δx $= (110-10)$ $= 100$ i. e., $p_k>0$	$x_{wmax}-x_1$ $= 100-10$ $= 90$	r_2 $= 90/100$ $= 0.9$ $=$ candidate for u_2
3	$-\Delta y$ $= -(40-10)$ $= -30$ i. e.$Pk<0$	y_1-y_{wmin} $= 10-0$ $= 10$	r_3 $= 10/-30$ $= -1/3$ $=$ candidate for u_1
4	Δy $= (40-10)$ $= 30$ i. e.$Pk>0$	$y_{wmax}-y_1$ $= 50-10$ $= 40$	r_4 $= 40/30$ $= 4/3$ $=$ candidate for u_2

We take $u_1 = 0$ and $u_2 = 0.9$

Clipped line

$x_1' = 10+0\times100 = 10$ $x_2' = x1+u_2\times100 = 10+0.9\times100 = 100$

$y_1' = 10+0\times30 = 10$ $y_2' = y1+u_2\times30 = 10+0.9\times30 = 37$

Example of Liang Barsky Line Clipping Algorithm

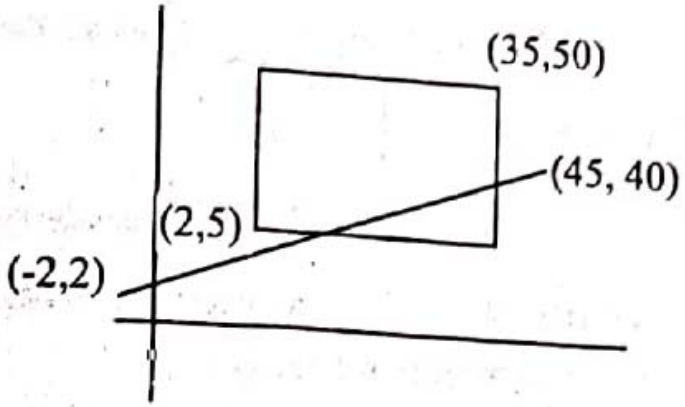
Clipping window: $(x_{min}, y_{min}) = (2,5)$.

And $(x_{max}, y_{max}) = (35,50)$

Line $(x_1, y_1) = (-2,2)$ and $(x_2, y_2) = (45,40)$

$\Delta x = x_2 - x_1 = 45 - (-2) = 47$

$\Delta y = y_2 - y_1 = 40 - 2 = 38$



k	p_k	q_k	$R_k = \frac{q_k}{p_k}$
0	$\Delta x = -47, p_k < 0$	$x_1 - x_{min} = -4$	$0.0851(u_1)$
1	$\Delta x = 47$	$x_{max} - x_1 = 37$	$0.787(u_2)$
2	$-\Delta y = -38$	$y_1 - y_{min} = -3$	$0.0789(u_1)$
3	$\Delta y = 38$	$y_{max} - y_1 = 48$	$1.263(u_2)$

$u_1 = \max(0, r_k)$

$= \max(0, 0.0851, 0.0789)$

$= 0.0851$

$u_2 = \min(1, r_k) = \min(1, 0.787, 1.263) = 0.787$

$x_1' = x_1 + u_1 \Delta x = -2 + 0.0851 \times 47 = 1.997 \approx 2$

$y_1' = y_1 + u_1 \Delta y = -2 + 0.0851 \times 38 = 5$

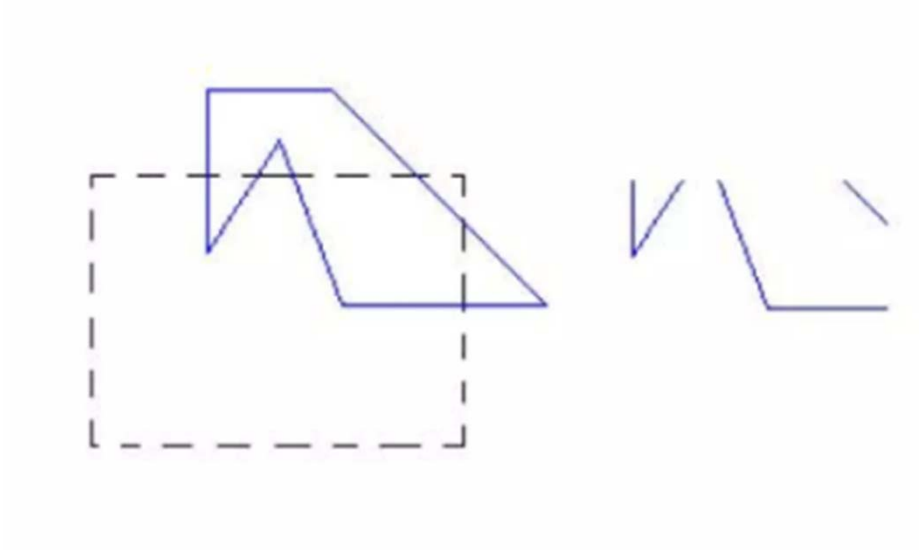
$x_2' = x_1 + u_2 \Delta x = -2 + 0.787 \times 47 = 35$

$y_2' = y_1 + u_2 \Delta y = 2 + 0.787 \times 38 = 32$

Required points are (2,5) and (35, 32)

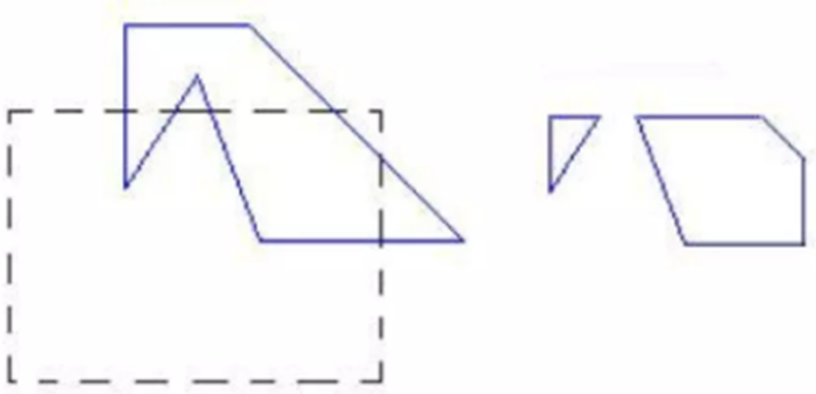
Polygon(Area) Clipping

- To clip a polygon, we cannot directly apply line clipping algorithm to individual polygon edges because this approach produce a series of unconnected line segments as show in figure below.



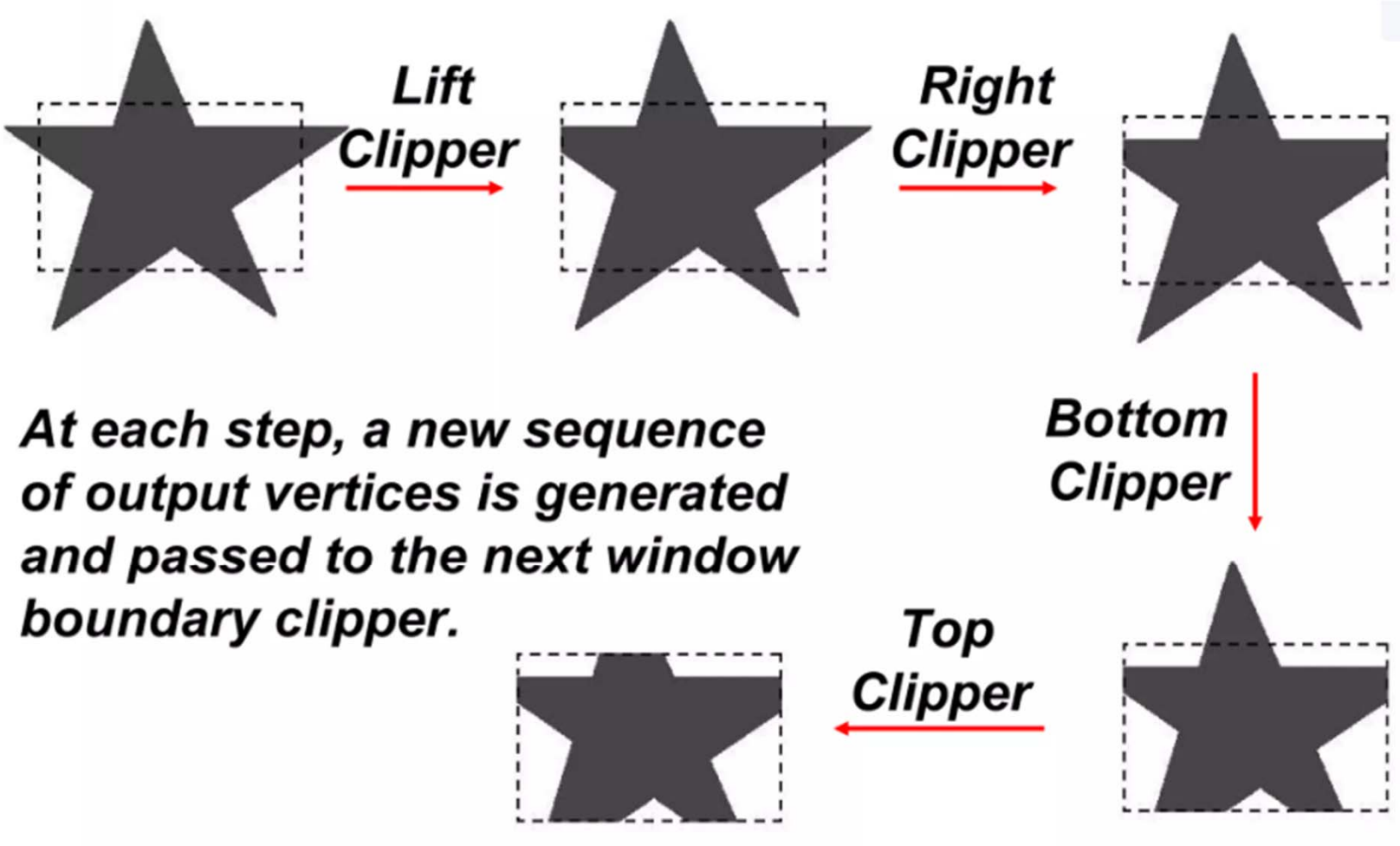
Polygon(Area) Clipping

- The clipped polygon must be a bounded area after clipping as shown in figure



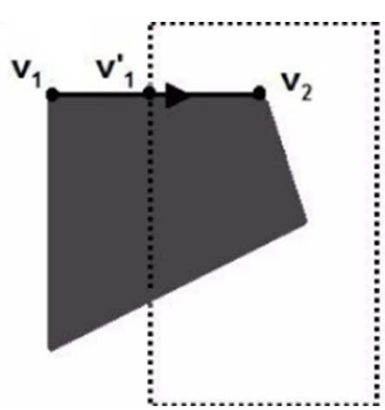
- For polygon clipping, we require an algorithm that will generate one or more closed areas that are then scan converted for area fill.

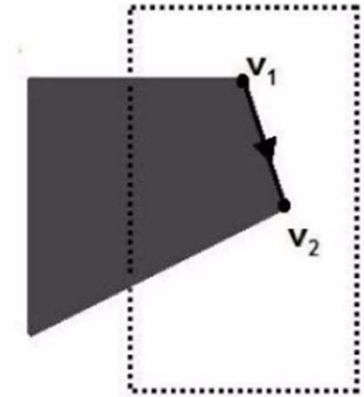
Sutherland-Hodgman Polygon Clipping

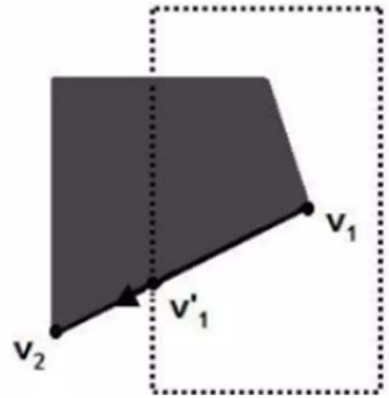


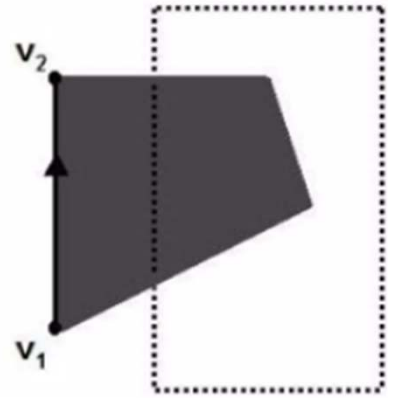
Sutherland-Hodgman Polygon Clipping

- There are four possible cases when processing vertices in sequence around the perimeter of a polygon
- As each pair of adjacent polygon vertices is passed to a next window boundary clipper, we make the following tests:
  - If the first vertex is outside the window boundary and the second vertex is inside, intersection point of the polygon edge with window boundary and the second vertex both are added to the output vertex list.
  - If both input vertices are inside the window boundary, only second vertex is added to output vertex list.
  - If the first vertex is inside the window boundary and the second vertex is outside, only edge intersection with the window boundary is added to the output vertex list.
  - If both input vertices are outside the window boundary, nothing is added to the output vertex list.



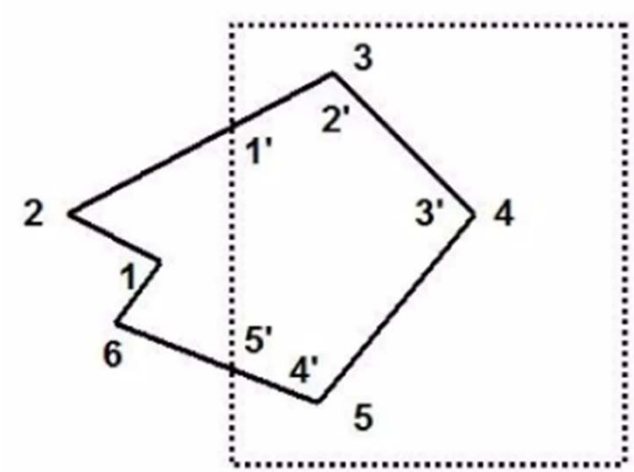




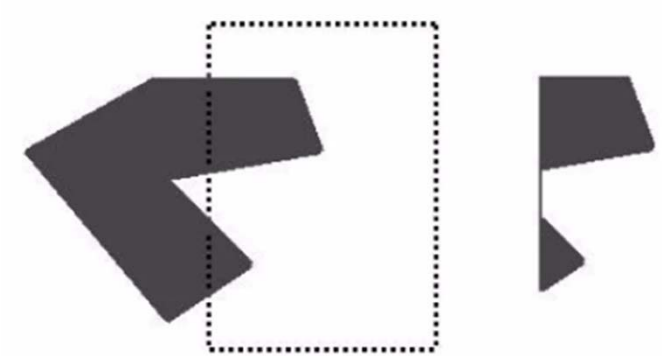


Sutherland-Hodgman Polygon Clipping

- **Example**
- Vertices 1 and 2 are outside of the boundary
- Vertex 3, which is inside, 1’ and vertex 3 are saved.
- Vertex 4 and 5 are inside and they are also saved
- Vertex 6 is outside, 5’ is saved.



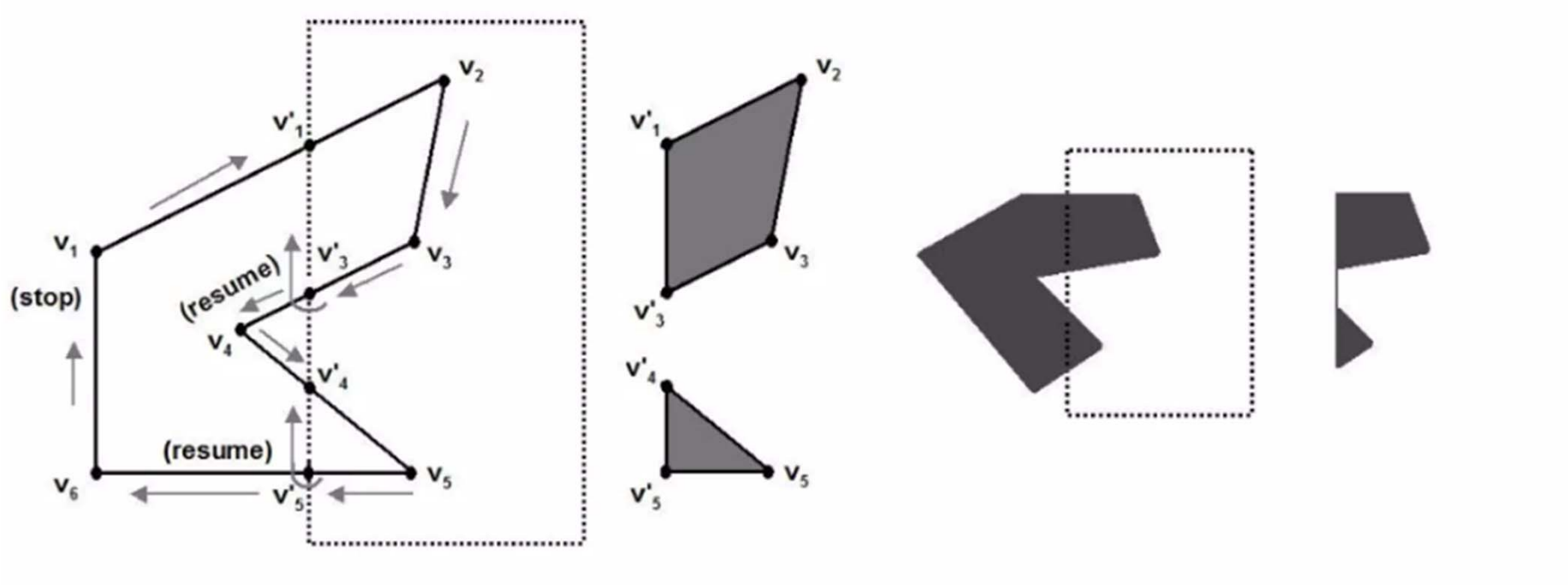
- **Drawback**
- The algorithm correctly clips convex polygons but concave polygons may be displayed with extraneous lines



Weiler-Atherton Polygon Clipping

- This algorithm was developed to clip a area that is either convex and concave polygon.
- The basic idea of this algorithm is that instead of proceeding around the polygon edges as vertices are processed, we will follow the window boundaries.
- The path we follow depends on:
  - Polygon-processing direction (clockwise or counterclockwise)
  - The pair of polygon vertices outside-to-inside or inside-to-outside
- For clockwise processing of polygon vertices, we use the following rules:
  - For an outside-to-inside pair of vertices, follow polygon boundaries.
  - For an inside-to-outside pair of vertices, follow window boundaries in a clockwise direction.

Weiler-Atherton Polygon Clipping



Clipped Polygon using Weiler-Atherton Polygon Clipping without connected component problem

Clipped Polygon using sutherland hogman algorithm with connected component problem

Inside/Outside Test

- Let P(x,y) be the polygon vertex which is to be tested against edge E defined by A(x1,y1) to B(x2,y2). Point P is said to be left of E or AB iff

$$\frac{y - y_1}{y_2 - y_1} - \frac{x - x_1}{x_2 - x_1} > 0$$

- Otherwise it is said to be right of edge E

Text Clipping

Text clipping relies on the concept of bounding rectangle

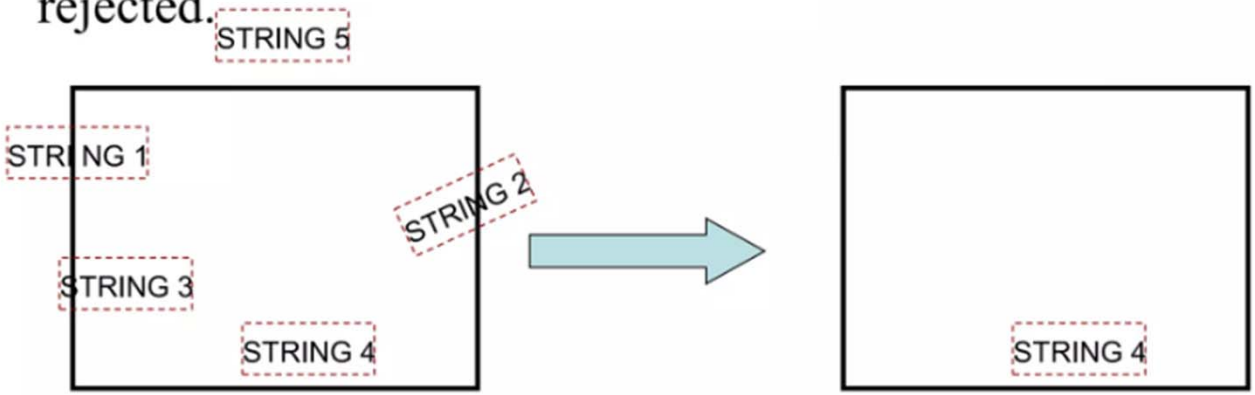
TYPES

1. All or None String Clipping
2. All or None Character Clipping
3. Component Character Clipping

Text Clipping

1. All or None String Clipping

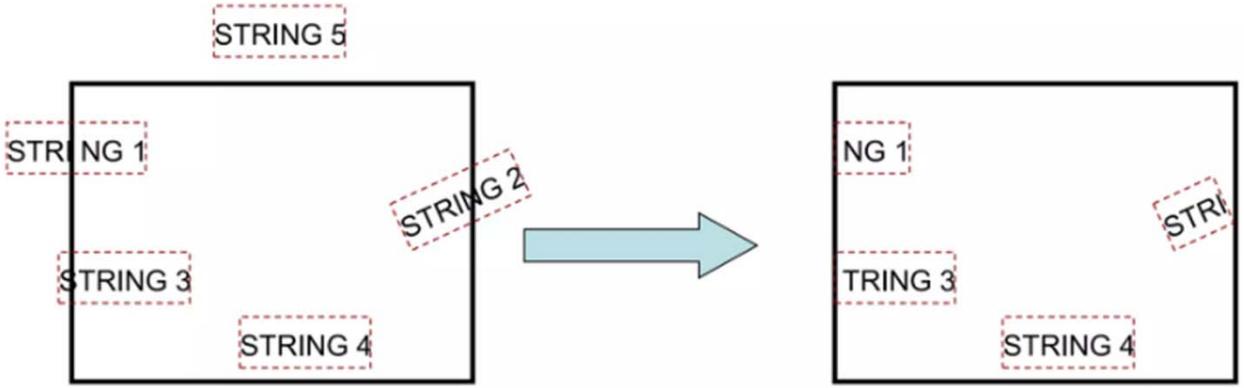
- In this scheme, if all of the string is inside window, we clip it, otherwise the string is discarded. This is the fastest method.
- The procedure is implemented by consider a ***bounding rectangle*** around the text pattern. The boundary positions are compared to the window boundaries. In case of overlapping the string is rejected.



Text Clipping

2. All or None Character Clipping

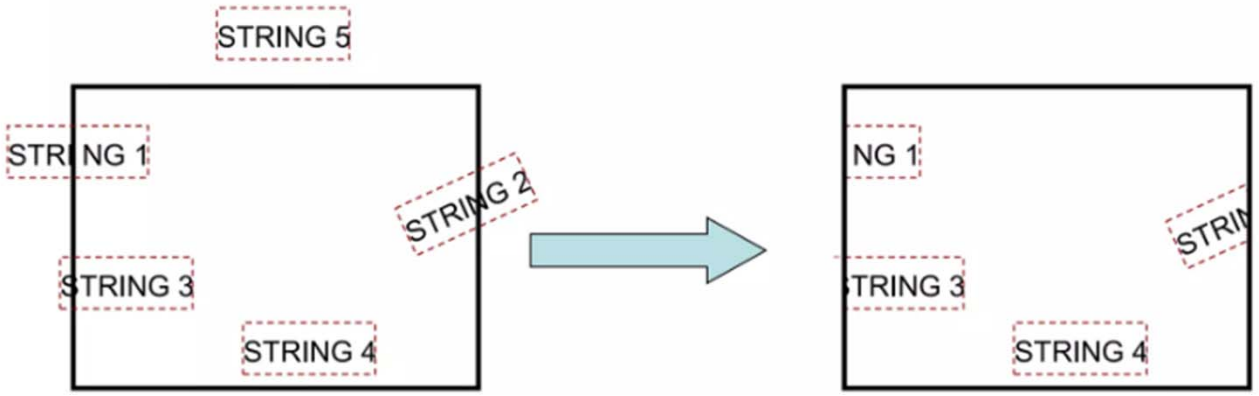
- In this scheme, we discard only those characters that are not completely inside window.
- Boundary limits of individual characters are compared against window. In case of overlapping the character is rejected.



Text Clipping

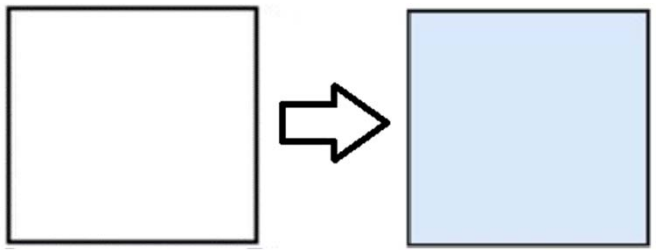
3. Component Character Clipping

- Characters are treated like graphic objects.
  - Bit Mapped Fonts : Point Clipping
  - Outlined Fonts : Line/Curve Clipping
- In case of overlapping the part of the character inside is displayed and the outside portion of the character is rejected.



Polygon Filling

- Highlighting all the pixels which lie inside the polygon
- Polygons are easier to fill
- There are two basic approaches
  - Seed Fill
    - Boundary Fill Algorithm
    - Flood Fill Algorithm
  - Scanline Algorithm

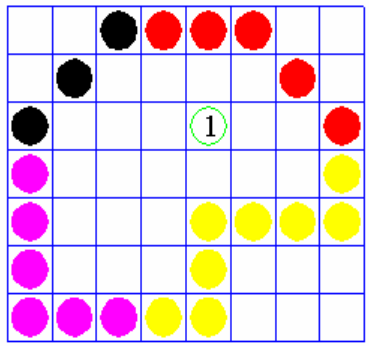


Boundary/Edge Fill Algorithm

- In this method, edges of the polygons are drawn.
- Then starting with any point inside the polygon (seed point), examine the neighboring pixels to check whether the boundary pixel is reached.
- If boundary pixels are not reached, pixels are highlighted and the process is continued until boundary pixels are reached.
- This algorithm is used for single color boundary only.

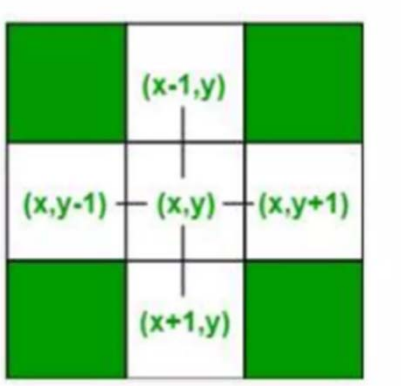
Flood Fill Algorithm

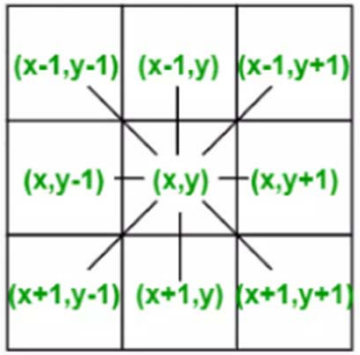
- Sometimes it is required to fill in an area that is not defined within a single color boundary.
- In such cases we can fill areas by replacing a specified interior color instead of searching for a boundary color.
- This approach is called a flood-fill algorithm. Like boundary fill algorithm, here we start with some seed and examine the neighboring pixels.
- However, here pixels are checked for a specified interior color instead of boundary color and they are replaced by new color.



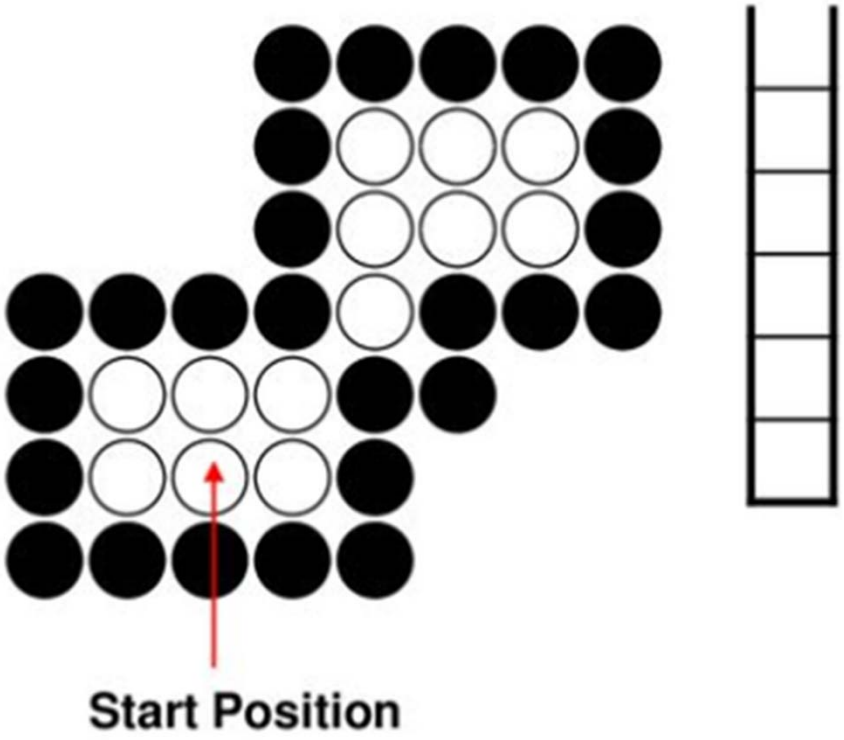
4 Connected/8 Connected Approach

- Filling adjacent pixels can be done using two approaches:
  - 4 Connected Approach
  - 8 Connected Approach

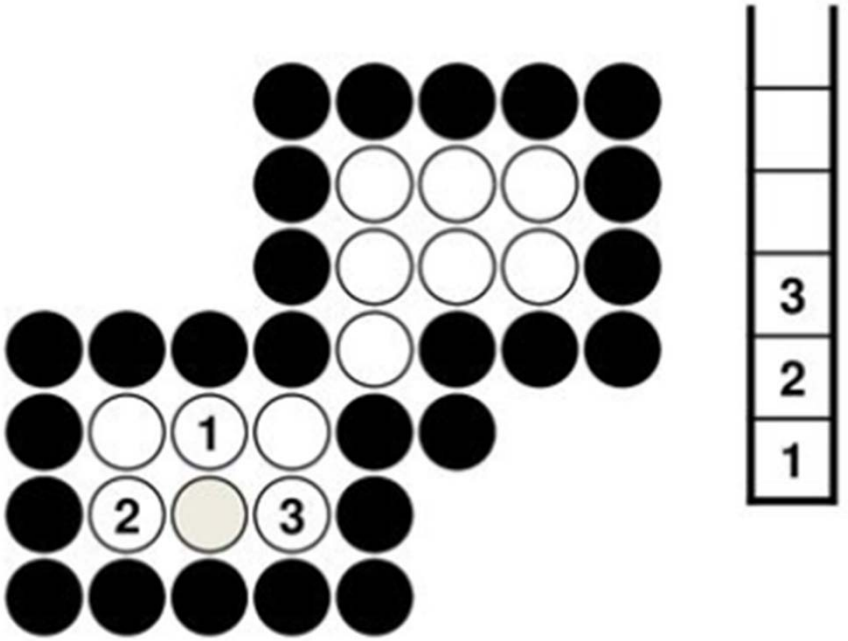




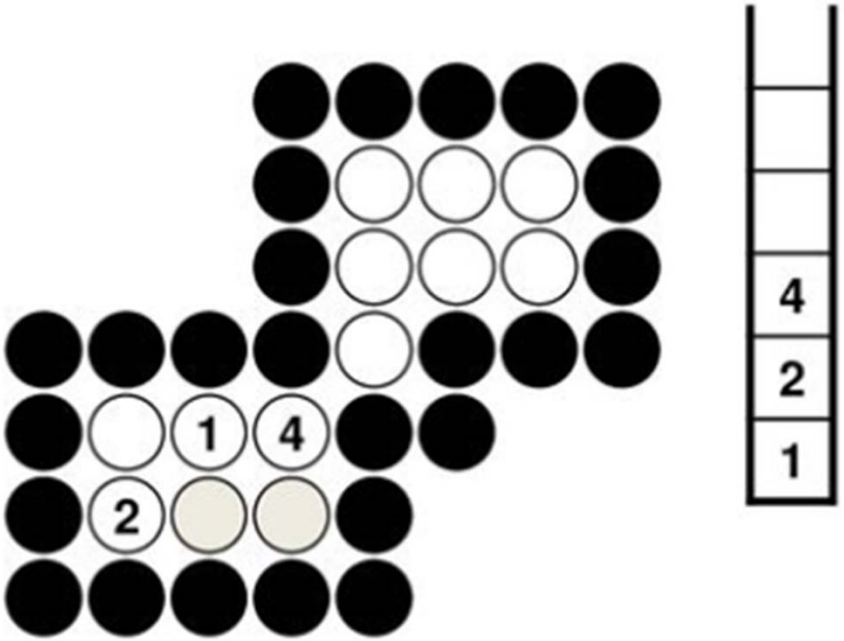
4 Connected Example



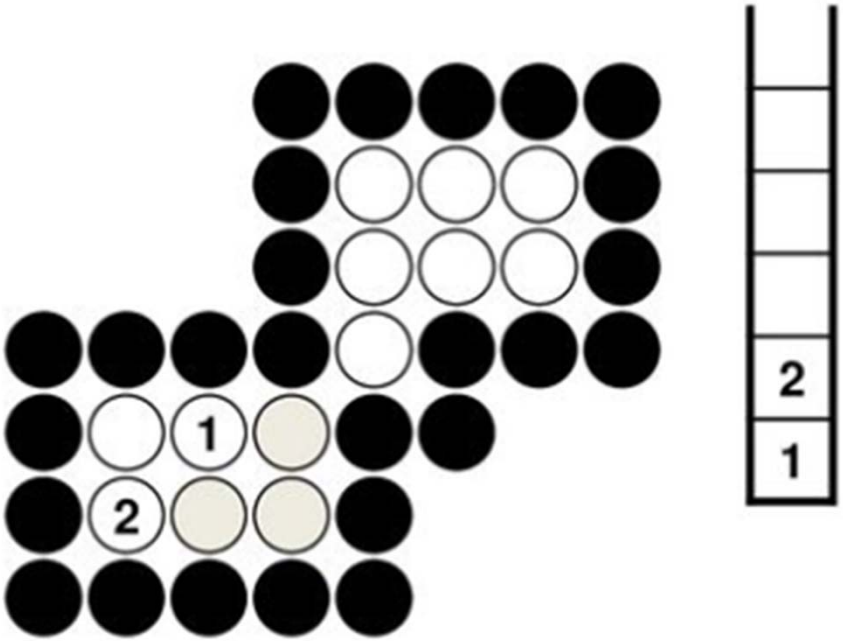
4 Connected Example



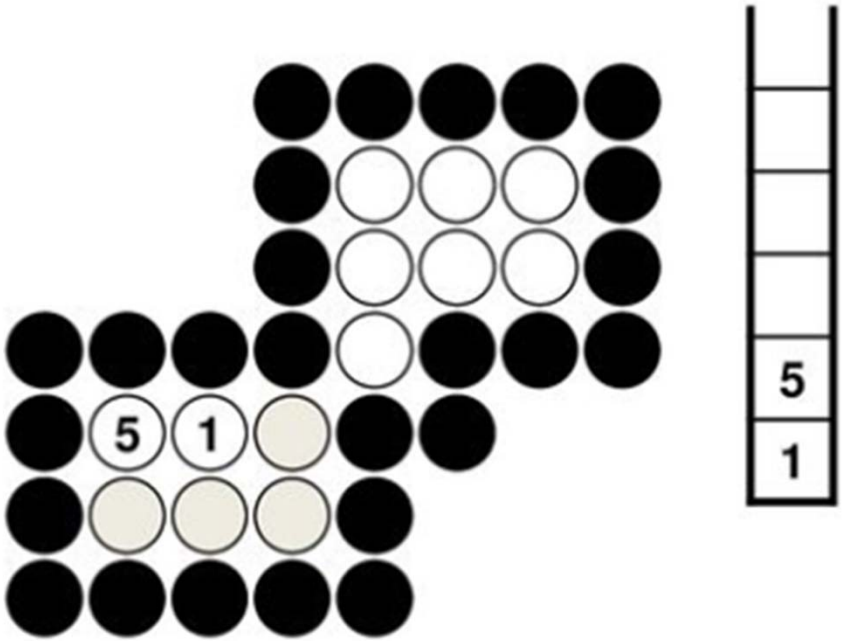
4 Connected Example



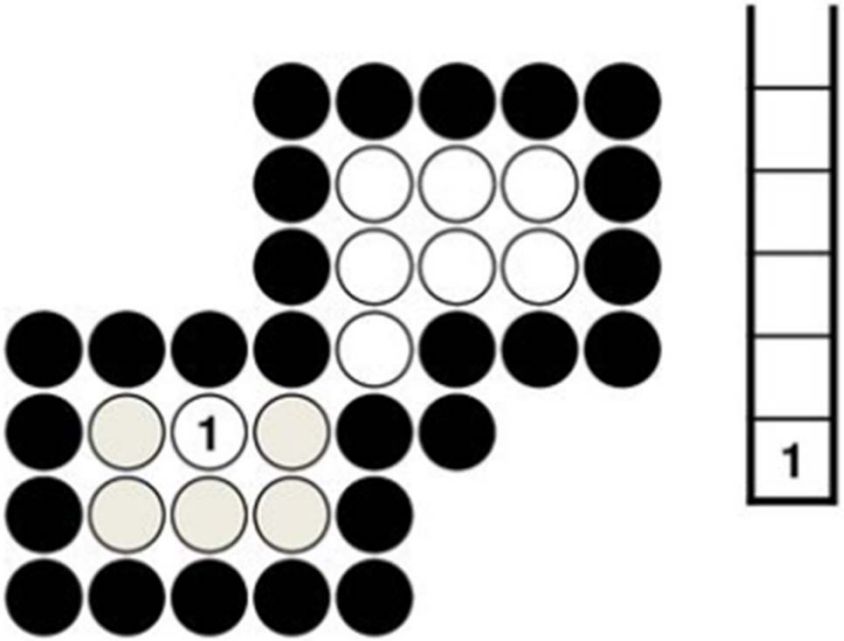
4 Connected Example



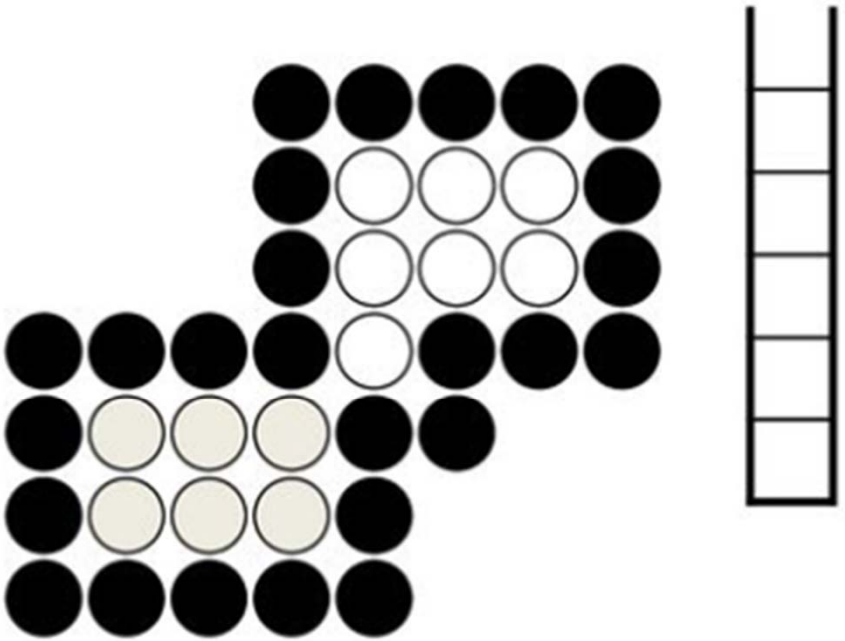
4 Connected Example



4 Connected Example



4 Connected Example



Some region remains unfilled

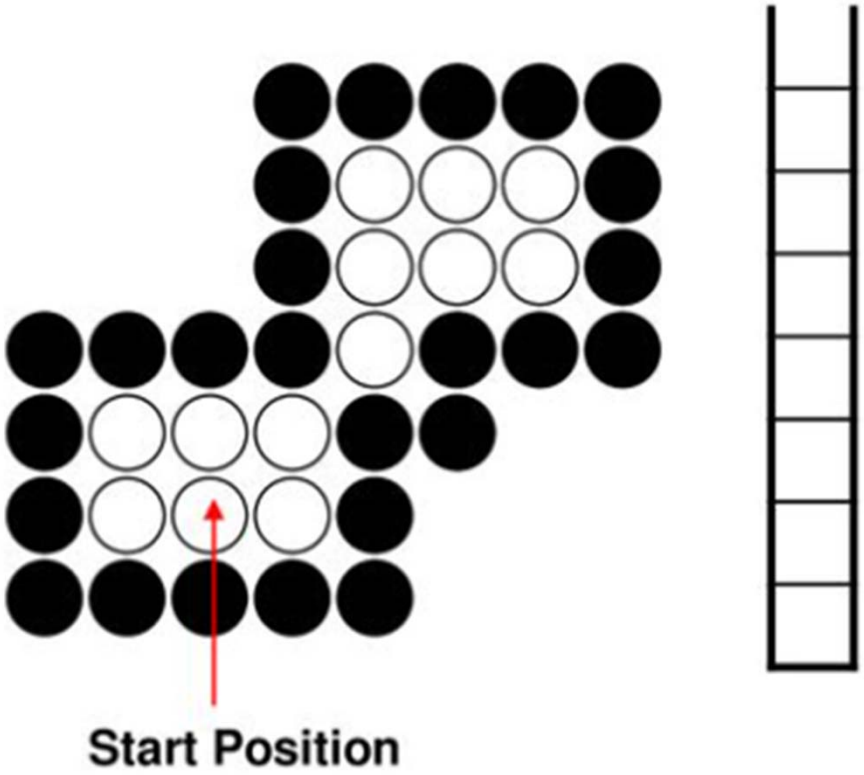
Boundary/Edge Fill Algorithm

```
void boundaryfill(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);
        boundaryfill(x+1,y,fill_color,boundary_color);
        boundaryfill(x-1,y, fill_color,boundary_color);
        boundaryfill(x,y+1, fill_color,boundary_color);
        boundaryfill(x,y-1, fill_color,boundary_color);
    }
}
```

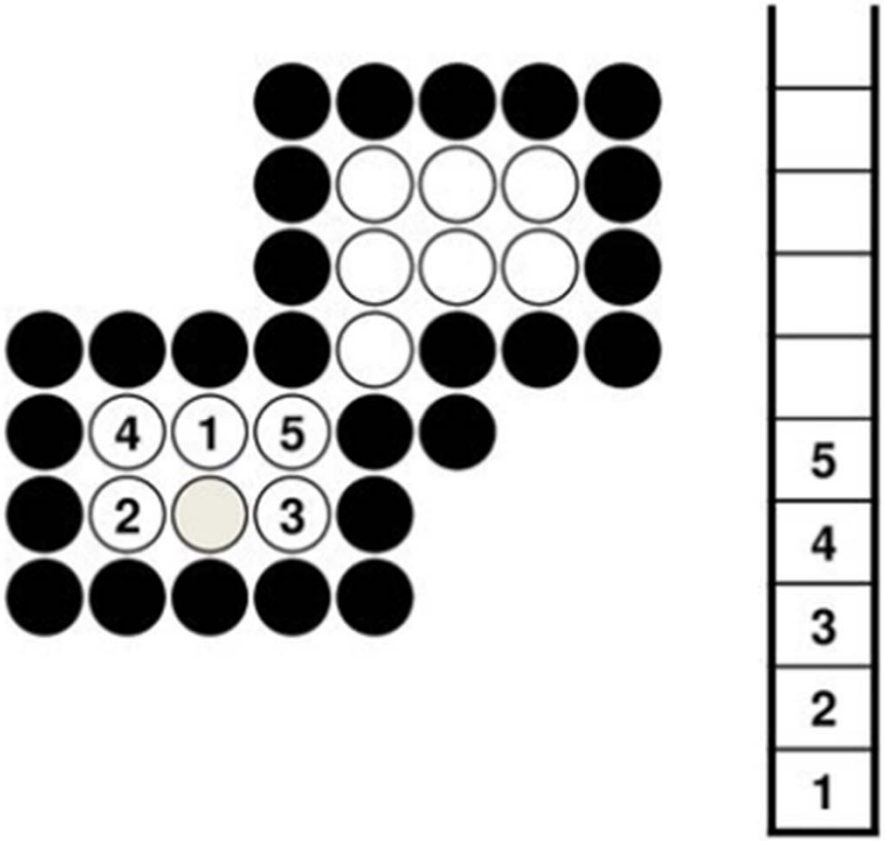
Flood Fill Algorithm

```
void floodfill(int x,int y,int fill_color,int old_color)
{
    if(getpixel(x,y)==old_color)
    {
        putpixel(x,y,fill_color);
        floodfill(x+1,y,fill_color,old_color);
        floodfill(x-1,y, fill_color,old_color);
        floodfill(x,y+1, fill_color,old_color);
       floodfill(x,y-1, fill_color,old_color);
    }
}
```

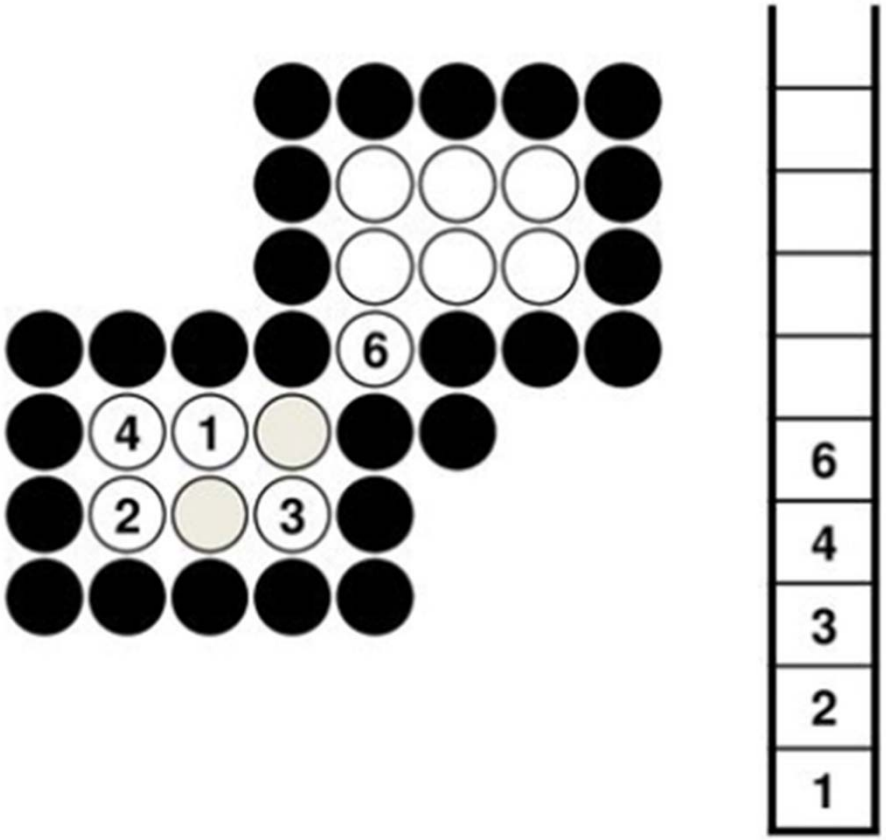
8 Connected Example



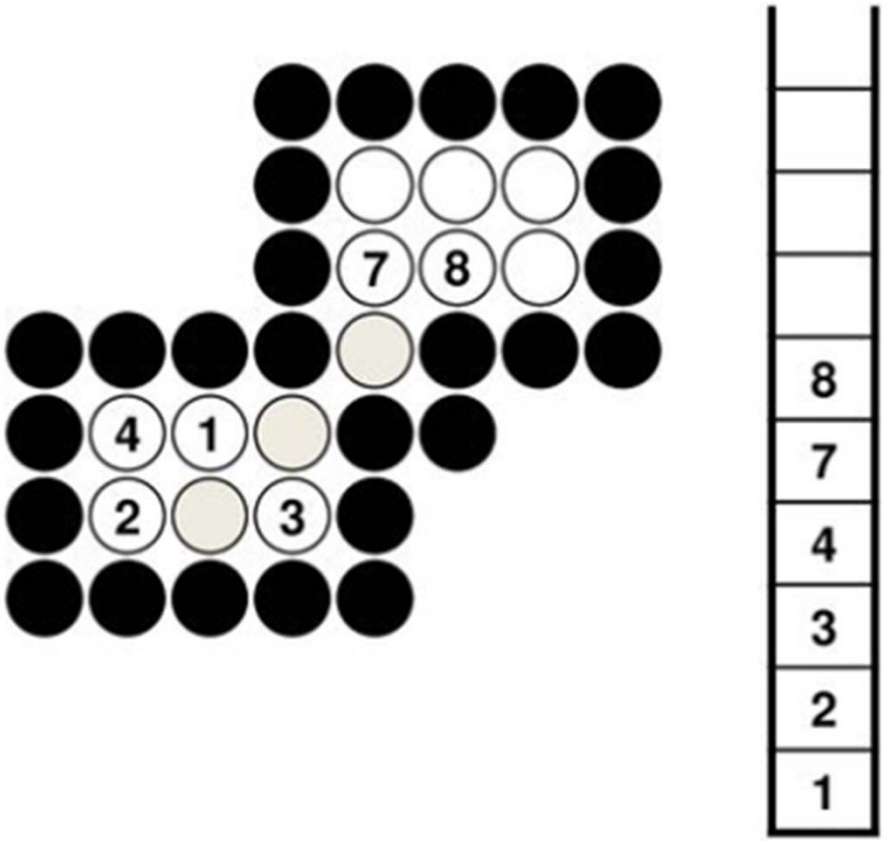
8 Connected Example



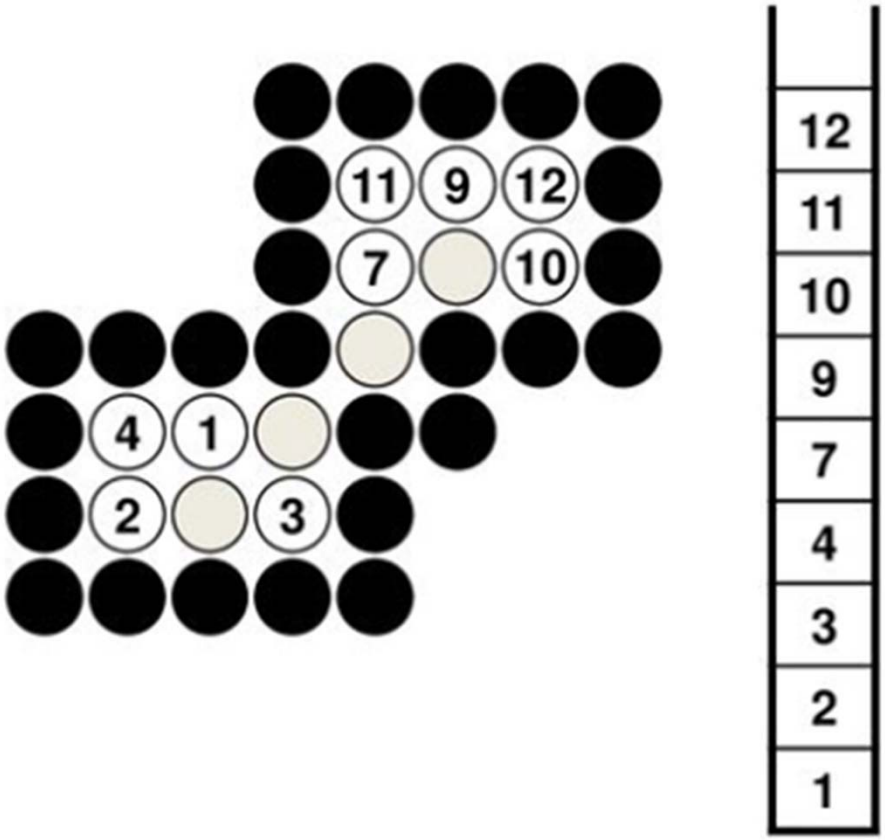
8 Connected Example



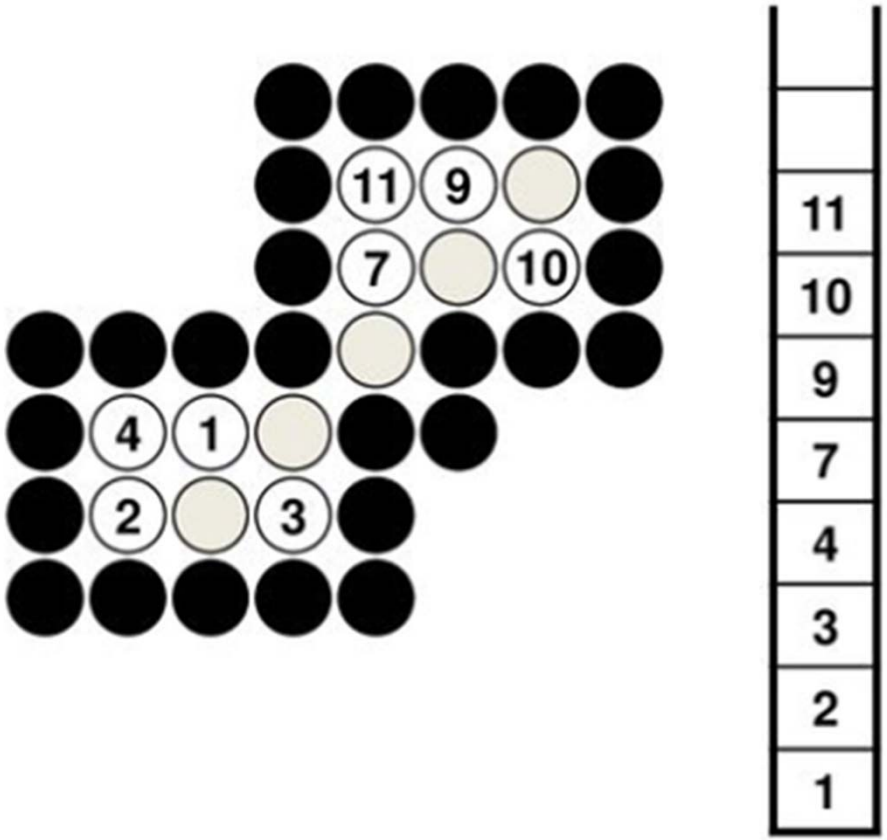
8 Connected Example



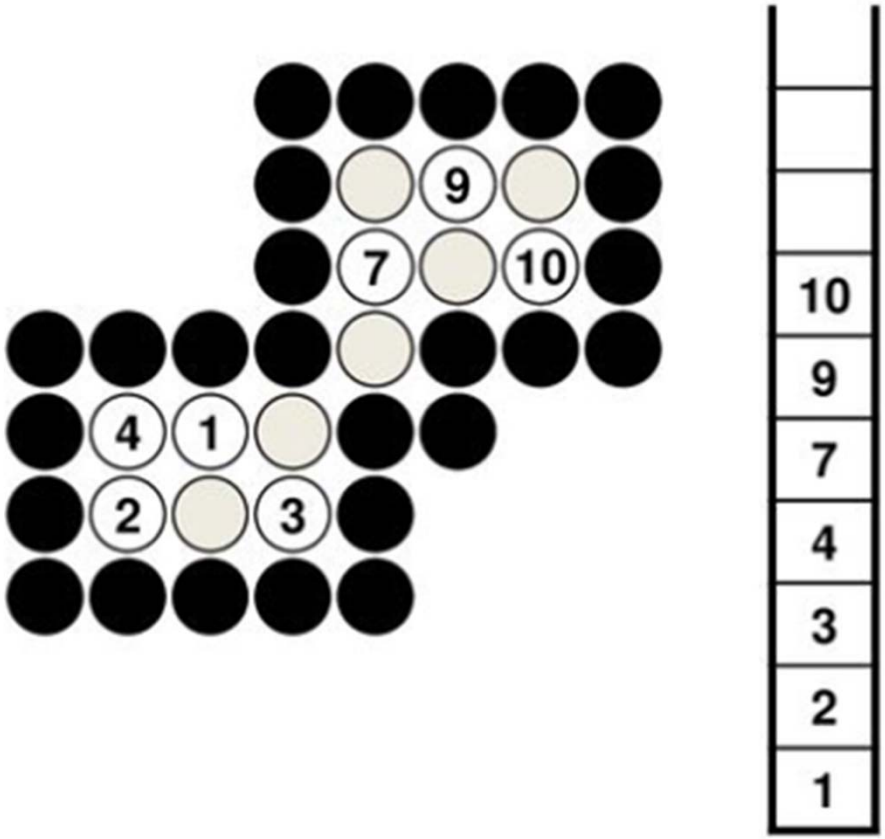
8 Connected Example



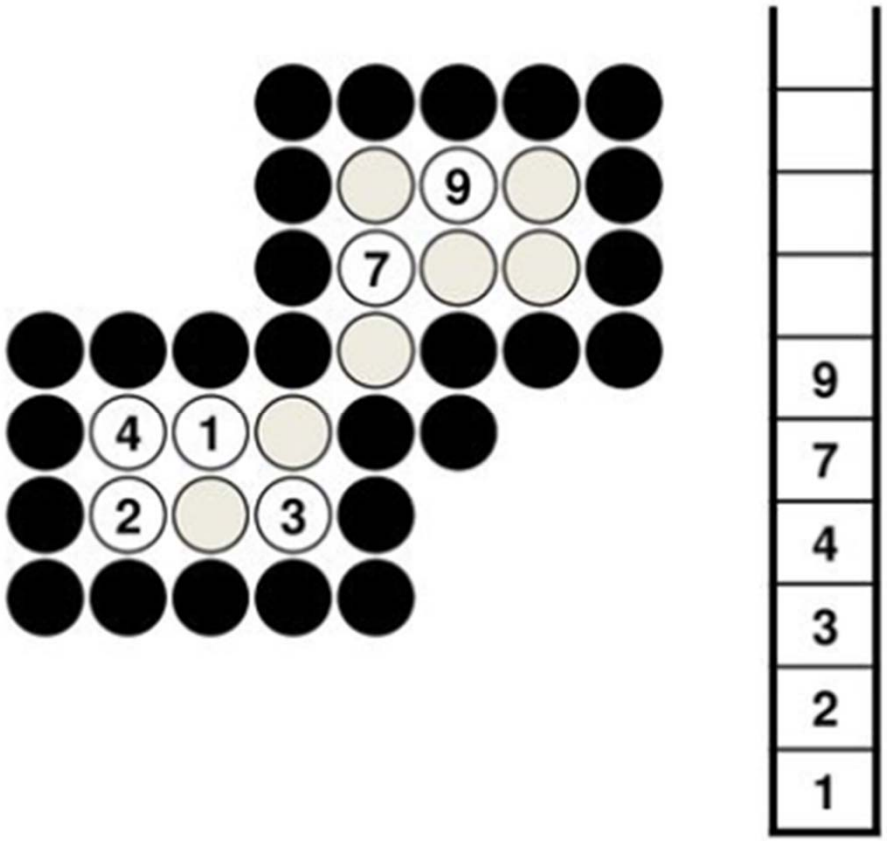
8 Connected Example



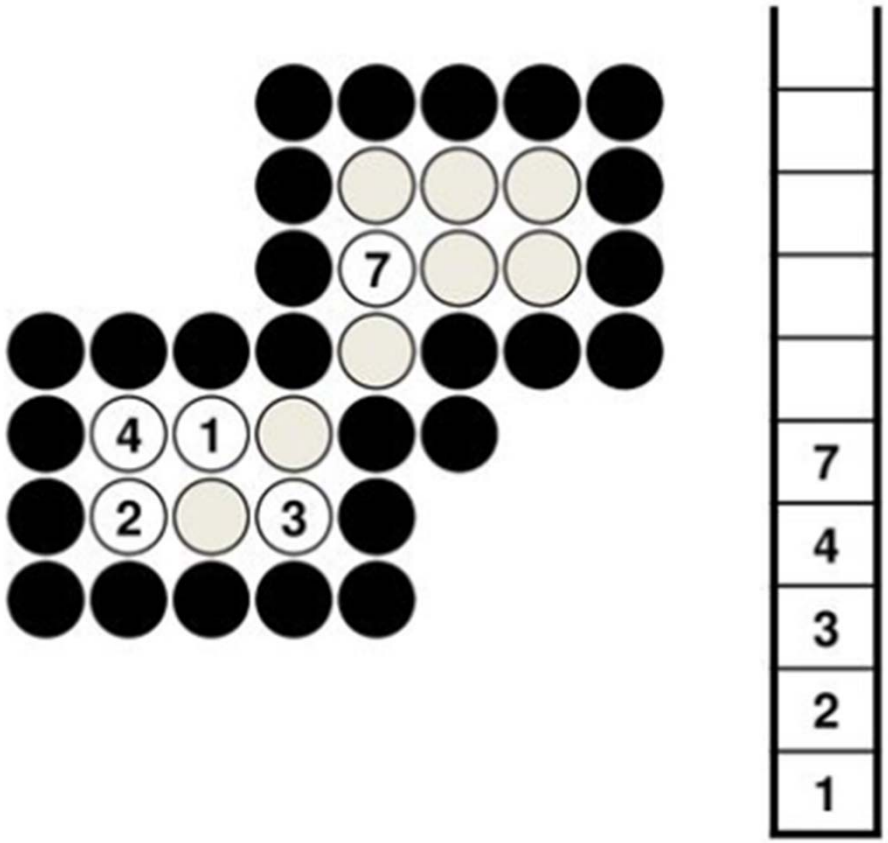
8 Connected Example



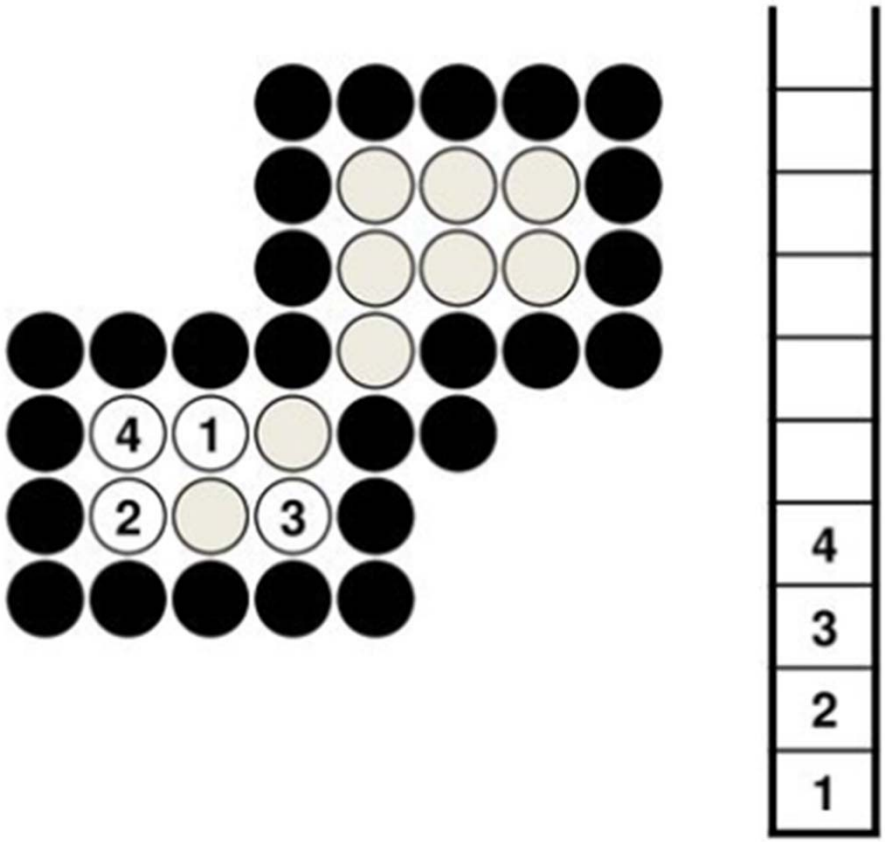
8 Connected Example



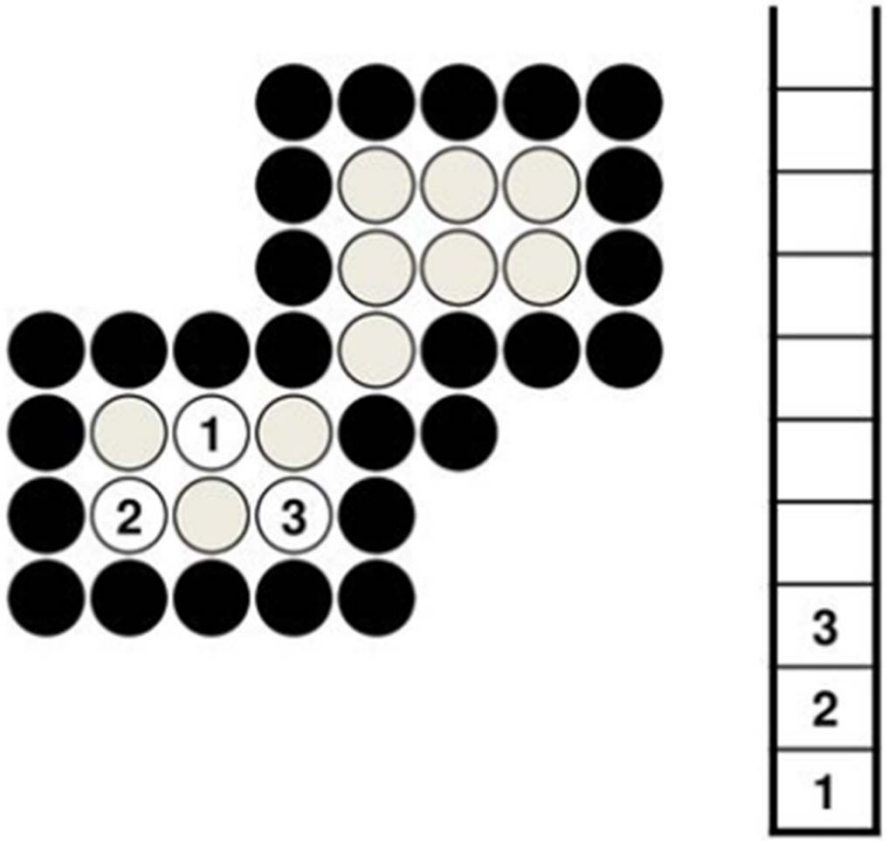
8 Connected Example



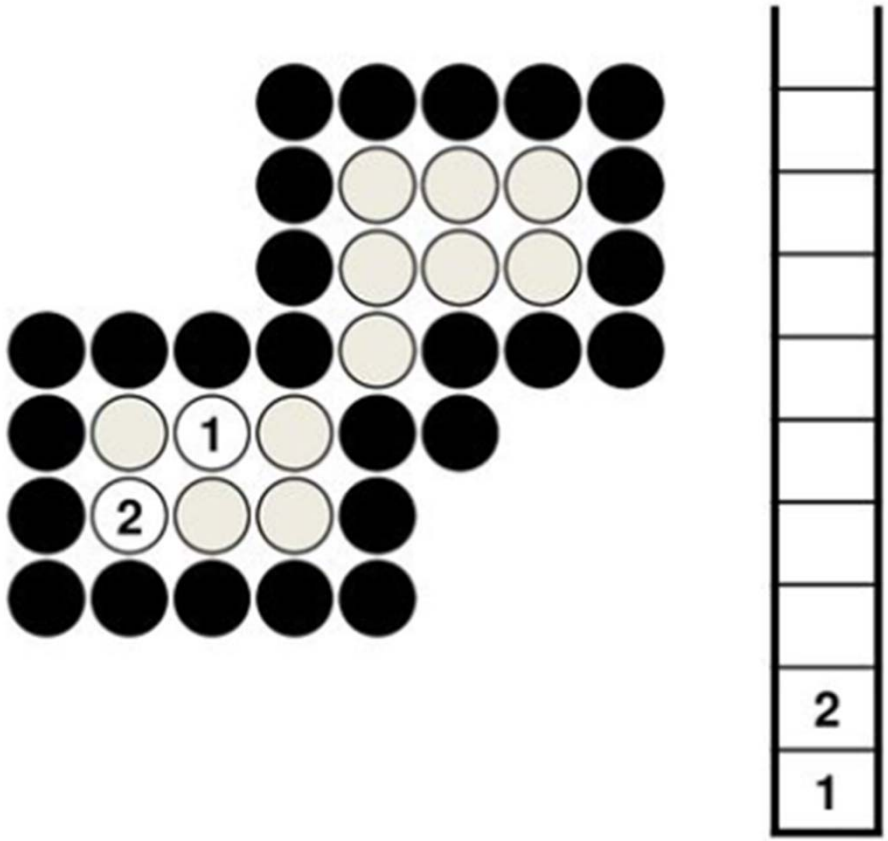
8 Connected Example



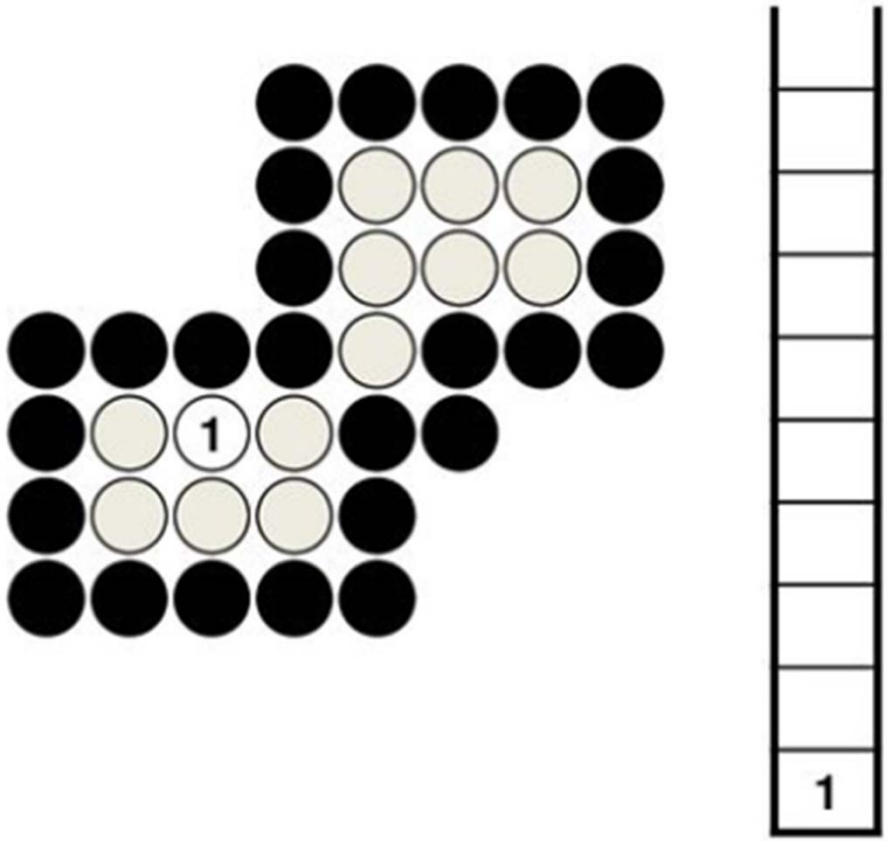
8 Connected Example



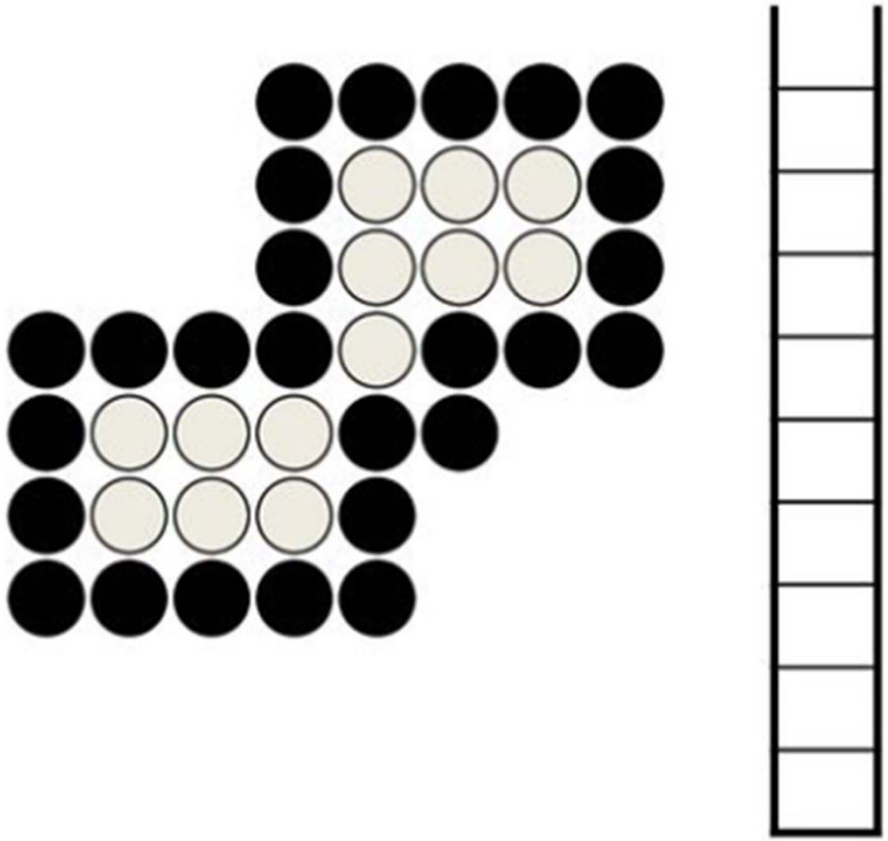
8 Connected Example



8 Connected Example



8 Connected Example



Flood-fill Algorithm	Boundary-fill Algorithm
It can process the image containing more than one boundary colors.	It can only process the image containing single boundary color.
Flood-fill algorithm is comparatively slower than the Boundary-fill algorithm.	Boundary-fill algorithm is faster than the Flood-fill algorithm.
In Flood-fill algorithm a random color can be used to paint the interior portion then the old one is replaced with a new one.	In Boundary-fill algorithm Interior points are painted by continuously searching for the boundary color.
It requires huge amount of memory.	Memory consumption is relatively low in Boundary-fill algorithm.
Flood-fill algorithms are simple and efficient.	The complexity of Boundary-fill algorithm is high.

Scanline Polygon Filling Algorithm

- Scanline filling is basically filling up of polygons using horizontal lines or scanlines.
- This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections.
- These intersection points are the sorted from left to right and the corresponding positions between each intersection pair are set to the specified fill color.

Scanline Polygon Filling Algorithm

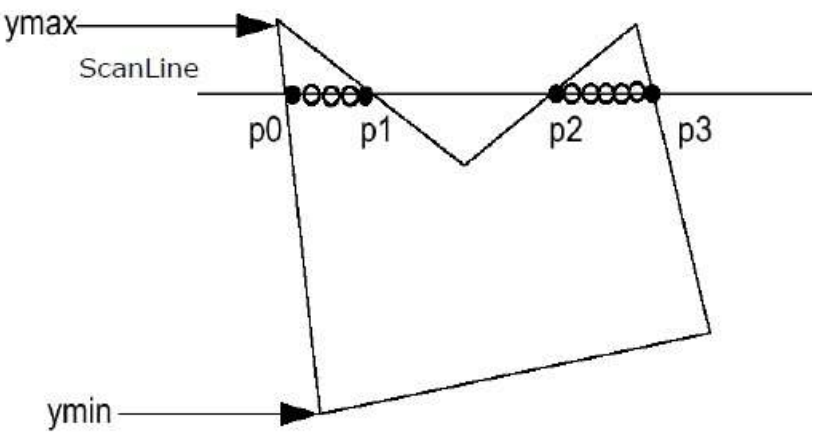
- The algorithm works according to following steps:

  Step 1: Find out the Ymin and Ymax from the given polygon

  Step 2: Scanline intersects with each edge of the polygon from Ymax to Ymin. Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.

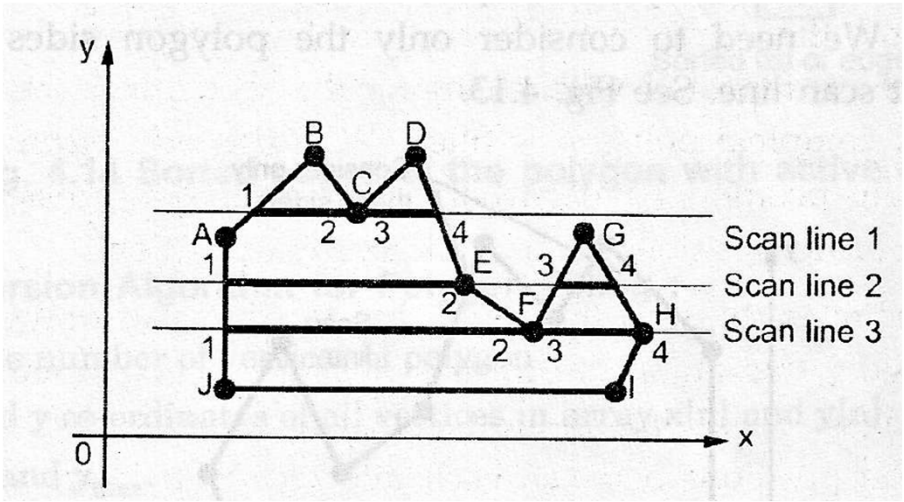
  Step 3: Sort the intersection point in the increasing order of X coordinate i.e. (p0, p1), (p1, p2), and (p2, p3).

  Step 4: Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.



Scanline Polygon Filling Algorithm

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



Edge Intersection Calculation with Scanline

- Coherence methods often involve incremental calculations applied along a single scan line or between successive scan lines.
- In determining edge intersections, we can set up incremental coordinate calculations along any edge by exploiting the fact that the slope of the edge is constant from one scan line to the next.
- For above figure we can write slope equation for polygon boundary as follows.
- $m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$
- Since change in y coordinates between the two scan lines is simply
- $y_{k+1} - y_k = 1$

Edge Intersection Calculation with Scanline

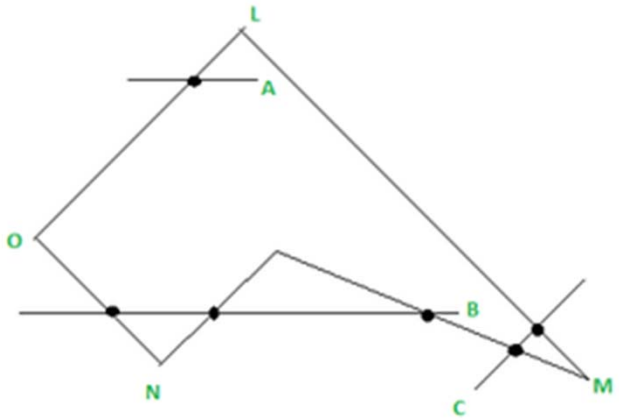
- So slope equation can be modified as follows
- $m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$
- $m = \frac{1}{x_{k+1} - x_k}$
- $x_{k+1} - x_k = \frac{1}{m}$
- $x_{k+1} = x_k + \frac{1}{m}$
- Each successive x intercept can thus be calculated by adding the inverse of the slope and rounding to the nearest integer.

Inside/Outside Test

- To determine a point lies inside a polygon or not, in computer graphics, we have two methods :
  (a) Even-Odd method (odd-parity rule)
  (b) Winding number Method

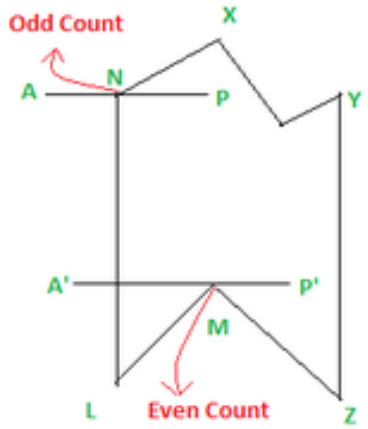
Inside/Outside Test

- **Even-Odd Method**
  - Constructing a line segment between the point to be examined and a known point outside the polygon is the one way to determine a point lies inside a polygon or not.
  - The number of times the line segment intersects the polygon boundary is then counted.
  - The point is an internal point if the number of polygon edges intersected by this line is odd; otherwise, the point is an external point.



Inside/Outside Test

- **Even-Odd Method**
  - *But this even-odd test fails when the intersection point is a vertex.*
  - To handle this case, few modifications are needed.
  - We must look at the other end points of the two segments of a polygon which meet at this vertex. If these points lies on the same side of the constructed line A'P', then the intersection point counts as an even number of intersection.
  - But if they lie on the opposite side of constructed line AP, then the intersection points counts as a single intersection.



Inside/Outside Test

- **Winding Number Method**
  - In this method, instead of just counting the intersections, we give each boundary line crossed a direction number, and we sum these directions numbers. The direction number indicates the direction of the polygon edge was drawn relative to the line segment we constructed for the test.
  - Example : To test a point (x_i, y_i), let us consider a horizontal line segment $y = y_i$ which runs from outside the polygon to (x_i, y_i). We find all the sides which crossed this line segment.
  - Now there are 2 ways for side to cross, the side could be drawn starting below end, cross it and end above the line. In this case we can give direction numbers – 1, to the side or the edge could start above the line & finish below it in this case, given a direction 1. The sum of the direction numbers for the sides that cross the constructed horizontal line segment yield the “Winding Number” for the point.

Inside/Outside Test

- **Winding Number Method**
  - If the winding number is non-zero , the point is interior to polygon, else, exterior to polygon.
  - In the above figure, the line segment crosses 4 edges having different direction numbers : 1, -1, 1& -1 respectively, then :

    Winding Number = 1 + (-1) + 1 + (-1) = 0

    So the point P is outside the Polygon. The edge has direction Number -1 because it starts below the line segment & finishes above. Similarly, edge has direction Number +1 because it starts from above the line segment & finishes below the line segment